Image-Based Modeling, Rendering, and Lighting

Paul Debevec (organizer)
USC Institute for Creative Technologies

Christoph Bregler
Stanford University

Michael Cohen
Microsoft Corporation

Leonard McMillan
Massachusetts Institute of Technology

François Sillion
iMAGIS - GRAVIR/IMAG

Richard Szeliski
Microsoft Corporation

SIGGRAPH 2000 Course 35 (Full Day)
Tuesday, July 25, 2000

Course Abstract

Image-based modeling, rendering, and lighting differs from traditional graphics in that the geometry, appearance, and lighting in a scene can be derived from real photographs. These techniques often allow for shorter modeling times, faster rendering speeds, and unprecedented levels of photorealism. In this course we will explain and demonstrate a variety of ways of turning images into models and then back into renderings, including movie maps, panoramas, image warping, photogrammetry, light fields, and 3D scanning. This course overviews the relevant topics in computer vision, and show how these methods relate to image-based rendering techniques. The course shows ways of applying the techniques to animation as well as to 3D navigation, and to both real and synthetic scenes. One underlying theme is that the various modeling techniques make tradeoffs between navigability, geometric accuracy, manipulability, ease of acquisition, and level of photorealism; another theme is the close connection between image-based techniques and global illumination. The course shows how image-based lighting techniques allow photorealistic additions and modifications to be made to image-based models. The described techniques are illustrated with results from recent research, pioneering projects, and creative applications in art and cinema.

Note: This course and SIGGRAPH 2000 Course #19, *3D Photography*, cover related topics and are designed to be complimentary.

Presenters

Christoph Bregler
Assistant Professor
Computer Science Department
Gates 138, 353 Serra Mall
Stanford University
Stanford, CA 94305
(650) 725-6359
(650) 725-1449 Fax
bregler@cs.stanford.edu
http://www.cs.stanford.edu/~bregler

Chris Bregler is an Assistant Professor in Computer Science at Stanford University. He received his Diplom in Computer Science from Karlsruhe University in 1993 and his M.S. and Ph.D. in Computer Science from U.C. Berkeley in 1995 and 1998. He also worked for several companies including IBM, Hewlett Packard, and Interval. He is a member of the Stanford Computer Graphics and the Robotics Laboratory. His research interests are in the areas of Computer Vision, Graphics, and Learning. Currently he focuses on topics in visual motion capture, human face, speech, and body gesture recognition and animation, and image based modeling and rendering.

Michael F. Cohen
Senior Researcher
Manager, Graphics Group
Microsoft Research
One Microsoft Way
Redmond WA 98052
(425) 703-0134
(425) 936-0502 Fax
mcohen@microsoft.com
http://www.research.microsoft.com/graphics/cohen/

Dr. Michael F. Cohen, senior researcher and manager of the Microsoft graphics research group, joined Microsoft Research in 1994 from Princeton University where he was an Assistant Professor of Computer Science. Dr. Cohen received his Ph.D. in 1992 from the University of Utah. He also holds undergraduate degrees in Art and Civil Engineering from Beloit College and Rutgers University respectively, and an M.S. in Computer Graphics from Cornell. Dr. Cohen also served on the Architecture faculty at Cornell University and was an adjunct faculty member at the University of Utah. His work at the University of Utah focused on spacetime control for linked figure animation. He is perhaps better known for his work on the radiosity method for realistic image synthesis as discussed in his recent book "Radiosity and Image Synthesis" (co-authored by John R. Wallace). Dr. Cohen has published and presented his work internationally in these areas. At Microsoft, Dr. Cohen has worked on a number of projects, including the IBMR projects "The Lumigraph" and "Layered Depth Images". He is also involved in the "Virtual Cinematographer" project to create automatic camera placement and sequencing of shots for interactive visual experiences, and in adding expressive refinements to the work in linked figure animation. Dr. Cohen served as the papers chair for SIGGRAPH 98, where he was also awarded the 1998 Computer Graphics Achievement Award for the development of practical radiosity methods for realistic image synthesis.

Paul Debevec
Executive Producer, Graphics Research
Institute for Creative Technologies
University of Southern California
13274 Fiji Way
Marina del Rey, CA 90292
(310) 574-5700
(310) 574-5725 fax
debevec@ict.usc.edu
http://www.debevec.org/

Paul Debevec earned degrees in Math and Computer Engineering at the University of Michigan in 1992 and completed his Ph.D. at the University of California at Berkeley in 1996; he now leads a research and production group at the University of Southern California's Institute for Creative Technologies. Debevec has worked on a number of image-based modeling and rendering projects, beginning in 1991 in deriving a 3D model of a Chevette from photographs for an animation project. Debevec has collaborated on projects at Interval Research Corporation in Palo Alto that used a variety of image-based techniques for interactive applications; the "Immersion '94" project done with Michael Naimark and John Woodfill developed an image-based walkthrough of the Banff national forest and his art installation "Rouen Revisited" done with Golan Levin showed at the SIGGRAPH 96 art show. His Ph.D. thesis in collaboration with C.J. Taylor presented an interactive method of modeling architectural scenes from sparse sets of photographs and for rendering these scenes realistically. Debevec led the creation of an image-based model of the Berkeley campus for "The Campanile Movie" shown at the SIGGRAPH 97 Electronic Theater and whose techniques were used in creating the "bullet time" shots in the 1999 film *The Matrix*. Debevec directed the animation "Rendering with Natural Light" at the SIGGRAPH 98 ET which demonstrated image-based lighting from high dynamic range photography. His film "Fiat Lux" featured in *The Story of Computer Graphics* combined the previous techniques to place dynamic objects in a reconstruction of St. Peter's Basilica. With Steve Gortler, Debevec organized the course "Image-Based Modeling and Rendering" at SIGGRAPH 98 and "Image-Based Modeling, Rendering, and Lighting" at SIGGRAPH 99.

Leonard McMillan
Assistant Professor
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139
(617) 258-0381
(617) 253-6652 Fax
mcmillan@graphics.lcs.mit.edu
http://graphics.lcs.mit.edu/~mcmillan/

Leonard McMillan is an assistant professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. He received B.S. and M.S. degrees in Electrical Engineering from the Georgia Institute of Technology in 1983 and 1984, and his Ph.D. in computer science in 1997 from the University of North Carolina at Chapel Hill. His experiences designing digital signal processing hardware have fueled his interest in making image-based rendering run at interactive speeds. His plenoptic modeling work from SIGGRAPH'95 demonstrated how the optical flow information derived from panoramic images could be used to simulate a three-dimensional immersive environments. Leonard is currently exploring new algorithms and hardware designs for the accelerating image-based rendering methods. He currently teaches introductory computer graphics and computer architecture and lectures on a wide range of issues related to image-based rendering.

François X. Sillion
Senior Researcher
French National Institute for Computer Science and Control (INRIA)
iMAGIS - GRAVIR/IMAG
B.P. 53, 38041 Grenoble Cedex 9
France
+33 4 76 51 43 54
+33 4 76 63 55 80 Fax
Francois.Sillion@imag.fr
http://www-imagis.imag.fr/~Francois.Sillion/

François Sillion is a senior researcher at the Institute for Research in Computer Science and Control (INRIA), working in the iMAGIS project in Grenoble, France. He received undergraduate and graduate degrees (1986) in Physics at the Ecole Normale Supérieure in paris, France, and a PhD in Computer Science from the University of Paris-XI/Orsay (1989). Dr. Sillion worked for two years as a post-doc at Cornell's Program of Computer Graphics, before joining France's National Center for Scientific Research (CNRS), working first in Paris, then in Grenoble (1993). His research interest include the simulation of illumination for realistic image synthesis (he worked on several extensions to the radiosity method, including non-diffuse reflection and hierarchical techniques using clusters); progressive rendering techniques allowing a continuous trade-off between quality and speed for interactive applications; image-based techniques for the acceleration of rendering; and the application of computer graphics techniques to the simulation of non-visible radiation (botanical studies and radio waves). Dr. Sillion published, with Claude Puech, a comprehensive book on radiosity and global illumination, and co-authored several papers on all the above subjects. In addition to participating in many conference program committees, he is an associate editor of ACM Transactions on Graphics, serves on the editorial board of Computer Graphics Forum, and chairs the EUROGRAPHICS working group on rendering, organizing a yearly workshop on rendering.

Richard Szeliski
Senior Researcher
Microsoft Corporation, Vision Technology Group
One Microsoft Way
Redmond, WA 98052-6399
(425) 936-4774
(425) 936-0502 Fax
szeliski@microsoft.com
http://www.research.microsoft.com/research/vision/szeliski/

Richard Szeliski is a Senior Researcher in the Vision Technology Group at Microsoft Research, where he is pursuing research in 3-D computer vision, video scene analysis, and image-based rendering. His current focus in on constructing photorealistic 3D scene models from multiple images and video, and on automatically parsing video for editing and retrieval applications. Dr. Szeliski received a B. Eng. degree in Honours Electrical Engineering from McGill University, Montreal, in 1979, a M. Appl. Sc. degree in Electrical Engineering from the University of British Columbia, Vancouver, in 1981, and a Ph. D. degree in Computer Science from Carnegie Mellon University, Pittsburgh, in 1988. He joined Microsoft Research in 1995. Prior to Microsoft, he worked at Bell-Northern Research, Montreal, at Schlumberger Palo Alto Research, Palo Alto, at the Artificial Intelligence Center of SRI International, Menlo Park, and at the Cambridge Research Lab of Digital Equipment Corporation, Cambridge. Dr. Szeliski has published over 60 research papers in computer vision, computer graphics, medical imaging, neural nets, and parallel numerical algorithms, as well as the book Bayesian Modeling of Uncertainty in Low-Level Vision. He is a member of the Association of Computing Machinery, the Institute of Electrical and Electronic Engineers, and Sigma Xi. He was an organizer of the first Workshop on Image-Based Modeling and Rendering, and is currently an Associate Editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence.

Course Schedule and Syllabus

Morning

1. 08:30 - 08:50, 20 minutes (Debevec)

Introduction and Overview

1. What is image-based modeling and rendering (IBMR)
2. Differences between image-based modeling and rendering and traditional 3D graphics
3. Why this is a promising area
4. Some Examples
5. Advantages and disadvantages
6. The spectrum of IBMR - from image indexing to 3D scanning

2. 08:50 - 10:00, 70 minutes (Sillion)

Image Formation Fundamentals and Using IBMR to Accelerate Rendering

1. What is an image?
2. Simple projective geometry, and the pin-hole camera model
3. How light interacts with matter
4. The relationship of global illumination to IBMR
5. Challenges posed by non-diffuse reflectance
6. Image caching techniques
7. Affine sprite warping

Break

3. 10:15 - 11:00, 45 Minutes (Szeliski)

Determining Geometry from Images

1. Why geometry is useful for image-based rendering
2. Computer Vision as Inverse Computer Graphics
3. Notes on camera calibration
4. Computing depth maps with stereo and multi-baseline stereo
5. Image correspondence techniques
6. Structure from Motion
7. Overview of other methods: Photogrammetric Modeling, 3D Scanning

Note: Additional material on determining geometry from images is available in the course notes for Course #19, *3D Photography*. Topics covered in detail include photogrammetric modeling, silhouette-based methods, 3D laser scanning, and other active sensing methods.

4. 11:00 - 12:00, 60 Minutes (McMillan)

Image-Based Rendering: With or Without Structure?

1. Image mosaicing and cylindrical panoramic viewing
2. Explanation of a depth map
3. Ways to warp an image based on depth
4. Panoramic image warping
5. Turning images and depth into a navigable environment

Lunch (12:00 – 01:30)

Afternoon

5. 01:30 - 02:20, 50 Minutes (Cohen)

LDI and Lightfield / Lumigraph representations

1. What is an image versus what is a model?
2. Layered depth images (LDIs)
3. The plenoptic function
4. Reduction to 4D
5. Light field rendering and the Lumigraph
6. Combining light fields with geometry
   - Silhouette models (Lumigraph)
   - View-dependent texture-mapping (Façade)

6. 02:20 - 03:00, 40 Minutes (Debevec)

Image-Based Lighting

1. Recovering lighting information from photographs
   - High dynamic range photography / light probes / inverse lighting
2. Illuminating synthetic objects with real light
3. Making additions and modifications to image-based models maintaining correct global illumination
4. Inverse global illumination: recovering material properties of real scenes from photographs
5. Communicating the sense of brightness using post-processing operations
6. The Light Stage: illuminating real objects/people with recorded light for compositing

Break

7. 03:15 - 04:05, 50 Minutes (Bregler)

Applications of IBMR in human animation

1. How IBMR generalizes from 3D navigation to kinematic domains
2. Facial animation with image-based rendering
3. Human figure animation with image-based modeling

8. 04:05 - 04:40, 35 Minutes (Debevec)

Applications of IBMR in Art and Cinema

1. Matte paintings vs. 3D Models in Movies (Gone with the Wind / Star Wars)
2. The Aspen and San Francisco Movie Map projects (Lippman)
3. Naimark's "Displacements" - physically projecting images onto geometry
4. Dayton Taylor's Timetrack system & "jump morphing"
5. Rouen Revisited (SIGGRAPH 96 art show), Mona Lisa Morph (SIGGRAPH 96)
   Buf Compagnie's Like a Rolling Stone (SIGGRAPH 96),
   Tour into the Picture (SIGGRAPH 97); What Dreams May Come (1998),
   The Matrix (1999); Prince of Egypt (1999); Fight Club (1999); Mission Impossible II (2000)

9. 04:40 - 05:00, 20 Minutes (Everyone)

Questions and Dialog

Table of Contents

5. LDI and Lightfield / Lumigraph representations

Slides: Image or Object? (Cohen)

Paper: *Layered Depth Images*
Jonathan Shade, Steven Gortler, Li-wei Hey, and Richard Szeliski, Proc. SIGGRAPH 97

Paper: *Light Field Rendering*
Marc Levoy and Pat Hanrahan, Proc. SIGGRAPH 96

Paper: *The Lumigraph*
S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, Proc. SIGGRAPH 96

Paper: *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*
Paul Debevec, George Borshukov, and Yizhou Yu, 9th Eurographics Rendering Workshop, 1998

6. Image-Based Lighting

Slides: Image-Based Lighting (Debevec)

Notes: *The Story of Reflection Mapping.*
Paul Debevec

Paper: *Recovering High Dynamic Range Radiance Maps from Photographs.*
Paul E. Debevec and Jitendra Malik, Proc. SIGGRAPH 97

Paper: *Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography*
Paul Debevec, Proc. SIGGRAPH 98

Paper: *Inverse Global Illumination:*
Recovering Reflectance Models of Real Scenes from Photographs.
Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins, Proc. SIGGRAPH 99

7. IBMR techniques for Animating People

Notes: *Video Based Animation Techniques for Human Motion* (Bregler)

Slides: IBMR Techniques for Animating People (Bregler)

Paper: *Video Rewrite: Driving Visual Speech with Audio*
Christoph Bregler, Michele Covell, Malcolm Slaney, Proc. SIGGRAPH 97

Paper: *Synthesizing Realistic Facial Expressions from Photographs*
Frédéric Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, and David H. Salesin, Proc. SIGGRAPH 98

Paper: *Making Faces*
Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Fredrick Pighin, Proc. SIGGRAPH 98

Paper: *Video Motion Capture*
Christoph Bregler and Jitendra Malik, *UCB Tech. Report CSD-97-973*

Paper: *Recovering Non-Rigid 3D Shape from Image Streams*
Christoph Bregler, Aaron Hertzman, and Henning Biermann, Proc. CVPR 2000

8. Applications of IBMR in Art and Cinema

Slides: Applications of IBMR in Art and Cinema (Debevec)

Notes: *Rouen Revisited*
Golan Levin and Paul Debevec

Introduction

What is Image-Based Modeling and Rendering?
And what is Image-Based Lighting?

Paul Debevec
USC Institute for Creative Technologies

A principal endeavor of computer graphics research has been the pursuit of photorealism. Early two-dimensional computer graphics gained a sense of depth by combining the simple algorithms for drawing lines with the mathematics of perspective projection. The wireframe look of such drawings fed a desire for a more solid appearance, which inspired the development of hidden surface removal algorithms. Shading algorithms allowed rendering surfaces with varying brightness levels as if they were being illuminated by sources of light, and shadow calculation techniques allowed objects to realistically cast shadows on each other. Techniques for representing and displaying curved surfaces expanded the variety of shapes that could be rendered, and we created modeling tools to help us generate complex models. Renderings that look as realistic as photographs have finally been achieved by using ray tracing and radiosity to simulate the myriad complex paths that light can take as it travels from its sources to the viewer.

The evolution of tools for modeling and rendering scenes with photorealistic fidelity - much of it represented in the twenty-seven years of the SIGGRAPH conference - is a monumental achievement that has had a profound influence on the visual medium. Nonetheless, the tools for creating complex models require a great deal of effort and skill to use, and the algorithms for rendering such images with accurate illumination remain computationally intensive and are still somewhat experimental. To wit: modeling is hard, and rendering is slow, which makes achieving truly compelling photorealism extremely difficult.

Suppose, for example, that we wanted to generate a photorealistic image of the cathedral of Notre Dame in Paris. We could start by figuring out the dimensions of the cathedral, perhaps by borrowing the architectural plans from the most recent restoration project, or by conducting our own surveying. We would then build up the towers and the rose window, brick by brick and pane by pane, and assign appropriate reflectance properties to each surface. We could use L-systems to generate synthetic trees in the adjacent garden, and we could specify an appropriate distribution of incident light from the sky. We could then use a global illumination algorithm that, with a great deal of computation, would simulate how light would bounce around the scene to generate a rendered image of the cathedral.

Alternately, we could simply visit the cathedral and take a picture of it. Taking the picture would not only require far less effort, but the picture would almost certainly be a far more convincing rendition of the scene - it is, by definition, photorealistic. But while a single photograph gives us an amazing amount of information about the scene's structure and appearance, it is a static frozen image. What we have lost is the ability to look in different directions, to move about in the scene, to collide with its surfaces, to change the light, to add objects, and to modify the scene itself. If we had constructed the computer model, all of this would have been possible, if not realistic.

Image-based modeling and rendering is about leveraging the ease with which photographs can be taken, the speed at which they can be displayed, and their amazing power to communicate, while at the same time transcending their limitations. The various forms of IBMR transcend the limitations by deriving some sort of representation of the scene from the photographs, and then using this representation to create renderings. The principal reason that image-based modeling and rendering is interesting is that these representations do not need to be as complete as traditional computer graphics models in order to transcend many of the limitations of photographs. To remove the restriction that it is impossible to look in different directions, we can take photographs of the scene looking in all directions, assemble the photographs into a panorama, and then allow the user to look around by displaying different sections of the panorama. To remove the restriction that one can't move about the scene, we can take many images of the scene from different locations, and then display the various images depending on where the user wants to go. To reduce the

number of images necessary, we can derive geometric representations of the scene through image correspondence, interactive photogrammetry, or active sensing, and then render this geometry from the desired viewpoint with colors projected on from the original photographs. As the techniques for deriving representations become more sophisticated, the fewer limitations there are.

Image-Based Modeling and Rendering is a relatively new field, but it has already produced degrees of interactivity and levels photorealism previously only dreamed of. With its current level of interest, it promises to continue to amaze us in the years to come. Furthermore, IBMR has the potential to fundamentally change the way we understand computer graphics. By starting with the answer - photorealistic renderings in the form of photographs and video - and discovering what it takes to transform them into models and then back into renderings, we have no choice but to gain an understanding of every perceptually relevant aspect of image synthesis.

This course again covers Image-Based Lighting, a technique which injects illumination from the real world into computer renderings. As such, it becomes a useful link in understanding the relationship between image-based and traditional computer graphics, and shows how the two can be combined while maintaining the photorealism we expect from image-based techniques.

This is already an exciting year for computer graphics and for image-based techniques in particular. Another excellent offering of innovative papers on image-based techniques is appearing in the papers session. Several new image-based software packages and hardware solutions have become available and will show at the exposition. *3D Photography*, a continuing SIGGRAPH course offered by Brian Curless and Steve Seitz, is a companion to this image-based modeling and rendering course. And perhaps most visibly, advanced image-based techniques continue to be employed in feature films such as *What Dreams May Come*, *The Prince of Egypt*, and *The Matrix*, *Fight Club*, *X-Men*, and *Mission Impossible II*, each of which offers an entirely different visual aesthetic. As the film industry helped inspire much of this recent image-based research by popularizing matte painting, environment mapping, and morphing (all forms of "image-based rendering" developed well before the term was in use), it's wonderful and fitting to see recent results from the research community help out in visual effects as well.

A central goal of this course is to give a basic understanding of the variety of techniques that have been developed in image-based modeling, rendering, and lighting. But the more important goal is to present the larger picture in which this variety of work can best be understood. To achieve this, an effort has been made to cover not just core material such as image warping and light fields, but to also present what lies near the frontier, such as movie maps, morphing, image-based human figure animation, and artistic applications. The result, I hope, will be a learning experience for all of us.

Paul Debevec
April 2000

Image-Based Modeling, Rendering, and Lighting

Paul Debevec
USC Institute for Creative Technologies

Michael Cohen
Microsoft Research

Leonard McMillan
MIT

Chris Bregler
Stanford University

Richard Szeliski
Microsoft Research

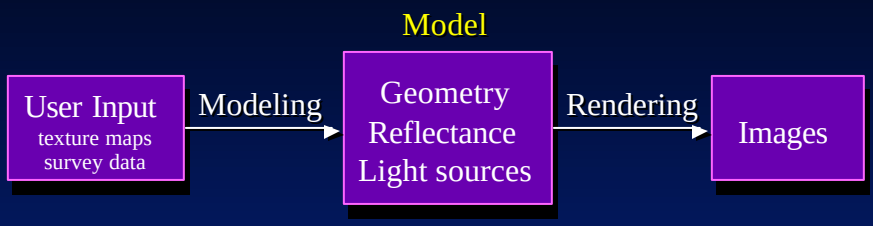
François Sillion
iMAGIS - GRAVIR/IMAG

SIGGRAPH 2000 Course #35

Tuesday, July 25, 2000

What is Image-Based Modeling and Rendering?

Traditional Modeling and Rendering

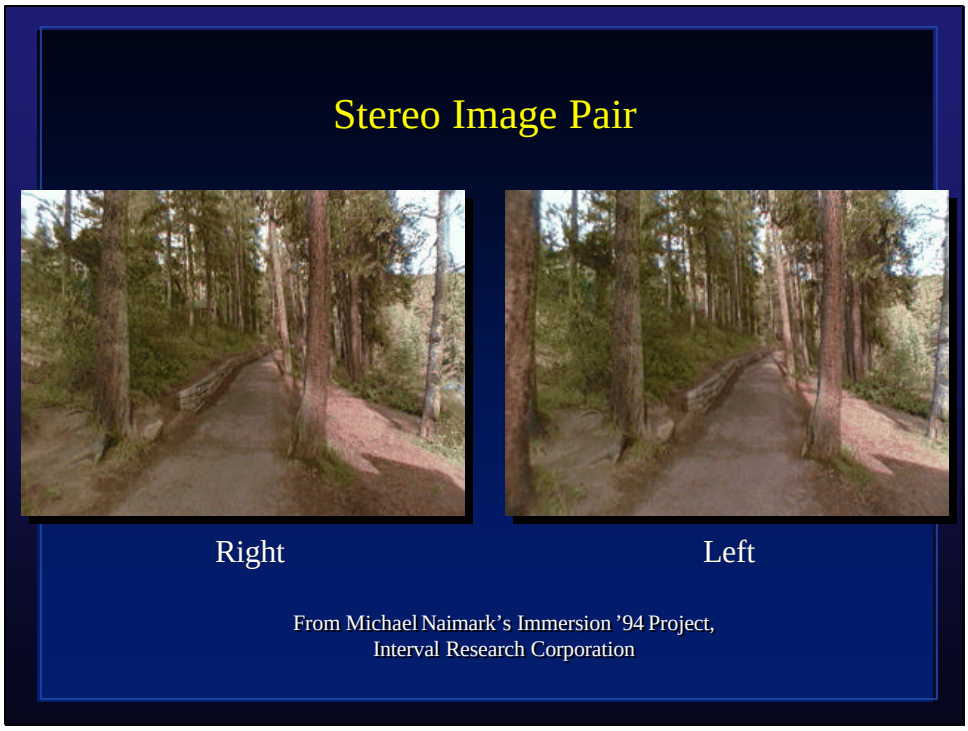


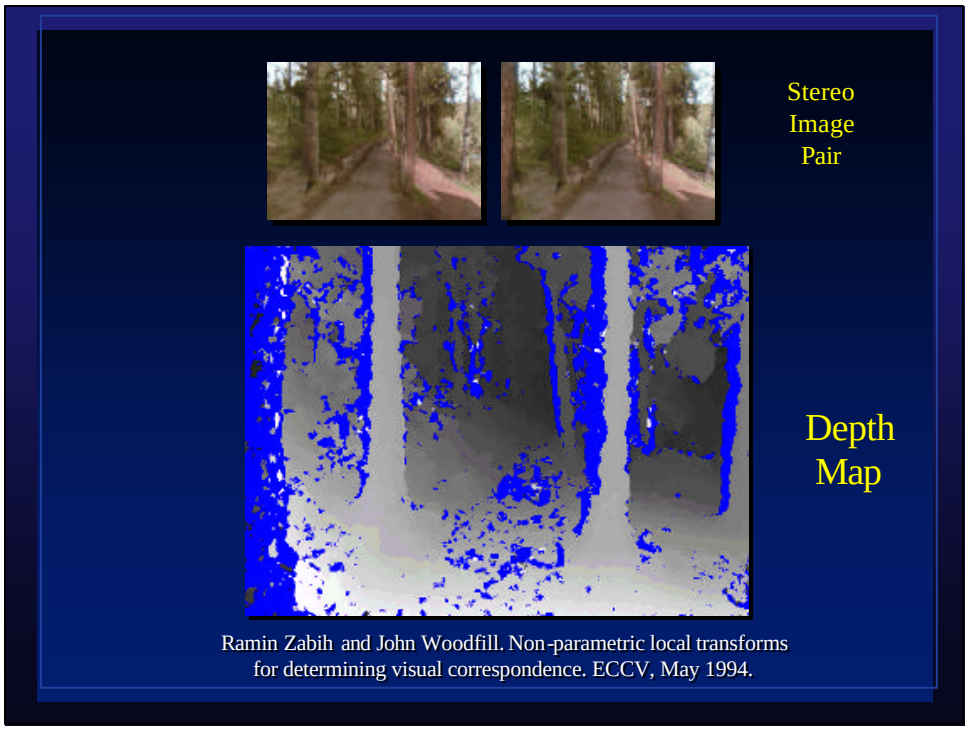
For Photorealism:

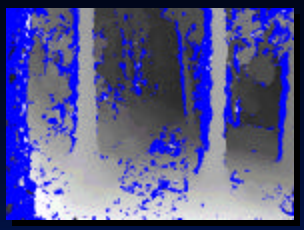
Modeling is Hard **Rendering is Slow**

Can we model and render something this complicated?

This depends on what we want to do with the model.


Stereo Image Pair
Right
Left
From Michael Naimark's Immersion '94 Project,
Interval Research Corporation


Stereo
Image
Pair
Depth
Map
Ramin Zabih and John Woodfill. Non-parametric local transforms
for determining visual correspondence. ECCV, May 1994.



Depth
Map

Synthetic Views

Immersion ’94

Michael Naimark
John Woodfill
Paul Debevec
Leo Villareal

Interval Research
Corporation

Stereo Image
Capture Rig

Video – Immersion '94

Image-Based Modeling and Rendering

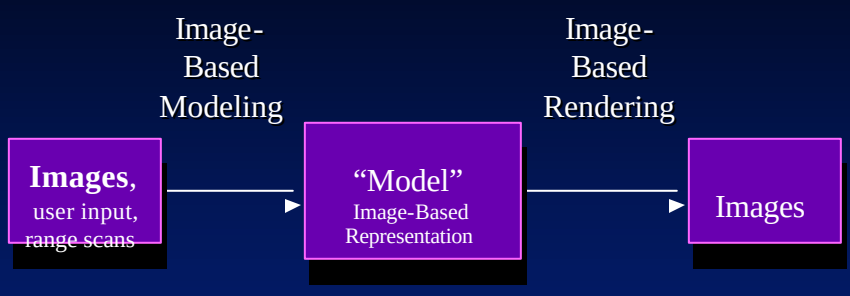


Image-Based *Modeling*

Images (photographs, renderings) are used to determine

- Scene Appearance
- Scene Geometry
- Lighting
- Reflectance Characterisics
- Kinematic Properties

=> Modeling scenes* photorealistically *is easier

Image-Based Rendering

Appearance in available views is used to determine appearance in novel views =>

Not necessary to perform full illumination computations =>

Rendering is faster

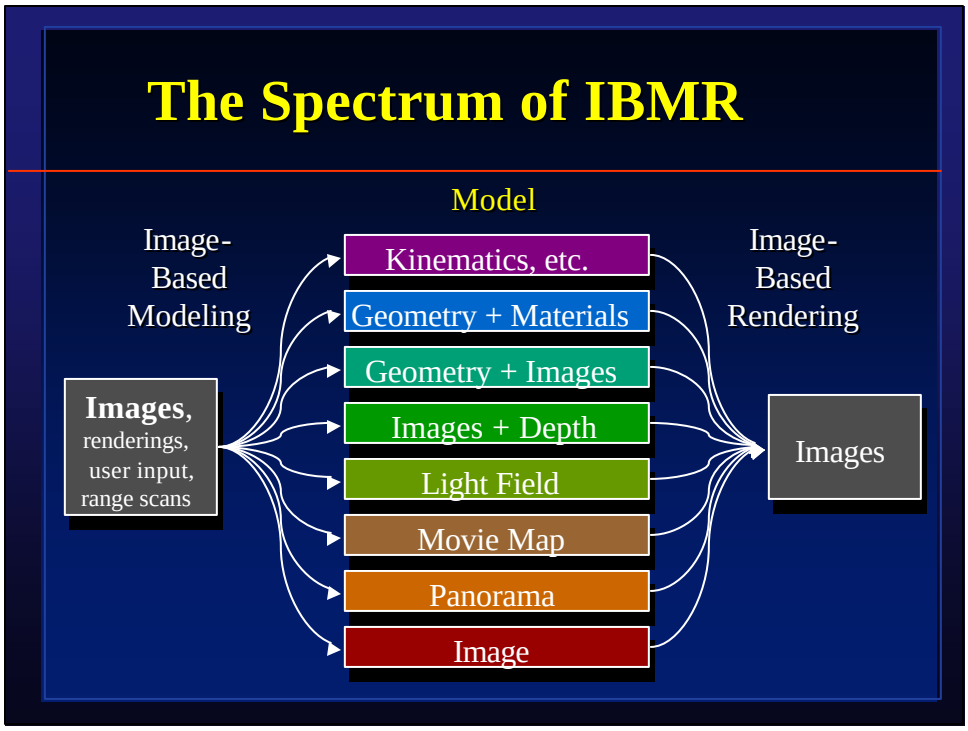


Image-Based Models: What do they allow?

Model	Movement	Geometry	Lighting
Geometry + Materials	Continuous	Global	Dynamic
Geometry + Images	Continuous	Global	Fixed
Images + Depth	Continuous	Local	Fixed
Light Field	Continuous	None	Fixed
Movie Map	Discrete	None	Fixed
Panorama	Rotation	None	Fixed
Image	None	None	Fixed

Global Illumination and Image-Based Lighting

Traditional Computer Graphics involves modeling with Matter: geometry with reflectance properties

Image-Based Modeling and Rendering involves modeling and rendering with Light, often deriving geometry and materials in the process

Image-Based Lighting allows us to combine real and synthetic graphics with consistent illumination, using images as light sources

Course Schedule

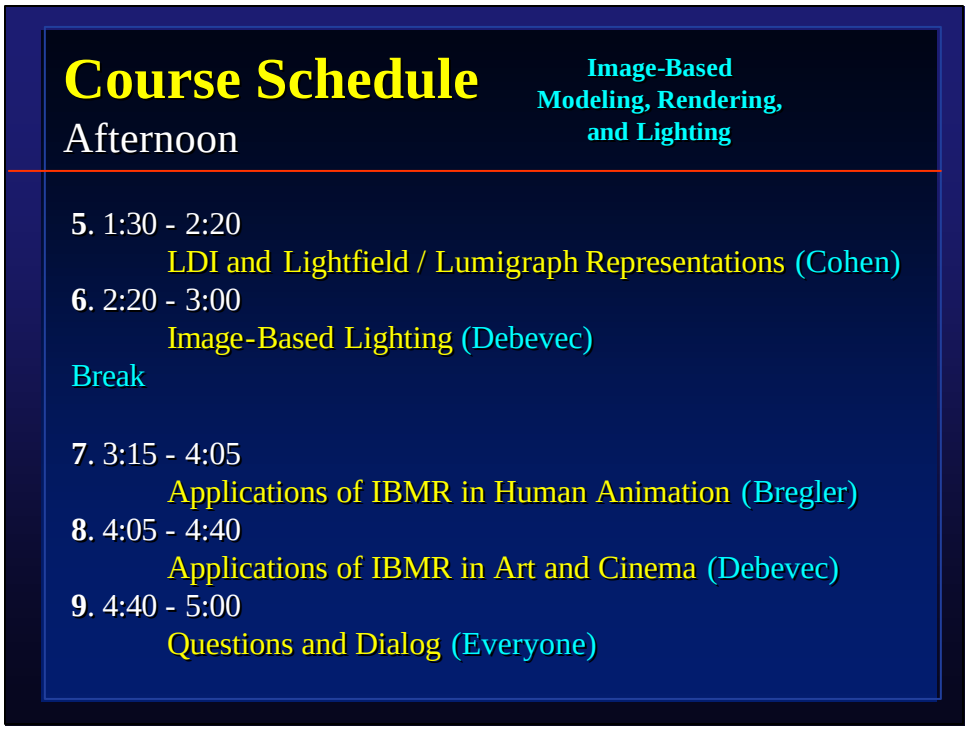
Image-Based Modeling, Rendering, and Lighting

Morning

1. 08:30 - 08:50
Introduction and Overview (Debevec)
2. 08:50 - 10:00
Fundamentals of Image Formation and Re-Use (Sillion)
Break

3. 10:15 - 11:00
Determining Geometry from Images (Szeliski)
4. 11:00 - 12:00
2D and 3D Image Warping (McMillan)
Lunch

http://www.cs.berkeley.edu/~debevec/IBMR2000


Course Schedule
Afternoon
Image-Based Modeling, Rendering, and Lighting
5. 1:30 - 2:20
LDI and Lightfield / Lumigraph Representations (Cohen)
6. 2:20 - 3:00
Image-Based Lighting (Debevec)
Break
7. 3:15 - 4:05
Applications of IBMR in Human Animation (Bregler)
8. 4:05 - 4:40
Applications of IBMR in Art and Cinema (Debevec)
9. 4:40 - 5:00
Questions and Dialog (Everyone)

Fundamentals of image formation and re-use

François X. Sillion

iMAGIS
INRIA, Laboratoire GRAVIR/IMAG
Grenoble, France.

In this section of the course we consider the basic questions of image content and re-use. In particular, we describe the physical information content of a typical image, based on the simplest camera model and the complex behavior of light in a 3D scene. After reviewing the important effects contributing to the appearance of an image, we focus on the issue of image re-use: when and how is it possible to re-use image fragments to assemble a plausible new view of a scene? Several techniques involving the re-use of synthetically generated images to accelerate the display of a complex synthetic scene are discussed.

What is an image?

The very notion of image is not well defined, especially in the context of computer graphics. Here we focus on the usual notion of an image, such as a typical photograph of a real scene, or a synthetic image viewed on a computer screen. Furthermore, we consider only digital images, composed of a finite set of points. Thus the image is a set of colored points (pixels), usually arranged on a rectangular grid. More elaborate “images”, containing even more information (such as depth or multiple samples) and resulting from the application of a computer process will be considered later in the course.

The image as a set of radiance samples

When the image represents a view of a three-dimensional environment, each pixel can be thought of as representing the “appearance” of a particular portion of this environment. A typical model of the image generation process considers a pinhole camera placed in the scene (see Figure 1). In such a case, each point on the image plane defines, together with the optical center of the camera, a direction in space. Leaving aside the whole issue of the camera’s radiometric and colorimetric response, we can consider for now that we record at each pixel the amount of incident light from the corresponding direction.

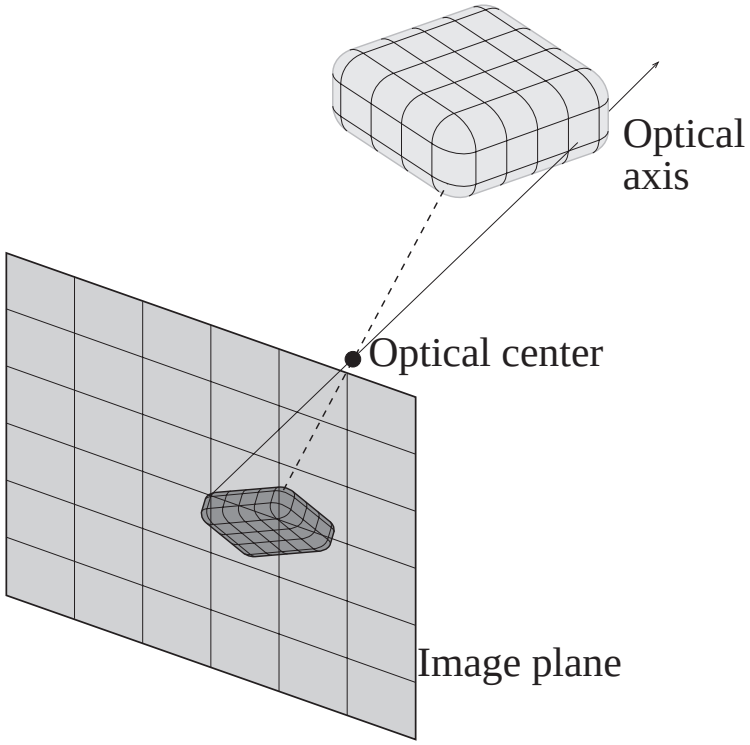


FIGURE 1: Pinhole camera model.

In terms of physical units, the relevant quantity to which our eye or photographic sensors are sensitive is called *radiance*, with units of $W/m^2/sr$, and measures the radiated power received per unit area and per unit incident solid angle (Figure 2).

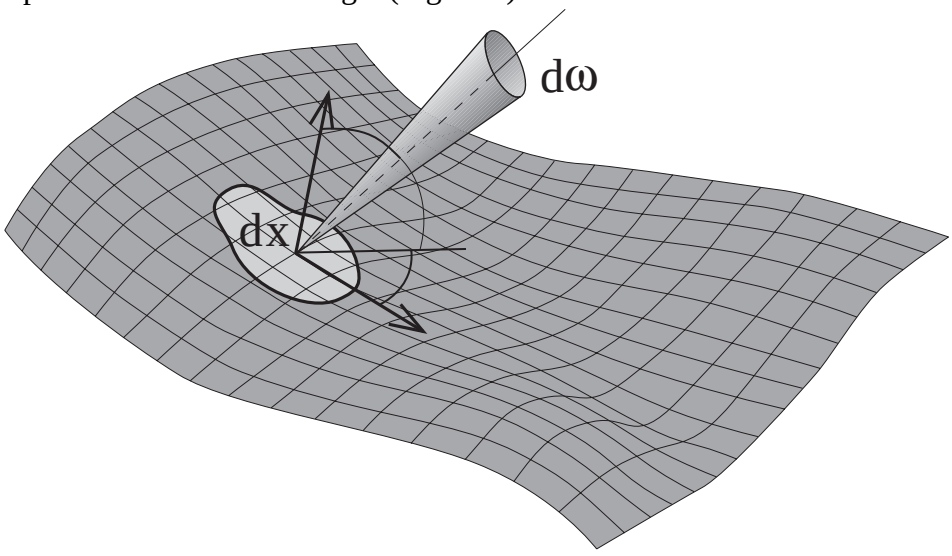


FIGURE 2: Radiance is the power received per unit area (dx) per unit solid angle (dω).

Finally, we define an image (of a real or synthetic scene) as *an array of radiance samples*. Of course this is an ideal view, assuming that we can record or represent radiance directly. This turns out to be difficult in many cases: for synthetic images where it may seem obvious to just record the result of a lighting formula, the problem may arise from the lack of well-established data formats to represent radiance images. For real images, we shall see that measuring radiance is difficult because of the very high dynamic range present in most images, which is almost always degraded by the capture processes.

The above definition actually assumes a monochromatic light distribution. Color variations must be represented by combining multiple radiance samples (for different wavelengths) at each pixel. Although most popular image formats simply use RGB color representations, more accurate color models are clearly needed for the most elaborate image-based techniques such as image re-lighting.

What are wee seeing in an image?

We just decided to consider an image as a set of incident radiance samples. We now need to further study where this incident radiance came from, in order to understand what we are seeing in the image. Radiance has a very nice conservation property, namely that

> *«In the absence of interaction between light and the medium in which it travels, the radiance incident on a surface from a given direction equals the radiance leaving the surface which is visible in that direction."*

In other words, radiance is conserved along its path through a non-participating medium. Examples of participating media include water and other dense fluids, smoke or fog. We can safely assume for now that we are not concerned with such media. The radiance conservation property tells us that our image can also be understood as a set of *outgoing* radiance samples. Each pixel records the outgoing radiance leaving the surface visible in its associated direction (towards the direction of the camera).

If we consider again the idealized case of the pinhole camera, the relationship between the three-dimensional location of a point and the pixel onto which it projects is expressed by a projective transform. This is best described with matrices, since the relationship between a 3D point $\begin{pmatrix} x_g & y_g & z_g \end{pmatrix}$ in world coordinates and its corresponding image point $\begin{pmatrix} x_i & y_i & z_i \end{pmatrix}$ is given by

$$\begin{pmatrix} x_i w_i \\ y_i w_i \\ z_i w_i \\ w_i \end{pmatrix} = P \cdot \begin{pmatrix} x_g \\ y_g \\ z_g \\ 1 \end{pmatrix}$$

The 4x4 matrix P describes the mapping using a fourth coordinate w_i to account for perspective division. P can be further decomposed into a euclidean transformation (defining the camera's reference axes with respect to the world), a projection transformation (defining the perspective parameters), and an affine transformation (defining the actual mapping to pixels, taking into account image resolution and possible distortions). These are discussed in more detail later.

Note that we associate a depth value z_i to a point in the image. This is common practice in computer graphics, since this "image-space" depth can be use to solve for visibility as in the depth buffer algorithm. Computer vision practitioners ignore this depth value, since it not readily available with images, and therefore use only 3x4 matrices.

The implications of this simple camera model are that

- If the "depth" of the visible surface at a pixel is known, the original 3D point can be recovered by applying the inverse transform P^{-1}.
- Otherwise, it is possible in certain cases to reproject the scene onto a different camera, without explicitly reconstructing the 3D scene, by a combination of projective maps. Depth is then approximated for instance by assuming the scene lies "close to" a plane in 3D. This is discussed later in the course.

Interactions of light and matter

We have now established that each pixel in our image records the radiance leaving an associated surface point, which happens to project onto that particular pixel. Understanding exactly what this radiance is requires a discussion of light and its interactions with a three-dimensional scene. This section introduces a number of principles governing the distribution and behavior of light, and discusses the implications on image-based rendering.

The behavior of light is governed by the following equation, called *the rendering equation* in the Computer Graphics field, after Kajiya [2].

$$L(x,\theta) = L_e(x,\theta) + \int L_i(x,\phi)\ \ \rho(x,\theta,\phi)\ \ d\omega$$

This equation states that $L(x,\theta)$, the radiance leaving a point (x) in a direction (θ) is the sum of

- the radiance $L_e(x,\theta)$ intrinsically emitted at this point (for a point on a light source),
- the radiance reflected at this point. That second term is an integral over all possible incident directions on x, where the incident radiance is weighted by a reflectance function ρ.

Strictly speaking, this equation is written for a singe wavelength, and involves spectral radiance. Several of the involved quantities vary with the wavelength λ, producing a particular spectral distribution for the total radiance field.

Each of the components of the rendering equation is described below in more detail.

Light sources

Light sources in the scene are objects that emit light by themselves, independently of their environment. Typical light sources include artificial light fixtures (in indoor scenes, or at night) and natural light sources such as the sun or sky. While computer graphics software often employs point light sources, almost all real light sources have a non-negligible extent in 3D. This is important because they create *soft shadows* or *penumbra regions*, where the light source is only partially visible.

Light sources are characterized by their location, size and shape, directional emission patterns, and spectral properties (in other words, their color). The color of a light source is described by the spectral variations of $L_e(x,\theta)$. As we shall see later in the course, light sources are particularly difficult to model from images, because of their very large radiance values.

Reflectance properties

The reflective behavior of surfaces is described by their *Bi-directional Reflectance Distribution Function* (BRDF), denoted by ρ in the rendering equation. This reflectance function basically expresses the probability that light arriving from a given direction will be reflected in another direction. Extreme, idealized cases for the BRDF are

- Ideal diffuse (Lambertian) reflectors. Light is reflected uniformly in all directions, the BRDF is constant.
- Ideal specular (mirror) reflectors. Light is reflected only in the Descartes-Snell direction, i.e. with the angle of reflection equal to the angle of incidence. The BRDF is a Dirac distribution.

However almost all real materials exhibit a more complex behavior, with a directional character resulting from surface finish and sub-scattering under the surface. The intermediate (directional but not specular) behavior is sometimes called “glossy”, or directional-diffuse in Computer Graphics (Figure 3).

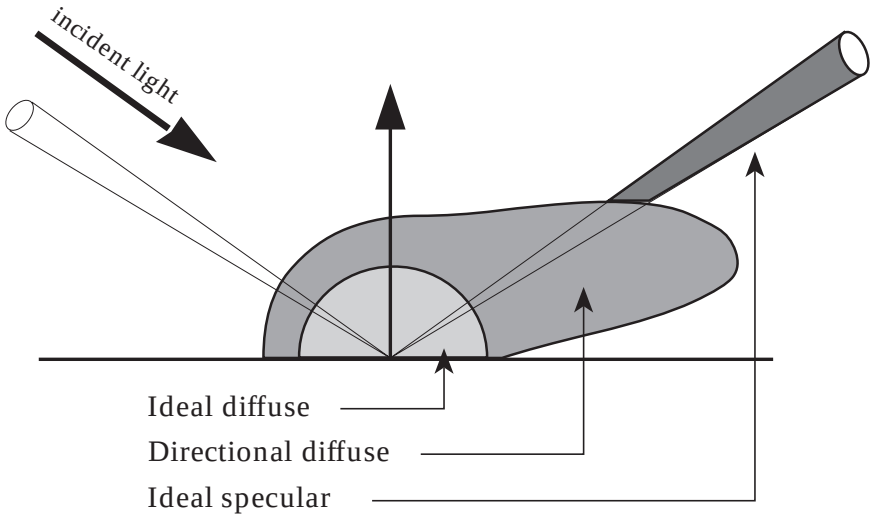


FIGURE 3: Three components of a BRDF.

Each component of a BRDF can have its distinct spectral variations, resulting in a number of different colors associated to a material: A base color corresponding to the ideal diffuse component, which will appear identical for all viewing directions, a glossy color (typically less saturated) for the view-dependent highlights, and an ideal specular color for mirror reflections.

Recovering the material properties from an image is a difficult challenge since these components are not available separately but only in the combined reflected radiance. Even assuming a carefully controlled illumination (e.g. from a single, parallel light source of known power and spectral properties) there are too many remaining unknowns unless the shape of the visible object is known. This condition is rarely met in practice, although an interesting exception is the image-based BRDF measuring device currently under development at Cornell University in the U.S.A. [12].

Image-based rendering, or the idea that radiance samples from an image may be re-used to create a new view, implicitly assumes an ideal diffuse behavior: only in that case will the radiance leaving an object be identical for all viewing directions. This assumption often works well for a wide range of materials and viewing directions. However, it should be explicitly recognized as an assumption, which is sometimes impossible to meet, as in the case of very glossy or ideal

specular objects. Highlights and reflections normally appear to move on the surface of the objects when the viewer moves, an effect that can not be recovered by simply re-using radiance samples from an original image. Consider for instance the image of a window reflecting a building across the street: when viewed from a direction other than the original one from which the picture was taken, the reflection will appear in the wrong place. Even worse, this reflection will not move properly in a sequence of images, distracting the viewer from the illusion of reality. In such cases, more elaborate processing is required, involving either some knowledge of the scene geometry or more radiance samples from nearby views.

Global illumination

The rendering equation shows that the radiance leaving a surface (for instance in the direction of the camera) is influenced by the radiance leaving other surfaces, creating a very complex set of inter-dependencies. The illumination of any surface point in the scene is potentially affected by the illumination of every other point, an effect captured in the name “global illumination.”

Global illumination accounts for all indirect light in a scene, and is responsible for the subtle exchanges of light and color between surfaces. Unfortunately, it is difficult and costly to simulate in synthetic images, requiring the use of Monte Carlo simulation or radiosity techniques [10]. On the other hand, of course, it is included “for free” in any real image acquired by photography! This is actually one of the major advantages of image-based rendering, as opposed to conventional model-based rendering; all natural illumination effects are present in the original set of images.

Taking into account the effects of global illumination amounts to considering each object as a (secondary) light source, because it reflects some light into the scene. This means that each object is subjected to a very complicated incident light distribution, coming from all directions with varying intensity and spectral characteristics. Therefore, the “advantage” of obtaining global illumination effects naturally in real images can sometimes be a curse for image-based modeling. Imagine for instance trying to extract material properties (BRDFs) from images. Even assuming the geometry of the object is known, the incident light distribution is very difficult to model!

Re-using images

The basic premise of image-based rendering, or “rendering from images”, is that image portions can be re-used to create new views of a scene. For instance, a projective mapping can be applied to reposition all pixels into a new image and simulate a new perspective of a nearly planar scene.

This idea is most often associated with “real” images, as opposed to synthetic ones. After all, if the original image (or set of images) is obtained with computer graphics techniques from a 3D model, it should be possible to also generate new views in the same way. However this vision is too simple, and even synthetic imaging can greatly benefit from image-based techniques.

For instance, it can be prohibitively expensive to synthesize new images for all desired viewpoints. An example of this case is virtual reality applications in which a sustained frame rate of more than 30 frames per second according to the tracked location and orientation of the viewer is a necessity. If images can only be rendered (or acquired from a network connection) at a slower rate, image-based techniques can be used with profit to fill in the missing frames. Another potential use of image-based representations is to replace very complex (and costly to render) models, in a level-of-detail approach.

We review here some of the proposed approaches for image re-use, in the context of synthetic imaging. This particular context is interesting because it implicitly assumes that all the relevant information (such as 3D model, material properties, lighting models and simulation) is available if needed. This makes it easier to understand differences between algorithms, in terms of what information they actually use, or how often they use it.

Perspective image caching

Let us consider what happens to the view of a user moving about in a scene. The apparent motion of objects in the image is obviously related to their distance to the viewer. Very distant objects appear to stay still, distant objects simply move in the image without much change of appearance, whereas nearby objects undergo the most severe changes, possibly exposing new areas or faces.

Therefore, not all parts of the image need to be refreshed at the same rate, and it is conceivable to render portions of the image separately for later compositing, and re-use some portions longer than others.

Regan and Pose [5] introduced this idea in a rendering system where the scene was segmented according to the distance to the viewer. Partial views of the scene are then rendered at appropriate rates, and always composited at the display rate.

Schaufler and Stürzlinger [7], and Shade *et al* [8] improved the idea by using a hierarchical set of images. In both their systems, the scene is encoded in a hierarchical data structure (BSP tree). For the first image, each node of the BSP tree is equipped with an image representing a view of the

corresponding subtree from the original viewpoint. For subsequent frames, rendering proceeds hierarchically, selecting at each node whether to use the stored image (and terminate the tree traversal locally) or to recursively draw children nodes. Therefore, the hierarchical tree is only traversed down to the appropriate representation, necessary to meet a quality criterion. A typical criterion bounds the disparity error between the re-used image and an image of the true geometry.

When a decision is made to use a pre-stored image, this image is texture-mapped on a billboard polygon. This explicit positioning in 3D means that the image undergoes the current view transformation which is a projective map.

Affine warping

Carrying further the idea that image fragments can be independently rendered at different rates and composited in the final view, the *Talisman* architecture was proposed in 1996 by Torborg and Kajiya [11]. The proposed architecture of a graphics subsystem combines a renderer and a warper/compositer, both operating on image fragments called *sprites*, representing individual objects or groups of objects. The compositer combines all sprites in real time at each frame, after subjecting each sprite to its own affine transformation to obtain the best approximation to the desired image. When no suitable affine transform can be found to produce correct results, a request is made to the renderer to produce a new sprite with the associated objects. The renderer therefore operates on-demand, at a slower rate than the display rate, and a controlling program makes decisions about which sprites should be updated at each frame, and which affine transforms should be used.

Lengyel and Snyder [3] studied the suitability of the affine transformation (as opposed to the more complete projective map) and found that it is sufficient in many cases. It is therefore a mapping of choice because it is cheaper to compute than a full projective map (no division is required). The same authors also describe a generic set of criteria to decide on the best affine warp, and show that shadows and highlights can be treated as independent sprites with their own refresh rate and transforms.

The resulting architecture is very flexible and appears quite promising. Until dedicated graphics hardware becomes available, however, its systematic use of compositing operations makes it quite expensive if run on conventional graphics cards. Note that in this organization of rendering, no explicit depth information is carried with, or extracted from, the image fragments. Instead, a high-level warping function is used for each sprite.

3D warping with points

When more information is available with the image, in the form of depth values for some or all the pixels, direct warping becomes possible. As we shall see later, pixels with depths can either be reprojected back in the 3D world or warped to the new image directly. In both cases, the new image can be reconstructed either by rendering points, splats, or polygons based on these pixels.

If many pixels have depth information, a dense mesh can be built from them, and simplified as desired according to any user-defined criteria. Note that this mesh can either be built in 3D space, a strategy used to construct *impostors* [9], or in the new image space to aid in the image reconstruction.

Layered warping

A clever warping technique has recently been proposed, which combines the simplicity of planar textured billboards (as in image caches) with the greater accuracy of full 3D warping. Schaufler [6] renders objects from a single source image using multiple stacked polygons at different locations in 3D. Using the available opacity test to perform a simple depth test at display time, the appropriate set of pixels is automatically selected for each layer, corresponding to a range of depth values in the input image. The number of layers can be dynamically adjusted to optimize the cost/quality tradeoff (Figure 4). Meyer and Neyret used a similar idea to render complex procedural 3D textures along the surface of objects [4].

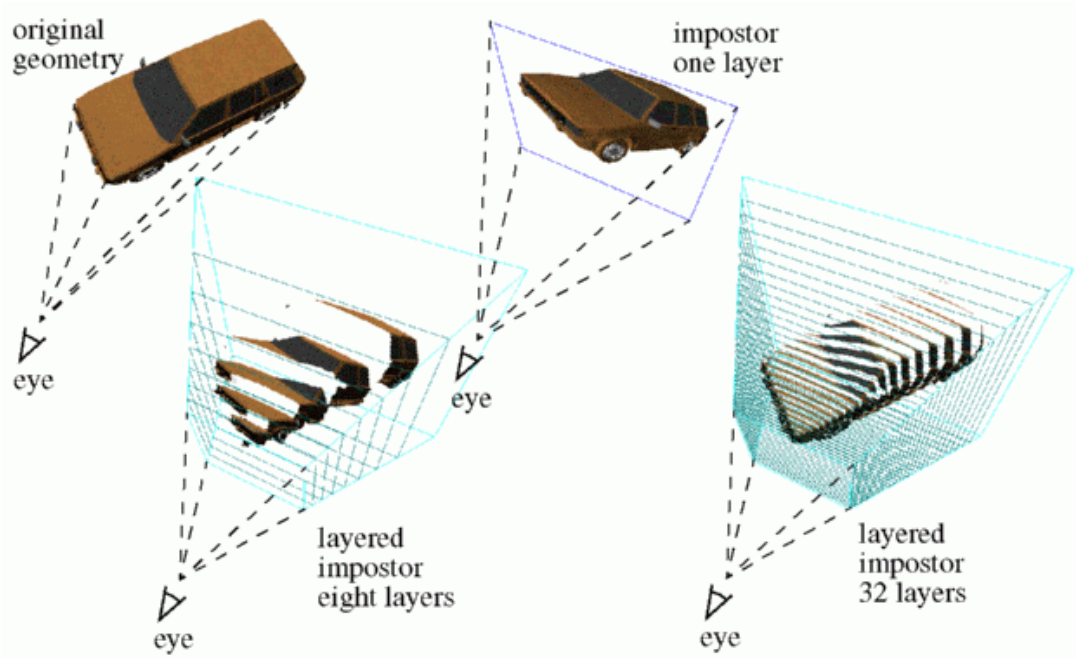


FIGURE 4: Layered rendering of a depth image (© Schaufler, 1998).

Developing a complete strategy for image re-use

Each of the above approaches has been shown to provide a substantial benefit in some practical situations. Yet it remains difficult to propose a general-purpose strategy for optimal image re-use in computer graphics applications. The Talisman approach is clearly well thought-out, but assumes all components of the suggested architecture are present. Algorithms running on today's hardware are also needed for current and upcoming applications.

We outline here a possible architecture for a display system, which is currently under development in a joint research project between MIT and iMAGIS. The proposed system employs a number of image-based acceleration techniques to dynamically optimize image quality and display speed.

First, a set of images of the model is created, most probably off-line in a pre-processing step. These images, representing well-chosen portions of the model, can then be used to generate *meshed impostors*, which will replace the underlying geometry whenever possible. The use of images to create impostors has two favorable properties: first, it automatically selects visible geometry with respect to a given viewpoint. Second, it samples the model at a chosen resolution, which can be adapted to the viewing conditions.

At display time, a segmenting process selects a set of 3D models and impostors to draw, based on a frame drawing time budget and a set of error estimates associated with impostors. Finally, a fraction of the frame drawing time is reserved for *dynamic updates* to the approximate impostor representation. Such updates can be performed on one or several impostors, using for instance the layered billboard approach mentioned above.

While this architecture is only one of many possible, it appears to offer a number of benefits:

- it uses image-based impostors to select potentially visible elements from specified scene locations.
- it uses meshed impostors to simplify the information visible on the reference images according to user-specified criteria. These criteria can involve bounds on de-occlusion error [1] or any application-specific information such as points of interest, etc.
- it is not limited to the accuracy of the pre-computed image-based impostors, thanks to the dynamic update capability. In fact, the segmentation in impostors also serves as a selection mechanism for the portions of the scene whose image-based representation can be dynamically updated: selected impostor layers can be individually tagged for update, concentrating resources on the most important areas.

Summary

In this section we have discussed the formation of real and synthetic images: we saw that images can be thought of as sets of radiance samples, taken for a number of directions arriving at the camera. Alternatively, these samples represent the radiance leaving the visible objects in these directions. Radiance obeys the “rendering equation”, mixing the properties of the light sources, the reflective behavior of surfaces, and the global illumination exchanges of light. Therefore, the radiance leaving a surface in a given direction is potentially influenced by the entire scene, and varies (sometimes dramatically) with direction. This places strong constraints on the possibility of image re-use or image re-lighting.

We reviewed a number of techniques based on image re-use for the creation of new images, in an image synthesis application. All of these techniques create images of portions of the scene, and attempt to control their re-use either by adjusting their refresh rate or by modifying the mapping to screen from frame to frame.

Finally we outlined a possible strategy for image-based acceleration of a visualization application, combining the simplification power of image representations with the quality obtained by selective image re-generation.

References

[1] Xavier Decoret, Gernot Schaufler, François Sillion, and Julie Dorsey. *Multi-layered impostors for accelerated rendering.* Computer Graphics Forum, 18(3), proceedings Eurographics'99. September 1999.

[2] James T. Kajiya. *The rendering equation.* Computer Graphics, 20(4):143-150, August 1986. Proceedings of SIGGRAPH '86 in Dallas (USA).

[3] Jed Lengyel and John Snyder. *Rendering with coherent layers*. In Turner Whitted, editor, SIGGRAPH 97 Conference Proceedings, Annual Conference Series, pages 233-242. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[4] Alexandre Meyer and Fabrice Neyret. *Interactive volumetric textures*. In George Drettakis and Nelson Max, editors, Eurographics Rendering Workshop 1998, pages 157-168, Wien, July 1998. Eurographics, Springer. ISBN.

[5] Matthew Regan and Ronald Pose. *Priority rendering with a virtual reality address recalculation pipeline*. In Andrew Glassner, editor, Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994), Computer Graphics Proceedings, Annual Conference Series, pages 155-162. ACM SIGGRAPH, ACM Press, July 1994.

[6] Gernot Schaufler. *Per-object image warping with layered impostors*. In George Drettakis and Nelson Max, editor, Eurographics Rendering Workshop 1998, pages 145-156, June 1998. Springer Wien.

[7] Gernot Schaufler and Wolfgang Stürzlinger. *A three dimensional image cache for virtual reality*. In J. Rossignac and F. Sillion, editors, Computer Graphics Forum, 15(3). Proceedings Eurographics '96, pages 227-236. Blackwell, September 1996.

[8] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. *Hierarchical image caching for accelerated walkthroughs of complex environments*. In Holly Rushmeier, editor, SIGGRAPH 96 Conference Proceedings, Annual Conference Series, pages 75-82. ACM SIGGRAPH, Addison Wesley, August 1996.

[9] François Sillion, George Drettakis, and Benoit Bodelet. *Efficient impostor manipulation for real-time visualization of urban scenery*. In D. Fellner and L. Szirmay-Kalos, editors,

Computer Graphics Forum (Proc. of Eurographics '97), volume 16, pages 207-218, Budapest, Hungary, September 1997.

[10] François Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann publishers, San Francisco, 1994.

[11] Jay Torborg and Jim Kajiya. *Talisman: Commodity Real-time 3D graphics for the PC*. In Holly Rushmeier, editor, SIGGRAPH 96 Conference Proceedings, Annual Conference Series, pages 353-364. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[12] *Workshop on Rendering, Perception and Measurement*. Cornell University, April 1999. `http://www.graphics.cornell.edu/workshop`.

Fundamentals of Image Formation and Re-use

François X. Sillion

*i*MAGIS*
Grenoble, France

*A joint research project of CNRS, INRIA, INPG and UJF

What is an image?

Digital image (point samples)

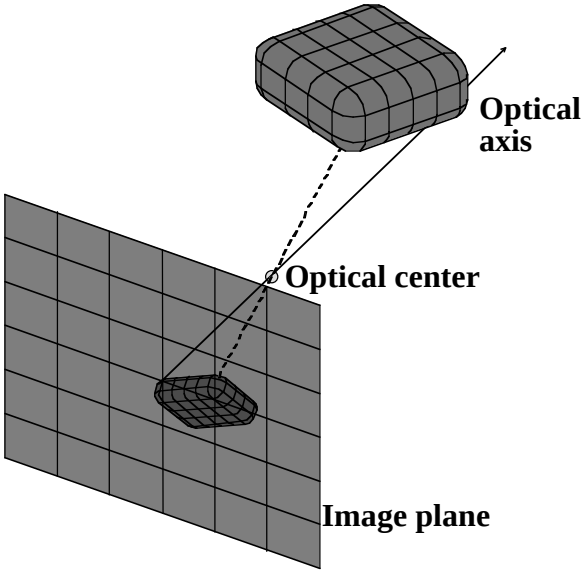
Ideal observation mechanism, captures *radiance* ($W/m^2/sr$)

Simple capture geometry

Goal: find out what can be done with an image.

iMAGIS

Pinhole camera model



iMAGIS

Light and matter

Light captured by the camera tells us about the object we see…

… and a lot more!

The *rendering equation*

$$L(x,\theta) = L_e(x,\theta) + \int L_i(x,\phi)\ \ \rho(x,\theta,\phi)\ \ d\omega$$

iMAGIS

Light sources

Extended shape (light fixtures, sky)

Complex near-field emission patterns

Difficult to model from images

- Dynamic range
- Visibility

$$L(x,\theta) = L_e(x,\theta) + \int L_i(x,\phi)\ \ \rho(x,\theta,\phi)\ \ d\omega$$

iMAGIS

Material properties

BRDF = Bidirectional Reflectance Distribution Function

Characteristic response of each material

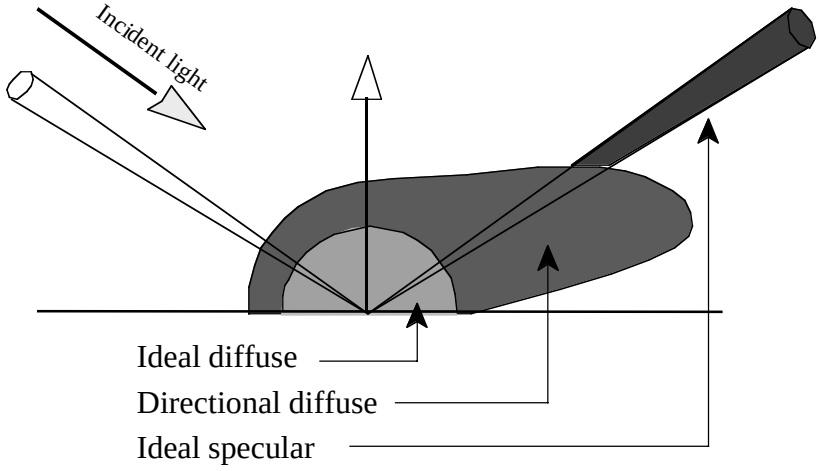
Spectral distribution defines object color(s)

Bi-directional nature of reflectance

$$L(x,\theta) = L_e(x,\theta) + \int L_i(x,\phi)\ \ \rho(x,\theta,\phi)\ \ d\omega$$

iMAGIS

Components of a BRDF model



iMAGIS

Global illumination

Visible objects are illuminated from all directions

Simulation is costly

Makes reflectance recovery difficult

$$L(x,\theta) = L_e(x,\theta) + \int L_i(x,\phi)\ \ \rho(x,\theta,\phi)\ \ d\omega$$

iMAGIS

IBR and specular reflections

iMAGIS

IBR and non-diffuse reflections

iMAGIS

Image-based relighting

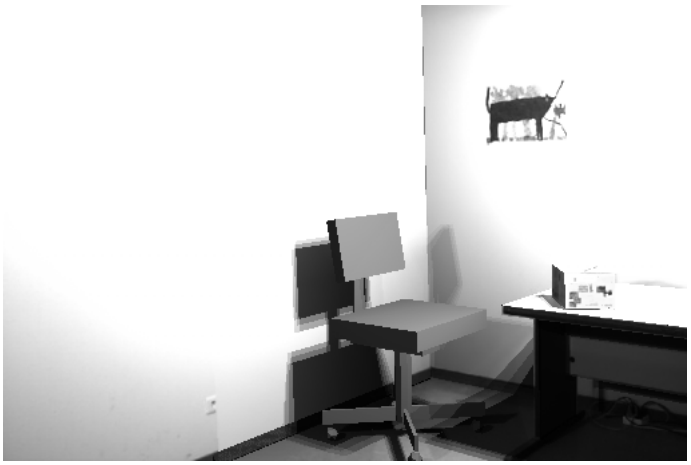
3D approach, requires
- Depth information
- Reflectance parameters
- Light source description

Radiance images are needed

What can we do without 3D lighting?

iMAGIS

Image-based relighting (3D)



Loscos and Drettakis, 1999

iMAGIS

Image-based relighting

3D approach, requires
- Depth information
- Reflectance parameters
- Light source description

Radiance images are needed

What can we do without 3D lighting?

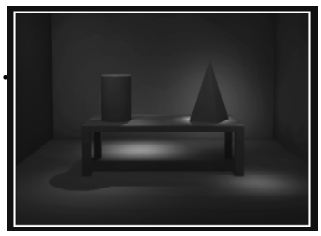
iMAGIS

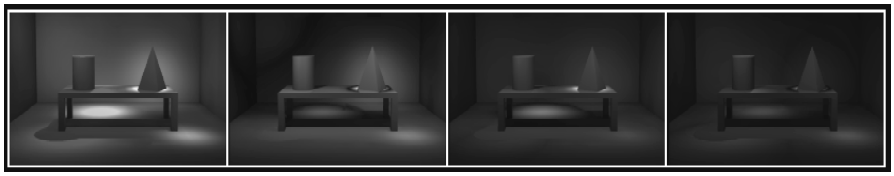
Image-based relighting (2D)

Linear combination of basis images

Use for lighting design: synthesize appropriate basis images using SVD.

Will work with radiance images





Dorsey et al. IEEE CG & A, 1995

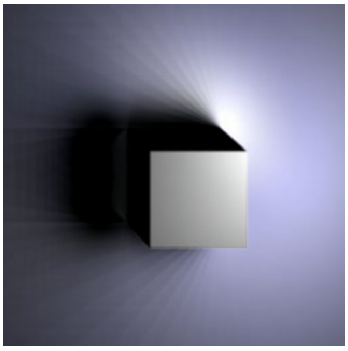
iMAGIS

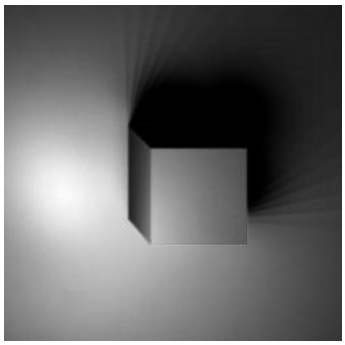
Image-based relighting (2D)

iMAGIS

Adding more variables to images

Add light source direction/position to the sampled variables





Wong *et al* 1997, EG rendering workshop

iMAGIS

Re-using images

- Move in a captured scene
- Special effects and compositing
- Synthetic images
  - Simplification, L.O.D.
  - Latency compensation (network, rendering power)

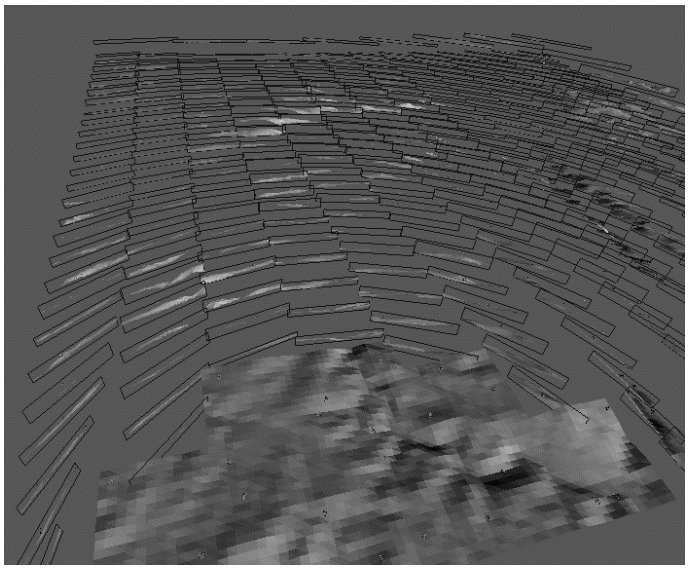
iMAGIS

Techniques for image re-use

- Image caching
  - Generate image fragments at their own refresh rate
  - Composite on the fly at user display rate
- Warping pixels
  - Render points
  - Interpolate (mesh)
- Layered warping

iMAGIS

Perspective image caching



Use 3D billboard and perspective mapping

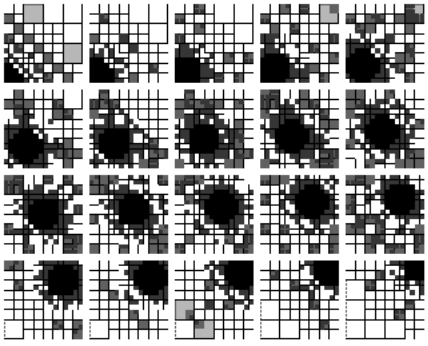
iMAGIS

Hierarchical image caching

Arrange scene in BSP

Render minimal subtree

Create images from subimages



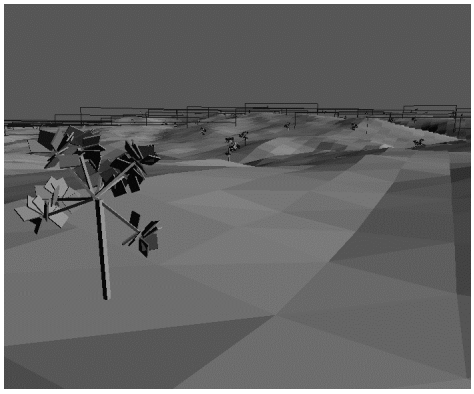
No update

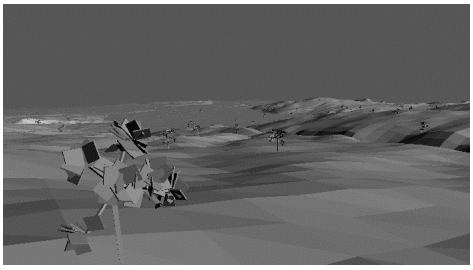
Move up in hierarchy

Re-generate image

iMAGIS

Hierarchical image caching





iMAGIS

Rendering With Coherent Layers

Jed Lengyel and John Snyder
Microsoft Research

Abstract

For decades, animated cartoons and movie special effects have factored the rendering of a scene into layers that are updated independently and composed in the final display. We apply layer factorization to real-time computer graphics. The layers allow targeting of resources, whether the ink and paint artists of cartoons or the graphics pipeline as described here, to those parts of the scene that are most important.

To take advantage of frame-to-frame coherence, we generalize layer factorization to apply to both dynamic geometric objects and terms of the shading model, introduce new ways to trade off fidelity for resource use in individual layers, and show how to compute warps that reuse renderings for multiple frames. We describe quantities, called *fiducials*, that measure the fidelity of approximations to the original image. Layer update rates, spatial resolution, and other quality parameters are determined by geometric, photometric, visibility, and sampling fiducials weighted by the content author's preferences. We also compare the fidelity of various types of reuse warps and demonstrate the suitability of the affine warp.

Using Talisman, a hardware architecture with an efficient layer primitive, the work presented here dramatically improves the geometric complexity and shading quality of scenes rendered in real-time.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Additional Keywords: sprite, affine transformation, image compositing, image-based rendering, Talisman

1 Introduction

The layered pipeline separates or *factor*s the scene into layers that represent the appearance of an object (e.g., a space ship separate from the star field background) or a special lighting effect (e.g., a shadow, reflection, highlight, explosion, or lens flare.) Each layer produces a 2D-image stream as well as a stream of 2D transformations that place the image on the display. We use *sprite* to refer to a layer's image (with alpha channel) and transformation together.

The layered pipeline decouples rendering of layers from their display. Specifically, the sprite transformation may be updated more frequently than the sprite image. Rendering (using 3D CG) updates the sprite image only when needed. Sprite transforming and compositing [Porter84] occur at display rates. The sprite transformation scales low-resolution sprites up to the display resolution, and transforms sprites rendered earlier to approximate their later appearance. In other words, the sprite transformation interpolates rendered image streams to display resolution in both space and time.

Layered rendering has several advantages for real-time CG. First, layered rendering better exploits coherence by separating fast-moving foreground objects from slowly changing background layers. Second, layered rendering more optimally targets rendering resources by allowing less important layers to be degraded to conserve resources for more important layers. Finally, layered rendering naturally integrates 2D elements such as overlaid video, offline rendered sprites, or hand-animated characters into 3D scenes.

As an architectural feature, decoupling rendering from compositing is advantageous. Compositing is 2D rather than 3D, requires no z-buffer, no lighting computations, no polygon edge antialiasing, and must handle few sprites (which are analogous to texture-mapped polygons) relative to the number of polygons in the rendered geometry. This simplicity allows compositing hardware to be made with pixel fill rates much higher than 3D rendering hardware. Our investigation demonstrates that the saving in 3D rendering justifies the extra hardware expense of a compositor.

Address: One Microsoft Way, Redmond, WA 98052-6399
Email: jedl@microsoft.com, johnsny@microsoft.com

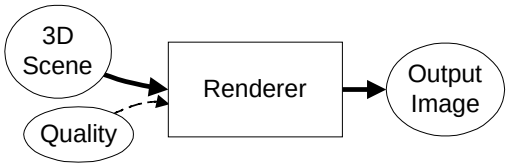


Figure 1: TRADITIONAL PIPELINE processes the entire scene database to produce each output image. The quality parameters for texture and geometry (such as level-of-detail) may be set independently for each object in the scene. However, the sampling resolutions in time (frame rate) and space (image resolution and compression) are the same for all objects in the scene.

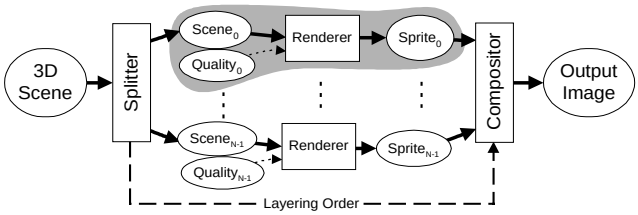


Figure 2: LAYERED PIPELINE partitions the scene into independent layers. A single layer's pipeline (highlighted at top) is similar to the traditional pipeline. By adjusting each layer's quality controls, the content author targets rendering resources to perceptually important parts of the scene. Slowly changing or unimportant layers are updated at lower frame rates, at lower resolution, and with higher compression.

The layered pipeline augments the set of traditional rendering quality parameters such as geometric level-of-detail and shading model (e.g., flat-, Gouraud-, or Phong-shaded), with the temporal and spatial resolution parameters of each layer. The *regulator* adjusts the quality parameters in order to achieve optimal quality within fixed rendering resources. The regulator dynamically measures both the costs of changing the quality parameters – how much more or less of the rendering budget they will consume – and the benefits – how much improvement or loss in fidelity will occur.

The specific contributions of this paper include extending the generality of factoring. While some authors have considered factoring over static geometric objects [Regan94, Maciel95, Shade96, Schaufler96ab], we consider dynamic situations and factoring over shading expressions (Section 2). We describe how to render using the layered pipeline (Section 3). We investigate different types of sprite transformations and show why an affine transformation is a good choice (Section 4). We discuss low-computation measures of image fidelity, which we call *fiducials*, and identify several classes of fiducials (Section 5). We add the spatial and temporal resolution of layers as regulation parameters and propose a simple regulator that balances them to optimize image fidelity (Section 6). Finally, we demonstrate that the ideas presented here enhance performance of the Talisman architecture by factors of 3-10, by using interpolated triple-framing or by regulating the heterogeneous update of sprite images (Section 7).

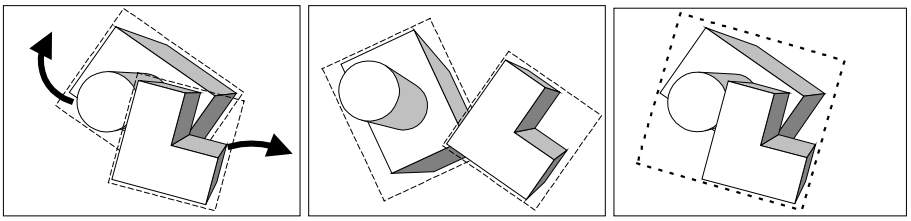


Figure 3: INDEPENDENT UPDATE depends on choice of factoring. The left and middle figures show how factoring the geometry into two separate layers allows each layer to be reused. The right figure shows a less effective partition.

1.1 The Layered Pipeline and Talisman

The Talisman reference hardware architecture was designed to support the layered pipeline (see [Torborg96] for details.) Rendering occurs within one 32×32 chunk at a time, so that z-buffer and fragment information for antialiasing can be stored on-chip. The resulting sprites with alpha channel are compressed and written to sprite memory. In parallel with 3D rendering, for every frame, the compositor applies an affine warp to each of an ordered set of sprites uncompressed from sprite memory and composites the sprites just ahead of the video refresh, eliminating the need for a frame buffer. Sprite composition is limited to the "over" operator [Porter84].

Although our experiments assume the Talisman reference architecture, the ideas can be usefully applied to traditional architectures. Layer composition can be emulated with rendering hardware that supports texture mapping with transparency by dedicating some of the pixel-fill rate of the renderer for sprite composition. This may be a good sacrifice if sprite rendering is polygon limited rather than pixel fill limited. Clearly though, Talisman is a superior layered pipeline in that sprite composition is "for free" (i.e., sacrifices few rendering resources) and very high speed (because of sprite compression and the simplicity of sprite composition in relation to rendering).[1]

1.2 Previous Work

To avoid visibility sorting of layers, alternative architectures use what are essentially sprites with z information per pixel [Molnar92, Regan94, Mark97]. Such systems are more costly in computation, bandwidth, and storage requirements since z must be stored and transmitted to a more complicated compositor. Z information is also difficult to interpolate and compress. We observe that z information per pixel is greatly redundant when used solely to determine a layering order. But such an ordering is necessary to ensure an antialiased result.[2] Our approach of factoring into layers allows warping per coherent object. It also avoids problems with uncovering of depth-shadowed information. Of course, sprites could store multiple z layers per pixel, a prohibitively costly approach for hardware, but one near to ours in spirit. Such a scheme stores all the layers within each pixel, rather than all the pixels for each layer.[3]

[Funkhouser93] adjusts rendering parameters to extract the best quality. We add sprite resolution and update rate to the set of regulated parameters and make photometric measurements rather than relying on *a priori* assignment of benefit to sampling rate. [Maciel95] takes a similar approach but use fiducials and impostor representations optimized for walkthroughs of static scenes.

Taking advantage of temporal coherence has been an ongoing theme in computer graphics [Hubschman81, Shelley82]. [Hofmann89] presents techniques for measuring how much camera movement is allowed before changes in the projected geometry exceed a given tolerance. [Chen93, Chen95] show how to take advantage of coherence between viewpoints to produce nearly constant cost per frame walkthroughs of static environments. [McMillan95] re-projects images to produce an arbitrary view.

Our use of temporal coherence is most similar to [Regan94], who observed that not all objects need updating at the display rate. Rather than factoring globally across object sets requiring a common update rate, our scheme factors over geometric objects and shading model terms, and accounts for relative motion between dynamic objects.

[Shade96] and [Schaufler96ab] use image caches and texture-mapped quadrilaterals to warp the image. This is conceptually similar to our work, but does not include the factoring across shading or the idea of real-time regulation of quality parameters. We harness simpler image transformations (affine rather than perspective) to achieve greater fidelity (Section 4.2). Our work also treats dynamic geometry.

Shading expressions [Cook84, Hanrahan90] have been studied extensively. [Dorsey95] factors shading expressions by light source and linearly combines the resulting images in the final display. [Guenter95] caches intermediate results. [Meier96] uses image processing techniques to factor shadow and highlight regions into separate layers which are then re-rendered using painterly techniques and finally composited. The novel aspects of our technique are the independent quality parameters for each layer and warp of cached terms.

2 Factoring

The guiding principle of our approach is to factor into separate layers elements that require different spatial or temporal sampling rates. This section discusses guidelines for manually factoring across geometry and shading, visibility sorting of layers, and annotating models with layer information.

2.1 Factoring Geometry

Geometry factoring should consider the following properties of objects and their motions:

1. *Relative velocity* – A sprite that contains two objects moving away from each other must be updated more frequently than two sprites each containing a single object (Figure 3). Relative velocity also applies to shading.
2. *Perceptual distinctness* – Background elements require fewer samples in space and time than foreground elements, and so must be separated into layers to allow independent control of the quality parameters.
3. *Ratio of clear to "touched" pixels* – Aggregating many objects into a single layer typically wastes sprite area where no geometry projects. Finer decompositions are often tighter. Reducing wasted sprite space saves rendering resources especially in a chunked architecture where some chunks can be eliminated, and makes better use of the compositor, whose maximum speed limits the average depth complexity of sprites over the display.

2.2 Visibility Sorting

Visibility sorting of dynamic layer geometry can be automated. We have implemented a preliminary algorithm for which we provide a sketch here. A full discussion along with experimental results is in progress [Snyder97].

To determine visibility order for layers containing moving geometry, we construct an incrementally changing kd-tree based on a set of constant directions. A convex polyhedron bounds each layer's geometry, for which we can incrementally compute bounding extents

[1] In a prototype implementation of Talisman, the compositor is planned to run at 320M pixels/second compared to 40Mps for the renderer.

[2] Penetrating z-sprites will have a point-sampled and thus aliased boundary where visibility switches.

[3] Post-warping of unfactored z-images also fails to address the case of independently moving objects.

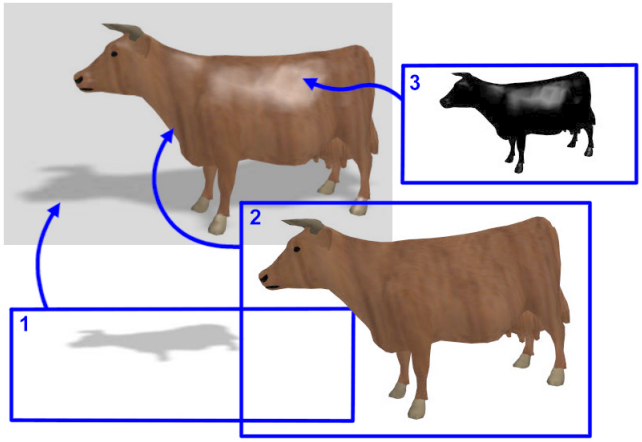


Figure 4: FACTORED SHADING EXPRESSION separates shadow, diffuse, and specular terms. In this example, the shadow and specular sprites are both computed at 25% (50% in x and y) of the display resolution. The shadow sprite modulates the color. The specular sprite adds to the output without changing alpha.

in each direction. The kd-tree quickly determines the set of objects that can possibly occlude a given object, based on these extents. Using this query, the visibility sort computes an incremental topological sort on the strongly connected components (which represent occlusion cycles) of the occlusion graph. Strongly connected components must be temporarily aggregated in the same layer, since a priority ordering does not exist.[4]

2.3 Factoring Shading

Shading may also be factored into separate layers. Figure 4 shows a typical multipass example, in which a shadow layer modulates the fully illuminated scene, and a reflection layer adds a reflection (in this case the reflection is the specular reflection from a light.) Figure 5 shows a schematic view of the steps needed to create the multipass image. The shadow layer is generated from a depth map rendered from the point of view of a light. The reflection layer is generated from a texture map produced by a separate rendering with a reflected camera. The layers shown in the figure represent post-modulation images using the same camera. With traditional architectures, the three layers are combined in the frame buffer using pixel blend operations supported by the 3D hardware, as described in [Segal92].

Shadows and reflections may instead be separated into layers as shown in Figure 6, so that the blend takes place in the compositor rather than the renderer. We call these *shade sprites* in reference to shade trees [Cook84]. To take advantage of temporal coherence, highlights from fast moving lights, reflections of fast moving reflected geometry, and animated texture maps should be in separate layers and rendered at higher frame rates than the receiving geometry. To take advantage of spatial coherence, blurry highlights, reflections, or shadows should be in separate layers and given fewer pixel samples.

For reflections, the correctness of using the compositor is evident because the reflection term is simply added to the rest of the shading. More generally, any terms of the shading expression that are combined with '+' or 'over' may be split into separate layers. 'A + B' can be computed using 'A over B' and setting A's alpha channel to zero.

The separation of shadows is slightly more difficult. The shadowing term multiplies each part of the shading expression that depends on a given light source. Many such terms can be added for

[4] At least, an ordering does not exist with respect to hulls formed by the set of bounding directions, which is a more conservative test than with the original bounding polyhedra. Note that visibility order for aggregated layers is computed simply by rendering into the same hardware z-buffer.

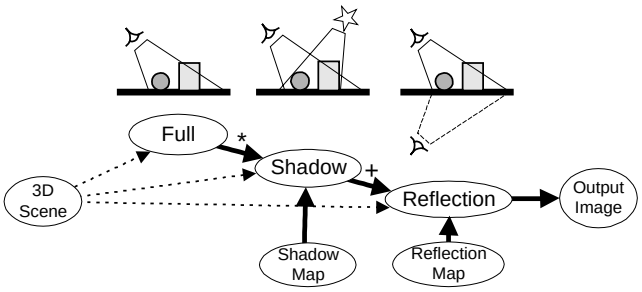


Figure 5: MULTIPASS RENDERING combines the results of several rendering passes to produce effects such as shadows and reflections. With a traditional architecture, the rendering passes are combined using blending operations in the 3D renderer (multiplication for shadow modulation and addition for adding reflections.)

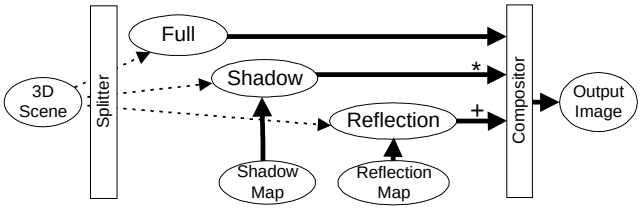


Figure 6: SHADE SPRITES are combined in the final composition phase to produce the multipass rendering. Each shading term may have different resolutions in space and time.

multiple shadowing light sources. We describe an approximation to multiplicative blending using the 'over' composition operator in an appendix.

Consider a simple example of a shading model with two textures and a shadow, $S(N \cdot L)(T_1 + T_2)$, where S is the shadowing term, N is the normal to the light, L is the light direction, and T_1 and T_2 are texture lookups. This shading model can be factored into three layers: S, $(N \cdot L)T_1$, and $(N \cdot L)T_2$, which are composited to produce the final image. The fact that this expression can be reordered and partial results cached is well known [Guenter95]. What we observe here is that each of these factors may be given different sampling resolutions in space and time, and interpolated to display resolutions.

As an aside, we believe shade sprites will be useful in authoring. When modifying the geometry and animation of a single primitive, the artist would like to see the current object in the context of the fully rendered and animated scene. By pre-rendering the layers that are not currently being manipulated, the bulk of the rendering resources may be applied to the current layer. The layers in front of the current layer may be made partially transparent (using a per-sprite alpha multiplier) to allow better manipulation in occluded environments. By using separate layers for each texture shading term, the artist can manipulate the texture-blending factors interactively at the full frame rate.

2.4 Model Annotation

The first step of model annotation is to break the scene into "parts" such as the base level joints in a hierarchical animated figure. The parts are containers for all of the standard CG elements such as polygon meshes, textures, materials, etc., required to render an image of the part. A part is the smallest renderable unit.

The second step is to group the parts into layers according to the guidelines described above. The distinction is made between parts and layers to allow for reuse of the parts, for example in both a shadow map layer and a shadow receiver layer.

The final step is to tag the layers with resource-use preferences relative to other layers in the scene. The preferences are relative so that total resource consumption can change when, for example, other applications are started (as discussed in Section 6).

3 Image Rendering

This section discusses how a layer's sprite image is created (i.e., rendered). Once created, the image can be warped in subsequent frames to approximate its underlying motion, until the approximation error grows too large. Although the discussion refers to the Talisman reference architecture with its 2D affine image warp, the ideas work for other warps as well.

3.1 Characteristic Bounding Polyhedron

The motion of the original geometry is tracked using a *characteristic bounding polyhedron*, usually containing a small number of vertices (Figure 7). For rigidly moving objects, the vertices of the characteristic polyhedron, called *characteristic points*, are transformed using the original geometry's time-varying transform. Nonrigidly deforming geometry can be tracked similarly by defining trajectories for each of the characteristic points. To group rigid bodies, we combine the characteristic bounding polyhedra, or calculate a single bounding polyhedron for the whole.

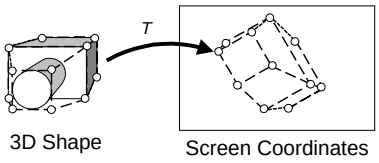


Figure 7: CHARACTERISTIC BOUNDING POLYHEDRON matches the shape of the geometry but has fewer vertices.

3.2 Sprite Extents

For a particular frame, there is no reason to render off-screen parts of the image. But in order to increase sprite reuse, it is often advantageous to expand the sprite image to include some off-screen area.

Figure 8a shows how clipping a sprite to the screen (solid box) prevents its later reuse because parts of the clipped image later become visible. In Figure 8b, the sprite extent (dashed box) has been enlarged to include regions that later become visible. The extra area to include depends on such factors as the screen velocity of the sprite (which suggests both where and how much the extents should be enlarged) and its expected duration of reuse.

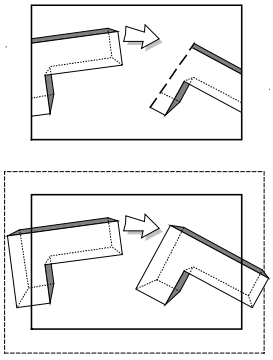


Figure 8: SPRITE EXTENTS enlarge the display extent to reuse sprites whose geometry lies partially off-screen.

3.3 Sprite Rendering Transformation

When creating a sprite image, we must consider a new transform in the pipeline in addition to the modeling, viewing, and projection transforms: a 2D affine transform that maps the sprite to the screen.

If T is the concatenation of the modeling, viewing, and projection matrices, a screen point p' is obtained from a modeling point p, by $p' = Tp$. For the sprite transformation, $p' = Aq$, where A is an affine transform and q is a point in sprite coordinates. To get the proper mapping of geometry to the display, the inverse 2D affine transform is appended to the projection matrix, so that $q = A^{-1} T p$ results in the same screen point $p' = A q = A A^{-1} T p = T p$ (Figure 9). The choice of matrix A determines how tightly the sprite fits the projected object. A tighter fit wastes fewer samples as discussed in Section 2.

To choose the affine transform that gives the tightest fit, we first project the vertices of the characteristic bounding polyhedron to the screen, clipping to the expanded sprite extent. Then, using discrete directions (from 2-30, depending on the desired tightness), we calculate 2D bounding slabs [Kay86]. Alternately, the slab directions may be chosen by embedding preferred axes in the original model, and transforming the axes to screen space.

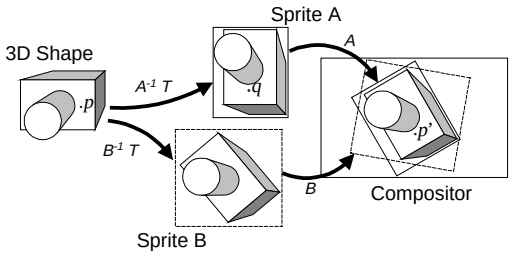


Figure 9: SPRITE RENDERING TRANSFORMATION maps the 3D shape into the sprite image. Affine transform B does not make the best use of image samples, while A fits the projected shape tightly.

Using the bounding slabs, we find the bounding rectangle with the smallest area (Figure 10). The origin and edges of the rectangle determine the affine matrix. Initially, we searched for the smallest area parallelogram, but found the resulting affine transformation had too much anisotropy.

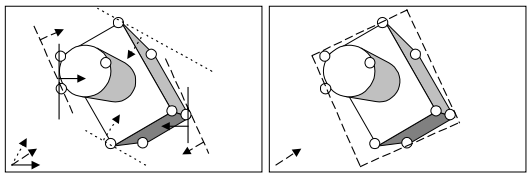


Figure 10: BOUNDING SLABS are obtained by taking the extremal values of the dot product of each slab direction with the characteristic points. A tight-fitting initial affine transform can be calculated by taking the minimum area rectangle or parallelogram that uses the slab directions.

3.4 Spatial Resolution

The choice of affine matrix A also determines how much the sprite is magnified on the display. Rendering using a sampling density less than the display's is useful for less important objects or for intentional blurring (Figure 11). The default is to use the same sampling density as the screen, by using the length in pixels of each side of the parallelogram from Section 3.3. See Figure 24 for an example of different sampling resolutions per sprite.

For a linear motion blur effect, the sprite sampling along one of the axes may be reduced to blur along that axis. The sprite rendering transformation should align one of the coordinate axes to the object's velocity vector by setting the bounding slab directions to the velocity vector and its perpendicular.

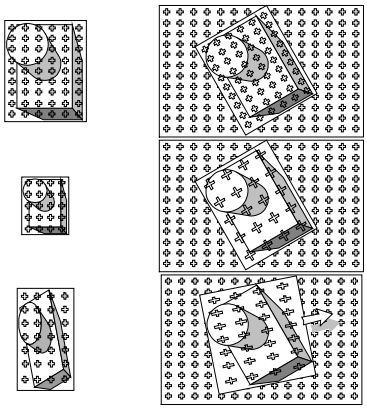


Figure 11: SPATIAL RESOLUTION is independent of the display resolution. The sampling density of the top sprite is the same as the screen. The middle sprite uses fewer samples than the screen, trading off pixel fill for blur. The bottom sprite aligns to the velocity vector and uses fewer samples along one dimension for a motion blur effect.

4 Image Warps

To reuse a rendered sprite image in subsequent frames, an image warp is used to approximate the actual motion of the object. We use the projected vertices of the bounding polyhedron (the characteristic points) to track the object's motion, as shown in Figure 12.

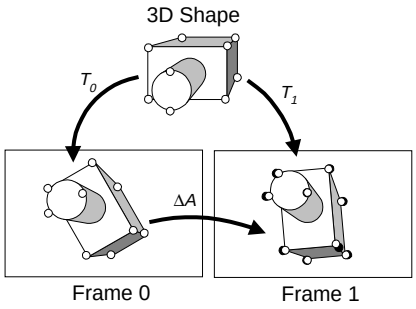


Figure 12: MATCHING CHARACTERISTIC POINTS on the 3D shape are projected to the screen to find a transform A that best matches the original points (white) to the points in the new frame (black).

To reuse images where objects are in transition from off-screen to on-screen, and to prevent large distortions (i.e., ill-conditioning of the resulting systems of equations), the characteristic bounding polyhedron is clipped to the viewing frustum, which may be enlarged from the display's as discussed in Section 3.2. The clipped points are added to the set of characteristic points (Figure 13) and used to determine an approximating sprite transformation as described below.

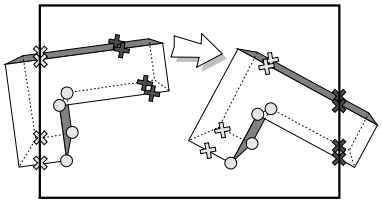


Figure 13: CLIPPED CHARACTERISTIC POLYHEDRON adds corresponding points introduced by clipping the characteristic polyhedron at the last-rendered and current frames.

4.1 Affine Warp

A 2D affine transform is represented by a 2x3 matrix, where the right column is translation and the left 2x2 is the rotation, scale, and skew.

$$A = \begin{bmatrix} a & b & t_X \\ c & d & t_Y \end{bmatrix}$$

Let P be the time-varying set of projected and clipped bounding polyhedron vertices, ignoring the z values and adding a row of 1's to account for the translation

$$P = \begin{bmatrix} x_0 & & x_{n-1} \\ y_0 & \cdots & y_{n-1} \\ 1 & & 1 \end{bmatrix}$$

where n is the number of points (at least 3 for the affine transform). Let $\hat{P}$ be the matrix of characteristic points at the initial time and P be the matrix at the desired time t. We solve for the best least-squares transform that matches the two sets of image-space points [Xie95].

In an affine transform, the x and y dimensions are decoupled and so may be solved independently. To solve $A\hat{P} = P$ at time t for the best A, in the least-squares sense, we use normal equations:

$$A\hat{P}\hat{P}^T = P\hat{P}^T$$
$$A = P\hat{P}^T\left(\hat{P}\hat{P}^T\right)^{-1}$$

The normal-equations technique works well in practice, as long as the projected points are reasonably distributed. Adding the clipped characteristic points ensures that $\hat{P}\hat{P}^T$ is not rank deficient. Much of the right hand side may be collected into a single vector K that may be reused for subsequent frames.

$$K = \hat{P}^T\left(\hat{P}\hat{P}^T\right)^{-1}$$
$$A = PK$$

To calculate K requires the accumulation and inverse of a symmetric 3×3 matrix.

4.2 Comparison of Warps

Clearly, other types of image warps can be used in place of the affine described above. In order to compare alternative image warps, we ran a series of experiments to

1. measure update rate as a function of maximum geometric error for various warps, and
2. measure perceptual quality as a function of update rate for various warps.

Each series involved the animation of a moving rigid body and/or moving camera to see how well image warping approximates 3D motion. We tried several types of rigid bodies, including nearly planar and non-planar examples. We also tried many animated trajectories for each body including translations with fixed camera, translations accompanied by rotation of the body along various axes with various rotation rates, and head turning animations with fixed objects.

The types of 2D image warps considered were

1. pure translation,
2. translation with isotropic scale,
3. translation with independent scale in x and y,
4. general affine, and
5. general perspective.

The fundamental simulation routine computes an animation given a geometric error threshold, attempting to minimize the number of renderings by approximating with an image warp of a particular type. A pseudo-code version is shown in Figure 14.

```
simulate(error-threshold, warp-type, animation)
{
    for each frame in animation
      compute screen position of characteristic points at current time
      compute transform (of warp-type) which best maps old
        cached positions to new positions
      compute maximum error for any characteristic point
      if error exceeds threshold
         re-render and cache current positions of characteristic points
      else
         display sprite with computed transformation
      endif
    endfor
    return total number of re-renderings
}
```

Figure 14: EXPERIMENT PSEUDOCODE shows steps used to compute update rates of various warps.

Ideally, we would like to compute approximations of each type that minimize the maximum error over all characteristic points, since this is the regulation metric. This is a difficult problem computationally, especially since the warping transformation happens at display rates for every layer in the animation. Minimizing the sum of squares of the error is much more tractable, yielding a simple linear system as we have already discussed. As a compromise, we simulated both kinds of error minimization: sum-of-squares and maximum error using an optimization method for L^∞ norms that iteratively applies the sum-of-square minimization, as described in [Gill81, pp. 96-98].

Further complicating matters, minimizing the error for perspective transformations is easier when done in homogeneous space rather than 2D space, again since the latter yields an 8×8 linear system rather than a difficult nonlinear optimization problem. We therefore included sum-of-square and maximum error methods for the first four (non-perspective) transformation types.

For perspective, we included sum-of-square minimization in homogeneous space (yielding a linear system as described above), maximum error in homogeneous space (using the technique of [Gill81]), and sum-of-square minimization in nonhomogeneous space (post-perspective divide), using gradient descent.[5] The starting point

[5] Minimization of the maximum nonhomogeneous error seemed wholly impractical for real-time implementation.

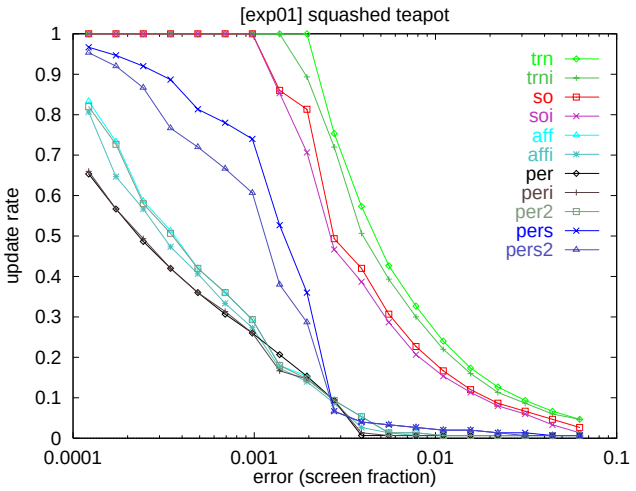


Figure 15: FLAT TEAPOT update-rate/error relations for the various warps show a surprisingly small difference between the affine and the perspective for a nearly flat object .

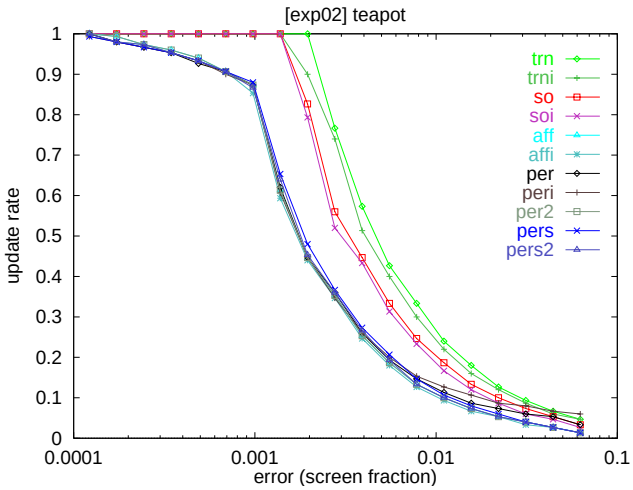


Figure 16: REGULAR TEAPOT update-rate/error relations show that affine and perspective are nearly indistinguishable.

for the gradient descent was the sum-of-squares-error-minimizing affine transformation.

We also included a perspective transformation derived using the method of [Shade96], in which objects are replaced by a quadrilateral placed perpendicular to the view direction and through the center of the object's bounding box. Our derivation projects the characteristic points onto this quadrilateral, bounds the projected points with a rectangle, and projects the corners of the rectangle to the screen. The perspective transformation that maps the old corners of the rectangle to their current locations is selected as the approximation. In yet another version, Shade's method is used as a starting point and then refined using gradient descent in nonhomogeneous space with the sum-of-square error metric.

Representative results of the first series of experiments are shown in Figure 15 and Figure 16. In both figures, the experiment involved a rotating and translating teapot which is scaled nearly flat[6] in Figure 15 and unscaled in Figure 16. Error thresholds ranging from 1/8 pixel to 64 pixels were used for each warp type/error minimization method, assuming an image size of 1024×1024, and the resulting rate of re-rendering measured via the simulation process described above. The meanings of the curve name keywords are as follows:

keyword	Warp type and minimization method
`trn`	translation, sum-of-square
`trni`	translation, max
`so`	translation with xy scale, sum-of-square
`soi`	translation with xy scale, max
`aff`	affine, sum-of-square
`affi`	affine, max
`per`	perspective, homogeneous, sum-of-square
`peri`	perspective, homogeneous, max
`per2`	perspective, nonhomogeneous, sum-of-square
`pers`	perspective, method of Shade
`pers2`	`pers`, followed by gradient descent

In the case of the flat teapot (Figure 15), note that the error/update curves cluster into groups – translation, translation with separate scale, Shade, affine, and perspective, in order of merit. Shade's method is significantly outperformed by affine. Note also that the sum-of-square error minimization is not much different than maximum error minimization for any of the warp types. The difference between perspective and affine is much less than one might expect in this case, given that perspective exactly matches motions of a perfectly flat object. Figure 16 (regular teapot) is similar, except that the clusters are translation, translation with separate scale, and all other warp types. In this case, perspective yields virtually no advantage over affine, and in fact is slightly worse towards the high-error/low update rate end of the curves for the homogeneous space metrics (`per` and `peri`).[7] This is because the homogeneous metric weights the errors unevenly over the set of characteristic points. The method of Shade is slightly worse than affine in this case.

Since geometric error is a rather indirect measure of the perceptual quality of the warp types, the second series of experiments attempted to compare the perceptual quality of the set of warps given an update rate (i.e., an equal consumption of rendering resources). We used binary search to invert the relation between error threshold and update rate for each warp type, and then recorded the same animation, at the same update rate[8], for various image warp approximations. Although subjective, the results confirm the merit of the affine transformation over less general transformations and the lack of improvement with the more general perspective transformation in typical scenarios.

4.3 Color Warp

Images can be "warped" to match photometry changes as well as geometry changes. For example, Talisman provides a per-sprite color multiplier that can be used to match photometry changes. To solve for this multiplier, we augment each characteristic point with a normal so that shading results can be computed (see Section 5.2). The color multiplier is selected using a simple least-squares technique that best matches the original color values of the shaded characteristic points to the new color values.

[6] The teapot, a roughly spherical object, was scaled along its axis of bilateral symmetry to 5% of its previous size.

[7] In the second series of experiments, the animations that used the homogeneous-weighted metric to determine an approximating perspective transformation looked visibly worse than those that used the simple affine transformation.

[8] The update rate is the fraction of frames re-rendered; this balances the total consumption of rendering resources over the whole animation.

5 Fiducials

Fiducials measure the fidelity of the approximation techniques. Our fiducials are of four types. Geometric fiducials measure error in the screen-projected positions of the geometry. Photometric fiducials measure error in lighting and shading. Sampling fiducials measure the degree of distortion of the image samples. Visibility fiducials measure potential visibility artifacts.

We use conservative measurements where possible, but are willing to use heuristic measurements if efficient and effective. Any computation expended on warping or measuring approximation quality can always be redirected to improve 3D renderings, so the cost of computing warps and fiducials must be kept small relative to the cost of rendering.

5.1 Geometric Fiducials

Let $\hat{P}$ be a set of characteristic points from an initial rendering, let P be the set of points at the current time, and let W be the warp computed to best match $\hat{P}$ to P. The geometric fiducial is defined as

$$F_{geom} = \max_i \left\| P_i - W\hat{P}_i \right\|$$

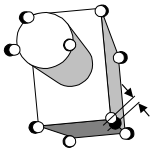


Figure 17: GEOMETRIC FIDUCIAL measures maximum pointwise distance between the warped original and current characteristic points.

5.2 Photometric Fiducials

We use two approaches to approximately measure photometric errors. The first uses characteristic points augmented with normals as described in Section 4.3 to point sample the lighting. Let $\hat{C}$ be the colors that result from sampling the lighting at the characteristic points at the initial time, and C be the sampled colors at the current time. Let W_C be the color warp used to best match $\hat{C}$ to C[9]. Then the shading photometric fiducial is defined to be the maximum pointwise distance from the matched color to the current color.

$$F_{photo} = \max_i \left\| C_i - W_C\hat{C}_i \right\|$$

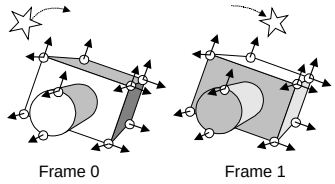


Figure 18: POINT-SAMPLED PHOTOMETRIC FIDUCIAL samples the shading at the initial and current characteristic points with normals.

Another approach is to abandon color warping and simply measure the change in photometry from the initial time to the current. Many measures of photometric change can be devised. Ours measures the change in the apparent position of the light. Let $\hat{L}$ be the position of the light at the initial time and L be its position at the current time (accounting for relative motion of the object and light). For light sources far away from the illuminated object, we can measure the angular change from $\hat{L}$ to L with respect to the object, and the change in distance to a representative object "center". For diffuse shading, the angular change essentially measures how much the object's terminator moves around the object, and the change in distance measures the increase or decrease in brightness. Light sources close to the object are best handled with a simple Euclidean norm. For specular shading, changes in the eye point can also be measured.

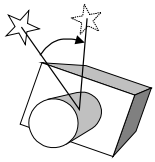


Figure 19: LIGHT SOURCE PHOTOMETRIC FIDUCIAL measures lighting change by the relative motion of light.

[9] Note that in architectures without color warping capability, W_C is the identity transform and we simply measure the maximum shading difference over all the characteristic points.

5.3 Sampling Fiducials

Sampling fiducials measure distortion of the samples in the image approximation. In Figure 20, both the geometric and photometric fiducials indicate high fidelity, but the image is blurry. The magnitudes of the singular values of the Jacobian of the image mapping function measure the greatest magnification and minification and the ratio measures the maximum anisotropy[10]. The affine warp has a spatially invariant Jacobian given by the left 2×2 part of the 2×3 matrix, for which the two singular values are easily calculated [Blinn96]. For transforms with spatially varying Jacobians, such as the perspective warp, the singular values vary over the image. In this case, bounds on the singular values over the input domain can be computed.

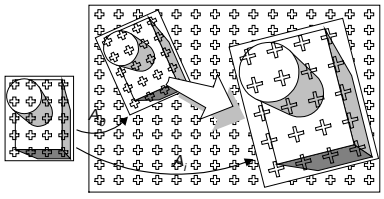


Figure 20: SAMPLING FIDUCIAL measures how the samples of a sprite are stretched or compressed.

5.4 Visibility Fiducials

Visibility fiducials measure potential visibility artifacts by tracking back-facing to front-facing transitions in the characteristic geometry (the simplified geometry makes these calculations tractable), and testing if the edges of clipped sprites become visible.

6 Regulation

A more complete treatment of regulation issues and directions may be found in [Horvitz96]. Our prototype regulator uses a simple cost-benefit scheduler and fiducial thresholds. The fiducial threshold provides a cutoff below which no attempt to re-render the layer is made (i.e., the image warp approximation is used). The regulator considers each frame separately, and performs the following steps:

1. Compute warp from previous rendering.
2. Use fiducials to estimate benefit of each warped layer.
3. Estimate rendering cost of each layer.
4. Sort layers according to benefit/cost.
5. Use fiducial thresholds to choose which layers to re-render.
6. Adjust parameters of chosen layers to fit within budget.
7. Render layers in order, stopping when all resources are used.

For a "budget-filling" regulator, the fiducial threshold is set to be small, on the order of a 1/1000 of the typical maximum error. All of the rendering resources are used in the attempt to make the scene as good as possible. For a "threshold" regulator, the threshold is raised to the maximum error that the user is willing to tolerate. This allows rendering resources to be used for other tasks.

Cost estimation [step 3] is based on a polygon budget, and measures the fraction of this budget consumed by the number of polygons in the layer's geometry. Parameter adjustments [step 6] are made to the sprite's spatial resolution using a budgeted total sprite size. This accounts for the rate at which the 3D rendering hardware can rasterize pixels.[11] Sprites that have been selected for re-rendering [step 5] are allocated part of this total budget in proportion to their desired area divided by the total desired area of the selected set. To dampen fluctuations in the regulation parameters which are perceptible when large, parameter changes are clamped to be no more than ±10% of their previous value at the time of last re-rendering. Note that factoring into many low-cost sprites allows smoother regulation.

[10] The filtering capabilities of the hardware limit the amount of minification and anisotropy that can be handled before perceptible artifacts arise.

[11] A global average depth-complexity estimate is used to reduce the budget to account for rasterization of hidden geometry. Note that the depth complexity of factored geometry in a single layer is lower than would be the case for frame-buffer renderings of the entire scene.

7 Results

For the results presented here, we assume we have an engine that will composite all of the sprite layers with minimal impact on the rendering resources (as in the Talisman architecture.) We track the number of polygons and the amount of pixel fill used for rendering, but disregard compositing.

Both of the sequences described below are intended to represent typical content. For scenes with a moving camera, the typical speedup is 3-5 times what the standard graphics pipeline is capable of producing.

7.1 Canyon Flyby

This 250-frame sequence (Figure 25) used 10 sprite layers. We interpolated using affine warps and regulated the update rate with a geometric fiducial threshold of 4 pixels. Each sprite layer was re-rendered only when the geometric threshold was exceeded. The fiducial threshold of 4 pixels may seem large, but is acceptable for this sequence since the ships are well separated from the rest of the world.

The average update rate was 19%, and the cost-weighted average (based on polygon count) was 32%. About a third of the total polygons were rendered per frame.

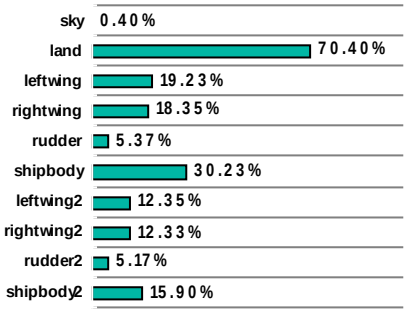


Figure 21: CANYON FLYBY AVERAGE UPDATE RATE for each sprite is the number of times each sprite was rendered divided by the number of frames in the sequence, 250

In the canyon flyby scene, the entire background was placed in one sprite. Parallax effects from the rapidly moving camera make this a poor layering decision that yields a high update rate for the landscape sprite. In contrast, the sky is rendered just once and then positioned with a new affine transformation each frame, and parts of the ships are updated relatively infrequently (5-30%).

7.2 Barnyard

The barnyard sequence (Figure 26) was chosen as an example in which a camera moves through a static scene[12]. This is a difficult case, because the eye is sensitive to relative motion between static objects. Approximation errors in sequences in which objects already have relative motion are far less perceptible (e.g., the ships in the canyon flyby above.) Even with this difficult case, our interpolation technique is dramatically better than triple framing.

The scene is factored into 119 standard layers, 2 shadow map layers, and 2 shadow modulation layers. The contiguous landscape geometry was split into separate sprites. As an authoring pre-process, the geometry along the split boundaries was expanded to allow for small separation of the warped sprites (the "seam" artifact.) This is similar to the expansion of the geometry along the split used by [Shade96]. More work is needed for automatic determination of the geometric expansion and better blending along the split boundaries.

The resource-use graph in Figure 22 shows three types of regulation. Simple triple framing, in which a frame is rendered and then held for three frames, requires the most resources. Interpolated triple-framing requires the same amount of rendering resources, but interpolates through time using the warp described in Section 4.1 – the sprites are still rendered in lock-step but track the underlying characteristic geometry between renderings. The rest of the curves show threshold regulation with increasing geometric error threshold, from 0.1-0.8 pixels – the sprites are updated heterogeneously when the geometric error threshold is exceeded. The graph is normalized to the resource use of triple-framing.

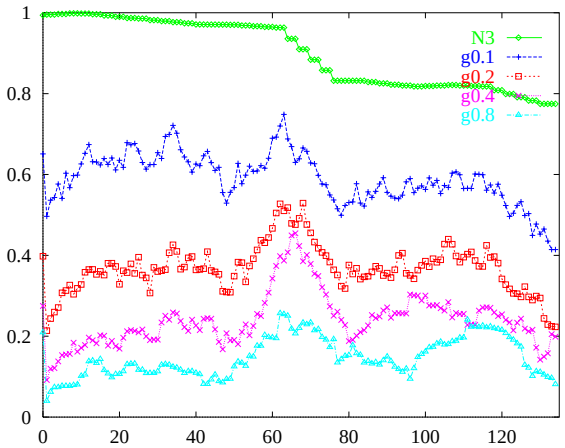


Figure 22: BARNYARD RESOURCE USE shows polygon counts as a 15-frame moving average. Pixel fill resource use is analogous. The top line is the triple-frame rendering. The lines below use threshold-regulation with increasing geometric threshold. As expected, as the pixel error tolerance goes up, the resource use goes down.

In the resulting animations, the most dramatic improvement in quality comes when the interpolation is turned on. The rest of the animations are nearly indistinguishable and use a fraction of the resources by rendering only those sprites whose geometric error exceeds the threshold.

Figure 23 shows the average pixel error for the same sequences shown in Figure 22. Each of the threshold-regulation sequences uses an error threshold smaller than the maximum error observed when triple-framing. Note that threshold-regulation is not the typical case and is shown here simply to demonstrate the performance advantage over the traditional graphics pipeline. Typically, all of the rendering resources are used to make the picture as good as possible.

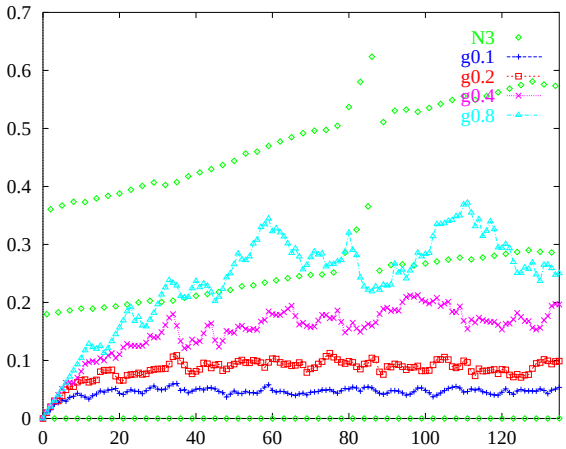


Figure 23: BARNYARD PIXEL ERROR shows average sprite pixel error per frame. Note that the triple-frame error is a saw-tooth that starts at 0, jumps to ½, and then jumps to full error value. The other curves oscillate below the given geometric threshold value.

[12] In the longer film from which the barnyard sequence is taken, many of the shots have fixed cameras. In these shots, only the main characters need to be updated, so the performance gain is extremely high. This is similar to the time-honored technique of saving the z-buffer for fixed camera shots.

8 Conclusions and Future Work

3D scenes should be factored into layers, with each layer having the proper sampling rates in both space and time to exploit the coherence of its underlying geometry and shading. By regulating rendering parameters using feedback from geometric, photometric, visibility, and sampling fiducials, rendering resources are applied where most beneficial. When not re-rendered, image warping suffices to approximate 3D motion of coherently factored layers. An affine warp provides a simple but effective interpolation primitive. These ideas yield 3-10 times improvement in rendering performance with the Talisman architecture with minor artifacts; greater performance can be controllably obtained with further sacrifices in fidelity.

Perceptual discontinuities may occur when a sprite's image is updated. Approximation with image warps captures the in-plane rotation, scale, and translation of an object, but not the out-of-plane rotation. The sprite image updates are sometimes perceived as a "clicking" or "popping" discontinuity. As the demand for higher quality 3D graphics increases display refresh rates, such artifacts will wane even at large factors of rendering amplification. More work is needed on the "seam" artifact (handling the boundaries of contiguous geometry placed in separate sprites.) Better modeling of the perceptual effects of regulation parameters is another area of future work [Horvitz97].

Factoring of shading terms is currently done using a fixed shading model that targets only the addition and over operations provided by hardware. Compilation of programmable shaders into layerable terms is an important extension. Many shading expressions, such as the shadow multiplication described in the appendix, can only be approximated with the over operator. We are interested in extending the system to target a fuller set of the image compositing operations.

Acknowledgements

The authors would like to thank the Microsoft Talisman Group and Microsoft Research, especially Jim Kajiya, Larry Ockene, Mark Kenworthy, Mike Toelle, Kent Griffin, Conal Elliott, Brian Guenter, Hugues Hoppe, Eric Horvitz, David Jenner, Andrew Glassner, Bobby Bodenheimer, Joe Chauvin, Howard Good, Mikey Wetzel, Susan O'Donnell, Mike Anderson, Jim Blinn, Steve Gabriel, Dan Ling, and Jay Torborg. Thanks to Nathan Myhrvold for the research direction and core ideas, and to Russell Schick for initial work on error-based sprite re-rendering. Thanks also to Allison Hart Lengyel and Julia Yang-Snyder.

Appendix: Shadow Sprites

For a fast-moving shadow on a slow-moving receiver, we update only the fast-moving shadow and use the compositor to compute the shadow modulation. Since the compositor supports only 'over', we use the following approximation.

Let $\mathbf{B}=[\beta B,\beta]$ be the receiver, where B is the color and β is the coverage. Let $\mathbf{A}=[\alpha A,\alpha]$ be the desired shadow sprite, where A is the color and α is the coverage. The compositor computes

$$\mathbf{A}\,\text{over}\,\mathbf{B}=[\alpha A+(1-\alpha)\beta B,\ \alpha+(1-\alpha)\beta].$$

Let s be the shadow modulation obtained by scan-converting the geometry of the background while looking up values in the shadow map of the fast moving object, where 0 means fully in shadow and 1 means fully illuminated.

The desired result is $\mathbf{C}'=[s\beta B,\beta]$. By letting $\mathbf{A}=[0,(1-s)\beta]$, we get $\mathbf{C}'=\mathbf{A}\,\text{over}\,\mathbf{B}$, or $\mathbf{C}'=[s\beta B+(1-s)(1-\beta)\beta B,\ \beta+(1-s)(1-\beta)\beta]$ which is close to the correct answer. Where there is no shadow, s is 1 and we get the correct answer of $[\beta B,\beta]$. Where coverage is complete, β is 1 and we get the correct answer of $[sB,1]$. The problem lies in regions of shadow and partial coverage.

Ideally, the shadow modulation needs a color multiply operator '$*$' defined by $s*[\beta B,\beta]=[s\beta B,\beta]$. This is a per-pixel version of the Porter-Duff 'darken' operator. Note that this operator is not associative, and so requires the saving of partial results when used in a nested expression.

References

[Blinn96] Consider the Lowly 2x2 Matrix, Jim Blinn, *IEEE Computer Graphics and Applications*, March 1996, pp. 82-88.

[Chen93] View Interpolation for Image Synthesis, Shenchang Eric Chen and Lance Williams, *SIGGRAPH* 93, pp. 279-288.

[Chen95] QuickTime VR – An Image-Based Approach to Virtual Environment Navigation, Shenchang Eric Chen, *SIGGRAPH* 95, pp. 29-38.

[Cook84] Shade Trees, Robert L. Cook, *SIGGRAPH* 94, pp. 223-232.

[Dorsey95] Interactive Design of Complex Time Dependent Lighting, Julie Dorsey, Jim Arvo, Donald P. Greenberg, *IEEE Computer Graphics and Application*, March 1995, Volume 15, Number 2, pp. 26-36.

[Funkhouser93] Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments, Thomas A. Funkhouser and Carlo H. Séquin, *SIGGRAPH* 93, pp. 247-254.

[Gill81] Practical Optimization, Philip E. Gill, Walter Murray, and Margaret H. Wright, Academic Press, London, 1981.

[Greene93] Hierarchical Z-Buffer Visibility, Ned Greene, Michael Kass, Gavin Miller, *SIGGRAPH* 93, pp. 231-238.

[Greene94] Error-Bounded Antialiased Rendering of Complex Environments, Ned Greene and Michael Kass, *SIGGRAPH* 94, pp. 59-66.

[Guenter95] Specializing Shaders, Brian Guenter, Todd B. Knoblock, and Erik Ruf, *SIGGRAPH* 95, pp. 343-350.

[Hanrahan90] A Language for Shading and Lighting Calculations, Pat Hanrahan and Jim Lawson, *SIGGRAPH* 90, pp. 289-298.

[Hofmann89] The Calculus of the Non-Exact Perspective Projection, Scene-Shifting for Computer Animation. Georg Rainer Hofmann. *Tutorial Notes for Computer Animation, Eurographics '89.*

[Horvitz96] Flexible Rendering of 3D Graphics Under Varying Resources: Issues and Directions, Eric Horvitz and Jed Lengyel, In *Symposium on Flexible Computation*, AAAI Notes FS-96-06, pp. 81-88. Also available as Microsoft Technical Report MSR-TR-96-18.

[Horvitz97] Decision-Theoretic Regulation of Graphics Rendering, Eric Horvitz and Jed Lengyel, In *Thirteenth Conference on Uncertainty in Artificial Intelligence*, D. Geiger and P. Shenoy, eds., August 1997.

[Hubschman81] Frame to Frame Coherence and the Hidden Surface Computation: Constraints for a Convex World, Harold Hubschman, and Steven W. Zucker, *SIGGRAPH* 81, pp. 45-54.

[Kay86] Ray Tracing Complex Scenes, Timothy L. Kay and James T. Kajiya, *SIGGRAPH* 86, pp. 269-278.

[Maciel95] Visual Navigation of Large Environments Using Textured Clusters, Paolo W. C. Maciel and Peter Shirley, *Proceedings 1995 Symposium on Interactive 3D Graphics*, April 1995, pp. 95-102.

[Mark97] Post-Rendering 3D Warping, William R. Mark, Leonard McMillan, and Gary Bishop, *Proceedings 1997 Symposium on Interactive 3D Graphics*, April 1997, pp. 7-16.

[Meier96] Painterly Rendering for Animation, Barbara J. Meier, *SIGGRAPH* 96, pp. 477-484.

[Molnar92] PixelFlow: High-Speed Rendering Using Image Composition, Steve Molnar, John Eyles, and John Poulton, *SIGGRAPH* 92, pp. 231-240.

[McMillan95] Plenoptic Modeling: An Image-Based Rendering System, Leonard McMillan, Gary Bishop, *SIGGRAPH* 95, pp. 39-46.

[Porter84] Compositing Digital Images, Thomas Porter and Tom Duff, *SIGGRAPH* 84, pp. 253-259.

[Regan94] Priority Rendering with a Virtual Reality Address Recalculation Pipeline, Matthew Regan and Ronald Pose, *SIGGRAPH* 94, pp. 155-162.

[Segal92] Fast Shadows and Lighting Effects Using Texture Mapping, Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul Haeberli, *SIGGRAPH* 92, pp. 249-252.

[Schaufler96a] Exploiting Frame to Frame Coherence in a Virtual Reality System, Gernot Schaufler, *Proceedings of VRAIS '96*, April 1996, pp. 95-102.

[Schaufler96b] A Three Dimensional Image Cache for Virtual Reality, Gernot Schaufler and Wolfgang Stürzlinger, *Proceedings of Eurographics '96*, August 1996, pp. 227-235.

[Shade96] Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments, Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, John Snyder, *SIGGRAPH* 96, pp. 75-82.

[Shelley82] Path Specification and Path Coherence, Kim L. Shelley and Donald P. Greenberg, *SIGGRAPH* 82, pp. 157-166.

[Snyder97] Visibility Sorting for Dynamic, Aggregate Geometry, John Snyder, Microsoft Technical Report, MSR-TR-97-11.

[Torborg96] Talisman: Commodity Real-time 3D Graphics for the PC, Jay Torborg, Jim Kajiya, *SIGGRAPH* 96, pp. 353-364.

[Xie95] Feature Matching and Affine Transformation for 2D Cell Animation, Ming Xie, Visual Computer, Volume 11, 1995, pp. 419-428.

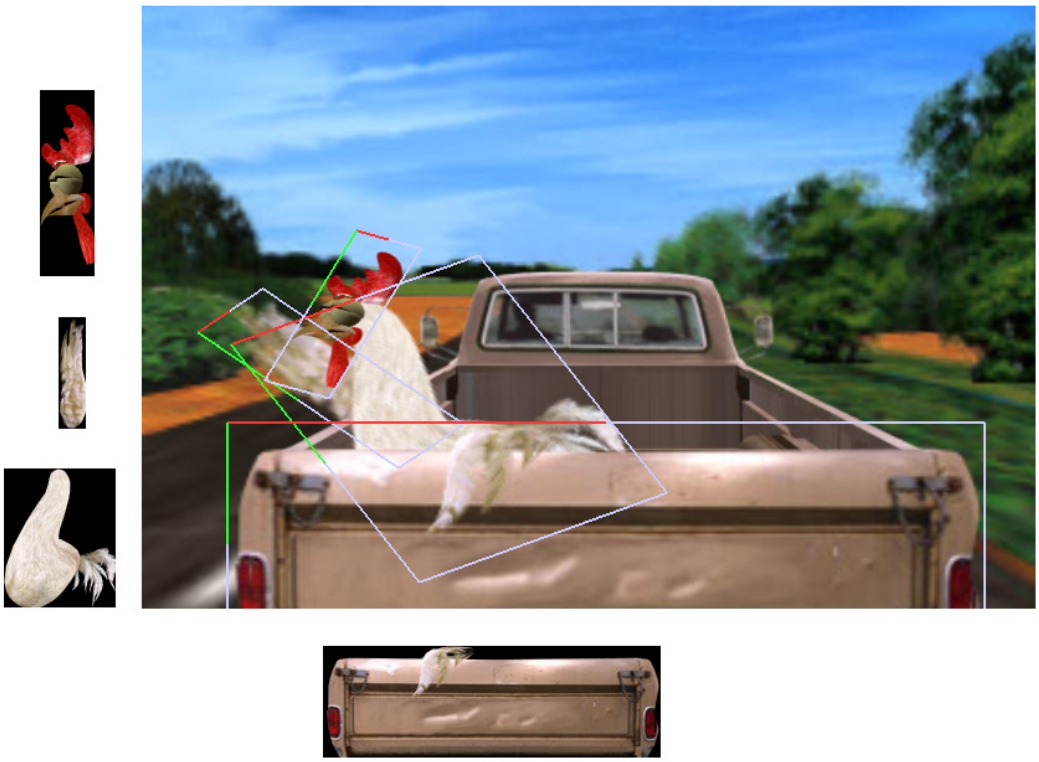


Figure 24: CHICKEN CROSSING sequence used 80 layers, some of which are shown separately (left and bottom) and displayed in the final frame with colored boundaries (middle). The sprite sizes reflect their actual rendered resolutions relative to the final frame. The rest of the sprites (not shown separately) were rendered at 40-50% of their display resolution. Since the chicken wing forms an occlusion cycle with the tailgate, the two were placed in a single sprite (bottom).

Figure 25: CANYON FLYBY used 10 layers with a geometric fiducial threshold of 4 pixels. The average sprite update rate was 19% with little loss of fidelity.

Figure 26: BARNYARD was factored into 119 geometry layers, 2 shadow map layers, and 2 shadow modulation layers. Threshold regulation for various geometric fiducial thresholds is compared in Figures 22 and 23.

EUROGRAPHICS '99 / P. Brunet and R. Scopigno
(Guest Editors)

Volume 18 (*1999*), *Number 3*

Multi-layered impostors for accelerated rendering

Xavier Decoret[†] Gernot Schaufler[‡] François Sillion[†] Julie Dorsey[‡]

[†] iMAGIS – GRAVIR/IMAG-INRIA
Grenoble, France

and

[‡] MIT Laboratory for Computer Science
Cambridge, MA, USA

Abstract

This paper describes the successful combination of pre-generated and dynamically updated image-based representations to accelerate the visualization of complex virtual environments. We introduce a new type of impostor, which has the desirable property of limiting de-occlusion errors to a user-specified amount. This impostor, composed of multiple layers of textured meshes, replaces the distant geometry and is much faster to draw. It captures the relevant depth complexity in the model without resorting to a complete sampling of the scene. We show that layers can be dynamically updated during visualization. This guarantees bounded scene complexity in each frame and also exploits temporal coherence to improve image quality when possible. We demonstrate the strengths of this approach in the context of city walkthroughs.

1. Introduction

The interactive visualization of extremely complex geometric datasets is becoming an increasingly important application of computer graphics. Although the performance of graphics hardware has improved dramatically in recent years, the demand for performance continues to grow, as environments containing many millions of polygons become commonplace. In order to visualize such scenes at interactive rates, it is necessary to limit the number of geometric primitives rendered in each frame.

Recently, image-based rendering (IBR) techniques have emerged as an attractive alternative to geometry-based systems for interactive scene display. With IBR methods, the three dimensional scene is replaced by a set of images, and traversing the scene is therefore independent of object space complexity.

To date, two distinct approaches have been explored in the construction of image impostors or caches [1, 2, 3, 4] for the interactive navigation of complex scenes. The first approach involves the pre-generation of an image-based representation for a collection of viewpoints. The advantage of this scheme is that it is possible to deal with excessive and potentially unbounded geometric complexity in a preprocessing step. The disadvantage is that the representation is fixed, so it must be used whenever the point of view is within the associated region of space for which this representation has been generated. Hence, artifacts that are introduced cannot be corrected, even if there is sufficient time and additional information about the viewer's position and viewing direction available during runtime.

In the second approach, impostors are updated dynamically. With this strategy, if the user moves slowly or comes to a complete halt, the image can be progressively refined to the correct image — that is, the one that would have been obtained by rendering the original geometry. A disadvantage of dynamically generated impostors arises from from the potentially unbounded complexity of the geometry that needs to be converted into the image-based representation. As the updates are done on the fly, they must fit into the frame-time budget allocated for generating the final images.

This paper introduces a novel impostor [3] representation that combines pre-generated and dynamically updated impostors into a single framework. We call this new representation the *multi-mesh impostor* (MMI). An advantage of the MMI is that does not enforce a single choice of how to incorporate its contents into the final image. Hence, it allows for a pre-calculated representation where necessary, but also supports a gradual transition to dynamic updates for better quality when feasible within the given frame-time constraints. In addition, the MMI improves on previous image-based scene representations by allowing the user to control the number of occlusion artifacts. By choosing a single quality parame-

ter, the occurrence of typical image-based rendering artifacts such as "rubber sheets" or cracks due to de-occlusion events can be restricted to a given size.

The remainder of the paper is organized as follows. The next section surveys previous work and discusses the strengths and weaknesses of pre-generated and dynamically updated impostors. Section 3 presents a taxonomy of the most common artifacts that occur with existing imposter techniques. Section 4 introduces the multi-meshed impostor (MMI). Section 5 presents an application of the MMI — combined with dynamically updated impostors — to the interactive visualization of large models. Section 6 reports results of the combination of MMIs and pre-generated impostors relative to previous approaches. Section 7 concludes with a discussion of the tradeoffs that have been made in the implementation.

2. Previous work

In this section, we review previous work on pre-generated and dynamically updated image-based representations

2.1. Pre-generated image-based representations

Pre-generated image-based representations make use of off-line resources to deal with the potentially unbounded geometric complexity of a given scene. Researchers have generated planar or omni-directional images to represent distant geometry. Such images must be available for sufficiently close viewpoints so that switching from one image to the next during a walkthrough is seamless. Using image warping usually reduces popping artifacts, although many warping methods introduce problems of their own, such as cracks or rubber-sheet triangles.

In order to cope with these artifacts, the number of viewpoints from which a representation is valid can be increased, but this adds to the considerable storage overhead of image-based methods (consider the memory requirements of one full-screen sized image in true-color of almost three megabytes). These representations must be paged from secondary storage during runtime, as this image data will generally not fit into main memory.

In order of ascending complexity of the sample organization, Grossman et al.[24] and Dally et al.[5] use individual points or small groups of points as samples in the image-based representation. Maciel et al.[3] and Aliaga[6] use planar projections to sample large portions of a model and texture-map them onto polygons for the final image. Chen[7], McMillan et al.[8] and Xiong[28] use environment maps to represent the distant portion of the scene that is surrounding the user. Sillion et al.[2], Darsa et al.[9] and Pulli et al.[10] use meshes to approximate a 3D image warp. Finally, Laveau et al.[11], Max et al.[12, 13] and Rafferty et al.[26] use 3D image warping of the samples.

An alternative approach extends the notion of an image to layered depth images[14] (LDIs) where each pixel contains all the intersections of the ray with the scene. Consequently, any portion of the object's surface is available for exposure, at the added cost of storage and processing of samples that will not become visible under the set of relevant viewing conditions. This representation must be generated during preprocessing, as current graphics hardware does not support the maintainence of multiple samples along the viewing ray per pixel.

2.2. Dynamically updated image-based representations

With dynamically updated image-based representations, there is little time to convert an image into a more elaborate representation. Instead, the data produced in the frame buffer by the graphics hardware must be reused largely in its current state. Special hardware [15] has been proposed to do this, and on some computers, a unified memory architecture [25] helps in reusing image data for rendering.

When flat image data is not augmented by depth information, one cannot deviate much from the point of view for which the image was generated without noticing the substitution. Warping a depth-image can help increase the lifetime of the image-based representation. However, as long as only one image is warped, occlusion artifacts will appear as surface sections hidden in the reference image become exposed.

In the work of Kajiya et al.[15] and Lengyel et al.[16], the final image is composited from individually updated image layers. The layers each contain a set of non-penetrating objects and are composited back to front by novel video hardware that removes the traditional frame buffer and decouples video refresh from object image refreshes. Shade et al.[1] and Schaufler et al.[4] use polygons with textures depicting portions of the scene. If the viewpoint moves too much, the textures are updated. Regan and Pose [17] use shells of environment maps around the user as the projection of distant objects changes slower under motion of the viewpoint. Each environment map is updated at a rate corresponding to the distance from the map to the user.

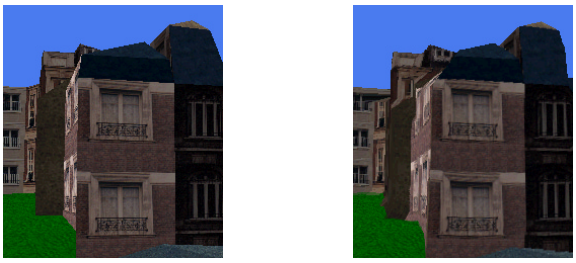
Schaufler[27] has presented a warping technique that is compatible with current graphics hardware architectures. Mark et al.[18] maintain the user's current view along with a depth map of this image as a representation of the complete scene. By warping this view, they can generate fast intermediate frames at low cost between full renderings of the scene's geometry. They also demonstrate this approach in a remote server-client setting. A related strategy was developed by Mann et al. [19] in the context of remote navigation of virtual environments.

Taken individually, the above approaches offer advantages, however, as explained in the introduction, a combination of these approaches would be desirable.

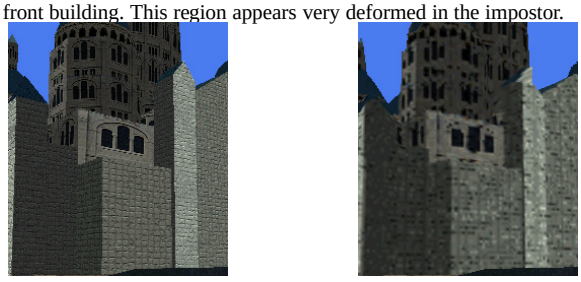
3. Limitations of existing impostor techniques

All of the image-based impostor techniques reviewed above perform some simplification of the underlying 3D model, replacing it with a meshed or point sampled representation. This simplification typically results in a number of potential image artifacts in the final application. The following table presents a taxonomy of the most common of such artifacts. In each case, the left-hand image is created from the geometry and the right-hand image from an impostor for the same viewpoint. The first two problems (**A** and **B**) are due to a poor representation of the model, while the last three (**C**-**E**) are linked to the dynamic behavior of the representation as the viewer moves.

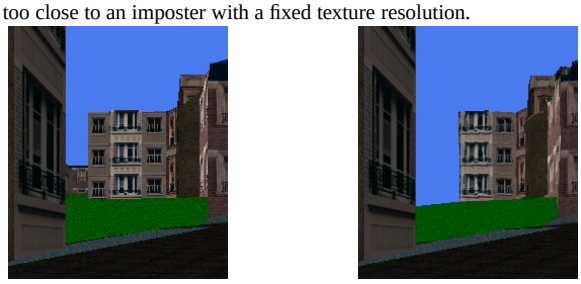
- *(A) Deformation by the impostor representation*: When the geometry of the underlying model is not properly sampled, the resulting imposter representation often appears deformed. To alleviate this problem, it is necessary to perform some analysis of the images to establish a precise correlation between the features of the geometry and the resulting mesh on the imposter [2].
- *(B) Resolution mismatch*:
  Since impostors are created from a sampled representation, they have a "built-in" resolution, which does not necessarily correspond to the image resolution at viewing time. This fixed resolution is usually determined by a predefined viewing distance and is associated with some region of the geometric model. If the viewer deters from these predefined viewing conditions, aliasing artifacts will occur. Appropriate filtering can of course reduce the resolution mismatch, but also creates unwanted blurring of image features and model textures.
- *(C) Incomplete representation*: Building impostors from images limits the amount of information about the geometry that is visible in the reference images. Hence, innapropriate selection of the reference images (or of the set of potentially visible objects) results in objects not appearing when they should be uncovered.
- *(D) Rubber sheet effect*:
  Meshed impostors implicitly force the representation of the model to be continuous in space, as if a sheet of stretchable material is wrapped onto the sampled mesh points. When the viewer moves in such a way that he looks in-between disjoint objects of the model, the view is blocked by this "rubber sheet" connecting foreground and background.
- *(E) Image cracks*:
  Point sampled impostors, on the other hand, suffer from the "cracking" problem, which occurs when no information is present for certain directions from a new viewpoint. This effect can be reduced by splatting or by using multiple depth samples [14].



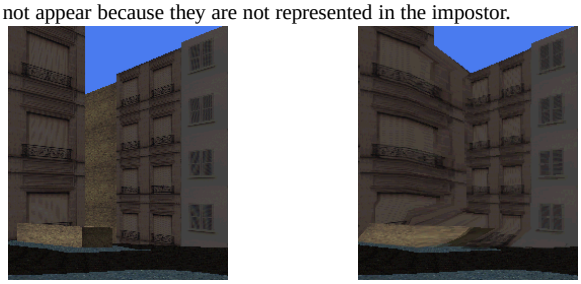
A: Deformation by the impostor representation. A meshed impostor is created without including in the mesh the top-left corner of the front building. This region appears very deformed in the impostor.



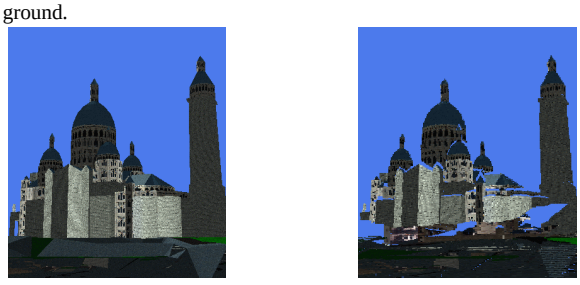
B: Resolution mismatch. Aliasing artifacts occur as the user moves too close to an imposter with a fixed texture resolution.



C: Incomplete representation. As we move past the building on the left, new objects should be uncovered. However, some objects do not appear because they are not represented in the impostor.



D: Rubber sheet effect. In this example, the view to the building in the center is blocked due to a "rubber sheet" that sits in the foreground.



E: Image cracks. "Cracks" appear in this view, especially around the church and tower.

In the next section, we describe a new type of pre-generated meshed impostor that addresses the artifacts caused by the movement of the viewer (items **C**-**E** above). In Sections 5 and 6, we show how this impostor can be combined with dynamic updates to refine the image quality on the fly and address items **A** and **B**.

4. Reducing visibility artifacts using multiple meshes

In order to improve meshed impostors [2, 9] by using multiple meshes, we assume that the environment is divided into a set of *viewing cells*. Given such a cell and three-dimensional geometry representing the model portion visible from that cell, we compute a suitable impostor that will (a) be fast to draw, and (b) correctly represent the geometry for all viewpoints within the viewing cell.

A correct visual representation requires that at least all visibility changes up to a certain magnitude are captured by our impostors. We measure the size of *visibility events* by the change of the angle under which two points on two different objects can be seen from within the viewing cell. This angular variation increases as the points become more separated in depth. Putting them into the same impostor will limit their relative positions in the final image substantially; putting them into different impostors will capture the parallax effects between them.

Considering a point A on an object $\mathcal{O}_1$ and another point B on an object $\mathcal{O}_2$ we need an upper bound for how A and B can move relative to each other in the image (see Figure 1).

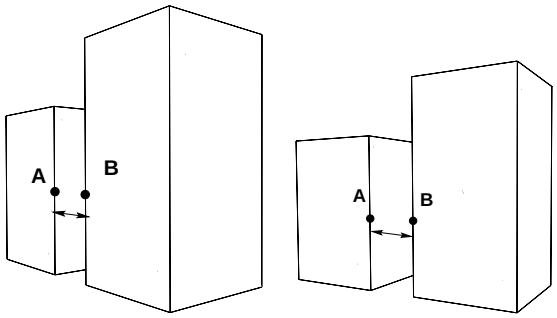


Figure 1: *Two cubes seen from two different points of view. Due to parallax, points A and B have different relative locations.*

4.1. Quantifying the importance of visibility events

Visibility events between two objects can only occur if the two objects overlap in image space as seen from somewhere within the viewing cell. Otherwise, the mesh of the impostor will sufficiently approximate the object's surfaces in 3D.

We define the *critical zone* S of an object as the convex hull of the object and the viewing cell (see Figure 2). If $\mathcal{O}_1$ does not intersect the critical zone of $\mathcal{O}_2$, there can be no overlap in image space. Hence, the importance $w_{i,j}$ of visibility events between $\mathcal{O}_1$ and $\mathcal{O}_2$ is zero. Otherwise, $\mathcal{O}_1$ may cover and uncover parts of $\mathcal{O}_2$, and $w_{i,j}$ must be set to quantify the amount of overlap.

In the general 3D case and with viewing cells of arbitrary shape, this is a difficult problem. We propose a solution for the restricted case where:

- objects have the shape of buildings that are extruded from their footprint on the ground plane,
- viewing cells are line segments (e.g. as streets, paths, etc.),
- the user moves at "ground level," that is mostly horizontally or on a designated terrain.

Under such assumptions, we can compute $w_{i,j}$ for two objects from their orthogonal projections onto a ground plane.

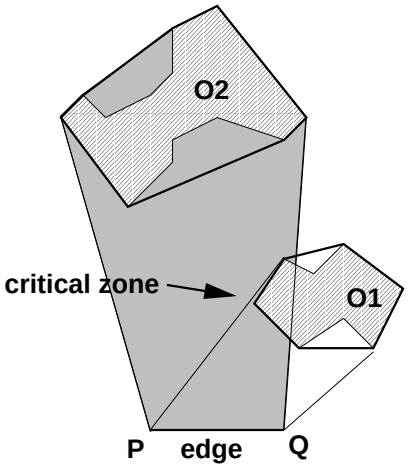


Figure 2: *Critical zone for the object $\mathcal{O}_2$ and the viewing cell $[PQ]$*

To obtain $w_{i,j}$, we derive a formula for the maximum and minimum angles under which a point $M \in [PQ]$ can see the two points A and B on $\mathcal{O}_1$ and $\mathcal{O}_2$. We set $w_{i,j}$ to the difference between these two extremal angles.

In the coordinate system indicated in Figure 3, (x_A, y_A) is defined to be at $(0,1)$.

There are two configurations:

- $y_B = 1$, that is $(AB) \parallel (PQ)$ as on the left of Figure 3.
  Let H be the intersection of $[AB]$'s *median* with (PQ). $\widehat{AMB}$ is increasing for $x_M \in (-\infty, x_H]$ and decreasing for $x_M \in [x_H, +\infty)$. Therefore, the maximum angle occurs for P, Q or H if $H \in [PQ]$, and the minimum is in P or Q.
- $y_B \neq 1$ as on the right of Figure 3.
  In this case, we have:

$$cos(\widehat{AMB}) = \frac{\overrightarrow{AM} \cdot \overrightarrow{BM}}{|AM| \cdot |BM|} \quad (1)$$

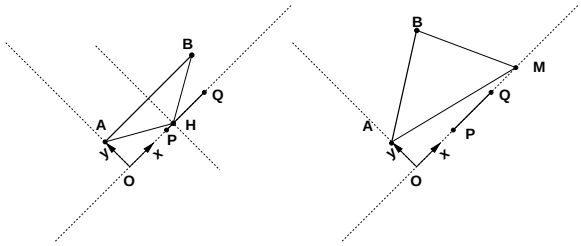


Figure 3: *The two cases for calculating the extremal viewing angles $\widehat{AMB}$.*

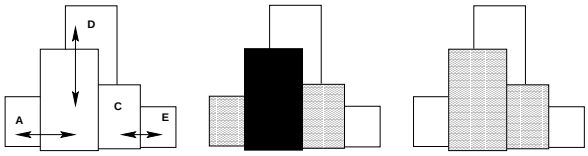


Figure 4: *The decomposition into layers is not unique: (a) represents 5 blocks that have relations as indicated by arrows. We do not consider any weight here. (b) and (c) show two ways of grouping blocks respecting the relations constraints, with 2 and 3 layers, respectively.*

$$= \frac{x(x-x_b)+y_b}{\sqrt{(x^2+1)((x-x_b)^2+y_B^2)}} \quad (2)$$

Since the angle of interest lies in $[0,\pi]$, it is extremal when its cosine is extremal. We found these extrema to be:

$$x_{M_1} = \frac{x_B}{1-y_B} \quad (3)$$

$$x_{M_2} = \frac{x_B + \sqrt{y_B * ((y_B-1)^2+x_B^2)}}{1-y_B} \quad (4)$$

$$x_{M_3} = \frac{x_B - \sqrt{y_B * ((y_B-1)^2+x_B^2)}}{1-y_B} \quad (5)$$

If those points lie on $[PQ]$ then one of them maximizes and one of them minimizes the angle $\widehat{AMB}$. Otherwise, we know that these extrema occur either at P or at Q.

4.2. A grouping criterion based on visibility events

To partition the objects into layers we construct a graph $\mathcal{G}$ of weighted relations between the objects in the following manner:

- $\mathcal{G}$ has a node per object,
- an edge is created between each pair of nodes with an associated weight $w_{i,j}$,
- the weight $w_{i,j}$ of the edge connecting nodes i and j is set to reflect the importance of visibility events between the two objects as described above.

$$w_{i,j} = \widehat{AMB}_{max} - \widehat{AMB}_{min}$$

- edge weights can be artificially increased to enforce relative importance of some objects such as landmarks or objects of particular interest.

We then partition $\mathcal{G}$ into subgraphs $\{\mathcal{G}_1 \ldots \mathcal{G}_l\}$ such that:

$\forall B_i, B_j \in \mathcal{G}_k \quad w_{i,j} \leq T,$ where T is a given threshold.

In other words, we place those objects into the same layer among which only negligible visibility events can occur. As shown in Figure 4 this decomposition is not unique.

We can express the graph partitioning problem as an n-color problem. If we represent each layer as a color, we want to attribute a color to each node of a graph such that connected nodes have different colors. This problem always has a solution: by allowing as many colors as the number of nodes, each node can be assigned a different color. However, finding the smallest number of colors required to color a non-planar graph is known to be NP-complete [20]. We will apply a heuristic to solve for our particular objective in tractable time.

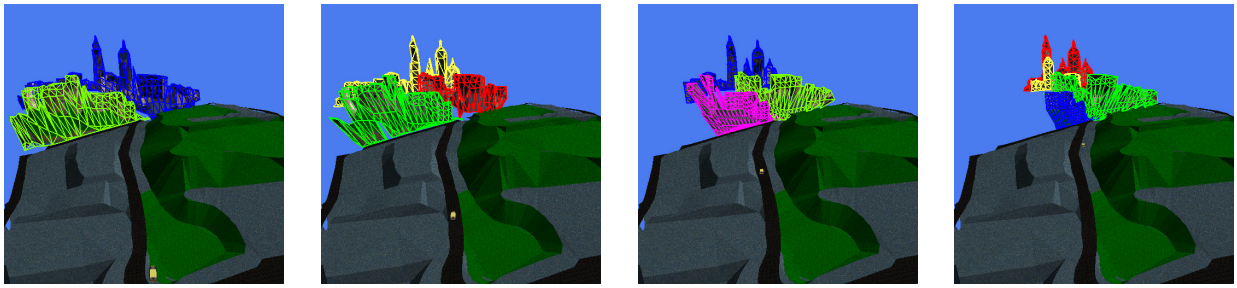
Our incremental algorithm considers one object at a time. It tries to place the object into an existing layer, ensuring that the weight of edges connecting it to objects already in this layer does not exceed a given threshold. If no such layer exists, we create a new layer for this object. When several layers could accept the object, we choose one such that the bounding box of objects in this layer (including the one we want to add) has minimal area. The rationale for this decision is that the closer objects are in the image, the smaller the texture size can be for a desired sampling rate.

4.3. Results of the layer organization

We implemented the described criterion in a city visualization software described further in this paper. For the moment, we just illustrate how our criterion behaves. Figure 5 (see also the color section) shows bird's eye views of the MMIs computed for four different viewing cells. Each viewing cell is a street segment (indicated by the small car in the street, positioned at the segment's starting point). The objects we consider are city blocks. The application of the selection criterion produces variable groupings of the blocks and different numbers of layers. The same 3D geometry set is processed for all four viewing cells. Each layer is used to build a meshed impostor, shown on the images. As the viewing cell gets closer to the sets of objects, the number of requested layers increases, which is the behavior we would expect.

5. Application to the interactive visualization of large models

Making the best use of the capabilities of rendering hardware in the visualization of large models enables a number



2 layers 3 layers 3 layers 4 layers

Figure 5: *MMIs with different numbers of layers are computed for different viewing cells*

of possibilities for how to combine off-line computation, the results of which need to be stored and loaded at runtime; online computation and its distribution among the host CPU; and the graphics subsystem.

5.1. Estimating storage requirements

On the storage-intensive end of the spectrum, MMIs are precalculated for every viewing cell, and several textures, several meshes, and a list of objects in the near geometry need to be stored with every viewing cell. During runtime, the images and meshes of viewing cells adjacent to the current viewing cell are fetched into main memory while the user is moving in the vicinity. Once the user enters the viewing cell, the local geometry together with the image-based representation for the distant geometry is rendered.

Let us illustrate the orders of magnitude of storage required for a sample application. We chose the example of an urban visualization application.

Our city model consists of 572 street segments. In order to represent the distant geometry for each segment with MMIs, we need to store a mesh and a texture for each layer. Expecting an average number of two to three layers per MMI, in the worst case this will result in 572×18 images to store, if we represent the 360^o around the viewing cell with six MMIs. Using images of 256 by 256 resolution, each image requires about 197kB of memory, resulting in a total of image storage of 2.02GB.

The resolution of the meshes of each MMI layer is much less of an issue. As texture coordinates need not be kept for projective texture mapping, a 30 by 30 regular grid of triangle pairs only requires 10.8kB of storage (assuming float coordinates with four bytes per coordinate). This is a very conservative estimate, as many of these triangles will lie outside the silhouettes of the geometric objects, and thus will not make it into the representation. Hence, for 572×18 MMIs, the storage for the mesh vertices would not exceed 111.2MB. With a total of 2.1GB of uncompressed storage, we can apply JPEG compression to the images and geometry compression to the meshes, which should realistically result in a 1:10 compression ratio.

Consequently, our model would fit into less than one third of the space available on a common CDROM, leaving ample space for other data and software.

5.2. Steering online computation from stored data

As the image data obviously requires the vast majority of the storage, we are investigating the tradeoff in terms of storage requirements vs. online generation of the images. Whenever an image is required of a certain model portion, it is only in the case of excessive complexity of this portion that we cannot afford to generate the image at runtime. However, we might want to generate the image off-line, analyze it, and build a very efficient mesh for it — an approach that is clearly too expensive at runtime. Therefore, the mesh and the camera settings are determined and stored for fast retrieval at runtime, but the image is generated online using the camera parameters to be textured onto the mesh.

Even greater flexibility is feasible when the combined rendering of the local geometry and the representation of the distant geometry does not completely consume the frame-time budget. The local geometry's complexity could be chosen to be low enough so that time remains to improve on imperfections that rendering pre-generated representations might exhibit. In particular, the pre-generation has no accurate information about from where the representation will be viewed. Only approximate information in the form of the shape and size of the viewing cell is available off-line.

In contrast, during the walkthrough phase, the precise position and viewing direction are known to the system and should be used to generate the most accurate image under the given time and resource constraints. The improvement will involve the accurate geometric positioning of all image elements, but also any view-dependent effects such as specular reflection effects, lighting modifications (if the virtual viewer carries a light source), or view-dependent selection of better textures. Image-based techniques allow to amortize the cost of more accurate rendering over several frames,

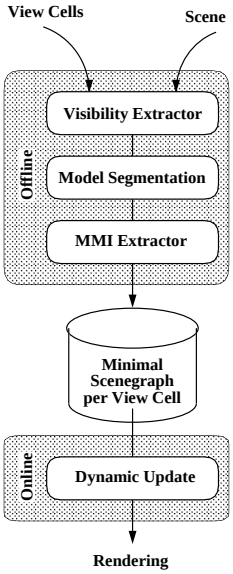


Figure 6: *The pipeline of geometry extractors.*

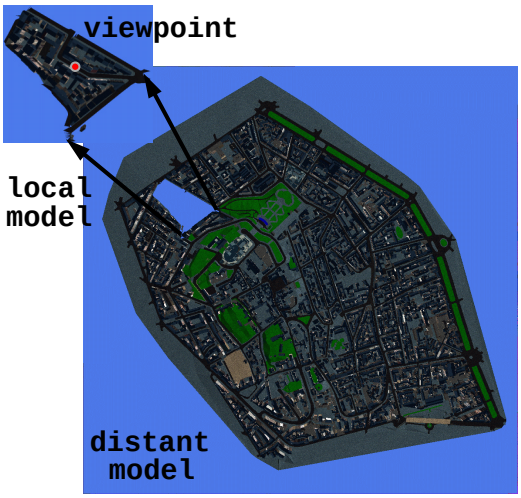


Figure 8: *Schematic overview of the scene partitioning enforced by our system. A zone of near geometry rendered as polygons surrounds the user. More distant sections are replaced by image-based representations.*

thereby exploiting the coherence in the frames to be generated. Off-line computations produce a single representation for a given viewing cell and only online techniques can improve this representation with the knowledge of the current viewing conditions.

In order to facilitate easy experimentation with such trade offs, our software architecture models the transformation of geometry from the full scene to the efficient representation to be generated per viewing cell as a processing pipeline. The geometry travels down this pipeline to be transformed into (approximatively) equivalent, more efficient representations. We refer to each stage in this pipeline as a geometry extractor. Our current pipeline of extractors is depicted in Figure 6.

First, we determine the potentially visible portion of the geometry by conservative visibility culling; next, we partition the model into near and far geometry. Finally, we extract an image-based representation for the distant part of the geometry. This arrangement also allows flexibility as to where to divide this pipeline into preprocessing and online computation. Currently, we support one pipeline stage of online computation, namely dynamic updates of image-based representations.

5.3. Visibility culling

A number of efficient techniques have been presented recently that facilitate the quick and accurate determination of a tight superset of the visible geometry as seen from a certain region of space (its PVS). We can pre-process our scene to find the PVS for all the viewing cells of our model.

In our current implementation, we use a sampling algorithm to estimate the PVS for viewing cells. It operates by taking full 360^o views of the environment from closely spaced points inside the viewing cell and recording the visible objects into an item buffer. This can be implemented efficiently in graphics hardware by assigning unique identifiers to objects coded into the RGB color channels and reading back the images. Any identifier found in the images denotes an element of the PVS. An example of the model portion identified by this approach is given in Figure 7. Note that in order to provide a frame of reference, the terrain geometry is not culled in these images.

We could easily plug in other algorithms by implementing them as geometry extractors. Interesting algorithms include those of Cohen-Or. et al.[21] and Coorg and Teller [22].

5.4. Model segmentation

Our approach is to partition the scene into two sections, the near geometry (the portion of the scene close to the viewpoint — this part will be rendered as geometry) and the far geometry (everything else — this part will be rendered with image-based representations) [2]. This partitioning, in particular the allowable complexity of the near geometry, depends on the rendering capabilities of the target system. In any case, the near geometry should be chosen in such a way that it can be rendered together with a suitable representation of the far geometry in less than the time available for a single frame. Figure 8 shows an example of this segmentation.

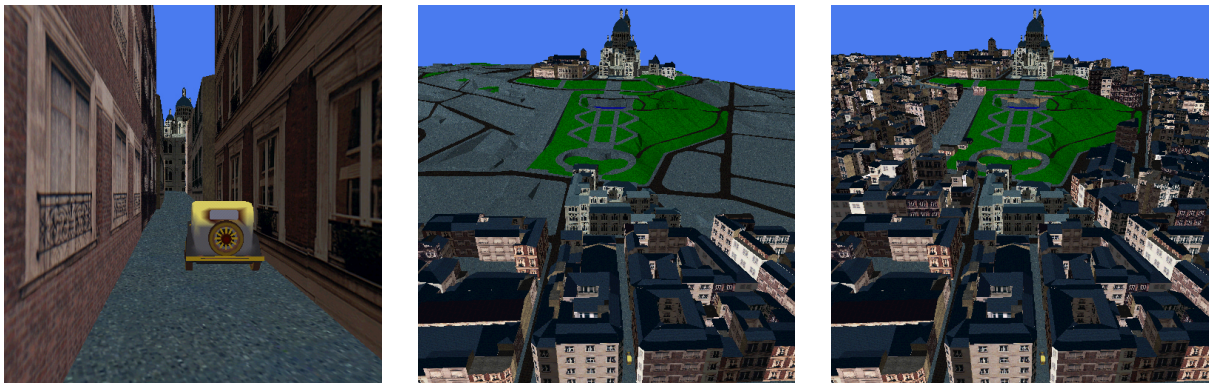


Figure 7: *From left to right: a view from the street, the set of potentially visible city blocks for the current viewing cells, and the entire model. Terrain geometry is not culled in order to maintain a frame of reference.*

5.5. Selective dynamic updates of impostors

An analysis of the primitive count in the near geometry together with its associated MMIs allows us to predict an accurate expected frame time for each viewing cell. We do not choose the near geometry's and the MMIs' combined complexity to deplete the whole frame time, but instead we strive to reduce both complexities as much as possible through visibility calculations and calculating efficient meshes for the MMIs in order to free frame time for dynamic updates. From the desired and the predicted frame times we can determine how much time we have to accomplish such updates.

Whenever free frame time remains in a viewing cell, we start to select the front-most meshes from the viewing cell's associated MMIs and convert them to dynamically updatable impostors. We store the textures of our meshes with a limited depth information of eight bits. For regular-grid impostors, this depth information is sub-sampled and converted into an efficient mesh of triangle strips. The resulting meshed impostor therefore makes no attempt to capture important depth disparities [2], under the assumption that it will be short-lived. Additional images are then rendered when needed to update the new representation in response to user movement within the viewing cell.

We borrow a criterion from the layered impostor approach to do the updates in such an order, that the gain in image fidelity is maximized. Schaufler [27] proposes to calculate an error angle as a measure of screen space deviation from the accurate image. MMI layers are considered for dynamic update in the order of decreasing error angle. Due to the inherent image warping of regular grid impostors, the images generated will be valid for some time, so more meshes can be switched to dynamic update as the images just generated can be reused.

This strategy continues as long as the coherence in the image sequence is sufficiently high and will eventually replace all the meshes of the current MMIs with a new texture. Sudden fast motion of the viewpoint will force us to switch back to MMIs only.

Figure 9: *View of a landmark in our model ("Sacre Cœur" church).*

6. Results

We have implemented the described walkthrough system in C++ using the OpenGL graphics API. The city model shown in the figures throughout this paper represents a section of the city of Paris in France known as Montmartre (see Figure 9). It consists of 143,500 polygons which have been textured using 34 texture maps. There are 146 blocks of houses and a street graph of 572 street segments connected by 427 junctions in the model.

6.1. Using MMIs

Figure 11 (see color section) shows the improvement in image quality obtained when pre-generating and rendering MMIs instead of SMIs. Note how the rubber skin triangles have disappeared from the houses toward the end of the sequence.

In order to document the performance of our approach we have recorded a walkthrough across the city model and calculated the average frame rate along this path. In our comparison, we have used the following rendering options:

- *Full Model:* In this rendering mode, the hardware renders the complete scene model without further optimization. This rendering mode gives an idea of the raw rendering performance of the platform.
- *Local Model:* Rendering just the local model sets an upper bound of the frame rate achievable. This geometry needs to be rendered as polygons since it is very close to the viewer and any simplification would be too obvious.
- *Visible Model:* The visible model is the output of our visibility culling extractor (described in Section 5.3). This is the model part, which is found to be visible from the current viewing cell. Its frame rate needs to be achieved by any alternative rendering method in order to be competitive with brute-force hardware rendering.
- *Single Mesh Impostors:* We include the performance of single mesh impostors [2] for comparison.
- *Multi Mesh Impostors:* Multiple meshes are used to capture visibility events amongst objects. Our error criterion was set to a maximum error of 10 pixels in the final frames of a resolution of 512 by 512 pixels, although visually the error is much less due to the image warping done by the meshes.
- *Multi Mesh Impostors with Dynamic Updates:* In order to overcome any artifacts introduced by the pre-generated representations, textures are updated online. Currently, our implementation is not fully optimized for the two rendering platforms used. (They offer different optimized approaches to improve speed by rendering directly into textures on the O2 platform or copying images directly from frame buffer to texture memory on the IR.)

The following table summarizes the frame rates obtained on a SGI O2 workstation, with one R10k processor running at 200 Mhz, and a SGI InfiniteReality workstation with one R10k processor running at 250 Mhz. We have used a single-buffer window configuration to eliminate the effects of synchronizing buffer switching with vertical retraces.

Frame Rate (Hz)	O2	IR
Full Model	1.0	3.3
Local Model	14.5	118.4
Visible Model	4.3	18.4
Single Mesh Impostors	15.9	103.3
Multi Mesh Impostors	10.9	85.0
MMI with Dynamic Updates	6.5	33.1

Table 1: *Frame rates achieved with the discussed rendering methods on two workstations with widely varying graphics performance.*

6.2. Dynamic impostor updates

Figure 10 shows examples of how image quality is improved when applying dynamic updates in addition to pre-generation of MMIs. Each row gives the MMI on the left, an image with dynamic updates in the middle, and an image rendered using geometry on the right.

In row one, dynamic updates are used to improve the polygonal silhouette of the meshes. Note that on the left, the silhouette of the towers in the background is of a triangular shape because of the mesh structure.

For the second row of images, we have selected an excessively large viewing cell to show how pre-calculated meshes cannot maintain a good look over all possible viewing directions of arbitrary geometry. In our model, the cathedral is a single object, and therefore, can only be assigned to a layer as a whole. Consequently, as the cathedral's tower in the front hides the rest of the cathedral, meshing artifacts occur under extreme viewing conditions. Again the middle image shows how this situation can be overcome with dynamic updates.

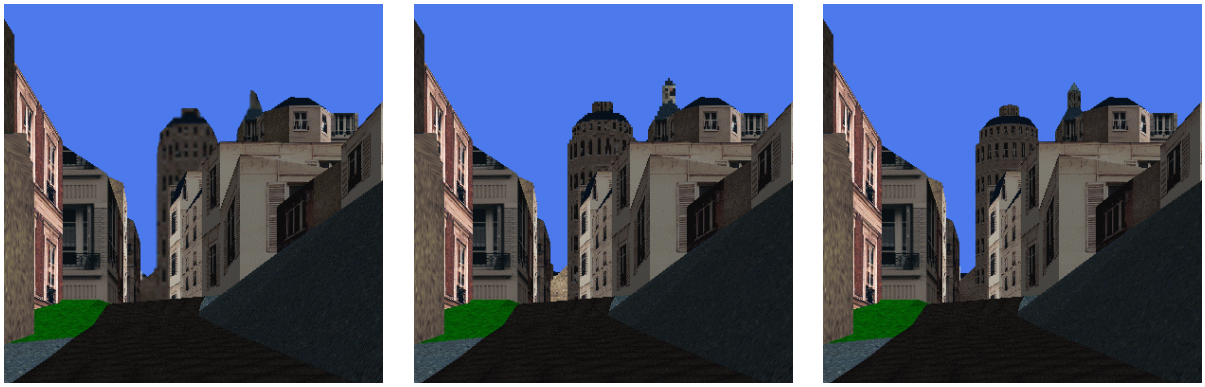
In the third row, an insufficient texture resolution stored in the MMI database is removed by a dynamic update.

7. Conclusions and Discussion

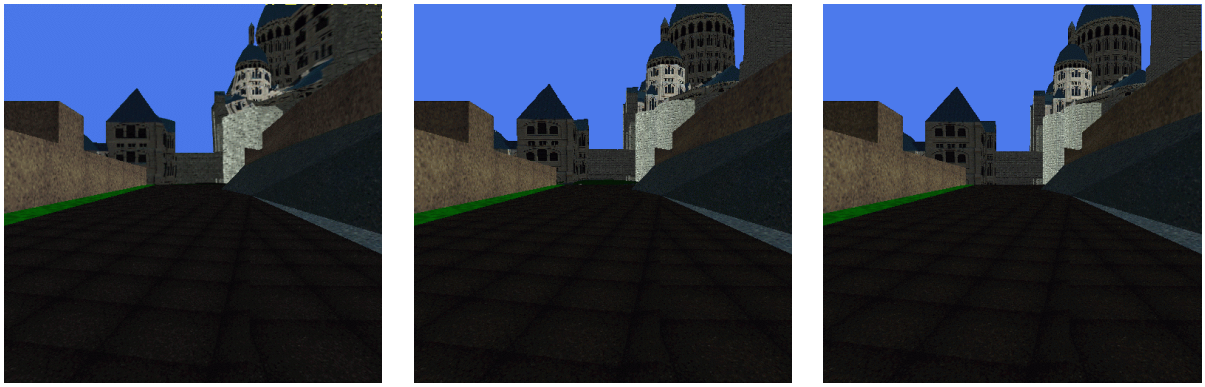
This paper has presented a new type of impostor, the multi-meshed impostor or MMI, which allows us to control the size of the visibility errors in the final image. We hope that this approach to image-based scene simplification will have a major impact on the general acceptance of image-based techniques in real world applications where uncontrolled artifacts are unacceptable.

The control over the visibility errors is obtained by generating MMIs for viewing cells of known size and analyzing the geometry to be replaced by the impostor. As a consequence of the depth differences between various parts of geometry, appropriate representation fidelity is selected. Geometry with too large an extent in depth is placed onto different impostor meshes in order to capture their perspective parallax effects.

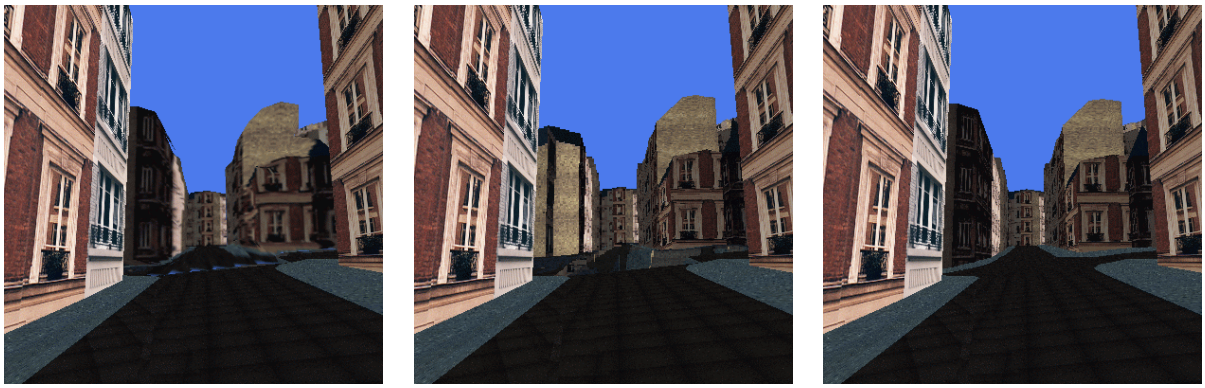
Since purely pre-generated image-based representations often do not allow us to obtain images identical to the ones obtained from geometry, we have combined MMIs with dynamic updates. Whenever frame time is available, our system strives to make the textures used in the image-based



Dynamic updates improve the silhouettes of the meshes.



Mesh distortions are removed from the final image.



Dynamic updates improve insufficient texture resolution.

Figure 10: *Improvement of image quality using dynamic updates. left: rendered using MMIs, center: rendered with dynamically updated mesh, right: rendered with geometry.*

representation converge towards the accurate image of the objects. Switching to dynamic updates or going back to pregenerated representations is especially easy in our system, as both representations build on the same image format.

The result of this combination is that the images generated by our system more closely resemble the images generated from geometry, and in the case of slow motion of the viewpoint, converge to the accurate image.

We still see potential to improve our approach in the way that we partition the model into different regions most suited for a particular type of image-based representation. In particular, we hope to decrease the size of the near model, where currently we are required to render the full geometric complexity. Level of detail approaches cannot help here either, as this part of the model dominates the user's field of view. Also, a more gradual transition from the near geometry to the distant representation would be desirable.

Acknowledgments

This work was supported in part by a joint collaborative research grant of NSF and INRIA (INT-9724005), an Alfred P. Sloan Foundation Research Fellowship (BR-3659), and by a grant from Intel Corporation. It also builds on preliminary work performed by Yann Argotti and Sami Shalabi. iMAGIS is a joint research project of CNRS, INPG, Université Joseph Fourier/Grenoble-I and INRIA.

References

1. Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. *Hierarchical image caching for accelerated walkthroughs of complex environments.* In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996.

2. François Sillion, George Drettakis, and Benoit Bodelet. *Efficient impostor manipulation for real-time visualization of urban scenery.* In D. Fellner and L. Szirmay-Kalos, editors, *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16, Budapest, Hungary, September 1997.

3. Paulo W. C. Maciel and Peter Shirley. *Visual navigation of large environments using textured clusters.* In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 95–102. ACM SIGGRAPH, April 1995.

4. Gernot Schaufler and Wolfgang Sturzlinger. *A three-dimensional image cache for virtual reality. Computer Graphics Forum*, 15(3):C227–C235, C471–C472, September 1996.

5. William J. Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs. *The delta tree: An object-centered approach to image-based rendering.* Technical Memo AIM-1604, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1996.

6. Daniel G. Aliaga. *Visualization of complex models using dynamic texture-based simplification.* In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

7. Shenchang Eric Chen. *Quicktime VR - an image-based approach to virtual environment navigation.* In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 29–38. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

8. Leonard McMillan and Gary Bishop. *Plenoptic modeling: An image-based rendering system.* In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

9. L. Darsa, B. Costa, and Amitabh Varshney. *Walkthroughs of complex environments using image-based simplification. Computers & Graphics*, 22(1):55–69, February 1998. ISSN 0097-8493.

10. Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. *View-based rendering: Visualizing real objects from scanned range and color data.* In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 23–34, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

11. Stephane Laveau and Olivier Faugeras. *3-D scene representation as a collection of images and fundamental matrices.* Technical Report RR-2205, Inria, Institut National de Recherche en Informatique et en Automatique.

12. Nelson Max. *Hierarchical rendering of trees from precomputed multi-layer Z-buffers.* In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 165–174, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

13. N. Max and K. Ohsaki. *Rendering trees from precomputed Z-buffer views.* In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

14. Jonathan W. Shade, Steven J. Gortler, Li-wei He, and Richard Szeliski. *Layered depth images.* In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 231–242. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

15. Jay Torborg and Jim Kajiya. *Talisman: Commodity Real-time 3D graphics for the PC.* In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 353–364. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

16. Jed Lengyel and John Snyder. *Rendering with coherent layers.* In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 233–242. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

17. Matthew Regan and Ronald Pose. *Priority rendering with a virtual reality address recalculation pipeline.* In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 155–162. ACM SIGGRAPH, ACM Press, July 1994.

18. William R. Mark, Leonard McMillan, and Gary Bishop. *Post-rendering 3D warping.* In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

19. Y. Mann and D. Cohen-Or. *Selective pixel transmissionfor navigating in remote virtual environments. Computer Graphics Forum*, 16(3):201–206, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055.

20. M.R. Garey and D.S. Johnson. *Computer and Intractibility: a guide to the theory of NP-completeness*, page 191. W.H. Freeman, 1979.

21. Gadi Fibich Dan Halperin Cohen-Or, Daniel and Eyal Zadicerio. *Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. Computer Graphics Forum*, 17(3):243–255, August 1998. Proceedings of Eurographics '98. ISSN 0167-7055.

22. Satyan Coorg and Seth Teller. *Real-time occlusion culling for models with large occluders.* In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 83–90. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

23. Gortler, Steven J., Li-Wei He, and Michael F. Cohen, *"Rendering Layered Depth Images"*, Microsoft Research Technical Report MSTR-TR-97-09.

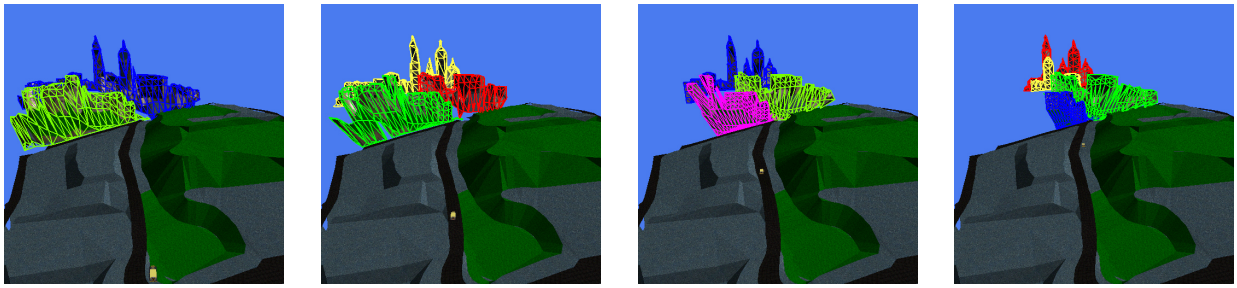
24. Grossman, J.P., and William J. Dally, *"Point Sample Rendering"*, 9th Eurographics Workshop on Rendering, August 1998, pp 181-192.

25. SGI O2, *"O2: Unified Memory Architecture"*, Silicon Graphics Datasheets and White Papers, April 1997, available at `http://www.sgi.com/o2/uma.html`.

26. Rafferty, Matthew M., Daniel G. Aliaga, Anselmo A. Lastra, *"3D Image Warping in Architectural Walkthroughs"*, VRAIS 98, pp. 228-233.

27. Schaufler, G., *"Per-Object Image Warping with Layered Impostors"*, in Proceedings of the 9th Eurographics Workshop on Rendering '98, Vienna, Austria, June 29-July 1, 1998, pp 145-156.

28. Xiong, Rebecca, *"CityScape - A Virtual Navigational System for Large Environments"*, MIT Masters Thesis, 1996.



2 layers 3 layers 3 layers 4 layers

Figure 5: *MMIs with different numbers of layers are computed for different viewing cells*

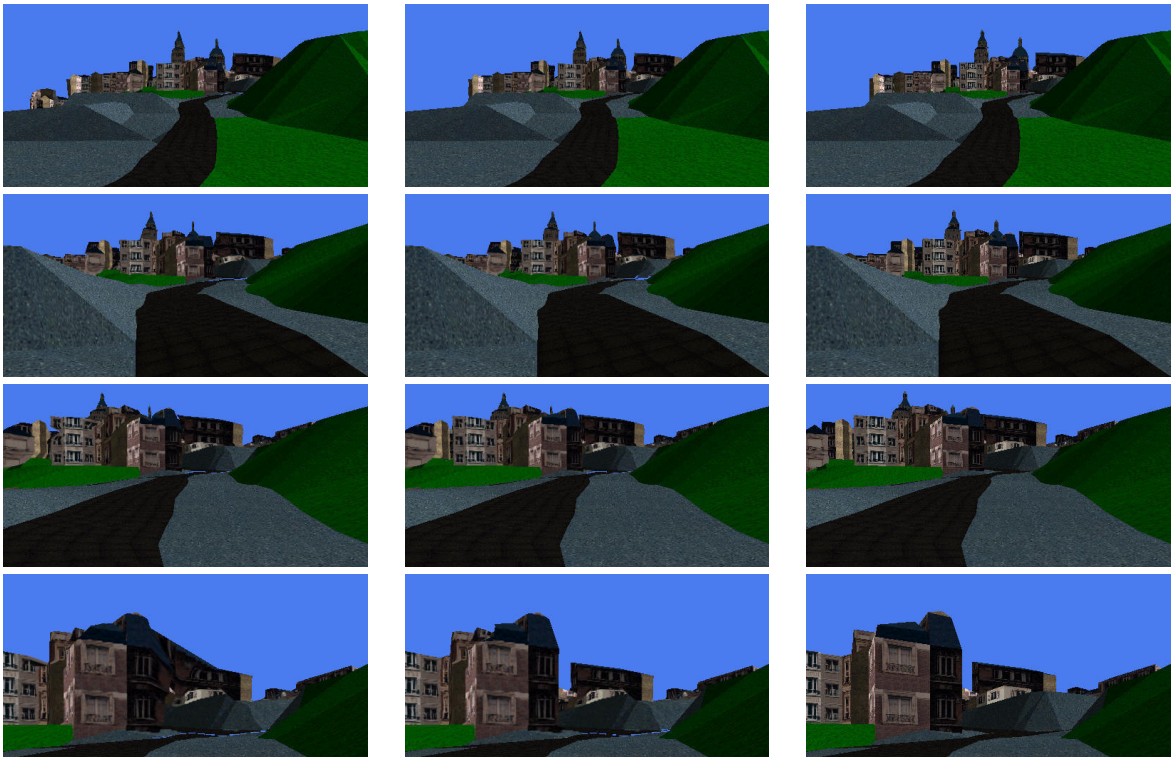


Figure 11: *Comparison of single-mesh impostors* [2] *and MMI impostors computed for a given distant model and the various viewing cells along a road. Left: Single-mesh impostors Middle: Multi-mesh impostor. Right: Actual geometry.*

A Three Dimensional Image Cache for Virtual Reality

Gernot Schaufler and Wolfgang Stürzlinger

From the Proceedings of Eurographics 1996

A Three Dimensional Image Cache for Virtual Reality

Gernot Schaufler and Wolfgang Stürzlinger

GUP, Johannes Kepler Universität Linz, Altenbergerstr.69, A-4040 Linz, Austria/Europe
schaufler@gup.uni-linz.ac.at

Abstract

Despite recent advances in rendering hardware, large and complex virtual environments cannot be displayed with a sufficiently high frame rate, because of limitations in the available rendering performance.
This paper presents a new approach of software accelerated rendering which draws from the concepts of impostors, hierarchical scene subdivision and levels of detail. So far software optimization in real-time rendering has merely considered individual objects. This work is actually optimizing the rendering of the whole virtual environment by implementing a three dimensional image cache. It speeds up rendering for large portions of the scene by exploiting the coherence inherent in any smooth frame sequence.
The implementation of the three dimensional image cache is discussed and the savings in rendering load achievable on a suitable hardware platform are presented.

