Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling

Barton T. Stander[1] John C. Hart[2]
School of EECS
Washington State University

Abstract

Morse theory shows how the topology of an implicit surface is affected by its function's critical points, whereas catastrophe theory shows how these critical points behave as the function's parameters change. Interval analysis finds the critical points, and they can also be tracked efficiently during parameter changes. Changes in the function value at these critical points cause changes in the topology. Techniques for modifying the polygonization to accommodate such changes in topology are given. These techniques are robust enough to guarantee the topology of an implicit surface polygonization, and are efficient enough to maintain this guarantee during interactive modeling. The impact of this work is a topologically-guaranteed polygonization technique, and the ability to directly and accurately manipulate polygonized implicit surfaces in real time.

Descriptors: I.3.5 Computational Geometry and Object Modeling — Modeling packages. **General Terms:** Algorithms.
Keywords: catastrophe theory, critical points, implicit surfaces, Morse theory, polygonization, topology, interval analysis, interactive modeling, particle systems.

1 Introduction

Shapes are represented implicitly by a function that classifies points in space as inside the shape, outside the shape or on the shape's surface, called the implicit surface. This representation provides computer graphics with geometric models that can be easily and smoothly joined together, but the incorporation of the implicit representation into graphics systems can be problematic. Polygonization of implicit surfaces allows graphics systems to reap the powerful modeling benefits of the implicit representation while retaining the rendering speed and flexibility of polygonal meshes.

The *topology* (specifically the *homotopy type*) of an implicit surface refers to the connectedness of the shape, including the number of disjoint components and the number of holes in each component (*genus*). This should not be confused with other instances of topology in computer graphics, such as the connection patterns of a mesh of polygons or parametric patches. Thus, for a polygonization to accurately represent the topology of an implicit surface the number of components and the genus of each component of the polygonization need to agree with that of the implicit surface.

Implicit surfaces are commonly polygonized to a given degree of geometric accuracy. This accuracy is in some cases adaptively related to the local curvature of the implicit surface, reducing the number of polygons without affecting the appearance of the surface. This work instead focuses on a polygonization that accurately discerns the topology of an implicit surface. Topologically-accurate polygonization can reduce the number of polygons without affecting the structure of the surface.

Guaranteeing the topology of a polygonization does not necessarily yield an accurate polygonization. A coffee cup is topologically equivalent to a torus, but the torus provides a poor representation for the geometry of a coffee cup. The topological guarantee becomes useful when coupled with a geometrically accurate polygonization scheme.

Guaranteeing the topology of an implicit surface polygonization solves several open problems in computer graphics.

Polygonization topology is coordinate dependent. Polygonization schemes often discern topology from point samples. Different polygonizations of the same implicit surface may differ in topology, and the same polygonization algorithm may return different topological structures depending on the coordinate system of the implicit surface, as demonstrated in Figure 1. For example, the implicit surface might appear connected when polygonized in its modeling coordinate system but could then appear disconnected when polygonized in a scene's coordinate system.

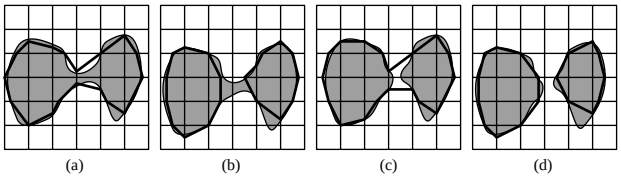
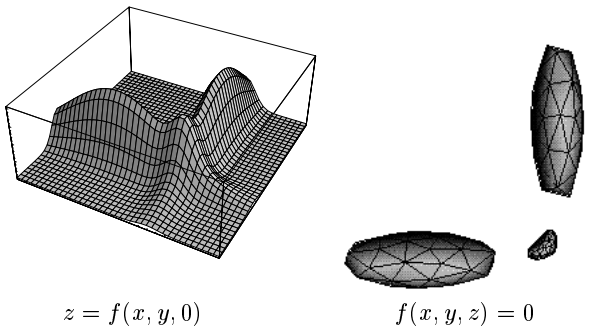


Figure 1: The topology of a connected implicit surface is correctly polygonized (a), but a translated instance is not (b). Two disjoint components are polygonized as a single component (c), but a translated instance is polygonized properly (d).

Interloping components. Some configurations of implicit surfaces can yield unexpected disjoint components. Such isolated components of the implicit surface occur because of an accumulation of neighboring potentials but do not themselves surround any "skele-

[1]Current address: Strata, 1562 El Vista Circle, Saint George, UT 84765, barts@strata3d.com
[2]Address: School of EECS, WSU, Pullman, WA 99164-2752, hart@eecs.wsu.edu

tal" geometry. For example, Figure 2 shows two blobby ellipsoids that produce a third component. Such components would not appear in a continuation-based polygonization[1], as might be used for modeling, but would appear as a surprise in a direct ray tracing of the implicit surface, as might be used for the final rendering.



$z = f(x, y, 0)$ $f(x, y, z) = 0$

Figure 2: An interloping component found at the intersection of the potential of two blobby ellipses (left) and two polygonized blobby ellipsoids (right). The algorithms described in this paper were used to correctly polygonize the interloping configuration on the right.

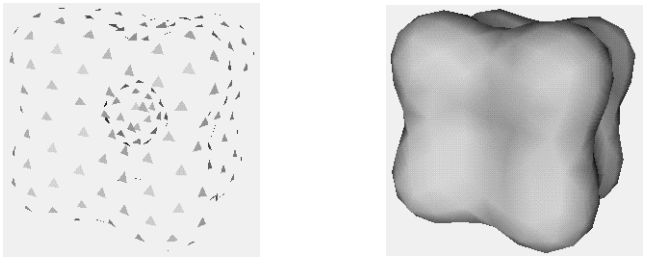


Figure 3: Implicit form displayed using a particle system (left) and polygonized (right).

