Fast Scanline Processing of Useful Implicitly-Defined Shapes: Sphylinders, Cone-Spheres and Rounded Polygons

Karansher Singh and Richard Parent*
Department of Computer and Information Science
The Ohio State University
2036 Neil Avenue, Columbus, OH 43210 USA

26 October 1993

Abstract

Implicit Surfaces provide an elegant approach to modeling and animating physically deformable and organic looking objects. A major drawback of this approach is its rendering complexity. Some work has been done on developing fast, robust techniques for ray tracing and polygonization of implicit surfaces. While a fast scanline technique has been proposed for rendering spheres, spheres alone are limited as a modeling primitive. This paper addresses techniques for fast scanline rendering of implicit surfaces in general, and provides details for implicit primitives: *spheres*, *sphylinders*, *cone-spheres* and *rounded polygons*. These primitives are particularly useful for modeling articulated figures.

1 Introduction

1.1 Background

An increasingly popular approach to object modeling, especially models which are organic-looking or which are to be animated, is to define the shape of the object as an *implicit surface* [4],[5] ,[13]. An implicit surface is defined as the set of points satisfying an equation $F(x, y, z) = 0$. A very useful set of implicit surfaces can be generated as the summation of polynomial functions each of which is defined over a finite volume, which is typically spherical. Thus $F(x,y,z) = \sum g_i(x,y,z) - T$, where i runs over primitive polynomial functions g_i and T is a threshold value $\in [0,1]$. The main drawback of this approach is the time taken to sample space in order to determine the implicitly defined surface. While a fast scanline technique has been proposed for rendering spheres [1], spheres alone have limited usefulness as a modeling primitive. This paper addresses the fast rendering of other primitives which are useful for constructing implicit surfaces.

*email: singh-k@cis.ohio-state.edu, parent@cis.ohio-state.edu, ph:(614)292-0055, fax:(614)292-9021, contact: R.Parent

Implicit surfaces have gained popularity due to several reasons.

- The modeling primitives over which the functions are specified usually have a simple convex geometric shape. Therefore, they are intuitive for the user to employ as primitives in building more complex shapes.
- A complex object can be deformed and animated easily, simply by manipulating the individual primitive shapes.
- The resulting surface is always smooth as the object is built from summed functions that are continuous. This makes implicit surfaces an attractive alternative to using higher order surface patches for object modeling [15].
- Topological changes to the object due to manipulation of the primitives are handled automatically and elegantly.

The simplest implicit shape to use as a modeling primitive is a sphere, within which the function is radially symmetric about a central point in space. The terms *metaball*, *blob* and *soft objects* have been used in previous work to refer to this shape and, by extension, to any such primitive used in constructing an implicit surface.

Wyvill [11] discusses the representation of what he refers to as *soft objects* defined by:

1. a set of *key points* each of which defines the center of a sphere.
2. a set of *radii of influence*, each of which is a distance from the corresponding key point beyond which the corresponding function has no effect.
3. a *density function* which is a sixth degree polynomial function of the normalized distance of a point from the key point; this is the same for all soft objects.

4. a *threshold* which is the density value in space that defines the implicit surface.

For a point in space, P, the density functions are evaluated at the point P and summed for all contributing primitives. If the sum is equal to the threshold, then the point P is on the surface of the object.

In this paper a *density function* is any function with certain properties (section 2.3) whose domain is the distance ratio of a point. The *distance ratio* for a point is a normalized value $\in [0, 1]$ obtained as the ratio of the distance of the point from the key point or set of points and a radius of influence for the primitive at that point.

The implicit surfaces addressed in this paper are **distance surfaces**, as defined in [3]. The difference between distance surfaces and **convolution surfaces** [3] lies in the method of density value calculation for a point in the primitive. The techniques discussed in this paper deal with the geometric properties of the implicit primitives, which are the same for either surface. These may thus be identically applied to the rendering of convolution surfaces.

Implicit surfaces can be rendered in one of two ways. One way is to construct a polygonal boundary representation from the summed density functions. This can be done as in [9], [2]. The polygonal surface can then be rendered by any of the various standard rendering algorithms. Alternatively, one may trace rays through space, sampling the summed functions as the rays progress away from the observer. Whenever the sum achieves the threshold, the surface has been encountered and display attributes can be determined [14], [12],[17]. Various techniques for fast and robust ray tracing of implicit surfaces have been studied [16],[8].

Blinn [1] addresses the issue of efficiently rendering surfaces defined by spherical density functions by employing scanline processing; modeling molecular structures are the motivating influence in his work. He extends the processing of the basic sphere shape to include affine transformations of spheres.

Blinn presents efficient scanline computations for spheres by processing the spheres in an unperspected space in which scanline-planes are parallel to the x-axis. He goes on to state that arbitrary quadrics are handled by considering them as affinely transformed spheres. Essentially, the space in which the quadric resides is inversely transformed into a space which takes the arbitrary quadric back into a sphere. However, this eliminates the simplifying assumption that makes calculating extents of spheres efficient, namely that a scanline-plane is parallel to the x-axis.

Blinn uses an exponential density function with two user-supplied parameters: the *radius* of the isolated metaball and the *blobbiness*; blobbiness controls how quickly the density function falls off as a function of distance. For a single sphere, an analytic solution to finding the isosurface along a ray can be found. For overlapping density functions, he resorts to a series of iterative solutions using heuristics to drive a hybrid Newton-iteration/regula-falsi approach.

To speed up processing of spheres, Blinn uses standard scanline techniques to manage ray-object intersection testing. He maintains an *active y list* and an *active x list.* In addition, to speed up the search for the isosurface, he maintains an *active z list.* He uses an algebraic approach to solving for extents in x, y and z.

1.2 Overview

The objective of this work is to develop techniques and algorithms which allow for the fast rendering of implicitly defined surfaces. This paper presents results along the direction set forth by Blinn and incorporates scanline techniques to speed up display processing of implicitly defined objects.

