Ray-Tracing Soft Objects

Geoff Wyvill and Andrew Trotman

Abstract

Soft objects, also known as metaballs or implicit surfaces, are deformable free-form shapes represented as a surface of constant value in a scalar field.

We present a simple, robust method for ray tracing soft objects defined by polynomial field functions. The method is guaranteed to find all the intersections of a ray with a soft object. Thus it is suitable for use in CSG systems where all intersections may be required.

Keywords: geometric modelling, ray tracing, soft objects, animation.

Introduction

Soft objects are solids described by scalar fields. A scalar field is a function that has a real value defined for every point of space. The surface of the soft object is a set of points that have the same field value. For this reason, they are sometimes called iso-surfaces. Soft objects have been described by Blinn (1982), Nishimura (1985), Wyvill (1986b) and others. Both Blinn and Nishimura used a form of ray casting to render their surfaces but both algorithms are of an ad hoc nature and have not been proven to find the intersection of any given ray with the surface. Our earlier papers on soft objects all used polygonal approximations for rendering (Wyvill 1986a, 1986b, 1987a, 1989). Kalra (1989) has described a provable algorithm for ray tracing, but it is more complicated, and (we believe) slower than the one presented here. Also, Kalra's algorithm, as described, finds only the nearest intersection of a ray with a soft object. In constructive solid geometry, a shape is built by adding and subtracting volumes. It often happens that an intersection point of one primitive object has disappeared in the final model because something has been subtracted from that region of space. In cases like this, we must find all the intersections and eliminate the ones that we don't need. It often happens that the correct intersection is not the nearest to the eye.

Kalra's algorithm appears to be more general than ours. They establish a criterion to determine whether a Newton's iteration will converge in a given region of space. If it does not, they divide the region until it will and apply Newton's method in each region. We have restricted our field functions to a polynomial type for which we can guarantee solvability without resorting to space division. Thus we believe our method, while less general, is more practical in most cases.

T.-S. Chua et al. (eds.), *CG International '90*

The field function

The purpose of using soft objects is to model free-form surfaces. These surfaces are usually edited by hand and it is important that we can exercise local control of their shape. Sometimes we need hundreds of data points, and it is important that we can calculate the field value at a point without having to refer to all of them. For this reason, we use a field function that is guaranteed to have only a local effect. Our basic field function is calculated from a set of *key points* as follows:

Key points form a kind of skeleton around which the soft object is drawn. We regard each key point as a source of energy, a hot spot around which the temperature drops off as a function of distance. The field value due to two or more key points is the sum of the values due to the individual key points. To achieve local control, the field produced by each key point has to conform to certain rules. In particular, the field function, f(r), and its derivative: $\frac{df}{dr}$ must drop to zero at some special distance, R from the key point. R is called the 'radius of influence' of a key point: the distance at which the field contribution falls to zero. In earlier work (Wyvill 1986a, 1986b, 1987a) we described a suitable function as a cubic in r^2:

$$C(r) = \begin{cases} -0.4444\frac{r^6}{R^6} + 1.8888\frac{r^4}{R^4} - 2.4444\frac{r^2}{R^2} + 1.0 & , r \le R \\ 0 & , r > R \end{cases} \quad (1)$$

where the coefficients were chosen so that the volumes of the soft objects behaved reasonably when key points were combined. The surface of our soft object lies by definition everywhere where C(r) = *magic*. The value of *magic* is chosen by evil art but in most cases, if we are using equation (1), *magic* = 0.5 works well.

In this paper, we demonstrate that this field function can be ray traced with a provable algorithm for any number of key points. The method can also be extended to non-spherical keys. That is key points for which the effective 'radius' is not simply the geometric distance from the key point, but may depend on direction too.

