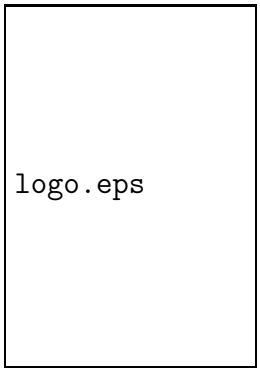
Otto-von-Guericke-Universität Magdeburg
University of Otago



Department of Computer Science

Painting with the Watching Window:
Design and Implementation of a Tracking-Based Painting Application

Report

Author:

Hanna von Tenspolde

August 12, 2004

Supervisors:

Prof. Maic Masuch (Universität Magdeburg)
Prof. Geoff Wyvill (University of Otago)

Contents

Chapter 1

Introduction

This thesis was written after having absolved an internship of 5 months length at the University of Otago, Dunedin, New Zealand. It is a report on the revision of a prototype virtual reality environment called "Watching Window". It gives an overview of different approaches of hardware interface design and explains our choice of using a vision based solution. The results of the updated system are presented. Several applications that demonstrate the capability of the system are described and the development of an additional one is accompanied.
There were mainly two students working on this project. Our task was to revise the existing code architecture and to develop more complex tracking methods and an additional demo application. My part of the task was to revise the graphics component of the system and implement a simple tracking based painting application. Robert Audenaerde was responsible for the vision part and implemented a basic network to which I added the client component.

1.1 Important Terms and Definitions

The following paragraphs of this Chapter have the purpose of clarifying some fundamental terms which are repeatedly used in this document. These terms are not unambiguous, we therefore give definitions to ascertain our own comprehension.

1.1.1 Immersion

The latin term for "to submerge", commonly used for the state of being fully concentrated on a single enthralling activity.
The sensation of being immersed into a virtual world can be achieved by

stimulating the user's senses in a way that convinces him to be present in a seemingly real environment. Current virtual reality systems are able to stimulate the visual, acoustic and haptic sense.

1.1.2 Virtual Reality (VR)

In the late seventies, the Massachusetts Institute of Technology (MIT) developed the first head mounted display which was the first important step toward Virtual Reality. They define the term as a synthetic, computer generated, three-dimensional (3D) and immersive environment. By using a subset of the variety of developed input and output devices a user is able to get a feel for the three-dimensional shape of virtual objects and to interact with them. Nowadays, there are many subcategories of Virtual Reality (Desktop VR, Mixed Reality, Fish Tank VR, etc.). In this report the term stands for an immersive VR System.

Fish Tank Virtual Reality (FTVR)

The phrase *fish tank virtual reality* was used to describe a Canadian VR system reported in the 1993 ACM CHI proceedings (17). We define the term FTVR as a VR system which uses eye/head tracking to control the user's view position. By moving the head in front of the screen and looking at the real-time generated correct perspective, the user gets the impression of looking onto a three-dimensional object. This creates the illusion of sensing a real scene behind the display screens (e.g. a fish tank). Comparable to looking at a real fish tank and not being able to, for example, stroke the fishes, the user can not manipulate the objects he sees.

1.1.3 Usability

> The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component (10).

This definition clarifies that usablitiy is not an characteristic of the system itself but an attribute of the interaction of a user with this system within a certain context. The quality of this attribute might be different between different users.

1.2 Interfaces

Humans are communicating constantly: by speech, gestures, facial expressions and body language. We see and train these natural behaviour patterns since we are born. We do not even have to think about most of our interpersonal communication, it happens subconsciously. But since we are living in this age of a highly technologized environment, another aspect of interaction should be considered: The communication between humans and machines. Most of us are spending several hours each day in front of a PC, trying to accomplish the tasks we want to utilize the computer for. Almost every domain of human activity, scientific or recreational, involves using a computer. Being effective should therefore be an important goal for co-operation of machine and user. An important feature of effective communication is common language, it is in fact the strongest means of verbal communication (7). Since humans and machines do not only 'speak' but also 'function' differently, a translator is required. Responsible for the translation of our requests into the language of the computer is the interface which is defined as "the part of the design of a computer or other device or program that accepts commands from and returns information to the user" (8) .

Conventional Interfaces

If we look at common interface devices of our desktop PCs (keyboard, mouse, joystick, monitor), we find that most of them were invented and designed decades ago: The first mouse was built in 1962, the typewriter, which is the functional basis for modern keyboards, was invented in 1868. Compared to the rapid technical progress over the last 10 years, these devices are relicts of the infancy of home-computers. While technology and software gets more and more complex, the time we have to invest learning how to tap the new potential increases drastically. Interaction strategies that were once adequate to control simple applications do not meet the new possibilities anymore. The result is a decreasing effectivity and therefore usefulness of interactive systems. This problem is being addressed by researchers in the field of human computer interaction.

Human-Computer Interaction

Concepts called human-computer interaction (HCI), usability, human- or user-centered design became part of the main research fields in computer

science over the last few years. The focus is on the user, not technology, because, after all, he is the one that wants to make use of it, he is the one who gives it a right to exist.
The introduction of Virtual Reality increased the need for new ways of communication which should be closer to human-human interaction. The quality of a these system is often judged by the degree of immersion it can provide (9). The smaller the gap between the user's intentions and the actions necessary to input them in the computer ("Gulf of Execution"(4)), the easier the feeling of being integrated in a virtual world can be accomplished.
Arising input devices such as data gloves, 3D magnetic tracker and eye tracking try to achieve this naturalness by being direct (3). The user does not have to move the mouse across his desk to manipulate an object on his screen, he can just directly point at it.

But in this approach an important factor is neglected, which can be easily recognized by looking at Figure 1.1: These devices are highly intrusive. The user has to wear inconvenient gloves, markers or magnets which disturb the immersive experience.
Non-intrusive and direct interfaces can be achieved by making computers 'see' - using camera signals as an input.With computer vision it is possible to use almost every object as an input device, including the user's body. The challenge in this field is to find methods that enable him to use his intuitive (non-verbal) ways of communication. Existing and emerging technology is concerned with extracting movements of the head, arms, legs, the torso (5). A lot of research is going on in the field of gesture recognition (6) and facial expressions. Even so most of the results only work for certain settings (such as special lighting conditions and only one user at a time) and many approaches could not be integrated into practical applications or systems (1), yet, the topic of computer vision based HCI is a step toward the visionary concept of sensing computers that we can control by behaving naturally.

1.3 Perception of 3D Objects and Scenes

Humans developed a fascinating ability to perceive relative depth and, to a lesser extend, judge absolute distances. This important ability, evolved as a means of survival, is based on a weighted sum of several *depth cues* (26) which are listed below.
We need to consider these depth indicators in order to create a plausible virtual world. The more depth cues we provide the more realistic the illusion gets.

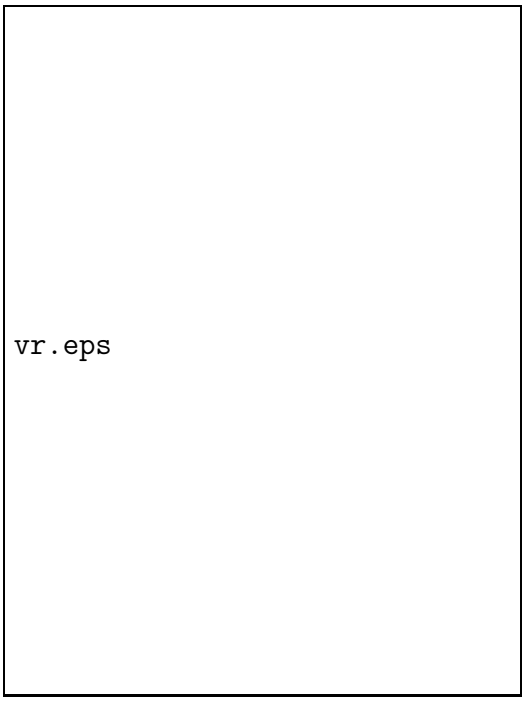


Figure 1.1: An example of input devices used in a virtual reality system. Courtesy of Ben Simons, Sydney VisLab, 2002

1.3.1 Physiological Depth Cues

The physiological depth cues are most important for depth perception as they are generated by a physical modification of the eyes and adjoining muscles.

Accommodation is the tension of the muscle that changes the focal length of the lens of eye. This is necessary to bring objects at different distances into focus. This depth cue operates only at short viewing distances (within 2 meters).

Convergence. To bring an object into focus that is placed close to the viewer, the eyes have to point slightly inward (compared to those at a greater distance). This also works only for short distances (within 10 meters).

Binocular parallax. As our eyes view objects from slightly different locations, the images projected onto the retina of each eye also differ a certain amount. This difference between correlating points in the projected images changes dependent on the distance to the object and is therefore an indicator for depth. It is in fact one of the strongest depth cues for small and medium distances. Even without any other cues we perceive depth using binocular parallax.

In the same way that our brain can calculate depth from the difference between two similar images seen by two eyes (binocular parallax), it can extract depth information from two similar images sensed after each other by one eye. This is called *motion parallax*.

1.3.2 Psychological Depth Cues

Psychological depth cues are constructed by our brains by using previous knowledge about the objects we are looking at. These cognitions are based on experience.

If we know the real size of an object or its relative size to the rest of the scene, the *retinal image size* gives a (rather vague) indication of the distance of the object.

Linear perspective can be detected e.g. when looking at parallel lines that seem to touch each other in the distance. The closer the lines at a particular

point, the further away this point is from the viewer.

Aerial perspective is caused by light being scattered dependent on the amount of atmosphere in between the viewer and the object. The result is a decreased contrast.

As soon as *occlusion* occurs in our scene we presume that the occluding object has a smaller distance to us than the occluded one. This helps us clarifying the relative order of the objects.

Shades and shadows are very important for depth perception. The Shading indicates the orientation of the object which can be used to relate it to other objects in a scene. Another depth hint results from the fact that lose objects tend to be brighter than the ones further away.
Objects casting shadows onto others tell us their position toward the light source. Taking this position into account we can determine depth information.