We first discuss properties of implicit functions that may be applied to various geometries and how they lend themselves to various optimizations. We then focus on applying a scanline approach to rendering geometries which could be used as primitives in constructing implicit surfaces. Algorithms are developed for primitives which have the following properties.

- They are *intuitive* to use. The user can place the primitives in space by specifying the coordinates of a few points and have a reasonable expectation of the result.

- They are *expressive.* The primitives are useful for modeling a variety of shapes. One of our motivations was the representation of articulated figures [7], [6].

- The primitives lend themselves to *scanline processing.* The calculations which determine extents in x, y and z must be tractable.

An object-oriented approach has been used so that pointers to necessary functions are encapsulated in the object data structure. To add a new primitive to the system, the basic rendering procedure does not have to be changed. We specify a set of functions that our scanline approach requires of any arbitrary primitive geometry.

Appendix A presents exhaustive and optimized computations for the following specific shapes:, the sphere, the sphylinder (a finite cylinder capped on either end by hemispheres), the cone-sphere (a truncated cone capped on either end by spherical segments), and rounded polygons (the surface enclosing a polygon with each point on it at a fixed shortest distance to the polygon). The computations presented here with regard to spheres are essentially equivalent to those presented by Blinn; however, the computations found here have a marked geometric flavor as opposed to the algebraic approach of Blinn.

Nelson Max [10] argues for the usefulness of cone-spheres as a modeling primitive. In his work, sphylinders are just considered as another example of cone-spheres. However, sphylinders present opportunity for computational simplifications which can be taken advantage of if treated as a separate case.

Computations for rendering the above specified primitives in image space are given along with the speed advantages and trade-offs that arise out of image space computation.

Example images are shown with associated timings; these are compared with a standard ray tracing approach. Scanline processing is shown to significantly speed up the rendering of implicit surfaces.

2 Polynomial Implicit Functions and Scanline Processing

The optimizations addressed in this work fall into three categories:

1. Characteristics of standard *scanline processing* are used to capitalize on coherence to speed up ray-object intersection tests. The bounding volumes here should be as tight as possible as the actual surface intersection computation is very expensive. Thus wasted computation should be minimized.

2. Properties of the implicit functions are used to speed up the search along a ray for surface intersection point. This is computationally the most expensive task and should be made as efficient as possible.

3. Simplifying approximations of the usual sixth degree density function are discussed, allowing for a precise analytical calculation of the surface in some cases.

While the organization of the processing is similar to that used in scanline algorithms, the system is still technically a ray tracer. The ray tracing, however, takes place in *eye-space*. In eye-space, the objects are defined relative to a left handed coordinate system centered at the observer with the positive z-axis aligned with the line of sight. Object geometries, defined by a few key points and parameters, can be transformed to eye-space with minimal computation. Processing in eye-space simplifies the scanline calculations. Processing in perspective space (image space or screen space) involves dealing with geometries distorted by the perspective transform. Normal vector calculations also need to be given special consideration. Image space however provides the opportunity of exploiting scanline and pixel coherence by using incremental computation for the primitives. Image space computations for the addressed primitives is discussed in section 4.

2.1 Scanline Processing

The system requires functions that calculate scanline ranges, spans across a scanline, and depth ranges at a pixel for every object primitive. This information is used to maintain an *active object list* which can be incrementally updated similar to standard scanline algorithms and produce a *span segment* from the object for each scanline. While the scanline-planes are not parallel to the z-axis, a z value can be computed based on the viewing parameters such that, at that z value, each integer x value in the span segment corresponds to a pixel on the screen.

For each such pixel covered by the span segment a *z-span* descriptor is generated which is added, in sorted order by closest z value, to a bucket for that x position in the scanline. For a given scanline, once this is done for all objects we have, for each pixel, all of the ray segments that intersect any of the primitives. These must then be searched for the nearest z value solution to the implicit surface equation.

In the case of incrementally building a desired surface or during animation, the intermediate z-span descriptions can be retained or updated to accelerate the rendering process for subsequent versions or frames. The z-spans contain all of the information necessary for evaluating the object density functions.

Because the basic result of the visibility processing is equivalent to that of a z-buffer, adding boundary-represented objects to the display processing is easily handled. The implicit surfaces can be rendered into a z-buffer and used to initialize its contents, or the boundary representation can be transformed into eye space and rendered by the same procedures used to render the implicit surfaces. The former technique requires the use of a full zbuffer while the efficiency of the latter technique suffers from performing all computation in unperspected space (the ubiquitous space-time tradeoff).

2.2 Implicit Function Properties

Blinn [1] uses a powerful heuristic which finds a number of initial guess points, one for each z-span. For each z-span this guess point is either the first point where the primitive's function achieves the threshold value or the point where it attains a maxima. In Blinn's case with spheres, this was either the intersection of the ray corresponding to the pixel with an appropriately scaled sphere or the center of the span. The implicit function for the entire object is evaluated at the guess points in a front to back fashion and the first pair of consecutive guess points with the first above the surface $F(x, y, z) < 0$ and the second below $F(x, y, z) > 0$, bound the interval with the solution. A composite Newton Raphson, Regula Falsi method is now used to obtain the surface point.

While this heuristic works well in practice for the symmetric and well behaved function generated by the span of a spherical primitive, it can give problems with other primitives and geometries that are subtracted from the surface. Even cases of overlapping spherical primitives can result in errors. The problems include missing a solution where one exists and returning a solution other than the one closest to the eye.

The polynomial functions used are monotonically decreasing functions with respect to single parameter. The desired property for its definition over a geometric volume is a smooth transition of its function value from a maximum at a skeletal center of the volume to zero at the boundary of the volume. Typically this is realized by making the parameter a function of the distance from some central element inside the volume. For points along a ray, this distance monotonically decreases to a minimum inside the volume after which it monotonically increases, for simple central elements like points, line segments, and convex polygons which form the centers for the primitives addressed in this paper. This fact is used to note that all z-spans from a single volume consist of a segment where the implicit function value monotonically increases from zero to a maximum and then monotonically decreases to zero.

