Ray Tracing with Cones

John Amanatides

Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4

Abstract

A new approach to ray tracing is introduced. The definition of a "ray" is extended into a cone by including information on the spread angle and the virtual origin. The advantages of this approach, which tries to model light propagation with more fidelity, include a better method of anti-aliasing, a way of calculating fuzzy shadows and dull reflections, a method of calculating the correct level of detail in a procedural model and texture map, and finally, a procedure for faster intersection calculation.

CR Categories and Subject Descriptions: I.3.3 [**Computer Graphics**]: Picture/Image Generation - display algorithms; I.3.7 [**Computer Graphics**]: Three-dimensional Graphics and Realism - Shading, Shadowing, Texture, Visible Line/Surface Algorithms;

General Terms: Algorithms

Additional Keywords and Phrases: Ray Tracing, Anti-Aliasing

Introduction

Ray tracing is a very powerful yet simple approach to image synthesis. Though expensive computationally, it has generated some of the most realistic scenes to date [10, 13, 17]. It is also used for entertainment and computer aided design [9, 15]. However, apart from the computationally expensive method of supersampling, no general method exists to remove artifacts created by aliasing. Furthermore, at present, there is no general procedure to decide what level of detail is sufficient in a texture map or in a procedural or hierarchical model of an object when ray tracing. This is primarily because individual rays are infinitesimally thick and thus we cannot exploit area-sampling techniques to avoid aliasing artifacts. This paper goes beyond some of these limitations by redefining the concept of "ray".

The Problem

In ray tracing, rays are shot from the eye into the world. They are constrained so that they pass through the center of the pixels in the virtual screen. Once they have left, any relationship between the ray and the pixel on which the results will be displayed is severed. This is because a ray is defined as a starting point and a direction which together form a line. This simple definition allows for straightforward and fast intersection calculations with various objects [11, 13, 14]. Unfortunately, it also has drawbacks.

The main drawback with the above standard approach is that there is not enough information associated with the ray to perform anti-aliasing [5]. Rays allow us only to sample at the one point in the center of a pixel. There is no way of knowing or calculating what else is visible in the neighborhood surrounding the sample point.

The only way to anti-alias within standard ray tracing is to go to higher resolution. Whitted proposed adaptive supersampling and it is now almost universally used [17]. There are a

couple of problems associated with this approach, however. First, the amount of computation can go up drastically in pixels where large variances of intensity occur. Small polygons with a texture mapped on them are a case in point. The second problem is that small details may "fall through the cracks" of the sample points. This is especially true of objects that are reflected or refracted by other objects.

One way of attacking the sampling problem outlined above is to modify the definition of "ray". The pixel should represent not a point but an area of the screen. This fact can be encorporated if the ray becomes a pyramid with the apex at the eye and the base defined by the four planes that cut the borders of the pixel. Intersection calculations between this extended ray and an object can decide not only if there is an intersection but also what fraction of the ray intersects the object. This fractional coverage information is sufficient to perform simple area anti-aliasing. Also, nothing can "fall through the cracks" as the ray covers the whole pixel. Now, only one ray per pixel is sufficient regardless of scene complexity.

The penalty of traveling this route, however, is that intersection calculations can become quite involved.[1] If a ray is reflected or refracted by a curved surface the resulting ray can be very distorted, furthering the complexity of the intersection calculations. Also, consider an object, A, that intersects only a portion of the ray. To correctly render objects behind A, the ray should be modified to indicate the portion that is blocked by A. The proposed new definition of a ray must be further extended to handle this. Approximations, such as coverage masks [7], may be used but these calculations quickly become prohibitive.

An Approximation: Cones

The above extended definition of a ray is too complex to be easily implemented so a simplifying approximation is proposed: Let the new definition of a ray be a circular pyramid or cone. This will be made possible by including the angle of spread and virtual origin of the ray in the definition which originally included only the origin and direction of the ray. The spread angle is defined as the angle between the center line of the cone and the cone boundary as measured at the apex of the cone. This angle is chosen such that when the ray is sent from the eye the radius of the cone at the distance of the virtual screen is the width of the pixel. The virtual origin is the distance from the apex of the cone to the origin. This will not be zero for reflected or refracted cones. Reflected and refracted rays continue to keep this symmetric shape, modifying the spread angle and distance to the virtual origin so that a good approximation of the cone is constantly maintained.

