Chapter 3: Implicit Surfaces

For my own part I am pleased enough with surfaces . . .
what else is there? What else do we need?
(Edward Abbey)

3.1 Introduction

An implicit surface is a set of points $\boldsymbol{p}$ such that $f(\boldsymbol{p}) = 0$, where f is a trivariate function (*i.e.*, $\boldsymbol{p} \in \Re^3$). The surface is also known as the *zero set* of f and may be written $f^{-1}(0)$ or $Z(f)$. According to the implicit surface theorem, if zero is a regular value of f, then the zero set is a two-dimensional manifold. An *iso-surface* is a similar set of points for which $f(\boldsymbol{p}) = c$, where c is the *iso-contour value* of the surface. The function f is sometimes called the *implicit function*, although we prefer *implicit surface function*. A review of the salient properties of implicit surfaces may be found in [Hoffmann 1989].

In many cases, f partitions space into an inside and an outside. By convention, f is usually written such that $f(\boldsymbol{p}) < 0$ describes a volume of points enclosed by the surface, $f(\boldsymbol{p}) = 0$. This ability to enclose volumes and the ability to represent blends of volumes endow implicit surfaces with inherent advantages in geometric surface design. This is particularly so for skeletal design, as the relation between skeleton and surface is, generally, volumetric. Inherent in this relation is the use of a distance metric between the point $\boldsymbol{p}$ and the skeleton. In chapter 5, we examine these metrics according to their blend properties and ease of implementation.

In chapter 2, we suggested it is easier for the designer of natural forms to work with skeletal geometry than with the usually more complex geometry of the corresponding surface. The volumetric relation described by $f(\boldsymbol{p}) < 0$ underlies the correspondence between skeleton and surface. Thus, the combination of skeletal and implicit techniques is particularly appropriate for many natural forms. The smoothness sometimes associated with natural forms may be obtained as a blend of component volumes, which we call implicit *primitives*. The skillful combination of primitives is an important task for designers who wish to define implicitly an interesting or useful shape. Primitives may also be combined by set operations and functional composition (such as deformations), but blends are the most important combination concerning the design of smooth surfaces.

Traditionally, computer graphics has favored the parametric surface over the implicit because the parametric is easier to render and is more convenient for certain geometric operations, such as the computation of curvature or the control of position and tangency. Specifically, ‘‘parametric surfaces are generally easier [than implicit surfaces] to draw, tessellate, subdivide, and bound, or to perform any operation on that requires a knowledge of ‘where’ on the surface’’ [Rockwood 1989]. Parametric and implicit surface representations are also distinguished by the compactness of their mathematical expression [Ricci 1973]. This seems particularly true for definitions that involve distance. For example, given a sphere centered at $\boldsymbol{c}$, with radius r, the parametric definition is:

(3.1) $(\boldsymbol{p}_x, \boldsymbol{p}_y, \boldsymbol{p}_z) = (\boldsymbol{c}_x + r\cos\theta\cos\phi,\ \boldsymbol{c}_y + r\sin\theta,\ \boldsymbol{c}_z + r\cos\theta\sin\phi),$
$\theta \in (0, \pi),\ \phi \in (0, 2\pi).$

The implicit definition is considerably more compact:

(3.2) $(\boldsymbol{p}_x - \boldsymbol{c}_x)^2 + (\boldsymbol{p}_y - \boldsymbol{c}_y)^2 + (\boldsymbol{p}_z - \boldsymbol{c}_z)^2 - r^2 = 0.$

Implicit surface definitions are very general; they can represent discrete pointsets, algebraic surfaces, and procedurally defined implicit surfaces. A discrete pointset can be represented by a function that returns 0 for $\boldsymbol{p}$ a member of the set, 1 otherwise. Generally, this is not useful because the function is discontinuous; pointsets can, however, be adapted for use in surface fitting [Hoppe *et al*. 1992].

Algebraic surfaces are commonly found in computer graphics, and include quadrics [Foley *et al*. 1990] and superquadrics [Barr 1981]. There has been a recent renewal of interest in general algebraic surfaces [Sederberg 1985], [Sederberg 1987], [Bajaj 1992]. They may be ray-traced [Blinn 1982], or, in the case of quadric surfaces, they are suitable for incremental scan-line techniques [Mathematical Applications Group 1968].

Although simple distance constraints can be expressed analytically, the defining function need not be analytic, but, as observed in [Ricci 1973], may be *procedural*. That is, a designer is free to specify any arbitrary process that, given a point in space, computes a real value. The procedure may employ conventional mathematical functions, conditionals, tables, and so on. A procedurally evaluated implicit surface function is not the same as a 'procedure model' [Newell 1975]. Both involve procedures, but the former is utilized to evaluate a point in space and the latter is utilized to construct a parametric surface. An example procedural implicit surface, discussed in section 5.6.14.1, performs several geometric operations to yield a value for f that would be difficult to express analytically.

In the following sections we consider several aspects of an implicit surface, including its relation to solid modeling, its application to skeletons, its visualization and polygonization, and its refinement by added surface detail.

3.2 Solid Modeling

In 1973, a 'constructive geometry' was introduced for the purpose of defining complex shapes derived from operations (such as union, intersection, and blend) upon primitives [Ricci 1973]. The surface was defined as the boundary between the half-spaces $f(\boldsymbol{p}) < 1$ and $f(\boldsymbol{p}) > 1$; the former was considered the 'inside,' or solid portion, of an object. From this initial approach to *solid modeling* evolved *constructive solid geometry*, or *CSG*. With CSG, an object is evaluated 'bottom-up' according to a binary tree. The leaf nodes are usually restricted to low degree polynomial primitives, such as spheres, cylinders, ellipsoids, and tori. The internal nodes represent Boolean set operations.

The difference between solid modeling and implicit modeling is somewhat subtle. The surface of a solid model must enclose a finite volume. Consequently, the surface is everywhere equivalent to a two-dimensional disk and is, therefore, a two-dimensional *manifold* [Mäntylä 1988]. Implicit surfaces can enclose finite volumes as well; for example, $f(x, y, z) = x^2+y^2+z^2-1$ represents the unit sphere. But implicit surfaces can also represent unbounded surfaces; for example, $f(x, y, z) = z$ represents the *xy*-plane.

