Eurographics Symposium on Geometry Processing (2006)
Konrad Polthier, Alla Sheffer (Editors)

Reliable Implicit Surface Polygonization using Visibility Mapping

Gokul Varadhan[1], Shankar Krishnan[2], Liangjun Zhang[1], and Dinesh Manocha[1†]

[1]Department of Computer Science, University of North Carolina, Chapel Hill, U.S.A
[2]AT&T Labs - Research, Florham Park, New Jersey, U.S.A
http://gamma.cs.unc.edu/VM/

Abstract

We present a new algorithm to compute a topologically and geometrically accurate triangulation of an implicit surface. Our approach uses spatial subdivision techniques to decompose a manifold implicit surface into star-shaped patches and computes a visibilty map for each patch. Based on these maps, we compute a homeomorphic and watertight triangulation as well as a parameterization of the implicit surface. Our algorithm is general and makes no assumption about the smoothness of the implicit surface. It can be easily implemented using linear programming, interval arithmetic, and ray shooting techniques. We highlight its application to many complex implicit models and boundary evaluation of CSG primitives.

Keywords: *contouring, Marching Cubes, set operations, implicit modeling, topology*

1. Introduction

Implicit surfaces are used to represent shapes of arbitrary topology in computer graphics and geometric modeling. As compared to other surface representations, implicits offer many advantages in terms of performing geometric operations like Boolean operations, blending, warping and offsets. Some early applications of implicit surfaces were the modeling of blobby shapes and of objects with biological or natural appearances. Recently, implicit surfaces have been shown to be useful for surface reconstruction, point-based modeling, and simulation.

In this paper, we address the problem of polygonizing an implicit surface. Our goal is to develop reliable techniques that preserve the topology of the implicit surface and compute a geometrically accurate polygonization. Topology of the implicit surface contains information about its connected components and genus. Many geometric operations like Boolean operations can result in a shape whose topology is very different from that of the primitives. It is important to capture all the topological features for CAD, medical and molecular modeling applications. Implicits are also used to reconstruct topologically accurate continuous surfaces from point clouds. These surfaces are defined as weighted combinations or blends of basis functions and have been applied to datasets consisting of millions of points. However, computing a topologically reliable polygonization of these complex implicit surfaces can be a challenge. Other applications of polygonization arise in mathematical visualization, where we want to accurately display the shape of complex surfaces.

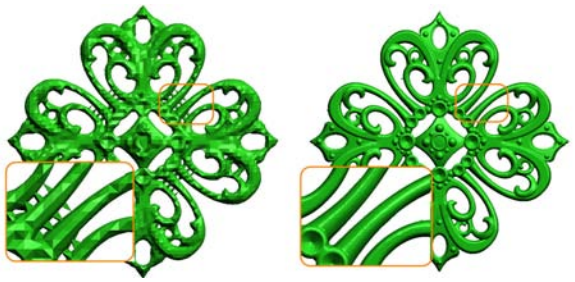


Figure 1: ***Polygonization of a complex MPU model:*** The right image shows a complex MPU model, Filigree, which has genus 65 and many topological features. This implicit was defined using 514K point samples by blending quadric surfaces using B-splines weights. The left image shows that previous spatial subdivision techniques may not compute an accurate polygonization, when the parameter used to select the grid size is not adequate. Our algorithm, based on Visibility mapping, generates an accurate polygonization of this model, as shown in the image on the right, in about 4 minutes.

The implicit surface polygonization problem has been studied for more than two decades. Prior methods can be classified into spatial subdivision techniques [WMW86, Blo88, HW90], scalar field isosurface reconstruction [LC87, KBSS01, JLSW02], and algorithms based on Morse theory [SH97]. Most applications use some combination of spatial subdivision and isosurface reconstruction. These algorithms are implemented using uniform or adaptive grids, and can handle complex models. In practice, their accuracy varies as a function of the grid resolution and the

† Supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043 and Intel.

coordinate system used for spatial subdivision. As a result, current spatial subdivision techniques may not provide rigorous guarantees on the topology of the reconstructed surface. On the other hand, polygonization algorithms based on Morse theory and critical point analysis can provide topological and geometric guarantees for smooth surfaces. However, these algorithms have been limited to relatively simple surfaces and do not extend to Boolean combinations.

Main Results: We present a new algorithm to polygonize implicit surfaces. Our approach is restricted to compact, manifold surfaces that can be represented as the zero set of a continuous function. We compute a topologically accurate polygonization of the implicit surface and construct a homeomorphism, i.e. a continuous bijection with a continuous inverse, between the implicit surface and the polygonal approximation. This homeomorphism is evaluated by computing ray intersections with the implicit surface.

Our algorithm uses spatial subdivision techniques along with interval arithmetic to decompose the implicit surface into surface patches. We ensure that each patch satisfies the *star-shaped* property, i.e. there exists a guard in space that can 'see' all the points on the patch. We use this property to compute a *visibility map* that projects each point on the patch to the boundary of the grid cell. Our algorithm computes a triangulation of the image of the visibility map on the boundary of the grid cell. Finally, these triangles are back-projected using the inverse of the visibility map to compute a homeomorphic triangulation of the patch. We ensure continuity between adjacent patches and generate a watertight polygonization of the overall surface. Some of the main *features* of our algorithm are:

Generality: Our algorithm is applicable to a broad class of implicits. These include zero sets of elementary functions, blends of locally fit smooth functions such as Multiple Partition-of-Unity (MPUs), Moving Least Squares (MLS), and Boolean combinations.

Reliable Polygonization: Given a manifold surface, our algorithm ensures that the polygonized approximation is topologically equivalent to the implicit surface. We also satisfy a two-sided Hausdorff bound between the original model and the polygonal approximation. Moreover, we do not need to perform any crack patching to handle adaptive grids.

Complex Models: Our approach can handle complex implicit surfaces that arise from point-cloud reconstruction algorithms or Boolean combinations. We use techniques based on spatial subdivision and voxelization to accelerate the computation.

Parameterization: Our algorithm computes a piecewise *star-shaped parameterization* of the implicit surface. Furthermore, the homeomorphism between the implicit surface and our polygonal approximation can be combined with mesh parameterization algorithms to compute a global parameterization of the implicit surface.

Quality of Polygonization: Star-shaped parameterization is a special case of spherical parameterization. This property can be exploited to generate a triangulation with good aspect ratios using the method proposed by Praun and Hoppe [PH03].

There exist prior implicit surface polygonization algorithms that provide topological guarantees. However, all of these algorithms assume a smooth implicit surface. As a result, they are not applicable to surfaces defined using Boolean operations. To the best of our knowledge, our algorithm is the first topology preserving polygonization algorithm that can handle Boolean operations. We have applied our algorithm to polygonize MPU surfaces generated using point cloud reconstruction algorithms, algebraic surfaces, and boundary evaluation of CSG models.

Organization: The rest of the paper is organized in the following manner. Section 2 presents our polygonization algorithm based on visibility mapping. We show that our algorithm computes a topologically and geometrically accurate polygonization in Section 3. We describe our spatial subdivision algorithm based on star-shaped decomposition in Section 4. We highlight the performance of our algorithm on different benchmarks in Section 5 and compare it with prior approaches in Section 6. Finally, we highlight some of its limitations in Section 7.

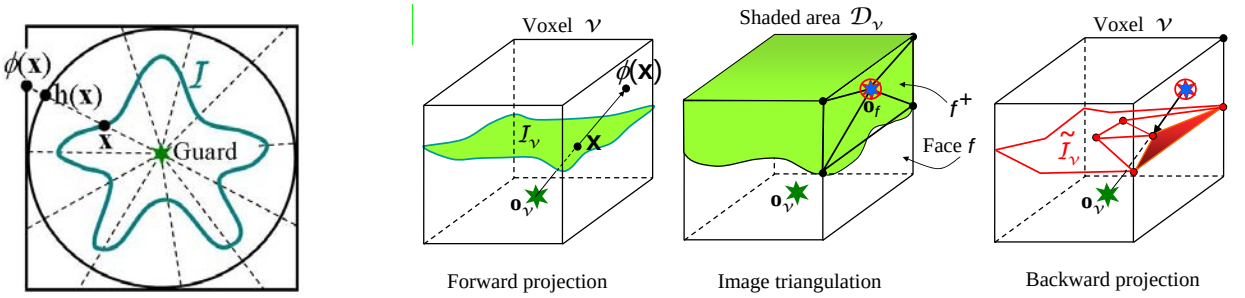
2. Implicit Surface Polygonization

Our algorithm uses a divide-and-conquer approach, subdividing the implicit surface into patches. In this section, we present our algorithm to polygonize each patch independently and to generate a watertight triangulation of the implicit surface. We first present our notation and define visibility mapping based on the star-shaped property.

