Ray Tracing Objects Defined by Sweeping Planar Cubic Splines

JARKE J. VAN WIJK
Delft University of Technology

The crucial step in a program based on ray tracing is the calculation of the intersection of a line with an object. In this paper, algorithms are presented for performing this calculation for objects defined by sweeping a planar cubic spline through space. Translational, rotational, and conic sweeping are treated. Besides solutions for the exact calculation, rectangle tests for improving efficiency are given. Possible extensions and improvements are discussed.

Categories and Subject Descriptors: I.3.3 [**Computing Methodologies**]: Computer Graphics—*picture/image generation; computational geometry and object modeling; three-dimensional graphics and realism*

General Terms: Algorithms

Key Words and Phrases: Image synthesis, raster graphics, ray tracing, solid modeling, sweeping

1. INTRODUCTION

Among available techniques for rendering shaded images, ray tracing is the most flexible and powerful. It provides a very simple and elegant way to handle optical effects such as cast shadows, reflection, and refraction. It has also proved to be a useful tool for rendering solid models [9] by reducing the complex task of evaluating objects defined by constructive solid geometry (CSG) to a one-dimensional problem, which can be solved in a simple and straightforward way.

The crucial step in ray casting is the calculation of the intersection of a ray (line) with a surface or solid. Intersection procedures have been published for several kinds of surfaces, including polygons and simple analytic surfaces [1, 4, 9], general algebraic surfaces [5], surfaces with a superimposed density distribution [2], cubic patches [6], and procedurally defined objects [7].

In [7] the intersection problem of a line with fractal surfaces and with simple sweep-defined objects is discussed. A sweep-defined object can be described as an object that is created by sweeping a two-dimensional contour along a trajectory. A possible solution to the intersection problem would be to convert the surface representation to polygons or patches and then apply one of the standard

This work was supported by the Delfts Hogeschoolfonds.
Author's address: J. J. van Wijk, Department of Industrial Design, Delft University of Technology. Oude Delft 39a, 2611 BB Delft, The Netherlands.

techniques. However, by taking advantage of the geometrical and topological properties of this class of objects, more efficient algorithms can be developed. Kajiya gives a solution for translational sweeping (prisms) and rotational sweeping (surfaces of revolution) in which the contour is described by a strip tree. The main principle is to reduce the three-dimensional intersection problem to a two-dimensional problem. Another discussion of this principle can be found in [10].

This paper discusses sweep-defined objects with a cubic spline as contour. Rotational and translational sweeping, as well as conic sweeping, are treated.

In the next section a general review of the problem and its solution is given. The shape definition is described, as well as a global outline of the intersection algorithm. The following sections deal with the detailed solution for translational (Section 3), conic (Section 4), and rotational (Section 5) sweeps. In Section 6, the implementation of the algorithms is described and examples of pictures are shown. In the final section the results are discussed and suggestions for further refinements are made.

2. OBJECTS DEFINED BY SWEEPING

A simple sweep-defined object can be represented by three entities:

—a specification of the type of sweeping,
—a transformation matrix,
—a plane contour.

The first item is one of the three types: translational, rotational, or conic sweeping. Figure 1 shows three objects, generated by performing these three operations on the same contour. In the following sections, formal descriptions will be given.

In line with the instancing technique, as described by Roth [9], each object is defined in a local axis frame in a standard way. For example, in translational sweep all contours lie initially in the base $z = 0$ and are translated in the z-direction to lie finally in the plane $z = 1$. Thus it is possible to use a local, standard intersection algorithm. The user can translate and rotate, as well as scale and skew the objects. Information about a particular object is stored in an associated 4×4 homogeneous matrix M, which defines the transformation from world to local coordinates. When a picture has to be made, the view matrix (transformation from view to world coordinates) is concatenated with M, so that a single transformation of a line is sufficient to obtain its local counterpart.

The contour is a description of the cross section of the object. In the implementation, the contour is entered as a polygon in local (u, v)-coordinates, along with a specification of the kind of curve desired. A choice can be made between

—a line contour, the polygon itself;
—a cubic B-spline, a curve approximating the polygon with second-order continuity;
—a cubic Catmull–Rom spline, a curve passing through the vertices with first-order continuity.