Solid modeling is not limited to constructive models but may include, among others, *decomposition* models and *boundary* models [*ibid*.]. CSG is, however, the dominant form of solid modeling. The literature of constructive solid geometry emphasizes the robust representation of all intermediate results within the tree-structured evaluation. Usually this intermediate representation is the *boundary representation*, or *BRep*. It is a versatile representation from which several geometric properties, such as volume and center of gravity, are readily computed.[1] It is generally accepted in solid modeling that boundary representations must be closed under all Boolean operations.

The requirement to maintain intermediate boundary representations places an extraordinary demand on the process of CSG evaluation. These concerns are expressed in [*ibid*.]:

> Unfortunately, Boolean set operations algorithms for boundary representations are in general plagued by two kinds of problems: First, to be effective, a set operations algorithm must be able to treat all possible kinds of geometric intersections . . . [which] easily leads to a very [complex] case analysis. Second, the very case analysis must be based on various tests for overlap, coplanarity, and intersection which are difficult to implement

robustly in the presence of numerical errors.

This explains the preoccupation in CSG literature with the robustness of edge-edge and edge-surface intersections. In comparison, concrete representations for implicit surfaces are formed without intermediate evaluations, greatly reducing the affects of numerical instability. Additionally, implicit surfaces need not be defined according to a binary tree or any other graph.[2]

Early development of geometric modeling, which embraces both surface and solid modeling, was motivated by engineering applications in the automotive, aerospace, aviation, and shipping industries and by training applications such as real-time, interactive flight simulators. This development involved an interplay of visualization and geometric modeling techniques that inextricably linked computer graphics and geometric modeling. For example, as graphics systems became faster and more flexible, designers were encouraged to develop ever more sophisticated models, many of which required new techniques in geometric modeling. Indeed, much of the development in surface and solid modeling is reported in the literature of computer graphics.

3.3 Skeletal Design

Having discussed the value of skeletally defined implicit surfaces, we now consider specific methods for their evaluation and definition. As described in section 2.4, the skeleton is a collection of elements, each of which generates a volume. Within an implicit context, we call such a volume a *skeletal primitive*, which we denote by $P_i(\boldsymbol{p})$, for skeletal element i. Thus, P is a function from $\Re^3$ (or $\Re^2$ for illustrative purposes) to $\Re$, and, usually, is C^1 continuous. The implicit surface function may be a blend of these primitives, *i.e.*, $f(\boldsymbol{p}) = g(\boldsymbol{p}, P_1, P_2, \ldots P_n) = 0$, and the implicit surface is the covering, or manifold, of the skeleton. Ideally, we wish our implementation of implicit surface algorithms to be indifferent to skeletal complexity. This leads to two principal methods to evaluate the implicit surface:

(3.3) $f_{skeleton} = fr_{root} - c$, where $fr_{limb} = max\,(f_{limb}, \Sigma fr_{children})$, or

(3.4) $f_{skeleton} = \Sigma f_{limb} - c$

In both (3.3) and (3.4), f_{limb} refers to the implicit primitive defining the volume surrounding the particular skeletal element. In (3.3), *fr* is a recursive function equal to the implicit primitive of a limb or the sum of *fr* applied to each of the child limbs, whichever is greater. Contrary to solid modeling convention, we assume f_{limb} increases with decreasing distance to the limb; thus, *max*, rather than *min*, is appropriate. $f_{skeleton}(\boldsymbol{p})$ is the recursive function applied to the root limb of the skeleton; in (3.4), $f_{skeleton}(\boldsymbol{p})$ is simply the summation of all primitives. The recursive function yields a more smooth transition in limb radii at branch points.

The computational load for (3.3) and (3.4) can be reduced by providing an axis-aligned bounding box around each skeletal element. For ***p*** outside the bounding box of $limb_i$, the influence of $limb_i$ is presumed non-existent; that is, $f_{limb_i}(\boldsymbol{p}) = 0$. It is simple to test for ***p*** within an axis-aligned box.

The interpolation of two implicit surfaces can be accomplished in several ways. The most accessible method is to interpolate the individual functions that define the surfaces. For example, in the figure below, we interpolate a torus, $(x^2+y^2+z^2+r_{major}{}^2-r_{minor}{}^2)^2-4r_{major}{}^2(x^2+z^2)$, and a sphere, $x^2+y^2+z^2-r_{major}{}^2$.

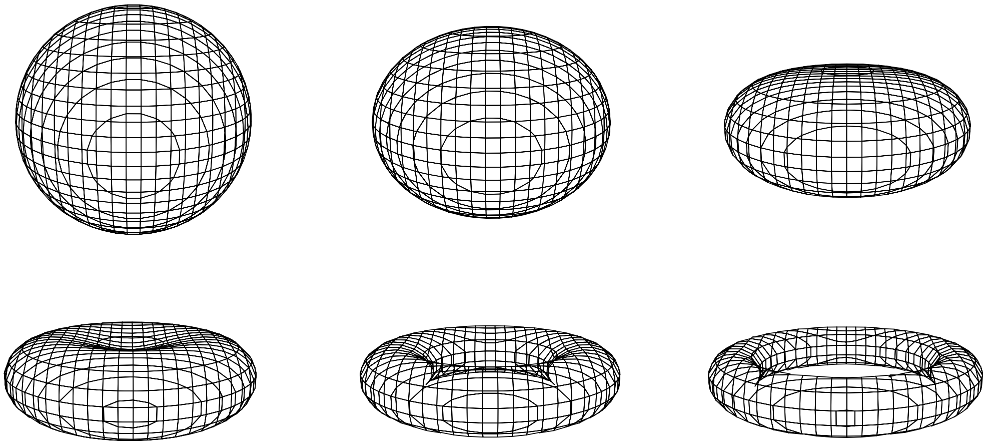


Figure 3.1 Interpolation of Sphere and Torus Functions

Interpolation of two functions, however, is not appropriate for skeletal forms because rigid body transformations can be lost. For example, in the following illustration, generated with a two-dimensional implicit contour follower, we employ three different interpolations. As in Figure 2.5, only rigid body rotation of the skeleton produces a realistic interpolation.