The algorithm

Because our field function is a polynomial in r, the ray intersection calculation for a single key is done by solving a simple polynomial. The geometry is shown in Fig. 1. For a ray from eyepoint **e** in a direction **v** the line is given by the parametric equation:

$$\mathbf{p} = \mathbf{e} + \mathbf{v}\,t \quad (2)$$

For a key point at **k**, the distance from **k** to **p** is $r = |\mathbf{p} - \mathbf{k}|$ and the field value is:

$$f = C(|\mathbf{e} + \mathbf{v}\,t - \mathbf{k}|) \quad (3)$$

where C(r) is given by equation (1), provided we are within the radius of influence of **k**. The significant point here is that f is a polynomial of degree six in t, and all the

coefficients are calculated from the geometry. If there are more key points, then the field is given by the sum of the effects of the individual keys:

$$f = \sum_i C(|e + v\,t - k_i|) \tag{4}$$

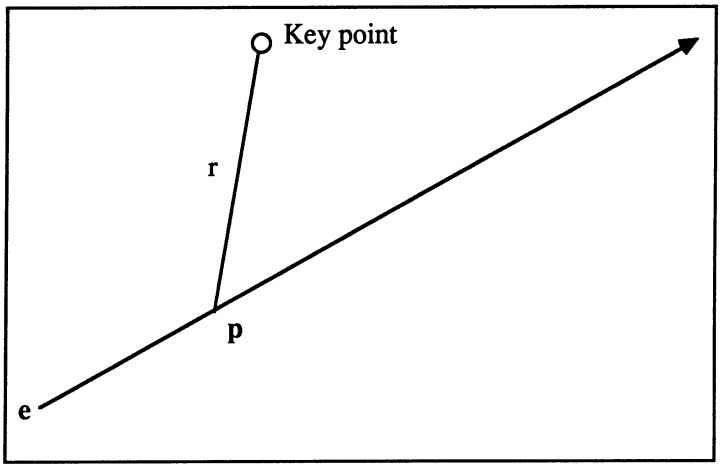


Fig. 1. Ray geometry.

Since this is still a polynomial of degree six, we can solve it for f = *magic* by standard methods. We use Laguerre's method which is guaranteed to find all the roots and, in the case of single roots, has third order convergence. Details of this method can be found in standard texts on numerical analysis; see, for example, Ralston (1965). Substituting the value of t back into (2) gives the point **p**, the intersection of the ray with the surface.

The function defined in (2), of course, is polynomial only within the radius of influence, R. So the simple polynomial solution works only in regions where we remain within the radius of influence of all the key points being considered.

In the general case, the ray will pass into and out of the radius of influence of key points. Each radius of influence defines a sphere around a keypoint and we find the points of intersection of the ray with all of the bounding spheres. Of course, in many cases, most of these spheres will be well away from the ray and it will not be necessary to check them for intersection. Spatial sorting (Wyvill 1986a) can eliminate most of these tests and this will improve the efficiency. Here we are mostly concerned with provability.

The intersection points are sorted according to distance from the eyepoint, **e**. Between any pair of points, the ray neither enters nor leaves a radius of influence so we can check for intersections in that interval using the simple polynomial solution. Each of these intervals is defined by an upper and lower limit of the parameter, t, of equation (2) and the solutions of the polynomials are also values of t. If any solution is outside these limits, then this is not an intersection in that interval and it can be discarded.

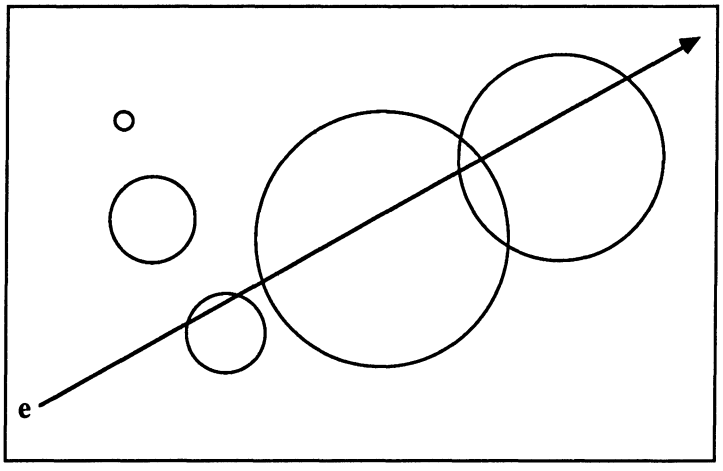


Fig. 2. Between any pair of intersections with bounding spheres, the field is influenced by a fixed set of key points.