2.1. Notation and Terminology

We use lower case bold letters such as $\mathbf{p}, \mathbf{q}$ to refer to points in $\mathbb{R}^3$. The notation $\mathbf{pq}$ denotes the line segment between the two points $\mathbf{p}$ and $\mathbf{q}$. The notation $(\mathbf{p}_1, \ldots, \mathbf{p}_n)$ denotes the n-sided polygon whose vertices are the points $\mathbf{p}_1, \ldots, \mathbf{p}_n$. Let $\mathscr{I} : \mathbb{R}^3 \rightarrow \mathbb{R}$ denote the implicit function; our goal is to compute a triangulation of its zero set. In practice, $\mathscr{I}$ could represent a function of the form $f(x,y,z)$. Furthermore, we allow CSG combinations of simple closed-form functions or other procedural models that can be expressed as elementary functions or weighted combination of simple elementary functions (e.g. blends, MPUs and MLS surfaces). With a slight abuse of notation, we will also use $\mathscr{I}$ to denote the implicit surface; we assume that $\mathscr{I}$ is a manifold. Our algorithm computes a topology-preserving polygonal approximation $\widetilde{\mathscr{I}}$ of the implicit surface $\mathscr{I}$. We construct a homeomorphism $\mathscr{H}$ between $\mathscr{I}$ and $\widetilde{\mathscr{I}}$.

Our algorithm uses a volumetric grid that has convex polyhedral cells. In the rest of the paper, for the sake of simplicity, we will assume that all the cells are cube-shaped. When referring to the cell as a geometric primitive, we will refer to it as a voxel. The boundary of a cell consists of faces, edges, and vertices and each of these is represented as a closed set. We use the symbols ϑ, f, e, and v to refer a voxel, a face, an edge, and a vertex, respectively.



(a) Visibility map (b) Our 3-stage approach

Figure 2: Fig. (a) shows that each point **x** on a star-shaped surface $\mathscr{I}$ can be mapped onto the the unit cube (or the unit sphere) using the visibility map ϕ (or h). Fig. (b) highlights the three steps of our algorithm: Forward projection that projects the star-shaped surface to the voxel boundary, image triangulation which triangulates the image on the voxel boundary, and backward projection, where the triangulation on the boundary is back-projected to compute triangulation of the implicit surface. We only show a subset of the triangles on the surface.

We use the symbol ∂S to denote the boundary of a set S. By a *restriction* of a set S with respect to another set T, we mean $S \cap T$, which we denote as S_T. Typically, S is a 2D manifold surface and T may correspond to a voxel or a face. The restriction operator has higher precedence over the boundary operator, i.e. $\partial S_T = \partial(S_T)$.

2.2. Visibility Mapping

In this subsection, we introduce visibility mapping and use it to polygonize implicit surfaces. We first present the intuition behind our algorithm. Let us assume that the implicit surface $\mathscr{I}$ is star-shaped. A surface is *star-shaped* if there exists a point in $\mathbb{R}^3$ (called a guard) that can 'see' every point on the surface. In essence, the star-shaped property of a surface captures its visibility. We exploit the fact that a star-shaped surface $\mathscr{I}$ has a star-shaped parametrization, a special case of spherical parameterization. Without loss of generality, we assume that the origin is the guard of $\mathscr{I}$. There exists a one-to-one map $h : \mathscr{I} \rightarrow \mathbb{S}^2$ that maps each point on $\mathscr{I}$ to a point on the unit sphere $\mathbb{S}^2$. This function is expressed as: $h(\mathbf{x}) = \frac{\mathbf{x}}{\|\mathbf{x}\|_2}$, where $\|\|_2$ denotes the Euclidean norm. The map $h()$ is the spherical projection operation. See Fig. 2(a). Note that if $\mathscr{I}$ is a closed star-shaped surface then h is a bijection and we can map $\mathscr{I}$ to $\mathbb{S}^2$ and vice-versa. In practice, we consider a mapping function $\phi : \mathscr{I} \rightarrow \mathbb{C}^2$ that maps $\mathscr{I}$ to the unit cube $\mathbb{C}^2$: $\phi(\mathbf{x}) = \frac{\mathbf{x}}{\|\mathbf{x}\|_\infty}$ where $\| \, . \, \|_\infty$ denotes the max-norm. Like h, ϕ is also a bijective mapping. We refer to ϕ as the *visibility map*. See Fig. 2(a). The visibility map can be thought of as a perspective projection operation onto the unit cube. It provides a simple method for triangulating $\mathscr{I}$: we first triangulate the boundary of $\mathbb{C}^2$ and then map these triangles to $\mathscr{I}$ using ϕ^{-1}. Since ϕ is a bijection, ϕ^{-1} is well-defined. This yields a triangulation of $\mathscr{I}$. Evaluating the visibility map ϕ (or its inverse ϕ^{-1}) is simple: it reduces to shooting a ray from the guard and computing intersection of the ray with the boundary of $\mathbb{C}^2$ (or with $\mathscr{I}$).

We extend the above idea to the case of a general implicit surface by adopting a divide-and-conquer approach. Conceptually, we decompose $\mathscr{I}$ into a set of star-shaped patches and polygonize each star-shaped patch using the visibility maps for that patch. Note that we compute a different map for each patch. In practice, we do not compute such a star-shaped decomposition explicitly but rather use spatial subdivision techniques (see Section 4). We also ensure that there is continuity between the polygonization of neighboring star-shaped patches. This results in a continuous water-tight triangulation of $\mathscr{I}$.

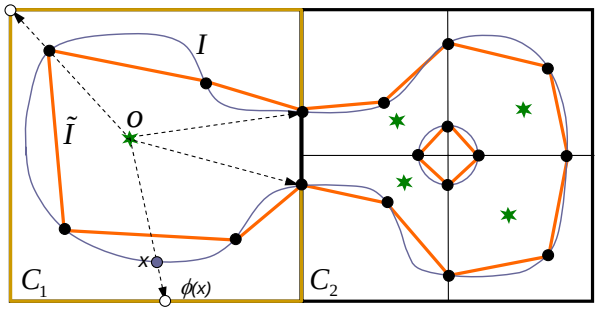


Figure 3: ***2D implicit curve approximation using our algorithm:*** The figure shows the 2D visibility grid for an implicit curve $\mathscr{I}$. The portion of $\mathscr{I}$ in cell C_1 is star-shaped property with respect to guard O. By projecting this portion onto the boundary of the cell C_1, we compute a one-to-one onto mapping ϕ between any point x on $\mathscr{I}$ and its projection $\phi(x)$on the boundary of the cell. We use this mapping to compute a homeomorphic polyline approximation of $\mathscr{I}$ within C_1. The cell C_2 is generated after an additional level of quadtree subdivision; the portion of $\mathscr{I}$ in C_2 is also star-shaped.

2.3. Polygonization within a Cell

Our algorithm uses spatial subdivision techniques like octree decomposition to generate a grid that satisfies certain visibility properties. We call a volumetric grid $\mathscr{G}$ a **visibility grid** if it satisfies the following conditions:

1. Every voxel ϑ in $\mathscr{G}$, $\mathscr{I} \cap \vartheta$ is star-shaped with respect to (w.r.t) some point $\mathbf{o}_\vartheta$ in ϑ. We call the point $\mathbf{o}_\vartheta$ a guard of ϑ.
2. Every face f in $\mathscr{G}$, $\mathscr{I} \cap f$ is star-shaped w.r.t some point $\mathbf{o}_f$ in f. We call the point $\mathbf{o}_f$ a guard of f.

