Topology Preserving Isosurface Extraction for Geometry Processing

Gokul Varadhan[1], Shankar Krishnan[2], TVN Sriram[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/recons

ABSTRACT

We address the problem of computing a topology preserving isosurface from a volumetric grid using Marching Cubes for geometry processing applications. We present a novel adaptive subdivision algorithm to generate a volumetric grid. Our algorithm ensures that every grid cell satisfies certain sampling criteria. We show that these sampling criteria are sufficient to ensure that the isosurface extracted from the grid using Marching Cubes is topologically equivalent to the exact isosurface: both the exact isosurface and the extracted isosurface have the same genus and connectivity. We use our algorithm for accurate boundary evaluation of Boolean combinations of polyhedra and low degree algebraic primitives, Minkowski sum computation, model simplification, and remeshing. The running time of our algorithm varies between a few seconds for simple models composed of a few thousand triangles and tens of seconds for complex polyhedral models represented using hundreds of thousands of triangles.

1. INTRODUCTION

Implicit surface representations have become increasingly common in computer graphics and geometric modeling. This representation uses a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to represent a closed surface $\mathcal{S}$. The function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is known as the *implicit function* or the *scalar field*. $\mathcal{S}$ is the set of points $\boldsymbol{p}$ where $f(\boldsymbol{p}) = 0$; $\mathcal{S}$ is referred to as an *implicit surface* or an *isosurface*. A commonly used scalar field is the *signed distance field*. The signed distance field $D : \mathbb{R}^d \rightarrow \mathbb{R}$ is a continuous function that at a point $\boldsymbol{p}$ measures the distance between $\boldsymbol{p}$ and $\mathcal{S}$. This value is positive or negative depending on whether the point lies outside or inside $\mathcal{S}$. The distance can be defined under any reasonable norm (e.g., Euclidean, max-norm).

A common way of representing a scalar field is to discretize the continuous scalar field into discrete samples – to compute the value of the scalar field at the vertices of a volumetric grid. We refer to this step as a *sampling* of the scalar field. The grid is an approximate representation of the scalar field; the accuracy of the approximate representation depends on the *rate of sampling* – the resolution of the grid. An explicit boundary representation of the implicit surface can be obtained by extracting the zero-level isosurface using Marching Cubes (MC) [2] or any of its variants [3–5]. We refer to these isosurface extraction algorithms collectively as *MC-like* algorithms. The output of an MC-like algorithm is an approximation – usually a polygonal approximation – of the implicit surface. We refer to this step as *reconstruction* of the implicit surface.

[3]A preliminary version of this work appeared in the second European Symposium on Geometry Processing 2004 [1].

Compared to other surface representations (e.g. parametric surfaces), implicit surface representations are easy to use to perform geometric operations like union, intersection, difference, blending, and warping. Specifically, they map Boolean operations into simple minimum/maximum operations on the scalar fields of the primitives. Suppose we have two primitives $\mathcal{P}_1$ and $\mathcal{P}_2$ with scalar fields f_1 and f_2. Then we have

$$\begin{aligned}
\boldsymbol{p} \in \partial(\mathcal{P}_1 \cup \mathcal{P}_2) &\iff \min(f_1(\boldsymbol{p}), f_2(\boldsymbol{p})) = 0 \\
\boldsymbol{p} \in \partial(\mathcal{P}_1 \cap \mathcal{P}_2) &\iff \max(f_1(\boldsymbol{p}), f_2(\boldsymbol{p})) = 0 \\
\boldsymbol{p} \in \partial(\mathcal{P}_1 \setminus \mathcal{P}_2) &\iff \max(f_1(\boldsymbol{p}), -f_2(\boldsymbol{p})) = 0
\end{aligned}$$

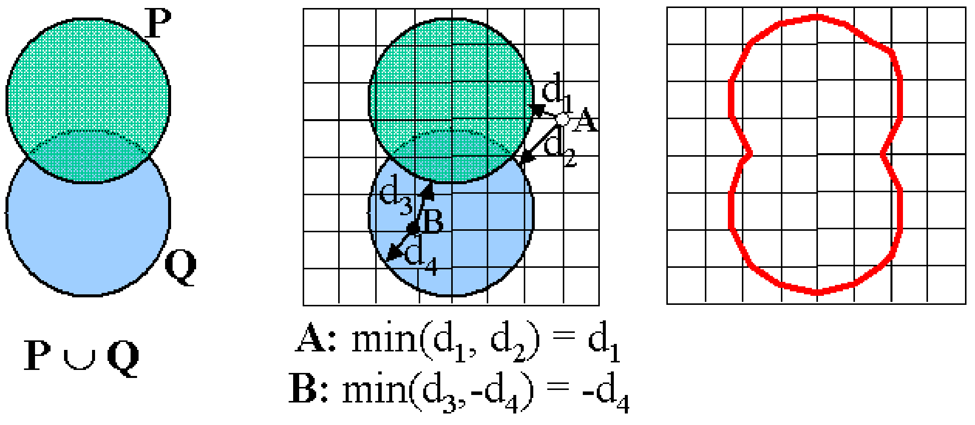
where $\partial\mathcal{P}$ denotes the boundary of $\mathcal{P}$. Because of the above property, implicit representations are frequently used to perform Boolean operations. They have been used for numerous applications including geometric modeling, volume rendering, morphing, path planning, swept volume computation, and sculpting digital characters [3, 4, 6–11].

Our goal is to exploit the desirable properties of implicit surface representations for geometric computations such as Boolean operations (i.e. union, intersection and difference), Minkowski sum computation, simplification, and remeshing. In each case, we wish to obtain an accurate polygonal approximation of the boundary of the final solid. Let $\mathcal{E}$ denote this boundary. We represent $\mathcal{E}$ implicitly – as an isosurface of a scalar field. This scalar field is obtained by performing minimum/maximum operations over the scalar fields defined for the primitives. At a broad level, our approach performs three main steps.

1. **Sampling:** Generate a volumetric grid and compute a scalar field (e.g, a signed distance field) at its corner grid points.

2. **Operation:** For each geometric operation (union or intersection), perform an analogous operation (e.g., min/max) on the scalar fields of the primitives. At the end of this step, the scalar values at the grid points define a sampled scalar field for $\mathcal{E}$.

3. **Reconstruction:** Perform isosurface extraction using



(a) Union (b) Sampling and Operation (c) Reconstruction

Figure 1: Sampling and reconstruction: ***This figure shows how to perform a union operation using the sampling and and reconstruction approach. The sampling step generates a volumetric grid (shown as a uniform grid). At each grid point, it computes a signed distance to the boundaries of each of the two primitives,* P *and* Q*. Next, a minimum operation is performed on the two signed distances. This generates another distance field on which the reconstruction is performed using an MC-like algorithm. The rightmost figure shows the reconstruction, which is an approximation to the union.***

an MC-like algorithm to obtain a polygonal approximation $\mathcal{A}$ of $\mathcal{E}$.

This approach is illustrated in Fig. 1. We will refer to $\mathcal{E}$ as the *exact isosurface* and $\mathcal{A}$ as the *extracted isosurface* or the *reconstructed isosurface*.

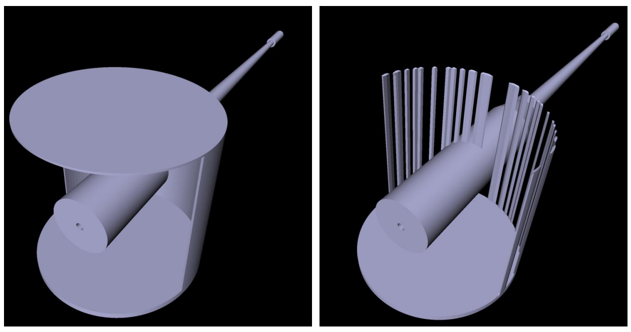


Figure 2: Accuracy problems with MC-like methods: ***This figure highlights the errors that can be present in the output when MC-like methods are used to reconstruct surfaces with thin features. The left image shows the "gun model" of the Bradley Fighting Vehicle, which is generated using 8 Boolean operations. The right image shows the output of a MC-like algorithm (dual contouring) on a distance field sampled on a uniform*** $64 \times 64 \times 64$ ***grid. The output has many artifacts such as unwanted holes and extraneous handles.***

Two important advantages of the above approach are simplicity and efficiency. Each step is easy to implement. A uniform grid or an adaptive grid (e.g. octree) may be chosen. Geometric operations such as union or intersection are cheap – we only need to perform simple min/max operations on the corresponding scalar fields. Isosurface extraction is also reasonably straightforward: MC-like algorithms are both simple and fast. Many public domain implementations of MC-like algorithms [12] are available.

Challenges/Issues: The above approach produces an approximation $\mathcal{A}$ to the final surface $\mathcal{E}$. The accuracy of the approximation mainly depends on the resolution of the underlying grid. Insufficient grid resolution can result in a poor approximation. This occurs especially if $\mathcal{E}$ has *small components*, *thin sheets*, or *needle-shaped features*. $\mathcal{A}$ may suffer from various kinds of errors: small components or handles present in $\mathcal{E}$ may not be captured in $\mathcal{A}$. The process of reconstruction may also introduce "extraneous topology", i.e., $\mathcal{A}$ may have unwanted additional components or undesirable handles that were not present in $\mathcal{E}$. Fig. 2 shows an example of such a situation.

The above problems occur on account of inadequate resolution of the grid. Therefore, to alleviate these problems, many applications generate samples on a fine grid. However, the use of fine grid can result in three problems. First, there may still be no guarantees on the accuracy of $\mathcal{A}$. Second, a fine grid increases the storage overhead and the reconstructed surface can have a high number of polygonal primitives. Finally, it is computationally expensive to use a fine grid. Recent work on adaptive grid generation and subdivision algorithms overcomes some of these problems [10, 13]. However, none of these algorithms give rigorous guarantees on the accuracy of $\mathcal{A}$.

Goals: Our goal is to ensure an accurate approximation by providing geometric and topological guarantees on $\mathcal{A}$. In particular, we wish to ensure that $\mathcal{A}$ has a bounded two-sided *Hausdorff error*, and is topologically equivalent to $\mathcal{E}$. The Hausdorff error measures the surface deviation between $\mathcal{A}$ and $\mathcal{E}$; bounding this error is important to ensure a geometrically close approximation.

Preserving topology is also important in many applications. In CAD, topological features such as tunnels often correspond to distinguishing characteristics of the model. The geometric models used to represent the organs in medical datasets often consist of handles. Retaining these topological features can be necessary in order to preserve the anatomical structure of the organ, which can be crucial for visualization and analysis. Apart from capturing important features present in $\mathcal{E}$, guaranteeing topology is important for another reason. An algorithm that preserves topology avoids the introduction of extraneous topology; its output does not have unwanted additional components or handles, and is immune from the kinds of errors shown in Fig. 2.

Main Contributions: We present a novel approach to compute a topology preserving isosurface using an MC-like algorithm for geometry processing applications. We present conservative sampling criteria such that if every cell in the volumetric grid satisfies the criteria, then the extracted isosurface will have the same topology as the exact isosurface. We present an adaptive subdivision algorithm to generate a volumetric grid such that every grid cell satisfies the sampling criteria. We present efficient computational techniques to verify the sampling criteria during grid generation. Our algorithm can easily perform these computations on polyhedra, algebraic or parametric primitives and their Boolean combinations. Furthermore, we extend the adaptive subdivision algorithm to also bound the Hausdorff distance between the exact isosurface and the extracted isosurface. This ensures that the extracted isosurface is geometrically close to the exact isosurface.

We have used our algorithm to perform accurate boundary evaluation of Boolean combinations of polyhedral and low degree algebraic primitives, model simplification, and remeshing of complex models. In each case, we compute a topology preserving polygonal approximation of the boundary of the final solid. The running time of our algorithm varies between a few seconds for simple models consisting of thousands of triangles and tens of seconds on complex primitives represented using hundreds of thousands of triangles on a Pentium IV PC.