Because the intersection procedures operate on the piecewise polynomial description of the splines, the input is converted into a linear list of segments,

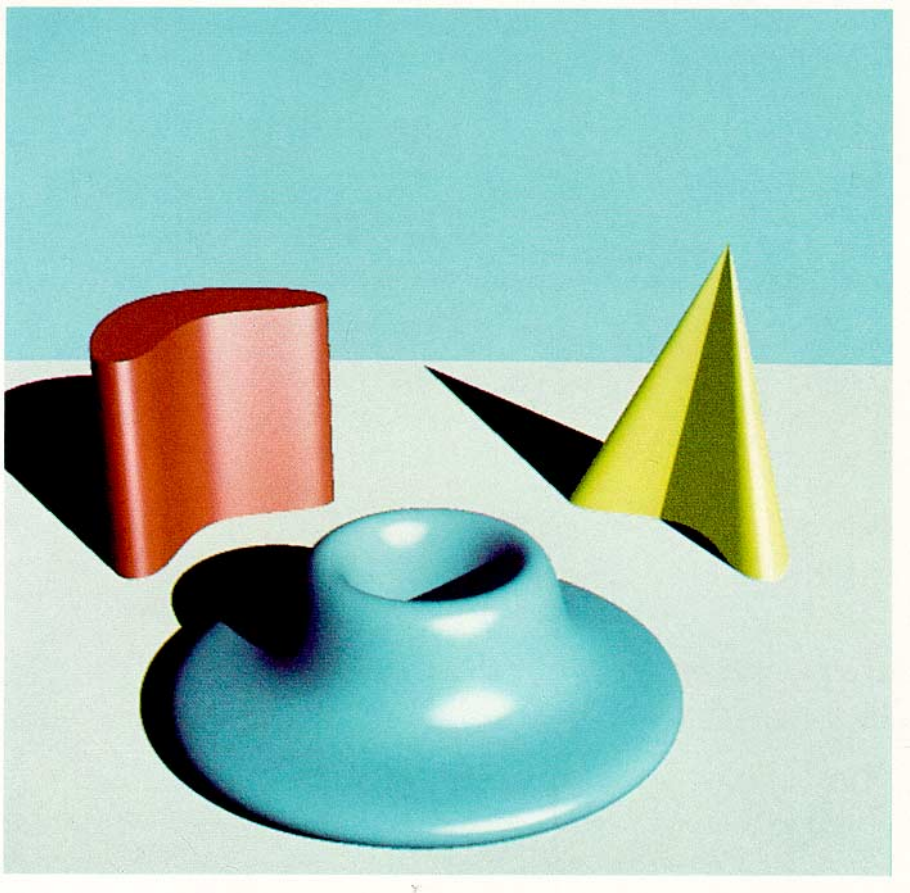

Fig. 1. Objects generated by performing translational, rotational, and conic sweeping on the same contour.

each defining a piece of the curve. A segment contains

—the degree of the two polynomials $u(s)$ and $v(s)$, 1 or 3;
—the coefficients of $u(s)$ and $v(s)$;
—a rectangle $(u_{min}, u_{max}, v_{min}, v_{max})$, enclosing the relevant piece of the curve.

The curve itself is defined by the set of points $(u(s), v(s))$ with $0 \leq s < 1$. An object does not contain a description of the contour, but a pointer to the first element, which enables several objects (with different M's) to point to the same contour, resulting in considerable savings in memory for regular patterns of objects.

Just as it is possible to give a global description of a sweep-defined object, so is it equally possible to give a general outline of the intersection algorithm. The algorithm consists of the following steps:

—transform the ray to local coordinates;
—project the ray on the local (u, v) contour plane;

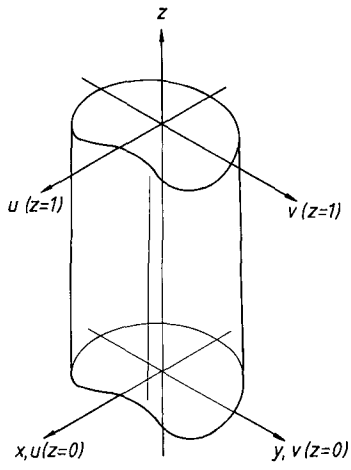

Fig. 2. Translational sweeping.

—calculate the intersections of the projected ray with the contour;
—determine, if necessary, intersections with the base and cap plane;
—calculate normals at intersection points;
—transform the results back to view coordinates.

In the following sections, this algorithm is elaborated for each type of object.