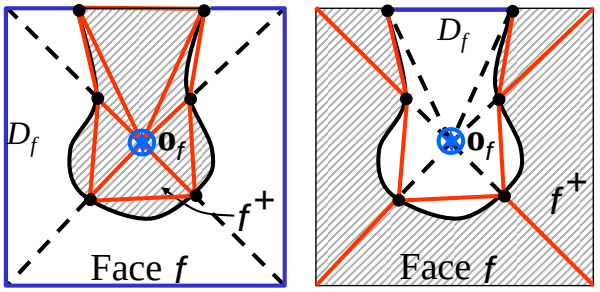


Figure 4: Image triangulation: The two figures show two different cases of image triangulation of a face f depending on whether the guard $\mathbf{o}_f$ of face f belongs to f^+ or not. In each case, we compute a triangulation $\widetilde{f^+}$ of f^+.

The visibility grid enforces the star-shaped property within every voxel and face of the grid. Our algorithm requires that that $\mathscr{I}$ intersect the cells in the visibility grid in a non-degenerate manner, i.e., $\mathscr{I}$ should not tangentially intersect any of the voxels, faces, or edges of the grid.

Let us consider $\mathscr{I}_\vartheta = \mathscr{I} \cap \vartheta$, the restriction of $\mathscr{I}$ to the voxel ϑ of a cell. $\mathscr{I}_\vartheta$ is star-shaped and has a guard $\mathbf{o}_\vartheta$ that lies inside ϑ. Without loss of generality, we assume that $\mathbf{o}_\vartheta$ lies inside $\mathscr{I}$. Our algorithm can be easily extended to handle the symmetric case when $\mathbf{o}_\vartheta$ lies outside $\mathscr{I}$. Our overall approach consists of three main steps:

1. **Forward projection:** We define a map $\phi_\vartheta : \mathscr{I}_\vartheta \rightarrow \mathscr{D}_\vartheta$ that projects each point on $\mathscr{I}_\vartheta$ to a portion $\mathscr{D}_\vartheta$ of the boundary of the voxel. We refer to ϕ_ϑ as the *forward visibility map* and $\mathscr{D}_\vartheta$ as the *image* of the forward visibility map.
2. **Image triangulation:** We triangulate $\mathscr{D}_\vartheta$ to compute a triangulation $\widetilde{\mathscr{D}}_\vartheta$.
3. **Backward projection:** We backproject the triangles in $\widetilde{\mathscr{D}}_\vartheta$ to obtain a triangulation $\widetilde{\mathscr{I}}_\vartheta$ of $\mathscr{I}$. The backward projection is achieved using a map $\widetilde{\phi}_\vartheta : \widetilde{\mathscr{D}}_\vartheta \rightarrow \widetilde{\mathscr{I}}_\vartheta$, which is defined using the inverse of the forward visibility map. We refer to $\widetilde{\phi}_\vartheta$ as the *backward visibility map*.

We now explain each of the above three steps in more detail. Fig. 3 illustrate our algorithm for a 2D planar curve.

2.3.1. Forward Projection

We define a function $\tau_\vartheta : \mathscr{I}_\vartheta \rightarrow \partial\vartheta$ that maps $\mathscr{I}_\vartheta$ to the boundary of the voxel (see Fig. 2).

$$\tau_\vartheta(\mathbf{x}) = \mathbf{o}_\vartheta + \parallel \vartheta \parallel \phi(\mathbf{x} - \mathbf{o}_\vartheta),$$

where $\parallel \vartheta \parallel$ denotes half of the length of voxel ϑ. It follows that τ_ϑ is a 1-1 map. In general, if $\mathscr{I}$ intersects the boundary of the voxel, τ_ϑ will not be an onto function. Let $\mathscr{D}_\vartheta = \tau_\vartheta(\mathscr{I}_\vartheta)$ denote the range of τ_ϑ, as shown in Fig. 2(b). In the rest of the paper, we will refer to the bijection $\phi_\vartheta : \mathscr{I}_\vartheta \rightarrow \mathscr{D}_\vartheta$, where $\phi_\vartheta(\mathbf{x}) = \tau_\vartheta(\mathbf{x})$. ϕ_ϑ is the *forward visibility map* and $\mathscr{D}_\vartheta$ is its *image*. Furthermore, $\phi_\vartheta(\mathbf{x})$ can be evaluated by shooting a ray from $\mathbf{o}_\vartheta$ in the direction of $\mathbf{x}$ and computing the intersection of the ray with the boundary of the voxel.

2.3.2. Triangulating the Image

Our goal is to compute a triangulation of $\mathscr{D}_\vartheta$. We need to ensure C^0 continuity between the triangulations of $\mathscr{I}$ between adjacent voxels. We utilize the fact that $\mathscr{D}_\vartheta$ consists of points on the boundary of ϑ that are not visible to the guard of ϑ, i.e. the ray from the guard to such a point must intersect $\mathscr{I}$. The star-shaped property ensures that such a point must necessarily lie outside $\mathscr{I}$. We use this property to express $\mathscr{D}_\vartheta$ in terms of faces of ϑ. Let $f_i, i = 1, \ldots, 6$, denote the faces of ϑ. Let f_i^+ be the set of points in f_i that lie outside $\mathscr{I}$, i.e.,

$$f_i^+ = \{\mathbf{x} \mid \mathbf{x} \in f_i, \mathscr{I}(\mathbf{x}) > 0\}.$$

$\mathscr{D}_\vartheta$ is given by: $\mathscr{D}_\vartheta = \bigcup_{i=1}^{6} f_i^+$, as shown in Fig. 2. Therefore the problem of triangulating $\mathscr{D}_\vartheta$ reduces to triangulating each f_i^+, which is a portion of face f_i.

It suffices to triangulate just *minimal faces*; a face f is minimal if no face in the grid is a strict subset of f. A non-minimal face is a union of a set of minimal faces. We now present our algorithm to triangulate f^+ for a minimal face f in the visibility grid. We do not compute f^+ explicitly; rather, we take advantage of the star-shaped property, i.e., $\mathscr{I} \cap f$ is star-shaped w.r.t a point $\mathbf{o}_f \in f$. We define a visibility mapping in 2D to compute this triangulation. Our approach again performs three main steps:

1. **Forward projection:** We define a 2D forward visibility map $\phi_f : \mathscr{I}_f \rightarrow \mathscr{D}_f$, which projects $\mathscr{I}_f$ onto a portion of ∂f. This visibility map is defined on a 2D domain and is similar to ϕ_ϑ. $\mathscr{D}_f$, the image of ϕ_f, is a polyline.
2. **Image triangulation:** Divide $\mathscr{D}_f$ into a set of line segments as explained below.
3. **Backward projection:** Backproject the line segments to obtain a 'triangulation $\widetilde{f^+}$ of f^+.

We consider two separate cases based on whether or not $\mathbf{o}_f$ belongs to f^+:

Case 1: $\mathbf{o}_f \in f^+$
First we compute $\mathscr{D}_f$, the image of ϕ_f. In this case, $\mathscr{D}_f$ consists of those points in ∂f that lie inside $\mathscr{I}$. Therefore, $\mathscr{D}_f$ can be computed by finding the intersections points between $\mathscr{I}$ and the edges of f. These intersection points together with the vertices of f partition ∂f into two sets of line segments. A subset $\mathscr{L}_1$ of these line segments belong to $\mathscr{D}_f$. Let $\mathscr{L}_2$ denote the remainder of line segments.

1. For each edge, $\mathbf{ab} \in \mathscr{L}_1$, output a triangle $(\mathbf{o}_f, \phi_f^{-1}(\mathbf{a}), \phi_f^{-1}(\mathbf{b}))$.
2. For each edge $\mathbf{ab} \in \mathscr{L}_2$, output a triangle $(\mathbf{o}_f, \mathbf{a}, \mathbf{b})$.

This is illustrated in Fig. 4. The union of all the triangles generated by these steps is the triangulation $\widetilde{f^+}$ of f^+. We evaluate ϕ_f^{-1} by shooting a ray from $\mathbf{o}_f$ and computing the intersection of the ray with $\mathscr{I}$.

Case 2: $\mathbf{o}_f \notin f^+$
As in Case 1, we first compute a set of line segments, $\mathscr{L}_1$, which belong to $\mathscr{D}_f$. In this case, $\mathscr{D}_f$ consists of those points

in ∂f that lie outside $\mathscr{I}$. For each edge $\mathbf{ab} \in \mathscr{L}_1$, we output a quadrilateral $(\mathbf{a}, \mathbf{b}, \phi_f^{-1}(\mathbf{b}), \phi_f^{-1}(\mathbf{a}))$. We triangulate each quadrilateral using one of its diagonals. This quadrilateral may degenerate into a triangle if either **a** or **b** corresponds to an intersection point between $\mathscr{I}$ and ∂f (Fig. 4). The union of all the triangles generated above is the triangulation $\widetilde{f^+}$ of f^+.