Real-time implicit surface modeling. Implicit surfaces can be modeled and displayed in real-time using a particle system representation [35]. The viewer must then infer the shape of the implicit surface from the positions and orientations of the particles. Polygonization of the particles provides a better visual representation of the implicit surface, as demonstrated in Figure 3, but adding a polygonization step after every modeling change significantly degrades the otherwise real-time performance of the system. The ability to detect and correct topology changes allows the system to dynamically maintain the polygonization in real time by reconnecting the vertices of a polygonization only when a topology change has occurred, and only in the neighborhood of the topology change.

This work hence has two objectives. The first is to generate a polygonization of an implicit surface that is guaranteed to agree with its topology. The second objective is to maintain this topological guarantee during interactive manipulation of the surface. The topology of implicit surface polygonizations are guaranteed by tracking a few special points, called critical points, that dictate the topology of the implicit surface.

Section 2 examines previous polygonization techniques that consider topology. Section 3 describes critical points and adapts existing interval search methods and constraint techniques to the specific tasks of finding and tracking critical points. Section 4 analyzes the effect of critical points on topology and describes techniques for adjusting the topology of a polygonization to match the topology of the implicit surface. Section 5 describes a new polygonization algorithm based on Morse theory that polygonizes an implicit surface with a guarantee that the topology of the polygonization matches the topology of the implicit surface. Section 6 describes several new algorithms for maintaining this topological guarantee fast enough to support direct manipulation at interactive speeds. Section 7 concludes with a summary, implementation details and directions for further investigation.

[1]The techniques developed in this paper could be used to assist a continuation-based polygonization schemes detect such isolated components.

2 Previous Work

One method for interrogating an implicit surface subdivides space into cells and samples the implicit surface at the corners of these cells [22, 36, 16, 2, 20].

Some cellular polygonizations can guarantee that the implicit surface is contained in the union of a set of arbitrarily small cells, hence yielding a guarantee on surface topology accurate to a given geometric precision (e.g. the diameter of the smallest cells). Both the Lipschitz condition [14] and interval analysis [30, 17] can guarantee that an implicit surface does not pass through some cells. Cells for which this guarantee fails are "ambiguous" and can be subdivided until a given level of precision is reached and the implicit surface is assumed to lie within the union of the ambiguous cells. These ambiguous cells can then be further subdivided into "globally parameterizable" components [28]. The topology of the surface passing through such a component can be determined with a few point samples. Such cellular subdivision schemes yield a geometrically precise guaranteed representation of the implicit surface topology, though at the expense of a polygonization composed of an unnecessarily large number of small polygons.

An alternative method for interrogation constrains a particle system to the implicit surface [3, 33, 31, 9, 8]. When the implicit surface remains static, such as during a pause in user manipulation, it's particle system can be polygonized [6, 35].

A polygonization of the particle system can be maintained during user manipulation in a previous work [24]. Changes in topology were detected by comparing the interpolated polygonization normals to the implicit surface gradients at the vertices of the polygonization. Significant differences in these two vectors implied that the topology of the polygonization might not match the topology of the underlying implicit surface.

Critical points have also been used to determine implicit surface topology during a "shrinkwrap" polygonization [4]. A Lipschitz condition on the derivative of the function was used to guarantee the absence of critical points in a neighborhood, and upon failure, Newton's method was used to find the possible critical point. If Newton's method failed to converge, then the neighborhood was assumed to contain no critical points. The shrinkwrapping work also described a technique for repolygonizing the vertices surrounding a critical point based on the positions and orientations of the nearby polygons. Section 5 further explains and analyzes the shrinkwrap algorithm.

The analysis of topology using critical points is not entirely new to computer graphics. Critical points of vector and tensor fields are used to delineate topologically-distinct regions in the visualization of flow [13, 7]. Catastrophes have been used to understand caustics [18] and to interpret projections [15]. Morse theory has been used to reconstruct surfaces from cross sections [27] and to find surface-surface intersection curves [5].

3 Critical Points

The *implicit surface* defined by a function $f : \mathbb{R}^3 \to \mathbb{R}$ is the set of points $\mathbf{x} \in \mathbb{R}^3$ that satisfy $f(\mathbf{x}) = 0$. We assume the function

returns positive values inside the object, so the *solid* modeled by the implicit surface is the set of points $\{\mathbf{x}|f(\mathbf{x}) \geq 0\}$.

This work requires the function f to be C^2 continuous, with continuous first and second derivatives, and its implicit surface must be a manifold with a well defined, continuously varying surface normal. These restrictions include exponential-based "blobby" models [1], but exclude some of the more efficient C^1 piecewise polynomial approximations [21, 36].

The implicit surface is extended into a family of surfaces defined by $f(\mathbf{x};\mathbf{q})$ continuously parameterized by the vector $\mathbf{q}$ consisting of various model parameters (e.g. the locations of blobby elements). For some values of $\mathbf{q}$, the implicit surface defined by $f(\mathbf{x};\mathbf{q}) = 0$ may contain a cusp, kink or crease, specifically when the implicit surface changes topology. We consider the implicit surfaces in this family before and after but not during such topology changes. An alternative technique exists for interactively manipulating implicit surfaces with cusps, kinks and creases [25].

The *critical points* of a function f occur where its gradient

$$\nabla f(\mathbf{x}) = (f_x(\mathbf{x}), f_y(\mathbf{x}), f_z(\mathbf{x})) \tag{1}$$

vanishes. (The notation $f_x = \partial f/\partial x$.) A *critical value* is the value of the function f at a critical point.