Chapter 2

Augmented Reality

We aim to build a VR system which adds to the current development in the area of human-centered interfaces. The system should provide an interaction which is easy to learn and to remember to ensure the effectiveness and attraction of the system.
The closer the implemented interaction of our interface is related to human-human interaction, the more satisfying it is to use it. Less effort has to be made getting familiar with the interaction before the user can actually start utilizing the system to achieve a goal. An increase in user-performance is an expected effect of the desired natural interface. Our research therefore concentrates on adopting human-human-interaction and human-environment-interaction mechanisms and developing applications to demonstrate the gained benefits. The research field of Augmented Reality provides a promising starting point for these purposes.

2.1 Theorie

The term Augmented Reality (AR) describes the idea of mixing virtual reality with reality to extend the possibilities of the real world instead of simulating it. One way of apply this concept is to project virtual objects onto physical scenes by using goggles or projectors. Another possibility, used in this project, is to include output devices or sensors into real world scenes or objects to provide additional information techniques (11).

The disciplines of virtual and augmented reality are not only concerned with creating virtual worlds, they also have to find solutions to interact with these worlds. Typical forms of interaction are navigation, selection of virtual objects and their manipulation and some kind of system control for more complex applications.

The most intuitive form of navigation and viewpoint control can be directly derived from our real-world experiences. We navigate by moving our body toward the aimed position and our field of view changes according to the orientation and position of our head. This can be done in AR and is already a fundamental technique. All we have to do is determine the position and orientation of the user and his head or preferrably his eyes. Basic manipulation of objects (e.g. grabbing, moving) can also be simulated by detecting hand positions and, for more advanced applications, can be supported by gestures recognition.
When dealing with direct interfaces, some kind of tracking is therefore indispensable. The following section introduces several approaches which address this task.

2.2 Existing tracking techniques

Mechanical trackers connect a reference point to the tracked object through a set of jointed linkages whose motion is measured. They have a large field of view and can yield a high accuracy and low latency. But their intrusiveness makes them cumbersome and they restrict the user's freedom of movement. There are only a few systems using mechanical tracking. One example is shown in figure 2.1.

Magnetic systems couple a transmitter producing magnetic fields and a receiver capable to determine the strength and angles of the fields. They are fairly accurate and have reasonable latency. A drawback is, however that almost all system use wires to connect the receivers to the system which results in a limited liberty of action. They also suffer from field distortion and electromagnetic interference. Metallic objects cause problems when placed close to the VR environment, which becomes an issue e.g. when demonstrating the system on conferences.

Standard *acoustic tracker systems* use three microphones and three emitters (for six degrees of freedom) to compute the distance (via time-of-flight) between a ultrasonic sound source and receiver via triangulation. The trackable area can be very large, compared to other systems (APR Inc. developed an ultrasonic "Gesture and Media System" (GAMS) which can cover up to 30*40 meters). The biggest disadvantage is the low accuracy due to noise interference, echoes and the varying speed of sound in different conditions of the air (e.g. temperature).

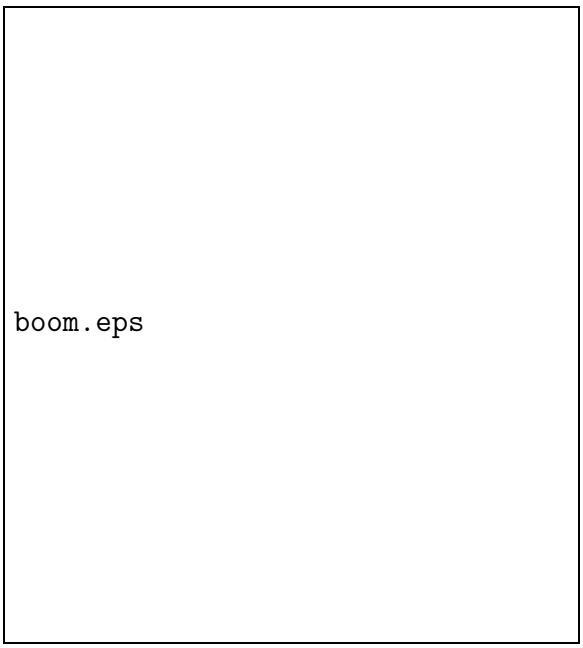


Figure 2.1: BOOM - Binocular Omni-Orientation Monitor (Fakespace)

Conventional *Optical Systems* determine the position of the user by tracking reflective markers under special lighting (often: infrared). They rely on vision algorithms to extract the location of targets from images captured by video cameras. This technique is often used for collecting body motion data which is needed in animations. Advantages are the comparative unrestricted freedom of movement, the small amount of noise caused by the cameras and the minimal encumbrances. Optical tracking, compared with the other types of tracking, is the most accurate technique for localisation. Unfortunately, most of these systems do not work in real-time and are very cost intensive.

Vision based tracking using natural features (no markers) is the least intrusive tracking method. Position and orientation of the user's body are extracted from 2D video camera data. A generalised statement about the accuracy cannot be made, the results depend on the used video cameras, lighting conditions, the computer vision algorithms and distance of the user to the cameras. The number of cameras and their lens coverage define the area which can be tracked, the technology itself does not set the limits.
While such systems have been implemented, practical and robust real time vision based tracking systems with a great potential for a variety of possibilities is not available, yet (22).

Assessment

All of the aforementioned techniques provide some advantages but also have drawbacks. The specific appliance and its requirements (accuracy, speed, etc.) governs the choice of one or a combination of several techniques.
Our emphasis is on developing a natural interface. From this follows that it should be preferably "invisible". We chose the only tracking variant that is non-intrusive: the one based on computer vision.
In addition to this requirement, we demand close to real-time reactivity and a relatively large area of movement. We intend to make the system as user- and background setting independent as possible as it is not only destined for research but also for presentation purposes.

Chapter 3

The Watching Window Framework

The setup and underlaying technology of the Watching Window have already gone through a few stages, the two most important steps in the development process are described in this chapter.

3.1 Overview

The *Watching Window* is a computer vision based virtual reality system that aims to develop, examine and utilize natural interaction between humans and computers to provide an intuitive interface.
It's name derives from the fact that it both watches it's users and provides a window into virtual worlds. To achieve a maximum of usability several requirements should be met or aspired.

- The perspective and, if used, the binocular parallax have to be convincing (low latency, high accuracy).
- It has to be robust: work stable and independent of the user's biological properties (such as skin/hair colour) and background conditions.
- The interaction should be as intuitive as possible.
- We aim for non-intrusive input devices.
- The system should be cost effective.

3.2 First Prototype

First research was done in 1997 by building and experimenting with "The Virtual Display Case" (19), a fish tank VR system. The idea was to give the user a clue about the three-dimensional shape of an object behind the screen by providing motion parallax. Similar to walking around a display case in a museum and getting a feel for the exhibit, he can change his head position and gets a response by seeing an updated perspective on the display.
Comparable functionality already existed at that time but could only be achieved by specialized and high priced hardware, whereas this project aimed for a cost effective solution.

3.2.1 Hardware

This set-up used only a standard computer and one standard video camera. This first version of the tracking system reduced the complexity of eye tracking by using easily trackable "glasses". These glasses had no lenses but were decorated with three coloured circles on a different coloured background. The circles were placed above the eyes and lay on a straight line. This system was accurate enough in simulating the perspective but for creating the desired illusion of an solid object being behind the screen, the latency was still too high ($\geq$ 80ms).
Later the single camera was replaced by two video cameras to provide a higher accuracy in the calculation of the 3D position. Since the usage of the mentioned spectacle frames should only be an interim solution, a new approach was made. By creating special lighting conditions (shown in Figure 3.1) and using a white sheet as a backdrop the silhouette of the user could be easily extracted.

3.2.2 Application

The application that utilized the new interface was a tool for modeling in 3D. Similar to a potter's wheel, a three-dimensional shape could be created by adjusting the with of a cylinder along its y-axis, see Figure 3.2. The hand positions were used for the modeling process, the head position defined the viewpoint from which the scene was viewed by the user. Due to binocular parallax, achieved by using red-blue glasses, the model seemed to float approximately half a meter in front of the screen.

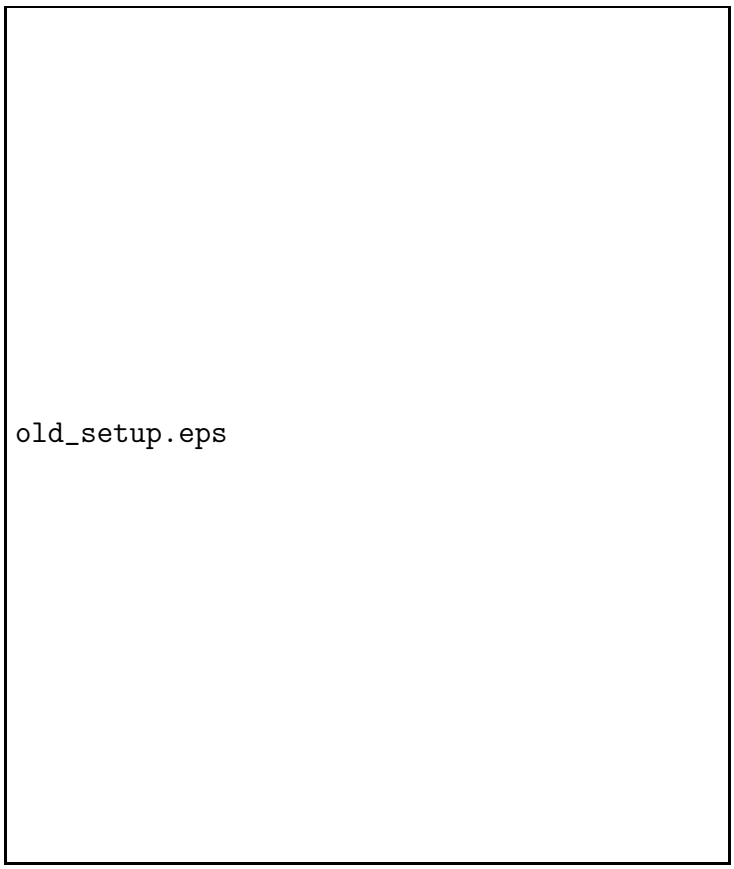


Figure 3.1: Setup of the Watching Window using background subtraction

3.2.3 Subsystems

Vision

The first set-up made use of special glasses with three spots in order to simplify the tracking process:
The first step was to locate the three spots by searching for pixels in a certain, predefined colour range. Then a flood filling algorithm was applied to localize the center of the three spots. The next step was to calculate the 3D position of the head by taking into account the known distance between the centers of the spots.
In the second set-up a silhouette was extracted and the head position was assumed as the topmost point of the silhouette minus a certain height which was dependent on the overall height of the silhouette.
Hand positions could also be found in this updated system. They were calculated as an average of the 50 foremost (closest to the screen) pixels.

