Techniques for Implicit Modeling

Jules Bloomenthal
Xerox PARC
Palo Alto, California

Abstract: Traditionally, computer graphics has favored the parametric over the implicit surface because the parametric is easier to render. Implicit representations, however, have natural advantages in surface design, particularly in their ability to express concisely constraints that a surface must obey; this paper discusses this advantage, emphasizing the use of distance as a constraint. Unfortunately, the implicit representation of surfaces does pose problems for certain computer graphics applications, particularly rendering and texturing. This paper introduces a number of new techniques to overcome these difficulties.

This paper originally appeared as Xerox PARC Technical Report P89-00106.

CR Categories and Subject Descriptors: I.3.5 **[Computer Graphics]:** Computational Geometry and Object Modeling - curve, surface, solid, and object representations;

Additional Keywords and Phrases: Implicit Surface, Parametric Surface, Octree.

Introduction

Two methods of surface representation are common to computer graphics. The first, the parametric, defines a surface as a set of three-dimensional points **p**, such that:

$$\mathbf{p} = (x(u, v), y(u, v), z(u, v)).$$

The second, the implicit, defines a surface as a set of three-dimensional points **p**, such that:

$$f(\mathbf{p}) = 0.$$

A primary distinction between these two methods is the ease with which **p** is generated. With the parametric representation an arbitrary number of points can be generated directly by sweeping u and v through their domains. This direct generation of points facilitates conventional image rendering of the surface. Implicit surfaces do not enjoy such direct generation and, consequently, computer graphicists have given less attention to the implicit form [Foley, 1982].

An early exception is Ricci, who, in 1973, introduced a constructive geometry for the purpose of defining complex shapes derived from operations (such as union, intersection, and blend) upon primitives [Ricci, 1973]; he defined the surface as the boundary between the half-spaces $f(\mathbf{p}) < 1$ and $f(\mathbf{p}) > 1$, in other words, those points $\mathbf{p}$ satisfying $f(\mathbf{p}) - 1 = 0$.

The constructive solid geometry that evolved has emphasized the set operations. The nature of the primitives received relatively little attention until recent discussions by Sederberg on the direct modeling of solids using implicit functions [Sederberg, 1985, Sederberg, 1987]. Sederberg restricts his primitives to algebraic polynomials; Ricci limited his examples to piecewise quadrics. Others restricted their definitions to exponentials [Barr, 1981, Blinn, 1982]. Sophisticated operations, such as blending, have generally been restricted to quadratics [Middleditch, 1985, Hoffman, 1985].

Another distinction between the two surface representations, noted by Ricci, is the ease of surface definition. Surfaces that must obey a constraint, specified in terms of distance, are easier to define implicitly. For example, compare these two representations for a sphere, centered at $\mathbf{c}$, with radius r:

parametric: $\mathbf{p} = \mathbf{c} + (r\text{Sin}\theta\text{Cos}\varphi, r\text{Sin}\theta\text{Sin}\varphi, r\text{Cos}\theta)$, $\theta \in (0, \pi)$, $\varphi \in (0, 2\pi)$.

implicit: $f(\mathbf{p}) = |\mathbf{p} - \mathbf{c}| - r$.

Parametric surfaces remain more convenient for certain geometric computations; for example, curvature is more readily computed from the parametric representation.

Although simple constraints may be expressed analytically, Ricci observed that the defining function need not be analytic, but could be *procedural*. That is, a designer is free to specify any procedure that, given a point in space, computes a value. We refer to such design as *procedural implicit modeling*. In the next section we examine some of its advantages and some of the techniques involved in its use.

Implicit modeling permits a designer certain freedoms in constraining a surface, but it presents problems for the computer graphicist. We will review techniques for the conversion of implicit surfaces to the more conventional polygonal model. We will also offer new solutions to problems associated with shaded rendering, line drawing, and texture parameterization.

Distance Constrained Implicit Models

Designing with implicitly defined surfaces offers various advantages; many geometric operations, for example, are simplified. These include the standard set operations (union, intersection, etc.) of constructive solid geometry (CSG), functional composition with other implicit functions, and inside/outside tests.

The advantage with which we are primarily concerned is that of surface constraint. When an implicit surface is constrained according to distance, it can define several classes of surfaces more conveniently than its parametric counterpart. In this section we discuss some of these surfaces.

Skeletal Surfaces

The simplest surface constraint is in terms of distance of the surface to a set of points, curves, or other surfaces. We call this set a *skeleton*. The simplest skeleton, of course, is a single point **c**; if the distance constraint is a fixed radius r, the resulting surface is a sphere:

$$f(\mathbf{p}) = |\mathbf{p}\cdot\mathbf{c}| - r.$$

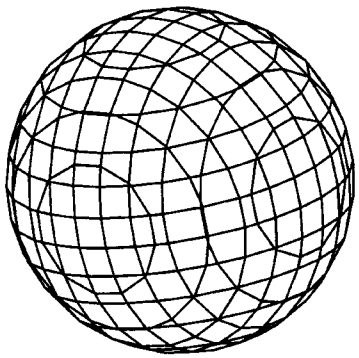

Figure 1: An implicitly defined sphere

If, instead of a center **c**, we use a segment $\overline{\mathbf{ab}}$, we obtain a cylinder:

$$f(\mathbf{p}) = |\mathbf{p} - \text{Closest}(\overline{\mathbf{ab}}, \mathbf{p})\text{-r}.$$

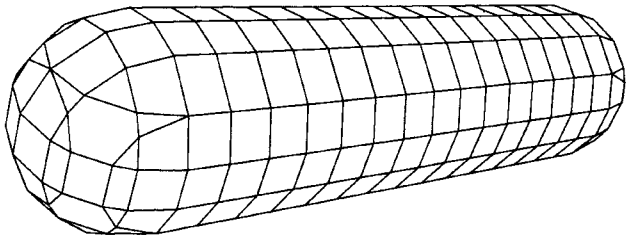

Figure 2: An implicitly defined cylinder

Closest($\overline{\mathbf{ab}}$, **p**) is the distance to the closest point on the line sgement, and is computed by:

$$\text{Closest} = \begin{cases} \mathbf{a} & \alpha \leq 0 \\ \mathbf{a} + \alpha\mathbf{d} & 0 < \alpha < 1 \\ \mathbf{b} & \alpha \geq 1 \end{cases}$$

where $\mathbf{d} = \mathbf{b} - \mathbf{a}$
$\mathbf{u} = \mathbf{p} - \mathbf{a}$
$\alpha = \mathbf{d}\cdot\mathbf{u}/(\mathbf{d}\cdot\mathbf{d})$.

The hemispherical ends are a consequence of the implicit function. Should we wish a truncated, right cylinder, *Closest* need only return any point greater than r distance from $\mathbf{p}$, when $\alpha \notin [0, 1]$.