Some *novel aspects* of our work include:

- Conservative sampling criteria for the volumetric grid such that the topology of the isosurface is preserved.
- An efficient adaptive subdivision algorithm to generate an octree satisfying the sampling criteria.
- Efficient and accurate algorithms for boundary evaluation of solids defined by Boolean operations.
- A fast algorithm to compute topology preserving simplification and remeshing of a complex polygonal model.

As compared to prior work, the main benefits of our algorithm are its speed, simplicity, and accuracy. It not only offers the simplicity and efficiency of MC-like reconstruction, but also the ability to guarantee the topology of the extracted isosurface.

Organization: The rest of our paper is organized as follows. Section 2 gives a brief overview of previous work on isosurface extraction. Section 3 introduces the notation and definitions used in the rest of the paper. Section 4 presents an overview of the MC-like methods and analyzes the errors in their output that can be caused due to inadequate resolution of the grid. In Section 5, we present a sampling condition on the volumetric grid to ensure topology preservation. In Section 6, we present an adaptive subdivision algorithm to generate a volumetric grid satisfying the sampling condition. In Section 7, we present a simple technique to guarantee a tight geometric error bound on the approximation. Section 8 disusses a few issues that arise when performing isosurface extraction on adaptive grids. In Section 9, we present techniques to improve the performance of the algorithm. Section 10 discusses the performance of the algorithm. Section 11 describes the implementation of the algorithm, and presents three different applications: Boolean operations, simplification, and remeshing. Section 12 analyzes the behavior of the adaptive subdivision algorithm and presents conditions for its termination. Section 13 discusses degenerate cases for our algorithm. Section 14 discuss limitations of our approach. Sections 15 and 16 conclude the paper and present directions for future work.

2. PRIOR WORK ON ISOSURFACE EXTRACTION

Given a continuous scalar field $f : \mathbb{R}^d \to \mathbb{R}$ and a scalar value s, the *isosurface* with *isovalue* s is the set, $\{\boldsymbol{x} \in \mathbb{R}^d \mid f(\boldsymbol{x} = s\}$, of points with identical scalar value s. Isosurface extraction refers to the process of constructing a (usually piecewise linear) approximation to the isosurface. In this section, we focus only on isosurface extraction in $\mathbb{R}^3$. Even this topic has been extensively studied, and we do not attempt to survey the entire literature on this topic. We refer the reader to [14, 15] for additional pointers.

The problem of isosurface extraction originated in two disparate domains – volumetric visualization and implicit modeling. We start by giving a brief overview of isosurface extraction methods in volumetric visualization and implicit modeling (Sections 2.1 and 2.2). Next we describe Marching Cubes [2] – the most popular method for isosurface extraction – and many of its variants (Section 2.3). Then we discuss various topological considerations during isosurface extraction (Section 2.4). We discuss a number of isosurface extraction methods that provide some form of topological guarantees on their output.

2.1 Volumetric Visualization

In volumetric visualization, isosurface extraction was used as a tool to visualize volumetric datasets acquired experimentally, e.g., medical datasets obtained using magentic resonance imaging (MRI), Computed Tomography (CT) [2,16–19]. Isosurface extraction was used to visualize the boundary of some important feature – typically, an anatomical organ.

The boundary corresponds to a region of constant value in the volumetric data. Volumetric datasets are also generated during scientific simulations in computational computational fluid dynamics (CFD) [20] and molecular dynamics [21]. Isosurface extraction was used to expose contours of constant value for understanding the structure of the scalar field. These contours isolate surfaces of interest, focusing attention on important features in the data such as material boundaries and shock waves.

2.2 Implicit Modeling

Implicit modeling became popular because of the simplicity and versatility of implicit representations in performing a wide variety of geometric operations. Many operations such as Boolean operations, offsetting, blending, warping, and sweep can be expressed elegantly using implicit representations [6,8,22–25]. Because of these advantages, implicit modeling techniques have been used by a large number of applications including geometric modeling [3, 4, 9–11], volume rendering [26], surface reconstruction [27–29], remeshing [3, 30], swept volume computation [31], animation [6], and sculpting digital characters [10]. Despite the above advantages of the implicit representation, many applications such as graphical rendering, collision detection, dynamic simulation, and model verification use an explicit representation such as a polygonal mesh representation. Isosurface extraction techniques are used to convert implicit surfaces to an "explicit form", i.e., a polygonal mesh approximating the implicit surface. Wyvill et al. [6] developed one of the early isosurface extraction methods for polygonizing implicit surfaces. Bloomenthal [7] developed another method that adaptively sampled the implicit function by surrounding the implicit surface by an octree. Subsequently, a large number of methods were developed for polygonizing implicit surfaces [3–5, 10, 32–41].

2.3 Marching Cubes

The Marching Cubes algorithm (MC), proposed by Lorensen and Cline [2], is a standard method to extract an isosurface from a volumetric dataset with scalar values. It performs reconstruction by extracting surfaces separately in every cell in a volumetric cubic grid. The algorithm iterates through all cells in the grid, hence the term marching cubes. Since each of the 8 vertices of the cubic cell can be either positive or negative, there are $2^8 = 256$ possible sign configurations. Lorenson & Cline used symmetries between different sign configurations to reduce them to 15 basic cases. They stored each of these cases in a look-up table, and used it to find the polygonal approximation of the isosurface in a given cell.

The original Marching Cubes algorithm examined all cells in the data set even though typically the isosurface intersects only a small subset of the cells. Wilhelms and Gelder [42] estimated that MC spent between 30% and 70% of the total time examining empty cells that do not intersect the isosurface. A tremendous amount of research has focused on reducing the number of cells visited while constructing an isosurface [42–45].

Another drawback of the original MC algorithm was that it generated an excessively large number of triangles to represent the isosurface. To overcome this drawback, many methods were developed for performing isosurface extraction adaptively using hierarchies such as octrees or k-D trees [4, 7, 13, 32, 33, 46–48].

A large number of variants of MC have also been developed that suggest alternative ways of reconstructing the isosurface within the cell [3–5, 10, 38, 40, 41, 49].

Implicit surfaces defined in terms of Boolean operations usually have sharp edges or corners. When MC is used for polygonizing such implicit surfaces, the output usually has aliasing artifacts in the vicinity of the sharp features. Recently, a few extensions have been proposed that can reconstruct sharp features and reduce aliasing artifacts in the reconstructed model [3–5, 37].

2.4 Topological Considerations in Isosurface Extraction

2.4.1 Topological Ambiguity

In the original Marching Cubes algorithm, some of the base cases were ambiguous. Given a face of a cube with two diagonally opposite corners above the surface, and the other two below the surface, some of the basic cases assumed that the higher corners fell inside the same connected component of the surface, while others assumed that they did not. Since each face of a cube was shared with an adjacent cube, it was possible to generate surfaces with holes by accident. This was noted by Durst [50], and many solutions were proposed [51–53].

The above algorithms deal with the problem of extracting a surface from a fixed volumetric data set. The continuous scalar field is not available for resampling. Since the scalar values are available only on the vertices of the cell, they model the scalar field in the interior of the cell. For example, the algorithms by [53–55] model the scalar field as a trilinear function that interpolates the scalar values at the vertices of the cell. Then they extract a surface that is topologically equivalent to the isosurface of the trilinear function. However, due to inadequate resolution of the data set, the trilinear function may not be an accurate model of the continuous scalar field. Hence the output of these algorithms need not be topologically equivalent to the isosurface of the continuous scalar field.

2.4.2 Topology Control and Simplification

Many volumetric approaches have used topological properties for generating an isosurface without additional handles or cavities from scanned data sets [30, 56, 57]. Often the input data contains noise due to the scanning process. During isosurface extraction, the inaccuracies in the input data result in a surface whose genus is much higher than the actual surface. In many applications, the topological type of the object under consideration is known beforehand, e.g., the cortex of a human brain is always homeomorphic to a sphere [57]. These methods exploit such a priori knowledge to eliminate unwanted handles that arise from the noise.

2.4.3 Topology Preserving Implicit Surface Polygonization

Few methods based on Morse theory [35, 36, 39] can guarantee a topology preserving polygonization of implicit surfaces. These methods assume that the implicit surface is smooth and require computation of all the critical points of the implicit surface.

Snyder [58] and Plantinga and Vegter [59] have presented adaptive subdivision methods for computing a isotopic approximation of an implicit curve/surface. These methods check whether the implicit curve/surface is locally parametrizable with respect to one of the axes of the cell. If a cell fails the condition, it is subdivided and the algorithm is recursively applied to the subdivided cells.

All the above methods assumes that the implicit surface is smooth. It is not clear how to apply them to perform Boolean combinations where the final surface is not smooth and whose topology can be very different from those of the input primitives.

3. NOTATION AND PRELIMINARIES

Input

We assume that the exact surface $\mathcal{E}$ is obtained by performing Boolean operations (union, intersection, difference, complement) on a set of primitives in $\mathbb{R}^3$. The input to our algorithm is a Boolean expression and a set $\Gamma = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ of primitives.

We assume that each primitive $\mathcal{P}_i$ is a closed 2-manifold, and is either a polyhedral or an algebraic object. We assume each $\mathcal{P}_i$ bounds a solid, which we denote as $\widetilde{\mathcal{P}}_i$. The Boolean operations are performed on the solids $\widetilde{\mathcal{P}}_i$, which yields a final solid $\widetilde{\mathcal{E}}$. The exact surface is the boundary of $\widetilde{\mathcal{E}}$ and is denoted as $\mathcal{E}$. We assume that $\mathcal{E}$ is a closed 2-manifold.

Output

The output of our algorithm is a polygonal approximation $\mathcal{A}$ to the exact surface $\mathcal{E}$.

Notation

We introduce the notation used in the rest of this paper.

- Lower case bold letters such as $\boldsymbol{p}, \boldsymbol{q}$ refer to points in $\mathbb{R}^3$. Upper case letters such as $\mathcal{P}, \mathcal{Q}, \mathcal{P}_1$ refer to geometric primitives. We assume that each primitive is a closed manifold. Each primitive bounds a solid, which we will refer to as the **primitive solid**, and denote it as $\widetilde{P}$.

- Boolean operations are defined on primitive solids. $\widetilde{\mathcal{P}}_1 \cup \widetilde{\mathcal{P}}_2$, $\widetilde{\mathcal{P}}_1 \cap \widetilde{\mathcal{P}}_2$, $\widetilde{\mathcal{P}}_1 \setminus \widetilde{\mathcal{P}}_2$ denote union, intersection, and difference operations on $\widetilde{\mathcal{P}}_1$ and $\widetilde{\mathcal{P}}_2$ respectively.

  Let $\widetilde{\mathcal{S}}$ denote a solid defined by Boolean operations. By a slight abuse of notation, we will use $\widetilde{\mathcal{S}}$ to also refer to the **Boolean expression** of the solid, which is the expression that defines $\widetilde{\mathcal{S}}$ in terms of Boolean operations over a set of primitive solids, e.g., $\widetilde{\mathcal{S}} = \widetilde{\mathcal{P}}_1 \cup \widetilde{\mathcal{P}}_2$. A Boolean expression of a surface $\mathcal{S}$ is defined as the Boolean expression of the corresponding solid $\widetilde{\mathcal{S}}$.

- The abbreviation *w.r.t* means *with respect to.*

- ∂S, $\overline{S}$, int S, and cl S respectively denote the boundary, complement, interior and closure of a set S.

- Let $d \geq 1$ be an integer. Let $\mathbf{o}$ be the origin of $\mathbb{R}^d$. $\mathbb{S}^{d-1}$ and $\mathbb{B}^d$ denote the $(d-1)$-dimensional sphere and d-dimensional ball respectively. They are defined as:

$$\begin{aligned} \mathbb{S}^{d-1} &= \{\mathbf{x} \in \mathbb{R}^d \mid |\mathbf{xo}| = 1\} \\ \mathbb{B}^d &= \{\mathbf{x} \in \mathbb{R}^d \mid |\mathbf{xo}| < 1\} \end{aligned}$$

  Also define the 0-ball $\mathbb{B}^0 = \{\mathbf{o}\}$.