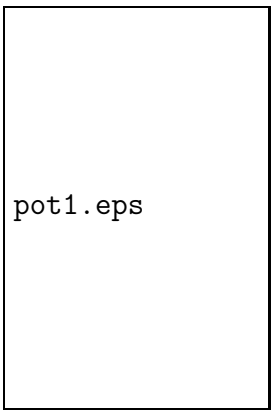


Figure 3.2: Example for a virtual pot modeled with the virtual potter's wheel

Graphics

The emphasis of the first attempt of this prototype was on developing an accurate tracking system so it used very basic graphics. A box was rendered and its changing perspective was compared to the one of a real box of the same size in the same x and z position.

The virtual potter's wheel application simulated a cylindrical deformation applied over a rotating cylinder. It used a blendeforming technique developed by Mason and Wyvill (15) for deforming objects based on local control points (see Figure 3.3). The deformation technique works by defining a deformation volume and squashing and stretching the space inside according to a user defined function. It compresses where the volume will be shifting into and expands the volume behind the deformation to keep it continuous. When the hand is near a control point the control point moves away from the hand position in a predefined direction (13).

3.2.4 Assessment

Even so this system was robust and stable enough for the purpose it was intended for: *an unencumbered interactive 3D VR model viewer* the feature extraction was still quite noisy. A Kalman Filter was used to smoothen the movements but this caused a small delay in the viewpoint updating process. Using sheets to create a uniform background simplified the silhouette extraction but to reduce the size of the system, a background independent solution should be found. The approach of assuming the head as the topmost point

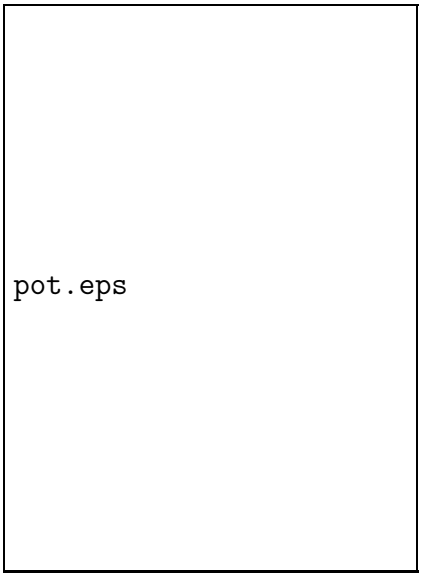


Figure 3.3: A bounding deformation mesh with control points deforming a cylinder.

and the hand as the furthermost point of the silhouette is not an ideal solution, as these assumptions may not always hold. As soon as the user moves one hand above his head or puts his hands behind his body, the system fails. The eye positions are only approximated in this approach, directly extracting the eye positions would lead to more accurate results. Shadows are part of the silhouette and can therefore lead to a wrong calculation of the body data. Generating the required lighting conditions is complicated as they can vary dependent on the weather and time of day.
Using red-blue glasses enhanced the 3D effect but reduces the quality of the rendered image, less colours can be seen. Better solutions could be to use polarised glasses with two different polarised projectors or shutter glasses.

3.3 Second Prototype

3.3.1 Hardware

In this section an overview of the hardware components of the second prototype is given. Figure 3.4 shows a sketch of the set-up.

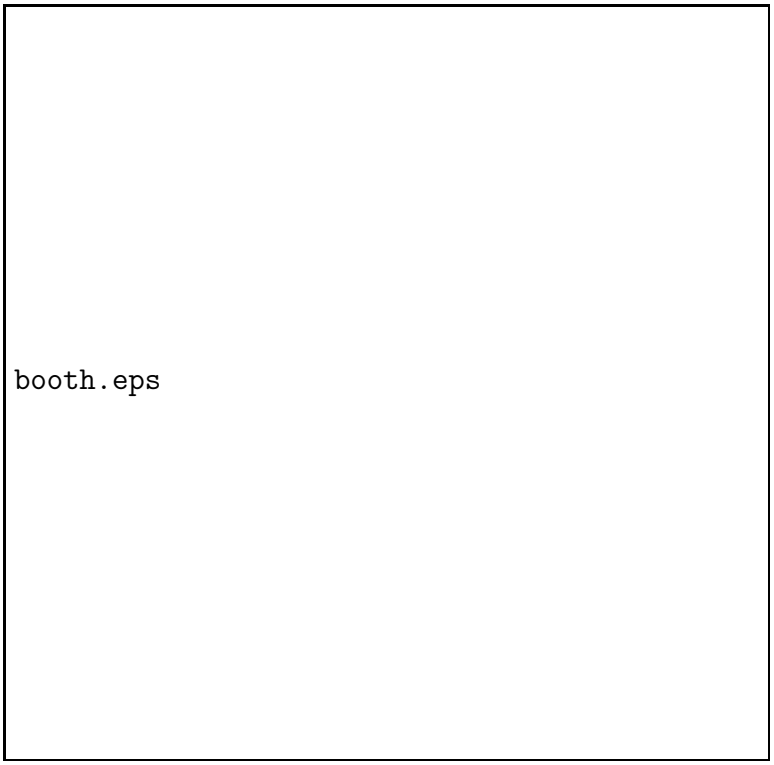


Figure 3.4: Current Setup of the Watching Window

Booth

The area of interaction is restricted by white wooden panels of approximately 2.5m height. On the ceiling we fixed 5 lamps to make the lighting as uniform as possible which simplifies the silhouette extraction in the vision process. The booth is divided into two sections. The smaller section contains two computers, a projector and a mirror that reflects the light of the projector onto a screen, which is located between these two sections. On both sides of the screen, at the same height, are small cameras pointing toward the middle of booth.
The user steps into the bigger part of the booth and as soon as he is in the view of the cameras, the vision system can begin to calculate required information, in our case the person's head and hand positions.

Cameras

We initially worked with simple video cameras that encoded the data to an interlaced analogue signal. This caused some serious problems because it distorted the image of a moving object and the calculation of the digital signal to an analogue signal and back to digital (to process the image) added a lot of noise. Another effect was that it decreased the frame rate (25 fps). To improve that situation, we switched to FireWire cameras that run at a higher frame rate (30 fps) and send a digital and non-interlaced signal. An advantage are their fish eye lenses whose viewports now cover a much bigger area of the booth so that the region, in which the user is seen by both cameras, is extended.

PCs

The machine that processes the video data and acts as the 3D server has two Pentium 500 processors. The other computer, which runs the graphics system, is a Pentium 3 930 MHz machine with 512 MB RAM and runs with a standard resolution of 1024 x 768 pixel. It is equipped with a new graphics card, a *GeForce FX 5900 Ultra*. Both machines run under Windows 2000, as there were no Unix drivers available for the TV cameras when the project was first initiated.

Assessment

Because of the large, heavy panels which make up the booth, this setup is not a perfect solution for demonstrations or exhibitions at various places (it

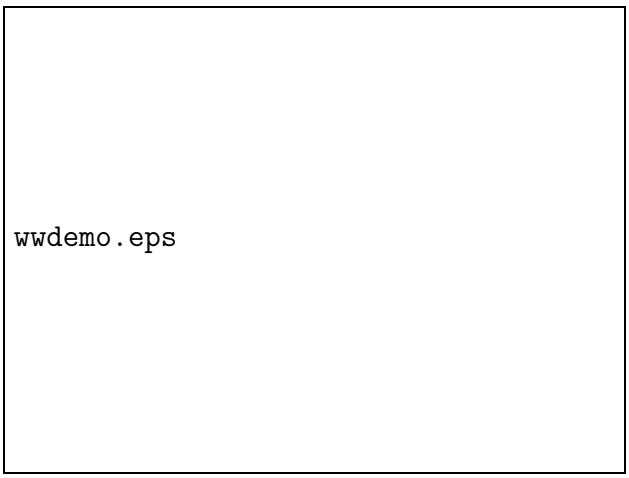


Figure 3.5: The Watching Window demonstrated at the Innovate Conference in Christchurch

took us 2 days to assemble it in the narrow Graphics Lab. To make it less dependent on the background, which is the only reason for the big booth, the vision system is being improved (see section 3.3.3).

3.3.2 Applications

Window Demo

The first demonstration shows a simple, static scene, consisting of a three-dimensional window frame and a flat background texture behind the window. The scene responses to the tracked head positions in oder to create the illusion of looking out of a real window. Moving along the x-axis of the screen (Figure 3.6 illustrates the coordinate system) results in a horizontal shift of the background scene. A movement along the z-axis suggest that the user can look at a bigger or smaller section of the scene. In both cases occurres a perspective distortion of the window frame. See Figure 3.7 for a screenshot. This relatively simple scene still impressed our visitors because of the accurate reactivity of the invisible interface they are controlling via movement.