Image Retriangulation: A finer triangulation $\widetilde{f^+}$ can be obtained by refining the initial triangulation of f^+. This can be computed in two ways: (a) by subdividing the line segments in $\mathscr{L}_1$ and $\mathscr{L}_2$ or (b) by subdividing the triangles in $\widetilde{f^+}$.

2.3.3. Backward Projection

We use the method presented above to triangulate f^+ for each face f of the cell. We represent this triangulation of $\mathscr{D}_\vartheta$ as $\widetilde{\mathscr{D}}_\vartheta$: $\widetilde{\mathscr{D}}_\vartheta = \bigcup_{i=1}^{6} \widetilde{f_i^+}$. Each triangle in $\widetilde{\mathscr{D}}_\vartheta$ has the property that all its vertices belong to $\mathscr{D}_\vartheta$. Therefore any point $\mathbf{x} \in \widetilde{\mathscr{D}}_\vartheta$ can be expressed in terms of barycentric coordinates:

$$\mathbf{x} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c},$$
$$u, v, w \geq 0, \quad u + v + w = 1, \quad \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathscr{D}_\vartheta.$$

We define a map $\widetilde{\phi}_\vartheta : \widetilde{\mathscr{D}}_\vartheta \to \mathbb{R}^3$:

$$\widetilde{\phi}_\vartheta(\mathbf{x}) = u\phi_\vartheta^{-1}(\mathbf{a}) + v\phi_\vartheta^{-1}(\mathbf{b}) + w\phi_\vartheta^{-1}(\mathbf{c}).$$

Since **a**,**b**, and **c** belong to $\mathscr{D}_\vartheta$, $\phi_\vartheta^{-1}(\mathbf{a})$,$\phi_\vartheta^{-1}(\mathbf{b})$, and $\phi_\vartheta^{-1}(\mathbf{c})$ are well-defined. We refer to $\widetilde{\phi}_\vartheta$ as the *backward visibility map*. In essence, the backward map back-projects the triangles in $\widetilde{\mathscr{D}}_\vartheta$ by using the inverse of the forward visibility map ϕ_ϑ and barycentric coordinates. The backward projection produces the approximation $\widetilde{\mathscr{I}}_\vartheta = \widetilde{\phi}_\vartheta(\widetilde{\mathscr{D}}_\vartheta)$ in the voxel. The overall approximation $\widetilde{\mathscr{I}}$ corresponds to the union of the approximations within the individual voxels:

$$\widetilde{\mathscr{I}} = \cup_\vartheta \widetilde{\mathscr{I}}_\vartheta$$

The resulting approximation $\widetilde{\mathscr{I}}$ is a continuous, watertight polygonal surface.

2.4. Continuity

The approximation $\widetilde{\mathscr{I}}$ has C^0 continuity. This follows from the fact that the approximation within two adjacent voxels "match" along the common face. Consider two voxels ϑ_i, ϑ_j that share a face f, i.e., $\vartheta_i \cap \vartheta_j = f$. Then the images of the forward visibility maps of ϑ_i and ϑ_j are identical along face f. Our image triangulation procedure also maintains this property. Therefore, we have $\mathscr{D}_{\vartheta_i} \cap f = \mathscr{D}_{\vartheta_j} \cap f$ and $\widetilde{\mathscr{D}}_{\vartheta_i} \cap f = \widetilde{\mathscr{D}}_{\vartheta_j} \cap f$. Since the approximations $\widetilde{\mathscr{I}}_{\vartheta_i}$ and $\widetilde{\mathscr{I}}_{\vartheta_j}$ are obtained by backprojecting $\widetilde{\mathscr{D}}_{\vartheta_i}$ and $\widetilde{\mathscr{D}}_{\vartheta_j}$, these approximations 'match' along f, i.e., $\partial(\widetilde{\mathscr{I}}_{\vartheta_i}) \cap f = \partial(\widetilde{\mathscr{I}}_{\vartheta_j}) \cap f$. The following lemma is used to formally prove the continuity:

LEMMA 1 *Let ϑ_i and ϑ_j be two adjacent voxels that share a face f, i.e., $\vartheta_i \cap \vartheta_j = f$. Then $\partial(\widetilde{\mathscr{I}}_{\vartheta_i}) \cap f = \partial(\widetilde{\mathscr{I}}_{\vartheta_j}) \cap f$.*

We note that the same formulation also applies to adaptive grids.

3. Accuracy of Polygonization

In this section, we analyze topological properties of our polygonization algorithm. We also extend the polygonization algorithm to improve its accuracy and compute a parameterization.

3.1. Topology Preservation

Our goal is to prove that $\widetilde{\mathscr{I}}$ is topologically equivalent to $\mathscr{I}$. In particular, we show that there exists a homeomorphism between $\widetilde{\mathscr{I}}$ and $\mathscr{I}$. Our proof follows the following outline:

1. We define a *local map* $\mathscr{H}_\vartheta : \mathscr{I}_\vartheta \to \widetilde{\mathscr{I}}_\vartheta$ in each voxel ϑ. We show that $\mathscr{H}_\vartheta$ is a bijection. Furthermore, we prove that the local maps of two adjacent voxels match each other along the common face. We combine these local maps to obtain a homeomorphism $\mathscr{H} : \mathscr{I} \to \widetilde{\mathscr{I}}$, and thereby establish topological equivalence between $\mathscr{I}$ and $\widetilde{\mathscr{I}}$.
2. The local map $\mathscr{H}_\vartheta$ is a composition of three bijections:
   a. The forward visibility map ϕ_ϑ.
   b. *The image transfer map*: For each face f, we define a map $\delta_f : f^+ \to \widetilde{f^+}$ that maps the image f^+ on face f to its triangulation $\widetilde{f^+}$. We refer to δ_f as the *image transfer map* of f. By combining the image transfer maps of the faces, we obtain an image transfer map $\delta_\vartheta : \mathscr{D}_\vartheta \to \widetilde{\mathscr{D}}_\vartheta$ for the voxel.
   c. The backward visibility map $\widetilde{\phi}_\vartheta$.

   The local map $\mathscr{H}_\vartheta$ is equal to $\widetilde{\phi}_\vartheta \circ \delta_\vartheta \circ \phi_\vartheta$.

We state the main results of topological equivalence. The proofs of all these lemmas and the theorem are presented in the appendix. We first state the properties of the image transfer map of a face and voxel.

LEMMA 2 *Given a face f in the visibility grid, there exists a bijection $\delta_f : f^+ \to \widetilde{f^+}$.*

LEMMA 3 *Given a voxel ϑ in the visibility grid, there exists a bijection $\delta_\vartheta : \mathscr{D}_\vartheta \to \widetilde{\mathscr{D}}_\vartheta$.*

Since $\mathscr{H}_\vartheta$ is a composition of bijective mappings, it is a bijection. See Appendix. This fact combined with the fact that our approximation generates a C^0 mesh can be used to arrive at the following main result (proof of result is in [VKZM06] and the supplementary material provided with this paper):

THEOREM 1 *$\mathscr{I}$ is homeomorphic to $\widetilde{\mathscr{I}}$ ($\mathscr{I} \approx \widetilde{\mathscr{I}}$).*

Evaluating the Homeomorphism: Our formulation of the homeomorphism is constructive. We can evaluate $\mathscr{H}$ and its inverse using ray-shooting. Consider a point $\mathbf{x} \in \mathscr{I}$. Let **x** belong to a voxel ϑ. Then $\mathscr{H}(\mathbf{x}) = \mathscr{H}_\vartheta(\mathbf{x})$. Recall that $\mathscr{H}_\vartheta$ is a composition of three bijections: (a) the forward visibility map ϕ_ϑ, (b) δ_ϑ, and (c) the backward visibility map $\widetilde{\phi}_\vartheta$. Each of the three bijections can be evaluated by ray-shooting. Therefore, computing $\mathscr{H}_\vartheta(\mathbf{x})$ also reduces to ray-shooting. We use this property in Section 3.5 to compute a parameterization of the implicit surfaces.