- Our algorithm uses a volumetric grid in $\mathbb{R}^3$. Unless otherwise stated, the grid is assumed to be an octree [60]. The letter C will refer to a single grid cell. 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. A cube-shaped grid cell consists of one voxel, six faces, twelve edges, and eight vertices. All of them are assumed to be closed sets. The symbols ϑ, f, e, and v will refer, respectively, to a voxel, a face, an edge, and a vertex.

  Let c be an edge/face/voxel of a cell C. The **size** of c, denoted as $\|c\|$, is the maximum distance between any two vertices in c. The **size** of C, denoted as $\|C\|$, is the size of its voxel. The **width** of c is the minimum distance between any two vertices in c. The **width** of C is the width of its voxel.

- By a restriction of a set S w.r.t another set T, we mean $S \cap T$, which is denoted as S_T. In particular, we will use the following notation frequently.

  The restrictions of $\mathcal{E}$ w.r.t a cell C, voxel ϑ, a face f, or an edge e are denoted as $\mathcal{E}_C$, $\mathcal{E}_\vartheta$, $\mathcal{E}_f$, and $\mathcal{E}_e$ respectively. Similarly, we can define $\mathcal{A}_C$, $\mathcal{A}_\vartheta$, $\mathcal{A}_f$, and $\mathcal{A}_e$. The restriction w.r.t the cell is defined as the restriction w.r.t the voxel of the cell.

- A **homeomorphism** is a continuous bijective mapping with a continuous inverse [61]. Two objects $\mathcal{P}$ and $\mathcal{Q}$ are **topologically equivalent** if there exists a homeomorphism $\mathcal{H} : \mathcal{P} \rightarrow \mathcal{Q}$. We denote this as $\mathcal{P} \approx \mathcal{Q}$.

  We will call an object $\mathcal{P}$ a **topological disk** if $\mathcal{P} \approx \mathbb{B}^d$ for $d > 0$.

  An object is **d-manifold** if every point has a neighborhood that is topologically equivalent to $\mathbb{R}^d$.

  A manifold is **connected** if for any two points on the manifold, there exists a path between them in the set. If two points $\mathbf{p} \in S$ and $\mathbf{q} \in S$ are connected in a set S, we denote this as $\mathbf{p} \xleftrightarrow{S} \mathbf{q}$.

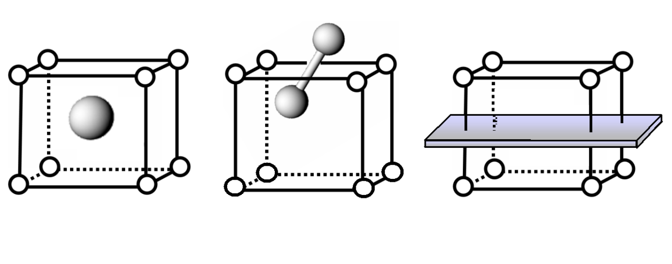
  A manifold is said to be **simply connected** if any simple closed curve on the manifold can be shrunk to a point continuously in the set.

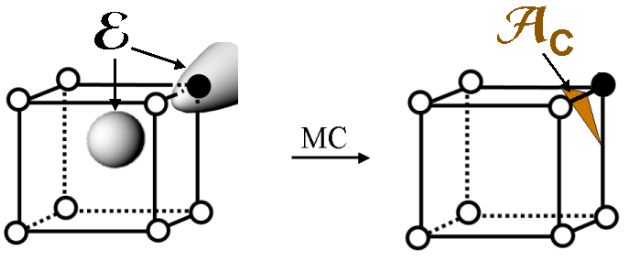
- Let $d(\boldsymbol{p}, \boldsymbol{q})$ denote the distance (in a suitable metric) between two points $\boldsymbol{p}$,$\boldsymbol{q} \in \mathcal{R}^n$. Unless explicitly stated, the metric is assumed to be Euclidean. Given a set $\mathcal{Q}$, we define the distance between a point $\mathbf{p}$ and $\mathcal{Q}$ as follows:

$$d(\boldsymbol{p}, \mathcal{Q}) = \min\{d(\boldsymbol{p}, \boldsymbol{q}) \mid \boldsymbol{q} \in \mathcal{Q}\}$$

  The diameter of a set P is defined as:

$$diam(\mathcal{P}) = \max\{d(\boldsymbol{p_1}, \boldsymbol{p_2}) \mid \boldsymbol{p_1}, \boldsymbol{p_2} \in \mathcal{P}\}$$


(a) Unreliable Detection



(b) Unreliable Reconstruction

Figure 3: Errors in MC-like reconstruction: ***When the isosurface has complicated features, MC-like methods are unreliable, and may produce inaccurate output. Fig. (a) shows three cases where the isosurface intersects the cell, but the MC-like methods cannot detect the presence of the isosurface with the cell. In these cases, MC-like methods produce no output within the cell. Fig. (b) shows a case where they output a polygon, but they do not reconstruct the surface component in the interior of the cell.***

The one-sided Hausdorff distance between two sets $\mathcal{P}$ and $\mathcal{Q}$ is defined as follows:

$$h(\mathcal{P}, \mathcal{Q}) \quad = \quad \max\{\min d(\boldsymbol{p}, \mathcal{Q}) \mid \boldsymbol{p} \in \mathcal{P}\}$$

Note that the above definition is not symmetric, i.e., $h(\mathcal{P}, \mathcal{Q})$ is not necessarily equal to $h(\mathcal{Q}, \mathcal{P})$. $h(\mathcal{P}, \mathcal{Q})$ and $h(\mathcal{Q}, \mathcal{P})$ are also referred to as the forward and backward Hausdorff distances respectively. The **two-sided Hausdorff** distance is defined as:

$$H(\mathcal{P}, \mathcal{Q}) \quad = \quad \max(h(\mathcal{P}, \mathcal{Q}), h(\mathcal{Q}, \mathcal{P}))$$

4. RELIABILITY OF MC-LIKE RECONSTRUCTION

4.1 Overview of Marching Cubes

We give a brief overview of the original Marching Cubes algorithm [2]. Given a continuous scalar field, $f : \mathbb{R}^3 \to \mathbb{R}$ and a scalar value s, an isosurface with isovalue s is the set, $\{\boldsymbol{p} \mid f(\boldsymbol{p}) = s\}$, of points with identical scalar value s. In the following discussion, we will assume that the isovalue s is zero. Marching Cubes (MC) is a simple method for generating a polygonal reconstruction $\mathcal{A}$ of an isosurface $\mathcal{E}$ in $\mathbb{R}^3$. The input to MC is a volumetric cubic grid with a scalar value at each grid vertex. MC performs reconstruction by extracting surfaces separately in every grid cell. The algorithm iterates through all grid cells, hence the term marching cubes. The algorithm operates on a single grid cell C to produce a polygonal approximation $\mathcal{A}_C$ of $\mathcal{E}_C$.

1. **Classification:** Classify each vertex of C as inside or outside $\mathcal{E}$. We refer to this inside/outside classification as the *sign* of the vertex. The sign of a point $\boldsymbol{p}$ is defined as the sign of the scalar value $f(\boldsymbol{p})$. The signs at the 8 vertices of C define a *sign configuration* $(s_1, \ldots, s_8)$ where s_i is 1 if the i^{th} vertex is positive and 0 otherwise, $i = 1, \ldots, 8$.

2. **Detection:** Test if $\mathcal{E}$ intersects C, i.e., if $\mathcal{E}_C = \mathcal{E} \cap C \neq \emptyset$. This test is performed by checking if C exhibits a *sign change*, i.e., not all the vertices of C have the same sign. In this case, proceed to Step 3. On the other hand, if all the grid vertices of C have the same sign, then assume that $\mathcal{E}_C = \emptyset$ and do not perform a reconstruction within C.

3. **Reconstruction:**
   - For each edge of C whose endpoints have different signs, estimate an edge point by linear interpolation of the scalar field along the edge.
   - Use the edge points enumerated in Step 2 to construct one or more polygonal facets separating the vertices with different signs. This is done as per the sign configuration of C. Since each of the 8 vertices of a cube can be either positive or negative, there are $2^8 = 256$ possible sign configurations. Lorenson & Cline [2] used symmetries between different sign configurations to reduce them to 15 basic cases. They stored each of these cases in a look-up table, and used them to find $\mathcal{A}_C$.

The union of $\mathcal{A}_C$ over all the grid cells produces the reconstructed isosurface $\mathcal{A}$.

In recent years, a large number of MC-like algorithms have been developed [3–5, 49] . All of them follow the same general approach outlined above.

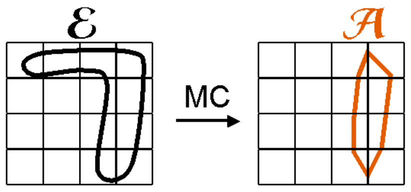
4.2 Geometric and Topological Errors

MC-like methods rely on the sign configuration of a cell C for two tasks: (a) to *detect the isosurface*, i.e., if $\mathcal{E}_C \neq \emptyset$ and (b) to reconstruct $\mathcal{E}_C$ – the portion of the isosurface within the cell. The reliance on sign configuration is merely a heuristic and not a fool-proof test.

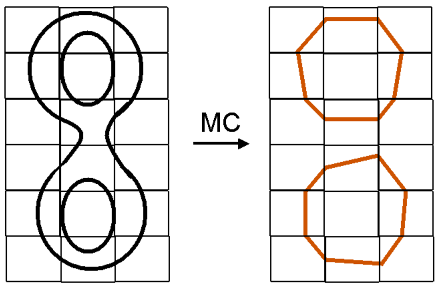
There can be two kinds of problems:

- **Unreliable Detection:** MC-like methods may wrongly assume that $\mathcal{E}$ does not intersect C and perform no reconstruction within C (Fig. 3(a)).

- **Unreliable Reconstruction:** MC-like methods produce $\mathcal{A}_C$ by indexing into a lookup table using the sign



(a) Geometric Error



(b) Topological Error

Figure 4: ***This figure shows 2D examples where MC-like methods produce output with geometric and topological errors. In Fig (a), a feature present at the top of $\mathcal{E}$ has not been captured in $\mathcal{A}$. In Fig (b) $\mathcal{A}$ does not capture all the components present in $\mathcal{E}$***

configuration of C. The sign configuration of C may not adequately capture $\mathcal{E}_C$. As a result, $\mathcal{A}_C$ may be a poor approximation to $\mathcal{E}_C$ (Fig. 3(b)).

The above two problems can lead to both geometric errors and topological errors in $\mathcal{A}$. We discuss each of them separately.

4.2.1 Geometric Errors

The inability of MC-like methods to detect $\mathcal{E}$ reliably can lead to a large geometric error in $\mathcal{A}$. See Fig. 4(a). One way of measuring the geometric error of $\mathcal{A}$ is to compute the Hausdorff distance between $\mathcal{E}$ and $\mathcal{A}$. For a definition of Hausdorff distance, see Sec. 3. From now on, we will assume that the geometric (or Hausdorff) error of $\mathcal{A}$ is equal to the two-sided Hausdorff distance $H(\mathcal{A}, \mathcal{E})$.

While it is possible for MC-like methods to guarantee a bound on the one-sided Hausdorff distance $h(\mathcal{A}, \mathcal{E})$, this guarantee is applicable only *one-way*: They do not bound the backward distance $h(\mathcal{E}, \mathcal{A})$; there can be points on $\mathcal{E}$ that are *far* from $\mathcal{A}$. See Fig. 4(a). Thus the geometric error $H(\mathcal{A}, \mathcal{E})$ can be quite large; consequently, $\mathcal{A}$ can be a poor approximation to $\mathcal{E}$.

4.2.2 Topological Errors

The problems of unreliable detection and unreliable reconstruction can cause topological errors in $\mathcal{A}$. Due to unreliable detection, MC-like methods may miss small surface components or handles present in $\mathcal{E}$. As a result, these features may not be captured in $\mathcal{A}$. See Fig. 4(b).

Due to the problem of unreliable reconstruction, the topology of $\mathcal{A}_C$ in a cell C may not match the topology of $\mathcal{E}_C$. This can introduce "extraneous topology" in $\mathcal{A}$, i.e., $\mathcal{A}$ may have unwanted additional components or undesirable handles that were not present in $\mathcal{E}$ (Fig. 2).