3. TRANSLATIONAL SWEEPS

The local geometry of an object defined by translational sweeping is shown in Figure 2. It consists of three parts:

—a base plane, consisting of the points inside the contour, with the u- and v-axis coinciding with the x- and y-axis, and $z = 0$;
—a cap plane, defined similarly, but with $z = 1$;
—the side planes of the object.

A transformed ray is defined by the vector equation

$$[r_x, r_y, r_z] = [o_x, o_y, o_z] + [d_x, d_y, d_z]t. \tag{1}$$

Determination of the intersections of this ray with the object is done as follows. First, the intersection values t_1 and t_2 of the ray with the planes $z = 0$ and $z = 1$ are calculated from

$$t = -\frac{o_z}{d_z}, \qquad t = 1 - \frac{o_z}{d_z}.$$

These values are stored in increasing order in an array IP of intersection points. We denote the smaller value by $t_{\min}$ and the larger by $t_{\max}$. Kajiya [7] proposes an *inside contour* test to check whether these points are valid intersection points. This test, however, can be omitted, as is shown later.

In the next step, the intersection points with the side planes are determined by processing the segments that occur in the list.

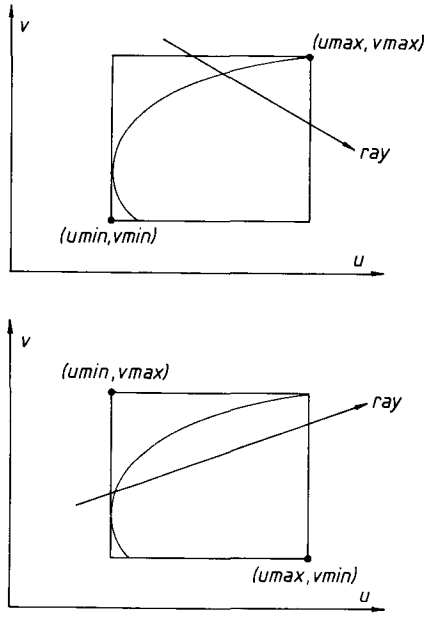

Fig. 3. Rectangle test applied for translational and conic sweeping.

The projection of the ray on the (u, v)-plane is

$$d_y u - d_x v + d_x o_y - d_y o_x = 0. \tag{2}$$

First, a simple rectangle test is carried out to determine whether this ray intersects a segment. Denoting the left-hand side of (2) by $P(u, v)$, then if $d_x d_y > 0$, intersection occurs if $P(u_{\min}, v_{\max})$ and $P(u_{\max}, v_{\min})$ have different signs. If $d_x d_y < 0$, intersection occurs if $P(u_{\min}, v_{\min})$ and $P(u_{\max}, v_{\max})$ have different signs (Figure 3). When intersection occurs, the polynomials $u(s)$ and $v(s)$ of the segment are substituted in (2), resulting in a polynomial in s. Because the maximum degree of this polynomial is 3, the roots can be found analytically. Coincident roots—corresponding with the ray touching the object—can be ignored. For each root s_0 satisfying $0 \leq s_0 < 1$,

$$t = \frac{u(s_0) - o_x}{d_x}$$

is inserted into its proper ordered position in the array of intersection points provided that $t > t_{\min}$.

After each segment has been tested in this way, a sorted array of n intersection points is obtained. We denote the position of $t_{\max}$ in the array with $i_{\max}$. For simplicity, in the remainder of this section $t_{\max}$ is assumed to be the intersection point with the plane $z = 0$. Selection of the valid intersection points is done with the following algorithm:

```
—if odd(n − i_max)
  then n := i_max          {delete intersections beyond plane z = 0}
  else n := i_max − 1;     {delete intersection with plane z = 0 and}
                           {intersections beyond it}
```

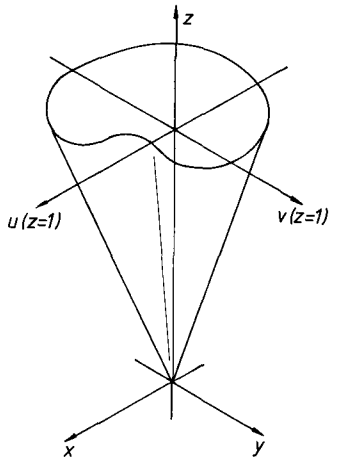

Fig. 4. Conic sweeping.

```
—if odd(n)
  then
  begin n := n − 1;
        for i := 1 to n do IP[i] := IP[i + 1]
  end     {delete intersection with z = 1}
```

This algorithm is based on the fact that the spaces defined by $z \geq 0$, $z \leq 1$, and the area within the (extended) side planes can be viewed as half spaces. The object itself can then be viewed as the intersection of these three half spaces. If the number of intersections of the ray with the side planes beyond t_{max} is odd, then the intersection point with the plane $z = 0$ is inside the half space of the side planes and thus a valid point. If this is so, the base point has to be omitted. The first point is treated slightly differently. If the resulting number of intersection points is odd, then it is deleted.