3.2. Geometric Accuracy

We extend our algorithm to generate a polygonization $\widetilde{\mathscr{I}}$ with a bounded Hausdorff error. Given any $\varepsilon > 0$, our algorithm outputs $\widetilde{\mathscr{I}}$ such that the two-sided Hausdorff distance between $\widetilde{\mathscr{I}}$ and $\mathscr{I}$ is less than ε. One way to bound this error is to require that all the grid cells in the volumetric grid be smaller than ε. However, this can incur a huge penalty in the performance as a large number of star-shaped tests would need to be performed (Section 4). Rather, we bound $\mathscr{I}$ by a set of voxels of size ε and check if $\widetilde{\mathscr{I}}$ is contained by this set of voxels. Such a voxelization is precomputed for the implicit surface using interval arithmetic (see Section 4.3).

When $\mathscr{I}$ is defined in terms of Boolean operations over a set of primitives, we exploit the fact that the boundary of $\mathscr{I}$ is a subset of the union of boundaries of the primitives. Consider a voxel ϑ that is intersected by n primitives, $\mathscr{S}_1, \ldots, \mathscr{S}_n$. In this case, we check if the Hausdorff distance between $\widetilde{\mathscr{I}}_\vartheta$ and $\mathscr{S}_i \cap \vartheta$ is less than ε for each $i = 1, \ldots, n$. This provides a sufficient condition under which the Hausdorff distance between $\widetilde{\mathscr{I}}$ and $\mathscr{I}$ is also bounded by ε. A different technique for bounding Hausdorff error is presented in [AAB02].

3.3. Quality of Polygonization

We can also generate a triangulation with good aspect ratios by using the remeshing techniques proposed by [PH03, THCM04]. Each patch in our case has a *star-shaped parameterization*, which is a special case of spherical parameterization – a property that is utilized by these remeshing techniques. We can use these techniques to guide the image triangulation step of our algorithm (Section 2.3.2). Since the image triangulation is done on the faces of the grid, we automatically ensure continuity across adjacent voxels.

3.4. Sharp Features

We use a technique similar to Dual contouring [JLSW02] to obtain an accurate polygonization near sharp features. For each triangle in $\widetilde{\mathscr{I}}$, we use the normals at the three vertices to estimate a *minimizing vertex* as described in [JLSW02]. Then we output a dual polygonal mesh by combining the minimizing vertices of adjacent triangles: each triangle in $\widetilde{\mathscr{I}}$ corresponds to a vertex in the dual mesh, an edge in $\widetilde{\mathscr{I}}$ corresponds to a dual edge, and a vertex in $\widetilde{\mathscr{I}}$ corresponds to a polygon in the dual mesh. By triangulating these polygons, we obtain a triangulation that can capture sharp features.

3.5. Parameterization

A star-shaped surface can be parameterized using spherical coordinates, i.e. latitude and longitude. Both the original implicit surface $\mathscr{I}$ and our approximation $\widetilde{\mathscr{I}}$ consist of star-shaped patches, denoted as $\mathscr{I}_\vartheta$ and $\widetilde{\mathscr{I}}_\vartheta$. As a result, our algorithm automatically computes a piecewise star-shaped parameterization of $\mathscr{I}$.

Our algorithm can also compute a global parameterization of implicit surfaces. In general, computing a good parameterization of implicit surfaces is a difficult problem [Ped95],

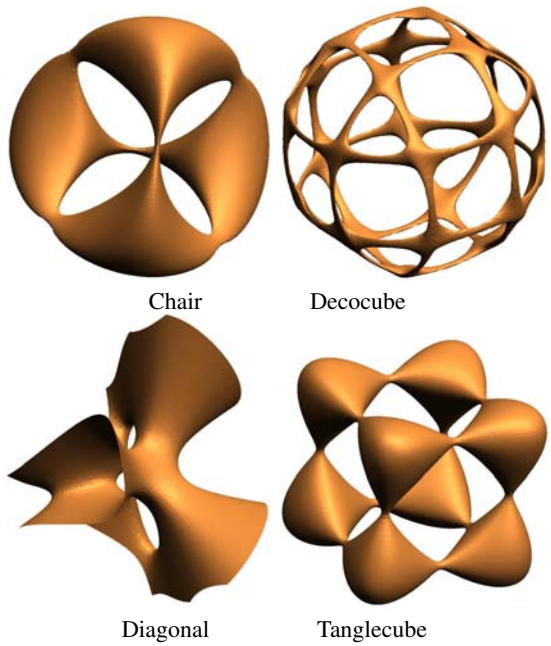


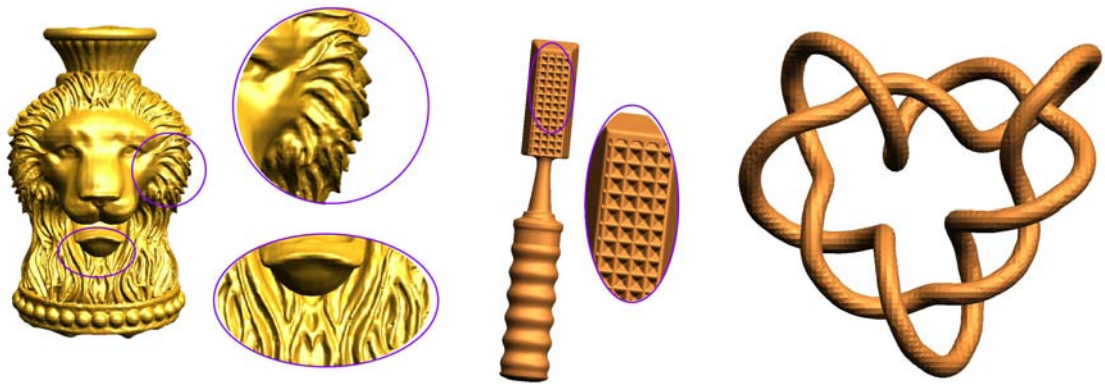
Figure 5: ***Polygonization of algebraic surfaces:*** Our algorithm is able to compute reliable polygonization of these high genus algebraic surfaces. The running time is about 10 sec for these surface, except for Decocube that takes about 52 seconds.

In contrast, there exist a host of techniques to parameterize polygonal meshes [FH05]. Our algorithm reduces the problem of implicit surface parameterization to a mesh parameterization problem. We exploit the fact that our algorithm constructs a homeomorphism $\mathscr{H} : \mathscr{I} \to \widetilde{\mathscr{I}}$ between the implicit surface and our polygonal approximation. Furthermore, $\mathscr{H}$ can be evaluated by ray-shooting. Therefore, by computing a mesh parameterization $\mu : \mathscr{X} \to \widetilde{\mathscr{I}}$, we obtain an implicit surface parameterization $\lambda : \mathscr{X} \to \mathscr{I}$, where $\lambda = \mathscr{H}^{-1} \circ \mu$.

4. Spatial Subdivision

Our polygonization algorithm assumes that each patch of the implicit surface restricted to a voxel/face of a cell is star-shaped. In this section, we present our spatial subdivision algorithm to decompose the space into a visibility grid with respect to $\mathscr{I}$. We perform an octree subdivision and check whether the portion of $\mathscr{I}$ restricted to a voxel ϑ, i.e. $\mathscr{I}_\vartheta$, is star-shaped. A similar test is also performed on the faces of each cell.

In general, a star-shaped surface does not have a unique guard. The set of all guards is called the *kernel*. Thus the surface is star shaped if its kernel is non-empty. For polygonal meshes, testing whether the kernel is non-empty can be performed efficiently using linear programming. However, it is hard to perform the kernel test exactly for non-linear primitives. We perform a conservative test using linear programming and interval arithmetic for many types of implicit surfaces, by extending the approach described by Varadhan *et al.* [VKSM04]. Our algorithm is based on two oracles:



(a) Vase Lion (b) Chisel (c) Knot

Figure 6: *Polygonization of complex MPU models: The figure shows three MPU models:* Vase Lion *with* 200*K points,* Chisel *with* 142*K points and and* Knot *with* 28*K points.*

- **O1:** Given a voxel ϑ and a parameter n, this oracle returns n points inside ϑ, which lie on $\mathscr{I}$. It returns an error code if $\mathscr{I}_\vartheta = \emptyset$. As long as the surface passes through a cell, this oracle can be implemented by generating line segments uniformly inside the cell and look for intersections with the surface. The generation of line segments is done by 3 sets of two-plane parameterization of the rays as described in Levoy *et al.* [LH96]. As the density of samples in each plane is increased, we get more line segments to intersect the surface.
  Computing the intersection of the line segments with the surface reduces to a ray shooting problem. Implicit surfaces like MPU implicits [OBA*03] are defined as the zero set of a convex combination of quadric surfaces. Dealing with quadric surfaces is fairly well studied in the literature. Further, techniques to compute roots of high-degree univariate polynomials have matured over the last decade and many efficient and reliable numerical algorithms exist [Bin96].
- **O2:** Given a point $\mathbf{p}$ on $\mathscr{I}$, this oracle returns the unit normal vector to $\mathscr{I}$ at $\mathbf{p}$. This is basically gradient computation.