Calculating the intersection between a cone and an object is still rather complex. This will be described in the next section. The result from the intersection calculations should indicate not only if there is an intersection but also the fraction of the cone that is blocked by the object. A sorted list is maintained of the eight closest objects that intersect the ray. This list is used for anti-aliasing. If the closest object does not completely fill the ray then the next object in the list contributes to the pixel value. Since at present only the fractional coverage value is used in mixing the contributions from the various objects, overlapping surfaces will be calculated correctly but abutting surfaces will not. Additional information in the sorted intersection list can be used to rectify this shortcoming.

Reflection and refraction calculations must take into account that the ray is now a cone. The new direction of the ray is calculated in the same manner as standard ray tracing and uses the center line of the cone. To calculate the new virtual origin and spread angle the surface curvature is required. A constant curvature is assumed throughout the area of intersection and the optical laws of spherical mirrors and lenses are used [12]. Unfortunately, we cannot use the simple lens equations that depend on the "paraxial" approximation (the incident angle of the incoming ray is close to zero). The more general equations are required.

Intersection Calculations

We now describe the intersection calculations between a cone and various objects. These objects include spheres, planes and polygons. In general, each intersection calculation should consist of two parts: a fast in/out test and then a more complicated area intersection approximation. The quick first test is desirable since most objects will intersect

1. In fact, Whitted started work in this direction but abandoned it due to the complexity of the intersection calculations [17].

relatively few rays.

The intersection calculation between a sphere and a cone consists of two parts. The first part tests if an intersection will occur and the second part calculates the fractional coverage.

This test begins by finding the point on the cone's center line (CP) that is closest to the center of the sphere and the distance between the two points (SEP). In standard ray tracing this must also be performed with the test being negative if SEP is greater than the radius of the sphere. The above comparison must be modified to take into account that the ray is a cone. Let the distance between CP and the virtual origin of the ray be T, the spread angle A and the radius of the sphere R. We calculate the following:

$$D = T*\tan(A) + R/\cos(A)$$

If D is less than SEP, then there is no intersection between the ray and the sphere (see fig. 1). The above calculation requires the evaluation of two trigonometric functions. Notice, however, that these functions only depend on the spread angle and need be evaluated once, before any intersection calculations begin. This results in a test that is only a few floating point operations more expensive that regular ray tracing.

The second part of the intersection calculation evaluates the fractional coverage of the sphere within the ray. This is equivalent to finding the area of intersection of two circles, the outline of the sphere and the outline of the cone where it is closest to the sphere. To calculate this quickly an approximate solution involving a simple polynomial evaluation is performed.

The intersection calculation between a ray and a plane is now described. The calculation begins with a quick test to make sure that the plane is not behind the origin of the ray by calculating the intersection between the center line of the ray and the plane. If the intersection is behind the origin of the ray then the plane is discarded. Otherwise, the angle between the centerline of the ray and the plane normal is computed. This angle and the spread angle of the ray are compared and it is a simple matter to test for intersection.

The next part of the intersection calculation computes the fractional coverage. The problem can be reduced to two dimensions and involves finding the area of intersection between a circle (the cross section of the cone) and a half plane (the horizon). The spread angle and the angle between the ray and plane computed above together indicate how the distance between the center of the circle and the edge of the half plane. Given this distance, the area of intersection is computed using a polynomial approximation. This completes the intersection calculation for planes.

The intersection calculation between a cone and a polygon is now outlined. There are two reasonable strategies we can use. The first requires we intersect the cone with the plane defined by the polygon and perform an intersection test between the polygon and the cross section obtained. The cross section of a conic can be either a circle, ellipse, hyperbola or a parabola. This makes intersection calculations more complicated. The second strategy requires that we project the vertices of the polygon onto a plane perpendicular to the direction vector of the cone. Now the cross section of the cone with that plane is always a circle. We then must calculate the intersection between the projected polygon and a circle. This can be accomplished by calculating the distance from the center of the circle to each of the edges and then using the circle - half plane intersection estimation mentioned earlier.