There are two cases for which the preceding method breaks down: $d_x = 0$ and $d_y = 0$ (top view), and $d_z = 0$ (side view). A practical solution is to give these parameters, if necessary, a very small absolute value and process them further in the standard way.

Calculation of the local normals $[n_x, n_y, n_z]$ at the intersection points is trivial. For points on the side planes, the corresponding root s_0 and the differentials of the polynomials $u(s)$ and $v(s)$ are used:

$$[n_x, n_y, n_z] = \left[\frac{-dv(s_0)}{ds}, \frac{du(s_0)}{ds}, 0\right].$$

4. CONIC SWEEPING

With translational sweeping, the size of the contour is constant as it is swept parallel to the z-axis. An obvious extension of the simple translation sweep is to combine scaling of the contour, for example, by multiplying the x- and y-coordinates by z, with translation. This is the basic idea of conic sweeping. Figure 4 shows an object resulting from a conic sweep. It consists of two parts: the cap plane, defined as for translational sweeping, and the side planes. The local contour axis frame (u, v) is a function of z. The position of a point $[x, y, z]$ on a

side plane can be defined in (u, v)-coordinates by

$$xv = yu, \tag{3}$$

$$x = zu. \tag{4}$$

Equation (3) means that the point is moved toward the z-axis, while (4) gives the scale factor z.

If we define a ray as in (1) and substitute it in (3), we get

$$v(d_x t + o_x) = u(d_y t + o_y).$$

Writing t explicitly yields

$$t = \frac{o_y u - o_x v}{d_x v - d_y u}. \tag{5}$$

Substituting (1) in (4), we find

$$o_x + d_x t = (o_z + d_z t)u.$$

Substitution of (5) into the above yields (6)

$$(o_y d_z - o_z d_y)u - (o_x d_z - o_z d_x)v + (o_x d_y - o_y d_x) = 0. \tag{6}$$

Thus, the projected ray can be written as a linear equation in u and v, which can be processed in the same way as for translational sweep. Calculation of t can be done with (5). Treatment of the cap plane is simpler than for translational sweeping. Only points satisfying $t_{\min} \leq t < t_{\max}$ (with $t_{\min}$ and $t_{\max}$ as defined in Section 3) are stored in the intersection point array. If the total number of intersection points is odd, then the cap plane point is included.

Calculation of the normals is somewhat more complex than for translational sweep (Figure 5). First, $[n_x, n_y, n_z]$ is calculated as in Section 3 and normalized. Next, n_z is determined via

$$n_z = -(u^2(s_0) + v^2(s_0))^{1/2}$$

with the sign changed if

$$n_x u(s_0) + n_y v(s_0) < 0.$$

Figure 5 shows that this ensures that a_1 (angle between the ruler on side and z-axis) is identical to angle a_2 (tilt of the normal with respect to the plane $z = 0$).

5. ROTATIONAL SWEEPING

The last type to be discussed is rotational sweeping, illustrated in Figure 6. The local v-axis is coincident with the z-axis, while the u-axis rotates in the XOY-plane. The problem now is to project a ray, as defined in (1), onto the (u, v)-plane. Consider the (u, v)-plane when the angle between u- and x-axis has a certain value θ. What do we know about the intersection point $[x, y, z]$ of the ray with the (u, v)-plane? First, since $v = z$, then

$$t = \frac{v - o_z}{d_z}. \tag{7}$$

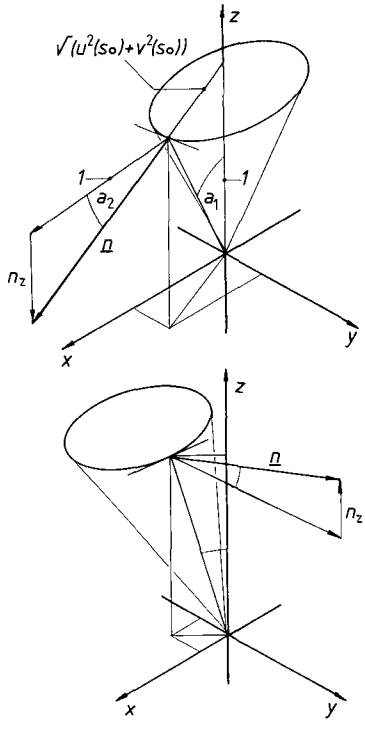

Fig. 5. Calculation of the normal for conic sweeping. In the case shown right, the sign of *nz* is changed.