The *Hessian* V (called the *stability matrix* in catastrophe theory) is defined as the Jacobian of the gradient

$$V(\mathbf{x}) = J(\nabla f(\mathbf{x})) = \begin{bmatrix} f_{xx}(\mathbf{x}) & f_{xy}(\mathbf{x}) & f_{xz}(\mathbf{x}) \\ f_{yx}(\mathbf{x}) & f_{yy}(\mathbf{x}) & f_{yz}(\mathbf{x}) \\ f_{zx}(\mathbf{x}) & f_{zy}(\mathbf{x}) & f_{zz}(\mathbf{x}) \end{bmatrix}. \tag{2}$$

Since $f_{xy} = f_{yx}$, etc., the stability matrix is symmetric.

A critical point $\mathbf{x}$ is classified based on the signs of the three eigenvalues $l_1 \leq l_2 \leq l_3$ of $V(\mathbf{x})$ [32]. If all three eigenvalues are non-zero, then the critical point is called *non-degenerate* and is either a maximum, minimum or some kind of saddle point. In three dimensions, saddle points come in two varieties. Table 1 indicates this classification.

l_1	l_2	l_3	Critical Point
-	-	-	Maximum Point
-	-	+	2-Saddle
-	+	+	1-Saddle
+	+	+	Minimum Point

Table 1: Classification of critical points based on the sign of the eigenvalues of the stability matrix.

The critical points are continuously dependent on the parameter vector $\mathbf{q}$. As the parameter vector $\mathbf{q}$ changes, the critical points move in space. They can also appear spontaneously in pairs, or collide in pairs, annihilating each other. If any of the three eigenvalues of the stability matrix of a critical point equal zero then $\mathbf{x}$ is called a *degenerate critical point*. The creation and destruction of critical points occur at degenerate critical points Critical point creation/destruction is demonstrated in Figure 4.

Isolated degenerate critical points are unstable. In the rare event that an isolated degenerate critical point does appear, it can be removed by a small perturbation of the implicit surface parameters without affecting the implicit surface topology.

Some functions can yields non-isolated degenerate critical points (critical sets). For example, the cylinder defined by $f(x,y,z) = x^2+y^2-1$ has a critical line along the z-axis. A small perturbation of the cylinder into an ellipsoid $f(x,y,z) = x^2+y^2+\epsilon z^2-1$ collapses the degenerate critical line into a single non-degenerate critical point at the origin. We assume the family of implicit surfaces is parameterized such that degenerate sets can be removed by such perturbation.

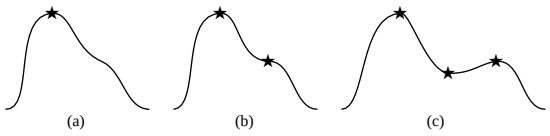


Figure 4: Creation of critical points in 1-D: $y = f(x,0,0)$. (a) Two summed Gaussian bumps, one large and one small, sufficiently close such that there is only a single maximum point in the domain shown. (b) Moving the smaller bump away from the larger creates a degenerate critical point. (c) Moving the smaller bump farther results in the creation of a pair of new critical points: a maximum point and a minimum point. Performing these steps in reverse demonstrates critical point annihilation.

3.1 Finding All Critical Points

Interval analysis searches can be guaranteed to find all points satisfying a given criterion in a given bounded domain to a desired degree of accuracy [19, 23]. Such a search can find all of the critical points of a given function to determine the topology of its implicit surface.

The interval search for critical points starts with an initial box bounding the space of interest. The simple interval search for critical points shown in Figure 5 eliminates large portions of space that cannot contain a critical point.

Given a box (a vector of intervals) $\mathbf{X} = [x_0,x_1] \times [y_0,y_1] \times [z_0,z_1]$ the algorithm checks whether the intervals returned by all of the partial derivatives contain zero. If not, then $\mathbf{X}$ contains no critical points. If so, then the algorithm subdivides $\mathbf{X}$ and tests each component individually. Note that $F_x(\mathbf{X})$ is an interval arithmetic implementation of $\partial f/\partial x$, and likewise for F_y, F_z.

```
Procedure SimpleSearch(X)
  If diam(X) < ε then indicate critical point in X.
  If 0 ∈ F_x(X) and 0 ∈ F_y(X) and 0 ∈ F_z(X) then
    Subdivide X and continue the search recursively.
```

Figure 5: Simple interval divide and conquer search algorithm.

In these algorithms, subdivision means dividing into halves with respect to its widest axis, although any number of subdivision techniques could be used. The diameter of a box $\mathrm{diam}(\mathbf{X})$ is measured using the chessboard metric, and is simply the width of the widest interval-element in the vector $\mathbf{X}$.

Simple subdivision performs remarkably well, discarding large portions of space known not to contain critical points. This technique will eventually finds all critical points to any degree of accuracy within a given bounding box, but with only linear convergence.

When the box diameter reaches a given size, the quadratically-convergent interval Newton's method shown in Figure 6 refines and/or subdivides the box down to the desired numerical precision [28, 11].