To extend this to include the generalized cylinder [Agin, 1972], we define *Closest* to return the distance from $\mathbf{p}$ to an arbitrary space curve, S. Here, we restrict S to the familiar cubic parametric spline, defined by the vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$. A point $\mathbf{q}$ on the curve is $\mathbf{a}t^3+\mathbf{b}t^2+\mathbf{c}t+\mathbf{d}$, where t is the spline parameter, $t \in [0, 1]$. We find the closest point to $\mathbf{p}$ by solving $\partial|\mathbf{p}-\mathbf{q}|/\partial t = 0$. After some algebra,

$$\partial|\mathbf{p}-\mathbf{q}|/\partial t = 3\mathbf{a}\cdot\mathbf{a}t^5+5\mathbf{a}\cdot\mathbf{b}t^4+2(2\mathbf{a}\cdot\mathbf{c}+\mathbf{b}\cdot\mathbf{b})t^3+3(\mathbf{a}\cdot\mathbf{e}+\mathbf{b}\cdot\mathbf{c})t^2+(2\mathbf{b}\cdot\mathbf{e}+\mathbf{c}\cdot\mathbf{c})t+\mathbf{c}\cdot\mathbf{e},$$

where $\mathbf{e} = \mathbf{d}-\mathbf{q}$. Multiple roots must each be tested for the closest point. As shown in Figure 3, r may be a function of t.

This new method for computing generalized cylinders is simpler algorithmically than parametric methods [Shani, 1984]. An alternate method accelerates the distance computation by representing S as a series of points [Wyvill, 1986]; to obtain a smooth surface, the number of points must be sufficient with respect to curve length, curvature, and cylinder radius. As discussed below, an analytic, parametric representation for S is desirable for a number of applications, including surface texture and the construction of complex skeletons.

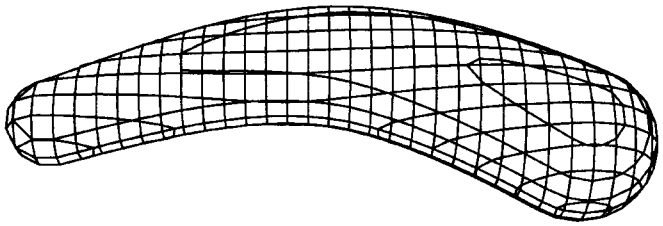

Figure 3: An implicitly defined generalized cylinder

Offset Surfaces

A surface constrained to be a fixed distance normal to another surface is an *offset surface* [Faux, 1979]. We consider here surfaces offset from polygons: these are defined conveniently as an implicit function of distance to the polygons:

$$f(\mathbf{p}) = \text{Min (Closest } (\mathbf{p}, \text{polygon}_i)$$

where *Closest* is either the distance from $\mathbf{p}$ to the plane of polygon_i, if $\mathbf{p}$ projects to the inside of the polygon, or the distance to the nearest edge or vertex of polygon_i. (The inside test is performed most conveniently in two-dimensions after dropping one of the coordinates from $\mathbf{p}$ and the polygon, provided the polygon is not perpendicular to the resulting plane).

12-5

A characteristic of offset surfaces is the rounded, or *chamfered*, results along convex edges or corners, as demonstrated in Figures 2-4. Thus we observe natural tendency for distance-constrained implicit surfaces to *smooth*.

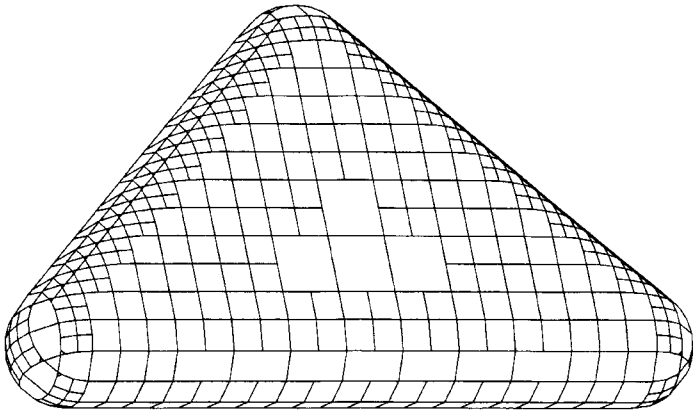

Figure 4: Offset surface of a triangle (adaptively sampled)

Metamorphosis

An implicit function can be an interpolation of two other functions. In the figure below, we interpolate the sphere and torus functions, evaluating them at various interpolation parameters between 0 and 1.

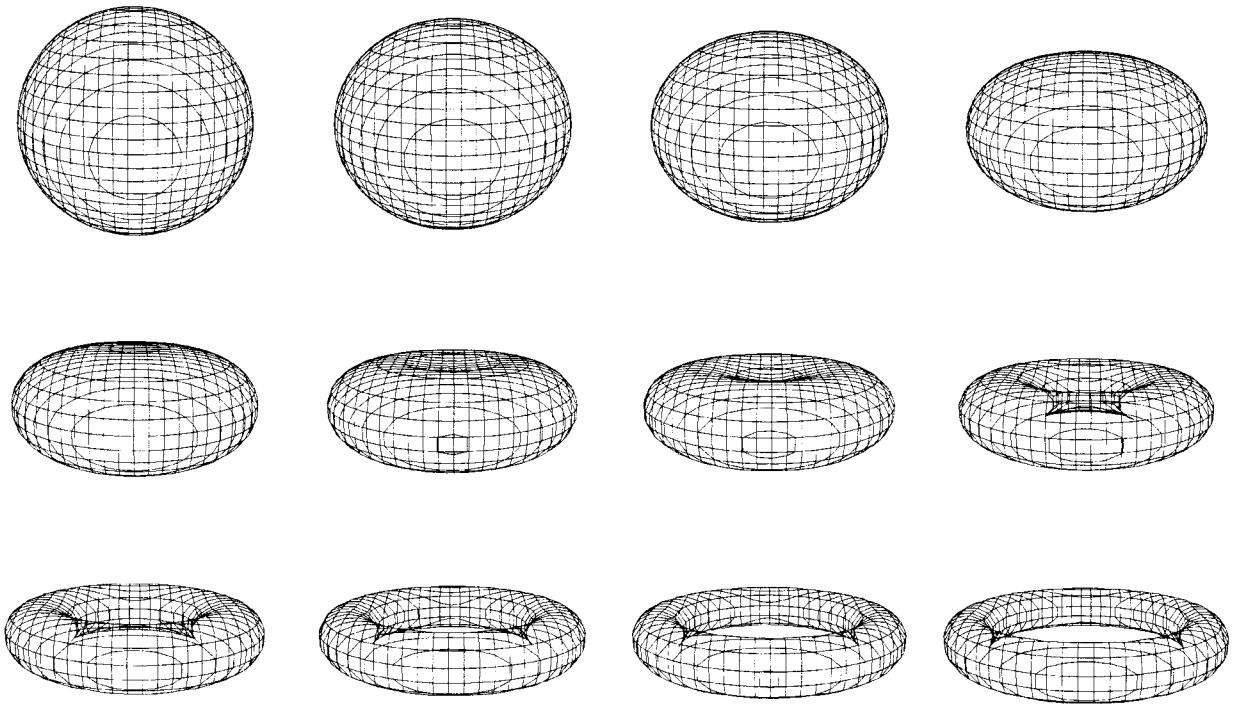

Figure 5: Metamorphosis of a sphere to a torus

Blended Surfaces

A blended surface, unlike an interpolation or metamorphosis, retains the individuality of its component primitives except where they intersect. We discuss blends to illustrate the convenience and flexibility of implicit functions and to demonstrate that blending can be based on skeletons. This section discusses some distance constraints useful in creating blends.