This allows further speed up of the traversal of the z-span segments and makes the heuristic described above more robust. By fragmenting the sorted spans into two monotonic spans then fragmenting overlapping spans into homogenous subspans, segments that remain monotonic after summation, can be disregarded altogether if their terminus values are below the threshold.

2.3 Simplified Density Functions

The desired properties of a density function f are: f is a function of the normalized distance (or distance ratio) between 0 and 1, $f(0) = 1$, $f(1) = 0$, $f'(0) = 0$, $f'(1) = 0$, f is monotonically decreasing. As developed in Wyvill [11], a simple Hermite cubic solution is well suited to the above. However, to avoid taking square roots in distance ratio calculations, a sixth degree polynomial with the added constraint $f(\frac{1}{2}) = \frac{1}{2}$ is used.

A close approximation to the original density function is provided by the function $f(x) = (x^2 - 1)^2$. See Figure 1. It has all the enumerated properties except $f(\frac{1}{2}) = \frac{9}{16}$; the visual difference between the blending obtained by the two functions is negligible. For primitives where explicit distance to the skeleton can be computed rather than the square of the distance using geometric arguments, a combination of two quadratics $f(x) = 1 - 2 * x^2$ for $x \in [0, 0.5]$ and $f(x) = 2 * (1 - x)^2$ for $x \in [0.5, 1]$ provides similar blending properties.

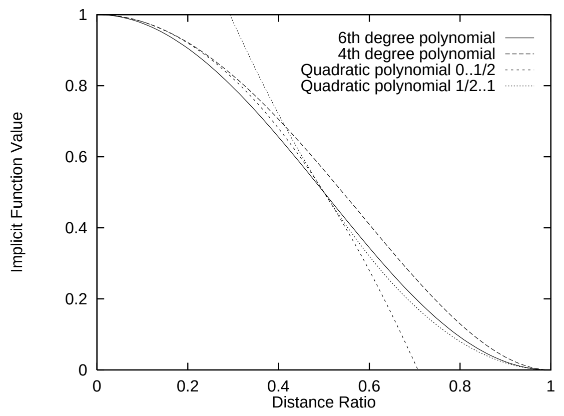

Figure 1: Polynomial Density Functions

The quartic and quadratic polynomials facilitate precise surface intersection calculation within a nonoverlapping depth span, for all object primitives whose geometry is defined in terms of a skeleton and a constant distance around it (spheres, sphylinders and rounded polygons). For a rounded polygon the distance to the skeleton computation falls under one of three cases for a point $P = A + B * t$ parametrically defined along a ray.

- Distance vector to one of the vertices V is $(P - V)$.
- Distance vector to one of the edges E with an adjacent vertex V is $P - (P.\overline{E})\overline{E} - (V - (V.\overline{E})\overline{E})$.
- Distance vector to polygon plane with normal N and a vertex V is $((P - V).\overline{N})\overline{N}$.

The distance vector is thus of the form C+D*t, fragmented over intervals defined by the case that determines the distance vector. Sphylinders and spheres are dengenerate cases of the above. Thus the parameter t can be plugged into a quartic or quadratic implicit equation and solved for explicitly over a non-overlapping interval fragment defined by all contributing primitives. This approach has the advantage of being robust and theoretically precise. There are two major drawbacks of this approach. First, its inapplicability to primitive shapes such as the cone-sphere where the implicit function cannot be represented precisely as a function of the interval parameter t. For cone-spheres we obtain a ratio of two polynomials in t which can only be approximated by a quartic or quadratic polynomials in t. Second, there is interval fragmentation at three levels: Due to polynomial domain if the quadratic approximation is used; within a primitive depending on which of the cases define the distance vector; between contributing primitives to obtain non-overlapping interval spans. For complex scenes the level of interval fragmentation may be upto 8 times the number of contributing primitives, which may make the solution inefficient.

A hybrid solution maybe implemented which employs initial pruning of intervals using the monotonic property, followed by Blinns initial guess heuristic to determine the locality of a solution. ' The explicit solution approach can then be applied to the few intervals that may be generated within the bounds of the guessed interval.

3 Handling Implicit Primitives and their Geometries

The system has been designed in an object oriented fashion. It handles any primitive shape for which a set of functions, which provide geometric and implicit function information about the object, are defined and registered with the system.

The functions return values in unperspected eye space. The following functions are required of each primitive:

- **calcIllumColor(object,point)** : returns the color contributed by **object** at **point**. This is generally dependent only on the normal vector of the object at the given point and not on the actual geometry of the primitive. Common texture map and color functions can be used for all primitives.
- **calcDensity(object,point)** : returns the value of the density due to **object** at **point**. The density calculation involves determining a *distance ratio* for the given point. First the distance from the given point to the central element is calculated. Then, the distance ratio is computed as the ratio of this distance to the radius of influence. The distance ratio is then passed to the density function which returns the density of the primitive at that point in space.
- **calcNormal(object,point)** : returns a weighted normal at **point**, to the surface contributed by **object**, on which **point** lies. The normal vector to an implicit surface $F(x, y, z)$ is given by $\overline{< \frac{\delta F}{\delta x}, \frac{\delta F}{\delta y}, \frac{\delta F}{\delta z} >}$. Thus for functions g_i summed over i primitives the normal vector is $\overline{\sum < \frac{\delta g_i}{\delta x}, \frac{\delta g_i}{\delta y}, \frac{\delta g_i}{\delta z} >}$. For primitives defined with a constant offset distance from a skeleton [3] r (spheres, sphylinders and rounded polygons), with an associated function of distance ratio f, the normal vector contribution for the primitive $< \frac{\delta g_i}{\delta x}, \frac{\delta g_i}{\delta y}, \frac{\delta g_i}{\delta z} >$ is identically $\frac{f'(d)*\overline{N}}{r}$ where d is the distance ratio at **point**, and N is the normal vector to the primitive shape at **point**. For primitives such as the cone-sphere the radius r also changes with **point**. The above form represents the surface normal in terms of normals to individual primitives and allows for effects such as blended textures.