Given two points $\mathbf{x},\mathbf{y}$ there exist points $\mathbf{z}$ between[2] $\mathbf{x}$ and $\mathbf{y}$ such that

$$\nabla f(\mathbf{x}) + V(\mathbf{z})(\mathbf{y}-\mathbf{x}) = \nabla f(\mathbf{y}), \tag{3}$$

where the stability matrix V is the Jacobian of ∇f. Let $\mathbf{m}(\mathbf{X})$ return the midpoint of box $\mathbf{X}$. The algorithm seeks $\mathbf{y} \in \mathbf{X}$ such that $\nabla f(\mathbf{y}) = 0$. Since both $\mathbf{x},\mathbf{y} \in \mathbf{X}$, the $\mathbf{z}$ satisfying (3) must be in $\mathbf{X}$ as well. Thus, solving

$$\nabla f(\mathbf{m}(\mathbf{X})) + V(\mathbf{X})(\mathbf{Y}-\mathbf{m}(\mathbf{X})) = 0. \tag{4}$$

yields $\mathbf{Y}$, a box containing all of the critical points in $\mathbf{X}$. Note that

Procedure NewtonSearch($\mathbf{X}$)
  Repeat.
    Solve $V(\mathbf{X})(\mathbf{Y} - \mathbf{m}(\mathbf{X})) = -\nabla f(\mathbf{m}(\mathbf{X}))$ for Y.
    If $\mathbf{Y} \supset \mathbf{X}$ then subdivide $\mathbf{X}$ and search recursively.
    If $\mathbf{Y} \subset \mathbf{X}$ then there is a unique c.p. in $\mathbf{Y}$ (and $\mathbf{X}$).
    If $\mathbf{Y} \cap \mathbf{X} = \emptyset$ then there is no c.p in $\mathbf{X}$. Return.
    Otherwise let $\mathbf{X} = \mathbf{X} \cap \mathbf{Y}$.
  Until $\mathrm{diam}(\mathbf{X}) < \epsilon$.
  Indicate critical point at $\mathbf{m}(\mathbf{X})$.

Figure 6: Interval Newton's method search for critical points.

$V(\mathbf{X})$ returns a matrix of intervals.

An interval version of Gauss-Seidel is recommended for solving (4), but this can lead to two problems. First, the diagonal elements of $V(\mathbf{X})$ might contain zero, requiring the interval arithmetic division operation to correctly perform a division by an interval containing zero. Division by intervals containing zero produce two intervals, leading to additional algorithm recursion [28, 10]. Solving the rows whose diagonal elements do not contain zero first reduces the occurrence of semi-infinite intervals [11].

Solving

$$V_c^{-1} V(\mathbf{X})(\mathbf{Y} - \mathbf{x}) = -V_c^{-1} \nabla f(\mathbf{m}(\mathbf{X})). \tag{5}$$

where $V_c = m(V(\mathbf{X}))$ instead of (4) yields a much tighter bound and hastens the NewtonSearch performance [11]. Note that the expression $m(V(\mathbf{X}))$ returns a scalar matrix consisting of the midpoints of the intervals of $V(\mathbf{X})$.

When a critical point lies on an edge of the box, NewtonSearch's convergence is less quadratic. Extending $\mathbf{X}$ outward by a small percentage each iteration avoids this problem. Time may also be incorporated into the search by crossing $\mathbf{X}$ with the time interval $[t_0, t_1]$.

3.2 Tracking Critical Points

Altering an implicit surface's parameters changes the positions of some or all of the critical points.

The same techniques that constrain particles to adhere to the implicit surface [35], can also cause particles to adhere to any selected critical point.

Let $\mathbf{x} = \mathbf{x}(t)$ be a particle constrained to follow one of the critical points of the function f. Its partial derivatives $f_x(\mathbf{x}), f_y(\mathbf{x})$, and $f_x(\mathbf{x})$ are all constrained to zero. To ensure that they remain zero, their time derivatives $\dot{f}_x(\mathbf{x}) = \frac{d^2 f}{dxdt}(\mathbf{x})$, $\dot{f}_y(\mathbf{x})$ and $\dot{f}_z(\mathbf{x})$ must also be set to zero. Given the parameter vector $\mathbf{q}$ and its velocity $\dot{\mathbf{q}}$, one can solve these equations to determine the critical point velocity $\dot{\mathbf{x}}$. This velocity is then passed to a differential equation solver (such as fourth-order Runge-Kutta) to approximate the new location of the critical point. Newton's method refines the approximation.

4 Detecting and Correcting Topology Changes

The identification of critical points simplifies topologically-guaranteed direct manipulation of implicit surfaces through a polygonal representation. The key to solving the topology problem is that a change in the topology of a surface is always accompanied by a change in the sign of a critical value [12]. Monitoring the critical points greatly simplifies the burden of detecting topological changes, and divides the problem into classifying topological changes, identifying polygons to remove and reconnecting the verticed of the removed polygons.

[2] The notion of "between" for points in space means that $\mathbf{z}$ is in a box with corners at $\mathbf{x}$ and $\mathbf{y}$.

4.1 Classifying Topological Changes

Table 2 enumerates all of the possible critical-point/sign combinations and their corresponding implications on the implicit surface topology. When an implicit surface topology change is detected, the polygonization must be altered to properly represent the new topology.

Critical Point	Sign Changes To	Action
Maximum	-	Destroy
Maximum	+	Create
2-Saddle	-	Cut
2-Saddle	+	Attach
1-Saddle	-	Pierce
1-Saddle	+	Spackle
Minimum	-	Bubble
Minimum	+	Burst

Table 2: The affect of critical point sign on topology.

4.2 Identifying Polygons to Remove

Changes in maximum and minimum critical values cause entire simply-connected components of polygonization to be removed or created. Changes in saddle points require the determination of specific polygons to be removed such that their vertices may be properly reconnected. These polygons intersect a separatrix extending from the saddle point.

The separatrix may be efficiently approximated by a line for 2-saddles, or a plane for 1-saddles. These lines and planes described by the eigenvectors of the stability matrix of a critical point approximate the separatrix.

When separatrixes are linearly approximated by lines and planes, certain errors might occur. For example, a 2-saddle may connect two components, but the line approximating its separatrix might not intersect either component. One must then assume that the parameter vector $\mathbf{q}$ is sufficiently close to the parameter vector at the topology change $\mathbf{q}_*$ that the linear approximation correctly intersect the proper polygonized implicit surface components.