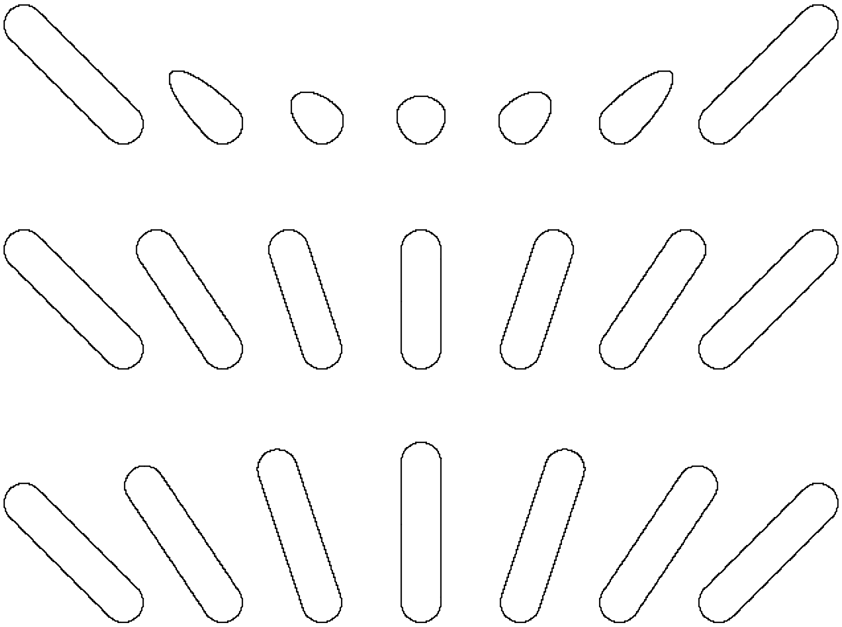


Figure 3.2 Implicit Interpolations

top: interpolation of implicit contour functions[3]

middle: interpolation of segment endpoints

bottom: interpolation of segment angle

3.4 Visualization

Because an implicit formulation does not produce surface points by substitution, root-finding must be employed to visualize an implicit surface.[4] This can be performed by ray-tracing, polygon scan conversion, or contour tracing. We briefly consider each of these methods.

Implicit surfaces may be rendered directly by ray tracing, assuming a ray-surface intersection procedure is provided for a given surface. Often the intersection calculation can be accelerated by culling those pieces of the surface bounded by axis-aligned boxes not intersected by the ray. This process is known as *spatial subdivision* [Glassner 1984], [Samet 1990] of the implicit volume, and was applied to a procedural implicit surface [Bloomenthal 1989] to produce the image below. Other methods to accelerate the ray-tracing of implicit surfaces include symbolic algebra for surfaces defined by polynomials [Hanrahan 1983], the Lipschitz condition for surfaces with bounded gradient derivatives [Kalra and Barr 1989], and sphere-tracing for surfaces with bounded derivatives [Hart 1993].

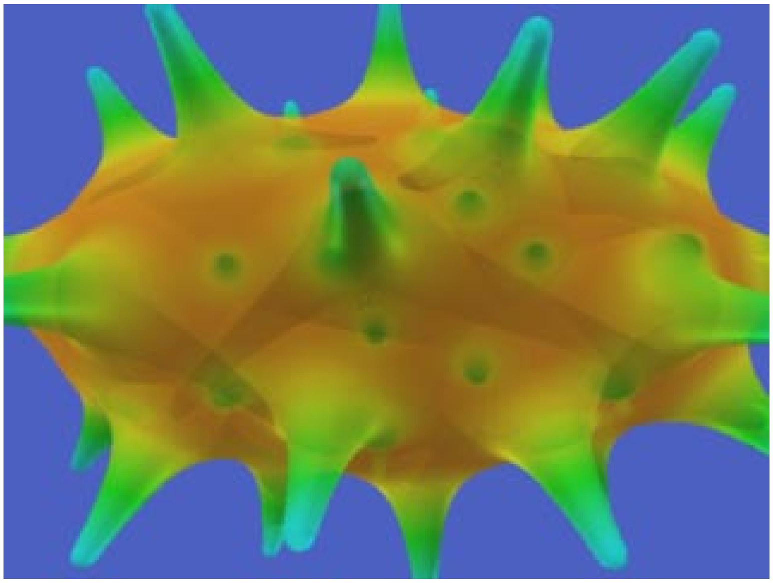


Figure 3.3 Ray-Traced Image

In addition to shaded images, it is possible to create contour-line (or *section*-line) drawings of implicit surfaces by intersecting the surface with a series of planes, each perpendicular to the line of sight and receding from the viewpoint [Ricci 1973]. For each plane, the zero-set contour is drawn, excepting those parts obscured by previously drawn contours. In [Bloomenthal 1989] the implicit surface was spatially partitioned by an octree [Meagher 1982] to produce the following image. It is simple to compute the intersection of octree and plane; each intersected terminal node of the octree produces a section of the contour. Contour line drawings are particularly useful for engineering applications [Forrest 1979].

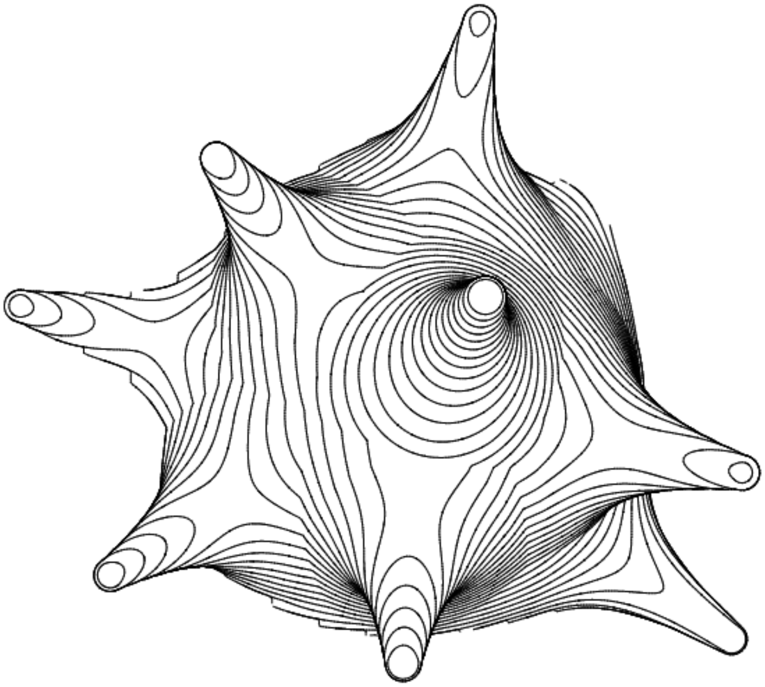


Figure 3.4 Contour Line Drawing

Although ray-tracing can produce excellent images, it cannot produce a *chartable* surface,[5] that is, a concrete surface representation. Such representations permit non-imaging operations such as movement from one surface point to another, positioning objects upon a surface, navigating along the surface to define texture, and associating predefined texture with specific portions of the surface.