  The above formulation for the quartic function (3) reduces to a very efficient form which is just $\frac{(1-d^2)*N}{r^2}$ where $N = point - C$, C being the skeletal point defining distance to **point**. This computation is required to determine the surface itself. Thus a weighted sum of the surface normals for each primitive based on a function of their contribution to the surface, using an approximate radius of influence gives a resultant normal that is a good visual approximation. This is both efficient and useful for primitives where exact analytical computation of the normal contribution is intractable.
- **calcYExtent(object)** : returns the extent of the **object** in Y (range of scanlines covered).
- **calcXExtent(object,scanline)** : returns the extent of the **object** in X at **scanline** (object span for a scanline).
- **calcZExtent(object,scanline,pixel)** : returns the extent of the **object** in Z at **pixel** (object depth for a pixel on the scanline).

The system currently handles the following primitives: spheres, sphylinders, rounded polygons and cone-spheres. The necessary calculations for each primitive are given in the appendix.

The overall structure of the renderer is then

1. Transform all primitives to eye space.
2. Y-bucket sort the primitives.
3. Maintain an active primitive list and for every scanline compute extent in X for the active primitives.
4. For every pixel compute depth Z spans for all primitives with an active X-extent.
5. Fragment the Z spans into intervals as desired (section 2.2, 2.3). Solve for the intersection point.
6. Compute the normal to the surface at the intersection point and obtain the color as the sum of colors of the individual primitives to the surface weighted by their contribution to the functions value. The color of each primitive is computed using the given values for intersection point and surface normal.

4 Image Space Rendering

A significant portion of the computation is still spent on depth span computation per pixel with the X range for the scanline, especially for complex geometries. Pixel coherence in image space may be used to speed up this computation. A further advantage in image space computation lies in the fact that x and y values for an entire depth span are a constant and so the implicit function can be collapsed to a single variable function in z which may then be solved for explicitly in some cases. Unfortunately the perspective transformation may distort the geometries of primitives in a manner so as to make image space computation infeasible. Quadric surfaces, however, are closed under the perspective transformation.

Under perspective, unclipped spheres transform to ellipsoids. As the eigenvectors of the transformation remain the same over slices of the ellipsoid with a constant Y value, all ellipses of intersection have the same orientation with respect to the XZ plane. This orientation angle need be solved once for the sphere. The center and major, minor axes magnitudes can be computed incrementally in a Bresenham fashion. Given an ellipse specification in the plane, the depth span calculations reduces to the problem of scan-conversion of the ellipse in the XZ plane. Sphylinders are similar to two spheres and the cylindrical component yields congruent ellipses or rectangular cross-sections both of which can be similarly handled. Depth span computation here involves scan-converting up to three component cross-sections within appropriate bounds. Rounded polygons and cone-spheres can also be transformed similarly.

Thus image space computation is an alternative which is a speed up subject to possible geometry restrictions.

5 Implementation and Results

The techniques discussed above have been implemented in C. The timings given in Table 1 are from a SUN IPX. Four images are considered: dog; face; skull; plane (Images 1-4). All timings are given in seconds and were taken using the UNIX *time* command. In each case the object occupies about a 1x1x1 area in world space, centered at the origin with the observer at (5,5,5) and an angle of view of 10 degrees. The pixel resolution is 400x400.

The composition of each image in terms of spheres, sphylinders and cone-spheres and rounded polygons is given. This is followed by the rendering time using scanline processing (referred to as *scan*) and the rendering time using a ray caster, equivalent in complexity to the scanline renderer but without most of the described optimizations. The surface reconstruction time for a polygonization algorithm has been found to be slower than either type of display, in addition to the subsequent rendering time.

Image	*Sph*	*Sphy*	*Csph*	*Rpgn*	*Scan*	*Rtrace*
dog	16	20	0	0	58	387
face	106	0	0	0	75	428
face'	13	30	0	0	66	415
skull	2	20	4	0	45	304
plane	4	3	2	3	28	275

Table 1: Timings (in seconds) for the Images.

6 Conclusion

As can be seen from Table 1, scanline processing of implicit surfaces offers a distinct advantage over standard ray tracing. In addition, intermediate structures can be retained if a single image was being worked on repeatedly. Currently, we are looking at a simplified interactive system that will aid the user in positioning the primitives for object creation.

Primitives are used in a fashion similar to that in Constructive Solid Geometry (CSG) unions. It is also the case that *negative* primitives can be defined in which the function value is subtracted from the sum instead of being added (e.g., Image 2). In our system, the density values of primitives can have an arbitrary *weight*, which may be negative. For example, the hair on the man's head in Image 2 are spheres of weight 3. Weights minimize the number of primitives, as they simulate coincident primitives.

Once negative primitives are allowed, the need for selective blending becomes evident. For example, when gouging out the man's nasal cavity with a negative primitive, one needs to specify that the negative primitive should not subtract from the upper lip. This has led us to the conclusion that a full CSG organization is needed to control primitive combination. We have also found a color weight useful which applied in conjunction with the density contribution determines color contribution of a primitive.

As mentioned before, the use of what amounts to a scanline ray-tracer, which outputs visible z-values at pixels, allows incorporating results of a standard z-buffer so that implicit surfaces and boundary-defined objects can be composited. In addition, the z-spans can be retained so that incremental object modeling can be made fast and efficient.

The power of the introduced primitives sphylinders, cone-spheres and rounded polygons can also be seen (Image 4). These are especially useful for modeling articulated characters. In Table 1 *face'* is generates a visually similar image to *face* (Image 2), using sphylinders to reduce the number of primitives as well as the rendering time.

Finally, accommodating additional primitive types is relatively easy with the system. This is due to the object-oriented nature of its design. Further, as the various scanline functions are primitive specific, efficiency for each primitive can be can be based on its specific properties.

References

[1] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[2] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[3] J. Bloomenthal and K. Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, 1991.