The separatrix extending from a 2-saddle can be treated as an initial value problem, using the positive eigenvector $\mathbf{v}_3(\mathbf{x})$ of the stability matrix to define the ordinary differential equation

$$\dot{\mathbf{x}} = \mathbf{v}_3(\mathbf{x}) \tag{6}$$

and using numerical integration to trace out the path of the separatrix. The midpoint method provided sufficient numerical accuracy for this task in our experiments.

The case where a separatrix intersects a polygonization vertex can be removed with a topology-preserving perturbation.

4.3 Reconnection

The following procedures describe which polygons must be removed, and how their vertices are reconnected to update the topology of the polygonization.

Destroy. When the value at a maximum goes negative, an isolated component in the implicit surface disappears. A ray cast from the maximum point in any direction will first intersect a polygon in this simply-connected component. All polygons connected to this polygon are then removed. Figure 7 pseudocodes this algorithm.

Create. When the value at a maximum goes positive, a new, simply-connected component in the implicit surface appears. A sufficiently small tetrahedron may be placed around the maximum point, letting its vertices adhere to the implicit surface component and adding

```
Procedure Destroy/Burst
  Cast a ray from maximum point in any direction.
  Let p be the first polygon the ray intersects.
  Push p onto stack.
  While stack not empty.
    Let p be the result of popping the stack.
    For all polygons q sharing an edge with p.
      Push q onto stack.
    Remove p from polygonization.
```

Figure 7: The repolygonization algorithm for *Destroy* and *Burst.*

new polygons when necessary. Alternatively, a ray may be cast from the maximum point and intersected with the implicit surface. The first intersection denotes the location where any standard continuation polygonization technique may be applied. Figure 8 pseudocodes this algorithm.

```
Procedure Create/Bubble
  Cast a ray from maximum point in any direction.
  Let x ∈ f^{-1}(0) be the first ray intersection.
  Polygonize the component containing x.
```

Figure 8: The repolygonization algorithm for *Create* and *Bubble.*

Cut. When the value at a 2-saddle goes negative, part of the implicit surface disconnects. The separatrix surface extending from the 2-saddle is found by integrating the two negative eigenvalues of the stability matrix will intersect the polygons in a ring surrounding the 2-saddle. In practical cases, this separatrix is sufficiently approximated by a plane passing through the 2-saddle perpendicular to its positive eigenvector. The ring of polygons intersecting the separatrix surface are removed, yielding two disjoint rings of polygonization vertices. These rings are "sewn up" individually via triangulation. Figure 9 pseudocodes this algorithm, and Figure 10 illustrates the polygon configuration.

```
Procedure Cut/Spackle
  Let P be a plane containing the critical point x perpendicular to the uniquely-signed eigenvector of V(x).
  Cast a ray from x in any direction within P.
  Let p_0 be the first polygon intersected by the ray.
  Initialize i = 0 and repeat.
    Let p_{i+1} be a polygon intersecting P, sharing an edge with p_i, and not equal to any p_j for j ≤ i.
    If no such p_{i+1} exists, break.
    Increment i.
  Let v_0 be any vertex of p_0.
  Call Procedure Ring.
  Triangulate v_i.
  Let v_0 be any vertex of p_0 not currently triangulated.
  Call Procedure Ring.
  Triangulate v_i.

Procedure Ring
  Initialize i = 0 and repeat.
    Let e be an edge connecting vertex v_i to v_{i+1} separating a p_k polygon from a non-p_k polygon, and vertex v_{i+1} not equal to any v_j for j ≤ i.
    If no such v_{i+1} exists, break.
    Increment i.
```

Figure 9: The repolygonization algorithm for *Cut* and *Spackle.*

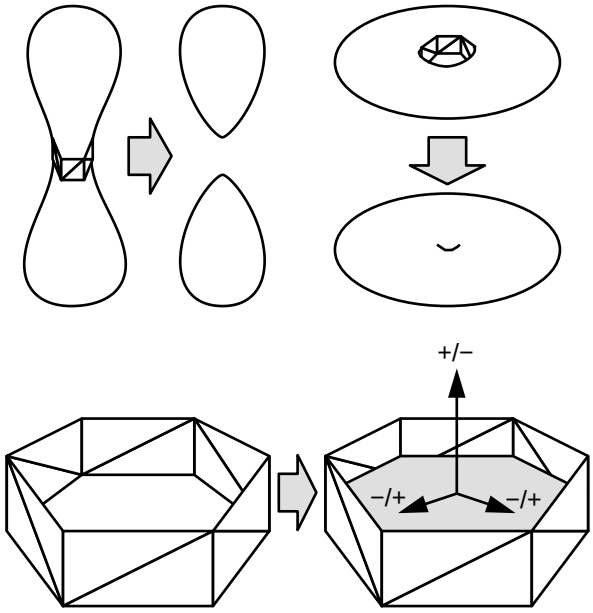


Figure 10: Polygons, eigenvalues and eigenvectors for *Cut* and *Spackle.*

Attach. When the value at a 2-saddle goes positive, two components of the implicit surface connect. The separatrix curve extends from the 2-saddle in the direction of its positive eigenvalue to a maximum point inside each component. The first polygon in each direction the separatrix intersects is removed. This leaves two disjoint rings of vertices that need to be connected. Proper correspondence algorithms between the two polygons can be found (e.g. [26]), but such techniques are not necessary if the polygonization is restricted to triangles. Figure 11 pseudocodes this algorithm.

```
Procedure Attach/Pierce
  Extend separatrix curves from the critical point.
  Let polygons p_0 and p_1 first intersect each separatrix.
  Connect the vertices of p_0 with the vertices of p_1.
  Remove p_0 and p_1.
```

Figure 11: The repolygonization algorithm for *Attach* and *Pierce.*

Pierce. When the value at a 1-saddle goes negative, a hole is pierced in the implicit surface. Similar to the *attach* case (a hole in the implicit surface of f is a connection in the implicit surface of $-f$), the two polygons that intersect the separatrix curve passing through the 1-saddle in the direction of the eigenvector corresponding to the one negative eigenvalue of the stability matrix are identified. These two polygons are removed and now form the ends of the hole. Corresponding and connecting the resulting two rings of vertices form the walls of the hole. The algorithm in Figure 11 also repolygonizes the *pierce* case.