The algorithm can be summarised as follows.

1 Find the intersections with all bounding spheres and order them along the ray.

2 Solve the polynomial equation (4) for f = *magic* in each interval.

3 Return those intersection points that fall within each interval.

Quadric surfaces

The basic algorithm can be adapted easily to handle primitives that are not spherical. We represent a point in space by a vector

$$\mathbf{u} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$ and we write its transpose, (x, y, z, 1) as **u**'.

The general quadratic form:

$$Q = \mathbf{u}' M \mathbf{u} \tag{5}$$

is a scalar value, a function of x, y and z, where the coefficients of each term are defined by the elements of M. It is convenient to regard Q as analogous to r^2 in the equation of a sphere and we can use the same cubic in Q to manufacture a soft object field of general quadric shape. The surface $Q = R^2$ gives us a bound and our algorithm can be extended to ray trace soft objects made by combining these shapes.

The quadric surfaces include ellipsoids, cylinders and cones and provide a convenient extension of the shapes we can make using only a few primitives. This extension was used by Blinn (1982) and it is important to note that our algorithm is not limited to spheres.

Negative key points

Blinn (1982) also introduced negative weights for some key points enabling holes and concavities to appear in the soft shapes. Use of such points can create rather complicated and sharply curving shapes. It is worth pointing out that these negative key points are handled equally well by our algorithm.

CSG operations

The soft object ray tracer has been incorporated into our *Katachi* solid modelling system at the University of Otago. *Katachi* supports set operations on volumes described by combining many primitive types (Roth 1982, Wyvill 1987b, 1988). Because we find all the intersections of a ray with the soft object, we are able to incorporate it into the solid modelling scheme. We can add and subtract the soft objects to and from any of the other primitives of *Katachi*.

Figures 3 and 4 illustrate all these things. A teapot has been built from five quadric key points, one of them negative. In Fig. 3, it has been coloured with a symmetrical synthetic marble texture. In Fig. 4, the teapot is given a metallic appearance and is shown with half of a mould from which it could have been cast. The mould was created in the CSG system by subtracting the teapot from a block.

Conclusion

We have developed and tested a robust and simple method for ray tracing soft objects that is guaranteed to find all the intersections of a ray with an implicit polynomial surface defined by key points.

The method has been tested in a CSG environment and works well.

Acknowledgements

The computer graphics project at Otago has been jointly funded by Otago University and the University Grants Committee. Our thanks also go to Television New Zealand for loan of equipment and studio time.

Fig. 3. Textured teapot.