The most universal, concrete surface representation is, arguably, a set of polygons. Conversion of a functionally specified implicit surface to a polygonal approximation can require considerable computation, but is required only once per surface and allows rendering of the surface by conventional polygon scan conversion, which is considerably more efficient than ray-tracing an arbitrary implicit surface.

It is possible to convert a low order algebraic function to a parametric equivalent, and then generate polygons by sweeping the surface parameters through their domains. This becomes difficult or impossible for higher order functions [Sederberg 1986]. Instead, we employ numerical techniques to *sample* the implicit surface function and *polygonize* the surface. These techniques generally work well for smooth implicit surfaces. Conversion to polygons allows the computer graphicist to manipulate the surface in traditional ways and to render the surface without repeated solution of the implicit surface function. Because of these advantages, we devote the next section to polygonization.

3.5 Polygonization

A flexible design environment should free the designer from difficulties associated with the computer representation of a surface, whether these difficulties be storage in a concrete form, fitting together surface pieces, or visualization. We rely on polygonization to shield the designer from the productive process wherein the abstract design expression (*i.e.*, *f*) is converted to a concrete representation (*i.e.*, polygons). In other words, the polygonizer treats *f* as a 'black box' and in so doing isolates the designer's definition of skeleton, volumetric primitives, and their blends from the topological and geometrical complexity of the resulting surface. Thus, complex implicit surfaces do not require correspondingly complex polygonization software, whereas the polygonization of parametric surfaces often requires an *ad hoc* implementation for each surface type. In this section we present a brief description of implicit surface polygonization.

An evaluation of polygonizers according to their implementation complexity, number of triangles produced, topological consistency, and topological correctness is given in [Ning and Bloomenthal 1993]. Other criteria, such as the number of function evaluations required, the adaptive distribution of polygons, and the visual appearance of the resulting surface, are not evaluated.

Polygonization consists of two principal steps: a) the partitioning of space into cells, and b) the processing of each cell to produce polygons. In the first step, space is partitioned into non-overlapping, space-filling cells that collectively contain the surface. The implicit surface function is evaluated at the corner.[6] negative values are considered to lie on one side of the surface, positive values on the other. In the second step, a vertex of the implicit surface is presumed to lie on any cell edge that connects oppositely signed corners. Within each cell the surface vertices are connected to form one or more polygons. Throughout this and the following chapter, we use the term ‘‘corner’’ to mean a vertex of the polygonizing cell and the term ‘‘vertex’’ to mean a vertex of the implicit surface.

3.5.1 Partitioning

An important distinction concerning polygonizers is the function *f*, which can be discrete or continuous. Many polygonizers are intended for use with discrete data, such as obtained from scanning devices. In other words, the value of *f* is available only at cell corners. For continuous functions, such as those that define the geometric models described in this dissertation, the value of *f* may be obtained at arbitrary locations. There are two practical consequences of this distinction. The first relates to the accuracy of the polygonal surface. For discrete functions, the location of a surface vertex must be approximated; for continuous functions, the location of a surface vertex can be determined with arbitrary precision.

A second practical consequence concerns the number of function evaluations. In the discrete case, *f* is evaluated throughout a fixed volume, and at a fixed interval. In this process, known as *enumeration*, the number of function evaluations is fixed and, usually, the time per evaluation is a constant determined by the scanning hardware. In the continuous case, the volume enclosing a surface is not known *a priori* and each function evaluation usually requires considerably more computation than in the discrete case. Thus, obtaining all cells within a given volume is less practical for the continuous than for the discrete case. Instead, polygonizers based on continuous data usually begin with an initial cell enclosing a piece of the surface. Additional cells are propagated across cell faces that intersect the surface until the implicit surface is fully contained by the collection of cells. This process is known as *numerical continuation* [Allgower and Georg 1990] and was first applied to implicit surfaces by [Wyvill *et al*. 1986].

Continuation methods require O(n^2) function evaluations, where n is a measure of the size of the object (thus, n^2 corresponds to the object's surface area). Methods that employ enumeration require O(n^3) function evaluations. We know of only two published polygonization implementations. In [Watt and Watt 1993] a 'marching cubes' implementation is provided that employs enumeration, and in [Bloomenthal 1994] an implementation is provided that employs continuation, which will be described in this section.[7]

A benefit of enumeration is that it detects all pieces of a set of disjoint surfaces. This is not provided by continuation methods, which require a starting point for each piece. The continuation method used in our research automatically detects a starting point by random search; thus, only a single object is detected and polygonized. The designer may, however, explicitly provide a starting point, in which case random search is not needed and disjoint objects may be polygonized by repeated use of the polygonizer, each time with a different starting point. It may be possible to automate the detection of starting points for skeletally defined models by searching the neighborhood of each skeleton's root. In practice, providing a starting point (or starting neighborhood) does not appear to be a serious burden for the designer.

The cube is a useful partitioning cell because it is a regular polyhedron that fully packs space [Coxeter 1963]. Its symmetries provide a) a simple means to compute and index corner locations, which is useful in storing surface vertices, and b) a simple means to index the cell itself, which is useful for the prevention of cycles during cell propagation.

3.5.2 Root-Finding

In skeletal design, an object is defined by a continuous, real-valued function. Such functions allow the location of a surface vertex to be accurately computed, rather than approximated by linear interpolation as is commonly employed with discrete data. The affects of interpolation are shown below.

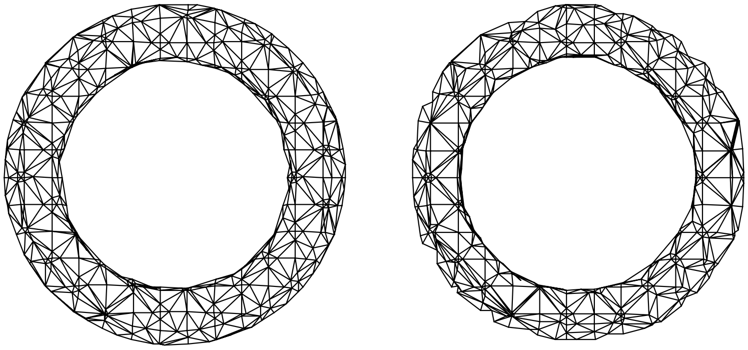


Figure 3.5 Surface Vertex Computation

left: accurate, right: interpolated

Binary subdivision is a reliable and simple method to compute accurate surface vertex locations. Given one point inside the surface and one outside, the method converges to a point on the surface by repeatedly subdividing the segment connecting oppositely signed function values. Ten iterations appears to provide acceptable accuracy. Subdivision can, at times, be more efficient than other convergence methods, such as *regula falsi* [Bloomenthal 1988].

