Ray Tracing in Calgary and Otago

John Cleary and Brian Wyvill

Department of Computer Science, University of Calgary.
2500 University Drive N.W.
Calgary, Alberta, Canada, T2N 1N4

Geoff Wyvill

Department of Computer Science, University of Otago
Dunedin, New Zealand

Summary This paper gives a brief summary of the work in Ray Tracing at the University of Calgary department of Computer Science in cooperation with the University of Otago in New Zealand. There are various projects: architecture for multi-processor ray tracing machine, parallel cellular ray tracing, theory of ray tracing, ray tracing soft objects and sketching with ray tracing.

The Graphicsland Environment Graphicsland is a hierarchical computer animation system which serves as a testbed for modeling, motion control and rendering programs. The following renderers have been implemented:

- Z-Buffer (including multi-process and CDC 205 version)
- A-Buffer
- Cellular (Voxel) Ray Tracing
- Fuch's Algorithm
- Warnock's Algorithm

Primitive objects supported by the system include: planar polygons (convex or non-convex), B-spline surfaces, soft objects, and particles. Various attributes are handed to the renderers, amongst them are: quality control, degraded objects, haze, abstract texture spaces, transparency, specular reflection, diffuse reflection multiple light sources, anti-aliasing, shadows and smooth shading of polygon meshes. One of the objectives is that new renderers can be tested easily. The system passes the objects and attributes to the selected renderer via a file which contains some header information to distinguish its type, then a sequence of op-codes followed by the appropriate data. The renderers supply routines to handle each of the op-codes and associated data and output a raster in a common format. Display routines exist to produce output on the chosen device.

Particles Particles [Reeves 83] are treated as spheres. Z-buffer, A-buffer and Ray Tracing handle particles, however our particle systems also have an attribute that allows them to be Gauss filtered into the frame buffer. The radius of the filter is a parameter of the individual particle, controlled by an attribute of the parent particle system.

Cheating B-Splines B-spline surfaces are currently rendered as polygons by all the renderers.

Surface Attributes All the renderers implement Phong smooth shading, however many of the other attributes are only provided by the ray tracer or one particular renderer. We do not include conventional texture maps, but instead implement 3D texture functions similar to the ideas of [Perlin 85] and [Peachey 85]. The abstract texture space is described in [Wyvill 87]. The A-buffer algorithm conveniently lends itself to 3D textures, since the buffer contains sufficient information (such as surface normals) to relate each pixel back to the world space.

Light Sources A sun light source is at infinity the user defines a vector between the origin and the light source. A lamp light source is a point in space which emits light in all directions but the effect decays with distance to zero at a specified distance. A spotlight light source is a point in space that emits light within a cone. The bounding cone of unit height which determines the deviance of the light directions is specified by the length of the base of the cone. The sharpness of the spotlight can also be controlled. Again only Ray tracing and A-buffer have all the light source types implemented. Shadows are only calculated with Ray Tracing.

Choosing the appropriate rendering algorithm is usually a matter of trading time and quality. The quality attribute in *Graphicsland* is a number between 0 and 1 and mostly affects anti-aliasing, depending on the individual renderer chosen. For example low quality A-buffer would not anti-alias and in fact default to Z-buffer. High quality Z-buffer would super-sample.

Ray Tracing Soft Objects We have reported on our experiments with an iso-surface in a scalar field to model objects which can deform as they move. Our soft objects are represented by a surface of constant value in a scalar field. This is a logical extension of the idea of a contour map. In a contour map the lines connect points of constant height. If you think of the field as a three dimensional map of temperature where every point in the space can have a temperature value, then a soft object is a surface connecting points with a certain value. The skeleton of the object which controls its shape and movement, consists of a structure of hot points, or wires, which affect the shape of the field. This method was first reported by Jim Blinn [Blinn 82]. Both of these descriptions concentrate on the use of a skeleton of control points to create the field. The main disadvantage of using simple points, or hot spots, to define the field, is that a large number can be required for some fairly simple shapes. For this reason we use an extended notion of a control point which we call a *key*. In general a *key* consists of three line segments at right angles representing the axes of an ellipsoid each of which has an associated radius of influence. A point which is within a radius of influence of more than one *key* has a field value equal to the sum of the values calculated for each *key*. This field determines the shape of the soft object because the surface of the object is an iso-surface of the field. The surface created by just one control point in the original model, is a sphere. The surface created by a single key, is an ellipsoid. To render the soft object we need to find the position of the surface. This is done by finding the field value at successive points along one axis (starting at a control key). Near the key the value will be greater than the iso-value (hot), when the field falls below the iso-value the surface has been crossed. At this point we erect a cube and find the field value at each vertex. It can then be ascertained if the surface cuts any edge of the cube by comparing the field value of each vertex to the iso-value. If the surface cuts a particular edge the adjacent cube is manufactured and the process repeated recursively until the entire surface has been covered by containing cubes. The surface can then be approximated by examining each cube and substituting polygons. Alternatively we now fire a ray at the cube which gives an approximation to the intersection with the surface. An iteration technique is used to determine the actual intersection.