4.2.3 Sampling Issues

The above geometric and topological errors occur due to insufficient resolution of the volumetric grid. These errors could be avoided by choosing a grid with a sufficiently high resolution. The main question that arises is how much resolution is sufficient to ensure an accurate approximation. Our goal is to come up with a condition for what constitutes a *sufficient resolution* of the volumetric grid.

Our overall approach proceeds by sampling and reconstruction. In any approach based on sampling and reconstruction, the key to an accurate output is to ensure that the sampling satisfies the requirements of the reconstruction. The nature of requirements depend on the reconstruction method. To ensure accuracy of MC-like reconstruction methods, we impose the following requirements on the sampling.

We require that the every cell in the volumetric grid must satisfy two requirements:

1. **Reliable Detection:** $\mathcal{E}_C \neq \emptyset$ if and only if C exhibits a sign change.
2. **Reliable Reconstruction:** $\mathcal{E}_C \approx \mathcal{A}_C \approx \mathbb{B}^2$.

Failure to satisfy the above requirement implies an insufficient rate of sampling for MC-like reconstruction methods, which can cause both geometric and topological errors in $\mathcal{A}$. We avoid these errors by using a sampling condition that enforces the requirement. We present this condition in the following section.

5. SAMPLING CONDITION

We present a sampling condition that ensures accuracy during isosurface extraction. If the volumetric grid satisfies the sampling condition, then we can apply an MC-like method to obtain an approximation $\mathcal{A}$ with a bounded two-sided Hausdorff error and same topology as $\mathcal{E}$.

We first address the issue of ensuring that MC-like methods preserve topology during isosurface extraction. We defer the problem of bounding the geometric error to Sec. 7. We can achieve our goal of topology preservation by making sure that the requirements of MC-like methods are satisfied, i.e., $\mathcal{E}$ should intersect a grid cell C in a simple manner, and should have a simple topology within C. In particular, we ensure that $\mathcal{E}_C$ is a topological disk. We present a simple condition to ensure this property. Then we show that this condition is sufficient – if the every cell in the volumetric grid satisfies this condition, then an MC-like method can be reliably applied to the grid to obtain a topologically correct approximation.

Our sampling condition consists of two geometric criteria: complex cell criterion and star-shaped criterion. For these criteria to be well defined, we require that the isosurface intersect the grid cells in a *non-degenerate* manner.

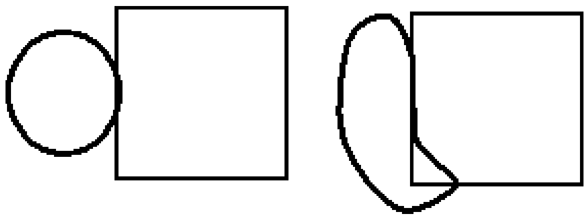


Figure 5: Grazing Contact: ***This figure shows (in 2D) two instances of an isosurface touching the boundary of a grid cell. Our algorithm requires that all the grid cells satisfy a*** **non-degeneracy condition** ***that prohibits such contacts.***

We present the non-degeneracy requirement followed by the sampling criteria.

5.1 Non-Degeneracy Condition

We require that the isosurface should not *graze* the boundary of the cell. See Fig. 5. To avoid such situations, we require that the grid cells satisfy a non-degeneracy condition.

Edelsbrunner & Shah [62] presented a condition for non-degeneracy in context of Delaunay triangulation. We use their condition to define a non-degenerate intersection of $\mathcal{E}$ and a grid cell C. Let c denote a voxel, face, or edge of C. We say $\mathcal{E}$ *intersects c generically* if

1. $\mathcal{E} \cap c = \emptyset$ or
2. $\mathcal{E} \cap (\text{int } c) = \text{relative int}(\mathcal{E} \cap c)$ and $\mathcal{E} \cap c$ has the *right dimension* where the right dimension for a voxel, a face, and an edge is 2, 1, and 0 respectively.

DEFINITION 1 *A cell is* **non-degenerate** *if $\mathcal{E}$ intersects each of its voxel, faces, and edges generically.*

5.2 Complex Cell Criterion

We define a voxel (face) of a grid cell to be *complex* if it intersects $\mathcal{E}$ and the grid vertices belonging to the voxel (face) do not exhibit a sign change (see Figs. 6(a) & 6(b)). An edge of the grid cell is said to be *complex* if $\mathcal{E}$ intersects the edge more than once (see Fig. 6(c)).

MC-like methods that operate on cubical grid cells cannot handle certain sign combinations in a topologically reliable manner. There are two types of ambiguity — *face ambiguity* and *voxel ambiguity* [51]. When the signs at the vertices of a single face alternate during counterclockwise (or clockwise) traversal, the resulting configuration is a face ambiguity. A voxel ambiguity results when any pair of diagonally opposite vertices have one sign while the other vertices have a different sign (see Fig. 6(d)). We refer to both face ambiguity and voxel ambiguity as an ambiguous sign configuration.

DEFINITION 2 *1.* **Complex cell:** A non-degenerate cell is *complex* if it has a *complex voxel, complex face, complex edge,* or an *ambiguous sign configuration.*

2. **Complex cell criterion ($\mathcal{C}^{\square}$) :** *A non-degenerate cell C satisfies $\mathcal{C}^{\square}$ if C is not complex.*

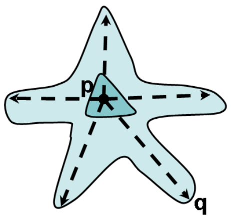


Figure 7: Star-shaped Primitive ***The figure shows a star-shaped primitive and its kernel (shaded region in the middle). P is a guard of the kernel.***

Intuitively, the complex cell criterion ensures that $\mathcal{E}$ intersects C in a simple manner most of the times. However, this criterion by itself is not sufficient. There are cases where a C may not be complex, but $\mathcal{E}_C$ may have a complicated topology (see Figs. 6(e) & 6(f)). In such cases, $\mathcal{A}_C$ will be a poor approximation to $\mathcal{E}_C$. In Fig. 6(e), MC-like methods miss the surface component present in the interior of the cell[1]. Similarly in Fig. 6(f), MC-like methods will not reconstruct the handle present in $\mathcal{E}$. We avoid such situations by enforcing a star-shaped criterion within all the grid cells.

5.3 Star-shaped Criterion

We begin by defining the star-shaped property. We consider a few cases:

1. Let S be a d-dimensional nonempty subset of $\mathbb{R}^d$. The set Kernel(S) consists of all $\boldsymbol{s} \in S$ such that for any $\boldsymbol{x} \in S$, we have $\boldsymbol{sx} \subseteq S$. Set S is *star-shaped* if Kernel(S) $\neq \emptyset$. We call a point belonging to Kernel(S) a *guard* of S. Intuitively, a guard can see every point within a star-shaped primitive. See Fig. 7.
2. Let S be a $(d-1)$-dimensional closed manifold in $\mathbb{R}^d$. The set Kernel(S) consists of all $\boldsymbol{o} \in \mathbb{R}^d$ such that for any $\boldsymbol{x} \in S$ we have $\mathbf{ox} \cap S = \{\mathbf{x}\}$. S is *star-shaped* if Kernel(S) $\neq \emptyset$. A point belonging to Kernel(S) is a *guard* of S. We assume that $\mathcal{E}$ is a 2-manifold in $\mathbb{R}^3$. Therefore we use this definition when we say $\mathcal{E}$ is star-shaped.
3. Next we define the star-shaped property for a cell. We say $\mathcal{E}$ is *star-shaped with respect to* (w.r.t) *a voxel* ϑ if there exists a point $\mathbf{o} \in \mathbb{R}^3$ such that for any $\boldsymbol{x} \in \mathcal{E}_\vartheta = \mathcal{E} \cap \vartheta$ we have $\mathbf{ox} \cap \mathcal{E}_\vartheta = \{\mathbf{x}\}$. Point $\mathbf{o}$ is a guard of $\mathcal{E}_\vartheta$. We note that it is not required to lie within the voxel. This makes the condition less restrictive.
   Consider a face f. We treat $\mathcal{E}_f = \mathcal{E} \cap f$ as a curve in $\mathbb{R}^2$. Let Π_f denote the plane containing f. We say $\mathcal{E}$ is *star-shaped w.r.t f* if there exists a point $\mathbf{o} \in \Pi_f$ such that for any $\boldsymbol{x} \in \mathcal{E}_f$, we have $\mathbf{ox} \cap \mathcal{E}_f = \{\mathbf{x}\}$. Point $\mathbf{o}$ is a guard of $\mathcal{E}_f$. We define $\mathcal{E}$ to be *star-shaped w.r.t a cell* if it is star-shaped w.r.t the cell's voxel, and each of its faces.

DEFINITION 3 Star-shaped criterion ($\mathcal{C}^{\star}$) : *A cell C satisfies $\mathcal{C}^{\star}$ if $\mathcal{E}$ is star-shaped w.r.t C.*

[1]We cannot detect the presence of the internal surface component and reconstruct it independently because we do not have an explicit representation of $\mathcal{E}$.

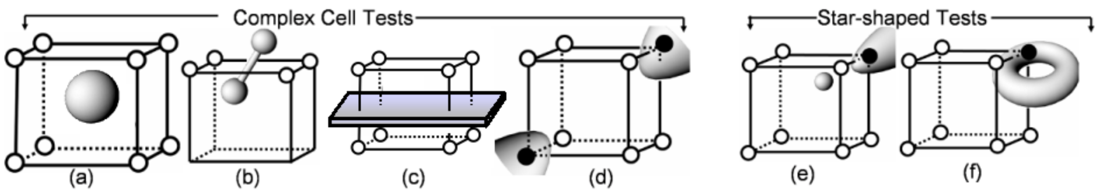


Figure 6: Complex cell and Star-shaped Test Cases: ***This figure shows the different cases corresponding to the complex cell and star-shaped test. Figs (a), (b), (c) and (d) show cases of complex voxel, complex face, complex edge, and topological ambiguity. The white and black circles denote positive and negative grid points respectively. Fig. (e) shows the case where the isosurface is not star-shaped w.r.t a voxel. In Fig (f), the restriction of the isosurface to the right face of the cell is not star-shaped.***

The surface is star-shaped w.r.t the cell in Figs. 6(a), (b), (c). On the other hand, Figs. 6(e) & 6(f) show cases where the surface is not star-shaped w.r.t a voxel or a face of the cell.

5.4 Topology Preserving Isosurface Extraction

DEFINITION 4 *A cell C satisfies $\mathcal{C}^{\square\star}$ if*

1. *$\mathcal{E}_C = \emptyset$ or*
2. *(a) C is non-degenerate,*
   (b) C satisfies $\mathcal{C}^{\square}$, and
   (c) C satisfies $\mathcal{C}^{\star}$.
3. *If C satisfies $\mathcal{C}^{\square\star}$, we refer to it as a $\mathcal{C}^{\square\star}$-*cell.

We now present our main result on topology-preserving isosurface extraction: If all the grid cells are $\mathcal{C}^{\square\star}$-cells, then MC-like algorithms extract an approximation $\mathcal{A}$ that has the same topology as $\mathcal{E}$. In order to prove this result, we first show that the intersection of $\mathcal{E}$ with a voxel, face, or edge is homeomorphic to a disk in the right dimension (provided the intersection is non-empty). This property is then used to establish topological equivalence. We begin by defining the properties of MC-like methods.

5.4.1 Properties of MC-like Reconstruction Methods

Recall that $\mathcal{A}$ is a piece-wise linear approximation of $\mathcal{E}$ obtained by performing isosurface extraction using an MC-like method. We require that the MC-like method satisfy the following properties:

Property 1: The signs of the scalar field at all the grid vertices are preserved during isosurface extraction; every grid vertex has an identical sign w.r.t both $\mathcal{E}$ and $\mathcal{A}$.

This property is satisfied by the original Marching Cubes algorithm [2] and most of its extensions [3, 49]. However, there are a few methods such as Dual Contouring [4] that may not satisfy the property. We propose an extension to the Dual Contouring algorithm that satisfies the property (Sec. 8).