As described in [Wyvill *et al.* 1986], the performance of function evaluation can be greatly improved by culling those skeletal elements that do not contribute to the function value at a given point (because the point is beyond the influence of the element). The skeletal primitives can be organized into a set of bounding boxes or a hierarchical structure such as an octree.

Performance is also improved by minimizing the number of function evaluations. For example, the location of a surface vertex is computed only once; it and its function value are cached and subsequently indexed according to the coordinates of its edge endpoints, using a hashing technique similar to that reported in [Wyvill *et al*. 1986]. Function values at cube corners are similarly cached. Cached cube corner values accelerate the computation of surface vertex locations for adjacent cubes; cached surface vertex values accelerate the computation of surface normals. The overhead in caching function values may exceed the cost of additional evaluations for simple functions. For complex functions, however, the elimination of redundant evaluation significantly improves performance.

3.5.3 Cell Polygonization

Cell polygonization is the approximation by one or more polygons of that part of the surface contained within a cell. This approximation consists of three steps. The first is the computation of *edge vertices* that approximate the intersection of the surface with the cell edges. The second step is the connection of these vertices to form lines across the faces of a cell; these approximate the intersection of the surface with the cell faces. The third step is the connection of these lines to form polygons, which approximate the surface itself.

Unfortunately, some polarity combinations of cube corners do not disambiguate between conflicting polygonal configurations within a cube. The ‘marching cubes’ polygonization method [Lorensen and Cline 1987] produces errant holes in the surface because it treats these ambiguities inconsistently [Düurst 1988]. The method employed by the skeletal design system treats cube ambiguities consistently, in one of two user-selectable ways. Either the cube is directly polygonized according to an algorithm given in [Bloomenthal 1988], or it is decomposed into tetrahedra that are then polygonized according to an algorithm given in [Koide *et al*. 1986]. Thus, either a cube or a tetrahedron serves as the polygonizing cell. The continuation, decomposition, and polygonization steps are illustrated below.

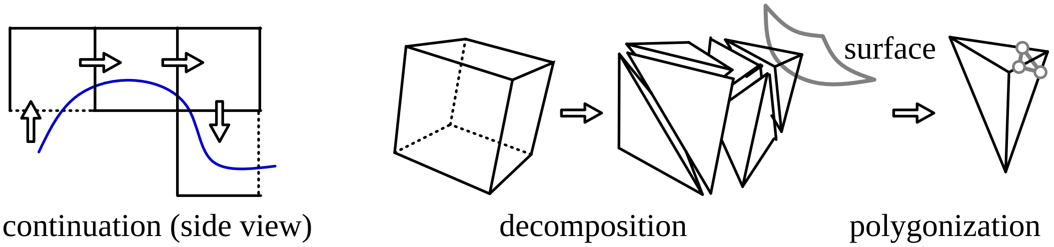


Figure 3.6 Overview of a Polygonizer

Each edge of the polygonizing cell that connects corners of differing polarity yields a surface vertex. When connected together, the surface vertices form a polygon. The ordering of these vertices is given by a table that contains one entry for each of the possible configurations of the cell corner polarities. For the cube, the table has 256 entries and may be generated according to methods described in [Bloomenthal 1988] and [Wyvill and Jevans 1993]. For the tetrahedron, which is the default polygonizing cell, the table has 16 entries and may be generated by inspection. For each tetrahedron, either no polygon, a triangle, or a quadrilateral (*i.e.*, two triangles) is produced, as shown below.

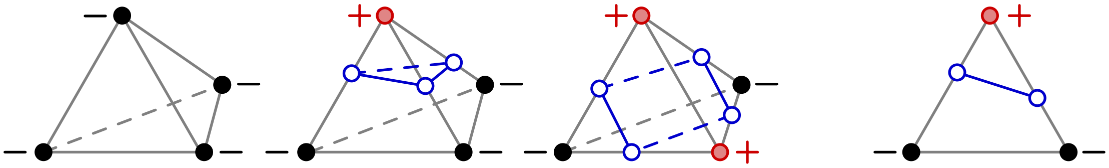


Figure 3.7 Tetrahedral Cell Polygonization

left: zero, three, or four edge vertices are produced within a tetrahedron

right: at most a single line crosses a tetrahedral face

Because the tetrahedral edges include the diagonals of the cube faces, the tetrahedral decomposition yields a greater number of surface vertices per surface area than does cubical polygonization, as compared below. Cubical polygonization requires less computation than tetrahedral polygonization, but requires a more complex implementation.

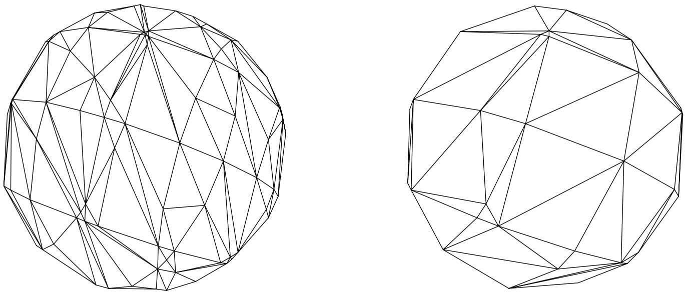
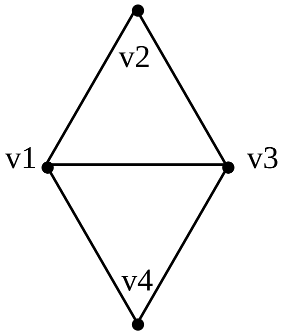


Figure 3.8 Two Methods of Cell Polygonization

left: with tetrahedral decomposition, right: without

In [Wyvill *et al*. 1986] polygons are produced in a 'polygon' format, wherein surface vertices shared across adjoining polygons are replicated. If vertices are stored by reference, not by value, a 'points/polygons' format results and coincident vertices are not replicated. Rather, the vertex reference (or 'id') can be associated with its containing edge. An edge that contains a surface vertex may be stored in a hash table, indexed according to the lattice indices of its two endpoints. This sparse storage, whose memory requirements are proportional to the object's surface area (whereas dense storage is proportional to the object's volume) is desirable because the domain extent of the function is not known *a priori*. Because tetrahedral cells utilize the corners of the cubical cell, the cube corner lattice indices work well for cubical or tetrahedral partitionings.