[4] J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(4):109–116, 1990.

[5] M. Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Proc. of SIGGRAPH*, pages 313–320, 1993.

[6] D. Haumann J. Chadwick and R. Parent. Layered construction for deformable animated characters. *Computer Graphics*, 23(3):243–252, 1989.

[7] N. Thalmann J. Gourret and D. Thalmann. Simulation of object and human skin deformations in a grasping task. *Computer Graphics*, 23(3):21–30, 1989.

[8] D. Kalra and A. Barr. Guaranteed ray intersections with implicit surfaces. *Computer Graphics*, 23(3):297–306, 1989.

[9] W. Lorenson and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[10] N. Max. Cone-spheres. *Computer Graphics*, 24(4):59–62, 1990.

[11] G. Wyvill C. McPheeters and B. Wyvill. Data structures for soft objects. *Visual Computer*, 2:227–234, 1986.

[12] P.Hanrahan. Ray tracing algebraic surfaces. *Computer Graphics*, 17(3):83–90, 1983.

[13] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics*, 26(2):185–194, 1992.

[14] A. Fujimoto T. Tanaka and K. Iwata. Arts: Accelerated ray tracing system. *Computer Graphics and Applications*, 6(4):16–26, 1986.

[15] J. Warren. Blending algebraic surfaces. *ACM Transactions on Graphics*, 8(4):263–278, 1989.

[16] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[17] G. Wyvill and A. Trotman. Ray tracing soft objects. *Proc. of Computer Graphics International*, 1990.

A Useful Implicit Primitives

Some comments concerning notation are in order. In the calculations that follow, a bar over a vector variable indicates normalization of the vector. For example, $\overline{X}$ is the unit vector in the direction of X. Upper case letters are used to denote vectors; lower case letters are used to denote scalars. The *dot product* operation is denoted by '$\cdot$'; the *cross product* operation is denoted by '$\otimes$'.

A.1 Spheres

Spheres are defined by a center point C and a radius r. The methods for computing the object functions which are dependent on the geometry of the primitive are given below.

A.1.1 Calculate Distance Ratio and Normal

Given the object information and a point P, the distance ratio is the ratio of the distance between the point and the center to the radius of the sphere. $distanceratio = \frac{|P-C|}{r}$.

Given the object information and a point P, the normal is the normalized vector N from the center of the sphere to the point: $N = \overline{P - C}$ (Figure 2).

A.1.2 Calculate Y-Extent

Given the object information, the Y-Extent can be calculated by forming the highest and lowest planes whose normals are in the YZ plane and which pass through the origin. In order to find the plane, the YZ projection of the plane (a line) and the sphere (a circle) is used. Points $P1$ and $P2$ are found on the lines through the origin tangent to the circle such that the line joining $P1$, C and $P2$ is perpendicular to the line from the origin to C. The y coordinates of $P1, P2$ give the scanline extent when projected onto the picture plane (Figure 2).

1. $V = \overline{<1,0,0> \otimes C}$
2. $k = \frac{r|C|}{sqrt(|C|^2 - r^2)}$
3. $P1 = C + kV, P2 = C - kV$

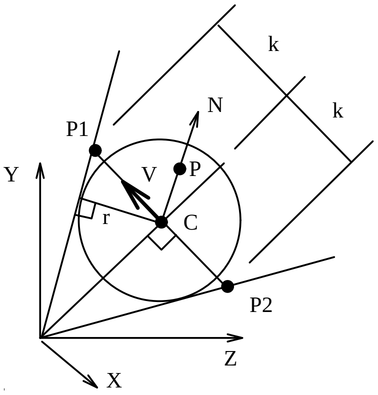

Figure 2: SPHERE: Normal, Y-Extent Calculation

A.1.3 Calculate X-Extent

For spheres, calculating the X-extent is similar to the Y-extent. A given scanline defines a plane in eye-space. The intersection of the plane, with unit normal N, and the sphere in eye-space is a circle of center C' and radius r' (Figure 3).

1. $s = C \cdot N$
2. $C' = C - sN$
3. $r' = sqrt(r^2 - s^2)$
4. calculate X-extent using C', r'

A.1.4 Calculate Z-Extent

For each pixel of each scanline covered by the sphere, the points of intersection of the ray with the sphere, $I1$ and $I2$, can be computed, given a unit ray through the pixel, RAY. Calculation of a midpoint, $MIDP$, allows for breaking the extent into two monotonic depth spans (Figure 3).

1. $d = C' \cdot RAY$
2. $MIDP = dRAY$
3. $middist = |C - MIDP|$
   (used in calculating monotonic z-spans)
4. $m = sqrt(r'^2 - |C' - MIDP|^2)$
5. $I1 = (d - m)RAY, I2 = (d + m)RAY$

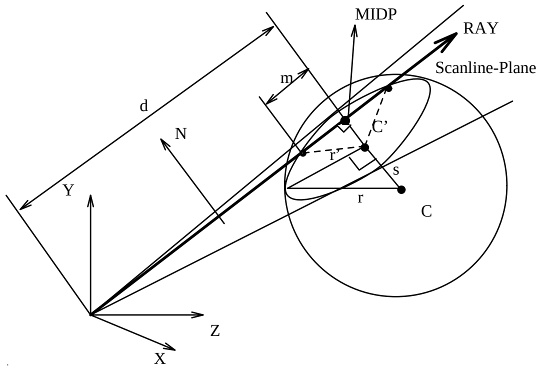

Figure 3: SPHERE: X-Extent, Z-Extent Calculation

A.2 Sphylinders

Sphylinders are finite extent cylinders with hemispheres capping both ends. They are useful for modeling a variety of shapes, especially longitudinal ones. A sphylinder is defined by two center points, $C1$ and $C2$, and a radius r (Figure 4). The line segment between $C1$ and $C2$ will be referred to as the *center axis.*

A.2.1 Calculate Distance Ratio and Normal

Given the object information and a point P, the distance ratio is computed as in the case of the sphere, if the closest point on the center axis to P is one of the end points. If the closest point to P, call it P', is interior to the center axis, then the distance ratio is calculated as the ratio of the distance between P and P' to the radius of the sphylinder. The normal is the normalized vector N from P' to P (Figure 4):

1. $P' = C1 + (P - C1) \cdot \overline{(C2 - C1)} * \overline{(C2 - C1)}$.
2. $distanceratio = \frac{|P - P'|}{r}$.
3. $N = \overline{P - P'}$.