The implicit definition of the *branching* generalized cylinder is nicely concise. Consider a generalized cylinder, *Parent*, that branches into *Children*:

$$f(\mathbf{p}) = \mathrm{Max}(f_{Parent}(\mathbf{p}), \Sigma f_{Children}(\mathbf{p})),$$

where f_{Parent} and $f_{Children}$ are modified functions for the generalized cylinder:

$$f_{GeneralizedCylinder} = r/(|\mathbf{p} - \mathrm{Closest}(\mathbf{p}, S)| - r).$$

As $|\mathbf{p} - \mathrm{Closest}|$ increases, f decreases smoothly; this accounts for the smooth blending of the children, shown in Figure 6. That is, the blending of branches is accomplished by superposition of their functions. In order to maintain surface continuity, the radius of *Parent* is scaled such that $f_{Parent} = \Sigma f_{Children}$ at the branch point. The simplicity of this definition contrasts with parametric methods, in which a pentagonal parametric patch must be integrated into the model [Charrot, 1984]. A large number of branches can greatly complicate the topology of the parametric patch network; the implicit method, however, readily supports an arbitrary number of branches.

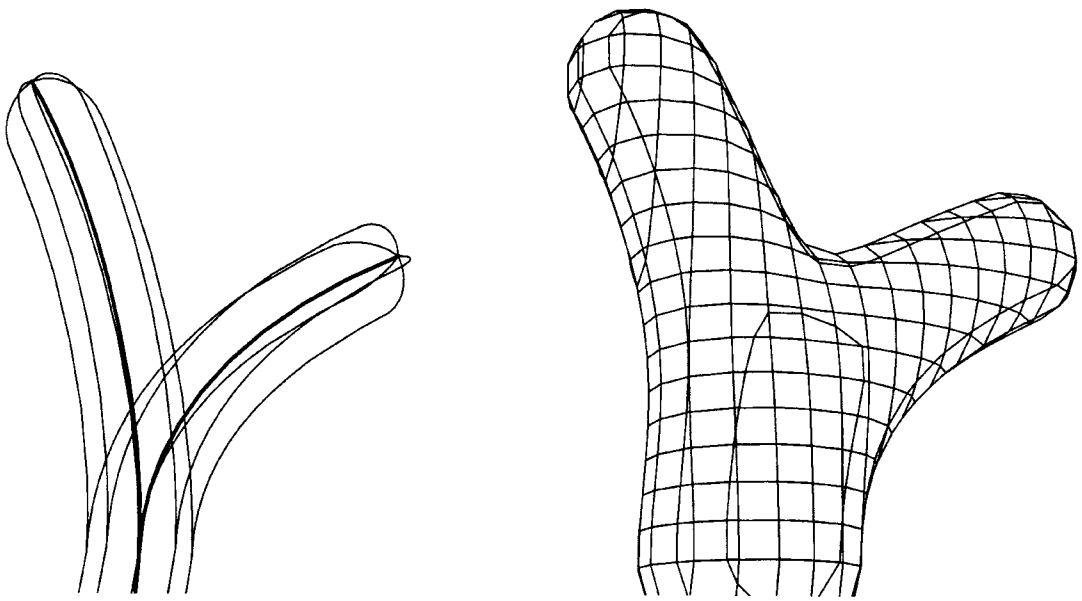

Figure 6: A branching generalized cylinder

A Complex Example

A consequence of the implicit definition for a branching cylinder, above, is that the radius of the parent is greater than any individual child radius. For a blend of two branches that maintains the original parent and child radii, we develop a method using an *envelope curve*. An envelope curve is a curve that is tangential to each member of a family of curves [Faux, 1979]. A well known example is derived from lines connecting evenly spaced points along two axes; the envelope curve for such a family of lines is a hyperbola, as shown in Figure 7, left.

To utilize the envelope curve, we construct a line segment between the closest points of the branches and compute the distance d between the segment and **p** (Fig. 7, middle). With several positions of **p**, an envelope curve becomes apparent (Fig. 7, right). The actual value of the implicit function is $(d-r)/r$.