Keywords: viewing algorithms, geometric algorithms, object hierarchies, virtual reality.

1 Introduction

The basic capability of any virtual reality system is to provide the user with a view of the environment. As the user navigates within the environment this view must be updated at least several times per second and recent user movements must be reflected in the view with minimal latency [6]. Usually high-end graphics workstations are used to meet these performance requirements. The environment is modelled with textured polygons which can be rendered with hardware support on these workstations. Typically the number of potentially visible polygons must not exceed several thousands if frame rates of 20 Hz or more should be achieved. A lot of work has been published how to determine the potentially visible polygons for each frame and how to make best use of the available rendering performance in order to generate images of good quality with an acceptable frame rate.
The severe limitation in the number of polygons is due to the fact that every frame is rendered from scratch: from the current point of view the set of potentially visible objects is determined and the images of these objects must be rendered within the available frame time. It would be desirable to reuse most of the image data generated during previous frames if the changes to these images can be neglected in the current frame. So far two systems have been presented which are capable of reusing previously generated images - a hardware solution called the virtual reality address recalculation pipeline [8] and a software approach called dynamically generated impostors [9].
This paper presents a new approach to software accelerated rendering which improves and generalizes the concept of impostors. Unlike dynamically generated impostors the new method can handle intersecting objects and indoor as well as outdoor scenes. It does not rely on the scene to be actually organized into objects but groups together spatially close primitives automatically. A hierarchical three-dimensional image cache stores and allows to reuse image data of parts of the scene, which was generated during previous frames. An update algorithm ensures that images are replaced with new ones if necessary. As a result, those parts of the scene for which images are already available from previous frames need not be rendered from scratch in the

current frame. This paper's examples are based on polygonal rendering. However, the 3d image cache can accelerate any kind of rendering such as free-form surface rendering or ray tracing as well.
The two sections to follow familiarize the reader with previous work in the field of real-time rendering and with the concept of dynamically generated impostors. In section 4 the hierarchical image cache is introduced. The idea is to store the scene in a hierarchical space partitioning data structure, e.g. a k-d-tree, in which images generated for any subtree can be cached and reused if appropriate. The error introduced by this caching can be limited to pixel resolution as will be shown in section 5. Conclusions are drawn in section 6 which will lead on to some possible future work.

2 Previous Work

When generating images of polygonal scenes performance can be improved if only those parts of the scene are sent to the rendering hardware which are visible in the final image. Finding the visible parts can be efficiently implemented as a traversal of a hierarchical space partitioning data structure [3].
If a uniform frame rate is desired, an appropriate level of detail can be chosen for every visible object thereby trading rendering speed for image quality [4]. Automatic algorithms exist to generate such levels of detail from polygonal objects [5][10].
Multiple frame buffer hardware can be used to overcome the drawback of having to render every visible object for every frame [8]. Observing that the images of distant objects move slower on screen than the images of close objects, the objects are rendered into different frame buffers based on their distance from the viewer. The frame buffers containing distant objects can be updated less frequently. The final image is obtained by overlapping the buffers front to back.
Object images are generated as a preprocessing step and are used during rendering as appropriate [7]. For every object images from any viewing direction must be stored in the scene database so that images from any point of view can be rendered (at the cost of high memory requirements).
Such object images can also be generated on the fly for individual objects in sparse outdoor scenes [9]. Instead of the original object model a transparent polygon is rendered with an opaque image of the object mapped onto it (an impostor). However, errors occur in the final images if objects are too close together so that visibility is not resolved correctly by rendering impostors (see figure 10 upper right). The correct image without impostors is shown in the upper left and a difference image in the lower left of figure 10.
Rendering based entirely on images was proposed for walkthroughs of static scenes [2]. The correct view can be extracted from a number of environment maps by real-time image processing. Intermediate frames can also be interpolated from available keyframes [1] and an appropriate image is shown to the user when he moves between the points for which environment maps are available.
As the image caching proposed in this paper is based upon impostors a short introduction to them follows.

3 Dynamically Generated Impostors

The concept of dynamically generated impostors is depicted in figure 1 (for a more in depth discussion of dynamically generated impostors refer to [9]):
A complex object is replaced by a textured polygon facing the viewer (see figure 1) which cannot be distinguished from the original object from the proper point of view. The texture on the polygon is generated by finding a rectangle surrounding the object (or the object's bounding box for convenience) and rendering the object into this rectangle using the graphics hardware. The resulting image is read from the frame buffer and used to define the texture on the impostor rectangle.

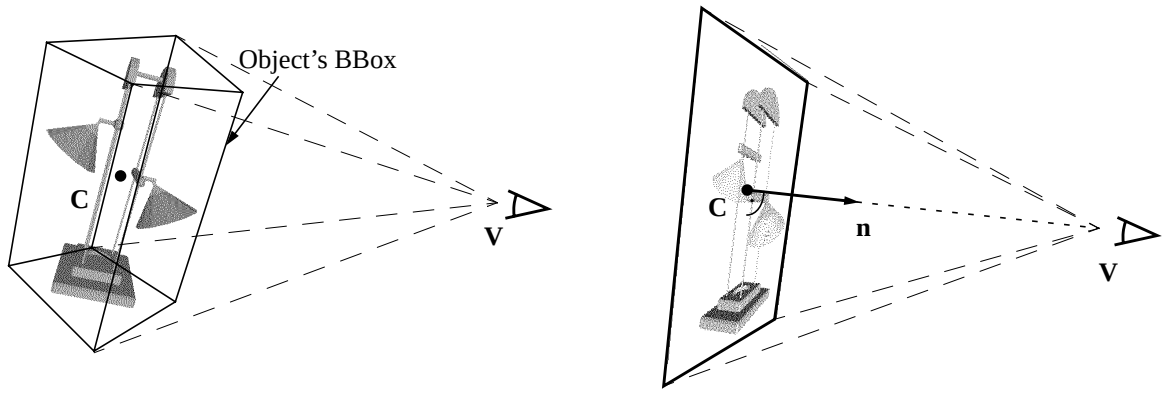


Figure 1. Original Object and Impostor

An impostor can be used as soon as the texels of the texture map are not visible in the final image. This fact can be expressed using angles under which the user sees one texel ($\alpha_{\mathbf{tex}}$), one pixel on screen ($\alpha_{\mathbf{screen}}$) and the whole screen (**fov**):

$$\alpha_{\mathbf{tex}} < \alpha_{\mathbf{screen}} = \frac{\mathbf{fov}}{\mathbf{screenres}} \tag{1}$$

As long as the user only changes his direction of view the images on the impostors need not be changed. This observation allows fast display updates during user head rotations which keeps the display lag short.
When moving sideways in front of an object the image on the impostor and the object's actual image will eventually become different. As soon as the differences would be visible in the final image a new impostor must be generated.
The need for an update can be determined by comparing the angle $\alpha_{\mathbf{trans}}$ to $\alpha_{\mathbf{screen}}$ (see figure 2). $\alpha_{\mathbf{trans}}$ is the angle under which extreme points on the object's bounding box ($\mathbf{B_1}$ and $\mathbf{B_2}$) and their image on the impostor (**B'**) are observed by the user.

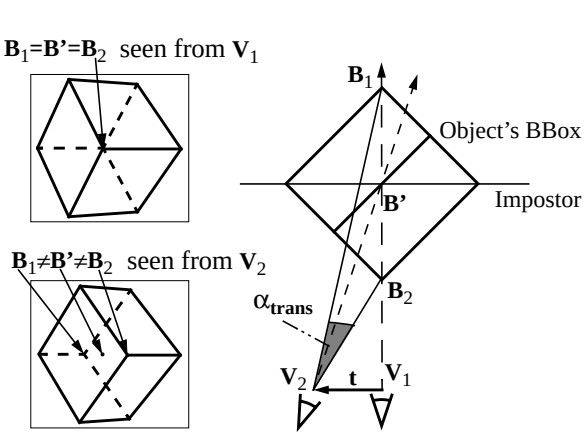


Figure 2. Error angle $\alpha_{\mathbf{trans}}$ due to translation

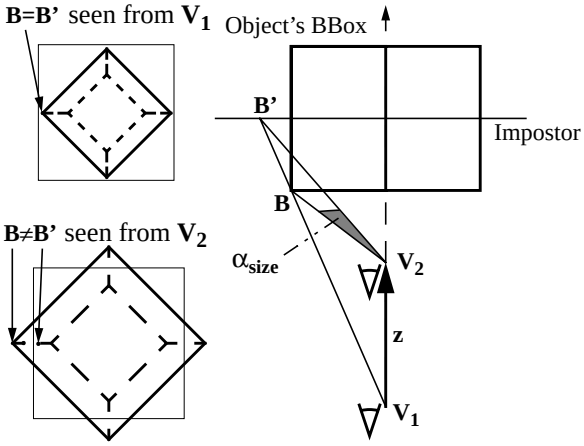


Figure 3. Error angle $\alpha_{\mathbf{trans}}$ due to move-in

A similar situation occurs when moving towards an object. In this case $\alpha_{\mathbf{size}}$ must be compared to $\alpha_{\mathbf{screen}}$ as shown in figure 3.
From those two orthogonal cases one can see that it is necessary to generate a new impostor every time either $\alpha_{\mathbf{size}}$ or $\alpha_{\mathbf{trans}}$ becomes larger than $\alpha_{\mathbf{screen}}$ (see equation (1)).
Problems occur when objects in a cluster or intersecting objects are replaced individually by impostors. For such objects impostors cannot resolve the visibility correctly and the object on the closer impostor will erroneously occlude the other object. These problems are especially annoying in indoor scenes. Examples are books in a shelf or chairs put in under a table (see figure 10 upper right for an extreme case).
Another problem occurs in scenes with large numbers of objects which all must be checked for a necessary impostor refresh. Considerable overhead results especially if the objects do not consist of many polygons. In addition some scenes are given as unstructured polygon lists which makes the use of dynamically generated impostors impossible.

4 A Three Dimensional Image Cache

In order to maintain a three dimensional image cache methods are needed to generate and store image data for parts of a scene. The current implementation of the 3d image cache generates and stores impostors for the contents of axially aligned bounding boxes (bboxes). In a hierarchy of bboxes image data in lower layers can be used in a bottom up fashion to generate images of the contents of higher nodes (considering several impostors as the contents of a bbox).
This hierarchy of bboxes is stored in a hierarchical space partitioning data structure. A k-d-tree is preferable to an octree, because it offers more flexibility: for the 3d image cache this flexibility is used to obtain approximately cubic bboxes. Such bboxes are desirable because they let scene subdivision adapt to overall scene shape and local scene complexity. Moreover the k-d-tree allows to tune the number of sub-boxes contained in one bbox to the overhead involved when generating an impostor from several impostors. Texture maps must be quadratic in this implementation which also makes approximately cubic boxes desirable as impostors then tend to be quadratic as well.
First a bbox for the whole scene is calculated. This bbox is recursively subdivided by the k-d-tree along it's longest side. Primitives (or object models) are partitioned into sub-boxes. If more than one sub-box is intersected, the primitive is stored in all the intersected sub-boxes.
Subdivision terminates if the number of primitives in a bbox falls below a given threshold or if the maximum subdivision level is reached. The number of primitives per bbox must compensate for the overhead in generating an impostor (which depends on the hardware).

4.1 Generating Impostors for BBox Contents

An impostor for the contents of a bbox can be generated in almost the same way as an impostor for an object. In a preprocessing step the part of the scene lying within the bbox is identified and stored with the bbox. Instead of clipping the primitives (lines, polygons, ...) to the bbox boundaries during preprocessing, hardware clipping planes can be used during creating the impostor. Such clipping planes are available on current graphics workstations. Cracks in objects crossing bbox boundaries can be avoided by using a slightly bigger bbox for clipping.
The generated impostor stores an image of the bbox's contents when viewed from the point of view for which the impostor was generated. Rendering the impostors for the bboxes of the k-d-tree correctly resolves visibility for all the primitives as the bboxes in the tree are disjoint.

4.2 Generating Impostors from Impostors

When the impostor of a bbox is invalid for the new point of view, it must be regenerated. If the bbox is not a leaf of the hierarchy, then a new impostor can be generated from the impostors of it's sub-boxes. As for an object a rectangle enclosing the bbox is determined and the sub-boxes' impostors are rendered into it. No clipping needs to be done in this case, as the bboxes form a hierarchy. Rendering the few impostors for the sub-

boxes is much cheaper than rendering all the primitives contained in the bbox. Re-rendering primitives is avoided until impostors in the leaves of the hierarchy become invalid. However, leaf-boxes are small which makes their error angles small as well and therefore their impostors are valid for more frames in general.

4.3 Using the 3d Image Cache

Rendering the impostors of the 3d image cache for the living room scene gives the image shown in figure 11 upper left. Compared to an image obtained by pure polygonal rendering (figure 10 upper left) any differences are due to resampling and are limited to an extent of one pixel as can be seen in the difference image (figure 11 upper right).
The contents of the 3d image cache are visualized in a bird's eye view for two different viewpoints (bottom row of figure 11). The k-d-tree used has a branching factor of eight and a depth of three. Impostors which were generated in the current frame are shown with a red border, impostors which were generated from the node's children are shown with a green border. In the right view parts of the floor and the ceiling close to the viewpoint could not be replaced by impostors due to the texture resolution limit.
For every frame to be rendered the k-d-tree is traversed to determine for which nodes an impostor can be used because the maximum texture resolution will be invisible in the final image and it is also determined which impostors need to be regenerated. Subtree traversal can be pruned if a valid impostor is encountered (as impostors of sub-boxes are always valid for more frames because their images and corresponding error angles are smaller). Pseudocode is shown in figure 4:

```
UpdateCache(Cache *c)
if(IsLeaf(c)) {
   if(ImpostorPossible(c) and ImpostorInvalid(c)) {
      GenerateImpostorFromPrimitives(c)
   }
} else {
   if(ImpostorPossible(c)) {
      if(ImpostorInvalid(c)) {
         forAll(c->child) {
            UpdateCache(c->child)
         }
         GenerateImpostorFromChildren(c)
      }
   } else {
      forAll(c->child) {
         UpdateCache(c->child)
      }
   }
}
```

Figure 4. Pseudo Code to Update the Image Cache

To render the final image the k-d-tree is traversed a second time and impostors are drawn when encountered in the tree which prunes subtree traversal. For bboxes close to the user (which cannot be replaced by an impostor), the contained primitives are drawn and clipped to the bbox boundaries (see figure 5 for the pseudo code). Limiting the cache updates to the visible portions of the k-d-tree is also a possible variation. If the viewing direction is rotated, the impostors for previously invisible bboxes must then be generated during the rotation. This option was turned off during the tests in section 5.

```
DrawCache(Cache *c)
if(Visible(c->bbox)) {            /* clip to frustum */
   if(ImpostorPossible(c)) {
      DrawImpostor(c)
   } else {
      if(IsLeaf(c)) {
         ClipTo(c);
         DrawPrimitives(c);
         ClipOff();
      } else {
         ForAll(c->child) {
            DrawCache(c->child)
         }
      }
   }
}
```

Figure 5. Pseudo Code to Draw the Image Cache

If the primitives in the scene are organized into objects and multiple levels of detail are available for each object, then the objects could be drawn with an appropriate level of detail based on their size on the screen. Such a selection scheme would guarantee that the same level of detail is selected for one object in each bbox it intersects and would adapt the work for drawing a bbox's contents to the distance from the viewer. Moreover rendering an impostor for a node in the tree implies a certain distance to the user. Therefore, only one level of detail per object must be stored per node and this level of detail is always used for regenerating the impostor for this node. With levels of detail the caching of images can be limited to one layer in the k-d-tree, namely the first layer of nodes for which impostors are useful. The texture memory necessary to store impostors of lower levels can then be saved.

5 Results

The department of computer graphics at the local university does not own a graphics workstation which supports texture mapping in hardware. Therefore, no measurements of execution times on a suitable hardware platform can be presented. However three different kinds of investigations have been carried out: first, a visualization of the 3d image cache updates was done for a flat environment so that the update events in the cache can be shown in two dimensions. Second, a frame sequence was rendered on a graphics workstation having support for gouraud shaded polygons (an INDIGO R3000 capable of 39k triangles/second) at a small screen resolution to minimize the effects of the software implementation of textured polygons (where the pixel fill rate is the bottleneck for triangles covering more than a few pixels). Third, the performance of the 3d image cache on a suitable hardware platform was simulated by replacing the operations not available on the R3000 system by other operations which were selected to match the execution speed of the desired operations on the required high-end graphics workstation. The statistics for all three test are shown in figures 6 to 8.

5.1 Visualization of cache behaviour

In the following investigation a very large and complex scene is assumed to be stored in the k-d-tree of the 3d image cache. For illustration purposes the scene is assumed to be flat enough so that no subdivision occurs along the projecting z-axis. As a worst case the scene is of homogenous complexity and the k-d-tree is always subdivided to the predetermined maximum. (In a typical walkthrough scenario some of the bboxes will be empty and therefore will not be present in the k-d-tree.) The diagram of figure 6 and the images of figure 9 address the question: which impostors need to be regenerated during a diagonal walkthrough of the scene.

Views from above show the necessary impostor updates for every 6^{th} frame of a total of 120 frames. A red square indicates a leaf bbox in the k-d-tree for which a new impostor was generated in the current frame. A green square indicates an intermediate bbox for which the impostor could be updated by reusing the impostors of it's sub-boxes. White squares indicate bboxes for which an available impostor was used. In the black area impostors cannot be used because the maximum texture resolution of 256 by 256 pixels would have been exceeded. Note how the used level of the k-d-tree adapts to the viewer position and accounts for the lower amount of coherence near the user. This pattern is entirely due to the chosen maximum texture resolution and the decision whether a bbox can be replaced by an impostor or not.
Some green squares in figure 9 contain smaller squares signifying that the corresponding impostor was generated by combining impostors of subnodes. The black bordered squares indicate the level of nodes in the k-d-tree used to generate the final image.
For each of the 120 frames the number of reused impostors, the number of impostors which could be generated from sub-box impostors (combined impostors in figure 6) and the number of impostors which needed to be regenerated (new impostors) is shown and should be compared to the total number of 341 bboxes in the k-d-tree. The total amount of memory required for the texture maps of the impostors is given in 100k bytes.
As can be seen a small fraction of impostors in the k-d-tree actually needs regeneration. Only those objects or parts of the scene, which lie inside such a bbox need to be rendered for the current frame. Combining impostors from impostors one level down the k-d-tree is a cheap operation and saves drawing the whole bbox contents.
In all simulations no clipping to the viewing frustum was assumed. If the 3d image cache is held up to date without considering viewing direction or clipping to the viewing frustum, rapid rotation of the viewing direction can be performed efficiently because most of the needed image data is already in the cache - only the scene portion around the user needs to be drawn as well as the impostors.

5.2 Simulation of cache behaviour

The next two tests were run for a scene of a procedurally generated forest consisting of 100 trees (approx. 105000 polygons, see figure 10 lower right). The first frame sequence generated depicts a sideward translation in front of the forest with 100 generated frames. The k-d-tree of the 3d image cache had a maximum depth of four with a total of 585 bboxes. The left graph of figure 7 compares new impostors, combined impostors and reused impostors. The amount of bboxes which could not be replaced by an impostor is shown in blue. The right graph of figure 7 shows new impostors, combined impostors and reused impostors of a total of 585 for a zoom from one corner of the forest towards its centre. The amount of bboxes which could not be replaced by an impostor is also given. Although it seems that the number of used impostors is comparable to the number of bboxes drawn using the primitives it will be obvious from the drawing times that a far larger part of the scene is covered by these impostors.
To approximate the performance of the 3d image cache on a suitable graphics workstation as close as possible the functionality not supported in hardware on the available R3000 workstation were replaced by operations which are equivalent in execution time to the required operations on texturing hardware. From data sheets of high end graphics workstations it can be seen that textured polygons are about twice as costly to draw as gouraud shaded polygons because of the additional access to texture memory for every pixel. Therefore textured polygons were replaced by drawing a gouraud shaded polygon with the same vertices twice. The texture definition operation used for an impostor update was replaced by two read operations of the image data from the frame buffer - the second transfer accounts for the copying of the image to the texture memory.
Although this test was run on a machine with no hardware support for texture mapping the frame times for the proposed rendering method were only longer a few times than without caching. Moreover, changes of the viewing direction would have been possible at almost no cost, as the necessary image data is in the cache and is valid. The left graph of figure 8 compares the times to render the frames for both rendering without caching and rendering using the 3d image cache (in 1/10 of a second) for the translation sequence, the right graph for the zoom in sequence. Drawing times (shown in figure 8) using the 3d image cache stay below the frame times

of usual polygonal rendering even though the impostors were also updated for the part of the scene not currently in the user's field of view.

6 Conclusions and future work

This paper presented the three dimensional image cache, a hierarchical data structure to speed up any kind of rasterized rendering on a graphics workstation which supports texture mapping in hardware. The 3d image cache uses dynamically generated impostors to store image data for the primitives contained in the nodes of a k-d-tree. During rendering these impostors are reused for several frames and are updated only if the differences to the image of the original primitives would exceed a pre-specified threshold (i.e. one pixel).
Impostor updates for leaves of the k-d-tree involve drawing the primitives. The impostors for intermediate nodes are refreshed from the image data stored in their subnodes.
If levels of details are available for the objects in the scene, a static level of detail selection algorithm can be employed to bound the amount of rendering load per node content. In this case the hierarchical caching of image data need not be used resulting in less texture memory requirements.
As the system has been implemented using the portable graphics library OpenGL future work will be to tune the system to a graphics platform supporting the required operations in hardware. Such platforms include the Pixel Flow system developed at the University of North Carolina and the SGI Reality Engine system.
In the future the system will be employed in a large environment navigation tool which pages in the environment portions around the user from secondary storage automatically.
Moreover the 3d image cache will be incorporated into an existing animation system based on ray tracing as the presented algorithm is independent of the method used to render primitives. Considerable speed up should be possible for this kind of rendering as well.

References

[1] Chen, Shenchang Eric and Lance Williams, *"View Interpolation for Image Synthesis"*, Computer Graphics (SIGGRAPH `93 Proceedings) (Aug. 1993) pp 279-288.

[2] Chen, Shenchang Eric, *"Quicktime VR - An Image-Based Approach to Virtual Environment Navigation"*, Computer Graphics (SIGGRAPH `95 Proceedings) (Aug. 1995) pp 29-38.

[3] Clark, James H., *"Hierarchical Geometric Models for Visible Surface Algorithms"*, Communications of the ACM 19 10 (Oct. 1976) pp 547-554.

[4] Funkhouser, Thomas A. and Carlo H. Séquin, *"Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments"*, Computer Graphics (SIGGRAPH `93 Proceedings) 17 (Aug. 1993) pp 247-254.

[5] Heckbert, Paul and Michael Garland, *"Multi-Resolution Modeling for Fast Rendering"*, Proceedings of Graphics Interface `94 (May 1994) pp 43-50.

[6] Helman, James L., *"Designing Virtual Reality Systems to Meet Physio- and Psychological Requirements"*, Applied Virtual Reality Course (ACM SIGGRAPH `93) (Aug. 1993) pp 115-1 - 5-20.

[7] Maciel, Paulo W. and Peter Shirley, *"Visual Navigation of Large Environments Using Textured Clusters"*, Symposium on Interactive 3D Graphics (April 1995) pp 95-102.

[8] Regan, Matthew and Ronald Post, *"Priority Rendering with a Virtual Reality Address Recalculation Pipeline"*, Computer Graphics (SIGGRAPH `94 Proceedings) (July 1994) pp 155-162.

[9] Schaufler, Gernot, *"Dynamically Generated Impostors"*, MVD'95 Workshop (Nov. 95) pp 129-136.

[10] Schaufler, Gernot and Wolfgang Stürzlinger. *"Generating Multiple Levels of Detail for Polygonal Geometry Models"*, 2nd Eurographics Workshop on Virtual Environments 95 (Jan. 1995) pp 53-62.

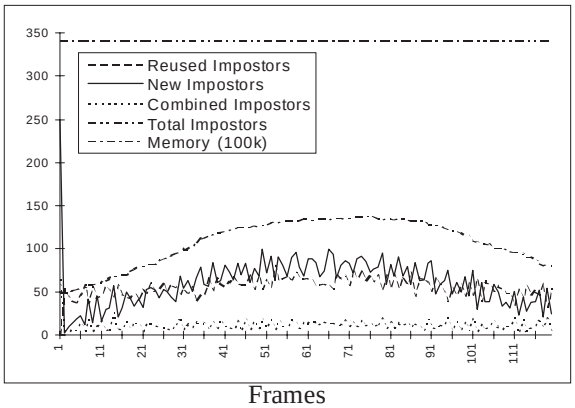


Figure 6. Statistics to Simulation of Frame Sequence in Section 5.1

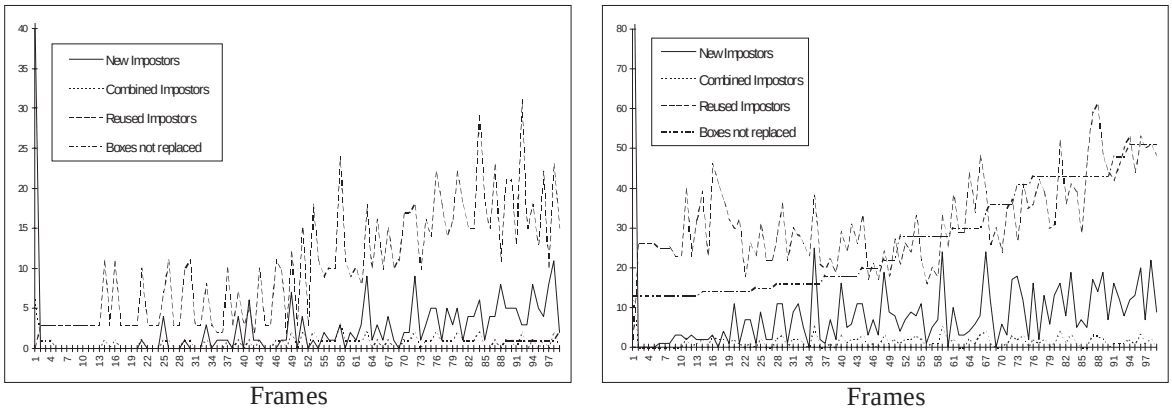


Figure 7. Statistics to Simulation of Frame Sequence in Section 5.2

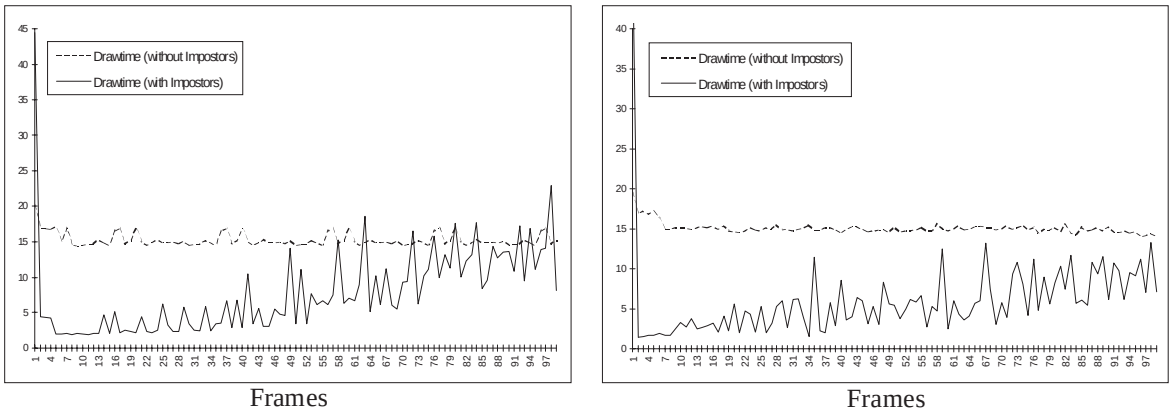


Figure 8. Comparison of Drawing Times in Section 5.2

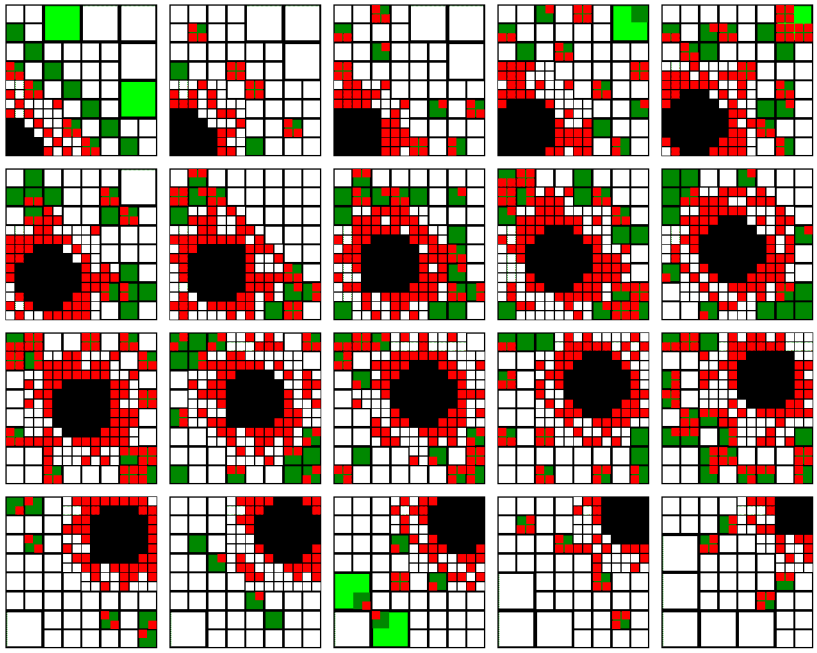


Figure 9. Selected Frames of Simulation Sequence Described in Section 5.1

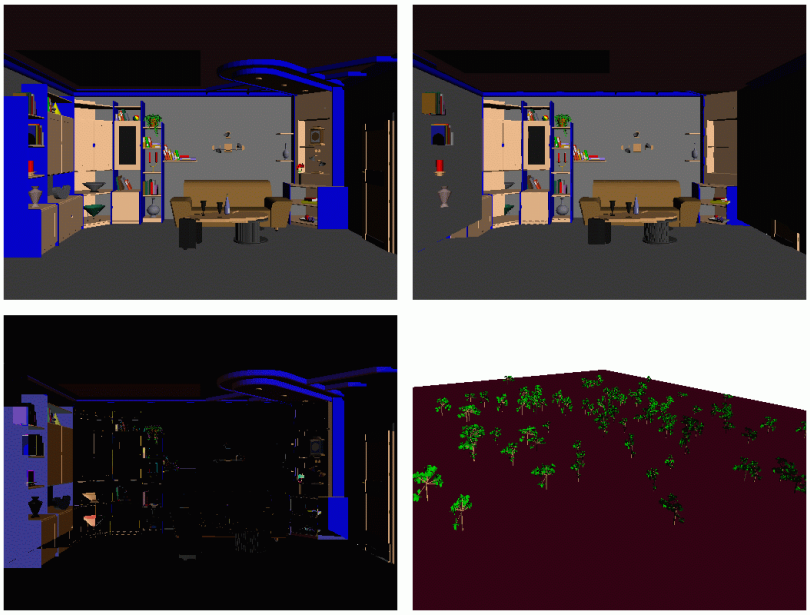


Figure 10. upper left: Original Living Room Scene; upper right: Problems with Object Impostors; lower left: Difference Image; lower right: Tree Scene

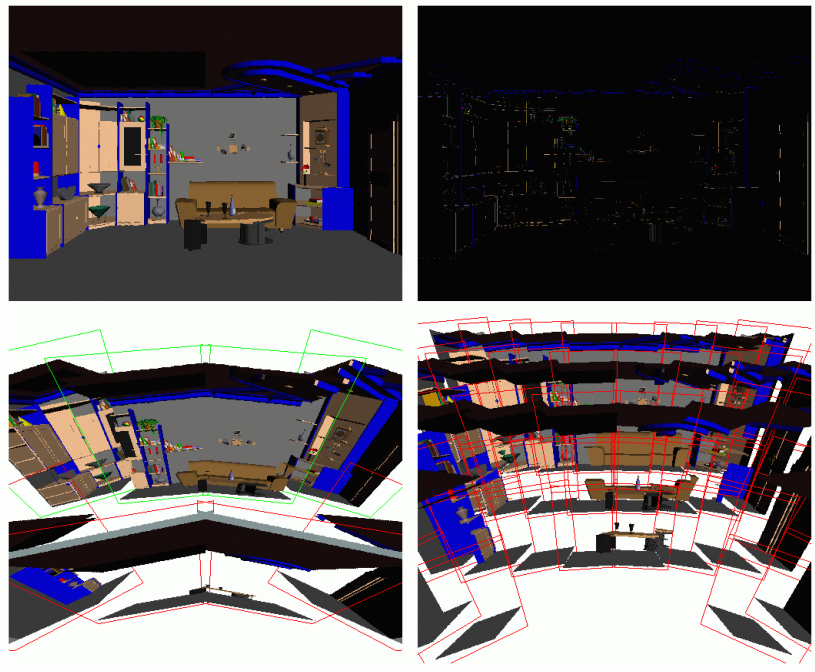


Figure 11. upper left: Result Generated with 3d Image Cache; upper right: Difference Image; lower row: Sample Bird's Eye Views of Cache Contents

Determining Geometry from Images

Richard Szeliski

Microsoft Research

SIGGRAPH Course on Image-Based Modeling, Rendering, and Lighting

July 25, 2000

Computer Vision

Computer Vision is the *inverse* of Computer Graphics:

- computer graphics:
  - given a 3D model, render it
- computer vision
  - given some images, create a 3D model

This talk describes some techniques for recovering 3D geometry from images.

Motivation

- model building for virtual reality, animation, and CAD is slow and tedious
- animators and designers want photo-realistic (texture-mapped) models
- video input, display, and processing hardware becoming ubiquitous(multimedia)
- computer vision algorithms becoming more mature and reliable

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 3

Applications

- recover camera location to superimpose graphics on image [Gleicher 92]
- extract texture maps from real world [Beardsley96, Debevec96]
- create a 3-D model object or world model, without extensive interactive modeling

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 4

Applications (example)

- 3D model building example



octree 3D curves texture-mapped

Outline

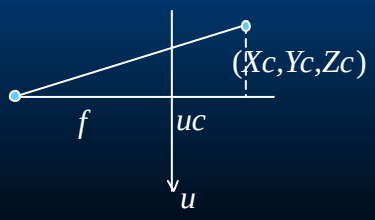
- camera calibration
- pose estimation (view correlation)
- triangulation
- structure from motion
- feature matching (correlation)
- stereo matching (dense shape estimates)
- volumes (octrees) from silhouettes
- surface curves from profiles
- inverse texture mapping
- applications

Camera calibration

- determine camera *internal* (focal length) and *external* (pose) parameters from known 3D points
- forward projection equations

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [\mathbf{R}]_{3\times3} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} f & 0 & u_c \\ 0 & f & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$



SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 7

Camera matrix calibration

- directly estimate 11 unknowns in 3×4 matrix projecting 3D ⇒ 2D

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- bring denominator over, solve set of linear equations

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 8

Camera matrix calibration

- Advantages:
  - very simple to formulate and solve
- Disadvantages:
  - doesn't compute internal parameters
  - more unknowns than true degrees of freedom
  - need a separate camera matrix for each new view

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 9

Pose estimation

- once the internal camera parameters are known, can compute camera pose
- application: superimpose 3D graphics onto video
- possible solution techniques:
  - use standard calibration code [Tsai87]
  - use *view correlation* [Bogart91]
  - use *through the lens camera control* [Gleicher92]
  - other techniques from computer vision

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 10

Triangulation (Stereo)

- Problem: Given some points in *correspondence* across two or more images (taken from calibrated cameras), $\{(u_j,v_j)\}$, compute the 3D location $\mathbf{X}$

Triangulation (Stereo)

- **Method I**: intersect viewing rays in 3D, minimize:

$$\arg\min_{\mathbf{X}} \sum_j \min_{s_j} \|\mathbf{C}_j + s_j\mathbf{V}_j - \mathbf{X}\|^2$$

  - $\mathbf{X}$ is the unknown 3D point
  - $\mathbf{C}_j$ is the optical center of camera j
  - $\mathbf{V}_j$ is the *viewing ray* for pixel (u_j,v_j)
  - s_j is unknown distance along $\mathbf{V}_j$
- advantage: geometrically intuitive

Triangulation (con't)

- **Method II**: solve linear equations in **X**
  - advantage: very simple
- Method III: non-linear minimization
  - advantage: most accurate (image plane error)

Structure from motion

- Given many points in *correspondence* across several images, $\{(u_{ij},v_{ij})\}$, simultaneously compute the 3D location $\mathbf{X}_i$ and camera (or *motion*) parameters $\mathbf{M}_j$
- two main variants: calibrated, and uncalibrated (sometimes associated with Euclidean and projective reconstructions)
- long history of research algorithms

[Longuet81,Tomasi92,Weng93a,Szeliski94e,Beardsley96a]

Structure from motion (con't)

- Simple iterative algorithm used for face reconstruction[Pighin98] assuming roughly known geometry and pose
  - assume $(u_c,v_c) = (0,0)$, but f is unknown

$$u_{ij} = s_j \frac{\mathbf{r}_j^x \cdot \mathbf{X}_i + \mathbf{t}_j^x}{1 + h_j \mathbf{r}_j^z \cdot \mathbf{X}_i}, v_{ij} = s_j \frac{\mathbf{r}_j^y \cdot \mathbf{X}_i + \mathbf{t}_j^y}{1 + h_j \mathbf{r}_j^z \cdot \mathbf{X}_i}$$

    where $h_j = 1/\mathbf{t}_j^z$ is the inverse distance to object, and $s_j = f_j/\mathbf{t}_j^z$ is a world-pixel scale factor
- *advantage*: works well for narrow fields of view when f and $\mathbf{t}_j^z$ are hard to estimate

Structure from motion (con't)

- bring denominator over to l.h.s.
- iteratively solve for: s_j, $\mathbf{X}_i$, $\mathbf{R}_j$, $\mathbf{t}_j^x$ and $\mathbf{t}_j^y$, h_j
- all equations are linear, except for $\mathbf{R}_j$, which is linearized by using a small angle (instantaneous velocity) approximation

Structure from motion: limitations

- very difficult to reliably estimate structure and motion unless:
  - large (x or y) rotation *or*
  - large field of view and depth variation
- camera calibration important for Euclidean reconstructions
- need good feature trackers
- postprocessing of the resulting 3-D points?

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 18

Feature matching (correlation)

- Find corresponding points in image video sequence
  - one simple technique: find two patches with minimal summed squared error[Anandan89]

$$E_{xy}(u,v) = \sum_{k=x-w}^{x+w} \sum_{l=y-w}^{y+w} [I_1(k+u, l+v) - I_0(k,l)]^2$$

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 19

Feature matching (optic flow)

- need *sub-pixel* precision to get best registration
- solution: Taylor series expansion of image function [Lucas81a]

$$E(\mathbf{u} + d\mathbf{u}) = \sum_i (e_i + g_i \cdot d\mathbf{u})^2$$

where $\mathbf{x}' = \mathbf{x}+\mathbf{u}$, $e_i = I_1(\mathbf{x}') - I_0(\mathbf{x})$, $\mathbf{g}_i = \nabla I_1(\mathbf{x}')$

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 20

Feature matching (optic flow)

- solve 2×2 system

$$\left[\sum_i g_i g_i^T\right] d\mathbf{u} = -\left[\sum_i e_i g_i\right]$$

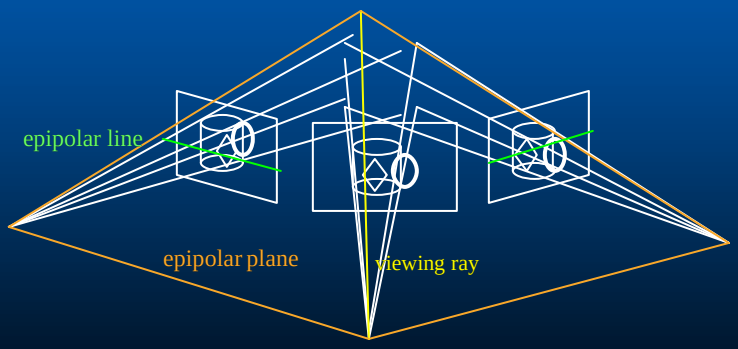
- use a *coarse-to-fine* pyramid to speed up search [Bergen92a]
- related to *Brightness Constancy Equation* [Horn81]

$$I_x u + I_y v - I_t = 0$$

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 21

Stereo: epipolar geometry

- Match features along epipolar lines



Stereo: epipolar geometry

- for *two* images (or images with collinear camera centers), can find epipolar lines
- epipolar lines are the projection of the *pencil* of planes passing through the centers
- **rectification:** warping the input images (perspective transformation) so that epipolar lines are horizontal [Faugeras '93; Loop & Zhang '99]

Stereo: dense depth

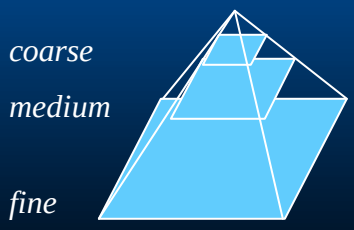
- apply feature matching criterion at *all* pixels simultaneously
- search only over epipolar lines (many fewer candidate positions)

- can also match features such as lines

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 24

Stereo: hierarchical matching

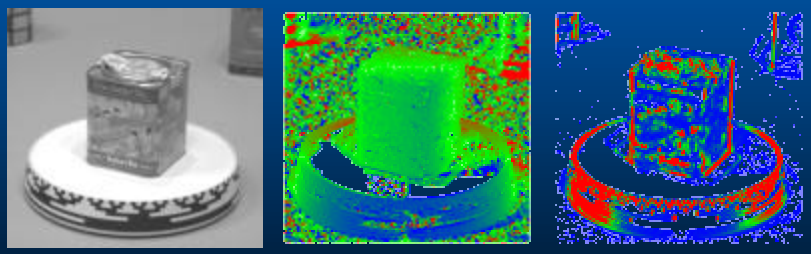
- Use coarse-to-fine search in an image pyramid to handle larger displacements
  [Bergen *et al.*'92]



SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 25

Stereo: certainty modeling

- Compute certainty map from correlations



input depth map certainty map

Stereo: dense depth

- recovered depth map can be used for *view interpolation* [Chen93,Szeliski95,Seitz96]

input depth image novel view

[Matthies,Szeliski,Kanade'88]

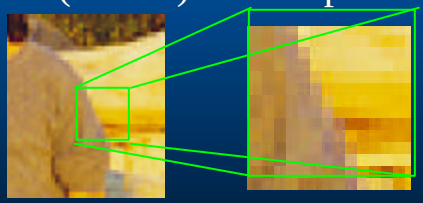
Dense Stereo Matching

- Advantages:
  - gives detailed surface estimates
  - multi-view aggregation improves accuracy
- Limitations:
  - narrow baseline ⇒ noisy estimates
  - fails in textureless areas
  - sparse, incomplete surface
  - sensitive to non-Lambertian effects

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 28

Stereo matching: limitations

- problems at and near occlusions
- incorrect color extraction, no partial occupancy in (mixed) border pixels



- solution: simultaneously recover *disparities*, *colors*, and *opacities*

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 29

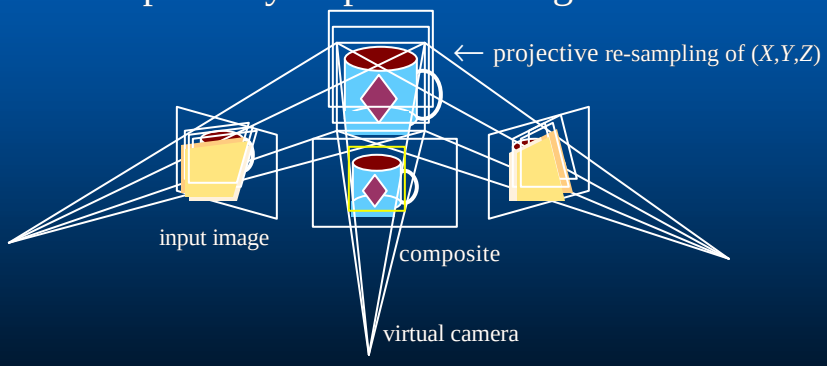
Multi-Image Scene Recovery

- Goals of new stereo algorithm
  - simultaneously recover *disparities*, *colors*, and *opacities* (c.f. blue screen matting)
  - explicitly handle occlusions
  - true multi-frame setting [Collins, CVPR'96]
  - details in [Szeliski & Golland, ICCV'98]

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 30

Plane Sweep Stereo

- Sweep family of planes through volume



  - each plane defines an image ⇒ composite homography

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 31

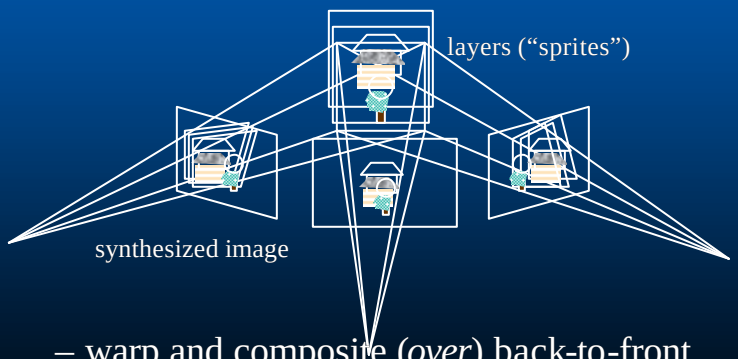
Plane Sweep Stereo

- For each depth plane
  - compute composite (mosaic) image — *mean*

  - compute error image — *variance*
  - convert to confidence and aggregate spatially
- Select winning depth at each pixel

Plane Sweep Stereo

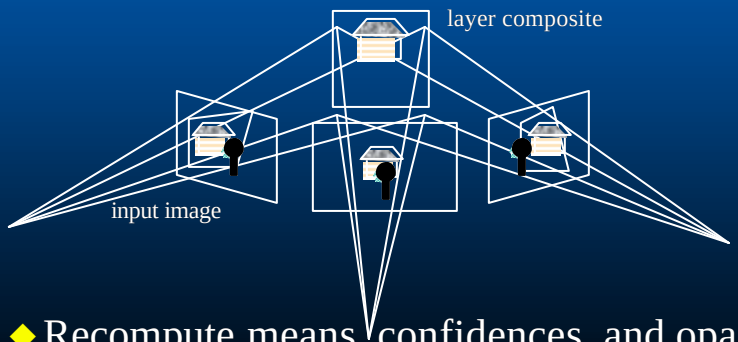
- "Stack of acetates" model (related to LDI)



  - warp and composite (*over*) back-to-front

Plane Sweep Stereo

- Compute *visibility* each input/layer pair



- Recompute means, confidences, and opacities

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 34

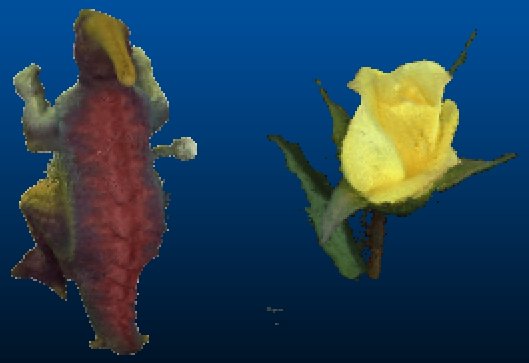
Voxel Coloring

- Generalizes plane sweep camera geometry
  - replace plane sweep with surface sweep
    [Seitz & Dyer, CVPR'97][Kutalakos & Seitz]

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 35

Voxel Coloring

- Results for dinosaur and rose

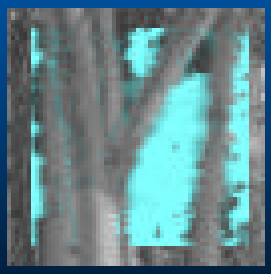


Stereo with Matting

- Estimate fractional opacities for pixels
  - adjust layer "sprites" (colors and opacities) to best match input images
  - optimization criteria:
    - re-synthesis error
    - color and opacity smoothness
    - prior distribution on opacities
  - corresponds to MAP Bayesian estimator

Stereo with Matting

- SRI Trees sequence example



input images

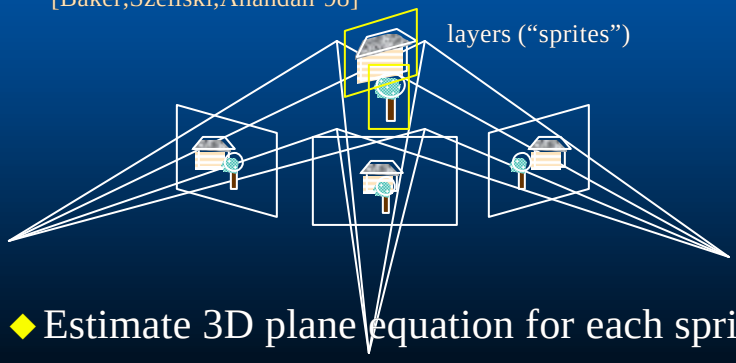
stereo layers

Stereo with Matting

- Advantages:
  - true multi-image matching
  - deals with occlusions and mixed pixels
- Limitations:
  - too many degrees of freedom (volume)
  - breaks up surfaces into “voxels”
  - no “sub-pixel” depths

Layered Stereo

- Use arbitrarily oriented sprites
  [Baker,Szeliski,Anandan'98]



- Estimate 3D plane equation for each sprite

Layered Stereo

- Assign pixel to different "layers" (objects, sprites)

Layered Stereo

- Track each layer from frame to frame, compute plane eqn. and composite mosaic

- Re-compute pixel assignment by comparing original images to sprites

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 42

Layered Stereo

- Resulting sprite collection

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 43

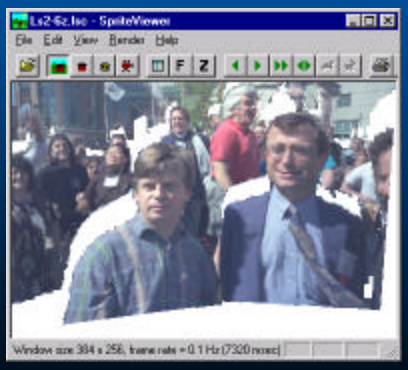
Layered Stereo

- Re-synthesize original or novel images from collection of sprites

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 44

Layered Stereo Demo

- *SpriteViewer*: renders sprites with depth



SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 45

Layered Stereo

- Per-pixel residual depth estimation
  - *plane plus parallax* [Anandan *et al.*]
  - *model-based stereo* [Debevec *et al.*]

  - better accuracy / fidelity
  - makes *forward warping* more difficult

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 46

Layered Stereo

- Advantages:
  - can represent occluded regions
  - can represent transparent and border (mixed) pixels (sprites have *alpha* value per pixel)
  - works on texture-less interior regions
- Limitations:
  - fails for high depth-complexity scenes
  - may need manual initialization / control

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 47

Multi-view Representation

- associate a depth (or motion) map with *each* input image (or some subset)

- enforce motion/depth *compatibility* across frames [Szeliski, CVPR'99]

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 48

Multi-view Representation

- no need to *discretize* space or determine the number of *layers*
- take *occlusions* into account
- capture natural *variation* in appearance
- better suited for view or frame *interpolation*

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 49

Multi-view matching framework

- minimize sum of inter-pair matching errors

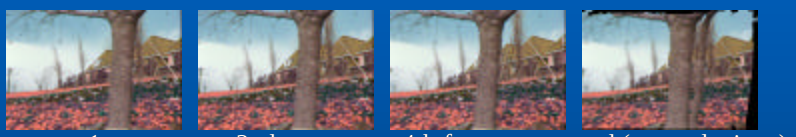
$$\sum_s \sum_{t \in N(s)} w_{st} \sum_{\boldsymbol{x}} r(I_s(\boldsymbol{x}_s) - I_t(\boldsymbol{x}_t))$$

where

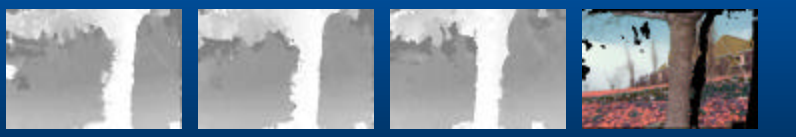
- s are the *key-frames* (or *key-views*)
- t are the *neighboring frames*
- $\boldsymbol{x}_s$ and $\boldsymbol{x}_t$ are corresponding pixels
- r is a robust penalty function

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 50

Experiments



1st 2nd 4th frames warped (no occlusions)

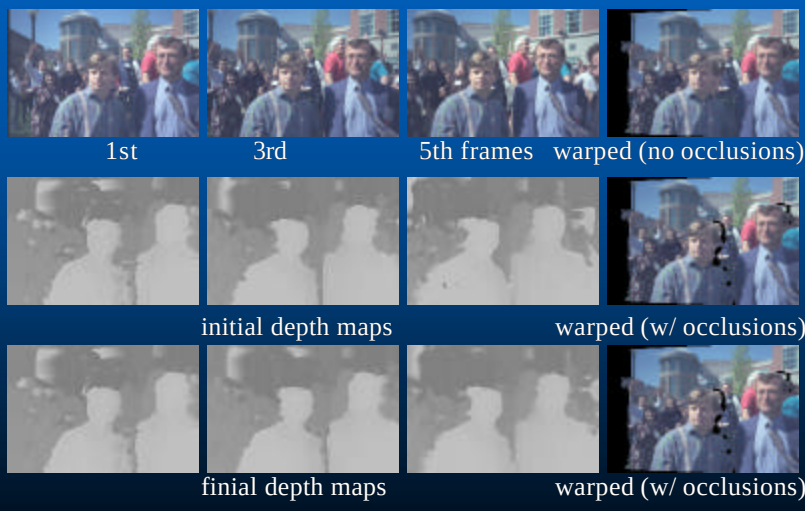


initial depth maps warped (w/ occlusions)

finial depth maps warped (w/ occlusions)

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 51

Experiments



SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 52

Volumes from silhouettes

- extract binary *silhouette* of object photographed against known background
- each silhouette + camera center defines enclosing conic region of space
- intersection of cones ⇒ bounding volume
- use octree representation of volume for efficiency [Szeliski93h]

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 53

Volumes from silhouettes

Cup on turntable example

Volumes from silhouettes

- Advantages:
  - simple to implement, fairly robust
  - fast execution
  - complete (closed) surface
- Disadvantages:
  - only produces *line hull*
  - limited resolution
  - sensitive to classification (thresholding)

Surface curves from profiles

- extract and link edges in each image
- match edges across image sequence
- infer 3-D location from 2 or more matched edges:
  - for *stationary edge* (surface marking, sharp crease), use regular triangulation
  - for smooth self-occluding *profile* (limb), use 3 or more edges, fit circular arc [Szeliski94]

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 56

Surface curves from profiles

Coffee jar example

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 57

Surface curves from profiles

- Advantages:
  - correct estimates at occluding contours
  - good for smoothly curved objects
  - provides intrinsic surface estimates, piecewise continuous surface mesh
  - works on interior surface markings
- Disadvantages:
  - fails in textureless *interior* areas
  - incomplete surface (not closed)

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 58

Inverse texture mapping (photometry)

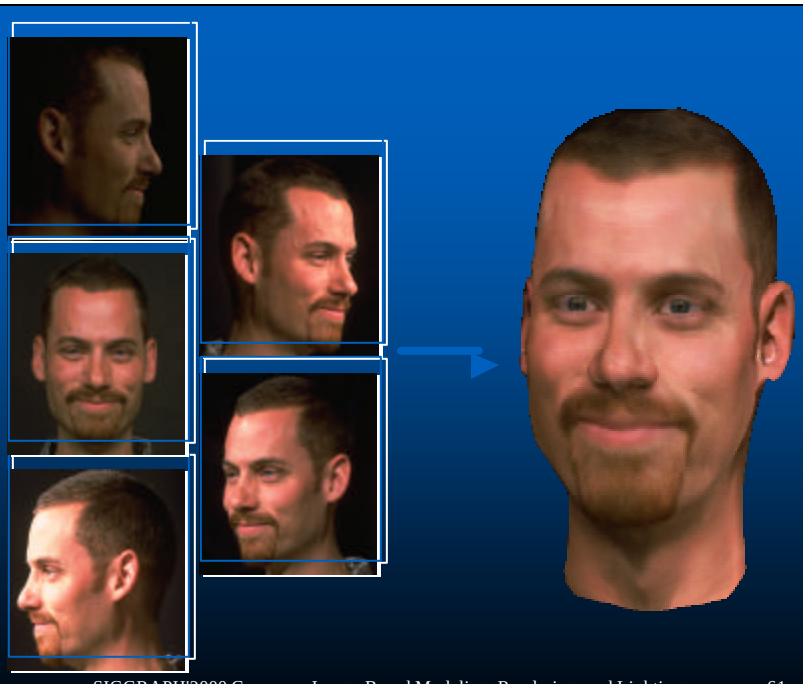
- recover color distribution over shape
- undo shading effects:
  - diffuse illumination
  - single source Lambertian
- weight contribution by surface normal
- smooth (and sharpen) results
  [Yu & Malik; Debevec]

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 59

Application: 3D face model building [Pighin98a]

- take several photos of a face from different views
- identify key points (eye and mouth corners, nose tip, ...) in each image
- recover camera position and coarse geometry using structure from motion
- add more correspondences, refine geometry, and interpolate to the rest of the mesh

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 60

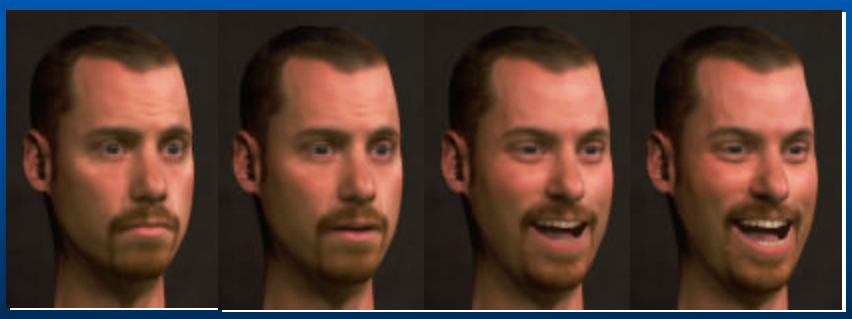


SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 61

Application: 3D face model building [Pighin98a]

- recover cylindrical texture map
- refine shape estimates using stereo
- animate by morphing between expressions

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 62



SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 63

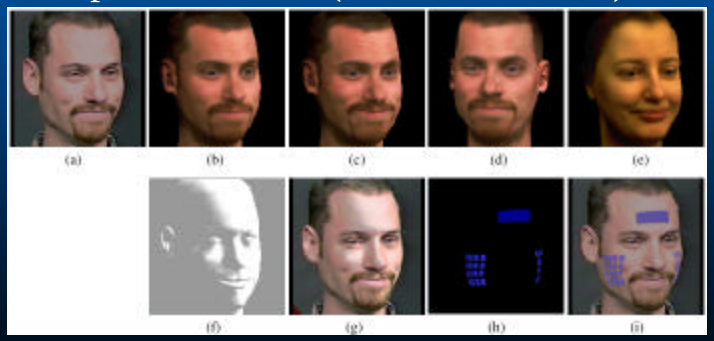
3D face model-based tracking

- Use “analysis by synthesis” to match 3D face model parameters to input video

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 64

3D model-based effects

- Change viewpoint, identity, illumination, or add special effects (scars, tatoos, …)



SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 65

Beyond IBMR

- What's next?
  - *Video-Based Rendering*
    - What if you could generate *computer video* instead of *computer graphics* ?
    - analyze video and synthsize new *frames*
    - *Video Rewrite* [Bregler *et al.*]
    - *Video Textures* [SIGGRAPH'2000]

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 66

Video Textures

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 67

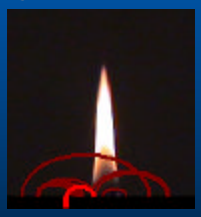
Video Textures

- Take a short *video clip* and turn it into an ∞ amount of continuous *video texture*
  - replace clips in Web pages and presentations
  - screen savers
  - alternative to 3D graphics animation?

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 68

Video Textures

1. find cyclic structure in the video




2. (optional) region-based analysis
3. play frames with random shuffle
4. smooth over discontinuities (morph)

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 69

Applications

- industrial applications
  - CAD/CAM
  - "3D Fax": collaborative design
  - architecture
  - biomedical (surgery, prostheses)
  - special effects (FX), virtual studio
  - fashion & clothing

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 70

Applications

- consumer applications:
  - 3D world building (travel, home sales, home page, ...)
  - 3D model construction (art, hobby, ..)
  - 3D avatar construction (heads)
  - "3D videophone"

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 71

Summary

- construct 3-D models from regular images
- processing steps:
  - calibrate cameras, determine pose
  - match features, build 3D geometry
  - extract appearance (texture maps)
- large variety of representations and algs.
- exciting new applications

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 72

To find out more

- general references on computer vision: [Ballard82,Horn86,Faugeras93,Nalwa93]
- recent survey of (some) 3D modeling techniques [Szeliski97]
- Computer Vision Home Page: http://www.cs.cmu.edu/afs/cs/ project/cil/ftp/html/vision.html
- Workshop on Image-Based Modeling and Rendering: http://graphics.stanford.edu/workshops/ibr98/

SIGGRAPH'2000 Course on Image-Based Modeling, Rendering, and Lighting 73

Bibliography

O. Faugeras, *Three-dimensional computer vision: A geometric viewpoint*, MIT Press, Cambridge, Massachusetts, 1993.

L. H. Matthies, R. Szeliski, and T. Kanade. Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision*, 3:209-236, 1989.

J. R. Bergen, P. Anandan, K. J. Hanna, and R, Hingorani. Hierarchical Model-Based Motion Estimation. In *Second European Conference on Computer Vision (ECCV'92)*, pages 234-252, May 1992.

R. T. Collins, A Space-Sweep Approach to True Multi-Image Matching, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)* , pages 358-363, June 1996.

S. M. Seitz and C. R. Dyer, Photorealistic Scene Reconstrcution by Space Coloring, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 1067-1073, June 1997.

Bibliography

R. Szeliski. From images to models (and beyond): a personal retrospective. In *Vision Interface '97*, pages 126-137, Kelowna, British Columbia, May 1997. Canadian Image Processing and Pattern Recognition Society.

R. Szeliski and P. Golland. Stereo matching with transparency and matting. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 517-524, Bombay, January 1998.

R. Szeliski and R. Weiss. Robust shape recovery from occluding contours using a linear smoother. *International Journal of Computer Vision*, 28(1):27-44, June 1998.

S. Baker, R. Szeliski, and P. Anandan. A layered approach to stereo reconstruction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 434-441, Santa Barbara, June 1998.

From images to models (and beyond): a personal retrospective

Richard Szeliski

In Vision Interface '97, pages 126-137, Kelowna, British Columbia, May 1997.

Canadian Image Processing and Pattern Recognition Society.

From images to models (and beyond): a personal retrospective

Richard Szeliski
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
szeliski@microsoft.com

Abstract

This paper surveys a number of techniques for extracting 3D geometry and photometry from multiple images. The paper describes the recovery of volumetric models from silhouettes, the extraction of 3D surface curves from profiles, the extraction and integration of 3D surface points from stereo, and a dynamic physically-based surface model for integrating the outputs of these procedures. It also presents recent work in image-based rendering, i.e., the representation of objects and scenes using collections of images which are warped and blended to create novel views.

1 Introduction

The reconstruction of 3D models from imagery has long been one of the central topics in computer vision [6, 5, 19]. Initially, it was believed that high-level 3D models were a prerequisite for image and scene understanding, as well as for robotics tasks such as navigation and manipulation. Today, it is widely accepted that visual information can be extracted and used at many levels. For instance, recognition tasks can often be solved using a collection of 2D views or features.

The last few years have seen a renewed interest in the acquisition of photorealistic 3D models for applications such as computer graphics, special effects, and the creation of virtual environments. Active range sensors such as laser scanners provide one way of acquiring such data [10, 43]. Passive image-based reconstruction techniques provide a potentially lower-cost alternative which can be applied to any sized object. These techniques, which reconstruct 3D shape from a number of views of an object or scene, also have the potential to capture the full visual richness of a complex object or scene through the use of view interpolation and view-dependent texture maps [24, 17].

This paper surveys a number of 3D reconstruction techniques which I have developed over the last ten years. My interest in 3D modeling began with an investigation of multiframe stereo algorithms [37]. I then developed a number of full 3D reconstruction algorithms using both surface [54, 65] and volumetric [55] models. I also became interested in the problem of surface reconstruction from sparse data [62, 64], and structure from motion techniques for recovering camera motion [59]. More recently, I have been investigating techniques for building large panoramic scene descriptions [56, 60, 57, 30] using a combination of projective structure from motion and multiframe stereo techniques. Most recently, I have been involved in research into image-based rendering [24].

This paper parallels the chronology of my research into these topics. Section 2 presents a technique for building volumetric models from binary foreground/background silhouettes. Section 3 describes our algorithm for estimating surface shape from tracked contours. Section 4 describes our method for integrating narrow-baseline stereo estimates using a cloud of 3D points with associated uncertainties. Section 5 discusses our technique for extracting and modeling 3D surfaces of arbitrary topology using collections of oriented surface elements. Section 6 discusses some approaches and recent results in extracting 3D surface color distributions (i.e., texture maps) from multiple images. Section 7 presents some recent results on image-based rendering. Section 8 briefly mentions issues related to camera calibration and structure from motion. We close with a discussion of potential applications and topics for future research.

2 Volumes from silhouettes

The first three reconstruction algorithms described in this paper use a controlled motion platform and a stationary video camera connected to a frame grabber. The motion platform is an inexpensive spring-loaded microwave turntable, to which we affixed a paper strip with an 8-bit pattern which encodes the turntable's angular position (Figure 1a) [54]. As the object spins on the turntable, video images are captured, the turntable angle is computed, and selected images (typically at 3° to 10° spacing) are retained for further processing. The relative position of the video camera and the turntable, and also the internal calibration parameters of the camera, must be known in order to recover 3D shape. We perform this calibration prior to model acquisition by placing a simple hexagonal pattern on the top of the

(a) (b) (c) (d)

Figure 1: Shape from silhouettes example: (a) input image, (b) silhouette image (non-white areas are considered part of the object), (c) lower-resolution octree model (5 levels), (d) higher-resolution octree model (6 levels).

turntable.[1] See Section 8 for alternative techniques for calibrating the camera and estimating its pose.

The first algorithm we developed for extracting 3D shape from multiple views constructs a volumetric representation of the object from the binary *silhouettes* of the object (Figure 1b). These silhouettes can easily be extracted from imagery by first acquiring an image of a blank turntable, and then doing simple pixel differencing.[2] This approach requires the colors of the object and background to be different, but this is not difficult to ensure in practice.

Each silhouette, together with the corresponding center of projection for the camera, defines a generalized cone in space within which the object must lie. The intersection of these cones in 3D defines a bounding volume for the object, which under many conditions is a reasonable approximation to the object's shape. Several different techniques have been developed in the past for computing this volume, including techniques which represent the volume as a polyhedron [53] and as an octree [42]. Our algorithm uses the octree representation, combined with the multi-resolution refinement algorithm described below, to minimize the overall amount of computation [55].

Octrees are tree-structured volumetric representations constructed by recursively subdividing cubes into eight equal sub-cubes [45]. Leaf nodes in the tree can be either *white* (empty) or *black* (occupied), while interior nodes are called *gray* and have children of different colors. Figure 2a shows a graphical view of a small octree, and Figure 2b shows its associated colored tree representation. The octree is usually more efficient than a pure voxel representation since the interior of the object is typically described by large cubes. For reasonably smooth objects, the total number of nodes in the tree is proportional to the object's surface area instead of its volume.

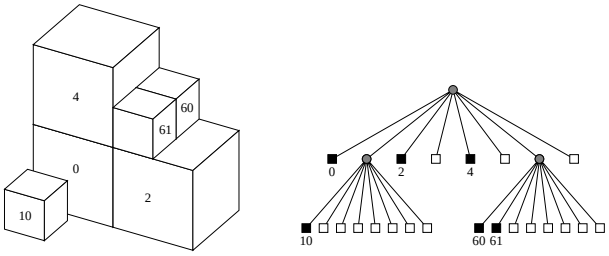


Figure 2: A simple two-level octree and its tree representation

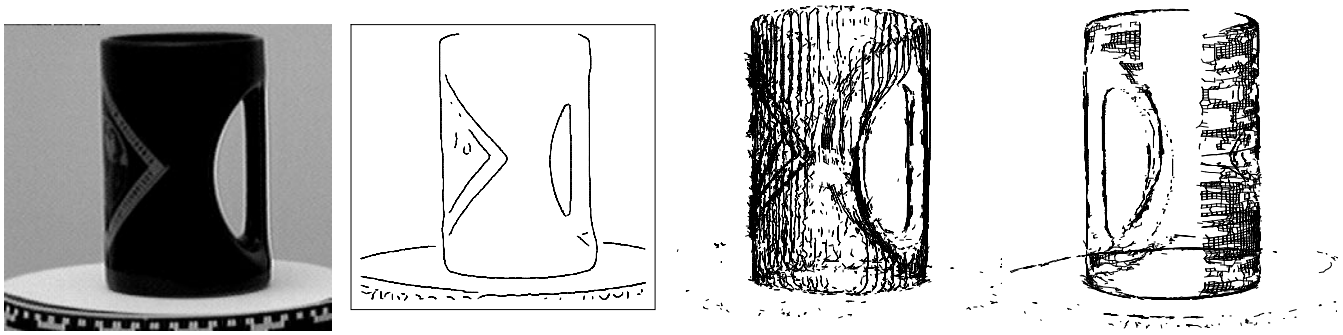
To construct the bounding volume from a sequence of silhouettes, we perform a series of forward projections from the 3D octree model to the 2D image plane using the known camera model (calibrated camera matrix plus turntable angle). Cubes in the octree which fall completely into the background can immediately be removed from the object model. Cubes which are partially inside the silhouette may be partially occupied, and must be subdivided to a finer resolution.

Our algorithm therefore constructs the octree in a hierarchical coarse-to-fine fashion. For a series of increasing octree resolutions, the inner loop of the algorithm refines the 3D volume by testing projected octree cubes against the sequence of silhouettes. After one complete revolution of the object, those cubes whose occupancy is uncertain are subdivided. Because the octree is completely constructed at one resolution before refining the next one, we perform fewer total computations than a non-hierarchical algorithm.

To rapidly compute whether a projected cube is inside the background or silhouette, we first compute a bounding box for the eight projected corners. A distance map, which is computed in time proportional to the image size, provides the minimum distance from any image point to the silhouette. This distance map applied to the bounding box gives a silhouette test that is $O(1)$

[1] The use of a planar calibration pattern requires that the aspect ratio of the camera be known. We have verified empirically that this ratio is close to 1 for our cameras.

[2] To close up small holes in the silhouette, we use a combination of morphological operations and connected components [44].



(a) (b) (c) (d)

Figure 3: Shape from contours example: (a) input image, (b) extracted edges, (c) reconstructed 3D edges (side view), (d) a portion of the surface mesh.

instead of $O(s)$ for each cube, where s is the number of pixels in the projected cube [55]. This optimization, combined with the efficiency of the hierarchical octree construction, results in an algorithm which can process several frames per second.

While our algorithm for recovering volumetric models from silhouettes can very quickly provide an approximate model of an object's shape, it suffers from a number of limitations. The greatest of these is the inability to sense certain kinds of cavities (such as the inside of the coffee cup). More precisely, our technique can recover the *visual hull* of an object, i.e., the complement to the space swept out by all lines that do not penetrate the object [32]. Other limitations include the susceptibility to misclassification errors which may be caused by specularities and accidental matches between object and background colors, as well as the limited precision available with volumetric representations.

3 Surface curves from profiles

The shape from silhouette method provides a bounding volume for a surface, but as described above it has limitations. To overcome some of these limitations, we developed an algorithm to directly compute 3D surface patches and curves from a sequence of image contours [65] (see [15, 71, 74, 12] for related techniques). Image contours come from surface markings, surface tangent discontinuities, and occluding contours. Surface markings and tangent discontinuities are viewpoint invariant, whereas occluding contours are not, and so cannot be used directly in two-frame stereo reconstruction algorithms.

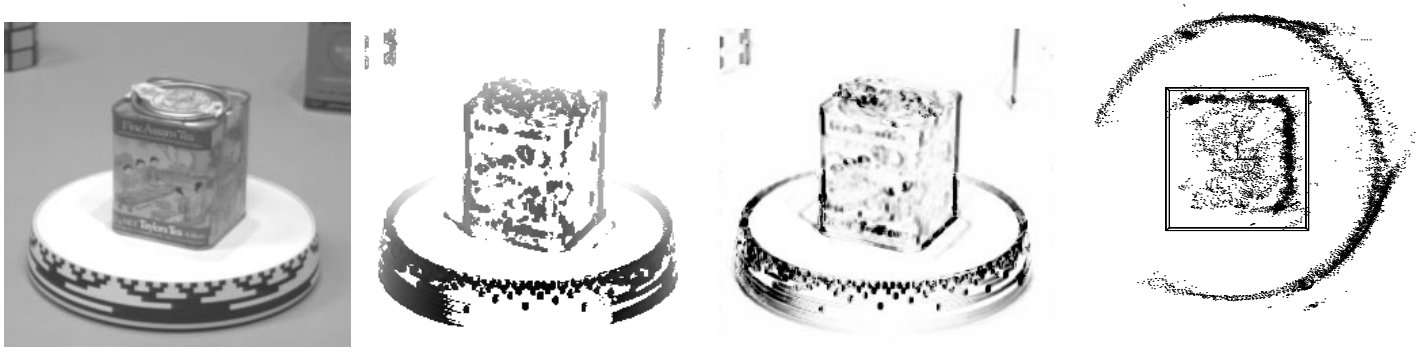
Given a sequence of images of an object, we extract edges and link them into contours (Figure 3b). The contours are tracked over the sequence of images, and 3D contours are estimated (Figure 3c). The 3D contours are linked by the trajectories of the tracks of the edge points in the images (Figure 3d). The known motion of the camera is used in two ways: it constrains the search for corresponding points during tracking, and it is used in the computation of the 3D points.

Edge detection is done using first order *steerable filters* [21], since they provide good angular resolution. Once discrete edgels have been detected, we use local search (based on proximity and continuity of orientation) to link the edgels into contours. Note that in contrast to some of the previous work in reconstruction from occluding contours [15], we do not fit a smooth parametric curve to the contour since we wish to directly use all of the edgels in the shape reconstruction, without losing detail.[3]

The pointwise correspondence between contours is based on the *epipolar constraint*. For two images, i.e., classical *binocular stereo*, this epipolar constraint is a set of straight lines which are the intersection of *epipolar planes* with the image plane [19]. For more than two images, the epipolar plane through a point is determined by the view direction at that point and the instantaneous camera velocity. The projection of the epipolar plane into the next frame, i.e., the epipolar line, is used to find the best matching edgel in the next frame. Our technique compares all candidate edgels within the epipolar line search range (defined by the expected minimum and maximum depths), and selects the one that matches most closely in orientation, contrast, and intensity. Once an initial estimate for the 3D location of an edgel has been computed, the search range can be dramatically reduced.

Given three or more edgels tracked with our technique, we compute the location of the surface and its curvature by fitting a circular arc to the lines defined by the view directions at those edgels. This curve fitting problem is *linear*, which allows us to use recursive least squares techniques [65]. To further improve the quality and reliability of our estimates, we apply *robust statistics* to reduce the effects of *outliers* which are due to grossly erroneous measurements as well as large changes in the surface curvature [28].

[3] However, we do perform a small amount of curvature-dependent smoothing along the curves to reduce noise. This can be viewed as part of the edge extraction stage.



(a) (b) (c) (d)

Figure 4: 3D depth map computed from *assam* image sequence: (a) first image in sequence, (b) depth map from flow (darker is nearer), (c) certainty in depth estimates (darker is higher certainty), (d) top view of 3D point cloud.

Once the circular arc is fitted, a point on the surface is computed which corresponds to the middle view direction. The topology of the surface in the form of a mesh is captured by maintaining the epipolar curves, which are given by the edge point trajectories, and the 3D contours. Figure 3c shows the complete set of 3D contours reconstructed from a 360° view of a cup, while Figure 3d shows a portion of the 3D mesh (contour curves plus epipolar curves).

Occluding contours sweep out surface patches on the *visible* region of the surface, which is the union of all critical sets that are not occluded. In general, these curves and patches will provide information in many places where the line hull differs from the original surface. The limitations of reconstruction from contours is that it may not produce a closed surface and the reconstruction degrades as the contours become parallel to the epipolar constraints. The technique may also fail in highly textured regions where it is difficult to consistently extract edges.

4 3D points from stereo

An alternative to extracting surface shape by matching 2D edges is to use dense (correlation-based) stereo, and then integrating the resulting depth maps into a complete 3D model [54].

Our algorithm first computes dense disparity maps from successive pairs of images in the input image stream. The images are re-sampled (*orthorectified*) so that corresponding epipolar lines are horizontal [19], and each image is interpolated horizontally by a factor of 4. Disparities are estimated using a Sum of Squared Differences (SSD) algorithm [3], since it provides not only sub-pixel disparity estimates, but also uncertainty estimates for each pixel. The sub-pixel disparity estimate at each pixel is computed by fitting a parabola to the minimum SSD value; the analytic minimum is used to compute the disparity, while the variance in this estimate is computed using the second derivative of the parabola and a local noise estimate [37].

Figure 4b shows the depth map computed from two frames of the video sequence, after thresholding out low-certainty pixels. Figure 4c shows the certainty (inverse variance). Notice how the estimates are more certain in highly textured regions.

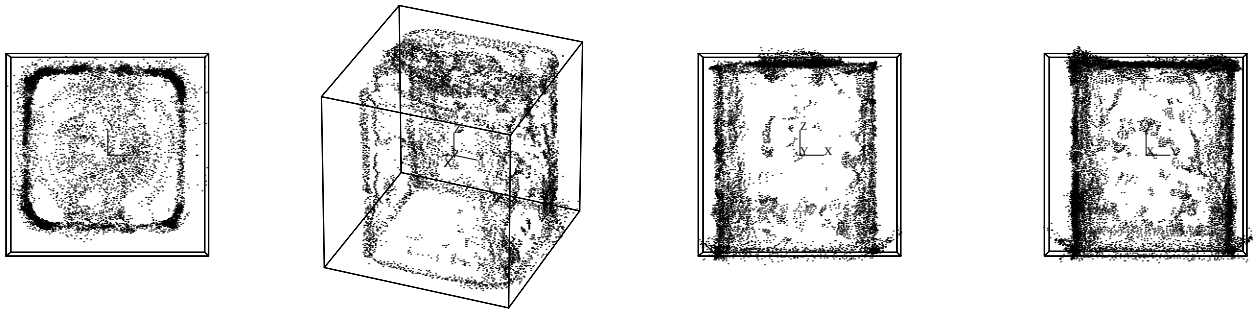
For each disparity estimate d we compute the corresponding 3-D object space location using triangulation.[4] We also compute a 3×3 covariance matrix for each point which characterizes the shape and magnitude of the point's positional uncertainty. The component of this covariance along the viewing ray is computed using the disparity estimate variance and the Jacobian of the inverse projection (triangulation) operator. The other two axes of the covariance ellipsoid can be chosen arbitrarily and their length (standard deviation) set to a suitably chosen constant.

The result of our two-frame stereo matching and backprojection into object space gives us a "cloud" of uncertainty-tagged points lying on the surface of the object (Figure 4d). As the object continues to rotate and more points are acquired, point collections from successive frames are merged in order to reduce the noise in point location estimates.

To merge neighboring 3D points from different frames, we start by computing an uncertainty-weighted distance measure. If this distance is sufficiently small, we merge the two points and replace them with a single measurement with a reduced uncertainty.

The problem with this simple approach is that there may be many candidate matches for a given point, especially if one elongated uncertainty ellipsoid overlaps several other points. To reduce this problem, we limit merges to points whose uncertainty ellipsoid major axes are nearly parallel and which also meet the previous distance criteria. In practice, we make the merging step simpler by re-projecting the 3D locations and their uncertainties into the camera image plane. Two points are

[4] Our *object-centered* coordinate reference frame is fixed to the top of the turntable and rotates with it.



(a) (b) (c) (d)

Figure 5: Final merged data from *assam* sequence: (a) top view, (b) oblique view, (c) front view, (d) side view. The wireframe bounding box is just a 3" reference cube aligned with the turntable top.

merged if their image plane centers lie within a small distance of each other (say, 1/2 pixel) and their depths overlap sufficiently (using a 1-D version of the uncertainty-weighted distance). The thresholds for merging points are set high enough so that neighboring measurements from the same frame are not merged (we want our final model to be at least as accurate as the input image) but low enough so that oversampling (the density of 3D points per image pixel) is not too great.

Figure 5 shows the results of processing a complete 250 image sequence. The data is shown as isolated points from 4 different views. As you can see, the overall shape of the tea tin is recovered well, although the surface is not very smooth. In general, these results are not as accurate as those obtained with our edge-based surface reconstruction technique, due mainly to the narrower baselines involved (the contour-based algorithm typically uses 5–7 frames for each reconstructed contour, whereas two-frame stereo was used for these results). However, the algorithm works better in highly textured areas where reliable edge extraction is difficult.

The algorithm we use for merging 3D points is related to other recent range-data merging algorithms [52, 70, 26, 16]. Unlike these algorithms, it explicitly models the 3D uncertainty associated with range measurements (as does [29]), which produces estimates which are more statistically optimal. Unlike these algorithms, however, it does not produce a single coherent surface. To produce such an estimate, we developed a novel topologically flexible surface interpolation algorithm, which we present next.

5 Oriented particle-based surface modeling

The problem of fitting 3D models to sparse range data has been extensively studied in computer vision. Popular models include include generalized cylinders [1], superquadrics [41], and triangular meshes [11]. Physically-based models, which have internal deformation energies and can be fitted through external forces to 2D images and 3D range data have also been widely studied [67, 18, 66, 38].

Unfortunately, all of these models require the user to specify the topology (and often the rough shape) of the object ahead of time. To overcome this limitation, we have developed a surface modeling and fitting system based on the concept of *oriented particles* (also known as surface elements, or *surfels*) [62, 64, 63].

Our surface modeling and reconstruction method has two components. The first is a dynamic particle system which discovers topological and geometric surface structure implicit in the data. The second component is an efficient triangulation scheme which connects the particles into a continuous global surface model that is consistent with the particle structure. The evolving global model supports the automatic extension of existing surfaces with few restrictions on connectivity, the joining of surfaces to form larger continuous surfaces, and the splitting of surfaces along arbitrary discontinuities as they are detected.

The most novel feature of our approach to surface reconstruction is the use of a *molecular dynamics* simulation in which particles interact through long-range attraction forces and short-range repulsion forces. In our system, each particle has an associated surface normal. To control the behavior of these particles, we designed new interaction potentials which favor locally planar or locally spherical arrangements of particles [62]. Thus, the oriented particles support smoothness constraints similar to those inherent in the deformation energies of physically-based surface models. When reconstructing an object of arbitrary topology, the particles can be made to "flow" over the data, extracting and conforming to meaningful surfaces.

Our oriented particles were used as the basis for an interactive surface modeling system [62]. With this system, users can spray collections of points into space to form elastic sheets, shape them using deformation tools, and then freeze the surfaces into the desired final shape. They can create any desired topology with this technique. For example, they can deform a flat sheet into an object with a protrusion and then change the topology

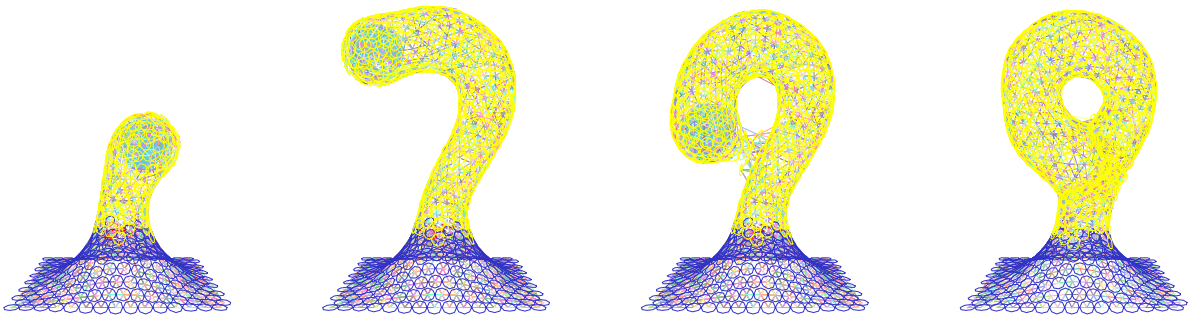


Figure 6: Forming a complex object. The initial surface is deformed upwards and then looped around. The new topology (a handle) is created automatically.

to create a looped handle (Figure 6). Forming such surfaces with traditional spline patches is a difficult problem that requires careful attention to patch continuities [36]. Other examples of modeling operations in this system include "cold welding" two surfaces together by abutting their edges, "cutting" surfaces with a knife-like constraint tool, and "creasing" surfaces by designating certain particles to be *unoriented* [62].

Another important application of our oriented particle systems is the interpolation and extrapolation of sparse 3-D data. This is a particularly difficult problem when the topology of the surface to be fitted is unknown. Oriented particles provide a solution to this problem by extending the surface out from known data points. This technique is particularly useful for interpolating the sparse position measurements obtained using the techniques described in this paper.

The basic components of our particle-based surface extension algorithm are two heuristic rules controlling the addition of new particles. These rules are based on the assumption that the particles on the surface are in a near-equilibrium configuration with respect to the flatness, bending, and inter-particle spacing potentials.

The first (*stretching*) rule checks to see if two neighboring particles have a large enough separation between them to add a new particle. If two particles are separated by an appropriate distance, we create a candidate particle at the midpoint. The second (*growing*) rule allows particles to be added in all directions with respect to a particle's local tangent plane. The rule is generalized to allow a minimum and maximum number of neighbors and to limit growth in regions of few neighboring particles, such as at the edge of a surface.

With these two rules, we can automatically build a surface from collections of 3-D points. We create particles at each sample location and fix their positions and orientations. We then start filling in gaps by growing particles away from isolated points and edges. After completing a rough surface approximation, we can release the original sampled particles to smooth the final surface, thereby eliminating excessive noise. If the set of data points is reasonably distributed, this approach will result in a smooth continuous closed surface.

In conjunction with fitting the surface, we can estimate local differential geometric quantities such as the principal directions and minimum and maximum curvatures. This can be achieved by simply adding an extra potential function that induces a torque around the local z axis and which forces the x and y axes to align themselves in the directions of minimum and maximum curvature [63]. The resulting system of oriented particles resembles the collection of interacting Darboux frames used by Sander and Zucker [46].

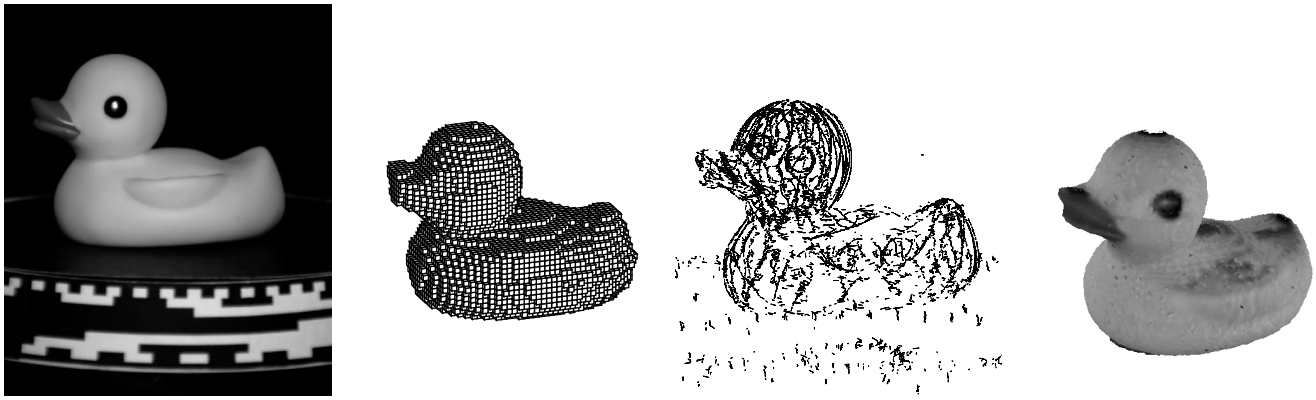
To summarize, oriented particles (surfels) allow us to interpolate and smooth both sparse and dense 3D data, without the need for any prior shape specification or user intervention (see [27, 23] for alternative approaches). Our system is flexible and general enough to model elastic surfaces of arbitrary topology with creases and discontinuities, and has also been extended to produce local measures of intrinsic geometry such as principal curvatures. Limitations of our approach include the large amounts of computation required to simulate the particle dynamics, the difficulty of tuning these dynamics, and occasional undesirable side-effects, such as the tendency of neighboring of surfaces to stick together.

6 Inverse texture mapping

The final step in creating a photorealistic model from imagery is to recover the color distribution over the surface of the object. In computer graphics, this is often called *texture map recovery* or *extraction*. A more general problem is to estimate the bidirectional reflectance distribution function (BRDF) at each pixel [47, 48].

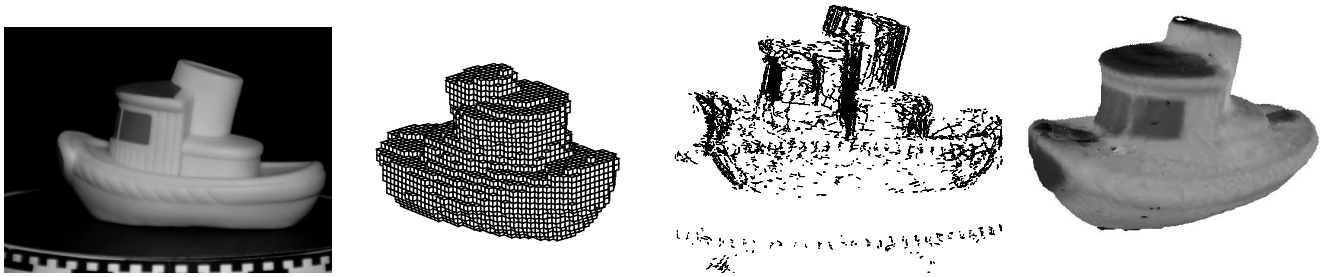
A simple way to obtain such distributions is to first segment the object or scene into piecewise planar regions, and to then project the input images onto these planar regions [13, 4, 17]. Another possibility is to use a triangulated surface model, and to project the images onto the triangles [7], or to just estimate the colors at the vertices (Figures 7–8).

Regardless of the approach used, the visibility of surfaces in each image must first be computed. The most convenient way to do this is to use a *z-buffer* or *item*



(a) (b) (c) (d)

Figure 7: Photometric recovery example (duck sequence): (a) input image from duck sequence, (b) octree model, (c) 3-D edges, (d) inverse texture-mapped model.

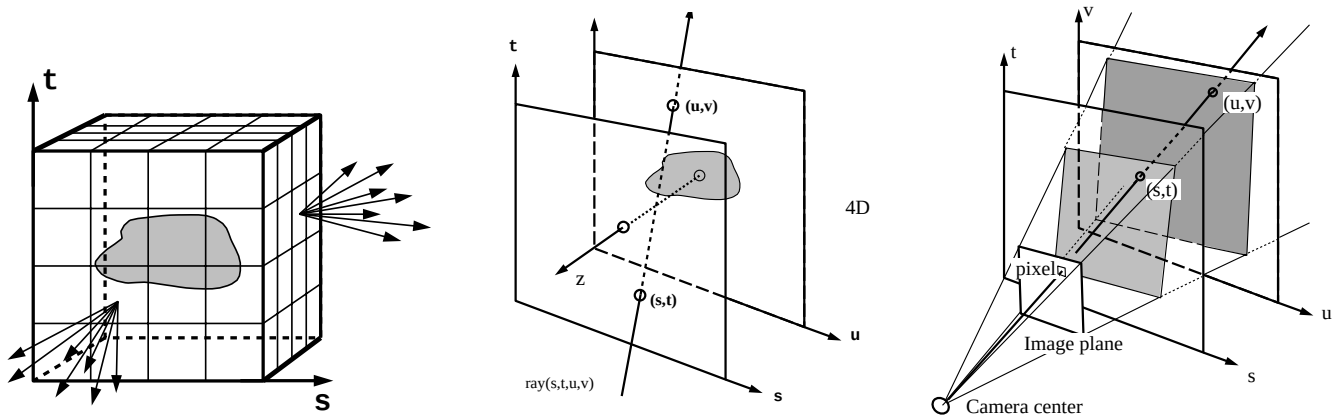


(a) (b) (c) (d)

Figure 8: Photometric recovery example (tug sequence): (a) input image from duck sequence, (b) octree model, (c) 3-D edges, (d) inverse texture-mapped model.

(a) (b) (c)

Figure 9: Projective depth recovery example—table with stacks of papers: (a) input image, (b) intensity-coded depth map (dark is farther back) (c) texture-mapped surface seen from novel viewpoint.



(a) (b) (c)

Figure 10: The Lumigraph: (a) the cube surface captures all light rays emanating from the object; (b) the 4D (s, t, u, v) parameterization of the rays; (c) computing the 4D coordinate of a pixel seen from a novel view.

buffer [72]. Furthermore, the contribution of each input pixel to the final color or texture map should be weighted so that pixels viewed at oblique angles contribute less. A convenient way to do this is to use the dot product between the camera viewing direction (or the ray corresponding to a pixel) and the surface normal. Finally, shading effects should be compensated for.

In the examples shown in Figures 7–8, we use a simple Lambertian shading model, with the light source placed directly behind the camera. While the recovered texture maps look plausible, they still suffer from visible artifacts. The first of these is a general darkening of colors near the edges of the objects' visible regions (e.g., the back of the duck, the top of the tugboat). This could be due to a failure of the Lambertian model, non-linearities in the image acquisition process, or simply a lack of information in highly compressed and poorly lit areas.

The second major problem is that the recovered color distributions (textures) are much blurrier than the original images. This is mostly due to residual inaccuracies in the recovered object's shape and possible miscalibrations in the acquisition system. Two approaches could be used to solve this problem. The first is to perturb the shape model in order to better align all of the input images [22]. A second approach is to simply deform each input image to better match the current re-rendered model, which should also compensate for other unmodeled sources of mis-registration errors. We are currently investigating these possibilities.

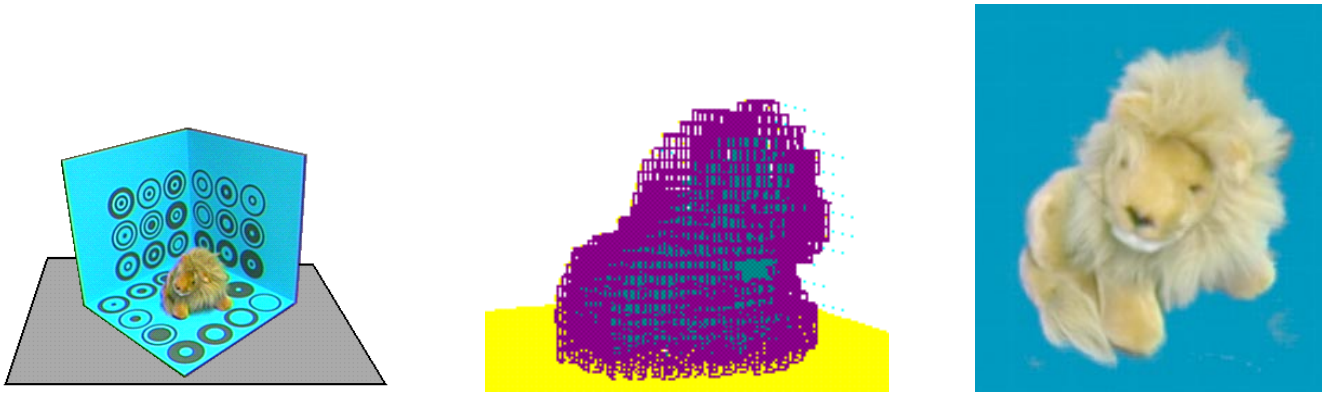
7 Beyond 3D models: image-based rendering

The techniques and algorithms presented thus far all have one goal in common: the reconstruction of detailed, photorealistic 3D models of arbitrary topology. My interest in this area was first stimulated while I was working on the recovery of accurate depth maps from multiframe stereo [37]. While we were able to obtain good depth maps and use these to re-project the scene into novel views, it quickly became apparent that only a very limited range of novel viewpoints could be supported with this representation.

Ironically, there has been a resurgence of interest in both the computer graphics and computer vision communities in such 2-$\frac{1}{2}$D representations. In computer graphics, this idea is called *image-based rendering* or *view interpolation* [14, 39]. The basic idea is to prerender (or alternatively, to capture with photographic means) a number of views of a complex scene. At rendering time, these images (views) are then combined using *morphing* techniques [9, 50], i.e., by warping and blending images. The advantage of this approach is that complex lighting simulations are eliminated (or precomputed), thus speeding up the rendering.

In computer vision, these representations are sometimes called *visual scene representations* [2]. The ideas here are similar, but with an emphasis on computing the necessary correspondences between input images to create large *mosaics* of scenes and to extract the required depth maps, often without any explicit knowledge of camera calibration or pose [31, 60]. One way to address this latter problem is to reconstruct a *projective* representation of the scene, i.e., one which is related to the true Euclidean structure through an unknown collineation [20, 25, 40, 59].

When combined with visual scene reconstruction, this often results in estimators based on first computing a planar homography, and then a residual parallax motion [31, 49, 58]. Figure 9 shows the depth map recovered using our plane + parallax technique, and also a novel view rendered by mapping one image onto the depth map. Note that the depth map is only known up to an arbitrary scale and planar offset. To generate the novel



(a) (b) (c)

Figure 11: Lumigraph example: (a) input image showing lion and calibration stage; (b) crude volumetric reconstruction of lion's shape; (c) novel image rendered from the Lumigraph.

view in Figure 9, we simply set the scale by hand to a reasonable value. Techniques for automatically deriving the proper mapping to novel views from additional correspondences have been developed [33].

The most recent work in image-based rendering suggests that correspondence between views may not be necessary if the spatial sampling of views is sufficiently high [34, 24]. The fundamental observation is that a set of images sampled on a surface enclosing any object (or conversely, surrounding some convex region of free-space) is sufficient to re-synthesize the appearance of the object or scene from any novel view in the free-space (Figure 10a). A convenient parametrization for the rays emanating from this volume uses two planes (Figure 10b). To compute the appearance of the object from a novel view, it then suffices to compute the 4D (s, t, u, v) index of each new pixel, and to look up the appropriate color in the 4D data set (Figure 10c).

Unfortunately, while this works perfectly well in theory, in practice the 4D light field must be sampled discretely, resulting in a finite number of images (say a 32×32 array of 256×256 images). When looking up a new ray value, we must interpolate this value from the neighboring discrete samples. If the neighboring samples are too far apart, then the blending of adjacent images will result in *ghosting* or double images, even if multi-dimensional (quadralinear) interpolation is used. This can be mitigated somewhat by pre-blurring the images (low-pass filtering in 4D space to remove aliasing), but results in blurry output images.

A sensible way to overcome this problem is to re-introduce some notion of depth or disparity into the 4D representation. Fortunately, we do not need to have a reliable depth estimate for each sample. A low-resolution geometric model is sufficient to remove most of the visible artifacts, and efficient rendering algorithms have been developed to exploit such models within a traditional graphics pipeline [24].

Figure 11 shows an input image from our Lumigraph system, a coarse geometric model computed using the volumetric technique described in Section 2, and a final re-synthesized image. Notice how this scene would be very difficult to model using a traditional 3D model because of the extremely fine detail in the lion's hair. The Lumigraph has similar advantages with respect to 3D models when it comes to capturing subtle photometric effects such as specularities and interreflections [24].

Our work on the Lumigraph suggests that there is a continuum of models and representations available for photorealistically representing the 3D world. At one end is the pure light field representation, which is simply a large collection of images taken from known vantage points. The rendering of such representations requires no correspondences, but does require very high sampling rates, and hence large storage costs (but see [34] on how these data sets may be compressed). At the other end are traditional texture-mapped 3D models. If appropriate reflectance models can also be estimated, then a standard graphics pipeline can be used to obtain photorealistic renderings (ignoring issues such as interreflections).

In between are the more interesting hybrid models. The Lumigraph uses a low-resolution representation of shape to perform a warping of the light field images in order to obtain better renderings at lower sampling rates. Visual scene representations and view interpolation techniques associate a depth value with each pixel in order to support re-projection through warping and blending. The rendering of such models on current graphics architectures (or in software) may be slow unless simplifying assumptions are made [14, 39]. Finally, view-dependent texture maps offer the possibility of increased realism when combined with traditional 3D models [17]. An alternative to view-dependent texture maps are full per-pixel BRDF models, but this research is still in its infancy [47, 48].

8 Camera pose recovery

Throughout our presentation, we have sidestepped two important issues which must be addressed before 3D models can be extracted from images, namely camera calibration and pose estimation. Camera calibration is a well-established discipline originating in photogrammetry [51]. Reg Willson's public implementation of Tsai's calibration algorithm [69] is freely available and widely used (see the Computer Vision Home Page at Carnegie Mellon University). Camera pose (location and orientation) estimation is intimately tied to camera calibration, and the same code is often used to accomplish both. The distinction is that it is sometimes preferable to calibrate the *intrinsic* camera parameters first using a more complex calibration setup , and to then use a simpler set of markers for determining the *extrinsic* (pose) parameters [19].

The most interesting design decision with respect to calibration and pose estimation is often the choice of stage and markers. For my research on shape from rotation, I used a simple hexagonal pattern affixed to the top of the turntable shown in Figure 4a [54]. For our Lumigraph project, we used a three-walled stage painted in blue (to support blue-screen matte extraction) with circular calibration targets (Figure 11a) [24]. In outdoor scenarios such as special effects for the film industry, it is common to place golf ball at known locations in the scene which are later manually removed from the live footage.

An alternative to marker-based pose estimation is to use structure from motion, i.e., to track a collection of features through several frames, and to then simultaneously reconstruct the point and camera positions. Many different structure from motion algorithms have been developed [35, 68, 73, 59]. Two of the more recent algorithms, which have been incorporated into full texture-mapped 3D model extraction systems are [4, 7]. The latter work is particularly interesting since it uses current structure and motion estimates to validate tracking hypotheses, and a robust (RANSAC) technique to throw out outliers.

Unfortunately, structure from motion algorithm are notoriously bad at estimating the true pose and structure unless a very wide range of views or baseline is used [61]. In some applications, however, such as the merging of live video and graphics, it may not be important to get a fully Euclidean reconstruction (unlike, say, for 3D model construction). Techniques which use stronger prior models of structure, e.g., building or room models, do not suffer from this problem as much, but can only be applied in restricted situations [8, 17].

9 Discussion

In this paper, we have surveyed a number of techniques for reconstructing and rendering photorealistic 3D models from multiple images. These techniques include both volume and surface reconstruction algorithms, physically-based surface models, and image-based rendering techniques. Each technique has its advantages and disadvantages. The selection of tools appropriate to the job will depend on the characteristics of the scene or object being imaged, the amount of computation time available, and the desired quality and flexibility at the output (rendering) stage.

Integrating all of these techniques into a coherent modeling system would be very interesting and useful, but would also require a careful design of the underlying representations. Such a task would go beyond a mere agglomeration of vision algorithms. It would imply interesting research problems in computer aided geometric design and multi-sensor fusion. Of course, each individual component (e.g., stereo matching) still has a lot of room for improvement, and we expect to see interesting future contributions in all of these areas.

Envisioning useful and exciting application of image-based modeling is another interesting way to predict their eventual deployment and to select research problems. Some possibilities include:

- *CAD/CAM*: the acquisition of shape and color models for design refinement and manufacturing
- *3D fax*: displaying the acquired models at a remote location or reconstructing them using stereolithography or other means
- *architecture*: constructing building and site models from photographs, with applications to design and remodeling, and even home sales
- *biomedical*: acquiring anatomical models (e.g., faces) for surgical planning and visualization
- *special effects (FX) and virtual studios*: merging live video or film with 3D graphics
- *3D world building*: for virtual environments (3D chat systems and games) and virtual travel
- *3D avatar construction*: head and body models to populate virtual environments, or to support a "3D videophone"

As these applications suggest, image-based 3D modeling is likely to be widely used in the future, and is bound to remain a fascinating area for future research.

Acknowledgements

It has been my honor to have worked over the last ten years with an outstanding collection of colleagues and collaborators, including Larry Matthies, Takeo Kanade, Demetri Terzopoulos, David Tonnesen, Richard Weiss, Sing Bing Kang, James Coughlan, Heung-Yeung Shum, Steven Gortler, Radek Grzeszczuk, and Michael Cohen, among others. The research described in this paper is a testament to their creativity, insight, and hard work.

References

[1] G. J. Agin and T. O. Binford. Computer description of curved objects. *IEEE Trans. Computers*, C-25(4):439–449, April 1976.

[2] *IEEE Work. Representations of Visual Scenes*, Cambridge, MA, June 1995.

[3] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *Int'l J. Computer Vision*, 2(3):283–310, January 1989.

[4] A. Azarbayejani and A. P. Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Trans. Pattern Analysis Machine Intelligence*, 17(6):562–575, June 1995.

[5] H. H. Baker. Three-dimensional modeling. In *Int'l Joint Conf. Artificial Intelligence*, pages 649–655, 1977.

[6] B. G. Baumgart. Geometric modeling for computer vision. Technical Report AIM-249, Artificial Intelligence Laboratory, Stanford University, October 1974.

[7] P. Beardsley, P. Torr, and A. Zisserman. 3D model acquisition from extended image sequences. In *European Conf. Computer Vision*, volume 2, pages 683–695, Cambridge, England, April 1996. Springer-Verlag.

[8] S. Becker and V. M. Bove. Semiautomatic 3-D model extraction from uncalibrated 2-d camera views. In *SPIE Vol. 2410, Visual Data Exploration and Analysis II*, pages 447–461, San Jose, CA, February 1995.

[9] T. Beier and S. Neely. Feature-based image metamorphosis. *Computer Graphics (SIGGRAPH'92)*, 26(2):35–42, July 1992.

[10] P. J. Besl and R. C.. Jain. Three-dimensional object recognition. *Computing Surveys*, 17(1):75–145, March 1985.

[11] J.-D. Boissonat. Representing 2D and 3D shapes with the Delaunay triangulation. In *Seventh Int'l Conf. Pattern Recognition*, pages 745–748, Montreal, July 1984.

[12] E. Boyer and M. O. Berger. 3D surface reconstruction using occluding countours. *Int'l J. Computer Vision*, 1997.

[13] I. Carlbom et al. Modeling and analysis of empirical data in collaborative environments. *Communications of the ACM*, 35(6):74–84, April 1992.

[14] S. Chen and L. Williams. View interpolation for image synthesis. *Computer Graphics (SIGGRAPH'93)*, pages 279–288, August 1993.

[15] R. Cipolla and A. Blake. Surface shape from the deformation of apparent contours. *Int'l J. Computer Vision*, 9(2):83–112, November 1992.

[16] B. Curless and M. Levoy. A volumetric method for building complex models from range images. Proc. SIGGRAPH'96 (New Orleans), pages 303–312, August 1996.

[17] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. Proc. SIGGRAPH'96 (New Orleans), pages 11–20, August 1996.

[18] H. Delingette, M. Hebert, and K. Ikeuichi. Shape representation and image segmentation using deformable surfaces. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 467–472, Maui, Hawaii, June 1991.

[19] O. Faugeras. *Three-dimensional computer vision: A geometric viewpoint*. MIT Press, Cambridge, MA, 1993.

[20] O. D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? In *European Conf. Computer Vision*, pages 563–578, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.

[21] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Analysis Machine Intelligence*, 13(9):891–906, September 1991.

[22] P. Fua and Y. G. Leclerc. Using 3–dimensional meshes to combine image-based and geometry-based constraints. In *European Conf. Computer Vision*, volume 2, pages 281–291, Stockholm, Sweden, May 1994. Springer-Verlag.

[23] P. Fua and P. Sander. Segmenting unstructured 3d points into surfaces. In *European Conf. Computer Vision*, pages 676–680, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.

[24] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. Proc. SIGGRAPH'96, pages 43–54, August 1996.

[25] R. Hartley, R. Gupta, and T. Chang. Estimation of relative camera positions for uncalibrated cameras. In *European Conf. Computer Vision*, pages 579–587, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.

[26] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *European Conf. Computer Vision*, volume 1, pages 117–126, Cambridge, England, April 1996. Springer-Verlag.

[27] W. Hoppe et al. Surface reconstruction from unorganized points. *Computer Graphics (SIGGRAPH'92)*, 26(2):71–78, July 1992.

[28] P. J. Huber. *Robust Statistics*. John Wiley & Sons, New York, 1981.

[29] A. E. Johnson and S. B. Kang. Registration and integration of textured 3-d data. In *Int'l Conf. Recent Advances in 3-D Digital Imaging and Modeling*, Ottawa, May 1997.

[30] S. B. Kang and R. Szeliski. 3-D scene data recovery using omnidirectional multibaseline stereo. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 364–370, San Francisco, June 1996.

[31] R. Kumar, P. Anandan, and K. Hanna. Shape recovery from multiple views: a parallax based approach. In *Image Understanding Work.*, pages 947–955, Monterey, CA, November 1994.

[32] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Analysis Machine Intelligence*, 16(2):150–162, February 1994.

[33] S. Laveau and O. D. Faugeras. 3-d scene representation as a collection of images. In *Int'l Conf. Pattern Recognition*, volume A, pages 689–691, Jerusalem, Israel, October 1994.

[34] M. Levoy and P. Hanrahan. Light field rendering. Proc. SIGGRAPH'96 (New Orleans), pages 31–42, August 1996.

[35] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[36] Charles Loop and Tony DeRose. Generalized B-spline surfaces of arbitrary topology. *Computer Graphics (SIGGRAPH'90)*, 24(4):347–356, August 1990.

[37] L. H. Matthies, R. Szeliski, and T. Kanade. Kalman filter-based algorithms for estimating depth from image sequences. *Int'l J. Computer Vision*, 3:209–236, 1989.

[38] T. McInerney and D. Terzopoulos. A finite element model for 3D shape reconstruction and nonrigid motion tracking.

In *Int'l Conf. Computer Vision*, pages 518–523, Berlin, May 1993.

[39] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics (SIGGRAPH'95)*, pages 39–46, August 1995.

[40] R. Mohr, L. Veillon, and L. Quan. Relative 3D reconstruction using multiple uncalibrated images. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 543–548, New York, June 1993.

[41] A. P. Pentland. Perceptual organization and the representation of natural form. *Artificial Intelligence*, 28(3):293–331, May 1986.

[42] M. Potmesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40:1–29, 1987.

[43] M. Rioux and T. Bird. White laser, synced scan. *IEEE Computer Graphics and Applications*, 13(3):15–17, May 1993.

[44] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Academic Press, New York, 1976.

[45] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1989.

[46] P. T. Sander and S. W. Zucker. Inferring surface trace and differential structure from 3-D images. *IEEE Trans. Pattern Analysis Machine Intelligence*, 12(9):833–854, September 1990.

[47] Y. Sato and K. Ikeuchi. Reflectance analysis for 3d computer graphics model generation. *Graphical Models and Image Processing*, 58(5):437–451, September 1996.

[48] Y. Sato and K. Ikeuchi. Object shape and reflectance modeling from observation. Proc. SIGGRAPH'97 (Los Angeles), August 1997.

[49] H. S. Sawhney. Simplifying motion and structure analysis using planar parallax and image warping. In *Int'l Conf. Pattern Recognition*, volume A, pages 403–408, Jerusalem, Israel, October 1994.

[50] S. M. Seitz and C. M. Dyer. View morphing. Proc. SIGGRAPH'96 (New Orleans), pages 21–30, August 1996.

[51] Chester C. Slama, editor. *Manual of Photogrammetry.* American Society of Photogrammetry, Falls Church, Virginia, Fourth Edition, 1980.

[52] M. Soucy and D. Laurendeau. Multi-resolution surface modeling from multiple range views. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 348–353, Champaign, Ill., June 1992.

[53] P. Srivasan, P. Liang, and S. Hackwood. Computational geometric methods in volumetric intersections for 3D reconstruction. *Pattern Recognition*, 23(8):843–857, 1990.

[54] R. Szeliski. Shape from rotation. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 625–630, Maui, Hawaii, June 1991.

[55] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

[56] R. Szeliski. Image mosaicing for tele-reality applications. In *IEEE Work. Applications of Computer Vision*, pages 44–53, Sarasota, Florida, December 1994.

[57] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, pages 22–30, March 1996.

[58] R. Szeliski and J. Coughlan. Hierarchical spline-based image registration. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 194–201, Seattle, Washington, June 1994.

[59] R. Szeliski and S. B. Kang. Recovering 3D shape and motion from image streams using nonlinear least squares. *J. Visual Communication and Image Representation*, 5(1):10–28, March 1994.

[60] R. Szeliski and S. B. Kang. Direct methods for visual scene reconstruction. In *IEEE Work. Representations of Visual Scenes*, pages 26–33, Cambridge, MA, June 1995.

[61] R. Szeliski and S. B. Kang. Shape ambiguities in structure from motion. In *European Conf. Computer Vision*, volume 1, pages 709–721, Cambridge, England, April 1996. Springer-Verlag.

[62] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (SIGGRAPH'92)*, 26(2):185–194, July 1992.

[63] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Curvature and continuity control in particle-based surface models. In *SPIE Vol. 2031 Geometric Methods in Computer Vision II*, pages 172–181, San Diego, July 1993.

[64] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling surfaces of arbitrary topology with dynamic particles. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 82–87, New York, New York, June 1993.

[65] R. Szeliski and R. Weiss. Robust shape recovery from occluding contours using a linear smoother. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 666–667, New York, June 1993.

[66] D. Terzopoulos and D. Metaxas. Dynamic 3D models with local and global deformations: Deformable superquadrics. *IEEE Trans. Pattern Analysis Machine Intelligence*, 13(7):703–714, July 1991.

[67] D. Terzopoulos, A. Witkin, and M. Kass. Constraints on deformable models: Rcovering 3D shape and nonrigid motion. *Artificial Intelligence*, 36(1):91–123, August 1988.

[68] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization method. *Int'l J. Computer Vision*, 9(2):137–154, November 1992.

[69] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE J. Robotics and Automation*, RA-3(4):323–344, August 1987.

[70] G. Turk and M. Levoy. Zippered polygonal meshes from range images. *Computer Graphics (SIGGRAPH'94)*, pages 311–318, July 1994.

[71] R. Vaillant and O. D.. Faugeras. Using extremal boundaries for 3-D object modeling. *IEEE Trans. Pattern Analysis Machine Intelligence*, 14(2):157–173, February 1992.

[72] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved computational methods for ray tracing. *ACM Trans. Graphics*, 3(1):52069, January 1984.

[73] J. Weng, T. S. Huang, and N. Ahuja. *Motion and Structure from Image Sequences.* Springer-Verlag, Berlin, 1993.

[74] J. Y. Zheng. Acquiring 3-D models from sequences of contours. *IEEE Trans. Pattern Analysis Machine Intelligence*, 16(2):163–178, February 1994.

10^{th} British Machine Vision Conference (BMVC'99)

Stereo Algorithms and Representations for Image-Based Rendering

Richard Szeliski
Vision Technology Group
Microsoft Research
One Microsoft Way
Redmond, WA 98052-6399

Abstract

This paper reviews a number of recently developed stereo matching algorithms and representations. It focuses on techniques that are especially well suited for *image-based rendering* applications such as novel view generation and the mixing of live imagery with synthetic computer graphics. The paper begins by reviewing some recent approaches to the classic problem of recovering a depth map from two or more images. It then describes a number of newer representations (and their associated reconstruction algorithms), including volumetric representations, layered plane-plus-parallax representations, and multiple depth maps. Each of these techniques has its own strengths and weaknesses, which are discussed.

1 Introduction

Stereo matching, which is one of the oldest problems in computer vision, now appears to be a maturing research area. Real-time stereo matching, which a few years ago required special-purpose hardware [16, 20], is now implementable on regular personal computers (see [20] for some references). Depth maps computed with such systems can now be used as basic building blocks for higher level processes such as background subtraction and tracking.

But is stereo really a solved problem? Consider, for example, one of the more recent applications of real-time stereo matching: the ability to composite live video with synthetic computer graphics using the process of *z-keying* [16]. Or, consider the ability to film or photograph a scene or activity from multiple views, and to then look at the same scene from novel viewpoints, i.e., *virtualized reality* [18]. These applications are certainly very exciting, but is the quality of existing algorithms adequate for their use in real production environments?

Judging from the results in recent papers, it appears that we are not there yet. The reconstructions produced by today's algorithms still often leave a "halo" of background pixels clinging to the foreground object. Furthermore, even if a stereo algorithm were to assign a correct depth to each pixel in an image, it would still fail to correctly handle *mixed pixels*, i.e., pixels whose color is a combination of foreground and background colors (which occur at nearly all pixels along a depth discontinuity). What we really need are

algorithms where a foreground layer can be pulled or *matted* from the background, based on the results of a stereo reconstruction algorithm.

Applications such as z-keying and virtualized reality are related to a recent trend in computer graphics, which is called *image-based rendering* [19]. While some of image-based rendering is concerned with re-using synthetically generated images to accelerate rendering speeds, a lot of recent work has focused on acquiring scene or object models from multiple images and re-synthesizing novel views from the original images [23, 14]. When stereo matching is used in such applications, there are several demanding requirements that are not present in more traditional robotics applications of stereo.

First of all, the stereo algorithm must be able to assign correct (or at least reasonable) depths at *all* pixels, especially those near depth discontinuities. In Section 2, I discuss how some recent stereo algorithms are able to avoid the systematic "fattening" of layers associated with traditional area-based methods. A second requirement is the ability to pull mattes, i.e., to separate foreground and background elements while correctly describing the true colors of individual pixels. A third requirement is to generate novel views with as few gaps (missing pixels) as possible, and to also account for partially occluded regions during the matching process. The single depth map representation used in Section 2 is inadequate on all of these counts.

Section 3 describes how a *volumetric* representation of space, combined with real-valued opacities, can be used to overcome most of these problems. Section 4 describes a different, more compact, representation based on arrangements of colored quasi-planar cutouts, which can also overcome these problems. Section 5 describes how to use multiple depth maps (and associated images) to solve the problem of partially occluded areas, and how this representation can also serve as a preliminary step towards a more complete reconstruction of the scene. I conclude this paper with a comparison of the approaches presented and some prospects for further progress in this field.

2 Depth maps

The classical problem of computing a dense depth map from two or more images has been extensively studied. Some good (although slightly dated) surveys of the field can be found in [3, 11, 8]. In this section, we first present a formulation for this problem, and then discuss several recently developed algorithm that attempt to accurately solve for depth near discontinuities.

2.1 Generalized disparity space

Assume we are given as input a collection of K images, $I_1(x, y), I_2(x, y), \ldots, I_K(x, y)$, captured by K cameras with known projection (camera) matrices, $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_K$. To formulate the multiframe stereo problem, we use a *generalized disparity space*, which can be any projective sampling (collineation) of 3-D space [9, 34]. This space is a generalization of the notion of *disparity space* [41, 15, 28], i.e., the enumeration of all possible disparities at every pixel. The goal of stereo matching is then to find the elements in disparity space which lie on the surfaces of the objects in the scene. The benefits of such an approach include the equal and efficient treatment of a large number of images [9] and the possibility of modeling occlusions [15].

To formulate the generalized disparity space, we first choose a *virtual camera* position and orientation. This virtual camera may be coincident with one of the input images, or it can be chosen based on the application demands and the desired accuracy of the results. For instance, if we wish to regularly sample a volume of 3-D space, we can make the camera orthographic, with the camera's (x, y, d) axes being orthogonal and evenly sampled (as in [29]).

Having chosen a virtual camera position, we then choose the orientation and spacing of the *disparity planes*, i.e., the constant d planes. The relationship between d and 3-D space can be projective. For example, we can choose d to be inversely proportional to depth, which is the usual meaning of disparity [24]. The information about the virtual camera's position and disparity plane orientation and spacing can be captured in a single 4×4 matrix $\hat{\mathbf{M}}_0$, which represents a collineation of 3-D space, $w(x\ y\ d\ 1)^T = \hat{\mathbf{M}}_0(X\ Y\ Z\ 1)^T$.

2.2 Area-based approaches

Having presented the representation used for describing the output of the matching algorithm, we can now state its goal: For each (x, y) location in disparity space, find the disparity d that aligns corresponding locations in the input images (ignoring, for now, the possibility that pixels may be occluded). In traditional area-based correlation, the quality of a match is measured by comparing windows centered at corresponding locations, for example, using the sum of squared intensity differences (SSD) [17].

A more general way of characterizing area-based algorithms is the following [28]:

1. For each disparity under consideration, compute a per-pixel matching cost, e.g., squared intensity difference or variance of colors across the k input images.
2. Aggregate support spatially (e.g., by summing over a window, or by diffusion).
3. At each pixel (x, y), find the best matching disparity d based on the aggregated support.
4. Compute a sub-pixel disparity estimate (optional).

Let us look at the components in this framework in more detail.

At the base of any matching algorithm is a matching cost that measures the similarity of corresponding locations. Matching costs can be defined locally (at pixel level), or over a certain area of support. Examples of local costs are absolute intensity differences, squared intensity differences, binary pixel matches, edges, filtered images, and measures based on gradient direction or gradient vectors. Matching costs that are defined over a certain area of support include correlation and non-parametric measures. These can be viewed as a combination of the matching cost and aggregation stages. More than two images are used in multiframe stereo to increase stability of the algorithm [24].

Aggregating support is necessary for stable matching. A support region can either be two-dimensional at a fixed disparity (favoring fronto-parallel surfaces), or three-dimensional in x-y-d space (supporting slanted surfaces). Two-dimensional evidence aggregation has been done using square windows (traditional), Gaussian convolution, multiple windows anchored at different points [15], and windows with adaptive sizes [17]. Three-dimensional support functions that have been proposed include limited disparity difference, limited disparity gradient [25], and Prazdny's coherence principle [26].

Sub-pixel disparity estimates can be computed by fitting a curve to the matching costs at the discrete disparity levels [22, 17]. This provides an easy way to increase the resolution of a stereo algorithm with little additional computation. However, to work well, the intensities being matched must vary smoothly. A related refinement is to compensate for discrete sampling of disparity space by linearly interpolating errors computed at adjacent disparity levels, and analytically finding the minimum matching error in this interval [4].

2.3 Global optimization approaches

Optimization (regularization) approaches start with the same computation of matching costs as area-based techniques, but then add a controlled smoothness penalty (prior) on the disparity field $d(x, y)$. A variety of optimization algorithms can then be used to find a good solution to this problem [2, 28, 7].

An elegant mathematical approach to formulating these energy function and finding their minimum is to use a Bayesian (probabilistic) estimation framework. The Bayesian model of stereo image formation consists of two parts. The first part, a *prior model* for the disparity surface, uses a traditional Markov Random Field (MRF) to encode preferences for smooth surfaces [13]. This model is specified as a Gibbs distribution p_P, the exponential of a potential function E_P:

$$p_P(\mathbf{d}) = \frac{1}{Z_P} \exp\left(-E_P(\mathbf{d})\right), \tag{1}$$

where $\mathbf{d}$ is the vector of all disparities $d(x, y)$ and Z_P is a normalizing factor. The potential function itself is the sum of clique potentials that only involve neighboring sites in the field. The simplest such field is a first order field, where

$$E_P(\mathbf{d}) = \sum_{i,j} \rho_P(d(x+1, y) - d(x, y)) + \rho_P(d(x, y+1) - d(x, y)) \tag{2}$$

(see [38, 31] for generalizations to higher order fields).

When $\rho(x)$ is a quadratic, $\rho(x) = x^2$, the field is a Gauss-MRF, and corresponds in a probabilistic sense to a first order regularized (*membrane*) surface model [38, 31]. When $\rho(x)$ is a unit impulse, $\rho(x) = 1 - \delta(x)$, it corresponds to a MRF that favors fronto-parallel surfaces [13]. In between these two extremes are functions derived from *robust statistics*, which behave much like surface models with discontinuities [5].

The second part of a Bayesian model is the *data* or *measurement model* which accounts for differences in intensities between corresponding image locations. This model assumes independent, identically distributed measurement errors,

$$p_M(I_1, \ldots, I_K|\mathbf{d}) = \prod_{i,j} p_M(x, y, d)), \tag{3}$$

where $\log p_M(x, y, d) = \rho_M(x, y, d)$ is the initial, unaggregated, matching cost. Traditional stereo matching methods use either a squared intensity error metric (Gaussian noise), or an exact binary matching criterion (e.g., for random-dot stereograms or binary features such as edges or the sign of the Laplacian). A more general model is a contaminated Gaussian model, which models both Gaussian noise and allows possible outliers due to occlusions or non-modeled photometric effects such as specularities.

The posterior distribution, $p(\mathbf{d}|I_1, \ldots, I_K)$ can be derived from the prior and measurement models using Bayes' rule,

$$p(\mathbf{d}|I_1, \ldots, I_K) \propto p_P(\mathbf{d}) p_M(I_1, \ldots, I_K|\mathbf{d}). \tag{4}$$

As is often the case, it is more convenient to study the negative log probability distribution

$$\begin{aligned} E(\mathbf{d}) &= -\log p(\mathbf{d}|I_1, \ldots, I_K) \\ &= \sum_{i,j} \rho_P(d_{i+1,j} - d_{i,j}) + \rho_P(d_{i,j+1} - d_{i,j}) + \sum_{i,j} \rho_M(x_i, y_j, d_{i,j}). \end{aligned} \tag{5}$$

While $p(\mathbf{d}|I_1, \ldots, I_K)$ specifies a complete distribution, usually only a single optimal estimate of $d(x, y)$ is desired ([31] explains why modeling of uncertainties may be useful). The most commonly studied estimate is the peak of the distribution, or *Maximum A Posteriori* (MAP) estimate, which is equivalent to minimizing the energy given in (5).

A variety of techniques have been developed for minimizing equations like (5). Two of the most popular are the Gibbs Sampler [13] and mean field theory [12]. The Gibbs Sampler randomly chooses values for each $d_{i,j}$ site according to the local distribution determined by the current guesses for a site's neighbors [13, 2]. This process will in theory converge to a statistically optimal sample, given enough time. Mean field theory updates an estimate of the *mean* value of $d_{i,j}$ at each site using a deterministic update rule derived from the original probability distribution [13].

The Gibbs Sampler and its variants can produce good solutions, but at the cost of long computation times. Mean field techniques, on the other hand, are not very good at modeling ambiguous estimates, such as multiple potential matches at each pixel. Instead of using either of these two traditional approaches, we developed a novel estimation algorithm based on modeling the probability distribution of $d(x, y)$ at each site [28] . To do this, we associate a scalar value between 0 and 1 with each possible discrete value of d at each pixel (x, y), and require that the probabilities sum up to 1. This representation is therefore the same as that used by aggregation-based algorithms, i.e., it explicitly models all possible disparities at each pixel, rather than modeling a single estimated disparity as in traditional Gibbs Sampler or mean-field approaches [2].

The algorithm is initialized by calculating the probability distribution for each pixel (x, y) based on the intensity errors between matching pixels, i.e., using the measurement model (3). To derive the update formula, we approximate the true Markov Random Field distribution with a *factored* approximation, i.e., we assume that the neighboring disparity columns have independent distributions. Minimizing the Kullback-Leibler divergence between the true posterior Gibbs distribution and its factored (mean-field) approximation leads to a set of update formulas on the probability distributions that use non-linear *diffusion* (see [28] for details).

The results of running this algorithm on difficult stereo pairs are quite promising. The algorithm is particularly good at correctly matching pixels near depth discontinuities, since the robust smoothness constraint can be violated at the appropriate places, and also at stereograms that have a lot of potential matches, such as random-dot stereograms.

Another recent development in optimization-based stereo matching is the use of graph algorithms [27, 7]. Here, techniques from discrete optimization are used to find good minima (in some cases, even global minima) of the global energy function (5). These algorithms have both good discontinuity localization, since they are based on robust smoothness

Input image | Graph cuts | Bayesian diffusion | SAD (13×13)

Figure 1: Results for several stereo algorithms on the University of Tsukuba imagery.

models, and also excel at filling in good disparities inside uniform color/intensity regions (since they take large steps in state space).

Figure 1 shows the results of applying three different matching algorithms on a multi-camera stereo data set provided by the University of Tsukuba. You can readily see that the two energy minimization-based approaches (Graph cuts and Bayesian) have much crisper depth discontinuities, compared with the sum of absolute differences (SAD) technique. The graph-cut approach also does an excellent job of filling in uniform intensity areas. These results are part of a larger evaluation of two-frame stereo matching algorithms that we are currently undertaking [37].

The original color image can be *texture-mapped* onto the surface defined by the depth map to produce novel views [22]. Unfortunately, the depth map behaves as a "rubber sheet", i.e., background regions that are not visible in the reference image are not correctly synthesized. As more and more images are used in stereo matching, this effect become even more pronounced. For this reason, we now turn our attention to algorithm that explicitly represent and reason about partially occluded regions.

3 Volumetric representations

The depth map representation presented in the previous section is unable to represent and hence render partially occluded background regions. This is due to our insistence (enforced during the winner-take-all stage) that only a single depth value be assigned to each pixel in the reference image.

What if we were to relax this assumption? What if in addition to being able to have several depth along each ray in the reference image, we also represented the colors of these pixels and their (potentially partial) opacities? In principle, we should be able to represent and reason about partially occluded pixels, and to correctly estimate the color values of mixed pixels. These are the intuitions that led to the development of the volumetric stereo reconstruction algorithm presented in [34]. (Simultaneously with our work, Seitz and Dyer [29] developed a volumetric stereo algorithm that uses binary (filled/empty) opacities and a front-to-back plane sweep (*voxel coloring*) algorithm. DeBonet and Viola also have a volumetric reconstruction technique that estimates partial opacities [10].)

The algorithm starts by performing the same matching cost computation, aggregation, and winning depth value selection as described in the previous section. However, instead of insisting that every pixel in the reference image pick a winning depth, we only select depth values that have a good match (good aggregated evidence), using a threshold to mark other pixels as currently "unassigned". These pixels will typically not have correspondences

either because they are partially occluded, or because they are mixed pixels with different backgrounds in different images. A new (x, y, d) volume can now be created, where each cell now contains a color value, initially set to the mean color computed in the first stage, and the opacity is set to 1 for cells which are winners, and 0 otherwise.

Once we have an initial (x, y, d) volume containing estimated RGBA (color and 0/1 opacity) values, we can re-project this volume into each of the input cameras using the known transformation

$$\mathbf{x}_k = \mathbf{P}_k \hat{\mathbf{M}}_0^{-1} \hat{\mathbf{x}}_0 \tag{6}$$

where $\hat{\mathbf{x}}_0$ is a (homogeneous) coordinate in (x, y, d) space, $\hat{\mathbf{M}}_0$ is the complete camera matrix corresponding to the virtual camera, $\mathbf{P}_k$ is the kth camera matrix, and $\mathbf{x}_k$ are the image coordinates in the kth image. In our approach, we interpret the (x, y, d) volume as a set of (potentially) transparent acetates stacked at different d levels. Each acetate is first warped into a given input camera's frame using the known homography $\mathbf{H}_k$. Once the layers have been resampled, they are then composited using the standard *over* operator [6].

After the re-projection step, we refine the disparity estimates by preventing visible surface pixels from voting for potential disparities in the regions they occlude. More precisely, we build an (x, y, d, k) *visibility map*, which indicates whether a given camera k can see a voxel at location (x, y, d) [34].

Once we have computed the visibility volumes for each input camera, we can update the list of color samples we originally used to get our initial disparity estimates to obtain a distribution of colors in (x, y, d, k) where each color has an associated visibility value. Voxels that are occluded by surfaces lying in front in a given view k will now have fewer (or potentially no) votes in their local color distributions. We can therefore recompute the local mean and variance estimates using weighted statistics, where the visibilities $V(x, y, d, k)$ provide the weights.

With these new statistics, we are now in position to refine the disparity map. In particular, voxels in disparity space that previously had an inconsistent set of color votes (large variance) may now have a consistent set of votes, because voxels in (partially occluded) regions will now only receive votes from input pixels that are not already assigned to nearer surfaces.

While the above process of computing visibilities and refining disparity estimates will in general lead to a higher quality disparity map (and better quality mean colors, i.e., texture maps), it will not recover the true colors and transparencies in *mixed pixels*, e.g., near depth discontinuities, which is one of the main goals of this research.

In the second phase of our algorithm, we adjust the opacity and color values $\hat{\mathbf{c}}(x, y, d)$ to match the input images (after re-projection), while favoring continuity in the color and opacity values. This can be formulated as a non-linear minimization problem, where the cost function has three parts: a weighted error norm on the difference between the re-projected images $\tilde{\mathbf{c}}_k(u, v)$ and the original input images $\mathbf{c}_k(u, v)$; a (weak) smoothness constraint on the colors and opacities; and a prior distribution on the opacities [34]. To minimize the total cost function, we use a preconditioned gradient descent algorithm. A complete description of this procedure is given in [34].

Figure 2 shows the results of this algorithm when run on a cropped portion of the *SRI Trees* multibaseline stereo dataset. A small region (64×64 pixels) was selected in order to better visualize pixel-level errors. While the overall reconstruction is somewhat noisy, the final reconstruction with a synthetic blue layer inserted shows that the algorithm has

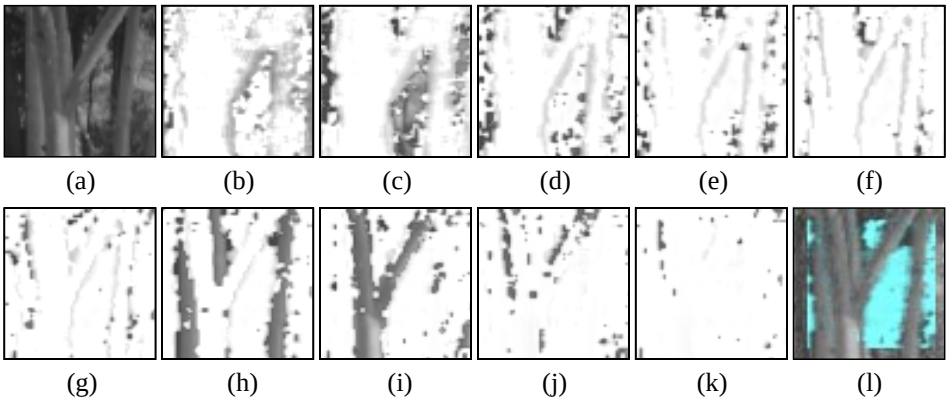


Figure 2: Volumetric reconstruction example: (a) cropped subimage from *SRI Trees* data set, (b–k) disparity layers $d = 0 \ldots 9$, (l) re-synthesized input image with inserted $d = 4$ blue layer.

done a reasonable job of assigning pixel depths and computing partial transparencies near the tree boundaries.

From this example, we see that the volumetric approach is a much more powerful representation for dealing with partially occluded regions and mixed pixels. Unfortunately, this power comes at the expense of two problems: the depth are quantized, which can lead to aliasing effects, and the representation has a very large number of degrees of freedom, which makes it difficult to find the optimal solution. The first problem could be fixed, in principle, by using fractional disparities (although these would have to be relative to one preferred camera). The second problem we address in the next section, where a much more parsimonious description is used.

4 Layered representations

To overcome the problem with the volumetric representation, we draw some inspiration from recent work in layered motion estimation [40]. Here, the goal is to decompose the images into sub-images, commonly referred to as *layers*, such that the pixels within each layer move in a manner consistent with a parametric transformation. The motion of each layer is determined by the values of the parameters. An important transformation is the 8–parameter homography (collineation), because it describes the motion of a rigid planar patch as either it or the camera moves.

While existing techniques have been successful in detecting multiple independent motions, layer extraction for scene modeling has not been fully developed. One fact that has not been exploited is that, when simultaneously imaged by several cameras, each of the layers implicitly lies on a fixed plane in the 3D world. Another omission is the proper treatment of transparency. With a few exceptions, the decomposition of an image into layers that are partially transparent has not been attempted. In contrast, scene modeling using multiple partially transparent layers is common in the graphics community [6].

In our own work [1], we have developed a framework for reconstructing a scene as a collection of approximately planar layers. Each of the layers has an explicit 3D

plane equation and is recovered as a *sprite*, i.e. a colored image with per-pixel opacity (transparency) [6]. To model a wider range of scenes, a per-pixel depth offset relative to the plane is also added.

Our layered approach to stereo shares many of the advantages of the volumetric approach. In addition, it offers a number of other advantages:

- The combination of the global model (the plane) with the local correction to it (the per-pixel depth offset) results in very robust performance. In this respect, the framework is similar to the *plane + parallax* work of [21].
- The output (a collection of approximately planar regions) is more suitable than a discrete collection of voxels for many applications, including, rendering [30] and video parsing.

Our representation consists of a collection of L approximately planar layers, each of which is an alpha-matted color image (layer sprite) $L_l(x, y)$ with *pre-multiplied opacities* [6]. We also associate a homogeneous vector $\mathbf{n}_l$ with each layer (which defines the plane equation of the layer via $\mathbf{n}_l^T\mathbf{x} = 0$) and a per-pixel residual depth offset $Z_l(x, y)$.

The goal of our layer decomposition algorithm is to estimate these quantities. To do so, we wish to use techniques for parametric motion estimation. Unfortunately, most such techniques assume boolean-valued opacities α_l (i.e., unique layer assignments). We therefore split our framework into two parts. In the first part, we assume boolean opacities to get a first approximation to the structure of the scene. If the opacities are boolean, each point in each image I_k is only the image of a point on one of the layers L_l. We therefore introduce boolean masks B_{kl} which denote the pixels in image I_k that are images of points on layer L_l. So, in addition to L_l, $\mathbf{n}_l$, and Z_l, we also need to estimate the masks B_{kl}. Once we have estimates of the masks, we immediately compute masked input images $M_{kl} = B_{kl} \cdot I_k$. In the second part of our framework, we use the initial estimates of the layers made by the first part as input into a re-synthesis algorithm which refines the layer sprites L_l, *including* the opacities α_l. This second step requires a generative or forward model of the image formation process.

Before we can compute the layer sprites L_l, we need to choose 2D coordinate systems for the sprite images. Such coordinate systems can be specified by a collection of arbitrary (rank 3) camera matrices $\mathbf{Q}_l$. In [1] we show that the image coordinates $\mathbf{x}_k$ of the pixel in image M_{kl} that is projected onto the pixel $\mathbf{x}_l$ on the plane $\mathbf{n}_l^T\mathbf{x} = 0$ is given by

$$\mathbf{x}_k \;=\; \mathbf{P}_k \left((\mathbf{n}_l^T \mathbf{q}_l)\mathbf{I} - \mathbf{q}_l \mathbf{n}_l^T \right) \mathbf{Q}_l^* \mathbf{x}_l \;\equiv\; \mathbf{H}_k^l \mathbf{x}_l, \tag{7}$$

where $\mathbf{Q}_l^*$ is the pseudo-inverse of $\mathbf{Q}_l$, and $\mathbf{q}_l$ is a vector in the null space of $\mathbf{Q}_l$. The homography $\mathbf{H}_k^l$ can be used to warp the image M_{kl} forward onto the coordinate frame of the plane $\mathbf{n}_l^T\mathbf{x} = 0$, the result of which is denoted $\mathbf{H}_k^l \circ M_{kl}$. Then, we can estimate the layer sprite (with boolean opacities) by *blending* the warped images [35].

To compute the homographies $\mathbf{H}_k^l$ that align all masked image pieces M_{kl} into a consistent coordinate frame, we use a previously developed parametric motion (*mosaicing*) technique [35]. Once we have an initial estimate for the $\mathbf{H}_k^l$, we use a structure-from-motion algorithm to compute the plane equations (and, in the case where the original camera matrices $\mathbf{P}_k$ are unknown, to estimate them as well) [36]. A better approach might be to directly optimize over the plane normal $\mathbf{n}_l$ used in (7).

Since in general, the scene will not be piecewise planar, we allow the point $\mathbf{x}_l$ on the plane $\mathbf{n}_l^T\mathbf{x} = 0$ to be displaced slightly. We assume it is displaced in the direction of the

ray through $\mathbf{x}_l$ defined by the camera matrix $\mathbf{Q}_l$. The distance it is displaced is denoted by $Z_l(\mathbf{x}_l)$, as measured in the direction *normal* to the plane. In this case, the homographic warps used in the previous section are not applicable, but using a similar argument, it is possible to show that

$$\mathbf{x}_k \;=\; \mathbf{H}^l_k \mathbf{x}_l + w(\mathbf{x}_l) Z_l(\mathbf{x}_l) \mathbf{t}_{kl}, \tag{8}$$

where $\mathbf{t}_{kl} = \mathbf{P}_k \mathbf{q}_l$ is the epipole and it is assumed that the vector $\mathbf{n}_l = (n_x, n_y, n_z, n_d)^T$ has been normalized such that $n_x^2 + n_y^2 + n_z^2 = 1$. The term $w(\mathbf{x}_l)$ is a projective scaling factor which equals the reciprocal of $\mathbf{Q}^3_l \mathbf{x}$, where $\mathbf{Q}^3_l$ is the third row of $\mathbf{Q}_l$ and $\mathbf{x}$ is the world coordinate of the point. Equation (8) can be used to map plane coordinates $\mathbf{x}_l$ backwards to image coordinates $\mathbf{x}_k$, or to map the image M_{kl} forwards onto the plane. We denote the result of this warp by $(\mathbf{H}^l_k, \mathbf{t}_{kl}, Z_l) \circ M_{kl}$, or more concisely $\mathbf{W}^l_k \circ M_{kl}$.

Almost any stereo algorithm can be used to compute $Z_l(\mathbf{x}_l)$, although it is preferable to use one favoring small disparities. Currently, we use a simple plane-sweep algorithm, a simplified version of the algorithm described in [28]. Once the residual depth offsets have been estimated, the layer sprite images can be re-estimated by blending the warped images $\mathbf{W}^l_k \circ M_{kl}$.

To re-compute the pixel assignments, we compare the warped images $\mathbf{W}^l_k \circ M_{kl}$ with the layer sprites L_l. If the pixel assignment was correct (and neglecting resampling issues) these images should be identical where they overlap. Details of the heuristics used in re-computing the layer assignments are given in [1].

In [1], we also describe how the estimates of the layer sprites can be refined, now assuming that their opacities α_l are real-valued. We begin by formulating a generative model of the image formation process. Afterwards, we propose a measure of how well the layers re-synthesize the input images, and show how the re-synthesis error can be minimized to refine the estimates of the layer sprites. This approach is similar to the one we developed for the volumetric model with transparent voxels [34]. We are currently implementing this portion of our algorithm. Once it is complete, we are hoping to have layer descriptions that will correctly account for mixed pixels, and may even be able to reconstruct scenes with translucent surfaces such as dirty windows or scenes with additive phenomena such as reflections.

Figure 3 shows some results of applying our algorithm to five images from a 40-image stereo data set taken at a graphics symposium. Figure 3(a) shows the middle input image, Figure 3(b) shows the initial pixel assignment to layers, Figure 3(c) shows the recovered planar depth map, and Figure 3(f) shows the residual depth map for one of the layers. Figures 3(d) and (e) show the recovered sprites. Figure 3(g) shows the middle image re-synthesized from these sprites. Finally, Figures 3(h–i) show the same sprite collection seen from a novel viewpoint (well outside the range of the original views), first with and then without residual depth correction. The gaps in Figure 3 correspond to parts of the scene that were not visible in any of the five input images.

To summarize, the layered approach to 3D reconstruction represents the scene as a collection of approximately planar layers. Each layer consists of a plane equation, a layer sprite image, and a residual depth map. The framework exploits the fact that each layer implicitly lies on a fixed plane in the 3D world. This is both the algorithm's strength (using a compact description) and its weakness (it is limited to scenes where objects are "cutouts with relief"). The layered approach also requires solving a combinatorial optimization problem, since the number of layers needs to be determined, as well as figuring out the assignment of pixels to layers [39].

Figure 3: Layered stereo results: (a) third of five images; (b) initial segmentation into six layers; (c) recovered depth map (darker denotes closer); (d) and (e) the five layer sprites; (f) residual depth image for fifth layer. (g) re-synthesized third image (note extended field of view). (h) novel view without residual depth; (i) novel view with residual depth (note the "rounding" of the people).

5 Multiple depth maps

In our most recent work, we have been investigating an alternative to volumetric and layered representations that can also represent and reason about semi-occluded regions. Rather than estimating a single depth map, we associate a depth map with *each* input image (or some subset of them) [32]. Furthermore, we try to ensure consistency between these different estimates using a *depth compatibility* constraint, and reason about occlusion relationships by computing pixel *visibilities*. Our representation can be used as is for image-based rendering (view interpolation) applications, or it can be used as a low-level representation from which segmentation and layer extraction (or 3D model construction) can take place.

To formulate the multi-view stereo problem, we take the matching costs for all reference images and sum them together. This *brightness compatibility* term, which measures the degree of agreement in brightness or color between corresponding pixels, can be written

(a) (b) (c)

Figure 4: Results of multi-view stereo algorithm: (a) depth estimate for first frame; (b) warped (resampled) images without visibilities; (c) with visibility computation.

as

$$\mathcal{C}(\{\mathbf{x}_s\}) = \sum_{s \in S} \sum_{t \in \mathcal{N}(s)} w_{st} \sum_{\mathbf{x}_s} v_{st}(\mathbf{x}_s) \rho\left(I_s(\mathbf{x}_s) - I_t(\mathbf{x}_t)\right). \tag{9}$$

The images I_s form the set S of *keyframes* (or *key-views*) for which we will estimate a depth estimate $d_s(\mathbf{x}_s)$. The decision as to which images are keyframes is problem-dependent, much like the selection of I and P frames in video compression. For 3D view interpolation, one possible choice of keyframes would be a collection of *characteristic views*.

Images $I_t, t \in \mathcal{N}(s)$ are *neighboring frames* (or views), for which we require that corresponding pixel brightnesses (or colors) agree. The pixel coordinate $\mathbf{x}_t$ corresponding to a given keyframe pixel $\mathbf{x}_s$ with depth d_s can be computed according to the rigid motion model (6). The constants w_{st} are the *inter-frame weights* which dictate how much neighboring frame t will contribute to the estimate of d_s. Corresponding pixel brightness or color differences are passed through a robust penalty function ρ. The visibility factor $v_{st}(\mathbf{x}_s)$, which encodes whether pixel $\mathbf{x}_s$ is *visible* in image I_t, can be computed by comparing corresponding depth values, i.e., checking whether $d_t(\mathbf{x}_t) \leq d_s(\mathbf{x}_s)$.

The cost function used in [32] consists of two additional terms. The controlled *depth compatibility* constraint, enforces *mutual consistency* between depth estimates at different neighboring keyframes. The controlled *depth smoothness* constraint, encourages the depth maps to be piecewise smooth. The shape of this robust penalty function is affected by the brightness/color difference between neighboring pixels (see [32] for details).

Our algorithm operates in two phases. During an initialization phase, we estimate the depths independently for each keyframe. Since we do not yet have any good motion estimates for other frames, the depth compatibility term $\mathcal{C}_T$ is ignored, and no visibilities are computed (i.e., $v_{st} = 1$). In the second phase, we enforce depth compatibility and compute visibilities based on the current collection of depth estimates $\{d_s\}$. Details on the optimization algorithm can be found in [32].

Figure 4 shows some representative results from running our algorithm. The depth map estimated by the algorithm is shown in Figure 4a. Figure 4b shows the results of warping the last image to the first image, based on the depth computed in the first image. Displaying these warped images as the algorithm progresses is a very useful way to debug the algorithm and to assess the quality of the motion estimates. Figure 4c shows the same warped image with invisible pixels flagged as black. Notice how the algorithm correctly labels most of the occluded pixels to the right of the two people's heads.

The experimental results we have obtained so far are encouraging, but still leave room for improvement. In particular, the smoothness of the final estimates and the *sharpness* of

the motion discontinuities is not as high as that obtainable with layered models [1]. This is particularly true in occluded regions: layered models will apply the layer's motion to the occluded regions, while we use a weak smoothness constraint.

The multi-view stereo matching framework described in this section produces estimates for a subset of the input images, thereby representing depth in partially occluded regions and explicitly modeling the variation in appearance between different views. Compared with the volumetric and layered representations, the multiple depth map representation is potentially not as compact (although it can be more compact than the search space of the volumetric technique), nor does it correctly model mixed pixels (because the concept of opacity is not built in). It also does not ensure that corresponding surface elements in different views have the same 3D location, although it attempts to ensure this with the weak compatibility constraint. The representation does, however, capture the variation in appearance between different view, for example, when there are strong illumination effects. The representation is useful for performing image-based rendering tasks such as novel view generation, and can also be used to *bootstrap* a more parsimonious representation such as 3D layers.

6 Discussion and Conclusions

In this paper, I have presented a number of representation and algorithms for reconstructing 3D scenes and objects using stereo matching techniques. My emphasis has been on techniques that are well suited to image-based rendering, i.e., approaches that can re-synthesize observed and novel views with a high degree of realism.

The desire to predict the performance of these approaches in image-based rendering applications has led me to propose a new quality metric for stereo matching. Instead of measuring deviations from ground truth depth maps (which are generally hard to come by), I suggest measuring how well the representation predicts *novel* views, i.e., images in a calibrated multi-image stereo data set that have intentionally been held back from the matcher [33] (this is similar to the statistical method of cross-validation). Such data sets are relatively easy to acquire, e.g., by taking a video of a rigid scene and applying a tracking and structure from motion algorithm to recover the camera positions.

Ramin Zabih and I are also currently performing a comparative evaluation of two-frame stereo matching algorithms [37]. While this study excludes some of the novel representations presented in this paper, we hope that it will shed light on underlying principles that make stereo matching work better.

To summarize, I have presented four different representations for stereo matching. A single depth map, the traditional representation used for matching, is a very compact and useful representation that can yield good results when the amount of occlusion is not large, i.e., when the surface is smoothly varying (e.g., a human face) and the range of viewpoints is limited. The volumetric representation (with partial opacities) can be used to represent and reason about partially occluded regions and mixed pixels. Unfortunately, it also has many degrees of freedom, which makes it tricky to find the best reconstruction. Layered representations have the same advantages as volumetric ones, and are potentially more compact, and hence easier to recover. However, determining the best number of planes and the correct pixel assignment is a tricky problem, which we are currently trying to solve. These representations are also inherently limited to scenes that are well approximated by a

collection of embossed cutouts. Finally, multiple depth maps can be used to obtain some of the same advantages with respect to partially occluded regions, and also to model the variation in appearance between viewpoints. Unfortunately, they are not guaranteed to have consistent representations of 3D shape, and also do not correctly predict the appearance of mixed pixels.

Thus, we see that all of the representations suggested so far have their limitations. Still, a tremendous amount of progress has been made in recent years in obtaining better and better stereo reconstructions, especially for image-based rendering applications where recovering the true shape of a scene is not paramount. I expect that by re-visiting issues in representation, e.g., by more closely studying the role of discontinuities in shape and depth representations, we will be able to make even further progress, and thereby expand the utility and applicability of stereo-based reconstruction techniques.

References

[1] S. Baker, R. Szeliski, and P. Anandan. A layered approach to stereo reconstruction. In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recog. (CVPR'98)*, pp. 434–441, June 1998.

[2] S. T. Barnard. Stochastic stereo matching over scale. *Intl. J. Comp. Vision*, 3(1):17–32, 1989.

[3] S. T. Barnard and M. A. Fischler. Computational stereo. *Computing Surveys*, 14(4):553–572, December 1982.

[4] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, 20(4):401–406, April 1998.

[5] M. J. Black and A. Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *Intl. J. Comp. Vision*, 19(1):57–91, 1996.

[6] J. F. Blinn. Jim Blinn's corner: Compositing, part 1: Theory. *IEEE Comp. Graphics and Applications*, 14(5):83–87, September 1994.

[7] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *Seventh Intl. Conf. on Comp. Vision (ICCV'99)*, September 1999.

[8] L. G. Brown. A survey of image registration techniques. *Computing Surveys*, 24(4):325–376, December 1992.

[9] R. T. Collins. A space-sweep approach to true multi-image matching. In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recog. (CVPR'96)*, pp. 358–363, June 1996.

[10] J. S. De Bonet and P. Viola. Poxels: Probabilistic voxelized volume reconstruction. In *Seventh Intl. Conf. on Comp. Vision (ICCV'99)*, September 1999.

[11] U. R. Dhond and J. K. Aggarwal. Structure from stereo—a review. *IEEE Trans. on Systems, Man, and Cybernetics*, 19(6):1489–1510, November/December 1989.

[12] D. Geiger and F. Girosi. Parallel and deterministic algorithms for MRF's: Surface reconstruction. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, 13(5):401–412, May 1991.

[13] S. Geman and D. Geman. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, PAMI-6(6):721–741, November 1984.

[14] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Comp. Graphics Proceedings, Annual Conf. Series*, pp. 43–54, Proc. SIGGRAPH'96, August 1996.

[15] S. S. Intille and A. F. Bobick. Disparity-space images and large occlusion stereo. In *Proc. Third European Conf. on Comp. Vision (ECCV'94)*, vol. 1, May 1994. Springer-Verlag.

[16] T. Kanade et al. A stereo machine for video-rate dense depth mapping and its new applications. In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recog. (CVPR'96)*, pp. 196–202, 1996.

[17] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory

and experiment. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, 16(9):920–932, September 1994.

[18] T. Kanade, P. W. Rander, and P. J. Narayanan. Virtualized reality: constructing virtual worlds from real scenes. *IEEE MultiMedia Magazine*, 1(1):34–47, Jan-March 1997.

[19] S. B Kang. A survey of image-based rendering techniques. In *Videometrics VI*, vol. 3641, pp. 2–16, 1999. SPIE.

[20] R. Kimura et al. A convolver-based real-time stereo machine (SAZAN). In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recog. (CVPR'99)*, vol. 1, pp. 457–463, June 1999.

[21] R. Kumar, P. Anandan, and K. Hanna. Direct recovery of shape from multiple views: A parallax based approach. In *Twelfth Intl. Conf. on Patt. Recognition (ICPR'94)*, volume A, pp. 685–688, Jerusalem, Israel, October 1994. IEEE Comp. Soc. Press.

[22] L. H. Matthies, R. Szeliski, and T. Kanade. Kalman filter-based algorithms for estimating depth from image sequences. *Intl. J. Comp. Vision*, 3:209–236, 1989.

[23] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Comp. Graphics (SIGGRAPH'95)*, pp. 39–46, August 1995.

[24] M. Okutomi and T. Kanade. A multiple baseline stereo. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, 15(4):353–363, April 1993.

[25] S. B. Pollard, J. E. W. Mayhew, and J. P. Frisby. PMF: A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14:449–470, 1985.

[26] K. Prazdny. Detection of binocular disparities. *Biological Cybernetics*, 52:93–99, 1985.

[27] S. Roy, , and I. J. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *Sixth Intl. Conf. on Comp. Vision (ICCV'98)*, pp. 492–499, January 1998.

[28] D. Scharstein and R. Szeliski. Stereo matching with nonlinear diffusion. *Intl. J. Comp. Vision*, 28(2):155–174, July 1998.

[29] S. M. Seitz and C. M. Dyer. Photorealistic scene reconstrcution by voxel coloring. In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recog. (CVPR'97)*, pp. 1067–1073, June 1997.

[30] J. Shade, S. Gortler, L.-W. He, and R. Szeliski. Layered depth images. In *Comp. Graphics (SIGGRAPH'98) Proceedings*, pp. 231–242, July 1998.

[31] R. Szeliski. *Bayesian Modeling of Uncertainty in Low-Level Vision*. Kluwer Academic Publishers, Boston, Massachusetts, 1989.

[32] R. Szeliski. A multi-view approach to motion and stereo. In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recog. (CVPR'99)*, volume 1, pp. 157–163, Fort Collins, June 1999.

[33] R. Szeliski. Prediction error as a quality metric for motion and stereo. In *Seventh Intl. Conf. on Comp. Vision (ICCV'99)*, September 1999.

[34] R. Szeliski and P. Golland. Stereo matching with transparency and matting. *Intl. J. Comp. Vision*, 32(1):45–61, August 1999. Special Issue for Marr Prize papers.

[35] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and texture-mapped models. *Comp. Graphics (SIGGRAPH'97)*, pp. 251–258, August 1997.

[36] R. Szeliski and P. Torr. Geometrically constrained structure from motion: Points on planes. In *European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE)*, pp. 171–186, Freiburg, Germany, June 1998.

[37] R. Szeliski and R. Zabih. An experimental comparison of stereo algorithms. In *Vision Algorithms 99 Workshop*, submitted 1999.

[38] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Trans. on Patt. Analysis and Machine Intelligence*, PAMI-8(4):413–424, July 1986.

[39] P. H. S. Torr, R. Szeliski, and P. Anandan. An integrated Bayesian approach to layer extraction from image sequences. In *Seventh Intl. Conf. on Comp. Vision (ICCV'98)*, September 1999.

[40] J.Y.A. Wang and E.H. Adelson. Layered representation for motion analysis. In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recognition*, pp. 361–366, 1993.

[41] Y. Yang, A. Yuille, and J. Lu. Local, global, and multilevel stereo matching. In *IEEE Comp. Soc. Conf. on Comp. Vision and Patt. Recog. (CVPR'93)*, pp. 274–279, June 1993.

Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach

Paul E. Debevec Camillo J. Taylor Jitendra Malik

University of California at Berkeley[1]

ABSTRACT

We present a new approach for modeling and rendering existing architectural scenes from a sparse set of still photographs. Our modeling approach, which combines both geometry-based and image-based techniques, has two components. The first component is a *photogrammetric modeling* method which facilitates the recovery of the basic geometry of the photographed scene. Our photogrammetric modeling approach is effective, convenient, and robust because it exploits the constraints that are characteristic of architectural scenes. The second component is a *model-based* stereo algorithm, which recovers how the real scene deviates from the basic model. By making use of the model, our stereo technique robustly recovers accurate depth from widely-spaced image pairs. Consequently, our approach can model large architectural environments with far fewer photographs than current image-based modeling approaches. For producing renderings, we present *view-dependent texture mapping*, a method of compositing multiple views of a scene that better simulates geometric detail on basic models. Our approach can be used to recover models for use in either geometry-based or image-based rendering systems. We present results that demonstrate our approach's ability to create realistic renderings of architectural scenes from viewpoints far from the original photographs.

CR Descriptors: I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding - *Modeling and recovery of physical attributes*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *Color, shading, shadowing, and texture* I.4.8 [**Image Processing**]: Scene Analysis - *Stereo*; J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD).

1 INTRODUCTION

Efforts to model the appearance and dynamics of the real world have produced some of the most compelling imagery in computer graphics. In particular, efforts to model architectural scenes, from the Amiens Cathedral to the Giza Pyramids to Berkeley's Soda Hall, have produced impressive walk-throughs and inspiring flybys. Clearly, it is an attractive application to be able to explore the world's architecture unencumbered by fences, gravity, customs, or jetlag.

[1]Computer Science Division, University of California at Berkeley, Berkeley, CA 94720-1776. {debevec,camillo,malik}@cs.berkeley.edu. See also http://www.cs.berkeley.edu/~debevec/Research

Unfortunately, current geometry-based methods (Fig. 1a) of modeling existing architecture, in which a modeling program is used to manually position the elements of the scene, have several drawbacks. First, the process is extremely labor-intensive, typically involving surveying the site, locating and digitizing architectural plans (if available), or converting existing CAD data (again, if available). Second, it is difficult to verify whether the resulting model is accurate. Most disappointing, though, is that the renderings of the resulting models are noticeably computer-generated; even those that employ liberal texture-mapping generally fail to resemble real photographs.

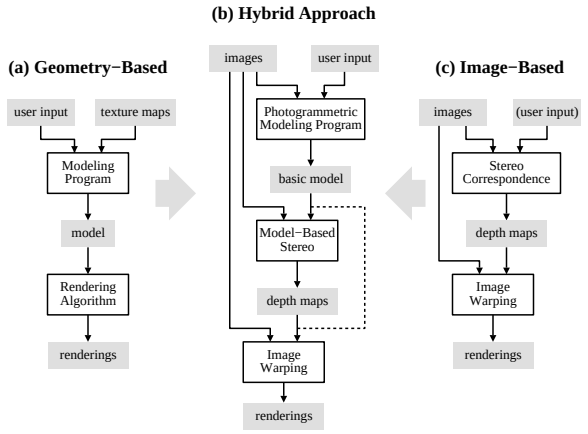


Figure 1: *Schematic of how our hybrid approach combines geometry-based and image-based approaches to modeling and rendering architecture from photographs.*

Recently, creating models directly from photographs has received increased interest in computer graphics. Since real images are used as input, such an image-based system (Fig. 1c) has an advantage in producing photorealistic renderings as output. Some of the most promising of these systems [16, 13] rely on the computer vision technique of computational stereopsis to automatically determine the structure of the scene from the multiple photographs available. As a consequence, however, these systems are only as strong as the underlying stereo algorithms. This has caused problems because state-of-the-art stereo algorithms have a number of significant weaknesses; in particular, the photographs need to appear very similar for reliable results to be obtained. Because of this, current image-based techniques must use many closely spaced images, and in some cases employ significant amounts of user input for each image pair to supervise the stereo algorithm. In this framework, capturing the data for a realistically renderable model would require an impractical number of closely spaced photographs, and deriving the depth from the photographs could require an impractical amount of user input. These concessions to the weakness of stereo algorithms bode poorly for creating large-scale, freely navigable virtual environments from photographs.

Our research aims to make the process of modeling architectural

scenes more convenient, more accurate, and more photorealistic than the methods currently available. To do this, we have developed a new approach that draws on the strengths of both geometry-based and image-based methods, as illustrated in Fig. 1b. The result is that our approach to modeling and rendering architecture requires only a sparse set of photographs and can produce realistic renderings from arbitrary viewpoints. In our approach, a basic geometric model of the architecture is recovered interactively with an easy-to-use photogrammetric modeling system, novel views are created using view-dependent texture mapping, and additional geometric detail can be recovered automatically through stereo correspondence. The final images can be rendered with current image-based rendering techniques. Because only photographs are required, our approach to modeling architecture is neither invasive nor does it require architectural plans, CAD models, or specialized instrumentation such as surveying equipment, GPS sensors or range scanners.

1.1 Background and Related Work

The process of recovering 3D structure from 2D images has been a central endeavor within computer vision, and the process of rendering such recovered structures is a subject receiving increased interest in computer graphics. Although no general technique exists to derive models from images, four particular areas of research have provided results that are applicable to the problem of modeling and rendering architectural scenes. They are: Camera Calibration, Structure from Motion, Stereo Correspondence, and Image-Based Rendering.

1.1.1 Camera Calibration

Recovering 3D structure from images becomes a simpler problem when the cameras used are *calibrated*, that is, the mapping between image coordinates and directions relative to each camera is known. This mapping is determined by, among other parameters, the camera's focal length and its pattern of radial distortion. Camera calibration is a well-studied problem both in photogrammetry and computer vision; some successful methods include [20] and [5]. While there has been recent progress in the use of uncalibrated views for 3D reconstruction [7], we have found camera calibration to be a straightforward process that considerably simplifies the problem.

1.1.2 Structure from Motion

Given the 2D projection of a point in the world, its position in 3D space could be anywhere on a ray extending out in a particular direction from the camera's optical center. However, when the projections of a sufficient number of points in the world are observed in multiple images from different positions, it is theoretically possible to deduce the 3D locations of the points as well as the positions of the original cameras, up to an unknown factor of scale.

This problem has been studied in the area of photogrammetry for the principal purpose of producing topographic maps. In 1913, Kruppa [10] proved the fundamental result that given two views of five distinct points, one could recover the rotation and translation between the two camera positions as well as the 3D locations of the points (up to a scale factor). Since then, the problem's mathematical and algorithmic aspects have been explored starting from the fundamental work of Ullman [21] and Longuet-Higgins [11], in the early 1980s. Faugeras's book [6] overviews the state of the art as of 1992. So far, a key realization has been that the recovery of structure is very sensitive to noise in image measurements when the translation between the available camera positions is small.

Attention has turned to using more than two views with image stream methods such as [19] or recursive approaches (e.g. [1]). [19] shows excellent results for the case of orthographic cameras, but direct solutions for the perspective case remain elusive. In general, linear algorithms for the problem fail to make use of all available information while nonlinear minimization methods are prone to difficulties arising from local minima in the parameter space. An alternative formulation of the problem [17] uses lines rather than points as image measurements, but the previously stated concerns were shown to remain largely valid. For purposes of computer graphics, there is yet another problem: the models recovered by these algorithms consist of sparse point fields or individual line segments, which are not directly renderable as solid 3D models.

In our approach, we exploit the fact that we are trying to recover geometric models of architectural scenes, not arbitrary three-dimensional point sets. This enables us to include additional constraints not typically available to structure from motion algorithms and to overcome the problems of numerical instability that plague such approaches. Our approach is demonstrated in a useful interactive system for building architectural models from photographs.

1.1.3 Stereo Correspondence

The geometrical theory of structure from motion assumes that one is able to solve the *correspondence* problem, which is to identify the points in two or more images that are projections of the same point in the world. In humans, corresponding points in the two slightly differing images on the retinas are determined by the visual cortex in the process called binocular stereopsis.

Years of research (e.g. [2, 4, 8, 9, 12, 15]) have shown that determining stereo correspondences by computer is difficult problem. In general, current methods are successful only when the images are similar in appearance, as in the case of human vision, which is usually obtained by using cameras that are closely spaced relative to the objects in the scene. When the distance between the cameras (often called the *baseline*) becomes large, surfaces in the images exhibit different degrees of foreshortening, different patterns of occlusion, and large disparities in their locations in the two images, all of which makes it much more difficult for the computer to determine correct stereo correspondences. Unfortunately, the alternative of improving stereo correspondence by using images taken from nearby locations has the disadvantage that computing depth becomes very sensitive to noise in image measurements.

In this paper, we show that having an approximate model of the photographed scene makes it possible to robustly determine stereo correspondences from images taken from widely varying viewpoints. Specifically, the model enables us to warp the images to eliminate unequal foreshortening and to predict major instances of occlusion *before* trying to find correspondences.

1.1.4 Image-Based Rendering

In an image-based rendering system, the model consists of a set of images of a scene and their corresponding depth maps. When the depth of every point in an image is known, the image can be re-rendered from any nearby point of view by projecting the pixels of the image to their proper 3D locations and reprojecting them onto a new image plane. Thus, a new image of the scene is created by warping the images according to their depth maps. A principal attraction of image-based rendering is that it offers a method of rendering arbitrarily complex scenes with a constant amount of computation required per pixel. Using this property, [23] demonstrated how regularly spaced synthetic images (with their computed depth maps) could be warped and composited in real time to produce a virtual environment.

More recently, [13] presented a real-time image-based rendering system that used panoramic photographs with depth computed, in part, from stereo correspondence. One finding of the paper was that extracting reliable depth estimates from stereo is "very difficult". The method was nonetheless able to obtain acceptable results for nearby views using user input to aid the stereo depth recovery: the correspondence map for each image pair was seeded with 100 to 500 user-supplied point correspondences and also post-processed. Even

with user assistance, the images used still had to be closely spaced; the largest baseline described in the paper was five feet.

The requirement that samples be close together is a serious limitation to generating a freely navigable virtual environment. Covering the size of just one city block would require thousands of panoramic images spaced five feet apart. Clearly, acquiring so many photographs is impractical. Moreover, even a dense lattice of ground-based photographs would only allow renderings to be generated from within a few feet of the original camera level, precluding any virtual fly-bys of the scene. Extending the dense lattice of photographs into three dimensions would clearly make the acquisition process even more difficult. The approach described in this paper takes advantage of the structure in architectural scenes so that it requires only a sparse set of photographs. For example, our approach has yielded a virtual fly-around of a building from just twelve standard photographs.

1.2 Overview

In this paper we present three new modeling and rendering techniques: photogrammetric modeling, view-dependent texture mapping, and model-based stereo. We show how these techniques can be used in conjunction to yield a convenient, accurate, and photorealistic method of modeling and rendering architecture from photographs. In our approach, the photogrammetric modeling program is used to create a basic volumetric model of the scene, which is then used to constrain stereo matching. Our rendering method composites information from multiple images with view-dependent texture-mapping. Our approach is successful because it splits the task of modeling from images into tasks which are easily accomplished by a person (but not a computer algorithm), and tasks which are easily performed by a computer algorithm (but not a person).

In Section 2, we present our **photogrammetric modeling** method. In essence, we have recast the structure from motion problem not as the recovery of individual point coordinates, but as the recovery of the parameters of a constrained hierarchy of parametric primitives. The result is that accurate architectural models can be recovered robustly from just a few photographs and with a minimal number of user-supplied correspondences.

In Section 3, we present **view-dependent texture mapping**, and show how it can be used to realistically render the recovered model. Unlike traditional texture-mapping, in which a single static image is used to color in each face of the model, view-dependent texture mapping interpolates between the available photographs of the scene depending on the user's point of view. This results in more lifelike animations that better capture surface specularities and unmodeled geometric detail.

Lastly, in Section 4, we present **model-based stereo**, which is used to automatically refine a basic model of a photographed scene. This technique can be used to recover the structure of architectural ornamentation that would be difficult to recover with photogrammetric modeling. In particular, we show that projecting pairs of images onto an initial approximate model allows conventional stereo techniques to robustly recover very accurate depth measurements from images with widely varying viewpoints.

2 Photogrammetric Modeling

In this section we present our method for photogrammetric modeling, in which the computer determines the parameters of a hierarchical model of parametric polyhedral primitives to reconstruct the architectural scene. We have implemented this method in *Façade*, an easy-to-use interactive modeling program that allows the user to construct a geometric model of a scene from digitized photographs. We first overview Façade from the point of view of the user, then we describe our model representation, and then we explain our reconstruction algorithm. Lastly, we present results from using Façade to reconstruct several architectural scenes.

2.1 The User's View

Constructing a geometric model of an architectural scene using Façade is an incremental and straightforward process. Typically, the user selects a small number of photographs to begin with, and models the scene one piece at a time. The user may refine the model and include more images in the project until the model meets the desired level of detail.

Fig. 2(a) and (b) shows the two types of windows used in Façade: image viewers and model viewers. The user instantiates the components of the model, marks edges in the images, and corresponds the edges in the images to the edges in the model. When instructed, Façade computes the sizes and relative positions of the model components that best fit the edges marked in the photographs.

Components of the model, called *blocks*, are parameterized geometric primitives such as boxes, prisms, and surfaces of revolution. A box, for example, is parameterized by its length, width, and height. The user models the scene as a collection of such blocks, creating new block classes as desired. Of course, the user does not need to specify numerical values for the blocks' parameters, since these are recovered by the program.

The user may choose to constrain the sizes and positions of any of the blocks. In Fig. 2(b), most of the blocks have been constrained to have equal length and width. Additionally, the four pinnacles have been constrained to have the same shape. Blocks may also be placed in constrained relations to one other. For example, many of the blocks in Fig. 2(b) have been constrained to sit centered and on top of the block below. Such constraints are specified using a graphical 3D interface. When such constraints are provided, they are used to simplify the reconstruction problem.

The user marks edge features in the images using a point-and-click interface; a gradient-based technique as in [14] can be used to align the edges with sub-pixel accuracy. We use edge rather than point features since they are easier to localize and less likely to be completely obscured. Only a section of each edge needs to be marked, making it possible to use partially visible edges. For each marked edge, the user also indicates the corresponding edge in the model. Generally, accurate reconstructions are obtained if there are as many correspondences in the images as there are free camera and model parameters. Thus, Façade reconstructs scenes accurately even when just a portion of the visible edges and marked in the images, and when just a portion of the model edges are given correspondences.

At any time, the user may instruct the computer to reconstruct the scene. The computer then solves for the parameters of the model that cause it to align with the marked features in the images. During the reconstruction, the computer computes and displays the locations from which the photographs were taken. For simple models consisting of just a few blocks, a full reconstruction takes only a few seconds; for more complex models, it can take a few minutes. For this reason, the user can instruct the computer to employ faster but less precise reconstruction algorithms (see Sec. 2.4) during the intermediate stages of modeling.

To verify the the accuracy of the recovered model and camera positions, Façade can project the model into the original photographs. Typically, the projected model deviates from the photographs by less than a pixel. Fig. 2(c) shows the results of projecting the edges of the model in Fig. 2(b) into the original photograph.

Lastly, the user may generate novel views of the model by positioning a virtual camera at any desired location. Façade will then use the view-dependent texture-mapping method of Section 3 to render a novel view of the scene from the desired location. Fig. 2(d) shows an aerial rendering of the tower model.

2.2 Model Representation

The purpose of our choice of model representation is to represent the scene as a surface model with as few parameters as possible: when

(a) (b) (c) (d)

Figure 2: **(a)** *A photograph of the Campanile, Berkeley's clock tower, with marked edges shown in green.* **(b)** *The model recovered by our photogrammetric modeling method. Although only the left pinnacle was marked, the remaining three (including one not visible) were recovered from symmetrical constraints in the model. Our method allows any number of images to be used, but in this case constraints of symmetry made it possible to recover an accurate 3D model from a single photograph.* **(c)** *The accuracy of the model is verified by reprojecting it into the original photograph through the recovered camera position.* **(d)** *A synthetic view of the Campanile generated using the view-dependent texture-mapping method described in Section 3. A real photograph from this position would be difficult to take since the camera position is 250 feet above the ground.*

the model has fewer parameters, the user needs to specify fewer correspondences, and the computer can reconstruct the model more efficiently. In Façade, the scene is represented as a constrained hierarchical model of parametric polyhedral primitives, called *blocks*. Each block has a small set of parameters which serve to define its size and shape. Each coordinate of each vertex of the block is then expressed as linear combination of the block's parameters, relative to an internal coordinate frame. For example, for the wedge block in Fig. 3, the coordinates of the vertex P_o are written in terms of the block parameters $width$, $height$, and $length$ as $P_o = (-width, -height, length)^T$. Each block is also given an associated bounding box.

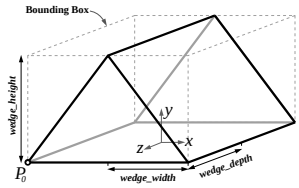
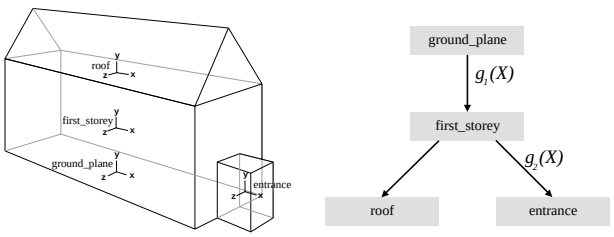


Figure 3: *A wedge block with its parameters and bounding box.*



(a) (b)

Figure 4: **(a)** *A geometric model of a simple building.* **(b)** *The model's hierarchical representation. The nodes in the tree represent parametric primitives (called* blocks*) while the links contain the spatial relationships between the blocks.*

The blocks in Façade are organized in a hierarchical tree structure as shown in Fig. 4(b). Each node of the tree represents an individual block, while the links in the tree contain the spatial relationships between blocks, called *relations*. Such hierarchical structures are also used in traditional modeling systems.

The relation between a block and its parent is most generally represented as a rotation matrix R and a translation vector t. This representation requires six parameters: three each for R and t. In architectural scenes, however, the relationship between two blocks usually has a simple form that can be represented with fewer parameters, and Façade allows the user to build such constraints on R and t into the model. The rotation R between a block and its parent can be specified in one of three ways: first, as an unconstrained rotation, requiring three parameters; second, as a rotation about a particular coordinate axis, requiring just one parameter; or third, as a fixed or null rotation, requiring no parameters.

Likewise, Façade allows for constraints to be placed on each component of the translation vector t. Specifically, the user can constrain the bounding boxes of two blocks to align themselves in some manner along each dimension. For example, in order to ensure that the roof block in Fig. 4 lies on top of the first story block, the user can require that the maximum y extent of the first story block be equal to the minimum y extent of the roof block. With this constraint, the translation along the y axis is computed ($t_y = (first\_story_y^{MAX} - roof_y^{MIN})$) rather than represented as a parameter of the model.

Each parameter of each instantiated block is actually a reference to a named symbolic variable, as illustrated in Fig. 5. As a result, two parameters of different blocks (or of the same block) can be equated by having each parameter reference the same symbol. This facility allows the user to equate two or more of the dimensions in a model, which makes modeling symmetrical blocks and repeated structure more convenient. Importantly, these constraints reduce the number of degrees of freedom in the model, which, as we will show, simplifies the structure recovery problem.

Once the blocks and their relations have been parameterized, it is straightforward to derive expressions for the world coordinates of the block vertices. Consider the set of edges which link a specific block in the model to the ground plane as shown in Fig. 4.

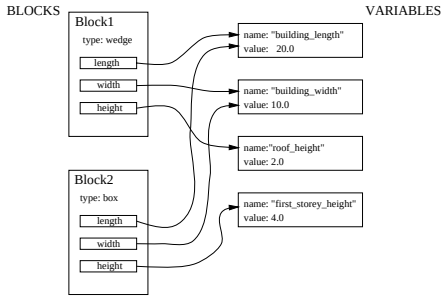


Figure 5: *Representation of block parameters as symbol references. A single variable can be referenced by the model in multiple places, allowing constraints of symmetry to be embedded in the model.*

Let $g_1(X)$,...,$g_n(X)$ represent the rigid transformations associated with each of these links, where X represents the vector of all the model parameters. The world coordinates $P_w(X)$ of a particular block vertex $P(X)$ is then:

$$P_w(X) = g_1(X)...g_n(X)P(X) \tag{1}$$

Similarly, the world orientation $v_w(X)$ of a particular line segment $v(X)$ is:

$$v_w(X) = g_1(X)...g_n(X)v(X) \tag{2}$$

In these equations, the point vectors P and P_w and the orientation vectors v and v_w are represented in homogeneous coordinates.

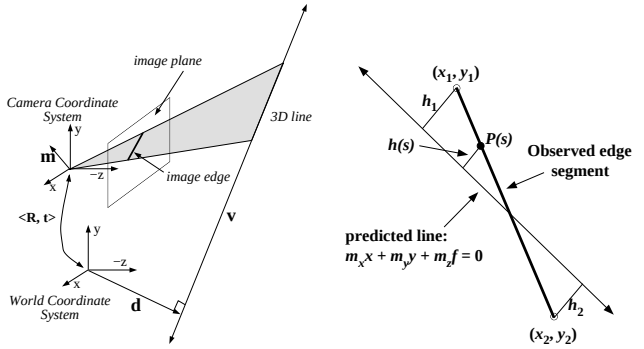
Modeling the scene with polyhedral blocks, as opposed to points, line segments, surface patches, or polygons, is advantageous for a number of reasons:

- Most architectural scenes are well modeled by an arrangement of geometric primitives.
- Blocks implicitly contain common architectural elements such as parallel lines and right angles.
- Manipulating block primitives is convenient since they are at a suitably high level of abstraction; individual features such as points and lines are less manageable.
- A surface model of the scene is readily obtained from the blocks, so there is no need to infer surfaces from discrete features.
- Modeling in terms of blocks and relationships greatly reduces the number of parameters that the reconstruction algorithm needs to recover.

The last point is crucial to the robustness of our reconstruction algorithm and the viability of our modeling system, and is illustrated best with an example. The model in Fig. 2 is parameterized by just 33 variables (the unknown camera position adds six more). If each block in the scene were unconstrained (in its dimensions and position), the model would have 240 parameters; if each line segment in the scene were treated independently, the model would have 2,896 parameters. This reduction in the number of parameters greatly enhances the robustness and efficiency of the method as compared to traditional structure from motion algorithms. Lastly, since the number of correspondences needed to suitably overconstrain the minimization is roughly proportional to the number of parameters in the model, this reduction means that the number of correspondences required of the user is manageable.

2.3 Reconstruction Algorithm

Our reconstruction algorithm works by minimizing an objective function $\mathcal{O}$ that sums the disparity between the projected edges of the model and the edges marked in the images, i.e. $\mathcal{O} = \sum Err_i$ where Err_i represents the disparity computed for edge feature i. Thus, the unknown model parameters and camera positions are computed by minimizing $\mathcal{O}$ with respect to these variables. Our system uses the the error function Err_i from [17], described below.



(a) (b)

Figure 6: **(a)** *Projection of a straight line onto a camera's image plane.* **(b)** *The error function used in the reconstruction algorithm. The heavy line represents the observed edge segment (marked by the user) and the lighter line represents the model edge predicted by the current camera and model parameters.*

Fig. 6(a) shows how a straight line in the model projects onto the image plane of a camera. The straight line can be defined by a pair of vectors $\langle v,d \rangle$ where v represents the direction of the line and d represents a point on the line. These vectors can be computed from equations 2 and 1 respectively. The position of the camera with respect to world coordinates is given in terms of a rotation matrix R_j and a translation vector t_j. The normal vector denoted by $\mathbf{m}$ in the figure is computed from the following expression:

$$\mathbf{m} = R_j(\mathbf{v} \times (\mathbf{d} - t_j)) \tag{3}$$

The projection of the line onto the image plane is simply the intersection of the plane defined by m with the image plane, located at $z = -f$ where f is the focal length of the camera. Thus, the image edge is defined by the equation $m_x x + m_y y - m_z f = 0$.

Fig. 6(b) shows how the error between the observed image edge $\{(x_1, y_1), (x_2, y_2)\}$ and the predicted image line is calculated for each correspondence. Points on the observed edge segment can be parameterized by a single scalar variable $s \in [0, l]$ where l is the length of the edge. We let $h(s)$ be the function that returns the shortest distance from a point on the segment, $p(s)$, to the predicted edge.

With these definitions, the total error between the observed edge segment and the predicted edge is calculated as:

$$Err_i = \int_0^l h^2(s)ds = \frac{l}{3}(h_1^2 + h_1 h_2 + h_2^2) = \mathbf{m}^T(A^T B A)\mathbf{m} \tag{4}$$

where:

$$\mathbf{m} = (m_x, m_y, m_z)^T$$

$$A = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} \frac{l}{3(m_x^2 + m_y^2)^{0.5}} & 0 \\ 0 & \frac{l}{3(m_x^2 + m_y^2)^{0.5}} \end{pmatrix}^{1.5}$$

The final objective function $\mathcal{O}$ is the sum of the error terms resulting from each correspondence. We minimize $\mathcal{O}$ using a variant of the Newton-Raphson method, which involves calculating the gradient and Hessian of $\mathcal{O}$ with respect to the parameters of the camera

and the model. As we have shown, it is simple to construct symbolic expressions for m in terms of the unknown model parameters. The minimization algorithm differentiates these expressions symbolically to evaluate the gradient and Hessian after each iteration. The procedure is inexpensive since the expressions for d and v in Equations 2 and 1 have a particularly simple form.