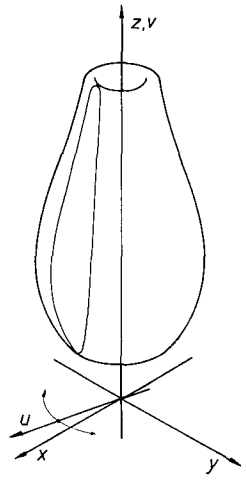

Fig. 6. Rotational sweeping.

Further, the distance of the point $[x, y, z]$ from the z-axis is u, so

$$u^2 = x^2 + y^2 \qquad \text{or} \qquad u = (o_x + d_x t)^2 + (o_y + d_y t)^2. \tag{8}$$

Substitution of (7) into (8) yields

$$av^2 + bv + c - du^2 = 0 \tag{9}$$

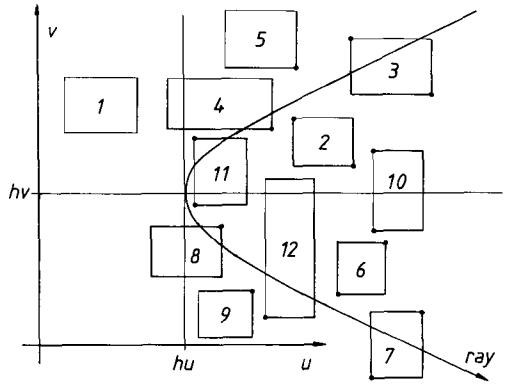

Fig. 7. Projected ray for rotational sweeping, with several rectangles. Numbers in rectangles refer to the algorithm in Sect. 5, thick edges and dotted vertices denote tested items.

with

$$a = d_x^2 + d_y^2,$$

$$b = 2(-o_z a + d_z(d_x o_x + d_y o_y)),$$

$$c = (d_x o_z - d_z o_x)^2 + (d_y o_z - d_z o_y)^2,$$

$$d = d_z^2.$$

Again, an equation in u and v, now of second degree, is obtained. Substitution of the two spline polynomials of a segment yields an equation of sixth degree, which has to be solved numerically. In this implementation the method of Laguerre [8] is used, together with synthetic division to decrease the degree of the polynomial when roots are found. Characteristics of the Laguerre method are stability, fast convergence, and location of all roots, real and complex. The first two properties led to its implementation here.

To bypass unnecessary determinations of and working with sixth-degree polynomials, we use a simple rectangle test, requiring in the worst case ten multiplications and six additions. Equation (9) defines a hyperbola. We need only consider $u \geq 0$, since the computation is simplified by restricting u to nonnegative values. The base point (h_u, h_v) of the hyperbola is given by

$$h_u = \left(\frac{c - b^2/(4a)}{d}\right)^{1/2},$$

$$h_v = \frac{-b}{2a}.$$

Figure 7 shows the geometry of the projected ray, along with a number of relative positions of the rectangle. The test applied, assuming that the values for a, b, c, d, h_u, and h_v are defined, is then

```
Function Outrect: boolean;
{Outrect is true if the rectangle (u_min, u_max, v_min, v_max)}
{is not intersected by the projected ray with}
{equation a * v * v + b * v + c - d * u * u = 0}
```

```
  Function Inshyp(u, v : real) : boolean;
  {Inshyp is true, if the point with coordinates}
  {(u, v) lies inside the hyperbola}
  begin
    Inshyp := (a * v + b) * v + c − d * u * u < 0
  end;
begin
  if u_max < h_u then Outrect := true                    {case 1}
  else
  if v_min > h_v then
  begin
    if Inshyp(u_max, v_min) then
    begin
      if u_min > h_u
      then Outrect := Inshyp(u_min, v_max)               {case 2, 3}
      else Outrect := false                              {case 4}
    end else Outrect := true                             {case 5}
  end else
  begin
    if v_max < h_v then
    begin
      if Inshyp(u_max, v_max) then
      begin
        if u_min > h_u
        then Outrect := Inshyp(u_min, v_min)             {case 6, 7}
        else Outrect := false                            {case 8}
      end else Outrect := true                           {case 9}
    end else
    if Inshyp(u_min, v_min)
    then Outrect := Inshyp(u_min, v_max)                 {case 10, 11}
    else Outrect := false                                {case 12}
  end
end;
```