Property 2: Let c be an edge, face, or voxel of a cell C such that c exhibits a sign change, i.e., not all the vertices of c have the same sign. If C is a $\mathcal{C}^{\square\star}$-cell then

$$\mathcal{A}_c \approx \mathbb{B}^k$$

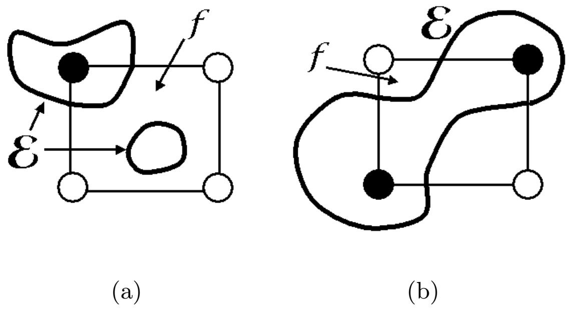


Figure 8: Intersection Curves: ***Fig. (a) supports proof of Lemma 2. It shows a grid cell face f and the intersection of $\mathcal{E}$ with f. The figure shows the intersection curve consisting of two curve components, one of which is closed. As a result, it is not star-shaped. Fig. (b) shows the case where the curve has two boundary curves. This results in a face ambiguity.***

where k is 0, 1, or 2 depending on whether c is an edge, face, or a voxel.

Consider an edge e that exhibits a sign change. MC-like algorithms output one intersection point along e. Therefore, we have $\mathcal{A}_e \approx \mathbb{B}^0$.

Let c be a face or voxel of a $\mathcal{C}^{\square\star}$-cell cell. The sign configurations for which MC-like reconstruction is not a topological disk are topologically ambiguous. Because $\mathcal{C}^{\square\star}$-cells, by definition, are not topologically ambiguous, the remaining sign configurations always result in a reconstruction such that $\mathcal{A}_c \approx \mathbb{B}^k$. This is a property of MC-like reconstruction methods.

5.4.2 Proof of Topology Preserving Reconstruction

LEMMA 1 *Let e be an edge of a $\mathcal{C}^{\square\star}$-cell such that $\mathcal{E}_e \neq \emptyset$. Then*

$$(i) \quad \mathcal{E}_e \approx \mathbb{B}^0$$
$$(ii) \quad \mathcal{A}_e \approx \mathbb{B}^0$$

Proof:
(i) Because $\mathcal{E}_e \neq \emptyset$, edge e intersects $\mathcal{E}$. There has to be exactly one intersection point: otherwise e will be complex. Therefore, $\mathcal{E}_e \approx \mathbb{B}^0$.

(ii) Because e is not complex and intersects $\mathcal{E}$, e will exhibit a sign change. By Property 2, we have $\mathcal{A}_e \approx \mathbb{B}^0$.

$\square$

DEFINITION 5 • *A set $\mathcal{P} \subseteq \mathcal{S}$ is a* **component** *of surface $\mathcal{S}$ if $\mathcal{P}$ is connected and there exists no point $\mathbf{p} \in \mathcal{P}$ such that $\mathbf{p} \overset{\mathcal{S}}{\longleftrightarrow} \mathbf{q}$ for some point $\mathbf{q} \in \mathcal{S} \setminus \mathcal{P}$.*

- *We call a component $\mathcal{P}$ a* **component with boundary** *if it has a nonempty boundary. Otherwise, we call $\mathcal{P}$ a* **closed component***.*

Similar definitions also hold in 2D for components of curves.

LEMMA 2 *Let f be a face of a $\mathcal{C}^{\square\bigstar}$-cell such that $\mathcal{E}_f \neq \emptyset$. Then*

$$\begin{aligned} (i) \quad & \mathcal{E}_f \approx \mathbb{B}^1 \\ (ii) \quad & \mathcal{A}_f \approx \mathbb{B}^1 \end{aligned}$$

Proof: **(i)** We will show that $\mathcal{E}_f$ is a curve component with boundary, or in short, a boundary curve. $\mathcal{E}_f$ can have no closed component. We prove this by contradiction. Suppose $\mathcal{E}_f$ has a closed component. Then $\mathcal{E}_f$ cannot have any other curve component because it will contradict the fact that $\mathcal{E}_f$ is star-shaped w.r.t f (see Fig. 8(a)). But if $\mathcal{E}_f$ is a single closed component, then f is a complex face. Therefore, $\mathcal{E}_f$ cannot have any closed components.

This means that $\mathcal{E}_f$ consists of a set of boundary curves. We will show that there exists only one boundary curve. We will prove this by contradiction. Suppose that $\mathcal{E}_f$ has multiple boundary curves. The complex edge criterion ensures that a boundary curve intersects exactly two edges of the face f. It also ensures that two boundary curves cannot intersect the same edge. Therefore, $\mathcal{E}_f$ can have at most two boundary curves, each intersecting two edges of f. However, this results in a face ambiguity (see Fig. 8(b)). Hence, $\mathcal{E}_f$ can have only a single boundary curve. This means $\mathcal{E}_f \approx \mathbb{B}^1$.

(ii) f intersects $\widetilde{\mathcal{E}}$. Because f is not complex, it must exhibit a sign change. By Property 2, we have $\mathcal{A}_f \approx \mathbb{B}^1$.

$\square$

LEMMA 3 *Let ϑ be a voxel of a $\mathcal{C}^{\square\bigstar}$-cell such that $\mathcal{E}_\vartheta \neq \emptyset$. Then*

$$\begin{aligned} (i) \quad & \mathcal{E}_\vartheta \approx \mathbb{B}^2 \\ (ii) \quad & \mathcal{A}_\vartheta \approx \mathbb{B}^2 \end{aligned}$$

Proof: **(i)** We prove that $\mathcal{E}_\vartheta$ cannot contain any closed component. Similar to the proof of Lemma 2, the presence of a closed component would imply that either the primitive is not star-shaped w.r.t. ϑ, or ϑ is a complex voxel.

We now prove that $\mathcal{E}_\vartheta$ has at most one surface component with a boundary and is connected. The boundary of each surface component corresponds to a boundary curve on the faces of the cell. From the result of Lemma 2, each face contains only one boundary curve. Furthermore, the complex face and complex edge criteria preclude the boundary curve from intersecting one or two faces. Therefore, each boundary curve intersects at least three faces. Since each face can have at most one boundary curve, $\mathcal{E}_\vartheta$ cannot have more than two surface components. The only way there can be two surface components is if two diagonally opposite cell vertices are inside the primitive while the others are outside (see Fig. 6(d)). This is the case of a voxel ambiguity. Therefore, $\mathcal{E}_\vartheta$ has at most one surface component with a boundary and is connected.

To show that $\mathcal{E}_\vartheta$ is a topological disk, we show that $\mathcal{E}_\vartheta$ is a simply connected surface. Suppose that $\mathcal{E}_\vartheta$ is not simply connected. This means that $\mathcal{E}_\vartheta$ has two or more boundaries. The remainder of the proof is similar to the above argument. Existence of two boundaries results in a voxel ambiguity. Therefore, $\mathcal{E}_\vartheta$ is simply connected. This proves that $\mathcal{E}_\vartheta$ is a topological disk. This concludes the proof.

(ii) ϑ intersects $\widetilde{\mathcal{E}}$. Because ϑ is not complex, it must exhibit a sign change. By Property 2, we have $\mathcal{A}_\vartheta \approx \mathbb{B}^2$.

$\square$

THEOREM 1 *If all the grid cells are $\mathcal{C}^{\square\bigstar}$-cells, then*

$$\mathcal{E} \approx \mathcal{A}$$

Proof: Lemmas 1, 2, and 3 establish that the restrictions of $\mathcal{E}$ and $\mathcal{A}$ to the edges, faces, and voxels of the grid are homeomorphic to each other. This fact can be used to construct a homeomorphism $\mathcal{H}$ between $\mathcal{E}$ and $\mathcal{A}$ inductively. We first define $\mathcal{H}$ on the edges, then faces, and finally voxels of the grid.

Edge Case: Consider an edge e of the grid that intersects $\mathcal{E}$. According to Lemma 1, e intersects both $\mathcal{E}$ and $\mathcal{A}$ once. Let $\mathbf{p}$ and $\mathbf{q}$ be the intersection of e with $\mathcal{E}$ and $\mathcal{A}$ respectively. We define $\mathcal{H}_e(\mathbf{p}) = \mathbf{q}$. In this manner, we define homeomorphism on the edges of the grid.

Face Case: Consider a face f of the grid such that $\mathcal{E}_f \neq \emptyset$. According to Lemma 2, both $\mathcal{E}_f \approx \mathbb{B}^1$ and $\mathcal{A}_f \approx \mathbb{B}^1$. Therefore, there exists a homeomorphism between $\mathcal{E}_f$ and $\mathcal{A}_f$. However, for our purpose, any homeomorphism does not suffice. We need to define a homeomorphism is consistent with the homeomorphisms defined on the edges of face f.

The complex edge criterion ensures that $\mathcal{E}_f$ must intersect two edges bounding the face. Let e_1 and e_2 be the two edges. $\mathcal{E}$ intersects e_1 and e_2 at intersection points $\mathbf{p_1}$ and $\mathbf{p_2}$, where $\mathbf{p_1} = \mathcal{E}_{e_1}$, and $\mathbf{p_2} = \mathcal{E}_{e_2}$. By Lemma 1, $\mathcal{A}$ also intersects e_1 and e_2 at intersection points $\mathbf{q_1}$ and $\mathbf{q_2}$, where $\mathbf{q_1} = \mathcal{A}_{e_1}$, and $\mathbf{q_2} = \mathcal{A}_{e_2}$. See Fig. 9.

Because $\mathcal{E}_f \approx \mathbb{B}^1$, there exists a homeomorphism $\mathcal{H}_1 : [0,1] \to \mathcal{E}_f$ such that $\mathcal{H}_1(0) = \mathbf{p_1}$ and $\mathcal{H}_1(1) = \mathbf{p_2}$. Similarly, there exists a homeomorphism $\mathcal{H}_2 : [0,1] \to \mathcal{A}_f$ such that $\mathcal{H}_2(0) = \mathbf{q_1}$ and $\mathcal{H}_2(1) = \mathbf{q_2}$. Define a *face homeomorphism* $\mathcal{H}_f : \mathcal{E}_f \to \mathcal{A}_f$ as $\mathcal{H}_f = \mathcal{H}_2 \circ \mathcal{H}_1^{-1}$ (Fig. 9).

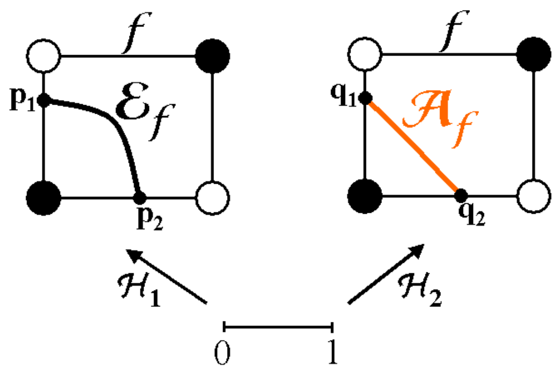


Figure 9: Face Homeomorphism: ***This figure shows how to construct a homeomorphism between $\mathcal{E}_f$ and $\mathcal{A}_f$ on a face f. The homeomorphism is given by $\mathcal{H}_2 \circ \mathcal{H}_1^{-1}$.***

Note that $\mathcal{H}_f$ is consistent with the homeomorphisms defined on the edges bounding the face. For example, we have

$$\begin{aligned}\mathcal{H}_f(\mathcal{E}_{e_1}) &= \mathcal{H}_f(\mathbf{p_1}) \\ &= \mathcal{H}_2 \circ \mathcal{H}_1^{-1}(\mathbf{p_1}) \\ &= \mathcal{H}_2(0) \\ &= \mathbf{q_1} \\ &= \mathcal{A}_{e_1}\end{aligned}$$

Voxel Case: Consider a voxel ϑ of the grid and let $\mathcal{E}_\vartheta \neq \emptyset$. Let $f_1, \ldots, f_k$ denote the cell faces that intersect $\mathcal{E}_\vartheta$. Our goal is to define a *voxel homeomorphism* $\mathcal{H}_\vartheta$ between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$ that is consistent with the face homeomorphisms $\mathcal{H}_{f_i}, i = 1, \ldots, k$.