2.4 Computing an Initial Estimate

The objective function described in Section 2.3 section is non-linear with respect to the model and camera parameters and consequently can have local minima. If the algorithm begins at a random location in the parameter space, it stands little chance of converging to the correct solution. To overcome this problem we have developed a method to directly compute a good initial estimate for the model parameters and camera positions that is near the correct solution. In practice, our initial estimate method consistently enables the non-linear minimization algorithm to converge to the correct solution.

Our initial estimate method consists of two procedures performed in sequence. The first procedure estimates the camera rotations while the second estimates the camera translations and the parameters of the model. Both initial estimate procedures are based upon an examination of Equation 3. From this equation the following constraints can be deduced:

$$m^T R_j \mathbf{v} = 0 \quad (5)$$

$$m^T R_j (\mathbf{d} - t_j) = 0 \quad (6)$$

Given an observed edge $\mathbf{u}_{ij}$ the measured normal $\mathbf{m}'$ to the plane passing through the camera center is:

$$\mathbf{m}' = \begin{pmatrix} x_1 \\ y_1 \\ -f \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ -f \end{pmatrix} \quad (7)$$

From these equations, we see that any model edges of known orientation constrain the possible values for R_j. Since most architectural models contain many such edges (e.g. horizontal and vertical lines), each camera rotation can be usually be estimated from the model independent of the model parameters and independent of the camera's location in space. Our method does this by minimizing the following objective function $\mathcal{O}_1$ that sums the extents to which the rotations R_j violate the constraints arising from Equation 5:

$$\mathcal{O}_1 = \sum_i (m^T R_j \mathbf{v}_i)^2, \quad \mathbf{v}_i \in \{\hat{x}, \hat{y}, \hat{z}\} \quad (8)$$

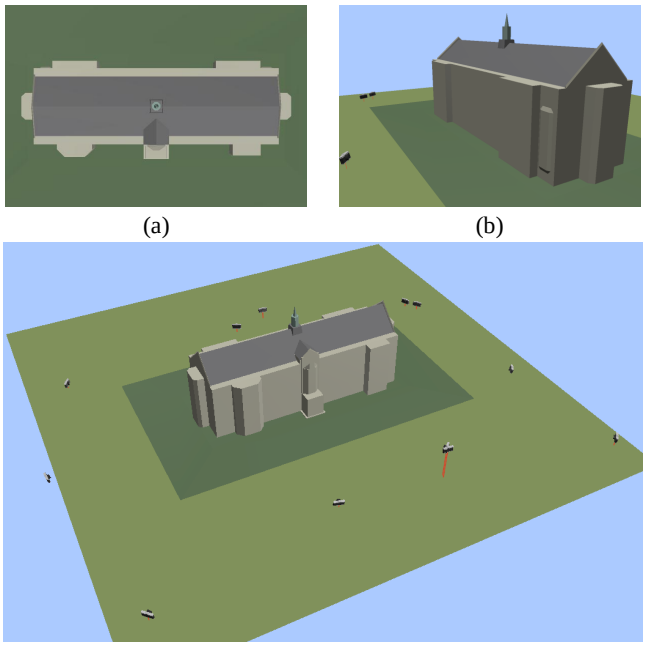
Once initial estimates for the camera rotations are computed, Equation 6 is used to obtain initial estimates of the model parameters and camera locations. Equation 6 reflects the constraint that all of the points on the line defined by the tuple $\langle \mathbf{v}, \mathbf{d} \rangle$ should lie on the plane with normal vector $\mathbf{m}$ passing through the camera center. This constraint is expressed in the following objective function $\mathcal{O}_2$ where $P_i(X)$ and $Q_i(X)$ are expressions for the vertices of an edge of the model.

$$\mathcal{O}_2 = \sum_i (m^T R_j (P_i(X) - t_j))^2 + (m^T R_j (Q_i(X) - t_j))^2 \quad (9)$$

In the special case where all of the block relations in the model have a known rotation, this objective function becomes a simple quadratic form which is easily minimized by solving a set of linear equations.

Once the initial estimate is obtained, the non-linear minimization over the entire parameter space is applied to produce the best possible reconstruction. Typically, the minimization requires fewer than ten iterations and adjusts the parameters of the model by at most a few percent from the initial estimates. The edges of the recovered models typically conform to the original photographs to within a pixel.

Figure 7: *Three of twelve photographs used to reconstruct the entire exterior of University High School in Urbana, Illinois. The superimposed lines indicate the edges the user has marked.*

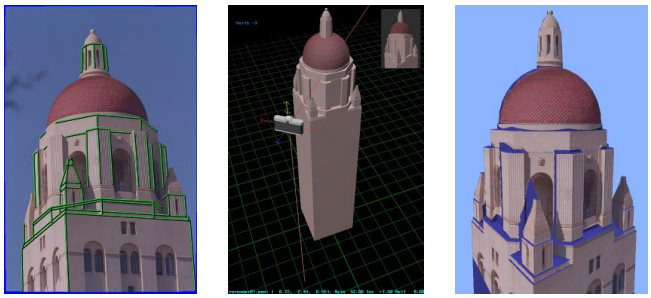


(a) (b)

(c)

Figure 8: *The high school model, reconstructed from twelve photographs.* **(a)** *Overhead view.* **(b)** *Rear view.* **(c)** *Aerial view showing the recovered camera positions. Two nearly coincident cameras can be observed in front of the building; their photographs were taken from the second story of a building across the street.*

Figure 9: *A synthetic view of University High School. This is a frame from an animation of flying around the entire building.*



(a) (b) (c)

Figure 10: *Reconstruction of Hoover Tower, Stanford, CA* **(a)** *Original photograph, with marked edges indicated.* **(b)** *Model recovered from the single photograph shown in (a).* **(c)** *Texture-mapped aerial view from the virtual camera position indicated in (b). Regions not seen in (a) are indicated in blue.*

2.5 Results

Fig. 2 showed the results of using Façade to reconstruct a clock tower from a single image. Figs. 7 and 8 show the results of using Façade to reconstruct a high school building from twelve photographs. (The model was originally constructed from just five images; the remaining images were added to the project for purposes of generating renderings using the techniques of Section 3.) The photographs were taken with a calibrated 35mm still camera with a standard 50mm lens and digitized with the PhotoCD process. Images at the 1536×1024 pixel resolution were processed to correct for lens distortion, then filtered down to 768×512 pixels for use in the modeling system. Fig. 8 shows some views of the recovered model and camera positions, and Fig. 9 shows a synthetic view of the building generated by the technique in Sec. 3.

Fig. 10 shows the reconstruction of another tower from a single photograph. The dome was modeled specially since the reconstruction algorithm does not recover curved surfaces. The user constrained a two-parameter hemisphere block to sit centered on top of the tower, and manually adjusted its height and width to align with the photograph. Each of the models presented took approximately four hours to create.

3 View-Dependent Texture-Mapping

In this section we present view-dependent texture-mapping, an effective method of rendering the scene that involves projecting the original photographs onto the model. This form of texture-mapping is most effective when the model conforms closely to the actual structure of the scene, and when the original photographs show the scene in similar lighting conditions. In Section 4 we will show how view-dependent texture-mapping can be used in conjunction with model-based stereo to produce realistic renderings when the recovered model only approximately models the structure of the scene.

Since the camera positions of the original photographs are recovered during the modeling phase, projecting the images onto the model is straightforward. In this section we first describe how we project a single image onto the model, and then how we merge several image projections to render the entire model. Unlike traditional texture-mapping, our method projects different images onto the model depending on the user's viewpoint. As a result, our view-dependent texture mapping can give a better illusion of additional geometric detail in the model.

3.1 Projecting a Single Image

The process of texture-mapping a single image onto the model can be thought of as replacing each camera with a slide projector that projects the original image onto the model. When the model is not convex, it is possible that some parts of the model will shadow others with respect to the camera. While such shadowed regions could be determined using an object-space visible surface algorithm, or an image-space ray casting algorithm, we use an image-space shadow map algorithm based on [22] since it is efficiently implemented using z-buffer hardware.

Fig. 11, upper left, shows the results of mapping a single image onto the high school building model. The recovered camera position for the projected image is indicated in the lower left corner of the image. Because of self-shadowing, not every point on the model within the camera's viewing frustum is mapped.

3.2 Compositing Multiple Images

In general, each photograph will view only a piece of the model. Thus, it is usually necessary to use multiple images in order to render the entire model from a novel point of view. The top images of Fig. 11 show two different images mapped onto the model and rendered from a novel viewpoint. Some pixels are colored in just one of the renderings, while some are colored in both. These two renderings can be merged into a composite rendering by considering the corresponding pixels in the rendered views. If a pixel is mapped in only one rendering, its value from that rendering is used in the composite. If it is mapped in more than one rendering, the renderer has to decide which image (or combination of images) to use.

It would be convenient, of course, if the projected images would agree perfectly where they overlap. However, the images will not necessarily agree if there is unmodeled geometric detail in the building, or if the surfaces of the building exhibit non-Lambertian reflection. In this case, the best image to use is clearly the one with the viewing angle closest to that of the rendered view. However, using the image closest in angle at every pixel means that neighboring rendered pixels may be sampled from different original images. When this happens, specularity and unmodeled geometric detail can cause visible seams in the rendering. To avoid this problem, we smooth these transitions through weighted averaging as in Fig. 12.

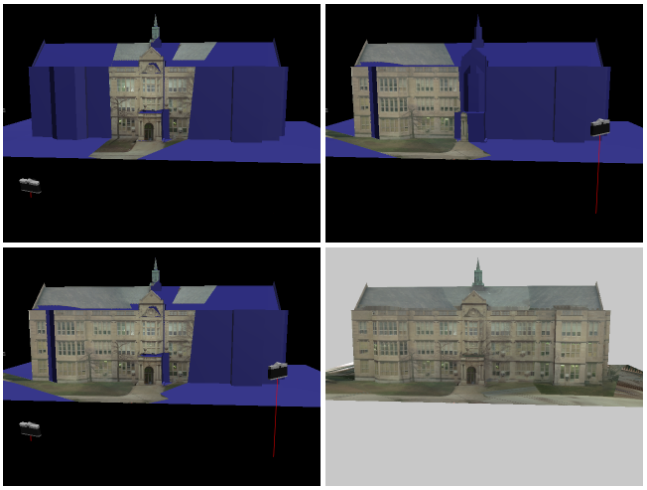


Figure 11: *The process of assembling projected images to form a composite rendering. The top two pictures show two images projected onto the model from their respective recovered camera positions. The lower left picture shows the results of compositing these two renderings using our view-dependent weighting function. The lower right picture shows the results of compositing renderings of all twelve original images. Some pixels near the front edge of the roof not seen in any image have been filled in with the hole-filling algorithm from [23].*

Even with this weighting, neighboring pixels can still be sampled from different views at the boundary of a projected image, since the contribution of an image must be zero outside its boundary. To

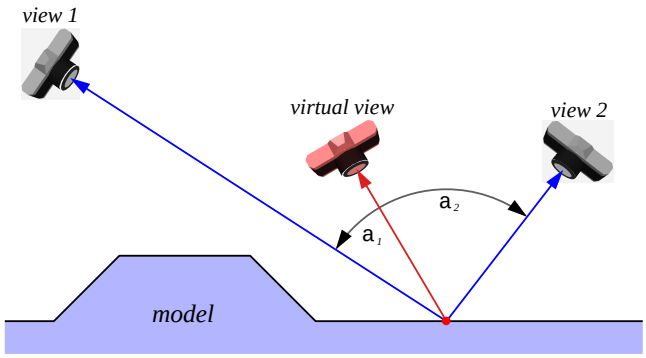


Figure 12: *The weighting function used in view-dependent texture mapping. The pixel in the virtual view corresponding to the point on the model is assigned a weighted average of the corresponding pixels in actual views 1 and 2. The weights w_1 and w_2 are inversely inversely proportional to the magnitude of angles α_1 and α_2. Alternately, more sophisticated weighting functions based on expected foreshortening and image resampling can be used.*

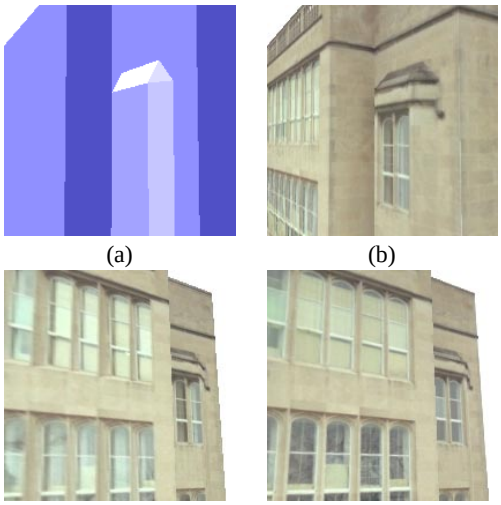


Figure 13: *View-dependent texture mapping.* **(a)** *A detail view of the high school model.* **(b)** *A rendering of the model from the same position using view-dependent texture mapping. Note that although the model does not capture the slightly recessed windows, the windows appear properly recessed because the texture map is sampled primarily from a photograph which viewed the windows from approximately the same direction.* **(c)** *The same piece of the model viewed from a different angle, using the same texture map as in (b). Since the texture is not selected from an image that viewed the model from approximately the same angle, the recessed windows appear unnatural.* **(d)** *A more natural result obtained by using view-dependent texture mapping. Since the angle of view in (d) is different than in (b), a different composition of original images is used to texture-map the model.*

address this, the pixel weights are ramped down near the boundary of the projected images. Although this method does not guarantee smooth transitions in all cases, we have found that it eliminates most artifacts in renderings and animations arising from such seams.

If an original photograph features an unwanted car, tourist, or other object in front of the architecture of interest, the unwanted object will be projected onto the surface of the model. To prevent this from happening, the user may mask out the object by painting over the obstruction with a reserved color. The rendering algorithm will then set the weights for any pixels corresponding to the masked regions to zero, and decrease the weights of the pixels near the boundary as before to minimize seams. Any regions in the composite image which are occluded in every projected image are filled in using the hole-filling method from [23].

In the discussion so far, projected image weights are computed at every pixel of every projected rendering. Since the weighting function is smooth (though not constant) across flat surfaces, it is not generally not necessary to compute it for every pixel of every face of the model. For example, using a single weight for each face of the model, computed at the face's center, produces acceptable results. By coarsely subdividing large faces, the results are visually indistinguishable from the case where a unique weight is computed for every pixel. Importantly, this technique suggests a real-time implementation of view-dependent texture mapping using a texture-mapping graphics pipeline to render the projected views, and α-channel blending to composite them.

For complex models where most images are entirely occluded for the typical view, it can be very inefficient to project every original photograph to the novel viewpoint. Some efficient techniques to determine such visibility *a priori* in architectural scenes through spatial partitioning are presented in [18].

4 Model-Based Stereopsis

The modeling system described in Section 2 allows the user to create a basic model of a scene, but in general the scene will have additional geometric detail (such as friezes and cornices) not captured in the model. In this section we present a new method of recovering such additional geometric detail automatically through stereo correspondence, which we call *model-based* stereo. Model-based stereo differs from traditional stereo in that it measures how the actual scene deviates from the approximate model, rather than trying to measure the structure of the scene without any prior information. The model serves to place the images into a common frame of reference that makes the stereo correspondence possible even for images taken from relatively far apart. The stereo correspondence information can then be used to render novel views of the scene using image-based rendering techniques.

As in traditional stereo, given two images (which we call the *key* and *offset*), model-based stereo computes the associated depth map for the key image by determining corresponding points in the key and offset images. Like many stereo algorithms, our method is *correlation-based*, in that it attempts to determine the corresponding point in the offset image by comparing small pixel neighborhoods around the points. As such, correlation-based stereo algorithms generally require the neighborhood of each point in the key image to resemble the neighborhood of its corresponding point in the offset image.

The problem we face is that when the key and offset images are taken from relatively far apart, as is the case for our modeling method, corresponding pixel neighborhoods can be foreshortened very differently. In Figs. 14(a) and (c), pixel neighborhoods toward the right of the key image are foreshortened horizontally by nearly a factor of four in the offset image.

The key observation in model-based stereo is that even though two images of the same scene may appear very different, they appear similar after being projected onto an approximate model of the scene. In particular, projecting the offset image onto the model and viewing it from the position of the key image produces what we call the *warped offset* image, which appears very similar to the key image. The geometrically detailed scene in Fig. 14 was modeled as two flat surfaces with our modeling program, which also determined the relative camera positions. As expected, the warped offset image (Fig. 14(b)) exhibits the same pattern of foreshortening as the key image.

In model-based stereo, pixel neighborhoods are compared between the key and warped offset images rather than the key and off-

(a) *Key Image* (b) *Warped Offset Image* (c) *Offset Image* (d) *Computed Disparity Map*

Figure 14: **(a)** *and* **(c)** *Two images of the entrance to Peterhouse chapel in Cambridge, UK. The Façade program was used to model the façade and ground as a flat surfaces and to recover the relative camera positions.* **(b)** *The warped offset image, produced by projecting the offset image onto the approximate model and viewing it from the position of the key camera. This projection eliminates most of the disparity and foreshortening with respect to the key image, greatly simplifying stereo correspondence.* **(d)** *An unedited disparity map produced by our model-based stereo algorithm.*

set images. When a correspondence is found, it is simple to convert its disparity to the corresponding disparity between the key and offset images, from which the point's depth is easily calculated. Fig. 14(d) shows a disparity map computed for the key image in (a).

The reduction of differences in foreshortening is just one of several ways that the warped offset image simplifies stereo correspondence. Some other desirable properties of the warped offset image are:

- Any point in the scene which lies on the approximate model will have zero disparity between the key image and the warped offset image.
- Disparities between the key and warped offset images are easily converted to a depth map for the key image.
- Depth estimates are far less sensitive to noise in image measurements since images taken from relatively far apart can be compared.
- Places where the model occludes itself relative to the key image can be detected and indicated in the warped offset image.
- A linear epipolar geometry (Sec. 4.1) exists between the key and warped offset images, despite the warping. In fact, the epipolar lines of the warped offset image coincide with the epipolar lines of the key image.

4.1 Model-Based Epipolar Geometry

In traditional stereo, the *epipolar constraint* (see [6]) is often used to constrain the search for corresponding points in the offset image to searching along an epipolar line. This constraint simplifies stereo not only by reducing the search for each correspondence to one dimension, but also by reducing the chance of selecting a false matches. In this section we show that taking advantage of the epipolar constraint is no more difficult in model-based stereo case, despite the fact that the offset image is non-uniformly warped.

Fig. 15 shows the epipolar geometry for model-based stereo. If we consider a point P in the scene, there is a unique *epipolar plane* which passes through P and the centers of the key and offset cameras. This epipolar plane intersects the key and offset image planes in *epipolar lines* e_k and e_o. If we consider the projection p_k of P onto the key image plane, the epipolar constraint states that the corresponding point in the offset image must lie somewhere along the offset image's epipolar line.

In model-based stereo, neighborhoods in the key image are compared to the warped offset image rather than the offset image. Thus, to make use of the epipolar constraint, it is necessary to determine where the pixels on the offset image's epipolar line project to in the warped offset image. The warped offset image is formed by projecting the offset image onto the model, and then reprojecting the model onto the image plane of the key camera. Thus, the projection p_o of P in the offset image projects onto the model at Q, and then reprojects to q_k in the warped offset image. Since each of these projections occurs within the epipolar plane, any possible correspondence

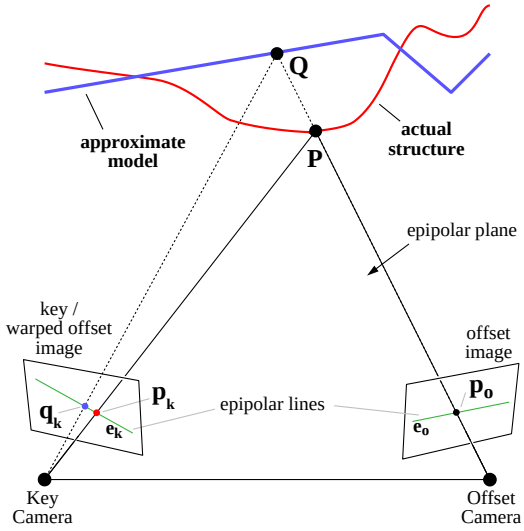


Figure 15: *Epipolar geometry for model-based stereo.*

for p_k in the key image must lie on the *key* image's epipolar line in the warped offset image. In the case where the actual structure and the model coincide at P, p_o is projected to P and then reprojected to p_k, yielding a correspondence with zero disparity.

The fact that the epipolar geometry remains linear after the warping step also facilitates the use of the ordering constraint [2, 6] through a dynamic programming technique.

4.2 Stereo Results and Rerendering

While the warping step makes it dramatically easier to determine stereo correspondences, a stereo algorithm is still necessary to actually determine them. The algorithm we developed to produce the images in this paper is described in [3].

Once a depth map has been computed for a particular image, we can rerender the scene from novel viewpoints using the methods described in [23, 16, 13]. Furthermore, when several images and their corresponding depth maps are available, we can use the view-dependent texture-mapping method of Section 3 to composite the multiple renderings. The novel views of the chapel façade in Fig. 16 were produced through such compositing of four images.

5 Conclusion and Future Work

To conclude, we have presented a new, photograph-based approach to modeling and rendering architectural scenes. Our modeling approach, which combines both geometry-based and image-based modeling techniques, is built from two components that we have developed. The first component is an easy-to-use photogrammet-

Figure 16: *Novel views of the scene generated from four original photographs. These are frames from an animated movie in which the façade rotates continuously. The depth is computed from model-based stereo and the frames are made by compositing image-based renderings with view-dependent texture-mapping.*

ric modeling system which facilitates the recovery of a basic geometric model of the photographed scene. The second component is a model-based stereo algorithm, which recovers precisely how the real scene differs from the basic model. For rendering, we have presented view-dependent texture-mapping, which produces images by warping and compositing multiple views of the scene. Through judicious use of images, models, and human assistance, our approach is more convenient, more accurate, and more photorealistic than current geometry-based or image-based approaches for modeling and rendering real-world architectural scenes.

There are several improvements and extensions that can be made to our approach. First, surfaces of revolution represent an important component of architecture (e.g. domes, columns, and minarets) that are not recovered in our photogrammetric modeling approach. (As noted, the dome in Fig. 10 was manually sized by the user.) Fortunately, there has been much work (e.g. [24]) that presents methods of recovering such structures from image contours. Curved model geometry is also entirely consistent with our approach to recovering additional detail with model-based stereo.

Second, our techniques should be extended to recognize and model the photometric properties of the materials in the scene. The system should be able to make better use of photographs taken in varying lighting conditions, and it should be able to render images of the scene as it would appear at any time of day, in any weather, and with any configuration of artificial light. Already, the recovered model can be used to predict shadowing in the scene with respect to an arbitrary light source. However, a full treatment of the problem will require estimating the photometric properties (i.e. the bidirectional reflectance distribution functions) of the surfaces in the scene.

Third, it is clear that further investigation should be made into the problem of selecting which original images to use when rendering a novel view of the scene. This problem is especially difficult when the available images are taken at arbitrary locations. Our current solution to this problem, the weighting function presented in Section 3, still allows seams to appear in renderings and does not consider issues arising from image resampling. Another form of view selection is required to choose which pairs of images should be matched to recover depth in the model-based stereo algorithm.

Lastly, it will clearly be an attractive application to integrate the models created with the techniques presented in this paper into forthcoming real-time image-based rendering systems.

Acknowledgments

This research was supported by a National Science Foundation Graduate Research Fellowship and grants from Interval Research Corporation, the California MICRO program, and JSEP contract F49620-93-C-0014. The authors also wish to thank Tim Hawkins, Carlo Séquin, David Forsyth, and Jianbo Shi for their valuable help in revising this paper.

References

[1] Ali Azarbayejani and Alex Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Trans. Pattern Anal. Machine Intell.*, 17(6):562–575, June 1995.

[2] H. H. Baker and T. O. Binford. Depth from edge and intensity based stereo. In *Proceedings of the Seventh IJCAI, Vancouver, BC*, pages 631–636, 1981.

[3] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. Technical Report UCB//CSD-96-893, U.C. Berkeley, CS Division, January 1996.

[4] D.J.Fleet, A.D.Jepson, and M.R.M. Jenkin. Phase-based disparity measurement. *CVGIP: Image Understanding*, 53(2):198–210, 1991.

[5] Oliver Faugeras and Giorgio Toscani. The calibration problem for stereo. In *Proceedings IEEE CVPR 86*, pages 15–20, 1986.

[6] Olivier Faugeras. *Three-Dimensional Computer Vision*. MIT Press, 1993.

[7] Olivier Faugeras, Stephane Laveau, Luc Robert, Gabriella Csurka, and Cyril Zeller. 3-d reconstruction of urban scenes from sequences of images. Technical Report 2572, INRIA, June 1995.

[8] W. E. L. Grimson. *From Images to Surface*. MIT Press, 1981.

[9] D. Jones and J. Malik. Computational framework for determining stereo correspondence from a set of linear spatial filters. *Image and Vision Computing*, 10(10):699–708, December 1992.

[10] E. Kruppa. Zur ermittlung eines objectes aus zwei perspektiven mit innerer orientierung. *Sitz.-Ber. Akad. Wiss., Wien, Math. Naturw. Kl., Abt. IIa.*, 122:1939–1948, 1913.

[11] H.C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981.

[12] D. Marr and T. Poggio. A computational theory of human stereo vision. *Proceedings of the Royal Society of London*, 204:301–328, 1979.

[13] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH '95*, 1995.

[14] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *SIGGRAPH '95*, 1995.

[15] S. B. Pollard, J. E. W. Mayhew, and J. P. Frisby. A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14:449–470, 1985.

[16] R. Szeliski. Image mosaicing for tele-reality applications. In *IEEE Computer Graphics and Applications*, 1996.

[17] Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Trans. Pattern Anal. Machine Intell.*, 17(11), November 1995.

[18] S. J. Teller, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity computations. In *SIGGRAPH '94*, pages 443–450, 1994.

[19] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, November 1992.

[20] Roger Tsai. A versatile camera calibration technique for high accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):323–344, August 1987.

[21] S. Ullman. *The Interpretation of Visual Motion*. The MIT Press, Cambridge, MA, 1979.

[22] L Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78*, pages 270–274, 1978.

[23] Lance Williams and Eric Chen. View interpolation for image synthesis. In *SIGGRAPH '93*, 1993.

[24] Mourad Zerroug and Ramakant Nevatia. Segmentation and recovery of shgcs from a real intensity image. In *European Conference on Computer Vision*, pages 319–330, 1994.

Image-Based Rendering using Image-Warping – Motivation and Background

Leonard McMillan
LCS Computer Graphics Group
MIT

The field of three-dimensional computer graphics has long focused on the problem of synthesizing images from geometric models. These geometric models, in combination with surface descriptions characterizing the reflective properties of each element, represent the scene that is to be rendered. Computationally, the classical computer-graphics image synthesis process is a simulation problem in which light's interactions with the supplied scene description are computed.

Conventional computer vision considers the opposite problem of generating geometric models from images. In addition to images, computer-vision systems depend on accurate camera models and estimates of a camera's position and orientation in order to synthesize the desired geometric models. Often a simplified surface reflectance model is assumed as part of the computer vision algorithm.

The efforts of computer graphics and computer vision are generally perceived as complementary because the results of one field can frequently serve as an input to the other. Computer graphics often looks to the field of computer vision for the generation of complex geometric models, whereas computer vision relies on computer graphics for viewing results. Three-dimensional geometry has been the fundamental interface between the fields of computer vision and computer graphics since their inception. Only recently has this interface come under question. To a large extent the field of *image-based rendering* suggests an alternative interface between the image analysis of computer vision and the image synthesis of computer graphics. In this course I will describe one class of methods for synthesizing images, comparable to those produced by conventional three-dimensional computer graphics methods, directly from other images without an explicit three-dimensional geometric representation.

Another motivation for the development of image-based rendering techniques is that, while geometry-based rendering technology has made significant strides towards achieving photorealism, the process of creating accurate models is still nearly as difficult today as it was twenty-five years ago. Technological advances in three-dimensional scanning methods provide some promise for simplifying the process of model building. However, these automated model acquisition methods also verify our worst suspicions—the geometry of the real world is exceedingly complex. Ironically, one of the primary subjective measures of image quality used in geometry-based computer graphics is the degree to which a rendered image is indistinguishable from a photograph. Consider, though, the advantages of using photographs (images) as the underlying scene representation. Photographs, unlike geometric models, are both plentiful and easily acquired, and, needless to say photorealistic. Images are capable of representing both the geometric complexity and photometric realism of a scene in a way that is currently beyond our modeling capabilities.

Throughout the three-dimensional computer graphics community, researchers, users, and hardware developers alike, have realized the significant advantages of incorporating

images, in the form of texture maps, into traditional three-dimensional models. Texture maps are commonly used to add fine surface property variations as well as to substitute for small-scale geometric details. Texture mapping can rightfully be viewed as the precursor to image-based computer graphics methods. In fact, the image-based approach that I present can be viewed as an extension of texture-mapping algorithms commonly used today. However, unlike a purely image-based approach, an underlying three-dimensional model still plays a crucial role with traditional texture maps.

In order to define an image-based computer graphics method, we need a principled process for transforming a finite set of known images, which I will henceforth refer to as reference images, into new images as they would be seen from arbitrary viewpoints. I will call these synthesized images, desired images. Techniques for deriving new images based on a series of reference images or drawings are not new. A skilled architect, artist, draftsman, or illustrator can, with relative ease, generate accurate new renderings of an object based on surprisingly few reference images. These reference images are frequently illustrations made from certain cardinal views, but it is not uncommon for them to be actual photographs of the desired scene taken from a different viewpoint. One characterization of image-based rendering is to emulate the finely honed skills of these artisans by using computational powers.

While image-based computer graphics has come many centuries after the discovery of perspective illustration techniques by artists, its history is still nearly as long as that of geometry-based computer graphics. Progress in the field of image-based computer graphics can be traced through at least three different scientific disciplines. In photogrammetry the problems of distortion correction, image registration, and photometrics have progressed toward the synthesis of desired images through the composition of reference images. Likewise, in computer vision, problems such as navigation, discrimination, and image understanding have naturally led in the same direction. In computer graphics, as discussed previously, the progression toward image-based rendering systems was initially motivated by the desire to increase the visual realism of the approximate geometric descriptions. Most recently, methods have been introduced in which the images alone constitute the overall scene description. The remainder of this introduction discusses previous works in image-based computer graphics and their relationship to the image-warping approach that I will present.

In recent years, images have supplemented the image generation process in several different capacities. Images have been used to represent approximations of the geometric contents of a scene. Collections of images have been employed as databases from which views of a desired environment are queried. And, most recently, images have been employed as full-fledged scene models from which desired views are synthesized. In this section, I will give an overview of the previous work in image-based computer graphics partitioned along these three lines.

Images, mapped onto simplified geometry, are often used as an approximate representation of visual environments. Texture mapping is perhaps the most obvious example of this use. Another more subtle approximation involves the assumption that all, or most, of the geometric content of a scene is located so far away from the viewer that its actual shape is inconsequential. Much of the pioneering work in texture mapping is attributable to the classic work of Catmull, Blinn, and Newell. The flexibility of image textures as three-dimensional computer graphics primitives has since been extended to include small perturbations in surface orientation (bump maps) [Blinn76] and approximations to global illumination (environment and shadow mapping) [Blinn76] [Greene86] [Segal92]. Recent developments in texture mapping have concentrated on the use of visually rich textures mapped onto very approximate geometric descriptions [Shade96] [Aliaga96][Schaufler96].

Texture mapping techniques rely on mapping functions to specify the relationship of the texture's image-space coordinates to their corresponding position on a three-dimensional

model. A comprehensive discussion of these mapping techniques was undertaken in [Heckbert86]. In practice the specification of this mapping is both difficult and time consuming, and often requires considerable human intervention. As a result, the most commonly used mapping methods are restricted to very simple geometric descriptions, such as polygonal facets, spheres and cylinders.

During the rendering process, these texture-to-model mapping functions undergo another mapping associated with the perspective-projection process. This second mapping is from the three-dimensional space of the scene's representation to the coordinate space of the desired image. In actual rendering systems, one or both of these mapping processes occurs in the opposite or inverse order. For instance, when ray tracing, the mapping of the desired image's coordinates to the three-dimensional coordinates in the space of the visible object occurs first. Then, the mapping from the three-dimensional object's coordinates to the texture's image-space coordinates is found. Likewise, in z-buffering based methods, the mapping from the image-space coordinate to texture's image-space occurs during the rasterization process. These inverse methods are known to be subject to aliasing and reconstruction artifacts. Many techniques, including mip-maps [Williams83] and summed-area tables [Crow84] [Glassner86], have been suggested to address these problems.

Another fundamental limitation of texture maps is that they rely solely on the geometry of the underlying three-dimensional model to specify the object's shape. The precise representation of three-dimensional shape using primitives suitable for the traditional approach to computer graphics is, in itself, a difficult problem that has long been an active topic in computer graphics research. When the difficulties of representing three-dimensional shape are combined with the issues of associating a texture coordinate to each point on the surface (not to mention the difficulties of acquiring suitable textures in the first place), the problem becomes even more difficult. It is conceivable that, given a series of photographs, a three-dimensional computer model could be assembled. And, from those same photographs, various figures might be identified, cropped, and the perspective distortions removed so that a texture might be extracted. Then, using traditional three-dimensional computer graphics methods, renderings of any desired image could be computed. While the process outlined is credible, it is both tedious and prone to errors. The image-based approach to computer graphics described in this thesis attempts to sidestep many of these intermediate steps by defining mapping functions from the image-space of one or more reference images directly to the image-space of a desired image.

A new class of scene approximation results when an image is mapped onto the set of points at infinity. The mapping is accomplished in exactly the same way that texture maps are applied to spheres, since each point on a sphere can be directly associated with another point located an infinite distance from the sphere's center. This observation is also the basis of environment maps. However, environment maps are observed indirectly as either reflections within other objects or as representations of a scene's illumination environment. When such an image mapping is intended for direct viewing, a new type of scene representation results. An image-based computer graphics system of this type, called QuickTimeVR [Chen95], has been developed by Apple Computer. In QuickTimeVR, the underlying scene is represented by a set of cylindrical images. The system is able to synthesize new planar views in response to a user's input by warping one of these cylindrical images. This is accomplished at highly interactive rates (greater than 20 frames per second) and is done entirely in software. The system adapts both the resolution and reconstruction filter quality based on the rate of the interaction. QuickTimeVR must be credited with exposing to a wide audience the vast potential of image-based computer graphics. The QuickTimeVR system is a reprojection method. It is only capable of describing image variations due to changes in viewing orientation. Translations of the viewing position can only be approximated by selecting the cylindrical image whose center-of-projection is closest to the current viewing position.

The panoramic representation afforded by the cylindrical image description provides many practical advantages. It immerses the user within the visual environment, and it eliminates the need to consider the viewing angle when determining which reference image is closest to a desired view. However, several normal photographs are required to create a single cylindrical projection. These images must be properly registered and then reprojected to construct the cylindrical reference image. QuickTimeVR's image-based approach has significant similarity to the approach described here. Its rendering process is a special case of the cylinder-to-plane warping equation in the case where all image points are computed as if they were an infinite distance from the observer.

The movie-map system by Lippman [Lippman80] was one of the earliest attempts at constructing a purely image-based computer graphics system. In a movie-map, many thousands of reference images were stored on interactive video laser disks. These images could be accessed randomly, according to the current viewpoint of the user. The system could also accommodate simple panning, tilting, or zooming about these fixed viewing positions. The movie-map approach to image-based computer graphics can also be interpreted as a table-based approach, where the rendering process is replaced by a database query into a vast set of reference images. This database-like structure is common to other image-based computer graphics systems. Movie-maps were unable to reconstruct all possible desired views. Even with the vast storage capacities currently available on media such as laser disks, and the rapid development of even higher capacity storage media, the space of all possible desired images appears so large that any purely database-oriented approach will continue to be impractical in the near future. Also, the very subtle differences between images observed from nearby points under similar viewing conditions bring into question the overall efficiency of this approach. The image-based rendering approach described here could be viewed as a reasonable compression method for movie maps.

Levoy and Hanrahan [Levoy96] have developed another database approach to image-based computer graphics in which the underlying modeling primitives are rays rather than images. A key innovation of this technique, called light-field rendering, is the recognition that all of the rays that pass through a slab of empty space enclosed between two planes can be described using only four parameters rather than the five dimensions required for the typical specifications of a ray. They also describe an efficient technique for generating the ray parameters needed to construct any arbitrary view. The subset of rays originating from a single point on a light field's entrance plane can be considered as an image corresponding to what would have been seen at that point. The entire two-parameter family of images originating from points on the entrance plane can be considered as a set of reference images. During the rendering process, the three-dimensional entrance and exit planes are projected onto the desired viewing plane. The final image is constructed by determining the image-space coordinates of the two points visible at a specified pixel coordinate one coordinate from the projected image of the entrance plane and the second from the image of the exiting plane). The desired ray can be looked up in the light field's database of rays using these four parameter values. Image generation using light fields is inherently a database query process, much like the movie map image-based process. The storage requirements for a lightfield's database of rays can be very large. Levoy and Hanrahan discuss a lossy method for compressing light fields that attempts to minimize some of the redundancy in the light-field representation.

The lumigraph [Gortler96] is another ray-database query algorithm closely related to the light-field. It also uses a four-dimensional parameterization of the rays passing through a pair of planes with fixed orientations. The primary differences in the two algorithms are the acquisition methods used and the final reconstruction process. The lumigraph, unlike the light field, considers the geometry of the underlying models when reconstructing desired views. This geometric information is derived from image segmentations based on the silhouettes of image features. The preparation of the ray database represented in a lumigraph requires considerable preprocessing when compared to the light field. This is a result of the arbitrary

camera poses that are used to construct the database of visible rays. In a lightfield, though, the reference images are acquired by scanning a camera along a plane using a motion platform. The lumigraph reconstruction process involves projecting each of the reference images as they would have appeared when mapped onto the exit plane. The exit plane is then viewed through an aperture on the entrance plane surrounding the center-of-projection of the reference image. When both the image of the aperture on the entrance plane and the reference image on the exit plane are projected as they would be seen from the desired view, the region of the reference image visible through the aperture can be drawn into the desired image. The process is repeated for each reference image. The lumigraph's approach to image-based three-dimensional graphics uses geometric information to control the blending of the image fragments visible through these apertures. Like the lightfield, the lumigraph is a data intensive rendering process.

The image-warping approach to IBR discussed here attempts to reconstruct desired views based on far less information. First, the reference image nearest the desired view is used to compute as much of the desired view as possible. Regions of the desired image that cannot be reconstructed based on the original reference image are subsequently filled in from other reference images. The warping approach to IBR can also be considered as a compression method for both light fields and lumigraphs. Considering the projective constraints induced by small variations in the viewing configuration reduces redundancy of the database representation. Thus, an image point, along with its associated mapping function, can be used to represent rays in many different images from which the same point is visible.

Many other computer graphics methods have been developed where images serve as the underlying representation. These methods handle the geometric relationships between image points very differently. In the case of image morphing, the appearance of a dynamic geometry is often a desired effect. Another method, known as *view interpolation* relies on an approximation to a true projective treatment in order to compute the mapping from reference images to desired images. Also, additional geometric information is required to determine correct visibility. A third method, proposed by Laveau and Faugeras, is based on an entirely projective approach to image synthesis. However, they have chosen to make a far more restrictive set of assumptions in their model, which allows for an ambiguous Euclidean interpretation.

Image morphing is a popular image-based computer graphics technique [Beier92], [Sietz96], [Wolberg90]. Generally, morphing describes a series of images representing a transition between two reference images. These reference images can be considered as endpoints along some path through time and/or space. An interpolation process is used to reconstruct intermediate images along the path's trajectory. Image morphing techniques have been used to approximate dynamic changes in camera pose [Sietz96], dynamic changes in scene geometry [Wolberg90], and combinations of these effects. In addition to reference images, the morphing process requires that some number of points in each reference be associated with corresponding points in the other. This association of points between images is called a *correspondence*. This extra information is usually hand crafted by an animator.

Most image-morphing techniques make the assumption that the transition between these corresponding points occurs at a constant rate along the entire path, thus amounting to a linear approximation. Also, a graduated blending function is often used to combine the reference images after they are mapped from their initial configuration to the desired point on the path. This blending function is usually some linear combination of the two images based on what percentage of the path's length has been traversed. The flexibility of image-morphing methods, combined with the fluidity and realism of the image transitions generated, have made a dramatic impact on the field of computer graphics, especially when considering how recently they have been developed. A subset of image morphing, called view morphing, is a special case of image-based computer graphics. In view morphing the scene geometry is fixed, and the pose of the desired views lies on a locus connecting the centers-of-projection of

the reference images. With the notable exception of the work done by Seitz, general image morphing makes no attempt to constrain the trajectory of this locus, the characteristics of the viewing configurations, or the shapes of the objects represented in the reference images. In this thesis, I will propose image-mapping functions that will allow desired images to be specified from any viewing point, including those off the locus. Furthermore, these mapping functions, like those of Sietz, are subject to constraints that are consistent with prescribed viewing conditions and the static Euclidean shape of the objects represented in the reference images.

Chen and Williams [Chen93] have presented a view interpolation method for three-dimensional computer graphics. It uses several reference images along with image correspondence information to reconstruct desired views. Dense correspondence between the pixels in reference images is established by a geometry-based rendering preprocess. During the reconstruction process, linear interpolation between corresponding points is used to map the reference images to the desired viewpoints, as in image morphing. In general, this interpolation scheme gives a reasonable approximation to an exact reprojection as long as the change in viewing position is slight. Indeed, as the authors point out, in some viewing configurations this interpolation is exact. Chen and Williams acknowledge, and provide a solution for, one of the key problems of image-based rendering—visible surface determination. Chen and Williams presort a quadtree-compressed flow-field in a back-to-front order according to the scene's depth values. This approach works only when all of the partial sample images share a common gaze direction and the synthesized viewpoints are restricted to stay within 90 degrees of this gaze angle. The underlying problem is that correspondence information alone (i.e., without depth values) still allows for many ambiguous visibility solutions unless we restrict ourselves to special flow fields that cannot fold (such as rubber-sheet local spline warps or thin-plate global spline warps).

Establishing the dense correspondence information required for a view interpolation system can also be problematic. Using pre-rendered synthetic images, Chen and Williams were able to determine the association of points by using the depth values stored in a z-buffer. In the absence of a geometric model, they suggest that approximate correspondence information can be established for all points using correlation methods. The image-based approach to three-dimensional computer graphics described in this research has a great deal in common with the view interpolation method. For instance, both methods require dense correspondence information in order to generate the desired image, and both methods define image-space to image-space mapping functions. In the case of view interpolation, the correspondence information is established on a pairwise basis between reference images. As a result the storage requirements for the correspondence data associating N reference images is O(N). My approach is able to decouple the correspondence information from the difference in viewing geometries. This allows a single flow field to be associated with each image, requiring only O(N) storage. Furthermore, the approach to visibility used in my method does not rely on any auxiliary geometric information, such as the presorted image regions based on the z-values, used in view interpolation.

Laveau's and Faugeras' [Laveau94] image-based computer-graphics system takes advantage of many recent results from computer vision. They consider how a particular projective geometric structure called epipolar geometry can be used to constrain the potential reprojections of a reference image. They explain how a fundamental matrix can describe the projective shape of a scene with scalar values defined at each image point. They also provide a two-dimensional ray-tracing-like solution to the visibility problem that does not require an underlying geometric description. Yet, it might require several images to assure an unambiguous visibility solution. Laveau and Faugeras also discuss combining information from several views, though primarily for the purpose of resolving visibility as mentioned before. By relating the reference views and the desired views by the homogenous transformations between their projections, Laveau and Faugeras can compute exact perspective depth solutions. However, the solutions generated using Laveau and Faugeras'

techniques do not reflect an unambiguous Euclidean environment. Their solution is consistent with an entire family of affine coordinate frames. They have pointed out elsewhere [Faugeraus92b] that when additional constraints are applied, such as the addition of more reference images and the requirement that a fixed camera model be used for all references images, then a unique Euclidean solution can be assured.

The methods described by Laveau and Faugeras are very similar to the image-based approach to computer graphics described here. The major difference in my approach is that I assume that more information is available than simply the epipolar geometries between reference images. Other significant differences are that the forward-mapping approach described here has fewer restrictions on the desired viewing position, and I provide a simpler solution to the visibility problem.

The delineation between computer graphics and computer vision has always been at the point of a geometric description. In IBR we tend to draw the lines somewhat differently. Rather than beginning the image synthesis task with a geometric description, we begin with images. In this overview I presented a summary of the various image-based rendering methods frequently used today.

Images have already begun to take on increasingly significant roles in the field of computer graphics. They have been successfully used to enhance the apparent visual complexity of relatively simple geometric screen descriptions. The discussion of previous work showed how the role of images has progressed from approximations, to databases, to actual scene descriptions.

In this course I will present an approach to three-dimensional computer graphics in which the underlying scene representation is composed of a set of reference images. I will present algorithms that describe the mapping of image-space points in these reference images to their image-space coordinates in any desired image. All information concerning the three-dimensional shape of the objects seen in the reference images will be represented implicitly using a scalar value that is defined for each point on the reference image plane. This value can be established using point correspondences between reference images. I will also present an approach for computing this mapping from image-space to image-space with correct visibility, independent of the scene's geometry.

Image-Based Rendering using Image Warping

Leonard McMillan
LCS Computer Graphics Group
Massachusetts Institute of Technology

In this report, I present a complete image-based rendering system. This includes the derivation of a mapping function from first principles, an algorithm for determining the visibility of these mapped points in the resulting image, and a method for reconstructing a continuous image from these mapped points. I refer to this type of mapping function as *image warping*, because it processes the elements of an image according to their image coordinates and produces outputs that are image coordinates in the resulting image.

In addition to the coordinates of the reference image additional information is required for each pixel. This information is related to the distance of the object seen at a particular pixel from the image plane. There are many different measures that can be used to describe this distance. Distance can be specified as *range values* describing the Euclidean distance from the visible object to image's center-of-projection. If the viewing or image plane is known and the coordinate system is chosen so that the normal of this plane lies a unit distance along the z-axis, then this distance information is called *depth* or the pixel's *z-value*. However, there are many other reasonable choices for representing this same distance. For instance distance values can be described indirectly by to the relative motion of image points induced by a change in the camera's position, this distance representation is frequently called *optical flow*, and it is inversely related to the point's range. *Disparity* and *projective-depth* are two more representations of distance for which a warping equation can be developed. The choice of a distance metric often depends on knowing some additional information about how the image was formed (ex. knowledge of the image plane), but in some applications knowledge of this relationship will be unnecessary to perform the desired image warp. Rather than selecting a particular distance measure, and deriving the warping function based on it, I will instead develop a notion of depth that leads to the simplest expression for the desired warping function.

The warping function developed here will not be a one-to-one mapping. In those places where multiple points map to the same result a method to resolve which of the candidate points is visible is required. I will describe a simple method for determining these visible regions. This method will not rely on any geometric information from the scene, but only on the change in pose between the reference and viewing positions.

Finally, since an image will usually be represented as two-dimensional array of discrete samples, reconstruction methods are developed so that the transformed discrete points of the reference image can be used to estimate the appearance of the desired continuous image. I will suggest two methods for this reconstruction.

1. From Images to Rays

A perspective image describes a collection of rays from a given viewing position. Relative to this position any particular ray is uniquely determined by two angles. While the use of two angles sufficiently

describes the full range and dimension of the set of rays, it is not a very convenient representation for analysis or computation. Here I will consider an alternative parameterization of rays based on a general *planar-pinhole camera model*. This is the same planar-pinhole camera model that is commonly used in traditional three-dimensional computer graphics and computer vision.

The planar-pinhole camera is an idealized device for describing the rays that pass through a single point in space, called the *center-of-projection*, and are contained within some solid angle defined by a bounded planar section, called the *image plane*. This solid angle is well defined as long as the center-of-projection does not lie on the extended image plane. As the bounds of the image plane are extended indefinitely, the solid angle approaches 2π steradians, exactly half of the visual sphere.

Consider the rays emanating from the origin of a three-dimensional system with basis vectors $(\hat{i}, \hat{j}, \hat{k})$. Suppose also, that a second two-dimensional coordinate system is defined in the image plane, allowing each point on it to be identified by an image coordinate, (u,v), where *u* and *v* are scalar multiples of the two basis vectors, $(\hat{s}, \hat{t})$, defined in the image plane. Points on this image-plane can be given coordinates that are specified relative to this coordinate system. These points, along with their assigned coordinates, are referred to as *image-space points*. Without loss of generality, we can assume that the origin of image space lies at one of the corners of the bounded image plane. The following figure depicts these coordinate systems:

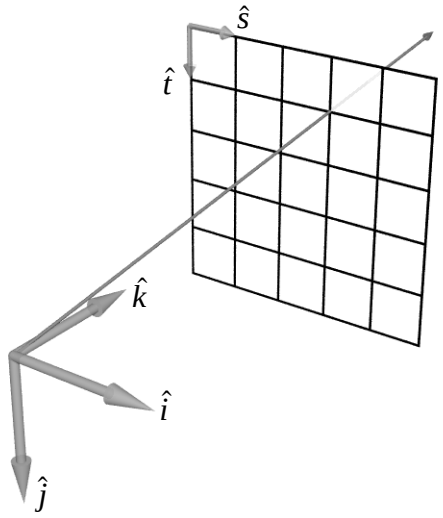


Figure 1: Mapping image-space point to rays

Each image-space point can be placed into one-to-one correspondence with a ray that originates from the Euclidean-space origin. This mapping function from image-space coordinates to rays can be described with a linear system:

$$\bar{d} = \begin{bmatrix} d_i \\ d_j \\ d_k \end{bmatrix} = \begin{bmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_k & b_k & c_k \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Equation 1: Mapping from image coordinates to coordinates in three-space

where the coordinate of the image-space point is specified by the coordinate (u, v), and the resulting vector $\bar{d}$ represents the corresponding ray's direction. The entries of the mapping matrix, $\mathbf{P}$, can be easily understood by considering each column as a vector, $\bar{a}$, $\bar{b}$, and $\bar{c}$ in the same coordinate system as $\bar{d}$. This relationship is shown in Figure 2, where $\bar{a}$ and $\bar{b}$ are images of the $\hat{s}$ and $\hat{t}$ basis vectors in the $(\hat{i}, \hat{j}, \hat{k})$ coordinate system, and $\bar{c}$ is a vector from the ray origin to the origin of the image plane. Thus, while this parameterization is general, it still has a reasonable physical interpretation.

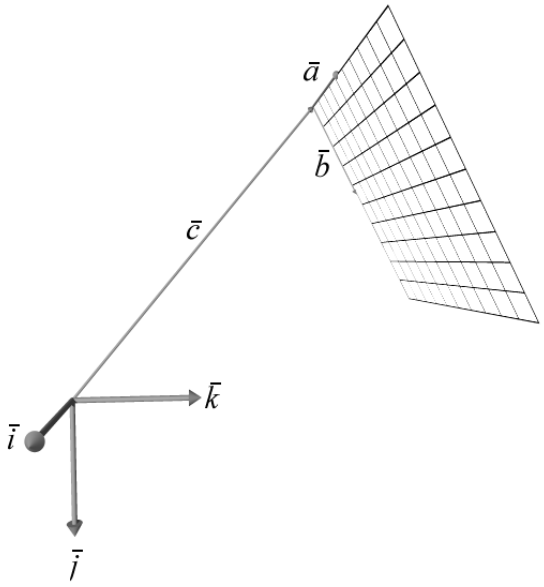


Figure 2: Relationship of the image-space basis vectors to the ray origin

The mapping function from image coordinates to rays is not uniquely defined. Any scalar multiple of the $\mathbf{P}$ matrix will also yield an equivalent set of image-space point-to-ray correspondences. This independence of scale is a consequence of the fact that the space of possible directions from any point in a three-dimensional space can be described using only two parameters (i.e., two angles). Thus, any representation of rays that uses unconstrained three-dimensional vectors will allow for multiple representations of the same ray.

2. A Warping Equation for Synthesizing Projections of a Scene

Equipped with only the simple planar-pinhole-camera model described in the previous section, an image-warping equation, which remaps those rays visible from a given viewpoint to any arbitrary viewing position, can be derived. As before, I will refer to the source image, or domain, of the warp as the *reference image*, and the resulting image, after the mapping is applied, as the *desired image*. For the moment, I will assume that both the pinhole-camera parameters of the desired and reference views are known. In addition, I will also assume that a single scalar value, called *generalized disparity* or *projective depth*, is known for

all points of the reference image. The precise nature of this quantity will be discussed in more detail later in this section. For the moment it is sufficient to say that this quantity can be determined from a set of correspondences specified between images.

Consider the implications of the same point, $\dot{X}$, being sighted along rays from two different centers-of-projection, $\dot{C}_1$ and $\dot{C}_2$, specified relative to their pinhole camera models. The following diagram illustrates this configuration.

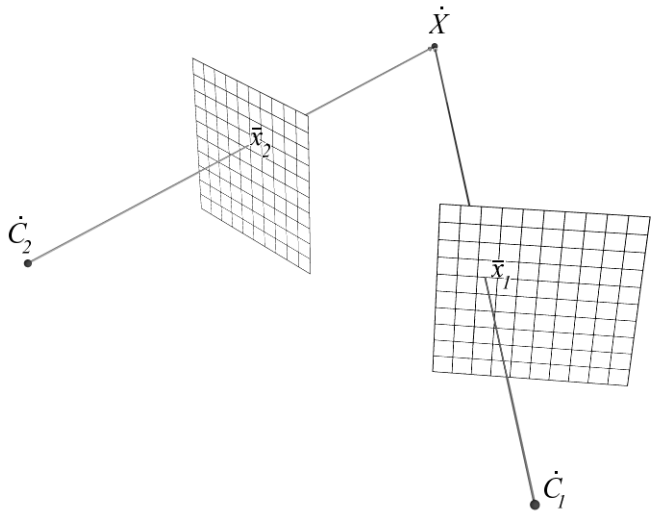


Figure 3: A point in three-dimensional space as seen from two pinhole cameras

The image coordinate, $\bar{x}_1$, in the first image determines a ray via the pinhole camera mapping $\bar{d}_1 = P_1\bar{x}_1$ with an origin of $\dot{C}_1$. Likewise, the image coordinate, $\bar{x}_2$, in the second image determines a ray, $\bar{d}_2 = P_2\bar{x}_2$, with origin $\dot{C}_2$. The coordinate of the point $\dot{X}$ can, therefore, be expressed using either of the following:

$$\dot{X} = \dot{C}_1 + t_1\mathbf{P}_1\bar{x}_1 = \dot{C}_2 + t_2\mathbf{P}_2\bar{x}_2$$

Equation 2: Specification of a 3D point in terms of pinhole-camera parameters

where t_1 and t_2 are the unknown scaling factors for the vector from the origin to the viewing plane which make it coincident with the point $\dot{X}$. This expression can be reorganized to give

$$\tfrac{t_2}{t_1}\mathbf{P}_2\bar{x}_2 = \tfrac{1}{t_1}\left(\dot{C}_1 - \dot{C}_2\right) + \mathbf{P}_1\bar{x}_1$$

Equation 3: Transformation of a ray in one camera to its corresponding ray in another

The left-hand side of this expression is now a ray, as is the second term on the right hand side. If we relax our definition of equivalence to mean “equal down to some non-zero scale factor” (which is consistent with the notion that rays having the same direction are equivalent regardless of the length of the three-space vector specifying this direction), then the t_2/t_1 factor can be eliminated. I will use the symbol,

$\doteq$, to represent this equivalence relationship. Alternatively, we could take advantage of the property that both $\mathbf{P}_1$ and $\mathbf{P}_2$ are defined independent of scale, to absorb the scalar quantity, t_2/t_1, into the matrix $\mathbf{P}_2$. Substituting the *generalized disparity* term $\delta(\bar{x})$ for $\frac{1}{t_1}$ gives

$$\mathbf{P}_2\bar{x}_2 \doteq \delta(\bar{x}_1)\left(\dot{C}_1 - \dot{C}_2\right) + \mathbf{P}_1\bar{x}_1$$

Equation 4: Simplified planar ray-to-ray mapping

The name, *generalized disparity*, comes from the notion of stereo disparity. In the normal depth-from-stereo case, the cameras are assumed to have a particular geometric configuration. Both image planes are required to have the same pinhole-camera model. The vector connecting the centers-of-projection must be parallel to both image planes. And, the coordinate system is selected so that the $\hat{i}$ basis vector of the camera space is parallel to the $\hat{s}$ basis vector of the image planes, as shown below.

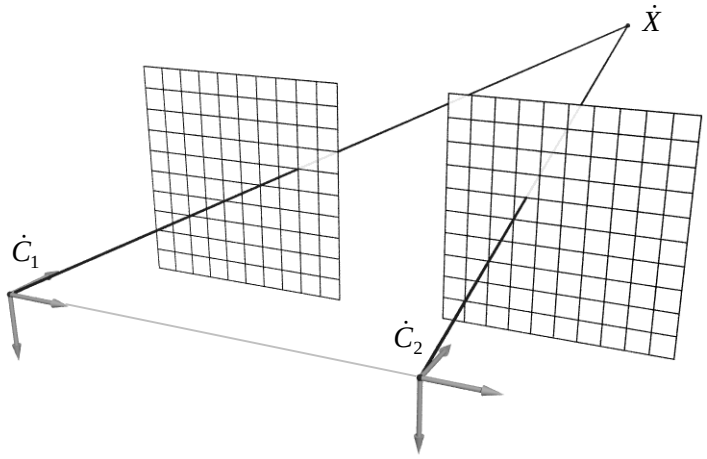


Figure 4: The depth-from-stereo camera configuration

This configuration can be accomplished either by accurate alignments of the cameras or by a post-processing rectification (using re-projection see Equation 13) of the acquired data. Under these conditions the planar ray-to-ray mapping equation simplifies to

$$\mathbf{P}_1\bar{x}_2 \doteq \delta(\bar{x}_1)\left(\begin{bmatrix} C_{1x} \\ C_{1y} \\ C_{1z} \end{bmatrix} - \begin{bmatrix} C_{2x} \\ C_{2y} \\ C_{2z} \end{bmatrix}\right) + \mathbf{P}_1\bar{x}_1 = \delta(\bar{x}_1)\begin{bmatrix} C_{1x} - C_{2x} \\ 0 \\ 0 \end{bmatrix} + \mathbf{P}_1\bar{x}_1$$

Equation 5:

Multiplying both sides by the inverse of the pixel-to-ray mapping gives

$$\bar{x}_2 \doteq \delta(\bar{x}_1)\mathbf{P}_1^{-1}\begin{bmatrix} C_{1x} - C_{2x} \\ 0 \\ 0 \end{bmatrix} + \bar{x}_1$$

Equation 6:

The alignment constraint on the camera-space and image-space basis vectors (that the $\hat{i}$ basis vector of the camera space is parallel to the $\hat{s}$ basis vector) implies that a unit step in image space produces a unit step in camera space. This is equivalent to

$$\mathbf{P}_1 = \begin{bmatrix} 1 & p_{12} & p_{13} \\ 0 & p_{22} & p_{23} \\ 0 & p_{23} & p_{33} \end{bmatrix} \quad \text{so} \quad \mathbf{P}_1^{-1} = \begin{bmatrix} 1 & q_{12} & q_{13} \\ 0 & q_{22} & q_{23} \\ 0 & q_{23} & q_{33} \end{bmatrix}$$

Equation 7:

After multiplying through and reorganizing terms we get the following:

$$\bar{x}_2 - \bar{x}_1 = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} - \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \delta(\bar{x}_1) \begin{bmatrix} C_{1x} - C_{2x} \\ 0 \\ 0 \end{bmatrix}$$

Equation 8:

Thus, when camera geometries satisfy the depth-from-stereo requirement, the image-space coordinates of corresponding points differ only along a single dimension in the images. The size of this change is proportional to both the distance between the centers-of-projection, which is often called the stereo baseline, and the generalized disparity quantity, which has units of pixels per unit length. Therefore, generalized disparity, $\delta(\bar{x})$, is equivalent to stereo disparity, $u_2 - u_1$, when normalized by dividing through by the length of the baseline.

$$\delta_{stereo}(\bar{x}) = \frac{u_2 - u_1}{C_{1x} - C_{2x}}$$

Equation 9:

The term, *projective depth*, is sometimes used instead of generalized disparity. This name is somewhat misleading since it increases in value for points closer to the center-of-projection. Generalized disparity, $\delta(\bar{x}_1)$, is inversely related to depth by a constant scale factor, and to range by a scale factor that varies from point to point on the viewing plane.

Let us consider further the generalized-disparity term, $\delta(\bar{x}_1)$. We can construct the following diagram of Equation 4 in the plane containing the three points $\dot{C}_1$, $\dot{C}_2$, and a visible point $\dot{X}$.

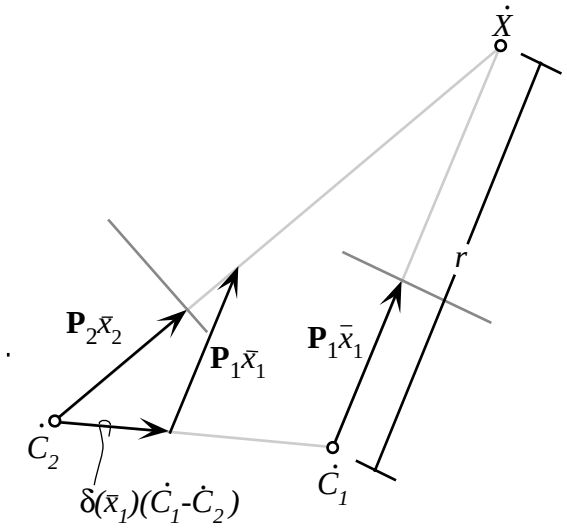


Figure 5: A point as seen from two images

Using similar triangles it can be shown that

$$\frac{r}{\left|\dot{C}_1 - \dot{C}_2\right|} = \frac{\left|\mathbf{P}_1\bar{x}_1\right|}{\delta(\bar{x}_1)\left|\dot{C}_1 - \dot{C}_2\right|}$$

Solving for generalized disparity gives the following expression:

$$\delta(\bar{x}_1) = \frac{\left|\mathbf{P}_1\bar{x}_1\right|}{r}$$

Thus, the generalized disparity depends only on the distance from the center-of-projection to the position on the image plane where the point is observed and the range value of the actual point. The image-space coordinate in the second image of the actual point can be found using the following transformation:

$$\bar{x}_2 \doteq \delta(\bar{x}_1)\mathbf{P}_2^{-1}\left(\dot{C}_1 - \dot{C}_2\right) + \mathbf{P}_2^{-1}\mathbf{P}_1\bar{x}_1$$

Equation 10: Planar image-warping equation

Since the generalized-disparity term, $\delta(\bar{x}_1)$, is independent of both the desired center-of-projection, $\dot{C}_2$, and the desired pinhole viewing parameters, $\mathbf{P}_2$, Equation 10 can also be used to determine the image-space coordinate of the observed point on any other viewing plane. This is accomplished by simply substituting the desired center-of-projection and pinhole-camera model into Equation 10.

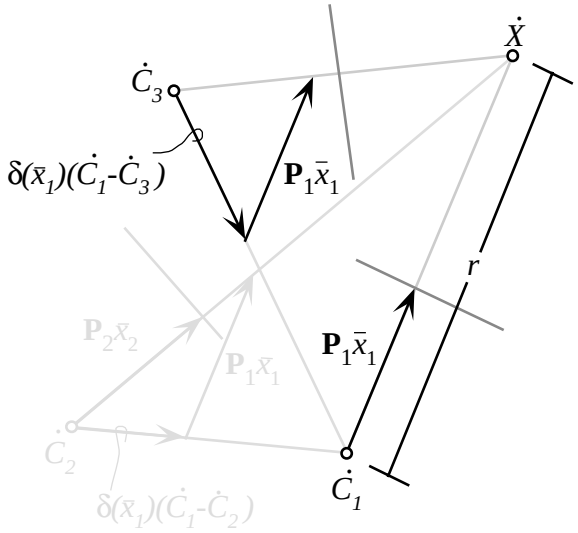


Figure 6: A third view of the point $\dot{\mathbf{X}}$

Figure 6 illustrates how the planar warping equation (Equation 10) can be used to synthesize arbitrary views. The third viewing position, $\dot{C}_3$, need not be in the same plane as $\dot{C}_1$, $\dot{C}_2$ and $\dot{X}$. Figure 6 also depicts how generalized disparity is an invariant fraction of the baseline vector. This fraction of the baseline vector applies to all potential views, and therefore, it can be used to align corresponding rays of a given reference image to their direction in any desired image. In addition, the resulting projection will be consistent to a fixed three-dimensional point. Generalized disparity is a scalar quantity that determines how a translation of the center-of-projection affects the coordinates of points in image space. The remaining matrix quantity, $\mathbf{P}_2^{-1}\mathbf{P}_1$, determines how changes in the pinhole-camera model, independent of translation, affect the coordinates of points in image space. A further explanation of this last claim will be presented in the next section.

The warping equation can be used to synthesize arbitrary views of a given reference image via the following procedure. Given a reference image, the matrix describing its planar-pinhole projection model, $\mathbf{P}_1$, its center-of-projection, $\dot{C}_1$, a generalized-disparity value, $\delta(\bar{x}_1)$, for each pixel, and the center-of-projection of the desired image, $\dot{C}_2$, and its projection model, the mapping of the reference image to a desired image can be computed. Using the vectors described in the generalized pinhole-camera model, the warping equation can be rewritten as

$$\alpha\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{a}_2 & \bar{b}_2 & \bar{c}_2 \end{bmatrix}^{-1}\left(\dot{C}_1 - \dot{C}_2\right)\delta(u_1, v_1) + \begin{bmatrix} \bar{a}_2 & \bar{b}_2 & \bar{c}_2 \end{bmatrix}^{-1}\begin{bmatrix} \bar{a}_1 & \bar{b}_1 & \bar{c}_1 \end{bmatrix}\begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

where α is an arbitrary scale factor. This matrix sum can be rewritten as shown below:

$$\alpha\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{a}_2 & \bar{b}_2 & \bar{c}_2 \end{bmatrix}^{-1}\begin{bmatrix} \bar{a}_1 & \bar{b}_1 & \bar{c}_1 & \left(\dot{C}_1 - \dot{C}_2\right) \end{bmatrix}\begin{bmatrix} u_1 \\ v_1 \\ 1 \\ \delta(u_1, v_1) \end{bmatrix}$$

Multiplying both sides by the determinant of $\mathbf{P}_2$ and substituting for its inverse gives the following 4×3 matrix equation:

$$\alpha\left(\bar{c}_2 \cdot \left(\bar{a}_2 \times \bar{b}_2\right)\right)\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{a}_1 \cdot \left(\bar{b}_2 \times \bar{c}_2\right) & \bar{b}_1 \cdot \left(\bar{b}_2 \times \bar{c}_2\right) & \bar{c}_1 \cdot \left(\bar{b}_2 \times \bar{c}_2\right) & \left(\dot{C}_1 - \dot{C}_2\right) \cdot \left(\bar{b}_2 \times \bar{c}_2\right) \\ \bar{a}_1 \cdot \left(\bar{c}_2 \times \bar{a}_2\right) & \bar{b}_1 \cdot \left(\bar{c}_2 \times \bar{a}_2\right) & \bar{c}_1 \cdot \left(\bar{c}_2 \times \bar{a}_2\right) & \left(\dot{C}_1 - \dot{C}_2\right) \cdot \left(\bar{c}_2 \times \bar{a}_2\right) \\ \bar{a}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right) & \bar{b}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right) & \bar{c}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right) & \left(\dot{C}_1 - \dot{C}_2\right) \cdot \left(\bar{a}_2 \times \bar{b}_2\right) \end{bmatrix}\begin{bmatrix} u_1 \\ v_1 \\ 1 \\ \delta(u_1, v_1) \end{bmatrix}$$

Equation 11: 4×3 matrix formulation of warping equation

This results in the following rational expressions for computing the reprojection of pixel coordinates from a reference image to the coordinates of a desired image:

$$u_2 = \frac{\bar{a}_1 \cdot \left(\bar{b}_2 \times \bar{c}_2\right)u_1 + \bar{b}_1 \cdot \left(\bar{b}_2 \times \bar{c}_2\right)v_1 + \bar{c}_1 \cdot \left(\bar{b}_2 \times \bar{c}_2\right) + \left(\dot{C}_1 - \dot{C}_2\right) \cdot \left(\bar{b}_2 \times \bar{c}_2\right)\delta(u_1, v_1)}{\bar{a}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right)u_1 + \bar{b}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right)v_1 + \bar{c}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right) + \left(\dot{C}_1 - \dot{C}_2\right) \cdot \left(\bar{a}_2 \times \bar{b}_2\right)\delta(u_1, v_1)}$$

$$v_2 = \frac{\bar{a}_1 \cdot \left(\bar{c}_2 \times \bar{a}_2\right)u_1 + \bar{b}_1 \cdot \left(\bar{c}_2 \times \bar{a}_2\right)v_1 + \bar{c}_1 \cdot \left(\bar{c}_2 \times \bar{a}_2\right) + \left(\dot{C}_1 - \dot{C}_2\right) \cdot \left(\bar{c}_2 \times \bar{a}_2\right)\delta(u_1, v_1)}{\bar{a}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right)u_1 + \bar{b}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right)v_1 + \bar{c}_1 \cdot \left(\bar{a}_2 \times \bar{b}_2\right) + \left(\dot{C}_1 - \dot{C}_2\right) \cdot \left(\bar{a}_2 \times \bar{b}_2\right)\delta(u_1, v_1)}$$

Since the centers-of-projection and the planar-pinhole camera models for the reference and desired images are fixed for a given mapping, the warping equation simplifies to a pair of constant-coefficient linear rational expressions of the form

$$r(u_1, v_1, \delta(u_1, v_1)) = w_{11}u_1 + w_{12}v_1 + w_{13} + w_{14}\delta(u_1, v_1)$$

$$s(u_1, v_1, \delta(u_1, v_1)) = w_{21}u_1 + w_{21}v_1 + w_{23} + w_{24}\delta(u_1, v_1)$$

$$t(u_1, v_1, \delta(u_1, v_1)) = w_{31}u_1 + w_{32}v_1 + w_{33} + w_{34}\delta(u_1, v_1)$$

$$u_2 = \frac{r(u_1, v_1, \delta(u_1, v_1))}{t(u_1, v_1, \delta(u_1, v_1))}$$

$$v_2 = \frac{s(u_1, v_1, \delta(u_1, v_1))}{t(u_1, v_1, \delta(u_1, v_1))}$$

Equation 12: Warping equation as rational expressions

The mapping of a point in the reference image to its corresponding point in the desired image can be computed using nine adds, eleven multiplies, and one inverse calculation. When points of the reference image are processed sequentially, the number of adds is reduced to six, and the number of multiplies is reduced to five. Additional tests for a positive denominator, $t(u_1, v_1, \delta(u_1, v_1))$, and a valid range of the numerator can avoid two multiplies and the inverse calculation. This operation is the equivalent of screen-space clipping in traditional three-dimensional computer graphics.

3. Relation to Previous Results

Many other researchers [Szeliski96] [Faugeras92] have described similar warping equations. In most of these applications the image-space coordinates of the points $\bar{x}_1$ and $\bar{x}_2$ were given, and the projective depth, $\delta(\bar{x})$, was the quantity to be solved for. When this equation is used for image warping, coordinates of image points in the desired view, $\bar{x}_2$, are computed from points in the reference image, $\bar{x}_1$, and their projective depths.

This warping equation is also closely related to several other well-known results from computer graphics, image-warping, and projective geometry. Consider the situation where the reference image and the desired view share a common center-of-projection. In this case the planar-warping equation simplifies to

$$\bar{x}_2 \doteq \mathbf{P}_2^{-1}\mathbf{P}_1\bar{x}_1$$

Equation 13: Image reprojection

This illustrates the well known result that images defined on planar viewing surfaces sharing a common center-of-projection are related by a *projective transformation* or *planar homography*.

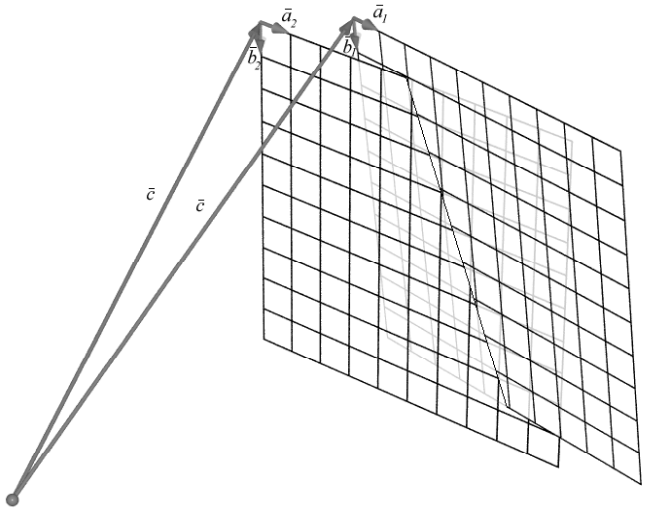


Figure 7: Reprojection of an image with the same center-of-projection

This projective transformation is merely the composition of the reference image's viewing matrix with the inverse of the desired image's viewing matrix, $\mathbf{H}_{reproject} = \mathbf{P}_2^{-1}\mathbf{P}_1$. This is indicative of the fact that, ignoring the clipping that occurs at the boundaries of the view plane, a change of viewing surface does not change the set of rays visible from the center-of-projection. It only changes their spatial distribution on the viewing surface. I will refer to mappings of this sort as reprojections.

A second well known result from the fields of computer graphics and projective geometry is that all images of a common planar surface seen in planar projection are also related by a projective transform as long as the plane does not project to a line. This result is the underlying basis for the texture mapping of images onto planar surfaces. It allows for the rasterization of textured planar primitives in screen space using a rational linear expression (an alternate formulation for a projective transform). The figure below illustrates the projection of a planar region onto several viewing planes and the resulting image.

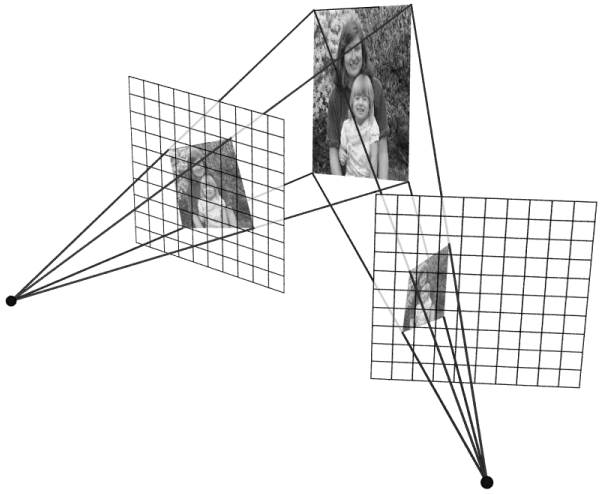


Figure 8: A planar region seen from multiple viewpoints

The mapping function that describes the possible views of a three-dimensional planar surface can be derived as a special case of Equation 10 by the following progression. The equation of a plane in the coordinate system having the reference image's center-of-projection as its origin is given by

$$\begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = D$$

Equation 14: Equation of a plane

where the scalars *A*, *B*, *C*, and *D* are four parameters defining the plane, and x, y, and z are the coordinates of a point in space. These three-space coordinates can be re-expressed in terms of image coordinates by using a planar-pinhole model image-to-ray mapping function as follows:

$$\begin{bmatrix} A & B & C \end{bmatrix} t(u,v) \mathbf{P}_1 \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = D$$

where $t(u,v)$ is a multiple of the distance from the center-of-projection to the viewing plane for the ray. Dividing both sides by the scalar quantities appearing on opposite sides gives

$$\delta(u,v) = \frac{1}{t(u,v)} = \begin{bmatrix} \frac{A}{D} & \frac{B}{D} & \frac{C}{D} \end{bmatrix} \mathbf{P}_1 \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

The inverse of $t(u,v)$ is equivalent to the generalized-disparity term discussed in Equation 3. When the generalized-disparity value from above is substituted into the warping equation (Equation 10), the following expression results:

$$\bar{x}_2 \doteq \mathbf{P}_2^{-1}\left(\dot{C}_1 - \dot{C}_2\right)\begin{bmatrix} \frac{A}{D} & \frac{B}{D} & \frac{C}{D} \end{bmatrix} \mathbf{P}_1 \bar{x}_1 + \mathbf{P}_2^{-1} \mathbf{P}_1 \bar{x}_1 = \mathbf{P}_2^{-1}\left(\left(\dot{C}_1 - \dot{C}_2\right)\begin{bmatrix} \frac{A}{D} & \frac{B}{D} & \frac{C}{D} \end{bmatrix} + I\right) \mathbf{P}_1 \bar{x}_1$$

Equation 15: Mapping of a common plane seen at two centers-of-projection

The planar homography, $\mathbf{H}_{plane} = \mathbf{P}_2^{-1}\left(\left(C_1 - C_2\right)\begin{bmatrix} \frac{A}{D} & \frac{B}{D} & \frac{C}{D} \end{bmatrix} + I\right)\mathbf{P}_1$, is a projective mapping of the reference-image points on the plane to their coordinates in the desired image. When the reference and desired images share a common center-of-projection, the projective mapping reduces to the reprojection given in Equation 13 as expected.

The image plane of a reference image is but a plane in three-dimensional space. Therefore, its image in any reprojection is related by a projective mapping. The equation of the three-dimensional image plane of a given pinhole-camera model is given by

$$\left(\bar{a}_1 \times \bar{b}_1\right)^T \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \bar{c}_1 \cdot \left(\bar{a}_1 \times \bar{b}_1\right)$$

Substitution into Equation 15, gives

$$\bar{x}_2 \doteq \mathbf{P}_2^{-1}\left(\left(\dot{C}_1 - \dot{C}_2\right)\frac{1}{\bar{c}_1 \cdot \left(\bar{a}_1 \times \bar{b}_1\right)}\left(\bar{a}_1 \times \bar{b}_1\right)^T + I\right)\mathbf{P}_1 \bar{x}_1 = \mathbf{P}_2^{-1}\left(\left(\dot{C}_1 - \dot{C}_2\right)\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} + \mathbf{P}_1\right)\bar{x}_1$$

which simplifies to

$$\bar{x}_2 \doteq \left(\begin{bmatrix} \bar{0} & \bar{0} & \mathbf{P}_2^{-1}\left(\dot{C}_1 - \dot{C}_2\right) \end{bmatrix} + \mathbf{P}_2^{-1}\mathbf{P}_1\right)\bar{x}_1$$

Equation 16: Projection of the reference image plane in the desired image

Notice that the projective mapping, $H_{viewplane} = \lfloor 0 \quad 0 \quad \mathbf{P}_2^{-1}(C_1 - C_2) \rfloor + \mathbf{P}_2^{-1}\mathbf{P}_1$, describes the projection of the reference image's viewing plane in the desired image.

4. Resolving visibility

The mapping function described by the planar image warping equation (Equation 10) is not one-to-one. The locus of potential points in three-space, $\dot{X}(t)$, that project to the same image coordinate, $\bar{x} = (u, v)$, is described by the center-of-projection and a planar pinhole-camera model using the following equation:

$$\dot{X}(t) = \dot{C} + t\mathbf{P}\bar{x}$$

Equation 17:

The parameter *t* identifies specific three-dimensional points that project to a given image-space coordinate. A policy of selecting the smallest of all positive *t* values for a given screen-space point can be used to determine visibility for opaque objects. This candidate *t* value is analogous to the z values stored in a z-buffer [Catmull74], or the ray-length maintained by a ray-casting algorithm [Appel68]. While the *t* parameter is an essential element of the reprojection process, (via its relationship to $\delta(\bar{x}_1)$) it is, surprisingly, not required to establish the visibility of a warped image.

In this section, an algorithm is presented for computing the visible surface at each image-space point of a desired image as it is being derived from a reference image via a warping equation. This is accomplished by establishing a warping order that guarantees a correct visibility solution. This ordering is established independently of the image contents. Only the centers-of-projection of the desired and reference images, as well as the pinhole-camera model for the reference image are needed.

In order to simplify the following discussion, the reference image is assumed to be stored as a two-dimensional array whose entries represent uniformly spaced image samples. This simplification is not strictly necessary for the algorithm to operate correctly, but it allows for a concise statement of the algorithm, and it is representative of many typical applications.

The approach of this visibility algorithm is to specify an ordering, or enumeration, of points from the reference image which guarantees that any scene element that is hidden by some other scene element in the desired image will always be drawn prior to its eventual occluder. This type of ordering is commonly called back-to-front. It is well known that a simple painter's algorithm [Rogers85] can be used to display any collection of scene elements with correct visibility when a back-to-front ordering can be established.

Given the reference image's center-of-projection, $\dot{C}_1$, and ray-mapping function, $\mathbf{P}_1$, and the desired center-of-projection, $\dot{C}_2$, the projection of $\dot{C}_2$ onto the reference image is first computed as follows:

$$\begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} = \mathbf{P}_1^{-1}\left(\dot{C}_2 - \dot{C}_1\right)$$

Equation 18:

An example of this projection is illustrated in Figure 9.

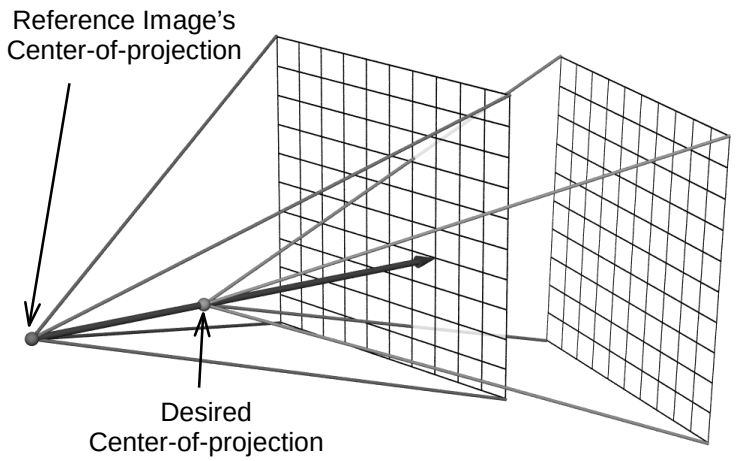


Figure 9: A desired center-of-projection projected onto the reference image

The image coordinate of $\dot{C}_2$ on the reference image is given by $\bar{e} = (e_x/e_z, e_y/e_z)$. It will fall into one of nine regions relative to the reference image shown below:

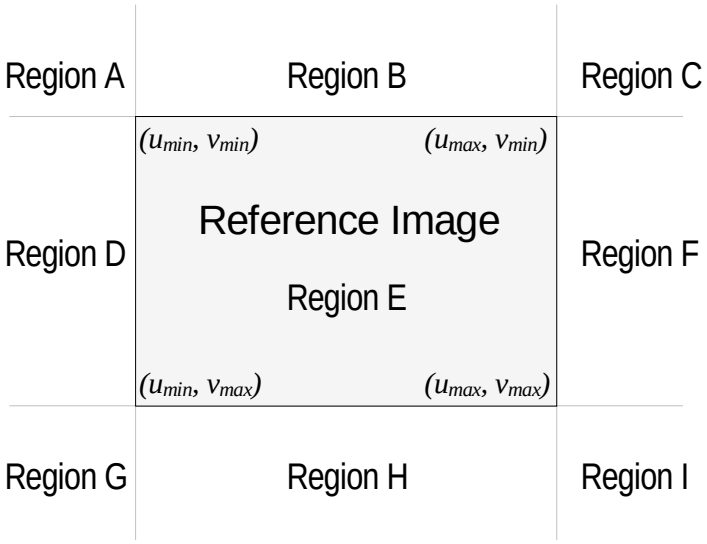


Figure 10: Figure of nine regions

Next, the reference image is subdivided into sheets that can be processed independently. Any set of orthogonal image-space basis vectors can be used to partition each sheet, but it is simplest to choose a coordinate basis aligned with the image's sampling grid. The reference image is partitioned into 1, 2, or 4 sheets depending on the image-space coordinates of the projected point, $\overline{e}$. When $\overline{e}$ projects within the domain of the reference image, the image is divided into four sections separated along the row and column of $\overline{e}$.

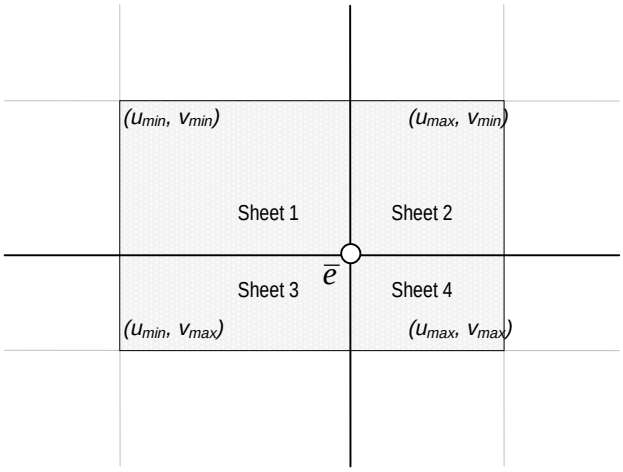


Figure 11: A desired center-of-projection that divides the reference image into 4 sheets

When only one coordinate of $\overline{e}$ falls within the reference image, (i.e., when $\overline{e}$ falls into regions B, D, F, and H) the image is subdivided into two sheets whose boundary is determined by either the row or column of the one coordinate that lies within the domain.

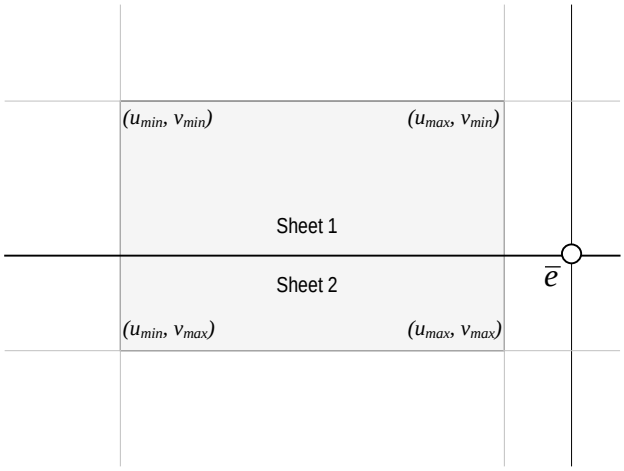


Figure 12: A desired center-of-projection that divides the reference image into 2 sheets

If neither component of $\bar{e}$ falls within the reference image's domain, (when $\bar{e}$ projects into regions A, C, H or J) then the entire image is treated as a single sheet.

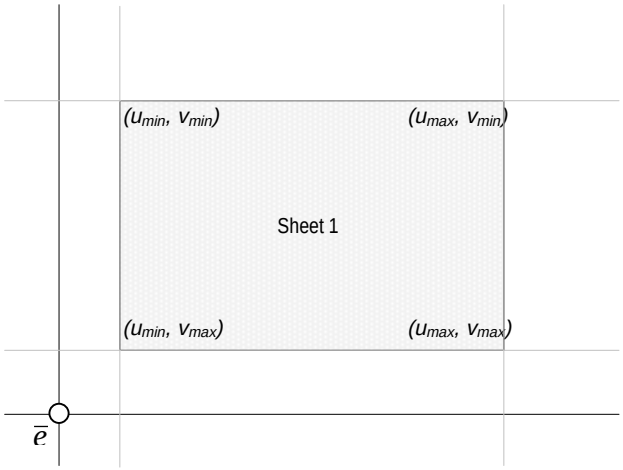


Figure 13: A desired center-of-projection that divides the reference image into 1 sheet

Once the reference image's domain is subdivided according to the projected position of the desired center-of-projection, the warping order for each sheet can be determined as follows. The sign of the e_z component from Equation 18 determines the enumeration direction. When e_z is non-negative the warping order of each sheet progresses toward the point $\bar{e}$, otherwise the warping progresses away from $\bar{e}$, as shown in the figure below. The case where e_z is zero indicates that the desired viewing position has no proper projection onto the reference image, because the vector $\dot{C}_1 - \dot{C}_2$ is parallel to the reference image's viewing plane. In this case, only one sheet will be enumerated, and the warping order progresses in the direction determined by the quadrant indicated by the signs of the remaining two vector components, e_x and e_y.

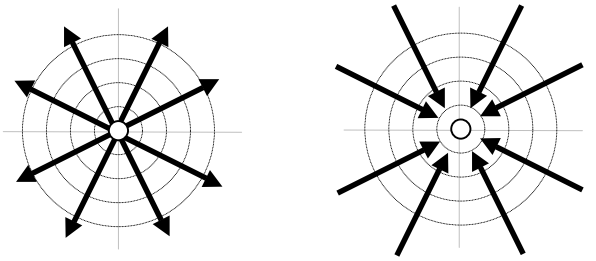


Figure 14: Enumeration direction

During the warp, each radial line originating from the projected image of the desired center-of-projection can be traversed independently. Alternatively, the warp can progress along either rows or columns of the sheets so long as the image of the desired center-of-projection, $\overline{e}$, is drawn at the appropriate time (i.e., p is drawn first when e_z is negative, and last otherwise), allowing either a row major or column major traversal. The advantage of the latter approach is that it allows the reference image's traversal to take maximum advantage of the access coherence of most memory systems.

The entire visibility algorithm involves three simple steps. First, the three-dimensional coordinate of the desired center-of-projection, $\dot{C}_2$, is projected on the reference image's viewing plane. Second, the image-plane is divided into sheets determined by the image-space coordinate of the projected center-of-projection, $\overline{e}$, and whose boundaries are aligned with the image-space coordinate axes[1]. Computing this coordinate involves a projective normalization. Finally, the sign of the planar normalizing element of the projective coordinate determines the traversal of the reference image[2].

The algorithm presented here is similar to Anderson's algorithm for bivariate functions [Anderson82]. The difference is that his visibility algorithm was defined for a different class of surfaces (i.e., a height field, or Monge patch, rather than a projective surface), and his algorithm enumerates the facets in a front-to-back occlusion order. Anderson's choice of a front-to-back order requires that some representation of the grid perimeter be maintained to aid in deciding what parts or edges of subsequent facets need to be rendered. Representing this perimeter requires auxiliary storage and extra clipping computations. This list must then be queried before each new facet's edge is displayed, and the display must be updated if any part of the facet is visible. In contrast, a back-to-front ordering requires no additional storage because the proper occlusion is handled by the drawing order using the painter's algorithm.

---

[1] Along the rows and columns of a discretely sampled image array

[2] This is often called the homogeneous element of the projective coordinate.

5. Reconstruction Issues

The planar-warping equation describes a mapping of the image-space points on a reference viewing plane to image-space points on a desired viewing plane. The underlying assumption is that both the reference and desired images are continuous over their domains. This is not generally the case for typical images. Usually, images are represented by a two-dimensional array of discrete samples. There are many subtle implications of warping sampled images rather than continuous ones. While the warping equation can easily be applied to the discrete coordinates of a sampled image, the likelihood that any sample will map exactly onto a sampling-grid point of the desired image is negligible. In addition to warping to locations off the sampling grid, the points of a reference image will also distribute unevenly over the desired image. The desired image must then be synthesized from this irregular distribution of sampled and reprojected image points.

The process of mapping a sampled image from one image-space to another is called *image resampling*. Conceptually, image resampling constructs a continuous representation of a reference image which is then mapped onto the desired viewing plane using the warping function and resampled to form the final discrete image. When the three-dimensional points represented in the reference image lie on a common plane, as shown in Figure 8, and the scene contents are appropriately band limited, reconstructing a continuous representation is a straightforward application of signal processing theory [Wolberg90]. The same situation occurs when three-dimensional points are constrained to lie along the same rays of different viewing planes, as when reprojecting an arbitrary set of points from the same center-of-projection as shown in Figure 7. The ideal continuous reconstruction of these surfaces is the summation of two-dimensional sinc functions centered at each sample-grid point and scaled by the sample's intensity. Since the sinc function has an infinite extent, local polynomial approximations are often used instead [Mitchell88]. However, when the three-dimensional points visible in an image are not constrained to lie in a plane or share a center-of-projection, the reconstruction of a continuous reference image representation is more involved.

Three-dimensional surfaces can be built up from discrete sub-surfaces, called *surface patches*. The composite of a group of surface patches can represent the reconstruction of a three-dimensional point set. One measure of the continuity of a composite set of surface patches, called *derivative continuity*, is the number of matching terms in the Taylor series expansions at abutting surface patches. When only the first terms of the composite surfaces' Taylor series expansions match along their common boundaries, the surface patches will have the same three-dimensional coordinate values along their abutting regions, and the overall surface has C^0 continuity. When both the first and second terms of the expansion agree, the tangent spaces of the abutting surfaces coincide, and the composite surface will have C^1 continuity.

One approach to reconstructing a continuous function from a sampled reference image is to consider each sample as specifying a surface patch. The basis for these surface patch definitions is typically polynomial. However, this is not a requirement for defining a continuous reconstruction. All that is

necessary is that the basis functions are sufficiently differentiable. For instance, a sinc or a Gaussian basis are both valid representations of surface patches[3].

I will next describe two different methods for constructing a continuous representation of a reference image for use in image warping based on C^0 continuity models that use different surface patch bases. Recent work by Mark [Mark97] has extended these methods to include a model in which C^0 and C^1 continuity alternates throughout the domain. His approach requires a local estimate of the tangent space at each sample point. This requirement can be easily satisfied when the reference images are synthesized using traditional computer graphics techniques, but it is more difficult for acquired reference images. However, there is some promise in this area. It appears that many shape-from-shading [Horn89] and photometric-ratio-based correspondence methods [Wolff94] can be adapted to estimate a surface normal at each image-space point.

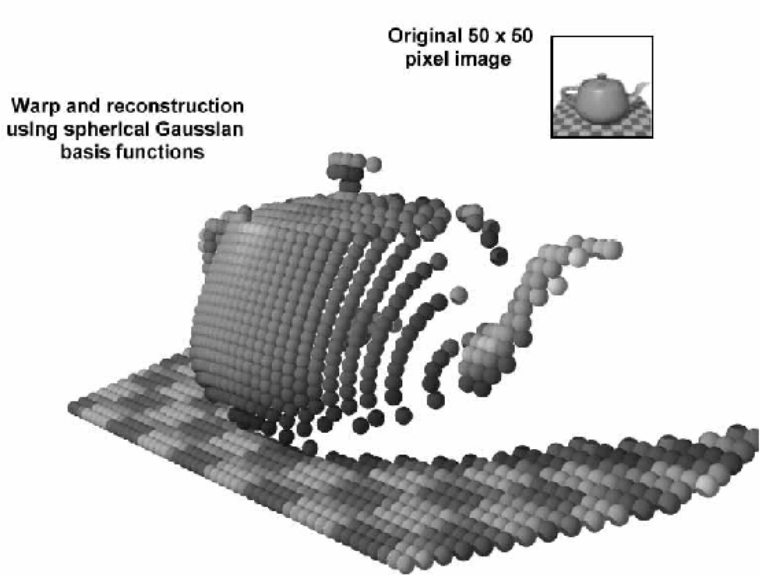


Figure 15: A Gaussian cloud representation of image-space points

The first reconstruction approach uses a spherical Gaussian basis function to represent each sample point. It assumes that every visible point in a reference image represents a three-dimensional spherical cloud density located somewhere along the ray determined by the image point. In three-dimensions this radius could be determined by considering the solid angle represented at each sample point and the distance of the cloud's center from the image's center-of-projection. A two-dimensional equivalent of this radius calculation can, however, be computed directly from the warping equation.

The image-space Gaussian reconstruction method described here is a straightforward adaptation of Heckbert's Elliptical Weighted Average (EWA) filter [Heckbert89], but it is used in a forward-mapping algorithm, similar in spirit to the Splatting algorithm described by Westover [Westover91]. The support of

[3] The use of a sinc or Gaussian basis is somewhat complicated by their infinite extents.

the Gaussian reconstruction function is determined by computing the change in shape of differential circular regions surrounding each sample as they undergo the image warp. One useful measure of the change in shape is provided by the *Jacobian determinant* of the mapping function. The Jacobian determinant is a direct measure of the local change in area induced by a transformation. However, changes in differential area are not necessarily a good indication of the changes in differential shape[4]. The additional assumptions required to address this discrepancy will be discussed shortly. The Jacobian matrix of the planar-warping function given in Equation 12 is

$$\mathbf{J} = \begin{bmatrix} \frac{w_{11}(w_{32}v+w_{33}+w_{34}\delta(u,v))-w_{31}(w_{12}v+w_{13}+w_{14}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} & \frac{w_{12}(w_{31}u+w_{33}+w_{34}\delta(u,v))-w_{32}(w_{11}u+w_{13}+w_{14}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} \\ \frac{w_{21}(w_{32}v+w_{33}+w_{34}\delta(u,v))-w_{31}(w_{22}v+w_{23}+w_{24}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} & \frac{w_{22}(w_{31}u+w_{33}+w_{34}\delta(u,v))-w_{32}(w_{21}u+w_{23}+w_{24}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} \end{bmatrix}$$

Equation 19: Jacobian of the warping equation

The determinant of the Jacobian matrix simplifies to

$$\frac{\partial(u',v')}{\partial(u,v)} = \frac{\det(\mathbf{H}) + \delta(u,v)\det(\mathbf{G})}{t(u,v,\delta(u,v))^3}$$

Equation 20: Jacobian determinant of the warping equation

where

$$\mathbf{H} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \text{ and } \mathbf{G} = \begin{bmatrix} w_{11} & w_{12} & w_{14} \\ w_{21} & w_{22} & w_{24} \\ w_{31} & w_{32} & w_{34} \end{bmatrix}$$

Equation 21: Camera-model matrix, H, and the structure matrix, G

The $\mathbf{H}$ component of the Jacobian determinant represents the change in projected area of an infinitesimal region of the reference image due entirely to the changes in the pinhole-camera model since $\mathbf{H} \doteq \mathbf{P}_2^{-1}\mathbf{P}_1$ and $\det(\text{Jacobian}(\mathbf{H}\bar{x})) = \det(\mathbf{H})/t(u,v,0)^3$. The $\mathbf{G}$ component represents the change in projected area due to the three-dimensional structure of the observed image point. This can be seen by letting $\mathbf{H} = \mathbf{I}$, which indicates no change in the pinhole camera model between the reference and desired image. This gives $\frac{\partial(u',v')}{\partial(u,v)} = \frac{1}{(1+\delta(u,v)w_{34})^2}$, indicating that the change in projected area is independent of the point's coordinate on the image plane, but instead it depends entirely on the generalized disparity value at that

---

[4] For instance, consider the mapping $\phi: \{x = 10u, y = 0.1v\}$ with $\frac{\partial(x,y)}{\partial(u,v)} = \text{Determinant}\begin{bmatrix} 10 & 0 \\ 0 & 0.1 \end{bmatrix} = 1$. The Jacobian determinant indicates that the differential area surrounding any sample remains constant; yet the mapping introduces considerable stretching along the u dimension and shrinking in the v dimension.

point. Since the matrices **H** and **G** are constant for a given pair of reference and desired views, they need only be calculated once per warp.

If we make the assumption that the warping function is well represented by a local linear approximation centered at each sample in the reference image with a constant generalized disparity value, then the Jacobian matrix can also be used to estimate the change in shape of a rotationally symmetric reconstruction kernel as follows. An infinitesimal circular region with a radius of dr is described by the expression

$$dr^2 = \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix}$$

Equation 22:

When that region undergoes an arbitrary mapping, $\phi: (u, v) \rightarrow (u', v')$, the best linear approximation to the differential mapping is given by the Jacobian matrix

$$\begin{bmatrix} du' \\ dv' \end{bmatrix} = \mathbf{J} \begin{bmatrix} du \\ dv \end{bmatrix}$$

Equation 23:

By substituting Equation 23 into Equation 22, the mapping of the differential circular region can be determined as follows:

$$dr^2 = \begin{bmatrix} du' & dv' \end{bmatrix} (\mathbf{J}^{-1})^T \mathbf{J}^{-1} \begin{bmatrix} du' \\ dv' \end{bmatrix}$$

Equation 24:

which expands to the following conic expression:

$$dr^2 = \frac{(j_{21}^2 + j_{22}^2)du'^2 - 2(j_{11}j_{21} + j_{12}j_{22})du'dv' + (j_{11}^2 + j_{12}^2)dv'^2}{(j_{11}j_{22} - j_{12}j_{21})^2}$$

Equation 25:

This equation describes an ellipse if the Jacobian determinant is positive. Therefore, any circular region centered about a reference image sample will project as an ellipse in the desired image. This property can be used to reconstruct a continuous representation of the reference image prior to resampling. The spatial domain response of any rotationally symmetric filter, such as a Gaussian, can be computed at a sample point in the desired image by evaluating Equation 25 at that point to determine the radius value in the undistorted filter's kernel. The resampling process can be optimized by first computing the extents of the ellipse in the desired image space. In terms of the Jacobian these extents are given by the following expressions:

$$\Delta u' = \pm \sqrt{\frac{dr^2 (j_{11} j_{22} - j_{12} j_{21})^2}{(j_{21}^2 + j_{22}^2) - \frac{(j_{11} j_{21} + j_{21} j_{22})^2}{(j_{11}^2 + j_{12}^2)}}} \qquad \Delta v' = \pm \sqrt{\frac{dr^2 (j_{11} j_{22} - j_{12} j_{21})^2}{(j_{11}^2 + j_{12}^2) - \frac{(j_{11} j_{21} + j_{21} j_{22})^2}{(j_{21}^2 + j_{22}^2)}}}$$

Equation 26:

Another approach to reconstructing a continuous representation of the reference image attempts to fit a bilinear surface patch between any four neighboring grid samples. This representation also has C^0 continuity, but it uses a polynomial basis. First, the individual sample points of the reference image are mapped to the desire image's coordinate system. These warped points will generally not correspond to sample grid positions. The connectivity of the eight-connected neighborhood of each reference sample-point is maintained after the warp. This can be managed by maintaining a buffer of two scan lines while enumerating the reference image in a visibility compatible order. The warped sample points stored in these two buffers can be considered a single strip of patches at the completion of each scan line. A standard polygonal rasterizer can be used to scan convert each patch in the strip. Therefore, this technique can easily take advantage of specialized rasterization hardware if it is available. Once a strip is rasterized, one of the scanline buffers becomes available for storing the mapped values of the next scan line.

Shown below is an example of the same warp using each of the two reconstruction methods discussed.

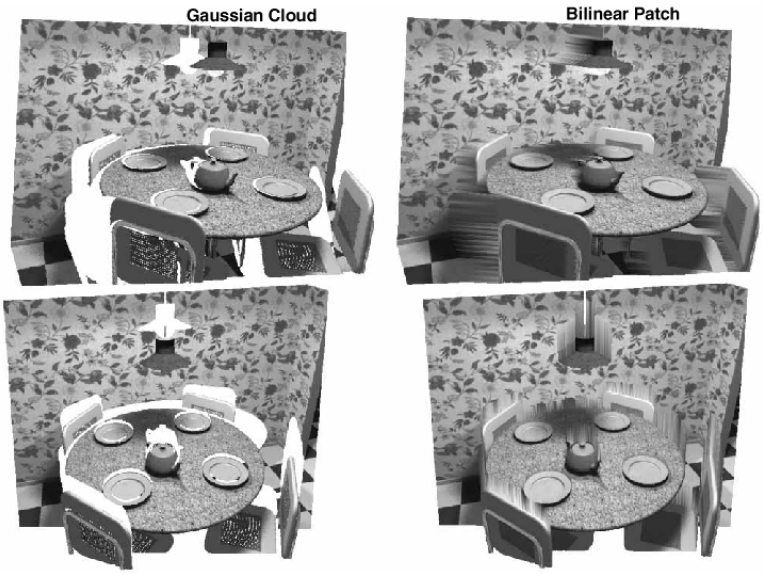


Figure 16: Example image warp using different reconstruction methods

6. Occlusion and Exposure Errors

The image warp can only correctly reproject those scene points visible in the reference image. In some cases, even the visibility of a point does not guarantee its proper reconstruction. These are not

weaknesses of either the image-warping or visibility methods described; they are, instead, inherent limitations of an image-based representation.

The major visual artifacts resulting from these limitations can be classified as one of two cases, *exposure errors* or *occlusion errors*. Exposure errors occur when a background region that should have been occluded is visible in a desired image because of the absence of some foreground element from the reference image. On the other hand, occlusion errors occur in a desired image when an interpolation error in the reconstruction process introduces a false foreground element that covers background regions visible in the actual scene. The choice of reconstruction methods plays a significant role in either amplifying or reducing these errors. However, a reduction in one class of artifact often causes an increase in the other.

Many exposures and occlusions are correct. For instance, when a viewpoint moves toward a foreground object the projection of the object will enlarge in the field-of-view such that it covers adjacent background points. As the viewpoint moves even closer to the foreground object, more of the background is occluded.

Exposure errors and occlusion errors take place either when the underlying assumptions of the reconstruction method are violated, or when there is insufficient information in the image to properly reconstruct the correct surface. These two error sources are closely related. The role of reconstruction kernel is to interpolate the missing gaps between samples. Often the information needed to correctly fill a missing image region is unavailable from the reference image.

Exposure errors are the subtler visual artifact. The region uncovered by a legitimate exposure lends itself to interpretation as a shadow produced by a light source placed at the reference image's center-of-projection. This is particularly noticeable when the exposure occurs along object boundaries. An exposure error occurs when a ray in the desired image passes through this shadow region, allowing some background element to be erroneously seen. Both exposure errors and actual exposures are illustrated below.

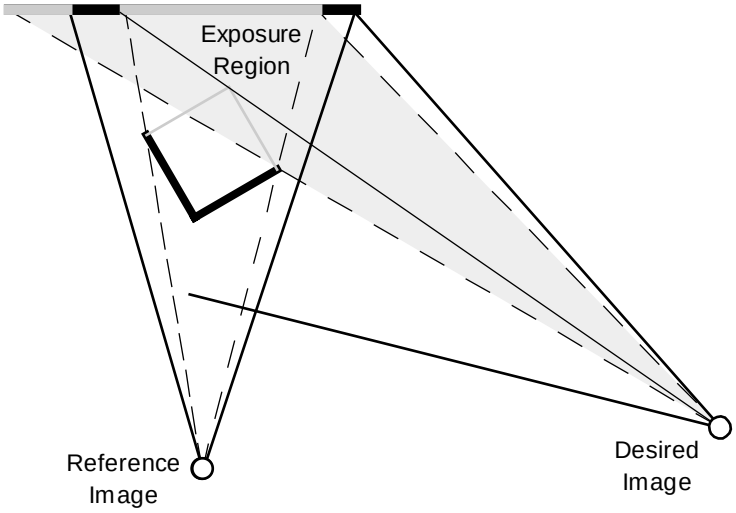


Figure 17: Exposures at occluding boundaries

The actual scene points that are visible from the reference image are shown darkened. The shaded region indicates where an exposure occurs in the desired image. The solid dividing line through the exposure region indicates the boundary between an actual exposure to the right and an exposure error to the left. However, without external information the difference between valid and invalid exposures cannot be resolved.

Exposure errors are most likely to occur at object silhouettes. They occur on smooth surfaces as well as along sharp depth discontinuities. This situation is depicted in Figure 18. Exposure errors occur immediately adjacent to those image points whose ray lies in the observed object's tangent plane. Therefore, as an observer moves to see around an object boundary, she should generally expect to see more of the object rather than any component of the background.

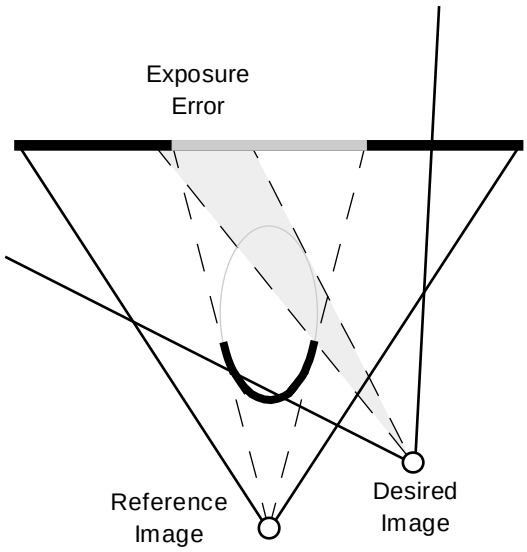


Figure 18: Exposure error on a smooth surface boundary

Merely changing the basis function used in the reconstruction of the reference image can eliminate exposure errors, but it introduces occlusion errors. Consider the example shown in Figure 19 when a polynomial basis, instead of the Gaussian cloud model, is used to approximate the underlying surface.

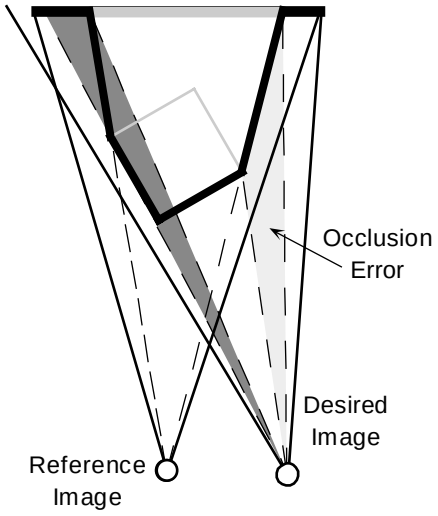


Figure 19: Occlusion errors introduced by polynomial reconstruction

The lighter shaded area indicates the extent of an occlusion error, whereas the darker shaded area represents an actual occlusion. The surface seen along the occlusion error corresponds to an extension of the foreground object's tangent plane. This surface will always enclose the actual surface. However, it is unlikely that any part of this extension will actually represent a point in the environment. The occlusion

error will usually hide valid exposures. Furthermore, since occlusion errors are not adjacent to an actual occlusion, they appear more unnatural than exposure errors.

The continuity of the polynomial interpolation basis causes an excessive estimation of the number of points in the scene. The polynomial interpolation model reconstructs images that are indistinguishable from a model that assumes that all of the points beyond each ray's observed point are occupied. Such a model will not miss any actual scene points, but it will erroneously include scene points that do not exist. The polynomial reconstruction method will, therefore, introduce a disproportionate number of occlusion errors. This can greatly hinder the usefulness of such a model.

In contrast, the Gaussian reconstruction method represents a vacuous estimate of a scene's contents. It assumes that the visible image point is the only scene point located along the ray's extent. Therefore, a Gaussian reconstruction basis will correctly represent all empty regions of an environment, while missing all scene points that are not visible in the reference image. It is, therefore, conservative in its estimate of the occupied volume of space, whereas the polynomial reconstruction makes a conservative estimate of the space's emptiness. Exposure errors should be expected when the Gaussian reconstruction model is used.

The visibility algorithm generally handles valid occlusions. One exception is the case when a scene point from outside the viewing frustum comes into view as a result of the change in the center-of-projection. This situation results in an *invisible occluder error*. A simple example of this case is shown in Figure 20. This problem is a direct result of the limited field-of-view available to a planar-pinhole camera. The use of panoramic pinhole-camera models can remedy this problem.

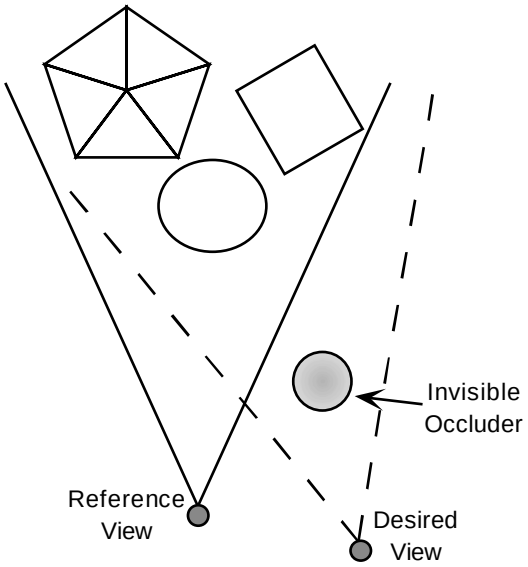


Figure 20: An external exposure error

The two reconstruction methods discussed previously represent two extreme assumptions concerning the structure of space. The use of either of these extreme positions introduces artifacts in the

final rendering. A more accurate reconstruction should combine elements of both methods. However, this might require additional information beyond that which is deducible from the reference image alone. This is an interesting area for future research.

7. Summary

This report has presented a complete example of an image-based method for synthesizing computer graphics. Special mapping functions, called image warps, were derived that enabled arbitrary views of a scene to be generated. This underlying scene was represented by a reference image that was augmented by a scalar value defined at each point, called generalized disparity. An algorithm was also presented to resolve the visibility of the mapped reference points at each coordinate in the desired image. Two methods were presented for reconstructing continuous representations of the warped reference image from these warped points.

8. References

[Anderson82] Anderson, D., *"Hidden Line Elimination in Projected Grid Surfaces,"* **ACM Transactions on Graphics**, October 1982.

[Appel67] Appel, A., *"The notion of quantitative invisibility and the machine rendering of solids,"* **Proceedings of the ACM National Conference**, ACM, New York, October, 1967, pp. 387-393.

[Catmull74] Catmull, E., *"A Subdivision Algorithm for Computer Display of Curved Surfaces,"* Ph.D. Thesis, Department of Computer Science, University of Utah, Tech. Report UTEC-CSc-74-133, December 1974.

[Chen93] Chen, S. E. and L. Williams. *"View Interpolation for Image Synthesis,"* **Computer Graphics** (SIGGRAPH'93 Conference Proceedings), July 1993, pp. 279-288.

[Faugeras92b] Faugeras, O. D., *"What can be seen in three dimensions with an uncalibrated stereo rig?,"* **Proceedings of the 2nd European Conference on Computer Vision**, Springer-Verlag, 1992, pp. 563-578.

[Heckbert89] Heckbert, P. S., *"Fundamentals of Texture Mapping and Image Warping,"* Masters Thesis, Department of EECS, UCB, Technical Report No. UCB/CSD 89/516, June 1989.

[Horn89] Horn, B.K.P., *"Obtaining Shape from Shading Information,"* **Shape From Shading**, Chapter 6, Edited by Berthold K. Horn and Michael J. Brooks, The MIT Press, Cambridge, Massachusetts, 1989.

[Mark97] Mark, W. R., L. McMillan, and G. Bishop**, Proceedings of 1997 Symposium on Interactive 3D Graphics** (Providence, Rhode Island, April 27-30, 1997).

[Mitchell88] Mitchell, D.P. and A.N. Netravali, *"Reconstruction Filters in Computer Graphics,"* **Computer Graphics** (SIGGRAPH'88 Conference Proceedings), Vol. 22, No. 4, August 1988, pp. 221-228.

[Rogers85] Rogers, D.F., **Procedural Elements for Computer Graphics**, McGraw-Hill, New York, NY, 1985.

[Szeliski96] Szeliski, R., "*Video Mosaics for Virtual Environments,*" **IEEE Computer Graphics and Applications**, March 1996, pp. 22-30.

[Westover90] Westover, L. A., "*Footprint Evaluation for Volume Rendering,*" **Computer Graphics** (SIGGRAPH'90 Conference Proceedings), Vol. 24, No. 4, August 1990, pp. 367-376.

[Wolberg90] Wolberg, G., **Digital Image Warping**, IEEE Computer Society Press, Los Alamitos, California, 1990.

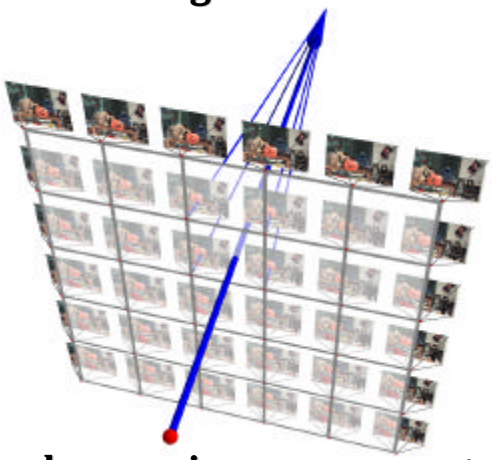
[Wolff94] Wolff, L. B., and E. Angelopoulou, *"3-D Stereo Using Photometric Ratios,"* **Proceedings of the European Conference on Computer Vision**, Springer-Verlag, 1994, pp. 247-257.

Image-Based Rendering: With or Without Structure?

Leonard McMillan
LCS Computer Graphics Group
Massachusetts Institute of Technology

Goals of IBR

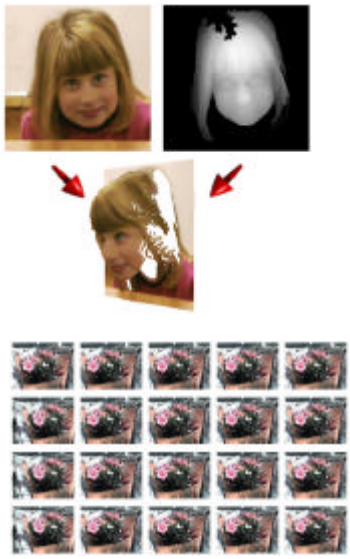
- To infer new images from old ones



- Appealing because images are easy to acquire

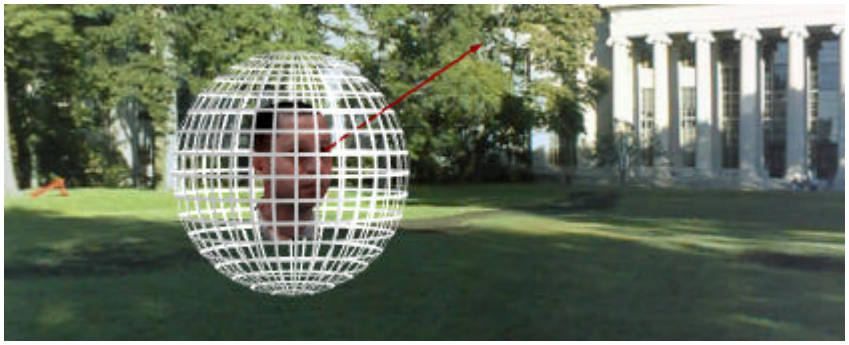
Two Schools of Thought

- **Images with structure**
  - Prerendering analysis
  - Small runtime footprint
  - Active image sensors
  - Backward compatible
- **Images without structure**
  - Can use raw images
  - Minimal analysis and assumptions
  - Large storage requirements
  - Incompatible / Radical



IBR Principles

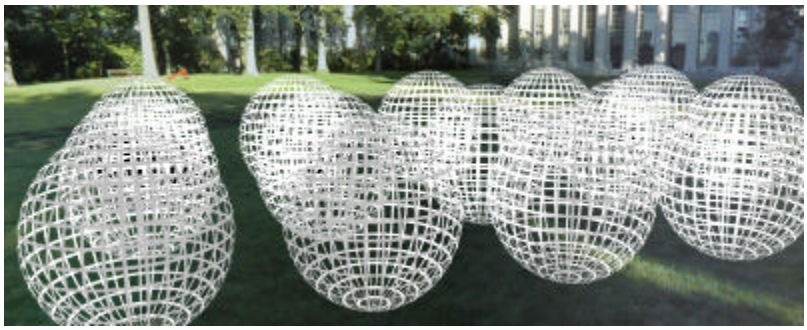
- **Consider images as a collection of rays,**



… rather than a collection of pixels.

The Plenoptic Function

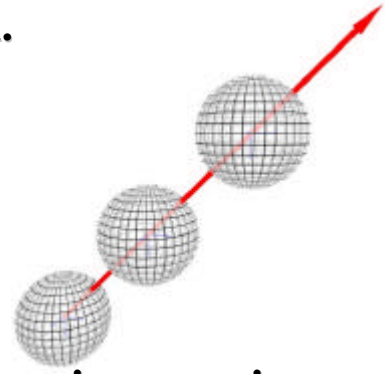
- Given enough sample rays, can we interpolate nearby ones?



IBR is a different approach to computer graphics.

Image-based rendering is about

… signal reconstruction rather than physical simulation.



5-D for static scenes in general

4-D in empty regions of static scenes

6-D for dynamic scenes

Where to Begin?

- **Next Generation Cameras**
  - Known Internal calibration
  - Know where they are
  - Photometric
  - High-dynamic range

Image Courtesy of MIT City Scanning Project, Seth Teller, Satyan Coorg , JP Mellor, George Chou, Doug De Couto, Neel Master, Barb Cutler, Eric Amram, Mike Bosse, Matt Antone, Stefano Totaro , and Manish Jethwa.

Warping Points with Depth

- **Advantages**
  - Simple warping transformation

$$\vec{\mathbf{x}}_2 = \frac{1}{\mathbf{z}_1}\vec{\mathbf{e}}_2 + \mathbf{H}_{21}\vec{\mathbf{x}}_1$$

  - Small footprint in memory
  - Occlusion compatible rendering order
  - Compatible with traditional graphics methods
- **Disadvantages**
  - View-independent shading
  - Reconstruction errors
  - Disocclusion

Acquiring Depth Images

- **Economical laser scanners**

Images and video courtesy of the University of North Carolina at Chapel Hill, "Office of the Future" Project, Wei-Chao Chen, Henry Fuchs, Lars Nyland, Herman Towles and Greg Welch.

Warping Prospects

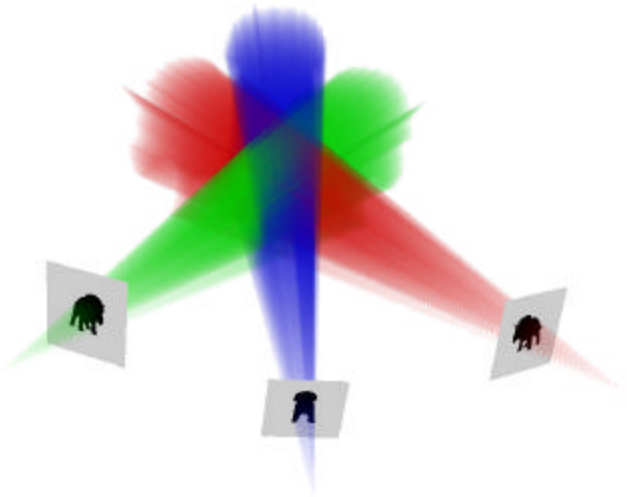
- **Modeling**
  - Active sensing
  - Passive sensing
  - Dynamic scenes
- **Rendering**
  - View-dependent shading
  - Disocclusion (LDIs, MCOPs)
  - Reconstruction
- **Hardware Acceleration**

Approximate Geometry

- **Many images with an approximate model**
  - Façade
  - View-dependent texture mapping
  - Image-based visual hulls
- **Approx shape for inter-object interactions**
  - Occlusions
  - Walk-around
- **Textures for detail**
  - View-dependent shading
  - Small geometric features

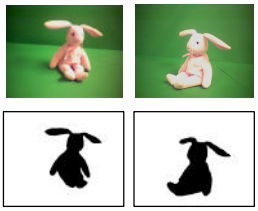
Visual Hulls

- **Intersection of all silhouette cones**

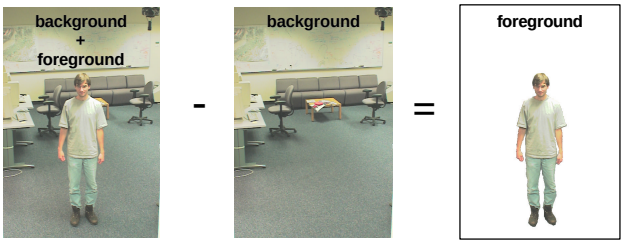


Why Use a Visual Hull?

- **Finding silhouettes is simple and robust**
  - Blue-screen methods
  - Image differenceing

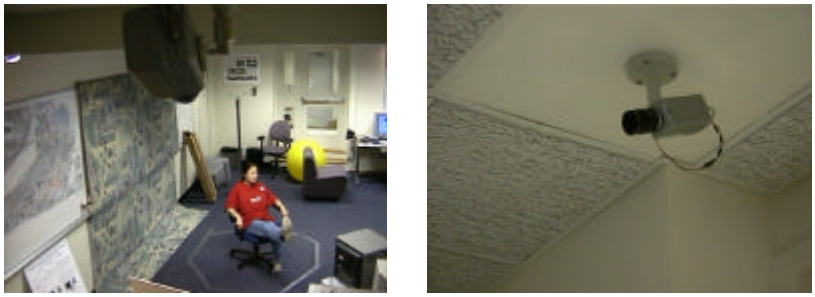


- **Contains actual object**
  - Can seed more sophisticated methods

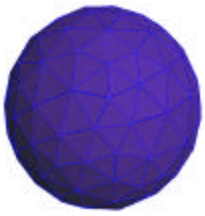


Acquisition

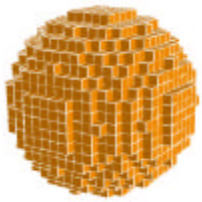
- **Several cameras with overlapping views**
  - Geometric calibration
  - Photometric calibration
  - Synchronization



Representing a Visual Hull



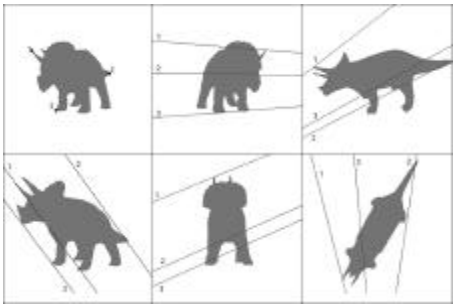
- **Geometric Model**
  - + Know how to render
  - • Calculating intersections isn't robust
  - • High polygonal complexity

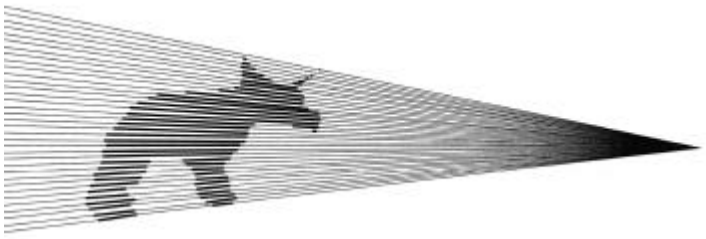


- **Volumetric Model**
  - + Easy to compute
  - • Expensive storage
  - • Result seldom predicts inputs

Image-based Visual Hulls

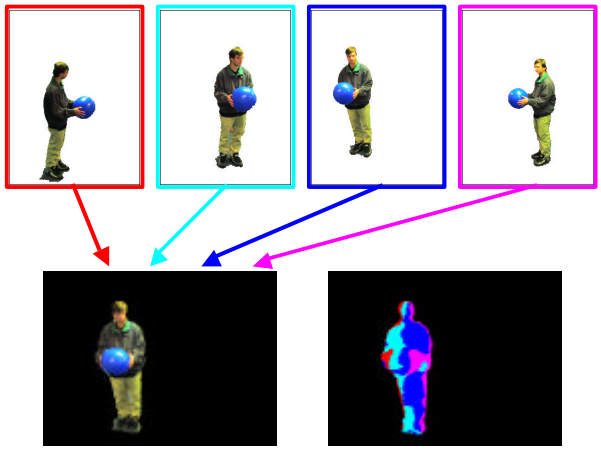
- **Volume-like**
- **Self-consistent**
- **Discrete-discrete-continuous**



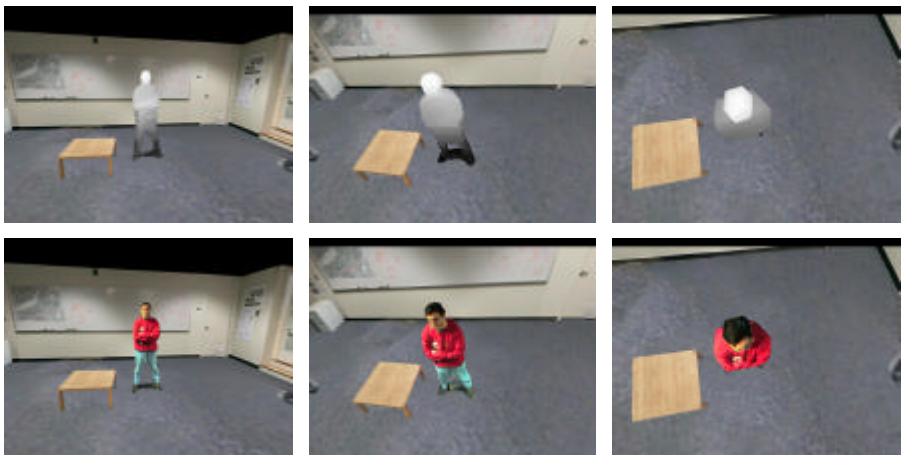


Shading Visual Hulls

- View-dependent illumination
- Visibility



Results



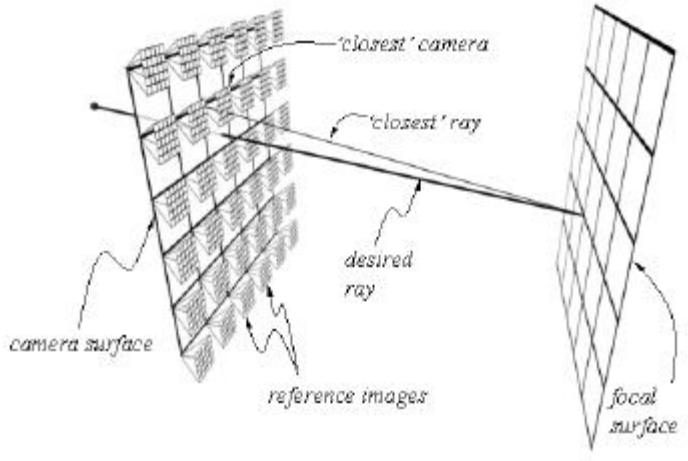
A range of different virtual viewpoints of a visual hull computed from four cameras in real-time. Top images show depth maps and bottom images show shaded visual hulls. The background is a textured polygonal model.

Visual Hull Prospects

- **Capture**
  - Passive - multiple synchronized calibrated cameras
  - Limited working volume
  - Approximate model (even in the limit)
- **Rendering**
  - Fast - can be computed at a constant cost / pixel
  - Limited view-dependent shading
  - Texture registration on approximate geometry
- **Acceleration**
  - Scalable to a large number of cameras

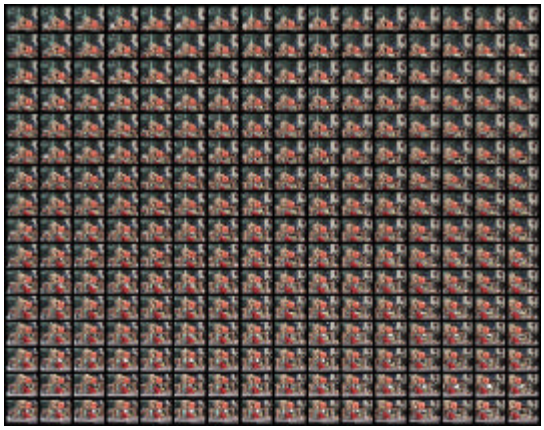
Structured Light fields

- **Focal planes as approximate structure**

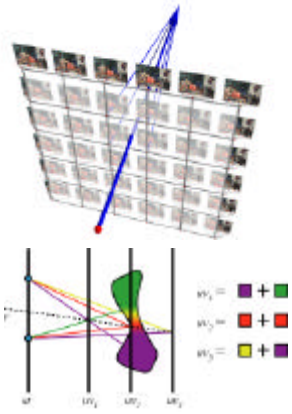


Interpolating Rays

- Less structure / More images
- Structure "on-the-fly"



Dynamic Reparameterization



Light Field Acquisition

- Motion Platforms
  - Precise positioning
  - Calibrated digital camera
  - Expensive (> $10K)
  - Very Slow (~20 mins)
- Light field cameras
  - Less precise
  - Calibration per aperture
  - Inexpensive (~ $100)
  - Slow (> 3 mins)

Focal Planes as Structure

- Structure is discovered by user interaction rather than recovered by computer vision
- Intuitive “camera-like” interface
- Reconstruction (Interpolation) controlled by variable focus and variable aperture

Focal Planes as UI

- Apparent object-centered navigation
- Segmenting into layers

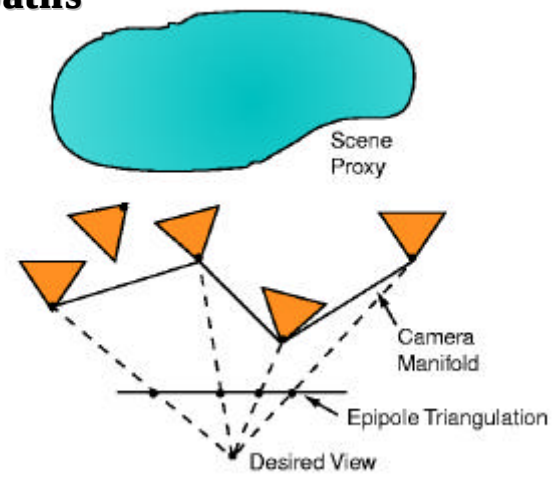
The apparent rotation in the scene above is about an axis through the pumpkin. However, the pumpkin’s position is unknown. The origin of the rotation in this case was determined by the placement of the focal plane.

Prospects of DRLF

- **Modeling**
  - Many images / cameras (expensive setup)
  - Minimal assumptions about the scene
  - Compression = Structure?
- **Rendering**
  - Supports view-dependent shading
  - Very fast - large memory requirements
  - Less structure - more aliasing
- **Hardware Acceleration**
  - Can use existing texture-mapping H / W

Unstructured Light Fields

- **Reference images along unconstrained camera paths**



Acquiring Unstructured LFs

- **Can use a wide range of source images**
  - Images from regular arrays or panoramas
  - Tracked or calibrated cameras
  - Hand-held camcorder
- **Position/Pose recovery**
  - Post-process source images by tracking features
  - Photometric equalization

Rendering Unstructured LFs

- **View-dependent parameterization via dynamic triangulation**
- **H/W texture mapping**

Results

- **Static scenes**

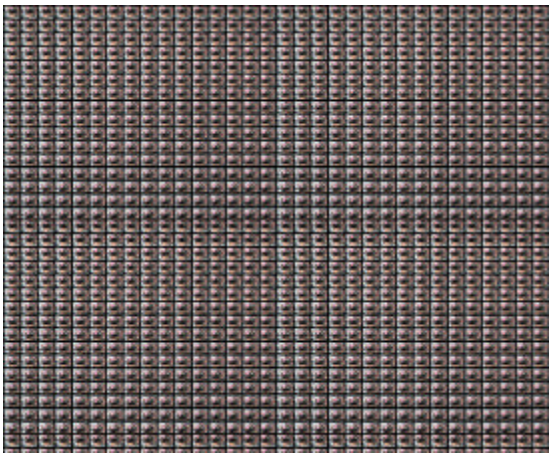
- **Dynamic scenes**

Unstructured LF Prospects

- **Modeling**
  - Wide range of source materials
  - Camera tracking is essential
- **Rendering**
  - Supports view-dependent shading
  - Very very fast - large memory requirements
  - Many artifacts to overcome (scattered data reconstruction)
- **Hardware Acceleration**
  - Can use existing texture-mapping H / W

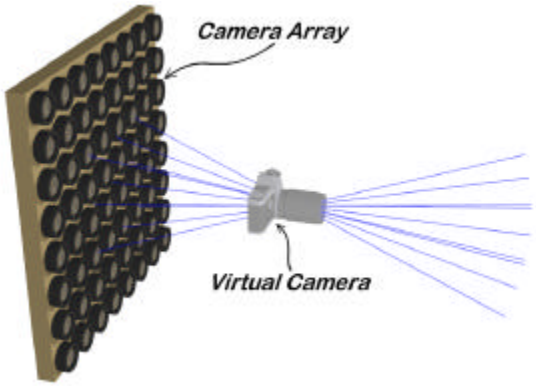
Pure Light Fields

- With enough images there is no need for focal planes and variable apertures aside from special effects



LF Acquisition

- Needs 10000s or more images in a small region
- Precision Motion platforms
- Tightly packed camera arrays

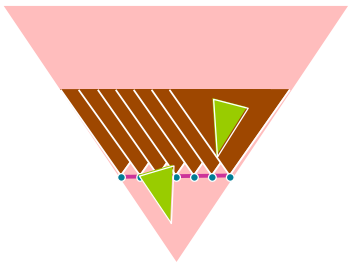


LF Rendering

- **Advantages**
  - Simple / Fast rendering algorithm
  - View-independent shading
  - High-quality reconstructions
- **Disadvantages**
  - Huge memory footprint (compression)
  - Incompatible with traditional graphics

LF Prospects

- **Modeling**
  - Images only, no analysis
  - Capture devices
  - Dynamic scenes
  - Immersive scenes



- **Rendering**
  - Moving around occluders
  - Stitching together multiple light fields
  - Compression
- **Hardware Acceleration**

Open Questions

- How much structure do we really need?
- How do we get it?
- How about adaptive systems?
- Compression versus model extraction?
- Structured versus unstructured capture?
- Storage versus analysis?
- Generality versus effectiveness?

Proceedings of SIGGRAPH 95 (Los Angeles, California, August 6-11, 1995)

Plenoptic Modeling: An Image-Based Rendering System

Leonard McMillan[†] and Gary Bishop[‡]

Department of Computer Science
University of North Carolina at Chapel Hill

ABSTRACT

Image-based rendering is a powerful new approach for generating real-time photorealistic computer graphics. It can provide convincing animations without an explicit geometric representation. We use the "plenoptic function" of Adelson and Bergen to provide a concise problem statement for image-based rendering paradigms, such as morphing and view interpolation. The plenoptic function is a parameterized function for describing everything that is visible from a given point in space. We present an image-based rendering system based on sampling, reconstructing, and resampling the plenoptic function. In addition, we introduce a novel visible surface algorithm and a geometric invariant for cylindrical projections that is equivalent to the epipolar constraint defined for planar projections.

CR Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation– *display algorithms*, *viewing algorithms*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism– *hidden line/surface removal*; I.4.3 [**Image Processing**]: Enhancement– *registration*; I.4.7 [**Image Processing**]: Feature Measurement– *projections*; I.4.8 [**Image Processing**]: Scene Analysis.

1. INTRODUCTION

In recent years there has been increased interest, within the computer graphics community, in image-based rendering systems. These systems are fundamentally different from traditional geometry-based rendering systems. In image-based systems the underlying data representation (i.e model) is composed of a set of photometric observations, whereas geometry-based systems use either mathematical descriptions of the boundary regions separating scene elements (B-rep) or discretely sampled space functions (volumetric).

The evolution of image-based rendering systems can be traced through at least three different research fields. In photogrammetry the initial problems of camera calibration, two-dimensional image registration, and photometrics have progressed toward the determination of three-dimensional models. Likewise, in computer vision, problems such as robot navigation, image discrimination, and image understanding have naturally led in the same direction. In computer graphics, the progression toward image-based rendering systems was initially motivated by the desire to increase the visual realism of the approximate geometric descriptions by mapping images onto their surface (texture mapping) [7], [12]. Next, images were used to approximate global illumination effects (environment mapping) [5], and, most recently, we have seen systems where the images themselves constitute the significant aspects of the scene's description [8].

Another reason for considering image-based rendering systems in computer graphics is that acquisition of realistic surface models is a difficult problem. While geometry-based rendering technology has made significant strides towards achieving photorealism, creating accurate models is still nearly as difficult as it was ten years ago. Technological advances in three-dimensional scanning provide some promise in model building. However, they also verify our worst suspicions— the geometry of the real-world is exceedingly complex. Ironically, the primary subjective measure of image quality used by proponents of geometric rendering systems is the degree with which the resulting images are indistinguishable from photographs.

One liability of image-based rendering systems is the lack of a consistent framework within which to judge the validity of the results. Fundamentally, this arises from the absence of a clear problem definition. Geometry-based rendering, on the other hand, has a solid foundation; it uses analytic and projective geometry to describe the world's shape and physics to describe the world's surface properties and the light's interaction with those surfaces.

This paper presents a consistent framework for the evaluation of image-based rendering systems, and gives a concise problem definition. We then evaluate previous image-based rendering methods within this new framework. Finally, we present our own image-based rendering methodology and results from our prototype implementation.

2. THE PLENOPTIC FUNCTION

Adelson and Bergen [1] assigned the name *plenoptic* function (from the latin root *plenus*, meaning complete or full, and *optic* pertaining to vision) to the pencil of rays visible from any point in space, at any time, and over any range of wavelengths. They used this function to develop a taxonomy for evaluating models of low-level vision. The plenoptic function describes all of the radiant energy that can be perceived from the point of view of the observer rather than the point of view of the source. They postulate

> *"… all the basic visual measurements can be considered to characterize local change along one or two dimensions of a single function that describes the structure of the information in the light impinging on an observer."*

Adelson and Bergen further formalized this functional description by providing a parameter space over which the plenoptic function is valid, as shown in Figure 1. Imagine an idealized eye which we are free to place at any point in space (V_x, V_y, V_z). From there we can select any of the viewable rays by choosing an azimuth and elevation angle

†‡CB 3175 Sitterson Hall, Chapel Hill, NC 27599
† (919) 962-1797 mcmillan@cs.unc.edu http://www.cs.unc.edu/~mcmillan
‡ (919) 962-1886 gb@cs.unc.edu http://www.cs.unc.edu/~gb

(θ,ϕ) as well as a band of wavelengths, λ, which we wish to consider.

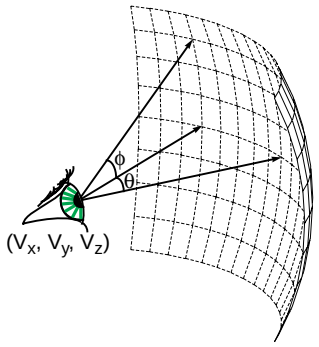


FIGURE 1. The plenoptic function describes all of the image information visible from a particular viewing position.

In the case of a dynamic scene, we can additionally choose the time, *t*, at which we wish to evaluate the function. This results in the following form for the plenoptic function:

$$p = P(\theta, \phi, \lambda, V_x, V_y, V_z, t) \quad (1)$$

In computer graphics terminology, the plenoptic function describes the set of all possible environment maps for a given scene. For the purposes of visualization, one can consider the plenoptic function as a scene representation. In order to generate a view from a given point in a particular direction we would need to merely plug in appropriate values for (V_x, V_y, V_z) and select from a range of (θ,ϕ) for some constant *t*.

We define a complete sample of the plenoptic function as a full spherical map for a given viewpoint and time value, and an incomplete sample as some solid angle subset of this spherical map.

Within this framework we can state the following problem definition for image-based rendering. *Given a set of discrete samples (complete or incomplete) from the plenoptic function, the goal of image-based rendering is to generate a continuous representation of that function.* This problem statement provides for many avenues of exploration, such as how to optimally select sample points and how to best reconstruct a continuous function from these samples.

3. PREVIOUS WORK

3.1 Movie-Maps

The Movie-Map system by Lippman [17] is one of the earliest attempts at constructing an image-based rendering system. In Movie-Maps, incomplete plenoptic samples are stored on interactive video laser disks. They are accessed randomly, primarily by a change in viewpoint; however, the system can also accommodate panning, tilting, or zooming about a fixed viewing position. We can characterize Lippman's plenoptic reconstruction technique as a nearest-neighbor interpolation because, when given a set of input parameters $(V_x, V_y, V_z, \theta, \phi, t)$, the Movie-Map system can select the nearest partial sample. The Movie-Map form of image-based rendering can also be interpreted as a table-based evaluation of the plenoptic function. This interpretation reflects the database structure common to most image-based systems.

3.2 Image Morphing

Image morphing is a very popular image-based rendering technique [4], [28]. Generally, morphing is considered to occur between two images. We can think of these images as endpoints along some path through time and/or space. In this interpretation, morphing becomes a method for reconstructing partial samples of the continuous plenoptic function along this path. In addition to photometric data, morphing uses additional information describing the image flow field. This information is usually hand crafted by an animator. At first glance, this type of augmentation might seem to place it outside of the plenoptic function's domain. However, several authors in the field of computer vision have shown that this type of image flow information is equivalent to changes in the local intensity due to infinitesimal perturbations of the plenoptic function's independent variables [20], [13]. This local derivative behavior can be related to the intensity gradient via applications of the chain rule. In fact, morphing makes an even stronger assumption that the flow information is constant along the entire path, thus amounting to a locally linear approximation. Also, a blending function is often used to combine both reference images after being partially flowed from their initial configurations to a given point on the path. This blending function is usually some linear combination of the two images based on what percentage of the path's length has been traversed. Thus, morphing is a plenoptic reconstruction method which interpolates between samples and uses local derivative information to construct approximations.

3.3 View Interpolation

Chen's and Williams' [8] view interpolation employs incomplete plenoptic samples and image flow fields to reconstruct arbitrary viewpoints with some constraints on gaze angle. The reconstruction process uses information about the local neighborhood of a sample. Chen and Williams point out and suggest a solution for one of the key problems of image-based rendering— determining the visible surfaces. Chen and Williams chose to presort the quadtree compressed flow-field in a back-to-front order according to its (geometric) z-value. This approach works well when all of the partial sample images share a common gaze direction, and the synthesized viewpoints are restricted to stay within 90 degrees of this gaze angle.

An image flow field alone allows for many ambiguous visibility solutions, unless we restrict ourselves to flow fields that do not fold, such as rubber-sheet local spline warps or thin-plate global spline warps. This problem must be considered in any general-purpose image-based rendering system, and ideally, it should be done without transporting the image into the geometric-rendering domain.

Establishing flow fields for a view interpolation system can also be problematic. Chen and Williams used pre-rendered synthetic images to determine flow fields from the z-values. In general, accurate flow field information between two samples can only be established for points that are mutually visible to both samples. This points out a shortcoming in the use of partial samples, because reference images seldom have a 100% overlap.

Like morphing, view interpolation uses photometric information as well as local derivative information in its reconstruction process. This locally linear approximation is nicely exploited to approximate perspective depth effects, and Chen and Williams show it to be correct for lateral motions relative to the gaze direction. View interpolation, however, adds a nonlinearity by allowing the visibility process to determine the blending function between reference frames in a closest-take-all (a.k.a. winner-take-all) fashion.

3.4 Laveau and Faugeras

Laveau and Faugeras [15] have taken advantage of the fact that the epipolar geometries between images restrict the image flow field in such a way that it can be parameterized by a single disparity value and a fundamental matrix which represents the epipolar relationship. They also provide a two-dimensional raytracing-like solution to the visibility problem which does not require an underlying geometric description. Their method does, however, require establishing correspondences for each image point along the ray's path. The Laveau and Faugeras system also uses partial plenoptic samples, and results are shown only for overlapping regions between views.

Laveau and Faugeras also discuss the combination of information from several views but primarily in terms of resolving visibility. By relating the reference views and the desired views by the homogenous transformations between their projections, Laveau and Faugeras can compute exact perspective depth solutions. The recon-

struction process again takes advantage of both image data and local derivative information to reconstruct the plenoptic function.

3.5 Regan and Pose

Regan and Pose [23] describe a hybrid system in which plenoptic samples are generated on the fly by a geometry-based rendering system at available rendering rates, while interactive rendering is provided by the image-based subsystem. At any instant, a user interacts with a single plenoptic sample. This allows the user to make unconstrained changes in the gaze angle about the sample point. Regan and Pose also discuss local reconstruction approximations due to changes in the viewing position. These approximations amount to treating the objects in the scene as being placed at infinity, resulting in a loss of the kinetic depth effect. These partial updates can be combined with the approximation values.

4. PLENOPTIC MODELING

We claim that all image-based rendering approaches can be cast as attempts to reconstruct the plenoptic function from a sample set of that function. We believe that there are significant insights to be gleaned from this characterization. In this section, we propose our prototype system in light of this plenoptic function framework.

We call our image-based rendering approach Plenoptic Modeling. Like other image-based rendering systems, the scene description is given by a series of reference images. These reference images are subsequently warped and combined to form representations of the scene from arbitrary viewpoints. The warping function is defined by image flow field information that can either be supplied as an input or derived from the reference images.

Our discussion of the plenoptic modeling image-based rendering system is broken down into four sections. First, we discuss the representation of the plenoptic samples. Next, we discuss their acquisition. The third section covers the determination of image flow fields, if required. And, finally, we describe how to reconstruct the plenoptic function from these sample images.

4.1 Plenoptic Sample Representation

The most natural surface for projecting a complete plenoptic sample is a unit sphere centered about the viewing position. One difficulty of spherical projections, however, is the lack of a representation that is suitable for storage on a computer. This is particularly difficult if a uniform (i.e. equal area) discrete sampling is required. This difficulty is reflected in the various distortions which arise in planar projections of world maps in cartography. Those uniform mappings which do exist are generally ill-suited for systematic access as a data structure. Furthermore, those which do map to a plane with consistent neighborhood relationships are generally quite distorted and, therefore, non-uniform.

A set of six planar projections in the form of a cube has been suggested by Greene [10] as an efficient representation for environment maps. While this representation can be easily stored and accessed by a computer, it provides significant problems relating to acquisition, alignment, and registration when used with real, non-computer-generated images. The orthogonal orientation of the cube faces requires precise camera positioning. The wide, 90 degree field-of-view of each face requires expensive lens systems to avoid optical distortion. Also, the planar mapping does not represent a uniform sampling, but instead, is considerably oversampled in the edges and corners. However, the greatest difficulty of a cube-oriented planar projection set is describing the behavior of the image flow fields across the boundaries between faces and at corners. This is not an issue when the six planar projections are used solely as an environment map, but it adds a considerable overhead when it is used for image analysis.

We have chosen to use a cylindrical projection as the plenoptic sample representation. One advantage of a cylinder is that it can be easily unrolled into a simple planar map. The surface is without boundaries in the azimuth direction, which simplifies correspondence searches required to establish image flow fields. One shortcoming of a projection on a finite cylindrical surface is the boundary conditions introduced at the top and bottom. We have chosen not to employ end caps on our projections, which has the problem of limiting the vertical field of view within the environment.

4.2 Acquiring Cylindrical Projections

A significant advantage of a cylindrical projection is the simplicity of acquisition. The only acquisition equipment required is a video camera and a tripod capable of continuous panning. Ideally, the camera's panning motion would be around the exact optical center of the camera. In practice, in a scene where all objects are relatively far from the tripod's rotational center, a slight misalignment offset can be tolerated.

Any two planar perspective projections of a scene which share a common viewpoint are related by a two-dimensional homogenous transform:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (2)$$

$$x' = \frac{u}{w} \qquad y' = \frac{v}{w}$$

where x and y represent the pixel coordinates of an image I, and x' and y' are their corresponding coordinates in a second image I'. This well known result has been reported by several authors [12], [28], [22]. The images resulting from typical camera motions, such as pan, tilt, roll, and zoom, can all be related in this fashion. When creating a cylindrical projection, we will only need to consider panning camera motions. For convenience we define the camera's local coordinate system such that the panning takes place entirely in the x-z plane.

In order to reproject an individual image into a cylindrical projection, we must first determine a model for the camera's projection or, equivalently, the appropriate homogenous transforms. Many different techniques have been developed for inferring the homogenous transformation between images sharing common centers of projection. The most common technique [12] involves establishing four corresponding points across each image pair. The resulting transforms provide a mapping of pixels from the planar projection of the first image to the planar projection of the second. Several images could be composited in this fashion by first determining the transform which maps the Nth image to image N-1. These transforms can be catenated to form a mapping of each image to the plane of the first. This approach, in effect, avoids direct determination of an entire camera model by performing all mappings between different instances of the same camera. Other techniques for deriving these homogeneous transformations without specific point correspondences have also been described [22], [25].

The set of homogenous transforms, $\boldsymbol{H}_i$, can be decomposed into two parts which will allow for arbitrary reprojections in a manner similar to [11]. These two parts include an intrinsic transform, $\boldsymbol{S}$, which is determined entirely by camera properties, and an extrinsic transform, $\boldsymbol{R}_i$, which is determined by the rotation around the camera's center of projection:

$$\bar{u} = \boldsymbol{H}_i \bar{x} = \boldsymbol{S}^{-1} \boldsymbol{R}_i \boldsymbol{S} \bar{x} \qquad (3)$$

This decomposition decouples the projection and rotational components of the homogeneous transform. By an appropriate choice of coordinate systems and by limiting the camera's motion to panning, the extrinsic transform component is constrained to a function of a single parameter rotation matrix describing the pan.

$$\boldsymbol{R}_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad (4)$$

Since the intrinsic component's properties are invariant over all of the images, the decomposition problem can be broken into two parts: the determination of the extrinsic rotation component, $\boldsymbol{R}_i$, followed by the determination of an intrinsic projection component, $\boldsymbol{S}$. The first step in our method determines estimates for the extrinsic panning angle between each image pair of the panning sequence. This is accomplished by using a linear approximation to an infinitesimal rotation by the angle θ. This linear approximation results from substituting $1 + \boldsymbol{O}(\theta^2)$ for the cosine terms and $\theta + \boldsymbol{O}(\theta^3)$ for the sine terms of the rotation matrix. This infinitesimal perturbation has been shown by [14] to reduce to the following approximate equations:

$$\begin{aligned} x' &= x - f\theta - \frac{\theta\,(x - C_x)^2}{f} + \boldsymbol{O}(\theta^2) \\ y' &= y - \frac{\theta\,(x - C_x)\,(y - C_y)}{f} + \boldsymbol{O}(\theta^2) \end{aligned} \tag{5}$$

where f is the apparent focal length of the camera measured in pixels, and (C_x, C_y) is the pixel coordinate of the intersection of the optical axis with the image plane. (C_x, C_y) is initially estimated to be at the center pixel of the image plane. A better estimate for (C_x, C_y) is found during the intrinsic matrix solution.

These equations show that small panning rotations can be approximated by translations for pixels near the image's center. We require that some part of each image in the sequence must be visible in the successive image, and that some part of the final image must be visible in the first image of the sequence. The first stage of the cylindrical registration process attempts to register the image set by computing the optimal translation in x which maximizes the normalized correlation within a region about the center third of the screen. This is first computed at a pixel resolution, then refined on a 0.1 subpixel grid, using a Catmull-Rom interpolation spline to compute subpixel intensities. Once these translations, t_i, are computed, Newton's method is used to convert them to estimates of rotation angles and the focal length, using the following equation:

$$2\pi - \sum_{i=1}^{N} \operatorname{atan}\left(\frac{t_i}{f}\right) = 0 \tag{6}$$

where N is the number of images comprising the sequence. This usually converges in as few as five iterations, depending on the original estimate for f. This first phase determines an estimate for the relative rotational angles between each of the images (our extrinsic parameters) and the initial focal length estimate measured in pixels (one of the intrinsic parameters).

The second stage of the registration process determines the **S**, or structural matrix, which describes various camera properties such as the tilt and roll angles which are assumed to remain constant over the group of images. The following model is used:

$$\boldsymbol{S} = \Omega_x \Omega_z \boldsymbol{P} \tag{7}$$

where **P** is the projection matrix:

$$\boldsymbol{P} = \begin{bmatrix} 1 & \sigma & -C_x \\ 0 & \rho & -C_y \\ 0 & 0 & f \end{bmatrix} \tag{8}$$

and (C_x, C_y) is the estimated center of the viewplane as described previously, σ is a skew parameter representing the deviation of the sampling grid from a rectilinear grid, ρ determines the sampling grid's aspect ratio, and f is the focal length in pixels as determined from the first alignment stage.

The remaining terms, Ω_x and Ω_z, describe the combined effects of camera orientation and deviations of the viewplane's orientation from perpendicular to the optical axis. Ideally, the viewplane would be normal to the optical axis, but manufacturing tolerances allow these numbers to vary slightly [27].

$$\Omega_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega_x & -\sin\omega_x \\ 0 & \sin\omega_x & \cos\omega_x \end{bmatrix} \tag{9}$$

$$\Omega_z = \begin{bmatrix} \cos\omega_z & -\sin\omega_z & 0 \\ \sin\omega_z & \cos\omega_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

In addition, the ω_z term is indistinguishable from the camera's roll angle and, thus, represents both the image sensor's and the camera's rotation. Likewise, ω_x, is combined with an implicit parameter, ϕ, that represents the relative tilt of the camera's optical axis out of the panning plane. If ϕ is zero, the images are all tangent to a cylinder and for a nonzero ϕ the projections are tangent to a cone.

This gives six unknown parameters, $(C_x, C_y, \sigma, \rho, \omega_x, \omega_z)$, to be determined in the second stage of the registration process. Notice that, when combined with the θ_i and f parameters determined in the first stage, we have a total of eight parameters for each image, which is consistent with the number of free parameters in a general homogeneous matrix.

The structural matrix, **S**, is determined by minimizing the following error function:

$$\text{error}(C_x, C_y, \sigma, \rho, \omega_x, \omega_z) = \sum_{i=1}^{n} 1 - \text{Correlation}(I_{i-1}, S^{-1} R_{y_i} S I_i) \tag{11}$$

where I_{i-1} and I_i represent the center third of the pixels from images i-1 and i respectively. Using Powell's multivariable minimization method [23] with the following initial values for our six parameters,

$$\begin{aligned} C_x &= \frac{\text{image width}}{2} \qquad C_y = \frac{\text{image height}}{2} \\ \sigma &= 0 \qquad \rho = 1 \qquad \omega_x = 0 \qquad \omega_z = 0 \end{aligned} \tag{12}$$

the solution typically converges in about six iterations. At this point we will have a new estimate for (C_x, C_y) which can be fed back into stage one, and the entire process can be repeated.

The registration process results in a single camera model, $\mathbf{S}(C_x, C_y, \sigma, \rho, \omega_x, \omega_z, f)$, and a set of the relative rotations, θ_i, between each of the sampled images. Using these parameters, we can compose mapping functions from any image in the sequence to any other image as follows:

$$\boldsymbol{I}'_i = \boldsymbol{S}^{-1} \boldsymbol{R}_{y_{i+1}} \boldsymbol{R}_{y_{i+2}} \boldsymbol{R}_{y_{i+3}} \ldots \boldsymbol{R}_{y_j} \boldsymbol{S} \boldsymbol{I}_j \tag{13}$$

We can also reproject images onto arbitrary surfaces by modifying **S**. Since each image pixel determines the equation of a ray from the center-of-projection, the reprojection process merely involves intersecting these rays with the projection manifold.

4.3 Determining Image Flow Fields

Given two or more cylindrical projections from different positions within a static scene, we can determine the relative positions of centers-of-projection and establish geometric constraints across all potential reprojections. These positions can only be computed to a scale factor. An intuitive argument for this is that from a set of images alone, one cannot determine if the observer is looking at a model or a full-sized scene. This implies that at least one measurement is required to establish a scale factor. The measurement may be taken either between features that are mutually visible within images, or the distance between the acquired image's camera positions can be used. Both techniques have been used with little difference in results.

To establish the relative relationships between any pair of cylindrical projections, the user specifies a set of corresponding points that are visible from both views. These points can be treated as rays in space with the following form:

$$\bar{x}_a(\theta, v) = \bar{C}_a + t\bar{D}_a(\theta, v) \qquad \bar{D}_a(\theta, v) = \begin{bmatrix} \cos(\phi_a - \theta) \\ \sin(\phi_a - \theta) \\ k_a\left(C_{v_a} - v\right) \end{bmatrix} \quad (14)$$

where $\bar{C}_a = (A_x, A_y, A_z)$ is the unknown position of the cylinder's center of projection, ϕ_a is the rotational offset which aligns the angular orientation of the cylinders to a common frame, k_a is a scale factor which determines the vertical field-of-view, and C_{v_a} is the scanline where the center of projection would project onto the scene (i.e. the line of zero elevation, like the equator of a spherical map).

A pair of tiepoints, one from each image, establishes a pair of rays which ideally intersect at the point in space identified by the tiepoint. In general, however, these rays are skewed. Therefore, we use the point that is simultaneously closest to both rays as an estimate of the point's position, $\bar{p}$, as determined by the following derivation.

$$\bar{p}(\theta_a, v_a, \theta_b, v_b) = \frac{\bar{x}_a - \bar{x}_b}{2} \quad (15)$$

where (θ_a, v_a) and (θ_b, v_b) are the tiepoint coordinates on cylinders A and B respectively. The two points, $\bar{x}_a$ and $\bar{x}_b$, are given by

$$\begin{aligned} \bar{x}_a &= \bar{C}_a + t\bar{D}_a(\theta_a, v_a) \\ \bar{x}_b &= \bar{C}_b + s\bar{D}_b(\theta_b, v_b) \end{aligned} \quad (16)$$

where

$$\begin{aligned} t &= \frac{\mathrm{Det}\left[\bar{C}_a - \bar{C}_b, \bar{D}_b(\theta_b, v_b), \bar{D}_a(\theta_a, v_a) \times \bar{D}_b(\theta_b, v_b)\right]}{\left|\bar{D}_a(\theta_a, v_a) \times \bar{D}_b(\theta_b, v_b)\right|^2} \\ s &= \frac{\mathrm{Det}\left[\bar{C}_b - \bar{C}_a, \bar{D}_a(\theta_a, v_a), \bar{D}_a(\theta_a, v_a) \times \bar{D}_b(\theta_b, v_b)\right]}{\left|\bar{D}_a(\theta_a, v_a) \times \bar{D}_b(\theta_b, v_b)\right|^2} \end{aligned} \quad (17)$$

This allows us to pose the problem of finding a cylinder's position as a minimization problem. For each pair of cylinders we have two sets of six unknowns, $[(A_x,A_y,A_z,\phi_a,k_a,C_{va}), (B_x,B_y,B_z,\phi_b,k_b,C_{vb})]$. In general, we have good estimates for the k and C_v terms, since these values are found by the registration phase. The position of the cylinders is determined by minimizing the distance between these skewed rays. We also choose to assign a penalty for shrinking the vertical height of the cylinder in order to bring points closer together. This penalty could be eliminated by accepting either the k or C_v values given by the registration.

We have tested this approach using from 12 to 500 tiepoints, and have found that it converges to a solution in as few as ten iterations of Powell's method. Since no correlation step is required, this process is considerably faster than the minimization step required to determine the structural matrix, ***S***.

The use of a cylindrical projection introduces significant geometric constraints on where a point viewed in one projection might appear in a second. We can capitalize on these restrictions when we wish to automatically identify corresponding points across cylinders. While an initial set of 100 to 500 tiepoints might be established by hand, this process is far too tedious to establish a mapping for the entire cylinder. Next, we present a geometric constraint for cylindrical projections that determines the possible positions of a point given its position in some other cylinder. This constraint plays the same role that the epipolar geometries [18], [9], used in the computer vision community for depth-from-stereo computations, play for planar projections.

First, we will present an intuitive argument for the existence of such an invariant. Consider yourself at the center of a cylindrical projection. Every point on the cylinder around you corresponds to a ray in space as given by the cylindrical epipolar geometry equation. When one of the rays is observed from a second cylinder, its path projects to a curve which appears to begin at the point corresponding to the origin of the first cylinder, and it is constrained to pass through the point's image on the second cylinder.

This same argument could obviously have been made for a planar projection. And, since two points are identified (the virtual image of the camera in the second projection along with the corresponding point) and, because a planar projection preserve lines, a unique, so called epipolar line is defined. This is the basis for an epipolar geometry, which identifies pairs of lines in two planar projections such that if a point falls upon one line in the first image, it is constrained to fall on the corresponding line in the second image. The existence of this invariant reduces the search for corresponding points from an $\mathbf{O}(N^2)$ problem to $\mathbf{O}(N)$.

Cylindrical projections, however, do not preserve lines. In general, lines map to quadratic parametric curves on the surface of a cylinder. Surprisingly, we can completely specify the form of the curve with no more information than was needed in the planar case.

The paths of these curves are uniquely determined sinusoids. This *cylindrical epipolar geometry* is established by the following equation.

$$v(\theta) = \frac{N_x \cos(\phi_a - \theta) + N_y \sin(\phi_a - \theta)}{N_z k_a} + C_{v_a} \quad (18)$$

where

$$\bar{N} = (\bar{C}_b - \bar{C}_a) \times \bar{D}_a(\theta_a, v_a) \quad (19)$$

This formula gives a concise expression for the curve formed by the projection of a ray across the surface of a cylinder, where the ray is specified by its position on some other cylinder.

This cylindrical epipolar relationship can be used to establish image flow fields using standard computer vision methods. We have used correlation methods [9], a simulated annealing-like relaxation method [3], and the method of differences [20] to compute stereo disparities between cylinder pairs. Each method has its strengths and weaknesses. We refer the reader to the references for further details.

4.4 Plenoptic Function Reconstruction

Our image-based rendering system takes as input cylindrically projected panoramic reference images along with scalar disparity images relating each cylinder pair. This information is used to automatically generate image warps that map reference images to arbitrary cylindrical or planar views that are capable of describing both occlusion and perspective effects.

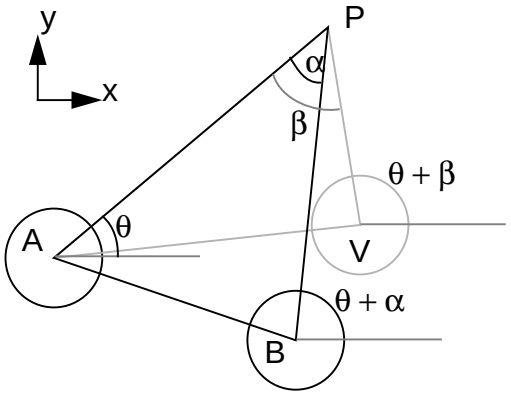


FIGURE 2. Diagram showing the transfer of the known disparity values between cylinders A and B to a new viewing position V.

We begin with a description of cylindrical-to-cylindrical mappings. Each angular disparity value, α, of the disparity images, can be readily converted into an image flow vector field, $(\theta + \alpha, v(\theta + \alpha))$ using the epipolar relation given by Equation 18 for each position on the cylinder, (θ, v). We can transfer disparity values from the known cylindrical pair to a new cylindrical projection

in an arbitrary position, as in Figure 2, using the following equations.

$$\begin{aligned} a &= (B_x - V_x)\cos(\phi_A - \theta) + (B_y - V_y)\sin(\phi_A - \theta) \\ b &= (B_y - A_y)\cos(\phi_A - \theta) - (B_x - A_x)\sin(\phi_A - \theta) \\ c &= (V_y - A_y)\cos(\phi_A - \theta) - (V_x - A_x)\sin(\phi_A - \theta) \end{aligned} \quad (20)$$

$$\cot(\beta(\theta, v)) = \frac{a + b\cot(\alpha(\theta, v))}{c}$$

By precomputing $[\cos(\phi_i - \theta), \sin(\phi_i - \theta)]$ for each column of the cylindrical reference image and storing $\cot(\alpha)$ in place of the disparity image, this transfer operation can be computed at interactive speeds.

Typically, once the disparity images have been transferred to their target, the cylindrical projection would be reprojected as a planar image for viewing. This reprojection can be combined with the disparity transfer to give a single image warp that performs both operations. To accomplish this, a new intermediate quantity, δ, called the *generalized angular disparity* is defined as follows:

$$d = (B_x - A_x)\cos(\phi_A - \theta) + (B_y - A_y)\sin(\phi_A - \theta)$$

$$\delta(\theta, v) = \frac{1}{d + b\cot(\alpha(\theta, v))} \quad (21)$$

This scalar function is the cylindrical equivalent to the classical stereo disparity. Finally, a composite image warp from a given reference image to any arbitrary planar projection can be defined as

$$\begin{aligned} x(\theta, v) &= \frac{\bar{r} \cdot D_A(\theta, v) + k_r\delta(\theta, v)}{\bar{n} \cdot D_A(\theta, v) + k_n\delta(\theta, v)} \\ y(\theta, v) &= \frac{\bar{s} \cdot D_A(\theta, v) + k_s\delta(\theta, v)}{\bar{n} \cdot D_A(\theta, v) + k_n\delta(\theta, v)} \end{aligned} \quad (22)$$

where

$$\begin{aligned} \bar{r} &= \bar{v} \times \bar{o} & k_r &= \bar{r} \cdot (\bar{C}_a - \bar{V}) \\ \bar{s} &= \bar{o} \times \bar{u} & k_s &= \bar{s} \cdot (\bar{C}_a - \bar{V}) \\ \bar{n} &= \bar{u} \times \bar{v} & k_n &= \bar{n} \cdot (\bar{C}_a - \bar{V}) \end{aligned} \quad (23)$$

and the vectors $\bar{p}$, $\bar{o}$, $\bar{u}$ and $\bar{v}$ are defined by the desired view as shown in Figure 3.

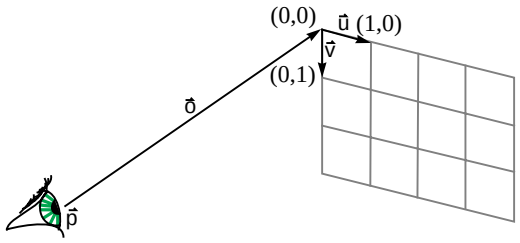


FIGURE 3. The center-of-projection, $\bar{p}$, a vector to the origin, $\bar{o}$, and two spanning vectors ($\bar{u}$ and $\bar{v}$) uniquely determine the planar projection.

In the case where $\delta(\theta, v) = \text{constant}$, the image warp defined by Equation 22, reduces to a simple reprojection of the cylindrical image to a desired planar view. The perturbation introduced by allowing $\delta(\theta, v)$ to vary over the image allows arbitrary shape and occlusions to be represented.

Potentially, both the cylinder transfer and image warping approaches are many-to-one mappings. For this reason we must consider visibility. The following simple algorithm can be used to determine an enumeration of the cylindrical mesh which guarantees a proper back-to-front ordering, (See Appendix). We project the desired viewing position onto the reference cylinder being warped and partition the cylinder into four toroidal sheets. The sheet boundaries are defined by the θ and v coordinates of two points, as shown in Figure 4. One point is defined by the intersection of the cylinder with the vector from the origin through the eye's position. The other is the intersection with the vector from the eye through the origin.

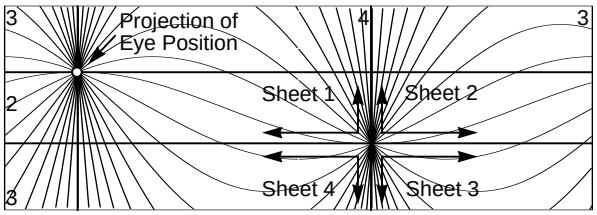


FIGURE 4. A back-to-front ordering of the image flow field can be established by projecting the eye's position onto the cylinder's surface and dividing it into four toroidal sheets.

Next, we enumerate each sheet such that the projected image of the desired viewpoint is the last point drawn. This simple partitioning and enumeration provides a back-to-front ordering for use by a painter's style rendering algorithm. This hidden-surface algorithm is a generalization of Anderson's [2] visible line algorithm to arbitrary projected grid surfaces. Additional details can be found in [21].

At this point, the plenoptic samples can be warped to their new position according to the image flow field. In general, these new pixel positions lie on an irregular grid, thus requiring some sort of reconstruction and resampling. We use a forward-mapping [28] reconstruction approach in the spirit of [27] in our prototype. This involves computing the projected kernel's size based on the current disparity value and the derivatives along the epipolar curves.

While the visibility method properly handles mesh folds, we still must consider the tears (or excessive stretching) produced by the exposure of previously occluded image regions. In view interpolation [8] a simple "distinguished color" heuristic is used based on the screen space projection of the neighboring pixel on the same scanline. This approach approximates stretching for small regions of occlusion, where the occluder still abuts the occluded region. And, for large exposed occluded regions, it tends to interpolate between the boundaries of the occluded region. These exposure events can be handled more robustly by combining, on a pixel-by-pixel basis, the results of multiple image warps according to the smallest-sized reconstruction kernel.

5. RESULTS

We collected a series of images using a video camcorder on a leveled tripod in the front yard of one of the author's home. Accurate leveling is not strictly necessary for the method to work. When the data were collected, no attempt was made to pan the camera at a uniform angular velocity. The autofocus and autoiris features of the camera were disabled, in order to maintain a constant focal length during the collection process. The frames were then digitized at a rate of approximately 5 frames per second to a resolution of 320 by 240 pixels. An example of three sequential frames are shown below.

Immediately after the collection of the first data set, the process was repeated at a second point approximately 60 inches from the first. The two image sequences were then separately registered using the methods described in Section 4.2. The images were reprojected onto the surface of a cylinder with a resolution of 3600 by 300 pixels. The results are shown in Figures 5a and 5b. The operating room scene, in Figure 5c, was also constructed using these same methods.

Next, the epipolar geometry was computed by specifying 12 tiepoints on the front of the house. Additional tiepoints were gradually added to establish an initial disparity image for use by the simulated

FIGURE 5. Cylindrical images a and b are panoramic views separated by approximately 60 inches. Image c is a panoramic view of an operating room. In image d, several epipolar curves are superimposed onto cylindrical image a.

annealing and method of differences stereo-correspondence routines. As these tiepoints were added, we also refined the epipolar geometry and cylinder position estimates. The change in cylinder position, however, was very slight. In Figure 5d, we show a cylindrical image with several epipolar curves superimposed. Notice how the curves all intersect at the alternate camera's virtual image and vanishing point.

After the disparity images are computed, they can be interactively warped to new viewing positions. The following four images show various reconstructions. When used interactively, the warped images provide a convincing kinetic depth effect.

6. CONCLUSIONS

The plenoptic function provides a consistent framework for image-based rendering systems. The various image-based methods, such as morphing and view interpolation, are characterized by the different ways they implement the three key steps of sampling, reconstructing, and resampling the plenoptic function.

We have described our approach to each of these steps. Our method for sampling the plenoptic function can be done with equipment that is commonly available, and it results in cylindrical samples about a point. All the necessary parameters are automatically estimated from a sequence of images resulting from panning a video camera through a full circle.

Reconstructing the function from these samples requires estimating the optic flow of points when the view point is translated. Though this problem can be very difficult, as evidenced by thirty years of computer vision and photogrammetry research, it is greatly simplified when the samples are relatively close together. This is because there is little change in the image between samples (which makes the estimation easier), and because the viewer is never far from a sample (which makes accurate estimation less important).

Resampling the plenoptic function and reconstructing a planar projection are the key steps for display of images from arbitrary viewpoints. Our methods allow efficient determination of visibility and real-time display of visually rich environments on conventional workstations without special purpose graphics acceleration.

The plenoptic approach to modeling and display will provide robust and high-fidelity models of environments based entirely on a set of reference projections. The degree of realism will be determined by the resolution of the reference images rather than the number of primitives used in describing the scene. Finally, the difficulty of producing realistic models of real environments will be greatly reduced by replacing geometry with images.

ACKNOWLEDGMENTS

We are indebted to the following individuals for their contributions and suggestions on this work: Henry Fuchs, Andrei State, Kevin Arthur, Donna McMillan, and all the members of the UNC/UPenn collaborative telepresence-research group.

This research is supported in part by Advanced Research Projects Agency contract no. DABT63-93-C-0048, NSF Cooperative Agreement no. ASC-8920219, Advanced Research Projects Agency order no. A410, and National Science Foundation grant no. MIP-9306208. Approved by ARPA for public release - distribution unlimited.

REFERENCES

[1] Adelson, E. H., and J. R. Bergen, *"The Plenoptic Function and the Elements of Early Vision,"* **Computational Models of Visual Processing**, Chapter 1, Edited by Michael Landy and J. Anthony Movshon. The MIT Press, Cambridge, Mass. 1991.

[2] Anderson, D., *"Hidden Line Elimination in Projected Grid Surfaces,"* **ACM Transactions on Graphics**, October 1982.

[3] Barnard, S.T. "A Stochastic Approach to Stereo Vision," SRI International, Technical Note 373, April 4, 1986.

[4] Beier, T. and S. Neely, *"Feature-Based Image Metamorphosis,"* **Computer Graphics** (SIGGRAPH'92 Proceedings), Vol. 26, No. 2, pp. 35-42, July 1992.

[5] Blinn, J. F. and M. E. Newell, *"Texture and Reflection in Computer Generated Images,"* ***Communications of the ACM***, vol. 19, no. 10, pp. 542-547, October 1976.

[6] Bolles, R. C., H. H. Baker, and D. H. Marimont, *"Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion,"* **International Journal of Computer Vision**, Vol. 1, 1987.

[7] Catmull, E., *"A Subdivision Algorithm for Computer Display of Curved Surfaces"* (Ph. D. Thesis), Department of Computer Sci-

ence, University of Utah, Tech. Report UTEC-CSc-74-133, December 1974.

[8] Chen, S. E. and L. Williams. "*View Interpolation for Image Synthesis,*" **Computer Graphics** (SIGGRAPH'93 Proceedings), pp. 279-288, July 1993.

[9] Faugeras, O., **Three-dimensional Computer Vision: A Geometric Viewpoint**, The MIT Press, Cambridge, Massachusetts, 1993.

[10] Greene, N., "*Environment Mapping and Other Applications of World Projections,*" **IEEE Computer Graphics and Applications**, November 1986.

[11] Hartley, R.I., "*Self-Calibration from Multiple Views with a Rotating Camera,*" **Proceedings of the European Conference on Computer Vision**, May 1994.

[12] Heckbert, P. S., "*Fundamentals of Texture Mapping and Image Warping,*" Masters Thesis, Dept. of EECS, UCB, Technical Report No. UCB/CSD 89/516, June 1989.

[13] Horn, B., and B.G. Schunck, "*Determining Optical Flow,*" **Artificial Intelligence**, Vol. 17, 1981.

[14] Kanatani, K., "*Transformation of Optical Flow by Camera Rotation,*" **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. 10, No. 2, March 1988.

[15] Laveau, S. and O. Faugeras, "*3-D Scene Representation as a Collection of Images and Fundamental Matrices,*" INRIA, Technical Report No. 2205, February, 1994.

[16] Lenz, R. K. and R. Y. Tsai, "*Techniques for Calibration of the Scale Factor and Image Center for High Accuracy 3D Machine Vision Metrology,*" **Proceedings of IEEE International Conference on Robotics and Automation**, March 31 - April 3, 1987.

[17] Lippman, A., "*Movie-Maps: An Application of the Optical Videodisc to Computer Graphics,*" **SIGGRAPH '80 Proceedings**, 1980.

[18] Longuet-Higgins, H. C., "*A Computer Algorithm for Reconstructing a Scene from Two Projections,*" **Nature**, Vol. 293, September 1981.

[19] Longuet-Higgins, H. C., "*The Reconstruction of a Scene From Two Projections - Configurations That Defeat the 8-Point Algorithm,*" **Proceedings of the First IEEE Conference on Artificial Intelligence Applications**, Dec 1984.

[20] Lucas, B., and T. Kanade, "*An Iterative Image Registration Technique with an Application to Stereo Vision,*" **Proceedings of the Seventh International Joint Conference on Artificial Intelligence**, Vancouver, 1981.

[21] McMillan, Leonard, "*A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces,*" Department of Computer Science, UNC, Technical Report TR95-005, 1995.

[22] Mann, S. and R. W. Picard, "*Virtual Bellows: Constructing High Quality Stills from Video,*" **Proceedings of the First IEEE International Conference on Image Processing**, November 1994.

[23] Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, **Numerical Recipes in C**, Cambridge University Press, Cambridge, Massachusetts, pp. 309-317, 1988.

[24] Regan, M., and R. Pose, "*Priority Rendering with a Virtual Reality Address Recalculation Pipeline,*" **SIGGRAPH'94 Proceedings**, 1994.

[25] Szeliski, R., "*Image Mosaicing for Tele-Reality Applications,*" **DEC and Cambridge Research Lab Technical Report**, CRL 94/2, May 1994.

[26] Tomasi, C., and T. Kanade, "*Shape and Motion from Image Streams: a Factorization Method; Full Report on the Orthographic Case,*" Technical Report, CMU-CS-92-104, Carnegie Mellon University, March 1992.

[27] Tsai, R. Y., "*A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses,*" **IEEE Journal of Robotics and Automation**, Vol. RA-3, No. 4, August 1987.

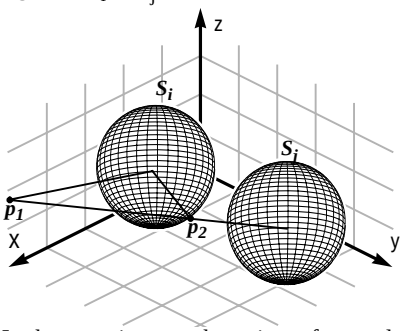
[28] Westover, L. A., "*Footprint Evaluation for Volume Rendering,*" **SIGGRAPH'90 Proceedings**, August 1990.

[29] Wolberg, G., **Digital Image Warping,** IEEE Computer Society Press, Los Alamitos, CA, 1990.

APPENDIX

We will show how occlusion compatible mappings can be determined on local spherical frames embedded within a global cartesian frame, ***W***. The projected visibility algorithm for cylindrical surfaces given in the paper can be derived by reducing it to this spherical case.

First, consider an isolated topological multiplicity on the projective mapping from $\boldsymbol{S}_i$ to $\boldsymbol{S}_j$, as shown below



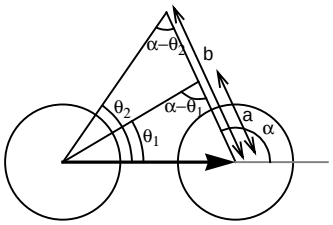
Theorem 1: In the generic case, the points of a topological multiplicity induced by a mapping from $\boldsymbol{S}_i$ to $\boldsymbol{S}_j$, and the two frame origins are coplanar.

Proof: The points of the topological multiplicity are colinear with the origin of $\boldsymbol{S}_j$ since they share angular coordinates. A second line segment connects the local frame origins, $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$. In general, these two lines are distinct and thus they define a plane in three space.

Thus, a single affine transformation, **A**, of ***W*** can accomplish the following results.

- Translate $\boldsymbol{S}_i$ to the origin
- Rotate $\boldsymbol{S}_j$ to lie on the x-axis
- Rotate the line along the multiplicity into the xy-plane
- Scale the system so that $\boldsymbol{S}_j$ has the coordinate (1,0,0).

With this transformation we can consider the multiplicity entirely within the xy-plane, as shown in the following figure.



Theorem 2: If $\cos\theta_1 > \cos\theta_2$ and $(\theta_1, \theta_2, \alpha) \in [0, \pi]$ then $a < b$.

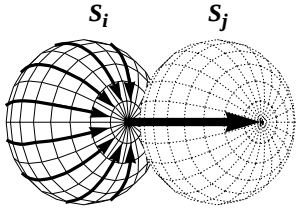
Proof: The length of sides *a* and *b* can be computed in terms of the angles θ_1, θ_2 and α using the law of sines as follows.

$$\frac{a}{\sin\theta_1} = \frac{1}{\sin(\alpha-\theta_1)} \qquad \frac{b}{\sin\theta_2} = \frac{1}{\sin(\alpha-\theta_2)}$$

$$\frac{a}{b} = \frac{\sin\alpha\cot\theta_2 - \cos\alpha}{\sin\alpha\cot\theta_1 - \cos\alpha}$$

$$\text{if } \cos\theta_1 > \cos\theta_2 \text{ then } \cot\theta_1 > \cot\theta_2, \text{ thus } a < b$$

Thus, an occlusion compatible mapping, can be determined by enumerating the topological mesh defined on $\boldsymbol{AS}_i$ in an order of increasing $\cos\theta$, while allowing later mesh facets to overwrite previous ones. This mapping is occlusion compatible since, by Theorem 2, greater range values will always proceed lesser values at all multiplicities. Notice, that this mapping procedure only considers the changes in the local frame's world coordinates, and makes no use of the range information itself.



View Morphing

Steven M. Seitz Charles R. Dyer

Department of Computer Sciences
University of Wisconsin—Madison[1]

ABSTRACT

Image morphing techniques can generate compelling 2D transitions between images. However, differences in object pose or viewpoint often cause unnatural distortions in image morphs that are difficult to correct manually. Using basic principles of projective geometry, this paper introduces a simple extension to image morphing that correctly handles 3D projective camera and scene transformations. The technique, called *view morphing*, works by prewarping two images prior to computing a morph and then postwarping the interpolated images. Because no knowledge of 3D shape is required, the technique may be applied to photographs and drawings, as well as rendered scenes. The ability to synthesize changes both in viewpoint and image structure affords a wide variety of interesting 3D effects via simple image transformations.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation– viewing algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism– animation; I.4.3 [Image Processing]: Enhancement– geometric correction, registration.