If an intersection point is found, the resulting value of t is given by (7). The normal on the surface can then be determined as follows. First, the normal lies in the plane through the z-axis and the intersection point, so we can state for the projection of the normal on the plane $z = 0$:

$$n_x = o_x + d_x t, \qquad n_y = o_y + d_y t.$$

To determine n_z, we first calculate a normal on the contour:

$$n_u = \frac{-d(v(s_0))}{ds}, \qquad n_v = \frac{d(u(s_0))}{ds}.$$

The z-component of the normal follows then from

$$\frac{n_z}{(n_x^2 + n_y^2)^{1/2}} = \frac{n_v}{n_u}, \qquad \text{or} \qquad n_z = \frac{(n_x^2 + n_y^2)^{1/2} n_v}{n_u}.$$

This is illustrated in Figure 8.

6. RESULTS

The routines for input and intersection calculation for sweep defined objects are embedded in an experimental package for solid modeling. The package was developed at the Delft University of Technology and is based on Roth [9]. It can

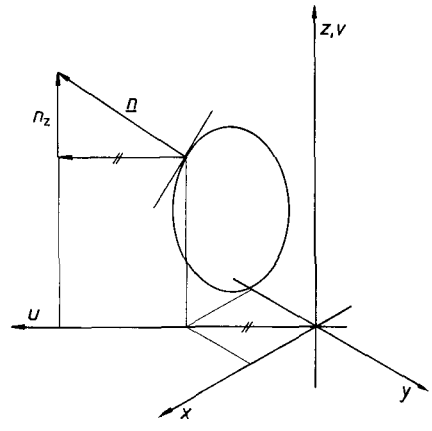

Fig. 8. Calculation of normal for rotational sweeping.

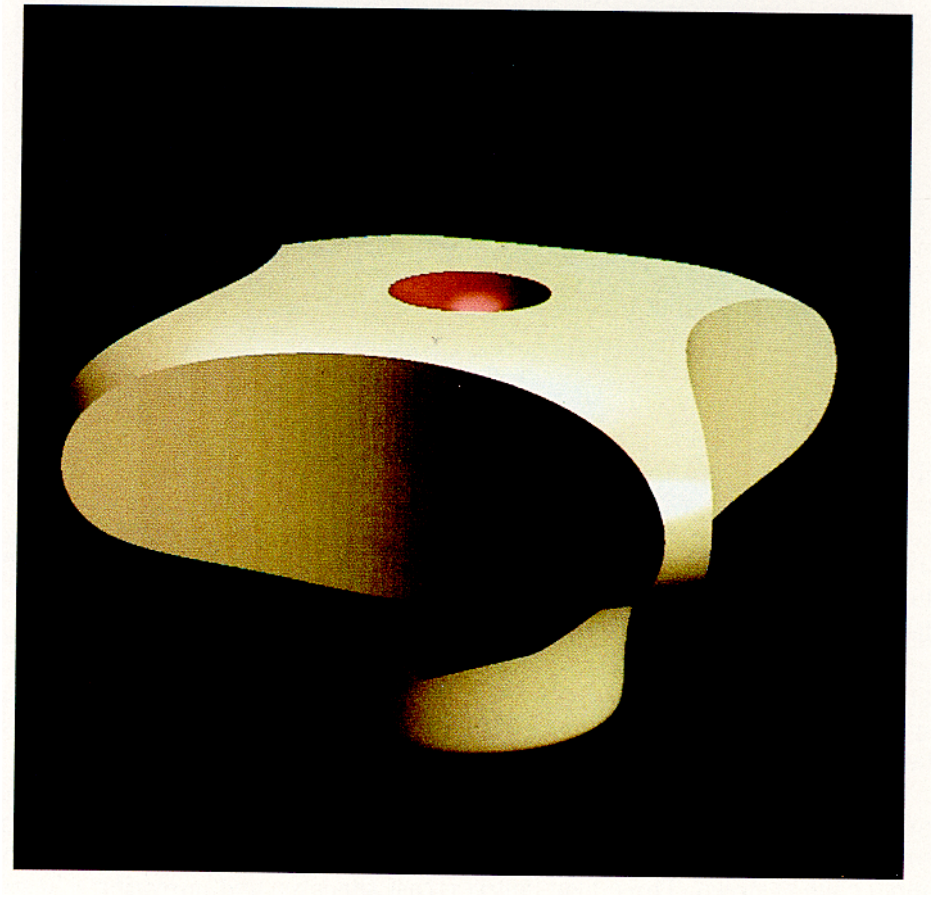

Fig. 9. Control knob.