The points/polygons format provides a basis for conversion to a boundary representation and may be more convenient for some polygon renderers A comparison of the two formats is given below.



polygon format

triangle:
(x1, y1, z1)
(x3, y3, z3)
(x4, y4, z4)

triangle:
(x2, y2, z2)
(x4, y4, z4)
(x3, y3, z3)

points/polygons format

vertices:
(x1, y1, z1)
(x2, y2, z2)
(x3, y3, z3)
(x4, y4, z4)

triangle: (1, 2, 3)
triangle: (1, 3, 4)

Figure 3.9 Two Polygon Formats

Either during or after polygonization, a surface vertex may be assigned a normal and, optionally, a color or other property. The normal at $\boldsymbol{p}$ is the function gradient, approximated by a forward difference (less evaluations than a central difference):

(3.5) $\nabla \mathrm{f}(\boldsymbol{p}) \approx (f(\boldsymbol{p}_x)-f(\boldsymbol{p}), f(\boldsymbol{p}_y)-f(\boldsymbol{p}), f(\boldsymbol{p}_z)-f(\boldsymbol{p}))/\Delta$,
where $\boldsymbol{p}_x$, $\boldsymbol{p}_y$, and $\boldsymbol{p}_z$ represent $\boldsymbol{p}$ displaced by Δ along the respective, positive axes.

3.5.4 Surface Bounds, Complexity, and Resolution

Many implicitly defined objects, such as the tori shown below, are two-dimensional manifolds. They may be bounded or unbounded, but, everywhere, must be homomorphic to a two-dimensional disk. The upper torus is given by $t_1(x, y, z) = (x^2+y^2+z^2+R^2-r^2)^2-4R^2(x^2+y^2) = 0$, where R and r are the major and minor radii (in this example, $r = R/4$). To achieve a rotation and offset, the lower torus is defined by $t_2(x, y, z) = (x^2+(y+R)^2+z^2+R^2-r^2)^2-4R^2((y+R)^2+z^2) = 0$. The points that satisfy $f(\boldsymbol{p}) = t_1(\boldsymbol{p}) = t_2(\boldsymbol{p}) = 0$ define a ‘Voronoi’ surface [Chandru *et al.* 1990] This surface is not bounded and requires a limit to the continuation propagation in all six (left, right, above, below, near, and far) directions from the start point. For the surface below, the propagation limit was 7.

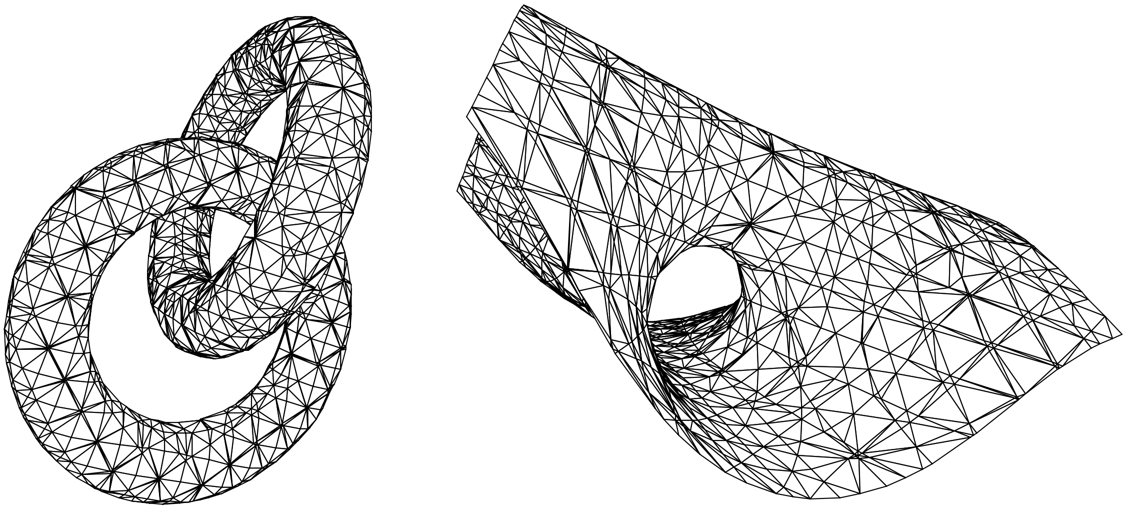


Figure 3.10 Torus R Us

left: two Tori, right: their Voronoi surface

There is no limit to the complexity of the implicit surface function. For example, an object resembling a piece in the game of ‘jacks’ may be given by:[8]

$$(1/(x^2/9+4y^2+4z^2)^4+1/(y^2/9+4x^2+4z^2)^4+1/(z^2/9+4y^2+4x^2)^4+$$
$$1/((4x/3-4)^2+16y^2/9+16z^2/9)^4+1/((4x/3+4)^2+16y^2/9+16z^2/9)^4+$$
$$1/((4y/3-4)^2+16x^2/9+16z^2/9)^4+1/((4y/3+4)^2+16x^2/9+16z^2/9)^4)^{-1/4}-1.$$

Indeed, ‘meta-objects’ have been specified with thousands of terms [Graves 1993].

Although its complexity is unrestricted, the implicit surface should be G^1 continuous. Otherwise, the surface that results from fixed size partitioning may be incomplete, have truncated edges, or contain small jutting pieces (such as shown in [Bloomenthal 1994]). A G^1 discontinuous surface requires adaptive function sampling to produce a concrete representation that is faithful, in terms of fine detail, to the design.

For polygonization, adaptive sampling is the variation of cell size according to several possible criteria, such as a) surface curvature within the cell [Bloomenthal 1988], b) interval analysis of the function at the cell corners [Snyder 1992], c) bounds on the gradient [Kalra and Barr 1989], or d) bounds on the derivative [Hart

1993].[9] An example of adaptive sampling according to approximate curvature is given in the following figure. This approach may be applied to all functions, but is not guaranteed to find all surface parts. The other three approaches find all parts but require precise analysis of the implicit surface function. Such analysis is possible for many functions [ibid.], but not for an arbitrary ‘black box.’ Indeed, there is always the possibility of a ‘pathological’ function that can never be polygonized in sufficient detail. For example, the Steiner surface, $f(x, y, z) = x^2y^2+x^2z^2+y^2z^2+xyz$, consists of an easily polygonized surface and the principal axes, which cannot be polygonized.