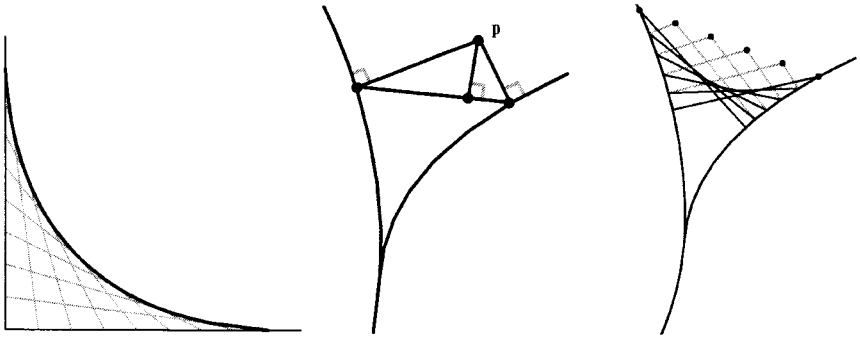

Figure 7: Distance measurements

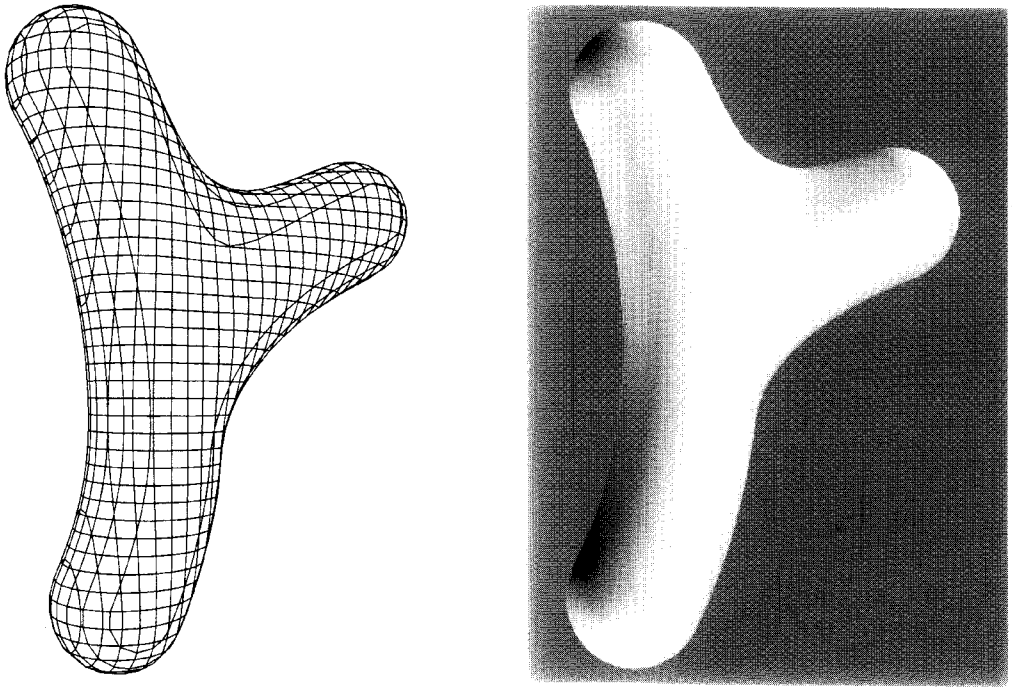

Figure 8: A branching generalized cylinder with constant radii

This method is considerably more complex than the original one given for branching generalized cylinders. This underscores an important characteristic of implicit modeling: the designer is free to specify an implicit function without regard for any subsequent conversion processes, such as polygonization or imaging. A properly implemented implicit surface polygonizer is unaffected by the inner workings of the implicit function. The function may be any arbitrary process; in addition to conventional mathematical functions, the designer may employ conditionals, tables, and so on.

Implicit Surfaces Derived from Parametric Surfaces

Let us consider the use of a complex parametric surface to define an implicit surface. We choose the shape of a flower petal, desiring a webbed surface to curve between two supporting veins. We can conveniently represent the veins by generalized cylinders. With some simple vector operations, we can loft cubic splines between the two veins. These are Bezier curves, defined by the points $\mathbf{p}_0$, **aa**, **bb**, and $\mathbf{p}_1$ (see Figure 9). $\mathbf{p}_0$ and $\mathbf{p}_1$ are along the two veins and **aa** and **bb** divide the segment $\overline{\mathbf{ab}}$ into thirds. $\mathbf{a} = \mathbf{p}_0 + \mathbf{v}$ and $\mathbf{b} = \mathbf{p}_1 + \mathbf{v}$, where $\mathbf{v} = \mathbf{v}_0 \times \mathbf{v}_1$. A lofted curve, Loft($t$), is defined in terms of a parameter t, where $t = 0$ at the base of a vein, and $t = 1$ at the tip.

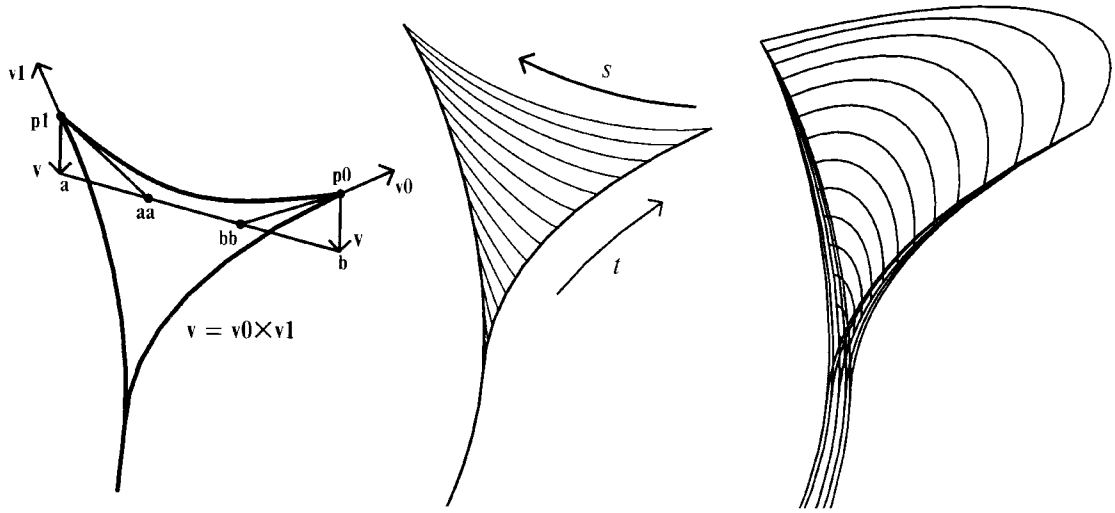

Figure 9: Method of lofting curves to create a parametric skeleton

To find the closest distance to the lofted surface from a point in space we seek the t for which the distance to Loft(t) is a minimum. It isn't possible to represent this minimum in a low order analytic form. Rather, we divide the computations into two steps: computing *Loft* and then computing the closest distance to *Loft*. We find t with a minimization technique [Press, 1986]. Although techniques exist that limit the search domain of t, the minimization is, nonetheless, very compute intensive. We intend to develop methods to accelerate finding the closest point on complex, parametrically defined skeletons. The surface shown in Figure 10 would be very difficult to specify parametrically.

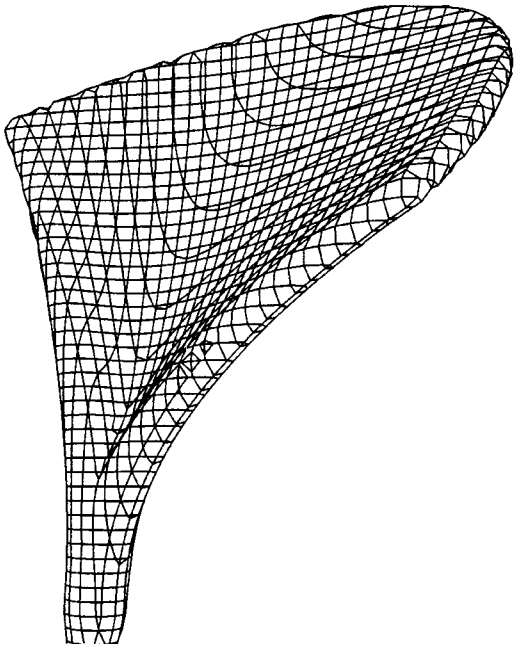

Figure 10: An implicitly defined "petal"

Hybrid Techniques

It is simple to bound an implicit object with a plane, but it is a quite different matter to create an object that consists of a planar piece; such a piece has no volume and can not be represented implicitly. Even if a thin piece had some thickness, the polygonization sampling size, although practical, may result in gaps in the surface. Blends may best be modeled as a combination of solids, but webs (as in Figure 10) are, in many cases, best modeled as a surface. Consider a cross-section of the vein and web, shown in Figure 11, left; there is no topologically consistent way to define an inside/outside test for this combination of primitives, as shown in Figure 11, middle.