A.2.2 Calculate Y-Extent

The Y-extent is calculated by computing the Y-extent of its end spheres and returning their combined Y-extent (Figure 4).

A.2.3 Calculate X-Extent

The X-extent, unlike the Y-extent, can either be delimited by the spheres or by the cylinder. The X-extents of the scanline-plane with the capping spheres are calculated. If the plane does not intersect the spheres, the X-extent of the scanline-plane (with normal N) and the infinite cylinder is computed as follows. Note that in *eye-space* a scanline-plane passes through the origin. It is

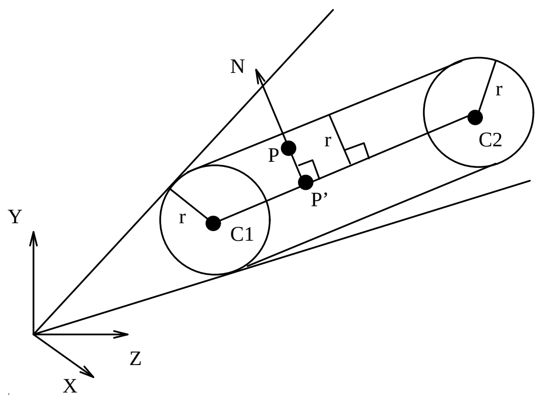

Figure 4: SPHYLINDER: Normal, Y-Extent Calculation

assumed that $C1$ is below the scanline-plane with no loss of generality.

1. $Y' = \overline{N \otimes (C2 - C1)}$
2. $X' = \overline{Y' \otimes N}$
3. $majoraxis = \frac{r}{|(C2-C1) \cdot N|}$
4. $minoraxis = r$
5. $C' = C1 + \frac{-N \cdot C1}{N \cdot (C2-C1)} * (C2 - C1)$ (Figure 5)
6. The intersection ellipse is defined by center, C', major and minor axes of symmetry, X' and Y', and their magnitudes,$majoraxis$ and $minoraxis$. Transform the origin to the local frame defined by axes of symmetry of the intersection ellipse centered at C' and solve the quadratic equation for the tangent points to the ellipse. The tangent points transformed back form the X-extent with the cylinder (Figure 6).

A.2.4 Calculate Z-Extent

As is the case with the sphere, the Z-extents of the sphylinder can be broken into monotonic segments. The closest point on the ray to the sphylinder's defining line is determined and the Z-extent is broken into the two segments at that point. The ray need only be intersected with the components of the sphylinder that define the surface of intersection in the scanline-plane. Thus the following check on X-extent definition can be made (Figure 7):

- Single defining component: Intersect ray with that component.
- Sphere and cylinder definitions: Intersect ray with that sphere and the cylinder and take the extremes.
- Spheres' definitions: Intersect ray with both spheres and the cylinder and take the extremes.

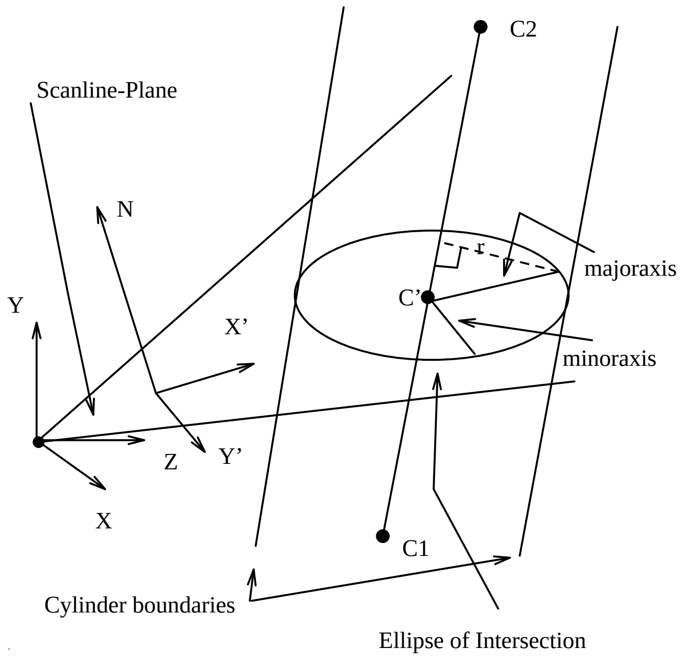

Figure 5: SPHYLINDER: X-Extent Calculation

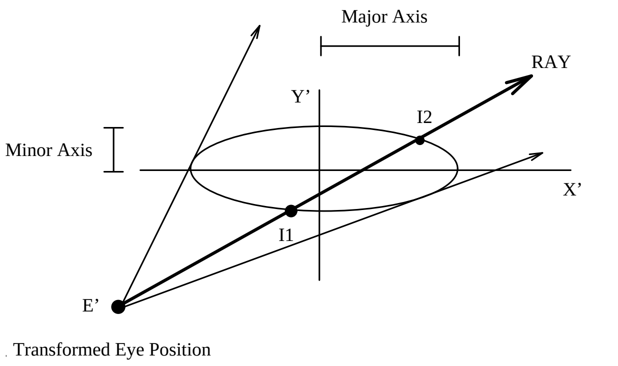

Figure 6: 2D Ellipse Tangent, Ray Intersection Calculation

The ray infinite cylinder intersection can be computed efficiently by using the 2D intersection ellipse information above to transform the ray and calculate the 2D ray ellipse intersection. A special case where the scanline plane is parallel to the sphylinder axis is handled as follows.