$\mathcal{E}_\vartheta$ is bounded by a set $\mathcal{E}_{f_1}, \ldots, \mathcal{E}_{f_k}$ of boundary curves. $\mathcal{E}_\vartheta$ also intersects k edges of the cell. Let $\{\mathbf{p_1}, \ldots, \mathbf{p_k}\}$ denote the set of intersection points along the edges. According to Lemma 3, we have $\mathcal{E}_\vartheta \approx \mathbb{B}^2$. Therefore, there exists a homeomorphism $\mathcal{H}_1 : \mathcal{E}_\vartheta \to \mathbb{B}^2$. Let $\mathcal{H}_1(\mathbf{p_i}) = \mathbf{a_i}$ and $\mathcal{H}_1(\mathcal{E}_{f_i}) = l_i, , i = 1, \ldots, k$. The points $\mathbf{a_1}, \ldots, \mathbf{a_k}$ belong to $\partial\mathbb{B}^2 = \mathbb{S}^1$, and partition $\mathbb{S}^1$ into k arcs $l_1, \ldots, l_k$. See Fig. 10.

Similar arguments hold for $\mathcal{A}_\vartheta$. $\mathcal{A}_\vartheta$ intersects the same set of cell faces and cell edges as $\mathcal{E}_\vartheta$. $\mathcal{A}_\vartheta$ is bounded by a set $\mathcal{A}_{f_1}, \ldots, \mathcal{A}_{f_k}$ of boundary curves. $\mathcal{A}_\vartheta$ intersects the cell edges giving rise to a set $\{\mathbf{q_1}, \ldots, \mathbf{q_k}\}$ of intersection points. Since $\mathcal{A}_\vartheta \approx \mathbb{B}^2$, there exists a homeomorphism $\mathcal{H}_2 : \mathcal{A}_\vartheta \to \mathbb{B}^2$. Let $\mathcal{H}_2(\mathbf{q_i}) = \mathbf{b_i}$ and $\mathcal{H}_2(\mathcal{A}_{f_i}) = m_i, , i = 1, \ldots, k$. The points $\mathbf{b_1}, \ldots, \mathbf{b_k}$ belong to $\mathbb{S}^1$, and partition $\mathbb{S}^1$ into k arcs $m_1, \ldots, m_k$ (Fig. 10).

It is possible to construct a homeomorphism $\mathcal{H}_3 : \mathbb{S}^1 \to \mathbb{S}^1$ that maps l_i to m_i. We can use an argument similar to the one in the **Face Case** to construct a homeomorphism $\mathcal{H}_3^i$ between l_i and m_i that maps a_i to b_i and a_{i+1} to b_{i+1}. $\mathcal{H}_3$ is then defined as an extension of $\mathcal{H}_3^i, i = 1, \ldots, k$; $\mathcal{H}_3(\mathbf{x}) = \mathcal{H}_3^i(\mathbf{x})$ if $\mathbf{x} \in l_i$

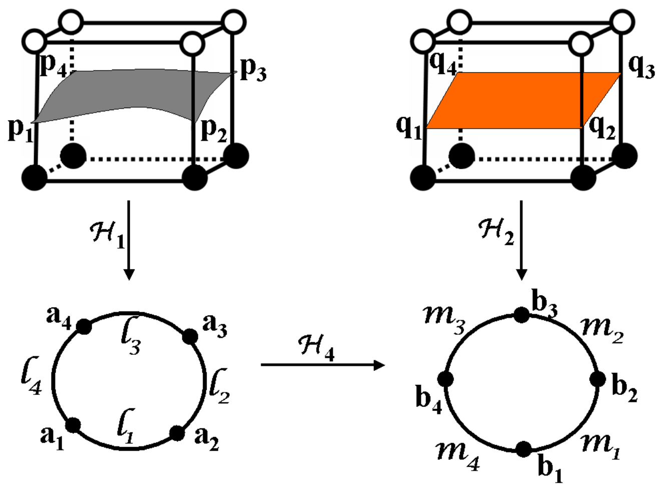


Figure 10: Voxel Homeomorphism: ***This figure shows how to construct a homeomorphism between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$ in a voxel ϑ. The homeomorphism is given by $\mathcal{H}_2^{-1} \circ \mathcal{H}_4 \circ \mathcal{H}_1$.***

We can then extend $\mathcal{H}_3$ to define a homeomorphism $\mathcal{H}_4 : \mathbb{B}^2 \to \mathbb{B}^2$. $\mathcal{H}_4$ is defined as follows:

$$\begin{aligned}\mathcal{H}_4(0) &= 0 \\ \mathcal{H}_4(\mathbf{x}) &= \|\mathbf{x}\|_2 * \mathcal{H}_3(\mathbf{x}/\|\mathbf{x}\|_2), \mathbf{x} \in \mathbb{B}^2, \|\mathbf{x}\|_2 > 0\end{aligned}$$

Since $\mathcal{H}_4$ is an extension of $\mathcal{H}_3$, it also maps l_i to m_i.

We are now ready to define a homeomorphism $\mathcal{H}_\vartheta$ between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$. Define $\mathcal{H}_\vartheta = \mathcal{H}_2^{-1} \circ \mathcal{H}_4 \circ \mathcal{H}_1$. $\mathcal{H}_\vartheta$ is a homeomorphism between $\mathcal{E}_\vartheta$ and $\mathcal{A}_\vartheta$. See Fig. 10. Note that it is consistent with the homeomorphisms defined on the faces of the cell. In particular, we have

$$\begin{aligned}\mathcal{H}_\vartheta(\mathcal{E}_{f_i}) &= \mathcal{H}_2^{-1} \circ \mathcal{H}_4 \circ \mathcal{H}_1(\mathcal{E}_{f_i}) \\ &= \mathcal{H}_2^{-1} \circ \mathcal{H}_4(l_i) \\ &= \mathcal{H}_2^{-1}(m_i) \\ &= \mathcal{A}_{f_i}\end{aligned}$$

We define the homeomorphism $\mathcal{H} : \mathcal{E} \to \mathcal{A}$ in terms of the homemorphisms defined on the edges, faces, and voxels of the grid.

$$\begin{aligned}\mathcal{H}(\mathbf{x}) &= \mathcal{H}_\vartheta(\mathbf{x}) && \text{if } \mathbf{x} \text{ belongs to a voxel } \vartheta \\ &= \mathcal{H}_f(\mathbf{x}) && \text{if } \mathbf{x} \text{ belongs to a face } f \\ &= \mathcal{H}_e(\mathbf{x}) && \text{if } \mathbf{x} \text{ belongs to an edge } e\end{aligned}$$

□

The above proof assumes that the reconstructed isosurface in two adjacent cells matches along the common face shared by the two cells. While the original Marching Cubes algorithm ensures this property for uniform grids, applying the algorithm to an adaptive grid violates the property, resulting in cracks in the reconstructed isosurface. Several extensions have been proposed to rectify this problem [4,46]. We assume that one of these algorithms is used for reconstruction in our applications. We will address this issue in more detail in Sec. 8.

In the above proofs, we showed that $\mathcal{E}$ restricted to a cell is a topological disk in the right dimension. This topological disk property is then used to establish topological equivalence of $\mathcal{E}$ and $\mathcal{A}$. This property is similar to the *topological ball* property proposed by Edelsbrunner & Shah [62]. They used the topological ball property to ensure topology preservation in the context of Delaunay triangulation. While our approach shares the goal of topology preservation, it is different in that it is geared towards MC-like reconstruction methods. The novelty of our overall approach lies in the use of two simple criteria – complex cell and star-shaped criteria – to guarantee topology preserving MC-like reconstruction.

6. TOPOLOGY PRESERVING SAMPLING

In this section, we provide adaptive subdivision criteria to generate a grid such that each grid cell is a $\mathcal{C}^{\square\bigstar}$-cell. Our sampling algorithm performs two tests on each grid cell. The first test checks if a cell satisfies $\mathcal{C}^{\square}$ using a *cell intersection query* (Sec. 6.1). The second test checks if a cell satisfies $\mathcal{C}^{\bigstar}$ using a *star-shaped query* (Sec. 6.2). If the grid cell fails to satisfy either of the two tests, the cell is subdivided and the two tests are recursively applied to the children cells. If the grid cell satisfies both the tests, the cell is returned as a leaf node in the octree grid. Fig. 11 shows a 2D illustration of the adaptive subdivision.

We first describe computational techniques for polyhedral models and their Boolean combinations, and later extend them to non-linear primitives. We also discuss details of the adaptive subdivision algorithm.

6.1 Cell Intersection Query

The objective of cell intersection query is to test if $\mathcal{E}$ intersects the cell. Specifically, we need to test if $\mathcal{E}$ intersects a voxel, a face, or an edge of a cell. We refer to these three tests collectively as cell intersection queries, and individually as voxel, face, and edge intersection query.

6.1.1 Interval Arithmetic

One technique for performing the cell intersection query is using interval arithmetic. Early work on interval arithmetic was done by Moore [63]. Since then, it has been widely used in a large number of domains including computer graphics [58]. Interval arithmetic is a general technique that is applicable to a wide variety of primitives. It is well suited to non-linear primitives. It also extends easily to higher dimensions. An overview of interval arithmetic is given in [58].

Given a primitive $\mathcal{P}$ defined by an algebraic function $f : \mathbb{R}^3 \rightarrow \mathbb{R} = 0$, we can write an interval form $\overline{f}$ of the function. Then we can use $\overline{f}$ to answer the cell intersection query. We take advantage of the fact that each edge/face/voxel corresponds to a product of intervals.

Consider a voxel ϑ of an axis-aligned grid. ϑ corresponds to a product of intervals defined by two *diametrically opposite* vertices of the voxel. Let $v^i, i = 0, \ldots, 7$ denote the vertices of the voxel. Let $v^i = (v^i.x, v^i.y, v^i.z)$. Let $s^i = v^i.x + v^i.y + v^i.z$. Given a set $\{a_1, \ldots, a_n\}, a_i \in \mathbb{R}, i = 1, \ldots, n$, let $\arg\min_i a_i$ return k if $a_k = \min_i a_i$. $\arg\max$ is defined similarly. Let

$$\mathbf{p} = v^k \quad \text{where} \quad k = \arg \min_i s^i$$
$$\mathbf{q} = v^l \quad \text{where} \quad l = \arg \max_i s^i$$

The voxel ϑ corresponds to a product of intervals $I_\vartheta = [\mathbf{p}.x, \mathbf{q}.x] \times [\mathbf{p}.y, \mathbf{q}.y] \times [\mathbf{p}.z, \mathbf{q}.z]$.

We can test if the boundary of $\mathcal{P}$ intersects ϑ by evaluating $\overline{f}$ on I_ϑ. We use the following fact:

$$\vartheta \text{ intersects the boundary of } \mathcal{P} \quad \text{if} \quad 0 \in \overline{f}(I_\vartheta)$$

Interval arithmetic can also handle voxels that are not axis-aligned by applying a rigid transformation to I_ϑ before evaluating $\overline{f}$.

The conservativeness of interval arithmetic makes the above intersection test conservative: While the test is guaranteed to be satisfied by a voxel that intersects the surface, it may also be satisfied by some voxels that do not actually intersect the surface. This does not, however, affect the correctness of our algorithm.