Spackle. When the value at a 1-saddle goes positive, a hole in the implicit surface is filled. Similar to the *cut* case, the separatrix surface is constructed at the 1-saddle perpendicular to the eigenvector corresponding to the one negative eigenvalue of the stability matrix. The local polygons this surface pierces are removed, and the two resulting polygonal holes are "sewn up" by triangulation. The algorithm in Figure 9 also repolygonizes the *spackle* case and Figure 10 illustrates the polygon configuration.

Bubble. When the value at a minimum goes negative, a pocket of

air forms inside the implicit solid. An air pocket in the implicit surface of f is a new component in the implicit surface of $-f$. This pocket of air may therefore be treated as a simply-connected implicit surface component, and polygonized using any of the existing techniques. The algorithm in Figure 8 also repolygonizes the *bubble* case.

Burst. When the value at a minimum goes positive, an air bubble within the implicit solid has burst. As in the Destroy case, a ray is cast from the minimum in any direction. The first polygon this ray intersects, as well as any other polygons with a connection to it, are then removed. The algorithm in Figure 7 also repolygonizes the *burst* case.

5 Polygonization

Morse theory provides the background for a topologically-guaranteed polygonization algorithm. Given a function f(**x**) implicitly defining the surface $f^{-1}(0)$, we consider the family of surfaces $f^{-1}(a)$ for non-negative a. Let a_0 be a value of sufficient magnitude such that the surface $f^{-1}(a_0) = \emptyset$. As a_0 decreases, it will pass critical values for maximum points, 2-saddles, 1-saddles and minimum points. As each critical value is encountered, the topology of the polygonization around its critical point is corrected. This "inflation" algorithm is pseudocoded in Figure 12.

```
Procedure Inflate
  X_c = Search X for {x : ∇f(x) = 0}.
  Let a_0 > max_{x∈X_c} f(x).
  Polygonize f^{-1}(a_0).
  For a = a_0 − ε to 0 step −ε.
    Adjust vertices to f^{-1}(a).
    If ∃x ∈ X_c : a < f(x) < a + ε then
      Correct topology change in polygonization.
  Return polygonization of f^{-1}(0).
```

Figure 12: The "Inflate" polygonization algorithm.

When a maximum point is passed, a new simply-connected component appears via the *create* routine. When a 2-saddle is passed, a connection formed via the *attach* routine. When a 1-saddle is passed, a hole is filled via the *spackle* routine. When a minimum point is passed, a hollow bubble is filled via the *burst* routine. The *Inflation* polygonization of a blobby cube is demonstrated in Figure 13

Shrinkwrapping similarly polygonizes implicit surfaces but from the opposite direction, approaching the isovalue from the negative side [34]. Hence, the polygonization begins with a large simply-connected spheroid, which shrinks and appears to adhere to the final implicit surface. Morse theory can be incorporated to detect changes in topology during the shrinkwrapping process [4].

One problem with shrinkwrapping is that its outside-in processing fails to account for hollow bubbles within an implicit surface whereas *inflate's* inside-out processing correctly detects and polygonizes these regions. While such regions are typically hidden from the viewer, they become visible when the surface is rendered translucently or when the surface's polygonization is later intersected, trimmed, clipped or blended.

6 Interactive Repolygonization

The interaction algorithm consists of an initialization stage followed by an interactive loop of user input, model update, and model display. The system assumes it is initialized with a topologically correct polygonization, such as is described in Section 5.

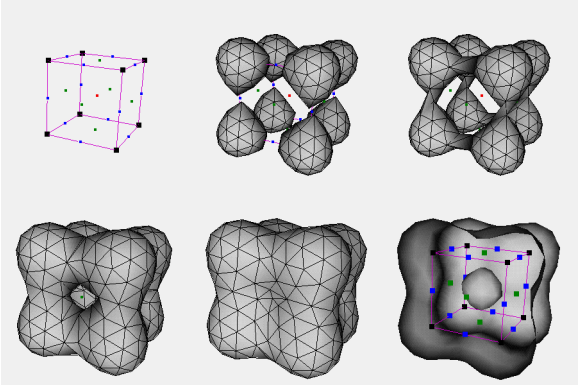


Figure 13: Polygonization via *Inflation* of the blobby cube. The last image illustrates the air bubble by only rendering back-facing polygons. Note that the definition of back facing may be the opposite of one's intuition for an air bubble.

```
Procedure Shrinkwrap
  X_c = Search X for {x : ∇f(x) = 0}.
  Let a_0 < min_{x∈X_c} f(x).
  Polygonize f^{-1}(a_0).
  For a = a_0 + ε to 0 step ε.
    Adjust vertices to f^{-1}(a).
    If ∃x ∈ X_c : a < f(x) < a + ε then
      Correct topology change in polygonization.
  Return polygonization of f^{-1}(0).
```

Figure 14: The "Shrinkwrap" polygonization algorithm [4].

For each time step, the interaction algorithm performs the steps in Figure 15.

```
Procedure InteractionLoop
  Repeat.
    Alter model parameters based on user manipulation.
    Adjust vertex positions, fix mesh.
    Determine critical points.
    Correct polygonization topology.
    Render.
```

Figure 15: The interaction loop for interactive implicit surface modeling.