Unfortunately small variations in the surface can be missed using this technique, for example if the surface is folded so that there is more than one part of the surface passing through the cube. Work is on-going in solving this problem. We are also interested in applying the surface following technique to other kinds of mathematical surfaces since the method is general for any field function that is reasonably well behaved. However, each surface component must be found so you need a hint about the shape of the field. Our key points/ellipsoids provide us with a set of points which are useful because they have the following property: No closed surface component exists which does not contain a key point. Thus the key points are good places to start looking for the surfaces.

Sketching with Ray Tracing For many applications, we do not need the realism of the ray tracer, but we still value its generality. By using a sparse grid of rays and by following edges in the picture plane, we can extract most of the information from the model at greatly reduced cost.

A simple example of this strategy is the adaptive subdivision of pixels in regions of large changes in intensity or colour. This is used to get sub-pixel values for antialiasing. This technique, first described by [Whitted 1980], is widely used and has set a standard for comparison with other approaches to antialiasing. [Fujimoto 1986b] and [Heckbert 1984], for example, both mention the method. Why can we not extend the idea to reduce the number of primary rays traced? It would be splendid, for example, to start with a very coarse grid (50 by 50 pixels) and, by recursive division, still obtain a high quality picture. This leads to loss of detail. Heckbert and Hanrahan (1984) overcame this problem with their beam tracer for polygonal objects. The sketching algorithm applies to a CSG model. A coarse grid of rays are cast and we examine every adjacent pair of sample points. If their intensities are different, we look for edges between them by recursive sub-division. [Roth 1982] mentions the use of a binary search to find edges, but this is at the sub-pixel level. Further details of this algorithm are available in [Wyvil 87b].

Theory of Ray Tracing A paper is currently in preparation by John Cleary and Geoff Wyvill which analyzes the performance of cellular ray tracing. This algorithm assumes that the scene being ray traced is subdivided into equal sized voxels which have a record of the objects which intersect them. This equal subdivision of cells allows a very fast next-cell skipping algorithm requiring about 8 machine operations per cell.

The major problem in analyzing this type of situation is the expected number of objects which will have to be checked for an intercept against any given ray. By averaging over all possible orientations, and relative positions of the ray the result is a closed expression in the areas and circumferences of the polygonal objects making up the scene. The upshot of this is that for any given scene divided into $n \times n \times n$ voxels the time has terms in n, n^{-1}, n^{-2}, and n^{-3}. From this an optimum value for n can be chosen.

Given a particular scene it is possible to compute how the execution time will vary if the scene is scaled, for example by looking at a larger scene which has similar average characteristics and includes the original scene as part of it. The result is that under some reasonable assumptions the execution time is constant although the space requirements can be more than linear in the amount of scaling. Work is continuing on a similar analysis of oct-tree subdivision.

References

Blinn, J (1982) "A Generalization of Algebraic Surface Drawing" *ACM Transactions on Graphics, 1,* 235.

Cleary, J and Wyvill, G (1981) "An Analysis of an Algorithm for Fast Ray-Tracing using Uniform Space Subdivision" *Research Report No. 87/264/12,* University of Calgary, Department of Computer Science.

Fujimoto, A, Tanaka, T, and Iwata, K (1986b) "ARTS: Accelerated Ray-Tracing System" *IEEE CG&A, 6* (4) 16-26.

Heckbert, P and Hanrahan, P (1984) "Beam Tracing Polygonal Objects" *Computer Graphics (Proc. ACM SIGGRAPH 84), 18* (3) 119-127.

Peachey, D (1985) "Solid Texturing of COmplex Surfaces" *Computer Graphics (Proc. SIGGRAPH 85), 19* (3) 279-286.

Perlin, K (1985) "An Image Synthesizer" *Computer Graphics (Proc. SIGGRAPH 85), 19* (3) 287-296..

Reeves, William. (Apr 1983) "Particle systems- A technique for modeling a class of fuzzy objects" *ACM Transactions on Graphics, 2,* 91-108.

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

Whitted,T (1980) "An Improved Illumination Model for Shaded Images" *CACM, 23* (6) 343-349.

Wyvill, G, Wyvill, B, and McPheeters, C (1986a) "Data Structures for Soft Objects" *Visual Computer, 2* (4) 227-234.

Wyvill, G, Wyvill, B, and McPheeters, C (1987) "Solid Texturing of Soft Objects (in press)" *CG International 87,* Tokyo, May 1987.

Wyvill, G, Ward, A, and Brown, T (1987b) "Sketches by Ray Tracing" *Research Report No. 1/1/87,* Department of Computer Science, University of Otago.