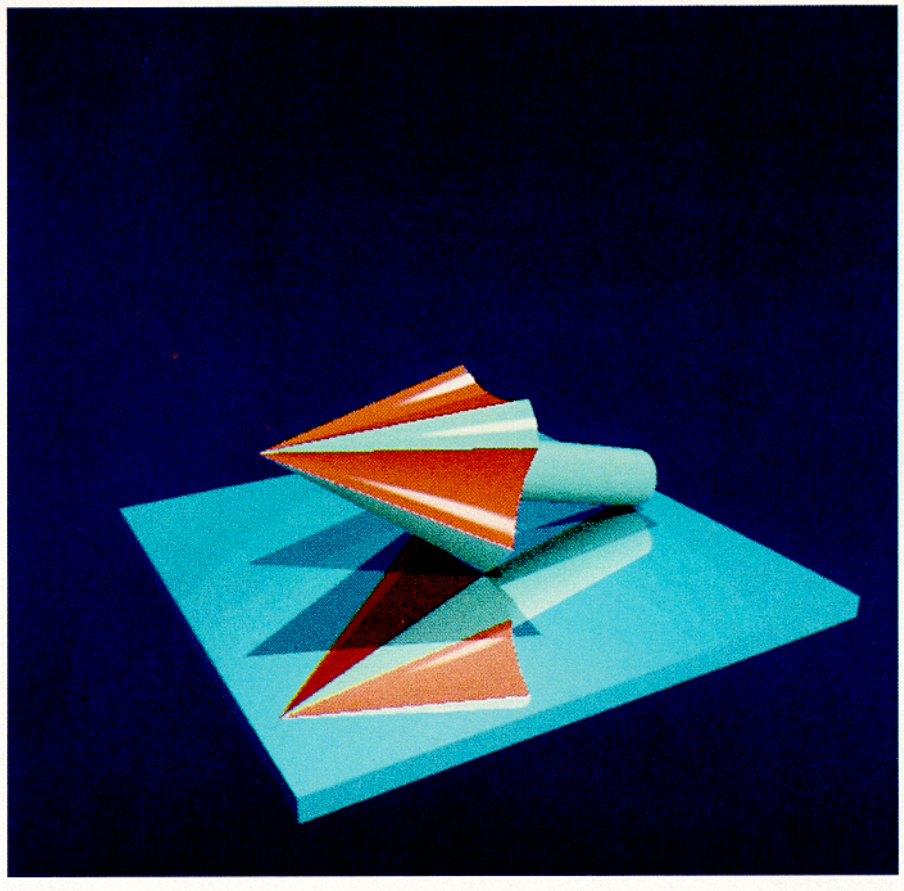

Fig. 10. Countersink bore.

thus be described as a solid modeler working with a CSG representation which is evaluated by ray casting. Instances implemented are cube, sphere, cylinder, cone, and torus, to which the sweep defined objects are a considerable extension. The first application of the modeler was the generation of shaded pictures. The luminance model is similar to the one described by Whitted [12], and supports features like specular and mirroring reflection, cast shadows, and transparency.

The package is written in Pascal and runs on a DEC PDP 11/44 under RSX-11M. The graphics device used is a Grinnell GMR 270 image processing display system with a $512 \times 512 \times 8$ frame buffer. The pictures shown in this paper were taken directly from its screen.

Figure 9 shows a control knob. It was modeled by taking the intersection of a rotational and a translational sweep. The red mark was made by subtracting a small sphere. Figure 10 shows a countersink bore, modeled by intersecting a cone and a conical sweep. Three light sources, cast shadows, and mirroring reflection were used. Figure 11 shows a glass, modeled by rotational sweeping.

Fig. 11. Glass.