Bell Demo

The second demonstration uses head and hand tracking. The scene consists of a closed room with a bell stand and three bells with different sizes. The height of the bell stand can be adjusted for different sized people. The

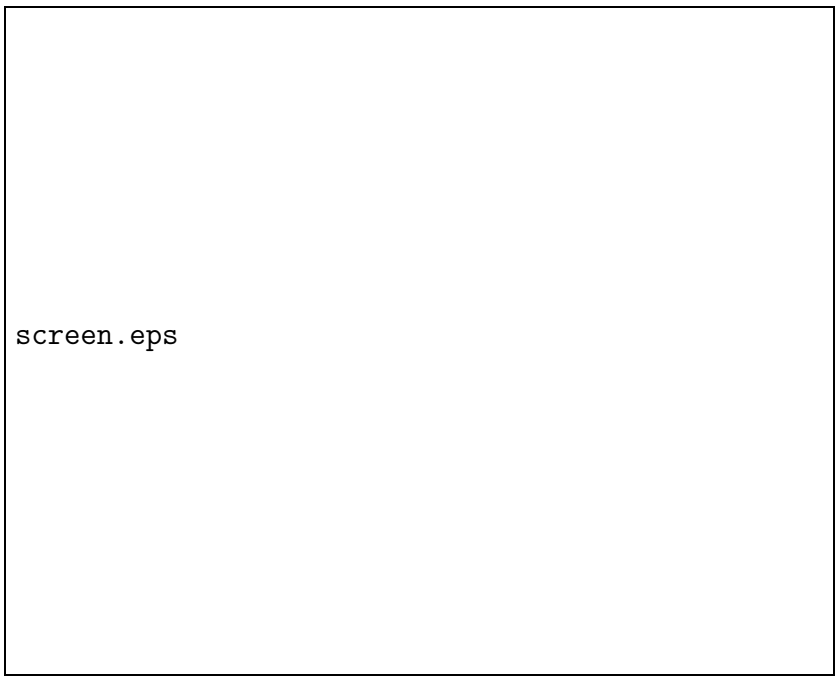


Figure 3.6: Coordinate system of the screen

user reaches out his hand and tries to "touch" the bells which are located approximately half a meter in front of the screen. After activating a bell by moving the hand to its 3D location, a ring sound gives acoustic feedback and the "touched" bell swings for a few seconds. The head movement is processed to control the view, just as in the window scene application. This demo can only be used with stereo vision and even then it is hard to judge where exactly the bells are. Most users just experimented with moving their hand along the z-axis, within the x- and y-axis range of the bell on the screen. See Figure 3.8 for a screenshot of the bell application.

Asteroid Game

This scene is situated in outer space. Asteroids and crystals fly toward the player. He tries to collect crystals while avoiding the asteroids. The hit test is done with a box around the user's head. Feedback is given by playing different sounds for the two different kinds of rocks and a vibration of the whole scene to simulate a short, sudden loss of balance. The goal is to collect as many crystals as possible in a short amount of time. At the end of this time or after being hit by a certain number of asteroids, the game is over and a feedback of the achieved points is given. See Figure 3.9 for a screenshot.

Figure 3.7: Screenshot of the window application

This application provides the most fun to interact with. The user has to duck and walk or jump around the booth. That is why this game is not only fun to play but also fun to watch for other visitors.

3.3.3 Subsystems

As the updated version of the Watching Window was created under a pressure of time, all of the components of the system were build only with respect to how fast and easy they were to implement. Programming concepts like maintainability, extendability, readability, fault tolerance and reusability (20) were neglected. The structure was inconsistent with concepts of object oriented programming like modularisation, usage of classes (encapsulation of attributes and methods) and information hiding. Many methods and functions were dispensable, no documentation was available and useful comments were rare.
With the start of this project a new development phase began and we decided to revise and restructure all of these components before starting to extend the subsystems.

Figure 3.8: Screenshot of the Bells Application

Graphics

Our three applications had a nice way of demonstrating the correlation of the rendered image with the head position of the user. Using these head information for other purposes than adjusting the view point would downgrade the intuitiveness of the interaction.
The bells demonstration showed a basic manipulation of the scene by using the hand position. But as we worked on an improved version of the hand tracking, a new application should get developed that taps the full potential of this data. After making the old graphics code more reliable and building a solid structure the new demo application could get designed and implemented. We agreed upon building a painting application which uses head tracking in the same fashion as for the other demonstrations and single hand tracking to draw virtual paint on the screen. This functionality fit well with the data provided from the vision system. The position extraction offers accurate enough informations about the position of the hand but was at that state not fast and robust enough to provide more specific data e.g. about the velocity of the movement. A complete description of the development process, implementation and features of the new application is given in Chapter

Figure 3.9: Screenshot of the Asteroid Game

4.

Application Architecture Upgrade After analysing the code and removing dispensable methods, the code was made portable by replacing Windows commands with much shorter and easier to read glut commands. The methods replaced were *WndProc*,*CreateGLWindow* and *WinMain* with a total amount of 425 lines. The equivalent glut methods summed up to only about 170 lines. The old program was equipped with a Kalman Filter on both the vision and the graphics side. This filter predicted the future position of tracked points. This can lead to a more robust tracking system and a smaller latency but at the same time makes the tracking less sensitive toward abrupt changes in position, direction and speed of the movement. As we wanted to experiment with a new tracking system and test its accuracy we decided to remove all of these expensive calculations of the filter code. Another reason for the removal was that the filter code was lengthy and hard to comprehend but we wanted to base our work on a completely familiar code.

Afterward a new structure was designed and object-oriented reorganisation took place, building classes for each application and separating different concerns from each other like network functionality, rendering, calculation, data storage, texture loading and control. The program structure is illustrated in Figure 3.10.
As the program is written in C++, the so-called const correctness (23) was considered. Useful comments in Doxygen-syntax were added which means that they can also be used to automatically generate a documentation, e.g. in html-format.

More functionality was also added to the old system:

- A stereo vision (red/green-) effect can be switched on or off by using a right-click-menu or the keyboard.
- The former situation of cycling through the different demo applications by pressing a key was improved by assigning one key per application and making them directly selectable.
- It is still possible to cycle through the demonstrations by using a timer but this process can now be paused by pressing a key.
- The applications got a reset key.

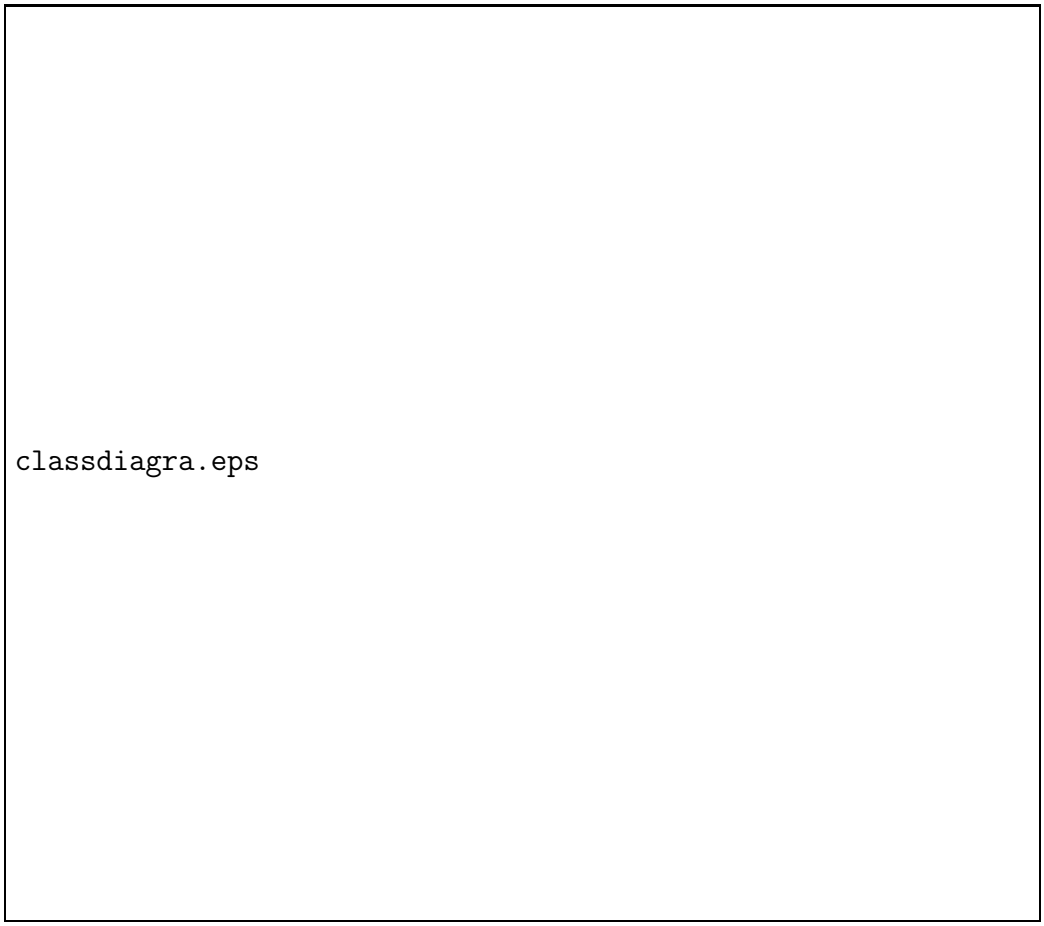


Figure 3.10: class diagram of the graphics system

- We included the functionality of loading *.png-files as textures. These are an advantage over *.bmp-files, which the old version used, as they have an alpha channel.
- The speed of the rendering process was increased by using display lists.

As the work on the tracking system was still under construction and some coordinates were needed for testing purposes, a method to interact by using mouse positions was implemented.