Choosing a Representative

Most anti-aliasing schemes make some assumptions within a pixel to simplify the algorithm. Two common assumptions are that the depth (z value) and intensity of any individual object are constant within the pixel [5, 7]. But the choice of the representative, the sample point on the object within the pixel at which the intersection calculations will be performed, must be made with care or the attempt at anti-aliasing will falter.

There are two variables that can be altered when making this decision. The first is which point on the object should be chosen? The center of the pixel is universally used. Problems arise when the object does not occupy this point. Solutions that only pick the point on the object that is closest to the center (as seen from the eye) can lead one astray. For example, consider the case in which a cone partially intersects an infinite plane but the center of the cone does not intersect. The point closest

to the center would be on the horizon. If we model the attenuation of light with distance this will result in an intensity of zero for this point since the distance to the horizon is infinite.

The second variable one has when choosing the representative is what surface properties should the sample point be given? They do not have to be exactly the same as the values found at the sample point. Why you may want to change these values is illustrated below:

Specular highlights can be a problem when the surface normal varies greatly within a pixel [6, 18]. By reducing the directional dependence of the specular highlight in these trouble spots the problem is diminished. For example, if we are using Phong shading, we can clamp the value of n, the power to which the specular dot product is raised, to a value that will subdue this form of aliasing.

Levels of Detail

A recurring problem with procedural and hierarchical models of objects is the level of detail to which they should be generated [3]. In classical ray tracing there is no good answer as there is no way of knowing how much of the screen an object will fill. Stochastic surfaces are a good example [8, 13]. If we do not subdivide the surface enough we will see the resulting polygons. If we subdivide too far, however, we will encounter two problems: First, we will waste computing resources and second, we will be forced to undersample. This is because further subdivision will introduce higher frequency components into the stochastic surface, frequencies that cannot be reproduced faithfully. We can use cones to advantage here. By calculating the size of the intersection, we can decide what level of detail is sufficient. Thus two different views of the same object, one direct and one reflecting off another surface, can both be rendered at the correct level of detail. Kajiya has performed ray tracing with prisms and surfaces of revolution [13]. His implementation of these objects require the use of strip trees, a hierarchical structure which represents a curve at various resolutions. We can speed up the intersection calculation with these objects by only subdividing the strip tree to a level sufficient for display.

Cones can also be used to anti-alias texture. At each intersection an estimate of the size and shape of the intersection can be calculated. This information can be used to generate the filter to average the texture map.

Fuzzy Shadows

Virtually all graphics systems model light as either a point source or as a direction from which parallel light beams emanate [4]. Consequently, shadows cast by these light sources exhibit sharp boundaries. Cones allow us to extend our repertoire of light sources to include ones that cast fuzzy boundaries at almost no extra cost. For example, we can add spheres of varying radii as light sources. At each intersection, when a ray is sent to the light source to calculate the shadow, we broaden the ray to the size of the light source. By calculating how much of the light source is blocked by intervening objects, we have enough information to generate fuzzy shadows. Note that this does not produce completely correct shadows. The shadows of transparent objects will still be wrong. The concentration of light by these refracting surfaces cannot be generated using this simple approach.

Dull Reflections

In his classic paper [17], Turner Whitted raised the issue of generating specular highlights using ray tracing techniques. His approach, however, suffered from aliasing and fired off many rays at each intersection point. This is very expensive computationally and was thus abandoned. We produce similar results by simply broadening the reflected ray. In this manner, only one ray is required. When rays are broadened, reflecting surfaces become less glossy. This results in reflections that are less detailed. In a similar manner, translucency can be modeled by broadening the transmitted ray.

The above remarks suggest that the amount of ambient lighting can be estimated by firing very broad rays from each surface and using simple lighting models to prevent an infinite regress of rays.

Reducing Intersection Calculations

The cone approach provides a basis for reducing the number of intersection calculations required for ray tracing. By recursively firing cones of various sizes at the screen, we can perform a Warnock style culling process [16]. This can significantly reduce the number of intersection calculations required at each

pixel by capitalizing on image coherence. A test case of six spheres (without reflection, refraction or shadows) resulted in an order of magnitude reduction in intersection calculations. This result can be immediately applied to ray casting [1], and with some modifications, to ray tracing in general.