Consider, then, this proposal whereby parametric surfaces may be combined with implicit surfaces. Referring to Figure 11, right, we construct two spatial partitionings, one that represents the web and one that represents the vein. In cell 1 the web is defined but the vein is not, since its thickness is less than the size of the cell. Cell 3 contains the web but no edges of the vein, although the entire cell is contained within the vein volume. Cell 2 contains both web and vein surfaces. We propose that the two partitionings be combined into one, observing these rules:

> where the parametric surface (the web) exists and the implicit surface and volume (the vein) do not, as in cell 1, the cell is polygonized according to the parametric surface,

where the implicit surface exists and the parametric does not, as in cell 4, the cell is polygonized according to the implicit surface,

where the parametric surface and the implicit volume exists, but the implicit surface does not, as in cell 3, the cell is not polygonized,

where the parametric surface and the implicit surface exist, as in cell 2, the cell is polygonized by connecting the parametric-cell intersection with the implicit-cell intersections, as shown by the dashed lines.

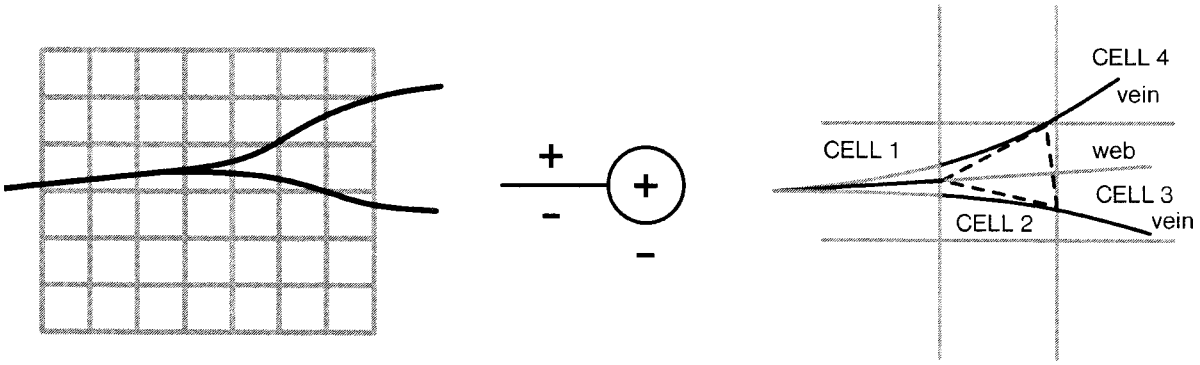

Figure 11: Combining parametric and implicit surfaces

Difficulties

We have discussed a number of advantages in modeling with implicit surfaces. They include ease of blending and offseting. Often constraints based upon distance are more simply and intuitively expressed implicitly. Complex implicit surfaces do not result in correspondingly complex polygonization software, whereas the polygonization of parametric surfaces often requires a custom implementation for each surface type. Implicit functions are, however, less tractable in a number of ways; we discuss some of these in this section.

Surface Generation

We define surface generation to be the generation of points on a surface. This is conveniently performed for a parametric surface by sweeping each parameter through its domain as the parametric equation is evaluated. It is possible to convert a low order implicit function to its parametric equivalent, and then generate the surface: but this becomes difficult or impossible for higher order functions [Sederberg, 1986].

For the arbitrary, procedural implicit function this is not possible. Instead, the function must be sampled, in search of those points that satisfy the implicit equation. This is a numeric technique, sensitive to those problems associated with sampling; the sampling errors may be minimized with adaptive techniques [Bloomenthal, 1988].

The underlying principal of numerical implicit surface generation is that the surface must intersect the segment $\overline{\mathbf{ab}}$ if $f(\mathbf{a}) < 0$ and $f(\mathbf{b}) > 0$; thus space must be sampled for pairs of points with oppositely signed function values. The simplest approach is to sample space on a regular, three-dimensional grid; the resulting *voxel array* may be searched for segments spanning $f = 0$, and points on the surface interpolated from these segments [Artzy, 1980]. Generating a voxel array is computationally intensive; further, the array may suffer from aliasing due to under-sampling, resulting in loss of detail. Generally, voxel imaging techniques are employed where data is relatively easy to obtain, such as in medical laboratories; it is less appropriate for modeling surfaces, which are sparse in three dimensions.

A more efficient method of surface generation is to track the implicit function, adaptively partitioning space into cells. A number of n-dimensional approaches [Dobkin, 1986, Allgower, 1987] utilize the simplex as a cell; others utilize the cube [Wyvill, 1986]. In all cases, surface points are found by interpolation along cell edges that span the surface. The partitioned cells may be organized by hashing [Wyvill, 1986], or by a more structured representation such as an octree [Bloomenthal, 1988]. Figure 12 illustrates this method; an octree, shown in blue, is created either by recursive subdivision of a root cube, thus converging to the implicit surface, or by propagation from a smaller, seed cube. Once the octree is created, its terminal nodes, shown in green, are tested for intersection with the implicit surface. The resulting points, shown in yellow, are connected together to form a polygonal approximation to the implicit surface, shown in red.

Adaptive sampling of the function is possible; a restricted octree [Von Herzen, 1987] is used in order to retain the polygonal structure and avoid discontinuities along faces of differing sizes [Bloomenthal, 1988].

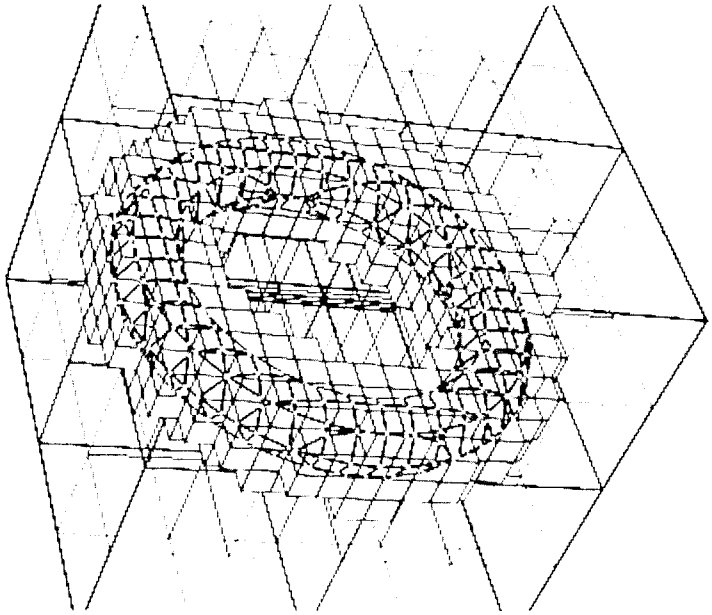

Figure 12: Implicit surface organized by an octree

If an implicit surface is polygonized, it becomes *navigable*. (compare with the *boundary* or *evaluated* CSG form [Mortensen 1985]) that is, one is able to move along the surface, from one surface point to another. Arrangements of objects, such as one resting upon another, as well as other applications, become possible, generally, only when a surface is navigable. To illustrate, Figure 13 presents an implicit surface that has been covered with blades of grass.