Our method initially estimates a point inside the cell that is likely to be a point in the exact kernel (if one exists). Once we have a witness point in the kernel, we use interval arithmetic to determine if the point obtained is indeed a witness to the exact kernel. If the interval arithmetic test is positive, we can guarantee that the surface is star-shaped. Otherwise, we subdivide the voxel until the condition is satisfied.

4.1. Kernel Witness Computation

We start by invoking oracle **O1** to generate a set of points inside the voxel. We use a simple heuristic to decide the number of points. If V is the volume of the voxel, we generate $m = cV^{2/3}$ points (roughly proportional to the surface area) with an appropriate constant c. The value of c depends on the curvature bounds of the surface inside the voxel and the lower bound of the voxel size during subdivision. Let the point set be $\mathscr{Q} = \{\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_m}\}$. We then invoke oracle **O2** to compute the unit normal vector on each of the points obtained from the first step. Let the normal vector at point $\mathbf{q_i}$ be $\mathbf{n_i}$. We define m linear constraints corresponding to halfspaces supported by the tangent planes $((\mathbf{q_i} - \mathbf{x})^T \mathbf{n_i} > 0, \mathbf{x} \in \mathbb{R}^3)$. We also include the six linear constraints defining the faces of the cell. We solve for the feasibility of the linear program to determine a witness.

We choose a point that is roughly in the center of the approximate kernel. To find this point, we augment the linear program by adding a slack variable δ to each of the constraints - $(\mathbf{q_i} - \mathbf{x})^T \mathbf{n_i} + \delta > 0$. We also set the objective function to maximize δ. This heuristic relies on the fact that as m increases, the kernel computed with linear programming converges to the true kernel. It relies on the following fact: Consider an ε-sampling of the implicit surface using the isophotic metric (Euclidean distance + arc length on Gaussian sphere) [PSH*04]. Then the kernel boundary computed using the samples will approximate the true kernel boundary (in terms of Hausdorff error) to within a monotonic function of ε.

It can be seen that the kernel monotonically shrinks (in terms of set containment) with increasing number of sampled points. The point most interior in the kernel is also the most stable to perturbations in the linear constraints. Hence, if the true surface is star-shaped, this point is likely to be (heuristically) the best candidate for being a guard. Let the point computed using linear programming be $\mathbf{p}$. We need to verify if $\mathbf{p}$ is actually a witness to the exact kernel. Such a witness would satisfy $0 \notin (\mathbf{x} - \mathbf{p})^T \mathbf{n}(\mathbf{x})$, for all points $\mathbf{x}$ on the primitive inside the voxel where $\mathbf{n}(\mathbf{x})$ is the normal to the surface at $\mathbf{x}$.

4.2. Application to Implicit Surfaces

The algorithm described above is applicable to all surfaces, as long as the oracles are available. As a result, our approach is general. However, if surfaces have a certain structure in terms of their description, we take advantage of that to provide more efficient interval tests. Many implicits are represented as zero set of an *elementary function*, i.e. built from a finite combination of constant functions, field operations algebraic, exponential, and logarithmic functions and their inverses under repeated compositions. For such surfaces, the normal vector at a point can be defined by the gradient function which are el-

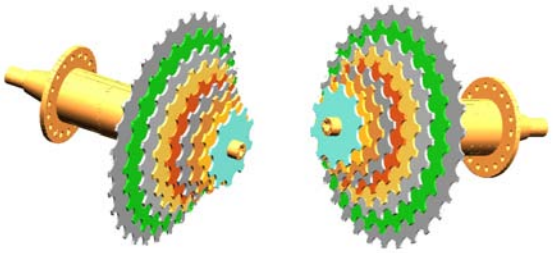


Figure 7: CAD model: This CAD model consists of 14 solids designed using a total of 84 Boolean operations on cones and cylinders. Our algorithm took on an average of 15 secs to compute the boundary of each solid.

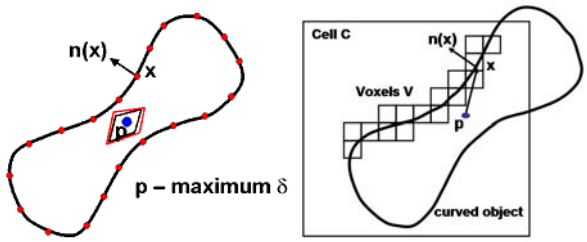


Figure 8: *Star-shaped test on implicit surfaces: The left figure shows the candidate point selection. The red points are a result of implementing oracle **O1**. The right figure shows the kernel membership test. Given a cell C, we compute an initial voxelization V of the primitive and use the intervals in V during the interval arithmetic step.*

ementary functions themselves. This allows us to define our star-shaped condition.

MPU Implicits: Many surface reconstruction algorithms use MPU implicits to define an approximate implicit surface for a given set of points with normals [OBA*03]. Inside localized regions of space, the set of points inside an octree voxel define a local shape function (a quadric function). The MPU implicit is defined as

$$\mathscr{I}(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x}) Q_i(\mathbf{x})}{\sum_i w_i(\mathbf{x})}, \tag{1}$$

where the weight functions $w_i(\mathbf{x})$ are defined as quadratic B-splines for each octree voxel and $Q_i(\mathbf{x})$ is a local quadric surface based on the points inside the voxel. Based on this definition, we express the gradient function of $f(\mathbf{x})$ as

$$\nabla\mathscr{I}(\mathbf{x}) = \frac{\sum_i (w_i(\mathbf{x})\nabla\mathbf{Q_i}(\mathbf{x}) + Q_i(\mathbf{x})\nabla\mathbf{w_i}(\mathbf{x}))}{\sum_j w_j(\mathbf{x})} \tag{2}$$

This expression is plugged into the interval test given a candidate kernel point. Since the interval test is a sign test, we eliminate the denominator since it is a positive weight function.

Moving Least Squares (MLS): Moving Least Squares are commonly used to render point-set surfaces and can be used to generate a topology preserving reconstruction of point samples, under certain conditions [Ame04, Kol05].

We use the formulation of Kolluri [Kol05] in the following discussion. Let S be the input point set. Let the unit normal at point $\mathbf{s_i} \in S$ be $\mathbf{n_i}$. According to Kolluri [Kol05], the MLS is the zero set of

$$\mathscr{I}(x) = \frac{\sum_{\mathbf{s_i}\in S} W_i(\mathbf{x})((\mathbf{x}-\mathbf{s_i})^T\mathbf{n_i})}{\sum_{\mathbf{s_j}\in S} W_j(\mathbf{x})}, \tag{3}$$

where $W_i(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{s_i}\|^2/\varepsilon^2}/A_i$. A_i is the number of samples inside a ball of radius ε centered at $\mathbf{s_i}$.

The implicit surface, defined above, is star-shaped with respect to a point $\mathbf{p}$ if $(\mathbf{x}-\mathbf{p})^T\nabla\mathbf{I}(\mathbf{x}) > 0$. Let the cardinality of set S be k. Let N be the $3\times k$ matrix whose i^{th} column is $\mathbf{n_i}$, W be the $k\times 1$ vector whose i^{th} element is $W_i(\mathbf{x})$ and S be the $k\times 3$ matrix whose i^{th} row is $(\mathbf{x}-\mathbf{s_i})^T$. Then the star-shaped test boils down to evaluating the interval expression

$$tr(NW(\mathbf{x}-\mathbf{p})^T) - 2tr(S^TNW(\mathbf{x}-\mathbf{p})^TS)/\varepsilon^2, \tag{4}$$

$tr()$ is the matrix trace function. The first term of the expression is a 3×3 matrix and the 2nd term is a $k\times k$ matrix.