Additional Keywords: Morphing, image metamorphosis, view interpolation, view synthesis, image warping.

1 INTRODUCTION

Recently there has been a great deal of interest in *morphing* techniques for producing smooth transitions between images. These techniques combine 2D interpolations of shape and color to create dramatic special effects. Part of the appeal of morphing is that the images produced can appear strikingly lifelike and visually convincing. Despite being computed by 2D image transformations, effective morphs can suggest a natural transformation between objects in the 3D world. The fact that realistic 3D shape transformations can arise from 2D image morphs is rather surprising, but extremely useful, in that 3D shape modeling can be avoided.

Although current techniques enable the creation of effective image transitions, they do not *ensure* that the resulting transitions appear natural. It is entirely up to the user to evaluate a morph transition and to design the interpolation to achieve the best results. Part of the problem is that existing image morphing methods do not account for changes in viewpoint or object pose. As a result, simple 3D transformations (e.g., translations, rotations) become surprisingly difficult to convey convincingly using existing methods.

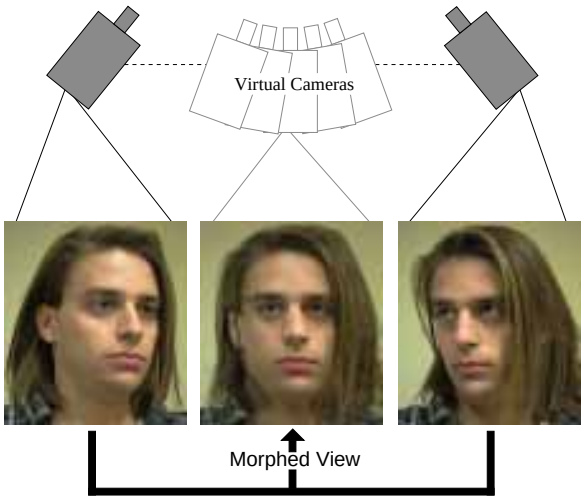


Figure 1: View morphing between two images of an object taken from two different viewpoints produces the illusion of physically moving a virtual camera.

In this paper, we describe a simple extension called *view morphing* that allows current image morphing methods to easily synthesize changes in viewpoint and other 3D effects. When morphing between different views of an object or scene, the technique produces new views of the same scene, ensuring a realistic image transition. The effect can be described by what you would see if you physically moved the object (or the camera) between its configurations in the two images and filmed the transition, as shown in Fig. 1. More generally, the approach can synthesize 3D *projective transformations* of objects, a class including 3D rotations, translations, shears, and tapering deformations, by operating entirely on images (no 3D shape information is required). Because view morphing employs existing image morphing techniques as an intermediate step, it may also be used to interpolate between different views of *different* 3D objects, combining image morphing's capacity for dramatic shape transformations with view morphing's ability to achieve changes in viewpoint. The result is a simultaneous interpolation of shape, color, and pose, giving rise to image transitions that appear strikingly 3D.

View morphing works by prewarping two images, computing a morph (image warp and cross-dissolve) between the prewarped images, and then postwarping each in-between image produced by the morph. The prewarping step is performed automatically, while the postwarping procedure may be interactively controlled by means of a small number of user-specified control points. Any of several image morphing techniques, for instance [15, 1, 8], may be used to compute the intermediate image interpolation. View morphing does

[1] 1210 W. Dayton St., Madison WI 53706
Email: {seitz | dyer}@cs.wisc.edu
Web: http://www.cs.wisc.edu/~dyer/vision.html

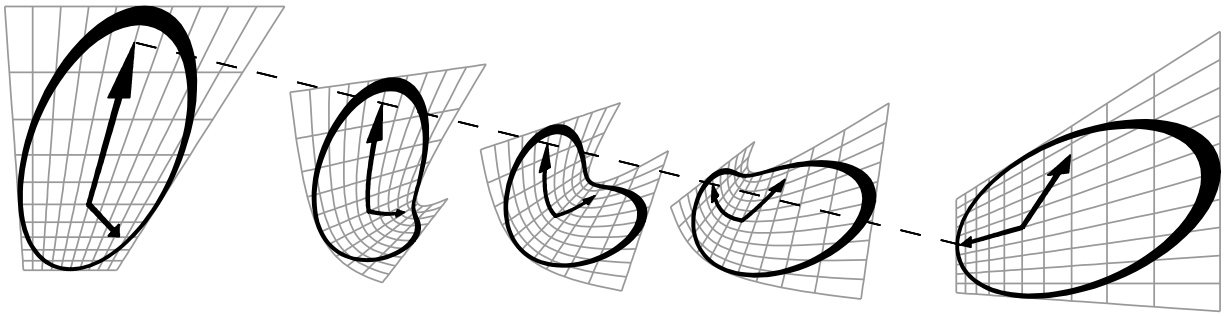


Figure 2: A Shape-Distorting Morph. Linearly interpolating two perspective views of a clock (far left and far right) causes a geometric bending effect in the in-between images. The dashed line shows the linear path of one feature during the course of the transformation. This example is indicative of the types of distortions that can arise with image morphing techniques.

not require knowledge of 3D shape, thereby allowing virtual manipulations of unknown objects or scenes given only as drawings or photographs.

In terms of its ability to achieve changes in viewpoint, view morphing is related to previous view-based techniques such as view synthesis [3, 7, 11, 12] and mosaics [10, 2, 14, 6]. However, this paper focuses on creating natural *transitions* between images rather than on synthesizing arbitrary views of an object or scene. This distinction has a number of important consequences. First, in computing the transition between two perspective views, we are free to choose a natural camera path. By choosing this path along the line connecting the two optical centers, we show that the formulation and implementation is greatly simplified. Second, our approach is general in that it can be used to compute transitions between any two images, thereby encompassing both rigid and nonrigid transformations. In contrast, previous view-based techniques have focused on rigid scenes. Finally, view morphing takes advantage of existing image morphing techniques, already in widespread use, for part of the computation. Existing image morphing tools may be easily extended to produce view morphs by adding the image prewarping and postwarping steps described in this paper.

The remainder of this paper is structured as follows: In Section 2 we review image morphing and argue that existing techniques may produce unnatural results when morphing between images of the same or similar shapes. Section 3 describes how to convert image morphing techniques into view morphing techniques by adding prewarping and postwarping steps. Section 4 extends the method to enable interpolations between views of arbitrary projective transformations of the same 3D object. In addition, interactive techniques for controlling the image transformations are introduced. We conclude with some examples in Section 5.

2 IMAGE MORPHING

Image morphing, or *metamorphosis*, is a popular class of techniques for producing transitions between images. There are a variety of morphing methods in the literature, all based on interpolating the positions and colors of pixels in two images. At present, there appears to be no universal criterion for evaluating the quality or realism of a morph, let alone of a morphing method. A natural question to ask, however, is does the method preserve 3D shape. That is, does a morph between two different views of an object produce new views of the same object? Our investigation indicates that unless special care is taken, morphing between images of similar 3D shapes often results in shapes that are mathematically quite different, leading to surprisingly complex and unnatural image transitions. These observations motivate *view morphing*, introduced in the next section, which preserves 3D shape under interpolation.

We write vectors and matrices in bold face and scalars in roman. Scene and image quantities are written in capitals and lowercase respectively. When possible, we also write corresponding image and scene quantities using the same letter. Images, $\mathcal{I}$, and 3D shapes or scenes, $\mathcal{S}$, are expressed as point sets. For example, an image point $(x, y) = \mathbf{p} \in \mathcal{I}$ is the projection of a scene point $(X, Y, Z) = \mathbf{P} \in \mathcal{S}$.

A morph is determined from two images $\mathcal{I}_0$ and $\mathcal{I}_1$ and maps $C_0 : \mathcal{I}_0 \Rightarrow \mathcal{I}_1$ and $C_1 : \mathcal{I}_1 \Rightarrow \mathcal{I}_0$ specifying a complete correspondence between points in the two images. Two maps are required because the correspondence may not be one-to-one. In practice, C_0 and C_1 are partially specified by having the user provide a sparse set of matching features or regions in the two images. The remaining correspondences are determined automatically by interpolation [15, 1, 8]. A warp function for each image is computed from the correspondence maps, usually based on linear interpolation:

$$W_0(\mathbf{p}_0, s) = (1-s)\mathbf{p}_0 + sC_0(\mathbf{p}_0) \quad (1)$$

$$W_1(\mathbf{p}_1, s) = (1-s)C_1(\mathbf{p}_1) + s\mathbf{p}_1 \quad (2)$$

W_0 and W_1 give the displacement of each point $\mathbf{p}_0 \in \mathcal{I}_0$ and $\mathbf{p}_1 \in \mathcal{I}_1$ as a function of $s \in [0, 1]$. The in-between images $\mathcal{I}_s$ are computed by warping the two original images and averaging the pixel colors of the warped images. Existing morphing methods vary principally in how the correspondence maps are computed. In addition, some techniques allow finer control over interpolation rates and methods. For instance, Beier et al. [1] suggested two different methods of interpolating line features, using linear interpolation of endpoints, per Eqs. (1) and (2), or of position and angle. In this paper, the term *image morphing* refers specifically to methods that use linear interpolation to compute feature positions in in-between images, including [15, 1, 8].

To illustrate the potentially severe 3D distortions incurred by image morphing, it is useful to consider interpolating between two different views of a planar shape. Any two such images are related by a 2D projective mapping of the form:

$$H(x, y) = \left(\frac{ax + by + c}{gx + hy + i}, \frac{dx + ey + f}{gx + hy + i}\right)$$

Projective mappings are not preserved under 2D linear interpolation since the sum of two such expressions is in general a ratio of quadratics and therefore not a projective mapping. Consequently, morphing is a *shape-distorting* transformation, as in-between images may not correspond to new views of the same shape. A particularly disturbing effect of image morphing is its tendency to bend straight lines, yielding quite unintuitive image transitions. Fig. 2 shows a Dali-esque morph between two views of a clock in which it appears to bend in half and then straighten out again during the

course of the transition. The in-between shapes were computed by linearly interpolating points in the two views that correspond to the same point on the clock.

3 VIEW MORPHING

In the previous section we argued that unless special care is taken, image interpolations do not convey 3D rigid shape transformations. We say that an image transformation is *shape-preserving* if from two images of a particular object, it produces a new image representing a view of the same object. In this section we describe an interpolation-based image morphing procedure that is shape-preserving. Morphs generated by this technique create the illusion that the object moves rigidly (rotating and translating in 3D) between its positions in the two images.

Computing the morph requires the following: (1) two images $\mathcal{I}_0$ and $\mathcal{I}_1$, representing views of the same 3D object or scene, (2) their respective projection matrices $\mathbf{\Pi}_0$ and $\mathbf{\Pi}_1$, and (3) a correspondence between pixels in the two images. Note that no a priori knowledge of 3D shape information is needed. The requirement that projection matrices be known differentiates this technique from previous morphing methods. However, there exist a variety of techniques for obtaining the projection matrices from the images themselves and knowledge of either the internal camera parameters or the 3D positions of a small number of image points. For an overview of both types of techniques, consult [4]. In Section 4 we introduce a variant that does not require knowledge of the projection matrices and also allows interpolations between views of *different* 3D objects or scenes.

The pixel correspondences are derived by a combination of user-interaction and automatic interpolation provided by existing morphing techniques. When the correspondence is correct, the methods described in this section guarantee shape-preserving morphs. In practice, we have found that an approximate correspondence is often sufficient to produce transitions that are visually convincing. Major errors in correspondence may result in visible artifacts such as "ghosting" and shape distortions. Some examples of these effects are shown in Section 5. Other errors may occur as a result of changes in visibility. In order to completely infer the appearance of a surface from a new viewpoint, that surface must be visible in both $\mathcal{I}_0$ and $\mathcal{I}_1$. Changes in visibility may result in *folds* or *holes*, as discussed in Section 3.4.

Following convention, we represent image and scene quantities using homogeneous coordinates: a scene point with Euclidean coordinates (X, Y, Z) is expressed by the column vector $\mathbf{P} = [X\ Y\ Z\ 1]^T$ and a Euclidean image point (x, y) by $\mathbf{p} = [x\ y\ 1]^T$. We reserve the notation $\mathbf{P}$ and $\mathbf{p}$ for points expressed in Euclidean coordinates, i.e., whose last coordinate is 1. Scalar multiples of these points will be written with a tilde, as $\tilde{\mathbf{P}}$ and $\tilde{\mathbf{p}}$. A camera is represented by a 3×4 homogeneous projection matrix of the form $\mathbf{\Pi} = [\mathbf{H} \mid -\mathbf{HC}]$. The vector $\mathbf{C}$ gives the Euclidean position of the camera's optical center and the 3×3 matrix $\mathbf{H}$ specifies the position and orientation of its image plane with respect to the world coordinate system. The perspective projection equation is

$$\tilde{\mathbf{p}} = \mathbf{\Pi}\mathbf{P} \tag{3}$$

The term *view* will henceforth refer to the tuple $\langle \mathcal{I}, \mathbf{\Pi} \rangle$ comprised of an image and its associated projection matrix.

3.1 Parallel Views

We begin by considering situations in which linear interpolation of images is shape-preserving. Suppose we take a photograph $\mathcal{I}_0$ of an object, move the object in a direction parallel to the image plane of the camera, zoom out, and take a second picture $\mathcal{I}_1$, as shown in Fig. 3. Alternatively, we could produce the same two images by moving the camera instead of the object. Chen and Williams [3] previously considered this special case, arguing that linear image interpolation should produce new perspective views when the camera moves parallel to the image plane. Indeed, suppose that the camera is moved from the world origin to position $(C_X, C_Y, 0)$ and the focal length changes from f_0 to f_1. We write the respective projection matrices, $\mathbf{\Pi}_0$ and $\mathbf{\Pi}_1$, as:

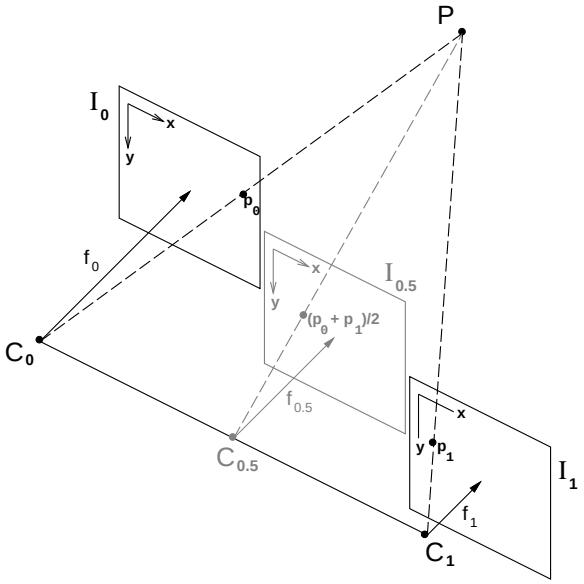


Figure 3: Morphing Parallel Views. Linear interpolation of corresponding pixels in parallel views with image planes $\mathcal{I}_0$ and $\mathcal{I}_1$ creates image $\mathcal{I}_{0.5}$, representing another parallel view of the same scene.

$$\mathbf{\Pi}_0 = \begin{bmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{\Pi}_1 = \begin{bmatrix} f_1 & 0 & 0 & -f_1 C_X \\ 0 & f_1 & 0 & -f_1 C_Y \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

We refer to cameras or views with projection matrices in this form as *parallel cameras* or *parallel views*, respectively. Let $\mathbf{p}_0 \in \mathcal{I}_0$ and $\mathbf{p}_1 \in \mathcal{I}_1$ be projections of a scene point $\mathbf{P} = [X\ Y\ Z\ 1]^T$. Linear interpolation of $\mathbf{p}_0$ and $\mathbf{p}_1$ yields

$$\begin{aligned} (1-s)\mathbf{p}_0 + s\mathbf{p}_1 &= (1-s)\frac{1}{Z}\mathbf{\Pi}_0\mathbf{P} + s\frac{1}{Z}\mathbf{\Pi}_1\mathbf{P} \\ &= \frac{1}{Z}\mathbf{\Pi}_s\mathbf{P} \end{aligned} \tag{4}$$

where

$$\mathbf{\Pi}_s = (1-s)\mathbf{\Pi}_0 + s\mathbf{\Pi}_1 \tag{5}$$

Image interpolation therefore produces a new view whose projection matrix, $\mathbf{\Pi}_s$, is a linear interpolation of $\mathbf{\Pi}_0$ and $\mathbf{\Pi}_1$, representing a camera with center $\mathbf{C}_s$ and focal length f_s given by:

$$\mathbf{C}_s = (sC_X, sC_Y, 0) \tag{6}$$

$$f_s = (1-s)f_0 + sf_1 \tag{7}$$

Consequently, interpolating images produced from parallel cameras produces the illusion of simultaneously moving the camera on the

line $\overline{\mathbf{C}_0\mathbf{C}_1}$ between the two optical centers and zooming continuously. Because the image interpolation produces new views of the same object, it is shape-preserving.

In fact, the above derivation relies only on the equality of the third rows of $\mathbf{\Pi}_0$ and $\mathbf{\Pi}_1$. Views satisfying this more general criterion represent a broader class of parallel views for which linear image interpolation is shape preserving. An interesting special case is the class of orthographic projections, i.e., projections $\mathbf{\Pi}_0$ and $\mathbf{\Pi}_1$ whose last row is $[0\ 0\ 0\ 1]$. Linear interpolation of any two orthographic views of a scene therefore produces a new orthographic view of the same scene.

3.2 Non-Parallel Views

In this section we describe how to generate a sequence of in-between views from two non-parallel perspective images of the same 3D object or scene. For convenience, we choose to model the transformation as a change in viewpoint, as opposed to a rotation and translation of the object or scene. The only tools used are image reprojection and linear interpolation, both of which may be performed using efficient scanline methods.

3.2.1 Image Reprojection

Any two views that share the same optical center are related by a planar projective transformation. Let $\mathcal{I}$ and $\hat{\mathcal{I}}$ be two images with projection matrices $\mathbf{\Pi} = [\mathbf{H} \mid -\mathbf{HC}]$ and $\hat{\mathbf{\Pi}} = [\hat{\mathbf{H}} \mid -\hat{\mathbf{H}}\mathbf{C}]$. The projections $\tilde{\mathbf{p}} \in \mathcal{I}$ and $\tilde{\hat{\mathbf{p}}} \in \hat{\mathcal{I}}$ of any scene point $\mathbf{P}$ are related by the following transformation:

$$\begin{aligned}\hat{\mathbf{H}}\mathbf{H}^{-1}\tilde{\mathbf{p}} &= \hat{\mathbf{H}}\mathbf{H}^{-1}\mathbf{\Pi}\mathbf{P} \\ &= \hat{\mathbf{\Pi}}\mathbf{P} \\ &= \tilde{\hat{\mathbf{p}}}\end{aligned}$$

The 3×3 matrix $\hat{\mathbf{H}}\mathbf{H}^{-1}$ is a projective transformation that reprojects the image plane of $\mathcal{I}$ onto that of $\hat{\mathcal{I}}$. More generally, any invertible 3×3 matrix represents a planar projective transformation, a one-to-one map of the plane that transforms points to points and lines to lines. The operation of reprojection is very powerful because it allows the gaze direction to be modified *after* a photograph is taken, or a scene rendered. Our use of projective transforms to compute reprojections takes advantage of an efficient scanline algorithm [15]. Reprojection can also be performed through texture-mapping and can therefore exploit current graphics hardware.

Image reprojection has been used previously in a number of applications [15]. Our use of reprojection is most closely related to the techniques used for rectifying stereo views to simplify 3D shape reconstruction [4]. Image mosaic techniques [10, 2, 14, 6] also rely heavily on reprojection methods to project images onto a planar, cylindrical, or spherical manifold. In the next section we describe how reprojection may be used to improve image morphs.

3.2.2 A Three Step Algorithm

Using reprojection, the problem of computing a shape-preserving morph from two non-parallel perspective views can be reduced to the case treated in Section 3.1. To this end, let $\mathcal{I}_0$ and $\mathcal{I}_1$ be two perspective views with projection matrices $\mathbf{\Pi}_0 = [\mathbf{H}_0 \mid -\mathbf{H}_0\mathbf{C}_0]$ and $\mathbf{\Pi}_1 = [\mathbf{H}_1 \mid -\mathbf{H}_1\mathbf{C}_1]$. It is convenient to choose the world coordinate system so that both $\mathbf{C}_0$ and $\mathbf{C}_1$ lie on the world X-axis, i.e., $\mathbf{C}_0 = [X_0\ 0\ 0]^T$ and $\mathbf{C}_1 = [X_1\ 0\ 0]^T$. The two remaining axes should be chosen in a way that reduces the distortion incurred by image reprojection. A simple choice that works well in practice is to choose the Y axis in the direction of the cross product of the two image plane normals.

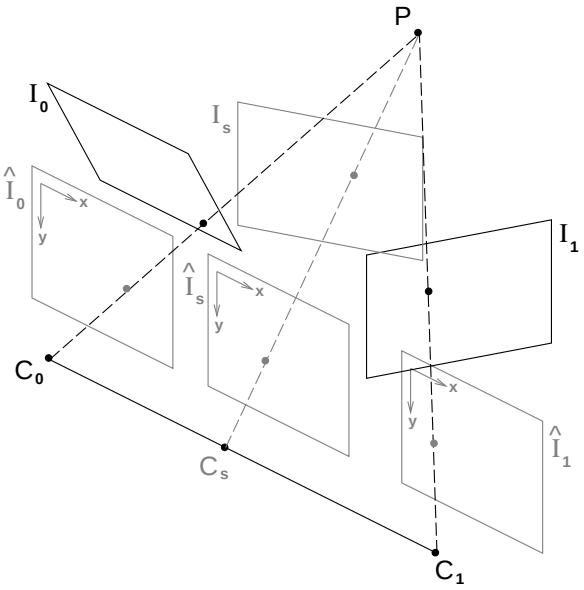


Figure 4: View Morphing in Three Steps. (1) Original images $\mathcal{I}_0$ and $\mathcal{I}_1$ are prewarped to form parallel views $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$. (2) $\hat{\mathcal{I}}_s$ is produced by morphing (interpolating) the prewarped images. (3) $\hat{\mathcal{I}}_s$ is postwarped to form $\mathcal{I}_s$.

In between perspective views on the line $\overline{\mathbf{C}_0\mathbf{C}_1}$ may be synthesized by a combination of image reprojections and interpolations, depicted in Fig. 4. Given a projection matrix $\mathbf{\Pi}_s = [\mathbf{H}_s \mid -\mathbf{H}_s\mathbf{C}_s]$, with $\mathbf{C}_s$ fixed by Eq. (6), the following sequence of operations produces an image $\mathcal{I}_s$ corresponding to a view with projection matrix $\mathbf{\Pi}_s$:

1. **Prewarp:** apply projective transforms $\mathbf{H}_0^{-1}$ to $\mathcal{I}_0$ and $\mathbf{H}_1^{-1}$ to $\mathcal{I}_1$, producing prewarped images $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$
2. **Morph:** form $\hat{\mathcal{I}}_s$ by linearly interpolating positions and colors of corresponding points in $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$, using Eq. (4) or any image morphing technique that approximates it
3. **Postwarp:** apply $\mathbf{H}_s$ to $\hat{\mathcal{I}}_s$, yielding image $\mathcal{I}_s$

Prewarping brings the image planes into alignment without changing the optical centers of the two cameras. Morphing the prewarped images moves the optical center to $\mathbf{C}_s$. Postwarping transforms the image plane of the new view to its desired position and orientation.

Notice that the prewarped images $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$ represent views with projection matrices $\hat{\mathbf{\Pi}}_0 = [\mathbf{I} \mid -\mathbf{C}_0]$ and $\hat{\mathbf{\Pi}}_1 = [\mathbf{I} \mid -\mathbf{C}_1]$, where $\mathbf{I}$ is the 3×3 identity matrix. Due to the special form of these projection matrices, $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$ have the property that corresponding points in the two images appear in the same scanline. Therefore, the interpolation $\hat{\mathcal{I}}_s$ may be computed one scanline at a time using only 1D warping and resampling operations.

The prewarping and postwarping operations, combined with the intermediate morph, require multiple image resampling operations that may contribute to a noticeable blurring in the in-between images. Resampling effects can be reduced by supersampling the input images [15] or by composing the image transformations into one aggregate warp for each image. The latter approach is especially compatible with image morphing techniques that employ inverse mapping, such as the Beier and Neely method [1], since the inverse postwarp, morph, and prewarp can be directly concatenated into a single inverse map. Composing the warps has disadvantages however,

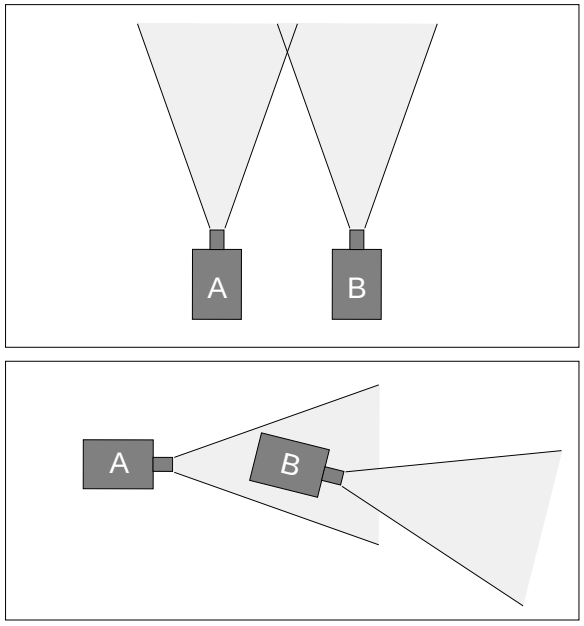


Figure 5: Singular Views. In the parallel configuration (top), each camera's optical center is out of the field of view of the other. A singular configuration (bottom) arises when the optical center of camera B is in the field of view of camera A. Because prewarping does not change the field of view, singular views cannot be reprojected to form parallel views.

including loss of both the scanline property and the ability to use off-the-shelf image morphing tools to compute the intermediate interpolation.

3.3 Singular View Configurations

Certain configurations of views cannot be made parallel through reprojection operations. For parallel cameras, (Fig. 5, top) the optical center of neither camera is within the field of view of the other. Note that reprojection does not change a camera's field of view, only its viewing direction. Therefore any pair of views for which the optical center of one camera is within the field of view of the other cannot be made parallel through prewarping[1]. Fig. 5 (bottom) depicts such a pair of *singular* views, for which the prewarping procedure fails. Singular configurations arise when the camera motion is roughly parallel to the viewing direction, a condition detectable from the images themselves (see the Appendix). Singular views are not a problem when the prewarp, morph, and postwarp are composed into a single aggregate warp, since prewarped images are never explicitly constructed. With aggregate warps, view morphing may be applied to arbitrary pairs of views, including singular views.

3.4 Changes in Visibility

So far, we have described how to correct for distortions in image morphs by manipulating the projection equations. Eq. (3), however, does not model the effects that changes in *visibility* have on image content. From the standpoint of morphing, changes in visibility result in two types of conditions: *folds* and *holes*. A fold occurs in an in-between image $\mathcal{I}_s$ when a visible surface in $\mathcal{I}_0$ (or $\mathcal{I}_1$) becomes occluded in $\mathcal{I}_s$. In this situation, multiple pixels of $\mathcal{I}_0$ map to the same point in $\mathcal{I}_s$, causing an ambiguity. The opposite case, of an occluded surface suddenly becoming visible, gives rise to a hole; a region of $\mathcal{I}_s$ having no correspondence in $\mathcal{I}_0$.

Folds can be resolved using Z-buffer techniques [3], provided depth information is available. In the absence of 3D shape information, we use point *disparity* instead. The disparity of corresponding points $\mathbf{p}_0$ and $\mathbf{p}_1$ in two parallel views is defined to be the difference of their x-coordinates. For parallel views, point disparity is inversely proportional to depth so that Z-buffer techniques may be directly applied, with inverse disparity substituted for depth. Because our technique makes images parallel prior to interpolation, this simple strategy suffices in general. Furthermore, since the interpolation is computed one scanline at a time, Z-buffering may be performed at the scanline level, thereby avoiding the large memory requirements commonly associated with Z-buffering algorithms. An alternative method using a Painter's method instead of Z-buffering is presented in [10].

Unlike folds, holes cannot always be eliminated using image information alone. Chen and Williams [3] suggested different methods for filling holes, using a designated background color, interpolation with neighboring pixels, or additional images for better surface coverage. The neighborhood interpolation approach is prevalent in existing image morphing methods and was used implicitly in our experiments.

3.4.1 Producing the Morph

Producing a shape-preserving morph between two images requires choosing a sequence of projection matrices $\mathbf{\Pi}_s = [\mathbf{H}_s \mid -\mathbf{H}_s\mathbf{C}_s]$, beginning with $\mathbf{\Pi}_0$ and ending with $\mathbf{\Pi}_1$. Since $\mathbf{C}_s$ is determined by Eq. (6), this task reduces to choosing $\mathbf{H}_s$ for each value of $s \in (0, 1)$, specifying a continuous transformation of the image plane from the first view to the second.

There are many ways to specify this transformation. A natural one is to interpolate the orientations of the image planes by a single axis rotation. If the image plane normals are denoted by 3D unit vectors $\mathbf{N}_0$ and $\mathbf{N}_1$, the axis $\mathbf{D}$ and angle of rotation θ are given by

$$\begin{aligned} \mathbf{D} &= \mathbf{N}_0 \times \mathbf{N}_1 \\ \theta &= cos^{-1}(\mathbf{N}_0 \cdot \mathbf{N}_1) \end{aligned}$$

Alternatively, if the orientations are expressed using quaternions, the interpolation is computed by spherical linear interpolation [13]. In either case, camera parameters such as focal length and aspect ratio should be interpolated separately.

4 PROJECTIVE TRANSFORMATIONS

By generalizing what we mean by a "view", the technique described in the previous section can be extended to accommodate a range of 3D shape deformations. In particular, view morphing can be used to interpolate between images of different 3D *projective* transformations of the same object, generating new images of the same object, projectively transformed. The advantage of using view morphing in this context is that salient features such as lines and conics are preserved during the course of the transformation from the first image to the second. In contrast, straightforward image morphing can cause severe geometric distortions, as seen in Fig. 2.

As described in Section 3.1, a 2D projective transformation may be expressed as a 3×3 homogeneous matrix transformation. Similarly, a 3D projective transformation is given by a 4×4 matrix $\mathbf{T}$. This class of transformations encompasses 3D rotations, translations, scales, shears, and tapering deformations. Applying $\mathbf{T}$ to a

[1] Prewarping is possible if the images are first cropped to exclude the epipoles (see the Appendix).

homogeneous scene point produces the point $\tilde{\mathbf{Q}} = \mathbf{TP}$. The corresponding point $\mathbf{Q}$ in 3D Euclidean coordinates is obtained by dividing $\tilde{\mathbf{Q}}$ by its fourth component. 3D projective transformations are notable in that they may be "absorbed" by the camera transformation. Specifically, consider rendering an image of a scene that has been transformed by a 3D projective transformation $\mathbf{T}$. If the projection matrix is given by $\mathbf{\Pi}$, a point $\mathbf{P}$ in the scene appears at position $\mathbf{p}$ in the image, where $\tilde{\mathbf{p}} = \mathbf{\Pi}(\mathbf{TP})$. If we define the 3×4 matrix $\tilde{\mathbf{\Pi}} = \mathbf{\Pi T}$, the combined transformation may be expressed as a single projection, representing a view with projection matrix $\tilde{\mathbf{\Pi}}$.

By allowing arbitrary 3×4 projections, we can model the changes in shape induced by projective transformations by changes in *viewpoint*. In doing so, the problem of interpolating images of projective transformations of an unknown shape is reduced to a form to which the three-step algorithm of Section 3.2.2 may be applied. However, recall that the three-step algorithm requires that the camera viewpoints be known. In order to morph between two different faces, this would require *a priori* knowledge of the 3D projective transformation that best relates them. Since this knowledge may be difficult to obtain, we describe here a modification that doesn't require knowing the projection matrices.

Suppose we wish to smoothly interpolate two images $\mathcal{I}_0$ and $\mathcal{I}_1$ of objects related by a 3D projective transformation. Suppose further that only the images themselves and pixel correspondences are provided. In order to ensure that in-between images depict the same 3D shape (projectively transformed), $\mathcal{I}_0$ and $\mathcal{I}_1$ must first be transformed so as to represent parallel views. As explained in Section 3.2.2, the transformed images, $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$, have the property that corresponding points appear in the same scanline of each image, i.e., two points $\mathbf{p}_0 \in \hat{\mathcal{I}}_0$ and $\mathbf{p}_1 \in \hat{\mathcal{I}}_1$ are projections of the same scene point only if their y-coordinates are equal. In fact, this condition is sufficient to ensure that two views are parallel. Consequently $\mathcal{I}_0$ and $\mathcal{I}_1$ may be made parallel by finding any pair of 2D projective transformations $\mathbf{H}_0$ and $\mathbf{H}_1$ that send corresponding points to the same scanline. One approach for determining $\mathbf{H}_0$ and $\mathbf{H}_1$ using 8 or more image point correspondences is given in the Appendix.

4.1 Controlling the Morph

To fully determine a view morph, $\mathbf{H}_s$ must be provided for each in-between image. Rather than specifying the 3×3 matrix explicitly, it is convenient to provide $\mathbf{H}_s$ indirectly by establishing constraints on the in-between images. A simple yet powerful way of doing this is to interactively specify the paths of four image points through the entire morph transition. These control points can represent the positions of four point features, the endpoints of two lines, or the bounding quadrilateral of an arbitrary image region[2]. Fig. 6 illustrates the process: first, four control points bounding a quadrilateral region of $\hat{\mathcal{I}}_{0.5}$ are selected, determining corresponding quadrilaterals in $\mathcal{I}_0$ and $\mathcal{I}_1$. Second, the control points are interactively moved to their desired positions in $\mathcal{I}_{0.5}$, implicitly specifying the postwarp transformation and thus determining the entire image $\mathcal{I}_{0.5}$. The postwarps of other in-between images are then determined by interpolating the control points. The positions of the control points in $\mathcal{I}_s$ and $\hat{\mathcal{I}}_s$ specify a linear system of equations whose solution yields $\mathbf{H}_s$ [15]. The four curves traced out by the control points may also be manually edited for finer control of the interpolation parameters.

The use of image control points bears resemblance to the view synthesis work of Laveau and Faugeras [7], who used five pairs of corresponding image points to specify projection parameters. However, in their case, the points represented the projection of a new image plane and optical center and were specified only in the original images. In our approach, the control points are specified in the *in-between* image(s), providing more direct control over image appearance.

[2] Care should be taken to ensure that no three of the control points are colinear in any image.

4.2 View Morphing Without Prewarping

Prewarping is less effective for morphs between different objects not closely related by a 3D projective transform. With objects that are considerably different, it is advisable to leave out the prewarp entirely, since its automatic computation becomes less stable [9]. The postwarp step should not be omitted, however, since it can be used to reduce image plane distortions for more natural morphs. For instance, a large change in orientation results in a noticeable 2D image contraction, as seen in Fig. 10.

Prewarping is not strictly necessary for images that are approximately orthographic, as noted in Section 3.1. Images taken with a telephoto lens often fall into this category, as do images of objects whose variation in depth is small relative to their distance from the camera. In either case, the images may be morphed directly, yielding new orthographic views. However, the prewarping step does influence the camera motion which, in the orthographic case, cannot be controlled solely by postwarping. The camera transformation determined by Eq. (5) may introduce unnatural skews and twists of the image plane due to the fact that linear matrix interpolation does not preserve row orthogonality. Prewarping the images ensures that the view plane undergoes a single axis rotation. More details on the orthographic case are given in [12].

5 RESULTS

Fig. 6 illustrates the view morphing procedure applied to two images of a bus. We manually selected a set of about 50 corresponding line features in the two images. These features were used to automatically prewarp the images to achieve parallelism using the method described in the Appendix. Inspection of the prewarped images confirms that corresponding features do in fact occupy the same scanlines. An implementation of the Beier-Neely field-morphing algorithm [1] was used to compute the intermediate images, based on the same set of features used to prewarp the images. The resulting images were postwarped by selecting a quadrilateral region delimited by four control points in $\hat{\mathcal{I}}_{0.5}$ and moving the control points to their desired positions in $\mathcal{I}_{0.5}$. The final positions of the control points for the image in the center of Fig. 6 were computed automatically by roughly calibrating the two images based on their known focal lengths and interpolating the changes in orientation [4]. Different images obtained by other settings of the control points are shown in Fig. 8. As these images indicate, a broad range of 3D projective effects may be achieved through the postwarping procedure. For instance, the rectangular shape of the bus can be skewed in different directions and tapered to depict different 3D shapes.

Fig. 7 shows some results on interpolating human faces in varying poses. The first example shows selected frames from a morph computed by interpolating views of the same person facing in two different directions. The resulting animation depicts the subject continuously turning his head from right to left. Because the subject's right ear is visible in only one of the original images, it appears "ghosted" in intermediate frames due to the interpolation of intensity values. In addition, the subject's nose appears slightly distorted as a result of similar changes in visibility. The second sequence shows a morph between different views of two *different faces*. Interpolating different faces is one of the most popular applications of image morphing. Here, we combine image morphing's capacity for dramatic facial interpolations with view morphing's ability to achieve changes in viewpoint. The result is a simultaneous interpolation of facial structure, color, and pose, giving rise to an image transition conveying a metamorphosis that appears strikingly 3D.

When an object has bilateral symmetry, view morphs can be computed from a single image. Fig. 9 depicts a view morph between an image of Leonardo da Vinci's *Mona Lisa* and its mirror reflection. Although the two sides of the face and torso are not perfectly symmetric, the morph conveys a convincing facial rotation.

Fig. 10 compares image morphing with view morphing using two ray-traced images of a helicopter toy. The image morph was computed by linearly interpolating corresponding pixels in the two original images. The change in orientation between the original images caused the in-between images to contract. In addition, the bending effects seen in Fig. 2 are also present. Image morphing techniques such as [1] that preserve lines can reduce bending effects, but only when line features are present. An interesting side-effect is that a large hole appears in the image morph, between the stick and propeller, but not in the view morph, since the eye-level is constant throughout the transition. To be sure, view morphs may also produce holes, but only as a result of a change in visibility.

6 CONCLUSIONS

Achieving realistic image transitions is possible but often difficult with existing image morphing techniques due to the lack of available 3D information. In this paper, we demonstrated how to accurately convey a range of 3D transformations based on a simple yet powerful extension to the image morphing paradigm called *view morphing*. In addition to changes in viewpoint, view morphing accommodates changes in projective shape. By integrating these capabilities with those already afforded by existing image morphing methods, view morphing enables transitions between images of different objects that give a strong sense of metamorphosis in 3D. Because no knowledge of 3D shape is required, the technique may be applied to photographs and drawings, as well as to artificially rendered scenes. Two different methods for controlling the image transition were described, using either automatic interpolation of camera parameters or interactive user-manipulation of image control points, based on whether or not the camera viewpoints are known.

Because view morphing relies exclusively on image information, it is sensitive to changes in visibility. In our experiments, the best morphs resulted when visibility was nearly constant, i.e., most surfaces were visible in both images. The visible effects of occlusions may often be minimized by experimenting with different feature correspondences. Additional user input could be used to reduce ghosting effects by specifying the paths of image regions visible in only one of the original images. A topic of future work will be to investigate ways of extending view morphing to handle extreme changes in visibility, enabling 180 or 360 degree rotations in depth.

ACKNOWLEDGEMENTS

The authors would like to thank Paul Heckbert for providing the image morphing code used to interpolate prewarped images in this paper. The original helicopter images in Fig. 10 were rendered using Rayshade with a Z-buffer output extension written by Mark Maimone. The support of the National Science Foundation under Grant Nos. IRI–9220782 and CDA–9222948 is gratefully acknowledged.

REFERENCES

[1] BEIER, T., AND NEELY, S. Feature-based image metamorphosis. Proc. SIGGRAPH 92. In *Computer Graphics* (1992), pp. 35–42.

[2] CHEN, S. E. Quicktime VR — An image-based approach to virtual environment navigation. Proc. SIGGRAPH 95. In *Computer Graphics* (1995), pp. 29–38.

[3] CHEN, S. E., AND WILLIAMS, L. View interpolation for image synthesis. Proc. SIGGRAPH 93. In *Computer Graphics* (1993), pp. 279–288.

[4] FAUGERAS, O. *Three-Dimensional Computer Vision, A Geometric Viewpoint*. MIT Press, Cambridge, MA, 1993.

[5] HARTLEY, R. I. In defence of the 8-point algorithm. In *Proc. Fifth Intl. Conference on Computer Vision* (1995), pp. 1064–1070.

[6] KUMAR, R., ANANDAN, P., IRANI, M., BERGEN, J., AND HANNA, K. Representation of scenes from collections of images. In *Proc. IEEE Workshop on Representations of Visual Scenes* (1995), pp. 10–17.

[7] LAVEAU, S., AND FAUGERAS, O. 3-D scene representation as a collection of images. In *Proc. International Conference on Pattern Recognition* (1994), pp. 689–691.

[8] LEE, S.-Y., CHWA, K.-Y., SHIN, S. Y., AND WOLBERG, G. Image metamorphosis using snakes and free-form deformations. Proc. SIGGRAPH 92. In *Computer Graphics* (1992), pp. 439–448.

[9] LUONG, Q.-T., AND FAUGERAS, O. The fundamental matrix: Theory, algorithms, and stability analysis. *Intl. Journal of Computer Vision 17*, 1 (1996), 43–75.

[10] MCMILLAN, L., AND BISHOP, G. Plenoptic modeling. Proc. SIGGRAPH 95. In *Computer Graphics* (1995), pp. 39–46.

[11] POGGIO, T., AND BEYMER, D. Learning networks for face analysis and synthesis. In *Proc. Intl. Workshop on Automatic Face- and Gesture-Recognition* (Zurich, 1995), pp. 160–165.

[12] SEITZ, S. M., AND DYER, C. R. Physically-valid view synthesis by image interpolation. In *Proc. IEEE Workshop on Representations of Visual Scenes* (1995), pp. 18–25.

[13] SHOEMAKE, K. Animating rotation with quaternion curves. Proc. SIGGRAPH 85. In *Computer Graphics* (1985), pp. 245–254.

[14] SZELISKI, R. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications 16*, 2 (1996), 22–30.

[15] WOLBERG, G. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, 1990.

APPENDIX

This appendix describes how to automatically compute the image prewarping transforms $\mathbf{H}_0$ and $\mathbf{H}_1$ from the images themselves. We assume that the 2D positions of 8 or more corresponding points are given in each image. The fundamental matrix of two views is defined to be the 3×3, rank-two matrix $\mathbf{F}$ such that for every pair of corresponding image points $\mathbf{p}_0 \in \mathcal{I}_0$ and $\mathbf{p}_1 \in \mathcal{I}_1$,

$$\mathbf{p}_1^T \mathbf{F} \mathbf{p}_0 = 0$$

$\mathbf{F}$ is defined up to a scale factor and can be computed from 8 or more such points using linear [5] or non-linear [9] methods.

A sufficient condition for two views to be parallel is that their fundamental matrix have the form:

$$\hat{\mathbf{F}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{8}$$

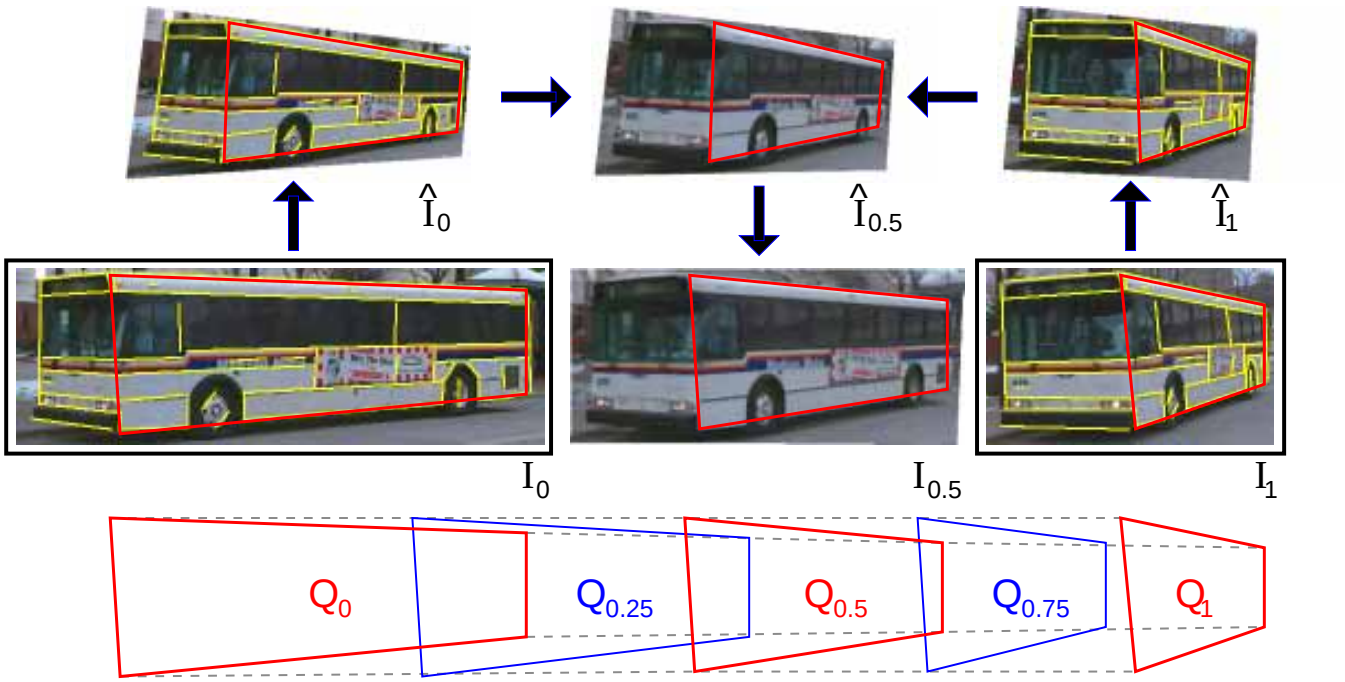


Figure 6: View Morphing Procedure: A set of features (yellow lines) is selected in original images $\mathcal{I}_0$ and $\mathcal{I}_1$. Using these features, the images are automatically prewarped to produce $\hat{\mathcal{I}}_0$ and $\hat{\mathcal{I}}_1$. The prewarped images are morphed to create a sequence of in-between images, the middle of which, $\hat{\mathcal{I}}_{0.5}$, is shown at top-center. $\hat{\mathcal{I}}_{0.5}$ is interactively postwarped by selecting a quadrilateral region (marked red) and specifying its desired configuration, $Q_{0.5}$, in $\mathcal{I}_{0.5}$. The postwarps for other in-between images are determined by interpolating the quadrilaterals (bottom).

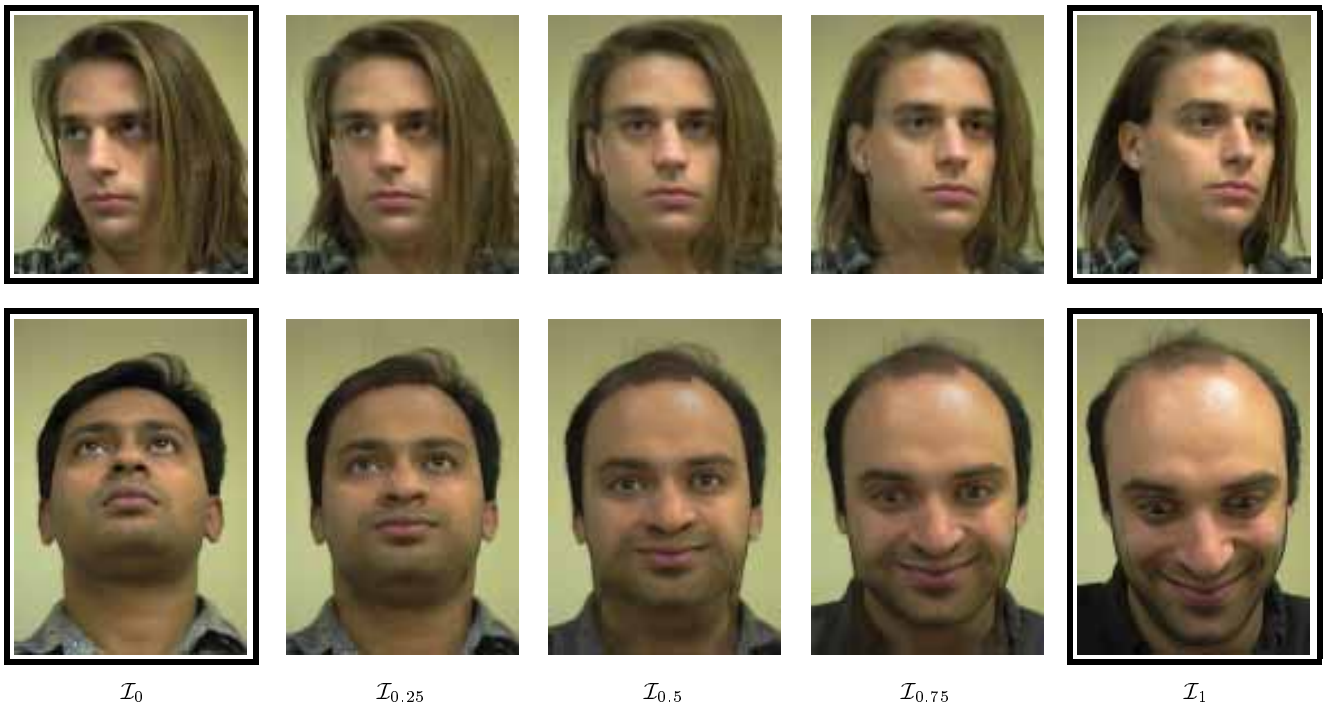


Figure 7: Facial View Morphs. Top: morph between two views of the same person. Bottom: morph between views of two different people. In each case, view morphing captures the change in facial pose between original images $\mathcal{I}_0$ and $\mathcal{I}_1$, conveying a natural 3D rotation.

Figure 8: Postwarping deformations obtained by different settings of the control quadrilateral.

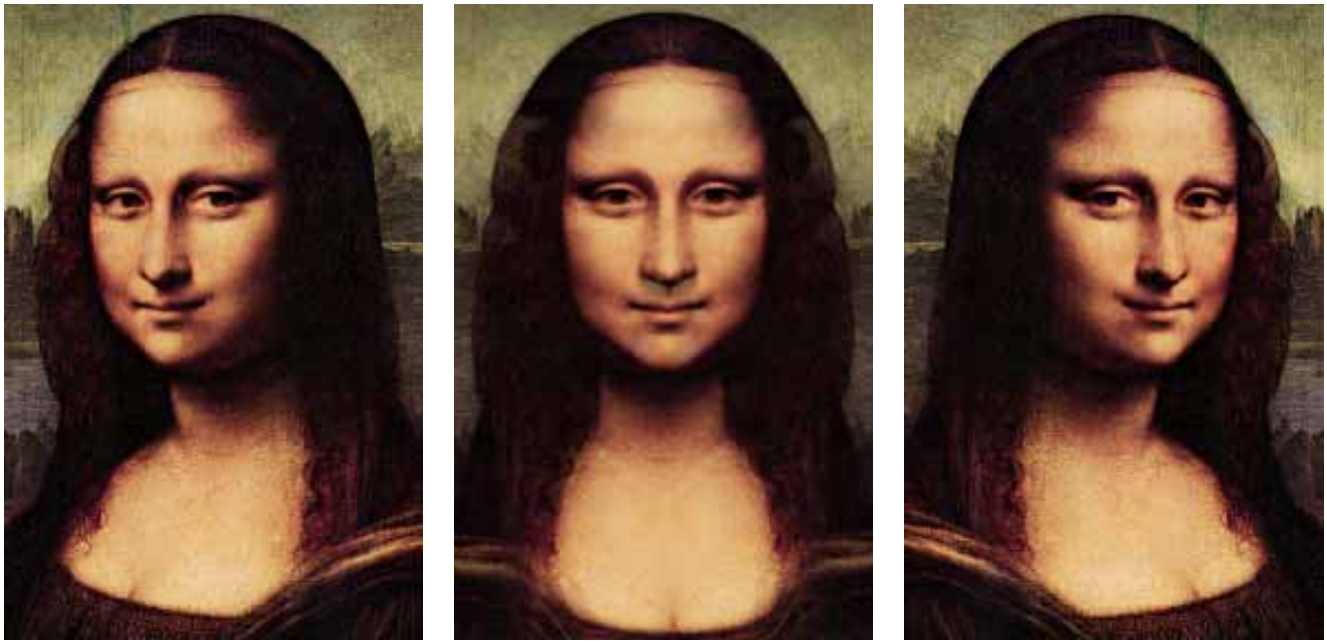


Figure 9: Mona Lisa View Morph. Morphed view (center) is halfway between original image (left) and it's reflection (right).

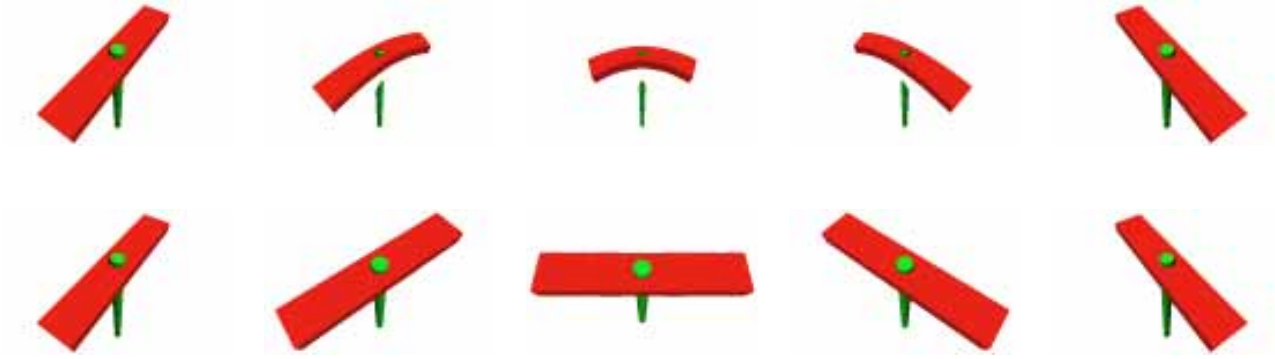


Figure 10: Image Morphing Versus View Morphing. Top: image morph between two views of a helicopter toy causes the in-between images to contract and bend. Bottom: view morph between the same two views results in a physically consistent morph. In this example the image morph also results in an extraneous hole between the blade and the stick. Holes can appear in view morphs as well.

Consequently, any two images with fundamental matrix $\mathbf{F}$ may be prewarped (i.e., made parallel) by choosing any two projective transforms $\mathbf{H}_0$ and $\mathbf{H}_1$ such that $\left(\mathbf{H}_1^{-1}\right)^T \mathbf{F} \mathbf{H}_0^{-1} = \hat{\mathbf{F}}$. Here we describe one method that applies a rotation in depth to make the images planes parallel, followed by an affine transformation to align corresponding scanlines. The procedure is determined by choosing an (arbitrary) axis of rotation $\mathbf{d}_0 = [d_0^x \; d_0^y \; 0]^T \in \mathcal{I}_0$. Given $[x \; y \; z]^T = \mathbf{F}\mathbf{d}_0$, the corresponding axis in $\mathcal{I}_1$ is determined according to $\mathbf{d}_1 = [-y \; x \; 0]^T$. To compute the angles of depth rotation we need the *epipoles*, also known as *vanishing points*, $\mathbf{e}_0 \in \mathcal{I}_0$ and $\mathbf{e}_1 \in \mathcal{I}_1$. $\mathbf{e}_0 = [e_0^x \; e_0^y \; e_0^z]^T$ and $\mathbf{e}_1 = [e_1^x \; e_1^y \; e_1^z]^T$ are the unit eigenvectors of $\mathbf{F}$ and $\mathbf{F}^T$ respectively, corresponding to eigenvalues of 0. A view's epipole represents the projection of the optical center of the other view. The following procedure will work provided the views are not *singular*, i.e., the epipoles are outside the image borders and therefore not within the field of view. The angles of rotation in depth about $\mathbf{d}_i$ are given by

$$\theta_i = -\frac{\pi}{2} - tan^{-1}\left(\frac{d_i^y e_i^x - d_i^x e_i^y}{e_i^z}\right)$$

We denote as $\mathbf{R}_{\theta_i}^{\mathbf{d}_i}$ the 3×3 matrix corresponding to a rotation of angle θ_i about axis $\mathbf{d}_i$. Applying $\mathbf{R}_{\theta_0}^{\mathbf{d}_0}$ to $\mathcal{I}_0$ and $\mathbf{R}_{\theta_1}^{\mathbf{d}_1}$ to $\mathcal{I}_1$ makes the two image planes parallel. Although this is technically sufficient for prewarping, it is useful to add an additional affine warp to align the scanlines. This simplifies the morph step to a scanline interpolation and also avoids bottleneck problems that arise as a result of image plane rotations [15].

The next step is to rotate the images so that epipolar lines are horizontal. The new epipoles are $[\tilde{e}_i^x \; \tilde{e}_i^y \; 0]^T = \mathbf{R}_{\theta_i}^{\mathbf{d}_i} \mathbf{e}_i$. The angles of rotation ϕ_0 and ϕ_1 are given by $\phi_i = -tan^{-1}(\tilde{e}_i^y / \tilde{e}_i^x)$. After applying these image plane rotations, the fundamental matrix has the form

$$\tilde{\mathbf{F}} = \mathbf{R}_{\phi_1} \mathbf{R}_{\theta_1}^{\mathbf{d}_1} \mathbf{F} \mathbf{R}_{-\theta_0}^{\mathbf{d}_0} \mathbf{R}_{-\phi_0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & a \\ 0 & b & c \end{bmatrix}$$

The 3×3 matrix $\mathbf{R}_\theta$ denotes an image plane (z axis) rotation of angle θ. Finally, to get $\mathbf{F}$ into the form of Eq. (8), the second image is translated and vertically scaled by the matrix

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -a & -c \\ 0 & 0 & b \end{bmatrix}$$

In summary, the prewarping transforms $\mathbf{H}_0$ and $\mathbf{H}_1$ are

$$\begin{aligned} \mathbf{H}_0 &= \mathbf{R}_{\phi_0} \mathbf{R}_{\theta_0}^{\mathbf{d}_0} \\ \mathbf{H}_1 &= \mathbf{T} \mathbf{R}_{\phi_1} \mathbf{R}_{\theta_1}^{\mathbf{d}_1} \end{aligned}$$

The entire procedure is determined by selecting $\mathbf{d}_0$. A suitable choice is to select $\mathbf{d}_0$ orthogonal to $\mathbf{e}_0$, i.e., $\mathbf{d}_0 = [-e_0^y \; e_0^x \; 0]^T$.

Creating and Rendering Image-Based Visual Hulls

Chris Buehler, Wojciech Matusik, Leonard McMillan
MIT, LCS Computer Graphics Group

Steven J. Gortler
Harvard University

Abstract

In this paper, we present efficient algorithms for creating and rendering image-based visual hulls. These algorithms are motivated by our desire to render real-time views of dynamic, real-world scenes. We first describe the visual hull, an abstract geometric entity useful for describing the volumes of objects as determined by their silhouettes. We then introduce the image-based visual hull, an efficient representation of an object's visual hull. We demonstrate two desirable properties of the image-based visual hull. First, it can be computed efficiently (i.e., in real-time) from multiple silhouette images. Second, it can be quickly rendered from novel viewpoints. These two properties motivate our use of the image-based visual hull in a real-time rendering system that we are currently developing .

Introduction

Computer graphics has long been concerned with the rendering of *static synthetic* scenes, or scenes composed of non-moving computer-created models. In time, attention turned to the rendering of *dynamic synthetic* scenes, as exemplified by virtual reality systems, most modern computer games, and the recent computer-animated movies. More recently, many researchers have embraced an image-based rendering approach in which scenes are represented by simple images that may be synthetic or acquired from the real world (say, with a digital camera). In this spirit, work has been done in rendering *static acquired* scenes, non-moving scenes acquired from real-world imagery (e.g., QuicktimeVR). However, relatively little work has been done in the case of *dynamic acquired* scenes. It is the goal of our work to develop an appropriate representation and rendering system for such scenes.

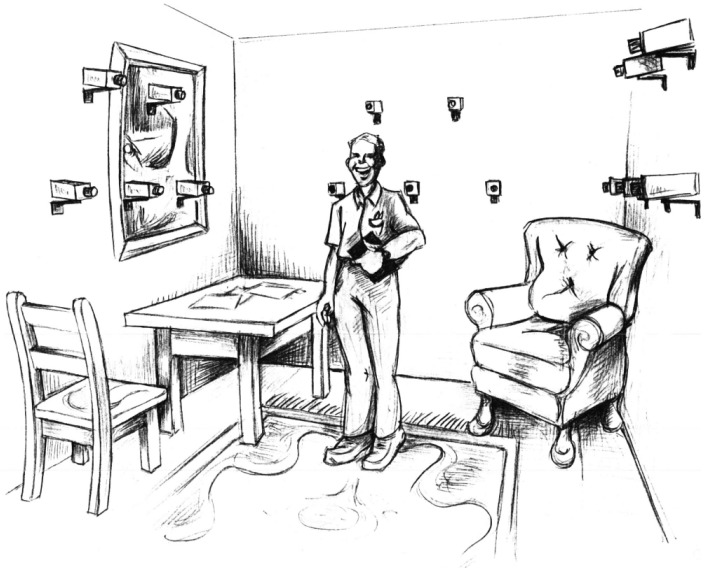


Figure 1. A hypothetical arrangement for acquiring dynamic scenes.

Using our system, a user can control a virtual camera within a moving scene that is acquired in real-time. Such a system has many potential uses. A commonly cited example is the virtual sports camera: users viewing a sporting event would be able to view the event from any angle, perhaps to focus on their favorite player or to see the action better. We are also targeting our current system at other tasks: teleconferencing and virtual sets. In a teleconferencing setting, our system would allow participants to navigate the virtual conference room or change their gaze while viewing the other participants moving in real-time. Applied to synthetic sets, our system would enable a director to see his actors perform in real-time in a dynamic three-dimensional virtual set.

A dynamic, acquired rendering system can be designed analogously to a static, acquired one. Static scenes (or objects) are typically acquired from many still photographs taken at different locations. Many photos are acquired, and often the same camera is used to take them. To extend this scenario to the dynamic case, we substitute video sequences for still photographs and place multiple, synchronized video cameras around the scene to acquire these sequences (see Figure 1). The dynamic setup is more restrictive than the static case: the number of input images is limited by the physical number of video cameras, and the cameras can only be placed in locations that do not impede the activity in the scene.

In both the static and dynamic cases, the acquired images are generally processed in some way—details vary from system to system—after which new images of the scene (or object) can be produced from arbitrary camera locations. In the dynamic case, a distinction can be made between *real-time* systems, those that process video and synthesize views at interactive rates, and *off-line* systems, those that require more extensive processing or rendering before viewing. In this paper, we are concerned with real-time systems.

There are a number of challenges inherent in real-time systems. The first is processing all the video frames at interactive rates. Obvious approaches for extracting useful information from multiple video streams, such as multi-baseline stereo algorithms, run too slow on current general-purpose hardware for a real-time system. The second challenge is rendering new views such that a virtual image exhibits as much visual fidelity as an image from one of the real cameras. For example, voxel-based systems often display noticeable artifacts in their images as a result of the low-resolution voxel data structure.

Our real-time system for rendering dynamic, acquired objects is designed to meet these challenges. We utilize between five and ten synchronized, digital video cameras to acquire continuous video streams. To achieve interactive rates, we process the video streams using efficient silhouette-based techniques to create a approximate on-the-fly models (called the *visual hull*) of the dynamic scene objects. We then create novel views of these dynamic objects using image-based rendering techniques, which are fast and preserve much of the detail of the original video sequences.

Related Work

Kanade's virtualized reality system [Kanade97] is perhaps closest in spirit to the dynamic acquired rendering system that we envision, although it is not currently a real-time system. They use a collection of cameras in conjunction with multi-baseline stereo techniques to extract models of dynamic scenes. Currently their method still requires significant off-line preprocessing time to perform the stereo correlation, but they are exploring special purpose hardware for this task, an option we wish to avoid. Recently, they have begun using silhouette methods such as the ones we use to improve the quality of their stereo reconstruction [Vedula98].

Pollard and Hayes [Pollard98] attempt to solve the problem of rendering real-time acquired data with their immersive video objects. Immersive video objects are annotated video streams that can be morphed in real-time to simulate three-dimensional camera motion. Their representation also utilizes object silhouettes, but in a different manner. They match silhouette edges across multiple views, and use these correspondences to compute a morph to a novel view. This approach has some problems, however, as silhouette edges are generally not consistent between views. These inconsistencies require their cameras to be placed close together, limiting the usefulness of the system.

Static Silhouette Methods

Silhouette contours have been used by computer vision researchers build approximate geometric models of static objects and scenes. These techniques are attractive because of the ease of extracting and working

with silhouettes.

Typically these object models are computed by using silhouettes to "carve" away regions of empty space. Potmesil describes a method for computing a voxel representation of objects from sequences of silhouettes [Potmesil87]. He uses an octree data structure to represent a binary volume of space, and does not attack the problem of reconstructing novel views of his objects.

Szeliski has implemented a similar idea [Szeliski92]. He uses a turntable to rotate objects in front of a real camera. After automatically extracting object silhouettes, he computes an octree-based voxel representation of the object by projecting octree nodes into the silhouette images.

Laurentini, recognizing the interest in silhouette methods, has introduced a formalism for analyzing object reconstruction from silhouettes [Laurentini94]. Central to his theory is the concept of the visual hull, which, is the best approximation to an object's shape that one can build from simple silhouettes. His framework is useful for understanding the limitations of silhouette methods, something that has not been quantified in earlier work.

Other volumetric carving methods, related to silhouette techniques, have also been suggested. These include volumetric reconstruction from active laser-range data [Curless96] and volumetric reconstruction based on photometric sample correspondences [Sietz97]. These techniques could be used to improve upon the approximate object models that are obtained from silhouettes. However, currently, they are not as well suited to real-time implementation.

Image-Based Rendering

Image-based rendering has been proposed as a practical alternative to the traditional modeling/rendering framework. In image based rendering, one starts with images and directly produces new images from this data. This avoids the traditional (i.e., polygonal) modeling process, and often leads to rendering algorithms whose running time is independent of the scene's geometric and photometric complexity.

Chen's QuicktimeVR [Chen95] is one of the first commercial static, acquired rendering systems. This system relies heavily on image-based rendering techniques to produce photo-realistic panoramic images of real scenes. Although successful, the system has some limitations: the scenes are static and the viewpoint is fixed.

McMillan's plenoptic modeling system [McMillan95] is QuicktimeVR-like, although it does allow a translating viewpoint. The rendering engine is based on three-dimensional image warping, a now commonplace image-based rendering technique. Dynamic scenes are not supported as the panoramic input images require much more off-line processing than the simple QuicktimeVR images.

Light field methods [Gortler96, Levoy96] represent scenes as a large database of images. Processing requirements are modest making real-time implementation feasible, if not for the large number of cameras required (on the order of hundreds). The cameras must also be placed close together, resulting in a small effective navigation volume for the system.

Paper Organization

In the next section we describe the visual hull, the approximate geometric representation that we use in our system. We demonstrate how it is related to object silhouettes, and why silhouette-based analysis techniques are well suited to this sort of system. We also point out some of the problems with using the visual hull as an object approximation.

In the second section, we describe various algorithms for computing visual hulls using a image-based representation. The first algorithm is slow, but conceptually simple, while the second algorithm is faster and more sophisticated. We present advantages and disadvantages and runtime analyses.

The third section discusses various rendering algorithms for image-based visual hulls. We have investigated at least four algorithms, each with strengths and weaknesses. In this paper, we discuss three of the algorithms.

Silhouettes and the Visual Hull

Silhouette methods are well suited to real-time analysis of object shape. First, computing object silhouettes is fast and relatively robust. Second, multiple silhouettes of an object give a strong indication

of that object's shape.

Computing Silhouettes

An object silhouette is essentially a binary segmentation of an image in which pixels are labeled "foreground" (belonging to the silhouette) or "background." In this paper, background pixels are typically drawn in white and foreground pixels non-white.

One common technique for computing silhouettes is chromakeying, or bluescreen matting [Smith96]. In this technique, the actual scene background is a single uniform color that is unlikely to appear in foreground objects. Foreground objects can then be segmented from the background by using color comparisons. Chromakey techniques are widely used in television weather forecasts and for cinematic special effects, which demonstrates their speed and quality. However, chromakey techniques do not admit arbitrary backgrounds, which is a severe limitation.

More general is a technique called background subtraction or image differencing [Bichsel94, Friedman97]. In background subtraction, a statistical model of a background scene is accumulated from many images. Changes in the scene, such as a figure walking into view, can then be detected by computing the difference between the new frame and the retained model. Differences that fall outside the allowed margins of the model are classified as foreground objects. There are many variations on the above two algorithms, but almost all of them are fast and robust enough to be used in a real-time system.

Shape from Silhouettes: The Visual Hull

It seems intuitive that the shape of an object can be recovered from many silhouettes. However, it is also clear that not all shapes can be recovered from silhouettes alone. For example, the concave region inside a bowl will never be evident in any silhouette, so any method based solely on silhouettes will fail to reconstruct it completely [Koenderink90].

Laurentini has introduced the concept of the visual hull for understanding the shapes of objects that can be reconstructed from their silhouettes [Laurentini94]. Loosely, the visual hull of an object is the closest approximation to that object that can be obtained from silhouettes alone.

The visual hull of an object depends both on the object itself and on a particular viewing region. A viewing region is a set of points in space from which silhouettes of an object are seen. The viewing region might be the set of all points enclosing the object, or, in a more practical case, a finite set of camera positions arranged around the object.

Formally, the visual hull of object S with respect to viewing region R, denoted $VH(S, R)$, is a volume in space such that for each point $P \in VH(S,R)$ and each viewpoint $V \in R$, the half-line from V through P contains at least one point of S [Laurentini94]. This definition simply states that the visual hull consists of all points in space whose images lie within all silhouettes viewed from the viewing region. Stated another way, the visual hull is the maximal object that has the same silhouettes as the original object, as viewed from the viewing region.

It is useful to think of an alternative, constructive definition of the visual hull with respect to a viewing region. Given a point V in the viewing region R, the silhouette of the object as seen from V defines a generalized cone in space with its apex at V (see Figure 2). The intersection of the cones from every point in R results in the visual hull with respect to R.

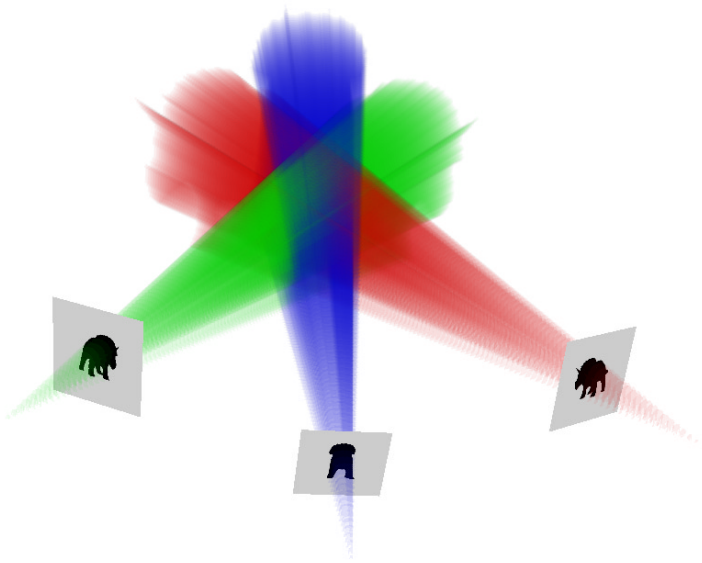


Figure 2. The intersection of the three silhouette cones defines the visual hull as seen from the viewing region. In this case, the viewing region contains only the apexes of the three silhouette cones.

This definition is useful because it implies a practical way to compute a visual hull. Almost all useful visual hull construction algorithms use some sort of volume intersection technique, as discussed in later sections.

Limitations of the Visual Hull

In the following discussion, we will assume that the viewing region for the visual hull is the set of all "reasonable" vantage points: those points outside the convex hull of the object. Using this special viewing region results in the closest possible approximation to the actual object. This viewing region is also assumed whenever reference is made to a visual hull whose viewing region is not implied by context.

The visual hull is a superset that contains the actual object's shape. It cannot represent concave surface regions (e.g., the inside of a bowl), in general, or even convex or hyperbolic points that are below the rim of a concavity (e.g., a marble inside a bowl). However, the visual hull is a tighter fit to the object than a convex hull, which only includes object regions that are globally convex. The visual hull of a convex object is the same as the object. However, the visual hull of an object composed of multiple, disjoint convex objects may not be the same as the actual objects, see Figure 3.

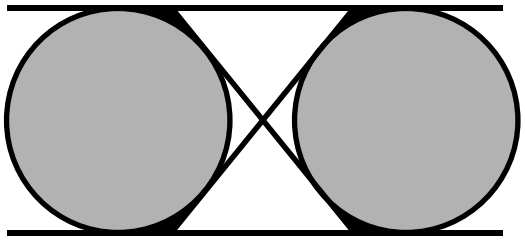


Figure 3. The visual hull of these two gray circles (black and gray regions) is slightly larger than the circles themselves. It is delimited by the bi-tangent lines drawn in the figure.

When the viewing region of the visual hull does not completely surround the object, the visual hull

becomes a coarser approximation and may even be larger than the convex hull. The visual hull becomes even worse for finite viewing regions, and may exhibit undesirable artifacts such as phantom volumes (Figure 4).

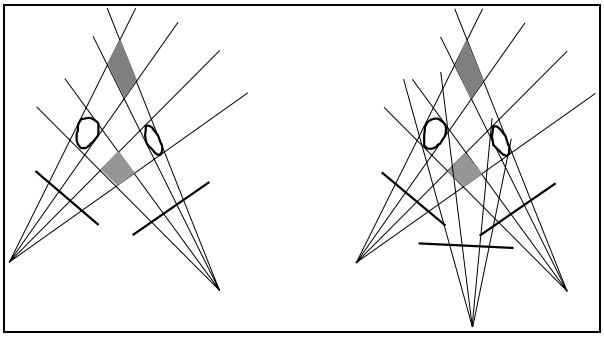


Figure 4. Intersecting the two silhouette cones results in "phantom" volumes, shown in gray on the left. A third silhouette can resolve the problem in this case (right).

In spite of these limitations, the visual hull is still a useful entity for approximating an object's shape in a dynamic rendering system. Object concavities can largely be camouflaged by object motion or hidden with surface texturing. Viewing regions that do not surround the object can be used as long as the virtual camera is confined to locations within the viewing region, as the visual hull is guaranteed to reproduce correctly all silhouettes seen from within the viewing region. Artifacts arising from using a finite viewing region (i.e., a finite amount of cameras) can be lessened by sampling a desired viewing region with appropriately placed viewpoints.

An Image-Based Visual Hull

One could attempt to compute a visual hull geometrically, but this approach, based on the intersection of multiple polytopes, is difficult to implement robustly and the resulting representation is composed of a great number of polygons if the silhouette contour is complex.

As a result, most visual hulls have been computed volumetrically by successively carving away all voxels outside of the projected silhouette cone. However, volumetric approaches suffer from problems with resolution. First, volumetric data structures are generally very memory intensive. This limitation is reduced somewhat by the fact that visual hull is a binary volume, and it is thus well suited to octree-type representations. However, it is still difficult to retain the full precision of the original silhouette images using a standard volumetric representation. If arbitrary configurations of input images are allowed then the intersection of the projected regions from them can have an arbitrarily high spatial frequency content. Thus no uniform spatial sampling is sufficient for exactly representing the final volume. Of course, reasonable approximations can be made by requiring the resulting volume to project to a silhouette contour that is within some error bound relative to the original.

In our approach, we prefer to use an image-based representation of the visual hull, which alleviates some of the problems with a standard voxel approach. In the graphics community, the term "image-based" has had many interpretations. In the strictest sense, an image-based representation consists solely of images (possibly along with matrices describing camera configurations). Along these lines, an image-based representation of the visual hull is simply the set of silhouettes themselves (along with the associated viewpoints). By definition, such a representation preserves the full resolution of the input images and contains no more or no less information than that provided by the silhouettes.

More generally, an "image-based" representation is often identified with a two-dimensional, sampled representation. For example, a standard color image is a rectangular grid of color samples, and a depth image is a grid of depth samples. Note that the samples are not considered connected in any way; they simply exist at regular intervals. The bulk of this paper is concerned with this second form of image-based visual hull.

This second type of image-based visual hull is constructed with respect to some viewpoint *V* in the viewing region of the visual hull. We can imagine that a camera at this viewpoint sees a silhouette image, which is discretized into a grid of pixels (i.e., samples). For each pixel in this silhouette image a list of occupancy intervals is stored. If a pixel does not belong to the silhouette (i.e., it is background), then the list is empty. Otherwise, the list contains intervals of space that are occupied by the visual hull of the object. These intervals, extruded over the solid angle subtended by the pixel, represent the region of the visual hull that projects to that pixel. The union of all such slices gives the visual hull as sampled from that viewpoint. In Figure 5, we show a slice of an image-based visual hull. The lines represent viewing rays along one column of the image, and the dark line segments denote occupied regions of space.

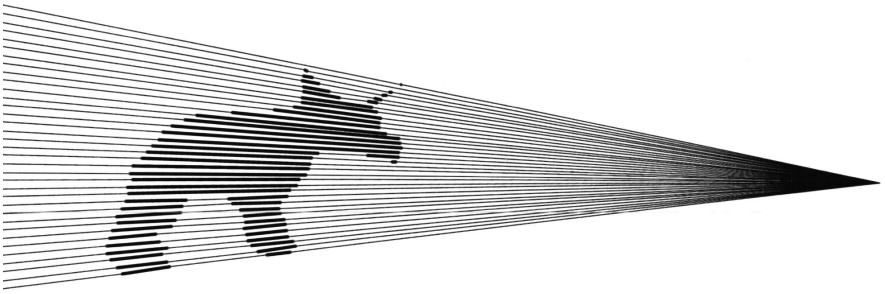


Figure 5. A single slice of an image-based visual hull. A full image-based visual hull contains many such slices, forming a volume in space.

Advantages of the Image-Based Representation

The image-based representation has a number of advantages in terms of storage requirements, computational efficiency, and ease of rendering.

The occupancy intervals can be stored as pairs of real numbers (where the numbers represent the minimum and maximum depths of the interval), similar to a run-length encoded volume. Thus, while the volume is discretized in two dimensions, the third dimension is continuous, allowing for higher resolution volumes than a voxel approach. Note also that this representation can be used for an arbitrary volume; it is not specialized for a visual hull. Similar data structures have been used by [VanHook86] and [Lacroute94] in traditional volume rendering settings.

Computing a visual hull using the image-based representation is much simpler than previous approaches.

As we will show in the next section, the three-dimensional generalized cone intersections and the volumetric carving operations of other methods are replaced with simple interval intersections in our method. These interval intersections are fast and robust, allowing for a real-time calculation of the visual hull.

Rendering the visual hull is also facilitated by the image-based representation. As we show in later sections, this representation can be rendered using only slight modifications to the standard three-dimensional image warp algorithm. This approach minimizes image resampling, as we only resample during rendering, and produces renderings of quality comparable to the input video images.

Mathematical Preliminaries

We first introduce the mathematical notation and concepts that we use in the rest of the paper. Dotted capital letters (e.g., $\dot{C}$) represent points in three-dimensional space, while lowercase over-bar letters (e.g., $\bar{x}$) represent homogeneous image (pixel) coordinates. Matrices (all are 3 x 3) are written in bold capital letters (e.g., **P**), while scalars are lowercase (e.g., *t*). We denote equality up to a scale factor with a dotted equals sign, $\doteq$.

One View

The basic quantity that we manipulate is a *view*, which is an image along with the viewpoint from which it was seen. We characterize a view [**P**, $\dot{C}$] by a center of projection $\dot{C}$ (i.e., the viewpoint) and an inverse projection matrix **P** that transforms homogeneous image coordinates $\bar{x}$ to rays in three-dimensional world space according to the following equation:

$$\dot{X}(t) = \dot{C} + t\mathbf{P}\bar{x},$$

where $\dot{X}(t)$ represents three-dimensional world points parameterized by the distance (or range) t along a ray. Conceptually, these rays originate at $\dot{C}$ and pass through the pixel $\bar{x} = [u, v, 1]^T$ in the imaging plane.

Often it is computationally more convenient to work with the reciprocal of the range parameter t. We call this quantity the *generalized disparity*, defined as

$$\delta = \frac{1}{t}.$$

Two Views

Two views [$\mathbf{P}_1$, $\dot{C}_1$] and [$\mathbf{P}_2$, $\dot{C}_2$] with different centers of projection (i.e., $\dot{C}_1 \neq \dot{C}_2$) are related by a so-called epipolar geometry. This geometry describes how a ray through a pixel in one view is seen as an *epipolar line* in the other view. Mathematically, this relationship between pixel coordinates in one view and epipolar lines in a second view is expressed by the *fundamental matrix* $\mathbf{F}_{21}$ between the two views [Faugeras93]. That is,

$$\bar{x}_2^T \mathbf{F}_{21} \bar{x}_1 = 0,$$

where the quantity $\mathbf{F}_{21}\bar{x}_1$ gives the coefficients of a line equation in the second image. Given two views [$\mathbf{P}_1$, $\dot{C}_1$] and [$\mathbf{P}_2$, $\dot{C}_2$], their fundamental matrix can be computed as

$$\mathbf{F}_{21} = \mathbf{E}_2 \mathbf{P}_2^{-1} \mathbf{P}_1.$$

Matrix $\mathbf{E}_2$ is a matrix representation of the cross product defined such that

$$\mathbf{E}_2 v = \bar{e}_2 \times v,$$

where v is an arbitrary vector and vector $\bar{e}_2$ is the *epipole*, or the projection of the first view's center of projection onto the second view's image plane. This epipole is computed as

$$\bar{e}_2 = \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2),$$

and the epipole of the first view with respect to the second is computed similarly.

Often we want to calculate a desired view from a known view. Given two views [$\mathbf{P}_1$, $\dot{C}_1$] and [$\mathbf{P}_2$, $\dot{C}_2$], where the first one is known and the second one is desired, we can transform pixels from the known view to pixels in the desired view using a three-dimensional image warping equation [McMillan96]:

$$\bar{x}_2 \doteq \mathbf{P}_2^{-1}\mathbf{P}_1\bar{x}_1 + \delta_1 \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2). \quad (1)$$

This equation gives pixel coordinates $\bar{x}_2$ in the desired view of the point defined by the pixel $\bar{x}_1$ and the disparity δ_1 in the first view. Thus, computing a desired view from a single known view requires auxiliary disparity information, which is often stored in the form of a depth image associated with the known view.

In computing image-based visual hulls, we are often interested in recovering the range (or disparity)

parameter t_2 given corresponding image points in two views. We solve this problem by computing the range parameters of the points of closest approach of the two rays defined by the corresponding pixels in two images as follows:

$$t_1 = \frac{\det\left[\dot{C}_2 - \dot{C}_1 \quad \mathbf{P}_2\bar{x}_2 \quad \mathbf{P}_1\bar{x}_1 \times \mathbf{P}_2\bar{x}_2\right]}{\left\|\mathbf{P}_1\bar{x}_1 \times \mathbf{P}_2\bar{x}_2\right\|^2}.$$

The parameter t_2 can be computed similarly.

Three Views

It has been shown [Shashua97] that three views are related by a mathematical entity called the trilinear tensor. Similar to the fundamental matrix for two views, the trilinear tensor describes the relationship between points and lines in the three views. A complete description of the trilinear tensor is beyond the scope of this paper, however, we do present four equations derived from the tensor which relate the coordinates of a pixel $\bar{p}'' = [x'', y'', 1]$ in a third view to the coordinates of pixels in two other views ($\bar{p} = [x, y, 1]$ and $\bar{p}' = [x', y', 1]$):

$$\begin{aligned}
x''\alpha_i^{13}\bar{p}^i - x''x'\alpha_i^{33}\bar{p}^i + x'\alpha_i^{31}\bar{p}^i - \alpha_i^{11}\bar{p}^i = 0, \\
y''\alpha_i^{13}\bar{p}^i - y''x'\alpha_i^{33}\bar{p}^i + x'\alpha_i^{32}\bar{p}^i - \alpha_i^{12}\bar{p}^i = 0, \\
x''\alpha_i^{23}\bar{p}^i - x''y'\alpha_i^{33}\bar{p}^i + y'\alpha_i^{31}\bar{p}^i - \alpha_i^{21}\bar{p}^i = 0, \\
y''\alpha_i^{23}\bar{p}^i - y''y'\alpha_i^{33}\bar{p}^i + y'\alpha_i^{32}\bar{p}^i - \alpha_i^{22}\bar{p}^i = 0.
\end{aligned}$$

In the above equations, α_i^{jk} (i,j,k = 1,2,3) is the 27 element trilinear tensor, and the notation $\alpha_i^{nm}\bar{p}^i$ denotes a dot-product of a row of the tensor with $\bar{p}$. The elements of α_i^{jk} are obtained from the three views [$\mathbf{P}_1$, $\dot{C}_1$], [$\mathbf{P}_2$, $\dot{C}_2$], and [$\mathbf{P}_3$, $\dot{C}_3$] according to the formulas given in [Shashua97].

The important quality of these equations, with regard to image synthesis, is that the third pixel's location is completely constrained by the locations of the two other pixels; no auxiliary depth image is needed. As we will demonstrate, these equations can be exploited when rendering novel views given two or more known views.

Creating Image-Based Visual Hulls

In the following sections, we describe algorithms for computing image-based visual hulls from a finite number of silhouette images. In all of these algorithms, the input is assumed to be a set of *k* silhouettes (i.e., binary images), their associated viewpoints, and a viewpoint from which the visual hull is to be constructed. The algorithms output a sampled image of the visual hull, in which each pixel of the image contains a list of occupied intervals of space.

To ease algorithm analysis, the input silhouettes are assumed to be square *m* x *m* arrays of pixels. The output resolution of the image-based visual hull is *n* x *n* pixels.

The Basic Algorithm

We implement the same basic idea in all of our visual hull construction algorithms. We cast a ray into space for each pixel in the desired view of the visual hull. We intersect this ray with the *k* silhouette cones defined by the *k* silhouette views and record the intersections as pairs of enter/exit points (i.e., intervals). This process results in *k* lists of intervals, which are then intersected together to form a single list. This final list, representing the intersection of the viewing ray with the visual hull, is stored in our data structure.

The key aspect of all our algorithms is that all of the ray/cone intersection calculations are done in two dimensions rather than three. Recall that each silhouette cone is defined by a two-dimensional silhouette

image and a center of projection. Instead of projecting these cones into three-dimensional space and then computing ray intersections, we can project the three-dimensional ray into the two-dimensional space of the silhouette image and perform intersections there. The ray simply projects to a line (in fact, the epipolar line as discussed in a previous section), and the resulting two-dimensional calculations are much more tractable.

The above observations lead directly to an algorithm for computing the image-based visual hull:

```
for each pixel p = [x,y,1] in VHULL
    initialize VHULL[x][y] = [depth_min, depth_max]

for each silhouette image SIL_i
    compute fundamental matrix F_i
    for each pixel p = [x,y,1] in VHULL
        compute epipolar line coefficients F_i p
        trace epipolar line in image SIL_i
        record list of silhouette contour intersection points [p_i,k]
        interval_list = []
        for each pair of intersection points p_i,2l and p_i,2l+1
            compute depth_i,l,min and depth_i,l,max measured w.r.t. VHULL
            interval_list = interval_list ∪ [depth_i,l,min, depth_i,l,max]
        endfor
    VHULL[x][y] = VHULL[x][y] ∩ interval_list
    endfor
endfor
```

The algorithm is illustrated in Figure 6. Six silhouettes from a synthetic dinosaur model are shown, and the desired image-based visual hull is computed from the viewpoint of the upper left silhouette (the primary view). Three pixels are labeled in this primary view. The corresponding epipolar line for each pixel is shown in the remaining five (secondary) images. The algorithm processes one secondary image at a time. First it detects each interval where the line crosses through the silhouette of the object. At each of these silhouette contour crossings the length along the ray of the primary image is computed using the equation for the point of closest approach. A list of these intervals is computed for each secondary image. Finally, the interval lists are merged by computing their intersections across all secondary images. This process is repeated for every pixel in the primary image.

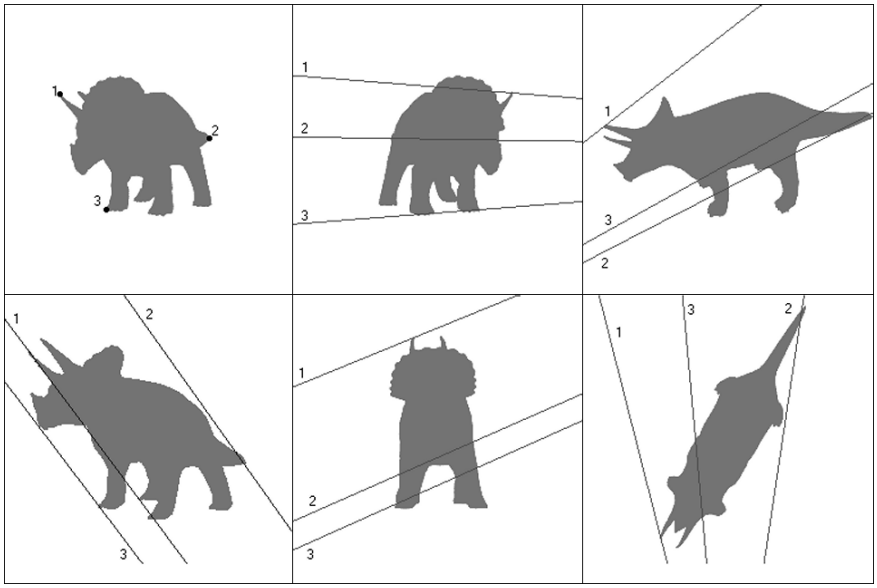


Figure 6. The image-based visual hull is computed from the viewpoint shown in the upper left. The epipolar lines corresponding to the three labeled pixels are shown in the five other silhouettes.

Analysis

The basic algorithm, while conceptually simple, is not a particularly efficient way to compute image-based visual hulls. The asymptotic running time is $O(km^2n)$, as the algorithm traces a line of length $O(n)$ in k images for each of m^2 pixels in the primary view. This analysis ignores the number of intervals traced and the cost of intersecting them. This omission is justified as there are typically far fewer intervals than the number of pixels in one dimension of a secondary image, and certainly not more than this number. When the primary and secondary images are of the same dimensions, a common case, then the running time is $O(kn^3)$. Thus, we generally consider this an n-cubed algorithm.

The algorithm also suffers from some quantization problems. The digital epipolar lines traced by the algorithm are generally not identical to the ideal epipolar lines. This discrepancy may cause the silhouette intersection points to be slightly off. In practice, such quantization problems have been largely unnoticeable.

Line-Cache Algorithm

The best running time we might expect from a visual hull construction algorithm is $O(km^2)$. This lower bound arises from the fact that we need to fill in interval lists for m^2 pixels, and we need to process k views. One might imagine a faster algorithm, based on a hierarchical decomposition (e.g., a quadtree) of the visual hull image, but here we will assume we want to create m^2 individual interval lists. A hierarchical decomposition, if desired, can then be applied to any of our algorithms.

The line-cache algorithm is an algorithm for computing the image-based visual hull that achieves the $O(km^2)$ running time. The increased efficiency is due to a simple observation: multiple three-dimensional rays from the primary image project to the same two dimensional line in the secondary images. This fact can be understood from the epipolar geometry between two views. A viewing ray from the primary image and the viewpoint of a secondary image are contained within a plane in space. This plane projects to an epipolar line in the secondary image. Any other viewing ray from the primary image which also lies in this plane projects to the same epipolar line in the secondary image.

The observation can also be demonstrated with a counting argument. It takes roughly $O(n)$ lines of length $O(n)$ to fill a discrete (pixelized) two-dimensional space of size $O(n^2)$. Thus, if we project $O(n^2)$ lines of length $O(n)$ into this space, we can expect that $O(n)$ lines will map to the same line. Of course, this argument is really only valid in a discrete setting, which is the setting in which we compute our image-based visual hulls.

Using the above observation, we amend our basic algorithm in the following way. When we attempt to compute the two dimensional line/silhouette intersection, we first check in an "epipolar line cache" data structure to see if the intersection intervals have already been computed. If so, we used the cached results. Otherwise, we compute the line intersections and store the resulting interval list in the line cache.

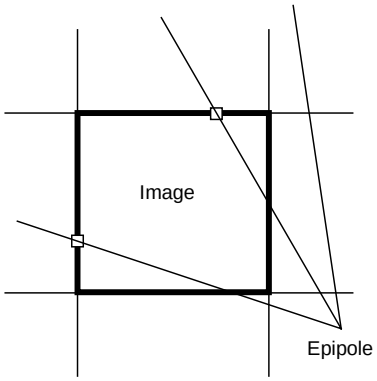


Figure 7. We determine line cache indices by the farthest intersection of the epipolar line with the image boundary. Lines that do not intersect this boundary need not be cached.

The only real issue to deal with in this algorithm is how to index the cache. That is, how do we determine that two lines are the same? There are many ways to do this; in our implementation we compute the intersection of the epipolar line with the farthest image boundary (see Figure 7). We use this intersection coordinate as the index to our cache. This indexing style allows us to vary the performance of our cache by changing the resolution of our coordinate system. For example, computing intersections to the nearest half-pixel gives a larger cache that better represents lines, but may result in fewer cache hits. Using the nearest double-pixel results in a smaller cache and more hits, but may group lines that are too dissimilar in the same cache location.

The line-cache algorithm is as follows:

```
for each pixel p = [x,y,1] in VHULL
    initialize VHULL[x][y] = [depth_min, depth_max]

for each silhouette image SIL_i
    for each cache index
        initialize CACHE_i[index] = EMPTY
    endfor
    compute fundamental matrix F_i
    for each pixel p = [x,y,1] in VHULL
        compute epipolar line coefficients F_i p
        compute line cache index = compute_index(F_i p)
        if(CACHE_i[index] = EMPTY)
            trace epipolar line in image SIL_i
            record list of silhouette contour intersection points [p_i,k]
            CACHE_i[index] = [p_i,k]
        else
            [p_i,k] = CACHE_i[index]
        endif
        interval_list = []
        for each pair of intersection points p_i,2l and p_i,2l+1
            compute depth_i,l,min and depth_i,l,max measured w.r.t. VHULL
            interval_list = interval_list ∪ [[depth_i,l,min, depth_i,l,max]]
        endfor
    VHULL[x][y] = VHULL[x][y] ∩ interval_list
    endfor
endfor
```

Analysis

We will consider a worst case running time for the line-cache algorithm in which all cache lines are accessed. The size of each cache is O(n), and for each cache entry a line of length O(n) is traversed, leading to a total time of O(kn^2) spent computing all cache entries. The algorithm spends time O(km^2) retrieving interval lists from the caches. Thus, the runtime is O(kn^2) if $n > m$, and O(km^2) otherwise. In practice, we find that 90% of the cache entries are accessed, so this worst case analysis is applicable.

The line-cache algorithm gains its speed by making some tradeoffs in the quality of the resulting visual hull. In addition to the quantization errors from the basic algorithm, the line cache algorithm introduces errors by mapping slightly different epipolar lines to the same cache location. In practice, such errors are small, although they may be noticeable near depth discontinuity edges.

Rendering Image-Based Visual Hulls

The rendering problem is to produce a novel image of the *original object* as seen from some desired view, given an image-based visual hull of the object along with its original source views (i.e., the camera pose and images before segmentation). Since we have already shown that the visual hull is an approximation to the object's true shape, it will generally be impossible to create the exact image of the object from the new view. Thus, the goal of our rendering algorithms is to reproduce as closely as possible the true object's shape and color with information from the visual hull (shape) and the original camera images (color).

We are interested in a number of additional sub-goals for our rendering algorithms. First, they should be fast enough so that they will be applicable in our dynamic, real-time system. Second, they should offer

high quality imagery in the sense that rendered images should be reasonably indistinguishable from the original camera images.

The inputs to each algorithm are assumed to be an image-based visual hull (*n* x *n* pixels), *k* original camera images (*n* x *n* pixels), and a desired view. The output is an *m* x *m* pixel image as seen from the desired view.

In all comparisons, we use the synthetic dinosaur images as inputs. The visual hull is computed from six 256 x 256 images. We generate novel renderings from three different viewpoints to exercise the strengths and weaknesses of the different algorithms. All six input dinosaur images are shown in Figure 8.

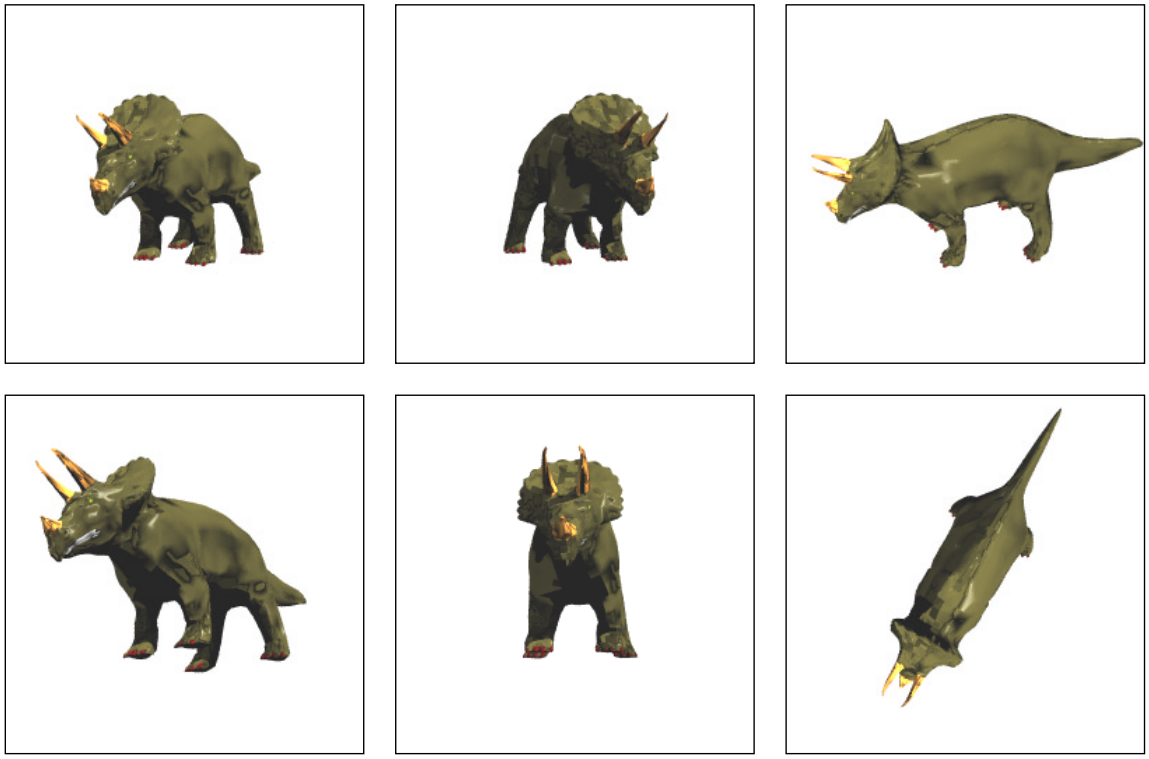


Figure 8. The six input dinosaur images (textures and silhouettes) used to create and render the image-based visual hull examples in this paper.

Texture Extrusion

The texture extrusion rendering method requires the image-based visual hull to be computed from the same viewpoint as one of the original camera images. In this special case, the pixels in the camera image are in one-to-one correspondence with the pixels in the visual hull image. In other words, each list of occupancy intervals in the visual hull image has a color assigned to it from the corresponding pixel in the camera image.

This special arrangement suggests a simple rendering technique: we can draw the occupancy intervals as seen from the new view, and we can color them with the colors assigned from the camera image. Such a rendering technique amounts to extruding the two-dimensional color image (or texture) along viewing rays to create a three-dimensional textured volume.

The basic requirement to use this technique is an ability to render a list of occupancy intervals from arbitrary viewpoints. The occupancy intervals are essentially long, thin cones in space. Calculating their projected shape exactly in the desired view would be prohibitively expensive for a real-time rendering algorithm. However, for viewpoints that are close to the viewpoint of the visual hull, the occupancy intervals can be approximated by simple line segments. Drawing these line segments can be done very

quickly since it is possible calculate the end points of the line segments efficiently.

The line segment endpoints can be incrementally computed using the three-dimensional warping equation (Equation 1). Recall that the image-based visual hull data structure stores a list of disparity values $[\delta_{1,\min}, \delta_{1,\max}, \ldots, \delta_{k,\min}, \delta_{k,\max}]$ for each pixel $\bar{p} = [x, y, 1]$, much like a Layered Depth Image [Shade98]. As is done when rendering Layered Depth Images, we exploit the fact that the warping equation reduces to a simple function of disparity for a fixed pixel $\bar{p}$:

$$\bar{x}_2(\delta) \doteq \bar{a} + \delta \bar{e}, \tag{2}$$

where $\bar{a} = \mathbf{P}_2^{-1}\mathbf{P}_1\bar{p}$ and $\bar{e} = \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2)$, which are constant for a given $\bar{p}$.

While a Layered Depth Image only stores depth values for front-facing surfaces, we store pairs of depth values that delimit occupied regions of space. Thus, to calculate the endpoints for the line segments, we evaulate this simple expression for each disparity pair $(\delta_{\min}, \delta_{\max})$ in the occupancy interval list. Given the endpoints, we draw the line segments using a fast digital line drawing routine. The complete texture extrusion algorithm is as follows:

```
compute H = P2^-1 P1
compute e = P2^-1 (C1 – C2)
for each pixel p = [x,y,1] in VHULL
    compute a = Hp
    for each interval [d_l,min, d_l,max] in VHULL[x][y]
        compute line segment endpoints [x_l,min, y_l,min] = a + d_l,min e
        and [x_l,max, y_l,max] = a + d_l,max e using the incremental
        three-dimensional warp equation (Equation 2)
        draw_line(x_l,min, y_l,min, x_l,max, y_l,max, VHULL[x][y].color)
    endfor
endfor
```

Analysis

The texture extrusion algorithm runs in time complexity $O(n^2 m)$, as it draws a line of length $O(m)$ for each of n^2 interval lists in the visual hull data structure. Although this may not seem fast, in practice it is fast enough for real-time rendering (~ 20 frames/sec). Texture extrusion also produces reasonably good looking images for viewpoints close to the viewpoint of the visual hull. Figure 9a demonstrates a novel viewpoint close to the original one. The visual hull in this case was computed from the viewpoint of the upper left-hand image in Figure 8.

Texture extrusion fails, however, when the desired viewpoint is far from the viewpoint at which the visual hull was sampled. This failure is primarily due to two factors. First, when the viewpoint is moved too far to one side, the extruded colors no longer approximate the true color of the object (see Figure 9b). This problem is unavoidable, as a single camera image can not see the entire object at one time. Second, when the viewpoint is moved very close to the object, the approximation of drawing line segments for the occupancy intervals is no longer valid and the images "explode" (see Figure 9c).

(a) (b) (c)

Figure 9. Images rendered from three novel viewpoints using texture extrusion.

Texture Projection

The texture projection algorithm extends the texture extrusion algorithm to handle a wider range of viewpoints. It corrects the second viewpoint problem, that of incorrect colors for distant viewpoints, by combining colors from multiple textures into a single rendering.

Texture projection is a simple extension to the texture extrusion algorithm. In texture extrusion, a single texture is essentially projected through the volume of the visual hull. Regions of the visual hull that are seen from the texture's viewpoint are colored correctly, while other regions are colored incorrectly. In texture projection, we project multiple textures onto the surface of the visual hull. Regions of the visual hull that are not seen by one texture can be colored with information from another texture.

We implement texture projection by a small modification to the texture extrusion algorithm. Instead of drawing each line segment with a constant color, we projectively texture map the line segment with colors from another texture. The projective texture mapping is done using the trilinear tensor equations. The tensor between the three views—the visual hull's view, the texture's view, and the desired view—allow us to compute texture coordinates in the texture's view given coordinates in the visual hull's view and the desired view. Pseudocode for the algorithm is give below. In the pseudocode $[\mathbf{P}_1, \dot{C}_1]$ refers to the visual hull's view, $[\mathbf{P}_2, \dot{C}_2]$ denotes the desired view, and $[\mathbf{P}_k, \dot{C}_k]$ is one of the texture views.

```
compute H = P2^-1 P1
compute e = P2^-1 (C1 – C2)
for each pixel p = [x,y,1] in VHULL
    compute a = Hp
    for each interval [dl,min, dl,max] in VHULL[x][y]
        compute line segment endpoints [xl,min, yl,min] = a + dl,min e
            and [xl,max, yl,max] = a + dl,max e using the incremental
            three-dimensional warp equation (Equation 1)
        k = select_texture(x, y, l)
        draw_line_proj_tex(x, y, xl,min, yl,min, xl,max, yl,max, P1, C1, P2, C2, Pk, Ck)
    endfor
endfor
```

The auxiliary function `draw_line_proj_tex` implements projective texture mapping using the trilinear tensor computed from $[\mathbf{P}_1, \dot{C}_1]$, $[\mathbf{P}_2, \dot{C}_2]$, and $[\mathbf{P}_k, \dot{C}_k]$. The function `select_texture` selects the texture to be mapped to the indicated visual hull interval. Many mappings are possible; we implemented a particularly simple strategy in our real-time implementation. We choose the texture with the minimum angle between the visual hull interval and the texture's viewpoint.

Analysis

The texture projection algorithm has the same asymptotic running time as the texture extrusion algorithm, $O(n^2m)$. However, because of the cost of the texture mapping, the hidden constant is much larger, which makes the algorithm slower in practice. The quality of the images is generally better, and the algorithm is useful for larger changes in the viewpoint (see Figures 10a and 10b). However, texture projection does suffer from the same zooming problem as the texture extrusion algorithm (see Figure 10c).

(a) (b) (c)

Figure 10. Images rendered from three novel viewpoints using texture projection.

Ray-Casting

Both the texture extrusion and the texture projection algorithms suffer from the same problem with viewpoints that are too close to the object: the image tends to break apart. This problem is directly related to the fact that both algorithms are *forward mapped*. They transform points from the visual hull to pixels in the desired view, and they may miss pixels along the way. Similar problems exist in other areas of computer graphics, and they are typically solved by using a *backward mapped* algorithm. In such an algorithm, pixels in the desired view are transformed to points in the visual hull. In this manner, every pixel in the desired view can be mapped to some point in the visual hull and colored appropriately.

To implement a backward mapped algorithm for rendering visual hulls, we would like to know for every pixel in the desired view whether or not the ray through that pixel intersects the visual hull. To compute this, we can cast a ray for every pixel in the desired view and test it for intersections with the k silhouette cones from the k cameras. Or, in other words, we can compute an image-based visual hull from the desired viewpoint.

An image-based visual hull computed from the desired viewpoint effectively gives the *shape* of the visual hull in the form of a depth image. However, we would like to have the proper colors along the with shape. We can compute the colors using a bit of additional computation to back project the visual hull to the k camera images and sample the colors. The complete algorithm is as follows:

```
compute VHULL_d from view [P_d, C_d]

for each pixel p = [x, y, 1] in VHULL_d
    extract depth_min from VHULL_d[x][y]
    for each camera image CAM_k
        backproject p to p_k = [x_k, y_k, 1] using Equation 1
        color_k = CAM_k[x_k][y_k]
    endfor
    VHULL_d[x][y] = weighted_avg(color_k)
endfor
```

The function **`weighted_avg`** simply computes some weighted average of the colors sampled from the k camera images. A color weight may be 0 if the camera makes no contribution to the color (e.g., it is occluded) or 1 if the camera contributes all the color (e.g., a winner-take-all strategy). In some cases, calculating the weights may be non-trivial. We use the winner-take-all approach in our implementation. That is, we assign a "best" camera a weight of 1 and assign all other cameras 0 weights. We define the best camera as the camera whose viewing ray is closest to that of the viewing direction. This strategy for assigning camera weights ignores the occlusion problem, and cameras may be selected which actually do not see the pixel to be colored.

Analysis

Due to its backward mapped nature, the ray-casting algorithm has a complexity fundamentally different than the previous two rendering algorithms. The running time is $O(km^2)$, as the visual hull calculation is $O(km^2)$, and the pixel coloring loop backprojects each of m^2 pixels k times. This running time is noteworthy as it is proportional to the size of the desired image and independent of the size of the camera images (for $m > n$). For $m = n$, the algorithm is n-squared, which compares favorably to the n-cubed forward mapped algorithms. However, the hidden constant is large, so this advantage is not realized at typical values of n.

This algorithm is slower than the forward mapped algorithms, but potentially produces images of higher quality (image quality and speed depend on the choice of color weighting). However, since the runtime of this algorithm includes the explicit visual hull calculation, the comparison is slightly unfair. Also, because it is backward mapped, problems with close range viewpoints are avoided. Ray-casting results are shown in Figures 11a, 11b, and 11c.

(a) (b) (c)

Figure 11. Images rendered from three novel viewpoints using ray-casting.

Conclusion

We have introduced the image-based visual hull as an approximate object representation for real-time dynamic acquired rendering systems. The needs of these systems require algorithms that allow for both the analysis of video inputs and the synthesis of rendered outputs to occur in real-time. Our algorithms for creating and rendering image-based visual hulls satisfy these requirements.

We have shown that the visual hull is a reasonable object representation to use in terms of accuracy and robustness. It provides a reasonable approximation to object shape in most cases, and requires only simple silhouette segmentation for acquisition.

We have demonstrated an efficient real-time algorithm for creating visual hulls. First, we exploit epipolar geometry to reduce three-dimensional volume intersections to simpler two-dimensional line intersections. Then, we use a line-caching approach to reuse previously computed results giving a further increase in performance.

Finally, we have presented a number of algorithms for rendering views of image-based visual hulls from novel viewpoints. The texture extrusion algorithm is fast but does not make use of all available color information. The texture projection algorithm, while slower, does utilize color information from all possible cameras. Both algorithms, however, suffer from a problem with viewpoints that are too close to the object. This problem is remedied by the ray-casting algorithm, which generates an image directly from the visual hull calculation.

Acknowledgements

Support for this research was provided by DARPA contract N30602-97-1-0283, and the Massachusetts Institute of Technology's Laboratory for Computer Science. We would also like to thank Anne McCarthy for providing artwork.

References

[Bichsel94] Bichsel, M., *"Segmenting Simply Connected Moving Objects in a Static Scene,"* **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. 16, No. 11, November 1994, pp. 1138-1142.

[Chen95] Chen, S.E., *"QuickTime VR - An Image-Based Approach to Virtual Environment Navigation,"* **Computer Graphics** (SIGGRAPH '95 Conference Proceedings), August 6-11, 1995, pp. 29-38.

[Curless96] Curless, B., and M. Levoy. *"A Volumetric Method for Building Complex Models from Range Images,"* **Computer Graphics** (SIGGRAPH '96 Conference Proceedings), August 4-9, 1996, pp. 43-54.

[Faugeras93] Faugeras, O., **Three-dimensional Computer Vision: A Geometric Viewpoint**, The MIT Press, Cambridge, Massachusetts, 1993.

[Friedman97] Friedman, N., and Russell, S., *"Image Segmentation in Video Sequences,"* **Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence**, 1997.

[Gortler96] Gortler, S.J., Grzeszczuk, R., Szeliski, R, and Cohen, M.F., *"The Lumigraph,"* **Computer Graphics** (SIGGRAPH'96 Conference Proceedings), August 4-9, 1996, pp. 43-54.

[Kanade97] Kanade, T., Rander, P. W., Marayanan, P. J., *"Virtualized Reality: Constructing Virtual Worlds from Real Scenes,"* **IEEE MultiMedia**, Vol.4, No.1, Jan. - Mar. 1997, pp.34-47.

[Koenderink90] Koenderink, J. J., **Solid Shape**, The MIT Press, Cambridge, Massachusetts, 1990.

[Lacroute94] Lacroute, P., Levoy, M., *"Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation,"* **Computer Graphics** (SIGGRAPH '94 Conference Proceedings), July 24-29, 1994, pp. 451-458.

[Laurentini94] Laurentini, A., *"The Visual Hull Concept for Silhouette-Based Image Understanding,"* **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. 16, No. 2, February 1994, pp. 150-162.

[Levoy96] Levoy, M. and P. Hanrahan, *"Light Field Rendering,"* **Computer Graphics** (SIGGRAPH'96 Conference Proceedings), August 4-9, 1996, pp. 31-42.

[McMillan95] McMillan, L., and Bishop, G., *"Plenoptic Modeling: An Image-Based Rendering System,"* **Computer Graphics** (SIGGRAPH '95 Conference Proceedings), August 6-11, 1995, pp. 39-46.

[McMillan96] McMillan, L., *"An Image-Based Approach to Three-Dimensional Computer Graphics,"* Ph.D. Thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1996.

[Pollard98] Pollard, S., and Hayes, S.,*"View Synthesis by Edge Transfer with Applications to the Generation of Immersive Video Objects,"* **Proceedings of the ACM Symposium on Virtual Reality Software and Technology**, November 2-5, 1998, pp. 91-98.

[Potmesil87] Potmesil, M., *"Generating Octree Models of 3D Objects from Their Silhouettes in a Sequence of Images,"* **Computer Vision, Graphics, and Image Processing**, Vol. 40, 1987, pp. 1-29.

[Seitz97] Seitz, S. M., Dyer, C. R., *"Photorealistic Scene Reconstruction by Voxel Coloring,"* **Computer Vision and Pattern Recognition Conference**, 1997, pp. 1067-1073.

[Shade98] Shade, J., Gortler, S., He, L., and Szeliski, R., *"Layered Depth Images,"* **Computer Graphics** (SIGGRAPH '98) Conference Proceedings), July 19-24, 1998, pp. 231-242.

[Shashua97] Shashua, A., *"Trilinear Tensor: The Fundamental Construct of Multiple-view Geometry and its Applications,"* **International Workshop on Algebraic Frames For The Perception Action Cycle (AFPAC)**, Kiel Germany Sep. 8-9, 1997.

[Smith96] Smith, A. R., and Blinn, J. F., *"Blue Screen Matting,"* **Computer Graphics** (SIGGRAPH '96 Conference Proceedings), August 4-9, 1996, pp. 21-30.

[Szeliski92] Szeliski, R., *"Rapid Octree Construction from Image Sequences,"* **CVGIP: Image Understanding**, Vol. 58, No. 1, July 1993, pp. 23-32.

[VanHook86] Van Hook, T., *"Real-time shaded NC milling display,"* **Computer Graphics** (SIGGRAPH '86 Conference Proceedings), 1986, pp. 15-20.

[Vedula98] Vedula, S., Rander, P., Saito, H., Kanade, T., *"Modeling, Combining, and Rendering Dynamic Real-World Events from Image Sequences,"* **4th International Conference on Virtual Systems and Multimedia conference proceedings**, Nov. 1998.

Image or Object?

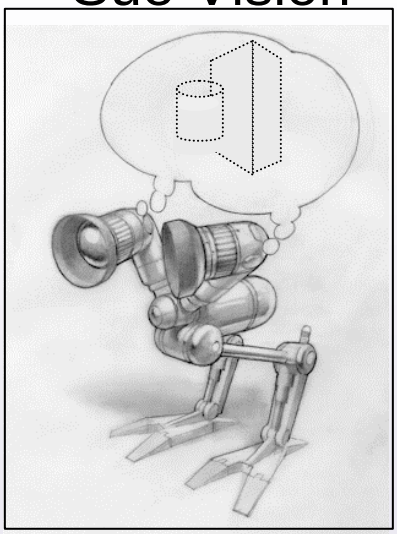
Michael F. Cohen
Microsoft Research

Is this real?

A mental model

Photo by Patrick Jennings (patrick@synaptic.bc.ca), Copyright 1995, 96, 97 Whistler B. C. Canada

Sue Vision

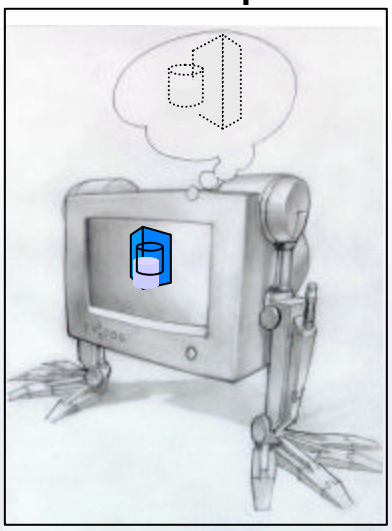


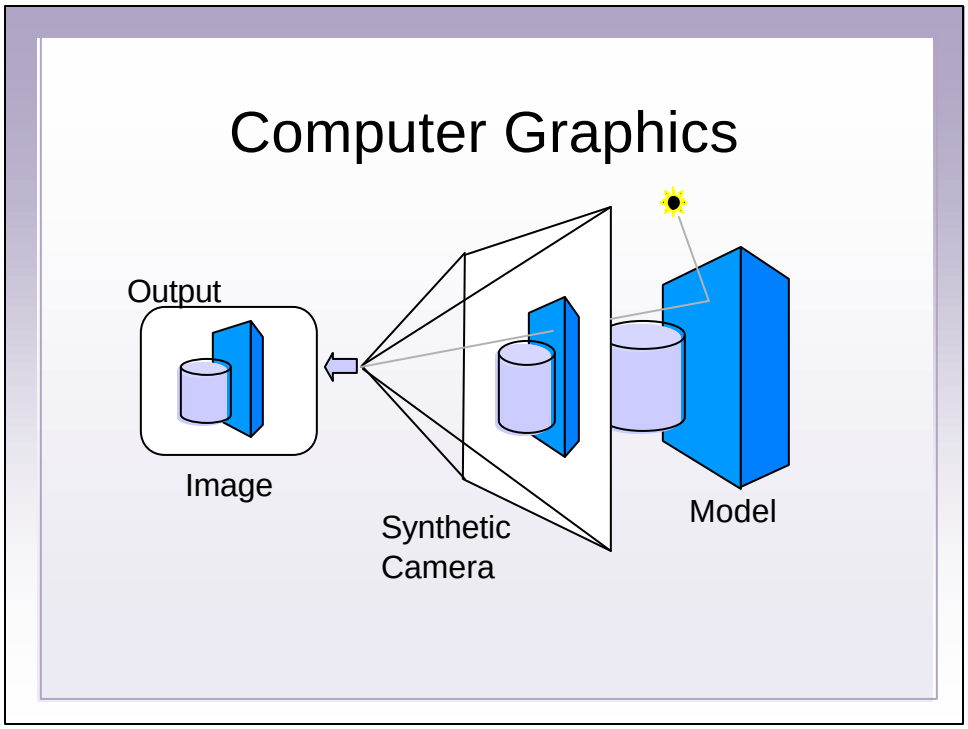
Is this real?

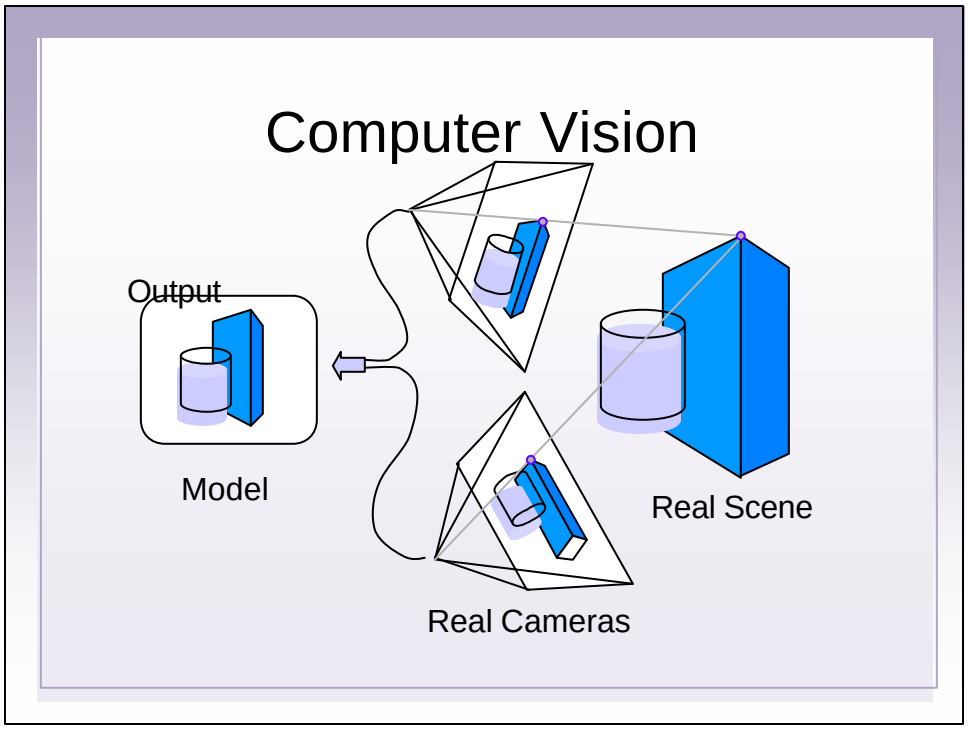
Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg

1988, *A Progressive Refinement Approach to Fast Radiosity Image Generation*.

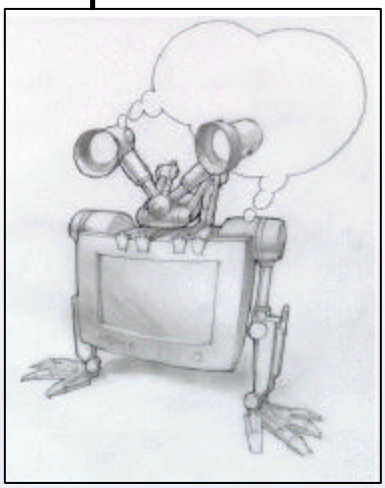
Bob Graphics



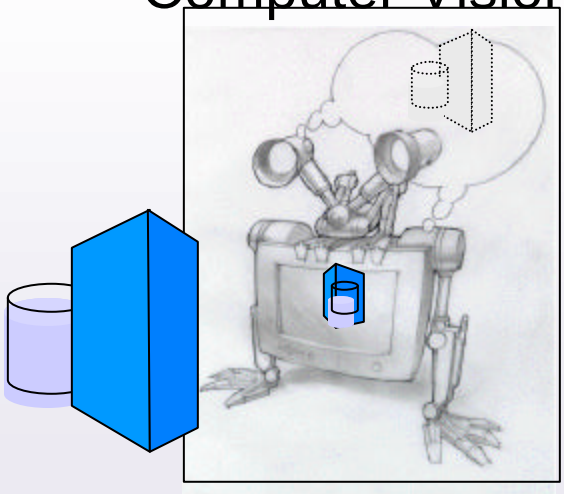

Computer Graphics
Output
Image
Synthetic Camera
Model

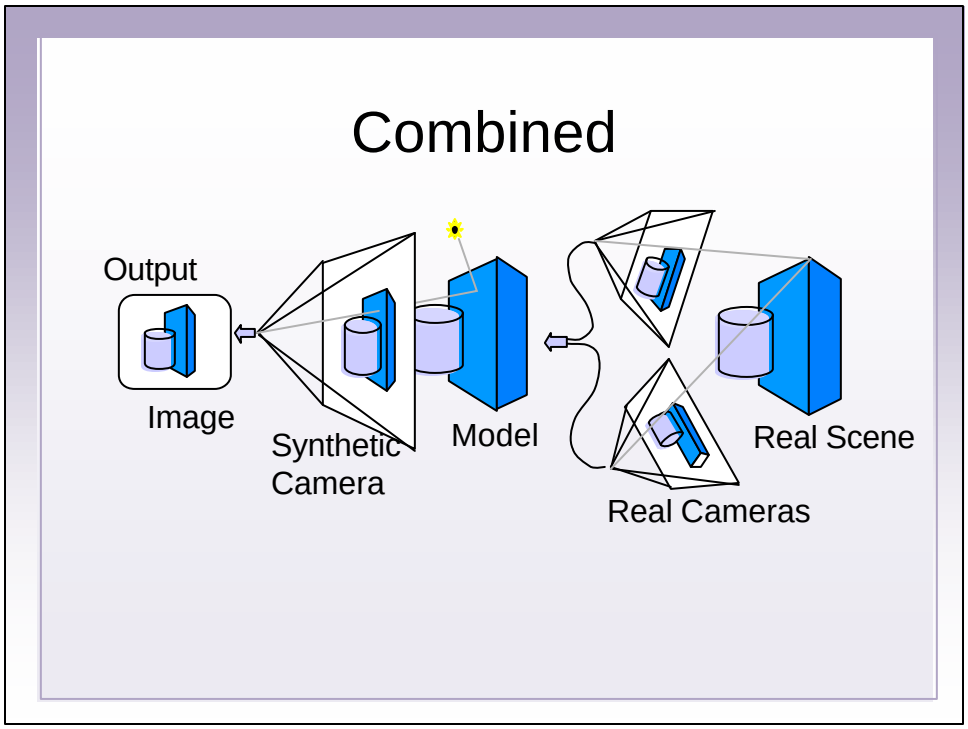

Computer Vision
Output
Model
Real Scene
Real Cameras

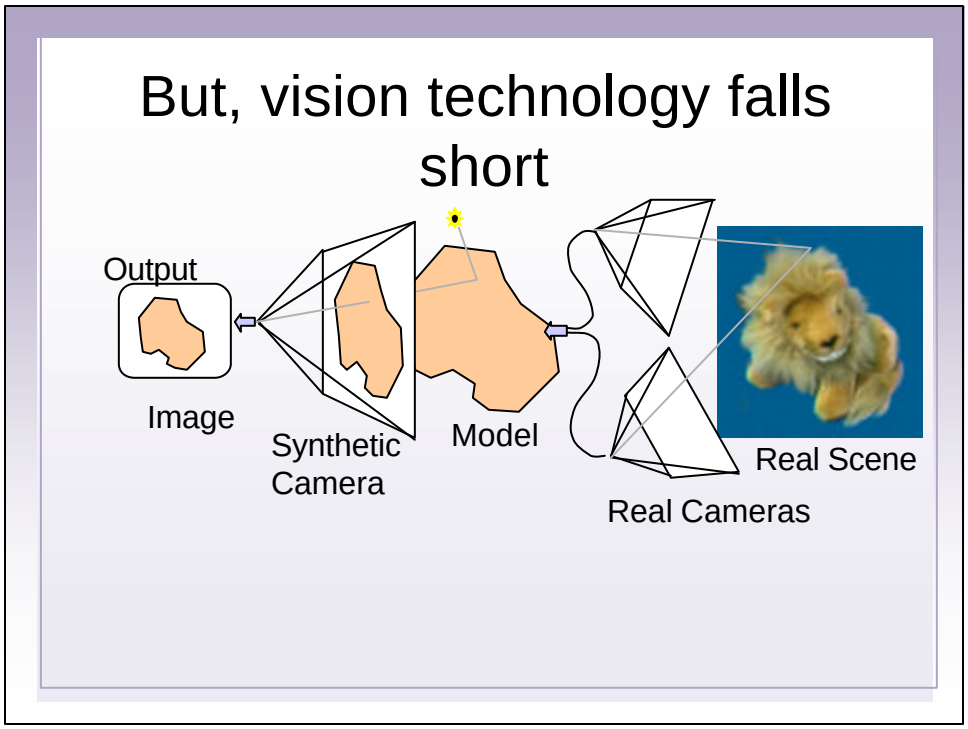
Computer Graphics Meets Computer Vision

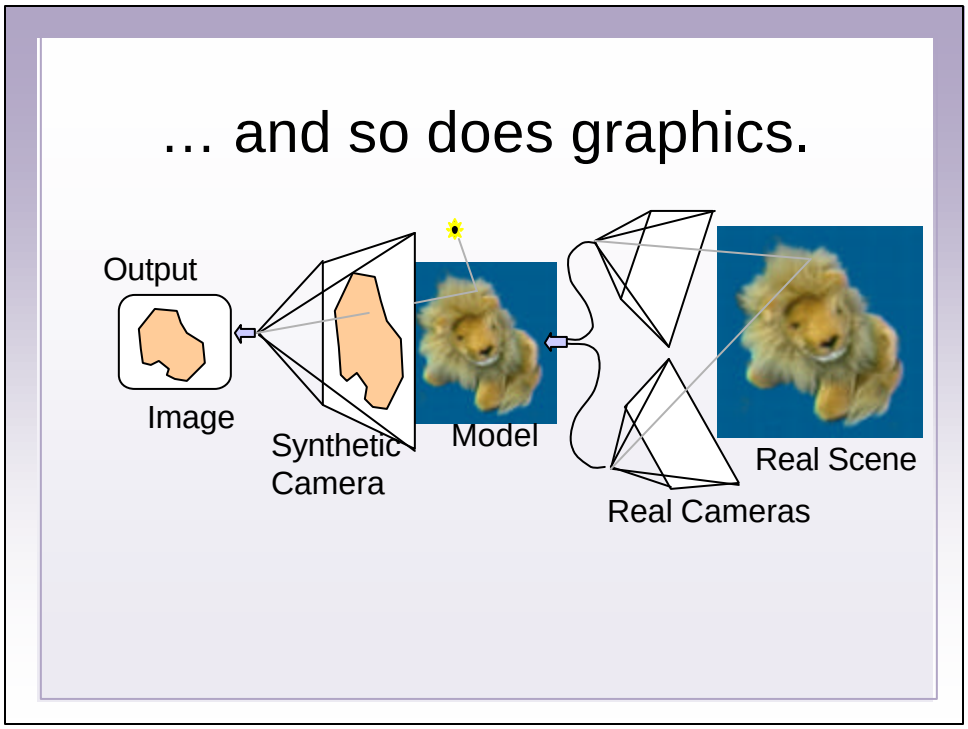


Computer Graphics Meets Computer Vision




Combined
Output
Image
Synthetic Camera
Model
Real Scene
Real Cameras


But, vision technology falls short
Output
Image
Synthetic Camera
Model
Real Scene
Real Cameras


… and so does graphics.
Output
Image
Synthetic Camera
Model
Real Scene
Real Cameras

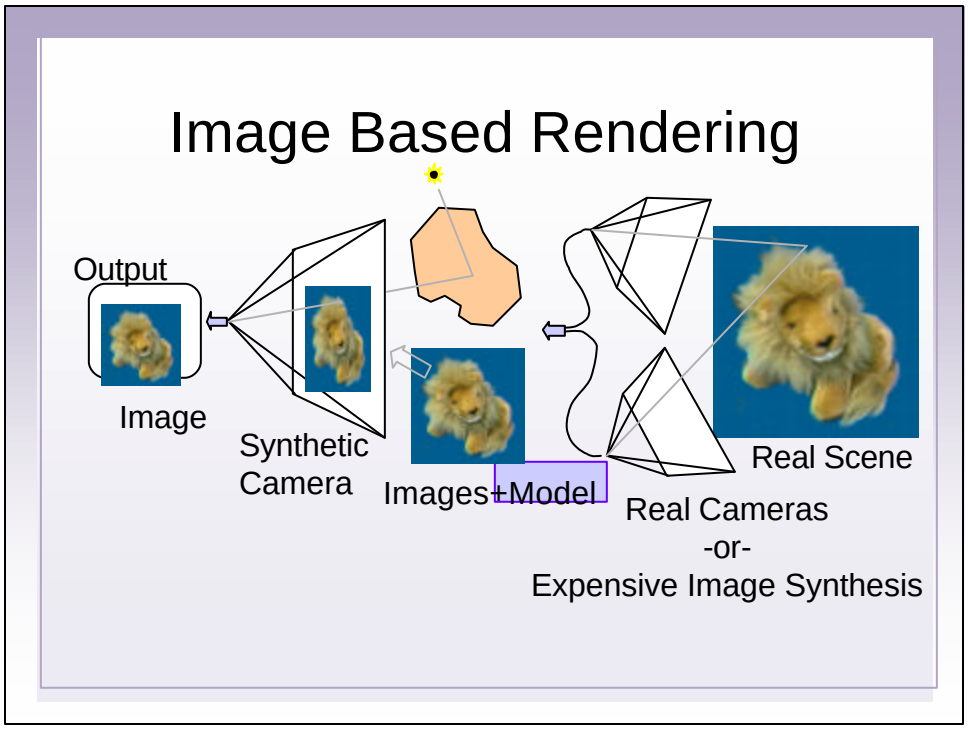


Image Based Rendering
Output
Image
Synthetic Camera
Images+Model
Real Scene
Real Cameras
-or-
Expensive Image Synthesis

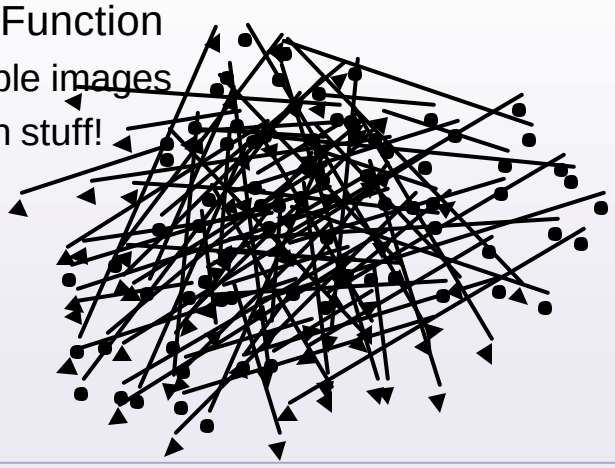
Ray

- Constant radiance
  - time is fixed

- 5D
  - 3D position
  - 2D direction

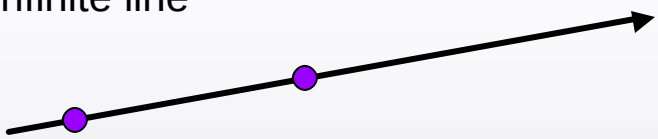
All Rays

- Plenoptic Function
  - all possible images
  - too much stuff!



Line

- Infinite line



- 4D
  - 2D direction
  - 2D position

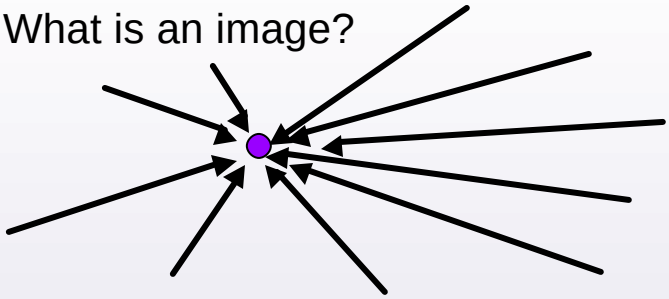
Ray

- Discretize

- Distance between 2 rays
  - Which is closer together?

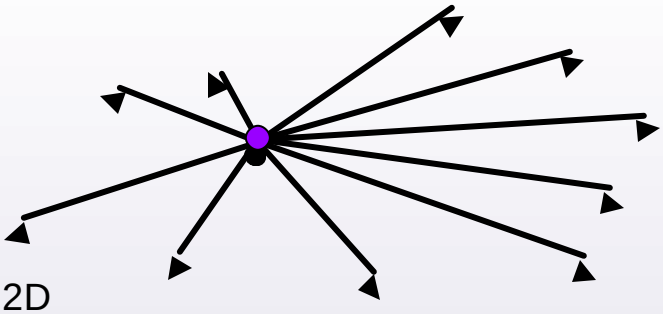
Image

- What is an image?



- All rays through a point
  - Panorama?

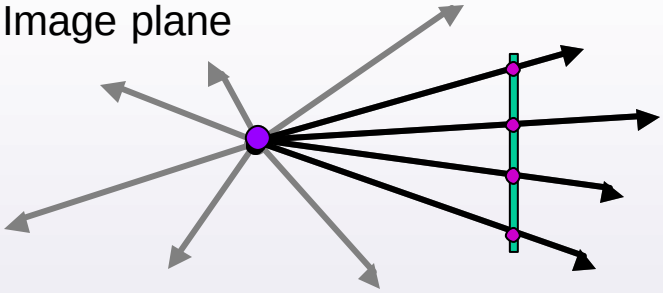
Image



- 2D
  - position of rays has been fixed
  - direction remains

Image

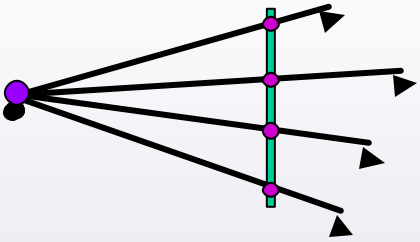
- Image plane



- 2D
  - position

Image

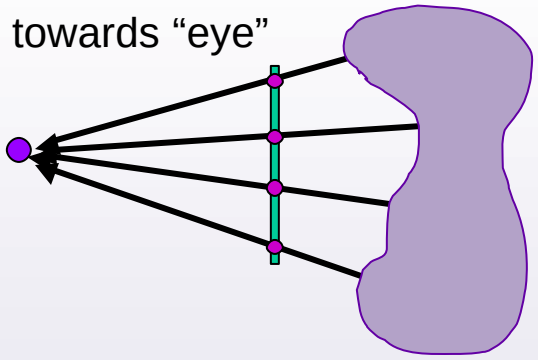
- Image plane



- 2D
  - position

Object

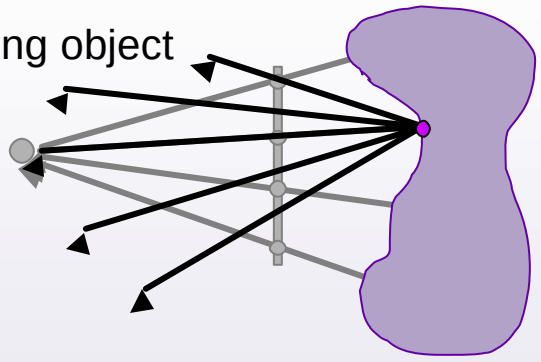
- Light leaving towards “eye”

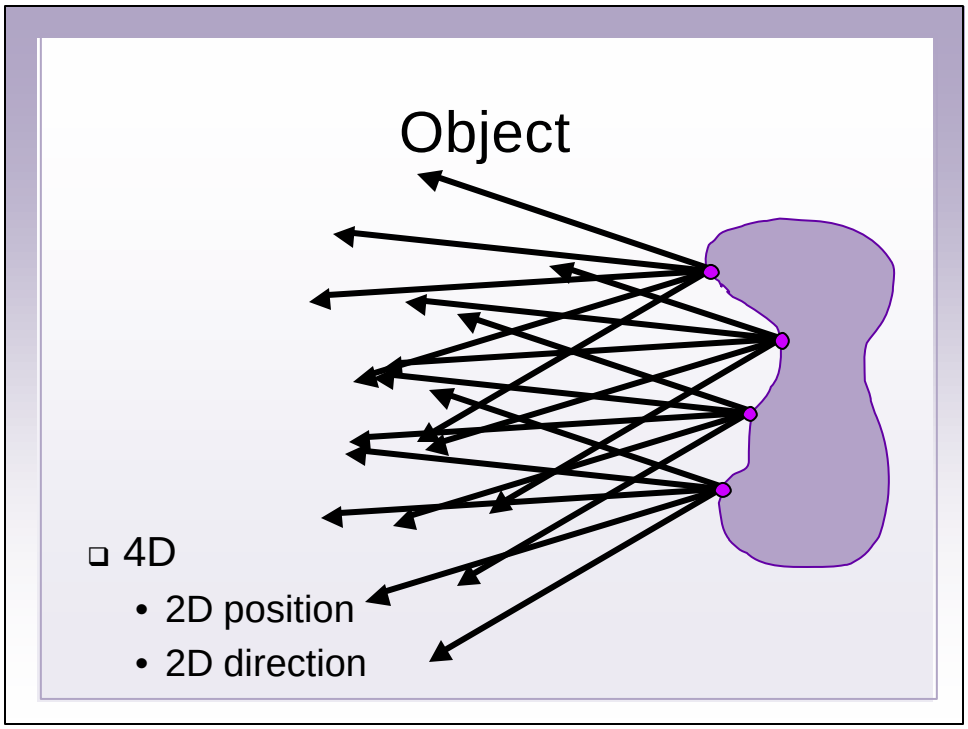


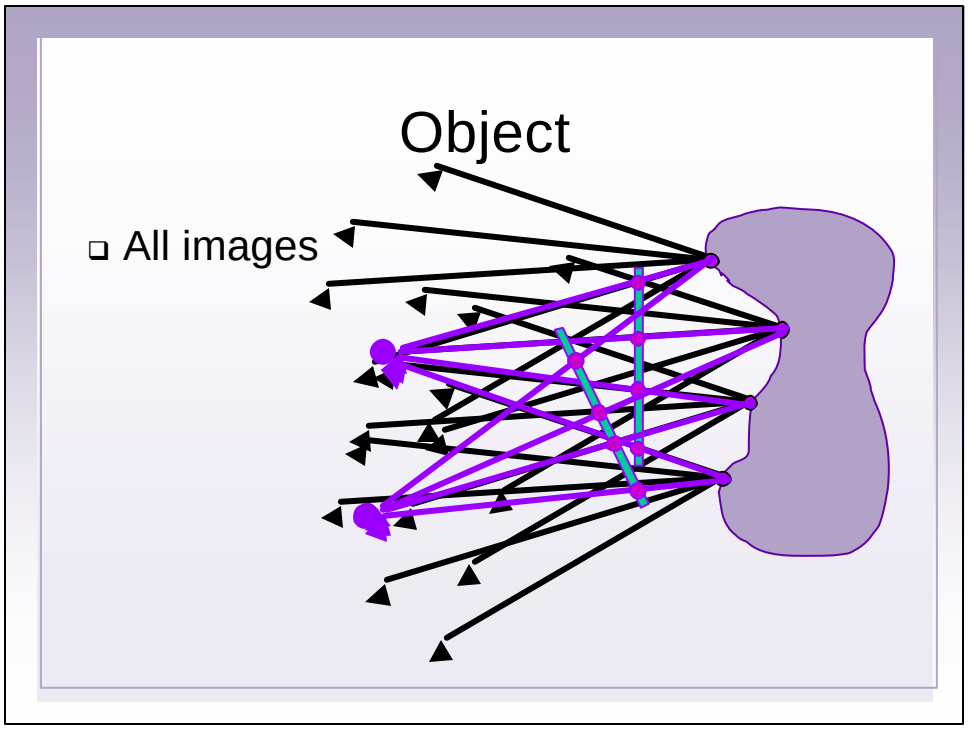
- 2D
  - just dual of image

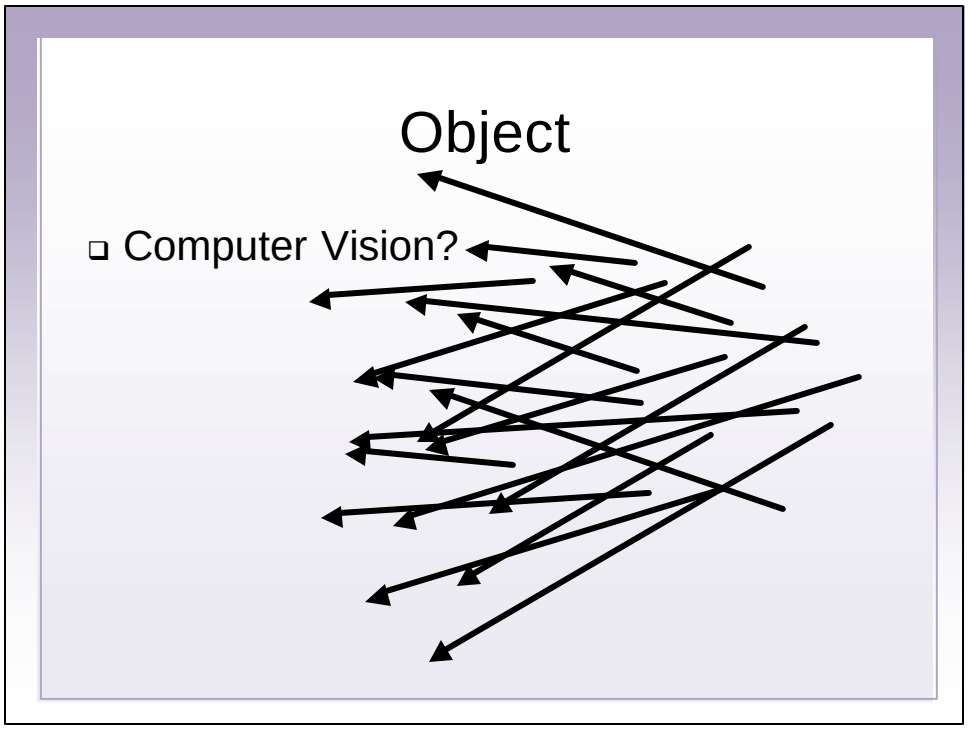
Object

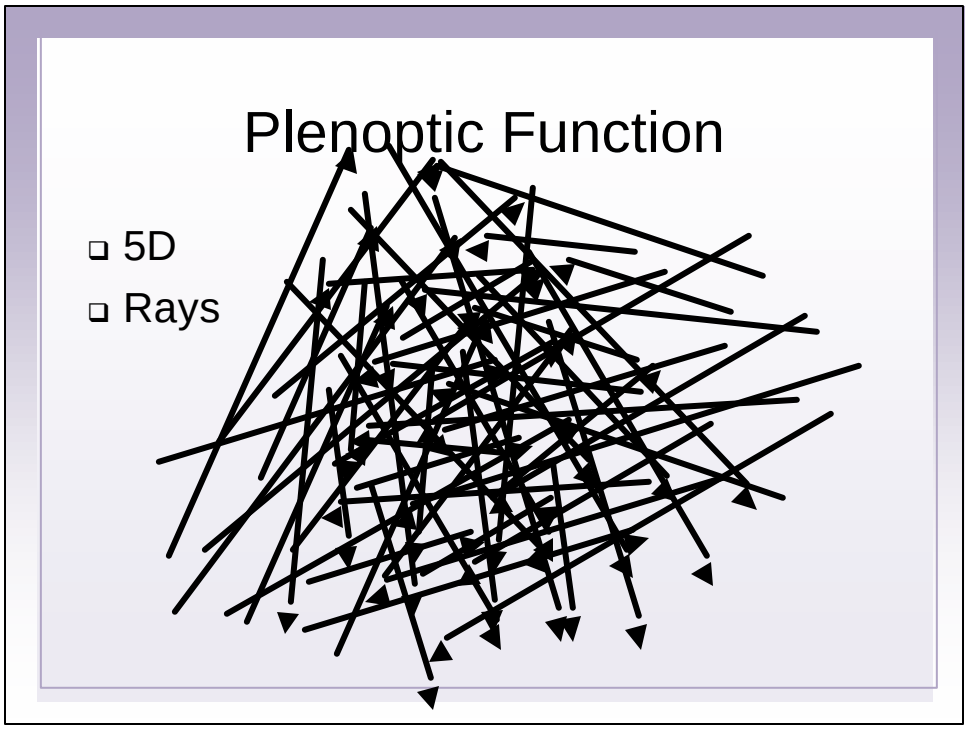
- All light leaving object

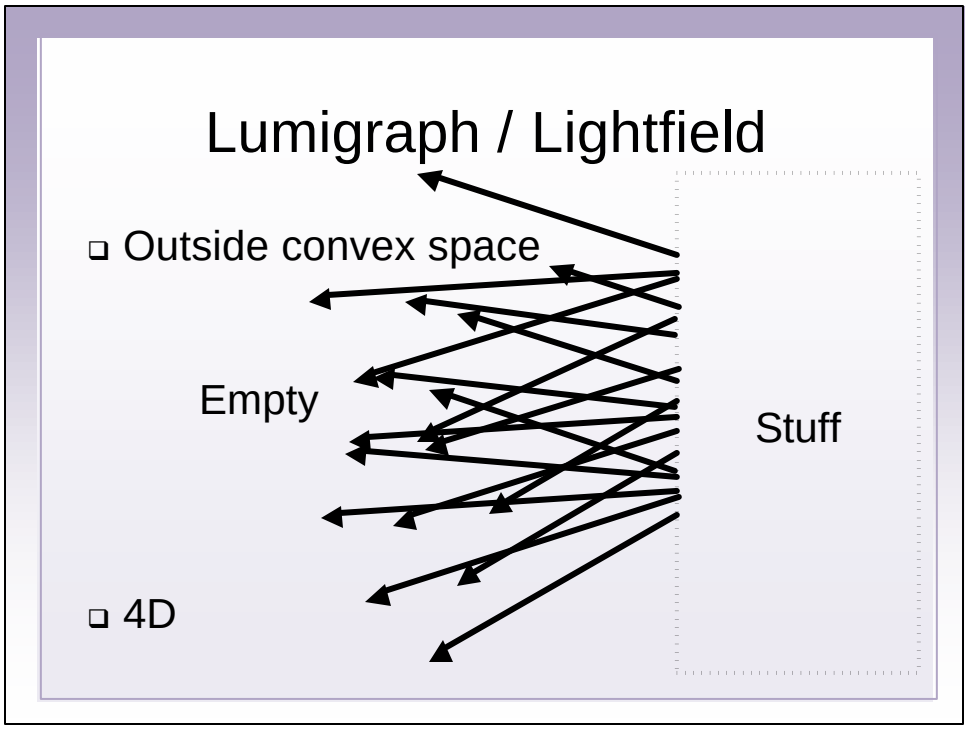


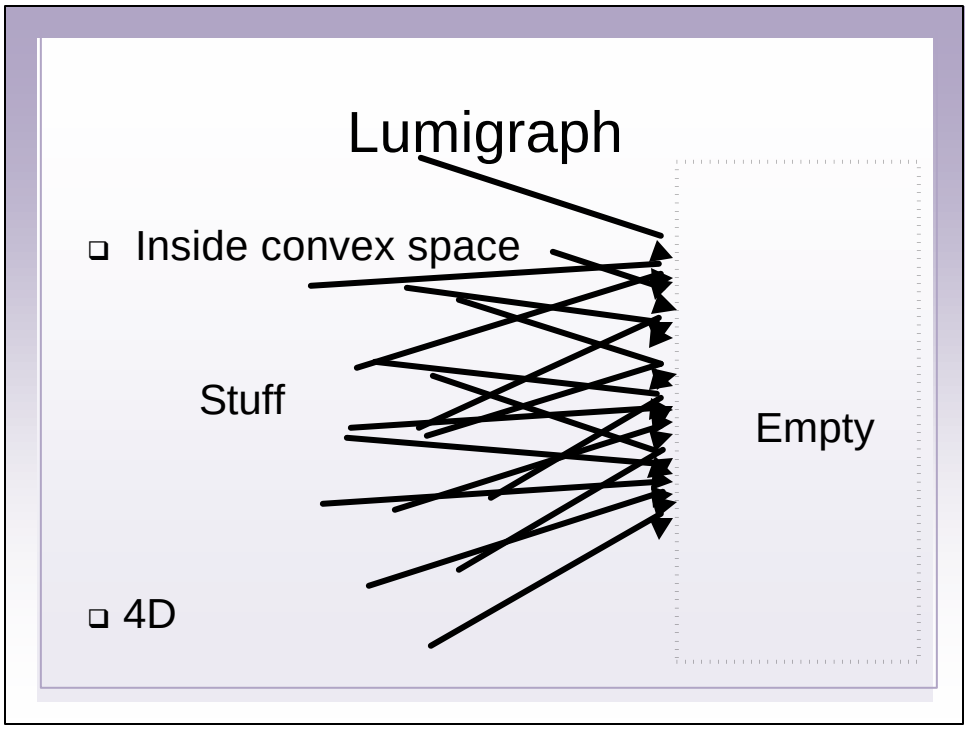

Object
4D
2D position
2D direction


Object
All images


Object
Computer Vision?

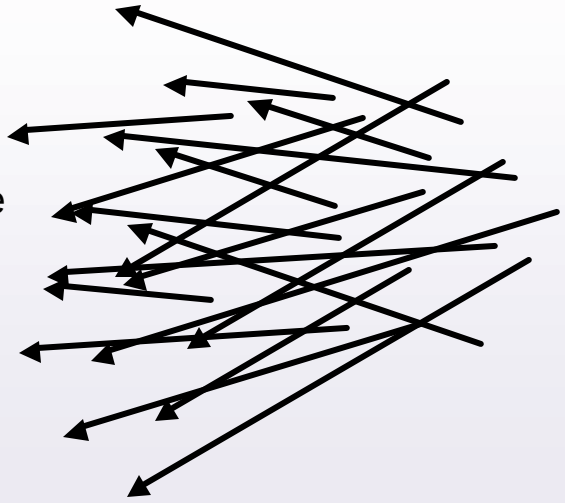

Plenoptic Function
5D
Rays


Lumigraph / Lightfield
Outside convex space
Empty
Stuff
4D


Lumigraph
Inside convex space
Stuff
Empty
4D

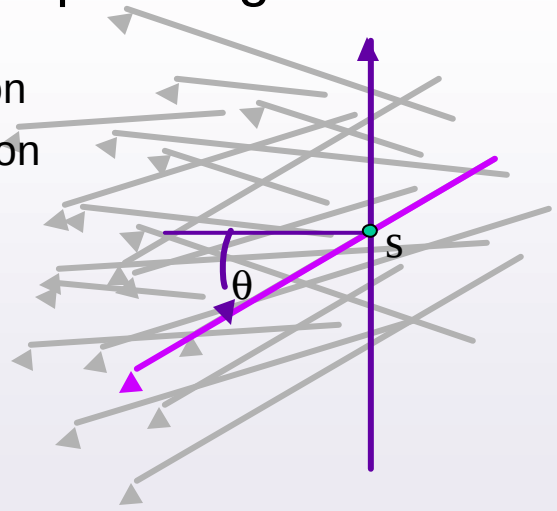
Lumigraph

- How to
  - organize
  - capture
  - render



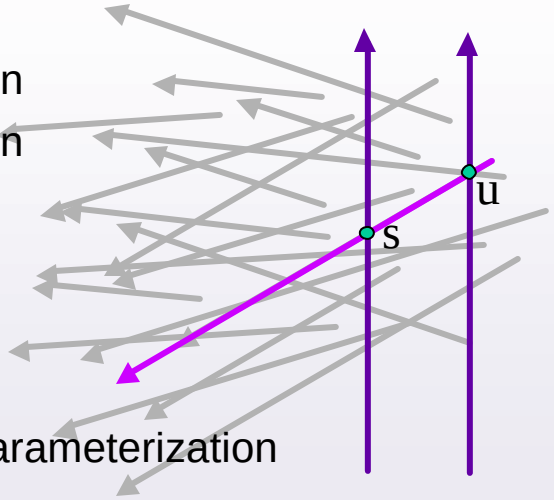
Lumigraph - Organization

- 2D position
- 2D direction



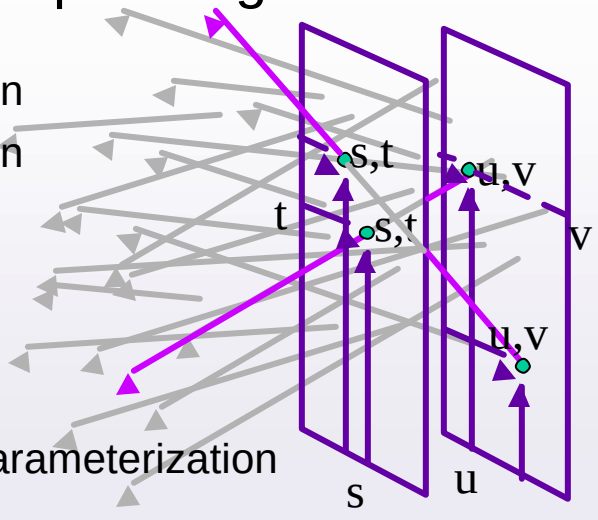
Lumigraph - Organization

- 2D position
- 2D position
- 2 plane parameterization



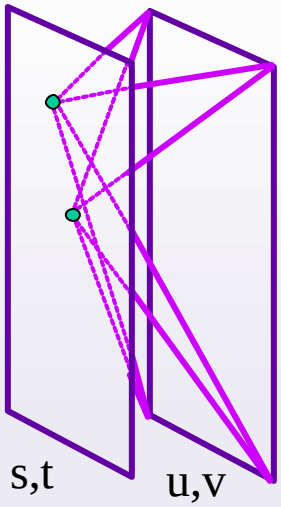
Lumigraph - Organization

- 2D position
- 2D position
- 2 plane parameterization



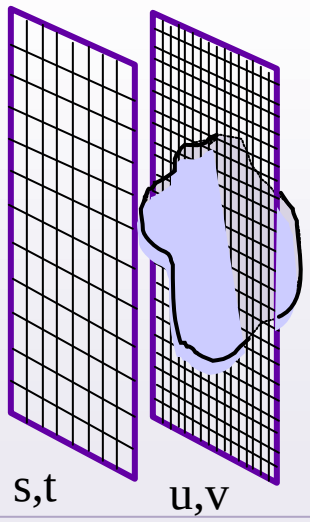
Lumigraph - Organization

- Hold s,t constant
- Let u,v vary
- An image



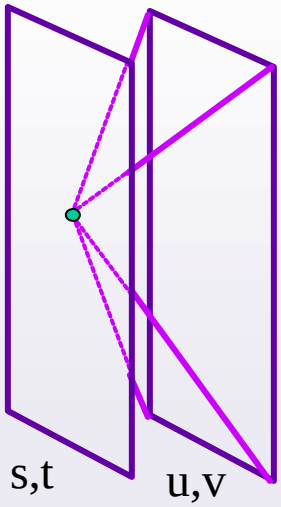
Lumigraph - Organization

- Discretization
  - higher res near object
    - if diffuse
    - captures texture
  - lower res away
    - captures directions



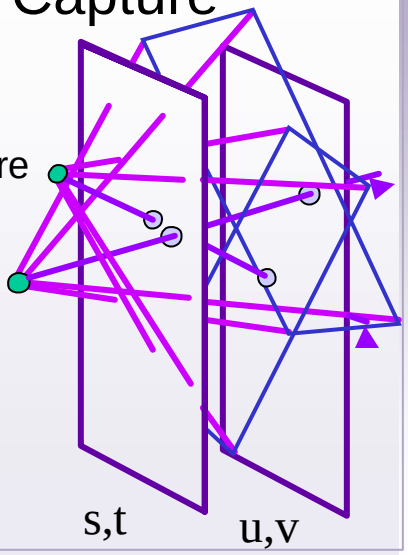
Lumigraph - Capture

- Idea 1
  - Move camera carefully over s,t plane
  - Gantry
    - see Lightfield paper



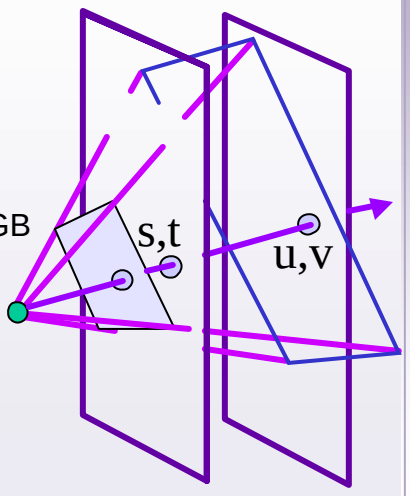
Lumigraph - Capture

- Idea 2
  - Move camera anywhere
  - Rebinning
    - see Lumigraph paper



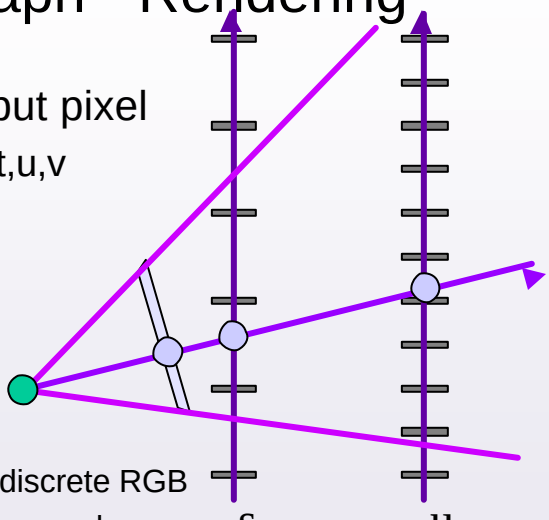
Lumigraph - Rendering

- For each output pixel
  - determine s,t,u,v
  - either
    - find closest discrete RGB
    - interpolate near values



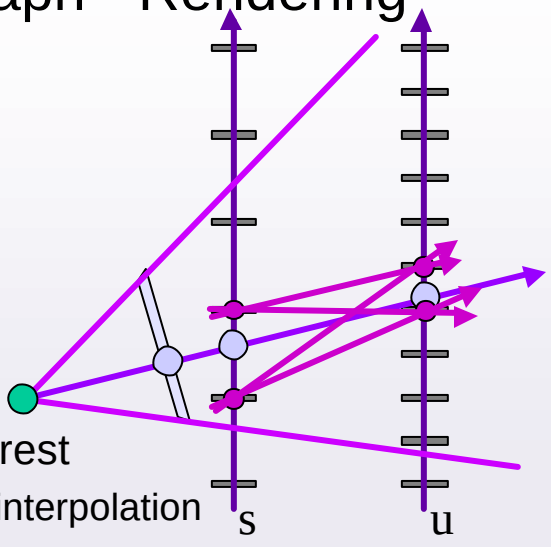
Lumigraph - Rendering

- For each output pixel
  - determine s,t,u,v
  - either
    - use closest discrete RGB
    - interpolate near values



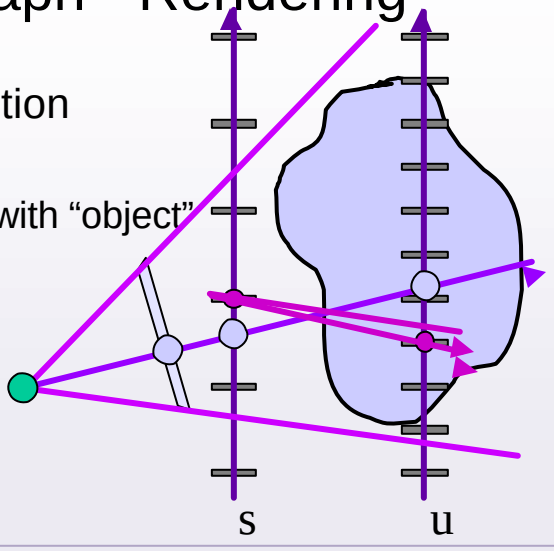
Lumigraph - Rendering

- Nearest
  - closest s
  - closest u
  - draw it
- Blend 16 nearest
  - quadrilinear interpolation



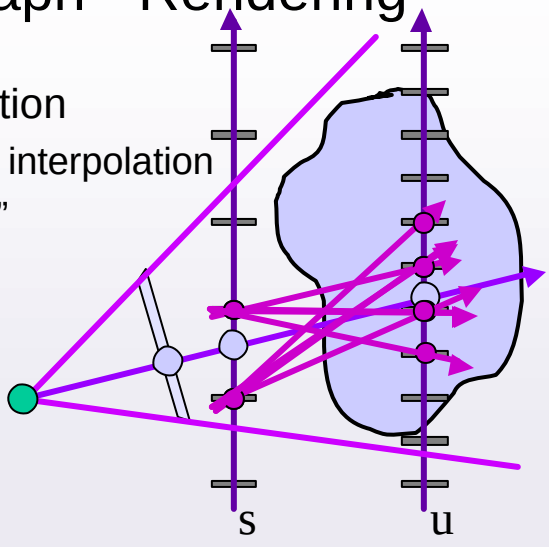
Lumigraph - Rendering

- Depth Correction
  - closest s
  - intersection with “object”
  - best u
  - closest u



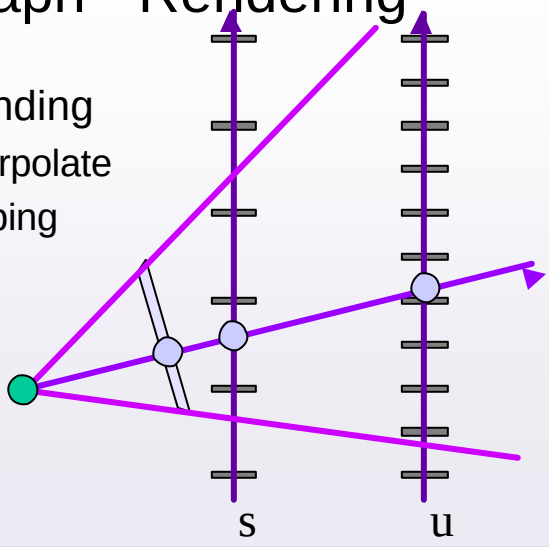
Lumigraph - Rendering

- Depth Correction
  - quadralinear interpolation
  - new “closest”
  - like focus



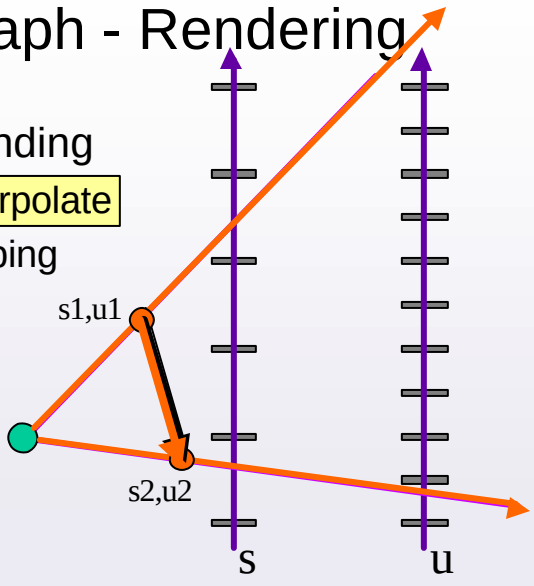
Lumigraph - Rendering

- Fast s,t,u,v finding
  - scanline interpolate
  - texture mapping
  - shear warp



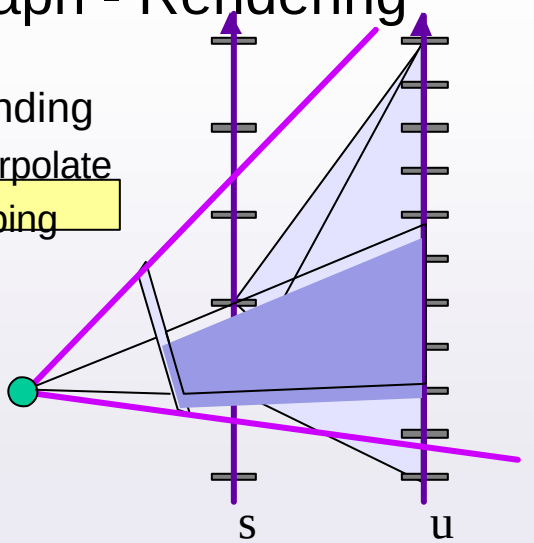
Lumigraph - Rendering

- Fast s,t,u,v finding
  - scanline interpolate
  - texture mapping
  - shear warp



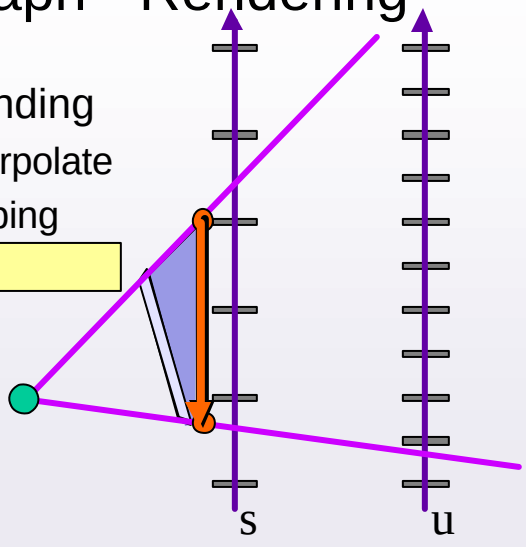
Lumigraph - Rendering

- Fast s,t,u,v finding
  - scanline interpolate
  - texture mapping
  - shear warp



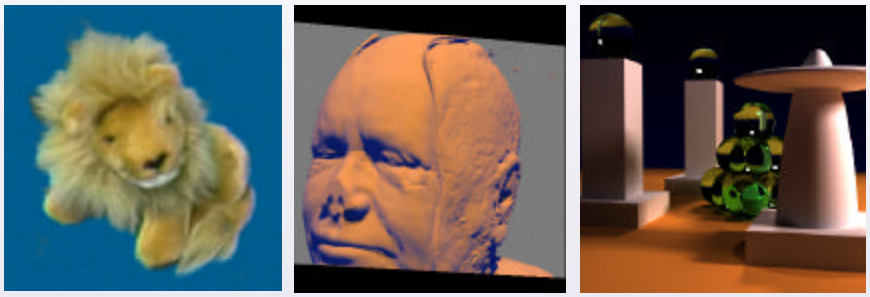
Lumigraph - Rendering

- Fast s,t,u,v finding
  - scanline interpolate
  - texture mapping
  - shear warp



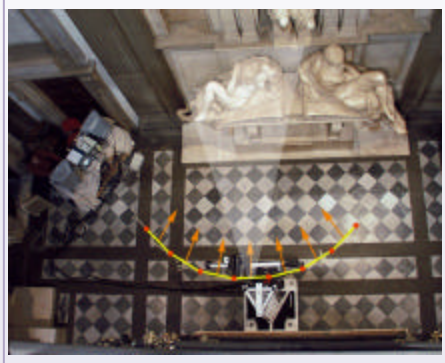
Lumigraph - Demo

- Lumigraph
  - Lion, Fruit Bowl, Visible Woman, Path Tracing



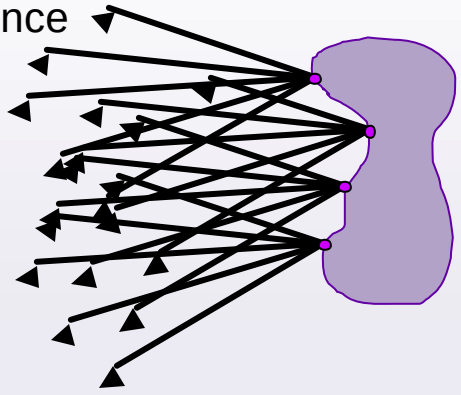
Lightfield - Demo

- Digital Michelangelo Project
  - Marc Levoy, Stanford University
  - Lightfield ("night") assembled by Jon Shade




Surface Lightfields

- Turn 4D parameterization around
- Leverage coherence



Surface Lightfields

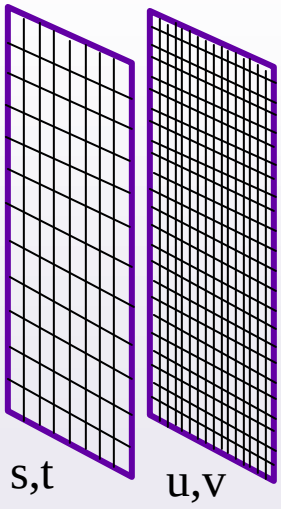
- Wood et al, SIGGRAPH 2000

3D Representations

- Image is 2D
- Lumigraph is 4D
- What happened to 3D?
  - 3D Lumigraph subset
  - Concentric mosaics

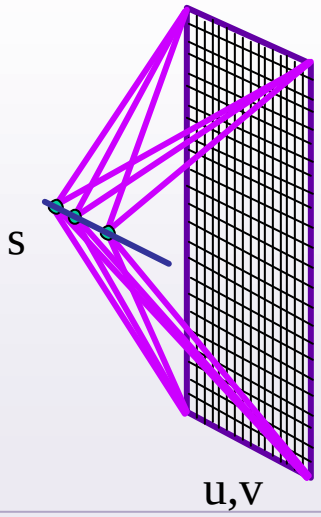
3D Lumigraph

- One row of s,t plane
  - i.e., hold t constant



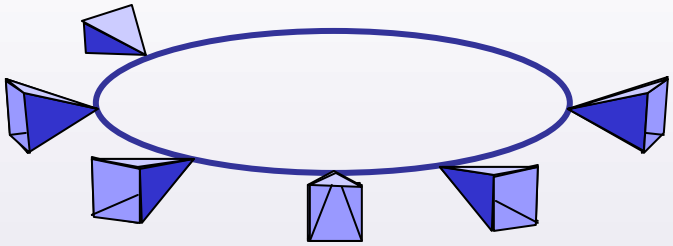
3D Lumigraph

- One row of s,t plane
  - i.e., hold t constant
  - thus s,u,v
  - a “row of images”



Concentric Mosaics

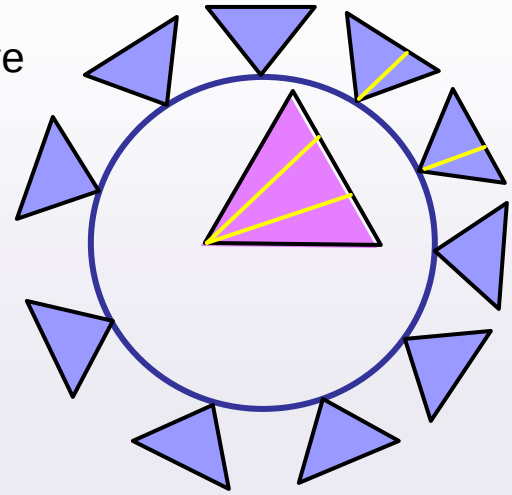
- Replace “row” with “circle” of images



Concentric Mosaics

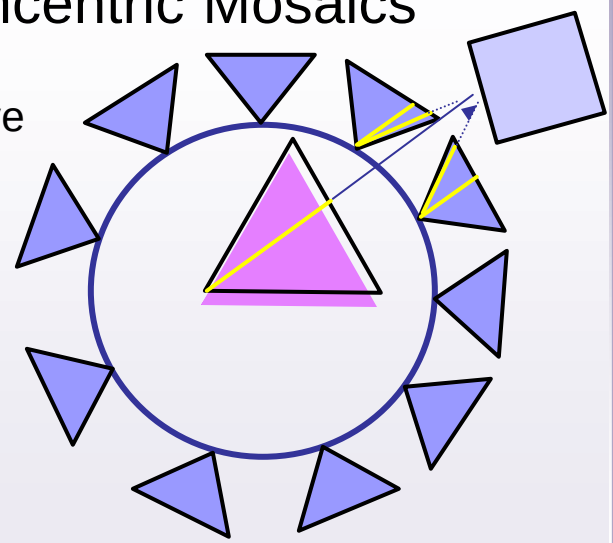
Concentric Mosaics

- From above



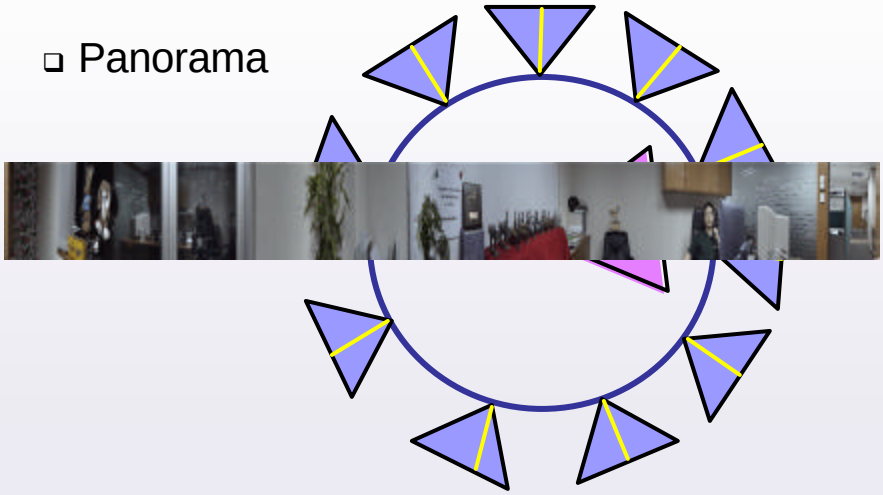
Concentric Mosaics

- From above



Concentric Mosaics

- Panorama

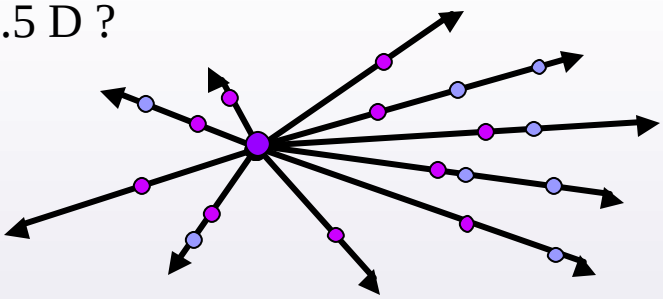


2.5D Representations

- Image is 2D
- Lumigraph is 4D
- 3D
  - 3D Lumigraph subset
  - Concentric mosaics
- 2.5D
  - Layered Depth Images
  - View Dependent Surfaces

Layered Depth Image

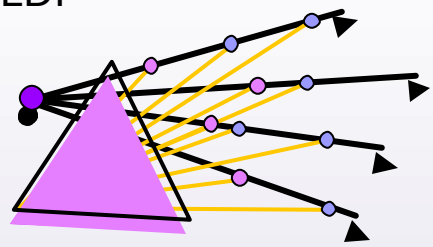
2.5 D ?



Layered Depth Image

Layered Depth Image

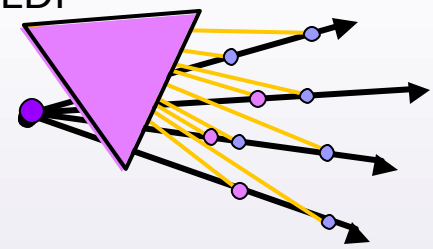
- Rendering from LDI



- Incremental in LDI X and Y
- Guaranteed to be in back-to-front order

Layered Depth Image

- Rendering from LDI

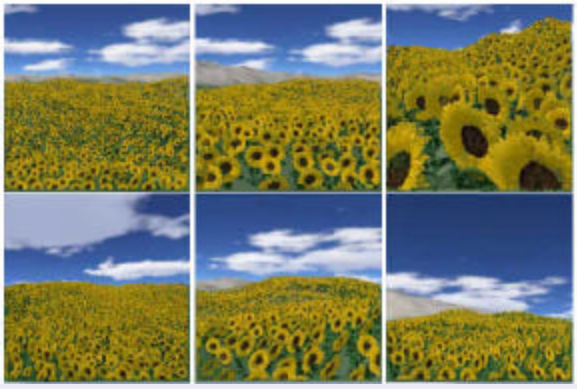


• Incremental in LDI X and Y
• Guaranteed to be in back-to-front order

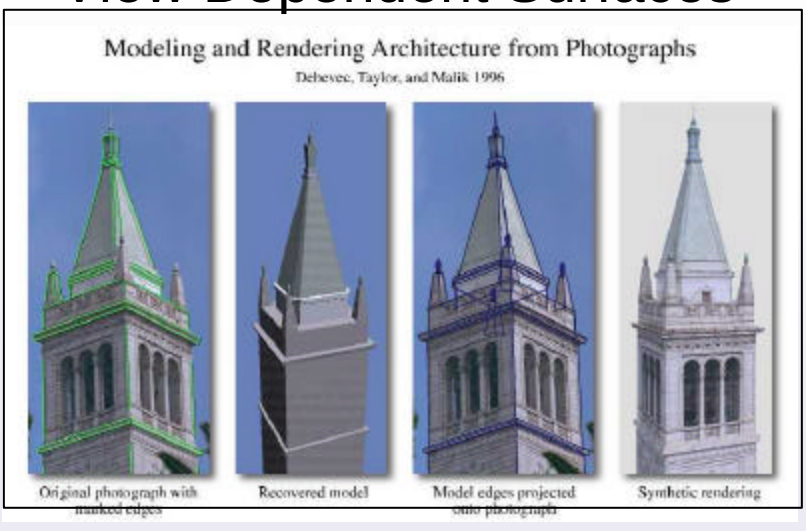
Layered Depth Image (LDI)

Tiled LDIs

- Multiresolution
- Directionally dependent



View Dependent Surfaces



Summary

- 5D: Plenoptic Function (Ray)
- 4D: Lumigraph / Lightfield
- 3D: Lumigraph Subset
- 3D: Concentric Mosaics
- 2.5D: Layered Depth Image
- 2.5D: View Dependent Models
- 2D: Image

Thanks

- Peter-Pike Sloan (Lumigraph)
- Jonathan Shade (Lightfield, LDI)
- Marc Levoy, Stanford University
  - Michaelangelo data set

Layered Depth Images

Jonathan Shade Steven Gortler* Li-wei He† Richard Szeliski‡

University of Washington **Harvard University* *†Stanford University* *‡Microsoft Research*

Abstract

In this paper we present a set of efficient image based rendering methods capable of rendering multiple frames per second on a PC. The first method warps Sprites with Depth representing smooth surfaces without the gaps found in other techniques. A second method for more general scenes performs warping from an intermediate representation called a Layered Depth Image (LDI). An LDI is a view of the scene from a single input camera view, but with multiple pixels along each line of sight. The size of the representation grows only linearly with the observed depth complexity in the scene. Moreover, because the LDI data are represented in a single image coordinate system, McMillan's warp ordering algorithm can be successfully adapted. As a result, pixels are drawn in the output image in back-to-front order. No z-buffer is required, so alpha-compositing can be done efficiently without depth sorting. This makes splatting an efficient solution to the resampling problem.

1 Introduction

Image based rendering (IBR) techniques have been proposed as an efficient way of generating novel views of real and synthetic objects. With traditional rendering techniques, the time required to render an image increases with the geometric complexity of the scene. The rendering time also grows as the requested shading computations (such as those requiring global illumination solutions) become more ambitious.

The most familiar IBR method is texture mapping. An image is remapped onto a surface residing in a three-dimensional scene. Traditional texture mapping exhibits two serious limitations. First, the pixelization of the texture map and that of the final image may be vastly different. The aliasing of the classic infinite checkerboard floor is a clear illustration of the problems this mismatch can create. Secondly, texture mapping speed is still limited by the surface the texture is applied to. Thus it would be very difficult to create a texture mapped tree containing thousands of leaves that exhibits appropriate parallax as the viewpoint changes.

Two extensions of the texture mapping model have recently been presented in the computer graphics literature that address these two difficulties. The first is a generalization of *sprites*. Once a complex scene is rendered from a particular point of view, the image that would be created from a nearby point of view will likely be similar. In this case, the original 2D image, or *sprite*, can be slightly altered by a 2D affine or projective transformation to approximate the view from the new camera position [30, 26, 14].

The sprite approximation's fidelity to the correct new view is highly dependent on the geometry being represented. In particular, the errors increase with the amount of depth variation in the real part of the scene captured by the sprite. The amount of virtual camera motion away from the point of view of sprite creation also increases the error. Errors decrease with the distance of the geometry from the virtual camera.

The second recent extension is to add depth information to an image to produce a *depth image* and to then use the optical flow that would be induced by a camera shift to warp the scene into an approximation of the new view [2, 21].

Each of these methods has its limitations. Simple sprite warping cannot produce the *parallax* induced when parts of the scenes have sizable differences in distance from the camera. Flowing a depth image pixel by pixel, on the other hand, can provide proper parallax but will result in gaps in the image either due to visibility changes when some portion of the scene become unoccluded, or when a surface is magnified in the new view.