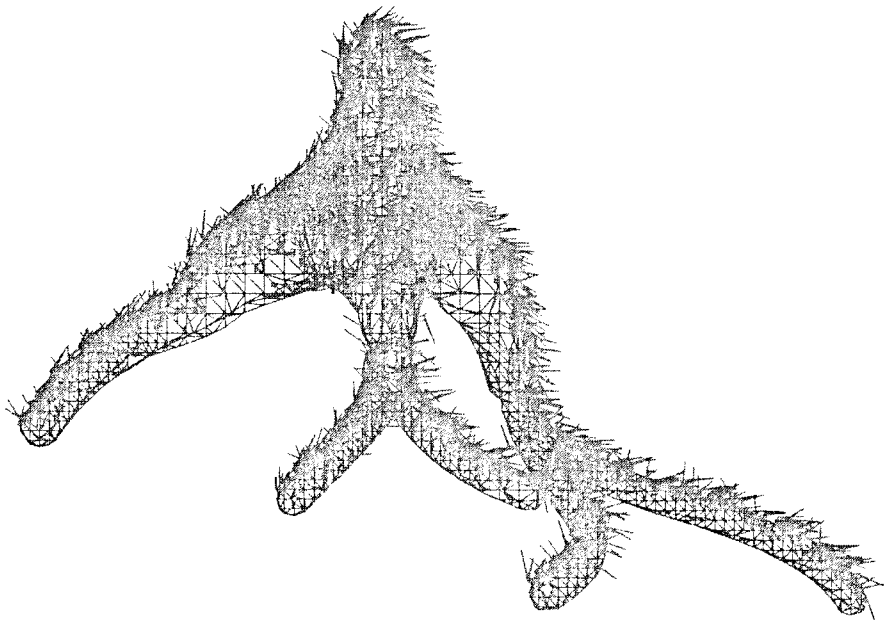

Figure 13: A navigable surface

Shaded Images

The difficulties in rendering an implicit surface are related to those of generating the surface; that is, there are no incremental methods comparable to the incremental scan-line techniques commonly employed for polygonal surfaces. Algebraic surfaces may be ray-traced [Hanrahan, 1983, Kalra, 1989] or rendered using incremental, scan-line techniques [Sederberg, 1989], but no incremental techniques exist for abitrary implicit surfaces. One method is to convert the implicit surface to a polygonal representation, to be rendered by a conventional polygon renderer. This is the method used for most of the images in this paper.

Alternatively, the implicit surface may be rendered by ray tracing. There are practical methods for ray tracing algebraic surfaces [Hanrahan, 1983], but if the implicit function is procedural, with roots that can not be predicted, then each ray must carefully sample space throughout its (infinite!) length. The task becomes more manageable if finite bounds can be placed on the function.

To reduce the amount of computation when ray tracing, Glassner suggested partitioning space with an octree, reducing the distance across which a ray must search for a root [Glassner, 1984]. (Partitioning an implicit surface for the purpose of ray-tracing is comparable to the subdivision used for parametric patches when rendering with a z-buffer [Catmull, 1974]). Significant computation is still required to find a root along even a short line segment, but this is compensated by the improved simulation of optical phenomena as compared with other hidden surface algorithms. In the figure below, we ray trace an implicit surface to simulate shadows and transparency.

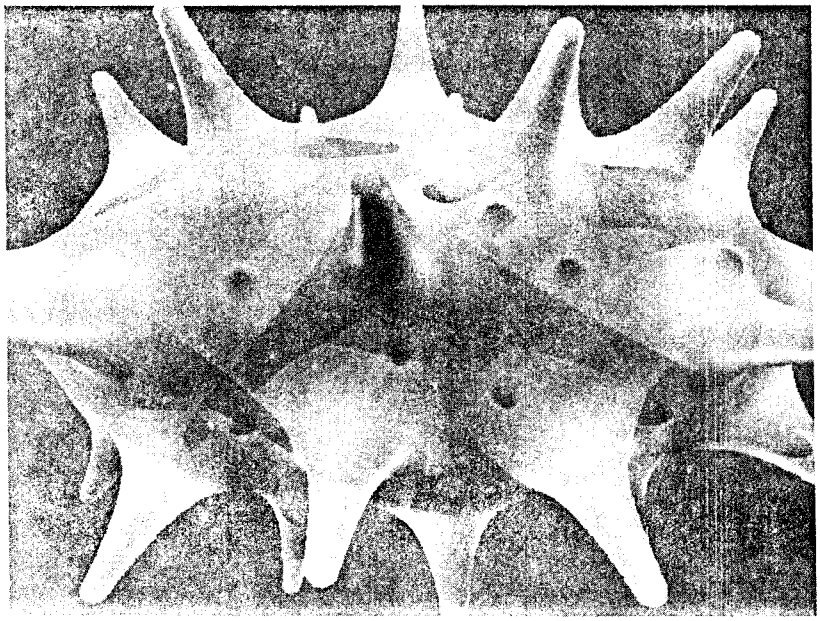

Figure 14: A ray-traced implicit surface

Ray tracing does not produce a navigable surface; thus, Figure 13 could not be created through conventional ray tracing techniques because those techniques would not permit the placement of grass along the object's surface.

Line Drawings

In addition to visible surface rendering, there is one known technique for visible line rendering of implicit surfaces, suggested by Ricci [Ricci, 1973]. The implicit surface is intersected with a series of planes, perpendicular to the line of sight and receding from the viewer. For each plane, the function contours are drawn, excepting those parts obscured by previous planes. Ricci's examples were limited to algebraic functions; we have extended the method to arbitrary implicit surfaces.