4.3. Interval Arithmetic and Termination

The main advantage of using interval arithmetic is that all our computations are validated and once it terminates we can be sure that the surface is divided into star-shaped pieces. In this section, we will discuss issues of termination with our interval arithmetic approach. Our subdivision algorithm terminates when one of the following interval conditions are satisfied:

1. The star-shaped condition is satisfied, i.e. $0 \notin (\mathbf{x}-\mathbf{p})^T\nabla\mathscr{I}(\mathbf{x})$
2. The cell does not contain the implicit surface, i.e. $0 \notin \mathscr{I}(x)$

For manifolds, cells far away from the surface will satisfy condition 2 and we terminate the subdivision. For smooth surfaces (C^1 or C^2), the gradient function will be away from zero sufficiently close to the surface and hence condition 1 will be satisfied after finite number of subdivisions.

There are two conditions under which our interval test, as described, will not be satisfied.

- If the candidate kernel witness point $\mathbf{p}$ lies inside the cell, $(\mathbf{x}-\mathbf{p})$ vanishes.
- If the medial axis of the surface lies inside the cell, the gradient of the implicit surface vanishes.

We alleviate these problems by precomputing a dense voxelization of the implicit surface as shown in Fig. 8(b). The motivation to perform the voxelization is to generate a set of intervals that are as close to the implicit surface as possible. While computing the kernel witness point, we ensure that it lies outside the intervals in the voxelization. Given a grid cell, we find the intervals that intersect the cell and perform the interval test on each of the intervals. If the test on each of the intervals does not contain zero, the candidate point is a valid witness to the kernel and we terminate the subdivision. For smooth surfaces, the medial axis is separated from the surface (local feature size is strictly positive) and the sufficiently

dense voxelization will not include the medial axis inside the intervals.

The only case that is left is when the cell contains a part of the surface that has a singularity (condition 1 above is not satisfied because gradient function is zero on the surface). In this case, interval arithmetic will not terminate. In our implementation, once the cells are roughly the size of the intervals we stop the subdivision process and resort only to linear programming to test for the star-shaped property. While this does not provide a rigorous guarantee, this heuristic will work unless we are dealing with highly pathological cases.

The performance of interval arithmetic depends on the number of false negatives which results in unnecessary subdivision. The precomputed voxelization serves the purpose of significantly improving the performance.

4.4. Boolean Combinations

We use the star-shaped property to perform the test on Boolean combinations of primitives. We use a conservative test based on the following property: if $\mathscr{S}_1$ and $\mathscr{S}_2$ are two star-shaped primitives with a common guard, then $\mathscr{S}_1 \odot \mathscr{S}_2$ is also star-shaped where $\odot$ denotes a Boolean operation such as union and intersection. This is because

$$Kernel(\mathscr{S}_1) \cap Kernel(\mathscr{S}_2) \subseteq Kernel(\mathscr{S}_1 \odot \mathscr{S}_2)$$

The above test reduces the star-shaped test for a Boolean combination to star-shaped tests on the individual primitives. We can test whether the individual primitives are star-shaped w.r.t a common guard by using linear programming and interval arithmetic techniques as discussed earlier in the section. There are two important advantages of the above test. First, this test can be performed without an explicit representation of the final solid corresponding to the Boolean operation. Second, this test does not require the final surface to be smooth; this is crucial because Boolean operations typically generate non-smooth surfaces with sharp features. Fig. 7 shows application of our algorithm to Boolean operations.

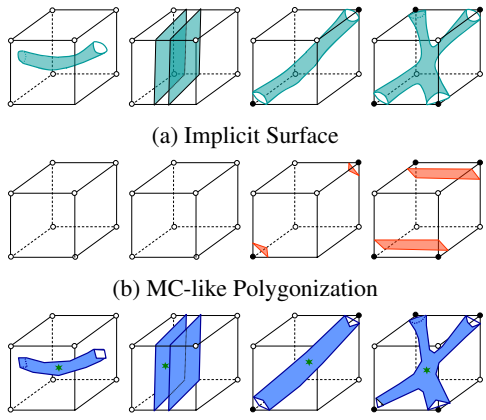
5. Implementation and Performance

Our algorithm uses three basic components: (a) linear programming to verify if a a polyhedral primitive or a set of points samples satisfy the star-shaped test, (b) interval arithmetic to check whether a curved primitive intersects a cell and if it satisfies the star-shaped test, and (c) ray shooting to evaluate the forward and backward visibility maps, image transfer map, and the homeomorphism. Each of these three components are relatively simple to implement. Moreover, there exist public domain packages for many of these components, e.g., *QSOPT* for linear programming.

We have tested our algorithms on three kinds of benchmarks: MPU-based reconstruction [OBA*03] of point cloud, polygonization of algebraic surfaces, and CSG models. Figs. 1, 5, 6, and 7 highlight our results on these benchmarks. Table 1 shows a performance comparison with Bloomenthal's algorithm [Blo88] and the algorithm by Boissonnat & Oudot [BO05]. While our algorithm is somewhat slower than Bloomenthal's algorithm, it has the advantage that it provides guarantees on the accuracy of the output. Compared to Boissonnat & Oudot's algorithm, which also provides guarantees, our algorithm is significantly faster and more scalable to complex models.

Model	Bloomenthal	Boissonnat [BO05]	Our Method
Filigree	380	519	241
Chisel	47	79	58
Vase-lion	205	13,582	234
Chair	1.0	?	10.6
Tanglecube	0.35	17.6	11
Decocube	1.89	?	52
Diagonal	0.93	9.0	9.1

Table 1: Performance comparison: This table compares the performance of our algorithm with Bloomenthal's algorithm and the algorithm by Boissonnat & Oudot for various models. All the timings are in seconds. The question mark indicates cases for which we were not able to successfully run the algorithm.



(a) Implicit Surface

(b) MC-like Polygonization

(c) Visibility Mapping Based Polygonization

Figure 9: This figure compares our algorithm with MC-like algorithms for a set of cell configurations. The top row of figures show different cell configurations; the middle row shows polygonization obtained using an MC-like algorithm; the bottom row shows polygonization produced by our algorithm. In some cases, MC-like algorithms generate no surface output.

6. Comparison with Prior Methods

In this section, we compare and contrast features of our algorithm with prior techniques to polygonize implicit surfaces.

6.1. Isosurface Extraction Methods

One of the commonly used approaches for isosurface extraction is Marching Cubes (MC) and its extensions [LC87, KBSS01, JLSW02]. We refer to these algorithms collectively as MC-like algorithms. The input to MC-like algorithms is a sampled scalar field obtained by discretizing the continuous implicit function upto a certain resolution. However, the main issue in these approaches is generating the scalar field at an appropriate resolution such that the reconstructed surface is accurate. An inadequate grid resolution may result in an inaccurate output when the implicit surface

has *small components*, *thin sheets*, or *needle-shaped features*, as shown in Fig. 9.

Assume that the subdivision algorithm has computed a guard within each cell in Fig. 9. It is possible that the subdivision algorithm may need to generate sub-intervals within the cell to reliably compute the guard. Once such a guard has been computed, our polygonization algorithm can reliably handle all these cases as shown in Fig. 9.

Topological ambiguity: Many solutions have been proposed to fix topological ambiguities that can arise in some cases of the original MC algorithm [WG90, CGMS00]. In some case, these algorithms can provide topological guarantees by assuming some type of interpolation, e.g., trilinar interpolation, on the sampled scalar field. These topological ambiguity handling algorithms do not address the issues of generating a grid with adequate resolution to provide global topological guarantees with respect to the original implicit surface.

Topology preserving sampling: Varadhan et al. [VKSM04] presented an adaptive grid generation algorithm such that the final surface reconstructed using MC has the same topology as the implicit surface. However, this algorithm is rather conservative and uses much stronger criteria for sample generation. Moreover, our polygonization algorithm can explicitly compute the homeomorphism and compute a parameterization of the implicit surface.