Network

Vision and graphics system are running on separate machines which are connected through a network. The initial network implementation was neither reliable, fast enough nor extendable. It also often caused both the graphic and vision application to crash without throwing error messages. The most serious drawbacks were (21):

- Only fixed number of two cameras were supported.
- All of the cameras had to be attached to the same machine.
- The network connection code was unstable and error-prone.
- The method used to transfer coordinates contained a lot of overhead because of an attempted time synchronization which was unnecessary.

Instead of spending a lot of time analysing and structuring this code, we decided to completely replace it by a new and more stable architecture. The new implementation is now easily extendable to allow an arbitrary number of cameras and multiple displays.
We revised the network implementing a server that receives 2D coordinates from one or more vision machines (cameras), calculates 3D positions out of this data and sends this 3D information upon request to the display clients. See Figure 3.11 for a diagram of the network.

Tasks of the Camera Client

- Stores camera calibration data and sends it upon request.
- Processes the video data (searches for head and hands) and forwards the results to the 3D Server.

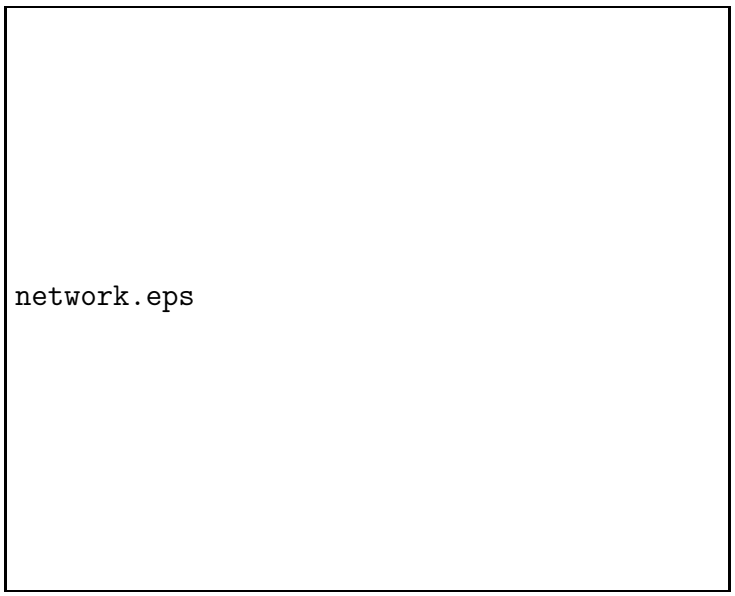


Figure 3.11: The revised network architecture

Tasks of the 3D Server

- Administrates Camera- and Display- Client.
- Calculates 3D positions from the data sent by each camera (needs data from at least 2 cameras).
- Sends results of 3D calculation to Display Client when requested.

Tasks of the Display Client

- Stores the information about its own location in world space.
- Retrieves 3D locations from 3D Server.
- Uses this information as input for demo applications.

Vision

The vision system is responsible for finding interesting pixels of frames captured by the video cameras and calculating three-dimensional positions of predefined points of interest. In order to do so, two steps have to be performed. The first step, the localization, is applied when the specific search point is not know in the previous frame. In this step all of the pixels of the

frame have to be taken into account. Once we found what we were looking for, the tracking process starts. This can be done via template matching within a restricted area of the frames.
A future goal for the system is to be independent of special background conditions. For that purpose, several filtering methods were implemented and analysed. These tested filters extracted movement, performed edge detection, found skin colour, enhanced the contrast, did fast blur and searched for connected pixels. We found not all of them were suitable for our task and only used a subset for our system. The ones which are integrated in the vision system are explained below, a complete description of the filtering methods and their results can be found in the report of R. Audenaerde (21).

Filters

Movement The necessity of this filter is based on the assumption that it is only necessary to update the position of body parts when movement occurs. We want to use this filter for locatisation but calculating the movement without knowing the coordinates of correlating pixels in two successive frames is not possible, in the conventional way of describing movement as the distance between these correlating pixels. We therefore define movement (m) as the maximum of difference in intensity (i) in each colour channel of each separate pixel of frame t.

$$\mathrm{m(t)} = \max\left(\left|i_{red}(t) - i_{red}(t-1)\right|, \left|i_{green}(t) - i_{green}(t-1)\right|, \left|i_{blue}(t) - i_{blue}(t-1)\right|\right)$$

Afterwards a threshold operation is performed to filter out noise and small changes.

Skin colour Studies have shown the skin colour can be modelled as a single Gaussian distribution (24). We use this knowledge to evaluate if found pixels are likely to be part of the user's body.

Silhouette detection For each interesting (here: moving) pixel the neighbors are regarded: if at least two of them are of interest, the algorithm will define a 16 by 16 area around the pixel as part of the silhouette. Afterwards, a quick horizontal scan is done in order to fill small gaps still between these areas.

Temporal Texture In order to track a found region of interest, this part of the image is stored as a *temporal texture*. To make the tracking process more stable in regard to sudden and short changes, this texture is not completely replaced by the image of the same region in the next frame but slowly updated:
$newTexture = \alpha * foundRegion + (1 - \alpha) * oldTexture$

TipTable The TipTable is a class for holding all the interesting information that is found during the image processing. It stores data like region of interest, the head bounding box, a head center, temporal textures. This list can be extended to fit other algorithms. This information can be used in the next image processing iteration, so the image can be analysed more effectively.
Currently there is just one method used to determine the position of the head and the hand. In advanced versions, more algorithms may run in parallel and then all the information in the TipTable can be used to perform an average, or a voting algorithm.

Localization algorithm

1. Find moving pixels
2. Build a silhouette
3. Defining the head position by using the top down edge walk applied to found silhouette (25)
4. Check if the head has eligible head-size
5. Check if the head is skin coloured
6. If all checks are OK, assume the head is found; create a temporal texture and update the TipTable

Tracking algorithm

1. Start to match the texture at the old position of the head
2. Try to match it to the neighbors, in a spiraling way
3. Stop after a certain number of searches
4. Consider the highest match:

(a) If bigger than a threshold

   i. Make sure there is no movement above the found match
   ii. Make sure the found match contains enough skin colour
   iii. Update the new position to be the position of this match
   iv. Update the temporal texture to the new found match

(b) If bigger than a threshold

   i. If smaller, consider the head "not found anymore" and start localizing algorithm

Chapter 4

Design and Implementation of a Painting Application for the Watching Window Framework

The reason for creating an additional demo application was to show one of the possible ways of manipulating a virtual scene by using hand tracking. At the same time we wanted to experiment with depth perception with and without using stereo vision and other depth cues. For demonstration purposes the interaction has to be relatively simple and comprehendable as there is only a short amount of time available per visitor. We chose an application that encourages creativity and enables the user to leave a trace of his presence in this virtual world. The new demo application is a tool for drawing virtual strokes on a virtual canvas which can be either static or continuously moving along the z-axis. The last alternative results in the creation of a three-dimensional painting (an example is given in Figure 4.1). The head position will control the angle of sight on the scene in the same manner as in the other demonstrations.

4.1 Usage Scenario

The user enters the booth and looks into a 3D screen sized box which is changing perspective according to his head movement. He sees a few real, coloured pots and one (or more) brushes. The user selects a brush which will be recognized and tracked by the vision system and dips it into one or more of the colour pots. Depending on the length of time the selected brush stays in the pot, the paint colour is adjusted and applied to the current brush texture. In front of the screen there is an area in which his hand positions

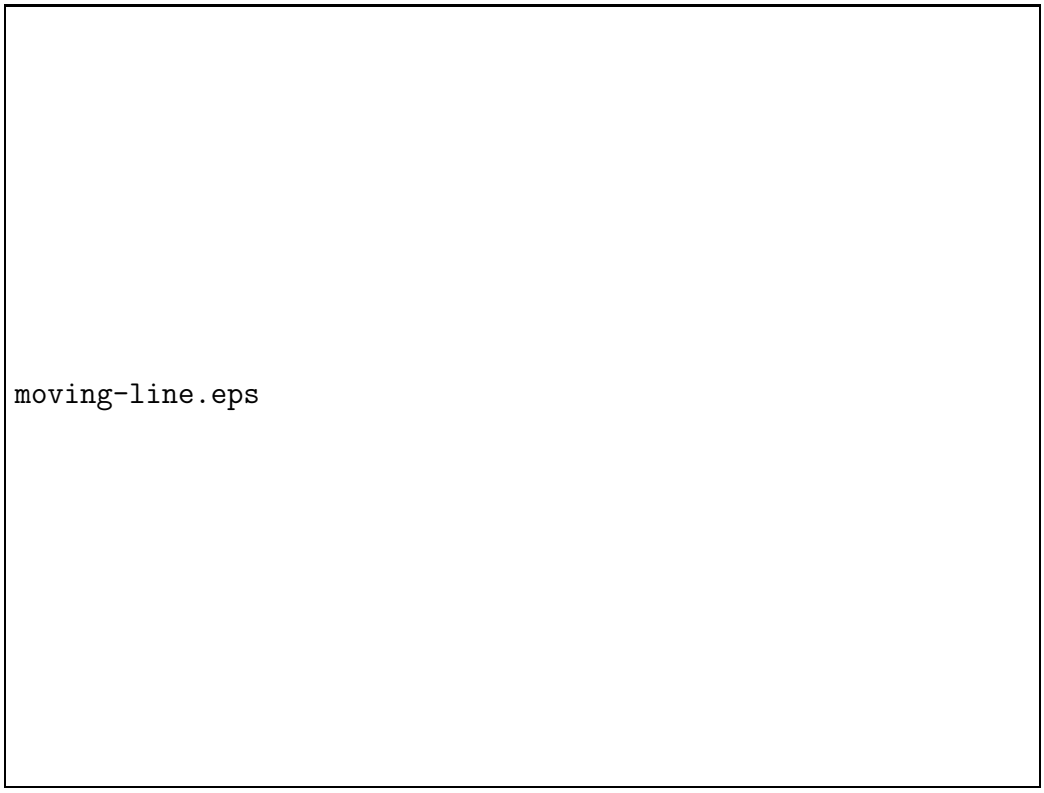


Figure 4.1: Drawing with continuous movement

are accepted. It has the width and height of the canvas on the screen and a certain depth, starting on the screen. Every time the tip of the brush is within this range of this box, forming the virtual canvas, the hand positions are used to draw lines with the selected colour. The saturation of this colour will be reduced to simulate a real brush losing ink with every stroke. As soon as the brush leaves the virtual canvas in z direction, the current line is finished. As the virtual painting continues over time the drawing turns into a 3D picture which can be viewed from different angles.