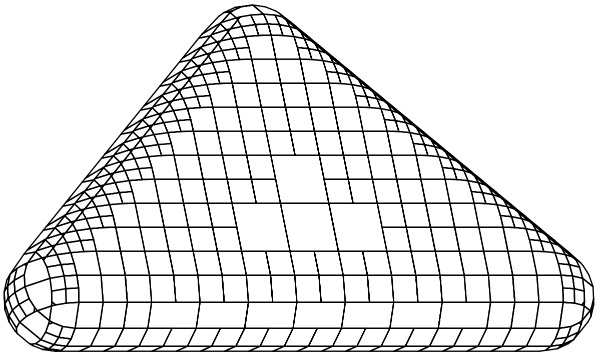


Figure 3.11 Adaptively Sampled Trianguloid

Adaptively sized cells complicate the cell processing step. For example, the hashing scheme described in section 3.5.3 would require substantial modification. In [Bloomenthal 1988] a *balanced octree* [Samet 1990], a three-dimensional corollary of the ‘restricted quadtree’ [Von Herzen and Barr 1987], is employed to maintain the polygonal structure without discontinuities along faces of differingly sized cells. The cube is a logical choice for adaptive sampling; it is the only regular polyhedron that can be recursively subdivided. [Moore 1992] discusses adaptive subdivision based upon tetrahedra.

In general, we have found a fixed cell size polygonizer provides good performance and works well for smooth, natural shapes. Fixed resolution polygonizers are simpler to implement, which explains their popularity over adaptive resolution methods. We do not employ automatic techniques to determine an appropriate cell size; this is set by the designer. In practice this has not been a significant burden. For fixed-size partitioning, the choice of cell size is important: too small a size produces an excessive number of polygons; too large a size obscures detail. The following figure demonstrates improved approximations as the cell size is reduced.

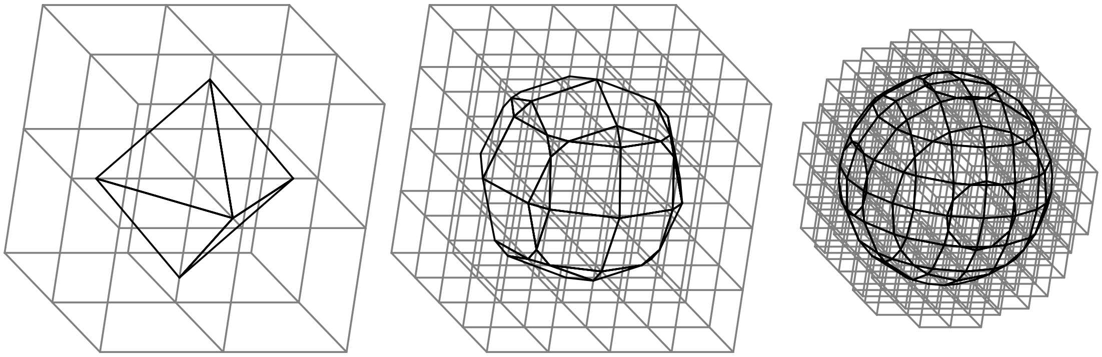


Figure 3.12 A Sphere Partitioned with Differently Sized Cells

Adaptive sampling can locate all subparts of an object, but cannot locate all objects within a scene. Some of these problems can be mitigated by skeletal techniques. For example, after each polygonization step, points along each skeletal primitive could be tested for inclusion within previously polygonized volumes.

3.6 Details, Details

In defining a smooth shape, we blend simple geometric primitives. Details are, as usual, a completely different matter. We wish to produce images with detail normally observable with the unaided eye, and suggest a finely detailed structure be represented as a compound shape in which minute detail refines a smooth, underlying surface.

Displacement techniques [Cook 1984] can be incorporated into the implicit definition, as shown below, to refine the surface geometry. In this case, the cell partitioning size can be proportional to the feature size of the added detail.

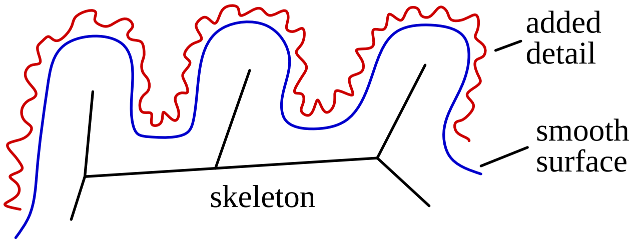


Figure 3.13 A Smooth Surface Refined by Detail

Surface refinement may be implemented as an implicit composition (or *deformation*) from $\Re^3$ to $\Re^3$. For example, consider the following two-dimensional function, $f(\boldsymbol{p}) = a(b(c(\boldsymbol{p})))$. The original function c is simply a vertical wash of gray (shown here with sine waves in the display's colormap). The function b causes two local, blended shifts downward, and the function c adds a vibration. This addition of detail upon an underlying, smooth form is similar to the addition of harmonics to a underlying audio sine wave [Chamberlin 1980].[10] Although this type of hierarchical detail could be recursive, in this example the character of the detail changes according to hierarchical depth.

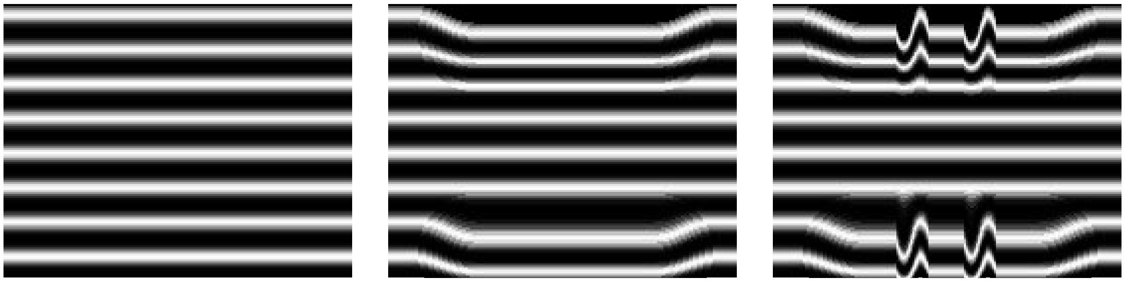


Figure 3.14 An Implicit Contour with Displacement

left: c, middle: b(c), right: a(b(c))

Patterns of complexity found on an object's surface often correspond to the object's skeleton. For example, surface creases occur at points of frequent stretching or compression and these points are predictable given skeletal articulation. Therefore, detail need not necessarily be given in the implicit representation. Instead, it can be added to the polygonized model directly, accessing the skeleton and ignoring the implicit definition. For example, a ‘bump-map’ can be associated with each surface polygon; the pattern of the bump-map can be determined from the relationship of the polygon to the skeleton.