Some solutions have been proposed to the latter problem. Laveau and Faugeras suggest performing a backwards mapping from the output sample location to the input image [13]. This is an expensive operation that requires some amount of searching in the input image. Another possible solution is to think of the input image as a mesh of micro-polygons, and to scan-convert these polygons in the output image. This is an expensive operation, as it requires a polygon scan-convert setup for each input pixel [17], an operation we would prefer to avoid especially in the absence of specialized rendering hardware. Alternatively one could use multiple input images from different viewpoints. However, if one uses n input images, one effectively multiplies the size of the scene description by n, and the rendering cost increases accordingly.

This paper introduces two new extensions to overcome both of these limitations. The first extension is primarily applicable to smoothly varying surfaces, while the second is useful primarily for very complex geometries. Each method provides efficient image based rendering capable of producing multiple frames per second on a PC.

In the case of sprites representing smoothly varying surfaces, we introduce an algorithm for rendering *Sprites with Depth*. The algorithm first forward maps (i.e., warps) the depth values themselves and then uses this information to add parallax corrections to a standard sprite renderer.

For more complex geometries, we introduce the *Layered Depth Image*, or LDI, that contains potentially multiple depth pixels at each discrete location in the image. Instead of a 2D array of depth pixels (a pixel with associated depth information), we store a 2D array of layered depth pixels. A layered depth pixel stores a set of depth pixels along one line of sight sorted in front to back order. The front element in the layered depth pixel samples the first surface seen along that line of sight; the next pixel in the layered depth pixel samples the next surface seen along that line of sight, etc. When rendering from an LDI, the requested view can move away from the original LDI view and expose surfaces that were not visible in the first layer. The previously occluded regions may still be rendered from data stored in some later layer of a layered depth pixel.

There are many advantages to this representation. The size of the

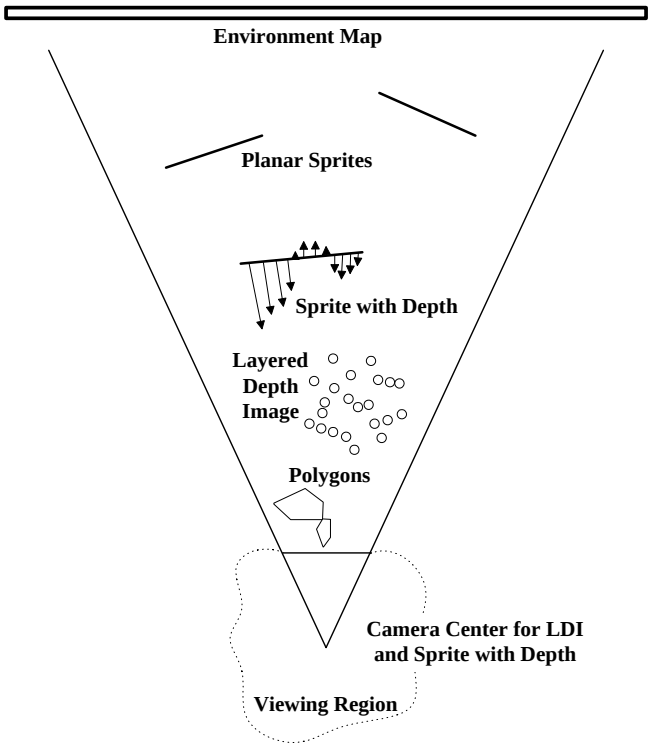


Figure 1 Different image based primitives can serve well depending on distance from the camera

representation grows linearly only with the depth complexity of the image. Moreover, because the LDI data are represented in a single image coordinate system, McMillan's ordering algorithm [20] can be successfully applied. As a result, pixels are drawn in the output image in back to front order allowing proper alpha blending without depth sorting. No z-buffer is required, so alpha-compositing can be done efficiently without explicit depth sorting. This makes splatting an efficient solution to the reconstruction problem.

Sprites with Depth and Layered Depth Images provide us with two new image based primitives that can be used in combination with traditional ones. Figure 1 depicts five types of primitives we may wish to use. The camera at the center of the frustum indicates where the image based primitives were generated from. The viewing volume indicates the range one wishes to allow the camera to move while still re-using these image based primitives.

The choice of which type of image-based or geometric primitive to use for each scene element is a function of its distance, its internal depth variation relative to the camera, as well as its internal geometric complexity. For scene elements at a great distance from the camera one might simply generate an environment map. The environment map is invariant to translation and simply translates as a whole on the screen based on the rotation of the camera. At a somewhat closer range, and for geometrically planar elements, traditional planar sprites (or *image caches*) may be used [30, 26]. The assumption here is that although the part of the scene depicted in the sprite may display some parallax relative to the background environment map and other sprites, it will not need to depict any parallax within the sprite itself. Yet closer to the camera, for elements with smoothly varying depth, Sprites with Depth are capable of displaying internal parallax but cannot deal with disocclusions due to image flow that may arise in more complex geometric scene elements. Layered Depth Images deal with both parallax and disocclusions and are thus useful for objects near the camera that also contain complex geometries that will exhibit considerable parallax. Finally, traditional polygon rendering may need to be used for immediate foreground objects.

In the sections that follow, we will concentrate on describing the data structures and algorithms for representing and rapidly rendering Sprites with Depth and Layered Depth Images.

2 Previous Work

Over the past few years, there have been many papers on image based rendering. In [16], Levoy and Whitted discuss rendering point data. Chen and Williams presented the idea of rendering from images [2]. Laveau and Faugeras discuss IBR using a backwards map [13]. McMillan and Bishop discuss IBR using cylindrical views [21]. Seitz and Dyer describe a system that allows a user to correctly model view transforms in a user controlled image morphing system [28]. In a slightly different direction, Levoy and Hanrahan [15] and Gortler *et al.* [7] describe IBR methods using a large number of input images to sample the high dimensional radiance function.

Max uses a representation similar to an LDI [18], but for a purpose quite different than ours; his purpose is high quality anti-aliasing, while our goal is efficiency. Max reports his rendering time as 5 minutes per frame while our goal is multiple frames per second. Max warps from *n* input LDIs with different camera information; the multiple depth layers serve to represent the high depth complexity of trees. We warp from a single LDI, so that the warping can be done most efficiently. For output, Max warps to an LDI. This is done so that, in conjunction with an A-buffer, high quality, but somewhat expensive, anti-aliasing of the output picture can be performed.

Mark *et al.*[17] and Darsa *et al.*[4] create triangulated depth maps from input images with per-pixel depth. Darsa concentrates on limiting the number of triangles by looking for depth coherence across regions of pixels. This triangle mesh is then rendered traditionally taking advantage of graphics hardware pipelines. Mark *et al.*describe the use of multiple input images as well. In this aspect of their work, specific triangles are given lowered priority if there is a large discontinuity in depth across neighboring pixels. In this case, if another image fills in the same area with a triangle of higher priority, it is used instead. This helps deal with disocclusions.

Shade *et al.*[30] and Shaufler *et al.*[26] render complex portions of a scene such as a tree onto alpha matted billboard-like sprites

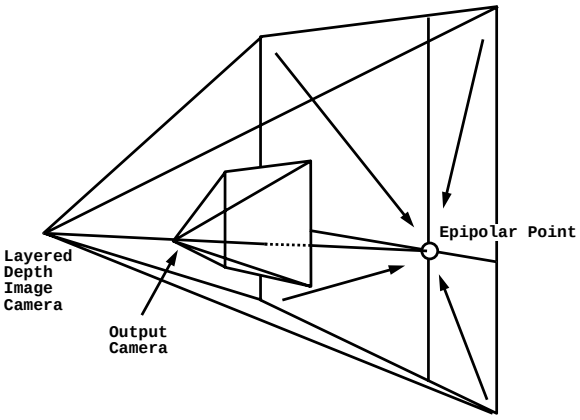


Figure 2 Back to front output ordering

and then reuse them as textures in subsequent frames. Lengyel and Snyder [14] extend this work by warping sprites by a best fit affine transformation based on a set of sample points in the underlying 3D model. These affine transforms are allowed to vary in time as the position and/or color of the sample points change. Hardware considerations for such system are discussed in [31].

Horry *et al.* [10] describe a very simple sprite-like system in which a user interactively indicates planes in an image that represent areas in a given image. Thus, from a single input image and some user supplied information, they can warp an image and provide approximate three dimensional cues about the scene.

The system presented here relies heavily on McMillan's ordering algorithm [20, 19, 21]. Using input and output camera information, a warping order is computed such that pixels that map to the same location in the output image are guaranteed to arrive in back to front order.

In McMillan's work, the depth order is computed by first finding the projection of the output camera's location in the input camera's image plane, that is, the intersection of the line joining the two camera locations with the input camera's image plane. The line joining the two camera locations is called the epipolar line, and the intersection with the image plane is called an epipolar point [6] (see Figure 1). The input image is then split horizontally and vertically at the epipolar point, generally creating 4 image quadrants. (If the epipolar point lies off the image plane, we may have only 2 or 1 regions.) The pixels in each of the quadrants are processed in a different order. Depending on whether the output camera is in front of or behind the input camera, the pixels in each quadrant are processed either inward towards the epipolar point or outwards away from it. In other words, one of the quadrants is processed left to right, top to bottom, another is processed left to right, bottom to top, etc. McMillan discusses in detail the various special cases that arise and proves that this ordering is guaranteed to produce depth ordered output [19].

When warping from an LDI, there is effectively only one input camera view. Therefore one can use the ordering algorithm to order the layered depth pixels visited. Within each layered depth pixel, the layers are processed in back to front order. The formal proof of [19] applies, and the ordering algorithm is guaranteed to work.

3 Rendering Sprites

Sprites are texture maps or images with alphas (transparent pixels) rendered onto planar surfaces. They can be used either for locally caching the results of slower rendering and then generating new views by warping [30, 26, 31, 14], or they can be used directly as drawing primitives (as in video games).

The texture map associated with a sprite can be computed by simply choosing a 3D viewing matrix and projecting some portion of the scene onto the image plane. In practice, a view associated with the current or expected viewpoint is a good choice. A 3D plane equation can also be computed for the sprite, e.g., by fitting a 3D plane to the z-buffer values associated with the sprite pixels. Below, we derive the equations for the 2D perspective mapping between a sprite and its novel view. This is useful both for implementing a backward mapping algorithm, and lays the foundation for our Sprites with Depth rendering algorithm.

A sprite consists of an alpha-matted image $I_1(x_1, y_1)$, a 4×4 camera matrix C_1 which maps from 3D world coordinates $(X, Y, Z, 1)$ into the sprite's coordinates $(x_1, y_1, z_1, 1)$,

$$\begin{bmatrix} w_1 x_1 \\ w_1 y_1 \\ w_1 z_1 \\ w_1 \end{bmatrix} = C_1 \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{1}$$

(z_1 is the z-buffer value), and a plane equation. This plane equation can either be specified in world coordinates, $AX + BY + CZ + D = 0$, or it can be specified in the sprite's coordinate system, $ax_1 + by_1 + cz_1 + d = 0$. In the former case, we can form a new camera matrix $\hat{C}_1$ by replacing the third row of C_1 with the row $[A\ B\ C\ D]$, while in the latter, we can compute $\hat{C}_1 = PC_1$, where

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & b & c & d \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(note that $[A\ B\ C\ D] = [a\ b\ c\ d]C_1$).

In either case, we can write the modified projection equation as

$$\begin{bmatrix} w_1 x_1 \\ w_1 y_1 \\ w_1 d_1 \\ w_1 \end{bmatrix} = \hat{C}_1 \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{2}$$

where $d_1 = 0$ for pixels on the plane. For pixels off the plane, d_1 is the scaled perpendicular distance to the plane (the scale factor is 1 if $A^2 + B^2 + C^2 = 1$) divided by the pixel to camera distance w_1.

Given such a sprite, how do we compute the 2D transformation associated with a novel view $\hat{C}_2$? The mapping between pixels $(x_1, y_1, d_1, 1)$ in the sprite and pixels $(w_2 x_2, w_2 y_2, w_2 d_2, w_2)$ in the output camera's image is given by the transfer matrix $T_{1,2} = \hat{C}_2 \cdot \hat{C}_1^{-1}$.

For a flat sprite ($d_1 = 0$), the transfer equation can be written as

$$\begin{bmatrix} w_2 x_2 \\ w_2 y_2 \\ w_2 \end{bmatrix} = H_{1,2} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{3}$$

where $H_{1,2}$ is the 2D planar perspective transformation (*homography*) obtained by dropping the third row and column of $T_{1,2}$. The coordinates (x_2, y_2) obtained after dividing out w_2 index a pixel address in the output camera's image. Efficient backward mapping techniques exist for performing the 2D perspective warp [8, 34], or texture mapping hardware can be used.

3.1 Sprites with Depth

The descriptive power (realism) of sprites can be greatly enhanced by adding an out-of-plane displacement component d_1 at each pixel in the sprite.[1] Unfortunately, such a representation can no longer be rendered directly using a backward mapping algorithm.

Using the same notation as before, we see that the transfer equation is now

$$\begin{bmatrix} w_2 x_2 \\ w_2 y_2 \\ w_2 \end{bmatrix} = H_{1,2} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} + d_1 e_{1,2}, \tag{4}$$

[1] The d_1 values can be stored as a separate image, say as 8-bit signed depths. The full precision of a traditional z-buffer is not required, since these depths are used only to compute local parallax, and not to perform z-buffer merging of primitives. Furthermore, the d_1 image could be stored at a lower resolution than the color image, if desired.

where $e_{1,2}$ is called *epipole* [6, 25, 11], and is obtained from the third column of $T_{1,2}$.

Equation (4) can be used to *forward map* pixels from a sprite to a new view. Unfortunately, this entails the usual problems associated with forward mapping, e.g., the necessity to fill gaps or to use larger splatting kernels, and the difficulty in achieving proper resampling. Notice, however, that Equation (4) could be used to perform a backward mapping step by interchanging the 1 and 2 indices, if only we knew the displacements d_2 in the output camera's coordinate frame.

A solution to this problem is to first *forward map* the displacements d_1, and to then use Equation (4) to perform a backward mapping step with the new (view-based) displacements. While this may at first appear to be no faster or more accurate than simply forward warping the color values, it does have some significant advantages.

First, small errors in displacement map warping will not be as evident as errors in the sprite image warping, at least if the displacement map is smoothly varying (in practice, the shape of a simple surface often varies more smoothly than its photometry). If bilinear or higher order filtering is used in the final color (backward) resampling, this two-stage warping will have much lower errors than forward mapping the colors directly with an inaccurate forward map. We can therefore use a quick single-pixel splat algorithm followed by a quick hole filling, or alternatively, use a simple 2×2 splat.

The second main advantage is that we can design the forward warping step to have a simpler form by factoring out the planar perspective warp. Notice that we can rewrite Equation (4) as

$$\begin{bmatrix} w_2 x_2 \\ w_2 y_2 \\ w_2 \end{bmatrix} = H_{1,2} \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix}, \tag{5}$$

with

$$\begin{bmatrix} w_3 x_3 \\ w_3 y_3 \\ w_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} + d_1 e^*_{1,2}, \tag{6}$$

where $e^*_{1,2} = H^{-1}_{1,2} e_{1,2}$. This suggests that Sprite with Depth rendering can be implemented by first shifting pixels by their local parallax, filling any resulting gaps, and then applying a global homography (planar perspective warp). This has the advantage that it can handle large changes in view (e.g., large zooms) with only a small amount of gap filling (since gaps arise only in the first step, and are due to variations in displacement).

Our novel two-step rendering algorithm thus proceeds in two stages:

1. forward map the displacement map $d_1(x_1, y_1)$, using only the parallax component given in Equation (6) to obtain $d_3(x_3, y_3)$;

2a. backward map the resulting warped displacements $d_3(x_3, y_3)$ using Equation (5) to obtain $d_2(x_2, y_2)$ (the displacements in the new camera view);

2b. backward map the original sprite colors, using both the homography $H_{2,1}$ and the new parallax d_2 as in Equation (4) (with the 1 and 2 indices interchanged), to obtain the image corresponding to camera C_2.

The last two operations can be combined into a single raster scan over the output image, avoiding the need to perspective warp d_3 into d_2. More precisely, for each output pixel (x_2, y_2), we compute (x_3, y_3) such that

$$\begin{bmatrix} w_3 x_3 \\ w_3 y_3 \\ w_3 \end{bmatrix} = H_{2,1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \tag{7}$$

to compute where to look up the displacement $d_3(x_3, y_3)$, and form the final address of the source sprite pixel using

$$\begin{bmatrix} w_1 x_1 \\ w_1 y_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} w_3 x_3 \\ w_3 y_3 \\ w_3 \end{bmatrix} + d_3(x_3, y_3) e_{2,1}. \tag{8}$$

We can obtain a quicker, but less accurate, algorithm by omitting the first step, i.e., the pure parallax warp from d_1 to d_3. If we assume the depth at a pixel before and after the warp will not change significantly, we can use d_1 instead of d_3 in Equation (8). This still gives a useful illusion of 3-D parallax, but is only valid for a much smaller range of viewing motions (see Figure 3).

Another variant on this algorithm, which uses somewhat more storage but fewer computations, is to compute a 2-D displacement field in the first pass, $u_3(x_3, y_3) = x_1 - x_3$, $v_3(x_3, y_3) = y_1 - y_3$, where (x_3, y_3) is computed using the pure parallax transform in Equation (6). In the second pass, the final pixel address in the sprite is computed using

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} + \begin{bmatrix} u_3(x_3, y_3) \\ v_3(x_3, y_3) \end{bmatrix}, \tag{9}$$

where this time (x_3, y_3) is computed using the transform given in Equation (7).

We can make the pure parallax transformation (6) faster by avoiding the per-pixel division required after adding homogeneous coordinates. One way to do this is to approximate the parallax transformation by first moving the epipole to infinity (setting its third component to 0). This is equivalent to having an *affine* parallax component (all points move in the same direction, instead of towards a common vanishing point). In practice, we find that this still provides a very compelling illusion of 3D shape.

Figure 3 shows some of the steps in our two-pass warping algorithm. Figures 3a and 3f show the original sprite (color) image and the depth map. Figure 3b shows the sprite warped with no parallax. Figures 3g, 3h, and 3i shows the depth map forward warped with only pure parallax, only the perspective projection, and both. Figure 3c shows the backward warp using the incorrect depth map d_1 (note how dark "background" colors are mapped onto the "bump"), whereas Figure 3d shows the backward warp using the correct depth map d_3. The white pixels near the right hand edge are a result of using only a single step of gap filling. Using three steps results in the better quality image shown in Figure 3e. Gaps also do not appear for a less quickly slanting d maps, such as the pyramid shown in Figure 3j.

The rendering times for the 256×256 image shown in Figure 3 on a 300 MHz Pentium II are as follows. Using bilinear pixel sampling, the frame rates are 30 Hz for no z-parallax, 21 Hz for "crude" one-pass warping (no forward warping of d_1 values), and 16 Hz for two-pass warping. Using nearest-neighbor resampling, the frame rates go up to 47 Hz, 24 Hz, and 20 Hz, respectively.

3.2 Recovering sprites from image sequences

While sprites and sprites with depth can be generated using computer graphics techniques, they can also be extracted from image sequences using computer vision techniques. To do this, we use a layered motion estimation algorithm [32, 1], which simultaneously segments the sequence into coherently moving regions, and computes a parametric motion estimate (planar perspective transformation) for each layer. To convert the recovered layers into sprites, we need to determine the plane equation associated with each region. We do this by tracking features from frame to frame and applying

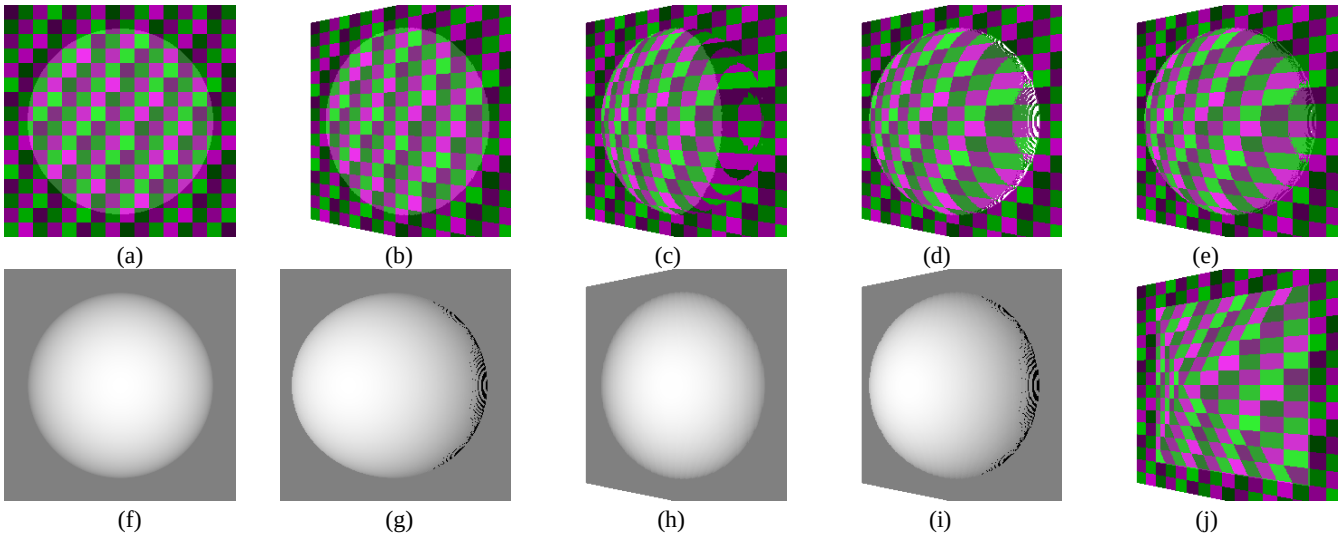


Figure 3 Plane with bump rendering example: (a) input color (sprite) image $I_1(x_1, y_1)$; (b) sprite warped by homography only (no parallax); (c) sprite warped by homography and crude parallax (d_1); (d) sprite warped by homography and true parallax (d_2); (e) with gap fill width set to 3; (f) input depth map $d_1(x_1, y_1)$; (g) pure parallax warped depth map $d_3(x_3, y_3)$; (h) forward warped depth map $d_2(x_2, y_2)$; (i) forward warped depth map without parallax correction; (j) sprite with "pyramid" depth map.

Figure 4 Results of sprite extraction from image sequence: (a) third of five images; (b) initial segmentation into six layers; (c) recovered depth map; (d) the five layer sprites; (e) residual depth image for fifth layer; (f) re-synthesized third image (note extended field of view); (g) novel view without residual depth; (h) novel view with residual depth (note the "rounding" of the people).

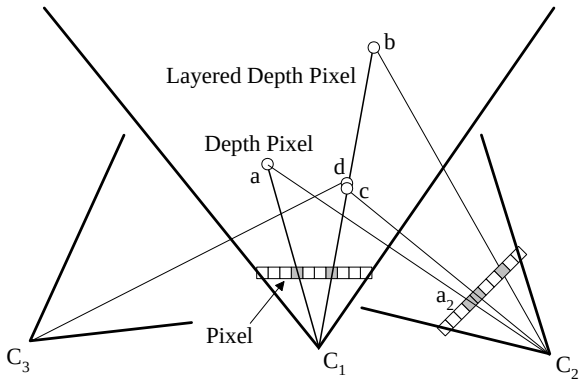


Figure 5 Layered Depth Image

a standard structure from motion algorithm to recover the camera parameters (viewing matrices) for each frame [6]. Tracking several points on each sprite enables us to reconstruct their 3D positions, and hence to estimate their 3D plane equations [1]. Once the sprite pixel assignment have been recovered, we run a traditional stereo algorithm to recover the out-of-plane displacements.

The results of applying the layered motion estimation algorithm to the first five images from a 40-image stereo dataset[2] are shown in Figure 4. Figure 4(a) shows the middle input image, Figure 4(b) shows the initial pixel assignment to layers, Figure 4(c) shows the recovered depth map, and Figure 4(e) shows the residual depth map for layer 5. Figure 4(d) shows the recovered sprites. Figure 4(f) shows the middle image re-synthesized from these sprites, while Figures 4(g–h) show the same sprite collection seen from a novel viewpoint (well outside the range of the original views), both with and without residual depth-based correction (parallax). The gaps visible in Figures 4(c) and 4(f) lie *outside* the area corresponding to the middle image, where the appropriate parts of the background sprites could not be seen.

4 Layered Depth Images

While the use of sprites and Sprites with Depth provides a fast means to warp planar or smoothly varying surfaces, more general scenes require the ability to handle more general disocclusions and large amounts of parallax as the viewpoint moves. These needs have led to the development of Layered Depth Images (LDI).

Like a sprite with depth, pixels contain depth values along with their colors (i.e., a *depth pixel*). In addition, a Layered Depth Image (Figure 5) contains potentially multiple depth pixels per pixel location. The farther depth pixels, which are occluded from the LDI center, will act to fill in the disocclusions that occur as the viewpoint moves away from the center.

The structure of an LDI is summarized by the following conceptual representation:

```
DepthPixel =
    ColorRGBA: 32 bit integer
    Z: 20 bit integer
    SplatIndex: 11 bit integer

LayeredDepthPixel =
    NumLayers: integer
    Layers[0..numlayers-1]: array of DepthPixel
```

[2]Courtesy of Dayton Taylor.

```
LayeredDepthImage =
    Camera: camera
    Pixels[0..xres-1,0..yres-1]: array of LayeredDepthPixel
```

The layered depth image contains camera information plus an array of size *xres* by *yres* layered depth pixels. In addition to image data, each layered depth pixel has an integer indicating how many valid depth pixels are contained in that pixel. The data contained in the depth pixel includes the color, the depth of the object seen at that pixel, plus an index into a table that will be used to calculate a splat size for reconstruction. This index is composed from a combination of the normal of the object seen and the distance from the LDI camera.

In practice, we implement Layered Depth Images in two ways. When creating layered depth images, it is important to be able to efficiently insert and delete layered depth pixels, so the *Layers* array in the *LayeredDepthPixel* structure is implemented as a linked list. When rendering, it is important to maintain spatial locality of depth pixels in order to most effectively take advantage of the cache in the CPU [12]. In Section 5.1 we discuss the compact render-time version of layered depth images.

There are a variety of ways to generate an LDI. Given a synthetic scene, we could use multiple images from nearby points of view for which depth information is available at each pixel. This information can be gathered from a standard ray tracer that returns depth per pixel or from a scan conversion and z-buffer algorithm where the z-buffer is also returned. Alternatively, we could use a ray tracer to sample an environment in a less regular way and then store computed ray intersections in the LDI structure. Given multiple real images, we can turn to computer vision techniques that can infer pixel correspondence and thus deduce depth values per pixel. We will demonstrate results from each of these three methods.

4.1 LDIs from Multiple Depth Images

We can construct an LDI by warping *n* depth images into a common camera view. For example the depth images C_2 and C_3 in Figure 5 can be warped to the camera frame defined by the LDI (C_1 in figure 5). [3] If, during the warp from the input camera to the LDI camera, two or more pixels map to the same layered depth pixel, their Z values are compared. If the Z values differ by more than a preset epsilon, a new layer is added to that layered depth pixel for each distinct Z value (i.e., *NumLayers* is incremented and a new depth pixel is added), otherwise (e.g., depth pixels *c* and *d* in figure 5), the values are averaged resulting in a single depth pixel. This preprocessing is similar to the rendering described by Max [18]. This construction of the layered depth image is effectively decoupled from the final rendering of images from desired viewpoints. Thus, the LDI construction does not need to run at multiple frames per second to allow interactive camera motion.

4.2 LDIs from a Modified Ray Tracer

By construction, a Layered Depth Image reconstructs images of a scene well from the center of projection of the LDI (we simply display the nearest depth pixels). The quality of the reconstruction from another viewpoint will depend on how closely the distribution of depth pixels in the LDI, when warped to the new viewpoint, corresponds to the pixel density in the new image. Two common events that occur are: (1) disocclusions as the viewpoint changes,

[3]Any arbitrary single coordinate system can be specified here. However, we have found it best to use one of the original camera coordinate systems. This results in fewer pixels needing to be resampled twice; once in the LDI construction, and once in the rendering process.

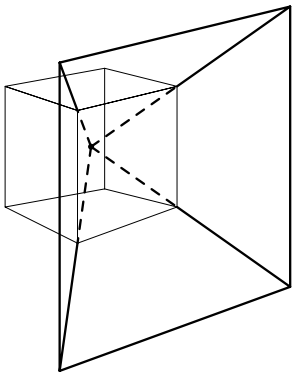


Figure 6 An LDI consists of the 90 degree frustum exiting one side of a cube. The cube represents the region of interest in which the viewer will be able to move.

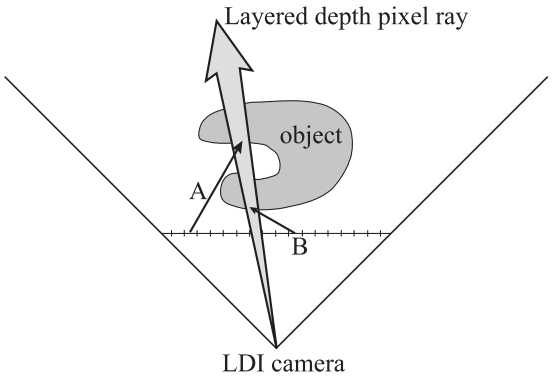


Figure 7 Intersections from sampling rays A and B are added to the same layered depth pixel.

and (2) surfaces that grow in terms of screen space. For example, when a surface is edge on to the LDI, it covers no area. Later, it may face the new viewpoint and thus cover some screen space.

When using a ray tracer, we have the freedom to sample the scene with any distribution of rays we desire. We could simply allow the rays emanating from the center of the LDI to pierce surfaces, recording each hit along the way (up to some maximum). This would solve the disocclusion problem but would not effectively sample surfaces edge on to the LDI.

What set of rays should we trace to sample the scene, to best approximate the distribution of rays from all possible viewpoints we are interested in? For simplicity, we have chosen to use a cubical region of empty space surrounding the LDI center to represent the region that the viewer is able to move in. Each face of the viewing cube defines a 90 degree frustum which we will use to define a single LDI (Figure 6). The six faces of the viewing cube thus cover all of space. For the following discussion we will refer to a single LDI.

Each ray in free space has four coordinates, two for position and two for direction. Since all rays of interest intersect the cube faces, we will choose the outward intersection to parameterize the position of the ray. Direction is parameterized by two angles.

Given no *a priori* knowledge of the geometry in the scene, we assume that every ray intersection the cube is equally important. To achieve a uniform density of rays we sample the positional coordinates uniformly. A uniform distribution over the hemisphere of directions requires that the probability of choosing a direction is proportional to the *projected* area in that direction. Thus, the direction is weighted by the cosine of the angle off the normal to the cube face.

Choosing a cosine weighted direction over a hemisphere can be accomplished by uniformly sampling the unit disk formed by the base of the hemisphere to get two coordinates of the ray direction, say x and y if the z-axis is normal to the disk. The third coordinate is chosen to give a unit length ($z = \sqrt{1 - x^2 - y^2}$). We make the selection within the disk by first selecting a point in the unit square, then applying a measure preserving mapping [23] that maps the unit square to the unit disk.

Given this desired distribution of rays, there are a variety of ways to perform the sampling:

Uniform. A straightforward stochastic method would take as input the number of rays to cast. Then, for each ray it would choose an origin on the cube face and a direction from the cosine distribution and cast the ray into the scene. There are two problems with this simple scheme. First, such *white noise* distributions tend to form unwanted clumps. Second, since there is no coherence between rays, complex scenes require considerable memory thrashing since rays will access the database in a random way [24]. The model of the chestnut tree seen in the color images was too complex to sample with a pure stochastic method on a machine with 320MB of memory.

Stratified Stochastic. To improve the coherence and distribution of rays, we employ a stratified scheme. In this method, we divide the 4D space of rays uniformly into a grid of $N \times N \times N \times N$ strata. For each stratum, we cast M rays. Enough coherence exists within a stratum that swapping of the data set is alleviated. Typical values for N and M are 32 and 16, generating approximately 16 million rays per cube face.

Once a ray is chosen, we cast it into the scene. If it hits an object, and that object lies in the LDI's frustum, we reproject the intersection into the LDI, as depicted in Figure 7, to determine which layered depth pixel should receive the sample. If the new sample is within an epsilon tolerance in depth of an existing depth pixel, the color of the new sample is averaged with the existing depth pixel. Otherwise, the color, normal, and distance to the sample create a new depth pixel that is inserted into the Layered Depth Pixel.

4.3 LDIs from Real Images

The dinosaur model in Figure 13 is constructed from 21 photographs of the object undergoing a 360 degree rotation on a computer-controlled calibrated turntable. An adaptation of Seitz and Dyer's voxel coloring algorithm [29] is used to obtain the LDI representation directly from the input images. The regular voxelization of Seitz and Dyer is replaced by a view-centered voxelization similar to the LDI structure. The procedure entails moving outward on rays from the LDI camera center and projecting candidate voxels back into the input images. If all input images agree on a color, this voxel is filled as a depth pixel in the LDI structure. This approach enables straightforward construction of LDI's from images that do not contain depth per pixel.

5 Rendering Layered Depth Images

Our fast warping-based renderer takes as input an LDI along with its associated camera information. Given a new desired camera position, the warper uses an incremental warping algorithm to efficiently create an output image. Pixels from the LDI are splatted into the output image using the *over* compositing operation. The size and

footprint of the splat is based on an estimated size of the reprojected pixel.

5.1 Space Efficient Representation

When rendering, it is important to maintain the spatial locality of depth pixels to exploit the second level cache in the CPU [12]. To this end, we reorganize the depth pixels into a linear array ordered from bottom to top and left to right in screen space, and back to front along a ray. We also separate out the number of layers in each layered depth pixel from the depth pixels themselves. The layered depth pixel structure does not exist explicitly in this implementation. Instead, a double array of offsets is used to locate each depth pixel. The number of depth pixels in each scanline is accumulated into a vector of offsets to the beginning of each scanline. Within each scanline, for each pixel location, a total count of the depth pixels from the beginning of the scanline to that location is maintained. Thus to find any layered depth pixel, one simply offsets to the beginning of the scanline and then further to the first depth pixel at that location. This supports scanning in right-to-left order as well as the clipping operation discussed later.

5.2 Incremental Warping Computation

The incremental warping computation is similar to the one used for certain texture mapping operations [9, 27]. The geometry of this computation has been analyzed by McMillan [22], and efficient computation for the special case of orthographic input images is given in [3].

Let C_1 be the 4×4 matrix for the LDI camera. It is composed of an affine transformation matrix, a projection matrix, and a viewport matrix, $C_1 = V_1 \cdot P_1 \cdot A_1$. This camera matrix transforms a point from the global coordinate system into the camera's projected image coordinate system. The projected image coordinates (x_1, y_1), obtained after multiplying the point's global coordinates by C_1 and dividing out w_1, index a screen pixel address. The z_1 coordinate can be used for depth comparisons in a z buffer.

Let C_2 be the output camera's matrix. Define the transfer matrix as $T_{1,2} = C_2 \cdot C_1^{-1}$. Given the projected image coordinates of some point seen in the LDI camera (e.g., the coordinates of a in Figure 5), this matrix computes the image coordinates as seen in the output camera (e.g., the image coordinates of a_2 in camera C_2 in Figure 5).

$$T_{1,2} \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_2 \cdot w_2 \\ y_2 \cdot w_2 \\ z_2 \cdot w_2 \\ w_2 \end{bmatrix} = \textbf{result}$$

The coordinates (x_2, y_2) obtained after dividing by w_2, index a pixel address in the output camera's image.

Using the linearity of matrix operations, this matrix multiply can be factored to reuse much of the computation from each iteration through the layers of a layered depth pixel; **result** can be computed as

$$T_{1,2} \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = T_{1,2} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} + z_1 \cdot T_{1,2} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \textbf{start} + z_1 \cdot \textbf{depth}$$

To compute the warped position of the next layered depth pixel along a scanline, the new **start** is simply incremented.

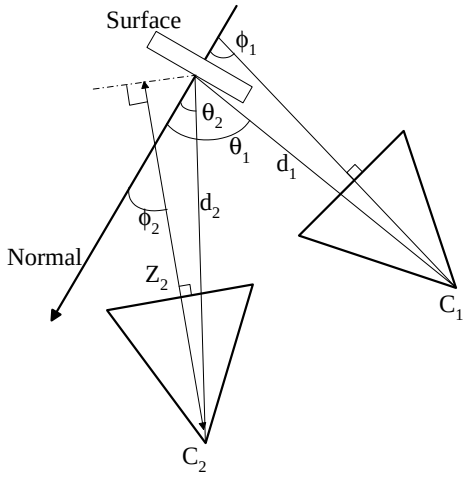


Figure 8 Values for size computation of a projected pixel.

$$T_{1,2} \cdot \begin{bmatrix} x_1 + 1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} = T_{1,2} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} + T_{1,2} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \textbf{start} + \textbf{xincr}$$

The warping algorithm proceeds using McMillan's ordering algorithm [20]. The LDI is broken up into four regions above and below and to the left and right of the epipolar point. For each quadrant, the LDI is traversed in (possibly reverse) scan line order. At the beginning of each scan line, **start** is computed. The sign of **xincr** is determined by the direction of processing in this quadrant. Each layered depth pixel in the scan line is then warped to the output image by calling *Warp*. This procedure visits each of the layers in back to front order and computes **result** to determine its location in the output image. As in perspective texture mapping, a divide is required per pixel. Finally, the depth pixel's color is splatted at this location in the output image.

The following pseudo code summarizes the warping algorithm applied to each layered depth pixel.

```
procedure Warp(ldpix, start, depth, xincr)
    for k ←0 to dpix.NumLayers-1
        z1 ←ldpix.Layers[k].Z
        result ←start + z1 ∗ depth
        //cull if the depth pixel goes behind the output camera
        //or if the depth pixel goes out of the output cam's frustum
        if result.w > 0 and IsInViewport(result) then
            result ←result / result.w
            // see next section
            sqrtSize ←z2 ∗ lookupTable[ldpix.Layers[k].SplatIndex]
            splat(ldpix.Layers[k].ColorRGBA, x2, y2, sqrtSize)
        end if
        // increment for next layered pixel on this scan line
        start ←start + xincr
    end for
end procedure
```

5.3 Splat Size Computation

To splat the LDI into the output image, we estimate the projected area of the warped pixel. This is a rough approximation to the footprint evaluation [33] optimized for speed. The proper size can be computed (differentially) as

$$size = \frac{(d_1)^2 \cos(\theta_2)\, res_2 \tan(fov_1/2)}{(d_2)^2 \cos(\theta_1)\, res_1 \tan(fov_2/2)}$$

where d_1 is the distance from the sampled surface point to the LDI camera, fov_1 is the field of view of the LDI camera, $res_1 = (w_1 h_1)^{-1}$ where w_1 and h_1 are the width and height of the LDI, and θ_1 is the angle between the surface normal and the line of sight to the LDI camera (see Figure 8). The same terms with subscript 2 refer to the output camera.

It will be more efficient to compute an approximation of the square root of size,

$$\begin{aligned}\sqrt{size} &= \frac{1}{d_2} \cdot \frac{d_1 \sqrt{\cos(\theta_2) res_2 \tan(fov_1/2)}}{\sqrt{\cos(\theta_1) res_1 \tan(fov_2/2)}} \\ &\approx \frac{1}{Z_2} \cdot \frac{d_1 \sqrt{\cos(\phi_2) res_2 \tan(fov_1/2)}}{\sqrt{\cos(\phi_1) res_1 \tan(fov_2/2)}} \\ &\approx z_2 \cdot \frac{d_1 \sqrt{\cos(\phi_2) res_2 \tan(fov_1/2)}}{\sqrt{\cos(\phi_1) res_1 \tan(fov_2/2)}}\end{aligned}$$

We approximate the θs as the angles ϕ between the surface normal vector and the z axes of the camera's coordinate systems. We also approximate d_2 by Z_2, the z coordinate of the sampled point in the output camera's unprojected eye coordinate system. During rendering, we set the projection matrix such that $z_2 = 1/Z_2$.

The current implementation supports 4 different splat sizes, so a very crude approximation of the size computation is implemented using a lookup table. For each pixel in the LDI, we store d_1 using 5 bits. We use 6 bits to encode the normal, 3 for n_x, and 3 for n_y. This gives us an eleven-bit lookup table index. Before rendering each new image, we use the new output camera information to precompute values for the 2048 possible lookup table indexes. At each pixel we obtain $\sqrt{size}$ by multiplying the computed z_2 by the value found in the lookup table.

$$\sqrt{size} \approx z_2 \cdot \texttt{lookup[nx, ny, d1]}$$

To maintain the accuracy of the approximation for d_1, we discretize d_1 nonlinearly using a simple exponential function that allocates more bits to the nearby d_1 values, and fewer bits to the distant d_1 values.

The four splat sizes we currently use have 1 by 1, 3 by 3, 5 by 5, and 7 by 7 pixel footprints. Each pixel in a footprint has an alpha value to approximate a Gaussian splat kernel. However, the alpha values are rounded to 1, 1/2, or 1/4, so the alpha blending can be done with integer shifts and adds.

5.4 Depth Pixel Representation

The size of a cache line on current Intel processors (Pentium Pro and Pentium II) is 32 bytes. To fit four depth pixels into a single cache line we convert the floating point Z value to a 20 bit integer. This is then packed into a single word along with the 11 bit splat table index. These 32 bits along with the R, G, B, and alpha values fill out the 8 bytes. This seemingly small optimization yielded a 25 percent improvement in rendering speed.

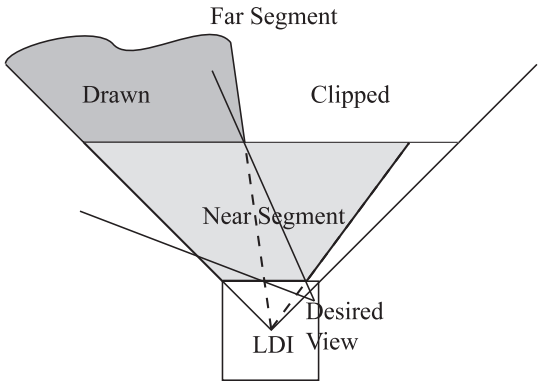


Figure 9 LDI with two segments

5.5 Clipping

The LDI of the chestnut tree scene in Figure 11 is a large data set containing over 1.1 million depth pixels. If we naively render this LDI by reprojecting every depth pixel, we would only be able to render at one or two frames per second. When the viewer is close to the tree, there is no need to flow those pixels that will fall outside of the new view. Unseen pixels can be culled by intersecting the view frustum with the frustum of the LDI. This is implemented by intersecting the view frustum with the near and far plane of the LDI frustum, and taking the bounding box of the intersection. This region defines the rays of depth pixels that could be seen in the new view. This computation is conservative, and gives suboptimal results when the viewer is looking at the LDI from the side (see Figure 9). The view frustum intersects almost the entire cross section of the LDI frustum, but only those depth pixels in the desired view need be warped. Our simple clipping test indicates that most of the LDI needs to be warped. To alleviate this, we split the LDI into two segments, a near and a far segment (see Figure 9). These are simply two frustra stacked one on top of the other. The near frustum is kept smaller than the back segment. We clip each segment individually, and render the back segment first and the front segment second. Clipping can speed rendering times by a factor of 2 to 4.

6 Results

Sprites with Depth and Layered Depth Images have been implemented in C++. The color figures show two examples of rendering sprites and three examples of rendering LDIs. Figures 3a through 3j show the results of rendering a sprite with depth. The hemisphere in the middle of the sprite pops out of the plane of the sprite, and the illusion of depth is quite good. Figure 4 shows the process of extracting sprites from multiple images using the vision techniques discussed in Section 3. There is a great deal of parallax between the layers of sprites, resulting in a convincing and inexpensive image-based-rendering method.

Figure 10 shows two views of a barnyard scene modeled in Softimage. A set of 20 images was pre-rendered from cameras that encircle the chicken using the Mental Ray renderer. The renderer returns colors, depths, and normals at each pixel. The images were rendered at 320 by 320 pixel resolution, taking approximately one minute each to generate. In the interactive system, the 3 images out of the 17 that have the closest direction to the current camera are chosen. The preprocessor (running in a low-priority thread) uses these images to create an LDI in about 1 second. While the LDIs are allocated with a maximum of 10 layers per pixel, the average depth complexity for these LDIs is only 1.24. Thus the use of three input images only increases the rendering cost by 24 percent. The fast

Figure 10 Barnyard scene

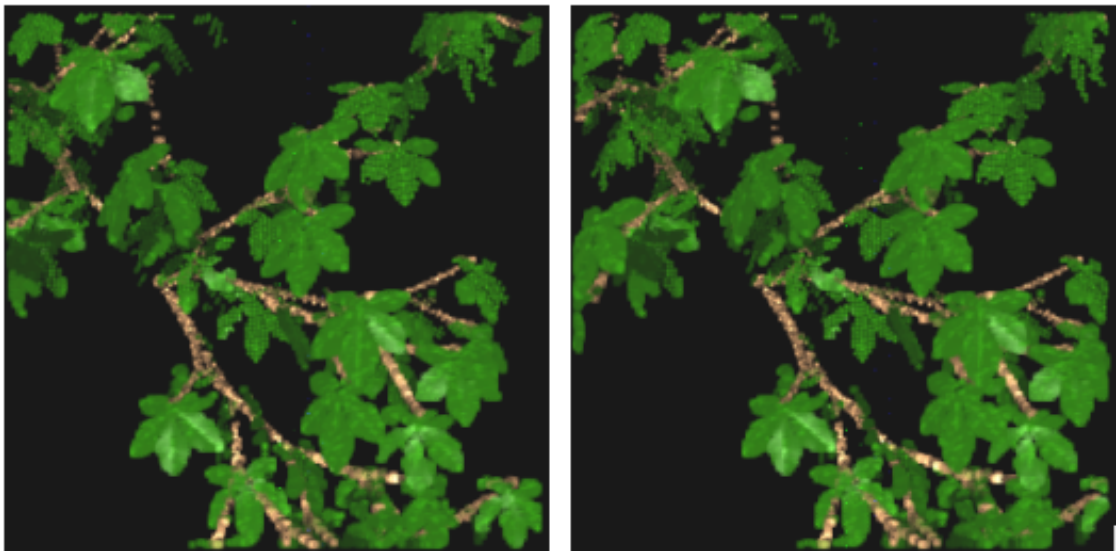


Figure 11 Near segment of chestnut tree

Figure 12 Chestnut tree in front of environment map

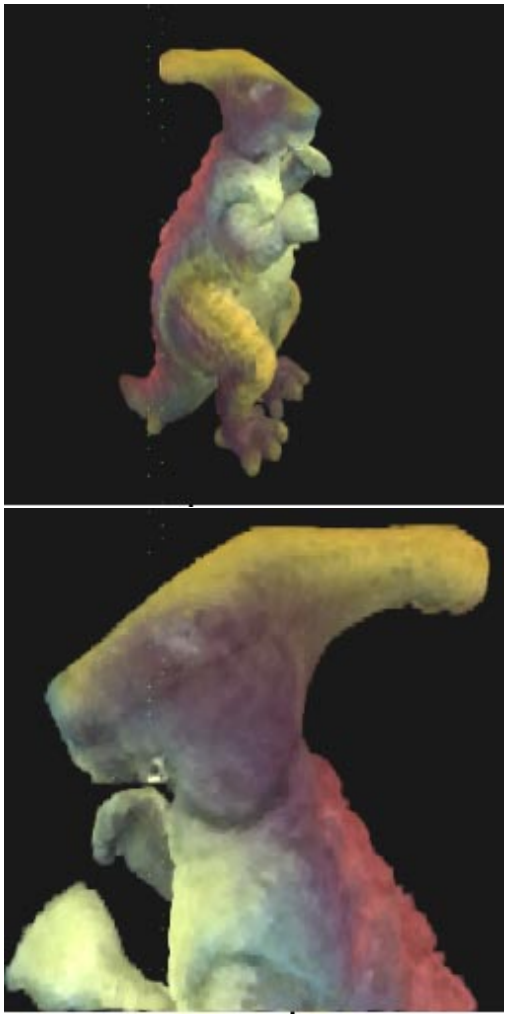


Figure 13 Dinosaur model reconstructed from 21 photographs

renderer (running concurrently in a high-priority thread) generates images at 300 by 300 resolution. On a Pentium II PC running at 300MHz, we achieved frame rate of 8 to 10 frames per second.

Figures 11 and 12 show two cross-eye stereo pairs of a chestnut tree. In Figure 11 only the near segment is displayed. Figure 12 shows both segments in front of an environment map. The LDIs were created using a modified version of the Rayshade raytracer. The tree model is very large; Rayshade allocates over 340 MB of memory to render a single image of the tree. The stochastic method discussed in Section 4.2 took 7 hours to trace 16 million rays through this scene using an SGI Indigo2 with a 250 MHz processor and 320MB of memory. The resulting LDI has over 1.1 million depth pixels, 70,000 of which were placed in the near segment with the rest in the far segment. When rendering this interactively we attain frame rates between 4 and 10 frames per second on a Pentium II PC running at 300MHz.

7 Discussion

In this paper, we have described two novel techniques for image based rendering. The first technique renders Sprites with Depth without visible gaps, and with a smoother rendering than traditional forward mapping (splatting) techniques. It is based on the observation that a forward mapped displacement map does not have to be as accurate as a forward mapped color image. If the displacement map is smooth, the inaccuracies in the warped displacement map result in only sub-pixel errors in the final color pixel sample positions.

Our second novel approach to image based rendering is a Layered Depth Image representation. The LDI representation provides the means to display the parallax induced by camera motion as well as reveal disoccluded regions. The average depth complexity in our LDI's is much lower that one would achieve using multiple input images (e.g., only 1.24 in the Chicken LDI). The LDI representation takes advantage of McMillan's ordering algorithm allowing pixels to be splatted back to Front with an *over* compositing operation.

Traditional graphics elements and planar sprites can be combined with Sprites with Depth and LDIs in the same scene if a back-to-front ordering is maintained. In this case they are simply composited onto one another. Without such an ordering a z-buffer approach will still work at the extra cost of maintaining depth information per frame.

Choosing a single camera view to organize the data has the advantage of having sampled the geometry with a preference for views very near the center of the LDI. This also has its disadvantages. First, pixels undergo two resampling steps in their journey from input image to output. This can potentially degrade image quality. Secondly, if some surface is seen at a glancing angle in the LDIs view the depth complexity for that LDI increases, while the spatial sampling resolution over that surface degrades. The sampling and aliasing issues involved in our layered depth image approach are still not fully understood; a formal analysis of these issues would be helpful.

With the introduction of our two new representations and rendering techniques, there now exists a wide range of different image based rendering methods available. At one end of the spectrum are traditional texture-mapped models. When the scene does not have too much geometric detail, and when texture-mapping hardware is available, this may be the method of choice. If the scene can easily be partitioned into non-overlapping sprites (with depth), then triangle-based texture-mapped rendering can be used without requiring a z buffer [17, 4].

All of these representations, however, do not explicitly account for certain variation of scene appearance with viewpoint, e.g., specularities, transparency, etc. View-dependent texture maps [5], and 4D representations such as lightfields or Lumigraphs [15, 7], have been designed to model such effects. These techniques can lead to greater realism than static texture maps, sprites, or Layered Depth Images, but usually require more effort (and time) to render.

In future work, we hope to explore representations and rendering algorithms which combine several image based rendering techniques. Automatic techniques for taking a 3D scene (either synthesized or real) and re-representing it in the most appropriate fashion for image based rendering would be very useful. These would allow us to apply image based rendering to truly complex, visually rich scenes, and thereby extend their range of applicability.

Acknowledgments

The authors would first of all like to thank Michael F. Cohen. Many of the original ideas contained in this paper as well as much of the discussion in the paper itself can be directly attributable to him. The authors would also like to thank Craig Kolb for his help in obtaining and modifying Rayshade. Steve Seitz is responsible for creating the LDI of the dinosaur from a modified version of his earlier code. Andrew Glassner was a great help with some of the illustrations in the paper. Finally, we would like to thank Microsoft Research for helping to bring together the authors to work on this project.

References

[1] S. Baker, R. Szeliski, and P. Anandan. A Layered Approach to Stereo Reconstruction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*. Santa Barbara, June 1998.

[2] Shenchang Eric Chen and Lance Williams. View Interpolation for Image Synthesis. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288. August 1993.

[3] William Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs. The Delta Tree: An Object Centered Approach to Image Based Rendering. AI technical Memo 1604, MIT, 1996.

[4] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Navigating Static Environments Using Image-Space Simplification and Morphing. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 25–34. 1997.

[5] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996.

[6] O. Faugeras. *Three-dimensional computer vision: A geometric viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.

[7] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The Lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996.

[8] Paul S. Heckbert. Survey of Texture Mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[9] Paul S. Heckbert and Henry P. Moreton. Interpolation for Polygon Texture Mapping and Shading. In David Rogers and Rae Earnshaw, editors, *State of the Art in Computer Graphics: Visualization and Modeling*, pages 101–111. Springer-Verlag, 1991.

[10] Youichi Horry, Ken ichi Anjyo, and Kiyoshi Arai. Tour Into the Picture: Using a Spidery Mesh Interface to Make Animation from a Single Image. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 225–232. ACM SIGGRAPH, Addison Wesley, August 1997.

[11] R. Kumar, P. Anandan, and K. Hanna. Direct recovery of shape from multiple views: A parallax based approach. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, volume A, pages 685–688. IEEE Computer Society Press, Jerusalem, Israel, October 1994.

[12] Anthony G. LaMarca. Caches and Algorithms. Ph.D. thesis, University of Washington, 1996.

[13] S. Laveau and O. D. Faugeras. 3-D Scene Representation as a Collection of Images. In *Twelfth International Conference on Pattern Recognition (ICPR'94)*, volume A, pages 689–691. IEEE Computer Society Press, Jerusalem, Israel, October 1994.

[14] Jed Lengyel and John Snyder. Rendering with Coherent Layers. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 233–242. ACM SIGGRAPH, Addison Wesley, August 1997.

[15] Marc Levoy and Pat Hanrahan. Light Field Rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996.

[16] Mark Levoy and Turner Whitted. The Use of Points as a Display Primitive. Technical Report 85-022, University of North Carolina, 1985.

[17] William R. Mark, Leonard McMilland, and Gary Bishop. Post-Rendering 3D Warping. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 7–16. 1997.

[18] Nelson Max. Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 165–174. Eurographics, Springer Wein, New York City, NY, June 1996.

[19] Leonard McMillan. Computing Visibility Without Depth. Technical Report 95-047, University of North Carolina, 1995.

[20] Leonard McMillan. A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces. Technical Report 95-005, University of North Carolina, 1995.

[21] Leonard McMillan and Gary Bishop. Plenoptic Modeling: An Image-Based Rendering System. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995.

[22] Leonard McMillan and Gary Bishop. Shape as a Pertebation to Projective Mapping. Technical Report 95-046, University of North Carolina, 1995.

[23] Don P. Mitchell. *personal communication*. 1997.

[24] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering Complex Scenes with Memory-Coherent Ray Tracing. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 101–108. ACM SIGGRAPH, Addison Wesley, August 1997.

[25] H. S. Sawhney. 3D Geometry from Planar Parallax. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 929–934. IEEE Computer Society, Seattle, Washington, June 1994.

[26] Gernot Schaufler and Wolfgang Stürzlinger. A Three-Dimensional Image Cache for Virtual Reality. In *Proceedings of Eurographics '96*, pages 227–236. August 1996.

[27] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul E. Haeberli. Fast shadows and lighting effects using texture mapping. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 249–252. July 1992.

[28] Steven M. Seitz and Charles R. Dyer. View Morphing: Synthesizing 3D Metamorphoses Using Image Transforms. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 21–30. ACM SIGGRAPH, Addison Wesley, August 1996.

[29] Steven M. seitz and Charles R. Dyer. Photorealistic Scene Reconstruction by Voxel Coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1067–1073. 1997.

[30] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996.

[31] Jay Torborg and Jim Kajiya. Talisman: Commodity Real-time 3D Graphics for the PC. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 353–364. ACM SIGGRAPH, Addison Wesley, August 1996.

[32] J. Y. A. Wang and E. H. Adelson. Layered Representation for Motion Analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 361–366. New York, New York, June 1993.

[33] Lee Westover. Footprint Evaluation for Volume Rendering. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 367–376. August 1990.

[34] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California, 1990.

Light Field Rendering

Marc Levoy and Pat Hanrahan
Computer Science Department
Stanford University

Abstract

A number of techniques have been proposed for flying through scenes by redisplaying previously rendered or digitized views. Techniques have also been proposed for interpolating between views by warping input images, using depth information or correspondences between multiple images. In this paper, we describe a simple and robust method for generating new views from arbitrary camera positions without depth information or feature matching, simply by combining and resampling the available images. The key to this technique lies in interpreting the input images as 2D slices of a 4D function - the light field. This function completely characterizes the flow of light through unobstructed space in a static scene with fixed illumination.

We describe a sampled representation for light fields that allows for both efficient creation and display of inward and outward looking views. We have created light fields from large arrays of both rendered and digitized images. The latter are acquired using a video camera mounted on a computer-controlled gantry. Once a light field has been created, new views may be constructed in real time by extracting slices in appropriate directions. Since the success of the method depends on having a high sample rate, we describe a compression system that is able to compress the light fields we have generated by more than a factor of 100:1 with very little loss of fidelity. We also address the issues of antialiasing during creation, and resampling during slice extraction.

CR Categories: I.3.2 [Computer Graphics]: Picture/Image Generation — *Digitizing and scanning*, *Viewing algorithms*; I.4.2 [Computer Graphics]: Compression — *Approximate methods*

Additional keywords: image-based rendering, light field, holographic stereogram, vector quantization, epipolar analysis

1. Introduction

Traditionally the input to a 3D graphics system is a scene consisting of geometric primitives composed of different materials and a set of lights. Based on this input specification, the rendering system computes and outputs an image. Recently a new approach to rendering has emerged: *image-based rendering*. Image-based rendering systems generate different views of an environment from a set of pre-acquired imagery. There are several advantages to this approach:

- The display algorithms for image-based rendering require modest computational resources and are thus suitable for real-time implementation on workstations and personal computers.
- The cost of interactively viewing the scene is independent of scene complexity.
- The source of the pre-acquired images can be from a real or virtual environment, i.e. from digitized photographs or from rendered models. In fact, the two can be mixed together.

The forerunner to these techniques is the use of environment maps to capture the incoming light in a texture map [Blinn76, Greene86]. An environment map records the incident light arriving from all directions at a point. The original use of environment maps was to efficiently approximate reflections of the environment on a surface. However, environment maps also may be used to quickly display any outward looking view of the environment from a fixed location but at a variable orientation. This is the basis of the Apple QuickTimeVR system [Chen95]. In this system environment maps are created at key locations in the scene. The user is able to navigate discretely from location to location, and while at each location continuously change the viewing direction.

The major limitation of rendering systems based on environment maps is that the viewpoint is fixed. One way to relax this fixed position constraint is to use view interpolation [Chen93, Greene94, Fuchs94, McMillan95a, McMillan95b, Narayanan95]. Most of these methods require a depth value for each pixel in the environment map, which is easily provided if the environment maps are synthetic images. Given the depth value it is possible to reproject points in the environment map from different vantage points to warp between multiple images. The key challenge in this warping approach is to "fill in the gaps" when previously occluded areas become visible.

Another approach to interpolating between acquired images is to find corresponding points in the two [Laveau94, McMillan95b, Seitz95]. If the positions of the cameras are known, this is equivalent to finding the depth values of the corresponding points. Automatically finding correspondences between pairs of images is the classic problem of stereo vision, and unfortunately although many algorithms exist, these algorithms are fairly fragile and may not always find the correct correspondences.

In this paper we propose a new technique that is robust and allows much more freedom in the range of possible views. The major idea behind the technique is a representation of the *light field*, the radiance as a function of position and direction, in regions of space free of occluders (free space). In free space, the light field is a 4D, not a 5D function. An image is a two dimensional slice of the 4D light field. Creating a light field from a set of images corresponds to inserting each 2D slice into the 4D light field representation. Similarly, generating new views corresponds to extracting and resampling a slice.

Address: Gates Computer Science Building 3B
Stanford University
Stanford, CA 94305

levoy@cs.stanford.edu
hanrahan@cs.stanford.edu
http://www-graphics.stanford.edu

Generating a new image from a light field is quite different than previous view interpolation approaches. First, the new image is generally formed from many different pieces of the original input images, and need not look like any of them. Second, no model information, such as depth values or image correspondences, is needed to extract the image values. Third, image generation involves only resampling, a simple linear process.

This representation of the light field is similar to the epipolar volumes used in computer vision [Bolles87] and to horizontal-parallax-only holographic stereograms [Benton83]. An epipolar volume is formed from an array of images created by translating a camera in equal increments in a single direction. Such a representation has recently been used to perform view interpolation [Katayama95]. A holographic stereogram is formed by exposing a piece of film to an array of images captured by a camera moving sideways. Halle has discussed how to set the camera aperture to properly acquire images for holographic stereograms [Halle94], and that theory is applicable to this work. Gavin Miller has also recognized the potential synergy between true 3D display technologies and computer graphics algorithms [Miller95].

There are several major challenges to using the light field approach to view 3D scenes on a graphics workstation. First, there is the choice of parameterization and representation of the light field. Related to this is the choice of sampling pattern for the field. Second, there is the issue of how to generate or acquire the light field. Third, there is the problem of fast generation of different views. This requires that the slice representing rays through a point be easily extracted, and that the slice be properly resampled to avoid artifacts in the final image. Fourth, the obvious disadvantage of this approach is the large amount of data that may be required. Intuitively one suspects that the light field is coherent and that it may be compressed greatly. In the remaining sections we discuss these issues and our proposed solutions.

2. Representation

We define the light field as the radiance at a point in a given direction. Note that our definition is equivalent to the *plenoptic function* introduced by Adelson and Bergen [Adelson91]. The phrase light field was coined by A. Gershun in his classic paper describing the radiometric properties of light in a space [Gershun36]. [1] McMillan and Bishop [McMillan95b] discuss the representation of 5D light fields as a set of panoramic images at different 3D locations.

However, the 5D representation may be reduced to 4D in free space (regions free of occluders). This is a consequence of the fact that the radiance does not change along a line unless blocked. 4D light fields may be interpreted as functions on the space of oriented lines. The redundancy of the 5D representation is undesirable for two reasons: first, redundancy increases the size of the total dataset, and second, redundancy complicates the reconstruction of the radiance function from its samples. This reduction in dimension has been used to simplify the representation of radiance emitted by luminaires [Levin71, Ashdown93]. For the remainder of this paper we will be only concerned with 4D light fields.

[1] For those familiar with Gershun's paper, he actually uses the term light field to mean the irradiance vector as a function of position. For this reason P. Moon in a later book [Moon81] uses the term photic field to denote what we call the light field.

Although restricting the validity of the representation to free space may seem like a limitation, there are two common situations where this assumption is useful. First, most geometric models are bounded. In this case free space is the region outside the convex hull of the object, and hence all views of an object from outside its convex hull may be generated from a 4D light field. Second, if we are moving through an architectural model or an outdoor scene we are usually moving through a region of free space; therefore, any view from inside this region, of objects outside the region, may be generated.

The major issue in choosing a representation of the 4D light field is how to parameterize the space of oriented lines. There are several issues in choosing the parameterization:

Efficient calculation. The computation of the position of a line from its parameters should be fast. More importantly, for the purposes of calculating new views, it should be easy to compute the line parameters given the viewing transformation and a pixel location.

Control over the set of lines. The space of all lines is infinite, but only a finite subset of line space is ever needed. For example, in the case of viewing an object we need only lines intersecting the convex hull of the object. Thus, there should be an intuitive connection between the actual lines in 3-space and line parameters.

Uniform sampling. Given equally spaced samples in line parameter space, the pattern of lines in 3-space should also be uniform. In this sense, a uniform sampling pattern is one where the *number of lines* in intervals between samples is constant everywhere. Note that the correct measure for number of lines is related to the form factor kernel [Sbert93].

The solution we propose is to parameterize lines by their intersections with two planes in arbitrary position (see figure 1). By convention, the coordinate system on the first plane is (u, v) and on the second plane is (s, t). An oriented line is defined by connecting a point on the uv plane to a point on the st plane. In practice we restrict u, v, s, and t to lie between 0 and 1, and thus points on each plane are restricted to lie within a convex quadrilateral. We call this representation a *light slab*. Intuitively, a light slab represents the beam of light entering one quadrilateral and exiting another quadrilateral.

A nice feature of this representation is that one of the planes may be placed at infinity. This is convenient since then lines may be parameterized by a point and a direction. The latter will prove useful for constructing light fields either from orthographic images or images with a fixed field of view. Furthermore, if all calculations are performed using homogeneous coordinates, the two cases may be handled at no additional cost.

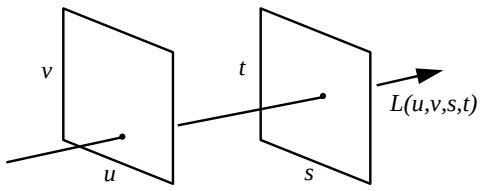


Figure 1: The light slab representation.

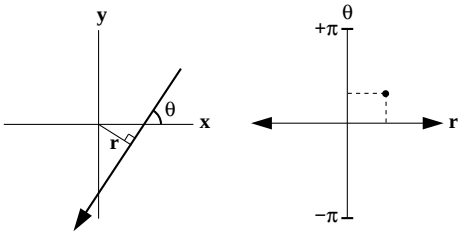


Figure 2: Definition of the line space we use to visualize sets of light rays. Each oriented line in Cartesian space (at left) is represented in line space (at right) by a point. To simplify the visualizations, we show only lines in 2D; the extension to 3D is straightforward.

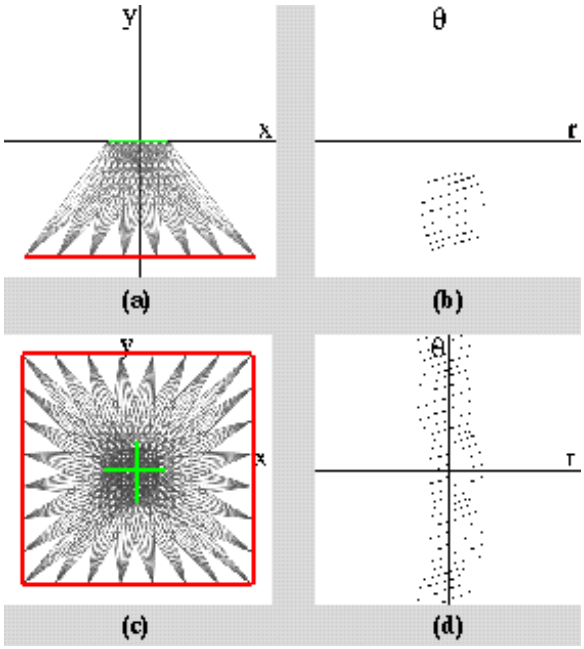


Figure 3: Using line space to visualize ray coverage. (a) shows a single light slab. Light rays (drawn in gray) connect points on two defining lines (drawn in red and green). (c) shows an arrangement of four rotated copies of (a). (b) and (d) show the corresponding line space visualizations. For any set of lines in Cartesian space, the envelope formed by the corresponding points in line space indicates our coverage of position and direction; ideally the coverage should be complete in θ and as wide as possible in r. As these figures show, the single slab in (a) does not provide full coverage in θ, but the four-slab arrangement in (c) does. (c) is, however, narrow in r. Such an arrangement is suitable for inward-looking views of a small object placed at the origin. It was used to generate the lion light field in figure 14d.

A big advantage of this representation is the efficiency of geometric calculations. Mapping from (u, v) to points on the plane is a projective map and involves only linear algebra (multiplying by a 3x3 matrix). More importantly, as will be discussed in section 5, the inverse mapping from an image pixel (x, y) to (u, v, s, t) is also a projective map. Methods using spherical or cylindrical coordinates require substantially more computation.

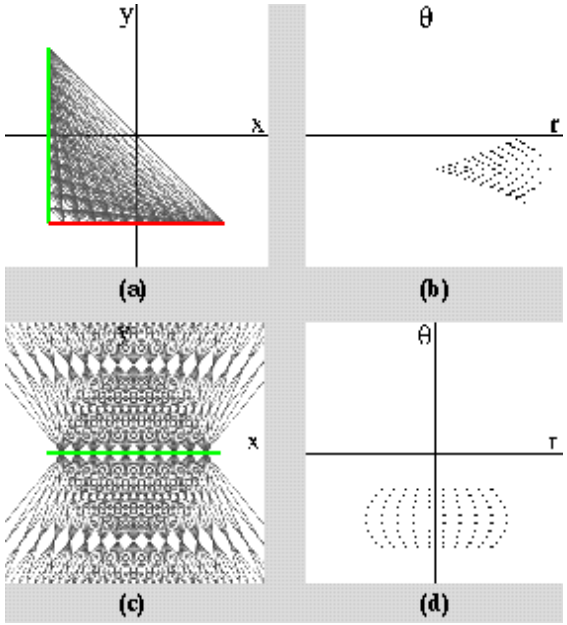


Figure 4: Using line space to visualize sampling uniformity. (a) shows a light slab defined by two lines at right angles. (c) shows a light slab where one defining line is at infinity. This arrangement generates rays passing through the other defining line with an angle between -45° and +45°. (b) and (d) show the corresponding line space visualizations. Our use of (r, θ) to parameterize line space has the property that equal areas in line space correspond to equally dense sampling of position and orientation in Cartesian space; ideally the density of points in line space should be uniform. As these figures show, the singularity at the corner in (a) leads to a highly nonuniform and therefore inefficient sampling pattern, indicated by dark areas in (b) at angles of 0 and $-\pi/2$. (c) generates a more uniform set of lines. Although (c) does not provide full coverage of θ, four rotated copies do. Such an arrangement is suitable for outward-looking views by an observer standing near the origin. It was used to generate the hallway light field in figure 14c.

Many properties of light fields are easier to understand in line space (figures 2 through 4). In line space, each oriented line is represented by a point and each set of lines by a region. In particular, the set of lines represented by a light slab and the set of lines intersecting the convex hull of an object are both regions in line space. All views of an object could be generated from one light slab if its set of lines include all lines intersecting the convex hull of the object. Unfortunately, this is not possible. Therefore, it takes multiple light slabs to represent all possible views of an object. We therefore tile line space with a collection of light slabs, as shown in figure 3.

An important issue related to the parameterization is the sampling pattern. Assuming that all views are equally likely to be generated, then any line is equally likely to be needed. Thus all regions of line space should have an equal density of samples. Figure 4 shows the density of samples in line space for different arrangements of slabs. Note that no slab arrangement is perfect: arrangements with a singularity such as two polygons joined at a corner (4a) are bad and should be avoided, whereas slabs formed

from parallel planes (3a) generate fairly uniform patterns. In addition, arrangements where one plane is at infinity (4c) are better than those with two finite planes (3a). Finally, because of symmetry the spacing of samples in uv should in general be the same as st. However, if the observer is likely to stand near the uv plane, then it may be acceptable to sample uv less frequently than st.

3. Creation of light fields

In this section we discuss the creation of both virtual light fields (from rendered images) and real light fields (from digitized images). One method to create a light field would be to choose a 4D sampling pattern, and for each line sample, find the radiance. This is easily done directly for virtual environments by a ray tracer. This could also be done in a real environment with a spot radiometer, but it would be very tedious. A more practical way to generate light fields is to assemble a collection of images.

3.1. From rendered images

For a virtual environment, a light slab is easily generated simply by rendering a 2D array of images. Each image represents a slice of the 4D light slab at a fixed uv value and is formed by placing the center of projection of the virtual camera at the sample location on the uv plane. The only issue is that the xy samples of each image must correspond exactly with the st samples. This is easily done by performing a sheared perspective projection (figure 5) similar to that used to generate a stereo pair of images. Figure 6 shows the resulting 4D light field, which can be visualized either as a uv array of st images or as an st array of uv images.

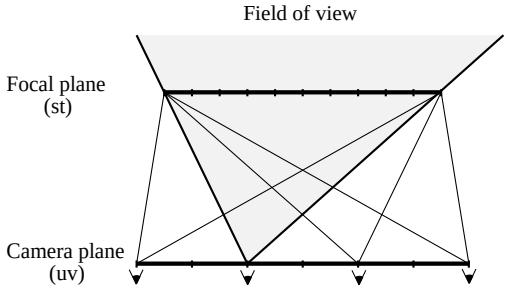


Figure 5: The viewing geometry used to create a light slab from an array of perspective images.

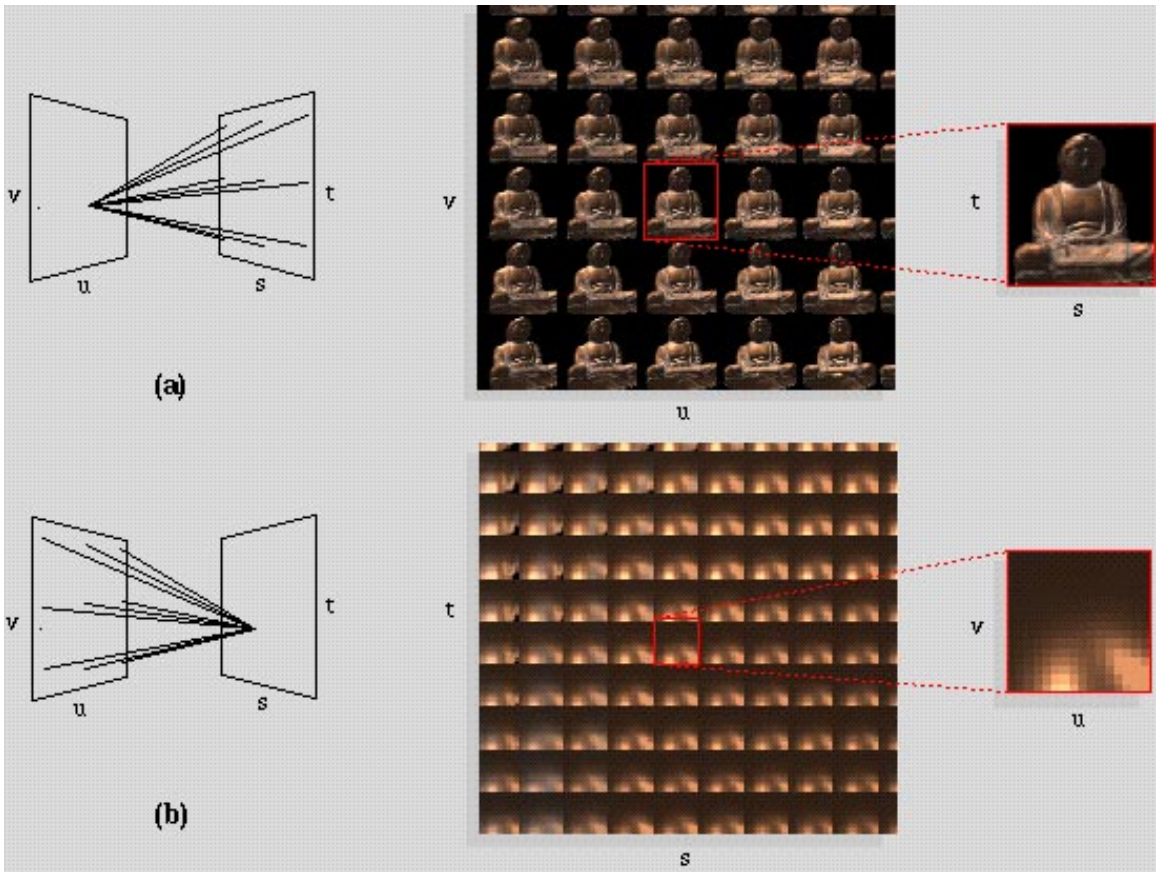


Figure 6: Two visualizations of a light field. (a) Each image in the array represents the rays arriving at one point on the uv plane from all points on the st plane, as shown at left. (b) Each image represents the rays leaving one point on the st plane bound for all points on the uv plane. The images in (a) are off-axis (i.e. sheared) perspective views of the scene, while the images in (b) look like reflectance maps. The latter occurs because the object has been placed astride the focal plane, making sets of rays leaving points on the focal plane similar in character to sets of rays leaving points on the object.

Two other viewing geometries are useful. A light slab may be formed from a 2D array of orthographic views. This can be modeled by placing the uv plane at infinity, as shown in figure 4c. In this case, each uv sample corresponds to the direction of a parallel projection. Again, the only issue is to align the xy and st samples of the image with the st quadrilateral. The other useful geometry consists of a 2D array of outward looking (non-sheared) perspective views with fixed field of view. In this case, each image is a slice of the light slab with the st plane at infinity. The fact that all these cases are equally easy to handle with light slabs attests to the elegance of projective geometry. Light fields using each arrangement are presented in section 6 and illustrated in figure 14.

As with any sampling process, sampling a light field may lead to aliasing since typical light fields contain high frequencies. Fortunately, the effects of aliasing may be alleviated by filtering before sampling. In the case of a light field, a 4D filter in the space of lines must be employed (see figure 7). Assuming a box filter, a weighted average of the radiances on all lines connecting sample squares in the uv and st planes must be computed. If a camera is placed on the uv plane and focussed on the st plane, then the filtering process corresponds to integrating both over a pixel corresponding to an st sample, and an aperture equal in size to a uv sample, as shown in figure 8. The theory behind this filtering process has been discussed in the context of holographic stereograms by Halle [Halle94].

Note that although prefiltering has the desired effect of antialiasing the light field, it has what at first seems like an undesirable side effect — introducing blurriness due to depth of field. However, this blurriness is precisely correct for the situation. Recall what happens when creating a pair of images from two adjacent camera locations on the uv plane: a given object point will project to different locations, potentially several pixels apart, in these two images. The distance between the two projected locations is called the stereo disparity. Extending this idea to multiple camera locations produces a sequence of images in which the object appears to jump by a distance equal to the disparity. This jumping is aliasing. Recall now that taking an image with a finite aperture causes points out of focus to be blurred on the film plane by a circle of confusion. Setting the diameter of the aperture to the spacing between camera locations causes the circle of confusion for each object point to be equal in size to its stereo disparity. This replaces the jumping with a sequence of blurred images. Thus, we are removing aliasing by employing finite depth of field.

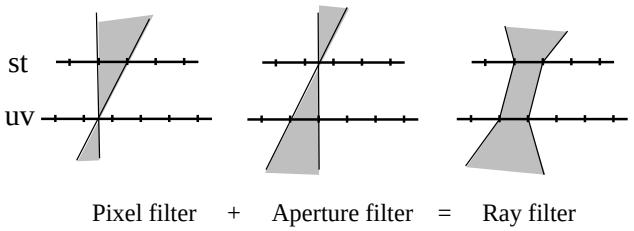


Figure 7: Prefiltering a light field. To avoid aliasing, a 4D low pass filter must be applied to the radiance function.

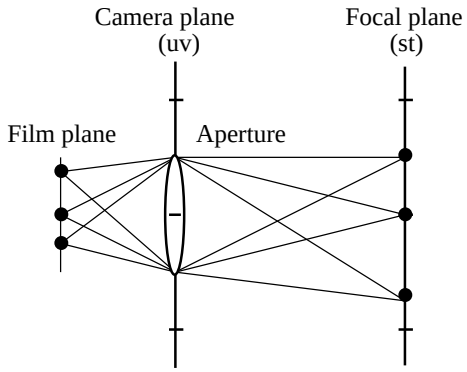


Figure 8: Prefiltering using an aperture. This figure shows a camera focused on the st plane with an aperture on the uv plane whose size is equal to the uv sample spacing. A hypothetical film plane is drawn behind the aperture. Ignore the aperture for a moment (consider a pinhole camera that precisely images the st plane onto the film plane). Then integrating over a pixel on the film plane is equivalent to integrating over an st region bounded by the pixel. Now consider fixing a point on the film plane while using a finite sized aperture (recall that all rays from a point on the film through the aperture are focussed on a single point on the focal plane). Then integrating over the aperture corresponds to integrating all rays through the uv region bounded by the aperture. Therefore, by simultaneously integrating over both the pixel and the aperture, the proper 4D integral is computed.

The necessity for prefiltering can also be understood in line space. Recall from our earlier discussion that samples of the light field correspond to points in line space. Having a finite depth of field with an aperture equal in size to the uv sample spacing insures that each sample adequately covers the interval between these line space points. Too small or too large an aperture yields gaps or overlaps in line space coverage, resulting in views that are either aliased or excessively blurry, respectively.

3.2. From digitized images

Digitizing the imagery required to build a light field of a physical scene is a formidable engineering problem. The number of images required is large (hundreds or thousands), so the process must be automated or at least computer-assisted. Moreover, the lighting must be controlled to insure a static light field, yet flexible enough to properly illuminate the scene, all the while staying clear of the camera to avoid unwanted shadows. Finally, real optical systems impose constraints on angle of view, focal distance, depth of field, and aperture, all of which must be managed. Similar issues have been faced in the construction of devices for performing near-field photometric measurements of luminaires [Ashdown93]. In the following paragraphs, we enumerate the major design decisions we faced in this endeavor and the solutions we adopted.

Inward versus outward looking. The first decision to be made was between a flyaround of a small object and a flythrough of a large-scale scene. We judged flyarounds to be the simpler case, so we attacked them first.

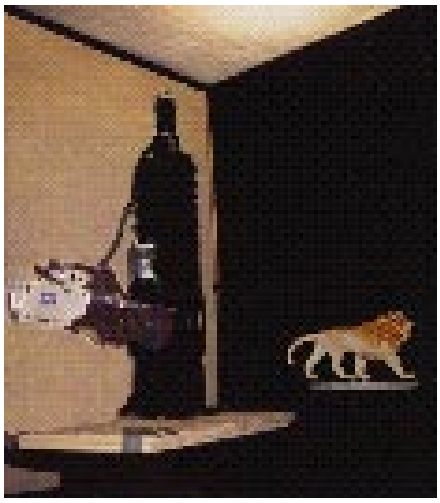


Figure 9: Our prototype camera gantry. A modified Cyberware MS motion platform with additional stepping motors from LinTech and Parker provide four degrees of freedom: horizontal and vertical translation, pan, and tilt. The camera is a Panasonic WV-F300 3-CCD video camera with a Canon f/1.7 10-120mm zoom lens. We keep it locked off at its widest setting (10mm) and mounted so that the pitch and yaw axes pass through the center of projection. While digitizing, the camera is kept pointed at the center of the focal plane. Calibrations and alignments are verified with the aid of a Faro digitizing arm, which is accurate to 0.3 mm.

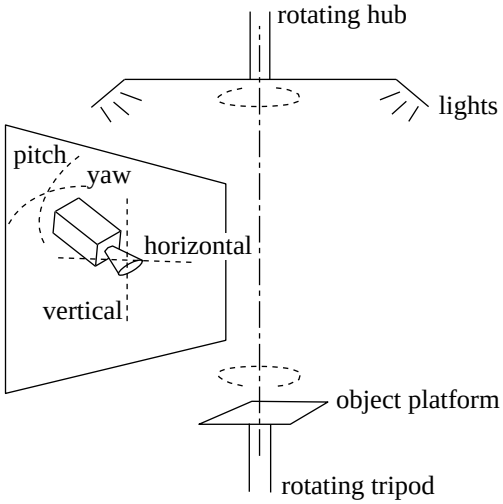


Figure 10: Object and lighting support. Objects are mounted on a Bogen fluid-head tripod, which we manually rotate to four orientations spaced 90 degrees apart. Illumination is provided by two 600W Lowell Omni spotlights attached to a ceiling-mounted rotating hub that is aligned with the rotation axis of the tripod. A stationary 6' x 6' diffuser panel is hung between the spotlights and the gantry, and the entire apparatus is enclosed in black velvet to eliminate stray light.

Human versus computer-controlled. An inexpensive approach to digitizing light fields is to move a handheld camera through the scene, populating the field from the resulting images [Gortler96]. This approach necessitates estimating camera pose at each frame and interpolating the light field from scattered data - two challenging problems. To simplify the situation, we chose instead to build a computer-controlled camera gantry and to digitize images on a regular grid.

Spherical versus planar camera motion. For flyarounds of small objects, an obvious gantry design consists of two concentric hemicycles, similar to a gyroscope mounting. The camera in such a gantry moves along a spherical surface, always pointing at the center of the sphere. Apple Computer has constructed such a gantry to acquire imagery for Quick-Time VR flyarounds [Chen95]. Unfortunately, the lighting in their system is attached to the moving camera, so it is unsuitable for acquiring static light fields. In general, a spherical gantry has three advantages over a planar gantry: (a) it is easier to cover the entire range of viewing directions, (b) the sampling rate in direction space is more uniform, and (c) the distance between the camera and the object is fixed, providing sharper focus throughout the range of camera motion. A planar gantry has two advantages over a spherical gantry: (a) it is easier to build; the entire structure can be assembled from linear motion stages, and (b) it is closer to our light slab representation. For our first prototype gantry, we chose to build a planar gantry, as shown in figure 9.

Field of view. Our goal was to build a light field that allowed 360 degrees of azimuthal viewing. To accomplish this using a planar gantry meant acquiring four slabs each providing 90 degrees. This can be achieved with a camera that translates but does not pan or tilt by employing a wide-angle lens. This solution has two disadvantages: (a) wide-angle lenses exhibit significant distortion, which must be corrected after acquisition, and (b) this solution trades off angle of view against sensor resolution. Another solution is to employ a view camera in which the sensor and optical system translate in parallel planes, the former moving faster than the latter. Horizontal parallax holographic stereograms are constructed using such a camera [Halle94]. Incorporating this solution into a gantry that moves both horizontally and vertically is difficult. We instead chose to equip our camera with pan and tilt motors, enabling us to use a narrow-angle lens. The use of a rotating camera means that, in order to transfer the acquired image to the light slab representation, it must be reprojected to lie on a common plane. This reprojection is equivalent to keystone correction in architectural photography.

Standoff distance. A disadvantage of planar gantries is that the distance from the camera to the object changes as the camera translates across the plane, making it difficult to keep the object in focus. The view camera described above does not suffer from this problem, because the ratio of object distance to image distance stays constant as the camera translates. For a rotating camera, servo-controlled focusing is an option, but changing the focus of a camera shifts its center of projection and changes the image magnification, complicating acquisition. We instead mitigate this problem by using strong lighting and a small aperture to maximize depth of field.

Sensor rotation. Each sample in a light slab should ideally represent the integral over a pixel, and these pixels should lie on a common focal plane. A view camera satisfies this constraint because its sensor translates in a plane. Our use of a rotating camera means that the focal plane also rotates. Assuming that

we resample the images carefully during reprojection, the presence of a rotated focal plane will introduce no additional error into the light field. In practice, we have not seen artifacts due to this resampling process.

Aperture size. Each sample in a light slab should also represent the integral over an aperture equal in size to a uv sample. Our use of a small aperture produces a light field with little or no uv antialiasing. Even fully open, the apertures of commercial video cameras are small. We can approximate the required antialiasing by averaging together some number of adjacent views, thereby creating a *synthetic aperture*. However, this technique requires a very dense spacing of views, which in turn requires rapid acquisition. We do not currently do this.

Object support. In order to acquire a 360-degree light field in four 90-degree segments using a planar gantry, either the gantry or the object must be rotated to each of four orientations spaced 90 degrees apart. Given the massiveness of our gantry, the latter was clearly easier. For these experiments, we mounted our objects on a tripod, which we manually rotate to the four positions as shown in figure 10.

Lighting. Given our decision to rotate the object, satisfying the requirement for fixed illumination means that either the lighting must exhibit fourfold symmetry or it must rotate with the object. We chose the latter solution, attaching a lighting system to a rotating hub as shown in figure 10. Designing a lighting system that stays clear of the gantry, yet provides enough light to evenly illuminate an object, is a challenging problem.

Using this gantry, our procedure for acquiring a light field is as follows. For each of the four orientations, the camera is translated through a regular grid of camera positions. At each position, the camera is panned and tilted to point at the center of the object, which lies along the axis of rotation of the tripod. We then acquire an image, and, using standard texture mapping algorithms, reproject it to lie on a common plane as described earlier. Table II gives a typical set of acquisition parameters. Note that the distance between camera positions (3.125 cm) exceeds the diameter of the aperture (1.25 mm), underscoring the need for denser spacing and a synthetic aperture.

4. Compression

Light field arrays are large — the largest example in this paper is 1.6 GB. To make creation, transmission, and display of light fields practical, they must be compressed. In choosing from among many available compression techniques, we were guided by several unique characteristics of light fields:

Data redundancy. A good compression technique removes redundancy from a signal without affecting its content. Light fields exhibit redundancy in all four dimensions. For example, the smooth regions in figure 6a tell us that this light field contains redundancy in s and t, and the smooth regions in figure 6b tell us that the light field contains redundancy in u and v. The former corresponds to our usual notion of interpixel coherence in a perspective view. The latter can be interpreted either as the interframe coherence one expects in a motion sequence or as the smoothness one expects in the bidirectional reflectance distribution function (BRDF) for a diffuse or moderately specular surface. Occlusions introduce discontinuities in both cases, of course.

Random access. Most compression techniques place some constraint on random access to data. For example, variable-bitrate coders may require scanlines, tiles, or frames to be decoded at once. Examples in this class are variable-bitrate vector quantization and the Huffman or arithmetic coders used in JPEG or MPEG. Predictive coding schemes further complicate random-access because pixels depend on previously decoded pixels, scanlines, or frames. This poses a problem for light fields since the set of samples referenced when extracting an image from a light field are dispersed in memory. As the observer moves, the access patterns change in complex ways. We therefore seek a compression technique that supports low-cost random access to individual samples.

Asymmetry. Applications of compression can be classified as symmetric or asymmetric depending on the relative time spent encoding versus decoding. We assume that light fields are assembled and compressed ahead of time, making this an asymmetric application.

Computational expense. We seek a compression scheme that can be decoded without hardware assistance. Although software decoders have been demonstrated for standards like JPEG and MPEG, these implementations consume the full power of a modern microprocessor. In addition to decompression, the display algorithm has additional work to perform, as will be described in section 5. We therefore seek a compression scheme that can be decoded quickly.

The compression scheme we chose was a two-stage pipeline consisting of fixed-rate vector quantization followed by entropy coding (Lempel-Ziv), as shown in figure 11. Following similar motivations, Beers et al. use vector quantization to compress textures for use in rendering pipelines [Beers96].

4.1. Vector quantization

The first stage of our compression pipeline is vector quantization (VQ) [Gersho92], a lossy compression technique wherein a vector of samples is quantized to one of a number of predetermined reproduction vectors. A reproduction vector is called a codeword, and the set of codewords available to encode a source is called the codebook, Codebooks are constructed during a training phase in which the quantizer is asked to find a set of codewords that best approximates a set of sample vectors, called the training set. The quality of a codeword is typically characterized

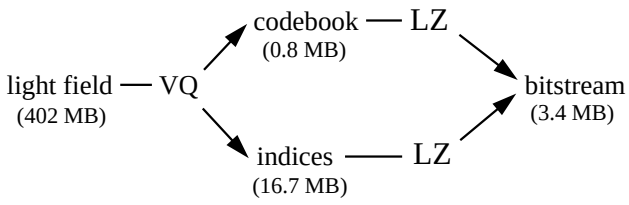


Figure 11 Two-stage compression pipeline. The light field is partitioned into tiles, which are encoded using vector quantization to form an array of codebook indices. The codebook and the array of indices are further compressed using Lempel-Ziv coding. Decompression also occurs in two stages: entropy decoding as the file is loaded into memory, and dequantization on demand during interactive viewing. Typical file sizes are shown beside each stage.

using mean-squared error (MSE), i.e. the sum over all samples in the vector of the squared difference between the source sample and the codeword sample. Once a codebook has been constructed, encoding consists of partitioning the source into vectors and finding for each vector the closest approximating codeword from the codebook. Decoding consists of looking up indices in the codebook and outputting the codewords found there — a very fast operation. Indeed, decoding speed is one of the primary advantages of vector quantization.

In our application, we typically use 2D or 4D tiles of the light field, yielding 12-dimensional or 48-dimensional vectors, respectively. The former takes advantage of coherence in s and t only, while the latter takes advantage of coherence in all four dimensions. To maximize image quality, we train on a representative subset of each light field to be compressed, then transmit the resulting codebook along with the codeword index array. Since light fields are large, even after compression, the additional overhead of transmitting a codebook is small, typically less than 20%. We train on a subset rather than the entire light field to reduce the expense of training.

The output of vector quantization is a sequence of fixed-rate codebook indices. Each index is log N bits where N is the number of codewords in the codebook, so the compression rate of the quantizer is (kl) / $(log\ N)$ where k is the number of elements per vector (i.e. the dimension), and l is the number of bits per element, usually 8. In our application, we typically use 16384-word codebooks, leading to a compression rate for this stage of the pipeline of (48 x 8) / (log 16384) = 384 bits / 14 bits = 27:1. To simplify decoding, we represent each index using an integral number of bytes, 2 in our case, which reduces our compression slightly, to 24:1.

4.2. Entropy coding

The second stage of our compression pipeline is an entropy coder designed to decrease the cost of representing high-probability code indices. Since our objects are typically rendered or photographed against a constant-color background, the array contains many tiles that occur with high probability. For the examples in this paper, we employed gzip, an implementation of Lempel-Ziv coding [Ziv77]. In this algorithm, the input stream is partitioned into nonoverlapping blocks while constructing a dictionary of blocks seen thus far. Applying gzip to our array of code indices typically gives us an additional 5:1 compression. Huffman coding would probably yield slightly higher compression, but encoding and decoding would be more expensive. Our total compression is therefore 24 x 5 = 120:1. See section 6 and table III for more detail on our compression results.

4.3. Decompression

Decompression occurs in two stages. The first stage — gzip decoding — is performed as the file is loaded into memory. The output of this stage is a codebook and an array of code indices packed in 16-bit words. Although some efficiency has been lost by this decoding, the light field is still compressed 24:1, and it is now represented in a way that supports random access.

The second stage — dequantization — proceeds as follows. As the observer moves through the scene, the display engine requests samples of the light field. Each request consists of a (u, v, s, t) coordinate tuple. For each request, a subscripting calculation is performed to determine which sample tile is being addressed. Each tile corresponds to one quantization vector and is thus represented in the index array by a single entry. Looking this index up in the codebook, we find a vector of sample values. A second subscripting calculation is then performed, giving us the offset of the requested sample within the vector. With the aid of precomputed subscripting tables, dequantization can be implemented very efficiently. In our tests, decompression consumes about 25% of the CPU cycles.

5. Display

The final part of the system is a real time viewer that constructs and displays an image from the light slab given the imaging geometry. The viewer must resample a 2D slice of lines from the 4D light field; each line represents a ray through the eye point and a pixel center as shown in figure 12. There are two steps to this process: step 1 consists of computing the (u, v, s, t) line parameters for each image ray, and step 2 consists of resampling the radiance at those line parameters.

As mentioned previously, a big advantage of the light slab representation is the efficiency of the inverse calculation of the line parameters. Conceptually the (u, v) and (s, t) parameters may be calculated by determining the point of intersection of an image ray with each plane. Thus, any ray tracer could easily be adapted to use light slabs. However, a polygonal rendering system also may be used to view a light slab. The transformation from image coordinates (x, y) to both the (u, v) and the (s, t) coordinates is a projective map. Therefore, computing the line coordinates can be done using texture mapping. The uv quadrilateral is drawn using the current viewing transformation, and during scan conversion the (uw, vw, w) coordinates at the corners of the quadrilateral are interpolated. The resulting $u = uw/w$ and $v = vw/w$ coordinates at each pixel represent the ray intersection with the uv quadrilateral. A similar procedure can be used to generate the (s, t) coordinates by drawing the st quadrilateral. Thus, the inverse transformation from (x, y) to (u, v, s, t) reduces essentially to two texture coordinate calculations per ray. This is cheap and can be done in real time, and is supported in many rendering systems, both hardware and software.

Only lines with (u, v) and (s, t) coordinates inside both quadrilaterals are represented in the light slab. Thus, if the texture coordinates for each plane are computed by drawing each quadrilaterial one after the other, then only those pixels that have both valid uv and st coordinates should be looked up in the light slab array. Alternatively, the two quadrilaterals may be simultaneously scan converted in their region of overlap to cut down on unnecessary calculations; this is the technique that we use in our software implementation.

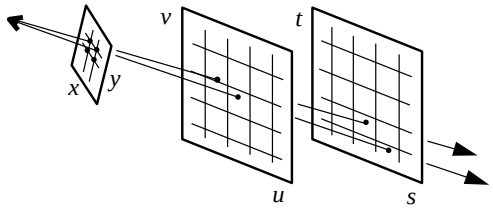


Figure 12: The process of resampling a light slab during display.

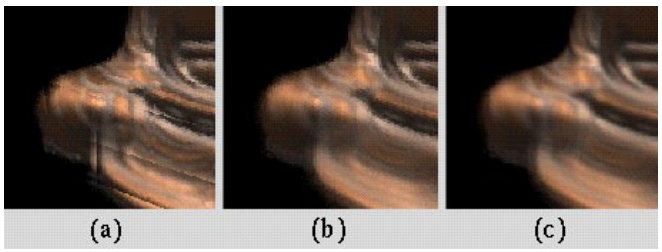


Figure 13: The effects of interpolation during slice extraction. (a) No interpolation. (b) Linear interpolation in uv only. (c) Quadralinear interpolation in uvst.

To draw an image of a collection of light slabs, we draw them sequentially. If the sets of lines in the collection of light slabs do not overlap, then each pixel is drawn only once and so this is quite efficient. To further increase efficiency, "back-facing" light slabs may be culled.

The second step involves resampling the radiance. The ideal resampling process first reconstructs the function from the original samples, and then applies a bandpass filter to the reconstructed function to remove high frequencies that may cause aliasing. In our system, we approximate the resampling process by simply interpolating the 4D function from the nearest samples. This is correct only if the new sampling rate is greater than the original sampling rate, which is usually the case when displaying light fields. However, if the image of the light field is very small, then some form of prefiltering should be applied. This could easily be done with a 4D variation of the standard mipmapping algorithm [Williams83].

Figure 13 shows the effect of nearest neighbor versus bilinear interpolation on the uv plane versus quadrilinear interpolation of the full 4D function. Quadralinear interpolation coupled with the proper prefiltering generates images with few aliasing artifacts. The improvement is particularly dramatic when the object or camera is moving. However, quadralinear filtering is more expensive and can sometimes be avoided. For example, if the sampling rates in the uv and st planes are different, and then the benefits of filtering one plane may be greater than the other plane.

6. Results

Figure 14 shows images extracted from four light fields. The first is a buddha constructed from rendered images. The model is an irregular polygon mesh constructed from range data. The input images were generated using RenderMan, which also provided the machinery for computing pixel and aperture antialiasing. The light field configuration was a single slab similar to that shown in figure 3a.

Our second light field is a human abdomen constructed from volume renderings. The two tan-colored organs on either side of the spine are the kidneys. In this case, the input images were orthographic views, so we employed a slab with one plane at infinity as shown in figure 4c. Because an orthographic image contains rays of constant direction, we generated more input images than in the first example in order to provide the angular range needed for creating perspective views. The images include pixel antialiasing but no aperture antialiasing. However, the dense spacing of input images reduces aperture aliasing artifacts to a minimum.

Our third example is an outward-looking light field depicting a hallway in Berkeley's Soda Hall, rendered using a radiosity program. To allow a full range of observer motion while optimizing sampling uniformity, we used four slabs with one plane at infinity, a four-slab version of figure 4c. The input images were rendered on an SGI RealityEngine, using the accumulation buffer to provide both pixel and aperture antialiasing.

Our last example is a light field constructed from digitized images. The scene is of a toy lion, and the light field consists of four slabs as shown in figure 3c, allowing the observer to walk completely around the object. The sensor and optical system provide pixel antialiasing, but the aperture diameter was too small to provide correct aperture antialiasing. As a result, the light field exhibits some aliasing, which appears as double images. These artifacts are worst near the head and tail of the lion because of their greater distance from the axis around which the camera rotated.

Table I summarizes the statistics of each light field. Table II gives additional information on the lion dataset. Table III gives the performance of our compression pipeline on two representative datasets. The buddha was compressed using a 2D tiling of the

	buddha	**kidney**	**hallway**	**lion**
Number of slabs	1	1	4	4
Images per slab	16x16	64x64	64x32	32x16
Total images	256	4096	8192	2048
Pixels per image	256^2	128^2	256^2	256^2
Raw size (MB)	50	201	1608	402
Prefiltering	uvst	st only	uvst	st only

Table I: Statistics of the light fields shown in figure 14.

Camera motion	
translation per slab	100 cm x 50 cm
pan and tilt per slab	90° x 45°
number of slabs	4 slabs 90° apart
total pan and tilt	360° x 45°
Sampling density	
distance to object	50 cm
camera pan per sample	3.6°
camera translation per sample	3.125 cm
Aperture	
focal distance of lens	10mm
F-number	f/8
aperture diameter	1.25 mm
Acquisition time	
time per image	3 seconds
total acquisition time	4 hours

Table II: Acquisition parameters for the lion light field. Distance to object and camera pan per sample are given at the center of the plane of camera motion. Total acquisition time includes longer gantry movements at the end of each row and manual setup time for each of the four orientations. The aperture diameter is the focal length divided by the F-number.

	buddha	lion
Vector quantization		
raw size (MB)	50.3	402.7
fraction in training set	5%	3%
samples per tile	2x2x1x1	2x2x2x2
bytes per sample	3	3
vector dimension	12	48
number of codewords	8192	16384
codebook size (MB)	0.1	0.8
bytes per codeword index	2	2
index array size (MB)	8.4	16.8
total size (MB)	8.5	17.6
compression rate	6:1	23:1
Entropy coding		
gzipped codebook (MB)	0.1	0.6
gzipped index array (MB)	1.0	2.8
total size (MB)	1.1	3.4
compression due to gzip	8:1	5:1
total compression	45:1	118:1
Compression performance		
training time	15 mins	4 hrs
encoding time	1 mins	8 mins
original entropy (bits/pixel)	4.2	2.9
image quality (PSNR)	36	27

Table III: Compression statistics for two light fields. The buddha was compressed using 2D tiles of RGB pixels, forming 12-dimensional vectors, and the lion was compressed using 4D tiles (2D tiles of RGB pixels from each of 2 x 2 adjacent camera positions), forming 48-dimensional vectors. Bytes per codeword index include padding as described in section 4. Peak signal-to-noise ratio (PSNR) is computed as $10 \log_{10}(255^2/MSE)$.

light field, yielding a total compression rate of 45:1. The lion was compressed using a 4D tiling, yielding a higher compression rate of 118:1. During interactive viewing, the compressed buddha is indistinguishable from the original; the compressed lion exhibits some artifacts, but only at high magnifications. Representative images are shown in figure 15. We have also experimented with higher rates. As a general rule, the artifacts become objectionable only above 200:1.

Finally, table IV summarizes the performance of our interactive viewer operating on the lion light field. As the table shows, we achieve interactive playback rates for reasonable image sizes. Note that the size of the light field has no effect on playback rate; only the image size matters. Memory size is not an issue because the compressed fields are small.

Display times (ms)	no bilerp	uv lerp	uvst lerp
coordinate calculation	13	13	13
sample extraction	14	59	214
overhead	3	3	3
total	30	75	230

Table IV: Display performance for the lion light field. Displayed images are 192 x 192 pixels. Sample extraction includes VQ decoding and sample interpolation. Display overhead includes reading the mouse, computing the observer position, and copying the image to the frame buffer. Timings are for a software-only implementation on a 250 MHz MIPS 4400 processor.

7. Discussion and future work

We have described a new light field representation, the light slab, for storing all the radiance values in free space. Both inserting images into the field and extracting new views from the field involve resampling, a simple and robust procedure. The resulting system is easily implemented on workstations and personal computers, requiring modest amounts of memory and cycles. Thus, this technique is useful for many applications requiring interaction with 3D scenes.

There are three major limitation of our method. First, the sampling density must be high to avoid excessive blurriness. This requires rendering or acquiring a large number of images, which may take a long time and consume a lot of memory. However, denser sample spacing leads to greater inter-sample coherence, so the size of the light field is usually manageable after compression. Second, the observer is restricted to regions of space free of occluders. This limitation can be addressed by stitching together multiple light fields based on a partition of the scene geometry into convex regions. If we augment light fields to include Z-depth, the regions need not even be convex. Third, the illumination must be fixed. If we ignore interreflections, this limitation can be addressed by augmenting light fields to include surface normals and optical properties. To handle interreflections, we might try representing illumination as a superposition of basis functions [Nimeroff94]. This would correspond in our case to computing a sum of light fields each lit with a different illumination function.

It is useful to compare this approach with depth-based or correspondence-based view interpolation. In these systems, a 3D model is created to improve quality of the interpolation and hence decrease the number of pre-acquired images. In our approach, a much larger number of images is acquired, and at first this seems like a disadvantage. However, because of the 3D structure of the light field, simple compression schemes are able to find and exploit this same 3D structure. In our case, simple 4D block coding leads to compression rates of over 100:1. Given the success of the compression, a high density compressed light field has an advantage over other approaches because the resampling process is simpler, and no explicit 3D structure must be found or stored.

There are many representations for light used in computer graphics and computer vision, for example, images, shadow and environment maps, light sources, radiosity and radiance basis functions, and ray tracing procedures. However, abstract light representations have not been systematically studied in the same way as modeling and display primitives. A fruitful line of future research would be to reexamine these representations from first principles. Such reexaminations may in turn lead to new methods for the central problems in these fields.

Another area of future research is the design of instrumentation for acquisition. A large parallel array of cameras connected to a parallel computer could be built to acquire and compress a light field in real time. In the short term, there are many interesting engineering issues in designing and building gantries to move

a small number of cameras and lights to sequentially acquire both inward- and outward-looking light fields. This same instrumentation could lead to breakthroughs in both 3D shape acquisition and reflection measurements. In fact, the interaction of light with any object can be represented as a higher-dimensional interaction matrix; acquiring, compressing, and manipulating such representations are a fruitful area for investigation.

8. Acknowledgements

We would like to thank David Addleman and George Dabrowski of Cyberware for helping us design and build the camera gantry, Craig Kolb and James Davis for helping us calibrate it, Brian Curless for scanning the buddha, Julie Dorsey for shading it and allowing us to use it, Carlo Sequin for the Soda Hall model, Seth Teller, Celeste Fowler, and Thomas Funkhauser for its radiosity solution, Lucas Pereira for rendering it, Benjamin Zhu for reimplementing our hardware-accelerated viewer in software, and Navin Chaddha for his vector quantization code. We also wish to thank Eric Chen and Michael Chen for allowing us to examine the Apple ObjectMaker, and Alain Fournier and Bob Lewis for showing us their wavelet light field work. Finally, we wish to thank Nina Amenta for sparking our interest in two-plane parameterizations of lines, Michael Cohen for reinforcing our interest in image-based representations, and Gavin Miller for inspiring us with his grail of volumetric hyperreality. This work was supported by the NSF under contracts CCR-9157767 and CCR-9508579.

9. References

[Adelson91] Adelson, E.H., Bergen, J.R., ‘‘The Plenoptic Function and the Elements of Early Vision,’’ In *Computation Models of Visual Processing*, M. Landy and J.A. Movshon, eds., MIT Press, Cambridge, 1991.

[Ashdown93] Ashdown, I., ‘‘Near-Field Photometry: A New Approach,’’ *Journal of the Illuminating Engineering Society*, Vol. 22, No. 1, Winter, 1993, pp. 163-180.

[Beers96] Beers, A., Agrawala, M., Chaddha, N., ‘‘Rendering from Compressed Textures.’’ In these proceedings.

[Benton83] Benton, S., ‘‘Survey of Holographic Stereograms,’’ *Processing and Display of Three-Dimensional Data*, Proc. SPIE, Vol. 367, 1983.

[Blinn76] Blinn, J.F., Newell, M.E., ‘‘Texture and Reflection in Computer Generated Images,’’ *CACM*, Vol. 19, No. 10, October, 1976, pp. 542-547.

[Bolles87] Bolles, R., Baker, H., Marimont, D., ‘‘Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion,’’ *International Journal of Computer Vision*, Vol. 1, No. 1, 1987, pp. 7-55.

[Chen93] Chen, S.E., Williams, L., ‘‘View Interpolation for Image Synthesis,’’ Proc. SIGGRAPH ’93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics Proceedings*, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 279-288.

[Chen95] Chen, S.E., ‘‘QuickTime VR — An Image-Based Approach to Virtual Environment Navigation,’’ Proc. SIGGRAPH ’95 (Los Angeles, CA, August 6-11, 1995). In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp. 29-38.

[Fuchs94] Fuchs, H., Bishop, G., Arthur, K., McMillan, L., Bajcsy, R., Lee, S.W., Farid, H., Kanade, T., ‘‘Virtual Space Teleconferencing Using a Sea of Cameras,’’ *Proc. First International Conference on Medical Robotics and Computer Assisted Surgery*, 1994, pp. 161-167.

[Gersho92] Gersho, A., Gray, R.M., *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[Gershun36] Gershun, A., ‘‘The Light Field,’’ Moscow, 1936. Translated by P. Moon and G. Timoshenko in *Journal of Mathematics and Physics*, Vol. XVIII, MIT, 1939, pp. 51-151.

[Gortler96] Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M., ‘‘The Lumigraph.’’ In these proceedings.

[Greene86] Greene, N., ‘‘Environment Mapping and Other Applications of World Projections,’’ *IEEE Computer Graphics and Applications*, Vol. 6, No. 11, November, 1986, pp. 21-29.

[Greene94] Greene, N. and Kass, M., ‘‘Approximating Visibility with Environment Maps,’’ Apple Technical Report No. 41, November, 1994.

[Halle94] Halle, M., ‘‘Holographic Stereograms as Discrete Imaging Systems.’’ *Practical Holography*, Proc. SPIE, Vol. 2176, February, 1994.

[Katayama95] Katayama, A., Tanaka, K., Oshino, T., Tamura, H., ‘‘Viewpoint-Dependent Stereoscopic Display Using Interpolation of Multiviewpoint Images,’’ *Stereoscopic Displays and Virtual Reality Systems II*, Proc. SPIE, Vol. 2409, S. Fisher, J. Merritt, B. Bolas eds. 1995, pp. 11-20.

[Laveau94] Laveau, S., Faugeras, O.D., ‘‘3-D Scene Representation as a Collection of Images and Fundamental Matrices,’’ INRIA Technical Report No. 2205, 1994.

[Levin71] Levin, R., ‘‘Photometric Characteristics of Light Controlling Apparatus,’’ *Illuminating Engineering*, Vol. 66, No. 4, 1971, pp. 205-215.

[McMillan95a] McMillan, L., Bishop, G., ‘‘Head-Tracked Stereoscopic Display Using Image Warping,’’ *Stereoscopic Displays and Virtual Reality Systems II*, Proc. SPIE, Vol. 2409, S. Fisher, J. Merritt, B. Bolas eds. 1995, pp. 21-30.

[McMillan95b] McMillan, L., Bishop, G., Plenoptic Modeling: An Image-Based Rendering System, Proc. SIGGRAPH ’95 (Los Angeles, CA, August 6-11, 1995). In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp. 39-46.

[Miller95] Miller, G., ‘‘Volumetric Hyper-Reality: A Computer Graphics Holy Grail for the 21st Century?,’’ *Proc. Graphics Interface ’95*, W. Davis and P. Prusinkiewicz eds., Canadian Information Processing Society, 1995, pp. 56-64.

[Moon81] Moon, P., Spencer, D.E., *The Photic Field*, MIT Press, 1981.

[Narayanan95] Narayanan, P.J., ‘‘Virtualized Reality: Concepts and Early Results,’’ *Proc. IEEE Workshop on the Representation of Visual Scenes*, IEEE, 1995.

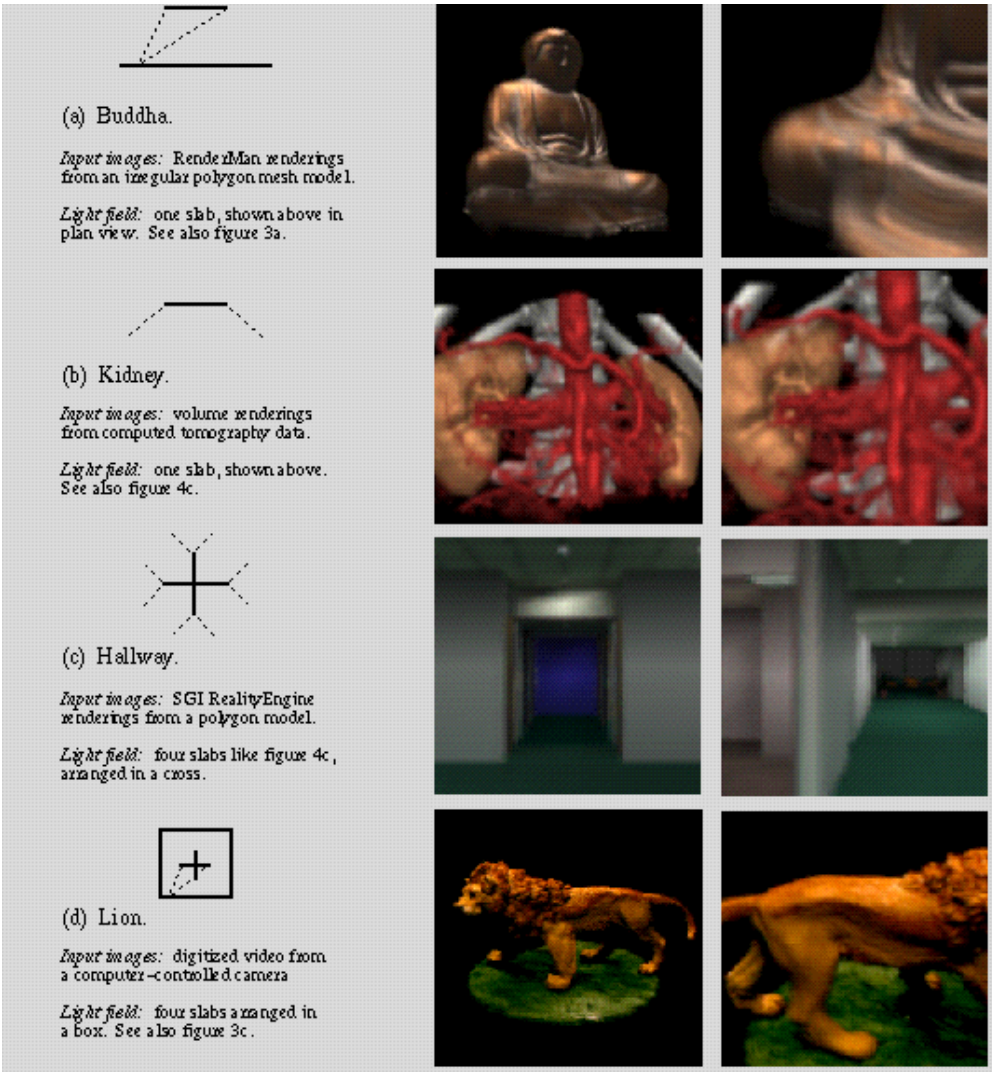
[Nimeroff94] Nimeroff, J., Simoncelli, E., Dorsey, J., ‘‘Efficient Rerendering of Naturally Illuminated Scenes,’’ *Proc. Fifth Eurographics Rendering Workshop*, 1994, pp. 359-373.

[Sbert93] Sbert, A.M., ‘‘An Integral Geometry Based Method for Form-Factor Computation,’’ *Computer Graphics Forum*, Vol. 13, No. 3, 1993, pp. 409-420.

[Seitz95] Seitz, S., Dyer, C., ‘‘Physically-Valid View Synthesis by Image Interpolation,’’ *Proc. IEEE Workshop on the Representation of Visual Scenes*, IEEE, 1995.

[Williams83] Williams, L., ‘‘Pyramidal Parametrics,’’ *Computer Graphics (Proc. Siggraph ’83)*, Vol. 17, No. 3, July, 1983, pp. 1-11.

[Ziv77] Ziv, J., Lempel, A., ‘‘A universal algorithm for sequential data compression,’’ *IEEE Transactions on Information Theory*, IT-23:337-343, 1977.



(a) Buddha.

Input images: RenderMan renderings from an irregular polygon mesh model.

Light field: one slab, shown above in plan view. See also figure 3a.

(b) Kidney.

Input images: volume renderings from computed tomography data.

Light field: one slab, shown above. See also figure 4c.

(c) Hallway.

Input images: SGI RealityEngine renderings from a polygon model.

Light field: four slabs like figure 4c, arranged in a cross.

(d) Lion.

Input images: digitized video from a computer-controlled camera

Light field: four slabs arranged in a box. See also figure 3c.

Figure 14: Example images from four light fields, extracted during a typical interactive viewing session.

(a) Buddha. *Vector dimension:* 12 *Compression:* 45:1

(b) Lion. *Vector dimension:* 48 *Compression:* 118:1

Figure 15: Images extracted from compressed light fields.

The Lumigraph

Steven J. Gortler
Microsoft Research

Radek Grzeszczuk
University of Toronto*

Richard Szeliski
Microsoft Research

Michael F. Cohen
Microsoft Research

Abstract

This paper discusses a new method for capturing the complete appearance of both synthetic and real world objects and scenes, representing this information, and then using this representation to render images of the object from new camera positions. Unlike the shape capture process traditionally used in computer vision and the rendering process traditionally used in computer graphics, our approach does not rely on geometric representations. Instead we sample and reconstruct a 4D function, which we call a Lumigraph. The Lumigraph is a subset of the complete plenoptic function that describes the flow of light at all positions in all directions. With the Lumigraph, new images of the object can be generated very quickly, independent of the geometric or illumination complexity of the scene or object. The paper discusses a complete working system including the capture of samples, the construction of the Lumigraph, and the subsequent rendering of images from this new representation.

1 Introduction

The process of creating a virtual environment or object in computer graphics begins with modeling the geometric and surface attributes of the objects in the environment along with any lights. An image of the environment is subsequently rendered from the vantage point of a virtual camera. Great effort has been expended to develop computer aided design systems that allow the specification of complex geometry and material attributes. Similarly, a great deal of work has been undertaken to produce systems that simulate the propagation of light through virtual environments to create realistic images.

Despite these efforts, it has remained difficult or impossible to recreate much of the complex geometry and subtle lighting effects found in the real world. The modeling problem can potentially be bypassed by capturing the geometry and material properties of objects directly from the real world. This approach typically involves some combination of cameras, structured light, range finders, and mechanical sensing devices such as 3D digitizers. When successful, the results can be fed into a rendering program to create images of real objects and scenes. Unfortunately, these systems are still unable to completely capture small details in geometry and material properties. Existing rendering methods also continue to be limited in their capability to faithfully reproduce real world illumination, even if given accurate geometric models.

*Work performed while visiting Microsoft Research.

Quicktime VR [6] was one of the first systems to suggest that the traditional modeling/rendering process can be skipped. Instead, a series of captured environment maps allow a user to *look around* a scene from fixed points in space. One can also flip through different views of an object to create the illusion of a 3D model. Chen and Williams [7] and Werner et al [30] have investigated smooth interpolation between images by modeling the motion of pixels (i.e., the *optical flow*) as one moves from one camera position to another. In Plenoptic Modeling [19], McMillan and Bishop discuss finding the disparity of each pixel in stereo pairs of cylindrical images. Given the disparity (roughly equivalent to depth information), they can then move pixels to create images from new vantage points. Similar work using stereo pairs of planar images is discussed in [14].

This paper extends the work begun with Quicktime VR and Plenoptic Modeling by further developing the idea of capturing the complete flow of light in a region of the environment. Such a flow is described by a *plenoptic function*[1]. The plenoptic function is a five dimensional quantity describing the flow of light at every 3D spatial position (x, y, z) for every 2D direction (θ, ϕ). In this paper, we discuss computational methods for capturing and representing a plenoptic function, and for using such a representation to render images of the environment from any arbitrary viewpoint.

Unlike Chen and Williams' view interpolation [7] and McMillan and Bishop's plenoptic modeling [19], our approach does not rely explicitly on any optical flow information. Such information is often difficult to obtain in practice, particularly in environments with complex visibility relationships or specular surfaces. We do, however, use approximate geometric information to improve the quality of the reconstruction at lower sampling densities. Previous flow based methods implicitly rely on diffuse surface reflectance, allowing them to use a pixel from a single image to represent the appearance of a single geometric location from a variety of viewpoints. In contrast, our approach regularly samples the full plenoptic function and thus makes no assumptions about reflectance properties.

If we consider only the subset of light leaving a bounded object (or equivalently entering a bounded empty region of space), the fact that radiance along any ray remains constant[1] allows us to reduce the domain of interest of the plenoptic function to four dimensions. This paper first discusses the representation of this 4D function which we call a Lumigraph. We then discuss a system for sampling the plenoptic function with an inexpensive hand-held camera, and "developing" the captured light into a Lumigraph. Finally this paper describes how to use texture mapping hardware to quickly reconstruct images from any viewpoint with a virtual camera model. The Lumigraph representation is applicable to synthetic objects as well, allowing us to encode the complete appearance of a complex model and to rerender the object at speeds independent of the model complexity. We provide results on synthetic and real sequences and discuss work that is currently underway to make the system more efficient.

[1] We are assuming the medium (i.e., the air) to be transparent.

2 Representation

2.1 From 5D to 4D

The plenoptic function is a function of 5 variables representing position and direction [2]. If we assume the air to be transparent then the radiance along a ray through empty space remains constant. If we furthermore limit our interest to the light leaving the convex hull of a bounded object, then we only need to represent the value of the plenoptic function along some surface that surrounds the object. A cube was chosen for its computational simplicity (see Figure 1). At any point in space, one can determine the radiance along any ray in any direction, by tracing backwards along that ray through empty space to the surface of the cube. Thus, the plenoptic function due to the object can be reduced to 4 dimensions [3].

The idea of restricting the plenoptic function to some surrounding surface has been used before. In full-parallax holographic stereograms [3], the appearance of an object is captured by moving a camera along some surface (usually a plane) capturing a 2D array of photographs. This array is then transferred to a single holographic image, which can display the appearance of the 3D object. The work reported in this paper takes many of its concepts from holographic stereograms.

Global illumination researchers have used the "surface restricted plenoptic function" to efficiently simulate light-transfer between regions of an environment containing complicated geometric objects. The plenoptic function is represented on the surface of a cube surrounding some region; that information is all that is needed to simulate the light transfer from that region of space to all other regions [17]. In the context of illumination engineering, this idea has been used to model and represent the illumination due to physical luminaires. Ashdown [2] describes a gantry for moving a camera along a sphere surrounding a luminaire of interest. The captured information can then be used to represent the light source in global illumination simulations. Ashdown traces this idea of the surface-restricted plenoptic function back to Levin [15].

A limited version of the work reported here has been described by Katayama et al. [11]. In their system, a camera is moved along a track, capturing a 1D array of images of some object. This information is then used to generate new images of the object from other points in space. Because they only capture the plenoptic function along a line, they only obtain horizontal parallax, and distortion is introduced as soon as the new virtual camera leaves the line. Finally, in work concurrent to our own, Levoy and Hanrahan [16] represent a 4D function that allows for undistorted, full parallax views of the object from anywhere in space.

2.2 Parameterization of the 4D Lumigraph

There are many potential ways to parameterize the four dimensions of the Lumigraph. We adopt a parameterization similar to that used in digital holographic stereograms [9] and also used by Levoy and Hanrahan [16]. We begin with a cube to organize a Lumigraph and, without loss of generality, only consider for discussion a single square face of the cube (the full Lumigraph is constructed from six such faces).

[2] We only consider a snapshot of the function, thus time is eliminated. Without loss of generality, we also consider only a monochromatic function (in practice 3 discrete color channels), eliminating the need to consider wavelength. We furthermore ignore issues of dynamic range and thus limit ourselves to scalar values lying in some finite range.

[3] In an analogous fashion one can reconstruct the complete plenoptic function inside an empty convex region by representing it only on the surface bounding the empty region. At any point inside the region, one can find the light entering from any direction by finding that direction's intersection with the region boundary.

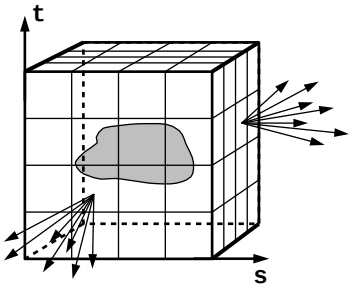


Figure 1: The surface of a cube holds all the radiance information due to the enclosed object.

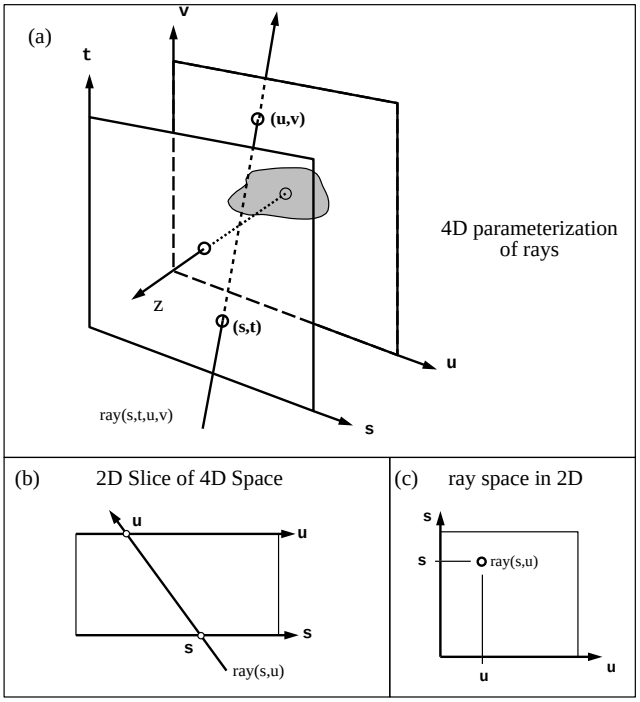


Figure 2: Parameterization of the Lumigraph

We choose a simple parameterization of the cube face with orthogonal axes running parallel to the sides labeled s and t (see Figure 1). Direction is parameterized using a second plane parallel to the st plane with axes labeled u and v (Figure 2). Any point in the 4D Lumigraph is thus identified by its four coordinates (s, t, u, v), the coordinates of a ray piercing the first plane at (s, t) and intersecting the second plane at (u, v) (see Ray(s, t, u, v) in Figure 2). We place the origin at the center of the uv plane, with the z axis normal to the plane. The st plane is located at $z = 1$. The full Lumigraph consists of six such pairs of planes with normals along the x, $-x$, y, $-y$, z, and $-z$ directions.

It will be instructive at times to consider two 2D analogs to the 4D Lumigraph. Figure 2(b) shows a 2D slice of the 4D Lumigraph that indicates the u and s axes. Figure 2(c) shows the same arrangement in *2D ray coordinates* in which rays are mapped to points (e.g., ray(s, u)) and points are mapped to lines.[4]

Figure 3 shows the relationship between this parameterization of the Lumigraph and a pixel in some arbitrary image. Given a Lu-

[4] More precisely, a line in ray space represents the set of rays through a point in space.

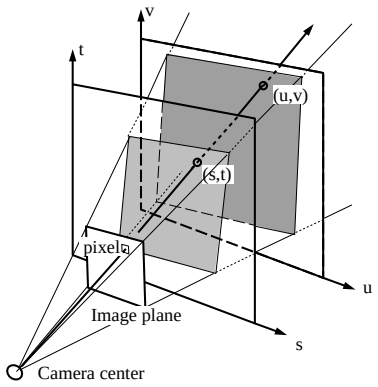


Figure 3: Relationship between Lumigraph and a pixel in an arbitrary image

migraph, L, one can generate an arbitrary new image coloring each pixel with the appropriate value $L(s,t,u,v)$. Conversely given some arbitrary image and the position and orientation of the camera, each pixel can be considered a sample of the Lumigraph value at (s,t,u,v) to be used to construct the Lumigraph.

There are many advantages of the two parallel plane parameterization. Given the geometric description of a ray, it is computationally simple to compute its coordinates; one merely finds its intersection with two planes. Moreover, reconstruction from this parameterization can be done rapidly using the texture mapping operations built into hardware on modern workstations (see section 3.6.2). Finally, in this parameterization, as one moves an eyepoint along the st plane in a straight line, the projection on the uv plane of points on the geometric object track along parallel straight lines. This makes it computationally efficient to compute the apparent motion of a geometric point (i.e., the *optical flow*), and to apply depth correction to the Lumigraph.

2.3 Discretization of the 4D Parameterization

So far, the Lumigraph has been discussed as an unknown, continuous, four dimensional function within a hypercubical domain in s,t,u,v and scalar range. To map such an object into a computational framework requires a discrete representation. In other words, we must choose some finite dimensional function space within which the function resides. To do so, we choose a discrete subdivision in each of the (s,t,u,v) dimensions and associate a coefficient and a basis function (reconstruction kernel) with each 4D grid point.

Choosing M subdivisions in the s and t dimensions and N subdivisions in u and v results in a grid of points on the st and uv planes (Figure 4). An st grid point is indexed with (i,j) and is located at (s_i,t_j). A uv grid point is indexed with (p,q) and is located at (u_p,v_q). A 4D grid point is indexed (i,j,p,q). The data value (in fact an RGB triple) at this grid point is referred to as $x_{i,j,p,q}$

2.3.1 Choice of Basis

We associate with each grid point a basis function $B_{i,j,p,q}$ so that the continuous Lumigraph is reconstructed as the linear sum

$$\tilde{L}(s,t,u,v) = \sum_{i=0}^{M}\sum_{j=0}^{M}\sum_{p=0}^{N}\sum_{q=0}^{N} x_{i,j,p,q}B_{i,j,p,q}(s,t,u,v)$$

where $\tilde{L}$ is a finite dimensional Lumigraph that exists in the space defined by the choice of basis.

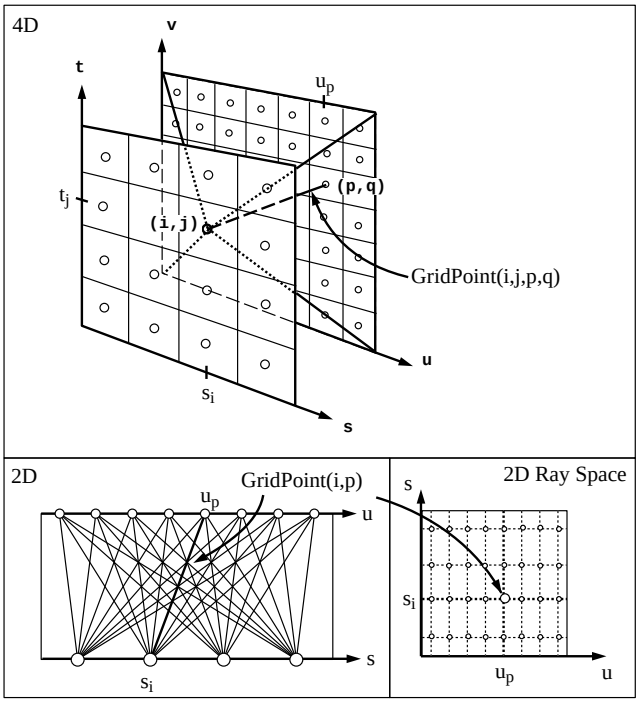


Figure 4: Discretization of the Lumigraph

For example, if we select constant basis functions (i.e., a 4D *box* with value 1 in the 4D region closest to the associated grid point and zero elsewhere), then the Lumigraph is piecewise constant, and takes on the value of the coefficient of the nearest grid point.

Similarly, a quadralinear basis function has a value of 1 at the grid point and drops off to 0 at all neighboring grid points. The value of $\tilde{L}(s,t,u,v)$ is thus interpolated from the 16 grid points forming the hypercube in which the point resides.

We have chosen to use the quadralinear basis for its computational simplicity and the C^0 continuity it imposes on $\tilde{L}$. However, because this basis is not band limited by the Nyquist frequency, and thus the corresponding finite dimensional function space is not shift invariant [24], the grid structure will be slightly noticeable in our results.

2.3.2 Projection into the Chosen Basis

Given a continuous Lumigraph, L, and a choice of basis for the finite dimensional Lumigraph, $\tilde{L}$, we still need to define a *projection* of L into $\tilde{L}$ (i.e., we need to find the coefficients x that result in an $\tilde{L}$ which is by some metric *closest* to L). If we choose the L^2 distance metric, then the projection is defined by integrating L against the *duals* of the basis functions [8], given by the inner products,

$$x_{i,j,p,q} = < L, \tilde{B}_{i,j,p,q} > \quad (1)$$

In the case of the box basis, $B = \tilde{B}$. The duals of the quadralinear basis functions are more complex, but these basis functions sufficiently approximate their own duals for our purposes.

One can interpret this projection as point sampling L after it has been low pass filtered with the kernel $\tilde{B}$. This interpretation is pursued in the context of holographic stereograms by Halle [9]. One can also interpret this projection as the result of placing a physical or synthetic "skewed" camera at grid point (s_i,t_j) with an aperture corresponding to the bilinear basis and with a pixel centered at

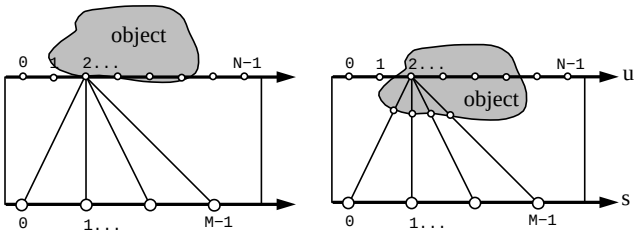


Figure 5: Choice of resolution on the uv plane

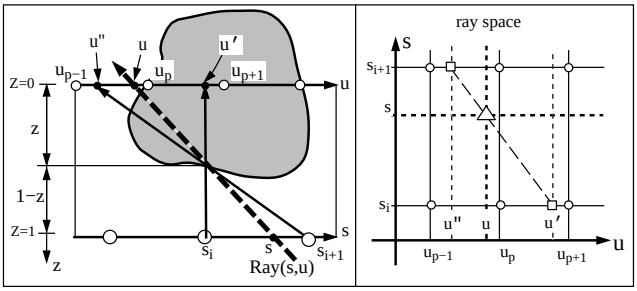


Figure 6: Depth correction of rays

(u_p, v_q) antialiased with a bilinear filter. This analogy is pursued in [16].

In Figure 16 we show images generated from Lumigraphs. The geometric scene consisted of a partial cube with the pink face in front, yellow face in back, and the brown face on the floor. These Lumigraphs were generated using two different quadrature methods to approximate equation 1, and using two different sets of basis functions, constant and quadralinear. In (a) and (c) only one sample was used to compute each Lumigraph coefficient. In these examples severe ghosting artifacts can be seen. In (b) and (d) numerical integration over the support of $\tilde{B}$ in st was computed for each coefficient. It is clear that best results are obtained using quadralinear basis function, with a full quadrature method.

2.3.3 Resolution

An important decision is how to set the resolutions, M and N, that best balance efficiency and the quality of the images reconstructed from the Lumigraph. The choices for M and N are influenced by the fact that we expect the visible surfaces of the object to lie closer to the uv plane than the st plane. In this case, N, the resolution of the uv plane, is closely related to the final image resolution and thus a choice for N close to final image resolution works best (we consider a range of resolutions from 128 to 512).

One can gain some intuition for the choice of M by observing the 2D subset of the Lumigraph from a single grid point on the uv plane (see $u = 2$ in Figure 5(a)). If the surface of the object lies exactly on the uv plane at a gridpoint, then all rays leaving that point represent samples of the radiance function at a single position on the object's surface. Even when the object's surface deviates from the uv plane as in Figure 5(b), we can still expect the function across the st plane to remain smooth and thus a low resolution is sufficient. Thus a significantly lower resolution for M than N can be expected to yield good results. In our implementation we use values of M ranging from 16 to 64.

2.3.4 Use of Geometric Information

Assuming the radiance function of the object is well behaved, knowledge about the geometry of the object gives us information about the coherence of the associated Lumigraph function, and can be used to help define the shape of our basis functions.

Consider the ray (s, u) in a two-dimensional Lumigraph (Figure 6). The closest grid point to this ray is (s_{i+1}, u_p). However, gridpoints (s_{i+1}, u_{p-1}) and (s_i, u_{p+1}) are likely to contain values closer to the true value at (s, u) since these grid points represent rays that intersect the object nearby the intersection with (s, u). This suggests adapting the shape of the basis functions.

Suppose we know the depth value z at which ray (s, u) first intersects a surface of the object. Then for a given s_i, one can compute a corresponding u' for a ray (s_i, u') that intersects the same geometric location on the object as the original ray (s, u)[5]. Let the depth z be 0 at the uv plane and 1 at the st plane. The intersections can then be found by examining the similar triangles in Figure 6,

$$u' = u + (s - s_i)\frac{z}{1-z} \tag{2}$$

It is instructive to view the same situation as in Figure 6(a), plotted in *ray space* (Figure 6(b)). In this figure, the triangle is the ray (s, u), and the circles indicate the nearby gridpoints in the discrete Lumigraph. The diagonal line passing through (s, u) indicates the *optical flow* (in this case, horizontal motion in 2D) of the intersection point on the object as one moves back and forth in s. The intersection of this line with s_i and s_{i+1} occurs at u' and u'' respectively.

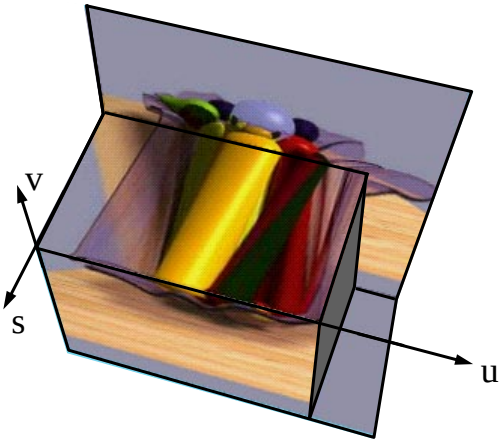


Figure 7: An (s, u, v) slice of a Lumigraph

Figure 7 shows an (s, u) slice through a three-dimensional (s, u, v) subspace of the Lumigraph for the ray-traced fruitbowl used in Figure 19. The flow of pixel motion is along straight lines in this space, but more than one motion may be present if the scene includes transparency. The slope of the flow lines corresponds to the depth of the point on the object tracing out the line. Notice how the function is coherent along these flow lines [4].

We expect the Lumigraph to be smooth along the optical flow lines, and thus it would be beneficial to have the basis functions adapt their shape correspondingly. The remapping of u and v values to u' and v' performs this reshaping. The idea of shaping the support of basis functions to closely match the structure of the function being approximated is used extensively in finite element methods. For example, in the Radiosity method for image synthesis, the mesh of elements is adapted to fit knowledge about the illumination function.

[5] Assuming there has been no change in visibility.

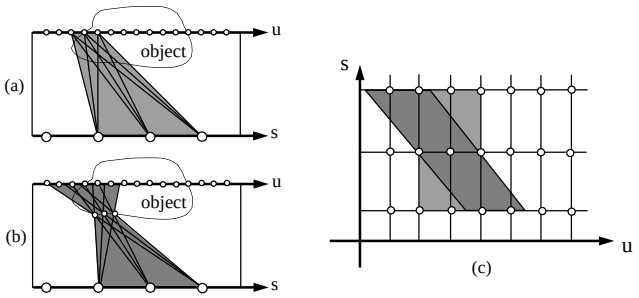


Figure 8: (a) Support of an uncorrected basis function. (b) Support of a depth corrected basis function. (c) Support of both basis functions in ray space.

The new basis function $B'_{i,j,p,q}(s,t,u,v)$ is defined by first finding u' and v' using equation 2 and then evaluating B, that is

$$B'_{i,j,p,q}(s,t,u,v) = B_{i,j,p,q}(s,t,u',v')$$

Although the shape of the new *depth corrected* basis is complicated, $\tilde{L}(s,t,u,v)$ is still a linear sum of coefficients and the weights of the contributing basis functions still sum to unity. However, the basis is no longer representable as a tensor product of simple boxes or hats as before. Figure 8 shows the support of an uncorrected (light gray) and a depth corrected (dark gray) basis function in 2D geometric space and in 2D ray space. Notice how the support of the depth corrected basis intersects the surface of the object across a narrower area compared to the uncorrected basis.

We use depth corrected quadralinear basis functions in our system. The value of $\tilde{L}(s,t,u,v)$ in the corrected quadralinear basis is computed using the following calculation:

```
QuadralinearDepthCorrect(s,t,u,v,z)
    Result = 0
    h_st = s_1 − s_0 /* grid spacing */
    h_uv = u_1 − u_0
    for each of the four (s_i, t_j) surrounding (s, t)
        u' = u + (s − s_i) * z/(1 − z)
        v' = v + (t − t_j) * z/(1 − z)
        temp = 0
        for each of the four (u_p, v_q) surrounding (u', v')
            interpWeight_uv =
                (h_uv − | u_p − u' |) * (h_uv − | v_q − v' |)/h_uv^2
            temp += interpWeight_uv * L(s_i, t_j, u_p, v_q)
        interpWeight_st =
            (h_st − | s_i − s |) * (h_st − | t_j − t |)/h_st^2
        Result += interpWeight_st * temp
    return Result
```

Figure 17 shows images generated from a Lumigraph using uncorrected and depth corrected basis functions. The depth correction was done using a 162 polygon model to approximate the original 70,000 polygons. The approximation was generated using a mesh simplification program [10]. These images show how depth correction reduces the artifacts present in the images.

3 The Lumigraph System

This section discusses many of the practical implementation issues related to creating a Lumigraph and generating images from it. Figure 9 shows a block diagram of the system. The process begins with capturing images with a hand-held camera. From known markers

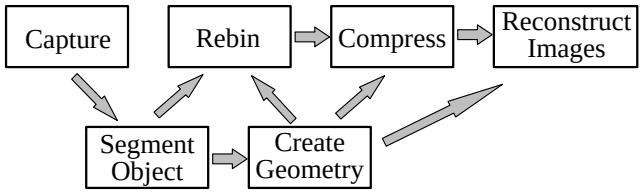


Figure 9: The Lumigraph system

in the image, the camera's position and orientation (its *pose*) is estimated. This provides enough information to create an approximate geometric object for use in the depth correction of (u,v) values. More importantly, each pixel in each image acts as a sample of the plenoptic function and is used to estimate the coefficients of the discrete Lumigraph (i.e., to *develop* the Lumigraph). Alternatively, the Lumigraph of a synthetic object can be generated directly by integrating a set of rays cast in a rendering system. We only briefly touch on compression issues. Finally, given an arbitrary virtual camera, new images of the object are quickly rendered.

3.1 Capture for Synthetic Scenes

Creating a Lumigraph of a synthetic scene is straightforward. A single sample per Lumigraph coefficient can be captured for each gridpoint (i,j) by placing the center of a virtual pin hole camera at (s_i,t_j) looking down the z axis, and defining the imaging frustum using the uv square as the film location. Rendering an image using this skewed perspective camera produces the Lumigraph coefficients. The pixel values in this image, indexed (p,q), are used as the Lumingraph coefficients $x_{i,j,p,q}$. To perform the integration against the kernel $\tilde{B}$, multiple rays per coefficient can be averaged by jittering the camera and pixel locations, weighting each image using $\tilde{B}$. For ray traced renderings, we have used the ray tracing program provided with the Generative Modeling package[25].

3.2 Capture for Real Scenes

Computing the Lumigraph for a real object requires the acquisition of object images from a large number of viewpoints. One way in which this can be accomplished is to use a special motion control platform to place the real camera at positions and orientations coincident with the (s_i,t_j) gridpoints [16]. While this is a reasonable solution, we are interested in acquiring the images with a regular hand-held camera. This results in a simpler and cheaper system, and may extend the range of applicability to larger scenes and objects.

To achieve this goal, we must first calibrate the camera to determine the mapping between directions and image coordinates. Next, we must identify special calibration markers in each image and compute the camera's pose from these markers. To enable depth-corrected interpolation of the Lumigraph, we also wish to recover a rough geometric model of the object. To do this, we convert each input image into a silhouette using a blue-screen technique, and then build a volumetric model from these binary images.

3.2.1 Camera Calibration and Pose Estimation

Camera calibration and pose estimation can be thought of as two parts of a single process: determining a mapping between screen pixels and rays in the world. The parameters associated with this process naturally divide into two sets: extrinsic parameters, which define the camera's pose (a rigid rotation and translation), and intrinsic parameters, which define a mapping of 3D camera coordinates onto the screen. This latter mapping not only includes a perspective (pinhole) projection from the 3D coordinates to undistorted

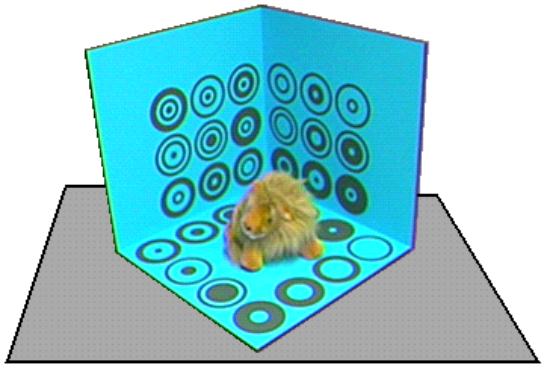


Figure 10: The capture stage

image coordinates, but also a radial distortion transformation and a final translation and scaling into screen coordinates [29, 31].

We use a camera with a fixed lens, thus the intrinsic parameters remain constant throughout the process and need to be estimated only once, before the data acquisition begins. Extrinsic parameters, however, change constantly and need to be recomputed for each new video frame. Fortunately, given the intrinsic parameters, this can be done efficiently and accurately with many fewer calibration points. To compute the intrinsic and extrinsic parameters, we employ an algorithm originally developed by Tsai [29] and extended by Willson [31].

A specially designed stage provides the source of calibration data (see Figure 10). The stage has two walls fixed together at a right angle and a base that can be detached from the walls and rotated in 90 degree increments. An object placed on such a movable base can be viewed from all directions in the upper hemisphere. The stage background is painted cyan for later blue-screen processing. Thirty markers, each of which consists of several concentric rings in a darker shade of cyan, are distributed along the sides and base. This number is sufficiently high to allow for a very precise intrinsic camera calibration. During the extrinsic camera calibration, only 8 or more markers need be visible to reliably compute a pose.

Locating markers in each image is accomplished by first converting the image into a binary (i.e., black or white) image. A *double thresholding* operator divides all image pixels into three groups separated by intensity thresholds T_1 and T_2. Pixels with an intensity below T_1 are considered black, pixels with an intensity above T_2 are considered white. Pixels with an intensity between T_1 and T_2 are considered black only if they have a black neighbor, otherwise they are considered white. The binary thresholded image is then searched for *connected components* [23]. Sets of connected components with similar centers of gravity are the likely candidates for the markers. Finally, the ratio of radii in each marker is used to uniquely identify the marker. To help the user correctly sample the viewing space, a real-time visual feedback displays the current and past locations of the camera in the view space (Figure 11). Marker tracking, pose estimation, feedback display, and frame recording takes approximately 1/2 second per frame on an SGI Indy.

3.3 3D Shape Approximation

The recovery of 3D shape information from natural imagery has long been a focus of computer vision research. Many of these techniques assume a particularly simple shape model, for example, a polyhedral scene where all edges are visible. Other techniques, such as stereo matching, produce sparse or incomplete depth estimates. To produce complete, closed 3D models, several approaches have been tried. One family of techniques builds 3D volumetric models

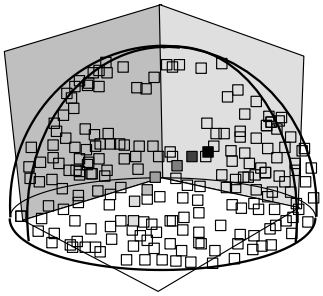


Figure 11: The user interface for the image capture stage displays the current and previous camera positions on a viewing sphere. The goal of the user is to "paint" the sphere.

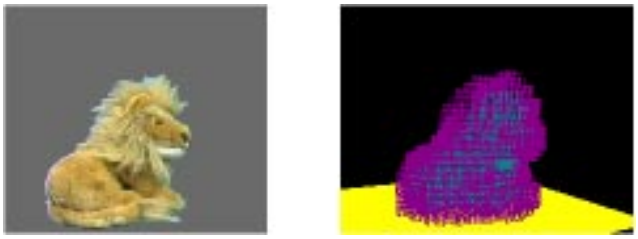


Figure 12: Segmented image plus volume construction

directly from silhouettes of the object being viewed [21]. Another approach is to fit a deformable 3D model to sparse stereo data. Despite over 20 years of research, the reliable extraction of accurate 3D geometric information from imagery (without the use of active illumination and positioning hardware) remains elusive.

Fortunately, a rough estimate of the shape of the object is enough to greatly aid in the capture and reconstruction of images from a Lumigraph. We employ the octree construction algorithm described in [26] for this process. Each input image is first segmented into a binary object/background image using a blue-screen technique [12] (Figure 12). An octree representation of a cube that completely encloses the object is initialized. Then for each segmented image, each voxel at a coarse level of the octree is projected onto the image plane and tested against the silhouette of the object. If a voxel falls outside of the silhouette, it is removed from the tree. If it falls on the boundary, it is marked for subdivision into eight smaller cubes. After a small number of images are processed, all marked cubes subdivide. The algorithm proceeds for a preset number of subdivisions, typically 4. The resulting 3D model consists of a collection of voxels describing a volume which is known to contain the object[6] (Figure 12). The external polygons are collected and the resulting polyhedron is then smoothed using Taubin's polyhedral smoothing algorithm [27].

3.4 Rebinning

As described in Equation 1, the coefficient associated with the basis function $B_{i,j,p,q}$ is defined as the integral of the continuous Lumigraph function multiplied by some kernel function $\tilde{B}$. This can be written as

$$x_{i,j,p,q} = \int L(s,t,u,v)\ \tilde{B}_{i,j,p,q}(s,t,u,v)\, ds\, dt\, du\, dv \qquad (3)$$

In practice this integral must be evaluated using a finite number of samples of the function L. Each pixel in the input video stream coming from the hand-held camera represents a single sample

[6] Technically, the volume is a superset of the *visual hull* of the object [13].

$L(s_k, t_k, u_k, v_k)$, of the Lumigraph function. As a result, the sample points in the domain cannot be pre-specified or controlled. In addition, there is no guarantee that the incoming samples are evenly spaced.

Constructing a Lumigraph from these samples is similar to the problem of multidimensional scattered data approximation. In the Lumigraph setting, the problem is difficult for many reasons. Because the samples are not evenly spaced, one cannot apply standard Fourier-based sampling theory. Because the number of sample points may be large ($\approx 10^8$) and because we are working in a 4 dimensional space, it is too expensive to solve systems of equations (as is done when solving thin-plate problems [28, 18]) or to build spatial data structures (such as Delauny triangulations).

In addition to the number of sample points, the distribution of the data samples have two qualities that make the problem particularly difficult. First, the sampling density can be quite sparse, with large gaps in many regions. Second, the sampling density is typically very non-uniform.

The first of these problems has been addressed in a two dimensional scattered data approximation algorithm described by Burt [5]. In his algorithm, a hierarchical set of lower resolution data sets is created using an image pyramid. Each of these lower resolutions represents a "blurred" version of the input data; at lower resolutions, the gaps in the data become smaller. This low resolution data is then used to fill in the gaps at higher resolutions.

The second of these problems, the non-uniformity of the sampling density, has been addressed by Mitchell [20]. He solves the problem of obtaining the value of a pixel that has been super-sampled with a non-uniform density. In this problem, when averaging the sample values, one does not want the result to be overly influenced by the regions sampled most densely. His algorithm avoids this by computing average values in a number of smaller regions. The final value of the pixel is then computed by averaging together the values of these strata. This average is not weighted by the number of samples falling in each of the strata. Thus, the non-uniformity of the samples does not bias the answer.

For our problem, we have developed a new hierarchical algorithm that combines concepts from both of these algorithms. Like Burt, our method uses a pyramid algorithm to fill in gaps, and like Mitchell, we ensure that the non-uniformity of the data does not bias the "blurring" step.

For ease of notation, the algorithm is described in 1D, and will use only one index i. A hierarchical set of basis functions is used, with the highest resolution labeled 0 and with lower resolutions having higher indices. Associated with each coefficient x_i^r at resolution r is a weight w_i^r. These weights determine how the coefficients at different resolution levels are eventually combined. The use of these weights is the distinguishing feature of our algorithm.

The algorithm proceeds in three phases. In the first phase, called *splat*, the sample data is used to approximate the integral of Equation 3, obtaining coefficients x_i^0 and weights w_i^0. In regions where there is little or no nearby sample data, the weights are small or zero. In the second phase, called *pull*, coefficients are computed for basis functions at a hierarchical set of lower resolution grids by combining the coefficient values from the higher resolution grids. In the lower resolution grids, the gaps (regions where the weights are low) become smaller (see figure 13). In the third phase, called *push*, information from the each lower resolution grid is combined with the next higher resolution grid, filling in the gaps while not unduly blurring the higher resolution information already computed.

3.4.1 Splatting

In the splatting phase, coefficients are computed by performing Monte-Carlo integration using the following weighted average estimator:

$$\begin{array}{rcll} w_i^0 & = & & \sum_k \tilde{B}_i(s_k) \\ x_i^0 & = & \frac{1}{w_i^0} & \sum_k \tilde{B}_i(s_k)\, L(s_k) \end{array} \tag{4}$$

where s_k denotes the domain location of sample k. If w_i^0 is 0, then the x_i^0 is undefined. If the $\tilde{B}_i$ have compact support, then each sample influences only a constant number of coefficients. Therefore, this step runs in time linear in the number of samples.

If the sample points s_k are chosen from a uniform distribution, this estimator converges to the correct value of the integral in Equation (3), and for n sample points has a variance of approximately $\frac{1}{n} \int \left(\tilde{B}_i(s)\, L(s) - x_i\, \tilde{B}_i(s)\right)^2 ds$. This variance is similar to that obtained using importance sampling, which is often much smaller than the crude Monte Carlo estimator. For a full analysis of this estimator, see [22].

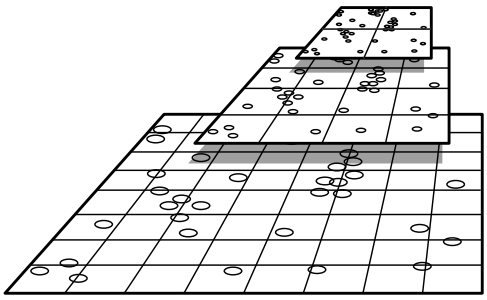


Figure 13: 2D pull-push. At lower resolutions the gaps are smaller.

3.4.2 Pull

In the *pull* phase, lower resolution approximations of the function are derived using a set of wider kernels. These wider kernels are defined by linearly summing together the higher resolution kernels ($\tilde{B}_i^{r+1} = \sum_k \tilde{h}_{k-2i} \tilde{B}_k^r$) using some discrete sequence $\tilde{h}$. For linear "hat" functions, $\tilde{h}[-1..1]$ is $\{\frac{1}{2}, 1, \frac{1}{2}\}$

The lower resolution coefficients are computed by combining the higher resolution coefficients using $\tilde{h}$. One way to do this would be to compute

$$\begin{array}{rcll} w_i^{r+1} & = & & \sum_k \tilde{h}_{k-2i}\, w_k^r \\ x_i^{r+1} & = & \frac{1}{w_i^{r+1}} & \sum_k \tilde{h}_{k-2i}\, w_k^r\, x_k^r \end{array} \tag{5}$$

It is easy to see that this formula, which corresponds to the method used by Burt, computes the same result as would the original estimator (Equation (4)) applied to the wider kernels. Once again, this estimator works if the sampling density is uniform. Unfortunately, when looking on a gross scale, it is imprudent to assume that the data is sampled uniformly. For example, the user may have held the camera in some particular region for a long time. This non-uniformity can greatly bias the estimator.

Our solution to this problem is to apply Mitchell's reasoning to this context, replacing Equation (5) with:

$$\begin{array}{rcll} w_i^{r+1} & = & & \sum_k \tilde{h}_{k-2i}\ \min(w_k^r, 1) \\ x_i^{r+1} & = & \frac{1}{w_i^{r+1}} & \sum_k \tilde{h}_{k-2i}\ \min(w_k^r, 1)\, x_k^r \end{array}$$

The value 1 represents full saturation[7], and the $\min$ operator is used to place an upper bound on the degree that one coefficient in a highly

[7] Using the value 1 introduces no loss of generality if the normalization of $\tilde{h}$ is not fixed.

sampled region, can influence the total sum [8].

The pull stage runs in time linear in the number of basis function summed over all of the resolutions. Because each lower resolution has half the density of basis functions, this stage runs in time linear in the number of basis functions at resolution 0.

3.4.3 Push

During the push stage, the lower resolution approximation is used to fill in the regions in the higher resolution that have low weight [9]. If a higher resolution coefficient has a high associated confidence (i.e., has weight greater than one), we fully disregard the lower resolution information there. If the higher resolution coefficient does not have sufficient weight, we blend in the information from the lower resolution.

To blend this information, the low resolution approximation of the function must be expressed in the higher resolution basis. This is done by upsampling and convolving with a sequence h, that satisfies $B_i^{r+1} = \sum_k h_{k-2i} B_k^r$.

We first compute temporary values

$$\begin{aligned} tw_i^r &= \sum_k h_{i-2k} \ \min(w_k^{r+1}, 1) \\ tx_i^r &= \tfrac{1}{tw_i^r} \sum_k h_{i-2k} \ \min(w_k^{r+1}, 1)\, x_k^{r+1} \end{aligned}$$

These temporary values are now ready to be blended with the values x and w values already at level r.

$$\begin{aligned} x_i^r &= tx_i^r\,(1 - w_i^r) + w_i^r\, x_i^r \\ w_i^r &= tw_i^r\,(1 - w_i^r) + w_i^r \end{aligned}$$

This is analogous to the blending performed in image compositing.

3.4.4 Use of Geometric Information

This three phase algorithm must be adapted slightly when using the depth corrected basis functions B'. During the splat phase, each sample ray $L(s_k, t_k, u_k, v_k)$ must have its u and v values remapped as explained in Section 2.3.4. Also, during the push and pull phases, instead of simply combining coefficients using basis functions with neighboring indices, depth corrected indices are used.

3.4.5 2D Results

The validity of the algorithm was tested by first applying it to a 2D image. Figure 18 (a) shows a set of scattered samples from the well known mandrill image. The samples were chosen by picking 256 random line segments and sampling the mandrill very densely along these lines [10]. Image (b) shows the resulting image after the *pull/push* algorithm has been applied. Image (c) and (d) show the same process but with only 100 sample lines. The success of our algorithm on both 2D image functions and 4D Lumigraph functions leads us to believe that it may have many other uses.

3.5 Compression

A straightforward sampling of the Lumigraph requires a large amount of storage. For the examples shown in section 4, we use, for a single face, a 32×32 sampling in (s, t) space and 256×256 (u, v) images. To store the six faces of our viewing cube with 24-bits per pixel requires $32^2 \cdot 256^2 \cdot 6 \cdot 3 = 1.125$GB of storage.

Fortunately, there is a large amount of coherence between (s, t, u, v) samples. One could apply a transform code to the 4D array, such as a wavelet transform or block DCT. Given geometric information, we can expect to do even better by considering the 4D array as a 2D array of images. We can then *predict* new (u, v) images from adjacent images, (i.e., images at adjacent (s, t) locations). Intraframe compression issues are identical to compressing single images (a simple JPEG compression yields about a 20:1 savings). Interframe compression can take advantage of increased information over other compression methods such as MPEG. Since we know that the object is static and know the camera motion between adjacent images, we can predict the motion of pixels. In addition, we can leverage the fact that we have a 2D array of images rather than a single linear video stream.

Although we have not completed a full analysis of compression issues, our preliminary experiments suggest that a 200:1 compression ratio should be achievable with almost no degradation. This reduces the storage requirements to under 6MB. Obviously, further improvements can be expected using a more sophisticated prediction and encoding scheme.

3.6 Reconstruction of Images

Given a desired camera (position, orientation, resolution), the reconstruction phase colors each pixel of the output image with the color that this camera would create if it were pointed at the real object.

3.6.1 Ray Tracing

Given a Lumigraph, one may generate a new image from an arbitrary camera pixel by pixel, ray by ray. For each ray, the corresponding (s, t, u, v) coordinates are computed, the nearby grid points are located, and their values are properly interpolated using the chosen basis functions (see Figure 3).

In order to use the depth corrected basis functions given an approximate object, we transform the (u, v) coordinates to the depth corrected (u', v') before interpolation. This depth correction of the (u, v) values can be carried out with the aid of graphics hardware. The polygonal approximation of the object is drawn from the point of view and with the same resolution as the desired image. Each vertex is assigned a *red, green, blue* value corresponding to its (x, y, z) coordinate resulting in a "depth" image. The corrected depth value is found by examining the blue value in the corresponding pixel of the depth image for the $\pm z$-faces of the Lumigraph cube (or the red or green values for other faces). This information is used to find u' and v' with Equation 2.

3.6.2 Texture mapping

The expense of tracing a ray for each pixel can be avoided by reconstructing images using texture mapping operations. The st plane itself is tiled with texture mapped polygons with the textures defined by slices of the Lumigraph: $\mathrm{tex}_{i,j}(u_p, v_q) = x_{i,j,p,q}$. In other words, we have one texture associated with each st gridpoint.

Constant Basis

Consider the case of constant basis functions. Suppose we wish to render an image from the desired camera shown in Figure 14. The set of rays passing through the shaded square on the st plane have (s, t) coordinates closest to the grid point (i, j). Suppose that the uv plane is filled with $\mathrm{tex}_{i,j}$. Then, when using constant basis functions, the shaded region in the desired camera's film plane should be filled with the corresponding pixels in the shaded region of the uv plane. This computation can be accomplished by placing a virtual camera at the desired location, drawing a square polygon on the

[8] This is actually less extreme that Mitchell's original algorithm. In this context, his algorithm would set all non-zero weights to 1.

[9] Variance measures could be used instead of weight as a measure of confidence in this phase.

[10] We chose this type of sampling pattern because it mimics in many ways the structure of the Lumigraph samples taken from a hand-held camera. In that case each input video image is a dense sampling of the 4D Lumigraph along a 2D plane.

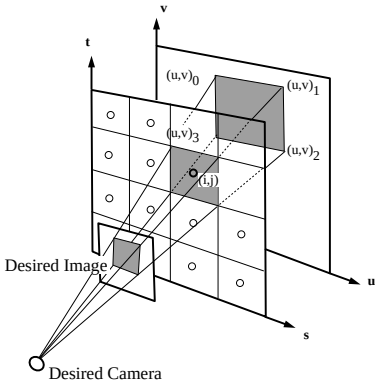


Figure 14: Texture mapping a portion of the *st* plane

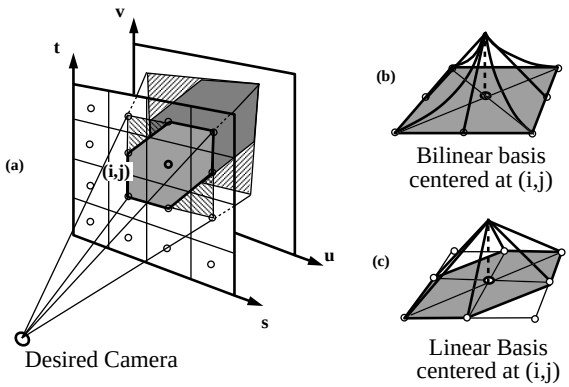


Figure 15: Quadralinear vs. linear-bilinear

st plane, and texture mapping it using the four texture coordinates $(u,v)_0$, $(u,v)_1$, $(u,v)_2$, and $(u,v)_3$ to index into $\mathrm{tex}_{i,j}$.

Repeating this process for each grid point on the st plane and viewing the result from the desired camera results in a complete reconstruction of the desired image. Thus, if one has an $M \times M$ resolution for the st plane, one needs to draw at most M^2 texture mapped squares, requiring on average, only one ray intersection for each square since the vertices are shared. Since many of the M^2 squares on the st plane are invisible from the desired camera, typically only a small fraction of these squares need to be rendered. The rendering cost is independent of the resolution of the final image.

Intuitively, you can think of the st plane as a piece of holographic film. As your eye moves back and forth you see different things at the same point in st since each point holds a complete image.

Quadralinear Basis

The reconstruction of images from a quadralinear basis Lumigraph can also be performed using a combination of texture mapping and alpha blending. In the quadralinear basis, the support of the basis function at i, j covers a larger square on the st plane than does the box basis (see Figure 15(a)). Although the regions do not overlap in the constant basis, they do in the quadralinear basis. For a given pixel in the desired image, values from 16 4D grid points contribute to the final value.

The quadralinear interpolation of these 16 values can be carried out as a sequence of bilinear interpolations, first in uv and then in st. A bilinear basis function is shown in Figure 15(b) centered at grid point (i, j). A similar basis would lie over each grid point in uv and every grid point in st.

Texture mapping hardware on an SGI workstation can automatically carry out the bilinear interpolation of the texture in uv. Unfortunately, there is no hardware support for the st bilinear interpolation. We could approximate the bilinear pyramid with a linear pyramid by drawing the four triangles shown on the floor of the basis function in Figure 15(b). By assigning α values to each vertex ($\alpha = 1$ at the center, and $\alpha = 0$ at the outer four vertices) and using alpha blending, the final image approximates the full quadralinear interpolation with a linear-bilinear one. Unfortunately, such a set of basis functions do not sum to unity which causes serious artifacts.

A different pyramid of triangles can be built that does sum to unity and thus avoids these artifacts. Figure 15(c) shows a hexagonal region associated with grid point (i, j) and an associated linear basis function. We draw the six triangles of the hexagon with $\alpha = 1$ at the center and $\alpha = 0$ at the outside six vertices[11]. The linear interpolation of α values together with the bilinear interpolation of the texture map results in a linear-bilinear interpolation. In practice we have found it to be indistinguishable from the full quadralinear interpolation. This process requires at most $6\,M^2$ texture mapped, α-blended triangles to be drawn.

Depth Correction

As before, the (u, v) coordinates of the vertices of the texture mapped triangles can be depth corrected. At interior pixels, the depth correction is only approximate. This is not valid when there are large depth changes within the bounds of the triangle. Therefore, we adaptively subdivide the triangles into four smaller ones by connecting the midpoints of the sides until they are (a) smaller than a minimum screen size or (b) have a sufficiently small variation in depth at the three corners and center. The α values at intermediate vertices are the average of the vertices of the parent triangles.

4 Results

We have implemented the complete system described in this paper and have created Lumigraphs of both synthetic and actual objects. For synthetic objects, Lumigraphs can be created either from polygon rendered or ray traced images. Computing all of the necessary images is a lengthy process often taking weeks of processing time.

For real objects, the capture is performed with an inexpensive, single chip Panasonic analog video camera. The capture phase takes less than one hour. The captured data is then "developed" into a Lumigraph. This off-line processing, which includes segmenting the image from its background, creating an approximate volumetric representation, and rebinning the samples, takes less than one day of processing on an SGI Indy workstation.

Once the Lumigraph has been created, arbitrary new images of the object or scene can be generated. One may generate these new images on a ray by ray basis, which takes a few seconds per frame at 450×450 resolution. If one has hardware texture mapping available, then one may use the acceleration algorithm described in Section 3.6.2. This texture mapping algorithm is able to create multiple frames per second from the Lumigraph on an SGI Reality Engine. The rendering speed is almost independent of the desired resolution of the output images. The computational bottleneck is moving the data from main memory to the smaller texture cache.

Figure 19 shows images of a synthetic fruit bowl, an actual fruit bowl, and a stuffed lion, generated from Lumigraphs. No geometric information was used in the Lumigraph of the synthetic fruit bowl. For the actual fruit bowl and the stuffed lion, we have used the approximate geometry that was computed using the silhouette information. These images can be generated in a fraction of a second, independent of scene complexity. The complexity of both the geometry and the lighting effects present in these images would be difficult to achieve using traditional computer graphics techniques.

[11] The alpha blending mode is set to perform a simple summation.

5 Conclusion

In this paper we have described a rendering framework based on the plenoptic function emanating from a static object or scene. Our method makes no assumptions about the reflective properties of the surfaces in the scene. Moreover, this representation does not require us to derive any geometric knowledge about the scene such as depth. However, this method does allow us to include any geometric knowledge we may compute, to improve the efficiency of the representation and improve the quality of the results. We compute the approximate geometry using silhouette information.

We have developed a system for capturing plenoptic data using a hand-held camera, and converting this data into a Lumigraph using a novel rebinning algorithm. Finally, we have developed an algorithm for generating new images from the Lumigraph quickly using the power of texture mapping hardware.

In the examples shown in this paper, we have not captured the complete plenoptic function surrounding an object. We have limited ourselves to only one face of a surrounding cube. There should be no conceptual obstacles to extending this work to complete captures using all six cube faces.

There is much future work to be done on this topic. It will be important to develop powerful compression methods so that Lumigraphs can be efficiently stored and transmitted. We believe that the large degree of coherence in the Lumigraph will make a high rate of compression achievable. Future research also includes improving the accuracy of our system to reduce the amount of artifacts in the images created by the Lumigraph. With these extensions we believe the Lumigraph will be an attractive alternative to traditional methods for efficiently storing and rendering realistic 3D objects and scenes.

Acknowledgments

The authors would like to acknowledge the help and advice we received in the conception, implementation and writing of this paper. Thanks to Marc Levoy and Pat Hanrahan for discussions on issues related to the 5D to 4D simplification, the two-plane parameterization and the camera based aperture analog. Jim Kajiya and Tony DeRose provided a terrific sounding board throughout this project. The ray tracer used for the synthetic fruit bowl was written by John Snyder. The mesh simplification code used for the bunny was written by Hugues Hoppe. Portions of the camera capture code were implemented by Matthew Turk. Jim Blinn, Hugues Hoppe, Andrew Glassner and Jutta Joesch provided excellent editing suggestions. Erynn Ryan is deeply thanked for her creative crisis management. Finally, we wish to thank the anonymous reviewers who pointed us toward a number of significant references we had missed.

References

[1] ADELSON, E. H., AND BERGEN, J. R. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, Landy and Movshon, Eds. MIT Press, Cambridge, Massachusetts, 1991, ch. 1.

[2] ASHDOWN, I. Near-field photometry: A new approach. *Journal of the Illumination Engineering Society 22*, 1 (1993), 163–180.

[3] BENTON, S. A. Survey of holographic stereograms. *Proceedings of the SPIE 391* (1982), 15–22.

[4] BOLLES, R. C., BAKER, H. H., AND MARIMONT, D. H. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision 1* (1987), 7–55.

[5] BURT, P. J. Moment images, polynomial fit filters, and the problem of surface interpolation. In *Proceedings of Computer Vision and Pattern Recognition* (June 1988), IEEE Computer Society Press, pp. 144–152.

[6] CHEN, S. E. Quicktime VR - an image-based approach to virtual environment navigation. In *Computer Graphics, Annual Conference Series, 1995*, pp. 29–38.

[7] CHEN, S. E., AND WILLIAMS, L. View interpolation for image synthesis. In *Computer Graphics, Annual Conference Series, 1993*, pp. 279–288.

[8] CHUI, C. K. *An Introduction to Wavelets*. Academic Press Inc., 1992.

[9] HALLE, M. W. Holographic stereograms as discrete imaging systems. *Practical Holography VIII (SPIE) 2176* (1994), 73–84.

[10] HOPPE, H. Progressive meshes. In *Computer Graphics, Annual Conference Series, 1996*.

[11] KATAYAMA, A., TANAKA, K., OSHINO, T., AND TAMURA, H. A viewpoint independent stereoscopic display using interpolation of multi-viewpoint images. *Steroscopic displays and virtal reality sytems II (SPIE) 2409* (1995), 11–20.

[12] KLINKER, G. J. *A Physical Approach to Color Image Understanding*. A K Peters, Wellesley, Massachusetts, 1993.

[13] LAURENTINI, A. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 2 (February 1994), 150–162.

[14] LAVEAU, S., AND FAUGERAS, O. 3-D scene representation as a collection of images and fundamental matrices. Tech. Rep. 2205, INRIA-Sophia Antipolis, February 1994.

[15] LEVIN, R. E. Photometric characteristics of light-controlling apparatus. *Illuminating Engineering 66*, 4 (1971), 205–215.

[16] LEVOY, M., AND HANRAHAN, P. Light-field rendering. In *Computer Graphics, Annual Conference Series, 1996*.

[17] LEWIS, R. R., AND FOURNIER, A. Light-driven global illumination with a wavelet representation of light transport. UBC CS Technical Reports 95-28, University of British Columbia, 1995.

[18] LITWINOWICZ, P., AND WILLIAMS, L. Animating images with drawings. In *Computer Graphics, Annual Conference Series, 1994*, pp. 409–412.

[19] MCMILLAN, L., AND BISHOP, G. Plenoptic modeling: An image-based rendering system. In *Computer Graphics, Annual Conference Series, 1995*, pp. 39–46.

[20] MITCHELL, D. P. Generating antialiased images at low sampling densities. *Computer Graphics 21*, 4 (1987), 65–72.

[21] POTMESIL, M. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing 40* (1987), 1–29.

[22] POWELL, M. J. D., AND SWANN, J. Weighted uniform sampling - a monte carlo technique for reducing variance. *J. Inst. Maths Applics 2* (1966), 228–236.

[23] ROSENFELD, A., AND KAK, A. C. *Digital Picture Processing*. Academic Press, New York, New York, 1976.

[24] SIMONCELLI, E. P., FREEMAN, W. T., ADELSON, E. H., AND HEEGER, D. J. Shiftable multiscale transforms. *IEEE Transactions on Information Theory 38* (1992), 587–607.

[25] SNYDER, J. M., AND KAJIYA, J. T. Generative modeling: A symbolic system for geometric modeling. *Computer Graphics 26*, 2 (1992), 369–379.

[26] SZELISKI, R. Rapid octree construction from image sequences. *CVGIP: Image Understanding 58*, 1 (July 1993), 23–32.

[27] TAUBIN, G. A signal processing approach to fair surface design. In *Computer Graphics, Annual Conference Series, 1995*, pp. 351–358.

[28] TERZOPOULOS, D. Regularization of inverse visual problems involving discontinuities. *IEEE PAMI 8*, 4 (July 1986), 413–424.

[29] TSAI, R. Y. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation RA-3*, 4 (August 1987), 323–344.

[30] WERNER, T., HERSCH, R. D., AND HLAVAC, V. Rendering real-world objects using view interpolation. In *Fifth International Conference on Computer Vision (ICCV'95)* (Cambridge, Massachusetts, June 1995), pp. 957–962.

[31] WILLSON, R. G. *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Carnegie Mellon University, 1994.

Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping

Paul Debevec, George Borshukov, and Yizhou Yu

9th Eurographics Rendering Workshop, Vienna, June 1998

George Drettakis and Nelson Max, eds.

Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping

Paul Debevec, Yizhou Yu, and George Borshukov
Univeristy of California at Berkeley
debevec@cs.berkeley.edu

Abstract. This paper presents how the image-based rendering technique ofview-dependent texture-mapping (VDTM) can be efficiently implementedusing projective texture mapping, a feature commonly available inpolygon graphics hardware. VDTM is a technique for generating novelviews of a scene with approximately known geometry making maximal useof a sparse set of original views. The original presentation of VDTMby Debevec, Taylor, and Malik required significant per-pixelcomputation and did not scale well with the number of original images.In our technique, we precompute for each polygon the set of originalimages in which it is visible and create a "view map" data structurethat encodes the best texture map to use for a regularly sampled setof possible viewing directions. To generate a novel view, the viewmap for each polygon is queried to determine a set of no more thanthree original images to blend together to render the polygon.Invisible triangles are shaded using an object-space hole-fillingmethod. We show how the rendering process can be streamlined forimplementation on standard polygon graphics hardware, and presentresults of using the method to render a large-scale model of theBerkeley bell tower and its surrounding campus environment.

1 Introduction

A clear application of image-based modeling and rendering techniques will be in the creation and display of realistic virtual environments of real places. Acquiring geometric models of environments has been the subject of research in interactive image-based modeling techniques, and is now becoming practical to perform with techniques such as laser scanning or interactive photogrammetry. Acquiring the corresponding appearance information (under given lighting conditions) is easily performed with a digital camera. The remaining challenge is to use the recovered geometry and the available real views to generate novel views of the scene quickly and realistically.

In addressing this problem, it is important to make judicious use of all the available views, especially when a particular surface is seen from different directions in multiple images. This problem was addressed in [2], which presented view-dependent texture mapping as a means to render each pixel of a novel view as a blend of its corresponding pixels in the original views. However, the technique presented did not guarantee smooth blending between images as the viewpoint changed and did not scale well with the number of available views.

In this paper we reformulate view-dependent texture-mapping to guarantee smooth blending between images, to scale well with the number of views, and to make efficient use of projective polygon texture-mapping hardware. The result is an effective and efficient technique for generating virtual views of a scene under the following conditions:

- A reasonably accurate geometric model of the scene is available

- A set of calibrated photographs (with known locations and known imaging geometry) is available
- The photographs are taken in the same lighting conditions
- The photographs generally observe each surface of the scene from a few different angles
- Surfaces in the scene are not extremely specular

2 Previous Work

Early image-based modeling and rendering work [16, 5, 8], presented methods of using image depth or image correspondences to reproject the pixels from one camera position to the viewpoint of another. However, the work did not concentrate on how to combine appearance information from multiple images to optimally produce novel views.

View-Dependent Texture Mapping (VDTM) was presented in [2] as a method of rendering interactively constructed 3D architectural scenes using images taken from multiple locations. The method attempted to make full use of the available imagery using the following principle: to generate a novel view of a particular surface patch in the scene, the best original image from which to sample reflectance information is the image that observed the patch from as close a direction as possible as the desired novel view. As an example, suppose that a particular surface of a building is seen in three original images from the left, front, and right. If one is generating a novel view from the left, one would want to use the surface's appearance in the left view as the texture map. Similarly, for a view in front of the surface one would most naturally use the frontal view. For an animation of moving from the left to the front, it would make sense to smoothly blend, as in morphing, between the left and front texture maps during the animation in order to prevent the texture map suddenly changing from one frame to the next. As a result, the view-dependent texture mapping approach allows renderings to be considerably more realistic than static texture-mapping allows, since it better represents non-diffuse reflectance and can simulate the appearance of unmodeled geometry.

Other image-based modeling and rendering work has addressed the problem of blending between available views of the scene in order to produce renderings. In [6], blending is performed amongst a dense regular sampling of images in order to generate novel views. Since scene geometry is not used, a very large number of views is necessary to produce even low-resolution renderings. [4] is similar to [6] but uses irregularly sampled views and leverages approximate scene geometry derived from object silhouettes. View-dependent texture-mapping, used with a dense sampling of images and with simple geometry, reduces to the light field approach. The representation used in the presented methods restricts the viewpoint to be outside the convex hull of an object or inside a convex empty region of space. This restriction, and the number of images necessary, could complicate using these methods for acquiring and rendering a large environment. The work in this paper leverages the light field methods to render each surface of a model as a light field constructed from a sparse set of views; since the model is assumed to conform well to the scene and the scene is assumed to be predominantly diffuse, far fewer images are necessary to achieve coherent results.

The implementation of VDTM in [2] computed texture weighting on a per-pixel basis, required visibility calculations to be performed at rendering time, examined every original view to produce every novel view, and only blended between the two closest viewpoints available. As a result, it was computationally expensive (several minutes per frame) and did not always guarantee the image blending to vary smoothly as the viewpoint changed. Subsequent work [9, 7] presented more efficient methods for op-

tically compositing multiple re-rendered views of a scene. In this work we associate appearance information with surfaces, rather than with viewpoints, in order to better interpolate between widely spaced viewpoints in which each sees only a part of the scene. We use visibility preprocessing, polygon view maps, and projective texture mapping to implement our technique.

3 Overview of the Method

Our method for VDTM first preprocesses the scene to determine which images observe which polygons from which directions. This preprocessing occurs as follows:

1. **Compute Visibility**: For each polygon, determine in which images it is seen. Split polygons that are partially seen in one of the images. (Section 5).
2. **Fill Holes**: For each polygon not seen in any view, choose appropriate vertex colors for performing Gouraud shading. (Section 5).
3. **Construct View Maps**: For each polygon, store the index of the image closest in viewing angle for each direction of a regularly sampled viewing hemisphere. (Section 7).

The rendering algorithm (Section 8) runs as follows:

1. Draw all polygons seen in none of the original views using the vertex colors determined during hole filling.
2. Draw all polygons which are seen in just one view.
3. For each polygon seen in more than one view, calculate its viewing direction for the desired novel view. Calculate where the novel view falls within the view map, and then determine the three closest viewing directions and their relative weights. Render the polygon using alpha-blending of the three textures with projective texture mapping.

4 Image-Based Rendering with Projective Texture Mapping

Projective texture mapping was introduced in [10] and is now part of the OpenGL graphics standard. Although the original paper used it only for shadows and lighting effects, it is directly applicable to image-based rendering because it can simulate the inverse projection of taking photographs with a camera. In order to perform projective texture mapping, the user specifies a virtual camera position and orientation, and a virtual image plane with the texture. The texture is then cast onto a geometric model using the camera position as the center of projection. The focus of this paper is to adapt projective texture-mapping to take advantage of multiple images of the scene via view-dependent texture-mapping.

Of course, we should only map a particular image onto the portions of the scene that are visible from its original camera viewpoint. The OpenGL implementation of projective texture mapping does not automatically perform such visibility checks; instead a texture map will project through any amount of geometry and be mapped onto occluded polygons as seen in Fig. 1. Thus, we need to explicitly compute visibility information before performing projective texture-mapping.

We could solve the visibility problem in image-space using ray tracing, an item buffer, or a shadow buffer (as in [2]). However, such methods would require us to compute visibility in image-space for each novel view, which is computationally expensive

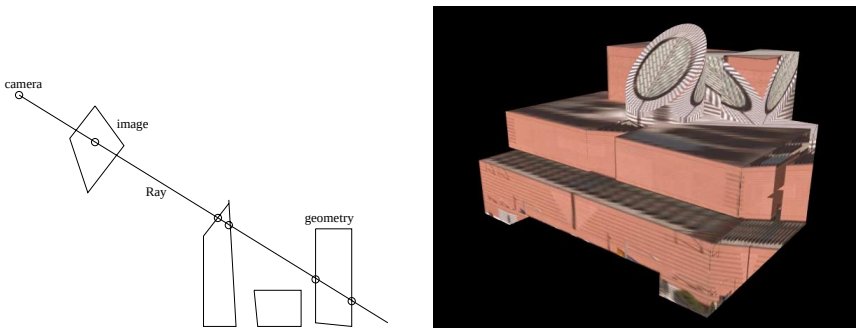


Fig. 1. The current hardware implementation of projective texture mapping in OpenGL lets the texture pass through the geometry and be mapped onto all backfacing and occluded polygons on the path of the ray, as can be seen in this rendering of a building on the right. Thus it is necessary to perform visibility pre-processing so that only polygons visible to a particular camera are texture-mapped with the corresponding image.

and not suited to interactive applications. Projective texture-mapping is extremely efficient if we know beforehand which polygons to texture-map, which suggests that we employ a visibility preprocessing step in object-space to determine which polygons are visible to which cameras. The next section describes such a visibility preprocessing method.