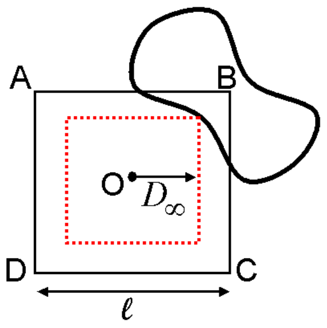


Figure 12: Voxel Intersection Test: ***We use the l_∞ distance (indicated by the dotted red cube) to perform a voxel-intersection test. The isosurface intersects the voxel if and only if l_∞ distance between the center of the voxel (o) and the isosurface is less than half the voxel size.***

6.1.2 Max-Norm Distance Computation

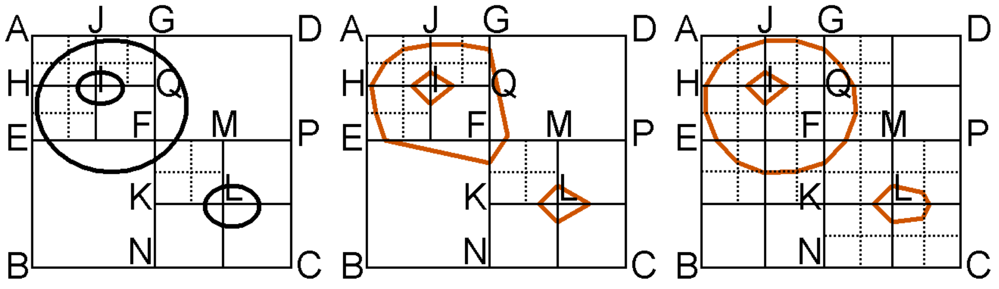
Another technique to answer the cell intersection query relies on max-norm distance computation [64]. It is efficient in practice, and is well suited to polyhedral and low degree non-linear primitives. Under the max-norm, the distance between two points $\mathbf{p}$ and $\mathbf{q}$ (in 3 dimensions) is denoted as $D_\infty(\mathbf{p}, \mathbf{q})$ and is defined as

$$D_\infty(\mathbf{p}, \mathbf{q}) = \max_i |p_i - q_i|, \qquad i = 1, 2, \ldots, 3$$

We can extend this definition for distance between a point $\mathbf{p}$ and a set $\mathcal{Q}$ in $\mathbb{R}^3$.

$$D_\infty(\mathbf{p}, \mathcal{Q}) = \min_{\mathbf{q} \in \mathcal{Q}} D_\infty(\mathbf{p}, \mathbf{q}) \qquad (1)$$

The iso-distance ball, i.e., the set of points at a constant distance from the origin, under max-norm is a cube; so it is a natural metric for cubical cells. The above definition can be extended to cuboids by defining a suitably weighted version of the max-norm along different dimensions.



(a) 2D Example (b) Correct Topology (c) Bounded Geometric Error

Figure 11: Adaptive Subdivision: ***This figure is a 2D illustration of our adaptive subdivision algorithm. Fig. (a) shows a volumetric grid generated by applying the sampling condition. Fig. (b) shows a topology preserving approximation obtained by applying an MC-like method to the volumetric grid. Fig. (c) shows an approximation with a bounded geometric error (see Sec. 7). Our algorithm performs adaptive subdivision until the isosurface within each cell is a topological disk. We ensure this condition by testing whether a cell is complex and if the isosurface is star-shaped with respect to the cell. In this figure, cell ABCD was subdivided because it corresponds to a complex voxel, cells AEFG and FNCP were subdivided because the isosurface within the cell was not star-shaped and FKLM was subdivided because of topological ambiguity. Edge IJ is complex; as a result, cells AHIJ and JIQG are subdivided.***

For a closed primitive, we use a signed version of the distance. Let $\mathcal{Q}$ denote the boundary of the closed primitive and let $\widetilde{\mathcal{Q}}$ be the solid bounded by $\mathcal{Q}$. We define the *signed max-norm distance* $D^s_\infty(\mathbf{p}, \mathcal{Q})$ as follows:

$$D^s_\infty(\mathbf{p}, \mathcal{Q}) = sign(\mathbf{p}, \mathcal{Q}) * \min_{\mathbf{q}\in\mathcal{Q}} D_\infty(\mathbf{p}, \mathbf{q}) \qquad (2)$$

where

$$\begin{aligned} sign(\mathbf{p}, \mathcal{Q}) &= -1 \quad \textit{if } \mathbf{p} \in \widetilde{\mathcal{Q}} \\ &= 1 \quad \textit{otherwise} \end{aligned}$$

We use max-norm distance computation to check whether $\mathcal{E}$ intersects a voxel of the cell. We use the fact that a voxel intersects the surface if and only if its unsigned three-dimensional max-norm (l_∞) distance from the center of the voxel is less than half the size of the cell. This is shown in Fig. 12. It is formally stated as the following lemma:

LEMMA 4 Voxel Intersection Test *Given a voxel ϑ,*

$$\mathcal{E}_\vartheta \neq \emptyset \quad \Leftrightarrow \quad |D^s_\infty(\mathbf{o}, \mathcal{E})| = D_\infty(\mathbf{o}, \mathcal{E}) \leq l/2$$

where $\mathbf{o}$ *and* l *are the center and width of* ϑ *respectively.*

Similarly, the face intersection test for a face f can be performed by computing two-dimensional max-norm distance between the center of f and $\mathcal{E}_f$. In this manner, we can use max-norm distance to perform the cell intersection query. We can efficiently compute max-norm distance for a wide variety of geometric primitives [64].

Boolean Expression
When $\mathcal{E}$ is defined by a Boolean expression involving a number of primitives, it is difficult to compute the signed distance $D^s_\infty(\mathbf{p}, \mathcal{E})$ because we do not have an explicit boundary representation of $\mathcal{E}$. Instead, we compute an estimate $\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})$ of the signed max-norm distance.

The Boolean operations on the primitives define a solid whose boundary is $\mathcal{E}$. Let $\widetilde{\mathcal{E}}$ denote this solid. We perform a case analysis on $\widetilde{\mathcal{E}}$ to define $\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})$. We note that $\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})$ is an estimate of the signed max-norm distance; hence it is also signed.

1. $\widetilde{\mathcal{E}}$ is a primitive solid $\widetilde{\mathcal{P}}$. Let $\mathcal{P}$ denote the boundary of $\widetilde{\mathcal{P}}$. We have
$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = D^s_\infty(\mathbf{p}, \mathcal{P})$$
2. $\widetilde{\mathcal{E}}$ is a union of two solids: $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$.
$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = \min(\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_1), \widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_2))$$
3. $\widetilde{\mathcal{E}}$ is an intersection of two solids: $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2$.
$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = \max(\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_1), \widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_2))$$
4. $\widetilde{\mathcal{E}}$ is the complement of a solid: $\widetilde{\mathcal{E}} = \overline{\widetilde{\mathcal{E}}_1}$.
$$\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}) = -\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E}_1)$$

We use $\widetilde{D}^s_\infty$ to perform the voxel intersection test. $\widetilde{D}^s_\infty$ may not be equal to the actual signed distance D^s_∞ at some points. However, its absolute value is always less than the absolute value of D^s_∞. This is stated in the following lemma.

LEMMA 5

$$|\widetilde{D}^s_\infty(\mathbf{p}, \mathcal{E})| < |D^s_\infty(\mathbf{p}, \mathcal{E})|$$

We skip the proof. It follows directly from the definition of $\widetilde{D}^s_\infty$.

□

While the above property makes the voxel intersection test conservative, it preserves its correctness: If a voxel intersects $\mathcal{E}$, we take it into account. On the other hand, we may also take into account some voxels that do not intersect $\mathcal{E}$. While this may result in some unnecessary computation, it does not affect the correctness of the algorithm.

A similar technique can also be used to perform interval arithmetic on Boolean combinations of primitives. Given a voxel ϑ, we first apply interval arithmetic to each of the primitives, and then perform min/max operations on the resulting intervals. This produces an interval $I_{\mathcal{E},\vartheta}$ for $\mathcal{E}$. We then check if $0 \in I_{\mathcal{E},\vartheta}$ to test whether $\mathcal{E}$ intersects ϑ.

Edge Intersection Query

We use directed distances [3] to answer the edge intersection query. The directed distance between a point $\mathbf{p}$ and a primitive $\mathcal{Q}$ along a unit vector $\vec{\mathbf{v}}$ is the distance to the closest point on the primitive along $\vec{\mathbf{v}}$. It is denoted as $D_{\vec{v}}(\mathbf{p},\mathcal{Q})$ and is defined as:

$$
\begin{aligned}
S &= \{\mathbf{q} \in \mathcal{Q} \mid \exists \lambda > 0 \text{ such that } \mathbf{q} - \mathbf{p} = \lambda \vec{\mathbf{v}}\} \quad (3)\\
D_{\vec{\mathbf{v}}}(\mathbf{p},\mathcal{Q}) &= \min\{d(\mathbf{p},\mathbf{q}) \mid \mathbf{q} \in S\} \quad \text{if } S \neq \emptyset \quad (4)\\
&= \infty \quad \text{otherwise}
\end{aligned}
$$

Our edge intersection test is based on the following property: If an edge $\mathbf{ab}$ intersects $\mathcal{E}$, then the directed distance at $\mathbf{a}$ along the direction vector $\vec{\mathbf{ab}}$ is less than the length of the vector $\vec{\mathbf{ab}}$. Based on this fact, we define an edge $\mathbf{ab}$ to be intersecting if

$$D_{\vec{\mathbf{ab}}}(\mathbf{a},\mathcal{E}) < d(\mathbf{a},\mathbf{b}).$$

Kobbelt et al. [3] have presented computational techniques for computing directed distance for a wide variety of geometric primitives. If $\mathcal{E}$ is defined as a Boolean expression, then we can compute a conservative estimate $\widetilde{D}_{\vec{v}}$ of the directed distance in a manner similar to the max-norm distance.

6.2 Star-shaped Query

The computation of the exact kernel of an orientable polyhedral primitive reduces to the intersection of halfspaces determined by the tangent planes of the faces of the primitive. Using the point-hyperplane duality, this is equivalent to convex hull computation. In $\mathbb{R}^3$, for a polyhedral surface with n facets, this can be performed in $O(n \log n)$ [65]. However, to test if a primitive is star-shaped, it suffices to check if the kernel is empty or not. We refer to this test as the *star-shaped query*.

For the sake of simplicity, we first consider the star-shaped query for polyhedral primitives. We discuss extension to non-linear primitives in Section 6.4. For a polyhedral primitive, testing for a non-empty kernel reduces to linear programming (LP) [66]. If $\boldsymbol{p}$ is a point belonging to the kernel, then each face of the polyhedron with centroid $\boldsymbol{c}$ and outward normal $\boldsymbol{n}$ defines the linear constraint $\boldsymbol{n} \cdot (\boldsymbol{c} - \boldsymbol{p}) > 0$ on $\boldsymbol{p}$. As a result, the kernel is non-empty if the set of constraints admits a feasible solution for $\boldsymbol{p}$. In fixed dimensions, LPs can be solved in linear time, and a number of efficient public domain implementations are available [67, 68].

6.2.1 Boolean Expression

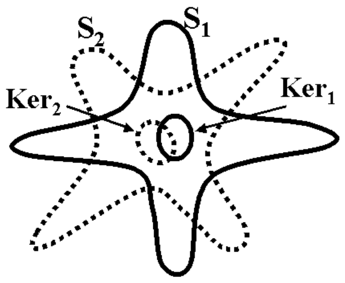


Figure 13: Star-shaped test for Boolean Combination: ***If two star-shaped primitives $\mathcal{S}_1$ and $\mathcal{S}_2$ are star-shaped w.r.t a common point, i.e. $Kernel(\mathcal{S}_1)$ overlaps with $Kernel(\mathcal{S}_2)$, then both $\mathcal{S}_1 \cup \mathcal{S}_2$ and $\mathcal{S}_1 \cap \mathcal{S}_2$ are star-shaped.***

Algorithm 1 `Adaptive_Subdivision`(C)
Input: Grid cell C associated with a Boolean expression $\widetilde{\mathcal{E}}_C$.
Output: An adaptive subdivision of C such that any MC-like algorithm generates topologically correct output.

```
if $C$ can be disregarded by cell culling (Sec. 9.1) then
  return
end if
if $C$ satisfies complex cell and star-shaped tests then
  return
end if
Subdivide $C$ into children cells $C_i, i = 1, \ldots, k$
for $i = 1$ to $k$ do
  Perform expression simplification w.r.t $C_i$: $\widetilde{\mathcal{E}}_C \xrightarrow{C_i} \widetilde{\mathcal{E}}_{C_i}$ (Sec. 9.2)
  Adaptive_Subdivision($C_i$)
end for
```

When $\mathcal{E}$ is defined by a Boolean expression involving a number of primitives, a sufficient condition for the star-shapedness of $\mathcal{E}$ is that the intersection of all the primitive kernels is non-empty. If $\mathcal{S}_1$ and $\mathcal{S}_2$ are two star-shaped primitives with a common guard, then $\mathcal{S}_1 \odot \mathcal{S}_2$ is also star-shaped where $\odot$ denotes a Boolean operation such as union and intersection. This is because

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

See Fig. 13.