For a procedural implicit function, the plane must be examined everywhere for roots, unless the implicit function has been spatially partitioned. If the partitioning is an octree, it is simple to intersect the plane with the terminal cubes of the octree (all terminal cubes are presumed to intersect the surface); the intersection proceeds hierarchically: child cubes need be tested for intersection only if their parent is intersected. The actual points of intersection can be computed linearly, once the corners of the cube have been evaluated with respect to the plane equation (see Figure 15). The intersection points may be ordered to form a *cube-plane polygon.*

Given a cube-plane polygon, the contours can be computed. First, points satisfying the implicit function are found along the polygon edges. A new point satisfying the implicit equation is found approximately midway along the segment; then the segment is divided into two, and the process repeated until the segment lengths are deemed small enough for presentation. Should $\overline{P_a P_b}$ fail to span a root, a is increased. The result, using this new technique, is a drawing of the surface's *contour lines.*

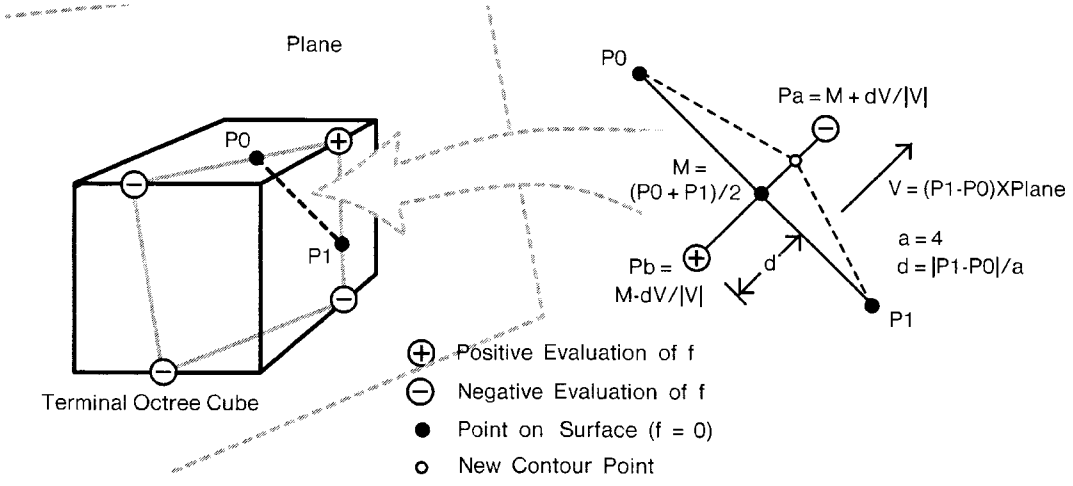

Figure 15: Planar contour generation

Forrest has discussed contour lines with respect to the display of shape information for surface patches [Forrest, 1979]. He stated that contour lines have an "exact, unambiguous, geometric interpretation . . . it is arguable that of all the line drawings possible, a set of surface sections is the most useful in determining shape." In the figure below we compare contour (section) lines with a visible line rendering of polygons.

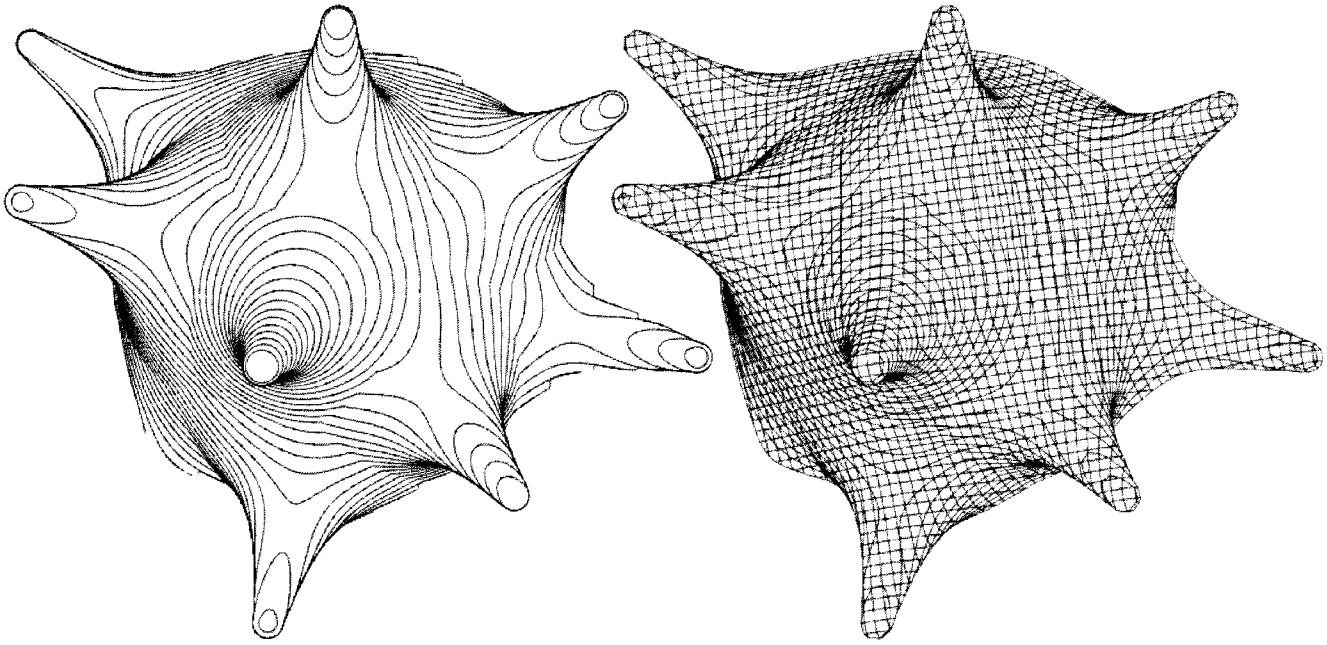

Figure 16: Visible line rendering of an implicit surface (left) and a polygonal equivalent (right)

Surface Texture

In this section, we are concerned with the assignment of parametric ("texture") coordinates to points on the implicit surface. Parameterization is important in generating images with realistic surface texture without explicitly modeling surface microstructure; bump and texture mapping both rely upon surface parameterization. For parametric surfaces, the obvious parameterization is trivial to construct, yet it may not be the desired one. Creating a "natural" parameterization that follows the skeleton of the surface may be desirable, and equally difficult to construct for implicit and parametric surfaces.

One approach is to parameterize a polygonal model derived from the implicit model, assigning parametric coordinates to the polygonal vertices in such a way that the parametric distance between two vertices is proportional to their geometric distance. For non-developable surfaces some variation in the proportion becomes necessary; for example, degeneracies occur at the poles of a sphere or along the seam of a tapered cylinder. We do not pursue this method, but refer the reader to related work [Gagalowicz, 1985].

Consider assigning parametric ("texture") coordinates to the branching cylinder. So as to avoid discontinuities in the mapping, we begin with the initial u-coordinate assignments shown in Figure 17. Note that the two branches and the parent agree with each other in their u-values for 0.0 and 0.5. The remaining values are assigned through a simple transformation of a surface vertex into the xy plane defined by the local coordinate system of the nearest point on the spline. It is this use of the cylinder's axis that associates the parameterization with the surface structure. (The v-value of the parametric coordinates is determined from the distance along the spline, but we discuss only the calculation of the u-value, which is more interesting).