5 Determining Visibility

The purpose of our visibility algorithm is to determine for each polygon in the model in which images it is visible, and to split polygons as necessary so that each is fully visible or fully invisible to any particular camera. Polygons are clipped to the camera viewing frustums, to each other, and to user-specified clipping regions. This algorithm operates in both object space [3, 15] and image space and runs as follows:

1. Assign each original polygon an ID number. If a polygon is subdivided later, all the smaller polygons generated share the same original ID number.
2. If there are intersecting polygons, subdivide them along the line of intersection.
3. Clip the polygons against all image boundaries and any user-specified clipping regions so that all resulting polygons lie either totally inside or totally outside the view frustum and clipping regions.
4. For each camera position, rendering the original polygons of the scene with Z-buffering using the polygon ID numbers as their colors.
5. For each frontfacing polygon, uniformly sample points and project them onto the image plane. Retrieve the polygon ID at each projected point from the color buffer. If the retrieved ID is different from the current polygon ID, the potentially occluding polygon is tested in object-space to determine whether it is an occluder or coplanar.
6. Clip each polygon with each of its occluders in object-space.
7. Associate with each polygon a list of photographs to which it is totally visible.

Using identification numbers to retrieve objects from the Z-buffer is similar to the item buffer technique introduced in [14]. The image-space steps in the algorithm can

quickly obtain the list of occluders for each polygon. Errors due to image-space sampling are largely avoided by checking the pixels in a neighborhood of each projection in addition to the pixels at the projected sample points.

Our technique also allows the user the flexibility to specify that only a particular region of an image be used in texture mapping. This is accomplished by specifying an additional clipping region in step 3 of the algorithm.

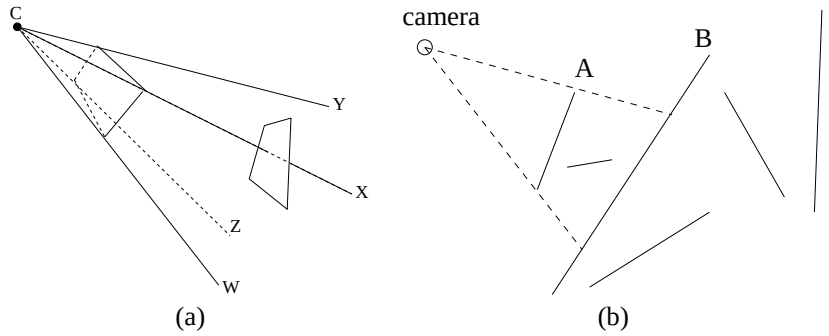


Fig. 2. (a) To clip a polygon against an occluder, we need to form a pyramid for the occluder with the apex at the camera position, and then clip the polygon with the bounding faces of the pyramid. (b) Our algorithm does *shallow clipping* in the sense that if polygon *A* occludes polygon *B*, we only use *A* to clip *B*, and any polygons behind *B* are unaffected.

The method of clipping a polygon against image boundaries is the same as that of clipping a polygon against an occluding polygon. In either case, we form a pyramid for the occluding polygon or image frame with the apex at the camera position (Fig. 2(a)), and then clip the polygon with the bounding faces of the pyramid. Our algorithm does *shallow clipping* in the sense that if polygon *A* occludes polygon *B*, we only use *A* to clip *B*, and any polygons behind *B* are unaffected(Fig. 2(b)). Only partially visible polygons are clipped; invisible ones are left intact. This greatly reduces the number of resulting polygons.

If a polygon P has a list of occluders $O = \{p_1, p_2, ..., p_m\}$, we use a recursive approach to do the clipping: First, we obtain the overlapping area on the image plane between each member of O and polygon P; we then choose the polygon p in O with maximum overlapping area to clip P into two parts P' and S where P' is the part of P that is occluded by p, and S is a set of convex polygons which make up the part of P not occluded by p. We recursively apply the algorithm on each member of S, first detecting its occluders and then performing the clipping.

To further reduce the number of resulting polygons, we set a lower threshold on the size of polygons. If the object-space area of a polygon is below the threshold, it is assigned a constant color based on the textures of its surrounding polygons. If a polygon is very small, it is not noticeable whether it is textured or simply a constant color. Fig. 3 shows visibility processing results for two geometric models.

6 Object-Space Hole Filling

No matter how many photographs we have, there may still be some polygons invisible to all cameras. Unless some sort of coloring is assigned to them, they will appear as undefined regions when visible in novel views.

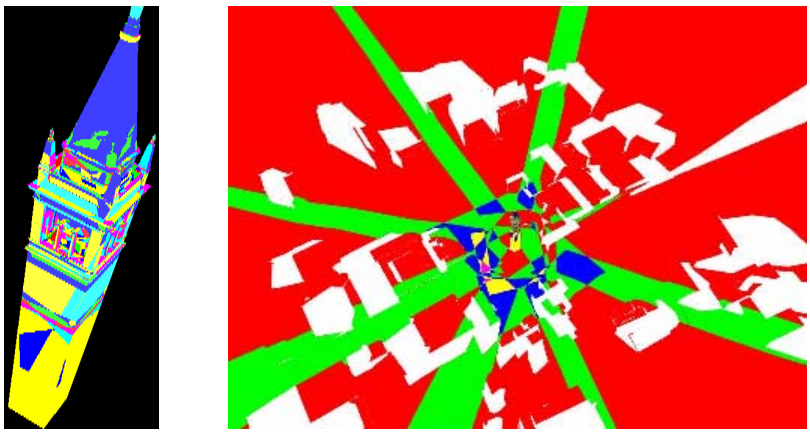


Fig. 3. Visibility results for a bell tower model with 24 camera positions and for the university campus model with 10 camera positions. The shade of each polygon encodes the number of camera positions from which it is visible; the white regions in the overhead view of the second image are "holes" invisible to all cameras.

Instead of relying on photographic data for these regions, we instead assign colors to them based on the appearance of their surrounding surfaces, a processed called *hole filling*. Previous hole-filling algorithms [16, 2, 7] have operated in image space, which can cause flickering in animations since the manner in which a hole is filled will not necessarily be consistent from frame to frame. Object-space hole-filling can guarantee that the derived appearance of each invisible polygon is consistent between viewpoints. By filling these regions with colors close to the colors of the surrounding visible polygons, the holes can be made difficult to notice.

Fig. 4. The image on the left exhibits black regions which were invisible to all the original cameras but not to the current viewpoint. The image on the right shows the rendering result with all the holes filled. See also Fig. 8.

The steps in hole filling are:

1. **Determine polygon connectivity**. At each shared vertex, set up a linked list for those polygons sharing that vertex. In this way, from a polygon, we can access all

its neighboring polygons.

2. **Determine colors of visible polygons**. Compute an "average" color for each visible polygon by projecting its centroid onto the image planes of each image in which it appears and sample the colors at those coordinates.
3. **Iteratively assign colors to the holes**. For each invisible polygon, if it has not yet been assigned a color, assign to each of its vertices the color of the closest polygon which is visible or that has been filled in a previous iteration.

The reason for the iterative step is that an invisible polygon may not have a visible polygon in its neighborhood. In this way its vertex colors can be determined after its neighboring invisible polygons are assigned colors.

Due to slight misalignments between the geometry and the original photographs, the textures of the edges of some objects may be projected onto the background. For example, a sliver of the edge of a building may project onto the ground nearby. In order to avoid filling the invisible areas with these incorrect textures, we do not sample polygon colors at regions directly adjacent to occlusion boundaries.

Fig. 4 shows the results of hole filling. The invisible polygons, filled will Gouraud-shaded low-frequency image content, are largely unnoticeable in animations. Because we assume that the holes will be relatively small and that the scene is mostly diffuse, we do not use the view-dependent information to render the holes.

7 Constructing and Querying Polygon View Maps

The goal of view-dependent texture-mapping is to always use surface appearance information sampled from the images which observed the scene closest in angle to the novel viewing angle. In this way, the errors in rendered appearance due to specular reflectance and incorrect model geometry will be minimized. Note that in any particular novel view, different visible surfaces may have different "best views"; an obvious case of this is when the novel view encompasses an area not entirely observed in any one view.

In order to avoid the perceptually distracting effect of surfaces suddenly switching between different best views from frame to frame, we wish to blend between the available views as the angle of view changes. This section shows how for each polygon we create a *view map* that encodes how to blend between at most three available views for any given novel viewpoint, with guaranteed smooth image weight transitions as the viewpoint changes. The view map for each polygon takes little storage and is simple to compute as a preprocessing step. A polygon's view map may be queried very efficiently: given a desired novel viewpoint, it quickly returns the set of images with which to texture-map the polygon and their relative weights.

To build a polygon's view map, we construct a local coordinate system for the polygon that represents the space of all viewing directions. We then regularly sample the set of viewing directions, and assign to each of these samples the closest original view in which the polygon is visible. The view maps are stored and used at rendering time to determine the three best original views and their blending factors by a quick look-up based on the current viewpoint.

The local polygon coordinate system is constructed as in Equation 1:

$$\mathbf{x} = \begin{cases} \mathbf{y^W} \times \mathbf{n} & \text{, if } \mathbf{y^W} \text{ and } \mathbf{n} \text{ are not collinear,} \\ \mathbf{x^W} & \text{otherwise} \end{cases}$$

$$\mathbf{y} = \mathbf{n} \times \mathbf{x} \tag{1}$$

where $\mathbf{x}^{\mathbf{W}}$ and $\mathbf{y}^{\mathbf{W}}$ are world coordinate system axes, and $\mathbf{n}$ is the triangle unit normal.

We transform viewing directions to the local coordinate system as in Fig. 5. We first obtain $\mathbf{v}$, the unit vector in the direction from the polygon centroid $\mathbf{c}$ to the original view position. We then rotate this vector into the $\mathbf{x} - \mathbf{y}$ plane of the local coordinate system for the polygon.

$$\mathbf{v}_r = (\mathbf{n} \times \mathbf{v}) \times \mathbf{n} \tag{2}$$

This vector is then scaled by the arc length $l = \cos^{-1}(\mathbf{n}^T\mathbf{v})$ and projected onto the $\mathbf{x}$ and $\mathbf{y}$ axes giving the desired view mapping.

$$\begin{aligned} x &= (l\mathbf{v}_r)^T\mathbf{x} \\ y &= (l\mathbf{v}_r)^T\mathbf{y} \end{aligned} \tag{3}$$

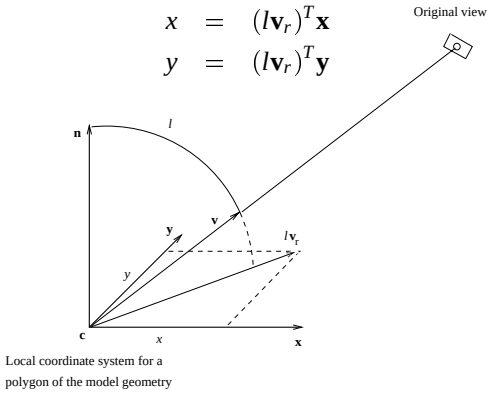


Fig. 5. The local polygon coordinate system for constructing view maps.

We pre-compute for each polygon of the model the mapping coordinates $\mathbf{p}_i = (x_i, y_i)$ for each original view i in which the polygon is visible. These points $\mathbf{p}_i$ represent a sparse sampling of view direction samples.

To extrapolate the sparse set of original viewpoints, we regularize the sampling of viewing directions as in Fig. 6. For every viewing direction on the grid, we assign to it the original view nearest to its location. This new regular configuration is what we store and use at rendering time. For the current virtual viewing direction we compute its mapping $\mathbf{p}_{virtual}$ in the local space of each polygon. Then based on this value we do a quick lookup into the regularly resampled view map. We find the grid triangle inside which $\mathbf{p}_{virtual}$ falls and use the original views associated with its vertices in the rendering ($\mathbf{p}_4$, $\mathbf{p}_5$, and $\mathbf{p}_7$ in the example of Fig. 6). The blending weights are computed as the barycentric coordinates of $\mathbf{p}_{virtual}$ in the triangle in which it lies. In this manner the weights of the various viewing images are guaranteed to vary smoothly as the viewpoint changes.

8 Efficient 3-pass View-Dependent Texture-Mapping

This section explains the implementation of the view-dependent texture-mapping rendering algorithm.

For each polygon visible in more than one original view we pre-compute and store the viewmaps described in Section 7. Before rendering begins, for each polygon we

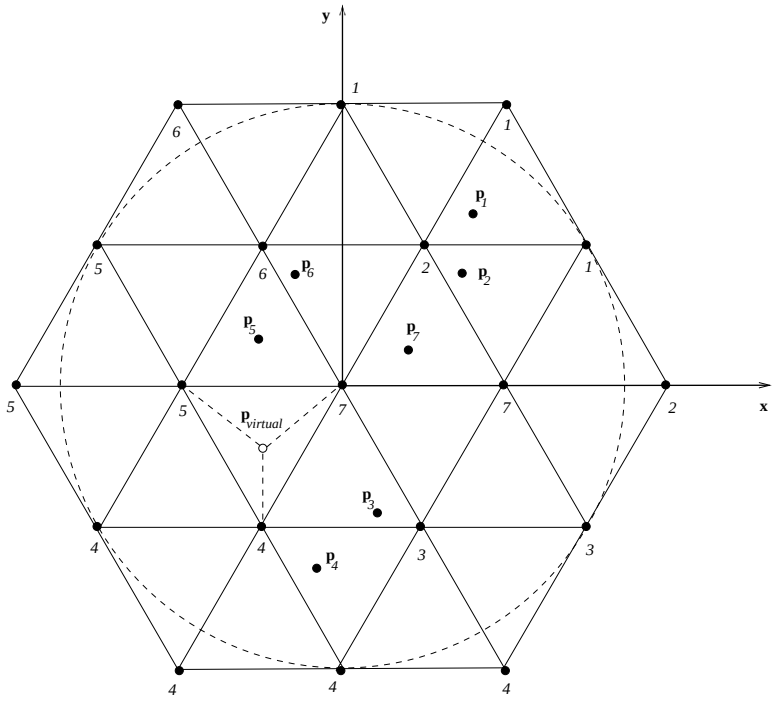


Fig. 6. A View Map. The space of viewing directions for each polygon is regularly sampled, and the closest original view is stored for each sample. To determine the weightings of original views to be used in a new view, the barycentric coordinates of the novel view within its containing triangle are used. This guarantees smooth changes of the set of three original views used for texture mapping when moving the virtual viewpoint. Here, for viewpoint $\mathbf{p}_{virtual}$, the polygon corresponding to this view map will be texture-mapped by an almost evenly weighted combination of original views $\mathbf{p}_4$, $\mathbf{p}_5$, and $\mathbf{p}_7$, since those are the views assigned to the vertices of $\mathbf{p}_{virtual}$'s view map triangle.

find the coordinate mapping of the current viewpoint $\mathbf{p}_{virtual}$ and do a quick lookup to determine which triangle of the grid it lies within. As explained in Section 7 this returns the three best original views and their relative weights $\alpha_1, \alpha_2, \alpha_3$.

Since each VDTM polygon must be rendered with three texture maps, the rendering is performed in three passes. Texture mapping is enabled in modulate mode, where the new pixel color C is obtained by multiplying the existing pixel color C_f and the texture color C_t. The Z-buffer test is set to *less than or equal* (`GL_LEQUAL`) instead of the default *less than* (`GL_LESS`) to allow a polygon to blend with itself as it is drawn multiple times with different textures. The first pass proceeds by selecting an image camera, binding the corresponding texture, loading the corresponding texture matrix transformation $\mathbf{M}_{texture}$ in the texture matrix stack, and rendering the part of the model geometry for which the first best camera is the selected one with modulation color $(\alpha_1, \alpha_1, \alpha_1)$. These steps are repeated for all image cameras. The results of this pass can seen on the tower in Fig. 8 (b). The first pass fills the depth buffer with correct depth values for the entire view. Before proceeding with the second pass we enable blending in the frame buffer, i.e. instead of replacing the existing pixel values with incoming values, we add those values

together. The second pass then selects cameras and renders polygons for which the second best camera is the selected one with modulation color $(\alpha_2, \alpha_2, \alpha_2)$. The results of the second pass can seen on the tower in Fig. 8 (c). The third pass proceeds similarly, rendering polygons for which the third best camera is the currently selected one with modulation color $(\alpha_3, \alpha_3, \alpha_3)$. The results of this last pass can seen on the tower in Fig. 8 (d). Polygons visible from only one original viewpoint are compiled in separate list and rendered during the first pass with modulation color $(1.0, 1.0, 1.0)$.

The polygons that are not visible in any image cameras are compiled in a separate OpenGL display list and their vertex colors are specified according to the results of the hole-filling algorithm. Those polygons are rendered before the first pass with Gouraud shading after the texture mapping is disabled.

The block diagram in Fig. 7 summarizes the display loop steps.

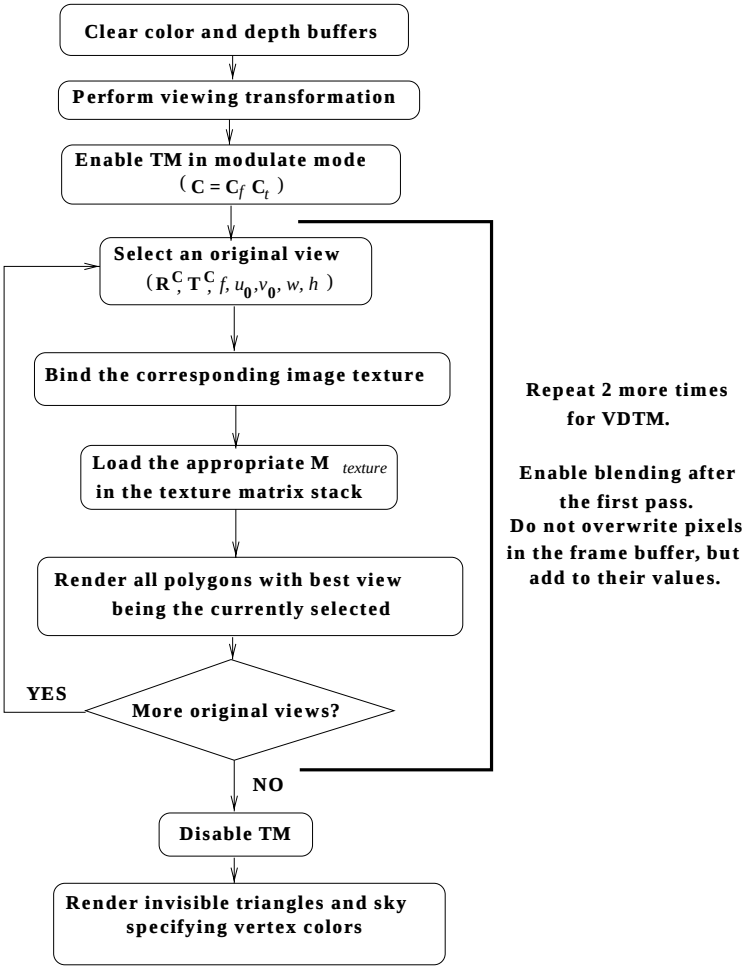


Fig. 7. The multi-pass view-dependent projective texture mapping rendering loop.

9 Discussion and Future Work

The presented method was effective at realistically rendering a relatively large-scale image-based scene at interactive rates on standard graphics hardware. Using relatively unoptimized code, we were able to achieve 20 frames per second on a Silicon Graphics InfiniteReality for the full tower and campus models. Nonetheless, many aspects of this work should be regarded as preliminary in nature. One problem with the technique is that it ignores the spatial resolution of the original images in its selection process – an image that shows a particular surface at very low resolution but at just the right angle would be given greater weighting than a high-resolution image from a slightly different angle. Having the algorithm blend between the images using a multiresolution image pyramid would allow low-resolution images to influence only the low-frequency content of the renderings. However, it is less clear how this could be implemented using standard graphics hardware.

While the algorithm guarantees smooth texture weight transitions as the viewpoint moves, it does not guarantee that the weights will transition smoothly across surfaces of the scene. As a result, seams can appear in the renderings where neighboring polygons are rendered with very different combinations of images. The problem is most likely to be noticeable near the frame boundaries of the original images, or near a shadow boundary of an image, where polygons lying on one side of the boundary include an image in their view maps but the polygons on the other side do not. [2] and [9] suggest feathering the influence of images in image-space toward their boundaries and near shadow boundaries to reduce the appearance of such seams; with some consideration this technique should be adaptable to the object-space method presented here.

The algorithm as we have presented it requires all the available images of the scene to fit within the main memory of the rendering computer. For a very large-scale environment, this is unreasonable to expect. To solve this problem, spatial partitioning schemes [13], image caching [11], and impostor manipulation [11, 12] techniques could be adapted to the current framework.

As we have presented the algorithm, it is only appropriate for models that can be broken into polygonal patches. The algorithm can also work for curved surfaces (such as those acquired by laser scanning); these surfaces would be need to be broken down by the visibility algorithm until they are seen without self-occlusion by their set of cameras.

Lastly, it seems as if it would be more efficient to analyze the set of available views of each polygon and distill a unified view-dependent function of its appearance, rather than the raw set of original views. One such representation is the Bidirectional Texture Function, presented in [1], or a yet-to-be-presented form of geometry-enhanced light field. Such a technique will require new rendering methods in order to render the distilled representations in real time. Lastly, extensions of techniques such as model-based stereo [2] might be able to perform a better job of interpolating between the various views than linear interpolation.

Images and Animations

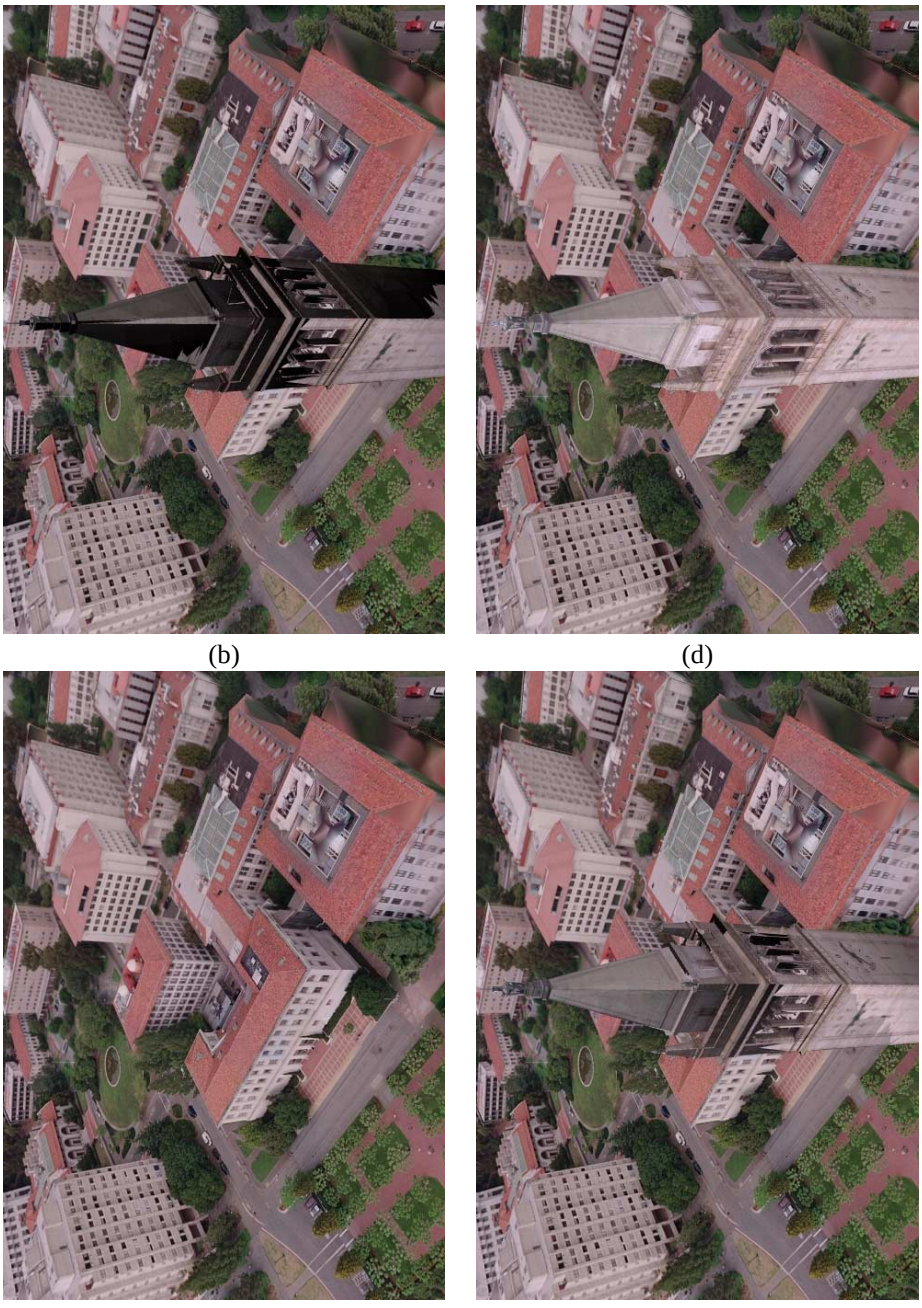
Images and Animations of the Berkeley campus model may be found at:
`http://www.cs.berkeley.edu/~debevec/Campanile`

Acknowledgments

The authors wish to thank Jason Luros, Vivian Jiang, Chris Wright, Sami Khoury, Charles Benton, Tim Hawkins, Charles Ying, Jitendra Malik, and Camillo Taylor for their contributions to the Berkeley Campus Animation. This research was supported by Silicon Graphics and a Multidisciplinary University Research Initiative on 3D Direct visualization from ONR and BMDO, grant FDN00014-96-1-1200.

References

1. DANA, K. J., GINNEKEN, B., NAYAR, S. K., AND KOENDERINK, J. J. Reflectance and texture of real-world surfaces. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.* (1997), pp. 151–157.
2. DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96* (August 1996), pp. 11–20.
3. FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. *Computer Graphics: principles and practice.* Addison-Wesley, Reading, Massachusetts, 1990.
4. GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. The Lumigraph. In *SIGGRAPH '96* (1996), pp. 43–54.
5. LAVEAU, S., AND FAUGERAS, O. 3-D scene representation as a collection of images. In *Proceedings of 12th International Conference on Pattern Recognition* (1994), vol. 1, pp. 689–691.
6. LEVOY, M., AND HANRAHAN, P. Light field rendering. In *SIGGRAPH '96* (1996), pp. 31–42.
7. MARK, W. R., MCMILLAN, L., AND BISHOP, G. Post-rendering 3D warping. In *Proceedings of the Symposium on Interactive 3D Graphics* (New York, Apr.27–30 1997), ACM Press, pp. 7–16.
8. MCMILLAN, L., AND BISHOP, G. Plenoptic Modeling: An image-based rendering system. In *SIGGRAPH '95* (1995).
9. PULLI, K., COHEN, M., DUCHAMP, T., HOPPE, H., SHAPIRO, L., , AND STUETZLE, W. View-based rendering: Visualizing real objects from scanned range and color data. In *Proceedings of 8th Eurographics Workshop on Rendering, St. Etienne, France* (June 1997), pp. 23–34.
10. SEGAL, M., KOROBKIN, C., VAN WIDENFELT, R., FORAN, J., AND HAEBERLI, P. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH '92* (July 1992), pp. 249–252.
11. SHADE, J., LISCHINSKI, D., SALESIN, D., DEROSE, T., AND SNYDER, J. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings* (1996), H. Rushmeier, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 75–82.
12. SILLION, F., DRETTAKIS, G., AND BODELET, B. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum (Proc. Eurographics 97) 16*, 3 (Sept. 4–8 1997), C207–C218.
13. TELLER, S. J., AND SEQUIN, C. H. Visibility preprocessing for interactive walkthroughs. In *SIGGRAPH '91* (1991), pp. 61–69.
14. WEGHORST, H., HOOPER, G., AND GREENBERG, D. P. Improved computational methods for ray tracing. *ACM Transactions on Graphics 3*, 1 (January 1984), 52–69.
15. WEILER, K., AND ATHERTON, P. Hidden surface removal using polygon area sorting. In *SIGGRAPH '77* (1977), pp. 214–222.
16. WILLIAMS, L., AND CHEN, E. View interpolation for image synthesis. In *SIGGRAPH '93* (1993).



(b) (d)

(a) (c)

Fig. 8. The different view-dependent projective texture-mapping passes in producing a frame of the Berkeley campus virtual fly-by. The complete model contains approximately 100,000 triangles. **(a)** The campus buildings and terrain after hole-filling; these areas were seen from only one viewpoint and are thus rendered before the VDTM passes. **(b)** The Berkeley tower after the first pass of view-dependent texture mapping. **(c)** The Berkeley tower after the second pass of view-dependent texture mapping. **(d)** The complete rendering after all three VDTM passes.

Image-Based Lighting

A Photometric Approach to Rendering and Compositing

Paul Debevec

Computer Science Division
University of California at Berkeley

August 1999

http://www.cs.berkeley.edu/~debevec

Reflection Mapping - 1982

Mike Chou and Lance Williams

Gene Miller and Ken Perlin

http://www.CS.Berkeley.EDU/~debevec/ReflectionMapping/

Motivations

Image-Based Modeling and Rendering

- IBMR allows us to model and render real scenes
- We want to add new objects
  - buildings, furniture, people
- We want the lighting to be correct

CGI / Background Plate Compositing

- Want to add synthetic actors, creatures, props to film and video
- Must be photorealistic
- Current techniques are challenging

CGI / Background Plate Compositing

Need to match:

- Camera Parameters
  - Pose, Focal length, Distortion, Focus
- Film Response
  - Contrast, Toe & Shoulder, Color Balance
- MTF / Film Grain
  - Modulation Transfer Function, Ag Particles
- Illumination
  - Highlights, Reflections, and Shadows

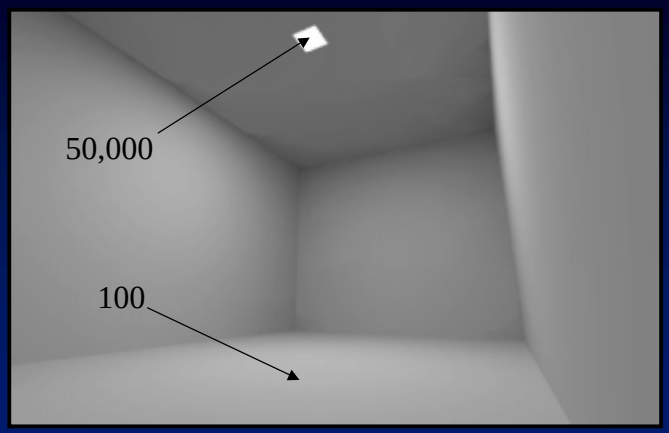
Photometric Approach:

Illuminate Synthetic Objects with Measurements of Real Light

How can we measure the light?

Light is difficult to measure

Concentrated Light Sources => High Dynamic Range



10’ x 15’ x 9’ room, 9” by 9” light, 50% reflective walls

Mirrored Ball - Records light in all directions

Brightest regions are saturated

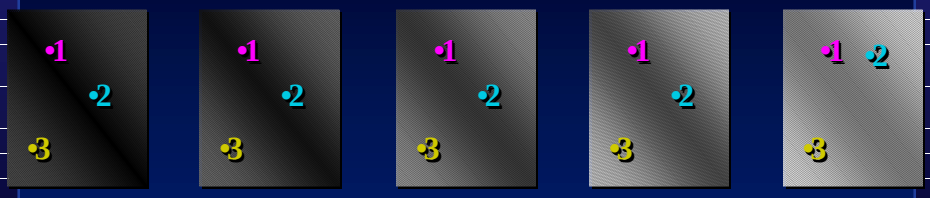
⬇

Intensity and color information lost

High Dynamic Range Photography

Debevec and Malik, SIGGRAPH 97

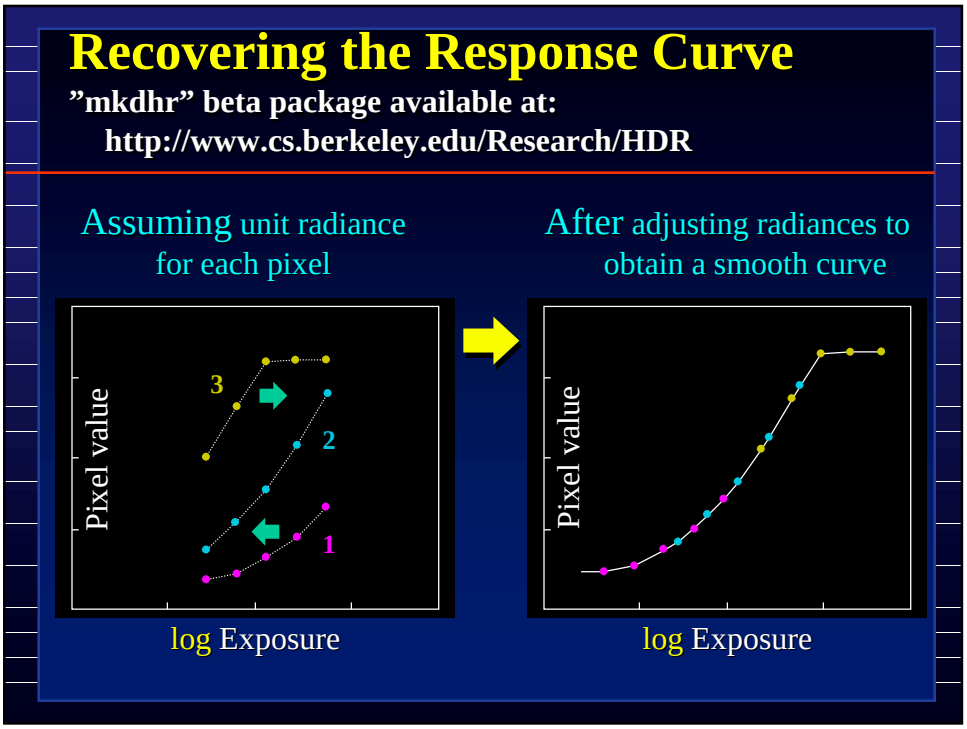
Image series

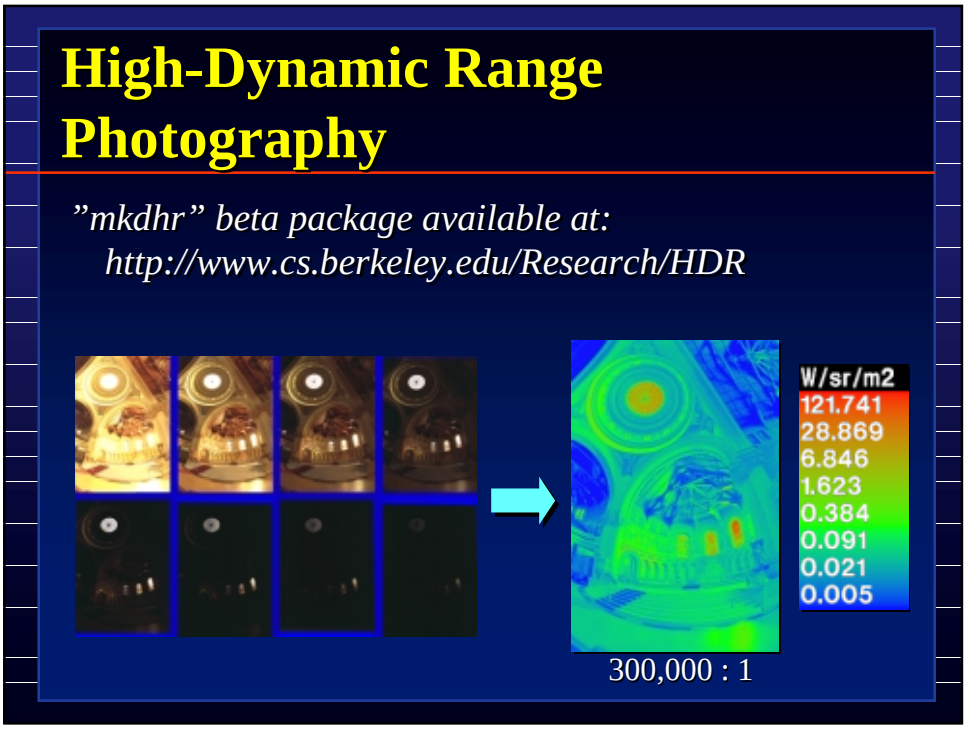


Δt = 1/64 sec	Δt = 1/16 sec	Δt = 1/4 sec	Δt = 1 sec	Δt = 4 sec

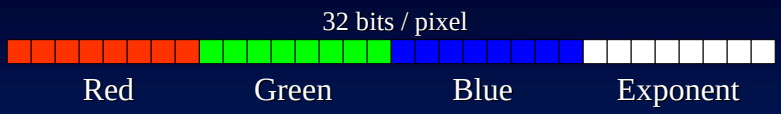
Exposure = Radiance × Δt

log Exposure = log Radiance + log Δt


Recovering the Response Curve
"mkdhr" beta package available at:
http://www.cs.berkeley.edu/Research/HDR
Assuming unit radiance for each pixel
After adjusting radiances to obtain a smooth curve
Pixel value
3
2
1
log Exposure
Pixel value
log Exposure


High-Dynamic Range Photography
"mkdhr" beta package available at:
http://www.cs.berkeley.edu/Research/HDR
W/sr/m2
121.741
28.869
6.846
1.623
0.384
0.091
0.021
0.005
300,000 : 1

Representing High Dynamic Range Radiance Images



(145, 215, 87, 149) =

(145, 215, 87) * 2^(149-128) =

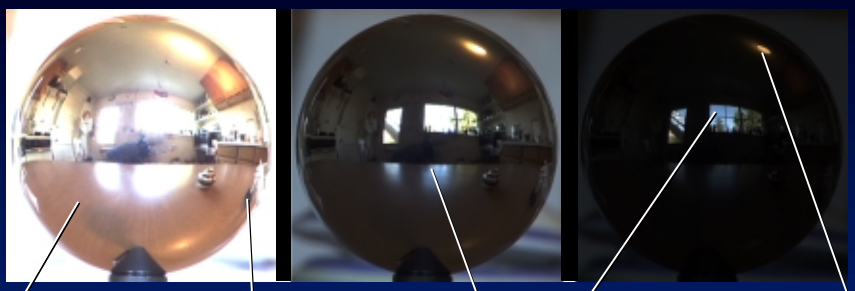
(1190000, 1760000, 713000) .

(145, 215, 87, 103) =

(145, 215, 87) * 2^(103-128) =

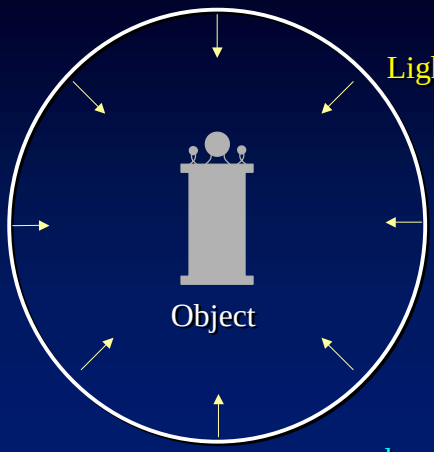
(0.00000432, 0.00000641, 0.00000259)

Radiance Map from the Mirrored Ball



Assembled from ten digital images,
Δ t = 1/4 to 1/10000 sec

Illuminating Objects using Measurements of Real Light

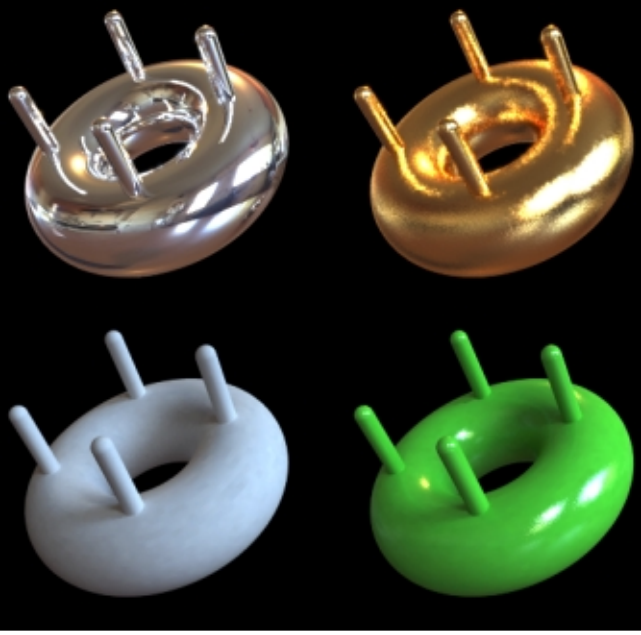


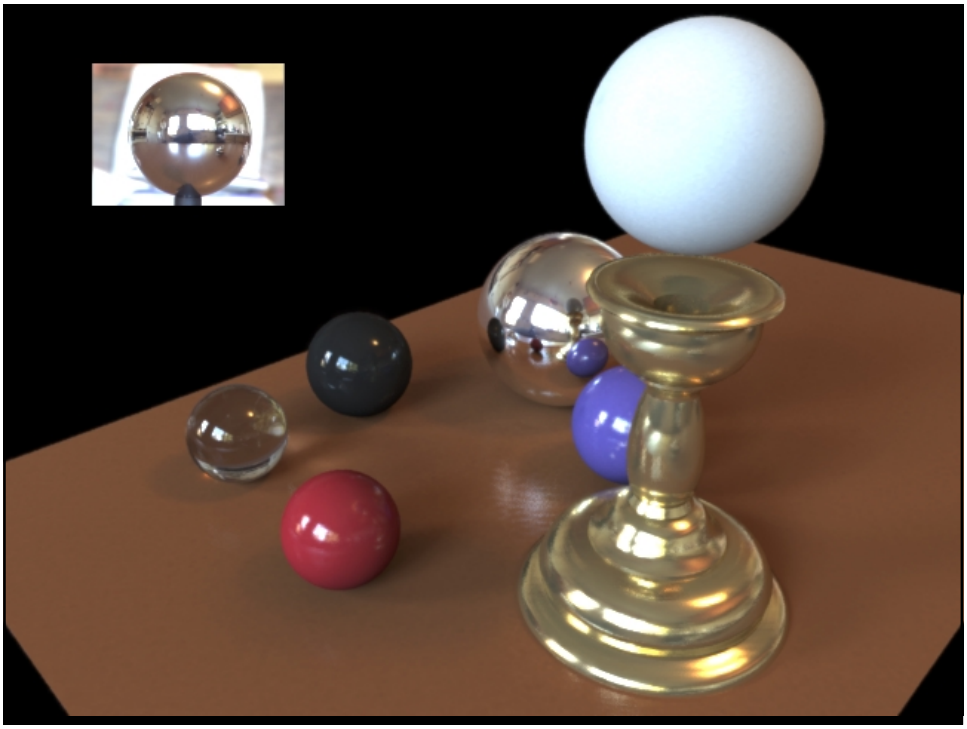
Environment assigned "glow" material property in Greg Larson's RADIANCE system.

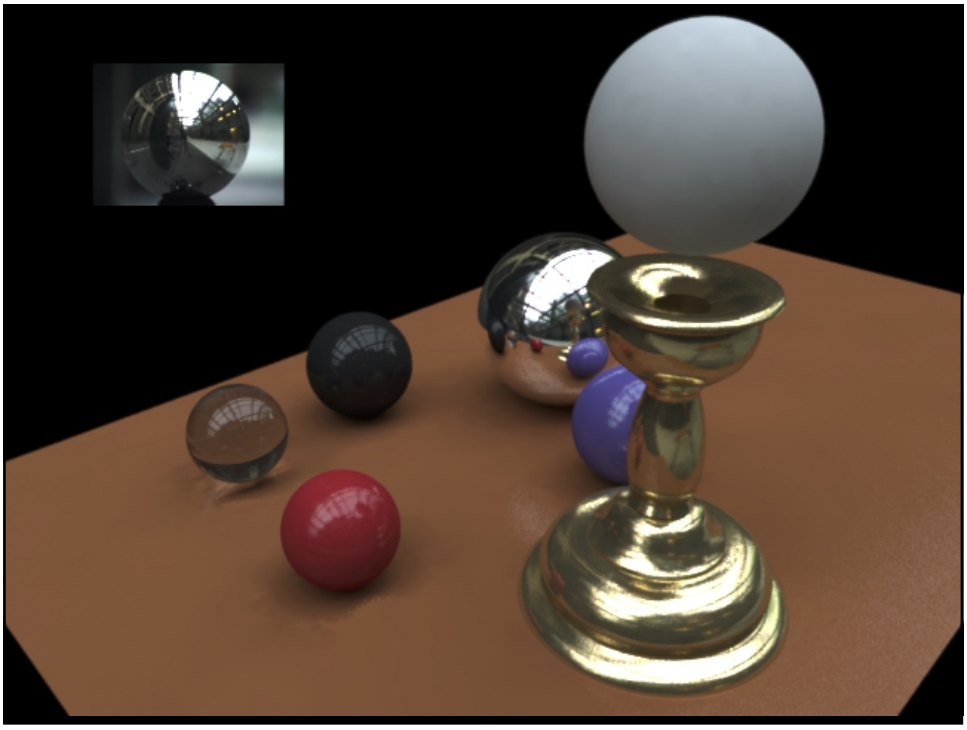
http://radsite.lbl.gov/radiance/

See also: Larson and Shakespeare, "Rendering with Radiance", 1998

Illumination Results







Early Tests of HDRI and Image-Based Lighting in LightWave 3D

Lighting Environments from the Light Probe Image Gallery
http://www.cs.berkeley.edu/~debevec/Probes/
www.debevec.org

Video

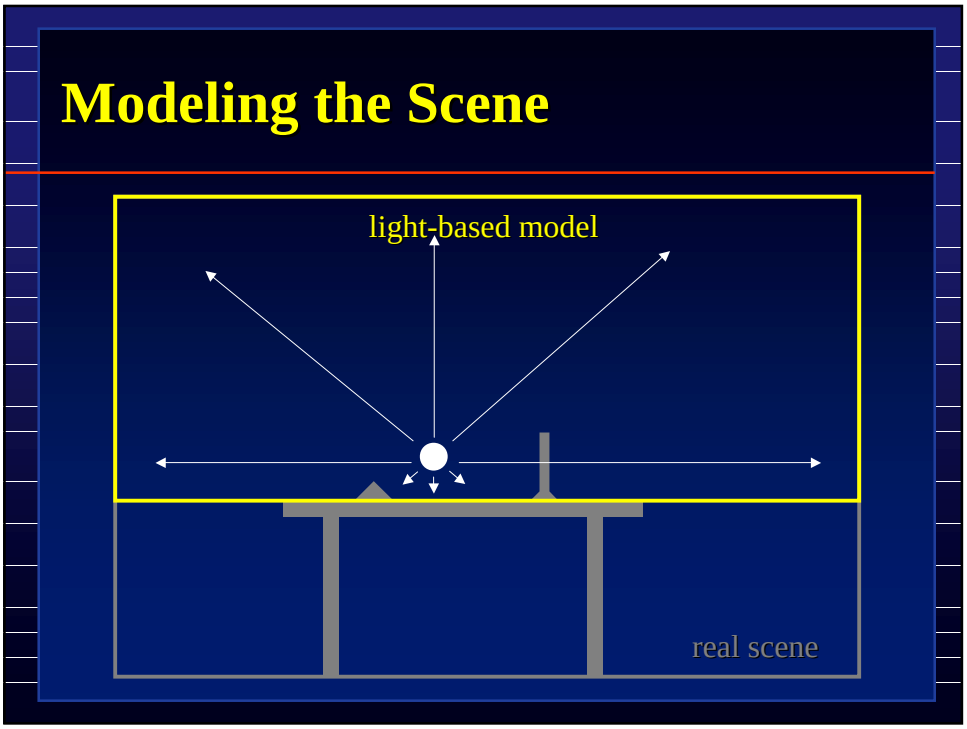
We can now illuminate synthetic objects with real light.

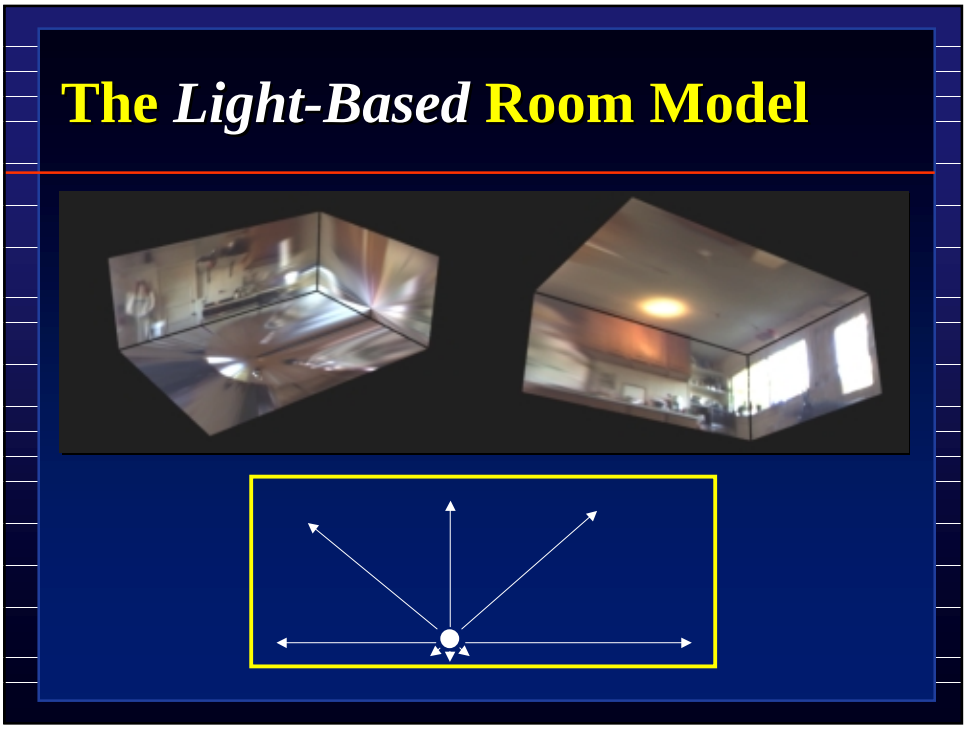
How do we add synthetic objects to a real scene?

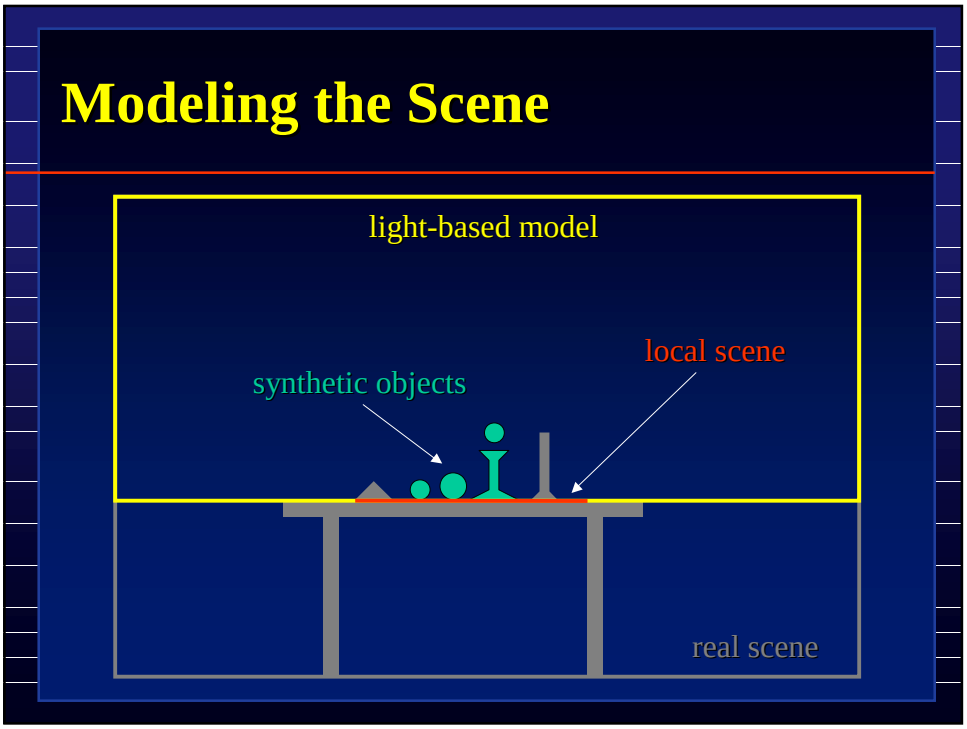
Real Scene Example

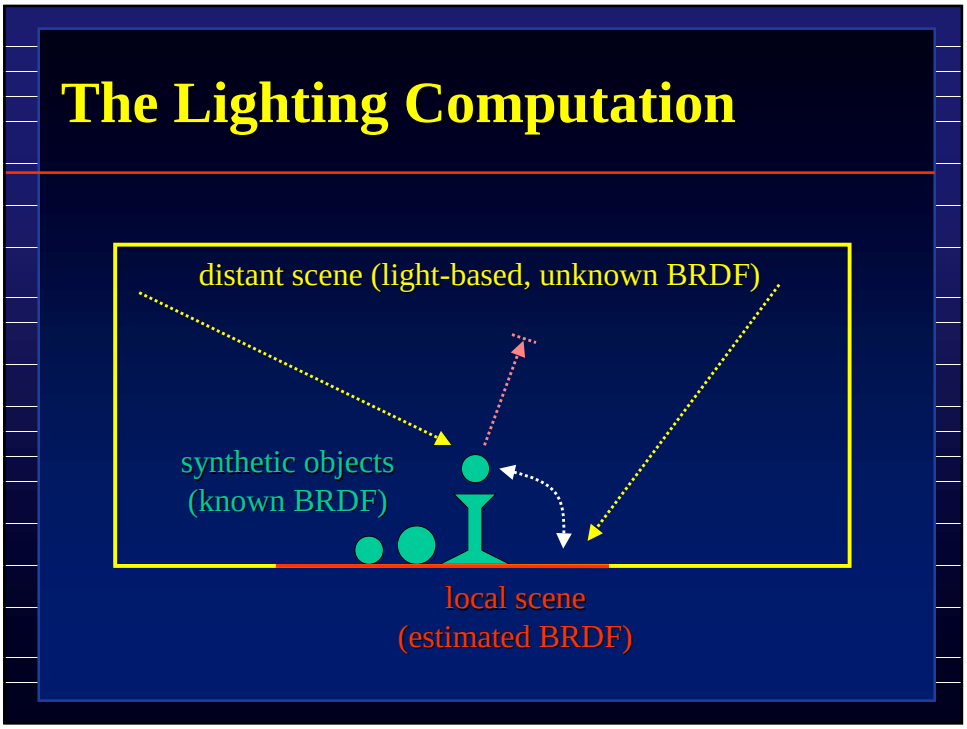
Goal: place synthetic objects on table

Light Probe / Calibration Grid


Modeling the Scene
light-based model
real scene


The Light-Based Room Model


Modeling the Scene
light-based model
local scene
synthetic objects
real scene


The Lighting Computation
distant scene (light-based, unknown BRDF)
synthetic objects
(known BRDF)
local scene
(estimated BRDF)

Rendering into the Scene

Background Plate

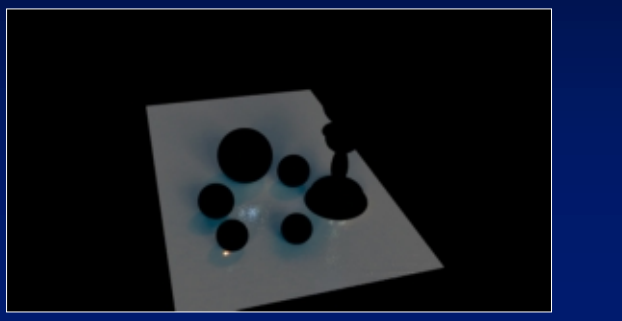
Rendering into the Scene

Objects and Local Scene matched to Scene

Differential Rendering

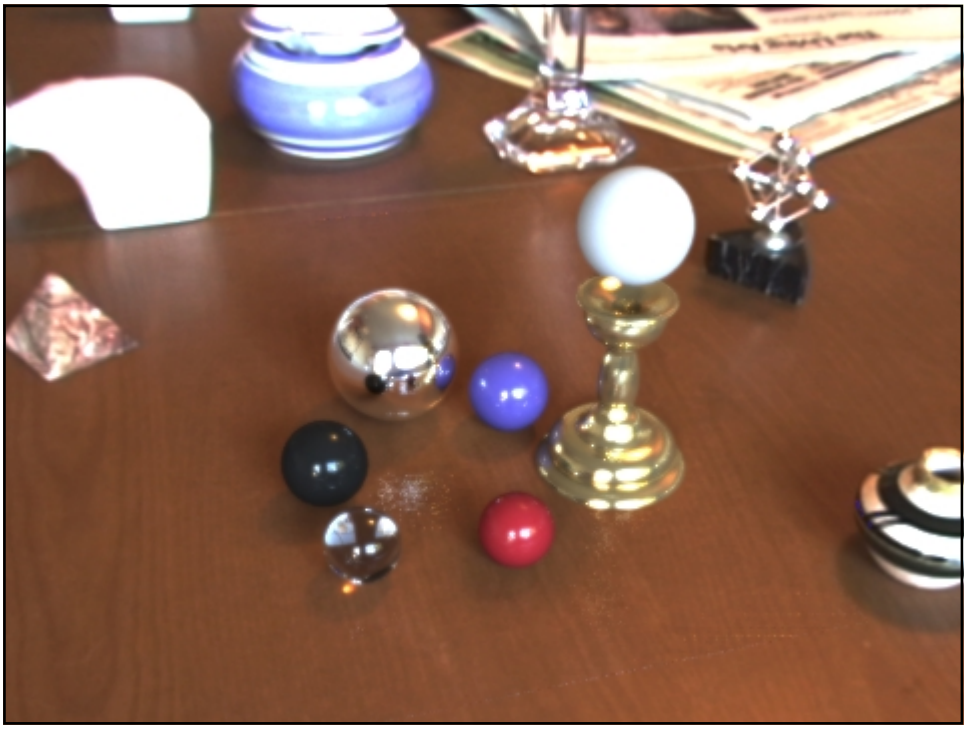
Local scene w/o objects, illuminated by model

Differential Rendering (2)
Difference in local scene



Differential Rendering (3)

Final Result



Video

Domino animation rendered by Son Chang and Christine Waggoner

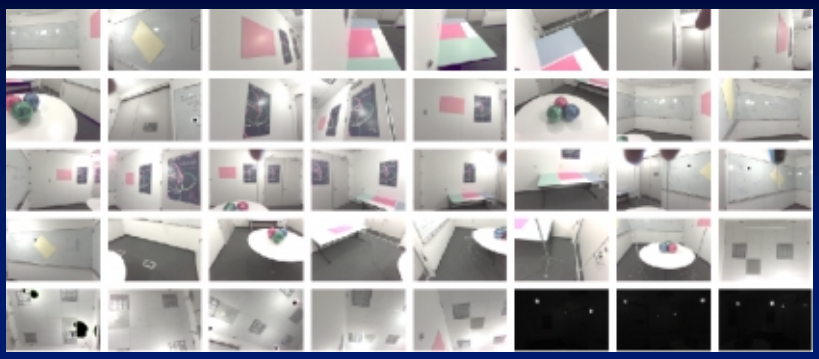
Estimating the local scene material properties

- Necessary for correct *shadows* and *reflections*
- For each part of the local scene, we know its *irradiance* from the light-based model
- If the material is *diffuse*, its albedo is its *radiance* divided by its *irradiance*
- *Non-diffuse* properties can be estimated by *iterative methods* or specified by hand
- See: Ward92, Karner20, Dana97, Sato97, Yu98, Debevec98, Yu99

Reflectance Properties for a Whole Scene: Inverse Global Illumination

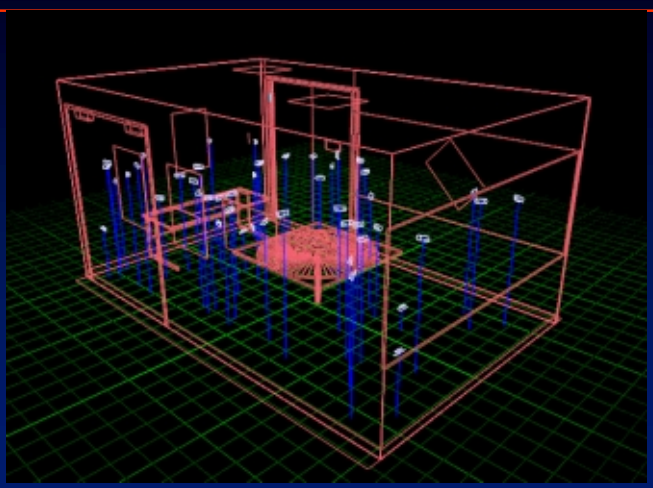
Yizhou Yu, Paul Debevec, Jitendra Malik, Tim Hawkins

SIGGRAPH 99, Thursday, 11:50-12:15pm, West Hall A



40 radiance maps of a room

Recovered Geometry and Viewpoints



Real/Synthetic Comparison

Same viewpoints, Same lighting, Same objects

Real/Synthetic Comparison

New viewpoint, New lighting, New object

Communicating the sense of Brightness

- **Fade In / Fade Out**
  - Bright areas appear first / fade last
- **Motion Blur**
  - Bright areas leave streaks
- **Blur / Glare / Soft Focus**
  - Bright areas blossom
- **Radial Light Falloff (Vignetting)**
  - Bright areas sear through corners
- **Color tinting**
  - Bright areas still ramp to white

RNL Example

Renderer Output

RNL Example

Defocus & Glare Added

RNL Example

Soft Focus Added

RNL Example

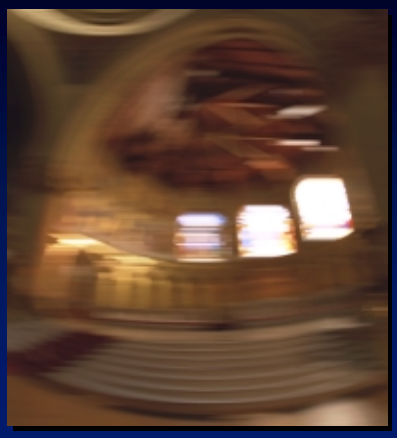
Light Falloff (Vignetting) Added

Motion Blur

Normal digitized photo

Synthetic blur added

Motion Blur



Blurred radiance map, virtually rephotographed

Actual blurred photograph

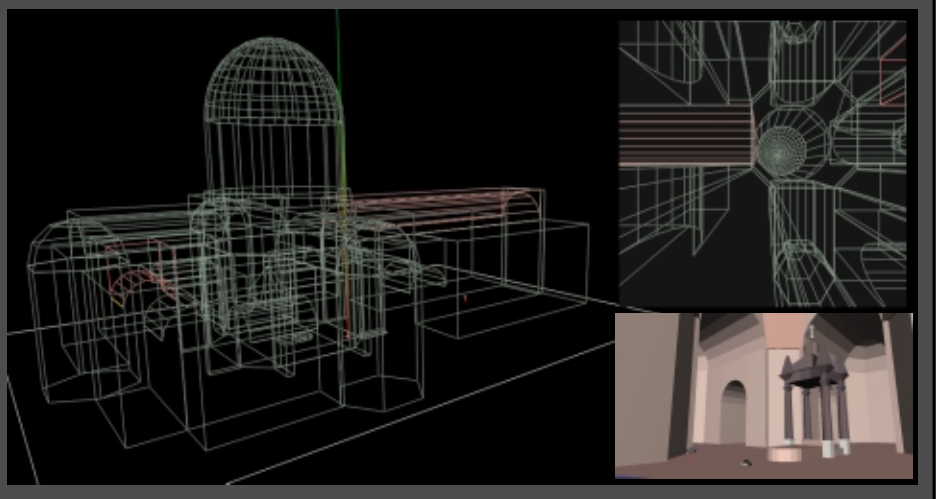
Color Tinting Example

Interior Illumination Model
St. Peter's Basilica for "Fiat Lux"

SIGGRAPH 99 ET

Christine Cheng, H.P. Duiker, Tal Garfinkel, Tim Hawkins, Jenny Huang, Westley Sarokin, Paul Debevec

http://fiatlux.berkeley.edu/



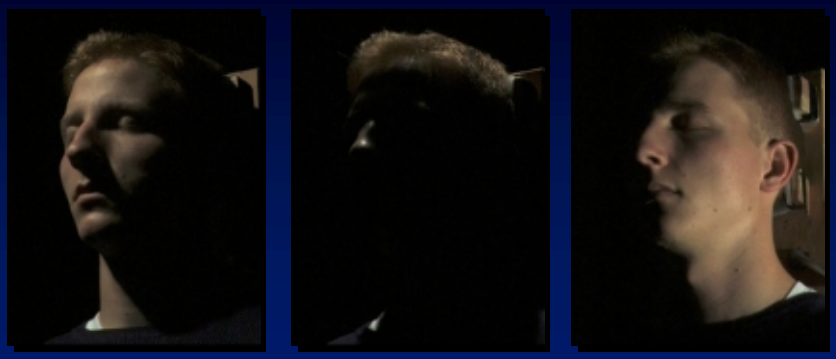
Interior of St. Peter's from one Viewpoint


http://fiatlux.berkeley.edu/


The Light Stage
See: “Acquiring the Reflectance Field of a Human Face” - Debevec, Hawkins, Tchou, Duiker, Sarokin, and Sagar - SIGGRAPH 2000

Light Stage

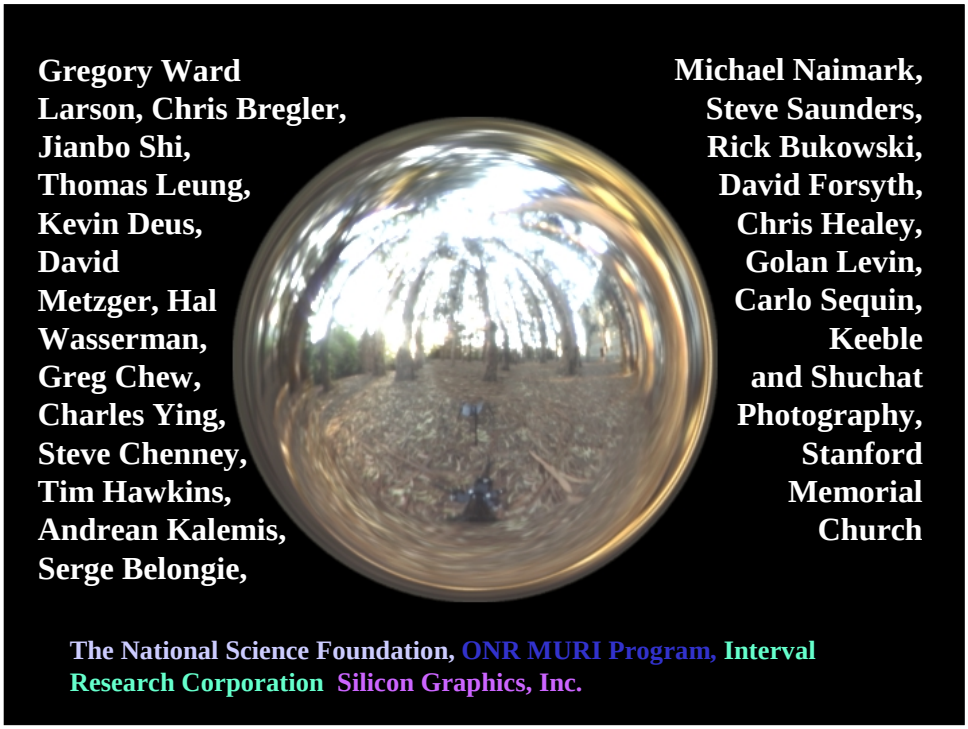
Original Data



Point Source Illumination

Light Stage Results

Lighting Environments from the Light Probe Image Gallery
http://www.cs.berkeley.edu/~debevec/Probes/
www.debevec.org


Gregory Ward Larson, Chris Bregler, Jianbo Shi, Thomas Leung, Kevin Deus, David Metzger, Hal Wasserman, Greg Chew, Charles Ying, Steve Chenney, Tim Hawkins, Andrean Kalemis, Serge Belongie,
Michael Naimark, Steve Saunders, Rick Bukowski, David Forsyth, Chris Healey, Golan Levin, Carlo Sequin, Keeble and Shuchat Photography, Stanford Memorial Church
The National Science Foundation, ONR MURI Program, Interval Research Corporation Silicon Graphics, Inc.

The Story of Reflection Mapping

(Title inspired by Frank Foster's " The Story of Computer Graphics")

The Quest Begins

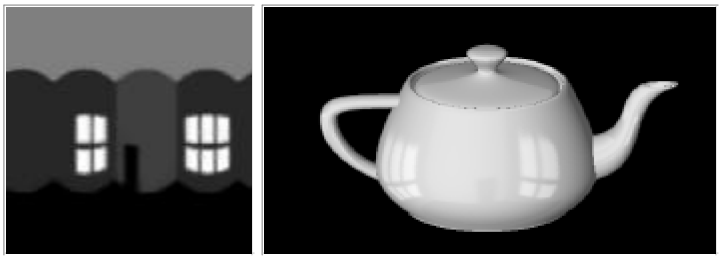
Some of the recent graphics research I've been working on builds on the techniques of reflection mapping and environment mapping developed in the late 70's and early 80's. I had a paper about the work at SIGGRAPH 98 ("Rendering Synthetic Objects into Real Scenes") which appears later in these notes.

Blinn and Newell 1976

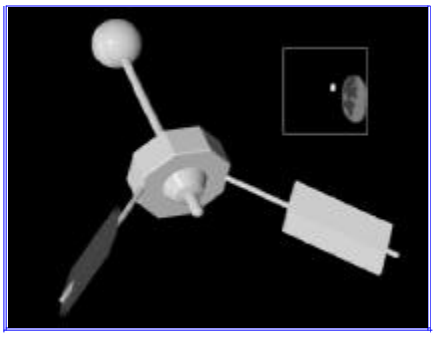
In the paper I referenced reflection mapping using synthetically rendered environment maps as presented by Jim Blinn in 1976:

- Blinn, J. F. and Newell, M. E. Texture and reflection in computer generated images. Communications of the ACM Vol. 19, No. 10 (October 1976), 542-547.

I met with Jim Blinn in June 1999 during a visit to Microsoft Research, and by coincidence he was just in the process of resurrecting some old files, including the images from this paper. The first environment-mapped object, quite appropriately, was the Utah Teapot, with a room image made with a paint program (which Blinn wrote) as the environment map:



In the paper, Blinn also included an image of a satellite, environment-mapped with an image of the earth and the sun which he drew, shown below. Note that in both cases the objects are also being illuminated by a traditional light source to create their diffuse appearance.



More images from Blinn's early environment mapping work may be found here.

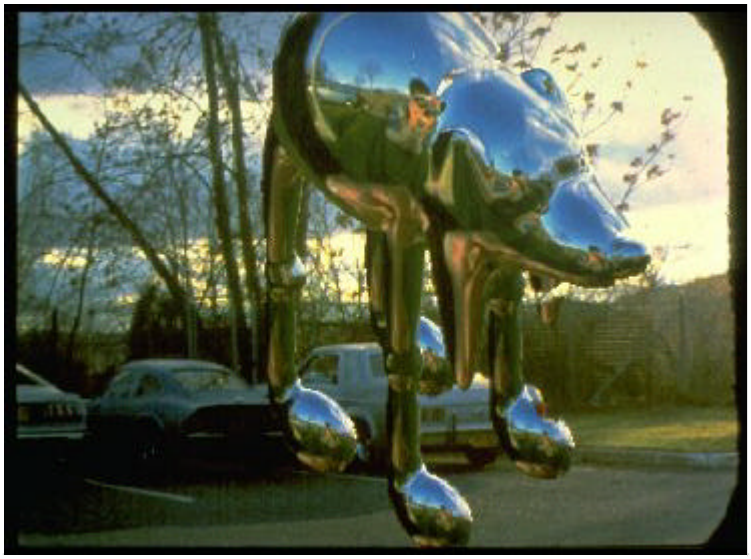
What about Photographs?

I was surprised in writing the paper that there didn't seem to be a good reference for using real omnidirectional *photographs* as reflection maps. The seemed odd, since the technique is in common usage in the computer graphics industry, and was used in creating some of the more memorable movie effects in the 80's and 90's (e.g. the spaceship in *Flight of the Navigator* (1986), and the metal man in *Terminator II* (1991)). Furthermore, it can be regarded as one of the earliest forms of image-based rendering. So I've tried to go about figuring out where the technique came from.

The First Renderings

While at SIGGRAPH 98 in Orlando, I talked to Paul Heckbert, Ned Greene, Michael Chou, Lance Williams, and Ken Perlin to try to find out the origin of the technique. The story that took shape was that the technique was developed independently by Gene Miller working with Ken Perlin, and also by Michael Chou working with Lance Williams, around 1982 or 1983. I heard that the first two images in which reflection mapping was used to place objects into scenes were of a synthetic shiny robot standing next to Michael Chou in a garden, and of a reflective blobby dog floating over a parking lot.

A few months later, with the help of Gene Miller, Lance Williams, and Paul Heckbert, I was able to see both of these images side-by side:



A reflection-mapped blobby dog floating in the MAGI parking lot. (Courtesy of Gene Miller)

A reflection-mapped robot standing next to Chou (In hi-res courtesy of Lance Williams)

At NYIT, it was Michael Chou who carried out the very first experiments on using images as reflection maps. For obtaining the reflection image, Chou used a ten-inch "Gazing Ball", which is a shiny glass sphere with a metallic coating on the inside, usually sold as a lawn ornament. Gene Miller used a three-inch Christmas tree ornament, which was held in place by Christine Chang while he took a 35mm photograph of it:

In January 1999, Gene Miller sent over a wealth of information and images about his knowledge of the origin of the technique. Click here to continue on to Gene Miller's stories and images about the development of reflection mapping.

Williams 1983

The Chou and robot image appeared in Lance Williams's 1983 SIGGRAPH paper "Pyramidal

Parametrics". The paper introduced MIP-mapping, an elegant pre-filtering scheme for avoiding aliasing in texture-mapping algorithms. MIP-mapping has since been implemented on scores of graphics architectures and is used everywhere from video games to PC graphics to high-end flight simulators. The reflection-mapped robot image was just one example used to demonstrate the technique.

- Williams, Lance, "Pyramidal Parametrics," Computer Graphics (SIGGRAPH), vol. 17, No. 3, Jul. 1983 pp. 1-11.

Miller and Hoffman 1984

In talking to Gene Miller, I learned that in December 1982 he and Bob Hoffman submitted a paper on the technique to SIGGRAPH 83 but it was not accepted for publication. However, a revised version of this work appeared in the SIGGRAPH 84 course notes on advanced computer graphics animation:

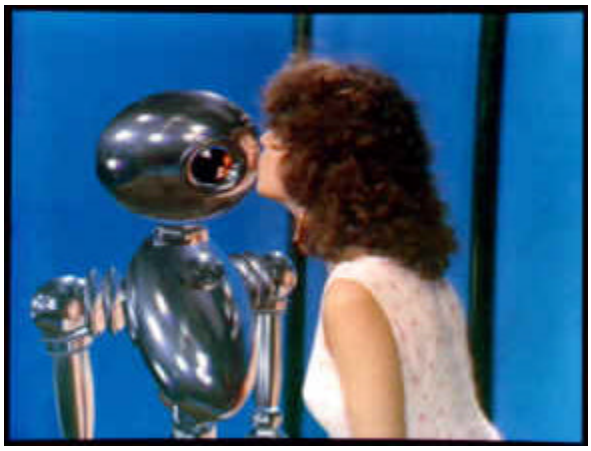
- Gene S. Miller and C. Robert Hoffman. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments, Course Notes for Advanced Computer Graphics Animation, SIGGRAPH 84.

Thanks to Gene Miller, these notes can be viewed in HTML and PDF formats.

Noteworthy is that the notes suggest that reflection maps can be used to render diffuse as well as specular objects, and that issues arising from the limited dynamic range of film could be addressed by combining a series of photographs taken with different exposure levels. Both issues were problems I worked to address in my SIGGRAPH 98 paper.

Interface - 1985

In 1985, Lance Williams was part of a team at the New York Institute of Technology that used reflection mapping in a moving scene with an animated CG element. The piece "Interface" featured a young woman kissing a shiny robot. In reality, she was filmed kissing a 10-inch shiny ball, and the reflection map was taken from the reflection of the ball. To make the animation, the reflection map was applied to the robot, and the robot was composited into the scene to replace the ball.



	QuickTime - "Video" Compressor, 160 by 120, 15fps, 2.4MB
	QuickTime - "Sorenson" Compressor, 240 by 160, 15fps, 2.6MB

"Interface", courtesy of Lance Williams.

Interface is the first use of photo-based reflection mapping in an animation, and also its first use to help tell a story. The woman quickly kisses the robot and then heads out for the evening. As the silent robot waves goodbye, her reflected image recedes, and you can't help but think that he might have wanted to go along with her.

Interface was also worked on by Carter Burwell and Ned Greene, and the actress was Ginevra Walker. Carter Burwell later composed music for feature films such as *Raising Arizona*, *Miller's Crossing*, *The Hudsucker Proxy*, and *Barton Fink*.

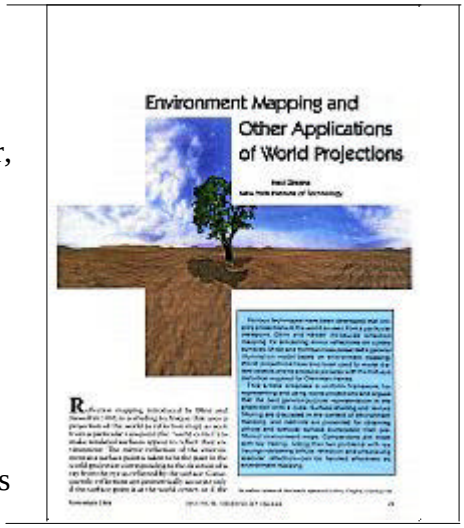
Lance Williams shortly thereafter added reflection mapping (as well as texture, bump, and transparency mapping) to Pacific Data Images' renderer, which was used to create Jose Dias' "Globo" reflection mapping images.

Flight of the Navigator - 1986

The first feature film to use the technique was Randal Kleiser's *Flight of the Navigator* in 1986. Bob Hoffman was part of the effects team that rendered a CG shiny spaceship flying over and reflecting airports, fields, and oceans. The technique was recently revisited to render the reflective Naboo spacecraft in *Star Wars: Episode I*.

A still from Randal Kleiser's 1986 film Flight of the Navigator,
demonstrating reflection mapping in a feature film.
Image courtesy of Bob Hoffman.

Greene 1986



Also in 1986, Ned Greene published a paper further developing and formalizing the technique of reflection mapping. In particular, he showed that environment maps could be pre-filtered and indexed with summed-area tables in order to perform a good approximation to correct antialiasing. Greene combined a real 180-degree fisheye image of the sky with a computer-generated image of desert terrain to create a full-view environment cube.

- Ned Greene. Environment Mapping and Other Applications of World Projections. IEEE Computer Graphics and Applications,
  Vol 6. No. 11. Nov. 1986.

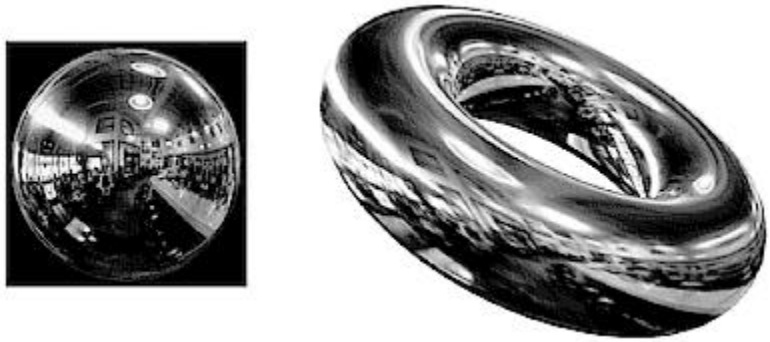
In this paper, Greene constructed an environment map using a photograph of the sky, and a rendering of the ground. In the rendering on the right, the map was re-warped to directly render the environment as well as to environment-map the ship.


Terminator II - 1991

Reflection Mapping made its most visible splash to date in 1991 in a film by James Cameron. Inspired by the use of reflection mapping (as well as shape morphing) in "Flight of the Navigator", Cameron used the technique to create the amazing look of the T1000 robot in "Terminator II".

Haeberli and Segal 1993

In 1993, Paul Haeberli and Mark Segal published a wonderful review of innovative uses of texture-mapping. Reflection mapping was one such application, and they demonstrated the technique by applying a mirrored ball image taken in a cafe to atorus shape.



A reflection mapping still from Haeberli and Segal, 1993.

- Paul Haeberli and Mark Segal. Texture Mapping as a Fundamental Drawing Primitive. Fourth Eurographics Workshop on Rendering. June 1993, pp. 259-266.

The full paper is available at Paul Haeberli's delightful Graphica Obscura website.

Paul Debevec / *debevec@cs.berkeley.edu* *http://www.cs.berkeley.edu/~debevec/ReflectionMapping/*

Recovering High Dynamic Range Radiance Maps from Photographs

Paul E. Debevec Jitendra Malik

University of California at Berkeley[1]

ABSTRACT

We present a method of recovering high dynamic range radiance maps from photographs taken with conventional imaging equipment. In our method, multiple photographs of the scene are taken with different amounts of exposure. Our algorithm uses these differently exposed photographs to recover the response function of the imaging process, up to factor of scale, using the assumption of reciprocity. With the known response function, the algorithm can fuse the multiple photographs into a single, high dynamic range radiance map whose pixel values are proportional to the true radiance values in the scene. We demonstrate our method on images acquired with both photochemical and digital imaging processes. We discuss how this work is applicable in many areas of computer graphics involving digitized photographs, including image-based modeling, image compositing, and image processing. Lastly, we demonstrate a few applications of having high dynamic range radiance maps, such as synthesizing realistic motion blur and simulating the response of the human visual system.

CR Descriptors: I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding - *Intensity, color, photometry and thresholding*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *Color, shading, shadowing, and texture*; I.4.1 [**Image Processing**]: Digitization - *Scanning*; I.4.8 [**Image Processing**]: Scene Analysis - *Photometry, Sensor Fusion*.

1 Introduction

Digitized photographs are becoming increasingly important in computer graphics. More than ever, scanned images are used as texture maps for geometric models, and recent work in image-based modeling and rendering uses images as the fundamental modeling primitive. Furthermore, many of today's graphics applications require computer-generated images to mesh seamlessly with real photographic imagery. Properly using photographically acquired imagery in these applications can greatly benefit from an accurate model of the photographic process.

When we photograph a scene, either with film or an electronic imaging array, and digitize the photograph to obtain a two-dimensional array of "brightness" values, these values are rarely true measurements of relative radiance in the scene. For example, if one pixel has twice the value of another, it is unlikely that it observed twice the radiance. Instead, there is usually an unknown, nonlinear mapping that determines how radiance in the scene becomes pixel values in the image.

This nonlinear mapping is hard to know beforehand because it is actually the composition of several nonlinear mappings that occur in the photographic process. In a conventional camera (see Fig. 1), the film is first exposed to light to form a latent image. The film is then developed to change this latent image into variations in transparency, or *density*, on the film. The film can then be digitized using a film scanner, which projects light through the film onto an electronic light-sensitive array, converting the image to electrical voltages. These voltages are digitized, and then manipulated before finally being written to the storage medium. If prints of the film are scanned rather than the film itself, then the printing process can also introduce nonlinear mappings.

In the first stage of the process, the film response to variations in exposure X (which is $E\Delta t$, the product of the irradiance E the film receives and the exposure time Δt) is a non-linear function, called the "characteristic curve" of the film. Noteworthy in the typical characteristic curve is the presence of a small response with no exposure and saturation at high exposures. The development, scanning and digitization processes usually introduce their own nonlinearities which compose to give the aggregate nonlinear relationship between the image pixel exposures X and their values Z.

Digital cameras, which use charge coupled device (CCD) arrays to image the scene, are prone to the same difficulties. Although the charge collected by a CCD element is proportional to its irradiance, most digital cameras apply a nonlinear mapping to the CCD outputs before they are written to the storage medium. This nonlinear mapping is used in various ways to mimic the response characteristics of film, anticipate nonlinear responses in the display device, and often to convert 12-bit output from the CCD's analog-to-digital converters to 8-bit values commonly used to store images. As with film, the most significant nonlinearity in the response curve is at its saturation point, where any pixel with a radiance above a certain level is mapped to the same maximum image value.

Why is this any problem at all? The most obvious difficulty, as any amateur or professional photographer knows, is that of limited dynamic range—one has to choose the range of radiance values that are of interest and determine the exposure time suitably. Sunlit scenes, and scenes with shiny materials and artificial light sources, often have extreme differences in radiance values that are impossible to capture without either under-exposing or saturating the film. To cover the full dynamic range in such a scene, one can take a series of photographs with different exposures. This then poses a problem: how can we combine these separate images into a composite radiance map? Here the fact that the mapping from scene radiance to pixel values is unknown and nonlinear begins to haunt us. The purpose of this paper is to present a simple technique for recovering this response function, up to a scale factor, using nothing more than a set of photographs taken with varying, known exposure durations. With this mapping, we then use the pixel values from all available photographs to construct an accurate map of the radiance in the scene, up to a factor of scale. This radiance map will cover

[1]Computer Science Division, University of California at Berkeley, Berkeley, CA 94720-1776. Email: debevec@cs.berkeley.edu, malik@cs.berkeley.edu. More information and additional results may be found at: http://www.cs.berkeley.edu/~debevec/Research/HDR

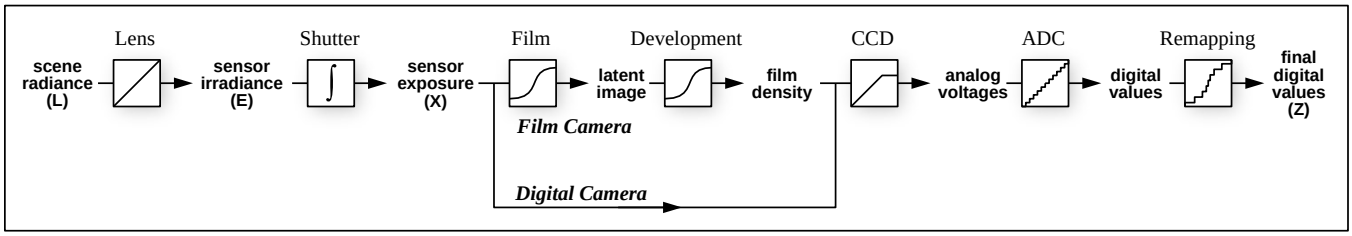


Figure 1: **Image Acquisition Pipeline** *shows how scene radiance becomes pixel values for both film and digital cameras. Unknown nonlinear mappings can occur during exposure, development, scanning, digitization, and remapping. The algorithm in this paper determines the aggregate mapping from scene radiance L to pixel values Z from a set of differently exposed images.*

the entire dynamic range captured by the original photographs.

1.1 Applications

Our technique of deriving imaging response functions and recovering high dynamic range radiance maps has many possible applications in computer graphics:

Image-based modeling and rendering

Image-based modeling and rendering systems to date (e.g. [11, 15, 2, 3, 12, 6, 17]) make the assumption that all the images are taken with the same exposure settings and film response functions. However, almost any large-scale environment will have some areas that are much brighter than others, making it impossible to adequately photograph the scene using a single exposure setting. In indoor scenes with windows, this situation often arises within the field of view of a single photograph, since the areas visible through the windows can be far brighter than the areas inside the building.

By determining the response functions of the imaging device, the method presented here allows one to correctly fuse pixel data from photographs taken at different exposure settings. As a result, one can properly photograph outdoor areas with short exposures, and indoor areas with longer exposures, without creating inconsistencies in the data set. Furthermore, knowing the response functions can be helpful in merging photographs taken with different imaging systems, such as video cameras, digital cameras, and film cameras with various film stocks and digitization processes.

The area of image-based modeling and rendering is working toward recovering more advanced reflection models (up to complete BRDF's) of the surfaces in the scene (e.g. [21]). These methods, which involve observing surface radiance in various directions under various lighting conditions, require absolute radiance values rather than the nonlinearly mapped pixel values found in conventional images. Just as important, the recovery of high dynamic range images will allow these methods to obtain accurate radiance values from surface specularities and from incident light sources. Such higher radiance values usually become clamped in conventional images.

Image processing

Most image processing operations, such as blurring, edge detection, color correction, and image correspondence, expect pixel values to be proportional to the scene radiance. Because of nonlinear image response, especially at the point of saturation, these operations can produce incorrect results for conventional images.

In computer graphics, one common image processing operation is the application of synthetic motion blur to images. In our results (Section 3), we will show that using true radiance maps produces significantly more realistic motion blur effects for high dynamic range scenes.

Image compositing

Many applications in computer graphics involve compositing image data from images obtained by different processes. For example, a background matte might be shot with a still camera, live action might be shot with a different film stock or scanning process, and CG elements would be produced by rendering algorithms. When there are significant differences in the response curves of these imaging processes, the composite image can be visually unconvincing. The technique presented in this paper provides a convenient and robust method of determining the overall response curve of any imaging process, allowing images from different processes to be used consistently as radiance maps. Furthermore, the recovered response curves can be inverted to render the composite radiance map as if it had been photographed with any of the original imaging processes, or a different imaging process entirely.

A research tool

One goal of computer graphics is to simulate the image formation process in a way that produces results that are consistent with what happens in the real world. Recovering radiance maps of real-world scenes should allow more quantitative evaluations of rendering algorithms to be made in addition to the qualitative scrutiny they traditionally receive. In particular, the method should be useful for developing reflectance and illumination models, and comparing global illumination solutions against ground truth data.

Rendering high dynamic range scenes on conventional display devices is the subject of considerable previous work, including [20, 16, 5, 23]. The work presented in this paper will allow such methods to be tested on real radiance maps in addition to synthetically computed radiance solutions.

1.2 Background

The photochemical processes involved in silver halide photography have been the subject of continued innovation and research ever since the invention of the daguerretype in 1839. [18] and [8] provide a comprehensive treatment of the theory and mechanisms involved. For the newer technology of solid-state imaging with charge coupled devices, [19] is an excellent reference. The technical and artistic problem of representing the dynamic range of a natural scene on the limited range of film has concerned photographers from the early days – [1] presents one of the best known systems to choose shutter speeds, lens apertures, and developing conditions to best coerce the dynamic range of a scene to fit into what is possible on a print. In scientific applications of photography, such as in astronomy, the nonlinear film response has been addressed by suitable calibration procedures. It is our objective instead to develop a simple self-calibrating procedure not requiring calibration charts or photometric measuring devices.

In previous work, [13] used multiple flux integration times of a CCD array to acquire extended dynamic range images. Since direct CCD outputs were available, the work did not need to deal with the

problem of nonlinear pixel value response. [14] addressed the problem of nonlinear response but provide a rather limited method of recovering the response curve. Specifically, a parametric form of the response curve is arbitrarily assumed, there is no satisfactory treatment of image noise, and the recovery process makes only partial use of the available data.

2 The Algorithm

This section presents our algorithm for recovering the film response function, and then presents our method of reconstructing the high dynamic range radiance image from the multiple photographs. We describe the algorithm assuming a grayscale imaging device. We discuss how to deal with color in Section 2.6.

2.1 Film Response Recovery

Our algorithm is based on exploiting a physical property of imaging systems, both photochemical and electronic, known as *reciprocity*.

Let us consider photographic film first. The response of a film to variations in exposure is summarized by the characteristic curve (or Hurter-Driffield curve). This is a graph of the optical density D of the processed film against the logarithm of the exposure X to which it has been subjected. The exposure X is defined as the product of the irradiance E at the film and exposure time, Δt, so that its units are $\mathrm{J m}^{-2}$. Key to the very concept of the characteristic curve is the assumption that only the product $E\Delta t$ is important, that halving E and doubling Δt will not change the resulting optical density D. Under extreme conditions (very large or very low Δt), the reciprocity assumption can break down, a situation described as reciprocity failure. In typical print films, reciprocity holds to within $\frac{1}{3}$ stop[1] for exposure times of 10 seconds to 1/10,000 of a second.[2] In the case of charge coupled arrays, reciprocity holds under the assumption that each site measures the total number of photons it absorbs during the integration time.

After the development, scanning and digitization processes, we obtain a digital number Z, which is a nonlinear function of the original exposure X at the pixel. Let us call this function f, which is the composition of the characteristic curve of the film as well as all the nonlinearities introduced by the later processing steps. Our first goal will be to recover this function f. Once we have that, we can compute the exposure X at each pixel, as $X = f^{-1}(Z)$. We make the reasonable assumption that the function f is monotonically increasing, so its inverse f^{-1} is well defined. Knowing the exposure X and the exposure time Δt, the irradiance E is recovered as $E = X/\Delta t$, which we will take to be proportional to the radiance L in the scene.[3]

Before proceeding further, we should discuss the consequences of the spectral response of the sensor. The exposure X should be thought of as a function of wavelength $X(\lambda)$, and the abscissa on the characteristic curve should be the integral $\int X(\lambda)R(\lambda)d\lambda$ where $R(\lambda)$ is the spectral response of the sensing element at the pixel location. Strictly speaking, our use of irradiance, a radiometric quantity, is not justified. However, the spectral response of the sensor site may not be the photopic luminosity function V_λ, so the photometric term *illuminance* is not justified either. In what follows, we will use the term irradiance, while urging the reader to remember that the quantities we will be dealing with are weighted by the spectral response at the sensor site. For color photography, the color channels may be treated separately.

The input to our algorithm is a number of digitized photographs taken from the same vantage point with different known exposure durations Δt_j.[4] We will assume that the scene is static and that this process is completed quickly enough that lighting changes can be safely ignored. It can then be assumed that the film irradiance values E_i for each pixel i are constant. We will denote pixel values by Z_{ij} where i is a spatial index over pixels and j indexes over exposure times Δt_j. We may now write down the film reciprocity equation as:

$$Z_{ij} = f(E_i \Delta t_j) \tag{1}$$

Since we assume f is monotonic, it is invertible, and we can rewrite (1) as:

$$f^{-1}(Z_{ij}) = E_i \Delta t_j$$

Taking the natural logarithm of both sides, we have:

$$\ln f^{-1}(Z_{ij}) = \ln E_i + \ln \Delta t_j$$

To simplify notation, let us define function $g = \ln f^{-1}$. We then have the set of equations:

$$g(Z_{ij}) = \ln E_i + \ln \Delta t_j \tag{2}$$

where i ranges over pixels and j ranges over exposure durations. In this set of equations, the Z_{ij} are known, as are the Δt_j. The unknowns are the irradiances E_i, as well as the function g, although we assume that g is smooth and monotonic.

We wish to recover the function g and the irradiances E_i that best satisfy the set of equations arising from Equation 2 in a least-squared error sense. We note that recovering g only requires recovering the *finite* number of values that $g(z)$ can take since the domain of Z, pixel brightness values, is finite. Letting Z_{min} and Z_{max} be the least and greatest pixel values (integers), N be the number of pixel locations and P be the number of photographs, we formulate the problem as one of finding the $(Z_{max} - Z_{min} + 1)$ values of $g(Z)$ and the N values of $\ln E_i$ that minimize the following quadratic objective function:

$$\mathcal{O} = \sum_{i=1}^{N} \sum_{j=1}^{P} [g(Z_{ij}) - \ln E_i - \ln \Delta t_j]^2 + \lambda \sum_{z=Z_{min}+1}^{Z_{max}-1} g''(z)^2 \tag{3}$$

The first term ensures that the solution satisfies the set of equations arising from Equation 2 in a least squares sense. The second term is a smoothness term on the sum of squared values of the second derivative of g to ensure that the function g is smooth; in this discrete setting we use $g''(z) = g(z-1) - 2g(z) + g(z+1)$. This smoothness term is essential to the formulation in that it provides coupling between the values $g(z)$ in the minimization. The scalar λ weights the smoothness term relative to the data fitting term, and should be chosen appropriately for the amount of noise expected in the Z_{ij} measurements.

Because it is quadratic in the E_i's and $g(z)$'s, minimizing $\mathcal{O}$ is a straightforward linear least squares problem. The overdetermined

[1] 1 stop is a photographic term for a factor of two; $\frac{1}{3}$ stop is thus $2^{\frac{1}{3}}$

[2] An even larger dynamic range can be covered by using neutral density filters to lessen to amount of light reaching the film for a given exposure time. A discussion of the modes of reciprocity failure may be found in [18], ch. 4.

[3] L is proportional E for any particular pixel, but it is possible for the proportionality factor to be different at different places on the sensor. One formula for this variance, given in [7], is $E = L\frac{\pi}{4}\left(\frac{d}{f}\right)^2 cos^4\alpha$, where α measures the pixel's angle from the lens' optical axis. However, most modern camera lenses are designed to compensate for this effect, and provide a nearly constant mapping between radiance and irradiance at f/8 and smaller apertures. See also [10].

[4] Most modern SLR cameras have electronically controlled shutters which give extremely accurate and reproducible exposure times. We tested our Canon EOS Elan camera by using a Macintosh to make digital audio recordings of the shutter. By analyzing these recordings we were able to verify the accuracy of the exposure times to within a thousandth of a second. Conveniently, we determined that the actual exposure times varied by powers of two between stops ($\frac{1}{64}$, $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, 1, 2, 4, 8, 16, 32), rather than the rounded numbers displayed on the camera readout ($\frac{1}{60}$, $\frac{1}{30}$, $\frac{1}{15}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, 1, 2, 4, 8, 15, 30). Because of problems associated with vignetting, varying the aperture is not recommended.

system of linear equations is robustly solved using the singular value decomposition (SVD) method. An intuitive explanation of the procedure may be found in Fig. 2.

We need to make three additional points to complete our description of the algorithm:

First, the solution for the $g(z)$ and E_i values can only be up to a single scale factor α. If each log irradiance value $\ln E_i$ were replaced by $\ln E_i + \alpha$, and the function g replaced by $g + \alpha$, the system of equations 2 and also the objective function $\mathcal{O}$ would remain unchanged. To establish a scale factor, we introduce the additional constraint $g(Z_{mid}) = 0$, where $Z_{mid} = \frac{1}{2}(Z_{min} + Z_{max})$, simply by adding this as an equation in the linear system. The meaning of this constraint is that a pixel with value midway between Z_{min} and Z_{max} will be assumed to have unit exposure.

Second, the solution can be made to have a much better fit by anticipating the basic shape of the response function. Since $g(z)$ will typically have a steep slope near Z_{min} and Z_{max}, we should expect that $g(z)$ will be less smooth and will fit the data more poorly near these extremes. To recognize this, we can introduce a weighting function $w(z)$ to emphasize the smoothness and fitting terms toward the middle of the curve. A sensible choice of w is a simple hat function:

$$w(z) = \begin{cases} z - Z_{min} & \text{for } z \leq \frac{1}{2}(Z_{min} + Z_{max}) \\ Z_{max} - z & \text{for } z > \frac{1}{2}(Z_{min} + Z_{max}) \end{cases} \tag{4}$$

Equation 3 now becomes:

$$\mathcal{O} = \sum_{i=1}^{N} \sum_{j=1}^{P} \{w(Z_{ij}) \left[g(Z_{ij}) - \ln E_i - \ln \Delta t_j\right]\}^2 + \lambda \sum_{z=Z_{min}+1}^{Z_{max}-1} \left[w(z) g''(z)\right]^2$$

Finally, we need not use every available pixel site in this solution procedure. Given measurements of N pixels in P photographs, we have to solve for N values of $\ln E_i$ and $(Z_{max} - Z_{min})$ samples of g. To ensure a sufficiently overdetermined system, we want $N(P-1) > (Z_{max} - Z_{min})$. For the pixel value range $(Z_{max} - Z_{min}) = 255$, $P = 11$ photographs, a choice of N on the order of 50 pixels is more than adequate. Since the size of the system of linear equations arising from Equation 3 is on the order of $N \times P + Z_{max} - Z_{min}$, computational complexity considerations make it impractical to use every pixel location in this algorithm. Clearly, the pixel locations should be chosen so that they have a reasonably even distribution of pixel values from Z_{min} to Z_{max}, and so that they are spatially well distributed in the image. Furthermore, the pixels are best sampled from regions of the image with low intensity variance so that radiance can be assumed to be constant across the area of the pixel, and the effect of optical blur of the imaging system is minimized. So far we have performed this task by hand, though it could easily be automated.

Note that we have not explicitly enforced the constraint that g must be a monotonic function. If desired, this can be done by transforming the problem to a non-negative least squares problem. We have not found it necessary because, in our experience, the smoothness penalty term is enough to make the estimated g monotonic in addition to being smooth.

To show its simplicity, the MATLAB routine we used to minimize Equation 5 is included in the Appendix. Running times are on the order of a few seconds.

2.2 Constructing the High Dynamic Range Radiance Map

Once the response curve g is recovered, it can be used to quickly convert pixel values to relative radiance values, assuming the exposure Δt_j is known. Note that the curve can be used to determine radiance values in any image(s) acquired by the imaging process associated with g, not just the images used to recover the response function.

From Equation 2, we obtain:

$$\ln E_i = g(Z_{ij}) - \ln \Delta t_j \tag{5}$$

For robustness, and to recover high dynamic range radiance values, we should use all the available exposures for a particular pixel to compute its radiance. For this, we reuse the weighting function in Equation 4 to give higher weight to exposures in which the pixel's value is closer to the middle of the response function:

$$\ln E_i = \frac{\sum_{j=1}^{P} w(Z_{ij})(g(Z_{ij}) - \ln \Delta t_j)}{\sum_{j=1}^{P} w(Z_{ij})} \tag{6}$$

Combining the multiple exposures has the effect of reducing noise in the recovered radiance values. It also reduces the effects of imaging artifacts such as film grain. Since the weighting function ignores saturated pixel values, "blooming" artifacts[5] have little impact on the reconstructed radiance values.

2.2.1 Storage

In our implementation the recovered radiance map is computed as an array of single-precision floating point values. For efficiency, the map can be converted to the image format used in the RADIANCE [22] simulation and rendering system, which uses just eight bits for each of the mantissa and exponent. This format is particularly compact for color radiance maps, since it stores just one exponent value for all three color values at each pixel. Thus, in this format, a high dynamic range radiance map requires just one third more storage than a conventional RGB image.

2.3 How many images are necessary?

To decide on the number of images needed for the technique, it is convenient to consider the two aspects of the process:

1. *Recovering the film response curve:* This requires a minimum of two photographs. Whether two photographs are enough can be understood in terms of the heuristic explanation of the process of film response curve recovery shown in Fig. 2. If the scene has sufficiently many different radiance values, the entire curve can, in principle, be assembled by sliding together the sampled curve segments, each with only two samples. Note that the photos must be similar enough in their exposure amounts that some pixels fall into the working range[6] of the film in both images; otherwise, there is no information to relate the exposures to each other. Obviously, using more than two images with differing exposure times improves performance with respect to noise sensitivity.
2. *Recovering a radiance map given the film response curve:* The number of photographs needed here is a function of the dynamic range of radiance values in the scene. Suppose the range of maximum to minimum radiance values that we are

[5] Blooming occurs when charge or light at highly saturated sites on the imaging surface spills over and affects values at neighboring sites.

[6] The *working range* of the film corresponds to the middle section of the response curve. The ends of the curve, in which large changes in exposure cause only small changes in density (or pixel value), are called the *toe* and the *shoulder*.

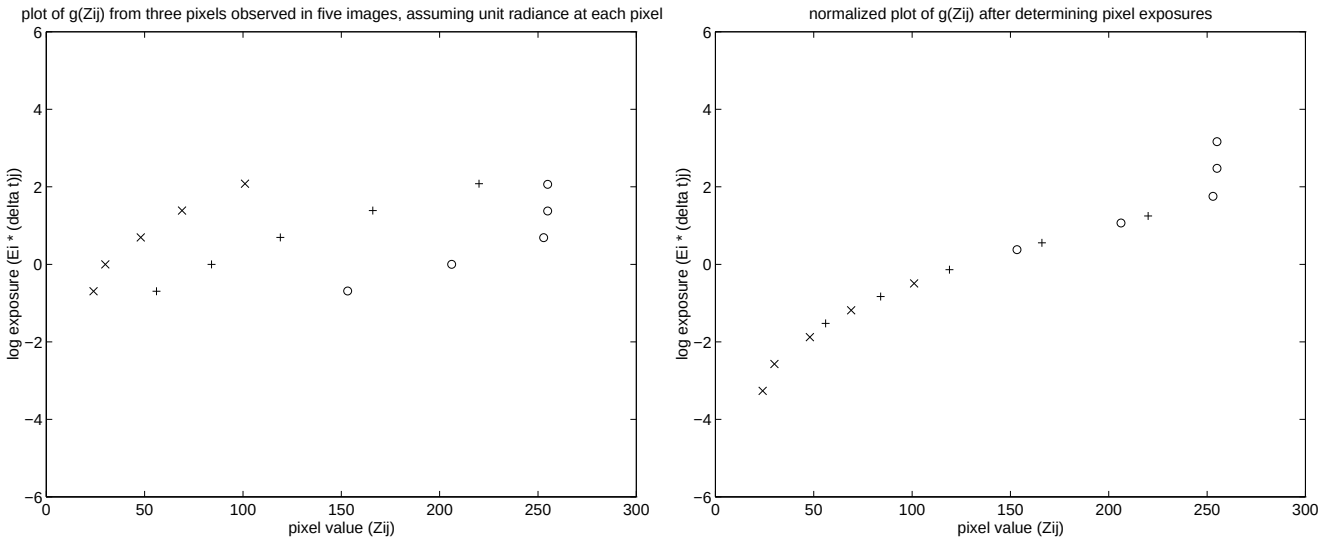


Figure 2: *In the figure on the left, the $\times$ symbols represent samples of the g curve derived from the digital values at one pixel for 5 different known exposures using Equation 2. The unknown log irradiance $\ln E_i$ has been arbitrarily assumed to be 0. Note that the shape of the g curve is correct, though its position on the vertical scale is arbitrary corresponding to the unknown $\ln E_i$. The $+$ and $\circ$ symbols show samples of g curve segments derived by consideration of two other pixels; again the vertical position of each segment is arbitrary. Essentially, what we want to achieve in the optimization process is to slide the 3 sampled curve segments up and down (by adjusting their $\ln E_i$'s) until they "line up" into a single smooth, monotonic curve, as shown in the right figure. The vertical position of the composite curve will remain arbitrary.*

interested in recovering accurately is R, and the film is capable of representing in its working range a dynamic range of F. Then the minimum number of photographs needed is $\lceil \frac{R}{F} \rceil$ to ensure that every part of the scene is imaged in at least one photograph at an exposure duration that puts it in the working range of the film response curve. As in recovering the response curve, using more photographs than strictly necessary will result in better noise sensitivity.

If one wanted to use as few photographs as possible, one might first recover the response curve of the imaging process by photographing a scene containing a diverse range of radiance values at three or four different exposures, differing by perhaps one or two stops. This response curve could be used to determine the working range of the imaging process, which for the processes we have seen would be as many as five or six stops. For the remainder of the shoot, the photographer could decide for any particular scene the number of shots necessary to cover its entire dynamic range. For diffuse indoor scenes, only one exposure might be necessary; for scenes with high dynamic range, several would be necessary. By recording the exposure amount for each shot, the images could then be converted to radiance maps using the pre-computed response curve.

2.4 Recovering extended dynamic range from single exposures

Most commericially available film scanners can detect reasonably close to the full range of useful densities present in film. However, many of these scanners (as well as the Kodak PhotoCD process) produce 8-bit-per-channel images designed to be viewed on a screen or printed on paper. Print film, however, records a significantly greater dynamic range than can be displayed with either of these media. As a result, such scanners deliver only a portion of the detected dynamic range of print film in a single scan, discarding information in either high or low density regions. The portion of the detected dynamic range that is delivered can usually be influenced by "brightness" or "density adjustment" controls.

The method presented in this paper enables two methods for recovering the full dynamic range of print film which we will briefly outline[7]. In the first method, the print negative is scanned with the scanner set to scan slide film. Most scanners will then record the entire detectable dynamic range of the film in the resulting image. As before, a series of differently exposed images of the same scene can be used to recover the response function of the imaging system with each of these scanner settings. This response function can then be used to convert individual exposures to radiance maps. Unfortunately, since the resulting image is still 8-bits-per-channel, this results in increased quantization.

In the second method, the film can be scanned twice with the scanner set to different density adjustment settings. A series of differently exposed images of the same scene can then be used to recover the response function of the imaging system at each of these density adjustment settings. These two response functions can then be used to combine two scans of any single negative using a similar technique as in Section 2.2.

2.5 Obtaining Absolute Radiance

For many applications, such as image processing and image compositing, the relative radiance values computed by our method are all that are necessary. If needed, an approximation to the scaling term necessary to convert to absolute radiance can be derived using the ASA of the film[8] and the shutter speeds and exposure amounts in the photographs. With these numbers, formulas that give an approximate prediction of film response can be found in [9]. Such an approximation can be adequate for simulating visual artifacts such as glare, and predicting areas of scotopic retinal response. If desired, one could recover the scaling factor precisely by photographing a calibration luminaire of known radiance, and scaling the radiance values to agree with the known radiance of the luminaire.

2.6 Color

Color images, consisting of red, green, and blue channels, can be processed by reconstructing the imaging system response curve for

[7]This work was done in collaboration with Gregory Ward Larson

[8]Conveniently, most digital cameras also specify their sensitivity in terms of ASA.

each channel independently. Unfortunately, there will be three unknown scaling factors relating relative radiance to absolute radiance, one for each channel. As a result, different choices of these scaling factors will change the color balance of the radiance map.

By default, the algorithm chooses the scaling factor such that a pixel with value Z_{mid} will have unit exposure. Thus, any pixel with the RGB value $(Z_{mid}, Z_{mid}, Z_{mid})$ will have equal radiance values for R, G, and B, meaning that the pixel is achromatic. If the three channels of the imaging system actually do respond equally to achromatic light in the neighborhood of Z_{mid}, then our procedure correctly reconstructs the relative radiances.

However, films are usually calibrated to respond achromatically to a particular color of light C, such as sunlight or fluorescent light. In this case, the radiance values of the three channels should be scaled so that the pixel value $(Z_{mid}, Z_{mid}, Z_{mid})$ maps to a radiance with the same color ratios as C. To properly model the color response of the entire imaging process rather than just the film response, the scaling terms can be adjusted by photographing a calibration luminaire of known color.

2.7 Taking virtual photographs

The recovered response functions can also be used to map radiance values back to pixel values for a given exposure Δt using Equation 1. This process can be thought of as taking a virtual photograph of the radiance map, in that the resulting image will exhibit the response qualities of the modeled imaging system. Note that the response functions used need not be the same response functions used to construct the original radiance map, which allows photographs acquired with one imaging process to be rendered as if they were acquired with another.[9]

3 Results

Figures 3-5 show the results of using our algorithm to determine the response curve of a DCS460 digital camera. Eleven grayscale photographs filtered down to 765×509 resolution (Fig. 3) were taken at f/8 with exposure times ranging from $\frac{1}{30}$ of a second to 30 seconds, with each image receiving twice the exposure of the previous one. The film curve recovered by our algorithm from 45 pixel locations observed across the image sequence is shown in Fig. 4. Note that although CCD image arrays naturally produce linear output, from the curve it is evident that the camera nonlinearly remaps the data, presumably to mimic the response curves found in film. The underlying registered $(E_i \Delta t_j, Z_{ij})$ data are shown as light circles underneath the curve; some outliers are due to sensor artifacts (light horizontal bands across some of the darker images.)

Fig. 5 shows the reconstructed high dynamic range radiance map. To display this map, we have taken the logarithm of the radiance values and mapped the range of these values into the range of the display. In this representation, the pixels at the light regions do not saturate, and detail in the shadow regions can be made out, indicating that all of the information from the original image sequence is present in the radiance map. The large range of values present in the radiance map (over four orders of magnitude of useful dynamic range) is shown by the values at the marked pixel locations.

Figure 6 shows sixteen photographs taken inside a church with a Canon 35mm SLR camera on Fuji 100 ASA color print film. A fisheye 15mm lens set at f/8 was used, with exposure times ranging from 30 seconds to $\frac{1}{1000}$ of a second in 1-stop increments. The film was developed professionally and scanned in using a Kodak PhotoCD film scanner. The scanner was set so that it would not individually

[9] Note that here we are assuming that the spectral response functions for each channel of the two imaging processes is the same. Also, this technique does not model many significant qualities of an imaging system such as film grain, chromatic aberration, blooming, and the modulation transfer function.

Figure 3: **(a)** *Eleven grayscale photographs of an indoor scene acquired with a Kodak DCS460 digital camera, with shutter speeds progressing in 1-stop increments from $\frac{1}{30}$ of a second to 30 seconds.*

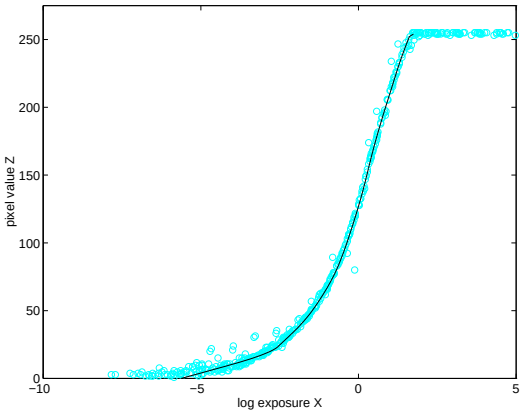


Figure 4: *The response function of the DCS460 recovered by our algorithm, with the underlying $(E_i \Delta t_j, Z_{ij})$ data shown as light circles. The logarithm is base e.*

Figure 5: *The reconstructed high dynamic range radiance map, mapped into a grayscale image by taking the logarithm of the radiance values. The relative radiance values of the marked pixel locations, clockwise from lower left: 1.0, 46.2, 1907.1, 15116.0, and 18.0.*

Figure 6: *Sixteen photographs of a church taken at 1-stop increments from 30 sec to* $\frac{1}{1000}$ *sec. The sun is directly behind the rightmost stained glass window, making it especially bright. The blue borders seen in some of the image margins are induced by the image registration process.*

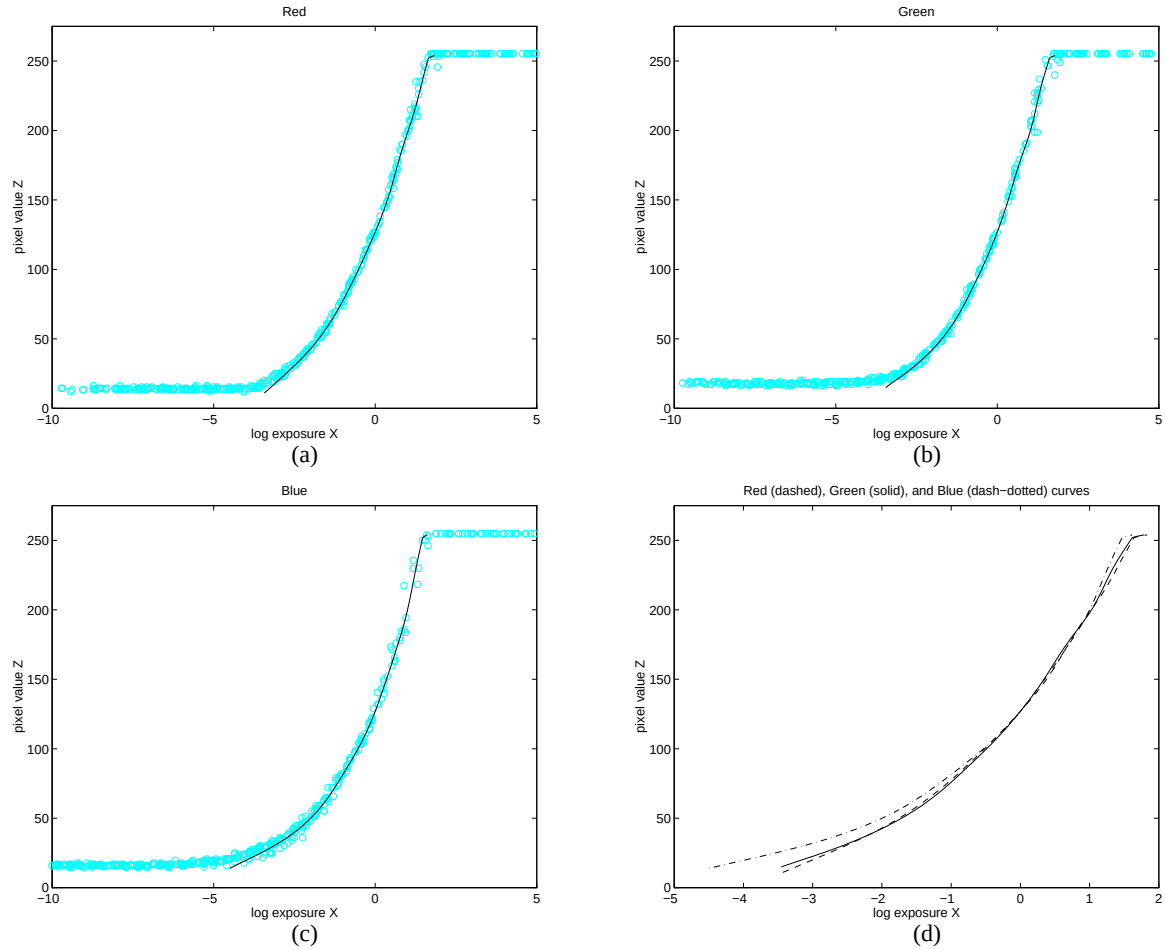
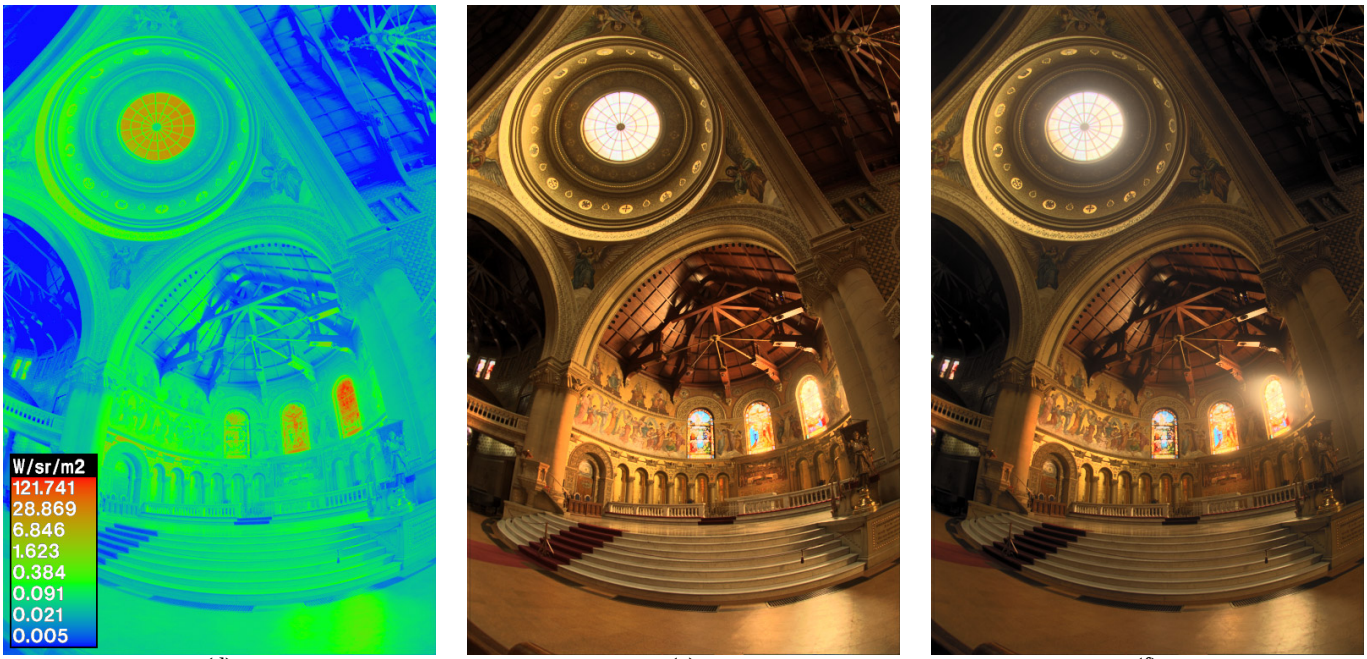


Figure 7: *Recovered response curves for the imaging system used in the church photographs in Fig. 8.* **(a-c)** *Response functions for the red, green, and blue channels, plotted with the underlying* $(E_i \Delta t_j, Z_{ij})$ *data shown as light circles.* **(d)** *The response functions for red, green, and blue plotted on the same axes. Note that while the red and green curves are very consistent, the blue curve rises significantly above the others for low exposure values. This indicates that dark regions in the images exhibit a slight blue cast. Since this artifact is recovered by the response curves, it does not affect the relative radiance values.*

(a) (b) (c)



(d) (e) (f)

Figure 8: **(a)** *An actual photograph, taken with conventional print film at two seconds and scanned to PhotoCD.* **(b)** *The high dynamic range radiance map, displayed by linearly mapping its entire dynamic range into the dynamic range of the display device.* **(c)** *The radiance map, displayed by linearly mapping the lower* 0.1% *of its dynamic range to the display device.* **(d)** *A false-color image showing relative radiance values for a grayscale version of the radiance map, indicating that the map contains over five orders of magnitude of useful dynamic range.* **(e)** *A rendering of the radiance map using adaptive histogram compression.* **(f)** *A rendering of the radiance map using histogram compression and also simulating various properties of the human visual system, such as glare, contrast sensitivity, and scotopic retinal response. Images (e) and (f) were generated by a method described in [23]. Images (d-f) courtesy of Gregory Ward Larson.*

adjust the brightness and contrast of the images[10] to guarantee that each image would be digitized using the same response function.

An unfortunate aspect of the PhotoCD process is that it does not scan precisely the same area of each negative relative to the extents of the image.[11] To counteract this effect, we geometrically registered the images to each other using a using normalized correlation (see [4]) to determine, with sub-pixel accuracy, corresponding pixels between pairs of images.

Fig. 7(a-c) shows the response functions for the red, green, and blue channels of the church sequence recovered from 28 pixel locations. Fig. 7(d) shows the recovered red, green, and blue response curves plotted on the same set of axes. From this plot, we can see that while the red and green curves are very consistent, the blue curve rises significantly above the others for low exposure values. This indicates that dark regions in the images exhibit a slight blue cast. Since this artifact is modeled by the response curves, it will not affect the relative radiance values.

Fig. 8 interprets the recovered high dynamic range radiance map in a variety of ways. Fig. 8(a) is one of the actual photographs, which lacks detail in its darker regions at the same time that many values within the two rightmost stained glass windows are saturated. Figs. 8(b,c) show the radiance map, linearly scaled to the display device using two different scaling factors. Although one scaling factor is one thousand times the other, there is useful detail in both images. Fig. 8(d) is a false-color image showing radiance values for a grayscale version of the radiance map; the highest listed radiance value is nearly 250,000 times that of the lowest. Figs. 8(e,f) show two renderings of the radiance map using a new tone reproduction algorithm [23]. Although the rightmost stained glass window has radiance values over a thousand times higher than the darker areas in the rafters, these renderings exhibit detail in both areas.

Figure 9 demonstrates two applications of the techniques presented in this paper: accurate signal processing and virtual photography. The task is to simulate the effects of motion blur caused by moving the camera during the exposure. Fig. 9(a) shows the results of convolving an actual, low-dynamic range photograph with a 37×1 pixel box filter to simulate horizontal motion blur. Fig. 9(b) shows the results of applying this same filter to the high dynamic range radiance map, and then sending this filtered radiance map back through the recovered film response functions using the same exposure time Δt as in the actual photograph. Because we are seeing this image through the actual image response curves, the two left images are tonally consistent with each other. However, there is a large difference between these two images near the bright spots. In the photograph, the bright radiance values have been clamped to the maximum pixel values by the response function. As a result, these clamped values blur with lower neighboring values and fail to saturate the image in the final result, giving a muddy appearance.

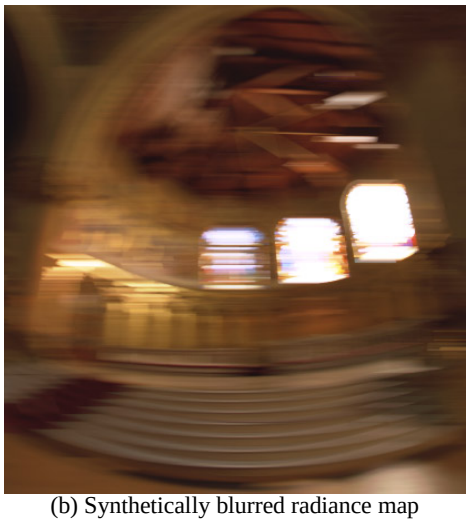
In Fig. 9(b), the extremely high pixel values were represented properly in the radiance map and thus remained at values above the level of the response function's saturation point within most of the blurred region. As a result, the resulting virtual photograph exhibits several crisply-defined saturated regions.

Fig. 9(c) is an actual photograph with real motion blur induced by spinning the camera on the tripod during the exposure, which is equal in duration to Fig. 9(a) and the exposure simulated in Fig. 9(b). Clearly, in the bright regions, the blurring effect is qualitatively similar to the synthetic blur in 9(b) but not 9(a). The precise shape of the real motion blur is curved and was not modeled for this demonstration.

[10] This feature of the PhotoCD process is called "Scene Balance Adjustment", or SBA.

[11] This is far less of a problem for cinematic applications, in which the film sprocket holes are used to expose and scan precisely the same area of each frame.

(a) Synthetically blurred digital image



(b) Synthetically blurred radiance map

(c) Actual blurred photograph

Figure 9: **(a)** *Synthetic motion blur applied to one of the original digitized photographs. The bright values in the windows are clamped before the processing, producing mostly unsaturated values in the blurred regions.* **(b)** *Synthetic motion blur applied to a recovered high-dynamic range radiance map, then virtually rephotographed through the recovered film response curves. The radiance values are clamped to the display device* after *the processing, allowing pixels to remain saturated in the window regions.* **(c)** *Real motion blur created by rotating the camera on the tripod during the exposure, which is much more consistent with (b) than (a).*

4 Conclusion

We have presented a simple, practical, robust and accurate method of recovering high dynamic range radiance maps from ordinary photographs. Our method uses the constraint of sensor reciprocity to derive the response function and relative radiance values directly from a set of images taken with different exposures. This work has a wide variety of applications in the areas of image-based modeling and rendering, image processing, and image compositing, a few of which we have demonstrated. It is our hope that this work will be able to help both researchers and practitioners of computer graphics make much more effective use of digitized photographs.

Acknowledgments

The authors wish to thank Tim Hawkins, Carlo Séquin, David Forsyth, Steve Chenney, Chris Healey, and our reviewers for their valuable help in revising this paper. This research was supported by a Multidisciplinary University Research Initiative on three dimensional direct visualization from ONR and BMDO, grant FDN00014-96-1-1200.

References

[1] ADAMS, A. *Basic Photo*, 1st ed. Morgan & Morgan, Hastings-on-Hudson, New York, 1970.

[2] CHEN, E. QuickTime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH '95* (1995).

[3] DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96* (August 1996), pp. 11–20.

[4] FAUGERAS, O. *Three-Dimensional Computer Vision*. MIT Press, 1993.

[5] FERWERDA, J. A., PATTANAIK, S. N., SHIRLEY, P., AND GREENBERG, D. P. A model of visual adaptation for realistic image synthesis. In *SIGGRAPH '96* (1996), pp. 249–258.

[6] GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. The Lumigraph. In *SIGGRAPH '96* (1996), pp. 43–54.

[7] HORN, B. K. P. *Robot Vision*. MIT Press, Cambridge, Mass., 1986, ch. 10, pp. 206–208.

[8] JAMES, T., Ed. *The Theory of the Photographic Process*. Macmillan, New York, 1977.

[9] KAUFMAN, J. E., Ed. *IES Lighting Handbook; the standard lighting guide*, 7th ed. Illuminating Engineering Society, New York, 1987, p. 24.

[10] KOLB, C., MITCHELL, D., AND HANRAHAN, P. A realistic camera model for computer graphics. In *SIGGRAPH '95* (1995).

[11] LAVEAU, S., AND FAUGERAS, O. 3-D scene representation as a collection of images. In *Proceedings of 12th International Conference on Pattern Recognition* (1994), vol. 1, pp. 689–691.

[12] LEVOY, M., AND HANRAHAN, P. Light field rendering. In *SIGGRAPH '96* (1996), pp. 31–42.

[13] MADDEN, B. C. Extended intensity range imaging. Tech. rep., GRASP Laboratory, University of Pennsylvania, 1993.

[14] MANN, S., AND PICARD, R. W. Being 'undigital' with digital cameras: Extending dynamic range by combining differently exposed pictures. In *Proceedings of IS&T 46th annual conference* (May 1995), pp. 422–428.

[15] MCMILLAN, L., AND BISHOP, G. Plenoptic Modeling: An image-based rendering system. In *SIGGRAPH '95* (1995).

[16] SCHLICK, C. Quantization techniques for visualization of high dynamic range pictures. In *Fifth Eurographics Workshop on Rendering (Darmstadt, Germany)* (June 1994), pp. 7–18.

[17] SZELISKI, R. Image mosaicing for tele-reality applications. In *IEEE Computer Graphics and Applications* (1996).

[18] TANI, T. *Photographic sensitivity : theory and mechanisms*. Oxford University Press, New York, 1995.

[19] THEUWISSEN, A. J. P. *Solid-state imaging with charge-coupled devices*. Kluwer Academic Publishers, Dordrecht; Boston, 1995.

[20] TUMBLIN, J., AND RUSHMEIER, H. Tone reproduction for realistic images. *IEEE Computer Graphics and Applications 13*, 6 (1993), 42–48.

[21] WARD, G. J. Measuring and modeling anisotropic reflection. In *SIGGRAPH '92* (July 1992), pp. 265–272.

[22] WARD, G. J. The radiance lighting simulation and rendering system. In *SIGGRAPH '94* (July 1994), pp. 459–472.

[23] WARD, G. J., RUSHMEIER, H., AND PIATKO, C. A visibility matching tone reproduction operator for high dynamic range scenes. Tech. Rep. LBNL-39882, Lawrence Berkeley National Laboratory, March 1997.

A Matlab Code

Here is the MATLAB code used to solve the linear system that minimizes the objective function $\mathcal{O}$ in Equation 3. Given a set of observed pixel values in a set of images with known exposures, this routine reconstructs the imaging response curve and the radiance values for the given pixels. The weighting function $w(z)$ is found in Equation 4.

```
%
% gsolve.m – Solve for imaging system response function
%
% Given a set of pixel values observed for several pixels in several
% images with different exposure times, this function returns the
% imaging system's response function g as well as the log film irradiance
% values for the observed pixels.
%
% Assumes:
%
%  Zmin = 0
%  Zmax = 255
%
% Arguments:
%
%  Z(i,j) is the pixel values of pixel location number i in image j
%  B(j)   is the log delta t, or log shutter speed, for image j
%  l      is lamdba, the constant that determines the amount of smoothness
%  w(z)   is the weighting function value for pixel value z
%
% Returns:
%
%  g(z)   is the log exposure corresponding to pixel value z
%  lE(i)  is the log film irradiance at pixel location i
%

function [g,lE]=gsolve(Z,B,l,w)

n = 256;

A = zeros(size(Z,1)*size(Z,2)+n+1,n+size(Z,1));
b = zeros(size(A,1),1);

%% Include the data-fitting equations

k = 1;
for i=1:size(Z,1)
  for j=1:size(Z,2)
    wij = w(Z(i,j)+1);
    A(k,Z(i,j)+1) = wij;  A(k,n+i) = -wij;       b(k,1) = wij * B(i,j);
    k=k+1;
  end
end

%% Fix the curve by setting its middle value to 0

A(k,129) = 1;
k=k+1;

%% Include the smoothness equations

for i=1:n-2
  A(k,i)=l*w(i+1);        A(k,i+1)=-2*l*w(i+1);  A(k,i+2)=l*w(i+1);
  k=k+1;
end

%% Solve the system using SVD

x = A\b;

g = x(1:n);
lE = x(n+1:size(x,1));
```

To appear in the SIGGRAPH 98 conference proceedings

Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography

Paul Debevec

University of California at Berkeley[1]

ABSTRACT

We present a method that uses measured scene radiance and global illumination in order to add new objects to light-based models with correct lighting. The method uses a high dynamic range image-based model of the scene, rather than synthetic light sources, to illuminate the new objects. To compute the illumination, the scene is considered as three components: the distant scene, the local scene, and the synthetic objects. The distant scene is assumed to be photometrically unaffected by the objects, obviating the need for reflectance model information. The local scene is endowed with estimated reflectance model information so that it can catch shadows and receive reflected light from the new objects. Renderings are created with a standard global illumination method by simulating the interaction of light amongst the three components. A differential rendering technique allows for good results to be obtained when only an estimate of the local scene reflectance properties is known.

We apply the general method to the problem of rendering synthetic objects into real scenes. The light-based model is constructed from an approximate geometric model of the scene and by using a light probe to measure the incident illumination at the location of the synthetic objects. The global illumination solution is then composited into a photograph of the scene using the differential rendering technique. We conclude by discussing the relevance of the technique to recovering surface reflectance properties in uncontrolled lighting situations. Applications of the method include visual effects, interior design, and architectural visualization.

CR Descriptors: I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding - *Intensity, color, photometry and thresholding*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *Color, shading, shadowing, and texture*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *Radiosity*; I.4.1 [**Image Processing**]: Digitization - *Scanning*; I.4.8 [**Image Processing**]: Scene Analysis - *Photometry, Sensor Fusion*.

[1]Computer Science Division, University of California at Berkeley, Berkeley, CA 94720−1776. Email: debevec@cs.berkeley.edu. More information and additional results may be found at: http://www.cs.berkeley.edu/~debevec/Research

1 Introduction

Rendering synthetic objects into real-world scenes is an important application of computer graphics, particularly in architectural and visual effects domains. Oftentimes, a piece of furniture, a prop, or a digital creature or actor needs to be rendered seamlessly into a real scene. This difficult task requires that the objects be lit consistently with the surfaces in their vicinity, and that the interplay of light between the objects and their surroundings be properly simulated. Specifically, the objects should cast shadows, appear in reflections, and refract, focus, and emit light just as real objects would.

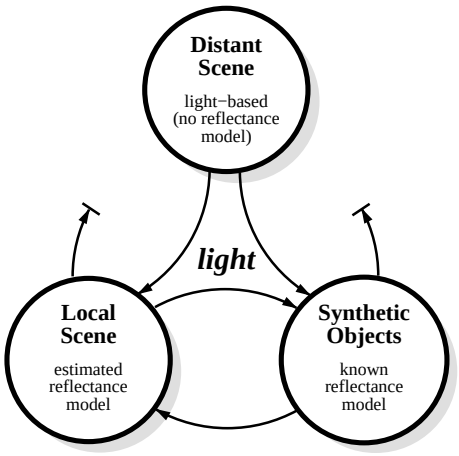


Figure 1: **The General Method** *In our method for adding synthetic objects into light-based scenes, the scene is partitioned into three components: the distant scene, the local scene, and the synthetic objects. Global illumination is used to simulate the interplay of light amongst all three components, except that light reflected back at the distant scene is ignored. As a result, BRDF information for the distant scene is unnecessary. Estimates of the geometry and material properties of the local scene are used to simulate the interaction of light between it and the synthetic objects.*

Currently available techniques for realistically rendering synthetic objects into scenes are labor intensive and not always successful. A common technique is to manually survey the positions of the light sources, and to instantiate a virtual light of equal color and intensity for each real light to illuminate the synthetic objects. Another technique is to photograph a reference object (such as a gray sphere) in the scene where the new object is to be rendered, and use its appearance as a qualitative guide in manually configuring the lighting environment. Lastly, the technique of reflection mapping is useful for mirror-like reflections. These methods typically require considerable hand-refinement and none of them easily simulates the effects of indirect illumination from the environment.

Accurately simulating the effects of both direct and indirect lighting has been the subject of research in global illumination. With a global illumination algorithm, if the entire scene were modeled with its full geometric and reflectance (BRDF) characteristics, one could correctly render a synthetic object into the scene simply by adding it to the model and recomputing the global illumination solution. Unfortunately, obtaining a full geometric and reflectance model of a large environment is extremeley difficult. Furthermore, global illumination solutions for large complex environments are extremely computationally intensive.

Moreover, it seems that having a full reflectance model of the large-scale scene should be unnecessary: under most circumstances, a new object will have no significant effect on the appearance of most of the of the distant scene. Thus, for such distant areas, knowing just its radiance (under the desired lighting conditions) should suffice.

Recently, [9] introduced a high dynamic range photographic technique that allows accurate measurements of scene radiance to be derived from a set of differently exposed photographs. This technique allows both low levels of indirect radiance from surfaces and high levels of direct radiance from light sources to be accurately recorded. When combined with image-based modeling techniques (e.g. [22, 24, 4, 10, 23, 17, 29]), and possibly active techniques for measuring geometry (e.g. [35, 30, 7, 27]) these derived radiance maps can be used to construct spatial representations of scene radiance.

We will use the term **light-based model** to refer to a representation of a scene that consists of radiance information, possibly with specific reference to light leaving surfaces, but not necessarily containing material property (BRDF) information. A light-based model can be used to evaluate the 5D plenoptic function [1] $P(\theta, \phi, V_x, V_y, V_z)$ for a given virtual or real subset of space[1]. A material-based model is converted to a light-based model by computing an illumination solution for it. A light-based model is differentiated from an image-based model in that its light values are actual measures of radiance[2], whereas image-based models may contain pixel values already transformed and truncated by the response function of an image acquisition or synthesis process.

In this paper, we present a general method for using accurate measurements of scene radiance in conjunction with global illumination to realistically add new objects to light-based models. The synthetic objects may have arbitrary material properties and can be rendered with appropriate illumination in arbitrary lighting environments. Furthermore, the objects can correctly interact with the environment around them: they cast the appropriate shadows, they are properly reflected, they can reflect and focus light, and they exhibit appropriate diffuse interreflection. The method can be carried out with commonly available equipment and software.

In this method (see Fig. 1), the scene is partitioned into three components. The first is the distant scene, which is the visible part of the environment too remote to be perceptibly affected by the synthetic object. The second is the local scene, which is the part of the environment which will be significantly affected by the presence of the objects. The third component is the synthetic objects. Our approach uses global illumination to correctly simulate the interaction of light amongst these three elements, with the exception that light radiated toward the distant environment will not be considered in the calculation. As a result, the BRDF of the distant environment need not be known — the technique uses BRDF information only for the local scene and the synthetic objects. We discuss the challenges in estimating the BRDF of the local scene, and methods for obtaining usable approximations. We also present a differential rendering technique that produces perceptually accurate results even when the estimated BRDF is somewhat inaccurate.

We demonstrate the general method for the specific case of rendering synthetic objects into particular views of a scene (such as background plates) rather than into a general image-based model. In this method, a light probe is used to acquire a high dynamic range panoramic radiance map near the location where the object will be rendered. A simple example of a light probe is a camera aimed at a mirrored sphere, a configuration commonly used for acquiring environment maps. An approximate geometric model of the scene is created (via surveying, photogrammetry, or 3D scanning) and mapped with radiance values measured with the light probe. The distant scene, local scene, and synthetic objects are rendered with global illumination from the same point of view as the background plate, and the results are composited into the background plate with a differential rendering technique.

1.1 Overview

The rest of this paper is organized as follows. In the next section we discuss work related to this paper. Section 3 introduces the basic technique of using acquired maps of scene radiance to illuminate synthetic objects. Section 4 presents the general method we will use to render synthetic objects into real scenes. Section 5 describes a practical technique based on this method using a *light probe* to measure incident illumination. Section 6 presents a differential rendering technique for rendering the local environment with only an approximate description of its reflectance. Section 7 presents a simple method to approximately recover the diffuse reflectance characteristics of the local environment. Section 8 presents results obtained with the technique. Section 9 discusses future directions for this work, and we conclude in Section 10.

2 Background and Related Work

The practice of adding new objects to photographs dates to the early days of photography in the simple form of pasting a cut-out from one picture onto another. While the technique conveys the idea of the new object being in the scene, it usually fails to produce an image that as a whole is a believable photograph. Attaining such realism requires a number of aspects of the two images to match. First, the camera projections should be consistent, otherwise the object may seem too foreshortened or skewed relative to the rest of the picture. Second, the patterns of film grain and film response should match. Third, the lighting on the object needs to be consistent with other objects in the environment. Lastly, the object needs to cast realistic shadows and reflections on the scene. Skilled artists found that by giving these considerations due attention, synthetic objects could be painted into still photographs convincingly.

In optical film compositing, the use of object mattes to prevent particular sections of film from being exposed made the same sort of cut-and-paste compositing possible for moving images. However, the increased demands of realism imposed by the dynamic nature of film made matching camera positions and lighting even more critical. As a result, care was taken to light the objects appropriately for the scene into which they were to be composited. This would still not account for the objects casting shadows onto the scene, so often these were painted in by an artist frame by frame [13, 2, 28]. Digital film scanning and compositing [26] helped make this process far more efficient.

Work in global illumination [16, 19] has recently produced algorithms (e.g. [31]) and software (e.g. [33]) to realistically simulate lighting in synthetic scenes, including indirect lighting with both specular and diffuse reflections. We leverage this work in order to create realistic renderings.

Some work has been done on the specific problem of compositing objects into photography. [25] presented a procedure for ren-

[1] Time and wavelength dependence can be included to represent the general 7D plenoptic function as appropriate.

[2] In practice, the measures of radiance are with respect to a discrete set of spectral distributions such as the standard tristimulus model.

dering architecture into background photographs using knowledge of the sun position and measurements or approximations of the local ambient light. For diffuse buildings in diffuse scenes, the technique is effective. The technique of *reflection mapping* (also called *environment mapping*) [3, 18] produces realistic results for mirror-like objects. In reflection mapping, a panoramic image is rendered or photographed from the location of the object. Then, the surface normals of the object are used to index into the panoramic image by reflecting rays from the desired viewpoint. As a result, the shiny object appears to properly reflect the desired environment[3]. However, the technique is limited to mirror-like reflection and does not account for objects casting light or shadows on the environment.

A common visual effects technique for having synthetic objects cast shadows on an existing environment is to create an approximate geometric model of the environment local to the object, and then compute the shadows from the various light sources. The shadows can then be subtracted from the background image. In the hands of professional artists this technique can produce excellent results, but it requires knowing the position, size, shape, color, and intensity of each of the scene's light sources. Furthermore, it does not account for diffuse reflection from the scene, and light reflected by the objects onto the scene must be handled specially.

To properly model the interaction of light between the objects and the local scene, we pose the compositing problem as a global illumination computation as in [14] and [12]. As in this work, we apply the effect of the synthetic objects in the lighting solution as a differential update to the original appearance of the scene. In the previous work an approximate model of the entire scene and its original light sources is constructed; the positions and sizes of the light sources are measured manually. Rough methods are used to estimate diffuse-only reflectance characteristics of the scene, which are then used to estimate the intensities of the light sources. [12] additionally presents a method for performing fast updates of the illumination solution in the case of moving objects. As in the previous work, we leverage the basic result from incremental radiosity [6, 5] that making a small change to a scene does not require recomputing the entire solution.

3 Illuminating synthetic objects with real light

In this section we propose that computer-generated objects be lit by actual recordings of light from the scene, using global illumination. Performing the lighting in this manner provides a unified and physically accurate alternative to manually attempting to replicate incident illumination conditions.

Accurately recording light in a scene is difficult because of the high dynamic range that scenes typically exhibit; this wide range of brightness is the result of light sources being relatively concentrated. As a result, the intensity of a source is often two to six orders of magnitude larger than the intensity of the non-emissive parts of an environment. However, it is necessary to accurately record both the large areas of indirect light from the environment and the concentrated areas of direct light from the sources since both are significant parts of the illumination solution.

Using the technique introduced in [9], we can acquire correct measures of scene radiance using conventional imaging equipment. The images, called *radiance maps*, are derived from a series of images with different sensor integration times and a technique for computing and accounting for the imaging system response function f. We can use these measures to illuminate synthetic objects exhibiting arbitrary material properties.

Fig. 2 shows a high-dynamic range lighting environment with electric, natural, and indirect lighting. This environment was recorded by taking a full dynamic range photograph of a mirrored ball on a table (see Section 5). A digital camera was used to acquire a series images in one-stop exposure increments from $\frac{1}{4}$ to $\frac{1}{10000}$ second. The images were fused using the technique in [9].

The environment is displayed at three exposure levels (-0, -3.5, and -7.0 stops) to show its full dynamic range. Recovered RGB radiance values for several points in the scene and on the two major light sources are indicated; the color difference between the tungsten lamp and the sky is evident. A single low-dynamic range photograph would be unable to record the correct colors and intensities over the entire scene.

Fig. 3(a-e) shows the results of using this panoramic radiance map to synthetically light a variety of materials using the RADIANCE global illumination algorithm [33]. The materials are: (a) perfectly reflective, (b) rough gold, (c) perfectly diffuse gray material, (d) shiny green plastic, and (e) dull orange plastic. Since we are computing a full illumination solution, the objects exhibit self-reflection and shadows from the light sources as appropriate. Note that in (c) the protrusions produce two noticeable shadows of slightly different colors, one corresponding to the ceiling light and a softer shadow corresponding to the window.

The shiny plastic object in (d) has a 4 percent specular component with a Gaussian roughness of 0.04 [32]. Since the object's surface both blurs and attenuates the light with its rough specular component, the reflections fall within the dynamic range of our display device and the different colors of the light sources can be seen. In (e) the rough plastic diffuses the incident light over a much larger area.

To illustrate the importance of using high dynamic range radiance maps, the same renderings were produced using just one of the original photographs as the lighting environment. In this single image, similar in appearance to Fig. 2(a), the brightest regions had been truncated to approximately 2 percent of their true values. The rendering of the mirrored surface (f) appears similar to (a) since it is displayed in low-dynamic range printed form. Significant errors are noticeable in (g-j) since these materials blur the incident light. In (g), the blurring of the rough material darkens the light sources, whereas in (b) they remain saturated. Renderings (h-j) are very dark due to the missed light; thus we have brightened by a factor of eight on the right in order to make qualitative comparisons to (c-e) possible. In each it can be seen that the low-dynamic range image of the lighting environment fails to capture the information necessary to simulate correct color balance, shadows, and highlights.

Fig. 4 shows a collection of objects with different material properties illuminated by two different environments. A wide variety of light interaction between the objects and the environment can be seen. The (synthetic) mirrored ball reflects both the synthetic objects as well as the environment. The floating diffuse ball shows a subtle color shift along its right edge as it shadows itself from the windows and is lit primarily by the incandescent lamp in Fig. 4(a). The reflection of the environment in the black ball (which has a specular intensity of 0.04) shows the colors of the light sources, which are too bright to be seen in the mirrored ball. A variety of shadows, reflections, and focused light can be observed on the resting surface.

The next section describes how the technique of using radiance maps to illuminate synthetic objects can be extended to compute the proper photometric interaction of the objects with the scene. It also describes how high dynamic range photography and image-based modeling combine in a natural manner to allow the simulation of arbitrary (non-infinite) lighting environments.

4 The General Method

This section explains our method for adding new objects to light-based scene representations. As in Fig. 1, we partition our scene into three parts: the distant scene, the local scene, and the synthetic

[3]Using the surface normal indexing method, the object will not reflect itself. Correct self-reflection can be obtained through ray tracing.

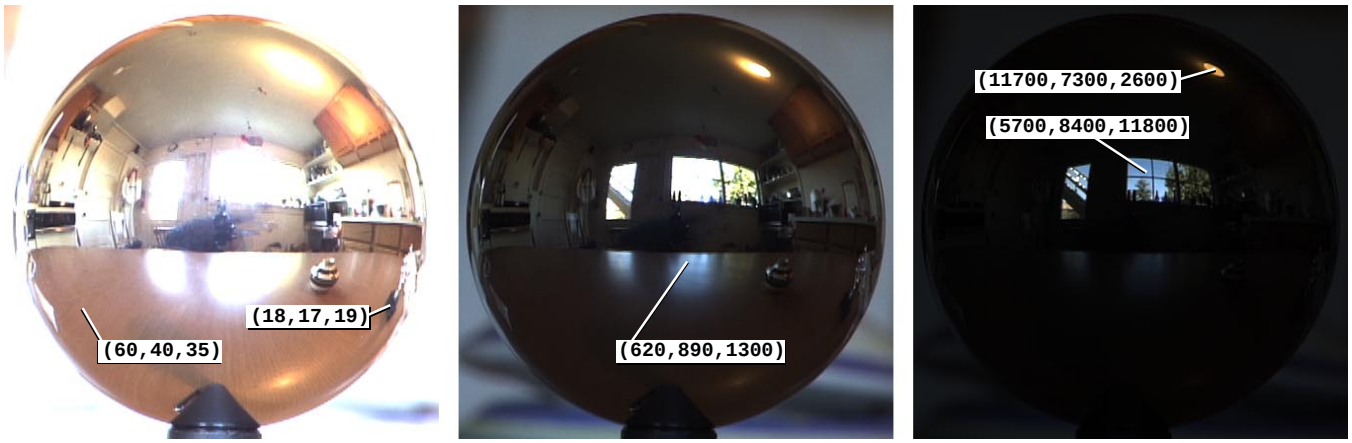


Figure 2: **An omnidirectional radiance map** This full dynamic range lighting environment was acquired by photographing a mirrored ball balanced on the cap of a pen sitting on a table. The environment contains natural, electric, and indirect light. The three views of this image adjusted to (a) +0 stops, (b) -3.5 stops, and (c) -7.0 stops show that the full dynamic range of the scene has been captured without saturation. As a result, the image usefully records the direction, color, and intensity of all forms of incident light.

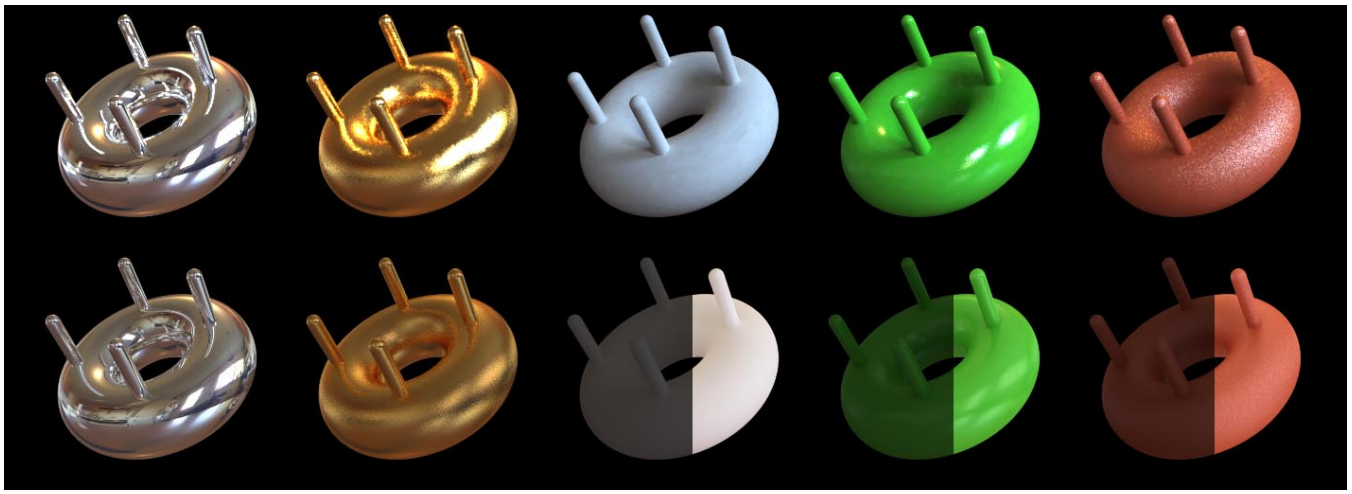


Figure 3: **Illuminating synthetic objects with real light (Top row: a,b,c,d,e)** *With full dynamic range measurements of scene radiance from Fig. 2.* **(Bottom row: f,g,h,i,j)** *With low dynamic range information from a single photograph of the ball. The right sides of images (h,i,j) have been brightened by a factor of six to allow qualitative comparison to (c,d,e). The high dynamic range measurements of scene radiance are necessary to produce proper lighting on the objects.*

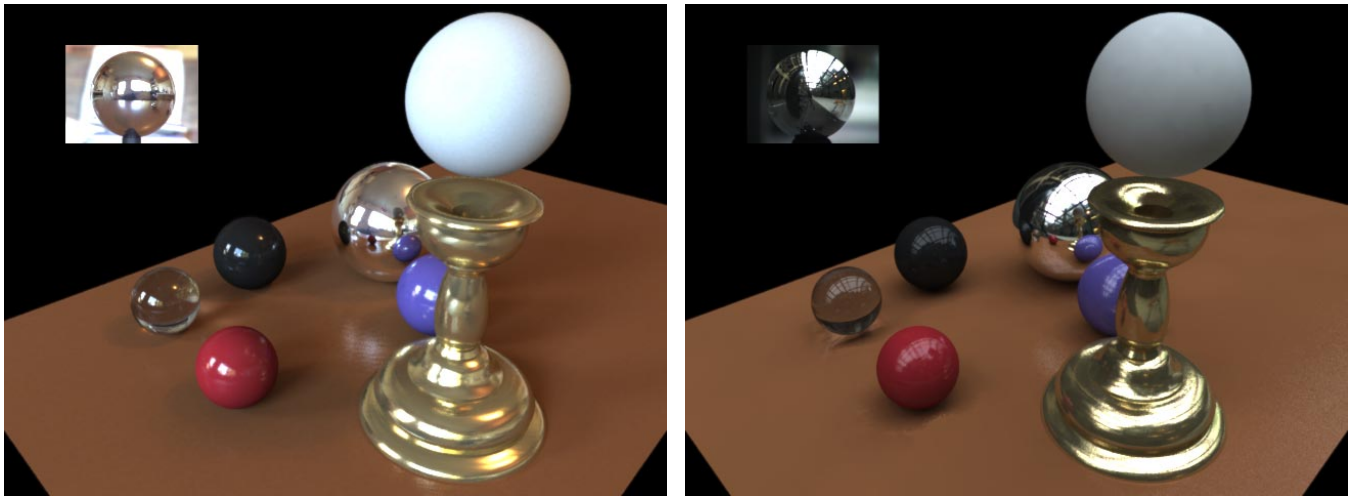


Figure 4: **Synthetic objects lit by two different environments (a)** *A collection of objects is illuminated by the radiance information in 2. The objects exhibit appropriate interreflection.* **(b)** *The same objects are illuminated by different radiance information obtained in an outdoor urban environment on an overcast day. The radiance map used for the illumination is shown in the upper left of each image. Candle holder model courtesy of Gregory Ward Larson.*

objects. We describe the geometric and photometric requirements for each of these components.

1. **A light-based model of the distant scene**
   The distant scene is constructed as a light-based model. The synthetic objects will receive light from this model, so it is necessary that the model store true measures of radiance rather than low dynamic range pixel values from conventional images. The light-based model can take on any form, using very little explicit geometry [23, 17], some geometry [24], moderate geometry [10], or be a full 3D scan of an environment with view-dependent texture-mapped [11] radiance. What is important is for the model to provide accurate measures of incident illumination in the vicinity of the objects, as well as from the desired viewpoint. In the next section we will present a convenient procedure for constructing a minimal model that meets these requirements.
   In the global illumination computation, the distant scene radiates light toward the local scene and the synthetic objects, but ignores light reflected back to it. We assume that no area of the distant scene will be significantly affected by light reflecting from the synthetic objects; if that were the case, the area should instead belong to the local scene, which contains the BRDF information necessary to interact with light. In the RADIANCE [33] system, this exclusively emissive behavior can be specified with the "glow" material property.
2. **An approximate material-based model of the local scene**
   The local scene consists of the surfaces that will photometrically interact with the synthetic objects. It is this geometry onto which the objects will cast shadows and reflect light. Since the local scene needs to fully participate in the illumination solution, both its geometry and reflectance characteristics should be known, at least approximately. If the geometry of the local scene is not readily available with sufficient accuracy from the light-based model of the distant scene, there are various techniques available for determining its geometry through active or passive methods. In the common case where the local scene is a flat surface that supports the synthetic objects, its geometry is determined easily from the camera pose. Methods for estimating the BRDF of the local scene are discussed in Section 7.
   Usually, the local scene will be the part of the scene that is geometrically close to the synthetic objects. When the local scene is mostly diffuse, the rendering equation shows that the visible effect of the objects on the local scene decreases as the inverse square of the distance between the two. Nonetheless, there is a variety of circumstances in which synthetic objects can significantly affect areas of the scene not in the immediate vicinity. Some common circumstances are:

   - If there are concentrated light sources illuminating the object, then the object can cast a significant shadow on a distant surface collinear with it and the light source.
   - If there are concentrated light sources and the object is flat and specular, it can focus a significant amount of light onto a distant part of the scene.
   - If a part of the distant scene is flat and specular (e.g. a mirror on a wall), its appearance can be significantly affected by a synthetic object.
   - If the synthetic object emits light (e.g. a synthetic laser), it can affect the appearance of the distant scene significantly.

   These situations should be considered in choosing which parts of the scene should be considered local and which parts distant. Any part of the scene that will be significantly affected in its appearance from the desired viewpoint should be included as part of the local scene.
   Since the local scene is a full BRDF model, it can be added to the global illumination problem as would any other object. The local scene may consist of any number of surfaces and objects with different material properties. For example, the local scene could consist of a patch of floor beneath the synthetic object to catch shadows as well as a mirror surface hanging on the opposite wall to catch a reflection. The local scene replaces the corresponding part of the light-based model of the distant scene.
   Since it can be difficult to determine the precise BRDF characteristics of the local scene, it is often desirable to have only the *change* in the local scene's appearance be computed with the BRDF estimate; its appearance due to illumination from the distant scene is taken from the original light-based model. This differential rendering method is presented in Section 6.
3. **Complete material-based models of the objects**
   The synthetic objects themselves may consist of any variety of shapes and materials supported by the global illumination software, including plastics, metals, emitters, and dielectrics such as glass and water. They should be placed in their desired geometric correspondence to the local scene.

Once the distant scene, local scene, and synthetic objects are properly modeled and positioned, the global illumination software can be used in the normal fashion to produce renderings from the desired viewpoints.

5 Compositing using a light probe

This section presents a particular technique for constructing a light-based model of a real scene suitable for adding synthetic objects at a particular location. This technique is useful for compositing objects into actual photography of a scene.

In Section 4, we mentioned that the light-based model of the distant scene needs to appear correctly in the vicinity of the synthetic objects as well as from the desired viewpoints. This latter requirement can be satisfied if it is possible to directly acquire radiance maps of the scene from the desired viewpoints. The former requirement, that the appear photometrically correct in all directions in the vicinity of the synthetic objects, arises because this information comprises the incident light which will illuminate the objects.

To obtain this part of the light-based model, we acquire a full dynamic range omnidirectional radiance map near the location of the synthetic object or objects. One technique for acquiring this radiance map is to photograph a spherical first-surface mirror, such as a polished steel ball, placed at or near the desired location of the synthetic object[4]. This procedure is illustrated in Fig. 7(a). An actual radiance map obtained using this method is shown in Fig. 2.

The radiance measurements observed in the ball are mapped onto the geometry of the distant scene. In many circumstances this model can be very simple. In particular, if the objects are small and resting on a flat surface, one can model the scene as a horizontal plane for the resting surface and a large dome for the rest of the environment. Fig. 7(c) illustrates the ball image being mapped onto a table surface and the walls and ceiling of a finite room; 5 shows the resulting light-based model.

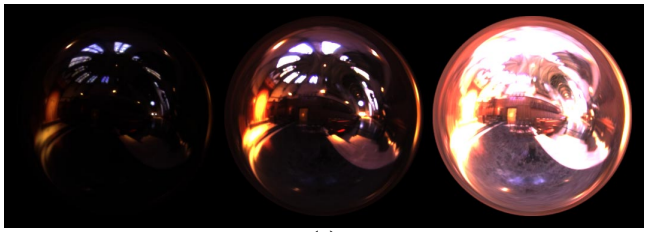
5.1 Mapping from the probe to the scene model

To precisely determine the mapping between coordinates on the ball and rays in the world, one needs to record the position of the ball

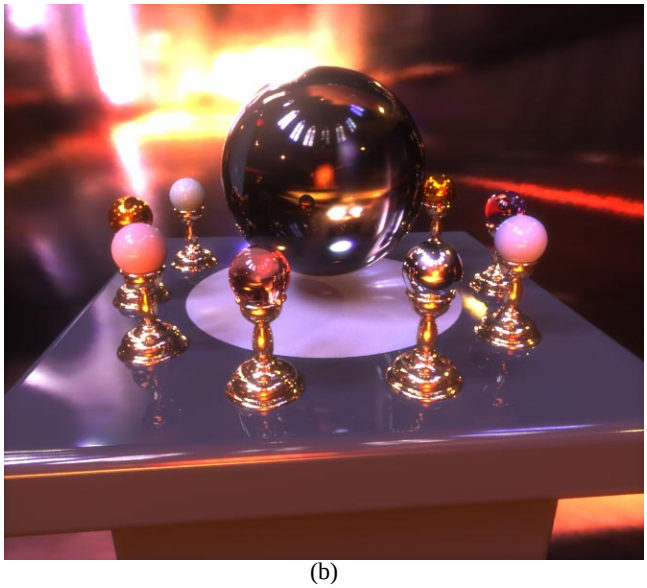
[4]Parabolic mirrors combined with telecentric lenses [34] can be used to obtain hemispherical fields of view with a consistent principal point, if so desired.

relative to the camera, the size of the ball, and the camera parameters such as its location in the scene and focal length. With this information, it is straightforward to trace rays from the camera center through the pixels of the image, and reflect rays off the ball into the environment. Often a good approximation results from assuming the ball is small relative to the environment and that the camera's view is orthographic.

The data acquired from a single ball image will exhibit a number of artifacts. First, the camera (and possibly the photographer) will be visible. The ball, in observing the scene, interacts with it: the ball (and its support) can appear in reflections, cast shadows, and can reflect light back onto surfaces. Lastly, the ball will not reflect the scene directly behind it, and will poorly sample the area nearby. If care is taken in positioning the ball and camera, these effects can be minimized and will have a negligible effect on the final renderings. If the artifacts are significant, the images can be fixed manually in image editing program or by selectively combining images of the ball taken from different directions; Fig. 6 shows a relatively artifact-free enviroment constructed using the latter method. We have found that combining two images of the ball taken ninety degrees apart from each other allows us to eliminate the camera's appearance and to avoid poor sampling.



(a)



(b)

Figure 6: **Rendering with a Combined Probe Image** *The full dynamic range environment map shown at the top was assembled from two light probe images taken ninety degrees apart from each other. As a result, the only visible artifact is small amount of the probe support visible on the floor. The map is shown at -4.5, 0, and +4.5 stops. The bottom rendering was produced using this lighting information, and exhibits diffuse and specular reflections, shadows from different sources of light, reflections, and caustics.*

5.2 Creating renderings

To render the objects into the scene, a synthetic local scene model is created as described in Section 4. Images of the scene from the desired viewpoint(s) are taken (Fig. 7(a)), and their position relative to the scene is recorded through pose-instrumented cameras or (as in our work) photogrammetry. The location of the ball in the scene is also recorded at this time. The global illumination software is then run to render the objects, local scene, and distant scene from the desired viewpoint (Fig. 7(d)).

The objects and local scene are then composited onto the background image. To perform this compositing, a mask is created by rendering the objects and local scene in white and the distant scene in black. If objects in the distant scene (which may appear in front of the objects or local scene from certain viewpoints) are geometrically modeled, they will properly obscure the local scene and the objects as necessary. This compositing can be considered as a subset of the general method (Section 4) wherein the light-based model of the distant scene acts as follows: if (V_x, V_y, V_z) corresponds to an actual view of the scene, return the radiance value looking in direction (θ, ϕ). Otherwise, return the radiance value obtained by casting the ray $(\theta, \phi, V_x, V_y, V_z)$ onto the radiance-mapped distant scene model.

In the next section we describe a more robust method of compositing the local scene into the background image.

6 Improving quality with differential rendering

The method we have presented so far requires that the local scene be modeled accurately in both its geometry and its spatially varying material properties. If the model is inaccurate, the appearance of the local scene will not be consistent with the appearance of adjacent distant scene. Such a border is readily apparent in Fig. 8(c), since the local scene was modeled with a homogeneous BRDF when in reality it exhibits a patterned albedo (see [21]). In this section we describe a method for greatly reducing such effects.

Suppose that we compute a global illumination solution for the local and distant scene models without including the synthetic objects. If the BRDF and geometry of the local scene model were perfectly accurate, then one would expect the appearance of the rendered local scene to be consistent with its appearance in the light-based model of the entire scene. Let us call the appearance of the local scene from the desired viewpoint in the light-based model LS_b. In the context of the method described in Section 5, LS_b is simply the background image. We will let LS_{noobj} denote the appearance of the local scene, without the synthetic objects, as calculated by the global illumination solution. The error in the rendered local scene (without the objects) is thus: $Err_{ls} = LS_{noobj} - LS_b$. This error results from the difference between the BRDF characteristics of the actual local scene as compared to the modeled local scene.

Let LS_{obj} denote the appearance of the local environment as calculated by the global illumination solution with the synthetic objects in place. We can compensate for the error if we compute our final rendering LS_{final} as:

$$LS_{final} = LS_{obj} - Err_{ls}$$

Equivalently, we can write:

$$LS_{final} = LS_b + (LS_{obj} - LS_{noobj})$$

In this form, we see that whenever LS_{obj} and LS_{noobj} are the same (i.e. the addition of the objects to the scene had no effect on the local scene) the final rendering of the local scene is equivalent to LS_b (e.g. the background plate). When LS_{obj} is darker than LS_{noobj}, light is subtracted from the background to form shadows,

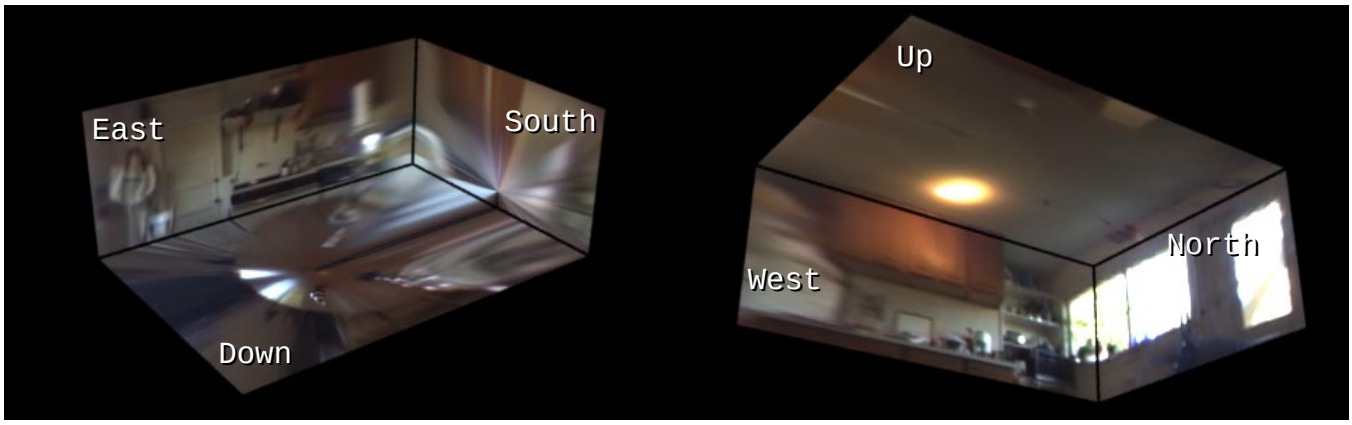


Figure 5: **A Light-Based Model** *A simple light-based model of a room is constructed by mapping the image from a light probe onto a box. The box corresponds to the upper half of the room, with the bottom face of the box being coincident with the top of the table. The model contains the full dynamic range of the original scene, which is not reproduced in its entirety in this figure.*

and when LS_{obj} is lighter than LS_{noobj} light is added to the background to produce reflections and caustics.

Stated more generally, the appearance of the local scene without the objects is computed with the correct reflectance characteristics lit by the correct environment, and the change in appearance due to the presence of the synthetic objects is computed with the modeled reflectance characteristics as lit by the modeled environment. While the realism of LS_{final} still benefits from having a good model of the reflectance characteristics of the local scene, the perceptual effect of small errors in albedo or specular properties is considerably reduced. Fig. 8(g) shows a final rendering in which the local environment is computed using this differential rendering technique. The objects are composited into the image directly from the LS_{obj} solution shown in Fig. 8(c).

It is important to stress that this technique can still produce abitrarily wrong results depending on the amount of error in the estimated local scene BRDF and the inaccuracies in the light-based model of the distance scene. In fact, Err_{ls} may be larger than LS_{obj}, causing LS_{final} to be negative. An alternate approach is to compensate for the *relative* error in the appearance of the local scene: $LS_{final} = LS_b(LS_{obj}/LS_{noobj})$. Inaccuracies in the local scene BDRF will also be reflected in the objects.

In the next section we discuss techniques for estimating the BRDF of the local scene.

7 Estimating the local scene BRDF

Simulating the interaction of light between the local scene and the synthetic objects requires a model of the reflectance characteristics of the local scene. Considerable recent work [32, 20, 8, 27] has presented methods for measuring the reflectance properties of materials through observation under controlled lighting configurations. Furthermore, reflectance characteristics can also be measured with commercial radiometric devices.

It would be more convenient if the local scene reflectance could be estimated directly from observation. Since the light-based model contains information about the radiance of the local scene as well as its irradiance, it actually contains information about the local scene reflectance. If we hypothesize reflectance characteristics for the local scene, we can illuminate the local scene with its known irradiance from the light-based model. If our hypothesis is correct, then the appearance should be consistent with the measured appearance. This suggests the following iterative method for recovering the reflectance properties of the local scene:

1. Assume a reflectance model for the local scene (e.g. diffuse only, diffuse + specular, metallic, or arbitrary BRDF, including spatial variation)
2. Choose approximate initial values for the parameters of the reflectance model
3. Compute a global illumination solution for the local scene with the current parameters using the observed lighting configuration or configurations.
4. Compare the appearance of the rendered local scene to its actual appearance in one or more views.
5. If the renderings are not consistent, adjust the parameters of the reflectance model and return to step 3.

Efficient methods of performing the adjustment in step 5 that exploit the properties of particular reflectance models are left as future work. However, assuming a diffuse-only model of the local scene in step 1 makes the adjustment in step 5 straightforward. We have:

$$L_{r1}(\theta_r, \phi_r) = \int_0^{2\pi} \int_0^{\pi/2} \rho_d L_i(\theta_i, \phi_i) \cos\theta_i \sin\theta_i \, d\theta_i \, d\phi_i =$$

$$\rho_d \int_0^{2\pi} \int_0^{\pi/2} L_i(\theta_i, \phi_i) \cos\theta_i \sin\theta_i \, d\theta_i \, d\phi_i$$

If we initialize the local scene to be perfectly diffuse ($\rho_d = 1$) everywhere, we have:

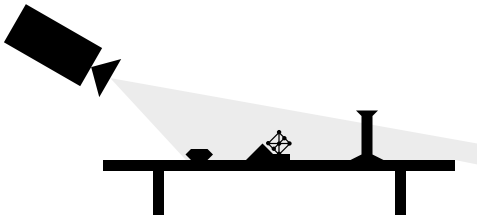
$$L_{r2}(\theta_r, \phi_r) = \int_0^{2\pi} \int_0^{\pi/2} L_i(\theta_i, \phi_i) \cos\theta_i \sin\theta_i \, d\theta_i \, d\phi_i$$

The updated diffuse reflectance coefficient for each part of the local scene can be computed as:

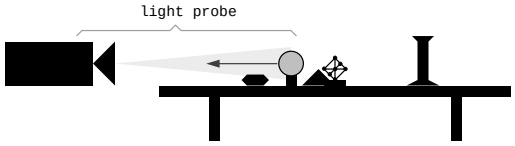
$$\rho_d' = \frac{L_{r1}(\theta_r, \phi_r)}{L_{r2}(\theta_r, \phi_r)}$$

In this manner, we use the global illumination calculation to render each patch as a perfectly diffuse reflector, and compare the resulting radiance to the observed value. Dividing the two quantities yields the next estimate of the diffuse reflection coefficient ρ_d'. If there is no interreflection within the local scene, then the ρ_d' estimates will make the renderings consistent. If there is interreflection, then the algorithm should be iterated until there is convergence.

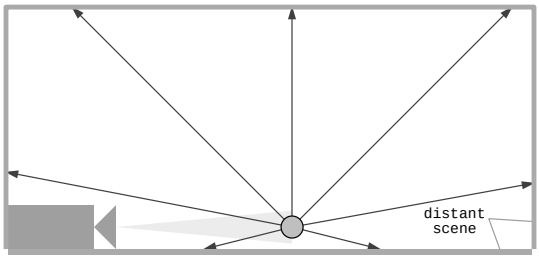
For a trichromatic image, the red, green, and blue diffuse reflectance values are computed independently. The diffuse characteristics of the background material used to produce Fig. 8(c) were



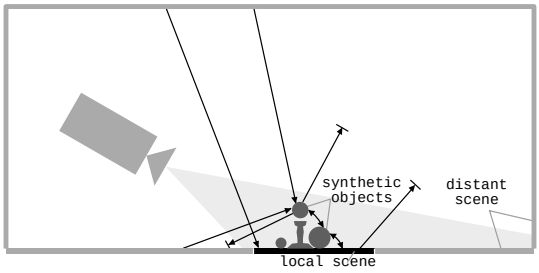
(a) Acquiring the background photograph



(b) Using the light probe



(c) Constructing the light-based model



(d) Computing the global illumination solution

Figure 7: **Using a light probe (a)** *The background plate of the scene (some objects on a table) is taken.* **(b)** *A light probe (in this case, the camera photographing a steel ball) records the incident radiance near the location of where the synthetic objects are to be placed.* **(c)** *A simplified light-based model of the distant scene is created as a planar surface for the table and a finite box to represent the rest of the room. The scene is texture-mapped in high dynamic range with the radiance map from the light probe. The objects on the table, which were not explicitly modeled, become projected onto the table.* **(d)** *Synthetic objects and a BRDF model of the local scene are added to the light-based model of the distant scene. A global illumination solution of this configuration is computed with light coming from the distant scene and interacting with the local scene and synthetic objects. Light reflected back to the distant scene is ignored. The results of this rendering are composited (possibly with differential rendering) into the background plate from (a) to achieve the final result.*

computed using this method, although it was assumed that the entire local scene had the same diffuse reflectance.

In the standard "plastic" illumination model, just two more coefficients – those for specular intensity and roughness – need to be specified. In Fig. 8, the specular coefficients for the local scene were estimated manually based on the specular reflection of the window in the table in Fig. 2.

8 Compositing Results

Fig. 5 shows a simple light-based model of a room constructed using the panoramic radiance map from Fig. 2. The room model begins at the height of the table and continues to the ceiling; its measurements and the position of the ball within it were measured manually. The table surface is visible on the bottom face. Since the room model is finite in size, the light sources are effectively local rather than infinite. The stretching on the south wall is due to the poor sampling toward the silhouette edge of the ball.

Figs. 4 and 6 show complex arrangements of synthetic objects lit entirely by a variety of light-based models. The selection and composition of the objects in the scene was chosen to exhibit a wide variety of light interactions, including diffuse and specular reflectance, multiple soft shadows, and reflected and focused light. Each rendering was produced using the RADIANCE system with two diffuse light bounces and a relatively high density of ambient sample points.

Fig. 8(a) is a background plate image into which the synthetic objects will be rendered. In 8(b) a calibration grid was placed on the table in order to determine the camera pose relative to the scene and to the mirrored ball, which can also be seen. The poses were determined using the photogrammetric method in [10]. In 8(c), a model of the local scene as well as the synthetic objects is geometrically matched and composited onto the background image. Note that the local scene, while the same average color as the table, is readily distinguishable at its edges and because it lacks the correct variations in albedo.

Fig. 8(d) shows the results of lighting the local scene model with the light-based model of the room, without the objects. This image will be compared to 8(c) in order to determine the effect the synthetic objects have on the local scene. Fig. 8(e) is a mask image in which the white areas indicate the location of the synthetic objects. If the distant or local scene were to occlude the objects, such regions would be dark in this image.

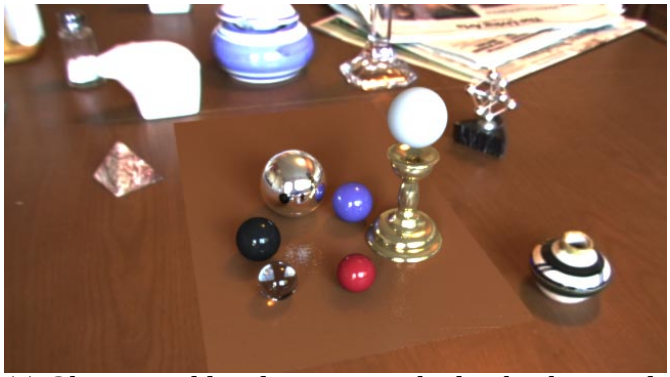
Fig. 8(f) shows the difference between the appearance of the local scene rendered with (8(c)) and without (8(d)) the objects. For illustration purposes, the difference in radiance values have been offset so that zero difference is shown in gray. The objects have been masked out using image 8(e). This difference image encodes both the shadowing (dark areas) and reflected and focussed light (light areas) imposed on the local scene by the addition of the synthetic objects.

Fig. 8(g) shows the final result using the differential rendering method described in Section 6. The synthetic objects are copied directly from the global illumination solution 8(c) using the object mask 8(e). The effects the objects have on the local scene are included by adding the difference image 8(f) (without offset) to the background image. The remainder of the scene is copied directly from the background image 8(a). Note that in the mirror ball's reflection, the modeled local scene can be observed without the effects of differential rendering — a limitation of the compositing technique.

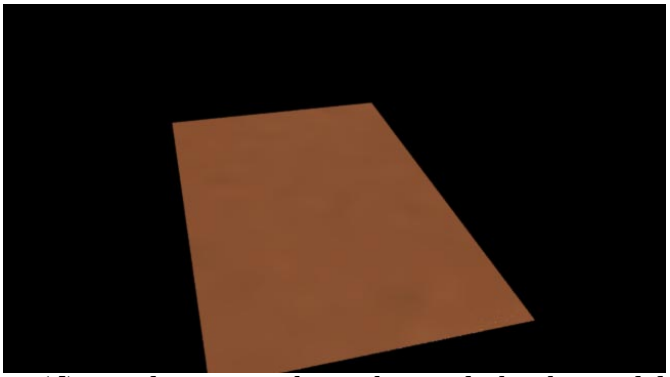
In this final rendering, the synthetic objects exhibit a consistent appearance with the real objects present in the background image 8(a) in both their diffuse and specular shading, as well as the direction and coloration of their shadows. The somewhat speckled nature of the object reflections seen in the table surface is due to

(a) Background photograph

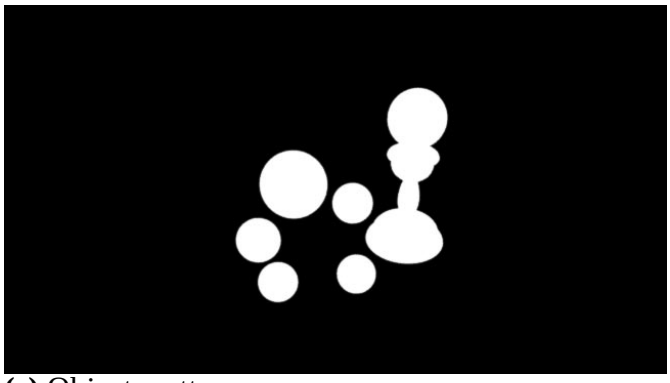
(b) Camera calibration grid and light probe



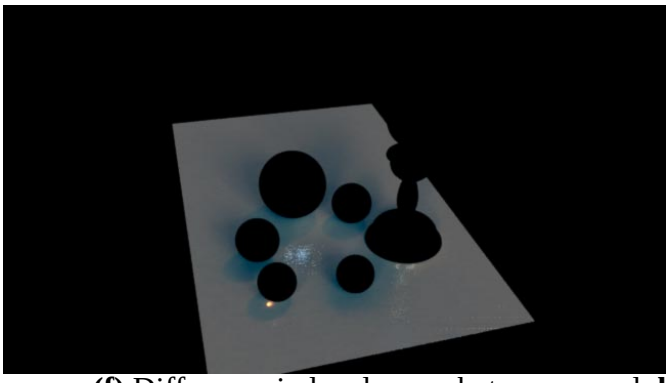
(c) Objects and local scene matched to background



(d) Local scene, without objects, lit by the model



(e) Object matte



(f) Difference in local scene between **c** and **d**

(g) Final result with differential rendering

Figure 8: Compositing synthetic objects into a real scene using a light probe and differential rendering

the stochastic nature of the particular global illumination algorithm used.

The differential rendering technique successfully eliminates the border between the local scene and the background image seen in 8(c). Note that the albedo texture of the table in the local scene area is preserved, and that a specular reflection of a background object on the table (appearing just to the left of the floating sphere) is correctly preserved in the final rendering. The local scene also exhibits reflections from the synthetic objects. A caustic from the glass ball focusing the light of the ceiling lamp onto the table is evident.

9 Future work

The method proposed here suggests a number of areas for future work. One area is to investigate methods of automatically recovering more general reflectance models for the local scene geometry, as proposed in Section 7. With such information available, the program might also also be able to suggest which areas of the scene should be considered as part of the local scene and which can safely be considered distant, given the position and reflectance characteristics of the desired synthetic objects.

Some additional work could be done to allow the global illumination algorithm to compute the ilumination solution more efficiently. One technique would be to have an algorithm automatically locate and identify concentrated light sources in the light-based model of the scene. With such knowledge, the algorithm could compute most of the direct illumination in a forward manner, which could dramatically increase the efficiency with which an accurate solution could be calculated. To the same end, use of the method presented in [15] to expedite the solution could be investigated. For the case of compositing moving objects into scenes, greatly increased efficiency could be obtained by adapting incremental radiosity methods to the current framework.

10 Conclusion

We have presented a general framework for adding new objects to light-based models with correct illumination. The method leverages a technique of using high dynamic range images of real scene radiance to synthetically illuminate new objects with arbitrary reflectance characteristics. We leverage this technique in a general method to simulate interplay of light between synthetic objects and the light-based environment, including shadows, reflections, and caustics. The method can be implemented with standard global illumination techniques.

For the particular case of rendering synthetic objects into real scenes (rather than general light-based models), we have presented a practical instance of the method that uses a light probe to record incident illumination in the vicinity of the synthetic objects. In addition, we have described a differential rendering technique that can convincingly render the interplay of light between objects and the local scene when only approximate reflectance information for the local scene is available. Lastly, we presented an iterative approach for determining reflectance characteristics of the local scene based on measured geometry and observed radiance in uncontrolled lighting conditions. It is our hope that the techniques presented here will be useful in practice as well as comprise a useful framework for combining material-based and light-based graphics.

Acknowledgments

The author wishes to thank Chris Bregler, David Forsyth, Jianbo Shi, Charles Ying, Steve Chenney, and Andrean Kalemis for the various forms of help and advice they provided. Special gratitude is also due to Jitendra Malik for helping make this work possible. Discussions with Michael Naimark and Steve Saunders helped motivate this work. Tim Hawkins provided extensive assistance on improving and revising this paper and provided invaluable assistance with image acquisition. Gregory Ward Larson deserves great thanks for the RADIANCE lighting simulation system and his invaluable assistance and advice in using RADIANCE in this research, for assisting with reflectance measurements, and for very helpful comments and suggestions on the paper. This research was supported by a Multidisciplinary University Research Initiative on three dimensional direct visualization from ONR and BMDO, grant FDN00014-96-1-1200.

References

[1] ADELSON, E. H., AND BERGEN, J. R. *Computational Models of Visual Processing*. MIT Press, Cambridge, Mass., 1991, ch. 1. The Plenoptic Function and the Elements of Early Vision.