Figure 16 shows a critical point tracking algorithm. During user interaction, the critical points move and change sign. Furthermore, one or more of the eigenvalues of the stability matrix can change sign at some degenerate critical point $\mathbf{x}$, resulting in the creation or annihilation of a pair of critical points. Critical point annihilation is revealed by the collision of two critical-point tracking particles. Critical point creation occurs spontaneously and is not detected by the tracking particles.

Searching in four-dimensions, as shown in Figure 17, allows us to find the location in space and time of degenerate critical points. Since critical points can be tracked and annihilation can be detected, the interval search would serve to detect the spontaneous creation of critical points. Such occurrences rarely happen, but when they do occur they appear as double zeros (both $\nabla f(\mathbf{x})$ and $|V(\mathbf{x})| = 0$) which degrades convergence.

An alternative search shown in Figure 18 finds the location in space and time for singular points in the family of implicit surfaces. where the value of a critical point changes sign. Such occurrences

Procedure Track-n-Search
Track critical points.
Search $\mathbf{X}$ for $\{\mathbf{x} : \nabla f(\mathbf{x}) = \mathbf{0}\}$.

Figure 16: The "Track-n-Search" critical point determination algorithm.

Procedure Track-n-SearchDegenerate
Track critical points.
Search $\mathbf{X} \times [t_0, t_1]$ for $\{\mathbf{x} : \nabla f(\mathbf{x}) = \mathbf{0}, |V(\mathbf{x})| = 0\}$.

Figure 17: The "Track-n-SearchDegenerate" critical point determination algorithm.

occur as rarely as degenerate critical points, so the interval search can quickly guarantee that such points do not exist. However, when they do exist, as with degenerate critical points, they appear as double zeros which degrades the rate of convergence.

7 Conclusion

Using techniques from catastrophe theory and Morse theory, the preceding sections developed (1) a new polygonization algorithm that can guarantee the topology of the polygonization matches that of the implicit surface, and (2) a new implicit surface modeling system capable of maintaining a topologically-accurate polygonized representation of the implicit surface during direct manipulation at interactive update rates.

Section 4 improves previous *ad hoc* geometry-only techniques [24, 4] by describing a mathematically sound method for using the separatrix to identify the polygons affected by a topology change, and robust algorithms for reconnecting the vertices of the polygonization.

Section 5 uses these techniques to polygonize an implicit surface. This method improves previous geometry-based interval methods [28] in that it is faster and does not return a large number of unnecessarily small polygons. The interval search is also guaranteed to find all critical points, which overcomes the uncertainty of previous methods [4], and also properly polygonizes hollow bubbles when they appear within an implicit surface.

Some initial experiments revealed that performance dropped below ten frames per second on scenes containing combinations of four or more interacting blobby ellipsoids. Modeling sessions that string a chain of blobby components operate in real time, but sessions with densely packed arrangements of blobby components appear sluggish in the current prototype implementation of the system. Even apparently simple configurations of blobby components can yield numerous nearly-degenerate critical points, and their detection is required for accurate topology management. This performance was measured using the SearchSingularity interaction loop, but is similar to the performance of the other interaction loop critical point search/tracking methods. This procedure becomes noticeably slow near topology changes. While any speed degradation and inconsistency is undesirable, the algorithm does focus its computation on the time and space where it is most needed.

Procedure SearchSingularity
Search $\mathbf{X} \times [t_0, t_1]$ for $\{\mathbf{x} : f(\mathbf{x}) = 0, \nabla f(\mathbf{x}) = \mathbf{0}\}$.

Figure 18: The "SearchSingularity" algorithm.

7.1 Some Implementation Tricks

One of the topological guarantee's restrictions was the lack of degenerate critical points. However, for speed, we were able to implement a C^2 cubic approximation to the exponential blobby model. The kernel of this approximation is uniform away from the center, and results in a three-dimensional degenerate critical set. We overcame this problem by assuming that the derivative intervals with zero for one endpoint did not contain a critical point, but instead contained a portion of this 3-D critical set. We avoided the possibility that a critical point fell on the boundary of the interval by expanding each interval by a small percentage.

Occasionally, the program errs in its attempt to process a topology change. In such cases, the system automatically initiates a full "inflation" repolygonization.

Further implementation details can be found in the dissertation [29].

7.2 Future Work

Tracking critical points is much faster than searching for them, but does not account for the pairs of critical points that can be created spontaneously. Tracking all of the derivatives of the function could detect degenerate critical points. This is not possible for exponentials because they are infinitely differentiable, and their derivatives become increasingly complex. Piecewise polynomials have finitely many derivatives that become increasingly simple, and might offer the opportunity to attempt such tracking.

Implicit surfaces still offer many challenges in modeling, texturing and animation due to the flexibility of their topology. This research solved the problem of interactive polygonization. Understanding the dynamics of critical points might lead to further solutions to other implicit surface problems, such as maintaining a consistent texturing during a topology change.

This research focused on 3-D implicit surfaces. Its application to the polygonization and modeling of 2-D implicit curves would be a useful, though perhaps now trivial, simplification.

7.3 Acknowledgments

This research was supported in part by the NSF Research Initiation Award #CCR-9309210. This research was performed in the Imaging Research Laboratory. The authors would like to thank the SIGGRAPH reviewers for their constructive criticism and positive comments. Further thanks are due to Dan Asimov, Jules Bloomethal and Jim Kajiya for their help in tracking down theorems in Morse theory. Special thanks to Andrew Glassner and Scott Lang for their assistance with the photoready copy of this paper.