Aside from general methods to assign texture [Gagalowicz 1985], [Turk 1992], we are aware only of *ad hoc* methods to assign texture coordinates given an implicit definition. Once such method is illustrated in figure 5.75. Although effective at moderate or long range, mappings (such as bump mapping or texture mapping) are usually unconvincing for natural objects viewed at close range. A convenient parameterization (*i.e.*, assignment of texture coordinates) free of singularities may not, however, exist for a given surface and, thus, it may be difficult to establish a realistic mapping.

It is important that details, such as veins on a hand, be located consistently on the underlying surface so that they remain appropriately attached to the surface should it undergo articulation or transformation. For example, if a skeletal joint is articulated in such a way as to suggest stretching of the surface, we expect the density of detail to decrease. Otherwise, the animated detail will not appear realistic. Rather than incorporate this detail into the implicit definition, it may be simpler to modify the polygonal surface produced by the implicit surface polygonizer. Polygonization produces a chartable surface. In other words, we are able to navigate on the surface and distribute detail, such as hair or veins, according to the relationship between surface position and skeleton. It is not immediately apparent how the distribution of detail can be accomplished within a visualization system such as ray-tracing, which does not produce a chartable surface from an implicit definition.

3.7 Interactivity

Although in this dissertation we emphasize the application of implicit modeling to particular natural forms, a number of general observations apply to the implicit design environment and, in particular, interactive techniques. A survey of techniques applicable to interactive implicit surface design may be found in [Bloomenthal and Wyvill 1990]. These techniques include:

- interactive manipulation of the skeleton
- 'diagramming' of geometric operations used by a procedural function
- interactive procedural specification 'by example'
- interactive definition of primitive functions for a skeletal element
- acceleration of ray-tracing or polygonization by
    - spatial partitioning,
    - bounding boxes,
    - frame coherence, or
    - user specified vertex accuracy
- adaptive subdivision
    - for surfaces with readily estimated local curvature
    - for local areas specified by the user
- piecewise linear approximation of curved skeletal elements
- alternate display methods
    - planar or volumetric function display, adjustable iso-value
    - physically based point 'scattering'
    - improved resolution during idle processor moments

Physically based sampling is a method whereby points on the surface are determined by heuristics, such as the function gradient [Figueirido and Gomes 1994]. Recent physically based methods provide improved design interactivity and rendering performance [Witkin and Heckbert 1994].

In our implementation, the polygonization process may be aborted at arbitrary time should the designer decide, for example, to revise the defining function or to modify polygonization parameters such as cell size.

3.8 Conclusions

The skeletal design models developed in this dissertation presume that the implicit surface functions are continuous, not discrete as with scanned volumetric data.[11] If we employed a discrete function, represented by a voxel array of function values, we might use an altogether different method to evaluate $f_{skeleton}$. Before sampling three-space for the surface, we could first proceed along the skeleton, dispersing values to accumulate within voxels. ‘Accumulation modeling’ has been used to impressive effect with *z*-buffers [Smith 1982], [Williams 1990] and voxel arrays [Greene 1989], [Greene 1991]. Although three-dimensional ‘value dispersal’ could be computationally expensive, the iterative accumulation within an enumerated volume appear to permit the simulation of developmental processes and the complex shapes that result [Greene 1989], [Greene 1991].

Polygonization is a method whereby a polygonal approximation to the implicit surface is created from the implicit surface function. This allows the surface to be rendered with conventional polygon renderers. Although ray-tracing can produce excellent images of implicit surfaces, we prefer the concrete representation afforded by polygonization. Unlike ray-tracing, the concrete representation is view independent. Thus, a given polygonal resolution may be excessive for some images and insufficient for others.

3.9 Notes

1. The BRep may be derived from a points-polygon set. Thus, we consider the product of polygonization, discussed in section 3.5, as equivalent to a boundary representation.

2. Nonetheless, many geometric shapes are best abstracted into Boolean set theoretic operations.

3. The top row of figure 3.2 was computed with $f(\boldsymbol{p}) = c - (\alpha\ d(\boldsymbol{p}, S_1)^2 + (1-\alpha)$

$d(\boldsymbol{p}, S_2)^2)^{½}$, where S_1 and S_2 are line segments and α ranged from 0 to 1. In the figure below, $f(\boldsymbol{p}) = c - \alpha\, d(\boldsymbol{p}, S_1) + (1-\alpha)\, d(\boldsymbol{p}, S_2)$. The tangent discontinuities of the contours result from C^1 discontinuities in the function d when $\boldsymbol{p}$ passes across S. In effect, unsigned distance d is discontinuous at $d = 0$; this discontinuity disappears for d^2.

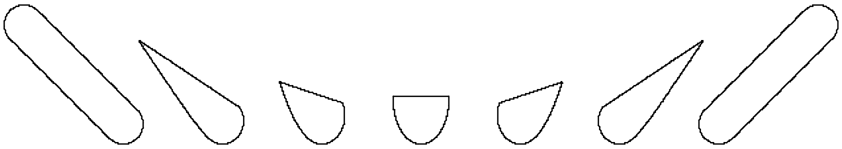


Figure 3.15 Contour Discontinuities

4. Incremental scan-line techniques can be used for certain classes of implicit surfaces [Blinn 1982, Sederberg 1989], but not for arbitrary implicit surfaces. As an alternative to surface rendering, the implicit *volume* may be visualized by volume rendering [Drebin *et al*. 1988] or slice rendering [Bloomenthal and Wyvill 1990], [Nielson *et al*. 1991].

5. ‘Chartable’ is a term suggested by Alyn Rockwood.

6. It is characteristic of implicit surfaces that spatial partitioning facilitates polygonization as well as accelerates ray-tracing and contour line-drawing.

7. This software is available through anonymous ftp from Princeton.edu, /pub/Graphics/GraphicsGems/GemsIV/.

8. This function is courtesy of Mark Ganter.

9. These same criteria can be used to determine an acceptable size for fixed-size polygonization.

10. And, just as ‘sampled data’ can be utilized in audio synthesizers, so can it be applied as textural detail to an otherwise smooth surface [Bloomenthal 1985].

11. Differences between discrete and continuous implicit surface functions are examined in [Ning and Bloomenthal 1993].

Do not worry about your difficulties in mathematics;
I can assure you that mine are still greater.
(Albert Einstein)