4.2 Design

The first prototype of a basic painting application was designed to show the correlation between the received data and the graphical output. It simply connected the tracked hand positions to a line in a predefined colour. The quality of this attempt was enough to show the responsiveness and the latency of the vision system and network connection but the visual appearance was unsatisfying because the lines appeared very flat and with an increased line width corner artifacts became visible as shown in Figure 4.2.

The idea behind the second line design is to calculate a quad out of two received hand positions and apply a texture to this quad. By choosing an appropriate texture and using the alpha channel of that texture a brush stroke with fading edges, different widths and different brush types can be simulated. One first approach of the calculation of these textured quads is shown in Figure 4.3. Certain problems arose in the process of calculating the four vertices of the quad. These problems and the different attempts to solve them are described in section 4.3.

4.3 Implementation

The initial version of the painting application, the simple line drawing version, used the OpenGL primitive GL_LINES to connect the received hand positions. All of the received positions were stored in a vector.

The points of the quads of the second line design were calculated as follows:
All calculations are done in two-dimensional space because the paint is applied onto the virtual canvas which has a fixed z-position. As soon as the

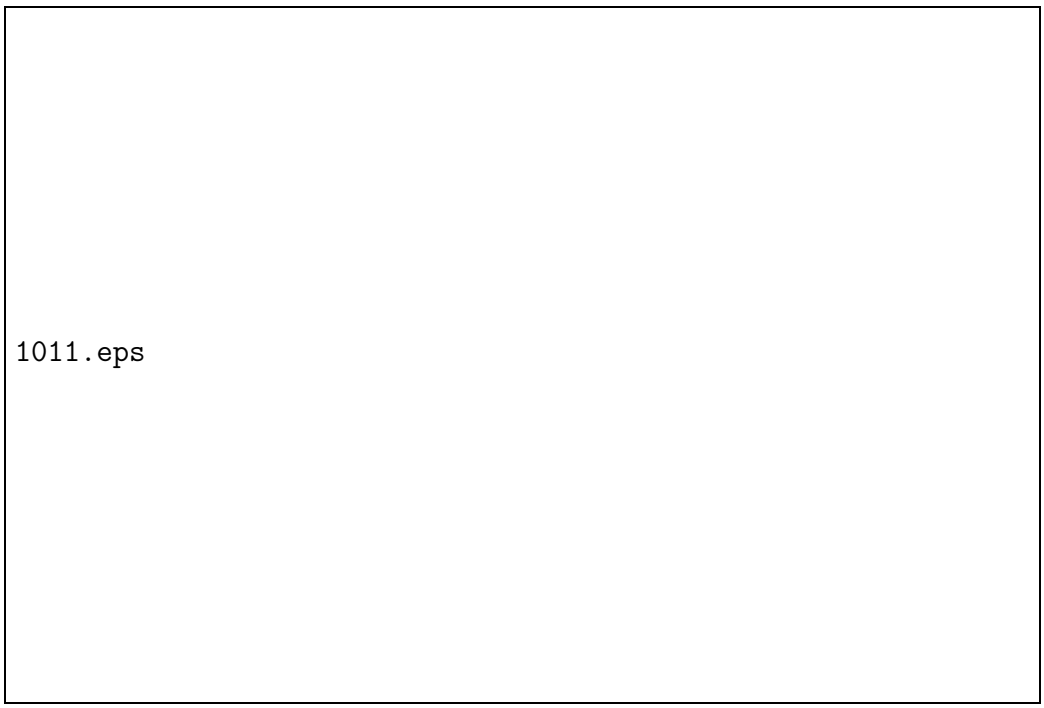


Figure 4.2: The first experimental prototype of the painting application

second hand position is received, the normal of the vector between the first and the second hand position is calculated and the new quad positions are the sum of the stored hand position plus or minus the normal, times a constant describing the brush size.
A problem arose when the angle between two neighboring vectors was more than 90 degrees so that the texture coordinates were in a wrong order. The first solution to this problem was to average the successing normals, as illustrated in Figure 4.4.

This attempt led to the effect that the wider the angle got the smaller the with of the quad was . Increasing the length of the averaged normal to keep the original width would have distorted the quad too much for extreme angles.

The next approach to avoid a twist in the texture was to simply invert the direction of the normal so that the left quad point of the first vector (L) is connected to the point that would have actually been the new right quad point (L'), R is connected to R'. The flag 'invertNormal' which stores the information whether the normal has to get inverted (multiplied with -1) is switched every time the angle is greater than 90 degrees. This method is

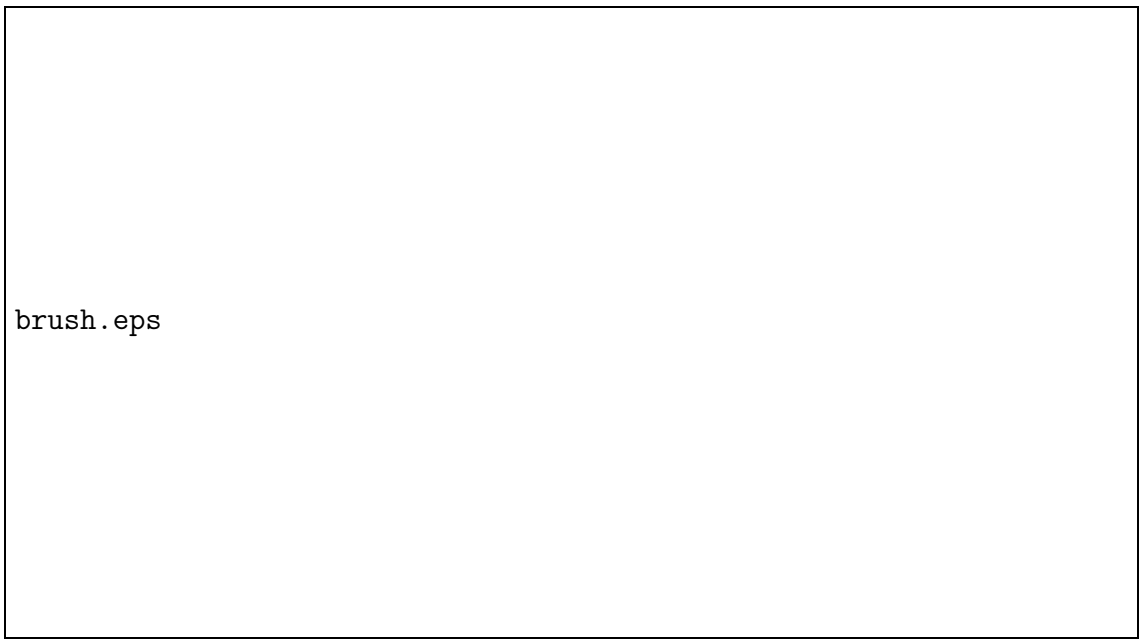


Figure 4.3: First design of drawing lines by constructing textured quads around received hand positions

illustrated in Figure 4.5.

4.3.1 Storage

The storage of the drawn lines is an important issue as it plays an essential role in terms of performance. Before the decision was made that the paint was going to be applied to a canvas (plane) and continuously moved backwards instead of moving a complete picture in discrete time intervals, the idea was to store the paintings as a texture and applying that texture to a quad. This seemed to be a logical solution as these paintings would not change and are nothing else than a picture on a texture. I would have avoided rendering old lines from the list of points every time the scene changed. This approach would have included rendering parts of the scene in a pixel buffer and binding the buffer on a quad as a texture. While attempting to implement this idea, we decided to add the functionality of a continuously moving canvas. This made the solution of rendering to a texture as a storage method impossible as the drawing could now become three-dimensional.
The new approach was to store all of the hand positions and calculated texture-quad positions in a vector before realizing that only the last 3 hand positions were needed to do the necessary calculations. The storage of the

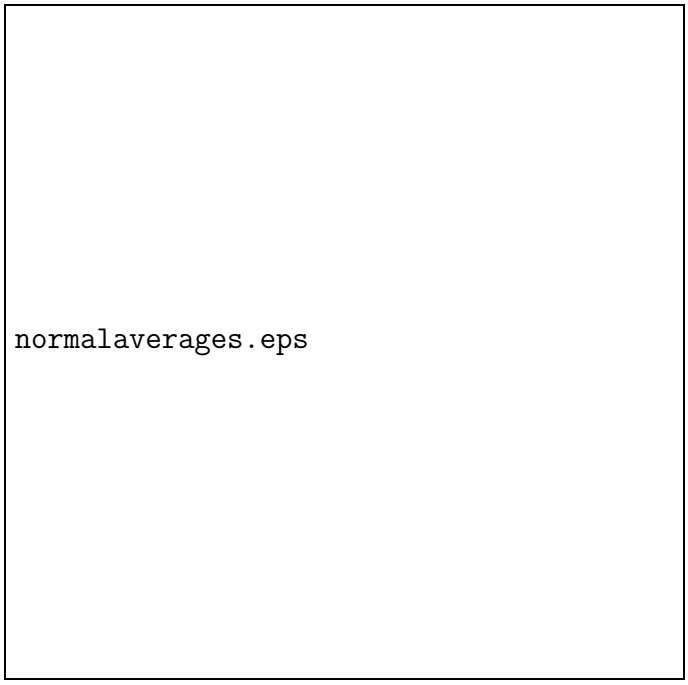


Figure 4.4: Quads points defined by adding an averaged normal vector (red) to a received hand position

Figure 4.5: Quad points defined by adding an inverted normal vector to a received hand position

hand positions of type "Point3d" was changed to use a double ended queue from the standard library as this container is more efficiently implemented. It provides front and back operations in a time of O(1).
After receiving a new hand position the according texture quad points are calculate and stored in the "QuadPoint" structure. As painted strokes do not change after they are complete, an effective way to store these lines is in a display list provided by the OpenGl library. This solution increases the speed of the rendering process and reduces the code for drawing all of the finished painted stokes to a few lines because a complete line can be rendered by calling the according list. These display lists are stored in a double ended queue of type GLuint.