[2] AZARMI, M. *Optical Effects Cinematography: Its Development, Methods, and Techniques*. University Microfilms International, Ann Arbor, Michigan, 1973.

[3] BLINN, J. F. Texture and reflection in computer generated images. *Communications of the ACM 19*, 10 (October 1976), 542–547.

[4] CHEN, E. QuickTime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH '95* (1995).

[5] CHEN, S. E. Incremental radiosity: An extension of progressive radiosity to an interactive synthesis system. In *SIGGRAPH '90* (1990), pp. 135–144.

[6] COHEN, M. F., CHEN, S. E., WALLACE, J. R., AND GREENBERG, D. P. A progressive refinement approach to fast radiosity image generation. In *SIGGRAPH '88* (1988), pp. 75–84.

[7] CURLESS, B., AND LEVOY, M. A volumetric method for building complex models from range images. In *SIGGRAPH '96* (1996), pp. 303–312.

[8] DANA, K. J., GINNEKEN, B., NAYAR, S. K., AND KOENDERINK, J. J. Reflectance and texture of real-world surfaces. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.* (1997), pp. 151–157.

[9] DEBEVEC, P. E., AND MALIK, J. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH '97* (August 1997), pp. 369–378.

[10] DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96* (August 1996), pp. 11–20.

[11] DEBEVEC, P. E., YU, Y., AND BORSHUKOV, G. D. Efficient view-dependent image-based rendering with projective texture-mapping. Tech. Rep. UCB//CSD-98-1003, University of California at Berkeley, 1998.

[12] DRETTAKIS, G., ROBERT, L., AND BOUGNOUX, S. Interactive common illumination for computer augmented reality. In *8th Eurographics workshop on Rendering, St. Etienne, France* (May 1997), J. Dorsey and P. Slusallek, Eds., pp. 45–57.

[13] FIELDING, R. *The Technique of Special Effects Cinematography*. Hastings House, New York, 1968.

[14] FOURNIER, A., GUNAWAN, A., AND ROMANZIN, C. Common illumination between real and computer generated scenes. In *Graphics Interface* (May 1993), pp. 254–262.

[15] GERSHBEIN, R., SCHRODER, P., AND HANRAHAN, P. Textures and radiosity: Controlling emission and reflection with texture maps. In *SIGGRAPH '94* (1994).

[16] GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84* (1984), pp. 213–222.

[17] GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. The Lumigraph. In *SIGGRAPH '96* (1996), pp. 43–54.

[18] HECKBERT, P. S. Survey of texture mapping. *IEEE Computer Graphics and Applications 6*, 11 (November 1986), 56–67.

[19] KAJIYA, J. The rendering equation. In *SIGGRAPH '86* (1986), pp. 143–150.

[20] KARNER, K. F., MAYER, H., AND GERVAUTZ, M. An image based measurement system for anisotropic reflection. In *EUROGRAPHICS Annual Conference Proceedings* (1996).

[21] KOENDERINK, J. J., AND VAN DOORN, A. J. Illuminance texture due to surface mesostructure. *J. Opt. Soc. Am. 13*, 3 (1996).

[22] LAVEAU, S., AND FAUGERAS, O. 3-D scene representation as a collection of images. In *Proceedings of 12th International Conference on Pattern Recognition* (1994), vol. 1, pp. 689–691.

[23] LEVOY, M., AND HANRAHAN, P. Light field rendering. In *SIGGRAPH '96* (1996), pp. 31–42.

[24] MCMILLAN, L., AND BISHOP, G. Plenoptic Modeling: An image-based rendering system. In *SIGGRAPH '95* (1995).

[25] NAKAMAE, E., HARADA, K., AND ISHIZAKI, T. A montage method: The overlaying of the computer generated images onto a background photograph. In *SIGGRAPH '86* (1986), pp. 207–214.

[26] PORTER, T., AND DUFF, T. Compositing digital images. In *SIGGRAPH 84* (July 1984), pp. 253–259.

[27] SATO, Y., WHEELER, M. D., AND IKEUCHI, K. Object shape and reflectance modeling from observation. In *SIGGRAPH '97* (1997), pp. 379–387.

[28] SMITH, T. G. *Industrial Light and Magic: The Art of Special Effects*. Ballantine Books, New York, 1986.

[29] SZELISKI, R. Image mosaicing for tele-reality applications. In *IEEE Computer Graphics and Applications* (1996).

[30] TURK, G., AND LEVOY, M. Zippered polygon meshes from range images. In *SIGGRAPH '94* (1994), pp. 311–318.

[31] VEACH, E., AND GUIBAS, L. J. Metropolis light transport. In *SIGGRAPH '97* (August 1997), pp. 65–76.

[32] WARD, G. J. Measuring and modeling anisotropic reflection. In *SIGGRAPH '92* (July 1992), pp. 265–272.

[33] WARD, G. J. The radiance lighting simulation and rendering system. In *SIGGRAPH '94* (July 1994), pp. 459–472.

[34] WATANABE, M., AND NAYAR, S. K. Telecentric optics for computational vision. In *Proceedings of Image Understanding Workshop (IUW 96)* (February 1996).

[35] Y.CHEN, AND MEDIONI, G. Object modeling from multiple range images. *Image and Vision Computing 10*, 3 (April 1992), 145–155.

Inverse Global Illumination: Recovering Reflectance Models of Real Scenes from Photographs

Yizhou Yu, Paul Debevec, Jitendra Malik, and **Tim Hawkins** *

Computer Science Division
University of California at Berkeley

ABSTRACT

In this paper we present a method for recovering the reflectance properties of all surfaces in a real scene from a sparse set of photographs, taking into account both direct and indirect illumination. The result is a lighting-independent model of the scene's geometry and reflectance properties, which can be rendered with arbitrary modifications to structure and lighting via traditional rendering methods. Our technique models reflectance with a low-parameter reflectance model, and allows diffuse albedo to vary arbitrarily over surfaces while assuming that non-diffuse characteristics remain constant across particular regions. The method's input is a geometric model of the scene and a set of calibrated high dynamic range photographs taken with known direct illumination. The algorithm hierarchically partitions the scene into a polygonal mesh, and uses image-based rendering to construct estimates of both the radiance and irradiance of each patch from the photographic data. The algorithm computes the expected location of specular highlights, and then analyzes the highlight areas in the images by running a novel iterative optimization procedure to recover the diffuse and specular reflectance parameters for each region. Lastly, these parameters are used in constructing high-resolution diffuse albedo maps for each surface.

The algorithm has been applied to both real and synthetic data, including a synthetic cubical room and a real meeting room. Re-renderings are produced using a global illumination system under both original and novel lighting, and with the addition of synthetic objects. Side-by-side comparisons show success at predicting the appearance of the scene under novel lighting conditions.

CR Categories: I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—modeling and recovery of physical attributes I.3.7 [**Computer Graphics**]: Three-dimensional Graphics and Realism—color, shading, shadowing, and texture I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—Radiosity I.4.8 [**Image Processing**]: Scene Analysis—Color, photometry, shading

Keywords: Global Illumination, Image-Based Modeling and Rendering, BRDF Models, Reflectance Recovery, Albedo Maps, Radiance, Radiosity, Rendering

*Email:{yyz,debevec,malik,tsh}@cs.Berkeley.edu, yizhouy@acm.org, Website: http://www.cs.berkeley.edu/∼{yyz,debevec,malik,tsh}

1 Introduction

Computer graphics is being increasingly used to visualize real objects and environments. Applications in entertainment, architecture, interior design, virtual reality, and digital museums often require that aspects of the real world be rendered realistically from novel viewpoints and/or under novel illumination. For example, one would want to see how a room in a house would look like with different lighting, or how a statue would look at various times of day in a different wing of a museum. Lastly, one might want to realistically render a film location in different lighting, and add in digital props and characters, with the expectation that the rendered results would be the same as what would have happened had it all been for real.

Work in image-based modeling and rendering e.g. [18, 3, 22, 19, 12, 9, 6, 29]) has shown that photographs of a scene can be used along with geometry to produce realistic renderings of diffuse scenes under the original lighting conditions. However, challenges remain in making modifications to such scenes. Whether it is changing the geometry or changing the lighting, generating a new rendering requires re-computing the interaction of light with the surfaces in the scene. Computing this interaction requires knowing the reflectance properties (diffuse color, shininess, etc.) of each surface. Unfortunately, such reflectance property information is not directly available from the scene geometry or from photographs.

Considerable work (e.g. [32, 16, 5, 27, 21]) has been done to estimate reflectance properties of real surfaces in laboratory settings from a dense set of measurements. However, reflectance properties of real scenes are usually spatially varying, and typically change with use and age, making *a priori* laboratory measurements impractical. It would clearly be preferable to estimate the reflectance properties of an entire scene at once, with the surfaces being illuminated *in situ* rather than as isolated samples, and from a relatively sparse set of photographs. This is difficult for two reasons.

The first is that we wish to use only a sparse set of photographs of the scene, rather than exhaustively photographing every point of every surface from a dense set of angles. With such a set of photographs, we can expect to observe each surface point from only a small number of angles. As a result, there will be too little data to determine fully general bi-directional reflectance distribution functions (BRDFs) for each surface. We address this problem in two ways. First, we limit ourselves to recovering low-parameter reflectance models of the surfaces in the scene. Second, we assume that the scene can be decomposed into areas with related reflectance properties. Specifically, we allow the diffuse reflectance, or *albedo*, of the object to vary arbitrarily over any surface; the estimated albedo is computed as an image called an *albedo map*[1]. In contrast, we require that the directional reflectance properties (such as specular reflectance and roughness) remain constant over each area. In this work, such areas are specified as part of the geometry

[1]The commonly used term *texture map* is sometimes used to refer to this same concept. However, texture maps are also sometimes used to store surface radiance information, which is not lighting-independent.

recovery process.

The second problem we face is that in a real scene, surfaces will exhibit mutual illumination. Thus, the light that any particular surface receives will arrive not just from the light sources, but also from the rest of the environment through indirect illumination. As a result, the incident radiance of an observed surface is a complex function of the light sources, the geometry of the scene, and the as-yet-undetermined reflectance properties of all of the scene's surfaces. In this work, we use radiance data from photographs and image-based rendering to estimate the incident radiances of surfaces in the scene. This allows us to estimate the reflectance properties of the surfaces in the scene via an iterative optimization procedure, which allows us to re-estimate the incident radiances. We refer to this procedure as *inverse global illumination*.

Addressing these two problems makes it possible to robustly recover reflectance parameters from the limited radiance information present in a sparse set of photographs, and the accommodations made are appropriate for a wide variety of real scenes. Even when they are not met, the algorithm will compute the reflectance property parameters that best fit the observed image data, which in many cases can still yield a visually acceptable result.

The input to our algorithm is a geometric model of the scene, a set of radiance maps taken under known direct illumination, and a partitioning of the scene into areas of similar non-diffuse reflectance properties. The algorithm outputs a set of high-resolution albedo maps for the surfaces in the scene along with their specular reflectance properties, yielding a traditional material-based model. This output is readily used as input to traditional rendering algorithms to realistically render the scene under arbitrary lighting conditions. Moreover, modifications to the scene's lighting and geometry and the addition of synthetic objects is easily accomplished using conventional modeling methods.

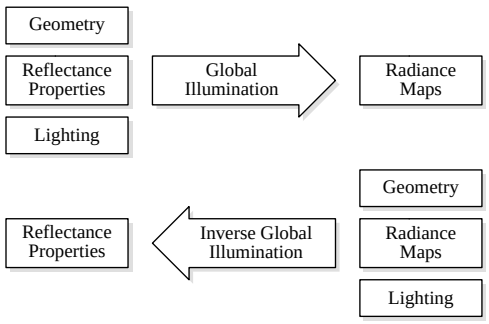


Figure 1: **Overview of the Method** *This figure shows the relationship between global illumination and inverse global illumination. Global illumination uses geometry, lighting, and reflectance properties to compute radiance maps (i.e. rendered images), and inverse global illumination uses geometry, lighting, and radiance maps to determine reflectance properties.*

1.1 Overview

The rest of this paper is organized as follows. In the next section we discuss work related to this paper. Section 3 describes *inverse radiosity*, a stepping stone to the full algorithm which considers diffuse scenes. Section 4 presents a technique for recovering specular reflectance properties for homogeneous surfaces considering direct illumination only. Section 5 describes how these two techniques are combined to produce our inverse global illumination algorithm. Section 6 completes the technical discussion by describing how high-resolution albedo maps are derived for the surfaces in the scene. Section 7 presents reflectance recovery results from both real and synthetic data, a description of our data acquisition, and synthetic renderings which are compared to real photographs. Section 8 presents some conclusions and avenues for future work.

2 Background and Related Work

The work we present in this paper has been made possible by previous work in BRDF modeling, measurement and recovery, geometry acquisition, image-based rendering, and global illumination.

In graphics, there is a long history of modeling surface reflectance properties using a small number of parameters. Recent efforts in this direction include models introduced in [14, 32, 25, 17]. These models have been shown to yield reasonable approximations to the reflectance properties of many real materials, and they have been used to produce realistic renderings.

On the other hand, considerable recent work has presented methods for measuring and recovering the reflectance properties of materials using imaging devices. [32] and [16] presented techniques and apparatus for measuring reflectance properties, including anisotropic reflection. [5] measured directional reflectance properties of textured objects. [27] and [21] showed that diffuse and specular reflectance properties could be recovered from multiple photographs of an object under direct illumination. [36] recovered reflectance properties of isolated buildings under daylight and was able to re-render them at novel times of day. [7] estimated material properties of parts of a scene so that they could receive shadows and reflections from synthetic objects. [10, 20] used a model of the scene and forward radiosity to estimate diffuse albedos to interactively modify the scene and its lighting. Although mutual illumination has been considered in the problem of shape from shading [23], it has not yet been fully considered for recovering non-diffuse reflectance properties in real environments. A survey of some of the methods is in Marschner [21].

Certain work has shown that changing the lighting in a scene does not necessarily require knowledge of the surface reflectance properties – taking linear combinations of a large set of basis images [24, 35] can yield images with novel lighting conditions.

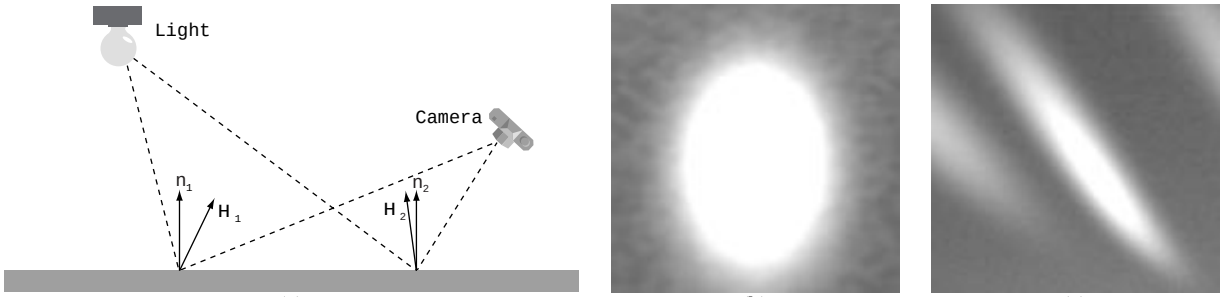
Recent work in laser range scanning and image-based modeling has made it possible to recover accurate geometry of real-world scenes. A number of robust techniques for merging multiple range images into complex models are now available [34, 30, 4, 27]. For architectural scenes involving regular geometry, robust photogrammetric techniques requiring only photographs can also be employed. The model used in this research was constructed using such a technique from [9]; however, our basic technique can be used regardless of how the geometry is acquired.

Work in global illumination (e.g. [11, 15, 31, 37]) has produced algorithms and software to realistically simulate light transport in synthetic scenes. In this work we leverage the hierarchical subdivision technique [13, 1] to efficiently compute surface irradiance. The renderings in this paper were produced using Gregory Ward Larson's RADIANCE system [33].

Photographs taken by a camera involve nonlinearities from the imaging process, and do not have the full dynamic range of real world radiance distributions. In this work we use the high dynamic range technique in [8] to solve these problems.

3 Inverse Radiosity

Most real surfaces exhibit specular as well as diffuse reflection. Recovering both diffuse and specular reflectance models simultaneously in a mutual illumination environment is complicated. In this section, we consider a simplified situation where all surfaces in an environment are pure diffuse (Lambertian). In this case, the global illumination problem simplifies considerably and can be treated in



(a) (b) (c)

Figure 2: (a) The lighting and viewing directions at different points on a surface are different with respect to a fixed light source and a fixed viewpoint. This fact can be used to recover a low-parameter BRDF model for the surface from a single image. $\mathbf{n_i}$'s and H_i's are the normals and halfway vectors between lighting and viewing directions at different locations on the surface. We can infer that surface point P_2 with normal n_2 is close to the center of the highlight, and point P_1 with normal n_1 is relatively far away from the center. (b) An example of an isotropic specular highlight, (c) An example of an anisotropic specular highlight.

the radiosity framework [28]. We define *inverse radiosity* as recovering the diffuse albedo at each surface patch in the environment, provided that the geometry, the lighting conditions and the radiance distribution in the scene are known. In the next section we will discuss another simple case — recovering more general reflectance models with specularity considering only direct illumination — and we address the full problem in Section 5.

In the radiosity framework [28], the surfaces in the environment are broken into a finite number of patches. The partitioning is assumed to be fine enough that the radiosity and diffuse albedo of each patch can be treated as constant. For each such patch,

$$B_i = E_i + \rho_i \sum_j B_j F_{ij} \tag{1}$$

where B_i, E_i, and ρ_i are the radiosity, emission, and diffuse albedo, respectively, of patch i, and F_{ij} is the form-factor between patches i and j. The form-factor F_{ij} is the proportion of the total power leaving patch i that is received by patch j. It can be shown that this is a purely geometric quantity which can be computed from the known geometry of the environment [28].

We take photographs of the surfaces, including the light sources, and use a high dynamic range image technique [8] to capture the radiance distribution. Since Lambertian surfaces have uniform directional radiance distributions, one camera position is sufficient for each surface. Then B_i and E_i in Eqn. (1) become known. Form-factors F_{ij} can be derived from the known geometry. Once these are done, $\rho_i = (B_i - E_i)/(\sum_j B_j F_{ij})$. The solution to inverse radiosity is so simple because the photographs capture the final solution of the underlying light transport among surfaces.

4 Recovering Parameterized BRDFs from Direct Illumination

Before tackling the general case of reflectance recovery from photographs of mutually illuminated surfaces with diffuse *and* specular components, we study another special case. Consider a single surface of uniform BRDF which is illuminated by a point light source in known position and photographed by a camera, also in a known geometric position with respect to the surface(Fig. 2). Every pixel in the radiance image provides a measurement of radiance L_i of the corresponding surface point P_i in the direction of the camera, and the known light source position lets us calculate the irradiance I_i incident on that point.

Our objective is to use these data (L_i, I_i) to estimate the BRDF of the surface. Since the BRDF is a function of four variables (azimuth and elevation of incident and viewing directions) it is obvious that the 2-dimensional set of measurements for a single camera/light source pairing is inadequate to do this in general. However for many materials it is possible to approximate the BRDF adequately by a parameterized BRDF model with a small number of parameters (e.g. Ward [32], Lafortune [17], He [14] etc). We use Ward's parameterization in which the BRDF is modeled as the sum of a diffuse term $\frac{\rho_d}{\pi}$ and a specular term $\rho_s K(\boldsymbol{\alpha}, \boldsymbol{\Theta})$. Here ρ_d and ρ_s are the diffuse and specular reflectance of the surface, respectively, and $K(\boldsymbol{\alpha}, \boldsymbol{\Theta})$ is a function of vector $\boldsymbol{\Theta}$, the azimuth and elevation of the incident and viewing directions, and parameterized by $\boldsymbol{\alpha}$, the surface roughness vector. For anisotropic surfaces $\boldsymbol{\alpha}$ has 3 components; for isotropic surfaces $\boldsymbol{\alpha}$ has only one component and reduces to a scalar. The precise functional form of $K(\boldsymbol{\alpha}, \boldsymbol{\Theta})$ in the two cases may be found in Appendix 1.

This leads us to the following equation for each surface point P_i,

$$L_i = (\frac{\rho_d}{\pi} + \rho_s K(\boldsymbol{\alpha}, \boldsymbol{\Theta}_i))\, I_i \tag{2}$$

where L_i, I_i and $\boldsymbol{\Theta}_i$ are known, and the parameters ρ_d,ρ_s, $\boldsymbol{\alpha}$ are unknowns to be estimated. Depending on whether we are using an isotropic or anisotropic model for the specular term we have a total of 3 or 5 unknown parameters, while there are as many constraining equations as the number of pixels in the radiance image of the surface patch. By solving a nonlinear optimization problem (see Appendix 1 for details), we can find the best estimate of ρ_d,ρ_s, $\boldsymbol{\alpha}$.

There are two important subtleties in the treatment of this optimization problem. One is that we need to solve a weighted least squares problem, otherwise the larger values from the highlight (with correspondingly larger noise in radiance measurements) cause a bias in parameter estimation. The second is the use of color information which needs to be done differently for dielectrics and metals. Both of these issues are discussed in Appendix 1.

To obtain an obvious global minimum for this optimization problem and achieve robust parameter recovery, the radiance image should cover the area that has a specular highlight as well as some area with very low specular component. If the highlight is missing, we do not have enough information for recovering specular parameters, and can only consider the surface to be diffuse.

5 Recovering Parameterized BRDFs in a Mutual Illumination Environment

We are now ready to study the general case when the environment consists of a number of surfaces and light sources with the surface reflectances allowed to have both diffuse and specular components.

Consider a point P_i on a surface patch seen by camera C_v (Fig. 3). The radiance from P_i in the direction of the camera is the re-

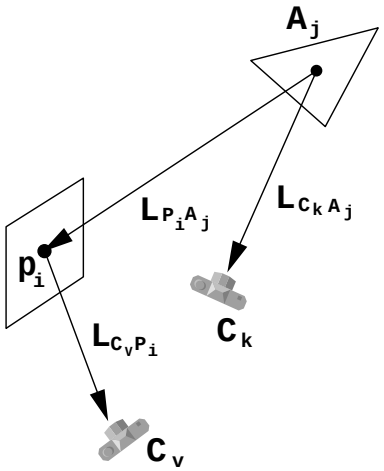


Figure 3: Patch A_j is in the radiance image captured by camera C_k. The specular component at A_j in the direction of sample point P_i is different from that in the direction of camera C_k. The difference is denoted by ΔS.

flection of the incident light contributed by all the light sources as well as all the surrounding surfaces. Eqn. (2) generalizes to

$$L_{C_vP_i} = E_{C_vP_i} + \rho_d \sum_j L_{P_iA_j} F_{P_iA_j} + \rho_s \sum_j L_{P_iA_j} K_{C_vP_iA_j}, \quad (3)$$

where $L_{C_vP_i}$ is the radiance value in the direction of camera C_v at some sample point P_i on the surface, $E_{C_vP_i}$ is the emission in the direction of camera C_v, $L_{P_iA_j}$ is the radiance value along the direction from patch A_j to point P_i on the surface, $F_{P_iA_j}$ is the analytical point-to-patch form-factor [2] between sample point P_i and patch A_j, and $\rho_s K_{C_vP_iA_j}$ is the specular term evaluated at P_i for a viewpoint at camera C_v and a light source position at patch A_j. The arguments, $\boldsymbol{\alpha}$ and $\boldsymbol{\Theta}$, of K have been dropped to simplify notation.

As before, our objective is to estimate ρ_d, ρ_s, and specular roughness parameters $\boldsymbol{\alpha}$. Of the other variables in Eqn. (3), $E_{C_vP_i} = 0$ for nonsources, and $L_{C_vP_i}$ can be measured directly from the radiance image at camera C_v. In general, the radiances $L_{P_iA_j}$ cannot be measured directly but have to be estimated iteratively. Suppose patch A_j in the environment appears in another radiance image taken by camera C_k(Fig. 3). Only if we assume A_j is Lambertian, does $L_{P_iA_j}$ in Eqn. (3) equal $L_{C_kA_j}$, the radiance from A_j to camera C_k. Otherwise, the diffuse components will be equal, but the specular components will differ.

$$L_{P_iA_j} = L_{C_kA_j} + \Delta S_{C_kP_iA_j} \quad (4)$$

Here $\Delta S_{C_kP_iA_j} = S_{P_iA_j} - S_{C_kA_j}$ is the difference between the specular components $S_{P_iA_j}$ and $S_{C_kA_j}$ of the radiances in the two directions. To compute the specular differences $\Delta S_{C_kP_iA_j}$, we need the BRDF of A_j, which is initially unknown. The estimation of ΔS (Section 5.1) therefore has to be part of an iterative framework. Assuming that the dominant component of reflectance is diffuse, we can initialize the iterative process with $\Delta S = 0$ (this sets $L_{P_iA_j} = L_{C_kA_j}$).

To recover BRDF parameters for all the surfaces, we need radiance images covering the whole scene. Each surface patch needs to be assigned a camera from which its radiance image is selected. At least one specular highlight on each surface needs to be visible in the set of images, or we will not be able to recover its specular reflectance and roughness parameters. Each sample point gives an

```
For each camera position C
  For each polygon T
    For each light source O
      Obtain the intersection P between plane of T and line CO'
        (O' and O are symmetric about T);
      Check if P falls inside polygon T;
      Check if there is any occlusion between P and O;
      Check if there is any occlusion between C and any point
        in a local neighborhood of P;
      /* A highlight area is detected if P passed all the above tests.*/
End
```

Figure 4: The specular highlight detection algorithm.

equation similar to Eqn. (3). From these equations, we can set up a weighted least-squares problem for each surface as in Appendix 1. During optimization, we need to gather irradiance at each sample point from the surface patches in the environment. One efficient way of doing this is to subdivide each surface into a hierarchy of patches [13, 1] and link different sample points to patches at different levels in the hierarchy. The solid angles subtended by the linked patches at the sample points should always be less than a prescribed threshold. There is a radiance value from the patch to the sample point and a ΔS associated with each hierarchical link.

For each sample point, we build hierarchical links to a large number of patches, and gather irradiance from these links. The amount of memory and computation involved in this process limits the number of samples for each highlight area. To make a reasonable tradeoff, we note that irradiance from indirect illumination caused by surrounding surfaces generally has little high-frequency spatial variation. Because of this, it makes sense to draw two sets of samples, one sparse set, and one dense set [2]. For the samples in the sparse set, we build hierarchical links and gather irradiance from the environment as usual. For the samples in the dense set, only their irradiance from light sources is computed explicitly, their irradiance from indirect illumination is computed by interpolation.

We are now ready to state the complete inverse global illumination algorithm. First detect all specular highlight blobs falling inside the radiance images using knowledge of the positions of the light sources, the camera poses, and the geometry (Fig. 4). Set the initial ΔS associated with each hierarchical link to zero. We can then recover an initial estimate of the BRDF parameters for each surface independently by solving a series of nonlinear optimization problems. The estimated specular parameters are used to update all ΔS's and $L_{P_iA_j}$'s associated with the hierarchical links. With the updated incident radiances, we can go back and re-estimate the BRDF parameters again. This optimization and update process is iterated several times to obtain the final solution of the BRDFs for all surfaces. The overall algorithm is shown in Fig. 5.

5.1 Estimation of ΔS

Suppose there is a hierarchical link $l_{P_iA_j}$ between a sample point P_i and a patch A_j which is visible to a camera C_k (Fig. 6). The ΔS for $l_{P_iA_j}$ is defined to be the difference of the specular component in directions $\vec{A_jP_i}$ and $\vec{A_jC_k}$. To estimate this difference, we need to obtain the specular component along these two directions given the BRDF parameters of patch A_j. A one-bounce approximation of ΔS for link $l_{P_iA_j}$ can be obtained by using Monte Carlo ray-tracing [32]. Because of off-specular components, multiple rays

[2] We choose the two sets of samples as follows. We first find the center of the highlight area in the image plane and rotate a straight line around this center to a number of different positions. The dense set of samples is the set of points on the surface corresponding to all the pixels on these lines. We choose the sparse set of samples on each line by separating two consecutive samples by some fixed distance in the object space.

Detect specular highlight blobs on the surfaces.
Choose a set of sample points inside and around each highlight area.
Build hierarchical links between sample points and patches in the environment and use ray tracing to detect occlusion.
Assign to each patch one radiance image and one average radiance value captured at the camera position.
Assign zero to ΔS at each hierarchical link.
For iter=1 to N
  For each hierarchical link,
    use its ΔS to update its associated radiance value;
  For each surface,
    optimize its BRDF parameters using the data from its sample points;
  For each hierarchical link,
    estimate its ΔS with the new BRDF parameters.
End

Figure 5: The *Inverse Global Illumination* algorithm.

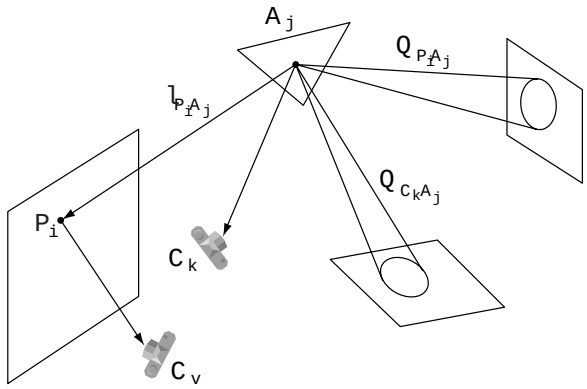


Figure 6: Random rays are traced around the two cones to obtain a one-bounce approximation of ΔS.

should be traced and the direction of the rays is randomized around the mirror directions of $A_j\vec{P}_i$ and $A_j\vec{C}_k$, respectively. For each possible ray direction, the probability density of shooting a ray in that direction is proportional to $K(\boldsymbol{\alpha_j}, \boldsymbol{\Theta})$ where $\boldsymbol{\Theta}$ encodes the incident and outgoing directions. Intuitively, most of the rays fall inside the two cones $Q_{P_iA_j}$ and $Q_{C_kA_j}$ centered at the two mirror directions. The width of each cone depends on the specular roughness parameters $\boldsymbol{\alpha_j}$ of patch A_j. The radiance along each ray is obtained from the patch hit by the ray. Suppose $L_{Q_{P_iA_j}}$ and $L_{Q_{C_kA_j}}$ are the average radiance values of the rays around the two cones, respectively, and $\rho_{s_{A_j}}$ is the specular reflectance of patch A_j. Because the average value of Monte Carlo sampling approximates the total irradiance modulated by $K(\boldsymbol{\alpha_j}, \boldsymbol{\Theta})$, ΔS can simply be estimated as $\rho_{s_{A_j}}(L_{Q_{P_iA_j}} - L_{Q_{C_kA_j}})$. This calculation could be extended to have multiple bounces by using path tracing [15]; we found that the one-bounce approximation was adequate for our purposes.

5.2 Practical Issues

We do not have a formal characterization of the conditions under which the inverse global illumination algorithm converges, or of error bounds on the recovered BRDF parameter values. In practice, we found it worked well (Section 7). Here we give some heuristic advice on how to acquire images to obtain good performance.

- *Use multiple light sources.* A specular highlight directly caused by one of the light sources should be captured on each surface. Having multiple light sources increases the probability that this can be achieved, and lets the whole scene receive more uniform illumination. This also increases the relative contribution of the diffuse component at any particular sample point P_i, and supports the $\Delta S = 0$ initialization, since highlights from different sources will usually occur at different locations on the surface.

- *Use concentrated light sources.* If the incoming radiance distribution is not very directional, the specular highlights will be quite extended and it will be difficult to distinguish the specular component from the diffuse one.

6 Recovering Diffuse Albedo Maps

In the previous sections, we modeled the reflectance properties as being uniform for each surface. In this section, we continue to do so for specular parameters because a small number of views of each surface does not provide enough information to reliably estimate specular parameters for each point individually. However, we relax this constraint on diffuse albedo and model it as a spatially varying function, an *albedo map*, on each surface. The diffuse albedo for any point x on a surface is computed as:

$$\rho_d(x) = \pi D(x)/I(x) \tag{5}$$

where $\rho_d(x)$ is the diffuse albedo map, $D(x)$ is the diffuse radiance map, and $I(x)$ is the irradiance map.

Suppose there is an image covering the considered surface which gives a radiance map $L(x) = D(x) + S(x)$ where $S(x)$ is the specular radiance map seen from the image's camera position. Then the diffuse radiance map in Eqn. (5) can be obtained by subtracting the specular component from each pixel of the radiance map $L(x)$ using the specular reflectance parameters already recovered. We estimate the radiance due to specular reflection as the sum of specular reflection due to direct and indirect illumination. The specular reflection due to direct illumination is computed from the knowledge of the direct lighting and the estimated reflectance properties, and we estimate the indirect specular reflectance by tracing a perturbed reflected ray into the environment in a manner similar to that in Section 5.1.

The irradiance $I(x)$ can be computed at any point on the surface from the direct illumination and by using analytical point-to-patch form-factors [2] as in previous sections of this paper. For efficiency, we compute the irradiance due to the indirect illumination only at certain sample points on the surfaces, and interpolate these indirect irradiance estimates to generate estimates for all surface points x. Of course, care must be taken to sufficiently sample the irradiance in regions of rapidly changing visibility to the rest of the scene.

Something that complicates estimating diffuse albedos in this manner is that in highlight regions the specular component of the reflectance $S(x)$ will be much larger than the diffuse component $D(x)$. As a result, relatively small errors in the estimated $S(x)$ will cause large relative errors in $D(x)$ and thus $\rho_d(x)$. However, just as a person might shift her view to avoid glare while reading a movie poster, we make use of multiple views of the surface to solve this problem.

Suppose at a point x on a surface, we have multiple radiance values $\{L_k(x)\}_{k=1}^p$ from different images. The highest value in this set will exhibit the strongest specular component, so we simply remove this value from consideration. For the remaining values, we subtract the corresponding specular estimates $S_k(x)$ from the radiance values $L_k(x)$, to obtain a set of diffuse radiance estimates $D_k(x)$. We compute a final diffuse radiance component $D(x)$ as a weighted average of the $D_k(x)$, with weights inversely proportional to the magnitude of the estimated specular components $S_k(x)$ to minimize the relative error in $D(x)$. We also weight the $D_k(x)$

values proportionally to the cosine of the viewing angle of the camera in order to reduce the influence of images at grazing angles; such oblique images typically have poor texture resolution and exhibit particularly strong specular reflection. Since we are combining information taken from different images, we smooth transitions at image boundaries using the image blending technique in [9].

Once diffuse albedo maps are recovered, they could be used to separate the diffuse and specular components in the specular highlight areas. This would allow recovering more accurate specular parameters in the BRDF model. In practice, however, we have found good estimates to be obtained without further refinements.

7 Results

7.1 Results for a Simulated Scene

We first tested our algorithm on a simple simulated cubical room with mutual illumination. This allowed us to verify the accuracy of the algorithm and compare its results to ground truth. All the six surfaces of the room have monochromatic diffuse and specular components, but each one has a distinct set of parameters. Each of the surfaces has spatially uniform specularity. We assigned two surfaces to be anisotropically specular and added 10-20% zero mean white noise to the uniform diffuse albedo of two surfaces to simulate spatial variations. We used the RADIANCE rendering system [33] to produce synthetic photographs of this scene. Six of the synthetic photographs were taken from the center of the cube with each one covering one of the six surfaces. Another set of six zoomed-in photographs were taken to capture the highlight areas. The scene was illuminated by six point light sources so that specular highlights could be observed on each surface. These twelve images along with the light source intensity and positions were used to solve the BRDF parameters. The images of the specular highlights are shown in Fig. 7. Some of the highlights are visually very weak, but corresponding parameters can still be recovered numerically. The original and recovered BRDF parameters are given in Table 1. For the last two surfaces with noisy diffuse albedo, the recovered albedo values are compared to the true average values. The total running time for BRDF recovery is about half an hour on a SGI O_2 180MHz workstation.

The numerical errors shown in Table 1 are obtained by comparing the recovered parameters with the original ones. There are three sources of error: BRDF modeling error, rendering error, and BRDF recovery error. BRDF modeling error comes from the inability of a given BRDF model to capture the behavior of a real material. By using the same model for recovery that RADIANCE uses for rendering, BRDF modeling error was eliminated for this test. However, because RADIANCE computes light transport only approximately, rendering error is present. We thus cannot determine the exact accuracy of our BRDF recovery. However, the test demonstrates that the algorithm works well in practice.

7.2 Results for a Real Scene

In this section we demonstrate the results of running our algorithm on a real scene. The scene we chose is a small meeting room with some furniture and two whiteboards; we also decorated the room with colored cards, posters, and three colored metallic spheres[3]. Once the BRDFs of the materials were recovered, we were able to re-render the scene under novel lighting conditions and with added virtual objects.

[3]The spheres were obtained from Baker's Lawn Ornaments, 570 Berlin Plank Road, Somerset PA 15501, (814) 445-7028.

	ρ_d	ρ_s	$\alpha_x\,(\alpha)$	α_y	γ
True	0.3	0.08	0.6	0.03	0
Recovered	0.318296	0.081871	0.595764	0.030520	-0.004161
Error(%)	6.10	2.34	0.71	1.73	
True	0.1	0.1	0.3		
Recovered	0.107364	0.103015	0.300194		
Error(%)	7.36	3.02	0.06		
True	0.1	0.01	0.1		
Recovered	0.100875	0.010477	0.101363		
Error(%)	0.88	4.77	1.36		
True	0.3	0.02	0.15		
Recovered	0.301775	0.021799	0.152331		
Error(%)	0.59	8.90	1.55		
True	0.2	0.05	0.05		
Recovered	0.206312	0.050547	0.050291		
Error(%)	3.16	1.09	0.58		
True	0.2	0.1	0.05	0.3	45
Recovered	0.209345	0.103083	0.050867	0.305740	44.997876
Error(%)	4.67	3.08	1.73	1.91	

Table 1: Comparison between true and recovered BRDF parameters for the six surfaces of a unit cube. The first and last surfaces have anisotropic specular reflection. They have two more parameters: second roughness parameter α_y and the orientation γ of the principal axes in a local coordinate system. The errors shown are the combined errors from both rendering and recovering stages.

7.2.1 Data Acquisition

We illuminated the scene with three heavily frosted 3-inch diameter tungsten light bulbs. Using high dynamic range photography, we verified that the lights produced even illumination in all directions. A DC power source was used to eliminate 60Hz intensity fluctuations from the alternating current power cycle.

We used a Kodak DCS520 color digital camera for image acquisition. The radiance response curve of the camera was recovered using the technique in [8]. We used a wide-angle lens with a 75 degree field of view so that we could photograph all the surfaces in the scene from a few angles with a relatively small number of shots. Forty high dynamic range radiance images, shown in Fig. 8, were acquired from approximately 150 exposures. Twelve of the images were taken specifically to capture specular highlights on surfaces.

The radiance images were processed to correct for radial light falloff and radial image distortion. Each of these corrections was modeled by fitting a polynomial of the form $1 + ar^2 + br^4$ to calibration data captured with the same lens settings used for the scene images. To reduce glare and lens flare, we shaded the lens from directly viewing the light sources in several of the images. Regions in the images corresponding to the light stands (which we did not model) or where excessive remaining glare was apparent were masked out of the images, and ignored by the algorithm. The thin cylindrical light stands which appear in the synthetic renderings have been added to the recovered model explicitly.

The radiance images were used to recover the scene geometry and the camera positions (Fig. 9) using the Façade [9] modeling system. Segmentation into areas of uniform specular reflectance was obtained by having each polygon of each block in the model (e.g. the front of each poster, the surface of each whiteboard, the top of each table) have its own uniform specular reflectance parameters.

The positions and intensities of the three light sources were recovered from the final three radiance images. During BRDF recovery, the area illumination from these spherical light sources was computed by stochastically casting several rays to each source.

7.2.2 BRDF Recovery

Given the necessary input data, our program recovered the surface BRDFs in two stages. In the first stage, it detected all the highlight regions and recovered parametrized BRDFs for the surfaces. In this stage, even if a surface had rich texture, only an average dif-

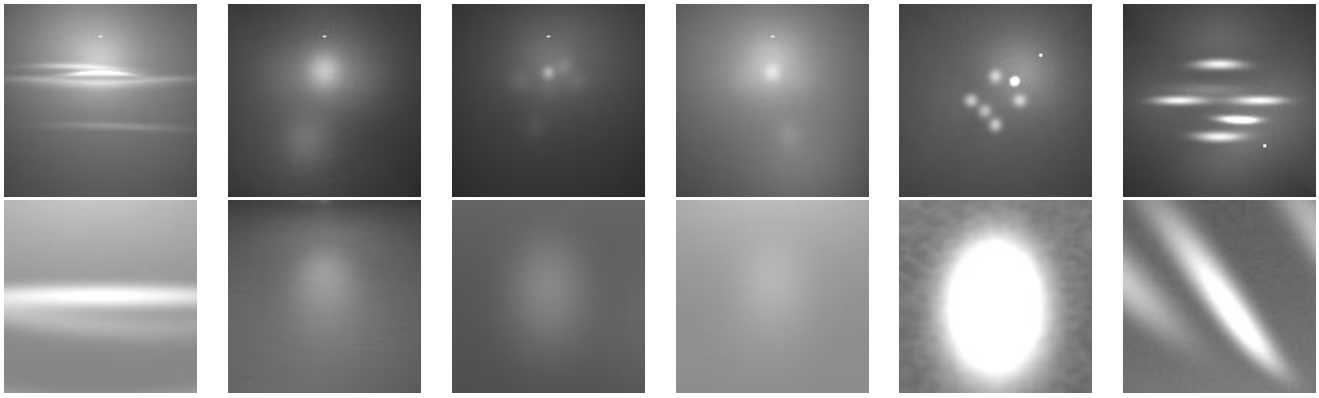


Figure 7: Synthetic grey-scale images of the interior of a unit cube in the presence of mutual illumination. These are used for recovering the BRDF model of each surface. The top row shows the six images taken at the center of the cube with each one covering one of the six surfaces. The bottom row shows the six zoomed-in images taken to capture one specular highlight area on each surface. The first and last surfaces have anisotropic specular reflection. The last two surfaces have 20 and 10 percent zero mean white noise added to their diffuse albedo, respectively.

	ρ_d(red)	ρ_d(green)	ρ_d(blue)	ρ_s(red)	ρ_s(green)	ρ_s(blue)	α
whiteboard	0.5794	0.5948	0.6121	0.0619	0.0619	0.0619	0.0137
roundtable top	0.7536	0.7178	0.7255	0.0366	0.0366	0.0366	0.0976
door	0.6353	0.5933	0.5958	0.0326	0.0326	0.0326	0.1271
wall	0.8543	0.8565	0.8036	0.0243	0.0243	0.0243	0.1456
poster	0.1426	0.1430	0.1790	0.0261	0.0261	0.0261	0.0818
red card	0.7507	0.2404	0.3977	0.0228	0.0228	0.0228	0.0714
yellow card	0.8187	0.7708	0.5552	0.0312	0.0312	0.0312	0.1515
teal card	0.4573	0.5951	0.5369	0.0320	0.0320	0.0320	0.1214
lavender card	0.3393	0.3722	0.4437	0.0077	0.0077	0.0077	0.1144
red ball	0	0	0	0.5913	0.1862	0.3112	0
green ball	0	0	0	0.2283	0.3694	0.3092	0
blue ball	0	0	0	0.2570	0.3417	0.4505	0

Table 2: BRDF parameters recovered for the materials in the test room. All of them are isotropic, and most of them are plastic. The balls are metallic.

fuse albedo was recovered. Surfaces for which no highlights were visible the algorithm considered diffuse. The second stage used the recovered specular reflection models to generate diffuse albedo maps for each surface by removing the specular components.

The running time for each of the two stages was about 3 hours on a Pentium II 300MHz PC. The results show our algorithm can recover accurate specular models and high-quality diffuse albedo maps. Fig. 10 shows how specular highlights on the white board were removed by combining the data from multiple images. Fig. 11 shows the albedo maps obtained for three identical posters placed at different places in the room. Although the posters were originally seen in different illumination, the algorithm successfully recovers very similar albedo maps for them. Fig. 12 shows that the algorithm can remove "color bleeding" effects: colors reflected onto a white wall from the cards on the table do not appear in the wall's diffuse albedo map. Table 2 shows the recovered specular parameters and average diffuse albedo for a variety of the surfaces in the scene. We indicated to the program that all the materials are isotropic, and that the metallic spheres only have ideal specular components[4].

[4]For surfaces that have only ideal specular reflection, such as mirrors, there is no diffuse component and the roughness parameter is zero. We can still recover their specular reflectance ρ_s from a single image by noting that the specular reflectance can be computed as the simple ratio between two radiance values. One is the radiance value in the image corresponding to the intersection between the surface and a ray shot from the camera position; the other is the radiance value of the environment along the reflected ray. In practice, we shoot a collection of rays from the camera position to obtain the average reflectance.

7.2.3 Re-rendering Results

We directly compared synthetic images rendered with our recovered BRDF models to real images. In Fig. 13, we show the comparison under the original lighting conditions in which we took the images for BRDF recovery. In Fig. 14, we show the comparison under a novel lighting condition obtained by removing two of the lights and moving the third to a new location, and adding a new object. There are a few differences between the real and synthetic images. Some lens flare appears in the real images of both figures, which we did not attempt to simulate in our renderings. We did not model the marker trays under the whiteboards, so their shadows do not appear in the synthetic images. In Fig. 14, a synthetic secondary highlight caused by specular reflection from the adjacent whiteboard appears darker than the one in the real image, which is likely due to RADIANCE's approximations for rendering secondary specularities. However, in both figures, real and synthetic images appear quite similar.

Fig 15 shows four panoramic views of the rendered scene. (a) shows the hierarchical mesh with the initial estimates of radiance obtained from the images. (b) shows the entire room rendered in the original illumination. (c) shows the entire scene rendered with novel lighting. The original lights were removed and three track lights were virtually installed on the ceiling to illuminate the posters. Also, a strange chandelier was placed above the spheres on the table. The new lights reflect specularly off of the posters and the table. Since the chandelier contains a point light source, it casts a hard shadow around the midsection of the room. The interior of the chandelier shade is turquoise colored which results in turquoise shadows under the spheres. A small amount of synthetic glare was added to this image. (d) shows the result of adding syn-

thetic objects to various locations in the room, including two chairs, a crystal ball, two metal boxes, and a floating diamond. In addition, a very large orange sculpture, was placed at the back of the room. All of the objects exhibit proper shadows, reflections, and caustics. The sculpture is large enough to turn the ceiling noticeably orange due to diffuse interreflection. The video for this paper shows a fly-through of each of these scenes.

8 Conclusions and Future Work

In this paper we have presented a new technique for determining reflectance properties of entire scenes taking into account mutual illumination. The properties recovered include diffuse reflectance that varies arbitrarily across surfaces, and specular reflectance parameters that are constant across regions. The technique takes as input a sparse set of geometrically and photometrically calibrated photographs taken under calibrated lighting conditions, as well as a geometric model of the scene. The algorithm iteratively estimates irradiances, radiances, and reflectance parameters. The result is a characterization of surface reflectance properties that is highly consistent with the observed radiances in the scene. We hope this work will be a useful step towards bringing visual spaces from the real world into the virtual domain, where they can be visualized from any angle, with any lighting, and with additions, deletions, and modifications according to our needs and imaginations.

There are a few directions for future research. We wish to apply our technique to more general geometrical and photometric data, such as multispectral radiance images and geometry accquired from laser scanners. It would be of significant practical value to be able to calibrate and use existing or natural illumination in recovering reflectance properties. The algorithm should be more robust to errors in the geometric model, misregistration of the photographs, and errors in the light source measurements. It would also be of theoretical value to obtain conditions under which the algorithm converges.

Acknowledgments

The authors wish to thank David Culler and the Berkeley NOW (Network of Workstations, http://now.cs.berkeley.edu/) project, and Tal Garfinkel for his help in using the NOW to render the video sequences. Thanks to Gregory Ward Larson for advice in using RADIANCE and estimating reflectance, Carlo Séquin for providing the sculpture model, and the reviewers for their valuable comments. This research was supported by a Multidisciplinary University Research Initiative on three dimensional direct visualization from ONR and BMDO, grant FDN00014-96-1-1200, the California MICRO program, Phillips Corporation, Interval Research Corporation, Pixar Animation Studios and Microsoft Graduate Fellowship.

Appendix 1. BRDF Model and Parameter Recovery

In this appendix we present more details on the BRDF model, introduced in Section 4, and how its parameters are recovered. We use Ward's [32] model for the specular term in the BRDF, which could be modeled as either isotropic or anisotropic. In the isotropic case,

$$K(\alpha, \mathbf{\Theta}) = \frac{1}{\sqrt{\cos\theta_i \cos\theta_r}} \frac{\exp[-\tan^2 \delta/\alpha^2]}{4\pi\alpha^2} \tag{6}$$

where α is a scalar surface roughness parameter, θ_i is the incident angle, θ_r is the viewing angle, and δ is the angle between the surface normal and the halfway vector H between the lighting and viewing directions. θ_i, θ_r are two components (along with ϕ_i, ϕ_r) of the vector $\mathbf{\Theta}$ which represents the incidence and viewing directions.

In the anisotropic case, we need two distinct roughness parameters α_x, α_y for two principal axes on the surface and an azimuth angle γ to define the orientation of these principal axes on the surface relative to a canonical coordinate system. Then, the parameter vector $\boldsymbol{\alpha}$ actually has three components $(\alpha_x, \alpha_y, \gamma)$ and we have:

$$K(\boldsymbol{\alpha}, \mathbf{\Theta}) = \frac{1}{\sqrt{\cos\theta_i \cos\theta_r}} \frac{\exp[-\tan^2 \delta(\cos^2 \phi/{\alpha_x}^2 + \sin^2 \phi/{\alpha_y}^2)]}{4\pi\alpha_x\alpha_y} \tag{7}$$

where δ is the same as in the isotropic case, and ϕ is the azimuth angle of the halfway vector H projected into the local 2D coordinate system on the surface patch defined by the two principal axes. To compute ϕ, γ, which relates this coordinate system to the canonical coordinate system, is necessary.

Now to parameter recovery. We wish to find ρ_d, ρ_s and $\boldsymbol{\alpha}$ that minimize the squared error between the measured and predicted radiance,

$$e(\rho_d, \rho_s, \boldsymbol{\alpha}) = \sum_{i=1}^{m} (L_i - \frac{\rho_d}{\pi} I_i - \rho_s K(\boldsymbol{\alpha}, \mathbf{\Theta}_i) I_i)^2 \tag{8}$$

where L_i is the measured radiance and I_i is the irradiance (computable from the known light source position) at sample point p_i on the surface, and m is the number of sample points.

Note that given a guess of $\boldsymbol{\alpha}$, $K(\boldsymbol{\alpha}, \mathbf{\Theta}_i)$ becomes a known quantity, and minimizing the error e reduces to a standard linear least-squares problem for estimating ρ_d and ρ_s. Plugging in these values in the right hand side of Eqn. (8) lets us compute e as a function of $\boldsymbol{\alpha}$. The optimization problem thus simplifies to a search for the optimum value of $\boldsymbol{\alpha}$ to minimize $e(\boldsymbol{\alpha})$. This is either a one-dimensional or three-dimensional search depending on whether an isotropic or anisotropic model of the specular term is being used. We use golden section search [26] for the isotropic case, and the downhill simplex method [26] in the anisotropic case. It is convenient that neither method requires evaluating the derivative $e'(\boldsymbol{\alpha})$, and both methods are fairly robust.

To deal with colored materials, we estimate both diffuse and specular reflectance in each of the red, green, blue color channels. The specular roughness parameters $\boldsymbol{\alpha}$ are the same for all color channels. The nonlinear optimization is still over 1 or 3 parameters, since given $\boldsymbol{\alpha}$, ρ_d and ρ_s estimation for each channel remains a linear least squares problem.

To make the parameter estimation additionally robust, we make two simple extensions to the basic strategy derived above. The first is to solve a weighted least squares problem instead of the vanilla version in Eqn. (8). Radiance measurements from the highlight area have much larger magnitude than those from the non-highlight area. Correspondingly the error in those measurements is higher both because of noise in imaging as well as error in the BRDF model. Giving all the terms in (8) equal weight causes biased fitting and gives poor estimation of the diffuse reflectance. From a statistical point of view, the correct thing to do is to weight each term by the reciprocal of the variance of expected error in that measurement. Not having a good model for the error term, we chose a heuristic strategy in which the weight w_i for the i-th term in the summation in Eqn. (8) is set to $\frac{1}{K(\boldsymbol{\alpha_c}, \mathbf{\Theta}_i)}$ where $\boldsymbol{\alpha_c}$ is some *ad hoc* or iteratively improved roughness vector. Since the roughness of most isotropic materials is less than 0.2, we used an initial value between 0.1 and 0.2 for scalar α_c.

The second refinement to improve parameter recovery is to use specular color information. For instance, specular highlights on dielectric and plastic materials have the same color as the light source, while the color of specular highlights on metals is the same as their diffuse components, which is the color of the light modulated by the diffuse albedo. For plastic objects, there would be one distinct variable ρ_d for each color channel, but the same variable ρ_s for all color channels. For metallic objects, there would be one variable ρ_d for each channel and a common ratio between the specular and diffuse reflectance in all channels. Thus, we can reduce the degree of freedom from $2N$ to N+1 where N is the number of color channels. For plastic, we can still obtain both analytic and numerical linear least-squares solutions for the N+1 variables provided the other parameters are fixed. The program performs a heuristic test to determine whether a material should be estimated with the metal or plastic specular reflectance model. Our program first solves for the specular reflectance of each color channel separately and then checks to see if they are larger than the estimated diffuse components. If they are larger, then the material is considered metallic. Otherwise, the plastic model is used. Then the smaller number of parameters corresponding to these material types are solved.

References

[1] AUPPERLE, L., AND HANRAHAN, P. A hierarchical illumination algorithm for surfaces with glossy reflection. In *SIGGRAPH 93* (August 1993), pp. 155–164.

[2] BAUM, D. R., RUSHMEIER, H. E., AND WINGET, J. M. Improving radiosity solutions through the use of analytically determined form factors. In *SIGGRAPH 89* (1989), pp. 325–334.

[3] CHEN, E. QuickTime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH '95* (1995).

[4] CURLESS, B., AND LEVOY, M. A volumetric method for building complex models from range images. In *SIGGRAPH '96* (1996), pp. 303–312.

[5] DANA, K. J., GINNEKEN, B., NAYAR, S. K., AND KOENDERINK, J. J. Reflectance and texture of real-world surfaces. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.* (1997), pp. 151–157.

[6] DEBEVEC, P., YU, Y., AND BORSHUKOV, G. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *9th Eurographics Workshop on Rendering*, (1998), pp. 105-116.

[7] DEBEVEC, P. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH 98* (July 1998).

[8] DEBEVEC, P. E., AND MALIK, J. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH 97* (August 1997), pp. 369–378.

[9] DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96* (August 1996), pp. 11–20.

[10] DRETTAKIS, G., ROBERT, L., AND BOUGNOUX, S. Interactive common illumination for computer augmented reality. In *8th Eurographics workshop on Rendering, St. Etienne, France* (May 1997), J. Dorsey and P. Slusallek, Eds., pp. 45–57.

[11] GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84* (1984), pp. 213–222.

[12] GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. The Lumigraph. In *SIGGRAPH '96* (1996), pp. 43–54.

[13] HANRAHAN, P., SALZMAN, P., AND AUPPERLE, L. A rapid hierarchical radiosity algorithm. In *SIGGRAPH 91* (1991), pp. 197–206.

[14] HE, X. D., TORRANCE, K. E., SILLION, F., AND GREENBERG, D. P. A comprehensive physical model for light reflection. In *SIGGRAPH 91*, (August 1991).

[15] KAJIYA, J. The rendering equation. In *SIGGRAPH '86* (1986), pp. 143–150.

[16] KARNER, K. F., MAYER, H., AND GERVAUTZ, M. An image based measurement system for anisotropic reflection. In *EUROGRAPHICS Annual Conference Proceedings* (1996).

[17] LAFORTUNE, E.P.F., FOO, S., TORRANCE,K.E., AND GREENBERG, D.P. Non-Linear Approximation of Reflectance Functions. In *SIGGRAPH 97*, (1997), pp.117-126.

[18] LAVEAU, S., AND FAUGERAS, O. 3-D scene representation as a collection of images. In *Proceedings of 12th International Conference on Pattern Recognition* (1994), vol. 1, pp. 689–691.

[19] LEVOY, M., AND HANRAHAN, P. Light field rendering. In *SIGGRAPH '96* (1996), pp. 31–42.

[20] LOSCOS, C., FRASSON, M.-C., DRETTAKIS, G., WALTER, B., GRANIER, X., AND POULIN, P. Interactive Virtual Relighting and Remodeling of Real Scenes. Technical Report, iMAGIS-GRAVIR/IMAG-INRIA, (May 1999), http://www-imagis.imag.fr/Membres/Celine.Loscos/relight.html.

[21] MARSHNER, S. *Inverse Rendering for Computer Graphics*. PhD thesis, Cornell University, August 1998.

[22] MCMILLAN, L., AND BISHOP, G. Plenoptic Modeling: An image-based rendering system. In *SIGGRAPH '95* (1995).

[23] NAYAR, S. K., IKEUCHI, K., AND KANADE, T. Shape from interreflections. *International Journal of Computer Vision 6*, 3 (1991), 173–195.

[24] NIMEROFF, J., SIMONCELLI, E., AND DORSEY, J. Efficient re-rendering of naturally illuminated environments. In *5th Eurographics Workshop on Rendering* (1994).

[25] Oren, M., and Nayar,S.K., "Generalization of Lambert's Reflectance Model", *Computer Graphics Proceedings, Annual Conference Series*, pp.239-246 (1994).

[26] PRESS, W., FLANNERY, B., TEUKOLSKY, S., AND VETTERLING, W. *Numerical Recipes in C*. Cambridge Univ. Press, New York, 1988.

[27] SATO, Y., WHEELER, M. D., AND IKEUCHI, K. Object shape and reflectance modeling from observation. In *SIGGRAPH '97* (1997), pp. 379–387.

[28] SILLION, F. X., AND PUECH, C. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers, San Francisco, 1994.

[29] SZELISKI, R., AND SHUM, H.-Y. Creating full view panoramic image mosaics and environment maps. In *SIGGRAPH 97* (1997), pp. 251–258.

[30] TURK, G., AND LEVOY, M. Zippered polygon meshes from range images. In *SIGGRAPH '94* (1994), pp. 311–318.

[31] VEACH, E., AND GUIBAS, L. J. Metropolis light transport. In *SIGGRAPH 97* (August 1997), pp. 65–76.

[32] WARD, G. J. Measuring and modeling anisotropic reflection. In *SIGGRAPH '92* (July 1992), pp. 265–272.

[33] WARD, G. J. The RADIANCE lighting simulation and rendering system. In *SIGGRAPH '94* (July 1994), pp. 459–472.

[34] Y.CHEN, AND MEDIONI, G. Object modeling from multiple range images. *Image and Vision Computing 10*, 3 (April 1992), pp.145–155.

[35] WONG T.-T., HENG P.-A., OR S.-H. AND NG W.-Y. Image-based Rendering with Controllable Illumination. In *8th Eurographics Workshop on Rendering*, (June 1997), pp.13–22.

[36] YU, Y., AND MALIK, J. Recovering photometric properties of architectural scenes from photographs. In *SIGGRAPH 98* (July 1998), pp. 207–217.

[37] YU, Y., AND WU, H. A Rendering Equation for Specular Transfers and its Integration into Global Illumination. Eurographics'97, In *J. Computer Graphics Forum*, 16(3), (1997), pp. 283-292.

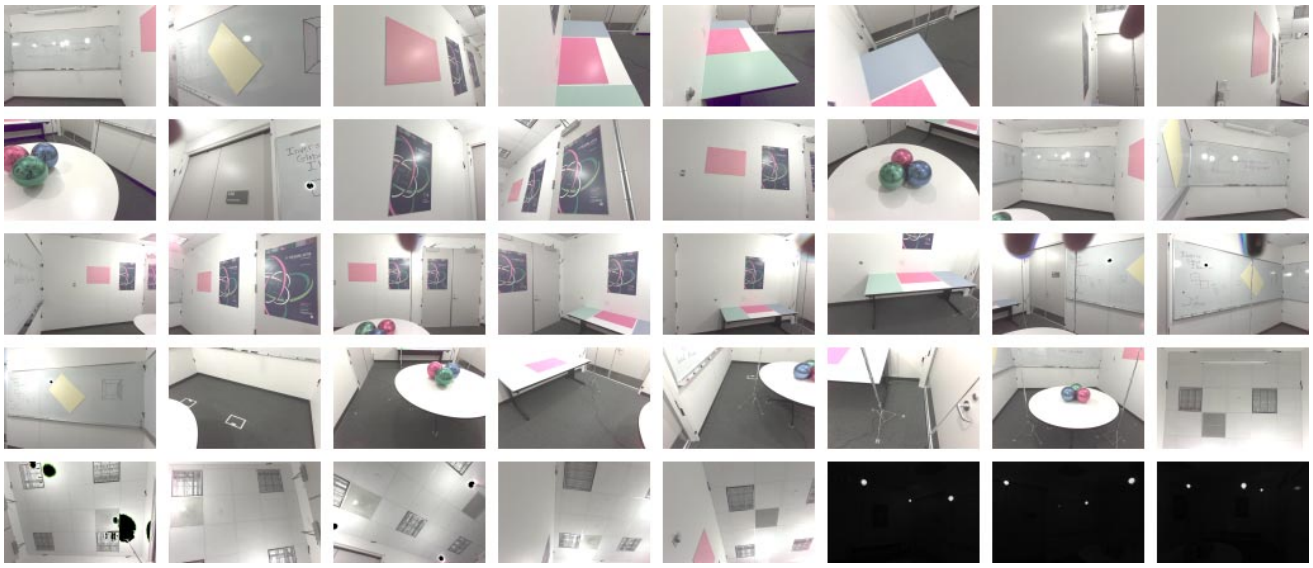


Figure 8: The complete set of forty radiance images of the room used to recover reflectance properties. Except for a few small areas, every surface in the room was seen in at least one radiance image. Each radiance image was constructed from between one and ten digital pictures depending on the dynamic range of the particular view. Black areas indicate regions which were saturated in all input images, and are not used by the recovery algorithm. The last three radiance images, reproduced ten stops darker than the rest, intentionally image the light bulbs. They were used to recover the positions and intensities of the sources.

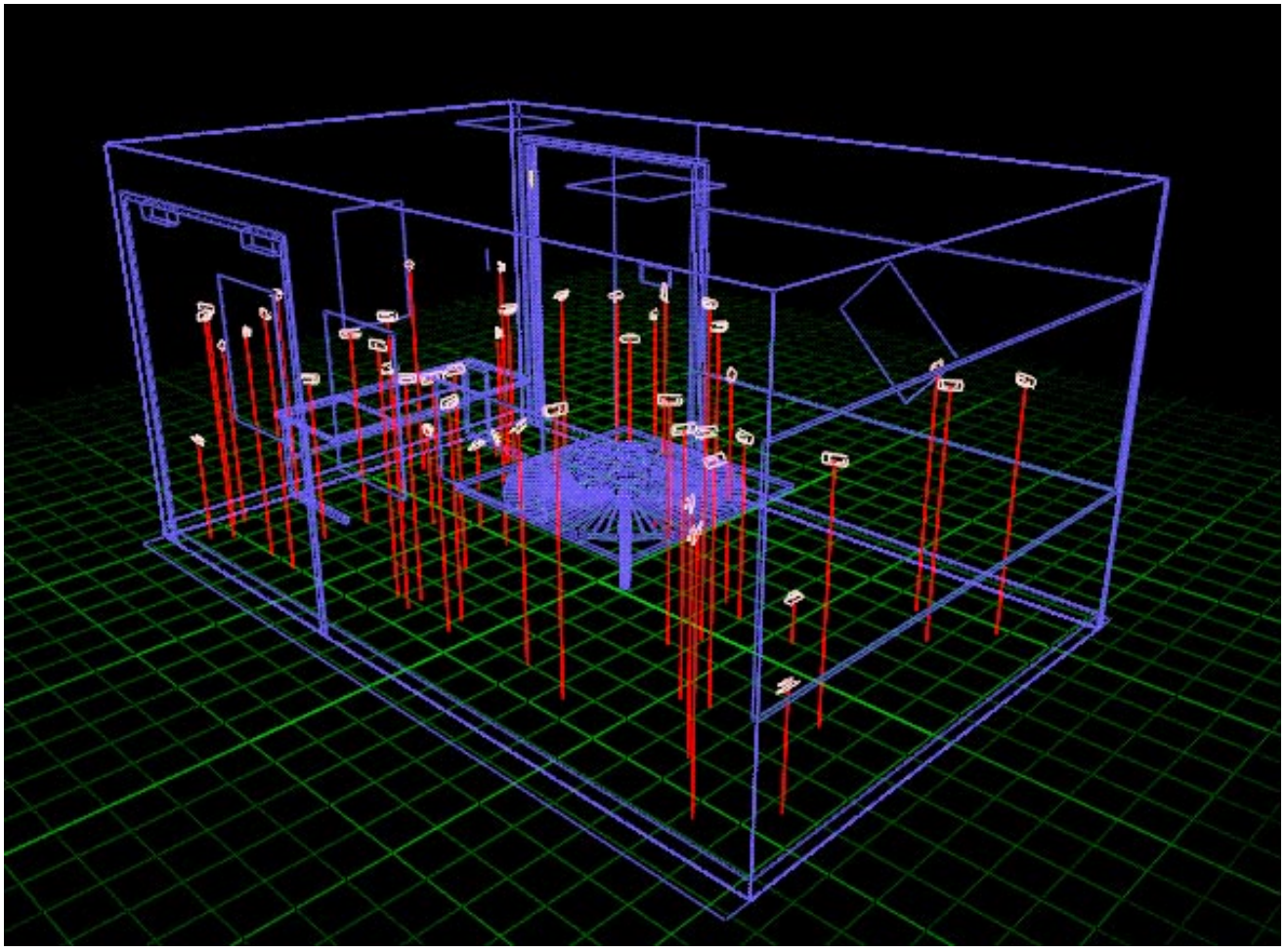


Figure 9: The model of the room, photogrammetrically recovered from the photographs in Fig 8. The recovered camera positions of the forty photographs are indicated.

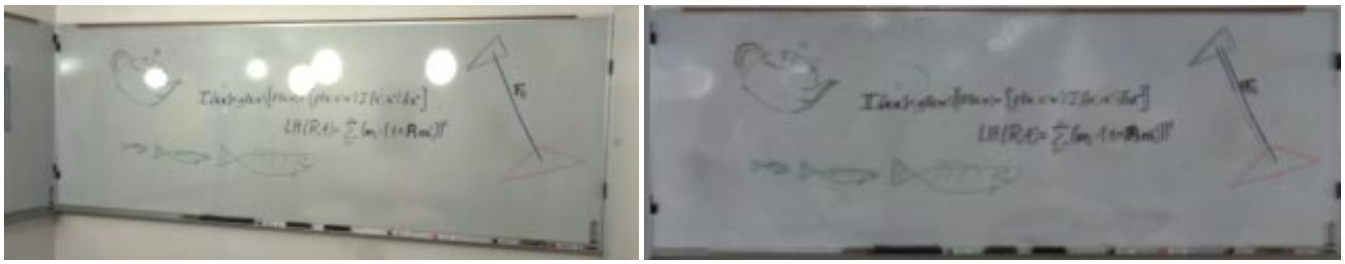


Figure 10: The left picture is a radiance image of a whiteboard, showing strong specular highlights. The right picture shows the diffuse albedo map of the whiteboard recovered from several images. Unlike the radiance image, the diffuse albedo map has a nearly uniform background, and is independent of the illumination.

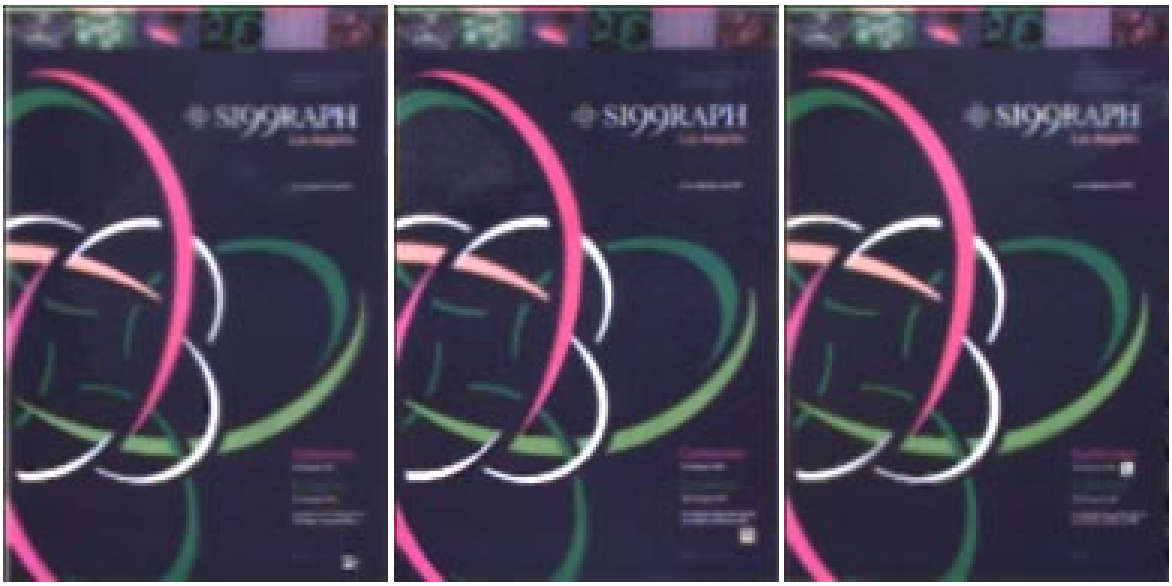


Figure 11: The diffuse albedo maps of three posters with the same texture. The posters were placed at different locations in the real scene with different illumination. Nonetheless, the recovered albedo maps are nearly the same. For identification purposes, a small yellow square was placed in a different location on the lower right of each poster.

Figure 12: The left image shows a part of a wall that becomes noticeably colored from light reflecting from the cards placed on the table below, an effect known as ”color bleeding”. The right image shows the recovered albedo map of the same part of the wall. It is nearly uniform, showing that the color bleeding was properly accounted for. The black line indicates where the table top aligned with the wall.

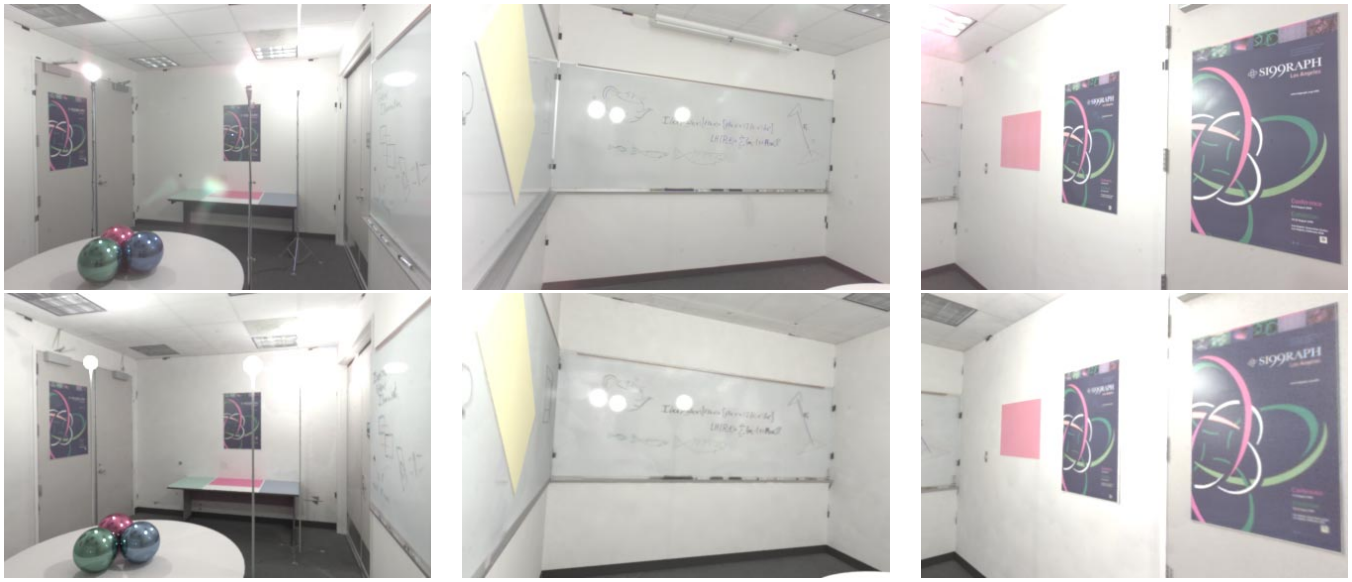


Figure 13: A comparison between real images (top) and synthetic renderings of our room with the recovered reflectance parameters (bottom). The simulated lighting is the same as in the original pictures, and the synthetic viewpoints have been matched to the recovered camera positions of the real images. The images show that good consistency was achieved.

Figure 14: A comparison between real and virtual, this time with novel lighting. Two of the lights were switched off and the third was moved to a new location. In addition, a real mirrored ball was placed on the red card. The scene was photographed from two locations and these real views are shown in the top row. To render the bottom row, we recovered the camera positions and light source position in the top views, estimated the material properties and position of the ball, and added a virtual ball to the model. The main noticeable difference is camera glare; however, some inaccuracies in the model (e.g. the whiteboard marker tray was not modeled) are also apparent. Otherwise, the illumination of the scene and appearance and shadows of the synthetic object are largely consistent.

(a) Initial hierarchical polygon mesh, with radiances assigned from images.

(b) Synthetic rendering of recovered properties under original illumination.

(c) Synthetic rendering of room under novel illumination.

(d) Synthetic rendering of room with seven virtual objects added.

Figure 15: Panoramic renderings of the room, with various changes to lighting and geometry.

Video Based Animation Techniques for Human Motion

Chris Bregler, Stanford University

Most image based rendering techniques are applied to rigid domains: Static environment maps, indoor scenes, or architectural scenes. Explicit geometric structures are combined with image data. Texture mapping and view morphing are simple examples. We can generate new images from a collection of recorded images. Simple geometry dictates coarse transformations of fine grained image texture. New views of a scene can be generated in blending between the transformed example textures. This is a trade-off between explicit structure (collection of views and geometric model) and implicit example data (the image texture).

Such trade-offs are applied to other domains as well. The most successful speech production systems (text-to-speech, concatenative speech) follow a similar philosophy. A collection of annotated example sounds are used to create new sounds. A sentence is build from phonemes (explicit structure). To blend the phonemes together, the sound examples are pitch and time warped (implicit data). We will show how this extends to video data and human motion animation.

Structure vs Data for Animation:

So far most graphical animation techniques do not exploit such trade-offs between explicit structure and implicit data. Many facial and body animations are generated by 3D volumetric models and physical simulations. Some facial animation systems texture map images onto the geometric model, or morph between a few example images. The appearance and motion become increasingly realistic. We will survey some of these systems.

In contrast to physical simulations, motion capture based animation techniques become increasingly popular. An actor performs the desired motion, and devices record body joint configurations or facial configurations. This data is mapped onto graphical computer models. Motion editing techniques allow to modify the motion data and create new motions. This has similarities to image morphing techniques. Instead of warping image texture, spatio-temporal configurations are warped.

Some systems allow to blend between different motion-capture data sets of different actions. New animations are assembled using existing examples.

Video Based Animation of People:

In order to create animations, that have natural motion **AND** have photo-realistic appearance, we need to combine motion-capture and image based (or video based) techniques. The goal is to build video based representations of annotated example motions.

Unlike standard motion capture techniques that are based on markers or other devices, we need to annotate body and facial configurations directly in unconstrained video. In static scenes the user could supply annotations by hand, but for video sequences, automatic techniques are crucial (10 min of video has 18,000 images, no-one has the budged, patience, and consistency to do this by hand). We will survey several visual tracking and annotation techniques that are tailored for full body movements and facial

movements. We demonstrate these visual annotation techniques on lab recordings of people walking and talking. We also demonstrate how to process historic footage. Examples are the famous Edweard Muybridge Plates from over 100 years ago of walking people, and stock-footage of John F. Kennedy giving a public speech.

To build libraries of example motions, we also need techniques that annotate coarse motion categories automatically. Again, this has to be done automatically. For example a 10 minute video of someone talking could be transformed into a video-based library of more then 2,000 phonetic lip motions (phonemes or visemes).

Once they are annotated, we can re-animate the data or create new data. We will present work-in-progress of photo-realistic animations of kinematic chain models, and we will cover in more detail work presented at last years SIGGRAPH conference on photo-realistic animation of talking heads: Video Rewrite.

More technical details of such techniques can be found in following papers:

- Video Rewrite: C. Bregler, M. Covell, M. Slaney

  Video Rewrite uses existing footage to create automatically new video of a person mouthing words that she did not speak in the original footage. This technique is useful in movie dubbing, for example, where the movie sequence can be modified to sync the actors' lip motions to the new soundtrack.
  Video Rewrite automatically labels the phonemes in the training data and in the new audio track. Video Rewrite reorders the mouth images in the training footage to match the phoneme sequence of the new audio track. When particular phonemes are unavailable in the training footage, Video Rewrite selects the closest approximations. The resulting sequence of mouth images is stitched into the background footage. This stitching process automatically corrects for differences in head position and orientation between the mouth images and the background footage.
  Video Rewrite uses computer-vision techniques to track points on the speaker's mouth in the training footage, and morphing techniques to combine these mouth gestures into the final video sequence. The new video combines the dynamics of the original actor's articulations with the mannerisms and setting dictated by the background footage.
  Video Rewrite is the first facial-animation system to automate all the labeling and assembly tasks required to resync existing footage to a new soundtrack.

- Video Motion Capture: C. Bregler, J. Malik

  This paper demonstrates a new vision based motion capture technique that is able to recover high degree-of-freedom articulated human body configurations in complex video sequences. It does not require any markers, body suits, or other devices attached to the subject. The only input needed is a video recording of the person whose motion is to be captured. For visual tracking we introduce the use of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation. This results in solving simple linear systems, and enables us to recover robustly the kinematic degrees-of-freedom in noise and complex self occluded configurations. We demonstrate this on several image sequences of people doing articulated full body movements, and visualize the results in re-animating an artificial 3D human model. We are also able to recover and re-animate the famous movements of Eadweard

Muybridge's motion studies from the last century. To the best of our knowledge, this is the first computer vision based system that is able to process such challenging footage and recover complex motions with such high accuracy.

Preliminary slides in PDF

References

- [Beier92] T. Beier, S. Neely. Feature-based image metamorphosis. SIGGRAPH 92, 26(2):35-42,1992.
- [Ezzat98] T. Ezzat, T. Poggio. MikeTalk: A Talking Facial Display Based on Morphing Visemes. Proc. Computer Animation Conference, Philadelphia, Penssylvania, 1998. (CD-version)
- [Guenter98] B. Guenter, C. Grimm, D. Wood, H.Malvar, F.Pighin. Making Faces. SIGGRAPH 98, 55-66. (web-pointer)
- [Litwinowicz94] P. Litwinowicz, L. Williams. Animating images with drawings. SIGGRAPH 94, Orlando, FL, pp. 409-412,1994
- [Moulines90] E. Moulines, P. Emerard, D. Larreur, J.L. Le Saint Milon, L. Le Faucheur, F. Marty, F. Charpentier, C. Sorin. A real-time French text-to-speech system generating high-quality synthetic speech. Proc. Int. Conf. Acoustics, Speech, and Signal Processing, Albuquerque, MN, pp. 309-312, 1990.
- [Pighin98] F. Pighin, J. Hecker, D.Lischinski, R.Szeliski, D.H.Salesin. Synthesizing Realistic Facial Expressions from Photographs.
- [Scott94] K.C. Scott, D.S. Kagels, S.H. Watson, H. Rom, J.R. Wright, M. Lee, K.J. Hussey. Synthesis of speaker facial movement to match selected speech sequences. Proc. Australian Conf. Speech Science and Technology, Perth Australia, pp. 620-625, 1994. (CD-version)
- [Water95] K. Waters, T.Levergood. DECface: A System for Synthetic Face Applications. J. Multimedia Tools and Applications, 1(4):349-366, 1995.(web-pointer)
- [Williams90] L. Williams. Performance-Driven Facial Animation. SIGGRAPH 90, 24(4):235-242, 1990.

IBMR Techniques for Animating People

Christoph Bregler

Computer Science Department
Stanford University

Static Scenes

- Drawing
- Rendering of Models
- IBMR

Disney

Hodgins et al

Animation

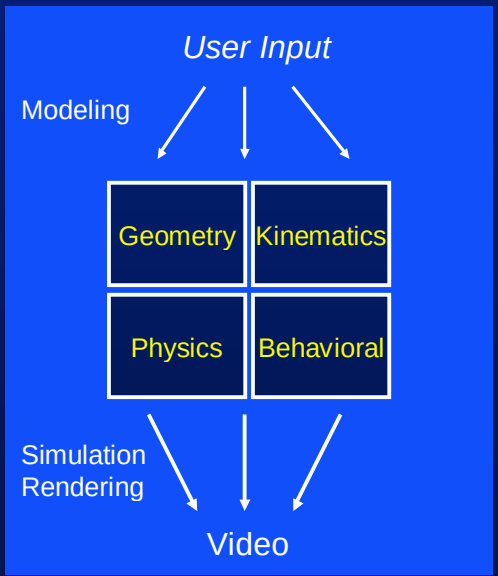
- Key-Framing
- Simulation of Models
- Motion Capture/ Video Based

DATA based Modeling and Rendering

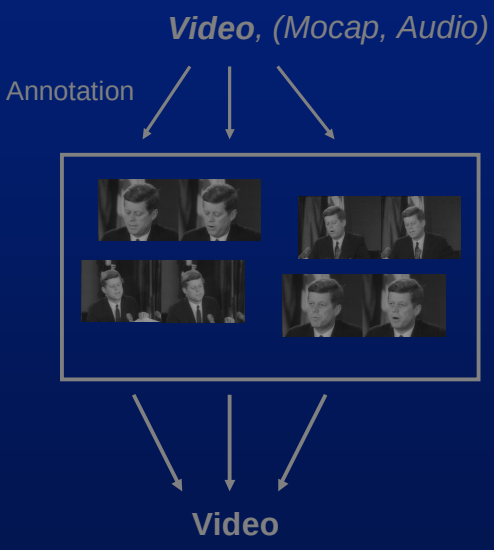
DATA =

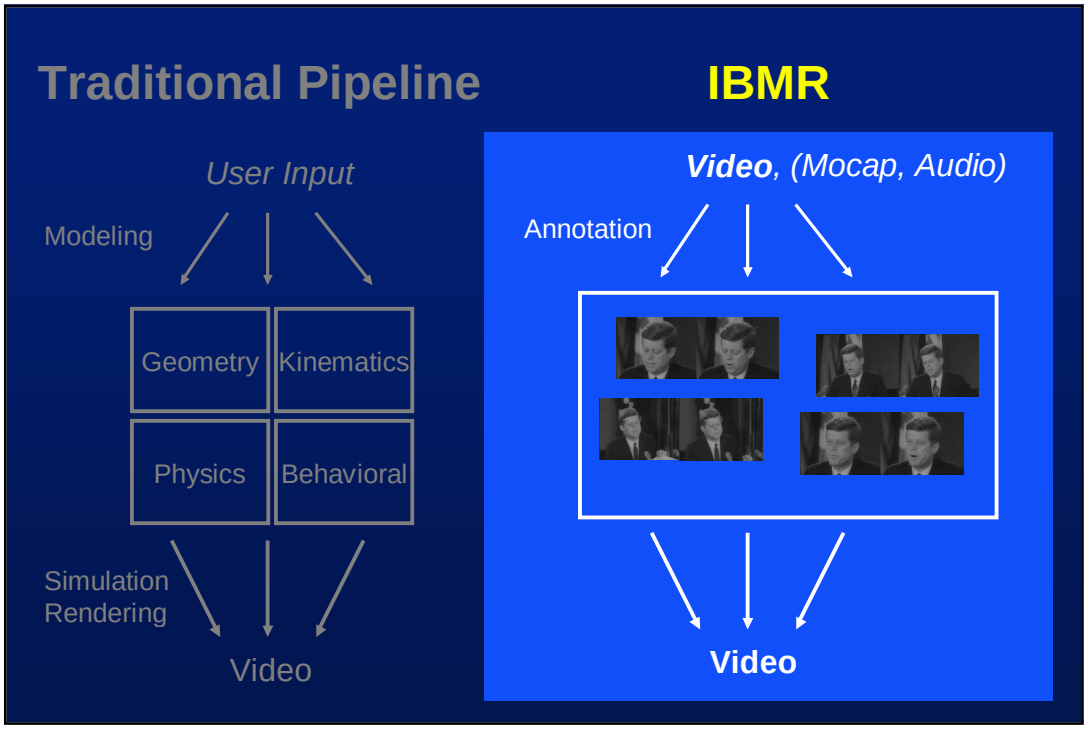
- *Images*
- *Range Scans*
- *Video*
- *Motion*
- *Audio*

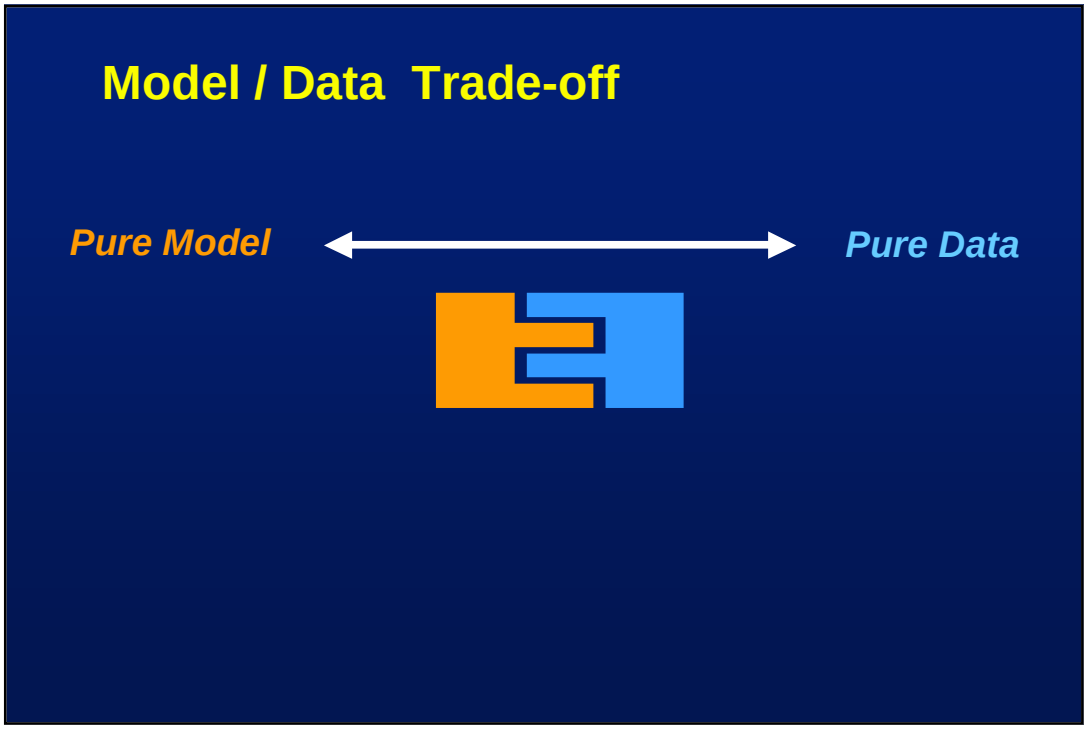
Traditional Pipeline



IBMR




Traditional Pipeline
IBMR
User Input
Modeling
Geometry
Kinematics
Physics
Behavioral
Simulation
Rendering
Video
Video, (Mocap, Audio)
Annotation
Video


Model / Data Trade-off
Pure Model
Pure Data

Model / Data Trade-off

Pure Model ⟷ *Pure Data*

Models =

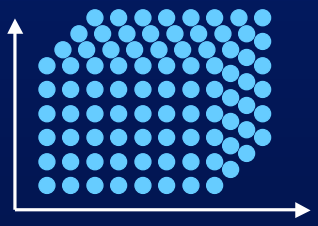
- *Geometric Models*
- *Kinematic Models*
- *Lighting Models*
- *Statistical Models*

Model / Data Trade-off

Pure Model ⟷ *Pure Data*

Models =

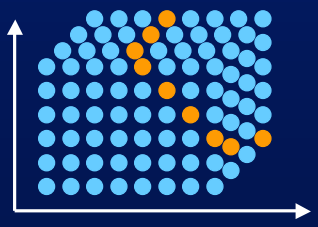
- *Geometric Models*
- *Kinematic Models*
- *Lighting Models*
- *Statistical Models*



Model / Data Trade-off

Pure Model ⟷ *Pure Data*

Subspaces in High-Dimensional Image Spaces

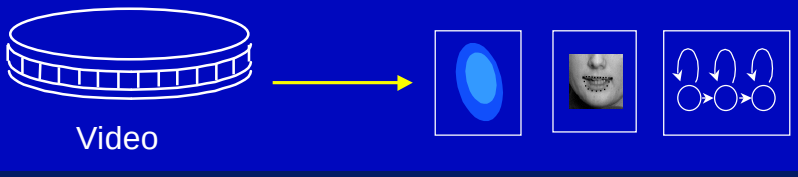


IBMR techniques for Animating People

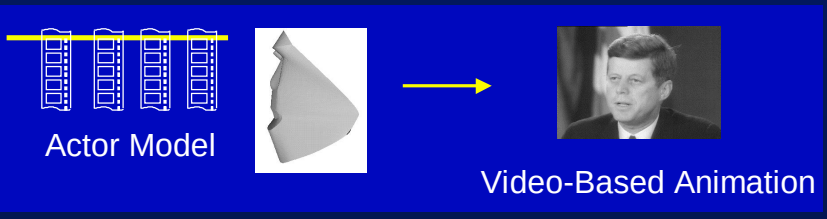
- ***Most Progress in Facial Modeling and Animation:***
  - ***Image Morphing:***
    Beyer+Neely, Seitz+Dyer, Litwinowicz+Williams
  - ***3D-Model + Images:***
    Terzopoulos+Waters, Pighin et al., Guenter et al.
  - ***Images + Lighting:***
    Raviv+Shashua
  - ***Audio-Data + Images:***
    Scott et al., Ezzat+Poggio, Cossato+Graf, Chen et al.

Outline

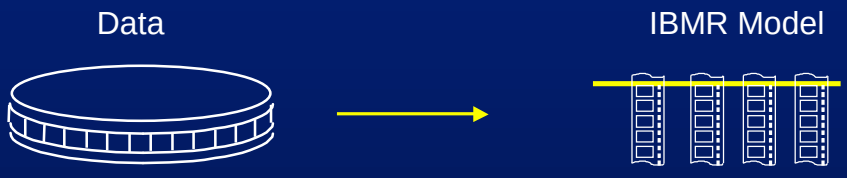
Analysis



Synthesis



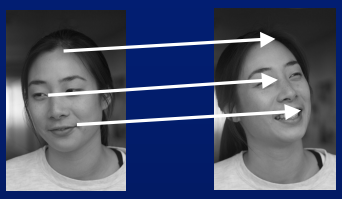
Challenges



10 min =
18,000 images

Visual Tracking

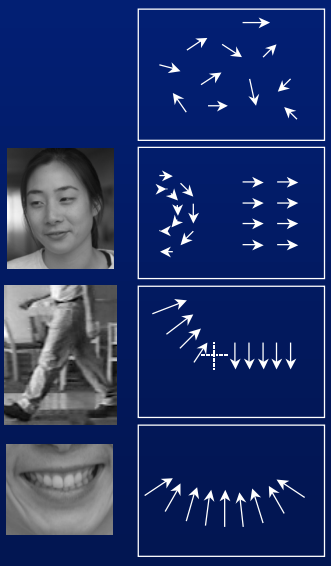
Visual Tracking



In General Very Challening

-> Make Generic “Model” Assumtions

Tracking: Motion Categories



- Optical Flow: no constraints

  [Horn81,Lucas81]

- Layered Motion: rigid constraints

  [Bergen92,Jepson93,Adelson95, Weiss96]

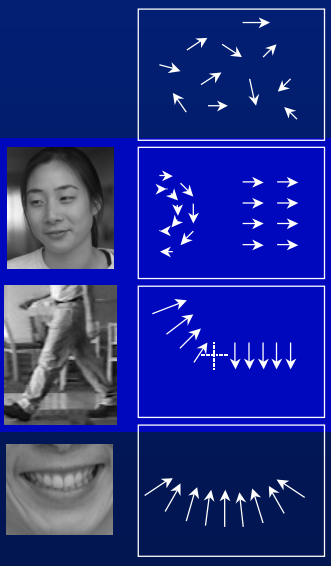
- Articulated: kinematic chain constraints

  [Yamamoto91,Rehg95,Kakadiaris96]

- Nonrigid: learned constraints

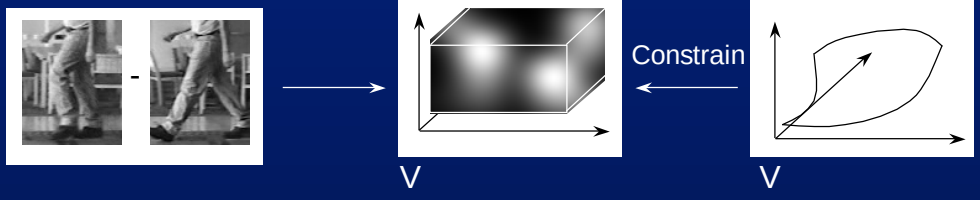
  [Eigen-XXX]

Tracking: Motion Categories



- Optical Flow: no constraints
- Layered Motion: rigid constraints
- Articulated: kinematic chain constraints
- Nonrigid: learned constraints

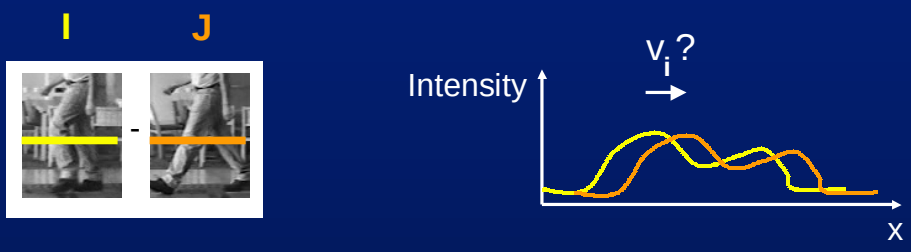
Tracking = Optimization



E(V)

$V = M(\theta)$

Function Minimization



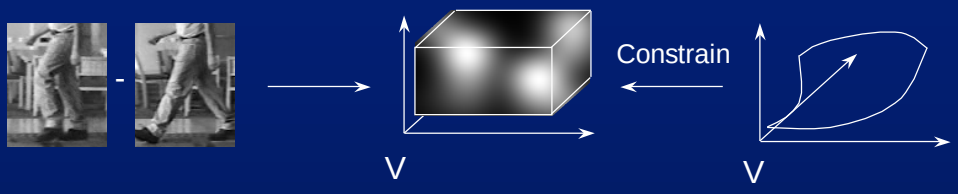
E(V)

$$\left\| \mathbf{I}(x_i) - \mathbf{J}(x_i + v_i) \right\|^2$$

linearize

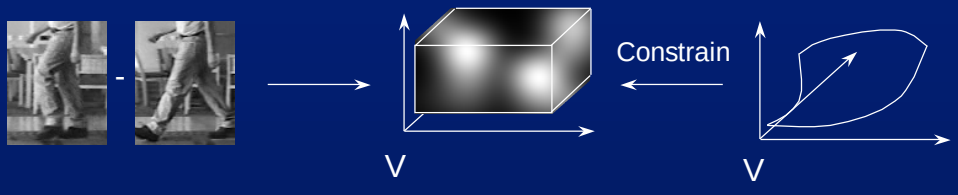
$$\left\| \mathbf{I}_t(x_i) - \nabla \mathbf{I}(x_i)\, v_i \right\|^2$$

Function Minimization



$$\left\| \begin{array}{l} I_t(1) - \nabla I(1)\,\vec{v}_1 \\ I_t(2) - \nabla I(2)\,\vec{v}_2 \\ \quad \ldots \\ I_t(n) - \nabla I(n)\,\vec{v}_n \end{array} \right\|^2 = E(V)$$

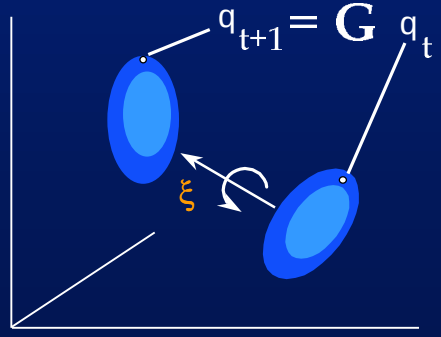
Constrained Function Minimization



$$\left\| \begin{array}{l} I_t(1) - \nabla I(1)\,\vec{v}_1 \\ I_t(2) - \nabla I(2)\,\vec{v}_2 \\ \quad \ldots \\ I_t(n) - \nabla I(n)\,\vec{v}_n \end{array} \right\|^2 = E(V) \qquad V = M(\theta)$$

Twist Motion / Exponential Map

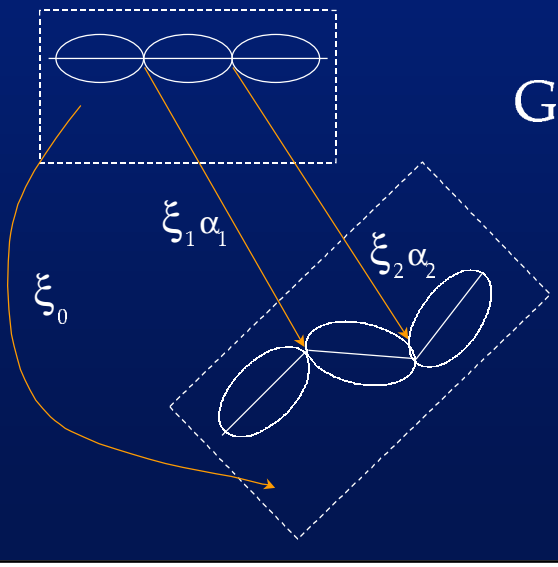
Murray, Li, Sastry



$$G = \begin{bmatrix} r1 & r2 & r3 & dx \\ r4 & r5 & r6 & dy \\ r7 & r8 & r9 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} = e^{\hat{\xi}}$$

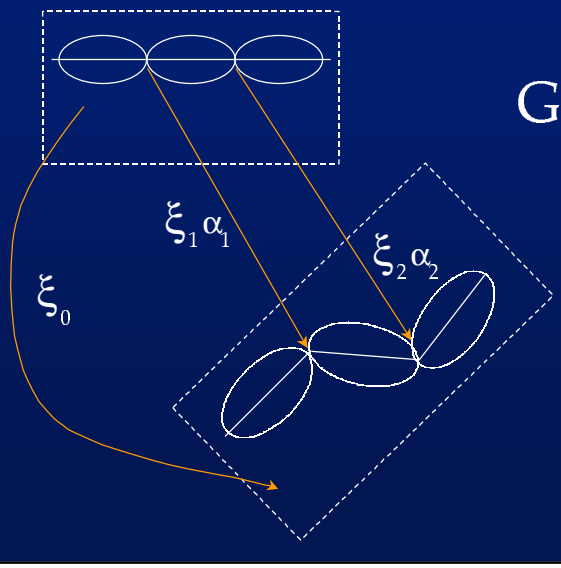
$$\hat{\xi} = \begin{bmatrix} 0 & -\omega 3 & \omega 2 & \upsilon 1 \\ \omega 3 & 0 & -\omega 1 & \upsilon 2 \\ -\omega 2 & \omega 1 & 0 & \upsilon 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Product of Exponential Map



$$G = e^{\hat{\xi}_0} e^{\hat{\xi}_1 \alpha_1} e^{\hat{\xi}_2 \alpha_2}$$

Product of Exponential Map



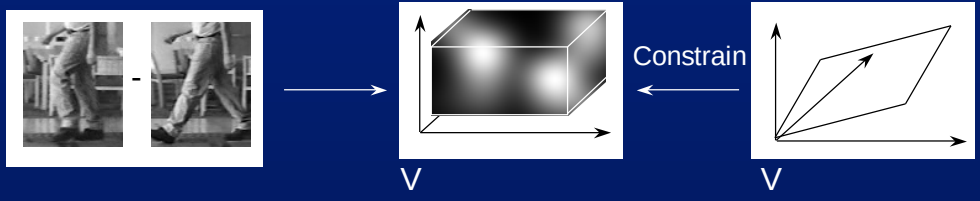
$$G = e^{\hat{\xi}_0} e^{\hat{\xi}_1 \alpha_1} e^{\hat{\xi}_2 \alpha_2}$$

$$\frac{\delta}{\delta t}$$

$$v_i = M_i \begin{bmatrix} \dot{\xi} \\ \dot{\alpha} \end{bmatrix}$$

Video Motion Capture

Bregler,Malik 98

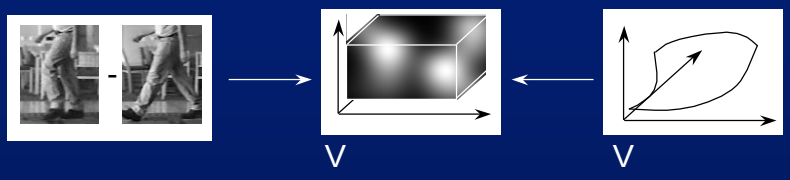


$$\left\| Z + H V \right\|^2$$

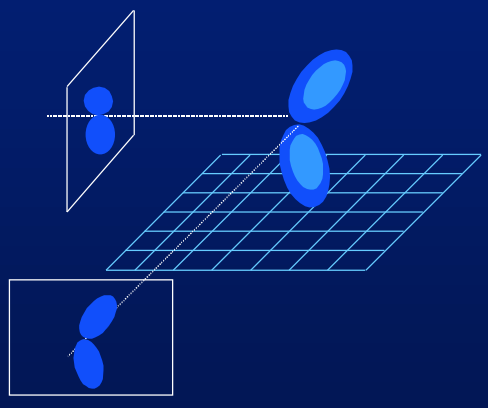
$$V = M \theta$$

$$\left\| Z + C \theta \right\|^2$$

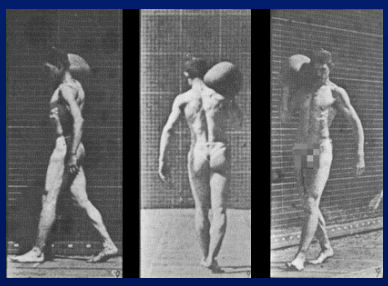
Example Track

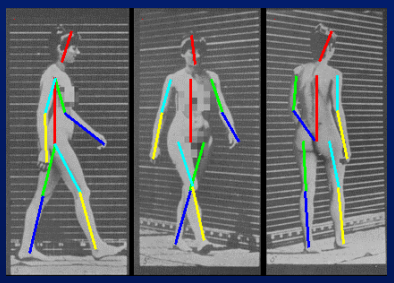


Multiple Views

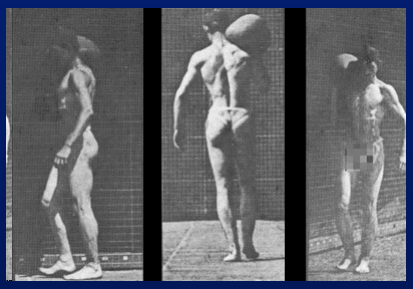


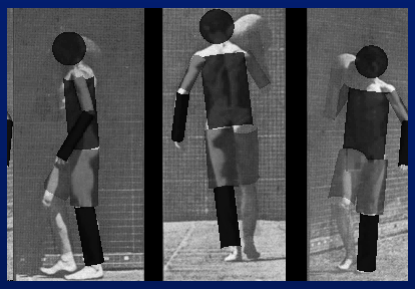
Eadweard Muybridge



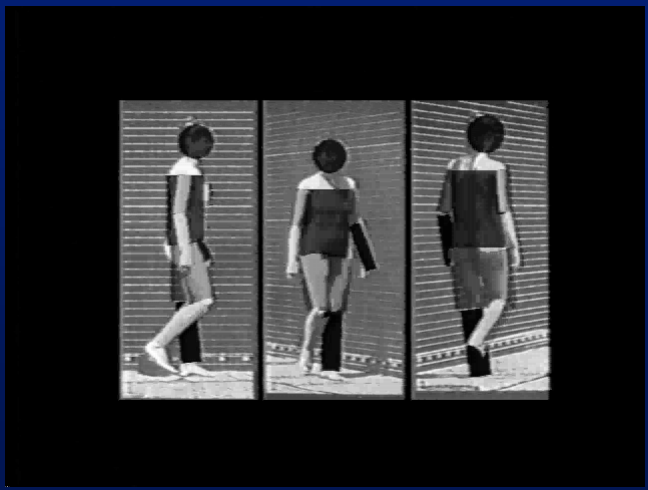


Example Track



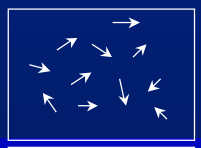


Example Track -2-

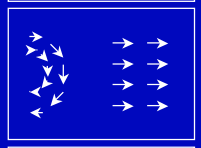


By Charles Ying

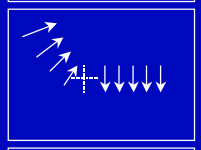
Tracking: Motion Categories



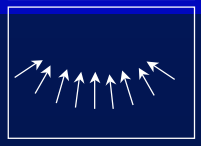
• Optical Flow: no constraints



• Layered Motion: rigid constraints



• Articulated: kinematic chain constraints

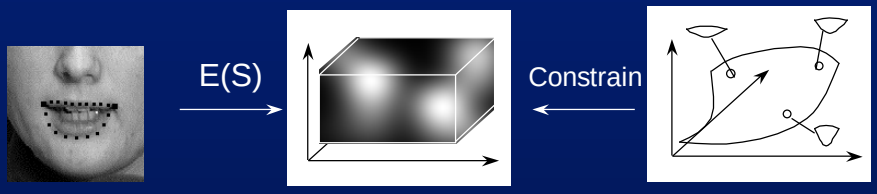


• Nonrigid: learned constraints

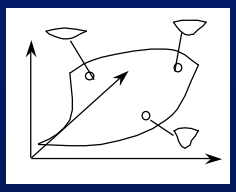
Tracking: Motion Categories

- Optical Flow: no constraints
- Layered Motion: rigid constraints
- Articulated: kinematic chain constraints
- Nonrigid: learned constraints

Non-Rigid Constrained Spaces

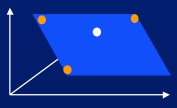


Non-Rigid Constrained Spaces

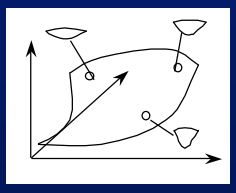


Linear Subspaces:

- Basis Set of Example Images

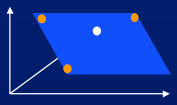


Non-Rigid Constrained Spaces

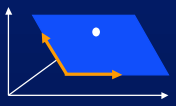


Linear Subspaces:

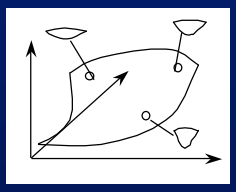
- Basis Set of Example Images



- Principal Components Analysis

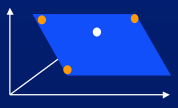


Non-Rigid Constrained Spaces

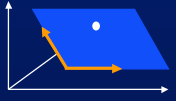


Linear Subspaces:

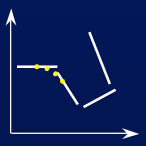
- Example Basis Set



- Principal Components Analysis

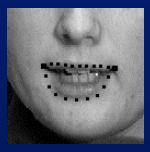


Nonlinear Manifolds:

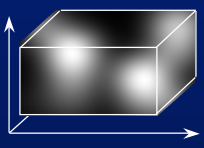


Mixture Models

Constrained Tracking



E(S)

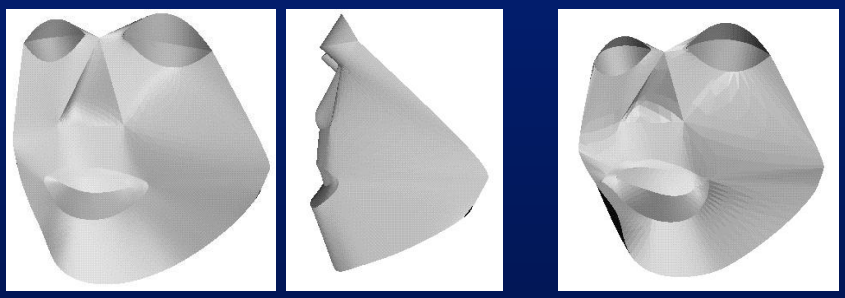


Constrain

$S = (p_1,\ldots,p_n)$

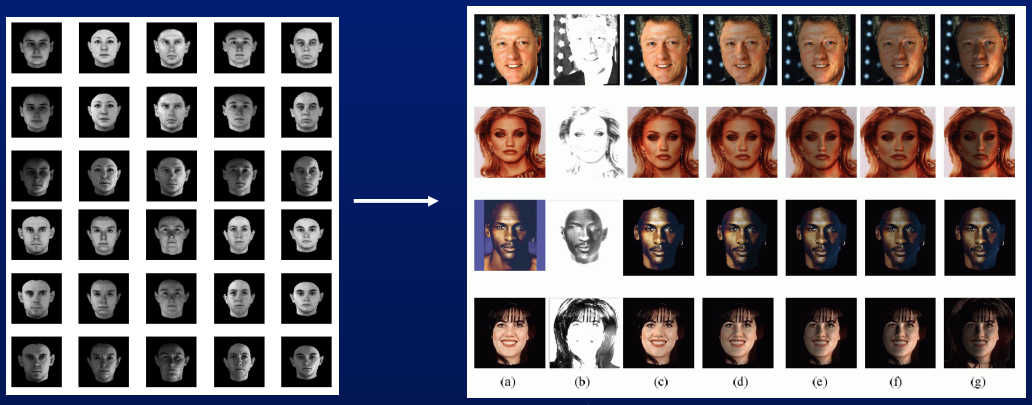
Example Track -3-

Building 3D Nonrigid Face Models



Bregler, Hertzmann, Biermann, ***NYU***

Image-Based Lighting for Faces
Riklin-Raviv + Shashua

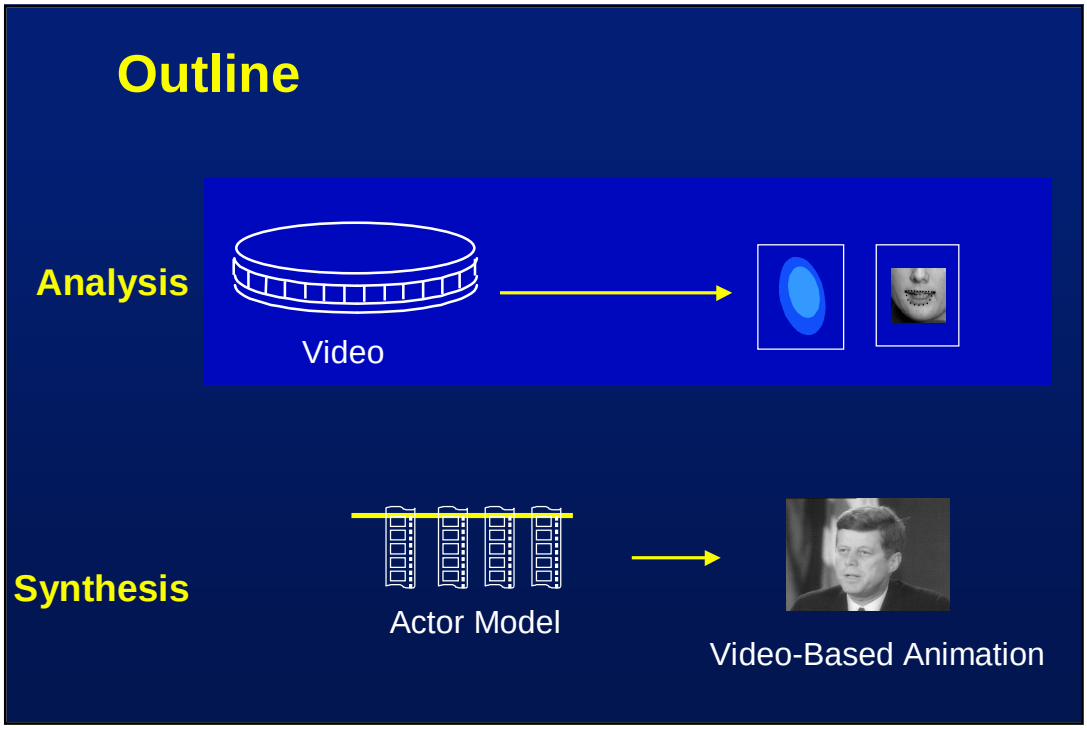


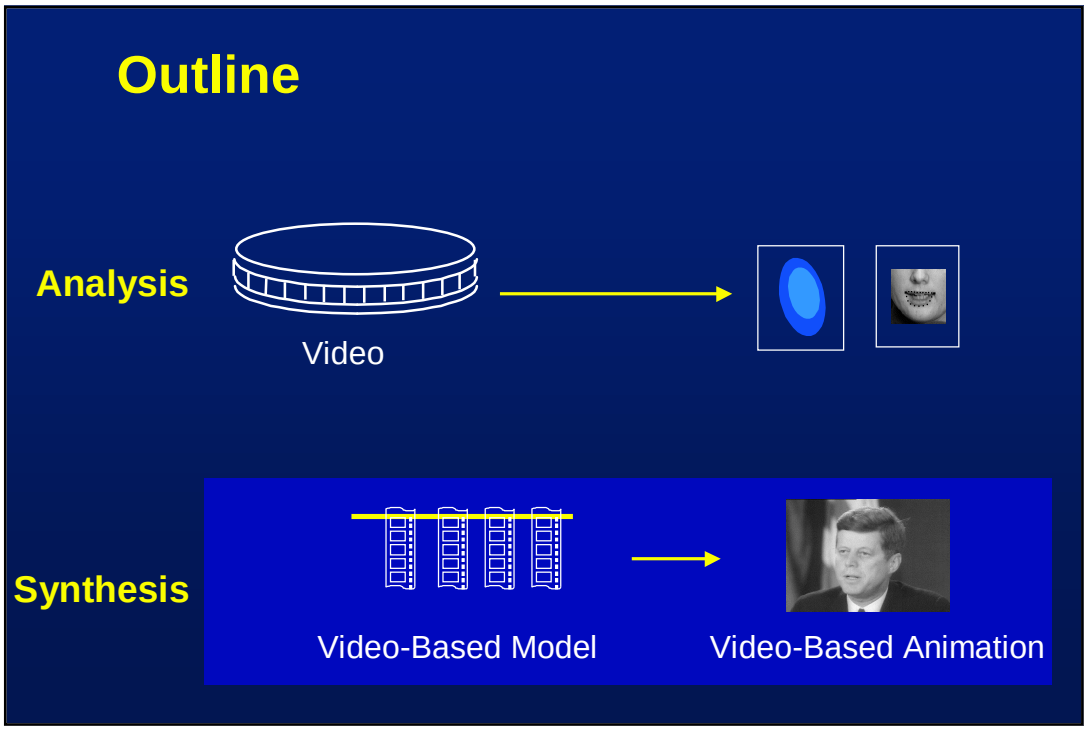
Bootstrap Database

Quotient-Image

Image-Based Lighting for Faces
Debevec et al.

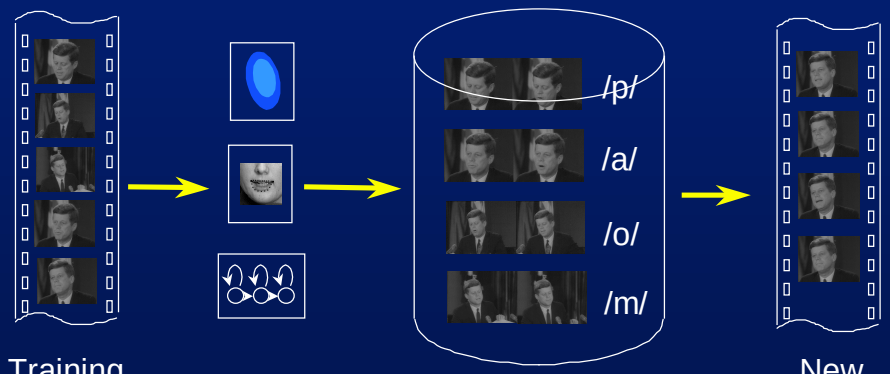
See ***Light-Stage***


Outline
Analysis
Video
Synthesis
Actor Model
Video-Based Animation


Outline
Analysis
Video
Synthesis
Video-Based Model
Video-Based Animation

Video Rewrite

Bregler, Covell, Slaney, *Interval Research*



Training

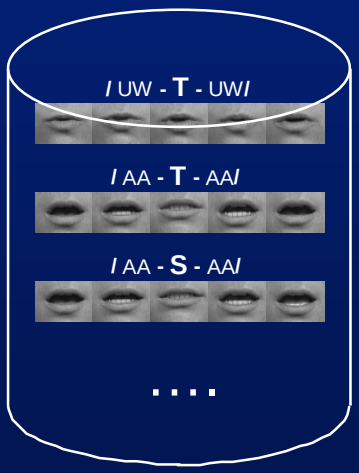
New

Video Model:

10

8 min =

1,700 triphones



"Ellen Model"

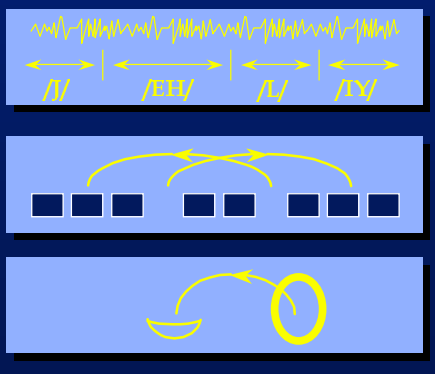
11

Synthesis - Overview -

12

Synthesis:

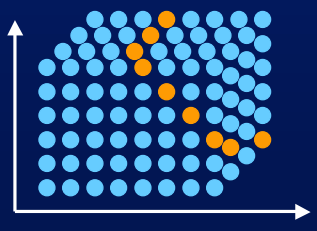
- *Transcribe*
- *Find Lip Clips*
- *Stitch Together*



Example 2D Animation

Model / Data Trade-off

Pure Model ⟷ *Pure Data*



Video Rewrite: Driving Visual Speech with Audio

Christoph Bregler, Michele Covell, Malcolm Slaney
Interval Research Corporation

ABSTRACT

Video Rewrite uses existing footage to create automatically new video of a person mouthing words that she did not speak in the original footage. This technique is useful in movie dubbing, for example, where the movie sequence can be modified to sync the actors' lip motions to the new soundtrack.

Video Rewrite automatically labels the phonemes in the training data and in the new audio track. Video Rewrite reorders the mouth images in the training footage to match the phoneme sequence of the new audio track. When particular phonemes are unavailable in the training footage, Video Rewrite selects the closest approximations. The resulting sequence of mouth images is stitched into the background footage. This stitching process automatically corrects for differences in head position and orientation between the mouth images and the background footage.

Video Rewrite uses computer-vision techniques to track points on the speaker's mouth in the training footage, and morphing techniques to combine these mouth gestures into the final video sequence. The new video combines the dynamics of the original actor's articulations with the mannerisms and setting dictated by the background footage. Video Rewrite is the first facial-animation system to automate all the labeling and assembly tasks required to resync existing footage to a new soundtrack.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Morphing; I.4.6 [Image Processing]: Segmentation—Feature Detection; I.3.8 [Computer Graphics]: Applications—Facial Synthesis; I.4.10 [Image Processing]: Applications—Feature Transformations.

Additional Keywords: Facial Animation, Lip Sync.

1 WHY AND HOW WE REWRITE VIDEO

We are very sensitive to the synchronization between speech and lip motions. For example, the special effects in *Forest Gump* are compelling because the Kennedy and Nixon footage is lip synched to the movie's new soundtrack. In contrast, close-ups in dubbed movies are often disturbing due to the lack of lip sync. Video Rewrite is a system for automatically synthesizing faces with proper lip sync. It can be used for dubbing movies, teleconferencing, and special effects.

1801 Page Mill Road, Building C, Palo Alto, CA, 94304. E-mail: bregler@cs.berkeley.edu, covell@interval.com, malcolm@interval.com. See the SIGGRAPH Video Proceedings or http://www.interval.com/papers/1997-012/ for the latest animations.

Video Rewrite automatically pieces together from old footage a new video that shows an actor mouthing a new utterance. The results are similar to labor-intensive special effects in *Forest Gump*. These effects are successful because they start from actual film footage and modify it to match the new speech. Modifying and reassembling such footage in a smart way and synchronizing it to the new sound track leads to final footage of realistic quality. Video Rewrite uses a similar approach but does not require labor-intensive interaction.

Our approach allows Video Rewrite to learn from example footage how a person's face changes during speech. We learn what a person's mouth looks like from a video of that person speaking normally. We capture the dynamics and idiosyncrasies of her articulation by creating a database of video clips. For example, if a woman speaks out of one side of her mouth, this detail is recreated accurately. In contrast, most current facial-animation systems rely on generic head models that do not capture the idiosyncrasies of an individual speaker.

To model a new person, Video Rewrite requires a small number (26 in this work) of hand-labeled images. This is the only human intervention that is required in the whole process. Even this level of human interaction is not a fundamental requirement: We could use face-independent models instead [Kirby90, Covell96].

Video Rewrite shares its philosophy with concatenative speech synthesis [Moulines90]. Instead of modeling the vocal tract, concatenative speech synthesis analyzes a corpus of speech, selects examples of phonemes, and normalizes those examples. Phonemes are the distinct sounds within a language, such as the /IY/ and /P/ in "teapot." Concatenative speech synthesizes new sounds by concatenating the proper sequence of phonemes. After the appropriate warping of pitch and duration, the resulting speech is natural sounding. This approach to synthesis is data driven: The algorithms analyze and resynthesize sounds using little hand-coded knowledge of speech. Yet they are effective at implicitly capturing the nuances of human speech.

Video Rewrite uses a similar approach to create new sequences of visemes. Visemes are the visual counterpart to phonemes. Visemes are visually distinct mouth, teeth, and tongue articulations for a language. For example, the phonemes /B/ and /P/ are visually indistinguishable and are grouped into a single viseme class.

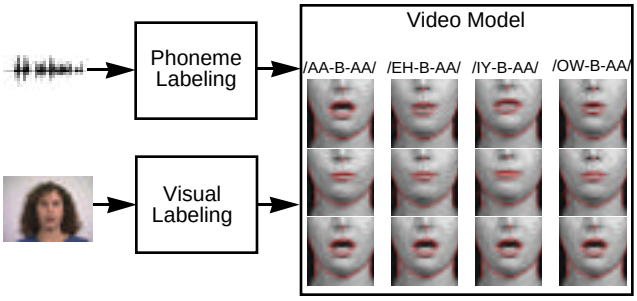


Figure 1: Overview of analysis stage. Video Rewrite uses the audio track to segment the video into triphones. Vision techniques find the orientation of the head, and the shape and position of the mouth and chin in each image. In the synthesis stage, Video Rewrite selects from this video model to synchronize new lip videos to any given audio.

Video Rewrite creates new videos using two steps: analysis of a training database and synthesis of new footage. In the *analysis* stage, Video Rewrite automatically segments into phonemes the audio track of the training database. We use these labels to segment the video track as well. We automatically track facial features in this segmented footage. The phoneme and facial labels together completely describe the visemes in the training database. In the *synthesis* stage, our system uses this video database, along with a new utterance. It automatically retrieves the appropriate viseme sequences, and blends them into a background scene using morphing techniques. The result is a new video with lip and jaw movements that synchronize to the new audio. The steps used in the analysis stage are shown in Figure 1; those of the synthesis stage are shown in Figure 2.

In the remainder of this paper, we first review other approaches to synthesizing talking faces (Section 2). We then describe the analysis and synthesis stages of Video Rewrite. In the analysis stage (Section 3), a collection of video is analyzed and stored in a database that matches sounds to video sequences. In the synthesis stage (Section 4), new speech is labeled, and the appropriate sequences are retrieved from the database. The final sections of this paper describe our results (Section 5), future work (Section 6), and contributions (Section 7).

2 SYNTHETIC VISUAL SPEECH

Facial-animation systems build a model of what a person's speech sounds and looks like. They use this model to generate a new output sequence, which matches the (new) target utterance. On the model-building side (analysis), there are typically three distinguishing choices: how the facial appearance is learned or described, how the facial appearance is controlled or labeled, and how the viseme labels are learned or described. For output-sequence generation (synthesis), the distinguishing choice is how the target utterance is characterized. This section reviews a representative sample of past research in these areas.

2.1 Source of Facial Appearance

Many facial-animation systems use a generic 3D mesh model of a face [Parke72, Lewis91, Guiard-Marigny94], sometimes adding texture mapping to improve realism [Morshima91, Cohen93, Waters95]. Another synthetic source of face data is hand-drawn images [Litwinowicz94]. Other systems use real faces for their source examples, including approaches that use 3D scans [Williams90] and still images [Scott94]. We use video footage to train Video Rewrite's models.

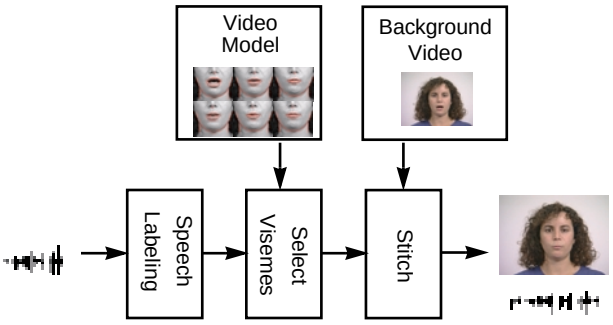


Figure 2: Overview of synthesis stage. Video Rewrite segments new audio and uses it to select triphones from the video model. Based on labels from the analysis stage, the new mouth images are morphed into a new background face.

2.2 Facial Appearance Control

Once a facial model is captured or created, the control parameters that exercise that model must be defined. In systems that rely on a 3D mesh model for appearance, the control parameters are the allowed 3D mesh deformations. Most of the image-based systems label the positions of specific facial locations as their control parameters. Of the systems that use facial-location labels, most rely on manual labeling of each example image [Scott94, Litwinowicz94]. Video Rewrite creates its video model by automatically labeling specific facial locations.

2.3 Viseme Labels

Many facial-animation systems label different visual configurations with an associated *phoneme*. These systems then match these phoneme labels with their corresponding labels in the target utterance. With synthetic images, the phoneme labels are artificial or are learned by analogy [Morshima91]. For natural images, taken from a video of someone speaking, the phonemic labels can be generated manually [Scott94] or automatically. Video Rewrite determines the phoneme labels automatically (Section 3.1).

2.4 Output-Sequence Generation

The goal of facial animation is to generate an image sequence that matches a target utterance. When phoneme labels are used, those for the target utterance can be entered manually [Scott94] or computed automatically [Lewis91, Morshima91]. Another option for phoneme labeling is to create the new utterance with synthetic speech [Parke72, Cohen93, Waters95]. Approaches, that do not use phoneme labels include motion capture of facial locations that are artificially highlighted [Williams90, Guiard-Marigny94] and manual control by an animator [Litwinowicz94]. Video Rewrite uses a combination of phoneme labels (from the target utterance) and facial-location labels (from the video-model segments). Video Rewrite derives all these labels automatically.

Video Rewrite is the first facial-animation system to automate all these steps and to generate realistic lip-synched video from natural speech and natural images.

3 ANALYSIS FOR VIDEO MODELING

As shown in Figure 1, the analysis stage creates an annotated database of example video clips, derived from unconstrained footage. We refer to this collection of annotated examples as a video model. This model captures how the subject's mouth and jaw move during speech. These training videos are labeled automatically with the phoneme sequence uttered during the video, and with the locations of fiduciary points that outline the lips, teeth, and jaw.

As we shall describe, the phonemic labels are from a time-aligned transcript of the speech, generated by a hidden Markov model (HMM). Video Rewrite uses the phonemic labels from the HMM to segment the input footage into short video clips, each showing three phonemes or a triphone. These triphone videos, with the fiduciary-point locations and the phoneme labels, are stored in the video model.

In Sections 3.1 and 3.2, we describe the visual and acoustic analyses of the video footage. In Section 4, we explain how to use this model to synthesize new video.

3.1 Annotation Using Image Analysis

Video Rewrite uses any footage of the subject speaking. As her face moves within the frame, we need to know the mouth position and the lip shapes at all times. In the synthesis stage, we use this information to warp overlapping videos such that they have the same lip shapes, and to align the lips with the background face.

Manual labeling of the fiduciary points around the mouth and jaw is error prone and tedious. Instead, we use computer-vision techniques to label the face and to identify the mouth and its shape. A major hurdle to automatic annotation is the low resolution of the images. In a typical scene, the lip region has a width of only 40 pixels. Conventional contour-tracking algorithms [Kass87, Yuille89] work well on high-contrast outer lip boundaries with some user interaction, but fail on inner lip boundaries at this resolution, due to the low signal-to-noise ratios. Grayscale-based algorithms, such as eigenimages [Kirby90, Turk91], work well at low resolutions, but estimate only the location of the lips or jaw, rather than estimating the desired fiduciary points. Eigenpoints [Covell96], and other extensions of eigenimages [Lanitis95], estimate control points reliably and automatically, even in such low-resolution images. As shown in Figure 3, eigenpoints learns how fiduciary points move as a function of the image appearance, and then uses this model to label new footage.

Video Rewrite labels each image in the training video using a total of 54 eigenpoints: 34 on the mouth (20 on the outer boundary, 12 on the inner boundary, 1 at the bottom of the upper teeth, and 1 at the top of the lower teeth) and 20 on the chin and jaw line. There are two separate eigenpoint analyses. The first eigenspace controls the placement of the 34 fiduciary points on the mouth, using 50×40 pixels around the nominal mouth location, a region that covers the mouth completely. The second eigenspace controls the placement of the 20 fiduciary points on the chin and jaw line, using 100×75 pixels around the nominal chin-location, a region that covers the upper neck and the lower part of the face.

We created the two eigenpoint models for locating the fiduciary points from a small number of images. We hand annotated only 26 images (of 14,218 images total; about 0.2%). We extended the hand-annotated dataset by morphing pairs of annotated images to form intermediate images, expanding the original 26 to 351 annotated images without any additional manual work. We then derived eigenpoints models using this extended data set.

We use eigenpoints to find the mouth and jaw and to label their contours. The derived eigenpoint models locate the facial features using six basis vectors for the mouth and six different vectors for the jaw. Eigenpoints then places the fiduciary points around the feature locations: 32 basis vectors place points around the lips and 64 basis vectors place points around the jaw.

Eigenpoints assumes that the features (the mouth or the jaw) are undergoing pure translational motion. It does a comparatively poor job at modeling rotations and scale changes. Yet, Video Rewrite is designed to use unconstrained footage. We expect rotations and scale changes. Subjects may lean toward the camera or turn away from it, tilt their heads to the side, or look up from under their eyelashes.

To allow for a variety of motions, we warp each face image into a standard reference plane, prior to eigenpoints labeling. We find the global transform that minimizes the mean-squared error between a large portion of the face image and a facial template. We currently use an affine transform [Black95]. The mask shown in Figure 4 defines the support of the minimization integral. Once the best global mapping is found, it is inverted and applied to the image, putting that face into the standard coordinate frame. We then perform eigenpoints analysis on this pre-warped image to find the fiduciary points. Finally, we back-project the fiduciary points through the global warp to place them on the original face image.

The labels provided by eigenpoints allow us automatically to (1) build the database of example lip configurations, and (2) track the features in a background scene that we intend to modify. Section 4.2 describes how we match the points we find in step 1 to each other and to the points found in step 2.

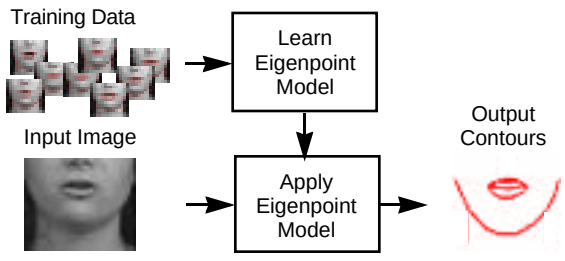


Figure 3: Overview of eigenpoints. A small set of hand-labeled facial images is used to train subspace models. Given a new image, the eigenpoint models tell us the positions of points on the lips and jaw.

3.2 Annotation Using Audio Analysis

All the speech data in Video Rewrite (and their associated video) are segmented into sequences of phonemes. Although single phonemes are a convenient representation for linguistic analysis, they are not appropriate for Video Rewrite. We want to capture the visual dynamics of speech. To do so correctly, we must consider *coarticulation*, which causes the lip shapes for many phonemes to be modified based on the phoneme's context. For example, the /T/ in "beet" looks different from the /T/ in "boot."

Therefore, Video Rewrite segments speech and video into triphones: collections of three sequential phonemes. The word "teapot" is split into the sequence of triphones /SIL-T-IY/ ,[1] /T-IY-P/ , /IY-P-AA/ , /P-AA-T/ , and /AA-T-SIL/ . When we synthesize a video, we emphasize the middle of each triphone. We cross-fade the overlapping regions of neighboring triphones. We thus ensure that the precise transition points are not critical, and that we can capture effectively many of the dynamics of both forward and backward coarticulation.

Video Rewrite uses HMMs [Rabiner89] to label the training footage with phonemes. We trained the HMMs using the TIMIT speech database [Lamel86], a collection of 4200 utterances with phonemic transcriptions that gives the uttered phonemes and their timing. Each of the 61 phoneme categories in TIMIT is modeled with a separate three-state HMM. The emission probabilities of each state are modeled with mixtures of eight Gaussians with diagonal covariances. For robustness, we split the available data by gender and train two speaker-independent, gender-specific systems, one based on 1300 female utterances, and one based on 2900 male utterances.

We used these gender-specific HMMs to create a fine-grained phonemic transcription of our input footage, using forced Viterbi

1. /SIL/ indicates silence. Two /SIL/ in a row are used at the beginnings and ends of utterances to allow all segments—including the beginning and end—to be treated as triphones.

Figure 4: Mask used to estimate the global warp. Each image is warped to account for changes in the head's position, size, and rotation. The transform minimizes the difference between the transformed images and the face template. The mask (left) forces the minimization to consider only the upper face (right).

search [Viterbi67]. Forced Viterbi uses unaligned sentence-level transcriptions and a phoneme-level pronunciation dictionary to create a time-aligned phoneme-level transcript of the speech. From this transcript, Video Rewrite segments the video automatically into triphone videos, labels them, and includes them in the video model.

4 SYNTHESIS USING A VIDEO MODEL

As shown in Figure 2, Video Rewrite synthesizes the final lip-synced video by labeling the new speech track, selecting a sequence of triphone videos that most accurately matches the new speech utterance, and stitching these images into a background video.

The background video sets the scene and provides the desired head position and movement. The background sequence in Video Rewrite includes most of the subject's face as well as the scene behind the subject. The frames of the background video are taken from the source footage in the same order as they were shot. The head tilts and the eyes blink, based on the background frames.

In contrast, the different triphone videos are used in whatever order is needed. They simply show the motions associated with articulation. For all the animations in this paper, the triphone images include the mouth, chin, and part of the cheeks, so that the chin and jaw move and the cheeks dimple appropriately as the mouth articulates. We use illumination-matching techniques [Burt83] to avoid visible seams between the triphone and background images.

The first step in synthesis (Figure 2) is labeling the new soundtrack. We label the new utterance with the same HMM that we used to create the video-model phoneme labels. In Sections 4.1 and 4.2, we describe the remaining steps: selecting triphone videos and stitching them into the background.

4.1 Selection of Triphone Videos

The new speech utterance determines the target sequence of speech sounds, marked with phoneme labels. We would like to find a sequence of triphone videos from our database that matches this new speech utterance. For each triphone in the new utterance, our goal is to find a video example with exactly the transition we need, and with lip shapes that match the lip shapes in neighboring triphone videos. Since this goal often is not reachable, we compromise by a choosing a sequence of clips that approximates the desired transitions and shape continuity.

Given a triphone in the new speech utterance, we compute a matching distance to each triphone in the video database. The matching metric has two terms: the *phoneme-context distance*, D_p, and the *distance between lip shapes* in overlapping visual triphones, D_s. The total error is

$$\text{error} = \alpha D_p + (1-\alpha) D_s,$$

where the weight, α, is a constant that trades off the two factors.

The phoneme-context distance, D_p, is based on categorical distances between phoneme categories and between viseme classes. Since Video Rewrite does not need to create a new soundtrack (it needs only a new video track), we can cluster phonemes into viseme classes, based on their visual appearance.

We use 26 viseme classes. Ten are consonant classes: (1) /CH/, /JH/, /SH/, /ZH/; (2) /K/, /G/, /N/, /L/; (3) /T/, /D/, /S/, /Z/; (4) /P/, /B/, /M/; (5) /F/, /V/; (6) /TH/, /DH/; (7) /W/, /R/; (8) /HH/; (9) /Y/; and (10) /NG/. Fifteen are vowel classes: one each for /EH/, /EY/, /ER/, /UH/, /AA/ /AO/, /AW/, /AY/, /UW/, /OW/, /OY/, /IY/, /IH/, /AE/, /AH/. One class is for silence, /SIL/.

The phoneme-context distance, D_p, is the weighted sum of phoneme distances between the target phonemes and the video-model phonemes within the context of the triphone. If the phonemic categories are the same (for example, /P/ and /P/), then this distance is 0. If they are in different viseme classes (/P/ and /IY/), then the distance is 1. If they are in different phonemic categories but are in the same viseme class (/P/ and /B/), then the distance is a value between 0 and 1. The intraclass distances are derived from published confusion matrices [Owens85].

In D_p, the center phoneme of the triphone has the largest weight, and the weights drop smoothly from there. Although the video model stores only triphone images, we consider the triphone's original context when picking the best-fitting sequence. In current animations, this context covers the triphone itself, plus one phoneme on either side.

The second term, D_s, measures how closely the mouth contours match in overlapping segments of adjacent triphone videos. In synthesizing the mouth shapes for "teapot" we want the contours for the /IY/ and /P/ in the lip sequence used for /T-IY-P/ to match the contours for the /IY/ and /P/ in the sequence used for /IY-P-AA/. We measure this similarity by computing the Euclidean distance, frame by frame, between four-element feature vectors containing the overall lip width, overall lip height, inner lip height, and height of visible teeth.

The lip-shape distance (D_s) between two triphone videos is minimized with the correct time alignment. For example, consider the overlapping contours for the /P/ in /T-IY-P/ and /IY-P-AA/. The /P/ phoneme includes both a silence, when the lips are pressed together, and an audible release, when the lips move rapidly apart. The durations of the initial silence within the /P/ phoneme may be different. The phoneme labels do not provide us with this level of detailed timing. Yet, if the silence durations are different, the lip-shape distance for two otherwise-well-matched videos will be large. This problem is exacerbated by imprecision in the HMM phonemic labels.

We want to find the temporal overlap between neighboring triphones that maximizes the similarity between the two lip shapes. We shift the two triphones relative to each other to find the best temporal offset and duration. We then use this optimal overlap both in computing the lip-shape distance, D_s, and in cross-fading the triphone videos during the stitching step. The optimal overlap is the one that minimizes D_s while still maintaining a minimum-allowed overlap.

Since the fitness measure for each triphone segment depends on that segment's neighbors in both directions, we select the sequence of triphone segments using dynamic programming over the entire utterance. This procedure ensures the selection of the optimal segments.

4.2 Stitching It Together

Video Rewrite produces the final video by stitching together the appropriate entries from the video database. At this point, we have already selected a sequence of triphone videos that most closely matches the target audio. We need to align the overlapping lip images temporally. This internally time-aligned sequence of videos is then time aligned to the new speech utterance. Finally, the resulting sequences of lip images are spatially aligned and are stitched into the background face. We describe each step in turn.

4.2.1 Time Alignment of Triphone Videos

We have a sequence of triphone videos that we must combine to form a new mouth movie. In combining the videos, we want to maintain the dynamics of the phonemes and their transitions. We need to time align the triphone videos carefully before blending

them. If we are not careful in this step, the mouth will appear to flutter open and closed inappropriately.

We align the triphone videos by choosing a portion of the overlapping triphones where the two lips shapes are as similar as possible. We make this choice when we evaluate D_S to choose the sequence of triphone videos (Section 4.1). We use the overlap duration and shift that provide the minimum value of D_S for the given videos.

4.2.2 Time Alignment of the Lips to the Utterance

We now have a self-consistent temporal alignment for the triphone videos. We have the correct articulatory motions, in the correct order to match the target utterance, but these articulations are not yet time aligned with the target utterance.

We align the lip motions with the target utterance by comparing the corresponding phoneme transcripts. The starting time of the center phone in the triphone sequence is aligned with the corresponding label in the target transcript. The triphone videos are then stretched or compressed such that they fit the time needed between the phoneme boundaries in the target utterance.

4.2.3 Combining of the Lips and the Background

The remaining task is to stitch the triphone videos into the background sequence. The correctness of the facial alignment is critical to the success of the recombination. The lips and head are constantly moving in the triphone and background footage. Yet, we need to align them all so that the new mouth is firmly planted on the face. Any error in spatial alignment causes the mouth to jitter relative to the face—an extremely disturbing effect.

We again use the mask from Figure 4 to help us find the optimal global transform to register the faces from the triphone videos with the background face. The combined tranforms from the mouth and background images to the template face (Section 3.1) give our starting estimate in this search. Re-estimating the global transform by directly matching the triphone images to the background improves the accuracy of the mapping.

We use a replacement mask to specify which portions of the final video come from the triphone images and which come from the background video. This replacement mask warps to fit the new mouth shape in the triphone image and to fit the jaw shape in the background image. Figure 5 shows an example replacement mask, applied to triphone and background images.

Local deformations are required to stitch the shape of the mouth and jaw line correctly. These two shapes are handled differently. The mouth's shape is completely determined by the triphone images. The only changes made to these mouth shapes are imposed to align the mouths within the overlapping triphone images: The lip shapes are linearly cross-faded between the shapes in the overlapping segments of the triphone videos.

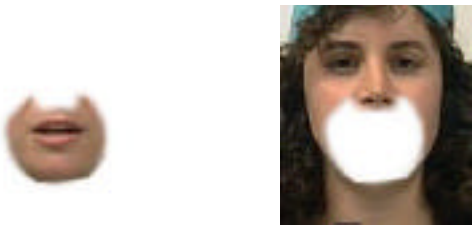


Figure 5: Facial fading mask. This mask determines which portions of the final movie frames come from the background frame, and which come from the triphone database. The mask should be large enough to include the mouth and chin. These images show the replacement mask applied to a triphone image, and its inverse applied to a background image. The mask warps according to the mouth and chin motions.

The jaw's shape, on the other hand, is a combination of the background jaw line and the two triphone jaw lines. Near the ears, we want to preserve the background video's jaw line. At the center of the jaw line (the chin), the shape and position are determined completely by what the mouth is doing. The final image of the jaw must join smoothly together the motion of the chin with the motion near the ears. To do this, we smoothly vary the weighting of the background and triphone shapes as we move along the jawline from the chin towards the ears.

The final stitching process is a three-way tradeoff in shape and texture among the fade-out lip image, the fade-in lip image, and the background image. As we move from phoneme to phoneme, the relative weights of the mouth shapes associated with the overlapping triphone-video images are changed. Within each frame, the relative weighting of the jaw shapes contributed by the background image and of the triphone-video images are varied spatially.

The derived fiduciary positions are used as control points in morphing. All morphs are done with the Beier-Neely algorithm [Beier92]. For each frame of the output image we need to warp four images: the two triphones, the replacement mask, and the background face. The warping is straightforward since we automatically generate high-quality control points using the eigenpoints algorithm.

5 RESULTS

We have applied Video Rewrite to several different training databases. We recorded one video dataset specifically for our evaluations. Section 5.1 describes our methods to collect this data and create lip-sync videos. Section 5.2 evaluates the resulting videos.

We also trained video models using truncated versions of our evaluation database. Finally, we used old footage of John F. Kennedy. We present the results from these experiments in Section 5.3.

5.1 Methods

We recorded about 8 minutes of video, containing 109 sentences, of a subject narrating a fairy tale. During the reading, the subject was asked to directly face the camera for some parts (still-head video) and to move and glance around naturally for others (moving-head video). We use these different segments to study the errors in local deformations separately from the errors in global spatial registration. The subject was also asked to wear a hat during the filming. We use this landmark to provide a quantitative evaluation of our global alignment. The hat is strictly outside all our alignment masks and our eigenpoints models. Thus, having the subject wear the hat does not effect the magnitude or type of errors that we expect to see in the animations—it simply provides us with a reference marker for the position and movement of her head.

To create a video model, we trained the system on all the still-head footage. Video Rewrite constructed and annotated the video model with just under 3500 triphone videos automatically, using HMM labeling of triphones and eigenpoint labeling of facial contours.

Video Rewrite was then given the target sentence, and was asked to construct the corresponding image sequence. To avoid unduly optimistic results, we removed from the database the triphone videos from training sentences similar to the target. A training sentence was considered similar to the target if the two shared a phrase two or more words long. Note that Video Rewrite would not normally pare the database in this manner: Instead, it would take advantage of these coincidences. We remove the similar sentences to avoid biasing our results.

We evaluated our output footage both qualitatively and quantitatively. Our qualitative evaluation was done informally, by a panel

of observers. There are no accepted metrics for evaluating lip-synced footage. Instead, we were forced to rely on the qualitative judgements listed in Section 5.2.

Only the (global) spatial registration is evaluated quantitatively. Since our subject wore a hat that moved rigidly with her upper head, we were able to measure quantitatively our global-registration error on this footage. We did so by first warping the full frame (instead of just the mouth region) of the triphone image into the coordinate frame of the background image. If this global transformation is correct, it should overlay the two images of the hat exactly on top of one another. We measured the error by finding the offset of the correlation peak for the image regions corresponding to the front of the hat. The offset of the peak is the registration error (in pixels).

5.2 Evaluation

Examples of our output footage can be seen at http://www.interval.com/papers/1997-012/. The top row of Figure 6 shows example frames, extracted from these videos. This section describes our evaluation criteria and the results.

5.2.1 Lip and Utterance Synchronization

How well are the lip motions synchronized with the audio? We evaluate this measure on the still-head videos. There occasionally are visible timing errors in plosives and stops.

5.2.2 Triphone-Video Synchronization

Do the lips flutter open and closed inappropriately? This artifact usually is due to synchronization error in overlapping triphone videos. We evaluated this measure on the still-head videos. We do not see any artifacts of this type.

5.2.3 Natural Articulation

Assuming that neither of the artifacts from Sections 5.2.1 or 5.2.2 appear, do the lip and teeth articulations look natural? Unnatural-looking articulation can result if the desired sequence of phonemes is not available in the database, and thus another sequence is used in its place. In our experiments, this replacement occurred on 31 percent of the triphone videos. We evaluated this measure on the still-head videos. We do not see this type of error when we use the full video model. Additional experiments in this area are described in Section 5.3.1.

5.2.4 Fading-Mask Visibility and Extent

Does the fading mask show? Does the animation have believable texture and motion around the lips and chin? Do the dimples move in sync with the mouth? We evaluated this measure on all the output videos. The still-head videos better show errors associated with the extent of the fading mask, whereas the moving-head videos better show errors due to interactions between the fading mask and the global transformation. Without illumination correction, we see artifacts in some of the moving-head videos, when the subject looked down so that the lighting on her face changed significantly. These artifacts disappear with adaptive illumination correction [Burt83].

5.2.5 Background Warping

Do the outer edges of the jaw line and neck, and the upper portions of the cheeks look realistic? Artifacts in these areas are due to incorrect warping of the background image or to a mismatch between the texture and the warped shape of the background image. We evaluated this measure on all the output videos. In some segments, we found minor artifacts near the outer edges of the jaw.

5.2.6 Spatial Registration

Does the mouth seem to float around on the face? Are the teeth rigidly attached to the skull? We evaluated this measure on the moving-head videos. No registration errors are visible.

We evaluated this error quantitatively as well, using the hat-registration metric described in Section 5.1. The mean, median, and maximum errors in the still-head videos were 0.6, 0.5, and 1.2 pixels (standard deviation 0.3); those in the moving-head videos were 1.0, 1.0, and 2.0 pixels (standard deviation 0.4). For comparison, the face covers approximately 85×120 pixels.

5.2.7 Overall Quality

Is the lip-sync believable? We evaluated this measure on all the output videos. We judged the overall quality as excellent.

Figure 6: Examples of synthesized output frames. These frames show the quality of our output after triphone segments have been stitched into different background video frames.

5.3 Other Experiments

In this section, we examine our performance using steadily smaller training databases (Section 5.3.1) and using historic footage (Section 5.3.2).

5.3.1 Reduction of Video Model Size

We wanted to see how the quality fell off as the number of data available in the video model were reduced. With the 8 minutes of video, we have examples of approximately 1700 different triphones (of around 19,000 naturally occurring triphones); our animations used triphones other than the target triphones 31 percent of the time. What happens when we have only 1 or 2 minutes of data? We truncated our video database to one-half, one-quarter, and one-eighth of its original size, and then reanimated our target sentences. The percent of mismatched triphones increased by about 15 percent with each halving of the database (that is, 46, 58, and 74 percent of the triphones were replaced in the reduced datasets). The perceptual quality also degraded smoothly as the database size was reduced. The video from the reduced datasets are shown on our web site.

5.3.2 Reanimation of Historic Footage

We also applied Video Rewrite to public-domain footage of John F. Kennedy. For this application, we digitized 2 minutes (1157 triphones) of Kennedy speaking during the Cuban missile crisis. Forty-five seconds of this footage are from a close-up camera, about 30 degrees to Kennedy's left. The remaining images are medium shots from the same side. The size ratio is approximately 5:3 between the close-up and medium shots. During the footage, Kennedy moves his head about 30 degrees vertically, reading his speech from notes on the desk and making eye contact with a center camera (which we do not have).

We used this video model to synthesize new animations of Kennedy saying, for example, “Read my lips” and “I never met Forrest Gump.” These animations combine the footage from both camera shots and from all head poses. The resulting videos are shown on our web site. The bottom row of Figure 6 shows example frames, extracted from these videos.

In our preliminary experiments, we were able to find the correct triphone sequences just 6% of the time. The lips are reliably synchronized to the utterance. The fading mask is not visible, nor is the background warping. However, the overall animation quality is not as good as our earlier results. The animations include some lip fluttering, because of the mismatched triphone sequences.

Our quality is limited for two reasons. The available viseme footage is distributed over a wide range of vertical head rotations. If we choose triphones that match the desired pose, then we cannot find good matches for the desired phoneme sequence. If we choose triphones that are well matched to the desired phoneme sequence, then we need to dramatically change the pose of the lip images. A large change in pose is difficult to model with our global (affine) transform. The lip shapes are distorted because we assumed, implicitly in the global transform, that the lips lie on a flat plane. Both the limited-triphone and pose problems can be avoided with additional data.

6 FUTURE WORK

There are many ways in which Video Rewrite could be extended and improved. The phonemic labeling of the triphone and background footage could consider the mouth- and jaw-shape information, as well as acoustic data [Bregler95]. Additional lip-image data and multiple eigenpoints models could be added, allowing larger out-of-plane head rotations. The acoustic data could be used in selecting the triphone videos, because facial expressions affect voice qualities (you can hear a smile). The synthesis could be made real-time, with low-latency.

In Sections 6.1 through 6.3, we explore extensions that we think are most promising and interesting.

6.1 Alignment Between Lips and Target

We currently use the simplest approach to time aligning the lip sequences with the target utterance: We rely on the phoneme boundaries. This approach provides a rough alignment between the motions in the lip sequence and the sounds in the target utterance. As we mentioned in Section 4.1, however, the phoneme boundaries are both imprecise (the HMM alignment is not perfect) and coarse (significant visual and auditory landmarks occur within single phonemes).

A more accurate way to time align the lip motions with the target utterance uses dynamic time warping of the audio associated with each triphone video to the corresponding segment of the target utterance. This technique would allow us to time align the auditory landmarks from the triphone videos with those of the target utterance, even if the landmarks occur at subphoneme resolution. This time alignment, when applied to the triphone image sequence, would then align the visual landmarks of the lip sequence with the auditory landmarks of the target utterance.

The overlapping triphone videos would provide overlapping and conflicting time warpings. Yet we want to keep fixed the time alignment of the overlapping triphone videos, as dictated by the visual distances (Section 4.1 and 4.2). Research is needed in how best to trade off these potentially conflicting time-alignment maps.

6.2 Animation of Facial Features

Another promising extension is animation of other facial parts, based on simple acoustic features or other criteria. The simplest version of this extension would change the position of the eyebrows with pitch [Ohala94]. A second extension would index the video model by both triphone and expression labels. Using such labels, we would select smiling or frowning lips, as desired. Alternatively, we could impose the desired expression on a neutral mouth shape, for those times when the appropriate combinations of triphones and expression are not available. To do this imposition correctly, we must separate which deformations are associated with articulations, and which are associated with expressions, and how the two interact. This type of factorization must be learned from examples [Tenenbaum97].

6.3 Perception of Lip Shapes

In doing this work, we solved many problems—automatic labeling, matching, and stitching—yet we found many situations where we did not have sufficient knowledge of how people perceive speaking faces. We would like to know more about how important the correct lip shapes and motions are in lip synching. For example, one study [Owens85] describes the confusibility of consonants in vowel–consonant–vowel clusters. The clustering of consonants into viseme class depends on the surrounding vowel context. Clearly, we need more sophisticated distance metrics within and between viseme classes.

7 CONTRIBUTIONS

Video Rewrite is a facial animation system that is driven by audio input. The output sequence is created from real video footage. It combines background video footage, including natural facial movements (such as eye blinks and head motions) with natural footage of mouth and chin motions. Video Rewrite is the first facial-animation system to automate all the audio- and video-labeling tasks required for this type of reanimation.

Video Rewrite can use images from unconstrained footage both to create the video model of the mouth and chin motions and to provide a background sequence for the final output footage. It preserves the individual characteristics of the subject in the original footage, even while the subject appears to mouth a completely new utterance. For example, the temporal dynamics of John F. Kennedy's articulatory motions can be preserved, reorganized, and reimposed on Kennedy's face.

Since Video Rewrite retains most of the background frame, modifying only the mouth area, it is well suited to applications such as movie dubbing. The setting and action are provided by the background video. Video Rewrite maintains an actor's visual mannerisms, using the dynamics of the actor's lips and chin from the video model for articulatory mannerisms, and using the background video for all other mannerisms. It maintains the correct timing, using the action as paced by the background video and speech as paced by the new soundtrack. It undertakes the entire process without manual intervention. The actor convincingly mouths something completely new.

ACKNOWLEDGMENTS

Many colleagues helped us. Ellen Tauber and Marc Davis graciously submitted to our experimental manipulation. Trevor Darrell and Subutai Ahmad contributed many good ideas to the algorithm development. Trevor, Subutai, John Lewis, Bud Lassiter, Gaile Gordon, Kris Rahardja, Michael Bajura, Frank Crow, Bill Verplank, and John Woodfill helped us to evaluate our results and the description. Bud Lassiter and Chris Seguine helped us with the video production. We offer many thanks to all.

REFERENCES

[Beier92] T. Beier, S. Neely. Feature-based image metamorphosis. *Computer Graphics*, 26(2):35–42, 1992. ISSN 0097-8930.

[Black95] M.J. Black, Y. Yacoob. Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motion. *Proc. IEEE Int. Conf. Computer Vision*, Cambridge, MA, pp. 374–381, 1995. ISBN 0-8186-7042-8.

[Bregler95] C. Bregler, S. Omohundro. Nonlinear manifold learning for visual speech recognition. *Proc. IEEE Int. Conf. Computer Vision,* Cambridge, MA, pp. 494–499, 1995. ISBN 0-8186-7042-8.

[Burt83] P.J. Burt, E.H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graphics,* 2(4): 217–236, 1983. ISSN 0730-0301.

[Cohen93] M.M. Cohen, D.W. Massaro. Modeling coarticulation in synthetic visual speech. In *Models and Techniques in Computer Animation*, ed. N.M Thalman, D. Thalman, pp. 139–156, Tokyo: Springer-Verlag, 1993. ISBN 0-3877-0124-9.

[Covell96] M. Covell, C. Bregler. Eigenpoints. *Proc. Int. Conf. Image Processing*, Lausanne, Switzerland, Vol. 3, pp. 471–474, 1996. ISBN 0-7803-3258-x.

[Guiard-Marigny94] T. Guiard-Marigny, A. Adjoudani, C. Benoit. A 3-D model of the lips for visual speech synthesis. *Proc. ESCA/IEEE Workshop on Speech Synthesis*, New Paltz, NY, pp. 49–52, 1994.

[Kass87] M. Kass, A. Witkin, D. Terzopoulos. Snakes: Active contour models. *Int. J. Computer Vision*, 1(4):321–331, 1987. ISSN 0920-5691.

[Kirby90] M. Kirby, L. Sirovich. Application of the Karhunen-Loeve procedure for the characterization of human faces. *IEEE PAMI*, 12(1):103–108, Jan. 1990. ISSN 0162-8828.

[Lamel86] L. F. Lamel, R. H. Kessel, S. Seneff. Speech database development: Design and analysis of the acoustic-phonetic corpus. *Proc. Speech Recognition Workshop (DARPA)*, Report #SAIC-86/1546, pp. 100–109, McLean VA: Science Applications International Corp., 1986.

[Lanitis95] A. Lanitis, C.J. Taylor, T.F. Cootes. A unified approach for coding and interpreting face images. *Proc. Int. Conf. Computer Vision,* Cambridge, MA, pp. 368–373, 1995. ISBN 0-8186-7042-8.

[Lewis91] J.Lewis. Automated lip-sync: Background and techniques. *J.Visualization and Computer Animation,* 2(4):118–122, 1991. ISSN 1049-8907.

[Litwinowicz94] P. Litwinowicz, L. Williams. Animating images with drawings. *SIGGRAPH 94*, Orlando, FL, pp. 409–412, 1994. ISBN 0-89791-667-0.

[Morishima91] S. Morishima, H. Harashima. A media conversion from speech to facial image for intelligent man-machine interface. *IEEE J Selected Areas Communications*, 9 (4):594–600, 1991. ISSN 0733-8716.

[Moulines90] E. Moulines, P. Emerard, D. Larreur, J. L. Le Saint Milon, L. Le Faucheur, F. Marty, F. Charpentier, C. Sorin. A real-time French text-to-speech system generating high-quality synthetic speech. *Proc. Int. Conf. Acoustics, Speech, and Signal Processing,* Albuquerque, NM, pp. 309–312, 1990.

[Ohala94] J.J. Ohala. The frequency code underlies the sound symbolic use of voice pitch. In *Sound Symbolism*, ed. L. Hinton, J. Nichols, J. J. Ohala, pp. 325–347, Cambridge UK: Cambridge Univ. Press, 1994. ISBN 0-5214-5219-8.

[Owens85] E. Owens, B. Blazek. Visemes observed by hearing-impaired and normal-hearing adult viewers. *J. Speech and Hearing Research*, 28:381–393, 1985. ISSN 0022-4685.

[Parke72] F. Parke. Computer generated animation of faces. *Proc. ACM National Conf.*, pp. 451–457, 1972.

[Rabiner89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Readings in Speech Recognition*, ed. A. Waibel, K. F. Lee, pp. 267–296, San Mateo, CA: Morgan Kaufmann Publishers, 1989. ISBN 1-5586-0124-4.

[Scott94] K.C. Scott, D.S. Kagels, S.H. Watson, H. Rom, J.R. Wright, M. Lee, K.J. Hussey. Synthesis of speaker facial movement to match selected speech sequences. *Proc. Australian Conf. Speech Science and Technology,* Perth Australia, pp. 620–625, 1994. ISBN 0-8642-2372-2.

[Tenenbaum97] J. Tenenbaum, W. Freeman. Separable mixture models: Separating style and content. In *Advances in Neural Information Processing 9*, ed. M. Jordan, M. Mozer, T. Petsche, Cambridge, MA: MIT Press, (in press).

[Turk91] M. Turk, A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, 1991. ISSN 0898-929X

[Viterbi67] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Informat. Theory*, IT-13:260–269, 1967. ISSN 0018-9448.

[Waters95] K. Waters, T. Levergood. DECface: A System for Synthetic Face Applications. *J. Multimedia Tools and Applications*, 1 (4):349–366, 1995. ISSN 1380-7501.

[Williams90] L. Williams. Performance-Driven Facial Animation. *Computer Graphics* (Proceedings of SIGGRAPH 90), 24(4):235–242, 1990. ISSN 0097-8930.

[Yuille89] A.L. Yuille, D.S. Cohen, P.W. Hallinan. Feature extraction from faces using deformable templates. *Proc. IEEE Computer Vision and Pattern Recognition,* San Diego, CA, pp. 104–109, 1989. ISBN 0-8186-1952-x.

Making Faces

Brian Guenter† Cindy Grimm† Daniel Wood‡
Henrique Malvar† Fredrick Pighin‡
†Microsoft Corporation ‡University of Washington

ABSTRACT

We have created a system for capturing both the three-dimensional geometry and color and shading information for human facial expressions. We use this data to reconstruct photorealistic, 3D animations of the captured expressions. The system uses a large set of sampling points on the face to accurately track the three dimensional deformations of the face. Simultaneously with the tracking of the geometric data, we capture multiple high resolution, registered video images of the face. These images are used to create a texture map sequence for a three dimensional polygonal face model which can then be rendered on standard 3D graphics hardware. The resulting facial animation is surprisingly life-like and looks very much like the original live performance. Separating the capture of the geometry from the texture images eliminates much of the variance in the image data due to motion, which increases compression ratios. Although the primary emphasis of our work is not compression we have investigated the use of a novel method to compress the geometric data based on principal components analysis. The texture sequence is compressed using an MPEG4 video codec. Animations reconstructed from 512x512 pixel textures look good at data rates as low as 240 Kbits per second.

CR Categories: I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism: Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1 Introduction

One of the most elusive goals in computer animation has been the realistic animation of the human face. Possessed of many degrees of freedom and capable of deforming in many ways the face has been difficult to simulate accurately enough to convince the average person that a piece of computer animation is actually an image of a real person.

We have created a system for capturing human facial expression and replaying it as a highly realistic 3D "talking head" consisting of a deformable 3D polygonal face model with a changing texture map. The process begins with video of a live actor's face, recorded from multiple camera positions simultaneously. Fluorescent colored 1/8" circular paper fiducials are glued on the actor's face and their 3D position reconstructed over time as the actor talks and emotes. The 3D fiducial positions are used to distort a 3D polygonal face model in mimicry of the distortions of the real face. The fiducials are removed using image processing techniques and the video streams from the multiple cameras are merged into a single texture map. When the resulting fiducial-free texture map is applied to the 3D reconstructed face mesh the result is a remarkably life-like 3D animation of facial expression. Both the time varying texture created from the video streams and the accurate reproduction of the 3D face structure contribute to the believability of the resulting animation.

Our system differs from much previous work in facial animation, such as that of Lee [10], Waters [14], and Cassel [3], in that we are not synthesizing animations using a physical or procedural model of the face. Instead, we capture facial movements in three dimensions and then replay them. The systems of [10], [14] are designed to make it relatively easy to animate facial expression manually. The system of [3] is designed to automatically create a dialog rather than faithfully reconstruct a particular person's facial expression. The work of Williams [15] is most similar to ours except that he used a single static texture image of a real person's face and tracked points only in 2D. The work of Bregler et al [2] is somewhat less related. They use speech recognition to locate visemes[1] in a video of a person talking and then synthesize new video, based on the original video sequence, for the mouth and jaw region of the face to correspond with synthetic utterances. They do not create a three dimensional face model nor do they vary the expression on the remainder of the face. Since we are only concerned with capturing and reconstructing facial performances out work is unlike that of [5] which attempts to recognize expressions or that of [4] which can track only a limited set of facial expressions.

An obvious application of this new method is the creation of believable virtual characters for movies and television. Another application is the construction of a flexible type of video compression. Facial expression can be captured in a studio, delivered via CDROM or the internet to a user, and then reconstructed in real time on a user's computer in a virtual 3D environment. The user can select any arbitrary position for the face, any virtual camera viewpoint, and render the result at any size.

One might think the second application would be difficult to achieve because of the huge amount of video data required for the time varying texture map. However, since our system generates accurate 3D deformation information, the texture image data is precisely registered from frame to frame. This reduces most of the variation in image intensity due to geometric motion, leaving primarily shading and self shadowing effects. These effects tend to be of low spatial frequency and can be compressed very efficiently. The compressed animation looks good at data rates of 240 kbits per second for texture image sizes of 512x512 pixels, updating at 30 frames per second.

The main contributions of the paper are a method for robustly capturing both a 3D deformation model and a registered texture image sequence from video data. The resulting geometric and texture data can be compressed, with little loss of fidelity, so that storage

[1] Visemes are the visual analog of phonemes.

Figure 1: The six camera views of our actress' face.

requirements are reasonable for many applications.

Section 2 of the paper explains the data capture stage of the process. Section 3 describes the fiducial correspondence algorithm. In Section 4 we discuss capturing and moving the mesh. Sections 5 and 6 describe the process for making the texture maps. Section 7 of the paper describes the algorithm for compressing the geometric data.

2 Data Capture

We used six studio quality video cameras arranged in the pattern shown in Plate 1 to capture the video data. The cameras were synchronized and the data saved digitally. Each of the six cameras was individually calibrated to determine its intrinsic and extrinsic parameters and to correct for lens distortion. The details of the calibration process are not germane to this paper but the interested reader can find a good overview of the topic in [6] as well as an extensive bibliography.

We glued 182 dots of six different colors onto the actress' face. The dots were arranged so that dots of the same color were as far apart as possible from each other and followed the contours of the face. This made the task of determining frame to frame dot correspondence (described in Section 3.3) much easier. The dot pattern was chosen to follow the contours of the face (i.e., outlining the eyes, lips, and nasio-labial furrows), although the manual application of the dots made it difficult to follow the pattern exactly.

The actress' head was kept relatively immobile using a padded foam box; this reduced rigid body motions and ensured that the actress' face stayed centered in the video images. Note that rigid body motions can be captured later using a 3D motion tracker, if desired.

The actress was illuminated with a combination of visible and near UV light. Because the dots were painted with fluorescent pigments the UV illumination increased the brightness of the dots significantly and moved them further away in color space from the colors of the face than they would ordinarily be. This made them easier to track reliably. Before the video shoot the actress' face was digitized using a cyberware scanner. This scan was used to create the base 3D face mesh which was then distorted using the positions of the tracked dots.

3 Dot Labeling

The fiducials are used to generate a set of 3D points which act as control points to warp the cyberware scan mesh of the actress' head. They are also used to establish a stable mapping for the textures generated from each of the six camera views. This requires that each dot have a unique and consistent label over time so that it is associated with a consistent set of mesh vertices.

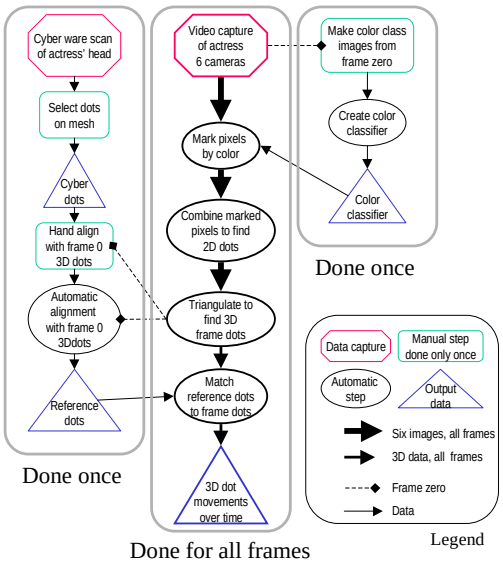


Figure 2: The sequence of operations needed to produce the labeled 3D dot movements over time.

The dot labeling begins by first locating (for each camera view) connected components of pixels which correspond to the fiducials. The 2D location for each dot is computed by finding the two dimensional centroid of each connected component. Correspondence between 2D dots in different camera views is established and potential 3D locations of dots reconstructed by triangulation. We construct a reference set of dots and pair up this reference set with the 3D locations in each frame. This gives a unique labeling for the dots that is maintained throughout the video sequence.

A flowchart of the dot labeling process is shown in Figure 2. The left side of the flowchart is described in Section 3.3.1, the middle in Sections 3.1, 3.2, and 3.3.2, and the right side in Section 3.1.1.

3.1 Two-dimensional dot location

For each camera view the 2D coordinates of the centroid of each colored fiducial must be computed. There are three steps to this process: color classification, connected color component generation, and centroid computation.

First, each pixel is classified as belonging to one of the six dot colors or to the background. Then depth first search is used to locate connected blobs of similarly colored pixels. Each connected colored blob is grown by one pixel to create a mask used to mark those pixels to be included in the centroid computation. This process is illustrated in Figure 4.

The classifier requires the manual marking of the fiducials for one frame for each of the six cameras. From this data a robust color classifier is created (exact details are discussed in Section 3.1.1). Although the training set was created using a single frame of a 3330 frame sequence, the fiducial colors are reliably labeled throughout the sequence. False positives are quite rare, with one major exception, and are almost always isolated pixels or two pixel clusters. The majority of exceptions arise because the highlights on the teeth and mouth match the color of the white fiducial training set. Fortunately, the incorrect white fiducial labelings occur at consistent 3D locations and are easily eliminated in the 3D dot processing stage.

The classifier generalizes well so that even fairly dramatic changes

in fiducial color over time do not result in incorrect classification. For example, Figure 5(b) shows the same green fiducial in two different frames. This fiducial is correctly classified as green in both frames.

The next step, finding connected color components, is complicated by the fact that the video is interlaced. There is significant field to field movement, especially around the lips and jaw, sometimes great enough so that there is no spatial overlap at all between the pixels of a fiducial in one field and the pixels of the same fiducial in the next field. If the two fields are treated as a single frame then a single fiducial can be fragmented, sometimes into many pieces.

One could just find connected color components in each field and use these to compute the 2D dot locations. Unfortunately, this does not work well because the fiducials often deform and are sometimes partially occluded. Therefore, the threshold for the number of pixels needed to classify a group of pixels as a fiducial has to be set very low. In our implementation any connected component which has more than three pixels is classified as a fiducial rather than noise. If just the connected pixels in a single field are counted then the threshold would have to be reduced to one pixel. This would cause many false fiducial classifications because there are typically a few 1 pixel false color classifications per frame and 2 or 3 pixel false clusters occur occasionally. Instead, we find connected components and generate lists of potential 2D dots in each field. Each potential 2D dot in field one is then paired with the closest 2D potential dot in field two. Because fiducials of the same color are spaced far apart, and because the field to field movement is not very large, the closest potential 2D dot is virtually guaranteed to be the correct match. If the sum of the pixels in the two potential 2D dots is greater than three pixels then the connected components of the two 2D potential dots are merged, and the resulting connected component is marked as a 2D dot.

The next step is to find the centroid of the connected components marked as 2D dots in the previous step. A two dimensional gradient magnitude image is computed by passing a one dimensional first derivative of Gaussian along the x and y directions and then taking the magnitude of these two values at each pixel. The centroid of the colored blob is computed by taking a weighted sum of positions of the pixel (x, y) coordinates which lie inside the gradient mask, where the weights are equal to the gradient magnitude.

3.1.1 Training the color classifier

We create one color classifier for each of the camera views, since the lighting can vary greatly between cameras. In the following discussion we build the classifier for a single camera.

The data for the color classifier is created by manually marking the pixels of frame zero that belong to a particular fiducial color. This is repeated for each of the six colors. The marked data is stored as 6 *color class images*, each of which is created from the original camera image by setting all of the pixels *not* marked as the given color to black (we use black as an out-of-class label because pure black never occurred in any of our images). A typical color class image for the yellow dots is shown in Figure 3. We generated the color class images using the "magic wand" tool available in many image editing programs.

A seventh color class image is automatically created for the background color (e.g., skin and hair) by labeling as out-of-class any pixel in the image which was previously marked as a fiducial in any of the fiducial color class images. This produces an image of the face with black holes where the fiducials were.

The color classifier is a discrete approximation to a nearest neighbor classifier [12]. In a nearest neighbor classifier the item

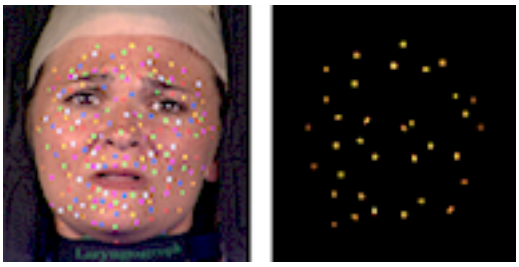


Figure 3: An image of the actress's face. A typical training set for the yellow dots, selected from the image on the left.

to be classified is given the label of the closest item in the training set, which in our case is the color data contained in the color class images. Because we have 3 dimensional data we can approximate the nearest neighbor classifier by subdividing the RGB cube uniformly into voxels, and assigning class labels to each RGB voxel. To classify a new color you quantize its RGB values and then index into the cube to extract the label.

To create the color classifier we use the color class images to assign color classes to each voxel. Assume that the color class image for color class C_i has n distinct colors, $c_1 ... c_n$. Each of the voxels corresponding to the color c_j is labeled with the color class C_i. Once the voxels for all of the known colors are labeled, the remaining unlabeled voxels are assigned labels by searching through all of the colors in each color class C_i and finding the color closest to p in RGB space. The color p is given the label of the color class containing the nearest color. Nearness in our case is the Euclidean distance between the two points in RGB space.

If colors from different color classes map to the same sub-cube, we label that sub-cube with the background label since it is more important to avoid incorrect dot labeling than it is to try to label every dot pixel. For the results shown in this paper we quantized the RGB color cube into a 32x32x32 lattice.

3.2 Camera to camera dot correspondence and 3D reconstruction

In order to capture good images of both the front and the sides of the face the cameras were spaced far apart. Because there are such extreme changes in perspective between the different camera views, the projected images of the colored fiducials are very different. Figure 5 shows some examples of the changes in fiducial shape and color between camera views. Establishing fiducial correspondence between camera views by using image matching techniques such as optical flow or template matching would be difficult and likely to generate incorrect matches. In addition, most of the camera views will only see a fraction of the fiducials so the correspondence has to be robust enough to cope with occlusion of fiducials in some of the camera views. With the large number of fiducials we have placed on the face false matches are also quite likely and these must be detected and removed. We used ray tracing in combination with a RANSAC [7] like algorithm to establish fiducial correspondence and to compute accurate 3D dot positions. This algorithm is robust to occlusion and to false matches as well.

First, all potential point correspondences between cameras are generated. If there are k cameras, and n 2D dots in each camera view then $\left(\begin{array}{c} k \\ 2 \end{array} \right) n^2$ point correspondences will be tested. Each correspondence gives rise to a 3D candidate point defined as the closest point of intersection of rays cast from the 2D dots in the

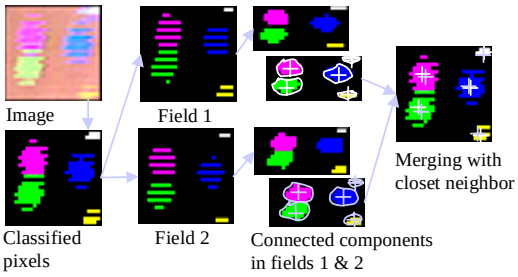


Figure 4: Finding the 2D dots in the images.

two camera views. The 3D candidate point is projected into each of the two camera views used to generate it. If the projection is further than a user-defined epsilon, in our case two pixels, from the centroid of either 2D point then the point is discarded as a potential 3D point candidate. All the 3D candidate points which remain are added to the 3D point list.

Each of the points in the 3D point list is projected into a reference camera view which is the camera with the best view of all the fiducials on the face. If the projected point lies within two pixels of the centroid of a 2D dot visible in the reference camera view then it is added to the list of potential 3D candidate positions for that 2D dot. This is the list of potential 3D matches for a given 2D dot.

For each 3D point in the potential 3D match list, $\binom{n}{3}$ possible combinations of three points in the 3D point list are computed and the combination with the smallest variance is chosen as the true 3D position. Then all 3D points which lie within a user defined distance, in our case the sphere subtended by a cone two pixels in radius at the distance of the 3D point, are averaged to generate the final 3D dot position. This 3D dot position is assigned to the corresponding 2D dot in the reference camera view.

This algorithm could clearly be made more efficient because many more 3D candidate points are generated then necessary. One could search for potential camera to camera correspondences only along the epipolar lines and use a variety of space subdivision techniques to find 3D candidate points to test for a given 2D point. However, because the number of fiducials in each color set is small (never more than 40) both steps of this simple and robust algorithm are reasonably fast, taking less than a second to generate the 2D dot correspondences and 3D dot positions for six camera views. The 2D dot correspondence calculation is dominated by the time taken to read in the images of the six camera views and to locate the 2D dots in each view. Consequently, the extra complexity of more efficient stereo matching algorithms does not appear to be justified.

3.3 Frame to frame dot correspondence and labeling

We now have a set of unlabeled 3D dot locations for each frame. We need to assign, across the entire sequence, consistent labels to the 3D dot locations. We do this by defining a reference set of dots D and matching this set to the 3D dot locations given for each frame. We can then describe how the reference dots move over time as follows: Let $d_j \in D$ be the neutral location for the reference dot j. We define the position of d_j at frame i by an offset, i.e.,

$$d_j^i = d_j + \vec{v}_j^i \tag{1}$$

Because there are thousands of frames and 182 dots in our data

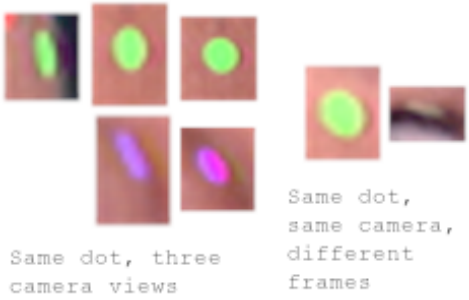


Figure 5: Dot variation. Left: Two dots seen from three different cameras (the purple dot is occluded in one camera's view). Right: A single dot seen from a single camera but in two different frames.

set we would like the correspondence computation to be automatic and quite efficient. To simplify the matching we used a fiducial pattern that separates fiducials of a given color as much as possible so that only a small subset of the unlabeled 3D dots need be checked for a best match. Unfortunately, simple nearest neighbor matching fails for several reasons: some fiducials occasionally disappear, some 3D dots may move more than the average distance between 3D dots of the same color, and occasionally extraneous 3D dots appear, caused by highlights in the eyes or teeth. Fortunately, neighboring fiducials move similarly and we can exploit this fact, modifying the nearest neighbor matching algorithm so that it is still efficient but also robust.

For each frame i we first move the reference dots to the locations found in the previous frame. Next, we find a (possibly incomplete) match between the reference dots and the 3D dot locations for frame i. We then move each matched reference dot to the location of its corresponding 3D dot. If a reference dot does not have a match we "guess" a new location for it by moving it in the same direction as its neighbors. We then perform a final matching step.

3.3.1 Acquiring the reference set of dots

The cyberware scan was taken with the dots glued onto the face. Since the dots are visible in both the geometric and color information of the scan, we can place the reference dots on the cyberware model by manually clicking on the model. We next need to align the reference dots and the model with the 3D dot locations found in frame zero. The coordinate system for the cyberware scan differs from the one used for the 3D dot locations, but only by a rigid body motion plus a uniform scale. We find this transform as follows: we first hand-align the 3D dots from frame zero with the reference dots acquired from the scan, then call the matching routine described in Section 3.3.2 below to find the correspondence between the 3D dot locations, f_i, and the reference dots, d_i. We use the method described in [9] to find the exact transform, T, between the two sets of dots. Finally, we replace the temporary locations of the reference dots with $d_i = f_i$.
and use T^{-1} to transform the cyberware model into the coordinate system of the video 3D dot locations.

3.3.2 The matching routine

The matching routine is run twice per frame. We first perform a conservative match, move the reference dots (as described below in Section 3.3.3), then perform a second, less conservative, match. By moving the reference dots between matches we reduce the problem of large 3D dot position displacements.

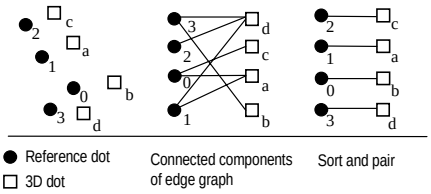


Figure 6: Matching dots.

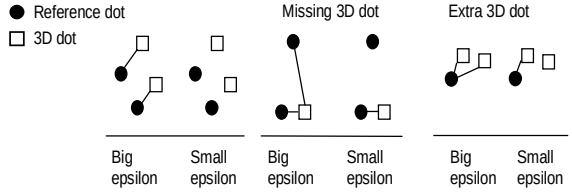


Figure 7: Examples of extra and missing dots and the effect of different values for ϵ.

The matching routine can be thought of as a graph problem where an edge between a reference dot and a frame dot indicates that the dots are potentially paired (see Figure 6). The matching routine proceeds in several steps; first, for each reference dot we add an edge for every 3D dot of the same color that is within a given distance ϵ. We then search for connected components in the graph that have an equal number of 3D and reference dots (most connected components will have exactly two dots, one of each type). We sort the dots in the vertical dimension of the plane of the face and use the resulting ordering to pair up the reference dots with the 3D dot locations (see Figure 6).

In the video sequences we captured, the difference in the 3D dot positions from frame to frame varied from zero to about 1.5 times the average distance separating closest dots. To adjust for this, we run the matching routine with several values of ϵ and pick the run that generates the most matches. Different choices of ϵ produce different results (see Figure 7): if ϵ is too small we may not find matches for 3D dots that have moved a lot. If ϵ is too large then the connected components in the graph will expand to include too many 3D dots. We try approximately five distances ranging from 0.5 to 1.5 of the average distance between closest reference dots.

If we are doing the second match for the frame we add an additional step to locate matches where a dot may be missing (or extra). We take those dots which have not been matched and run the matching routine on them with smaller and smaller ϵ values. This resolves situations such as the one shown on the right of Figure 7.

3.3.3 Moving the dots

We move all of the matched reference dots to their new locations then interpolate the locations for the remaining, unmatched reference dots by using their nearest, matched neighbors. For each reference dot we define a valid set of neighbors using the routine in Section 4.2.1, ignoring the blending values returned by the routine.

To move an unmatched dot d_k we use a combination of the offsets of all of its valid neighbors (refer to Equation 1). Let $n_k \subset D$ be the set of neighbor dots for dot d_k. Let $\hat{n}_k$ be the set of neighbors that have a match for the current frame i. Provided $\hat{n}_k \neq \emptyset$, the offset vector for dot d_k^i is calculated as follows: let $\vec{v}_j^i = d_j^i - d_j$ be the offset of dot j (recall that d_j is the initial position for the reference dot j).

$$\vec{v}_k^i = \frac{1}{||\hat{n}_k||} \sum_{d_j^i \in \hat{n}_k} \vec{v}_j^i$$

If the dot has no matched neighbors we repeat as necessary, treating the moved, unmatched reference dots as matched dots. Eventually, the movements will propagate through all of the reference dots.