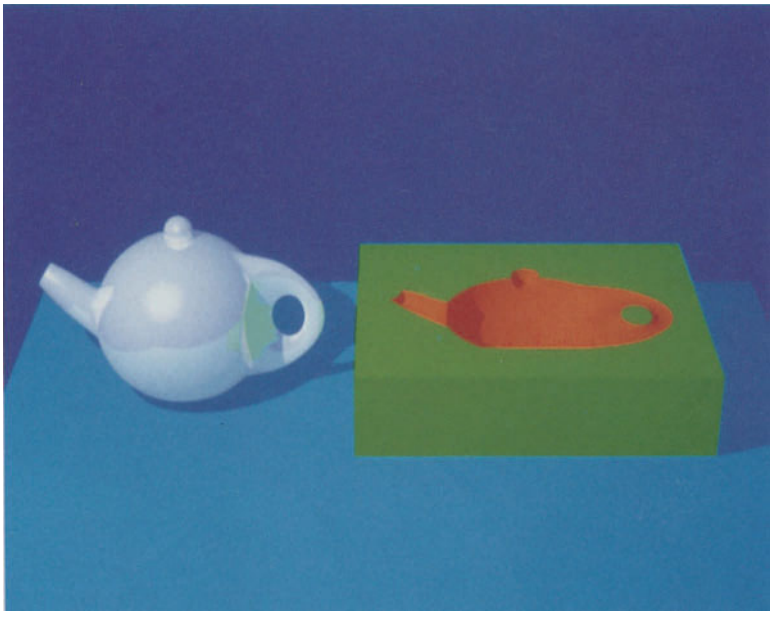


Fig. 4. CSG operations on soft objects.

References

Blinn J (1982) A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics* 1 : 235 - 256

Kalra D and Bar AH (1989) Guaranteed Ray Intersections with Implicit Surfaces. *Computer Graphics* (Proc. SIGGRAPH 1989) 23 (3) : 297-306

Nishimura H, Hirai M, Kawai T, Kawata T, Shirakawa I and Omura K (1985) Object Modeling by Distribution Function and a Method of Image Generation. *Journal of papers given at the Electronics Communication Conference '85* J68-D (4) (in Japanese)

Ralston A (1965) *A First Course in Numerical Analysis.* McGraw Hill 368-371

Roth SD (1982) Ray Casting for Modeling Solids. *Computer Graphics and Image Processing* 18 : 109-144

Wyvill BLM, McPheeters C and Wyvill G (1986a) Animating Soft Objects. *The Visual Computer* 2 (4) : 235-242

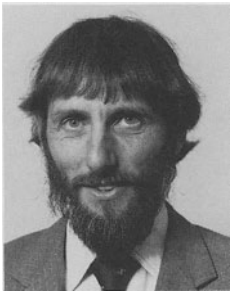
Wyvill G, McPheeters C and Wyvill BLM (1986b) Data Structure for Soft Objects. *The Visual Computer* 2 (4) : 227-234

Wyvill G, Wyvill B and McPheeters C (1987a) Solid Texturing of Soft Objects. *IEEE CG&A* 7 (12) : 20-26

Wyvill G, Ward A and Brown T (1987b) Sketches by Ray Tracing. *Computer Graphics 1987* (Proc. CG International '87, Karuizawa) 315-333

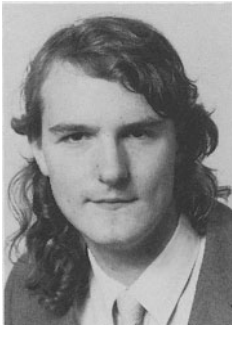
Wyvill G and Sharp P (1988) Volume and Surface Properties in CSG. *New Trends in Computer Graphics* (Proc. CG International '88, Geneva) 257-266

Wyvill BLM and Wyvill G (1989) Using Soft Objects in Computer Generated Character Animation. In: *Computers in Art Design and Animation,* Springer Verlag 283-297



Geoff Wyvill graduated in physics from Jesus College, Oxford, and started working with computers as a research technologist with the British Petroleum Company. He gained MSc and PhD degrees in computer science from the University of Bradford where he lectured in computer science from 1969 until 1978. He is currently senior lecturer in computer science at the University of Otago. He is on the editorial board of The Visual Computer and is a member of SIGGRAPH, ACM, CGS and NZCS.

Address: Department of Computer Science
University of Otago Box 56
Dunedin, New Zealand



Andrew Trotman is a graduate student at Otago University. His research interests include constructive solid geometry and computer animation. He completed a BA degree in computer science in 1988 and he is a student member of ACM and SIGGRAPH.

Address: Department of Computer Science
University of Otago Box 56
Dunedin, New Zealand