7. DISCUSSION AND FURTHER WORK

A method is presented for calculating the intersection of a ray with a object defined by sweeping a cubic spline contour. At present we cannot determine whether the method of using splines is more efficient than one using strip trees [7]. It does require fewer evaluations for a contour, but the calculations involved per evaluation are more complex.

We think the method described in this paper has advantages over conversion into patches or polygons. For instance, an object defined by rotational sweep requests a very large number of polygons if smooth shading is required. By using patches, fewer items are needed, but a single patch requires more complex processing than a section of a contour.

We have shown that by representing a three-dimensional ray by an equation in two variables, efficient rectangle tests can be developed that exploit the halfspace nature of these equations.

Further work falls naturally into parts: improvement of the algorithms presented and extensions to the range of objects that can be described.

The main area in which improvements can be expected is in the solution of the two-dimensional intersection problem for rotational sweeps. A variety of ways to solve the resulting equation can be implemented. We used the method of Laguerre, which, in general, yielded satisfactory results in terms of the number of iterations required and of stability. The synthetic division step, however, as we could have expected, turned out to be very sensitive to round-off error. The ability of the Laguerre method to find all real and complex roots is not relevant here, since we are only interested in real roots in the interval [0, 1]. The method used by Hanrahan [5] can search for real roots in a limited interval, and it would appear, therefore, to be a better choice. Another possibility is not to substitute the polynomials $u(s)$ and $v(s)$ in eq. (9), but to use the subdivision method of Catmull [3] for each piece of the contour, along with an efficient rectangle test. Future research is needed to determine which technique, in general, is the most efficient.

Other sweep-defined objects that can be handled by the methods described in this paper can arise from generalizations of conic sweeping. Instead of both the x- and y-coordinates being scaled, each coordinate can be separately scaled by separate general functions of z. However, to find meaningful applications for the use of these kind of shapes will be harder then the development of the necessary algorithms.

General sweeping, that is, sweeping a contour along an arbitrary trajectory in three-dimensional space, would be an obviously useful extension. The line–object intersection problem has been solved for objects defined by sweeping a sphere as a partial advance in this direction [11]. The method involves the solution of a 10-degree polynomial when the trajectory is a cubic spline.

ACKNOWLEDGMENTS

I would like to thank D. J. McConalogue, W. F. Bronsvoort, and F. W. Jansen for their support, comments, and suggestions during the development of the algorithms and the preparation of this paper.

REFERENCES

1. APPEL, A. Some techniques for shading machine renderings of solids. In *Proceedings of the Spring Joint Computer Conference* (1968). Thompson Books, Washington, D.C., 37–45.
2. BLINN, J. F. A generalization of algebraic surface drawing. *ACM Trans Graph 1*, 3 (July 1982), 235–236.
3. CATMULL, E. E. A subdivision algorithm for computer display of curved surfaces. Ph.D. thesis, Computer Science Dept., Univ. of Utah, Salt Lake City, 1974.
4. GOLDSTEIN, R. A., AND NAGEL, R. 3D visual simulation. *Simulation 16*, 1 (Jan. 1971), 25–31.
5. HANRAHAN, P. Ray tracing algebraic surfaces. *Comput. Graph. 17*, 3 (Jul. 1983), 83–90.
6. KAJIYA, J. T. Ray tracing parametric patches. *Comput. Graph. 16*, 3 (Jul. 1982), 245–254.
7. KAJIYA, J. T. New techniques for ray tracing procedurally defined objects. *Comput. Graph. 17*, 3 (1983), 91–102.
8. RALSTON, A., AND RABINOWITZ, P. *A first course in numerical analysis*, 2nd ed., McGraw-Hill, New York, 1978, pp. 380–383.
9. ROTH, S. D. Ray casting for modeling solids. *Comput. Graph. Image Process. 18*, (Feb. 1982), 109–144.
10. SEYBOLD, H. Construction of functions for the representation of surfaces. In *Surfaces in*

Computer Aided Geometric Design (Copenhagen, Sept. 12–14), R. E. Barnhill and W. Boelm, Eds. North–Holland, 1983, pp. 179–185.

11. VAN WIJK, J. J. Ray tracing objects defined by sweeping a sphere. In *Proceedings of the Eurographics' 84 Conference*, K. Bø and H. A. Tucker, Eds. North–Holland, Amsterdam, 1984, pp. 73–82.
12. WHITTED, T. An improved illumination model for shaded display. *Commun ACM 23*, 6 (June, 1980), 343–349.

Received November 1983; accepted November 1984