4 Mesh construction and deformation

4.1 Constructing the mesh

To construct a mesh we begin with a cyberware scan of the head. Because we later need to align the scan with the 3D video dot data, we scanned the head with the fiducials glued on. The resulting scan suffers from four problems:

- The fluorescent fiducials caused "bumps" on the mesh.
- Several parts of the mesh were not adequately scanned, namely, the ears, one side of the nose, the eyes, and under the chin. These were manually corrected.
- The mesh does not have an opening for the mouth.
- The scan has too many polygons.

The bumps caused by the fluorescent fiducials were removed by selecting the vertices which were out of place (approximately 10-30 surrounding each dot) and automatically finding new locations for them by blending between four correct neighbors. Since the scan produces a rectangular grid of vertices we can pick the neighbors to blend between in (u, v) space, i.e., the nearest valid neighbors in the positive and negative u and v direction.

The polygons at the mouth were split and then filled with six rows of polygons located slightly behind the lips. We map the teeth and tongue onto these polygons when the mouth is open.

We reduced the number of polygons in the mesh from approximately $460,000$ to 4800 using Hoppe's simplification method [8].

4.2 Moving the mesh

The vertices are moved by a linear combination of the offsets of the nearest dots (refer to Equation 1). The linear combination for each vertex v_j is expressed as a set of blend coefficients, α_k^j, one for each dot, such that $\sum_{d_k \in D} \alpha_k^j = 1$ (most of the α_k^js will be zero). The new location p_j^i of the vertex v_j at frame i is then

$$p_j^i = p_j + \sum_k \alpha_k^j ||d_k^i - d_k||$$

where p_j is the initial location of the vertex v_j.

For most of the vertices the α_k^js are a weighted average of the closest dots. The vertices in the eyes, mouth, behind the mouth, and outside of the facial area are treated slightly differently since, for example, we do not want the dots on the lower lip influencing vertices on the upper part of the lip. Also, although we tried to keep the head as still as possible, there is still some residual rigid body motion. We need to compensate for this for those vertices that are not directly influenced by a dot (e.g., the back of the head).

We use a two-step process to assign the blend coefficients to the vertices. We first find blend coefficients for a grid of points evenly distributed across the face, then use this grid of points to

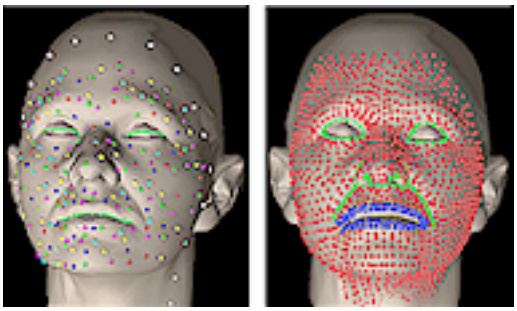


Figure 8: Left: The original dots plus the extra dots (in white). The labeling curves are shown in light green. Right: The grid of dots. Outline dots are green or blue.

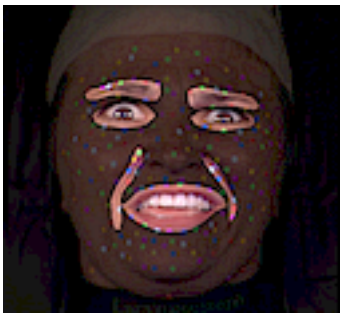


Figure 9: Masks surrounding important facial features. The gradient of a blurred version of this mask is used to orient the low-pass filters used in the dot removal process.

assign blend coefficients to the vertices. This two-step process is helpful because both the fluorescent fiducials and the mesh vertices are unevenly distributed across the face, making it difficult to get smoothly changing blend coefficients.

The grid consists of roughly 1400 points, evenly distributed and placed by hand to follow the contours of the face (see Figure 8). The points along the nasolabial furrows, nostrils, eyes, and lips are treated slightly differently than the other points to avoid blending across features such as the lips.

Because we want the mesh movement to go to zero outside of the face, we add another set of unmoving dots to the reference set. These new dots form a ring around the face (see Figure 8) enclosing all of the reference dots. For each frame we determine the rigid body motion of the head (if any) using a subset of those reference dots which are relatively stable. This rigid body transformation is then applied to the new dots.

We label the dots, grid points, and vertices as being *above*, *below*, or *neither* with respect to each of the eyes and the mouth. Dots which are *above* a given feature can not be combined with dots which are *below* that same feature (or vice-versa). Labeling is accomplished using three curves, one for each of the eyes and one for the mouth. Dots directly above (or below) a curve are labeled as *above* (or *below*) that curve. Otherwise, they are labeled *neither*.

4.2.1 Assigning blends to the grid points

The algorithm for assigning blends to the grid points first finds the closest dots, assigns blends, then filters to more evenly distribute the blends.

Finding the ideal set of reference dots to influence a grid point is complicated because the reference dots are not evenly distributed across the face. The algorithm attempts to find two or more dots distributed in a rough circle around the given grid point. To do this we both compensate for the dot density, by setting the search distance using the two closest dots, and by checking for dots which will both "pull" in the same direction.

To find the closest dots to the grid point p we first find δ_1 and δ_2, the distance to the closest and second closest dot, respectively. Let $D_n \subset D$ be the set of dots within $1.8\frac{\delta_1+\delta_2}{2}$ distance of p whose labels do not conflict with p's label. Next, we check for pairs of dots that are more or less in the same direction from p and remove the furthest one. More precisely, let $\hat{v}_i$ be the normalized vector from p to the dot $d_i \in D_n$ and let $\hat{v}_j$ be the normalized vector from p to the dot $d_j \in D_n$. If $\hat{v}_1 \cdot \hat{v}_2 > 0.8$ then remove the furthest of d_i and d_j from the set D_n.

We assign blend values based on the distance of the dots from p. If the dot is not in D_n then its corresponding α value is 0. For the dots in D_n let $l_i = \frac{1.0}{||d_i - p||}$. Then the corresponding α's are

$$\alpha_i = \frac{l_i}{(\sum_{d_i \in D_n} l_i)}$$

We next filter the blend coefficients for the grid points. For each grid point we find the closest grid points – since the grid points are distributed in a rough grid there will usually be 4 neighboring points – using the above routine (replacing the dots with the grid points). We special case the outlining grid points; they are only blended with other outlining grid points. The new blend coefficients are found by taking 0.75 of the grid point's blend coefficients and 0.25 of the average of the neighboring grid point's coefficients. More formally, let $g_i = [\alpha_0, \ldots, \alpha_n]$ be the vector of blend coefficients for the grid point i. Then the new vector g_i' is found as follows, where N_i is the set of neighboring grid points for the grid point i:

$$g_i' = 0.75 g_i + \frac{0.25}{||N_i||} \sum_{j \in N_i} g_j$$

We apply this filter twice to simulate a wide low pass filter.

To find the blend coefficients for the vertices of the mesh we find the closest grid point with the same label as the vertex and copy the blend coefficients. The only exception to this is the vertices for the polygons inside of the mouth. For these vertices we take β of the closest grid point on the top lip and $1.0 - \beta$ of the closest grid point on the bottom lip. The β values are 0.8, 0.6, 0.4, 0.25, and 0.1 from top to bottom of the mouth polygons.

5 Dot removal

Before we create the textures, the dots and their associated illumination effects have to be removed from the camera images. Interreflection effects are surprisingly noticeable because some parts of the face fold dramatically, bringing the reflective surface of some dots into close proximity with the skin. This is a big problem along the naso-labial furrow where diffuse interreflection from the colored dots onto the face significantly alters the skin color.

First, the dot colors are removed from each of the six camera image sequences by substituting skin texture for pixels which are covered by colored dots. Next, diffuse interreflection effects and any remaining color casts from stray pixels that have not been properly substituted are removed.

The skin texture substitution begins by finding the pixels which correspond to colored dots. The nearest neighbor color classifier

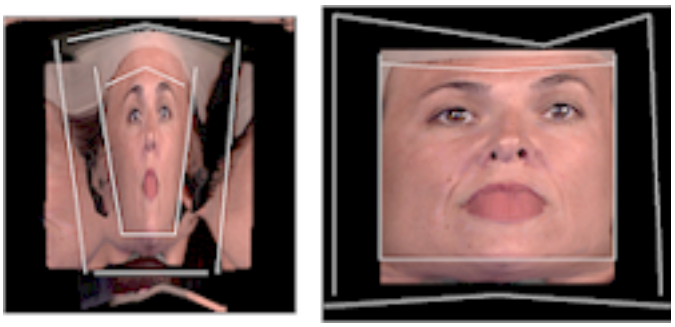


Figure 10: Standard cylindrical texture map. Warped texture map that focuses on the face, and particularly on the eyes and mouth. The warp is defined by the line pairs shown in white.

described in Section 3.1.1 is used to mark all pixels which have any of the dot colors. A special training set is used since in this case false positives are much less detrimental than they are for the dot tracking case. Also, there is no need to distinguish between dot colors, only between dot colors and the background colors. The training set is created to capture as much of the dot color and the boundary region between dots and the background colors as possible.

A dot mask is generated by applying the classifier to each pixel in the image. The mask is grown by a few pixels to account for any remaining pixels which might be contaminated by the dot color. The dot mask marks all pixels which must have skin texture substituted.

The skin texture is broken into low spatial frequency and high frequency components. The low frequency components of the skin texture are interpolated by using a directional low pass filter oriented parallel to features that might introduce intensity discontinuities. This prevents bleeding of colors across sharp intensity boundaries such as the boundary between the lips and the lighter colored regions around the mouth. The directionality of the filter is controlled by a two dimensional mask which is the projection into the image plane of a three dimensional polygon mask lying on the 3D face model. Because the polygon mask is fixed on the 3D mesh, the 2D projection of the polygon mask stays in registration with the texture map as the face deforms.

All of the important intensity gradients have their own polygon mask: the eyes, the eyebrows, the lips, and the naso-labial furrows (see 9). The 2D polygon masks are filled with white and the region of the image outside the masks is filled with black to create an image. This image is low-pass filtered. The intensity of the resulting image is used to control how directional the filter is. The filter is circularly symmetric where the image is black, i.e., far from intensity discontinuities, and it is very directional where the image is white. The directional filter is oriented so that its long axis is orthogonal to the gradient of this image.

The high frequency skin texture is created from a rectangular sample of skin texture taken from a part of the face that is free of dots. The skin sample is highpass filtered to eliminate low frequency components. At each dot mask pixel location the highpass filtered skin texture is first registered to the center of the 2D bounding box of the connected dot region and then added to the low frequency interpolated skin texture.

The remaining diffuse interreflection effects are removed by clamping the hue of the skin color to a narrow range determined from the actual skin colors. First the pixel values are converted from RGB to HSV space and then any hue outside the legal range is clamped to the extremes of the range. Pixels in the eyes and mouth, found using the eye and lip masks shown in Figure 9, are left unchanged.

Some temporal variation remains in the substituted skin texture due to imperfect registration of the high frequency texture from frame to frame. A low pass temporal filter is applied to the dot mask regions in the texture images, because in the texture map space the dots are relatively motionless. This temporal filter effectively eliminates the temporal texture substitution artifacts.

6 Creating the texture maps

Figure 11 is a flowchart of the texture creation process. We create texture maps for every frame of our animation in a four-step process. The first two steps are performed only once per mesh. First we define a parameterization of the mesh. Second, using this parameterization, we create a *geometry map* containing a location on the mesh for each texel. Third, for every frame, we create six preliminary texture maps, one from each camera image, along with weight maps. The weight maps indicate the relative quality of the data from the different cameras. Fourth, we take a weighted average of these texture maps to make our final texture map.

We create an initial set of texture coordinates for the head by tilting the mesh back 10 degrees to expose the nostrils and projecting the mesh vertices onto a cylinder. A texture map generated using this parametrization is shown on the left of Figure 10. We specify a set of line pairs and warp the texture coordinates using the technique described by Beier and Neely[1]. This parametrization results in the texture map shown on the right of Figure 10. Only the front of the head is textured with data from the six video streams.

Next we create the geometry map containing a mesh location for each texel. A mesh location is a triple (k, β_1, β_2) specifying a triangle k and barycentric coordinates in the triangle (β_1, β_2, $1 - \beta_1 - \beta_2$). To find the triangle identifier k for texel (u, v) we exhaustively search through the mesh's triangles to find the one that contains the texture coordinates (u, v). We then set the β_is to be the barycentric coordinates of the point (u, v) in the texture coordinates of the triangle k. When finding the mesh location for a pixel we already know in which triangles its neighbors above and to the left lie. Therefore, we speed our search by first searching through these triangles and their neighbors. However, the time required for this task is not critical as the geometry map need only be created once.

Next we create preliminary texture maps for frame f one for each camera. This is a modified version of the technique described in [11]. To create the texture map for camera c, we begin by deforming the mesh into its frame f position. Then, for each texel, we get its mesh location, (k, β_1, β_2), from the geometry map. With the 3D coordinates of triangle k's vertices and the barycentric coordinates β_i, we compute the texel's 3D location t. We transform t by camera c's projection matrix to obtain a location, (x, y), on camera c's image plane. We then color the texel with the color from camera c's image at (x, y). We set the texel's weight to the dot product of the mesh normal at t, $\hat{n}$, with the direction back to the camera, $\hat{d}$ (see Figure 12). Negative values are clamped to zero. Hence, weights are low where the camera's view is glancing. However, this weight map is not smooth at triangle boundaries, so we smooth it by convolving it with a Gaussian kernel.

Last, we merge the six preliminary texture maps. As they do not align perfectly, averaging them blurs the texture and loses detail. Therefore, we use only the texture map of our bottom, center camera for the center 46 % of the final texture map. We smoothly transition (over 23 pixels) to using a weighted average of each preliminary texture map at the sides.

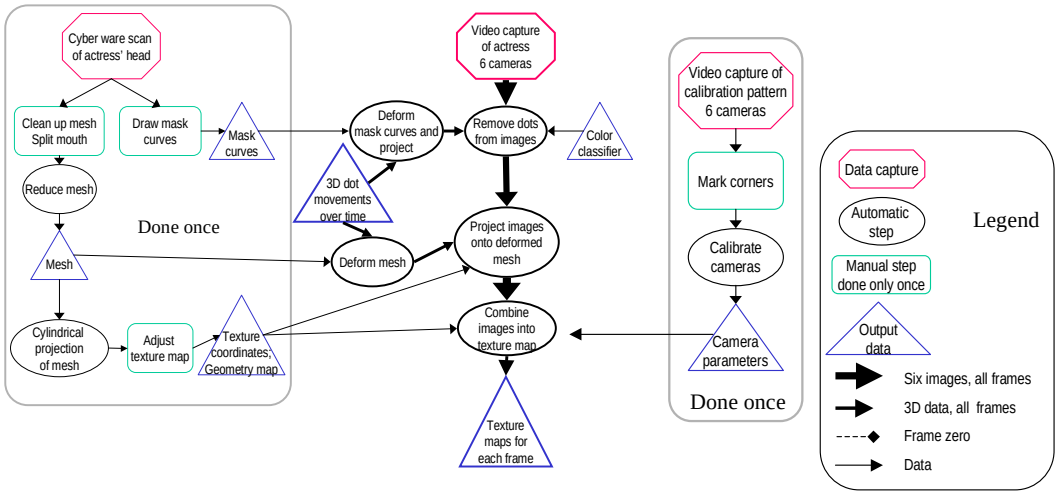


Figure 11: Creating the texture maps.

We texture the parts of the head not covered by the aforementioned texture maps with the captured reflectance data from our Cyberware scan, modified in two ways. First, because we replaced the mesh's ears with ears from a stock mesh (Section 4.1), we moved the ears in the texture to achieve better registration. Second, we set the alpha channel to zero (with a soft edge) in the region of the texture for the front of the head. Then we render in two passes to create an image of the head with both texture maps applied.

7 Compression

7.1 Principal Components Analysis

The geometric and texture map data have different statistical characteristics and are best compressed in different ways. There is significant long-term temporal correlation in the geometric data since similar facial expressions occur throughout the sequence. The short term correlation of the texture data is significantly increased over that of the raw video footage because in the texture image space the fiducials are essentially motionless. This eliminates most of the intensity changes associated with movement and leaves primarily shading changes. Shading changes tend to have low spatial frequencies and are highly compressible. Compression schemes such as MPEG, which can take advantage of short term temporal correlation, can exploit this increase in short term correlation.

For the geometric data, one way to exploit the long term correlation is to use principal component analysis. If we represent our data set as a matrix A, where frame i of the data maps column i of A, then the first principal component of A is

$$\max_u (A^T u)^T (A^T u) \tag{2}$$

The u which maximizes Equation 2 is the eigenvector associated with the largest eigenvalue of AA^T, which is also the value of the maximum. Succeeding principal components are defined similarly, except that they are required to be orthogonal to all preceding principal components, i.e., $u_i^T u_j = 0$ for $j \neq i$. The principal components form an orthonormal basis set represented by the matrix U where the columns of U are the principal components of A ordered by eigenvalue size with the most significant principal component in the first column of U.

The data in the A matrix can be projected onto the principal component basis as follows:

$$W = U^T A$$

Row i of W is the projection of column A_i onto the basis vector u_i. More precisely, the jth element in row i of W corresponds to the projection of frame j of the original data onto the ith basis vector. We will call the elements of the W matrix projection *coefficients*.

Similarly, A can be reconstructed exactly from W by multiplication by the basis set, i.e., $A = UW$.

The most important property of the principal components for our purposes is that they are the best linear basis set for reconstruction in the l_2 norm sense. For any given matrix U_k, where k is the number of columns of the matrix and $k < rank(A)$, the reconstruction error

$$e = ||A - U_k U_k^T A||_F^2 \tag{3}$$

where $||B||_F^2$ is the Frobenius norm defined to be

$$||B||_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij}^2 \tag{4}$$

will be minimized if U_k is the matrix containing the k most significant principal components of A.

We can compress a data set A by quantizing the elements of its corresponding W and U matrices and entropy coding them. Since the compressed data cannot be reconstructed without the principal component basis vectors both the W and U matrices have to be compressed. The basis vectors add overhead that is not present with basis sets that can be computed independent of the original data set, such as the DCT basis.

For data sequences that have no particular structure the extra overhead of the basis vectors would probably out-weigh any gain in compression efficiency. However, for data sets with regular frame to frame structure the residual error for reconstruction with the principal component basis vectors can be much smaller than for other bases. This reduction in residual error can be great enough to compensate for the overhead bits of the basis vectors.

The principal components can be computed using the singular value decomposition (SVD) [13]. Efficient implementations of this algorithm are widely available. The SVD of a matrix A is

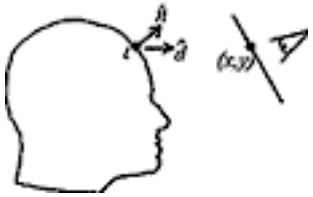


Figure 12: Creating the preliminary texture map.

$$A = U\Sigma V^T \tag{5}$$

where the columns of U are the eigenvectors of AA^T, the singular values, σ_i, along the diagonal matrix Σ are the square roots of the eigenvalues of AA^T, and the columns of V are the eigenvectors of A^TA. The ith column of U is the ith principal component of A. Computing the first k left singular vectors of A is equivalent to computing the first k principal components.

7.2 Geometric Data

The geometric data has the long term temporal coherence properties mentioned above since the motion of the face is highly structured. The overhead of the basis vectors for the geometric data is fixed because there are only 182 fiducials on the face. The maximum number of basis vectors is $182 * 3$ since there are three numbers, x, y, and z, associated with each fiducial. Consequently, the basis vector overhead steadily diminishes as the length of the animation sequence increases.

The geometric data is mapped to matrix form by taking the 3D offset data for the ith frame and mapping it the ith column of the data matrix A_g. The first k principal components, U_g, of A_g are computed and A_g is projected into the U_g basis to give the projection coefficients W_g.

There is significant correlation between the columns of projection coefficients because the motion of the dots is relatively smooth over time. We can reduce the entropy of the quantized projection coefficients by temporally predicting the projection coefficients in column i from column $i-1$, i.e., $c_i = c_{i-1} + \Delta_i$ where we encode Δ_i.

For our data set, only the projection coefficients associated with the first 45 principal components, corresponding to the first 45 rows of W_g, have significant temporal correlation so only the first 45 rows are temporally predicted. The remaining rows are entropy coded directly. After the temporal prediction the entropy is reduced by about 20 percent (Figure 13).

The basis vectors are compressed by choosing a peak error rate and then varying the number of quantization levels allocated to each vector based on the standard deviation of the projection coefficients for each vector.

We visually examined animation sequences with W_g and U_g compressed at a variety of peak error rates and chose a level which resulted in undetectable geometric jitter in reconstructed animation. The entropy of W_g for this error level is 26 Kbits per second and the entropy of U_g is 13 kbits per second for a total of 40 kbits per second for all the geometric data. These values were computed for our 3330 frame animation sequence.

8 Results

Figure 16 shows some typical frames from a reconstructed sequence of 3D facial expressions. These frames are taken from a 3330 frame animation in which the actress makes random expressions while reading from a script[2].

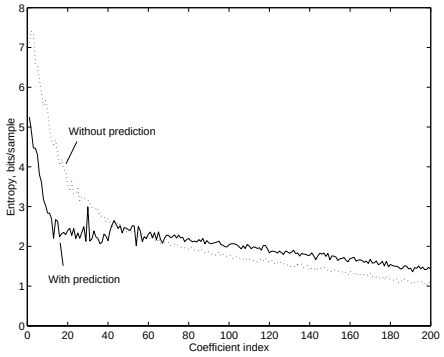


Figure 13: Reduction in entropy after temporal prediction.

The facial expressions look remarkably life-like. The animation sequence is similarly striking. Virtually all evidence of the colored fiducials and diffuse interreflection artifacts is gone, which is surprising considering that in some regions of the face, especially around the lips, there is very little of the actress' skin visible – most of the area is covered by colored fiducials.

Both the accurate 3D geometry and the accurate face texture contribute to the believability of the reconstructed expressions. Occlusion contours look correct and the subtle details of face geometry that are very difficult to capture as geometric data show up well in the texture images. Important examples of this occur at the nasolabial furrow which runs from just above the nares down to slightly below the lips, eyebrows, and eyes. Forehead furrows and wrinkles also are captured. To recreate these features using geometric data rather than texture data would require an extremely detailed 3D capture of the face geometry and a resulting high polygon count in the 3D model. In addition, shading these details properly if they were represented as geometry would be difficult since it would require computing shadows and possibly even diffuse interreflection effects in order to look correct. Subtle shading changes on the smooth parts of the skin, most prominent at the cheekbones, are also captured well in the texture images.

There are still visible artifacts in the animation, some of which are polygonization or shading artifacts, others of which arise because of limitations in our current implementation.

Some polygonization of the face surface is visible, especially along the chin contour, because the front surface of the head contains only 4500 polygons. This is not a limitation of the algorithm – we chose this number of polygons because we wanted to verify that believable facial animation could be done at polygon resolutions low enough to potentially be displayed in real time on inexpensive ($200) 3D graphics cards[3]. For film or television work, where real time rendering is not an issue, the polygon count can be made much higher and the polygonization artifacts will disappear. As graphics hardware becomes faster the differential in quality between offline and online rendered face images will diminish.

Several artifacts are simply the result of our current implementation. For example, occasionally the edge of the face, the tips of the nares, and the eyebrows appear to jitter. This usually occurs when dots are lost, either by falling below the minimum size threshold or by not being visible to three or more cameras. When a dot is lost the algorithm synthesizes dot position data which is

[2] The rubber cap on the actress' head was used to keep her hair out of her face.

[3] In this paper we have not addressed the issue of real time texture decompression and rendering of the face model, but we plan to do so in future work

usually incorrect enough that it is visible as jitter. More cameras, or better placement of the cameras, would eliminate this problem. However, overall the image is extremely stable.

In retrospect, a mesh constructed by hand with the correct geometry and then fit to the cyberware data [10] would be much simpler and possibly reduce some of the polygonization artifacts.

Another implementation artifact that becomes most visible when the head is viewed near profile is that the teeth and tongue appear slightly distorted. This is because we do not use correct 3D models to represent them. Instead, the texture map of the teeth and tongue is projected onto a sheet of polygons stretching between the lips. It is possible that the teeth and tongue could be tracked using more sophisticated computer vision techniques and then more correct geometric models could be used.

Shading artifacts represent an intrinsic limitation of the algorithm. The highlights on the eyes and skin remain in fixed positions regardless of point of view, and shadowing is fixed at the time the video is captured. However, for many applications this should not be a limitation because these artifacts are surprisingly subtle. Most people do not notice that the shading is incorrect until it is pointed out to them, and even then frequently do not find it particularly objectionable. The highlights on the eyes can probably be corrected by building a 3D eye model and creating synthetic highlights appropriate for the viewing situation. Correcting the skin shading and self shadowing artifacts is more difficult. The former will require very realistic and efficient skin reflectance models while the latter will require significant improvements in rendering performance, especially if the shadowing effect of area light sources is to be adequately modeled. When both these problems are solved then it will no longer be necessary to capture the live video sequence – only the 3D geometric data and skin reflectance properties will be needed.

The compression numbers are quite good. Figure 14 shows a single frame from the original sequence, the same frame compressed by the MPEG4 codec at 460 Kbps and at 260 KBps. All of the images look quite good. The animated sequences also look good, with the 260 KBps sequence just beginning to show noticeable compression artifacts. The 260 KBps video is well within the bandwidth of single speed CDROM drives. This data rate is probably low enough that decompression could be performed in real time in software on the fastest personal computers so there is the potential for real time display of the resulting animations. We intend to investigate this possibility in future work.

There is still room for significant improvement in our compression. A better mesh parameterization would significantly reduce the number of bits needed to encode the eyes, which distort significantly over time in the texture map space. Also the teeth, inner edges of the lips, and the tongue could potentially be tracked over time and at least partially stabilized, resulting in a significant reduction in bit rate for the mouth region. Since these two regions account for the majority of the bit budget, the potential for further reduction in bit rate is large.

9 Conclusion

The system produces remarkably lifelike reconstructions of facial expressions recorded from live actors' performances. The accurate 3D tracking of a large number of points on the face results in an accurate 3D model of facial expression. The texture map sequence captured simultaneously with the 3D deformation data captures details of expression that would be difficult to capture any other way. By using the 3D deformation information to register the texture maps from frame to frame the variance of the texture map sequence is significantly reduced which increases its compressibility. Image quality of 30 frame per second animations, reconstructed at approximately 300 by 400 pixels, is still good at data rates as low as 240 Kbits per second, and there is significant potential for lowering this bit rate even further. Because the bit overhead for the geometric data is low in comparison to the texture data one can get a 3D talking head, with all the attendant flexibility, for little more than the cost of a conventional video sequence. With the true 3D model of facial expression, the animation can be viewed from any angle and placed in a 3D virtual environment, making it much more flexible than conventional video.

References

[1] BEIER, T., AND NEELY, S. Feature-based image metamorphosis. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 35–42.

[2] BREGLER, C., COVELL, M., AND SLANEY, M. Video rewrite: Driving visual speech with audio. *Computer Graphics 31*, 2 (Aug. 1997), 353–361.

[3] CASSELL, J., PELACHAUD, C., BADLER, N., STEEDMAN, M., ACHORN, B., BECKET, T., DOUVILLE, B., PREVOST, S., AND STONE, M. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. *Computer Graphics 28*, 2 (Aug. 1994), 413–420.

[4] DECARLO, D., AND METAXAS, D. The integration of optical flow and deformable models with applications to human face shape and motion estimation. *Proceedings CVPR* (1996), 231–238.

[5] ESSA, I., AND PENTLAND, A. Coding, analysis, interpretation and recognition of facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 7 (1997), 757–763.

[6] FAUGERAS, O. *Three-dimensional computer vision*. MIT Press, Cambridge, MA, 1993.

[7] FISCHLER, M. A., AND BOOLES, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (Aug. 1981), 381–395.

[8] HOPPE, H. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 99–108. held in New Orleans, Louisiana, 04-09 August 1996.

[9] HORN, B. K. P. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America 4*, 4 (Apr. 1987).

[10] LEE, Y., TERZOPOULOS, D., AND WATERS, K. Realistic modeling for facial animation. *Computer Graphics 29*, 2 (July 1995), 55–62.

[11] PIGHIN, F., AUSLANDER, J., LISHINSKI, D., SZELISKI, R., AND SALESIN, D. Realistic facial animation using image based 3d morphing. Tech. Report TR-97-01-03, Department of Computer Science and Engineering, University of Washington, Seattle, Wa, 1997.

[12] SCHÜRMANN, J. *Pattern Classification: A Unified View of Statistical and Neural Approaches*. John Wiley and Sons, Inc., New York, 1996.

Figure 14: Left to Right: Mesh with uncompressed textures, compressed to 400 kbits/sec, and compressed to 200 kbits/sec

[13] STRANG. *Linear Algebra and its Application*. HBJ, 1988.

[14] WATERS, K. A muscle model for animating three-dimensional facial expression. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), M. C. Stone, Ed., vol. 21, pp. 17–24.

[15] WILLIAMS, L. Performance-driven facial animation. *Computer Graphics 24*, 2 (Aug. 1990), 235–242.

Figure 15: Face before and after dot removal, with details showing the steps in the dot removal process. From left to right, top to bottom: Face with dots, dots replaced with low frequency skin texture, high frequency skin texture added, hue clamped.

Figure 16: Sequence of rendered images of textured mesh.

Video Motion Capture

Christoph Bregler Jitendra Malik

Computer Science Division
University of California, Berkeley
Berkeley, CA 94720-1776
bregler@cs.berkeley.edu, malik@cs.berkeley.edu

Abstract

This paper demonstrates a new vision based motion capture technique that is able to recover high degree-of-freedom articulated human body configurations in complex video sequences. It does not require any markers, body suits, or other devices attached to the subject. The only input needed is a video recording of the person whose motion is to be captured. For visual tracking we introduce the use of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation. This results in solving simple linear systems, and enables us to recover robustly the kinematic degrees-of-freedom in noise and complex self occluded configurations. We demonstrate this on several image sequences of people doing articulated full body movements, and visualize the results in re-animating an artificial 3D human model. We are also able to recover and re-animate the famous movements of Eadweard Muybridge's motion studies from the last century. To the best of our knowledge, this is the first computer vision based system that is able to process such challenging footage and recover complex motions with such high accuracy.

CR Categories:

Keywords: Computer Vision, Animation, Motion Capture, Visual Tracking, Twist Kinematics, Exponential Maps, Muybridge

1 Introduction

In this paper, we offer a new approach to motion capture based just on ordinary video recording of the actor performing naturally. The approach does not require any markers, body suits or any other devices attached to the body of the actor. The actor can move about wearing his or her regular clothes. This implies that one can use historical footage–motion capture Charlie Chaplin's inimitable walk, for instance. Indeed in this paper we shall go even further back historically and show motion capture results from Muybridge sequences–*the* first examples of photographically recorded motion [15].

Motion capture occupies an important role in the creation of special effects. Its application to CG character animation has been much more controversial; SIGGRAPH 97 featured a lively panel debate[4] between its proponents and opponents. Our goal in this paper is not to address that debate. Rather we take it as a given that motion capture, like any other technology, can be correctly or incorrectly applied and we are merely extending its possibilities.

Our approach, from a user's point of view, is rather straightforward. The user marks limb segments in an initial frame; if multiple video streams are available from synchronized cameras, then the limb segments are marked in the corresponding initial frames in all of them. The computer program does the rest–tracking the multiple degrees of freedom of the human body configuration from frame to frame.

Attempts to track the human body without special markers go back quite a few years – we review past work in Sec. 2. However in spite of many years of work in computer vision on this problem, it is fair to describe it as not yet solved. There are many reasons why human body tracking is very challenging, compared to tracking other objects such as footballs, robots or cars. These include

1. *High Accuracy Requirements.* Especially in the context of motion capture applications, one desires to record all the degrees of freedom of the configuration of arms, legs, torso, head etc accurately from frame to frame. At playback time, any error will be instantly noticed by a human observer.

2. *Frequent inter-part occlusion* During normal motion, from any camera angle some parts of the body are occluded by other parts of the body

3. *Lack of contrast* Distinguishing the edge of a limb from, say the torso underneath, is made difficult by the fact that typically the texture or color of the shirt is usually the same in both regions.

Our contribution to this problem is the introduction of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation scheme. This formalism will be explained fully in Section 3. The advantage of this particular formulation is that it results in the equations that need to be solved to update the kinematic chain parameters from frame to frame being *linear*. Also the only parameters that need to be solved for are the true degrees of freedom and pose parameters–there are no intermediate stages which may be unnecessarily hard. For instance recovering the local affine motion parameters of each and every limb segment separately is harder than the final goal of knowing the configuration of all the joints from frame to frame–the fact that the joints are constrained to move together reduces considerably the number of degrees of freedom. This in turn provides robustness to self-occlusions, loss of contrast, large motions etc.

We applied this technique to several video recordings of walking people and to the famous photo plates of Edweard Muybridge. We achieved accurate tracking results with high degree-of-freedom full body models and could successfully re-animate the data. The accompanying video shows the tracking results and the naturalness of the animated motion capture data.

Section 2 reviews previous video tracking techniques, section 3 introduces the new motion tracking framework and its mathematical formulation, section 4 details our experiments, and we discuss the results and future directions in section 5.

2 Review

The earliest computer vision attempt to recognize human movements was reported by O'Rouke and Badler [16] working on syn-

thetic images using a 3D structure of rigid segments, joints, and constraints between them.

Marker-free visual tracking on video recordings of human bodies goes back to work by Hogg and by Rohr [8, 18]. Both systems are specialized to one degree-of-freedom walking models. Edge and line features are extracted from images and matched to a cylindrical 3D human body model. Higher degree-of-freedom articulated hand configurations are tracked by Regh and Kanade [17], full body configurations by Gravrila and Davis [7], and arm configurations by Kakadiaris and Metaxas [11] and by Goncalves and Perona [5]. All these approaches are demonstrated in constrained environments with high contrast edge boundaries. In most cases this is achieved by uniform backgrounds, and skintight clothing of uniform color. Also, in order to estimate 3D configurations, a camera calibration is needed. Alternatively, Weng et. al demonstrated how to track full bodies with color features [20], and Ju et. al showed motion based tracking of leg configurations [10]. No 3D kinematic chain models were used in the last two cases.

To the best of our knowledge, there is no system reported so far, which would be able to successfully track accurate high-degree-of freedom human body configurations in the challenging footage that we will demonstrate here.

3 Articulated Tracking

There exist a wide range of visual tracking techniques in the literature ranging from edge feature based to region based tracking, and brute-force search methods to differential approaches.

Edge feature based tracking techniques usually require clean data with high contrast object boundaries. Unfortunately on human bodies such features are very noisy. Clothes have many folds. Also if the left and right leg have the same color and they overlap, they are separated only by low contrast boundaries.

Region based techniques can track objects with arbitrary texture. Such techniques attempt to match areas between consecutive frames. For example if the area describes a rigid planar object, a 2D affine deformation of this area has to be found. This requires the estimation of 6 free parameters that describe this deformation (x/y translation, x/y scaling, rotation, and shear). Instead of exhaustively searching over these parameters, differential methods link local intensity changes to parameter changes, and allow for Newton-step like optimizations.

In the following we will introduce a new region based differential technique that is tailored to articulated objects modeled by kinematic chains. We will first review a commonly used motion estimation framework [2, 19], and then show how this can be extended for our task, using the twist and product of exponential formulation [14].

3.1 Preliminaries

Assuming that changes in image intensity are only due to translation of local image intensity, a parametric image motion between consecutive time frames t and $t+1$ can be described by the following equation:

$$I(x + \mathbf{u}_x(x, y, \phi), y + \mathbf{u}_y(x, y, \phi), t + 1) = I(x, y, t) \qquad (1)$$

$I(x, y, t)$ is the image intensity. The motion model $\mathbf{u}(x, y, \phi) = [\mathbf{u}_x(x, y, \phi), \mathbf{u}_y(x, y, \phi)]^T$ describes the pixel displacement dependent on location (x, y) and model parameters ϕ. For example, a 2D affine motion model with parameters $\phi = [a_1, a_2, a_3, a_4, d_x, d_y]^T$ is defined as

$$\mathbf{u}(x, y, \phi) = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \qquad (2)$$

The first-order Taylor series expansion of (1) leads to the commonly used gradient formulation [12]:

$$I_t(x, y) + [I_x(x, y), I_y(x, y)] \cdot \mathbf{u}(x, y, \phi) = 0 \qquad (3)$$

$I_t(x, y)$ is the temporal image gradient and $[I_x(x, y), I_y(x, y)]$ is the spatial image gradient at location (x, y). Assuming a motion model of K degrees of freedom (in case of the affine model $K = 6$) and a region of $N > K$ pixels, we can write an over-constrained set of N equations. For the case that the motion model is linear (as in the affine case), we can write the set of equations in matrix form (see [2] for details):

$$\mathbf{H} \cdot \phi + \vec{z} = \vec{0} \qquad (4)$$

where $\mathbf{H} \in \Re^{N \times K}$, and $\vec{z} \in \Re^N$. The least squares solution to (3) is:

$$\phi = -\left(\mathbf{H}^T \cdot \mathbf{H}\right)^{-1} \cdot \mathbf{H}^T \vec{z} \qquad (5)$$

Because (4) is the first-order Taylor series linearization of (1), we linearize around the new solution and iterate. This is done by warping the image $I(t + 1)$ using the motion model parameters ϕ found by (5). Based on the re-warped image we compute the new image gradients (3). Repeating this process is equivalent to a Newton-Raphson style minimization.

A convenient representation of the shape of an image region is a probability mask $w(x, y) \in [0, 1]$. $w(x, y) = 1$ declares that pixel (x, y) is part of the region. Equation (5) can be modified, such that it weights the contribution of pixel location (x, y) according to $w(x, y)$:

$$\phi = -\left((\mathbf{W} \cdot \mathbf{H})^T \cdot \mathbf{H}\right)^{-1} \cdot (\mathbf{W} \cdot \mathbf{H})^T \vec{z} \qquad (6)$$

$\mathbf{W}$ is an $N \times N$ diagonal matrix, with $\mathbf{W}(i, i) = w(x_i, y_i)$. We assume for now that we know the exact shape of the region. For example, if we want to estimate the motion parameters for a human body part, we supply a weight matrix $\mathbf{W}$ that defines the image support map of that specific body part, and run this estimation technique for several iterations. Section 3.4 describes how we can estimate the shape of the support maps as well.

Tracking over multiple frames can be achieved by applying this optimization technique successively over the complete image sequence.

3.2 Twists and the Product of Exponential Formula

In the following we develop a motion model $\mathbf{u}(x, y, \phi)$ for a 3D kinematic chain under scaled orthographic projection and show how these domain constraints can be incorporated into one linear system similar to (6). ϕ will represent the 3D pose and angle configuration of such a kinematic chain and can be tracked in the same fashion as already outlined for simpler motion models.

3.2.1 3D pose

The pose of an object relative to the camera frame can be represented as a rigid body transformation in $\Re^3$ using homogeneous coordinates (we will use the notation from [14]):

$$q_c = \mathbf{G} \cdot q_o \;\text{ with }\; \mathbf{G} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & d_x \\ r_{2,1} & r_{2,2} & r_{2,3} & d_y \\ r_{3,1} & r_{3,2} & r_{3,3} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (7)$$

$q_o = [x_o, y_o, z_o, 1]^T$ is a point in the object frame and $q_c = [x_c, y_c, z_c, 1]^T$ is the corresponding point in the camera frame. Using scaled orthographic projection with scale s, the point q_c in the camera frame gets projected into the image point $[x_{im}, y_{im}]^T = s \cdot [x_c, y_c]^T$.

The 3D translation $[d_x, d_y, d_z]^T$ can be arbitrary, but the rotation matrix:

$$\mathbf{R} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix} \in SO(3) \tag{8}$$

has only 3 degrees of freedom. Therefore the rigid body transformation $\mathbf{G} \in SE(3)$ has a total of 6 degrees of freedom.

Our goal is to find a model of the image motion that is parameterized by 6 degrees of freedom for the 3D rigid motion and the scale factor s for scaled orthographic projection. *Euler angles* are commonly used to constrain the rotation matrix to $SO(3)$, but they suffer from singularities and don't lead to a simple formulation in the optimization procedure (for example [1] propose a 3D ellipsoidal tracker based on Euler angles). In contrast, the *twist* representation provides a more elegant solution [14] and leads to a very simple linear representation of the motion model. It is based on the observation that every rigid motion can be represented as a rotation around a 3D axis and a translation along this axis. A twist ξ has two representations: (a) a 6D vector, or (b) a 4×4 matrix with the upper 3×3 component as a skew-symmetric matrix:

$$\xi = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad \text{or} \quad \hat{\xi} = \begin{bmatrix} 0 & -\omega_z & \omega_y & v_1 \\ \omega_z & 0 & -\omega_x & v_2 \\ -\omega_y & \omega_x & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{9}$$

ω is a 3D unit vector that points in the direction of the rotation axis. The amount of rotation is specified with a scalar angle θ that is multiplied by the twist: $\xi\theta$. The v component determines the location of the rotation axis and the amount of translation along this axis. See [14] for a detailed geometric interpretation. For simplicity, we drop the constraint that ω is unit, and discard the θ coefficient. Therefore $\xi \in \Re^6$.

It can be shown [14] that for any arbitrary $\mathbf{G} \in SE(3)$ there exists a $\xi \in \Re^6$ twist representation.

A twist can be converted into the $\mathbf{G}$ representation with following exponential map:

$$\mathbf{G} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & d_x \\ r_{2,1} & r_{2,2} & r_{2,3} & d_y \\ r_{3,1} & r_{3,2} & r_{3,3} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \mathbf{e}^{\hat{\xi}} = \mathbf{I} + \hat{\xi} + \frac{(\hat{\xi})^2}{2!} + \frac{(\hat{\xi})^3}{3!} + ... \tag{10}$$

3.2.2 Twist motion model

At this point we would like to track the 3D pose of a rigid object under scaled orthographic projection. We will extend this formulation in the next section to a kinematic chain representation. The pose of an object is defined as $[s, \xi^T]^T = [s, v_1, v_2, v_3, \omega_x, \omega_y, \omega_z]^T$. A point q_o in the object frame is projected to the image location (x_{im}, y_{im}) with:

$$\begin{bmatrix} x_{im} \\ y_{im} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot s \cdot \mathbf{e}^{\hat{\xi}} \cdot q_o \tag{11}$$

The image motion of point (x_{im}, y_{im}) from time t to time $t+1$ is:

$$\begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} = \begin{bmatrix} x_{im}(t+1) - x_{im}(t) \\ y_{im}(t+1) - y_{im}(t) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left(s(t+1) \cdot \mathbf{e}^{\hat{\xi}(t+1)} \cdot q_o - s(t) \cdot \mathbf{e}^{\hat{\xi}(t)} \cdot q_o \right)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left((1+s') \cdot \mathbf{e}^{\hat{\xi}'} - \mathbf{I} \right) \cdot s(t) q_c \tag{12}$$

$$\text{with} \quad \hat{\xi}(t+1) = \hat{\xi}(t) + \hat{\xi}'$$
$$s(t+1) = s(t) \cdot (1+s')$$

Using the first order Taylor expansion from (10) we can approximate:

$$(1+s') \cdot \mathbf{e}^{\hat{\xi}} \approx (1+s') \cdot \mathbf{I} + (1+s') \cdot \hat{\xi} \tag{13}$$

and can rewrite (12) as:

$$\begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} = \begin{bmatrix} s' & -\omega'_z & \omega'_y & v'_1 \\ \omega'_z & s' & -\omega'_x & v'_2 \end{bmatrix} \cdot q_c \tag{14}$$

with

$$\omega(t+1) = \omega(t) + \frac{1}{1+s'} \cdot \omega'$$

$$v(t+1) = v(t) + \frac{1}{1+s'} \cdot v'$$

$\phi = [s', v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z]^T$ codes the relative scale and twist motion from time t to $t+1$. Note that (14) does not include v'_3. Translation in the Z direction of the camera frame is not measurable under scaled orthographic projection.

Equation (14) describes the image motion of a point (x_i, y_i) in terms of the motion parameters ϕ and the corresponding 3D point $q_c(i)$ in the camera frame. The 3D point $q_c(i)$ is computed by intersecting the camera ray of the image point (x_i, y_i) with the 3D model. In this paper we assume that the body segments can be approximated by ellipsoidal 3D blobs. Therefore q_c is the solution of a quadratic equation. This computation has to be done only once for each new image. It is outside the Newton-Raphson iterations. It could be replaced by more complex models and rendering algorithms.

Inserting (14) into (3) leads to:

$$I_t + I_x \cdot [s', -\omega'_z, \omega'_y, v'_1] \cdot q_c + I_y \cdot [\omega'_z, s', -\omega'_x, v'_2] \cdot q_c = 0$$

$$\Leftrightarrow \quad I_t(i) + H_i \cdot [s, v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z]^T = 0 \tag{15}$$

$$\text{with} \quad I_t := I_t(x_i, y_i), I_x := I_x(x_i, y_i), I_y := I_y(x_i, y_i)$$

For N pixel positions we have N equations of the form (15). This can be written in matrix form:

$$\mathbf{H} \cdot \phi + \vec{z} = \mathbf{0} \tag{16}$$

with

$$\mathbf{H} = \begin{bmatrix} H_1 \\ H_2 \\ ... \\ H_N \end{bmatrix} \quad \text{and} \quad \vec{z} = \begin{bmatrix} I_t(x_1, y_1) \\ I_t(x_2, y_2) \\ ... \\ I_t(x_N, y_N) \end{bmatrix}$$

Finding the least-squares solution (3D twist motion ϕ) for this equation is done using (6).

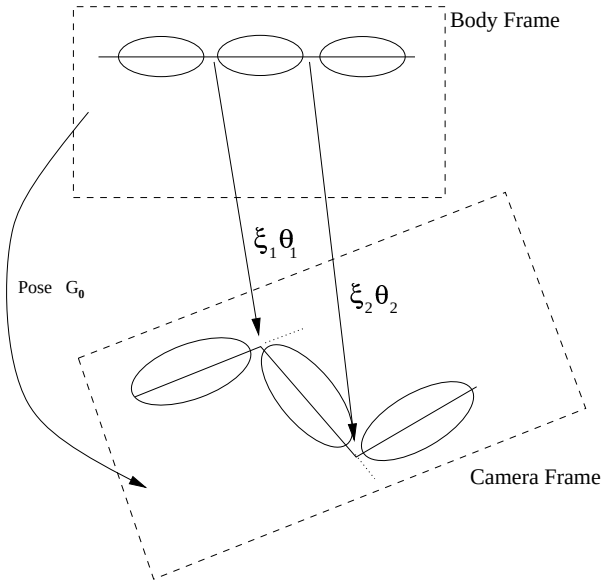


Figure 1: Kinematic chain defined by twists

3.2.3 Kinematic chain as a Product of Exponentials

So far we have parameterized the 3D pose and motion of a body segment by the 6 parameters of a twist ξ. Points on this body segment in a canonical object frame are transformed into a camera frame by the mapping $\mathbf{G_0} = e^{\hat{\xi}}$. Assume that a second body segment is attached to the first segment with a joint. The joint can be defined by an axis of rotation in the object frame. We define this rotation axis in the object frame by a 3D unit vector ω_1 along the axis, and a point q_1 on the axis (figure 1). This is a so called revolute joint, and can be modeled by a twist ([14]):

$$\xi_1 = \begin{bmatrix} -\omega_1 \times q_1 \\ \omega_1 \end{bmatrix} \tag{17}$$

A rotation of angle θ_1 around this axis can be written as:

$$\mathbf{g_1} = e^{\hat{\xi_1}\cdot\theta_1} \tag{18}$$

(19)

The global mapping from object frame points on the first body segment into the camera frame is described by the following product:

$$\begin{aligned} \mathbf{g}(\theta_1) &= \mathbf{G_0} \cdot e^{\hat{\xi_1}\cdot\theta_1} \\ q_c &= \mathbf{g}(\theta_1) \cdot q_o \end{aligned} \tag{20}$$

If we have a chain of $K+1$ segments linked with K joints (kinematic chain) and describe each joint by a twist ξ_k, a point on segment k is mapped from the object frame into the camera frame dependent on $\mathbf{G_0}$ and angles θ_1, θ_2, ..., θ_k:

$$\mathbf{g_k}(\theta_1, \theta_2, ..., \theta_k) = \mathbf{G_0} \cdot e^{\hat{\xi_1}\cdot\theta_1} \cdot e^{\hat{\xi_2}\cdot\theta_2} \cdot ... \cdot e^{\hat{\xi_k}\cdot\theta_k} \tag{21}$$

This is called the **product of exponential maps** for kinematic chains.

The velocity of a segment k can be described with a twist V_k that is a linear combination of twists $\xi_1', \xi_2', ..., \xi_k'$ and the angular velocities $\dot{\theta}_1, \dot{\theta}_2, ..., \dot{\theta}_k$ (see [14] for the derivations):

$$\begin{aligned} V_k &= \xi_1' \cdot \dot{\theta}_1 + \xi_2' \cdot \dot{\theta}_2 + ...\xi_k' \cdot \dot{\theta}_k \\ \xi_k' &= \mathbf{Ad}_{e^{\hat{\xi_1}\theta_1}\cdot ... \cdot e^{\hat{\xi_{k-1}}\theta_{k-1}}} \xi_k \end{aligned} \tag{22}$$

$\mathbf{Ad}_g$ is the adjoint transformation associated with g.[1]

Given a point q_c on the k'th segment of a kinematic chain, its motion vector in the image is related to the angular velocities by:

$$\begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left[\hat{\xi}_1' \cdot \dot{\theta}_1 + \hat{\xi}_2' \cdot \dot{\theta}_2 + ... + \hat{\xi}_k' \cdot \dot{\theta}_k\right] \cdot q_c \tag{23}$$

Recall (15) relates the image motion of a point q_c to changes in pose $\mathbf{G_0}$. We combine (15) and (23) to relate the image motion to the combined vector of pose change and angular change $\Phi = [s', v_1', v_2', \omega_x', \omega_y', \omega_z', \dot{\phi}_1, \dot{\phi}_2, ..., \dot{\phi}_K]^T$:

$$I_t + H_i \cdot [s, v_1', v_2', \omega_x', \omega_y', \omega_z']^T + J_i \cdot [\dot{\theta}_1, \dot{\theta}_2, ...\dot{\theta}_K]^T = 0 \tag{24}$$

$$[\mathbf{H}, \mathbf{J}] \cdot \Phi + \vec{z} = \mathbf{0} \tag{25}$$

with

$$\mathbf{J} = \begin{bmatrix} J_1 \\ J_2 \\ ... \\ J_N \end{bmatrix} \qquad \text{and } \mathbf{H}, \vec{z} \text{ as before}$$

$$J_i = [J_{i1}, J_{i2}, ..., J_{iK}]$$

$$J_{ik} = \begin{cases} [I_x, I_y] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \hat{\xi}_k \cdot q_c \\ 0 \qquad \text{if pixel } i \text{ is on a segment that is not affected by joint } \xi_k \end{cases}$$

The least squares solution to (25) is:

$$\Phi = -([\mathbf{H}, \mathbf{J}]^T \cdot [\mathbf{H}, \mathbf{J}])^{-1} \cdot [\mathbf{H}, \mathbf{J}]^T \cdot \vec{z} \tag{26}$$

Φ is the new estimate of the pose and angular change between two consecutive images. As outlined earlier, this solution is based on the assumption that the local image intensity variations can be approximated by the first-order Taylor expansion (3). We linearize around this new solution and iterate. This is done in warping the image $I(t+1)$ using the solution Φ. Based on the re-warped image we compute the new image gradients. Repeating this process of warping and solving (26) is equivalent to a Newton-Raphson style minimization.

3.3 Multiple Camera Views

In cases where we have access to multiple synchronized cameras, we can couple the different views in one equation system. Let's assume we have C different camera views at the same time. View c corresponds to following equation system (from (25)):

$$[\mathbf{H_c}, \mathbf{J_c}] \cdot \begin{bmatrix} \Omega_c \\ \dot{\phi}_1 \\ \dot{\phi}_2 \\ ... \\ \dot{\phi}_K \end{bmatrix} + \vec{z_c} = \mathbf{0} \tag{27}$$

$\Omega_c = [s_c', v_{1,c}', v_{2,c}', \omega_{x,c}', \omega_{y,c}', \omega_{z,c}']^T$ describes the pose seen from view c. All views share the same angular parameters, because

[1] $\mathbf{Ad}_g = \begin{bmatrix} R & \hat{p} \cdot R \\ \mathbf{0} & R \end{bmatrix}$, and $g = \begin{bmatrix} R & p \\ 000 & 1 \end{bmatrix}$

the cameras are triggered at the same time. We can simply combine all C equation systems into one large equation system:

$$\begin{bmatrix} \mathbf{H_1} & \mathbf{0} & ... & \mathbf{0} & \mathbf{J_1} \\ \mathbf{0} & \mathbf{H_2} & ... & \mathbf{0} & \mathbf{J_2} \\ ... & ... & ... & ... & ... \\ \mathbf{0} & \mathbf{0} & ... & \mathbf{H_C} & \mathbf{J_C} \end{bmatrix} \cdot \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ ... \\ \Omega_C \\ \dot{\phi}_1 \\ \dot{\phi}_2 \\ ... \\ \dot{\phi}_K \end{bmatrix} + \begin{bmatrix} \vec{z_1} \\ \vec{z_2} \\ ... \\ \vec{z_C} \end{bmatrix} = \mathbf{0} \tag{28}$$

Operating with multiple views has three main advantages. The estimation of the angular parameters is more robust: (1) the number of measurements and therefore the number of equations increases with the number of views, (2) some angular configurations might be close to a singular pose in one view, whereas they can be estimated in a orthogonal view much better. (3) With more camera views, the chance decreases that one body part is occluded in all views.

3.4 Adaptive Support Maps using EM

As in (3), the update can be constrained to estimate the motion only in a weighted support map $\mathbf{W}_k$ for each segment k using:

$$\Phi = -\left((\mathbf{W}_k \cdot [\mathbf{H},\mathbf{J}])^T \cdot [\mathbf{H},\mathbf{J}]\right)^{-1} \cdot (\mathbf{W}_k \cdot [\mathbf{H},\mathbf{J}])^T \vec{z} \tag{29}$$

We approximate the shape of the body segments as ellipsoids, and can compute the support map as the projection of the ellipsoids into the image. Such a support map usually covers a larger region, including pixels from the environment. That distracts the exact motion measurement. Robust statistics would be one solution to this problem [3]. Another solution is an EM-based layered representation [6, 9]. It is beyond the scope of this paper to describe this method in detail, but we would like to outline the method briefly: We start with an initial guess of the support map (ellipsoidal projection in this case). Given the initial $\mathbf{W}_k$, we compute the motion estimate Φ (M-step). Given such a Φ we can compute for each pixel location the probability that it complies with the motion model defined by Φ. We do this for each blob and the background (dominant motion) and normalize the sum of all probabilities per pixel location to 1. This results in new $\mathbf{W}_k$ maps that are better "tuned" to the real shape of the body segment. In this paper we repeat the EM iteration only once.

3.5 Tracking Recipe

We summarize the algorithm for tracking the pose and angles of a kinematic chain in an image sequence:

- **Input**: $I(t)$, $I(t+1)$, $\mathbf{G_0(t)}, \theta_1(t), \theta_2(t), ..., \theta_K(t)$ (Two images and the pose and angles for the first image).
- **Output:** $\mathbf{G_0(t+1)}, \theta_1(t+1), \theta_2(t+1), ..., \theta_K(t+1)$. (Pose and angles for second image).

1. Compute for each image location (x_i, y_i) in $I(t)$ the 3D point $q_c(i)$ (using ellipsoids or more complex models and rendering algorithm).
2. Compute for each body segment the support map W_k.
3. Set $\mathbf{G}_0(t+1) := \mathbf{G}_0(t)$, $\forall k : \theta_k(t+1) := \theta_k(t)$.
4. Iterate:
   (a) Compute spatiotemporal image gradients: I_t, I_x, I_y.
   (b) Estimate Φ using (29)
   (c) Update $G_0(t+1) := G_0(t+1) \cdot (1+s') \cdot e^{\frac{\xi'}{1+s'}}$
   (d) $\forall k$ Update $\theta_k(t+1) := \theta_k(t+1) + \dot{\theta}_k$.
   (e) $\forall k$ Warp the region inside W_k of $I(t+1)$ by $\mathbf{G}_0(t+1) \cdot g_k(t+1) \cdot (\mathbf{G}^(t) \cdot g_k(t))^{-1}$.

3.6 Initialization

The visual tracking is based on an initialized first frame. We have to know the initial pose and the initial angular configuration. If more than one view is available, all views for the first time step have to be known. A user clicks on the 2D joint locations in all views at the first time step. Given that, the 3D pose and the image projection of the matching angular configuration is found in minimizing the sum of squared differences between the projected model joint locations and the user supplied model joint locations. The optimization is done over the poses, angles, and body dimensions. Example body dimensions are "upper-leg-length", "lower-leg-length", or "shoulder-width". The dimensions and angles have to be the same in all views, but the pose can be different. Symmetry constraints, that the left and right body lengths are the same, are enforced as well. Minimizing only over angles, or only over model dimensions results in linear equations similar to what we have shown so far. Unfortunately the global minimization criteria over all parameters is a tri-linear equation system, that cannot be easily solved by simple matrix inversions. There are several possible techniques for minimizing such functions. We achieved good results with a Quasi-Newton method and a mixed quadratic and cubic line search procedure.

4 Results

We applied this technique to video recordings in our lab and to photo-plate sequence of Eadweard Muybdrige's motion studies.

4.1 Single camera recordings

Our lab video recordings were done with a single camera. Therefore the 3D pose and some parts of the body can not be estimated completely. Figure 2 shows one example sequences of a person walking in a frontoparallel plane. We defined a 6 DOF kinematic structure: One blob for the body trunk, three blobs for the frontal leg and foot, connected with a hip joint, knee joint, and ankle joint, and two blobs for the arm connected with a shoulder and elbow joint. All joints have an axis orientation parallel to the Z-axis in the camera frame. The head blob was connected with one joint to the body trunk. The first image in figure 2 shows the initial blob support maps.

After the hand-initialization we applied the motion tracker to a sequence of 53 image frames. We could successfully track all body parts in this video sequence (see video). The video shows that the appearance of the upper leg changes significantly due to moving folds on the subject's jeans. The lower leg appearance does not change to the same extent. The constraints were able to enforce compatible motion vectors for the upper leg, based on more reliable measurements on the lower leg.

We can compare the estimated angular configurations with motion capture data reported in the literature. Murray, Brought, and

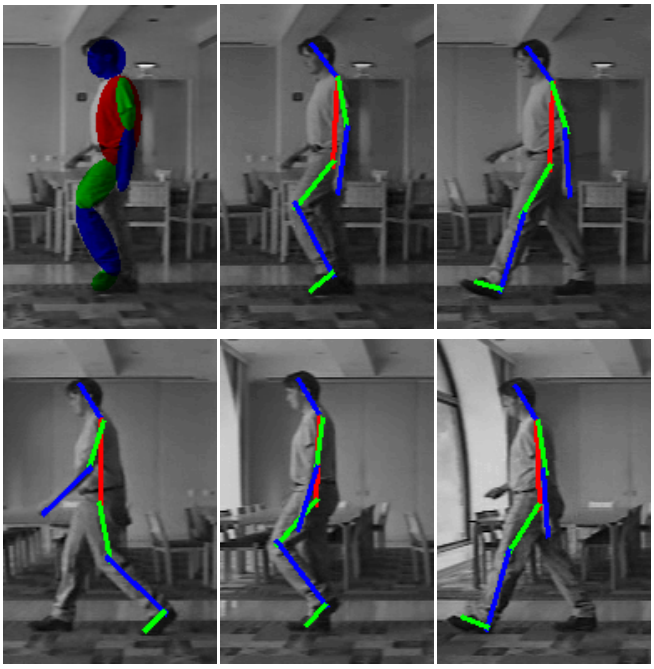


Figure 2: Example configurations of the estimated kinematic structure. First image shows the support maps of the initial configuration. In subsequent images the white lines show blob axes. The joint is the position on the intersection of two axes.

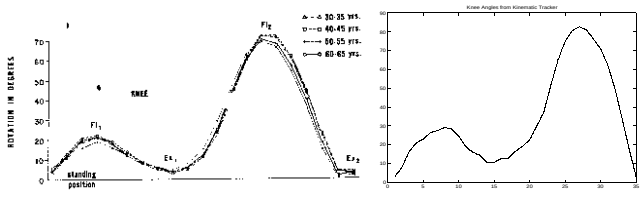


Figure 3: Comparison of a) data from [Murray et al] (left) and b) our motion tracker (right).

Kory published [13] such measurements for the hip, knee, and angle joints. We compared our motion tracker measurements with the published curves and found good agreement. Figure 4.1a shows the curves for the knee and ankle reported in [13], and figure 4.1b shows our measurements.

We also experimented with a walking sequence of a subject seen from an oblique view with a similar kinematic model. As seen in figure 4, we tracked the angular configurations and the pose successfully over the complete sequence of 45 image frames. Because we use a scaled orthographic projection model, the perspective effects of the person walking closer to the camera had to be compensated by different scales. The tracking algorithm could successfully estimate the scale changes.

4.2 Digital Muybridge

The final set of experiments was done on historic footage recorded by Eadweard Muybridge in 1884. His methods are of independent interest, as they predate motion pictures. Muybridge had his models walk in an open shed. Parallel to the shed was a fixed battery of 24 cameras. Two portable batteries of 12 cameras each were positioned at both ends of the shed, either at an angle of 90 deg relative to the shed or an angle of 60 deg. Three photographs were take

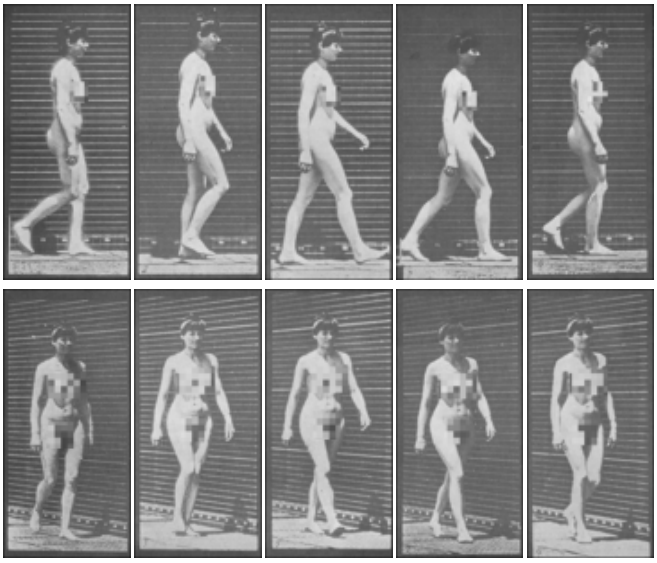


Figure 5: Eadweard Muybridge, The Human Figure in Motion, Plate 97: Woman Walking. The first 5 frames show part of a walk cycle from one example view, and the second 5 frames show the same time steps from a different view

simultaneously, one from each battery. The effective 'framerate' of his technique is about two times lower then current video frame rates; a fact which makes tracking a harder problem.. It is to our advantage that he took for each time step three pictures from different viewpoints.

Figure 4.2 and figure 4.2 shows example photo plates. We could initialize the 3D pose by labeling all three views of the first frame and running the minimization procedure over the body dimensions and poses. Figure 4.2 shows one example initialization. Every body segment was visible in at least one of the three camera views, therefore we could track the left and the right side of the person. We applied this technique to a walking woman and a walking man. For the walking woman we had 10 time steps available that contained 60 % of a full walk cycle (figure 4.2). For this set of experiments we extended our kinematic model to 19 DOFs. The two hip joints, the two shoulder joints, and the neck joint, were modeled by 3 DOFs. The two knee joints and two elbow joints were modeled just by one rotation axis. Figure 4.2 shows the tracking results with the model overlayed. As you see, we could successfully track the complete sequence. To animate the tracking results we mirrored the left and right side angles to produce the remaining frames of a complete walk cycle. We animated the 3D motion capture data with a stick figure model and a volumetric model (figure 10), and it looks very natural. The video shows some of the tracking and animation sequences from several novel camera views, replicating the walk cycle performed over a century ago on the grounds of University of Pennsylvania.

For the visualization of the walking man sequence, we did not apply the mirroring, because he was carrying a boulder on his shoulder. This made the walk asymmetric. We re-animated the original tracked motion (figure 4.2) capture data for the man, and it also looked very natural.

Given the successful application of our tracking technique to multi-view data, we are planning to record with higher frame-rates our own multi-view video footage. We also plan to record a wider range of gestures.

Figure 4: Example configurations of the estimated kinematic structure of a person seen from an oblique view.

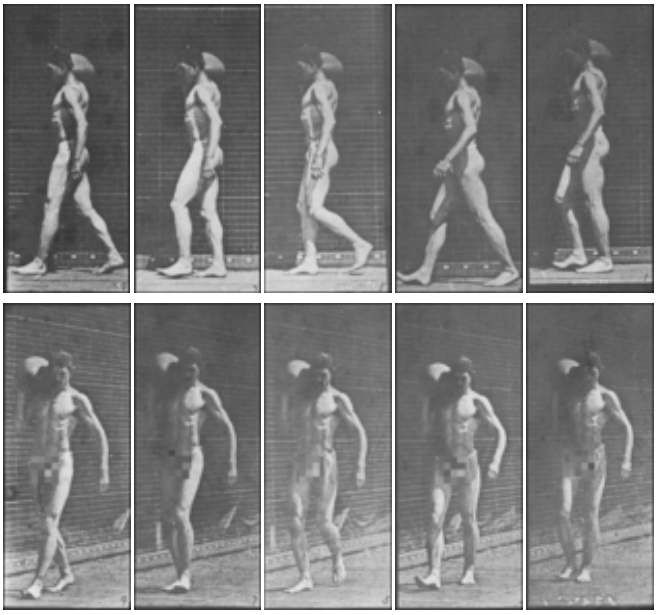


Figure 6: Eadweard Muybridge, The Human Figure in Motion, Plate 7: Man walking and carrying 75-LB boulder on shoulder. The first 5 frames show part of a walk cycle from one example view, and the second 5 frames show the same time steps from a different view

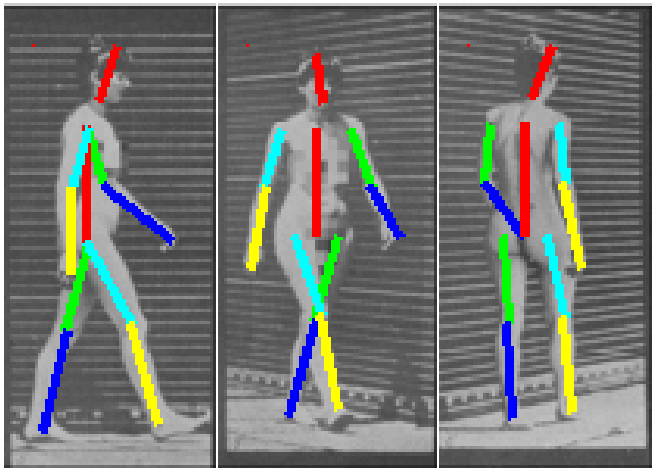


Figure 7: Initialization of Muybridge's Woman Walking: This visualizes the initial angular configuration projected to 3 example views.

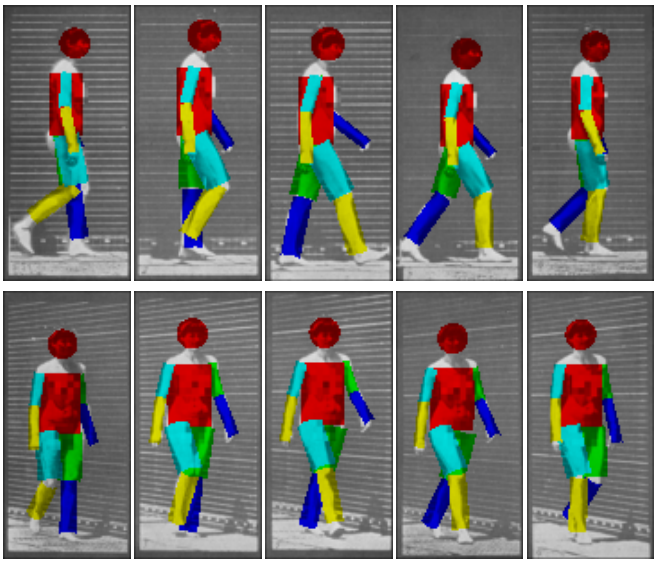


Figure 8: Muybridge's Woman Walking: Motion Capture results. This shows the tracked angular configurations and its volumetric model projected to 2 example views.

5 Conclusion

In this paper, we have developed and demonstrated a new technique for video motion capture. The approach does not require any markers, body suits or any other devices attached to the body of the actor. The actor can move about wearing his or her regular clothes. We demonstrated results on video recordings of people walking both in frontoparallel and oblique views, as well as on the classic Muybridge photographic sequences recorded more than a century ago.

Visually tracking human motion at the level of individual joints is a very challenging problem. Our results are due, in large measure, to the introduction of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation scheme. The advantage of this particular formulation is that it results in the equations that need to be solved to update the kinematic chain parameters from frame to frame being linear, and that it is not necessary to solve for any redundant or unnecessary variables.

Future work will concentrate on dealing with very large motions, as may happen, for instance, in videotapes of high speed running. The approach developed in this paper is a differential method, and therefore may be expected to fail when the motion from frame-to-frame is very large. We propose to augment the technique by the use of an initial coarse search stage. Given a close enough starting value, the differential method will converge correctly.

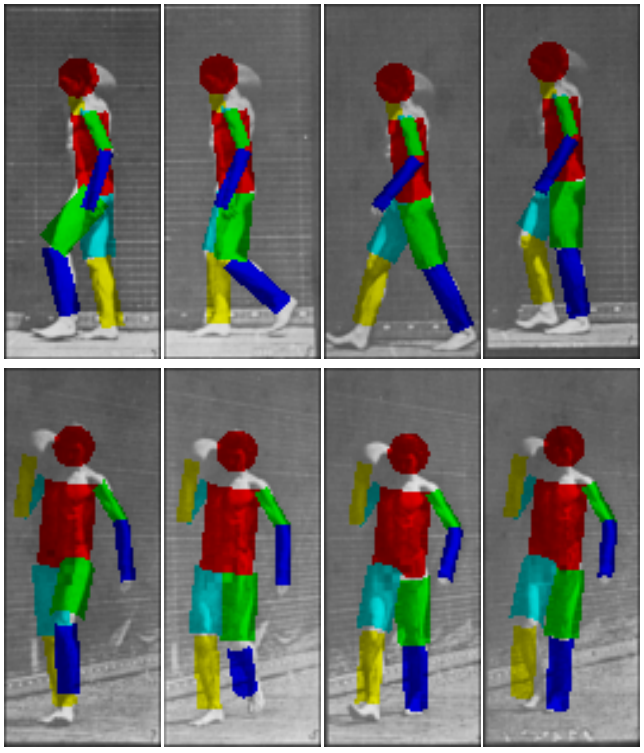


Figure 9: Muybridge's Man Walking: Motion Capture results. This shows the tracked angular configurations and its volumetric model projected to 2 example views.

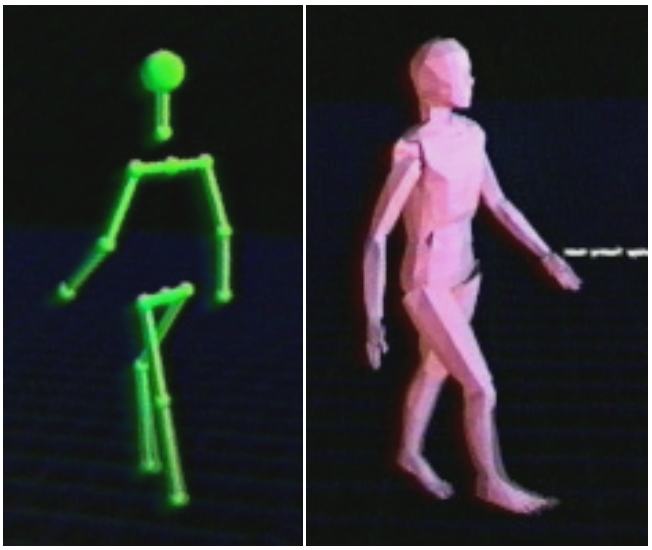


Figure 10: Computer models used for the animation of the Muybridge motion capture. Please check out the video to see the quality of the animation.

Acknowledgements

We would like to thank Charles Ying for creating the Open-GL animations and video editing, Shankar Sastry, Lara Crawford, Jerry Feldman, John Canny, and Jianbo Shi for fruitful discussions, Chad Carson for helping to write this document, and Interval Research Corp, and the California State MICRO program for supporting this research.

References

[1] S. Basu, I.A. Essa, and A.P. Pentland. Motion regularization for model-based head tracking. In *International Conference on Pattern Recognition*, 1996.

[2] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *ECCV*, pages 237–252, 1992.

[3] M.J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, Jan 1996.

[4] G. Cameron, A. Bustanoby, K. Cope, S. Greenberg, C. Hayes, and O. Ozoux. Panel on motion capture and cg character animation. *SIGGRAPH 97*, pages 442–445, 1997.

[5] L. Concalves, E.D. Bernardo, E. Ursella, and P. Perona. Monocular tracking of the human arm in 3d. In *Proc. Int. Conf. Computer Vision*, 1995.

[6] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39, 1977.

[7] D.M. Gavrila and L.S. Davis. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *Proc. of the Int. Workshop on Automatic Face- and Gesture-Recognition, Zurich, 1995*, 1995.

[8] D. Hogg. A program to see a walking person. *Image Vision Computing*, 5(20), 1983.

[9] A. Jepson and M.J. Black. Mixture models for optical flow computation. In *Proc. IEEE Conf. Computer VIsion Pattern Recognition*, pages 760–761, New York, 1993.

[10] S.X. Ju, M.J.Black, and Y.Yacoob. Cardboard people: A parameterized model of articulated motion. In *2nd Int. Conf. on Automatic Face- and Gesture-Recognition, Killington, Vermont*, pages 38–44, 1996.

[11] I.A. Kakadiaris and D. Metaxas. Model-based estimation of 3d human motion with occlusion based on active multi-viewpoint selection. In *CVPR*, 1996.

[12] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proc. 7th Int. Joinnt Conf. on Art. Intell.*, 1981.

[13] M.P. Murray, A.B. Drought, and R.C. Kory. Walking patterns of normal men. *Journal of Bone and Joint Surgery*, 46-A(2):335–360, March 1964.

[14] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, 1994.

[15] Eadweard Muybridge. *The Human Figure In Motion.* Various Publishers, latest edition by Dover Publications, 1901.

[16] J. O'Rourke and N. I. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2(6):522–536, November 1980.

[17] J.M. Regh and T. Kanade. Model-based tracking of self-occluding articulated objects. In *Proc. Int. Conf. Computer Vision*, 1995.

[18] K. Rohr. Incremental recognition of pedestrians from image sequences. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, pages 8–13, New York City, June, 1993.

[19] J. Shi and C. Tomasi. Good features to track. In *CVPR*, 1994.

[20] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. In *SPIE Conference on Integration Issues in Large Commercial Media Delivery Systems*, volume 2615, 1995.

Synthesizing Realistic Facial Expressions from Photographs

Frédéric Pighin *Jamie Hecker* *Dani Lischinski*[†] *Richard Szeliski*[‡] *David H. Salesin*

University of Washington [†]The Hebrew University [‡]Microsoft Research

Abstract

We present new techniques for creating photorealistic textured 3D facial models from photographs of a human subject, and for creating smooth transitions between different facial expressions by morphing between these different models. Starting from several uncalibrated views of a human subject, we employ a user-assisted technique to recover the camera poses corresponding to the views as well as the 3D coordinates of a sparse set of chosen locations on the subject's face. A scattered data interpolation technique is then used to deform a generic face mesh to fit the particular geometry of the subject's face. Having recovered the camera poses and the facial geometry, we extract from the input images one or more texture maps for the model. This process is repeated for several facial expressions of a particular subject. To generate transitions between these facial expressions we use 3D shape morphing between the corresponding face models, while at the same time blending the corresponding textures. Using our technique, we have been able to generate highly realistic face models and natural looking animations.

CR Categories: I.2.10 [Artificial Intelligence]: Vision and Scene Understanding — Modeling and recovery of physical attributes; I.3.7 [Computer Graphics]: Three-Dimensional Graphics — Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics — Color, shading, shadowing and texture.

Additional Keywords: facial modeling, facial expression generation, facial animation, photogrammetry, morphing, view-dependent texture-mapping

1 Introduction

> *There is no landscape that we know as well as the human face. The twenty-five-odd square inches containing the features is the most intimately scrutinized piece of territory in existence, examined constantly, and carefully, with far more than an intellectual interest. Every detail of the nose, eyes, and mouth, every regularity in proportion, every variation from one individual to the next, are matters about which we are all authorities.*
>
> — Gary Faigin [14],
> from *The Artist's Complete Guide to Facial Expression*

Realistic facial synthesis is one of the most fundamental problems in computer graphics — and one of the most difficult. Indeed, attempts to model and animate realistic human faces date back to the early 70's [34], with many dozens of research papers published since. The applications of facial animation include such diverse fields as character animation for films and advertising, computer games [19], video teleconferencing [7], user-interface agents and avatars [44], and facial surgery planning [23, 45]. Yet no perfectly realistic facial animation has ever been generated by computer: no "facial animation Turing test" has ever been passed.

There are several factors that make realistic facial animation so elusive. First, the human face is an extremely complex geometric form. For example, the human face models used in Pixar's *Toy Story* had several thousand control points each [10]. Moreover, the face exhibits countless tiny creases and wrinkles, as well as subtle variations in color and texture — all of which are crucial for our comprehension and appreciation of facial expressions. As difficult as the face is to model, it is even more problematic to animate, since facial movement is a product of the underlying skeletal and muscular forms, as well as the mechanical properties of the skin and subcutaneous layers (which vary in thickness and composition in different parts of the face). All of these problems are enormously magnified by the fact that we as humans have an uncanny ability to read expressions — an ability that is not merely a learned skill, but part of our deep-rooted instincts. For facial expressions, the slightest deviation from truth is something any person will immediately detect.

A number of approaches have been developed to model and animate realistic facial expressions in three dimensions. (The reader is referred to the recent book by Parke and Waters [36] for an excellent survey of this entire field.) Parke's pioneering work introduced simple geometric interpolation between face models that were digitized by hand [34]. A radically different approach is performance-based animation, in which measurements from real actors are used to drive synthetic characters [4, 13, 47]. Today, face models can also be obtained using laser-based cylindrical scanners, such as those produced by Cyberware [8]. The resulting range and color data can be fitted with a structured face mesh, augmented with a physically-based model of skin and muscles [29, 30, 43, 46]. The animations produced using these face models represent the state-of-the-art in automatic physically-based facial animation.

For sheer photorealism, one of the most effective approaches to date has been the use of 2D morphing between photographic images [3]. Indeed, some remarkable results have been achieved in this way — most notably, perhaps, the Michael Jackson video produced by PDI, in which very different-looking actors are seemingly transformed into one another as they dance. The production of this video, however, required animators to painstakingly specify a few dozen carefully chosen correspondences between physical features of the actors in almost every frame. Another problem with 2D image morphing is that it does not correctly account for changes in viewpoint or object pose. Although this shortcoming has been recently addressed by a technique called "view morphing" [39], 2D morphing still lacks some of the advantages of a 3D model, such as the complete freedom of viewpoint and the ability to composite the image with other 3D graphics. Morphing has also been applied in 3D: Chen *et al.* [6] applied Beier and Neely's 2D morphing technique [3] to morph between cylindrical laser scans of human heads. Still, even in this case the animator must specify correspondences for every pair of expressions in order to produce a transition between them. More recently,

Bregler *et al.* [5] used morphing of mouth regions to lip-synch existing video to a novel sound-track.

In this paper, we show how 2D morphing techniques can be combined with 3D transformations of a geometric model to automatically produce 3D facial expressions with a high degree of realism. Our process consists of several basic steps. First, we capture multiple views of a human subject (with a given facial expression) using cameras at arbitrary locations. Next, we digitize these photographs and manually mark a small set of initial corresponding points on the face in the different views (typically, corners of the eyes and mouth, tip of the nose, etc.). These points are then used to automatically recover the camera parameters (position, focal length, etc.) corresponding to each photograph, as well as the 3D positions of the marked points in space. The 3D positions are then used to deform a generic 3D face mesh to fit the face of the particular human subject. At this stage, additional corresponding points may be marked to refine the fit. Finally, we extract one or more texture maps for the 3D model from the photos. Either a single view-independent texture map can be extracted, or the original images can be used to perform view-dependent texture mapping. This whole process is repeated for the same human subject, with several different facial expressions. To produce facial animations, we interpolate between two or more different 3D models constructed in this way, while at the same time blending the textures. Since all the 3D models are constructed from the same generic mesh, there is a natural correspondence between all geometric points for performing the morph. Thus, transitions between expressions can be produced entirely automatically once the different face models have been constructed, without having to specify pairwise correspondences between any of the expressions.

Our modeling approach is based on photogrammetric techniques in which images are used to create precise geometry [31, 40]. The earliest such techniques applied to facial modeling and animation employed grids that were drawn directly on the human subject's face [34, 35]. One consequence of these grids, however, is that the images used to construct geometry can no longer be used as valid texture maps for the subject. More recently, several methods have been proposed for modeling the face photogrammetrically without the use of grids [20, 24]. These modeling methods are similar in concept to the modeling technique described in this paper. However, these previous techniques use a small predetermined set of features to deform the generic face mesh to the particular face being modeled, and offer no mechanism to further improve the fit. Such an approach may perform poorly on faces with unusual features or other significant deviations from the normal. Our system, by contrast, gives the user complete freedom in specifying the correspondences, and enables the user to refine the initial fit as needed. Another advantage of our technique is its ability to handle fairly arbitrary camera positions and lenses, rather than using a fixed pair that are precisely oriented. Our method is similar, in concept, to the work done in architectural modeling by Debevec *et al.* [9], where a set of annotated photographs are used to model buildings starting from a rough description of their shape. Compared to facial modeling methods that utilize a laser scanner, our technique uses simpler acquisition equipment (regular cameras), and it is capable of extracting texture maps of higher resolution. (Cyberware scans typically produce a cylindrical grid of 512 by 256 samples). The price we pay for these advantages is the need for user intervention in the modeling process.

We employ our system not only for creating realistic face models, but also for performing realistic transitions between different expressions. One advantage of our technique, compared to more traditional animatable models with a single texture map, is that we can capture the subtle changes in illumination and appearance (e.g., facial creases) that occur as the face is deformed. This degree of realism is difficult to achieve even with physically-based models, because of the complexity of skin folding and the difficulty of simulating interreflections and self-shadowing [18, 21, 32].

This paper also presents several new expression synthesis techniques based on extensions to the idea of morphing. We develop a morphing technique that allows for different regions of the face to have different "percentages" or "mixing proportions" of facial expressions. We also introduce a painting interface, which allows users to locally add in a little bit of an expression to an existing composite expression. We believe that these novel methods for expression generation and animation may be more natural for the average user than more traditional animation systems, which rely on the manual adjustments of dozens or hundreds of control parameters.

The rest of this paper is organized as follows. Section 2 describes our method for fitting a generic face mesh to a collection of simultaneous photographs of an individual's head. Section 3 describes our technique for extracting both view-dependent and view-independent texture maps for photorealistic rendering of the face. Section 4 presents the face morphing algorithm that is used to animate the face model. Section 5 describes the key aspects of our system's user interface. Section 6 presents the results of our experiments with the proposed techniques, and Section 7 offers directions for future research.

2 Model fitting

The task of the model-fitting process is to adapt a generic face model to fit an individual's face and facial expression. As input to this process, we take several images of the face from different viewpoints (Figure 1a) and a generic face model (we use the generic face model created with Alias|Wavefront [2] shown in Figure 1c). A few features points are chosen (13 in this case, shown in the frames of Figure 1a) to recover the camera pose. These same points are also used to refine the generic face model (Figure 1d). The model can be further refined by drawing corresponding curves in the different views (Figure 1b). The output of the process is a face model that has been adapted to fit the face in the input images (Figure 1e), along with a precise estimate of the camera pose corresponding to each input image.

The model-fitting process consists of three stages. In the *pose recovery* stage, we apply computer vision techniques to estimate the viewing parameters (position, orientation, and focal length) for each of the input cameras. We simultaneously recover the 3D coordinates of a set of *feature points* on the face. These feature points are selected interactively from among the face mesh vertices, and their positions in each image (where visible) are specified by hand. The *scattered data interpolation* stage uses the estimated 3D coordinates of the feature points to compute the positions of the remaining face mesh vertices. In the *shape refinement* stage, we specify additional correspondences between facial vertices and image coordinates to improve the estimated shape of the face (while keeping the camera pose fixed).

2.1 Pose recovery

Starting with a rough knowledge of the camera positions (e.g., frontal view, side view, etc.) and of the 3D shape (given by the generic head model), we iteratively improve the pose and the 3D shape estimates in order to minimize the difference between the predicted and observed feature point positions. Our formulation is based on the non-linear least squares structure-from-motion algorithm introduced by Szeliski and Kang [41]. However, unlike the method they describe, which uses the Levenberg-Marquardt algorithm to perform a complete iterative minimization over all of the unknowns simultaneously, we break the problem down into a series of linear least squares problems that can be solved using very simple

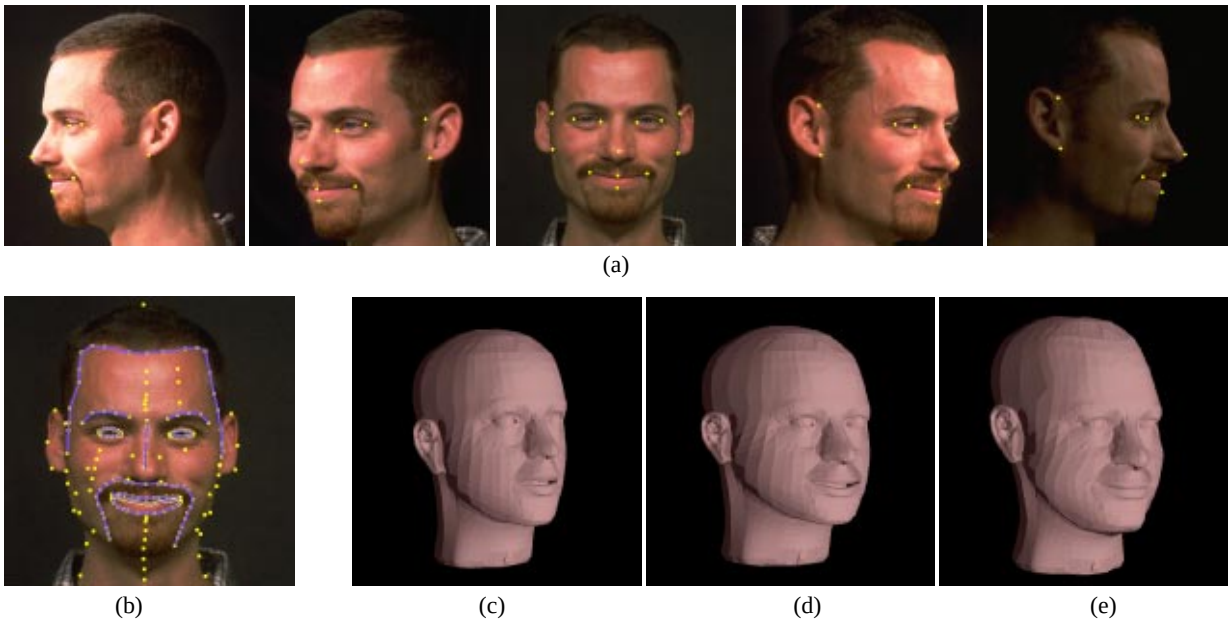


Figure 1 Model-fitting process: (a) a set of input images with marked feature points, (b) facial features annotated using a set of curves, (c) generic face geometry (shaded surface rendering), (d) face adapted to initial 13 feature points (after pose estimation) (e) face after 99 additional correspondences have been given.

and numerically stable techniques [16, 37].

To formulate the pose recovery problem, we associate a rotation matrix $\boldsymbol{R}^k$ and a translation vector $\boldsymbol{t}^k$ with each camera pose k. (The three rows of $\boldsymbol{R}^k$ are $\boldsymbol{r}_x^k$, $\boldsymbol{r}_y^k$, and $\boldsymbol{r}_z^k$, and the three entries in $\boldsymbol{t}^k$ are t_x^k, t_y^k, t_z^k.) We write each 3D feature point as $\boldsymbol{p}_i$, and its 2D screen coordinates in the k-th image as (x_i^k, y_i^k).

Assuming that the origin of the (x, y) image coordinate system lies at the optical center of each image (i.e., where the optical axis intersects the image plane), the traditional 3D projection equation for a camera with a focal length f^k (expressed in pixels) can be written as

$$x_i^k = f^k \frac{\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k}{\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i + t_z^k} \qquad y_i^k = f^k \frac{\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k}{\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i + t_z^k} \tag{1}$$

(This is just an explicit rewriting of the traditional projection equation $\boldsymbol{x}_i^k \propto \boldsymbol{R}^k \boldsymbol{p}_i + \boldsymbol{t}^k$ where $\boldsymbol{x}_i^k = (x_i^k, y_i^k, f^k)$.)

Instead of using (1) directly, we reformulate the problem to estimate inverse distances to the object [41]. Let $\eta^k = 1/t_z^k$ be this inverse distance and $s^k = f^k \eta^k$ be a world-to-image scale factor. The advantage of this formulation is that the scale factor s^k can be reliably estimated even when the focal length is long, whereas the original formulation has a strong coupling between the f^k and t_z^k parameters.

Performing these substitution, we obtain

$$x_i^k = s^k \frac{\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k}{1 + \eta^k \boldsymbol{r}_z^k \cdot \boldsymbol{p}_i}$$
$$y_i^k = s^k \frac{\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k}{1 + \eta^k \boldsymbol{r}_z^k \cdot \boldsymbol{p}_i}.$$

If we let $w_i^k = (1 + \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i))^{-1}$ be the inverse denominator, and collect terms on the left-hand side, we get

$$w_i^k \left(x_i^k + x_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k) \right) = 0 \tag{2}$$
$$w_i^k \left(y_i^k + y_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) - s^k (\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k) \right) = 0$$

Note that these equations are linear in each of the unknowns that we wish to recover, i.e., $\boldsymbol{p}_i$, t_x^k, t_y^k, η^k, s^k, and $\boldsymbol{R}^k$, if we ignore the variation of w_i^k with respect to these parameters. (The reason we keep the w_i^k term, rather than just dropping it from these equations, is so that the linear equations being solved in the least squares step have the same magnitude as the original measurements (x_i^k, y_i^k). Hence, least-squares will produce a *maximum likelihood* estimate for the unknown parameters [26].)

Given estimates for initial values, we can solve for different subsets of the unknowns. In our current algorithm, we solve for the unknowns in five steps: first s^k, then $\boldsymbol{p}_i$, $\boldsymbol{R}^k$, t_x^k and t_y^k, and finally η^k. This order is chosen to provide maximum numerical stability given the crude initial pose and shape estimates. For each parameter or set of parameters chosen, we solve for the unknowns using linear least squares (Appendix A). The simplicity of this approach is a result of solving for the unknowns in five separate stages, so that the parameters for a given camera or 3D point can be recovered independently of the other parameters.

2.2 Scattered data interpolation

Once we have computed an initial set of coordinates for the feature points $\boldsymbol{p}_i$, we use these values to deform the remaining vertices on the face mesh. We construct a smooth interpolation function that gives the 3D displacements between the original point positions and the new adapted positions for every vertex in the original generic face mesh. Constructing such an interpolation function is a standard problem in scattered data interpolation. Given a set of known displacements $\boldsymbol{u}_i = \boldsymbol{p}_i - \boldsymbol{p}_i^{(0)}$ away from the original positions $\boldsymbol{p}_i^{(0)}$ at every constrained vertex i, construct a function that gives the displacement $\boldsymbol{u}_j$ for every unconstrained vertex j.

There are several considerations in choosing the particular data interpolant [33]. The first consideration is the embedding space, that is, the domain of the function being computed. In our case, we use the original 3D coordinates of the points as the domain. (An alternative would be to use some 2D parameterization of the surface mesh, for instance, the cylindrical coordinates described in Section 3.) We therefore attempt to find a smooth vector-valued function $\boldsymbol{f}(\boldsymbol{p})$ fitted

to the known data $\boldsymbol{u}_i = \boldsymbol{f}(\boldsymbol{p}_i)$, from which we can compute $\boldsymbol{u}_j = \boldsymbol{f}(\boldsymbol{p}_j)$.

There are also several choices for how to construct the interpolating function [33]. We use a method based on *radial basis functions*, that is, functions of the form

$$\boldsymbol{f}(\boldsymbol{p}) = \sum_i \boldsymbol{c}_i \phi(\|\boldsymbol{p} - \boldsymbol{p}_i\|),$$

where $\phi(r)$ are radially symmetric basis functions. A more general form of this interpolant also adds some low-order polynomial terms to model global, e.g., affine, deformations [27, 28, 33]. In our system, we use an affine basis as part of our interpolation algorithm, so that our interpolant has the form:

$$\boldsymbol{f}(\boldsymbol{p}) = \sum_i \boldsymbol{c}_i \phi(\|\boldsymbol{p} - \boldsymbol{p}_i\|) + \boldsymbol{M}\boldsymbol{p} + \boldsymbol{t}, \quad (3)$$

To determine the coefficients $\boldsymbol{c}_i$ and the affine components $\boldsymbol{M}$ and $\boldsymbol{t}$, we solve a set of linear equations that includes the interpolation constraints $\boldsymbol{u}_i = \boldsymbol{f}(\boldsymbol{p}_i)$, as well as the constraints $\sum_i \boldsymbol{c}_i = 0$ and $\sum_i \boldsymbol{c}_i \boldsymbol{p}_i^{\mathrm{T}} = 0$, which remove affine contributions from the radial basis functions.

Many different functions for $\phi(r)$ have been proposed [33]. After experimenting with a number of functions, we have chosen to use $\phi(r) = e^{-r/64}$, with units measured in inches.

Figure 1d shows the shape of the face model after having interpolated the set of computed 3D displacements at 13 feature points shown in Figure 1 and applied them to the entire face.

2.3 Correspondence-based shape refinement

After warping the generic face model into its new shape, we can further improve the shape by specifying additional correspondences. Since these correspondences may not be as easy to locate correctly, we do not use them to update the camera pose estimates. Instead, we simply solve for the values of the new feature points $\boldsymbol{p}_i$ using a simple least-squares fit, which corresponds to finding the point nearest the intersection of the viewing rays in 3D. We can then re-run the scattered data interpolation algorithm to update the vertices for which no correspondences are given. This process can be repeated until we are satisfied with the shape.

Figure 1e shows the shape of the face model after 99 additional correspondences have been specified. To facilitate the annotation process, we grouped vertices into polylines. Each polyline corresponds to an easily identifiable facial feature such as the eyebrow, eyelid, lips, chin, or hairline. The features can be annotated by outlining them with hand-drawn curves on each photograph where they are visible. The curves are automatically converted into a set of feature points by stepping along them using an arc-length parametrization. Figure 1b shows annotated facial features using a set of curves on the front view.

3 Texture extraction

In this section we describe the process of extracting the texture maps necessary for rendering photorealistic images of a reconstructed face model from various viewpoints.

The texture extraction problem can be defined as follows. Given a collection of photographs, the recovered viewing parameters, and the fitted face model, compute for each point $\boldsymbol{p}$ on the face model its texture color $T(\boldsymbol{p})$.

Each point $\boldsymbol{p}$ may be visible in one or more photographs; therefore, we must identify the corresponding point in each photograph and decide how these potentially different values should be combined

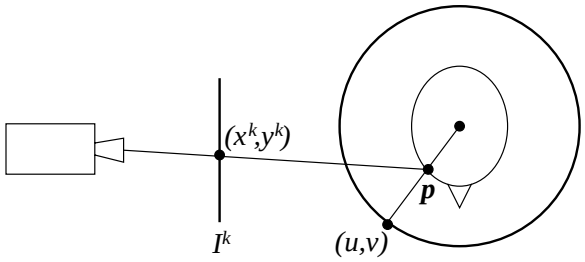


Figure 2 Geometry for texture extraction

(blended) together. There are two principal ways to blend values from different photographs: *view-independent blending*, resulting in a texture map that can be used to render the face from any viewpoint; and *view-dependent blending*, which adjusts the blending weights at each point based on the direction of the current viewpoint [9, 38]. Rendering takes longer with view-dependent blending, but the resulting image is of slightly higher quality (see Figure 3).

3.1 Weight maps

As outlined above, the texture value $T(\boldsymbol{p})$ at each point on the face model can be expressed as a convex combination of the corresponding colors in the photographs:

$$T(\boldsymbol{p}) = \frac{\sum_k m^k(\boldsymbol{p})\, I^k(x^k, y^k)}{\sum_k m^k(\boldsymbol{p})}.$$

Here, I^k is the image function (color at each pixel of the k-th photograph,) and (x^k, y^k) are the image coordinates of the projection of $\boldsymbol{p}$ onto the k-th image plane. The *weight map* $m^k(\boldsymbol{p})$ is a function that specifies the contribution of the k-th photograph to the texture at each facial surface point.

The construction of these weight maps is probably the trickiest and the most interesting component of our texture extraction technique. There are several important considerations that must be taken into account when defining a weight map:

1. *Self-occlusion:* $m^k(\boldsymbol{p})$ should be zero unless $\boldsymbol{p}$ is front-facing with respect to the k-th image and visible in it.
2. *Smoothness:* the weight map should vary smoothly, in order to ensure a seamless blend between different input images.
3. *Positional certainty:* $m^k(\boldsymbol{p})$ should depend on the "positional certainty" [24] of $\boldsymbol{p}$ with respect to the k-th image. The positional certainty is defined as the dot product between the surface normal at $\boldsymbol{p}$ and the k-th direction of projection.
4. *View similarity:* for view-dependent texture mapping, the weight $m^k(\boldsymbol{p})$ should also depend on the angle between the direction of projection of $\boldsymbol{p}$ onto the j-th image and its direction of projection in the new view.

Previous authors have taken only a subset of these considerations into account when designing their weighting functions. For example, Kurihara and Arai [24] use positional certainty as their weighting function, but they do not account for self-occlusion. Akimoto *et al.* [1] and Ip and Yin [20] blend the images smoothly, but address neither self-occlusion nor positional certainty. Debevec *et al.* [9], who describe a view-dependent texture mapping technique for modeling and rendering buildings from photographs, do address occlusion but do not account for positional certainty. (It should be noted, however, that positional certainty is less critical in photographs of buildings, since most buildings do not tend to curve away from the camera.)

To facilitate fast visibility testing of points on the surface of the face from a particular camera pose, we first render the face model using the recovered viewing parameters and save the resulting depth map from the Z-buffer. Then, with the aid of this depth map, we can quickly classify the visibility of each facial point by applying the viewing transformation and comparing the resulting depth to the corresponding value in the depth map.

3.2 View-independent texture mapping

In order to support rapid display of the textured face model from any viewpoint, it is desirable to blend the individual photographs together into a single texture map. This texture map is constructed on a virtual cylinder enclosing the face model. The mapping between the 3D coordinates on the face mesh and the 2D texture space is defined using a cylindrical projection, as in several previous papers [6, 24, 29].

For view-independent texture mapping, we will index the weight map m^k by the (u, v) coordinates of the texture being created. Each weight $m^k(u, v)$ is determined by the following steps:

1. Construct a feathered visibility map F^k for each image k. These maps are defined in the same cylindrical coordinates as the texture map. We initially set $F^k(u, v)$ to 1 if the corresponding facial point $\boldsymbol{p}$ is visible in the k-th image, and to 0 otherwise. The result is a binary visibility map, which is then smoothly ramped (feathered) from 1 to 0 in the vicinity of the boundaries [42]. A cubic polynomial is used as the ramping function.
2. Compute the 3D point $\boldsymbol{p}$ on the surface of the face mesh whose cylindrical projection is (u, v) (see Figure 2). This computation is performed by casting a ray from (u, v) on the cylinder towards the cylinder's axis. The first intersection between this ray and the face mesh is the point $\boldsymbol{p}$. (Note that there can be more than one intersection for certain regions of the face, most notably the ears. These special cases are discussed in Section 3.4.) Let $P^k(\boldsymbol{p})$ be the positional certainty of $\boldsymbol{p}$ with respect to the k-th image.
3. Set weight $m^k(u, v)$ to the product $F^k(u, v)\, P^k(\boldsymbol{p})$.

For view-independent texture mapping, we will compute each pixel of the resulting texture $T(u, v)$ as a weighted sum of the original image functions, indexed by (u, v).

3.3 View-dependent texture mapping

The main disadvantage of the view-independent cylindrical texture map described above is that its construction involves blending together resampled versions of the original images of the face. Because of this resampling, and also because of slight registration errors, the resulting texture is slightly blurry. This problem can be alleviated to a large degree by using a view-dependent texture map [9] in which the blending weights are adjusted dynamically, according to the current view.

For view-dependent texture mapping, we render the model several times, each time using a different input photograph as a texture map, and blend the results. More specifically, for each input photograph, we associate texture coordinates and a blending weight with each vertex in the face mesh. (The rendering hardware performs perspective-correct texture mapping along with linear interpolation of the blending weights.)

Given a viewing direction $\boldsymbol{d}$, we first select the subset of photographs used for the rendering and then assign blending weights to each of these photographs. Pulli *et al.* [38] select three photographs based on a Delaunay triangulation of a sphere surrounding the object. Since our cameras were positioned roughly in the same plane, we select just the two photographs whose view directions $\boldsymbol{d}^\ell$ and $\boldsymbol{d}^{\ell+1}$ are the closest to $\boldsymbol{d}$ and blend between the two.

Figure 3 Comparison between view-independent (left) and view-dependent (right) texture mapping. Higher frequency details are visible in the view-dependent rendering.

In choosing the view-dependent term $V^k(\boldsymbol{d})$ of the blending weights, we wish to use just a single photo if that photo's view direction matches the current view direction precisely, and to blend smoothly between the nearest two photos otherwise. We used the simplest possible blending function having this effect:

$$V^k(\boldsymbol{d}) = \begin{cases} \boldsymbol{d} \cdot \boldsymbol{d}^k - \boldsymbol{d}^\ell \cdot \boldsymbol{d}^{\ell+1} & \text{if } \ell \le k \le \ell + 1 \\ 0 & \text{otherwise} \end{cases}$$

For the final blending weights $m^k(\boldsymbol{p}, \boldsymbol{d})$, we then use the product of all three terms $F^k(x^k, y^k)\, P^k(\boldsymbol{p})\, V^k(\boldsymbol{d})$.

View-dependent texture maps have several advantages over cylindrical texture maps. First, they can make up for some lack of detail in the model. Second, whenever the model projects onto a cylinder with overlap, a cylindrical texture map will not contain data for some parts of the model. This problem does not arise with view-dependent texture maps if the geometry of the mesh matches the photograph properly. One disadvantage of the view-dependent approach is its higher memory requirements and slower speed due to the multi-pass rendering. Another drawback is that the resulting images are much more sensitive to any variations in exposure or lighting conditions in the original photographs.

3.4 Eyes, teeth, ears, and hair

The parts of the mesh that correspond to the eyes, teeth, ears, and hair are textured in a separate process. The eyes and teeth are usually partially occluded by the face; hence it is difficult to extract a texture map for these parts in every facial expression. The ears have an intricate geometry with many folds and usually fail to project without overlap on a cylinder. The hair has fine-detailed texture that is difficult to register properly across facial expressions. For these reasons, each of these facial elements is assigned an individual texture map. The texture maps for the eyes, teeth, and ears are computed by projecting the corresponding mesh part onto a selected input image where that part is clearly visible (the front view for eyes and teeth, side views for ears).

The eyes and the teeth are usually partially shadowed by the eyelids and the mouth respectively. We approximate this shadowing by modulating the brightness of the eye and teeth texture maps according to the size of the eyelid and mouth openings.

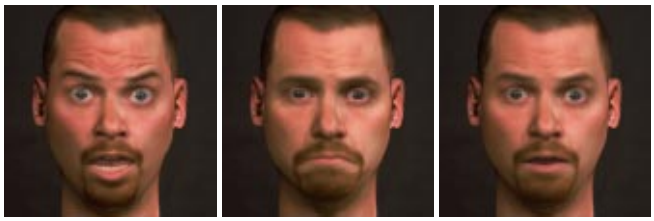


Figure 4 A global blend between "surprised" (left) and "sad" (center) produces a "worried" expression (right).

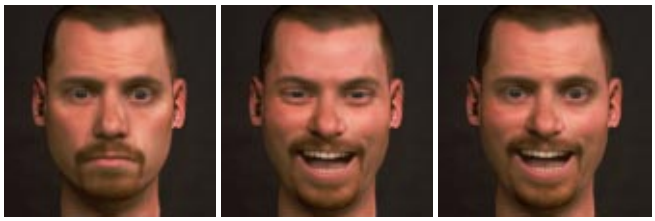


Figure 5 Combining the upper part of a "neutral" expression (left) with the lower part of a "happy" expression (center) produces a "fake smile" (right).

4 Expression morphing

A major goal of this work is the generation of continuous and realistic transitions between different facial expressions. We achieve these effects by morphing between corresponding face models.

In general the problem of morphing between arbitrary polygonal meshes is a difficult one [22], since it requires a set of correspondences between meshes with potentially different topology that can produce a reasonable set of intermediate shapes. In our case, however, the topology of all the face meshes is identical. Thus, there is already a natural correspondence between vertices. Furthermore, in creating the models we attempt to mark facial features consistently across different facial expressions, so that the major facial features correspond to the same vertices in all expressions. In this case, a satisfactory 3D morphing sequence can be obtained using simple linear interpolation between the geometric coordinates of corresponding vertices in each of the two face meshes.

Together with the geometric interpolation, we need to blend the associated textures. Again, in general, morphing between two images requires pairwise correspondences between images features [3]. In our case, however, correspondences between the two textures are implicit in the texture coordinates of the two associated face meshes. Rather than warping the two textures to form an intermediate one, the intermediate face model (obtained by geometric interpolation) is rendered once with the first texture, and again with the second. The two resulting images are then blended together. This approach is faster than warping the textures (which typically have high resolution), and it avoids the resampling that is typically performed during warping.

4.1 Multiway blend and localized blend

Given a set of facial expression meshes, we have explored ways to enlarge this set by combining expressions. The simplest approach is to use the morphing technique described above to create new facial expressions, which can be added to the set. This idea can be generalized to an arbitrary number of starting expressions by taking convex combinations of them all, using weights that apply both to the coordinates of the mesh vertices and to the values in the texture map. (Extrapolation of expressions should also be possible by allowing weights to have values outside of the interval $[0, 1]$; note, however, that such weights might result in colors outside of the allowable gamut.)

We can generate an even wider range of expressions using a localized blend of the facial expressions. Such a blend is specified by a set of blend functions, one for each expression, defined over the vertices of the mesh. These blend functions describe the contribution of a given expression at a particular vertex.

Although it would be possible to compute a texture map for each new expression, doing so would result in a loss of texture quality. Instead, the weights for each new blended expression are always factored into weights over the vertices of the original set of expressions. Thus, each blended expression is rendered using the texture map of an original expression, along with weights at each vertex, which control the opacity of that texture. The opacities are linearly interpolated over the face mesh using Gouraud shading.

4.2 Blend specification

In order to design new facial expressions easily, the user must be provided with useful tools for specifying the blending functions. These tools should satisfy several requirements. First, it should be possible to edit the blend at different resolutions. Moreover, we would like the specification process to be continuous so that small changes in the blend parameters do not trigger radical changes in the resulting expression. Finally, the tools should be intuitive to the user; it should be easy to produce a particular target facial expression from an existing set.

We explored several different ways of specifying the blending weights:

- *Global blend.* The blending weights are constant over all vertices. A set of sliders controls the mixing proportions of the contributing expressions. Figure 4 shows two facial expressions blended in equal proportions to produce a halfway blend.
- *Regional blend.* According to studies in psychology, the face can be split into several regions that behave as coherent units [11]. Usually, three regions are considered: one for the forehead (including the eyebrows), another for the eyes, and another for the lower part of the face. Further splitting the face vertically down the center results in six regions and allows for asymmetric expressions. We similarly partition the face mesh into several (softly feathered) regions and assign weights so that vertices belonging to the same region have the same weights. The mixing proportions describing a selected region can be adjusted by manipulating a set of sliders. Figure 5 illustrates the blend of two facial expressions with two regions: the upper part of the face (including eyes and forehead) and the lower part (including nose, mouth, and chin.)
- *Painterly interface.* The blending weights can be assigned to the vertices using a 3D painting tool. This tool uses a palette in which the "colors" are facial expressions (both geometry and color), and the "opacity" of the brush controls how much the expression contributes to the result. Once an expression is selected, a 3D brush can be used to modify the blending weights in selected areas of the mesh. The fraction painted has a gradual drop-off and is controlled by the opacity of the brush. The strokes are applied directly on the rendering of the current facial blend, which is updated in real-time. To improve the rendering speed, only the portion of the mesh that is being painted is re-rendered. Figure 7 illustrates the design of a debauched smile: starting with a neutral expression, the face is locally modified using three other expressions. Note that in the last step, the use of a partially transparent brush with the "sleepy" expression results in the actual geometry of the eyelids becoming partially lowered.

Figure 6 Animation interface. On the left is the "expression gallery"; on the right an expression is being designed. At the bottom expressions and poses are scheduled on the timeline.

Combining different original expressions enlarges the repertoire of expressions obtained from a set of photographs. The expressions in this repertoire can themselves be blended to create even more expressions, with the resulting expression still being representable as a (locally varying) linear combination of the original expressions.

5 User interface

We designed an interactive tool to fit a 3D face mesh to a set of images. This tool allows a user to select vertices on the mesh and mark where these curves or vertices should project on the images. After a first expression has been modeled, the set of annotations can be used as an initial guess for subsequent expressions. These guesses are automatically refined using standard correlation-based search. Any resulting errors can be fixed up by hand. The extraction of the texture map does not require user intervention, but is included in the interface to provide feedback during the modeling phase.

We also designed a keyframe animation system to generate facial animations. Our animation system permits a user to blend facial expressions and to control the transitions between these different expressions (Figure 6). The expression gallery is a key component of our system; it is used to select and display (as thumbnails) the set of facial expressions currently available. The thumbnails can be dragged and dropped onto the timeline (to set keyframes) or onto the facial design interface (to select or add facial expressions). The timeline is used to schedule the different expression blends and the changes in viewing parameters (pose) during the animation. The blends and poses have two distinct types of keyframes. Both types of keyframes are linearly interpolated with user-controlled cubic Bézier curves. The timeline can also be used to display intermediate frames at low resolution to provide a quick feedback to the animator. A second timeline can be displayed next to the composition timeline. This feature is helpful for correctly synchronizing an animation with live video or a soundtrack. The eyes are animated separately from the rest of the face, with the gaze direction parameterized by two Euler angles.

6 Results

In order to test our technique, we photographed both a man (J. R.) and a woman (Karla) in a variety of facial expressions. The photog-

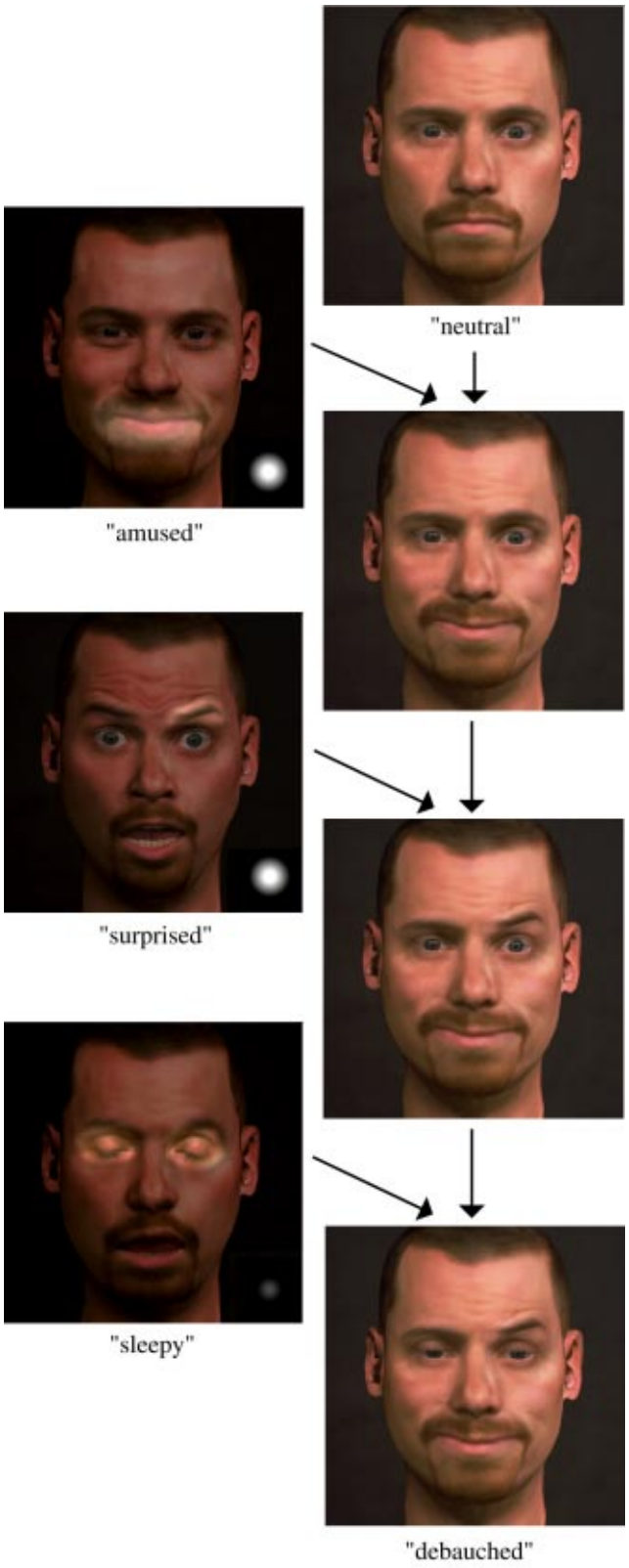


Figure 7 Painterly interface: design of a debauched smile. The right column shows the different stages of the design; the left column shows the portions of the original expressions used in creating the final expression. The "soft brush" used is shown at the bottom-right corner of each contributing expression.

raphy was performed using five cameras simultaneously. The cameras were not calibrated in any particular way, and the lenses had different focal lengths. Since no special attempt was made to illuminate the subject uniformly, the resulting photographs exhibited considerable variation in both hue and brightness. The photographs were digitized using the Kodak PhotoCD process. Five typical images (cropped to the size of the subject's head) are shown in Figure 1a.

We used the interactive modeling system described in Sections 2 and 3 to create the same set of eight face models for each subject: "happy," "amused," "angry," "surprised," "sad," "sleepy," "pained," and "neutral."

Following the modeling stage, we generated a facial animation for each of the individuals starting from the eight original expressions. We first created an animation for J. R. We then applied the very same morphs specified by this animation to the models created for Karla. For most frames of the animation, the resulting expressions were quite realistic. Figure 8 shows five frames from the animation sequence for J. R. and the purely automatically generated frames in the corresponding animation for Karla. With just a small amount of additional retouching (using the blending tools described in Section 4.2), this derivative animation can be made to look as good as the original animation for J. R.

7 Future work

The work described in this paper is just the first step towards building a complete image-based facial modeling and animation system. There are many ways to further enhance and extend the techniques that we have described:

Color correction. For better color consistency in facial textures extracted from photographs, color correction should be applied to simultaneous photographs of each expression.

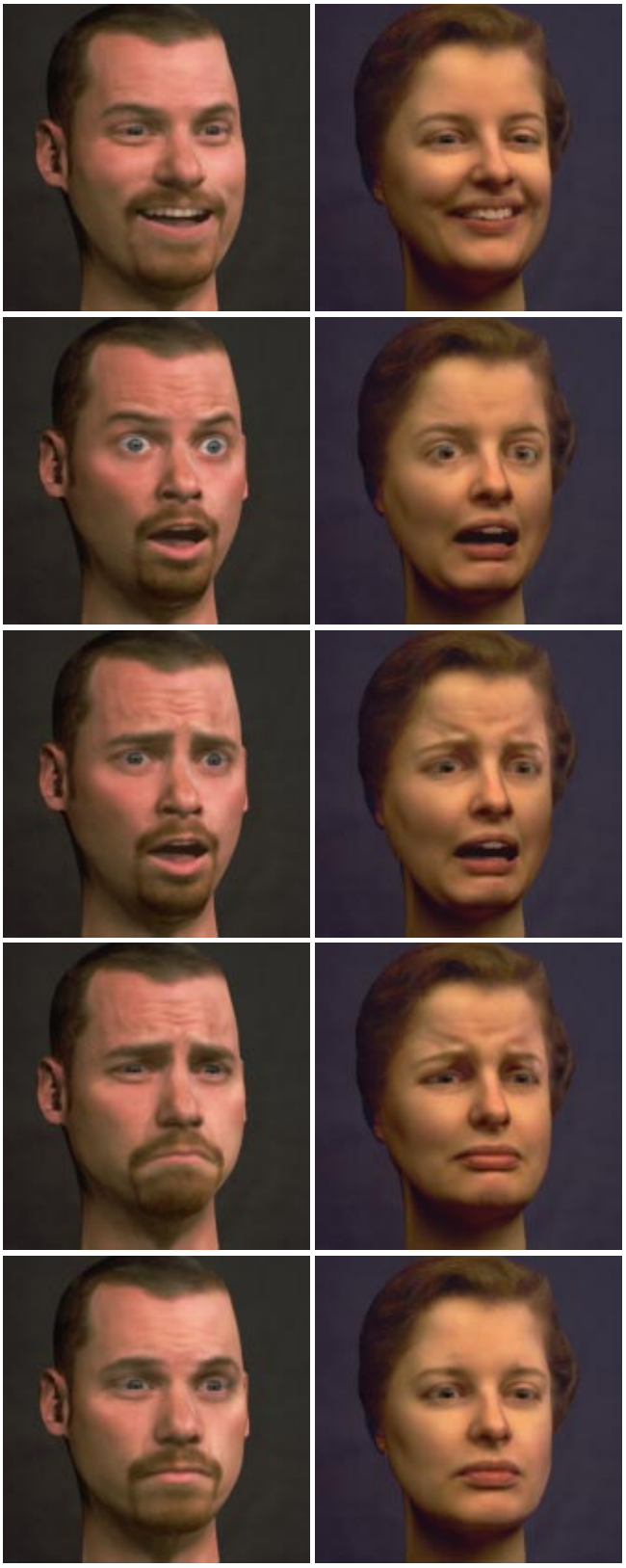
Improved registration. Some residual ghosting or blurring artifacts may occasionally be visible in the cylindrical texture map due to small misregistrations between the images, which can occur if geometry is imperfectly modeled or not detailed enough. To improve the quality of the composite textures, we could locally warp each component texture (and weight) map before blending [42].

Texture relighting. Currently, extracted textures reflect the lighting conditions under which the photographs were taken. Relighting techniques should be developed for seamless integration of our face models with other elements.

Automatic modeling. Our ultimate goal, as far as the facial modeling part is concerned, is to construct a fully automated modeling system, which would automatically find features and correspondences with minimal user intervention. This is a challenging problem indeed, but recent results on 2D face modeling in computer vision [25] give us cause for hope.

Modeling from video. We would like to be able to create face models from video or old movie footage. For this purpose, we would have to improve the robustness of our techniques in order to synthesize face meshes and texture maps from images that do not correspond to different views of the same expression. Adding anthropomorphic constraints to our face model might make up for the lack of coherence in the data [48].

Complex animations. In order to create complex animations, we must extend our vocabulary for describing facial movements beyond blending between different expressions. There are several potential ways to attack this problem. One would be to adopt an action-unit-based system such as the Facial Action Coding System



(a) (b)

Figure 8 On the left are frames from an original animation, which we created for J. R. The morphs specified in these frames were then directly used to create a derivative animation for Karla, shown on the right.

(FACS) [12]. Another possibility would be to apply modal analysis (principal component analysis) techniques to describe facial expression changes using a small number of motions [25]. Finding natural control parameters to facilitate animation and developing realistic-looking temporal profiles for such movements are also challenging research problems.

Lip-synching. Generating speech animation with our keyframe animation system would require a large number of keyframes. However, we could use a technique similar to that of Bregler *et al*. [5] to automatically lip-synch an animation to a sound-track. This would require the synthesis of face models for a wide range of visemes. For example, such database of models could be constructed using video footage to reconstruct face models automatically [17].

Performance-driven animation. Ultimately, we would also like to support performance-driven animation, i.e., the ability to automatically track facial movements in a video sequence, and to automatically translate these into animation control parameters. Our current techniques for registering images and converting them into 3D movements should provide a good start, although they will probably need to be enhanced with feature-tracking techniques and some rudimentary expression-recognition capabilities. Such a system would enable not only very realistic facial animation, but also a new level of video coding and compression techniques (since only the expression parameters would need to be encoded), as well as real-time control of avatars in 3D chat systems.

8 Acknowledgments

We would like to thank Katrin Petersen and Andrew Petty for modeling the generic face model, Cassidy Curtis for his invaluable advice on animating faces, and Joel Auslander and Jason Griffith for early contributions to this project. This work was supported by an NSF Presidential Faculty Fellow award (CCR-9553199), an ONR Young Investigator award (N00014-95-1-0728), and industrial gifts from Microsoft and Pixar.

References

[1] Takaaki Akimoto, Yasuhito Suenaga, and Richard S. Wallace. Automatic Creation of 3D Facial Models. *IEEE Computer Graphics and Applications*, 13(5):16–22, September 1993.

[2] Alias | Wavefront, Toronto, Ontario. *Alias V7.0*, 1995.

[3] Thaddeus Beier and Shawn Neely. Feature-based Image Metamorphosis. In *SIGGRAPH 92 Conference Proceedings*, pages 35–42. ACM SIGGRAPH, July 1992.

[4] Philippe Bergeron and Pierre Lachapelle. Controlling Facial Expressions and Body Movements in the Computer-Generated Animated Short "Tony De Peltrie". In *SIGGRAPH 85 Advanced Computer Animation seminar notes*. July 1985.

[5] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video Rewrite: Driving Visual Speech with Audio. In *SIGGRAPH 97 Conference Proceedings*, pages 353–360. ACM SIGGRAPH, August 1997.

[6] David T. Chen, Andrei State, and David Banks. Interactive Shape Metamorphosis. In *1995 Symposium on Interactive 3D Graphics*, pages 43–44. ACM SIGGRAPH, April 1995.

[7] Chang S. Choi, Kiyoharu, Hiroshi Harashima, and Tsuyoshi Takebe. Analysis and Synthesis of Facial Image Sequences in Model-Based Image Coding. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 4, pages 257 – 275. June 1994.

[8] Cyberware Laboratory, Inc, Monterey, California. *4020/RGB 3D Scanner with Color Digitizer*, 1990.

[9] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20. ACM SIGGRAPH, August 1996.

[10] Eben Ostby, Pixar Animation Studios. Personal communication, January 1997.

[11] Paul Ekman and Wallace V. Friesen. *Unmasking the Face. A guide to recognizing emotions fron facial clues*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[12] Paul Ekman and Wallace V. Friesen. *Manual for the Facial Action Coding System*. Consulting Psychologists Press, Inc., Palo Alto, California, 1978.

[13] Irfan Essa, Sumit Basu, Trevor Darrell, and Alex Pentland. Modeling, Tracking and Interactive Animation of Faces and Heads Using Input from Video. In *Computer Animation Conference*, pages 68–79. June 1996.

[14] Gary Faigin. *The Artist's Complete Guide to Facial Expression*. Watson-Guptill Publications, New York, 1990.

[15] Olivier Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.

[16] G. Golub and C. F. Van Loan. *Matrix Computation, third edition*. The John Hopkins University Press, Baltimore and London, 1996.

[17] Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Frédéric Pighin. Making Faces. In *SIGGRAPH 98 Conference Proceedings*. ACM SIGGRAPH, July 1998.

[18] Pat Hanrahan and Wolfgang Krueger. Reflection from Layered Surfaces Due to Subsurface Scattering. In *SIGGRAPH 93 Conference Proceedings*, volume 27, pages 165–174. ACM SIGGRAPH, August 1993.

[19] Bright Star Technologies Inc. *Beginning Reading Software*. Sierra On-Line, Inc., 1993.

[20] Horace H. S. Ip and Lijun Yin. Constructing a 3D Individualized Head Model from Two Orthogonal Views. *The Visual Computer*, 12:254–266, 1996.

[21] Gregory Ward J., Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *SIGGRAPH 88 Conference Proceedings*, volume 22, pages 85–92. August 1988.

[22] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape Transformation for Polyhedral Objects. In *SIGGRAPH 92 Proceedings Conference*, volume 26, pages 47–54. ACM SIGGRAPH, July 1992.

[23] Rolf M. Koch, Markus H. Gross, Friedrich R. Carls, Daniel F. von Büren, George Fankhauser, and Yoav I. H. Parish. Simulating Facial Surgery Using Finite Element Methods. In *SIGGRAPH 96 Conference Proceedings*, pages 421–428. ACM SIGGRAPH, August 1996.

[24] Tsuneya Kurihara and Kiyoshi Arai. A Transformation Method for Modeling and Animation of the Human Face from Photographs. In Nadia Magnenat Thalmann and Daniel Thalmann, editors, *Computer Animation 91*, pages 45–58. Springer-Verlag, Tokyo, 1991.

[25] A. Lanitis, C. J. Taylor, and T. F. Cootes. A Unified Approach for Coding and Interpreting Face Images. In *Fifth International Conference on Computer Vision (ICCV 95)*, pages 368–373. Cambridge, Massachusetts, June 1995.

[26] C. L. Lawson and R. J. Hansen. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, 1974.

[27] Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, and George Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. In *SIGGRAPH 95 Conference Proceedings*, pages 439–448. ACM SIGGRAPH, August 1995.

[28] Seung-Yong Lee, George Wolberg, Kyung-Yong Chwa, and Sung Yong Shin. Image Metamorphosis with Scattered Feature Constraints. *IEEE Transactions on Visualization and Computer Graphics*, 2(4), December 1996.

[29] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic Modeling for Facial Animation. In *SIGGRAPH 95 Conference Proceedings*, pages 55–62. ACM SIGGRAPH, August 1995.

[30] Yuencheng C. Lee, Demetri Terzopoulos, and Keith Waters. Constructing Physics-Based Facial Models of Individuals. In *Proceedings of Graphics Interface 93*, pages 1–8. May 1993.

[31] Francis H. Moffitt and Edward M. Mikhail. *Photogrammetry*. Harper & Row, New York, 3 edition, 1980.

[32] Shree K. Nayar, Katsushi Ikeuchi, and Takeo Kanade. Shape from Interreflections. *International Journal of Computer Vision*, 6:173–195, 1991.

[33] Gregory M. Nielson. Scattered Data Modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, January 1993.

[34] Frederic I. Parke. Computer Generated Animation of Faces. *Proceedings ACM annual conference.*, August 1972.

[35] Frederic I. Parke. A Parametric Model for Human Faces. PhD thesis, University of Utah, Salt Lake City, Utah, December 1974. UTEC-CSc-75-047.

[36] Frederic I. Parke and Keith Waters. *Computer Facial Animation*. A K Peters, Wellesley, Massachusetts, 1996.

[37] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, second edition, 1992.

[38] Kari Pulli, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Proc. 8th Eurographics Workshop on Rendering*. June 1997.

[39] Steven M. Seitz and Charles R. Dyer. View Morphing. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 21–30. ACM SIGGRAPH, August 1996.

[40] Chester C. Slama, editor. *Manual of Photogrammetry*. American Society of Photogrammetry, Falls Church, Virginia, fourth edition, 1980.

[41] Richard Szeliski and Sing Bing Kang. Recovering 3D Shape and Motion from Image Streams using Nonlinear Least Squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, March 1994.

[42] Richard Szeliski and Heung-Yeung Shum. Creating Full View Panoramic Image Mosaics and Texture-Mapped Models. In *SIGGRAPH 97 Conference Proceedings*, pages 251–258. ACM SIGGRAPH, August 1997.

[43] Demetri Terzopoulos and Keith Waters. Physically-based Facial Modeling, Analysis, and Animation. *Journal of Visualization and Computer Animation*, 1(4):73–80, March 1990.

[44] Kristinn R. Thórisson. Gandalf: An Embodied Humanoid Capable of Real-Time Multimodal Dialogue with People. In *First ACM International Conference on Autonomous Agents*. 1997.

[45] Michael W. Vannier, Jeffrey F. Marsh, and James O. Warren. Three-dimentional Computer Graphics for Craniofacial Surgical Planning and Evaluation. In *SIGGRAPH 83 Conference Proceedings*, volume 17, pages 263–273. ACM SIGGRAPH, August 1983.

[46] Keith Waters. A Muscle Model for Animating Three-Dimensional Facial Expression. In *SIGGRAPH 87 Conference Proceedings)*, volume 21, pages 17–24. ACM SIGGRAPH, July 1987.

[47] Lance Williams. Performance-Driven Facial Animation. In *SIGGRAPH 90 Conference Proceedings*, volume 24, pages 235–242. August 1990.

[48] Z. Zhang, K. Isono, and S. Akamatsu. Euclidean Structure from Uncalibrated Images Using Fuzzy Domain Knowledge: Application to Facial Images Synthesis. In *Proc. International Conference on Computer Vision (ICCV'98)*. January 1998.

A Least squares for pose recovery

To solve for a subset of the parameters given in Equation (2), we use linear least squares. In general, given a set of linear equations of the form

$$\boldsymbol{a}_j \cdot \boldsymbol{x} - b_j = 0, \tag{4}$$

we solve for the vector $\boldsymbol{x}$ by minimizing

$$\sum_j (\boldsymbol{a}_j \cdot \boldsymbol{x} - b_j)^2. \tag{5}$$

Setting the partial derivative of this sum with respect to $\boldsymbol{x}$ to zero, we obtain

$$\sum_j (\boldsymbol{a}_j \boldsymbol{a}_j^T)\boldsymbol{x} - b_j \boldsymbol{a}_j = 0, \tag{6}$$

i.e., we solve the set of *normal equations* [16]

$$\left(\sum_j \boldsymbol{a}_j \boldsymbol{a}_j^T \right) \boldsymbol{x} = \sum_j b_j \boldsymbol{a}_j. \tag{7}$$

More numerically stable methods such as *QR* decomposition or Singular Value Decomposition [16] can also be used to solve the least squares problem, but we have not found them to be necessary for our application.

To update one of the parameters, we simply pull out the relevant linear coefficient $\boldsymbol{a}_j$ and scalar value b_j from Equation (2). For example, to solve for $\boldsymbol{p}_i$, we set

$$\begin{aligned} \boldsymbol{a}_{2k+0} &= w_i^k (x_i^k \eta^k \boldsymbol{r}_z^k - s^k \boldsymbol{r}_x^k), & b_{2k+0} &= w_i^k (s^k t_x^k - x_i^k) \\ \boldsymbol{a}_{2k+1} &= w_i^k (y_i^k \eta^k \boldsymbol{r}_z^k - s^k \boldsymbol{r}_y^k), & b_{2k+1} &= w_i^k (s^k t_y^k - y_i^k). \end{aligned}$$

For a scalar variable like s^k, we obtain scalar equations

$$\begin{aligned} \boldsymbol{a}_{2k+0} &= w_i^k (\boldsymbol{r}_x^k \cdot \boldsymbol{p}_i + t_x^k), & b_{2k+0} &= w_i^k \left(x_i^k + x_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) \right) \\ \boldsymbol{a}_{2k+1} &= w_i^k (\boldsymbol{r}_y^k \cdot \boldsymbol{p}_i + t_y^k), & b_{2k+1} &= w_i^k \left(y_i^k + y_i^k \eta^k (\boldsymbol{r}_z^k \cdot \boldsymbol{p}_i) \right). \end{aligned}$$

Similar equations for $\boldsymbol{a}_j$ and b_j can be derived for the other parameters t_x^k, t_y^k, and η^k. Note that the parameters for a given camera k or 3D point i can be recovered independently of the other parameters.

Solving for rotation is a little trickier than for the other parameters, since $\boldsymbol{R}$ must be a valid rotation matrix. Instead of updating the elements in $\boldsymbol{R}_k$ directly, we replace the rotation matrix $\boldsymbol{R}^k$ with $\tilde{\boldsymbol{R}}\boldsymbol{R}^k$ [42], where $\tilde{\boldsymbol{R}}$ is given by Rodriguez's formula [15]:

$$\tilde{\boldsymbol{R}}(\hat{\boldsymbol{n}}, \theta) = \boldsymbol{I} + \sin\theta \boldsymbol{X}(\hat{\boldsymbol{n}}) + (1 - \cos\theta)\boldsymbol{X}^2(\hat{\boldsymbol{n}}), \tag{8}$$

where θ is an incremental rotation angle, $\hat{\boldsymbol{n}}$ is a rotation axis, and $\boldsymbol{X}(\boldsymbol{v})$ is the cross product operator

$$\boldsymbol{X}(\boldsymbol{v}) = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}. \tag{9}$$

A first order expansion of $\tilde{\boldsymbol{R}}$ in terms of the entries in $\boldsymbol{v} = \theta\hat{\boldsymbol{n}} = (v_x, v_y, v_z)$ is given by $\boldsymbol{I} + \boldsymbol{X}(\boldsymbol{v})$.

Substituting into Equation (2) and letting $\boldsymbol{q}_i = \boldsymbol{R}^k \boldsymbol{p}_i$, we obtain

$$\begin{aligned} w_i^k \left(x_i^k + x_i^k \eta^k (\tilde{\boldsymbol{r}}_z^k \cdot \boldsymbol{q}_i) - s^k (\tilde{\boldsymbol{r}}_x^k \cdot \boldsymbol{q}_i + t_x^k) \right) &= 0 \\ w_i^k \left(y_i^k + y_i^k \eta^k (\tilde{\boldsymbol{r}}_z^k \cdot \boldsymbol{q}_i) - s^k (\tilde{\boldsymbol{r}}_y^k \cdot \boldsymbol{q}_i + t_y^k) \right) &= 0, \end{aligned} \tag{10}$$

where $\tilde{\boldsymbol{r}}_x^k = (1, -v_z, v_y)$, $\tilde{\boldsymbol{r}}_y^k = (v_z, 1, -v_x)$, $\tilde{\boldsymbol{r}}_z^k = (-v_y, v_x, 1)$, are the rows of $[\boldsymbol{I} + \boldsymbol{X}(\boldsymbol{v})]$. This expression is linear in (v_x, v_y, v_z), and hence leads to a 3×3 set of normal equations in (v_x, v_y, v_z). Once the elements of $\boldsymbol{v}$ have been estimated, we can compute θ and $\hat{\boldsymbol{n}}$, and update the rotation matrix using

$$\boldsymbol{R}^k \leftarrow \tilde{\boldsymbol{R}}(\hat{\boldsymbol{n}}^k, \theta^k)\boldsymbol{R}^k.$$

DRAFT - Final version to appear in Proc. of IEEE Conf. Computer Vision and Pattern Recognition 2000

Recovering Non-Rigid 3D Shape from Image Streams

Christoph Bregler
Computer Science Department
Stanford University
Stanford, CA 94305
bregler@cs.stanford.edu

Aaron Hertzmann Henning Biermann
NYU Media Research Lab
719 Broadway, 12th floor
New York, NY 10003
hertzman@cs.nyu.edu, biermann@cs.nyu.edu

Abstract

This paper addresses the problem of recovering 3D non-rigid shape models from image sequences. For example, given a video recording of a talking person, we would like to estimate a 3D model of the lips and the full face and its internal modes of variation. Many solutions that recover 3D shape from 2D image sequences have been proposed; these so-called structure-from-motion techniques usually assume that the 3D object is rigid. For example Tomasi and Kanade's factorization technique is based on a rigid shape matrix, which produces a tracking matrix of rank 3 under orthographic projection. We propose a novel technique based on a non-rigid model, where the 3D shape in each frame is a linear combination of a set of basis shapes. Under this model, the tracking matrix is of higher rank, and can be factored in a three step process to yield to pose, configuration and shape. We demonstrate this simple but effective algorithm on video sequences of people and animals. We were able to recover 3D non-rigid facial models with high accuracy.

1 Introduction

This paper demonstrates a new technique for recovering 3D non-rigid shape models from 2D image sequences recorded with a single camera. For example, this technique can be applied to video recordings of a talking person. It extracts a 3D model of the human face, including all facial expressions and lip movements.

Previous work has treated the two problems of recovering 3D shapes from 2D image sequences and of discovering a parameterization of non-rigid shape deformations separately. Most techniques that address the *structure-from-motion* problem are limited to rigid objects. For example, Tomasi and Kanade's factorization technique [13] recovers a shape matrix from image sequences. Under orthographic projection, it can be shown that the 2D tracking data matrix has rank 3 and can be factored into 3D pose and 3D shape with the use of the singular value decomposition (SVD). Unfortunately these techniques can not be applied to nonrigid deforming objects, since they are based on the rigidity assumption.

Most techniques that learn models of shape variations do so on the 2D appearance, and do not recover 3D structure. Popular methods are based on Principal Components Analysis. If the object deforms with K linear degrees of freedom, the covariance matrix of the shape measurements has rank K. The principal modes of variation can be recovered with the use of SVD.

We show how 3D non-rigid shape models can be recovered under scaled orthographic projection. The 3D shape in each frame is a linear combination of a set of K basis shapes. Under this model, the 2D tracking matrix is of rank $3K$ and can be factored into 3D pose, object configuration and 3D basis shapes with the use of SVD. We demonstrated the effectiveness of this technique on several data sets, including challenging recordings of human faces during speech and varying facial expressions and animal body motions.

Section 2 summarizes related approaches, Section 3 describes our algorithm, and Section 4 discusses our experiments.

2 Previous Work

Many methods have been proposed to solve the *Structure-from-motion* problem. One of the most influential of these was proposed by Tomasi and Kanade [13] who demonstrated the factorization method for rigid objects and orthographic projections. Many extensions have been proposed, such as the multi-body factorization method of Coseira and Kanade [5] that relaxes the rigidity constraint. In this method, K independently moving objects are allowed, which results in a tracking matrix of rank $3K$ and a permutation algorithm that identifies the submatrix corresponding to each object. More recently, Bascle and Blake

[1] proposed a solution for factoring facial expressions and pose during tracking. Although it exploits the bilinearity of 3D pose and nonrigid object configuration, it requires a set of basis images selected before factorization is performed. The discovery of these basis images is not part of their algorithm.

Various authors have demonstrated estimation of nonrigid appearance in 2D using Principal Components Analysis [14, 9, 3].

The most impressive work for 3D reconstruction of human faces was presented by [4]. A high-resolution 3D model of the shape space was obtained by laser scanning a large face database a-priori. Using a hand initialization and iterative matching of shape, texture, and lighting, a very detailed 3D face shape could be recovered from one single image. Based on 2D image sequences, [6] and [10] were tracking the pose and configuration of human faces. A 3D face model was given a-priori as well. Basu [2] demonstrates how the parameters can be iteratively fitted to a video sequence, starting from an initial lip model. [11, 7] propose methods for recovering the 3D facial model itself using multiple views.

To the best of our knowledge, all existing methods for nonrigid 3D shapes either need an a-priori model, or need multiple views. In the next section, we demonstrate how a 3D nonrigid shape model can be recovered from single-view recordings in solving multiple factorization steps. No a-priori shape model is required. We demonstrate this technique on various recordings of human faces and animals.

3 Factorization Algorithm

We describe the shape of the non-rigid object as a key-frame basis set $S_1, S_2, ...S_k$. Each key-frame S_i is a $3 \times P$ matrix describing P points. The shape of a specific configuration is a linear combination of this basis set:

$$S = \sum_{i=1}^{K} l_i \cdot S_i \qquad S, S_i \in I\!R^{3\times P}, l_i \in I\!R \tag{1}$$

Under a scaled orthographic projection, the P points of a configuration S are projected into $2D$ image points (u_i, v_i):

$$\begin{bmatrix} u_1 & u_2 & ... & u_P \\ v_1 & v_2 & ... & v_P \end{bmatrix} = R \cdot \left(\sum_{i=1}^{K} l_i \cdot S_i \right) + T \tag{2}$$

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \end{bmatrix} \tag{3}$$

R contains the first 2 rows of the full $3D$ camera rotation matrix, and T is the camera translation. The scale of the projection is coded in $l_1, ...l_K$. As in Tomasi-Kanade, we eliminate T by subtracting the mean of all $2D$ points, and henceforth can assume that S is centered at the origin.

We can rewrite the linear combination in (2) as a matrix-matrix multiplication:

$$\begin{bmatrix} u_1 & ... & u_P \\ v_1 & ... & v_P \end{bmatrix} = \begin{bmatrix} l_1 R & ... & l_K R \end{bmatrix} \cdot \begin{bmatrix} S_1 \\ S_2 \\ ... \\ S_K \end{bmatrix} \tag{4}$$

We add a temporal index to each $2D$ point, and denote the tracked points in frame t as $(u_i^{(t)}, v_i^{(t)})$. We assume we have $2D$ point tracking data over N frames and code them in the tracking matrix W:

$$W = \begin{bmatrix} u_1^{(1)} & ... & u_P^{(1)} \\ v_1^{(1)} & ... & v_P^{(1)} \\ u_1^{(2)} & ... & u_P^{(2)} \\ v_1^{(2)} & ... & v_P^{(2)} \\ & ... & \\ u_1^{(N)} & ... & u_P^{(N)} \\ v_1^{(N)} & ... & v_P^{(N)} \end{bmatrix}$$

Using (4) we can write:

$$W = \underbrace{\begin{bmatrix} l_1^{(1)} R^{(1)} & ... & l_K^{(1)} R^{(1)} \\ l_1^{(2)} R^{(2)} & ... & l_K^{(2)} R^{(2)} \\ & ... & \\ l_1^{(N)} R^{(N)} & ... & l_K^{(N)} R^{(N)} \end{bmatrix}}_{Q} \cdot \underbrace{\begin{bmatrix} S_1 \\ S_2 \\ ... \\ S_K \end{bmatrix}}_{B} \tag{5}$$

3.1 Basis Shape Factorization

Equation (5) shows that the tracking matrix has rank $3K$ and can be factored into 2 matrixes: Q contains for each time frame t the pose $R^{(t)}$ and configuration weights $l_1^{(t)}, ..., l_K^{(t)}$. B codes the K key-frame basis shapes S_i. The factorization can be done using singular value decomposition (SVD) by only considering the first $3K$ singular vectors and singular values (first $3K$ columns in U, D, V):

$$\textbf{SVD:} \quad W^{2N\times P} = \hat{U} \cdot \hat{D} \cdot \hat{V}^T = \hat{Q}^{2N\times 3K} \cdot \hat{B}^{3K\times P} \tag{6}$$

3.2 Factoring Pose from Configuration

In the second step, we extract the camera rotations $R^{(t)}$ and shape basis weights $l_i^{(t)}$ from the matrix $\hat{Q}$. Although $\hat{Q}$ is a $2N \times 3K$ matrix, it only contains $N(K+6)$ free variables. Consider the 2 rows of $\hat{Q}$ that correspond to one single time frame t, namely rows $2t-1$ and row $2t$ (for convenience we drop the time index (t)):

$$\begin{aligned} q^{(t)} &= \begin{bmatrix} l_1^{(t)} R^{(t)} & ... & l_K^{(t)} R^{(t)} \end{bmatrix} \\ &= \begin{bmatrix} l_1 r_1 & l_1 r_2 & l_1 r_3 & ... & l_K r_1 & l_K r_2 & l_K r_3 \\ l_1 r_4 & l_1 r_5 & l_1 r_6 & ... & l_K r_4 & l_K r_5 & l_K r_6 \end{bmatrix} \end{aligned}$$

We can reorder the elements of $q^{(t)}$ into a new matrix $\overline{q}^{(t)}$:

$$\overline{q}^{(t)} = \begin{bmatrix} l_1 r_1 & l_1 r_2 & l_1 r_3 & l_1 r_4 & l_1 r_5 & l_1 r_6 \\ l_2 r_1 & l_2 r_2 & l_2 r_3 & l_2 r_4 & l_2 r_5 & l_2 r_6 \\ & & & \dots & & \\ l_K r_1 & l_K r_2 & l_K r_3 & l_K r_4 & l_K r_5 & l_K r_6 \end{bmatrix}$$

$$= \begin{bmatrix} l_1 \\ l_2 \\ \dots \\ l_K \end{bmatrix} \cdot \begin{bmatrix} r_1 & r_2 & r_3 & r_4 & r_5 & r_6 \end{bmatrix}$$

which shows that $\overline{q}^{(t)}$ is of rank 1 and can be factored into the pose $\hat{R}^{(t)}$ and configuration weights $l_i^{(t)}$ by SVD. We successively apply the reordering and factorization to all time blocks of $\hat{Q}$.

3.3 Adjusting Pose and Shape

In the final step, we need to enforce the orthonormality of the rotation matrices. As in [13], a linear transformation G is found by solving a least squares problem[1]. The transformation G maps all $\hat{R}^{(t)}$ into an orthonormal $R^{(t)} = \hat{R}^{(t)} \cdot G$. The inverse transformation must be applied to the key-frame basis $\hat{B}$ to keep the factorization consistent: $S_i = G^{-1} \cdot \hat{S}_i$.

We are now done. Given $2D$ tracking data W, we can estimate a non-rigid 3D shape matrix with K degrees of freedom, and the corresponding camera rotations and configuration weights for each time frame.

4 Experiments

Part of this work is motivated by our efforts in image-based facial animation, but the technique is not limited to the facial domain only. We collected several videos of people speaking sentences with various facial expressions. We also collected videos of animals in motion, to demonstrate the generality of this approach. The human face recordings contain rigid head motions, and non-rigid lip, eye, and other facial motions. We tracked important facial features with an appearance-based 2D tracking technique[2]. Figure 1 and 7 shows example tracking results for video-1 and video-2. For facial animation, we want explicit control over the rigid head pose and the implicit facial variations. In the following, we show how we were able to extract a 3D non-rigid face model parameterized by these degrees of freedom. Video-3 contains a walking giraffe (Figure 9). This video was tracked by a point feature tracker[3].

We applied our method to all three video sequences. The first is a public broadcast originally recorded on film in the early 1960's (video-1) and contains 1213 video frames.

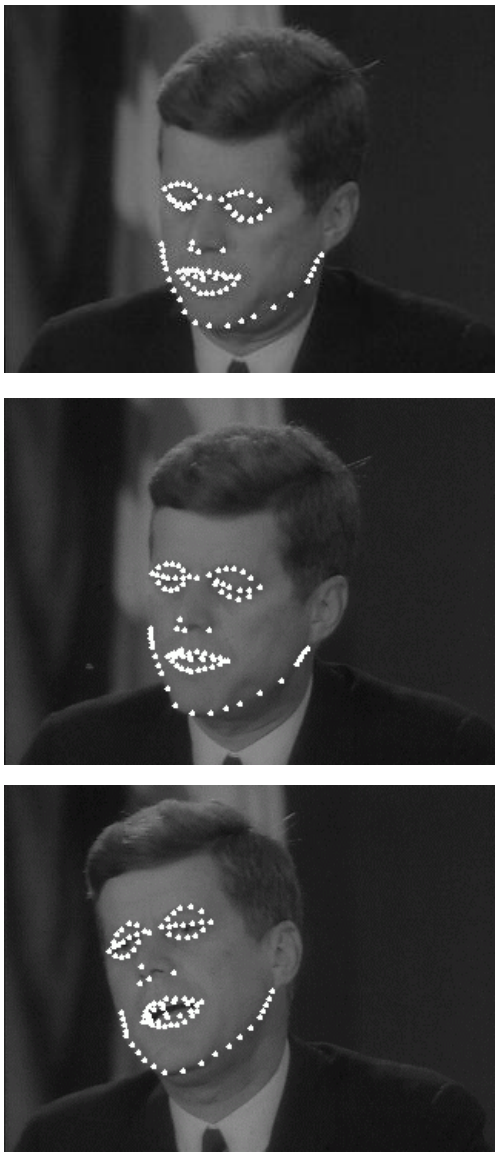


Figure 1: Example images from video-1 with overlayed tracking points. We track the eye brows, upper and lower eye lids, 5 nose points, outer and inner boundary of the lips, and the chin contour.

[1] The least squares problem enforces orthonormality of all $R^{(t)}$: $[r_1 r_2 r_3] G G^T [r_1 r_2 r_3]^T = 1$, $[r_4 r_5 r_6] G G^T [r_4 r_5 r_6]^T = 1$, $[r_1 r_2 r_3] G G^T [r_4 r_5 r_6]^T = 0$

[2] We used a learned PCA-based tracker similar to [9]

[3] We used for this experiment a tracking approach reported in [12]

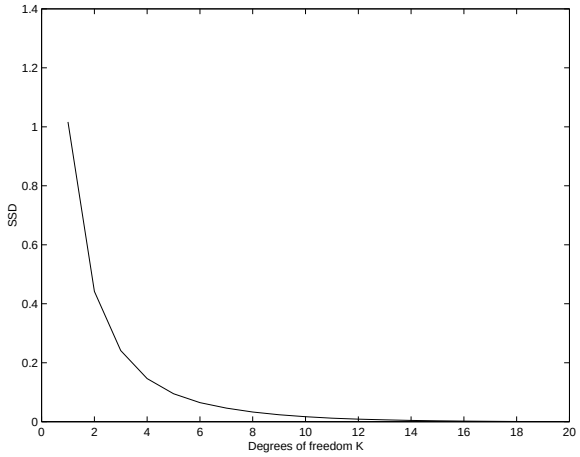


Figure 2: Average pixel SSD error of back-projected face model for different degrees of freedom: K

The second video was recorded in our lab (video-2) and contains 1000 video frames. The third video was recorded in a public zoo and only contains 60 frames. All recordings are challenging for 3D reconstructions, since they contain very few out-of-plane head or body motions. In a first experiment, we computed the reconstruction error based on the number of degrees of freedom (K) for video-1. We factorized the tracking data, and computed the back-projection of the estimated model, configuration, and pose into the image. Figure 2 shows the SSD error between the back-projected points and image measurements. For $K = 16$ the error vanishes. For the remainder of the paper, we set $K = 16$. Figure 3 and 4 shows for example frames of video-1 and the reconstructed 3D S matrix rotated by the corresponding $R^{(t)}$. To illustrate the 3D data better, we fit a shaded smooth surface to the 3D shape points.

We also investigated the discovered modes of variation. We computed the mean and standard deviations of $[l_1^t, ..., l_K^t]$ in video-1. Figure 5 and 6 shows 4 standard deviations of the second and third modes (S_1, S_2, S_3). Mode 1 covers scale change, mode 2 cover some aspect of mouth opening, and mode 3 covers eye opening. The remaining modes pick up more subtle and less intuitive variations.

Figure 8 shows the reconstruction results for video-2.

Figure 9 shows example frames of the walking giraffe. Tracking the complete surface of such an animal is much more difficult. Although it has very distinct features that makes it easier to track than other animals, there are still many local ambiguities to resolve. The reported experiments work in progress. For instance, we could only track features on the trunk, neck, and head with the technique in [12], but not the legs. We envision a combination of several different tracking strategies would be more robust.

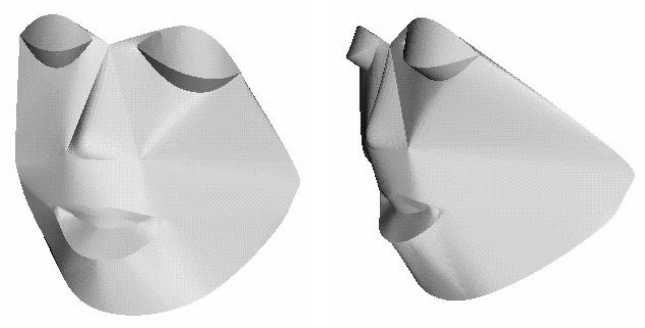


Figure 3: 3D reconstructed shape and pose for first frame of Figure 1

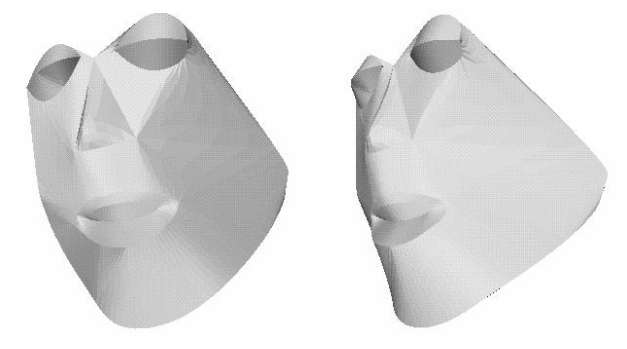


Figure 4: 3D reconstructed shape and pose for last frame of Figure 1

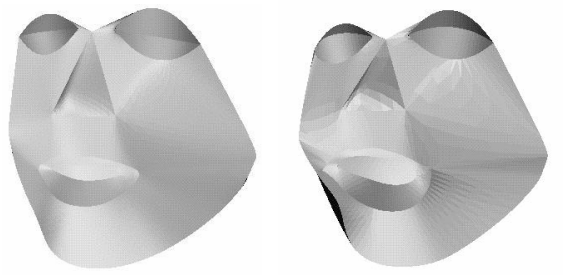


Figure 5: Variation along mode 2 of the nonrigid face model. The mouth deforms.

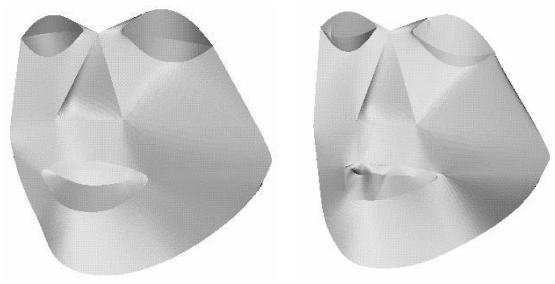


Figure 6: Variation along mode 3 of the nonrigid face model. The eyes close.

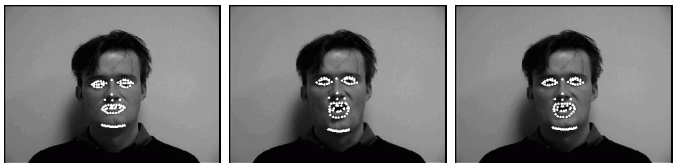


Figure 7: Example images from video-2 with overlayed tracking points.

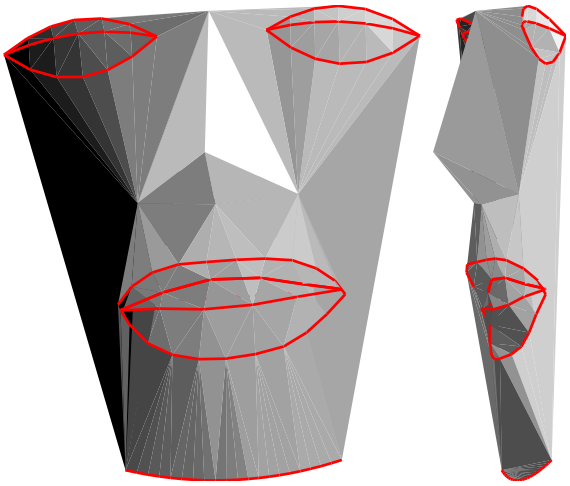


Figure 8: Front and side view for the reconstructions from video-2.

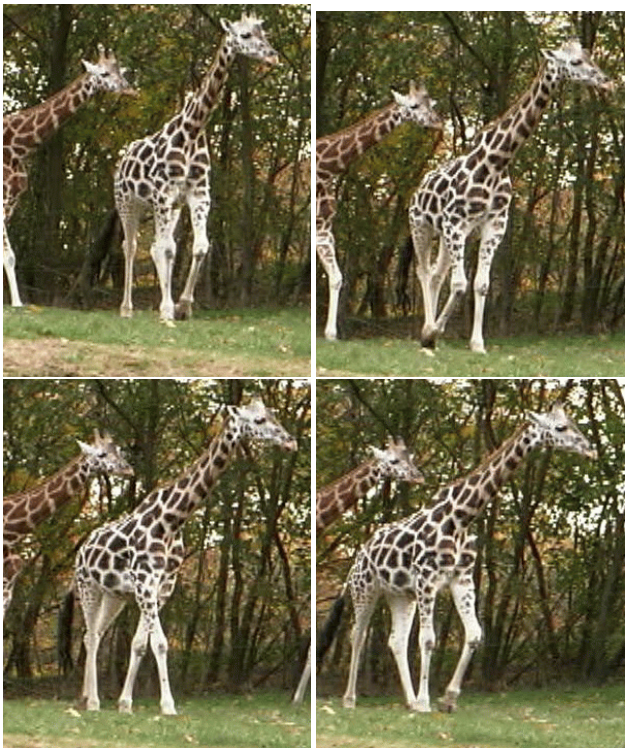


Figure 9: Example frames of the giraffe sequence

Another short-coming is that our technique can not deal with missing tracks yet (see discussion on our future plans). Therefore we could only track 161 features in a sequence of 60 frames total. Figure 10 and 11 shows the 3D reconstruction. Figure 12 illustrates the first mode of variation. The 2 different colored surfaces represent 2 opposing extremes. As you can see, this mode covers some of the head rotations and a deformation of the trunk due to internal bone motion. The second mode of variation is much more subtle and less intuitive (Figure 13).

The results on these 3 video databases are very encouraging. Given the limited range of out-of-plane face and body orientations, the 3D details that we could recover from the lip shapes and skin deformations are quite surprising.

5 Discussion

We have presented a simple but effective new technique for recovering 3D non-rigid shape models from 2D image streams without the use of any a-priori model. It is a three step procedure using multiple factorizations. We were able to recover 3D models for video recordings of human faces and animals. Although these are very encouraging results, we plan to evaluate this technique and its limitations on larger data sets. We also plan to extend this technique such that occluded feature tracks can be handled. For exam-

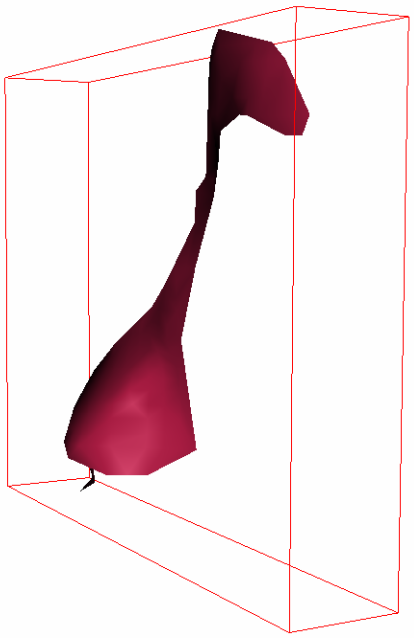


Figure 10: 3D reconstruction of the giraffe surface.

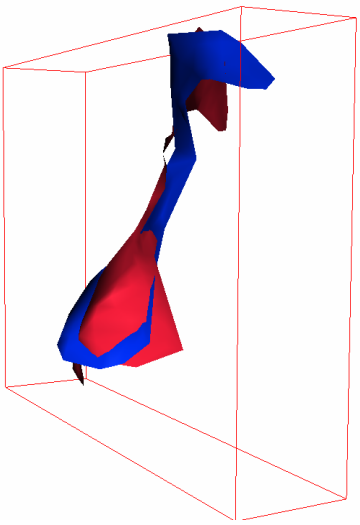


Figure 12: First mode of shape variation of giraffe model.

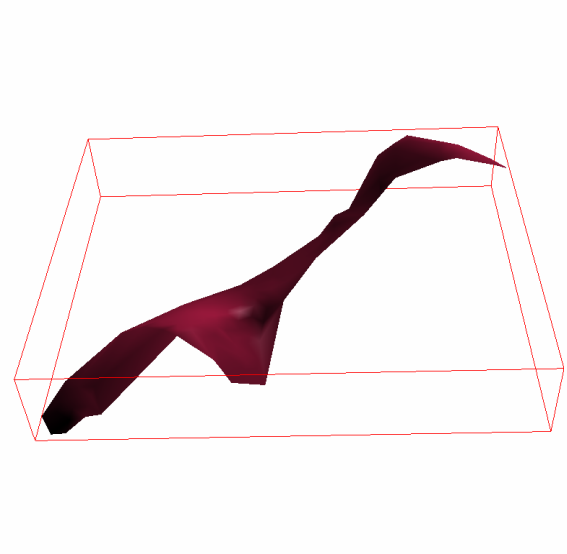


Figure 11: Other view of the 3D reconstruction of the giraffe surface.

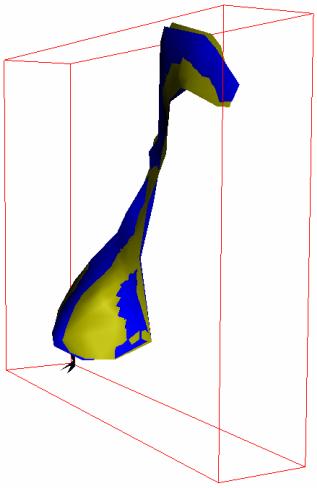


Figure 13: Second mode of shape variation of giraffe model.

ple, [8] demonstrated a technique that deals with missing feature tracks for rigid 3D reconstruction. It projects a incomplete measurement matrix into a matrix of rank 3. The same technique can be used to project the incomplete matrix W into a complete matrix of rank $3K$. With such extensions, we anticipate to track longer sequences that contain many more view angles of the object.

Reconstructing non-rigid models from single-view video recordings has many potential applications. In addition, we intend to apply this technique to our image-based facial and full-body animation system and to a model based tracking system.

Acknowledgments

We like to thank Ken Perlin, Denis Zorin, and Davi Geiger for fruitful discussions, and for supporting this research, Clilly Castiglia and Steve Cooney for helping with the data collection, and New York University, California State MICRO program and Interval Research for partial funding.

References

[1] B. Bascle and A. Blake. Separability of pose and expression in facial tracking and animation. In *Proc. Int. Conf. Computer Vision*, 1998.

[2] S. Basu. A three-dimensional model of muman lip motion. In *EECS Master Thesis, MIT Media Lab Report 417*, 1997.

[3] A. Blake, M. Isard, and D. Reynard. Learning to track the visual motion of contours. In *J. Artificial Intelligence*, 1995.

[4] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. *Proceedings of SIGGRAPH 99*, pages 187–194, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[5] J. Costeira and T. Kanade. A multi-body factorization method for motion analysis. *Int. J. of Computer Vision*, pages 159–180, Sep 1998.

[6] Douglas DeCarlo and Dimitris Metaxas. Deformable model-based shape and motion analysis from images using motion residual error. In *Proc. Int. Conf. Computer Vision*, 1998.

[7] Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Frédéric Pighin. Making faces. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 55–66. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[8] D. Jacobs. Linear fitting with missing data for structure-from-motion. In *Proc. IEEE. Conf. Computer Vision and Pattern Recognition*, 1997.

[9] A. Lanitis, Taylor C.J., Cootes T.F., and Ahmed T. Automatic interpretation of human faces and hand gestures using flexible models. In *International Workshop on Automatic Face- and Gesture-Recognition*, 1995.

[10] F. Pighin, D. H. Salesin, and R. Szeliski. Resynthesizing facial animation through 3d model-based tracking. In *Proc. Int. Conf. Computer Vision*, 1999.

[11] Frédéric Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, and David H. Salesin. Synthesizing realistic facial expressions from photographs. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 75–84. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[12] J. Shi and C. Tomasi. Good features to track. In *CVPR*, 1994.

[13] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *Int. J. of Computer Vision*, 9(2):137–154, 1992.

[14] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

Applications of Image-Based Modeling and Rendering in Art and Cinema

Paul Debevec

Overview

- **Image-based techniques are used in many ways; for each we can ask:**

What is the model?

How are images used?

What is the rendering method?

How are images made?

What is the effect?

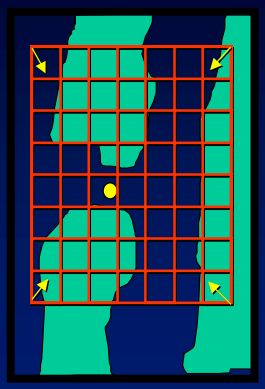
Is what we see what is really there?

Aspen Moviemap
(MIT, 1978-1980)

- Vehicle-mounted camera filmed driving down all streets of downtown Aspen
- Video transferred to random-access videodisc
- User interface allows real-time photorealistic exploration of the space

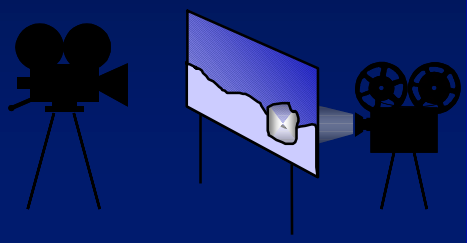
Golden Gate Moviemap
Michael Naimark, Exploratorium 1987

- Helicopter flown in 2D grid pattern over Bay Area
- Camera kept pointed toward Golden Gate Bridge
- Film transferred to videodisc
- Trackball interface allows real-time fly-over of San Francisco



Matte Painting

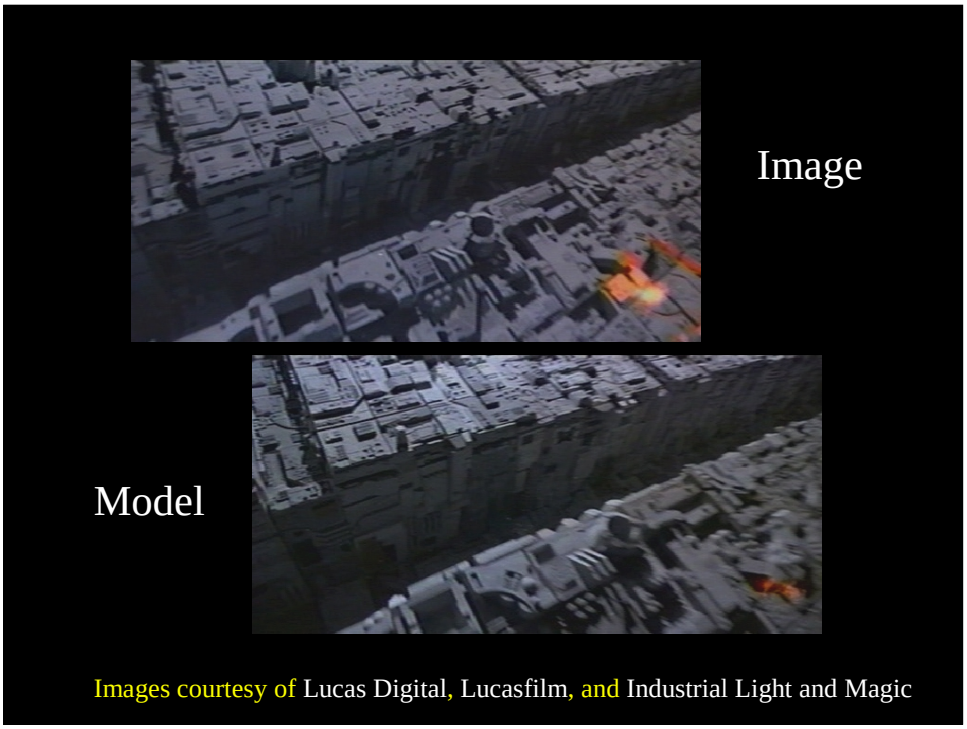
- Instead of building a set, just create an image!
- Used since the early days of the film industry
  - *Gone With the Wind (1939)*
- Add actors, etc.
- Can pan and zoom to simulate motion
- Now done digitally



Star Wars Trench Sequence

Industrial Light and Magic, 1977

Images courtesy of Lucas Digital, Lucasfilm, and Industrial Light and Magic



Images courtesy of Lucas Digital, Lucasfilm, and Industrial Light and Magic

Tour into the Picture

(Horry, Anjyo, Arai, Hitachi SIGGRAPH 97)

- **An approximate 3D model is constructed interactively with easy-to-use tools**
- **Foreground objects modeled as flat cut-outs**
- **User fills in the occluded areas**
- **Animations from the single photograph reveal depth as well as visual detail**
- **Software at:**
  http://koigakubo.hitachi.co.jp/little/DL_TipE.html

Displacements

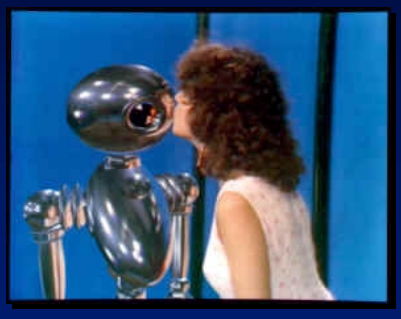
Michael Naimark, San Francisco Museum of Modern Art, 1984

- **Image-based modeling and rendering with real geometry and real light**
- **Living room filmed with rotating movie camera**
- **Room painted white**
- **Film reprojected with rotating movie projector**

"Interface"

Lance Williams, Carter Burwell, Ned Greene SIGGRAPH 85 Film Show

- **First use of Image-Based Reflection Mapping in an Animation**



http://www.CS.Berkeley.EDU/~debevec/ReflectionMapping/

Reflection Mapping in Films

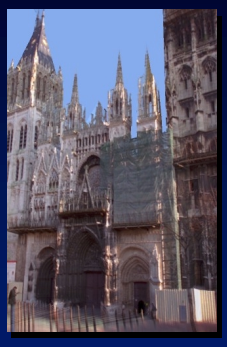
Flight of the Navigator
Randal Kleiser
1986

Terminator II
James Cameron
1991

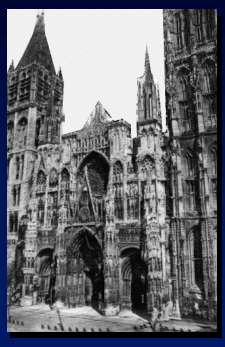
http://www.CS.Berkeley.EDU/~debevec/ReflectionMapping/

Rouen Revisited

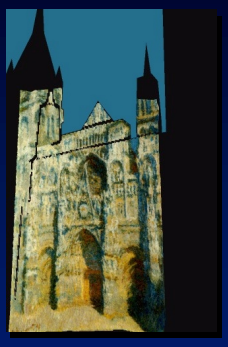
(Golan Levin and Paul Debevec, SIGGRAPH 96 Art Show)



Synthetic View:
1996



Synthetic View:
1896



Synthetic View:
Monet Painting

Like a Rolling Stone

Buf Compagnie, SIGGRAPH 96 Electronic Theatre

- View interpolation based on stereo reconstruction between two views of the same scene (Software by Arnauld Lamorlette)
- Free-style morphing between sparsely sampled frames
  - Artifacts serve artistic effect

BUF COMPAGNIE

3D computer effects: Buf Compagnie
Producer: Partizan-midi-minuit
Director: Michel Gondry
See Also: *City of Lost Children*, commercials for Reebok, Chanel, Gap, and *Fight Club*

Mona Lisa View Morph

Steve Seitz, SIGGRAPH 96

Seitz, S. and C. Dyer. *View Morphing*. SIGGRAPH 96.

Mona Lisa View Morph

Steve Seitz, SIGGRAPH 96

Seitz, S. and C. Dyer. *View Morphing*. SIGGRAPH 96.

The Prince of Egypt

IBR for NPR!

"Exposure"

- **Geometry constructed in 3D**
- **Shaded renderings given to artists to paint**
- **Digital paintings used as 3D textures**

Video courtesy of Dreamworks, LLC

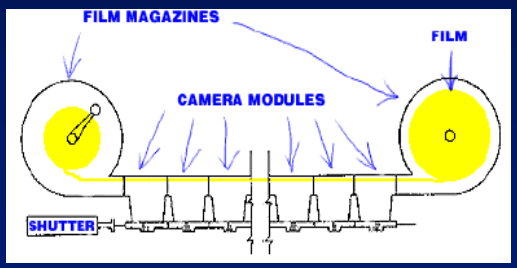
More use of IBR for NPR in Films

- *Tarzan –“ Deep Canvas”*
- *What Dreams May Come*
  Pierre Jasmin and Pete Litwinowicz, RE:Vision

Dayton Taylor’s Virtual Camera

http://www.virtualcamera.com/

- Array of pictures taken onto same roll of film
- Playback achieves “frozen time” effect



The Campanile Movie

SIGGRAPH 97 Electronic Theatre

Tower Photographs

Environment Photographs

- Model of Campus Built from 20 images
- Real-time virtual animation created w/ projective texture-mapping

The Matrix

www.mvfx.com

George Borshukov, Dan Piponi, Kim Libreri, and John Gaeta, MANEX Entertainment

www.whatisthematrix.com

Conclusion

IBMR in Art and Cinema

- Image-based techniques have a long lineage
- Allow reinterpretation of existing imagery
- Enable many new effects
- Artifacts can be interesting
- IBMR techniques ~~will likely~~ have become standard tools in filmmaking

Thanks

Ken Anjyo, George Borshukov, Michael Naimark, H.B. Seigel, Ellen Pasternack, Jeanne Cole, Arnauld Lamorlette, Steve Seitz, Linda Branagan, Dayton Taylor, Craig Barron, Lance Williams, Golan Levin, Industrial Light and Magic, Lucas Digital, Ltd., Lucasfilm, Ltd., Dreamworks LLC, Buf Compagnie, Virtual Camera, MANEX Entertainment

Rouen Revisited

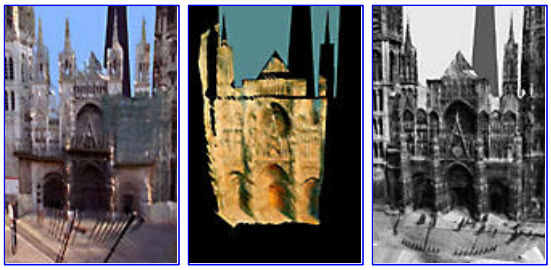
An interactive art installation by
Golan Levin *(Interval Research Corporation)* and
Paul Debevec *(University of California at Berkeley)*

Française • **Deutsche**

Presented at:
The Siggraph '96 Art Show,
New Orleans, USA, 4 - 9 August 1996

Fleshfactor: Ars Electronica Festival '97
Linz, Austria, 8 - 13 September 1997

Eighth International Symposium on Electronic Art (ISEA '97)
Chicago, USA, 22 - 27 September 1997



Synthetic views of the Rouen Cathedral derived (respectively) from:
photographs taken in January 1996; a painting by Claude Monet, made in 1894-95;
and photographs taken in the mid-1890's. All are shown from the same point of view.

Table of Contents

Overview

Between 1892 and 1894, the French Impressionist Claude Monet produced nearly 30 oil paintings of the main façade of the Rouen Cathedral in Normandy. Fascinated by the play of light and atmosphere over the Gothic church, Monet systematically painted the cathedral at different times of day, from slightly different angles, and in varied weather conditions. Each painting, quickly executed, offers a glimpse into a narrow slice of time and mood.

We are interested in widening these slices, extending and connecting the dots occupied by Monet's paintings in the multidimensional space of turn-of-the-century Rouen. In *Rouen Revisited*, we present an interactive kiosk in which users are invited to explore the façade of the Rouen Cathedral, as Monet might have painted it, from any angle, time of day, and degree of atmospheric haze. Users can contrast these re-rendered paintings with similar views synthesized from century-old archival photographs, as well as from recent photographs that reveal the scars of a century of weathering and war.

Rouen Revisited is our homage to the hundredth anniversary of Monet's cathedral paintings. Like Monet's series, our installation is a constellation of impressions, a document of moments and percepts played out over space and time. In our homage, we extend the scope of Monet's study to where he could not go, bringing forth his object of fascination from a hundred feet in the air and across a hundred years of history.

The Technology

To produce renderings of the cathedral's façade from arbitrary angles, we needed an accurate, three-dimensional model of the cathedral. For this purpose, we made use of new modeling and rendering techniques, developed by Paul Debevec at the University of California at Berkeley, that allow three-dimensional models of architectural scenes to be constructed from a small number of ordinary photographs. We traveled to Rouen in January 1996, where, in addition to taking a set of photographs from which we could generate the model, we obtained reproductions of Monet's paintings as well as antique photographs of the cathedral as it would have been seen by Monet.

Once the 3D model was built, the photographs and Monet paintings were registered with and projected onto the 3D model. Re-renderings of each of the projected paintings and photographs were then generated from hundreds of points of view; renderings of the cathedral in different atmospheric conditions and at arbitrary times of day were derived from our own time-lapse photographs of the cathedral and by interpolating between the textures of Monet's original paintings. The model recovery and image rendering was accomplished with custom graphics software on a Silicon Graphics Indigo2. The *Rouen Revisited* interface runs in Macromedia Director on a 166-MHz Pentium PC, and allows unencumbered exploration of more than 12,000 synthesized renderings.

The execution of *Rouen Revisited* entailed more than half a dozen novel technical achievements. The most basic of these were the techniques of view-dependent texture-mapping, photogrammetric modeling, and model-based stereo that Paul Debevec developed in his Berkeley Ph.D. thesis. Other achievements, however, were more specific to the *Rouen Revisited* installation itself. A brief survey of the technological accomplishments in *Rouen Revisited* can be found here.

Further information about the modeling and rendering algorithms used in *Rouen Revisited* can be found in:

- Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs. In SIGGRAPH '96 Proceedings, August 1996.

The Presentation

Rouen Revisited is presented in an arch-shaped maple cabinet, seven feet three inches tall. Its front face is articulated by three features: Near the top, a backlit stained-glass rosette (whose design is based on the rosette of the Rouen Cathedral) acts as a beacon for passers-by. Below that, a 17-inch color monitor, configured on its side, provides users with a view onto the cathedral's surface. Finally, a projecting wedge-shaped block at waist-level provides the interface controls for operating the kiosk.

Users explore the surface of the Rouen Cathedral by touching one of three force-sensitive regions exposed within a brass plate mounted on the interface wedge. Each region affords the user with control of a different dimension of the façade:

- Touching the corners of the upper, triangular region of the brass plate allows users to select between renderings of Monet paintings, archival photographs from the 1890's, or new photographs from 1996. Dragging one's finger along this groove creates a blend between these modes.

- Moving one's finger left and right inside the central, upside-down-T-shaped region of the brass control plate allows users to change the time of day. Moving one's finger up and down the vertical groove of this control changes the level of fog. This control is disabled for the archival photographs, for which time-series and fog-series source stills were unavailable. Nevertheless, this control is active for the new photographs and Monet paintings, and permits users to draw comparisons between the actual appearance of the cathedral (given certain lighting conditions) and Monet's interpretation of the cathedral so lit.

- Dragging one's finger across the rectangular, bottom region of the brass plate allows users to change their point of view around the Rouen Cathedral.

The Experience

The *Rouen Revisited* interactive kiosk allows its users to explore eight dimensions of the façade of the Rouen Cathedral in Normandy. We can examine the cathedral in various levels of fog; at different times of day; from different points of view on a three-dimensional viewing surface—and lastly, along three dimensions of interpretation and media: namely, as the cathedral appeared to photographers a hundred years ago, as it appears today, and as it would appear if Monet's impressionist paintings of it were aligned with and projected onto its surface.

Many more ways of exploring and understanding the Gothic cathedral are afforded by moving between and around these modes. We can observe the many ways in which the cathedral has changed in the past century, for example, by moving between the re-renderings of the old photographs and those of the new photographs. We can come to an understanding of which *geometric* details Monet chose to focus on, by moving between views of his paintings and the historic photographs from the same time period. We can come to understand how the play of light at a given time of day may have inspired Monet to paint the colors and textures he did—by moving between a painting and the new photograph which shares the same time of day. And, when we scrub through the time-series of Monet paintings, we have a unique opportunity to access the entire set of Monet's Cathedral paintings, and gain an appreciation for the both the range of Monet's exploration as well as the constraints within which he chose to work. Finally, by changing the time of day and our point of view around the cathedral, we may derive a *sense of place*—a feeling for the Rouen Cathedral as a real physical artifact, and a sense of the passage of a day in Rouen.

Rouen Revisited is an artifact about artifacts about an artifact—an interactive and open-ended interpretation of paintings and photographs, which are themselves interpretations of an ancient Gothic artwork. Ultimately, the interpretation which *Rouen Revisited* affords is a *dynamic* one, forged in the mind of the user when she creates, using the multidimensional interface, her *own* Rouen Cathedral composition.

Renderings from Rouen Revisited

The unwieldy size of the *Rouen Revisited* image database (nearly 3 gigabytes of renderings) prohibits us from creating an interactive, web-based version of the project at this time. In this section of the Rouen site, we instead attempt to convey the experience made possible by *Rouen Revisited* in a compact, easily-transmissable and (for now) non-interactive form. In addition to presenting still images, we have also converted select paths through the multidimensional space of possible renderings into short animations and digital videos. The animations are presented as animated gifs, reduced to one-ninth of their original size and dithered from 24-bit color down to a browser-safe 8-bit palette. Viewing these animations requires a Netscape 2.0 or better browser. The Quicktime and AVI videos are similarly reduced in size, and additionally compressed with the Apple Video or Microsoft Video compression formats.

To the animations and stills.

Artist Biographies

Golan Levin is an artist and designer of artifacts and experiences. Before he joined Interval Research in 1994, Golan completed his self-made undergraduate degree in Media Arts and Sciences at the Massachusetts Institute of Technology. Since then, he has focused on the design of expressive instruments, tools and toys for producing and playing with media.

Paul Debevec received degrees in Math and Computer Engineering from the University of Michigan in 1992, and recently completed his Ph.D. in Computer Science at the University of California at Berkeley. For his thesis, Paul developed a method of modeling and rendering architectural scenes photorealistically from ordinary photographs by synthesizing techniques from computer vision with those of computer graphics. With no current commitments after graduate school, Paul is interested in continuing to capture, visualize, and interpret the world in new and creative ways through novel photographic techniques.

A photograph of the authors with the *Rouen Revisited* kiosk, August 1996.
A photograph of the authors in front of the Rouen Cathedral, January 1996.

Exhibition Information

Rouen Revisited is available for public exhibition on an expenses-only basis. When it is not on loan to outside exhibitors, *Rouen Revisited* may be seen by appointment at the Interval Research Corporation Gallery. For further information about the *Rouen Revisited* installation, Please email gallery@interval.com or contact the Interval Gallery at (650) 842-6222 [*phone*], (650) 354-0872 [*fax*].

Credits

Rouen Revisited was conceived and developed by Golan Levin and Paul Debevec. The kiosk's maple cabinet was constructed by Warren H. Shaw, furniture-maker, of South San Francisco. The stained-glass rosette was hand-made by David Kaczor, glass artist, of Mountain View, California. The brass interface gate was machined by Shane Levin, president of the HAP Engraving Company, New York City. Invaluable suggestions and assistance with the presentation design were provided by Joe Ansel of Ansel Associates, and Charles "Bud" Lassiter of Interval Research Corporation. Scott Snibbe and Geoff Smith provided key software assistance; additional hardware, electronics and construction support were lent by Scott Wallters, Chris $eguine and Bernie Lubell.

Rouen Revisited would not have been possible without the generous support of Paul Allen, David Liddle, and Noel Hirst; Michael Naimark, Bud Lassiter, Sally Rosenthal, Carol Moran, Marc Davis, Frank Crow, Stephan Gehring, Marie-Dominique Baudot, Laurence Shelvin, and Mark Keen; Jitendra Malik and Camillo J. Taylor; Shane Levin; and Joe Ansel.

Golan Levin
Interval Research Corporation
1801-C Page Mill Road
Palo Alto, CA 94304
+1.650.424.0722
levin@interval.com

Paul Debevec
Computer Science Division
University of California at Berkeley
545 Soda Hall
Berkeley, CA 94720-1776
debevec@cs.berkeley.edu