Results

Figures 2 - 5 are examples of images generated using some of the above improvements to ray tracing. They all took approximately 50 minutes each to compute on a VAX 780. Figure 2 illustrates anti-aliasing and fuzzy shadows. The light source is a sphere of radius 20 units (each checkerboard is one unit wide) and approximately 300 units away from the scene. The checkerboard is modeled as a procedural texture map.

Figure 3 illustrates dull reflections. The balls become progressively less glossy from left to right. This is evidenced by the reflected checkerboard that varies in detail form ball to ball. The extra angular spread of the reflected rays for each of the balls in left to right order is 0., .2 and .4 radians.

Figures 4 and 5 illustrate ray tracing textures. The method of pyramidal parametrics [18] was used to filter the texture.

Conclusions

We have introduced a new approach to ray tracing: cones. With cones, only one ray per pixel is now required to perform anti-aliasing. The cone approach can also easily be used to decide the correct level of detail, generate fuzzy shadows and dull reflections and reduce intersection calculations. Work is still required to find efficient intersection algorithms for more complicated objects.

Acknowledgements

I wish to thank Alain Fournier for his support and valuable comments. I also wish to thank Eugene Fiume, Ralph Hill, Michael Hollosi and Delfin Montuno who were a sounding board for many ideas and contributed numerous suggestions.

References

1. Amanatides, J., and Fournier, A., "Ray Casting using Divide and Conquer in Screen Space", *Proc. Intl. Conf. of Engineering and Computer Graphics*, Beijing, China, Aug. 27 - Sept. 1 1984.

2. Blinn, J.F., and Newell, M.E., "Texture and Reflection in Computer Generated Images", *Comm. ACM*, Vol. 19(10), October 1976, pp. 542-547.

3. Clark, J.H., "Hierarchical Geometric Models for Visible Surface Algorithms", *Comm. ACM*, Vol. 19(10), October 1976, pp.547-554.

4. Crow, F.C., "Shadow Algorithms for Computer Graphics", *Computer Graphics*, Vol. 11(3), July 1977, pp. 242-248.

5. Crow, F.C., "The Aliasing Problem in Computer-Generated Shaded Images", *Comm. ACM*, Vol. 20(11), November 1977, pp. 799-805.

6. Crow, F.C., "A Comparison of Antialiasing Techniques", *IEEE Computer Graphics and Applications*, Vol. 1(1), January 1981, pp. 40-48.

7. Fiume, E., Fournier, A., and Rudolph, L., "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General Purpose Ultracomputer", *Computer Graphics*, Vol. 17(3), July 1983, pp. 141-150.

8. Fournier, A., Fussell, D., and Carpenter, L., "Computer Rendering of Stochastic Models", *Comm. ACM*, Vol. 25(6), June 1982, pp. 371-384.

9. Goldstein, R.A., and Nagel, R., "3-D Visual Simulation", *Simulation*, January 1971, pp. 25-31.

10. Hall, R.A., and Greenberg, D.P., "A Testbed for Realistic Image Synthesis", *IEEE Computer Graphics and Applications*, Vol. 3(8), November 1983, pp. 10-20.

11. Hanrahan, P., "Ray Tracing Algebraic Surfaces", *Computer Graphics*, Vol. 17(3), July 1983, pp.83-90.

12. Hect, E., and Zajac, A., **OPTICS**, Addison Wesley Publishing Company, Reading Massachusetts, 1974.

13. Kajiya, J.T., "New Techniques For Ray Tracing Procedurally Defined Objects", *Computer Graphics*, Vol. 17(3), July 1983, pp. 91-102.

14. Rubin, S.M., and Whitted, T., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", *Computer Graphics*, Vol. 14(3), July 1980, pp. 110-116.

15. Roth, S.D., "Ray Casting for Modeling Solids", *Computer Graphics and Image Processing*, Vol. 18, 1982, pp. 109-144.

16. Warnock, J., *A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures*, Univ. Utah Computer Sci. Dept., TR 4-15, 1969, NTIS AD-733 671.