The active and still uncompleted line is stored in a vector and has the type "QuadPoint". A line is terminated after the hand left the canvas or the brush type changed. As soon as the hand leaves the canvas along the x- or y-axis, the intersection with the canvas border is calculated and this point is added to the end of the line before finishing it.

4.3.2 Movement

The movement of the drawn picture is an essential element in the painting application as it is the main difference to common drawing programs. It transforms the 2D picture into 3D space. Changing the angle of view by moving the head helps looking at lines that are hidden behind others.
There are two different types of movement: Continuous and discrete. Drawing onto a 2D plane and moving the whole plane back can be compared to drawing onto layers. To emphasis that a picture was painted in a plane, a frame is drawn around it. Continuous movement adds another dimension while painting. For example drawing a circle on a moving canvas results in a spring-like shape, see Figure 4.1.

4.3.3 Textures

The "TextureHandler" was updated to be capable of reading ".png" files in addition to the functionality of reading bitmap. The new format is has an additional alpha channel in contrast to the bitmap format. Reading textures with alpha channel was necessary to provide different textures for the brush that have a smooth border. By enabling the alpha test, OpenGL discards texels that have an alpha value smaller than a predefined threshold.

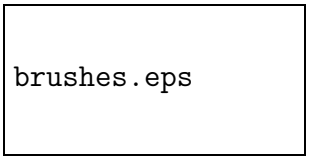


Figure 4.6: Currently used brush textures

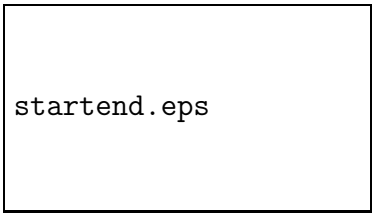


Figure 4.7: Start and end piece of line

The brush textures are only used to store alpha values which are represented by the gray value. To assign the current brush colour to these textures, the texture environment has to be set to GL_BLEND.
A texture for the first and last brush quad was designed to avoid sharp edges and simulate a real start and end of a brush line, see Figure 4.7.

4.3.4 Colour

To choose a colour for drawing the user dips his brush into pots. These pots represent the three colours red, green and blue. The selected colour is combined with the current one by increasing the according value and reducing the other colour values. The longer the brush is in one pot, the more of this colour is assigned. The amount of paint decreases while the brush is in use to simulate a loss of ink while painting. A fourth pot stands for rainbow mode which means that the colour values cycle smoothly through the seven rainbow colours by using linear interpolation.

4.3.5 Design Features

The final state of the visual appearance of the painting application can be viewed in Figure 4.8. To give a feedback of the current colour, a 'feedback-brush' was added to the screen whose tip represents the colour that the drawn line will have. To achieve this effect, the handle of the brush is drawn with the texture environment set to GL_REPLACE (because this part is never influenced by the painting colour) whereas the tip texture must be combined with the GL-colour by using GL_BLEND. For performance reasons not every

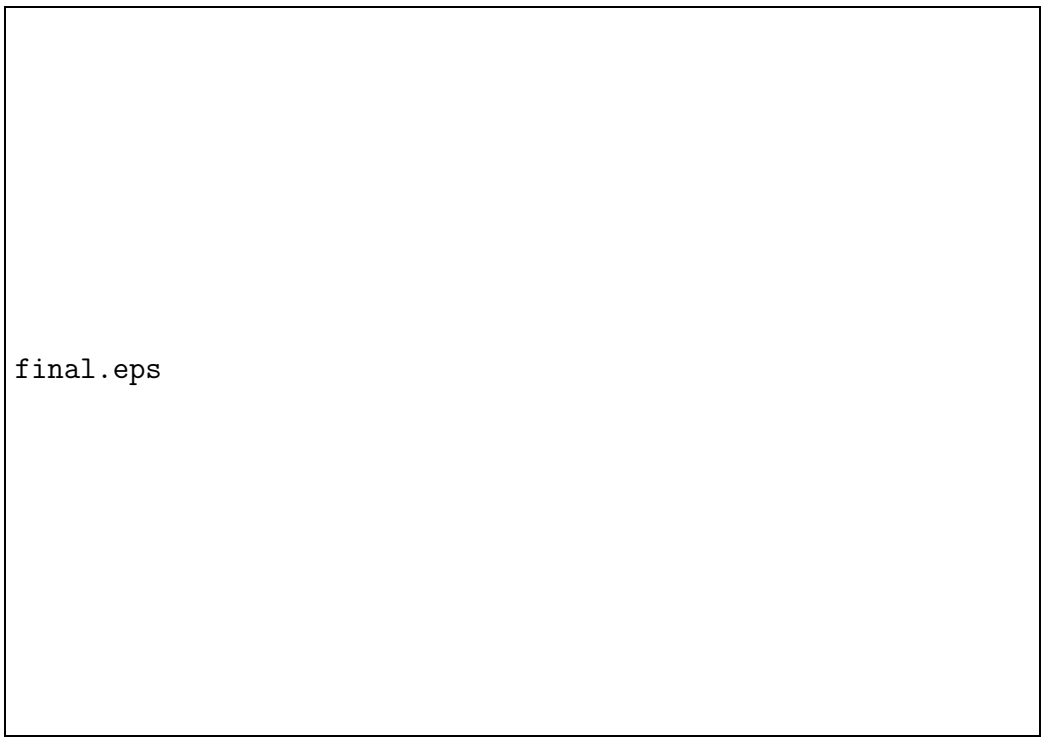


Figure 4.8: Current design of the painting application

line that was ever draw is stored. Lines that moved out of the visible range of the scene are deleted. The length of the room defines this range. To avoid the effect that a deleted line suddenly disappears, fog was added to the scene in the same colour as the back plane of the room. This leads to a smooth fading effect. Lines are deleted as soon as they seem to be completely dissolved. The coloured boxes below the canvas and the buttons on the left are just placeholders. The colour is going to be selected by dipping a brush into a real pot but these have not been put into place, yet.

A class description of the current implementation can be found in the appendix.

4.4 Depth perception

By stimulating as many depth cues as possible we try to simulate the perception of a three-dimensional world behind the screen. The depth cues used in this application are listed below:

- Linear perspective:

The textures on the walls of the painting room show a pattern of horizontal, parallel lines. Because of the perspective distortion the distance between these lines gets smaller in the distance which is a clear depth indicator.

- Aerial perspective:
  This effect is simulated using fog inside the painting area. The further away the lines are, the more fog is visible.

- Occlusion:
  Lines occlude each other as they move along the z-axis. When changing the head position the degree of occlusion changes as well as their relative positions which leads to:

- Motion parallax

- Binocular parallax:
  By generating one image of the scene per eye and wearing red-green-glasses we can serve this depth cue. But as this approach restricts the colour range and we aim for a system that works without any unnecessary devices, we try achieve enough depth perception without having to use glasses.

4.5 Feature List

- Drawing two-dimensional strokes onto a virtual canvas
- Discrete and continuous movement of the strokes
- Provision of several textures for different brush types
- Selection of colour for the paint by mixing red, green and blue
- Feedback of the current colour
- Fading paint
- Rainbow colour mode
- Functionality of clearing the complete canvas
- Selection of stereo vision with red-green-separation

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We implemented a cost effective VR system that provides intuitive, non-intrusive interactivity with a virtual scene. Being robust it is a good base for future research in the area of natural interfaces.

The vision system does not depend on background subtraction anymore but has not been tested with changing background conditions, yet. Becoming completely background independent is therefore an objective we could not achieve, yet. Even though we attempted to find a way of extracting the eye positions, all of our algorithms did not work stable and precise enough. We therefore had to abandon the plan during this project and concentrated on head and hand tracking.
The head tracking works more stable than the hand tracking as the head usually does not perform sudden, fast movements. The hand tracking has to get improved to provide satisfying results. Especially the coordinate changes along the x-axis are not precisely recognised. Adding a third camera which is fixed on top of the screen and in its center, looking toward the user's face would simplify this problem and the process of determining the distance of head and hand toward the screen. A head tracking that finds the positions of the eyes would be more accurate and is a recommended change for future revisions. A small latency can be noticed. This problem can not entirely be solved without using a predictive element.

The developed and improved applications still offer very basic reactivity and manipulation possibilities but often show an accurate correlation to the input.

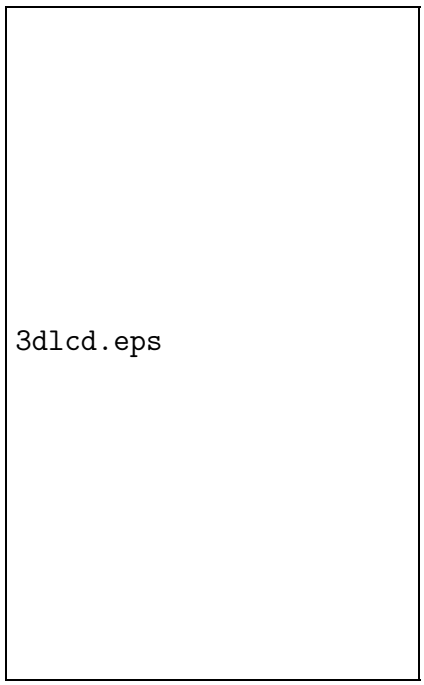


Figure 5.1: 3D LCD display for stereo vision

The painting application is not well tested, yet. At the end of this project the cameras were being replaced and the vision system updated so that this was not possible. But earlier versions of the demonstration responded well to the received hand positions as long as the movements were not too abrupt and within an area close to the center. Outside this area, an inaccuracy due to calibration errors could be noticed. When holding the hand still, the drawn stroke sometimes formed a zig-zag line because of noise in the video signal. A filter should be used to avoid this problem.
The binocular parallax using glasses can not be considered a final solution because of the restricted colour range and obtrusiveness. A better way of separating the two images for each eye would be to use an 3D LCD display as illustrated in Figure 5.1.