1. $d = C1 \cdot Y'$.
2. $d' = \sqrt{r^2 - (C1 \cdot N)^2}$.
3. $\frac{(d \pm d')}{RAY \cdot Y'}$ are distances of intersection along RAY.

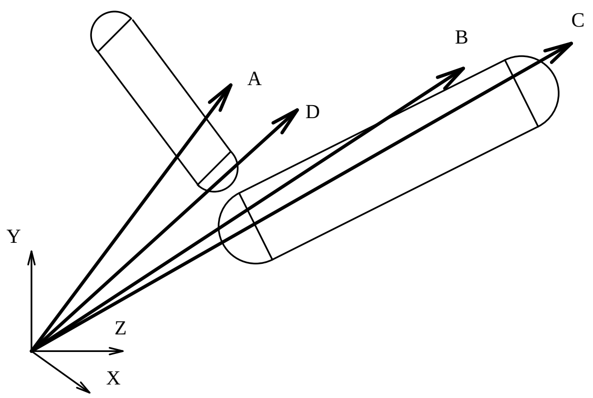

Figure 7: Ray Intersections: A) Cylinder-Cylinder B) Sphere-Cylinder C,D) Sphere-Sphere

A.3 Cone-Sphere

Cone-Spheres are a generalization of sphylinders in which the spherical segments on either end have unequal radii and the connecting surface is a truncated cone instead of a cylinder; interior radii are linear interpolations of the end radii. The cone is defined so that the surface has first order continuity at the junction between the cone and spherical segments. This means that the spherical segments at either end are only hemispheres in the case that degenerates to a sphylinder. Cone-spheres are also useful for modeling a variety of shapes, especially articulated animate figures.

Cone-Spheres are defined by two center points, $C1$ and $C2$, and two corresponding radii, $r1$ and $r2$. Using the above information, the following is computed (Figure 8):

- $\theta = cos^{-1}(\frac{r2-r1}{|C2-C1|})$
- $C21 = \overline{(C2 - C1)}$
- $C0 = C1 - C21 * \frac{r1}{cos\theta}$

A.3.1 Calculate Distance Ratio and Normal

Given the object information and a point P, the distance ratio, *distratio*, and normal, N, computation is as follows (Figure 8):

1. $P' = P - C0$
2. $t' = (\sqrt{|P'|^2 - (P' \cdot C21)^2})cotan\theta$
3. $t = P' \cdot C21 + t'$
4. $N = \overline{P' - t * C21}$
5. $distratio = \frac{t' * sec^2\theta}{t}$

A.3.2 Calculate Y-Extent

The Y-extent, as in the case of the sphylinder, is the combined extent of the end spheres.

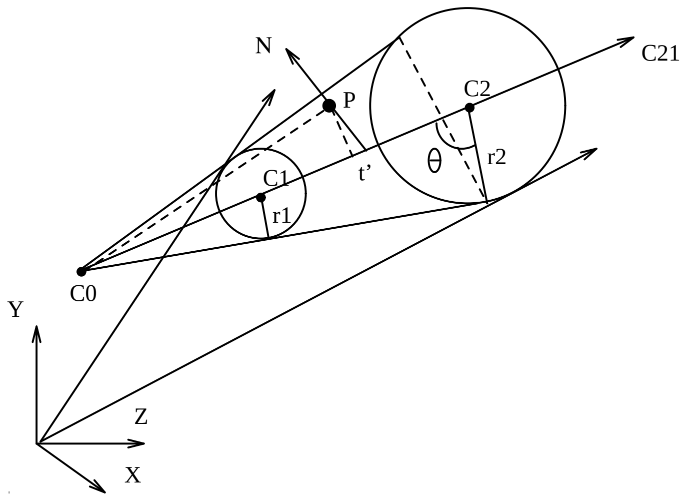

Figure 8: CONE-SPHERE: Normal, Y-Extent Calculation

A.3.3 Calculate X-Extent

The X-extent can either be delimited by the spheres or by the cone. The X-extents of the scanline-plane with the capping spheres are calculated. If the plane does not intersect the spheres the X-extent of the scanline-plane with the semi-infinite cone is computed as follows:

1. $\phi = cos^{-1}(X' \cdot C21)$ (Figure 9)
2. The conic of intersection is determined as follows:
   - Ellipse : $\phi > 90^o - \theta$
   - Hyperbola : $0 < \phi <= 90^o - \theta$
   - Parabola : $\phi = 0^o$
3. Axes of symmetry ,X' and Y', and, in the case of ellipse and hyperbola, intersection point, C', are calculated as in the case of the sphylinder.

The 3 cases are now treated individually for clarity,though some of the computation may overlap. The computations are similar to those used for sphylinders.

- **ELLIPSE**
  - $t = |C' - C0| * cos\theta$
  - $distC'P1 = \frac{t}{cos(\theta - \phi)}$
  - $distC'P2 = \frac{-t}{cos(\phi + \theta)}$
  - $P1 = C' - distC'P1 * X'$
  - $P2 = C' + distC'P2 * X'$
  - $MIDP = \frac{P1+P2}{2}$
  - $majoraxis = |MIDP - P1|$
  - $t' = (MIDP - C') \cdot C21$
  - $MIDP' = C' + t' * C21$
  - $rMIDP' = |MIDP' - C0| * cotan\theta$
  - $d = sqrt(|MIDP - C'|^2 - t'^2)$
  - $minoraxis = sqrt(rMIDP'^2 - d^2)$

  Solve for X-extent of the ellipse of intersection defined by the axes of symmetry centered at $MIDP'$ as in the case of the sphylinder.