17. Whitted, T., "An Improved Illumination Model for Shaded Display", *Comm. ACM*, Vol. 23(6), June 1980, pp. 343-349.

18. Williams, L., "Pyramidal Parametrics", *Computer Graphics*, Vol. 17(3), July 1983, pp. 1-11.

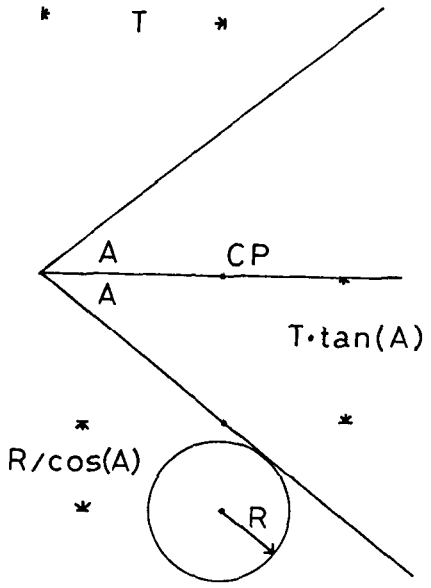

Figure 1

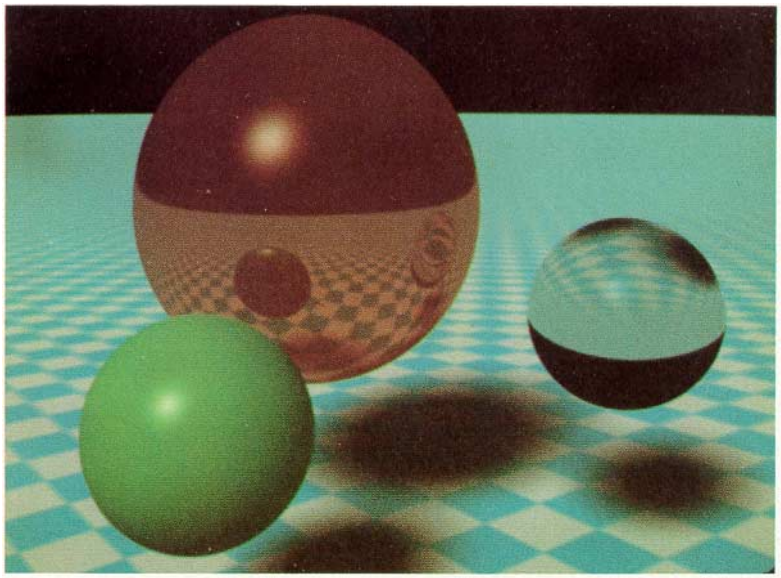

Figure 2

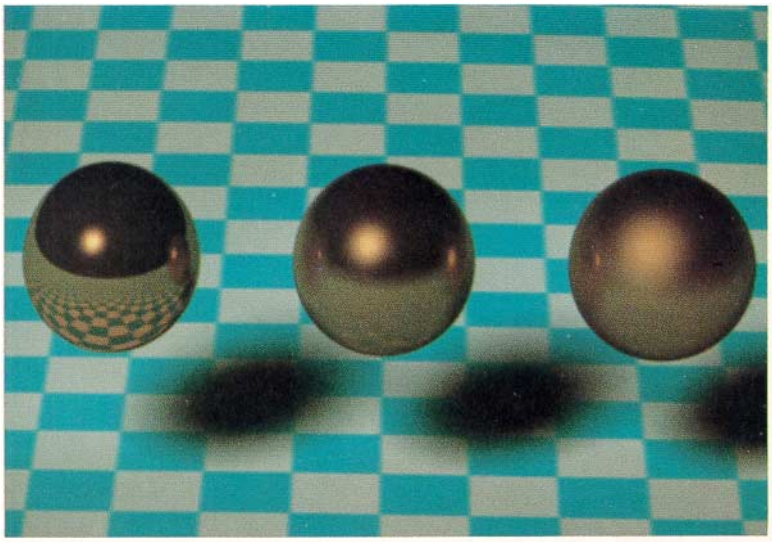

Figure 3

Figure 4

Figure 5