Personal Remarks

I enjoyed my stay at the University of Otago, the working climate was great and it was fun to be part of both the work and social aspect of the graphics community. Every week we had a Watching Window meeting with Geoff, Robert, Jayson and me. We talked about our progress, asked for advice,

fixed tasks for the next week and searched for ideas to improve the hardware set-up.
I learned a lot. I am now more experienced in using OpenGl and more confident with C++, improved my programming style and learned how to apply program- design rules (thanks to Rob and Phil for helping me with my attempts).

5.2 Future Work

- The vision system should be adjusted to precicely recognise the brush we bought. The current state is that it is still the hand which is tracked.
- The stereo separation is implemented by drawing the scene twice, first the red and then the green channel on top of it. OpenGL offers a nicer solution by using the right and left buffer for stereo colours.
- The user can see the border of the background texture when he steps out of one camera's view or walks to far to either of the sides. This could be fixed by sending positions only when the user is in the view of both cameras and clipping the positions to a certain range.
- The vision system sometimes sends packages with wrong data such as very small numbers for each of the values. We currently check for these numbers at the display client side after receiving the data but they should be filtered out on the vision side.
- The game mode of the Asteroid Demo should be extended. The level of difficulty stays the same, what makes it boring after a while. Here are some suggestions for variations:
  - According to how many crystals the user collects, the speed increases.
  - The size of crystals and asteroids could be changed depending on the user's skills.
  - The number of crystals decreases over time while the number of asteroids stays the same.
- The network code has almost no error handling and no method for dis- and reconnecting, yet. Occasionally it seems to slow down the graphics application. I did not have enough time to search for the cause of that. I assume the client waits for the server to send data and does not do any rendering during the waiting time.

Recommendations for improving the painting application:

- Replacing the quads that act as placeholders for selecting a colour with real pots.
- Designing nicer buttons to toggle between the continuous and concrete movement.
- Implementing an 'Undo'-functionality to delete the last drawn line.
- Implementing a rubber to delete parts of already drawn lines.
- Designing a mechanism to control the position of the drawn strokes. Currently the lines just float away and disappear. If the user had some sort of navigation to move the scene along the z-axis, more complex paintings could get created.
- The lines are currently applied to a virtual two-dimensional canvas and are therefore two-dimensional as well. But as they exist inside a three-dimensional scene and can be viewed from different angles (e.g. from the side) it would be an improvement to be able to create three-dimensional strokes in this virtual space. One attempt could be to render cylinders instead of textured quads.

Appendix A

Class Descriptions

BaseApp
This class contains basic functions and variables for the demo applications such as the 3D positions of hand and head (Figure A.1).

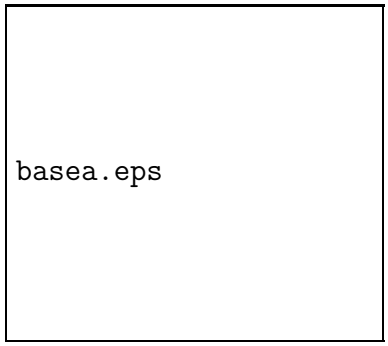


Figure A.1: Important Methods and Variables of Class BaseApp

Timer
Contains basic time methods for getting the time in milliseconds and storing a start time. This class is used in the main interface to decide if it's time to switch to the next application and for the game mode in the asteroid demo.

TextureHandler
Is used to load and store png- and bmp- textures.

DisplayClient
Connects to the 3D Server and requests data.

Figure A.2: Simplified Colour Class

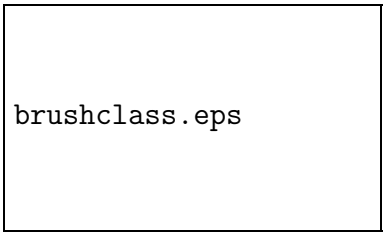


Figure A.3: Class Brush

ClientInterface
Interface between main program and DisplayClient. Initialises the connection and copies received data into the hand- and head- position structure that is used by the program.

Body3D
Data structure used in the vision system and to transmit information over the network
A complete html documentation can be found on the Watching Window graphics machine and on the report CD-R.

subsectionBasic classes I constructed a class 'Color', that stores the values for a RGBA colour and has methods to change these values, see figure A.2.

The Brush class (figure A.3) has data about the type of the brush, its size and how often it has to be repeated on a single line segment. This is dependant on the length of this segment and is only necessary if the brush texture is not symmetrical.

The class QuadPoint (figure A.4) is used to store points of the line, which is currently 'active', in other words, points are still being added to this line.

The elements of this structure have data about their 2D position in screen space the z- value is given by the z- value of the canvas. This class also stores

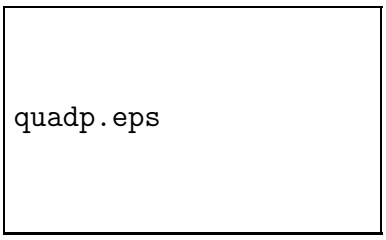


Figure A.4: Class QuadPoint

information about the type of brush is used and which colour this part of the line has.

Bibliography

[1] Roberto Cipolla, Alex Pentlax, *Computer Vision for Human-Machine Interaction* (Cambridge University Press, 1998).

[2] Lindsay MacDonald, Joe Vince, *Interaction with Virtual Environments* (John Wiley & Sons, 1994).

[3] B. Shneiderman, *Direct Manipulation: A Step Beyond Programming Languages* (IEEE Computer, 1983).

[4] E.L. Hutchins, J.D. Hollan, D.A. Norman, "Direct Manipulation Interfaces" in *User Centered Design: New Perspectives on Human-Computer Interaction,* (1986).

[5] Thomas B. Moeslund, Erik Granum, *Visual Interpretation of Hand Gestures for Humas-Computer Interaction: A review* (1997).

[6] Vladimir I. Pavlović, Rajeev Sharma, Thomas S. Huang, *A survey of Computer Vision-Based Human Motion Capture* (Moeslund, Granum 2001).

[7] National Institute of Open Schooling, Communication: Basic Concepts, *http://www.nos.org/secpsycour/unit-18.pdf.*

[8] Encarta Dictionary, Definition 'User Interface', *http://encarta.msn.com/dictionary_1861708850/user_interface.html(date: 23.03.2004).*

[9] Tom Hayward, *Adventures in Virtual Reality,* (Que Cooperation, 1993).

[10] Institute of Electrical and Electronics Engineers, *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries,* (New York, NY: 1990).

[11] Bederson, B.B., A. Druin, *Computer-Augmented Environments: New Places to Learn, Work, and Play". In: J. Nielsen. Advances in Human-Computer Interaction 5,* (Norwood, NJ, 1995)

[12] Marie-Laure Ryan, *Narrative as Virtual Reality: Immersion and Interactivity in Literature and Electronic Media,* (The Johns Hopkins University Press, 2001).

[13] Alister West, Keekim Heng, *Watching Window: A reactive real-time model-viewer,* (Proceedings of Image and Vision Computing (IVCNZ), 2001).

[14] Michael Treadgold, *Out of the Fish Tank: a report on a two-screen Fish Tank Virtual Reality (FTVR) system,* (2000).

[15] Mason, D., Wyvill, G., *Blendeforming: Ray traceable localized foldover-free space deformation,* (Proceedings of Computer Graphics International, 2001).

[16] Michael Treadgold, Kevin Novins, Geoff Wyvill, B. Niven, *What do You Think You're Doing? Measuring Perception in Fish Tank Virtual Reality,* (Proceedings of Computer Graphics International, IEEE Press, 2001).

[17] C. Ware, K. Arthur, K.S. Booth, *Fish tank virtual reality* (in ACM CHI 93 Proceedings, 1993).

[18] Enrico Gobbetti, Riccardo Scateni, *Virtual Reality: Past, Present and Future,* (Center for Advanced Studies, Research and Development in Sardinia Cagliari, Italy, 1999)

[19] Geoff Wyvill, Tracy Mason, Garry Downes, Mark Williams, *The Virtual Display Case,* (Proceedings of Computer Graphics International, Hasselt, Belgium 1997).

[20] Prof. Dr. Timm Grams, *Programmkonstruktion,* (Fulda, 2003).

[21] Robert Audenaerde, *The Watching Window Revisited A report on upgrading the Watching Window,* (Dunedin, 2004).

[22] Jerry Isdale, *Motion Tracking,* (Technology Review, Oct. 1999)

[23] Scott Meyers, *Effective C++: 50 Specific Ways to Improve Your Programs and Designs,* (Second Edition, Addison Wesley Professional, 1997)

[24] J. C. Terrillon, M. N. Shirazi, H. Fukamachi, *Comparative Performance of Different Skin Chrominance Chrominance Spaces for the Automatic Detection of Human Images,*(Proceedings of 4th IEEE International Conference Face and Gesture Recognition, pp. 54-61, 2000).

[25] Ghidary, S.S., Nakata, Y., Takamori, T., Hattori, M. *Head and Face Detection at Indoor Environment by Home Robot,* (Proceedings of IEEE2000, May 2000, Iran).

[26] Okoshi, T., *Three-Dimensional Imaging Techniques,* (Academic Press, New York, 1976)

[27] ACM SIGCHI/ACM Special Interest Group on Computer-Human Interaction, *ACM SIGCHI Curricula for Human-Computer Interaction,* (N.Y.: Association for Computing Machinery, 1992).