6.2. Spatial Subdivision Methods

Spatial subdivision techniques have been widely used to polygonize implicit surfaces [Blo88, Vel90, HW90, SFYC96, JLSW02, WGG99, AG01]. These algorithms either generate a uniform grid, octree or a kd-tree, or perform tetrahedral-based polygonization. These algorithms can also handle non-manifold implicit surfaces [BF95]. However, to the best of our knowledge, these spatial subdivision algorithms do not provide topological guarantees on the final polygonization. Few algorithms such as [KB89] choose to visualize implicit surfaces by ray-tracing rather than polygonization, but these are complementary to our work.

6.3. Topology Preserving Polygonization

Some algorithms can guarantee a topology preserving polygonization of implicit surfaces [BNO96, SH97, BCSV04, vOW04]. These algorithms assumes a C^2 continuous implicit surface. Under this assumption, these algorithms attempt to isolate the critical points. They do not apply to Boolean operations wherein the critical points corresponding to the final solid are unknown.

We have demonstrated the application of the meshing algorithm described in [BO05] in Table 1. It can only handle smooth surfaces and can not deal with Boolean operations. It is significantly slower than our algorithm and can not deal with very complex inputs.

6.4. Interval Arithmetic

Synder [Sny92] presented an adaptive subdivision method for computing a topologically accurate polygonal approximation of an implicit curve/surface. This method keeps subdividing cells until a parameterizability criterion is satisfied This parameterizability criterion would not be satisfied if the implicit surface has a sharp feature, e.g., in the case of a surface derived from a Boolean operation. Furthermore, the computational techniques presented in Snyder's paper to verify the parameterizability criterion are not applicable to Boolean operations.

Plantinga and Vegter [PV04] present a similar method for implicit surface polygonization. The condition of Plantinga and Vegter imposes the constraint that the gradient of the implicit function should not vary by more than $\pi/2$. As a result, it is applicable to only smooth surfaces and cannot handle objects defined by Boolean operations.

7. Limitations

Our approach has a few limitations. We assume that the implicit surface is a manifold. Our interval arithmetic based star-shaped decomposition algorithm can be conservative in practice. Furthermore, the star-shaped test can generate a high number of sub-intervals in order to compute a guard inside a voxel. Our algorithm cannot handle cases where the implicit surface has singularities such as self-intersections or tangential intersections.

8. Conclusion and Future Work

We present a new algorithm to polygonizing implicit surfaces based on star-shaped decomposition. We compute a visibility mapping for each star-shaped patch within the cell and use these maps to compute a polygonization that is homeomorphic to the original surface. Our algorithm is relatively simple to implement and not prone to robustness problems or crack patching that arise in other techniques based on adaptive subdivision. We demonstrate the performance of our algorithm on many complex benchmarks, including non-smooth surfaces, and compute a watertight polygonization. In terms of performance, our algorithm is considerably faster than previous methods that can provide rigorous guarantees on the topology and can handle complex models. However, our current implementation is not fast enough for interactive applications.

There are many avenues for future work. We would like to extend our approach to handle non-manifold surfaces. We would like to compute a global parameterization of implicit surface that can give bounds on distortion. It may be useful to develop point-reconstruction algorithms for non-smooth surfaces based on the MLS formulation [Kol05] and generate a topology preserving polygonization using our approach. Another challenge is to faithfully reconstruct all the sharp features in the input. Finally, we would like to improve the performance of our algorithm for real-time implicit modeling.

Acknowledgements

We would like to acknowledge SensAble Technologies Inc. for providing the Filigree and Vase-lion models (`http://shapes.aim-at-shape.net`).

References

[AAB02] ANDUJAR C., AYALA D., BRUNET P.: Topology simplification trough discrete models. pp. 88–105.

[AG01] AKKOUCHE S., GALIN E.: Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum 20*, 2 (2001), 67 –80. ISSN 1067-7055.

[Ame04] AMENTA N.: Defining point-set surfaces. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 23* (2004), 264–270.

[BCSV04] BOISSONNAT J., COHEN-STEINER D., VEGTER G.: Isotopic implicit surface meshing. *ACM Symposium on Theory of Computing* (2004), 301–309.

[BF95] BLOOMENTHAL J., FERGUSON K.: Polygonization of Non-Manifold implicit surfaces. In *SIGGRAPH 95 Conference Proceedings* (1995), pp. 309–316.

[Bin96] BINI D.: Numerical computation of polynomial zeros by means of aberth's method. *Numerical Mathematics 13* (1996), 179–200.

[Blo88] BLOOMENTHAL J.: Polygonization of implicit surfaces. *Comput. Aided Geom. Design 5*, 4 (1988), 341–355.

[BNO96] BOTTION A., NUIJ W., OVERVELD K. V.: How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of surfaces with arbitrary topology. *Implicit Surfaces* (1996), 53–72.

[BO05] BOISSONNAT J., OUDOT S.: Provably good sampling and meshing of surfaces. *Graphical Models* (2005).

[CGMS00] CIGNONI P., GANOVELLI F., MONTANI C., SCOPIGNO R.: Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics 24*, 3 (2000), 399–418.

[FH05] FLOATER M., HORMANN K.: *Surface parameterization:: A tutorial and survey*. Tech. rep., 2005.

[HW90] HALL M., WARREN J.: Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics and Applications 10*, 6 (Nov. 1990), 33–42.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Trans. on Graphics (Proc. SIGGRAPH) 21*, 3 (2002).

[KB89] KALRA D., BARR A. H.: Guaranteed ray intersections with implicit surfaces. In *Computer Graphics (SIGGRAPH '89 Proceedings)* (1989), vol. 23, pp. 297–306.

[KBSS01] KOBBELT L., BOTSCH M., SCHWANECKE U., SEIDEL H. P.: Feature-sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH* (2001), pp. 57–66.

[Kol05] KOLLURI R.: Provably good moving least squares. *ACM-SIAM Symposium on Discrete Algorithms* (2005).

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (1987), vol. 21, pp. 163–169.

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM Press, pp. 31–42.

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.: Multi-level partition of unity implicits. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 22* (2003), 463–470.

[Ped95] PEDERSEN H. K.: Decorating implicit surfaces. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), Cook R., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 291–300. held in Los Angeles, California, 06-11 August 1995.

[PH03] PRAUN E., HOPPE H.: Spherical parameterization and remeshing. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 22* (2003).

[PSH*04] POTTMANN H., STEINER T., HOFER M., HAIDER C., HANBURY A.: The isophotic metric and its application to feature sensitive morphology on surfaces. 560–572.

[PV04] PLANTINGA S., VEGTER G.: Isotopic approximation of implicit curves and surfaces. *Symposium on Geometry Processing* (2004).

[SFYC96] SHEKHAR R., FAYYAD E., YAGEL R., CORNHILL F.: Octree-based decimation of marching cubes surfaces. *Proc. of IEEE Visualization* (1996), 335–342.

[SH97] STANDER B. T., HART J. C.: Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proc. of ACM SIGGRAPH* (1997), pp. 279–286.

[Sny92] SNYDER J. M.: Interval analysis for computer graphics. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), Catmull E. E., (Ed.), vol. 26, pp. 121–130.

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. *ACM Trans. Graph. 23*, 3 (2004), 853–860.

[Vel90] VELHO L.: Adaptive polygonization of implicit surfaces using simplicial decomposition and boundary constraints. In *Eurographics '90* (Sept. 1990), Vandoni C. E., Duce D. A., (Eds.), North-Holland, pp. 125–136.

[VKSM04] VARADHAN G., KRISHNAN S., SRIRAM T. V. N., MANOCHA D.: Topology preserving surface extraction using adaptive subdivision. In *Eurographics Symposium on Geometry Processing* (2004).

[VKZM06] VARADHAN G., KRISHNAN S., ZHANG L., MANOCHA D.: Reliable implicit surface polygonization using visibility mapping. In *http://gamma.cs.unc.edu/VM/supplem.pdf* (2006).

[vOW04] VAN OVERVELD K., WYVILL B.: Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface . *The Visual Computer 20*, 6 (2004), 362– 369.

[WG90] WILHELMS J., GELDER A. V.: Topological considerations in isosurface generation extended abstract. *Computer Graphics 24*, 5 (1990), 79–86.

[WGG99] WYVILL B., GALIN E., GUY A.: Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum 18*, 2 (1999), 149–158.

[WMW86] WYVILL B., MCPHEETERS C., WYVILL G.: Data structure for soft objects. *The Visual Computer 2*, 4 (1986), 227–234.