The difficulty arises with the gap between lines of equal u that occurs in a roughly triangular region in the center of the branching cylinder. If we think of this as an "island" and the lines of constant u as "topographic" contours, then, intuitively, we wish to create a "hill" on this island. We can do so by increasing the u-value with decreasing proximity to the center of the island. We define this center as the point at which the u-value is 0.25 and the distances from the surface vertex to the two splines are equal. The image below, middle, is without compensation for the island; the image to the right has this compensation. The important criterion is the evenness of line spacing (the ripples are a rendering artifact).

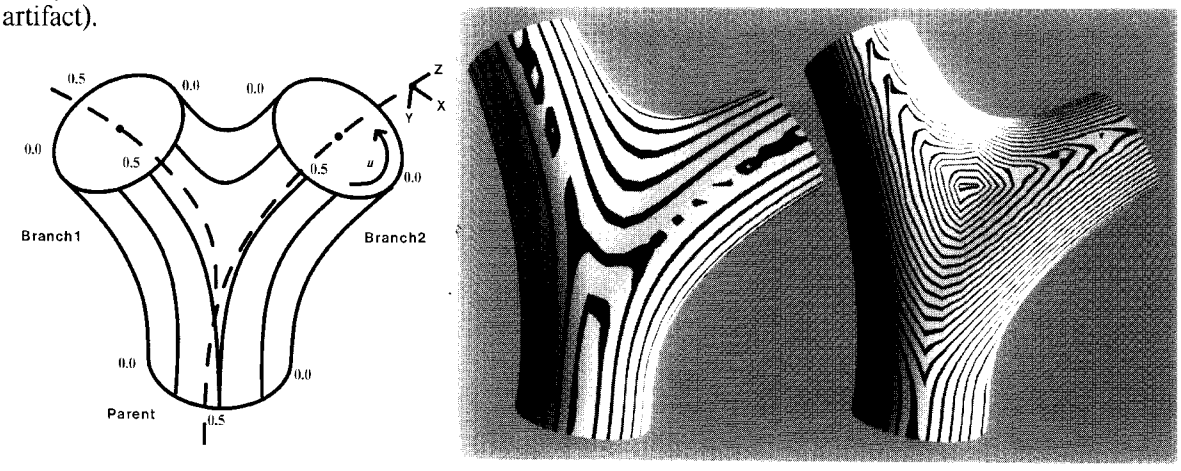

Figure 17: Texture assignment and adjustment to an implicit surface

This is an area for additional investigation, but the example demonstrates that a mapping can adhere to the defining structure of the surface.

Conclusions

Implicit surfaces offer a number of advantages for the designer, but pose a number of difficulties for the computer graphicist who needs to manipulate the surface in traditional ways: in this paper we have categorized the problems and offered some solutions. Although experience with implicit surface design is limited, it is clear that complex, constrainable shapes can be expressed, often with comparative ease.

Acknowledgements

Thanks to Robin Forrest for his thoughtful comments, and to Paul Heckbert who patiently offered valuable and extensive suggestions to improve this paper. To Nancy Gill, my thanks for General Critique and Major Support. And thanks to Lynn Klein, for Tolkien's A Leaf by Niggle.

References

Allgower, E.L. and Gnutzmann, S. An Algorithm for Piecewise Linear Approximation of Implicitly Defined Two-Dimensional Surfaces. *SIAM Journal of Numerical Analysis, 24*, 2 (April 1987), 452-469.

Agin, G.J. Representation and Description of Curved Objects. *Memo AIM-173*, Stanford Artificial Intelligence Report (October 1972).

Artzy, E., Frieder, G., and Herman, G.T. The Theory, Design, Implementation and Evaluation of a Three-Dimensional Surface Detection Algorithm. Proceedings of SIGGRAPH'80 (Seattle, Washington, July 14–18, 1980). In *Computer Graphics 14*, 3 (July 1980), 2-9.

Barr, A. Superquadrics and Angle-Preserving Transformations. *IEEE Computer Graphics and Applications, 1*, 1 (January 1981), 11-23.

Blinn, J.F. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics, 1*, 3 (July 1982), 235-256.

Bloomenthal, J., Polygonization of Implicit Surfaces. *Computer Aided Geometric Design, 5*, 4 (November 1988).

Catmull E. A Subdivision Algorithm for Computer Display of Curved Surfaces. *Technical Report UTEC-CSc-74-113*, Computer Science Dept., University of Utah, December 1974.

Charrot, P. and Gregory J.A. A Pentagonal Surface Patch for Computer Aided Geometric Design. *Computer Aided Geometric Design, 1*, 1 (July 1984), 87-94.

Dobkin, D.P., Thurston W.P. and Wilks A.R. Robust Contour Tracing. *Technical Report CS-TR-054-86*, Computer Science Dept., Princeton University, September 1986.

Faux, I.D. and Pratt, M.J. *Computational Geometry for Design and Manufacture.* John Wiley and Sons, New York, NY, 1979.

Foley, J.D. and Van Dam, A. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, Reading, Mass, 1982.

Forrest, R. On the Rendering of Surfaces. Proceedings of SIGGRAPH'79 (Chicago, Illinois, August 8-10, 1979). In *Computer Graphics 13*, 2 (August 1979), 253-257.

Gagalowicz A. and De Ma S. Model Driven Synthesis of Natural Textures for 3-D Scenes. *Eurographics '85, C.E. Vandoni, ed.*, Elsevier Science Publishers, 1985.

Glassner, A.S. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics and Applications, 4*, 10 (October 1984), 15-22.

Hanrahan, P. Ray Tracing Algebraic Surfaces. Proceedings of SIGGRAPH'83 (Detroit, Michigan, July 25-29, 1983). In *Computer Graphics 17*, 3 (July 1983), 83-90.

Hoffman, C. and Hopcroft, J. The Potential Method for Blending Surfaces and Corners. Technical Report TR 85-674 Computer Science Dept., Cornell University, 1985.

Mortensen, M.E. *Geometric Modeling*. Wiley and Sons, New York, 1985.

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. *Numerical Recipes*. Cambridge University Press, Cambridge, England, 1986.

Ricci, A. A Constructive Geometry for Computer Graphics. *The Computer Journal 16*, 2 (May 1973), 157-160.

Sederberg, T.W. and Goldman, R.N. Algebraic Geometry for Computer-Aided Geometric Design. *IEEE Computer Graphics and Applications, 6* (June 1986), 52-59.

Sederberg, T.W. Algebraic Piecewise Algebraic Surface Patches. *Computer Aided Geometric Design, 2* (1985), 53-59.

Sederberg, T.W. Algebraic Geometry for Surface and Solid Modeling. *Geometric Modeling: Algorithms and Trends*, G. Farin, ed., SIAM Press, 1987.

Shani, U. and Ballard D.H. Splines as Embeddings for Generalized Cylinders. *Computer Vision, Graphics, and Image Processing, 27*, 2 (August 1984), 129-156.

Woodwark, J.R. Blends in Geometric Modelling. *Proceedings of the 2nd IMA Conference on the Mathematics of Surfaces*, (Cardiff, September 8-10, 1986).

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