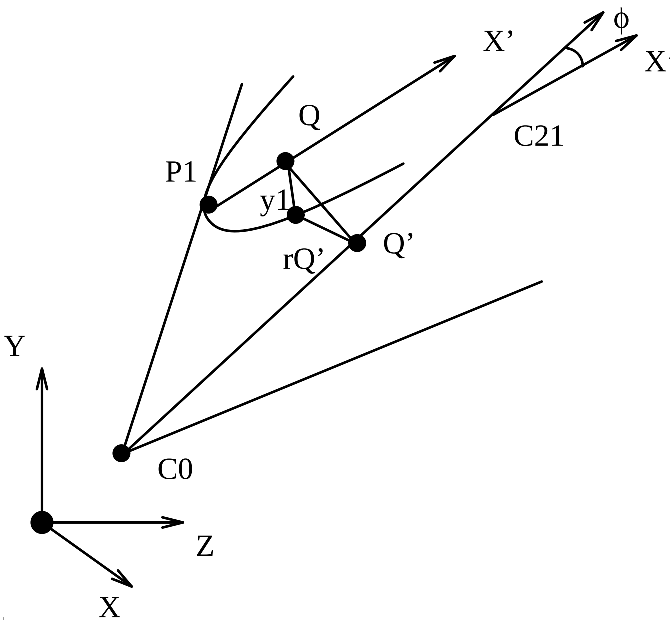

Figure 9: Hyperbolic Intersection of Cone/Plane

- **HYPERBOLA**
  - Calculate P1 as for ellipse
  - The general equation of a hyperbola symmetric about positive X-axis, passing through the origin is :
    $y^2 = a^2x^2 - 2abx$. The defining constants a, b are obtained by computing points $(1, y1)$ and $(2, y2)$ on the hyperbola and solving the linear equations to get
    $a^2 = \frac{y2^2 - 2*y1^2}{2}$
    $ab = \frac{y2^2 - 4*y1^2}{4}$
  - $y1^2, y2^2$ are computed as follows (Figure 9):
    * $Q = P1 + 1 * X'$
    * $Q' = C0 + ((Q - C0) \cdot C21)C21$
    * $rQ' = |Q' - C0|cotan\theta$
    * $y1^2 = rQ'^2 - |Q - Q'|^2$
    * $y2^2$ is calculated similarly by a distance of 2 along X'
  - We thus obtain the definition of the intersection hyperbola with axes X', Y', constants a^2, ab and local origin $P1$.
  - Tangent calculations after transforming the origin to local space results in a quadratic equation of whose solutions one will lie within the bounds of the truncated cone (Figure 10).

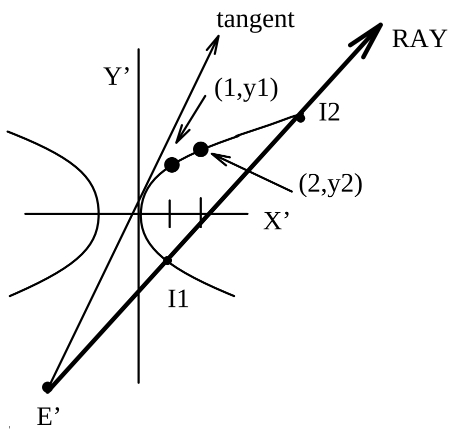

Figure 10: 2D Hyperbola Tangent, Ray Intersection Calculation

- **PARABOLA**
  - $d = (N \cdot C0)tan\theta$

- $P1 = C0 + d * C21 + (N \cdot C0)N$
- The general equation of a parabola symmetric about positive X-axis, passing through the origin is :
  $y^2 = a^2 x$. The defining constant a is obtained by computing point $(1, y1)$ on the parabola giving $a^2 = y1^2$.
- $y1^2 = ((d+1) * cotan\theta)^2 - (N \cdot C0)^2$
- We thus obtain the definition of the intersection parabola with axes X', Y', constant a^2 and local origin $P1$.
- Tangent calculations after transforming the origin to local space results in a quadratic equation of whose solutions one will lie within the bounds of the truncated cone.

A.3.4 Calculate Z-Extent

Z-Extent calculation is similar to that of the sphylinder. The ray intersection for the infinite cylinder component (accomplished by a 2D ray, intersection ellipse, intersection), however, must be replaced for the infinite cone component by 2D intersection with the appropriate conic of intersection the definition of which is made once for the scan-line, while computing X-Extents.

A.4 Rounded Polygons

Rounded Polygons are defined by a list of vertices $C_1, .., C_n$ defining a planar polygon and a radius r (Figure 11). For the most part all the computation is done by treating the rounded polygon as n sphylinders with consecutive vertices forming the two centers and a radius r, within appropriate bounds. Care should be taken not to repeat the computation for the region common to two adjacent sphylinders.

Further computation is required for the case when the surface normal and Z spans are specified by the polygon plane. Let N be the normal to the plane. Given a point P for which the plane defines the normal and distance ratio, the normal is N and $distratio = \frac{(P - C_1) \cdot N}{r}$. For a RAY whose Z spans are defined by the plane the distances of intersection along RAY are $\frac{(C_1 \cdot N) \pm r}{RAY \cdot N}$.

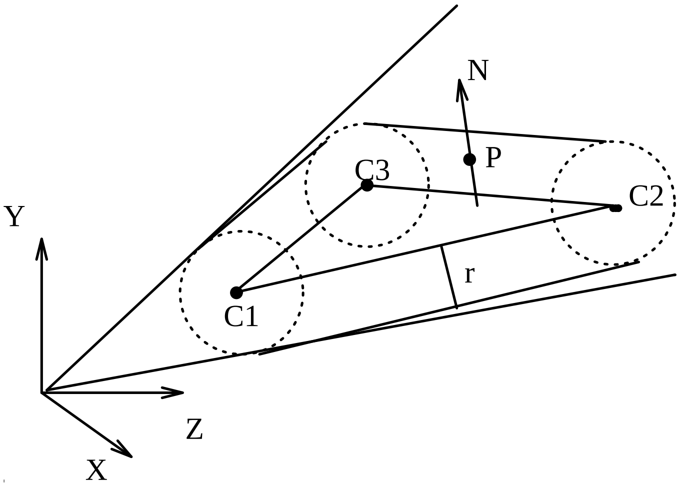

Figure 11: ROUNDED POLYGON: Normal, Y-Extent Calculation