REFERENCES

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

[2] BLOOMENTHAL, J. Polygonization of implicit surfaces. *Computer Aided Geometric Design 5*, 4 (Nov. 1988), 341–355.

[3] BLOOMENTHAL, J., AND WYVILL, B. Interactive techniques for implicit modeling. *Computer Graphics 24*, 2 (Mar. 1990), 109–116.

[4] BOTTINO, A., NUIJ, W., AND VAN OVERVELD, K. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 53–72.

[5] CHENG, K.-P. Using plane vector fieldsto obtain all the intersection curves of two general surfaces. In *Theory and Practice of Geometric Modeling* (New York, 1989), Springer-Verlag.

[6] DE FIGUEIREDO, L. H., DE MIRANDA GOMES, J., TERZOPOULOS, D., AND VELHO, L. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of Graphics Interface '92* (May 1992), pp. 250–257.

[7] DELMARCELLE, T., AND HESSELINK, L. The topology of symmetric, second-order tensor fields. *Proceedings IEEE Visualization '94* (October 1994), 140–147.

[8] DESBRUN, M., TSINGOS, N., AND GASCUEL, M.-P. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Implicit Surfaces '95 Proceedings* (April 1995), 171–185.

[9] FLEISCHER, K. W., LAIDLAW, D. H., CURRIN, B. L., AND BARR, A. H. Cellular texture generation. In *Computer Graphics (Annual Conference Series)* (Aug. 1995), pp. 239–248.

[10] HANSEN, E. A globally convergent interval method for computing and bounding real roots. *BIT 18* (1978), 415–424.

[11] HANSEN, E. R., AND GREENBERG, R. I. An interval newton method. *Applied Mathematics and Computation 12* (1983), 89–98.

[12] HART, J. C. Morse theory for computer graphics. Tech. Rep. EECS-97-002, Washington State University, May 1997. Also in: SIGGRAPH '97 Course #14 Notes "New Frontiers in Modeling and Texturing".

[13] HELMAN, J. L., AND HESSELINK, L. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications* (May 1991), 36–46.

[14] KALRA, D., AND BARR, A. H. Guaranteed ray intersections with implicit surfaces. *Computer Graphics 23*, 3 (July 1989), 297–306.

[15] KERGOSIEN, Y. L. Generic sign systems in medical imaging. *IEEE Computer Graphics and Applications 11*, 5 (Sep. 1991), 46–65.

[16] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3-d surface construction algorithm. *Computer Graphics 21*, 4 (July 1987), 163–170.

[17] MITCHELL, D. Three applications of interval analysis in computer graphics. In *Frontiers of Rendering*. SIGGRAPH '91 Course Notes, 1991.

[18] MITCHELL, D., AND HANRAHAN, P. Illumination from curved reflectors. *Computer Graphics 26*, 2 (July 1992), 283–291.

[19] MOORE, R. E. *Interval Analysis*. Prentice Hall, 1966.

[20] NING, P., AND BLOOMENTHAL, J. An evaluation of implicit surface tilers. *Computer Graphics and Applications 13*, 6 (Nov. 1993), 33–41.

[21] NISHIMURA, H., HIRAI, M., KAWAI, T., KAWATA, T., SHIRAKAWA, I., AND OMURA, K. Object modeling by distribution function and a method of image generation. In Proc. of *Electronics Communication Conference '85* (1985), pp. 718–725. (Japanese).

[22] NORTON, A. Generation and rendering of geometric fractals in 3-D. *Computer Graphics 16*, 3 (1982), 61–67.

[23] RATSCHEK, H., AND ROKNE, J. *Computer Methods for the Range of Functions*. John Wiley and Sons, 1984.

[24] RODRIAN, H.-C., AND MOOCK, H. Dynamic triangulation of animated skeleton-based implicit surfaces. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 37–52.

[25] ROSCH, A., RUHL, M., AND SAUPE, D. Interactive visualization of implicit surfaces with singularities. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 73–87.

[26] SEDERBERG, T. W., AND GREENWOOD, E. A physically based approach to 2-D shape blending. *Computer Graphics 26*, 2 (July 1992), 25–34.

[27] SHINAGAWA, Y., KUNII, T. L., AND KERGOSIEN, Y. L. Surface coding based on morse theory. *IEEE Computer Graphics and Applications 11*, 5 (Sep. 1991), 66–78.

[28] SNYDER, J. *Generative Modeling for Computer Graphics and CAD*. Academic Press, 1992.

[29] STANDER, B. T. *Polygonizing Implicit Surfaces with Guaranteed Topology*. PhD thesis, School of EECS, Washington State University, May 1997.

[30] SUFFERN, K., AND FACKERELL, E. Interval methods in computer graphics. In *Proc. AUSGRAPH 90* (1990), pp. 35–44.

[31] SZELISKI, R., AND TONNESEN, D. Surface modeling with oriented particle systems. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 185–194.

[32] TAYLOR, A. E. *Advanced Calculus*. Ginn and Company, 1955.

[33] TURK, G. Generating textures for arbitrary surfaces using reaction-diffusion. In *Computer Graphics (SIGGRAPH '91 Proceedings)* (July 1991), T. W. Sederberg, Ed., vol. 25, pp. 289–298.

[34] VAN OVERVELD, C., AND WYVILL, B. Shrinkwrap: an adaptive algorithm for polygonizing and implicit surface. Tech. Rep. 93/514/19, University of Calgary, Dept. of Computer Science, March 1993.

[35] WITKIN, A. P., AND HECKBERT, P. S. Using particles to sample and control implicit surfaces. In *Computer Graphics (Annual Conference Series)* (July 1994), pp. 269–278.

[36] WYVILL, G., MCPHEETERS, C., AND WYVILL, B. Data structure for soft objects. *Visual Computer 2*, 4 (1986), 227–234.