For polyhedral primitives, we check for the above condition by combining the linear constraints defined by the individual primitives and testing for feasibility by solving the resulting LP. The difference operation can be rewritten as an intersection by inverting the linear constraints of the negated primitive.

We note here that the above condition is sufficient, but not necessary. We do not perform an exact test as we do not have an explicit representation of $\mathcal{E}$.

6.3 Adaptive Subdivision Algorithm

We start with a single grid cell that is guaranteed to bound $\mathcal{E}$. We perform two tests, *complex cell test* and *star-shaped* *test*, to decide whether to subdivide a grid cell. We now describe each of these tests.

6.3.1 Complex Cell Test

To check whether a cell is complex, we perform the following tests:

- **Complex Voxel/Face**: We use the cell intersection query to check whether $\mathcal{E}$ intersects a voxel or face of the cell. If $\mathcal{E}$ intersects the voxel (face), then we determine if the voxel (face) is complex by checking for a sign change at the cell vertices. If $\mathcal{E}$ does not intersect the voxel (face), then the voxel (face) is not considered complex.

- **Complex Edge**: We use directed distances [3] to test if an edge is complex. An edge is complex if the sum of the directed distances (along the edge) from the two endpoints of the edge is less than the edge length.

- **Ambiguity**: We use the signs at the grid vertices to resolve cases corresponding to face and voxel ambiguity (see Fig. 8(b) and Fig. 6(d)).

If any of these tests results in the affirmative, the cell is complex, and we subdivide it and apply the algorithm recursively to the new cells. See Fig. 11. Alg. 1 shows the pseudo-code of our algorithm.

6.3.2 Star-shaped Test

Linear programming (LP) is used to test for the star-shapedness of a polyhedral primitive. As described earlier in this section, $\mathcal{E}$ described by a single primitive or implicitly by a collection of primitives can be conservatively checked for the star-shaped property. We need to perform two tests on each cell C – (a) star-shaped w.r.t voxel of C, and (b) star-shaped w.r.t. each face of C.

For each polyhedral primitive, we consider only those faces of the polyhedron that intersect the voxel. This set of faces defines the constraints for an LP in $\mathbb{R}^3$, whose solution answers the test (a). For each face of the cell, we consider those faces of the polyhedron that intersect it. These faces result in a collection of piecewise linear segments on the face of the cell. Solving a similar LP defined by these linear curves in $\mathbb{R}^2$ answers test (b). When there are multiple primitives intersecting the cell, we combine the linear constraints arising from each primitive and then solve the resulting LP. Since we are only dealing with linear programs in two and three dimensions, a dual formulation of constraints and objective function is much more efficient in practice. We use such a formulation to perform the star-shaped test.

If either of these tests turns out to be negative, we subdivide the cell. Fig. 11 illustrates the working of the algorithm on a 2D example.

6.4 Star-shaped Query for Non-linear Primitives

In the previous subsections, we presented methods for performing the complex cell and star-shaped tests. We used interval arithmetic and max-norm distance computation for the complex cell test. Both of these methods are applicable to a wide class of geometric primitives. On the other hand,

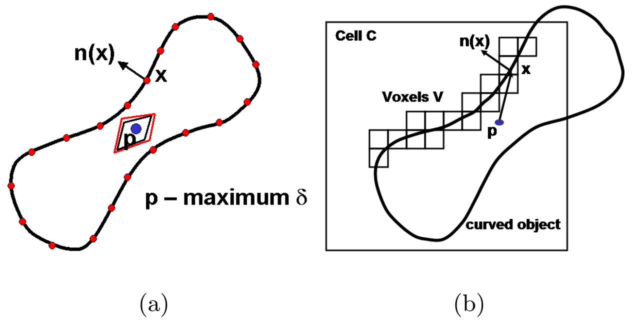


Figure 14: Star-shaped test on curved primitives: ***Fig. (a) shows the candidate point selection. The black curve is the curved primitive, while the red points form a discretized approximation. The same color scheme is used for the respective kernels. The linear program is set up so that if an approximate kernel exists, the objective function δ is optimized near its center. Fig. (b) shows the kernel membership test. After choosing candidate point $\mathbf{p}$, we use interval arithmetic to perform the test. Given a cell C, we compute an initial voxelization V of the primitive and use the intervals generated by each of the voxels inside C in the interval arithmetic step. Since the voxels closely approximate the primitive, the performance of the membership test is significantly improved.***

our method for the star-shaped query using linear programming was restricted to polyhedral primitives.

In this section, we describe a method to extend this computation to non-linear primitives. It is well-known that the kernel for non-linear closed primitives like parametric and algebraic surfaces can be computed by finding the intersection of the tangent planes at points on the surface with zero Gaussian curvature [69]. However, this approach involves solving a system of high degree equations and curve tracing, which is computationally intensive. Therefore, the applicability of this approach is limited.

We describe a method that avoids exact kernel computation. As previously observed, the star-shaped query reduces to testing whether the kernel is empty or not. Therefore, all we need is a point that is witness to the actual kernel. We describe a simple method to conservatively perform this test. This method proceeds by selecting a candidate point that lies in the interior of an approximate kernel of a discretized version of the primitive. This point is computed by linear programming. We check if the candidate point belongs to the kernel of the curved primitive by using interval arithmetic. In this section, we provide the details of the candidate point selection and kernel membership test.

6.4.1 Candidate Point Selection

We start with a discretization of the curved primitive. We compute a set of sample points $\mathbf{p_i}, i = 1, \ldots, n$ on the curved surface. Let the unit outward normal at $\mathbf{p_i}$ be $\mathbf{n_i}$.

These sample points define a set of linear constraints on the kernel. The constraints are of the form $\mathbf{n_i}^T\mathbf{x} \leq d_i = \mathbf{n_i}^T\mathbf{p_i}, i = 1, \ldots, n$. We add a new slack variable δ to each of the constraints - $\mathbf{n_i}^T\mathbf{x} + \delta \leq d_i$ subject to $\delta > 0$. We set the objective function to maximize δ. Intuitively, the modified constraints can be viewed as a family of parallel planes (planes moving away from the normal) defining a kernel parameterized by δ. As δ increases, the kernel shrinks. If the original constraints define a valid kernel, the maximum value of δ for the new constraints is reached at a point that is maximally interior in the kernel (see Fig. 14(a)). Further, the maximum δ value also gives a lower bound on the volume of the approximate kernel, and hence an indication of the existence of a non-empty exact kernel.

6.4.2 Kernel Membership Test using Interval Arithmetic

The candidate point computed above will lie inside the exact kernel (if non-empty) provided we computed a sufficient number of sample points ($\mathbf{p_i}$'s) on the curved surface. In general, we do not know how many such points are needed; so we choose a fixed number of points. To ensure correctness, we need to check if the candidate point is actually a witness to the exact kernel. Such a witness $\mathbf{p}$ would satisfy $\mathbf{n}(\mathbf{x})^T(\mathbf{x} - \mathbf{p}) > 0$, for all points $\mathbf{x}$ on the primitive where $\mathbf{n}(\mathbf{x})$ is the normal to the surface at $\mathbf{x}$. For an algebraic surface $f(x, y, z) = 0$, the expression becomes $\bigtriangledown f^T(\mathbf{x} - \mathbf{p})$. We can derive a similar expression for parametric surfaces.

Consider a closed algebraic surface $f(\mathbf{x}) = 0$ and a point $\mathbf{p}$. The expression defining the kernel membership test is $\Gamma(f, \mathbf{p})\ :\ \bigtriangledown f^T(\mathbf{x} - \mathbf{p})$ subject to the condition that $f(\mathbf{x}) = 0$. Consider the case of a quadric surface, where $f()$ is given by $\mathbf{x}^T\mathbf{A}\mathbf{x}$. In this case, the expression for the gradient is $\mathbf{A}\mathbf{x}$. Therefore, $\Gamma(f, \mathbf{p}) = (\mathbf{x} - \mathbf{p})^T\mathbf{A}\mathbf{x} = -\mathbf{p}^T\mathbf{A}\mathbf{x}$, since $\mathbf{x}^T\mathbf{A}\mathbf{x} = 0$. In this special case, the expression to be tested is a linear expression.

Given such an expression and an axis-aligned cell, we need to verify if it is positive inside the cell. We use interval arithmetic to perform this test reliably on the interval determined by the cell. If the expression turns out to be positive inside the cell, $\mathbf{p}$ is a witness to the exact kernel and we stop subdivision. Otherwise, the cell is subdivided and the tests are repeated on each child cell.

The above approach is conservative and may result in some unnecessary subdivision. The main benefit of this method is that it is similar in flavor to the test for polyhedral primitives, and this makes it efficient. This approach requires an explicit expression for the normal field of the surface. This is a reasonable assumption because such expressions are available for the class of surfaces representable in algebraic or rational parametric form [70].

The performance of interval arithmetic depends on the degree of the expression and the tightness of the interval used. If the cells are big, interval arithmetic can return false negatives. This is primarily due to the fact that we are unable to impose the restriction that $\mathbf{x}$ should satisfy $f(\mathbf{x}) = 0$. Furthermore, if $\mathbf{p}$ lies inside the interval box on which we perform the evaluation using interval arithmetic, $\Gamma(f, \mathbf{p})$ will never be positive. We alleviate these problems as follows:

- We do not impose the restriction that $\mathbf{p}$ be inside the grid cell. This gives us candidate points that are usually far away from the grid cell.

- Along with a discretization of the original primitive, we can also precompute a voxelization. Given a grid cell, we find voxels that intersect the cell and test for $\Gamma(f, \mathbf{p})$'s sign in each of the voxel intervals (see Fig. 14(b)). If all the tests return a positive sign, the point is a valid witness to the kernel. The motivation to perform the voxelization is to generate intervals that are as close to the original surface as possible. This also serves the purpose of significantly improving the performance of the interval arithmetic step.

The combination of these two steps eliminates most of the problems mentioned earlier, and avoids unnecessary subdivision.

7. GEOMETRIC ERROR BOUND

We extend our adaptive subdivision algorithm to generate $\mathcal{A}$ with a bounded geometric error. We define the geometric error as the two-sided Hausdorff distance $H(\mathcal{A}, \mathcal{E})$, which we call the *Hausdorff error*. For a definition of Hausdorff distance, see Sec. 3.

We describe a simple extension to the adaptive subdivision algorithm that bounds the Hausdorff error. We exploit the fact that $\mathcal{E}$ is a subset of the boundaries of a set of primitives. We bound the Hausdorff distance between $\mathcal{A}$ and the boundaries of these primitives. Assume that we are given an error tolerance $\epsilon > 0$. Let $\Gamma = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ denote the set of primitives that define $\mathcal{E}$. We augment the subdivision algorithm with the following criterion:

DEFINITION 6 Hausdorff criterion ($\mathcal{C}^\epsilon$) : *Given an $\epsilon > 0$, a cell C satisfies $\mathcal{C}^\epsilon$ if*

$$H(\mathcal{A}_C, \mathcal{P}_{i,C}) < \epsilon \quad \forall \quad \mathcal{P}_i \in \Gamma \textit{ such that } \mathcal{P}_{i,C} \neq \emptyset$$

where $\mathcal{P}_{i,C} = \mathcal{P}_i \cap C$ is the restriction of $\mathcal{P}_i$ to C.

If every grid cell bounding $\mathcal{E}$ satisfies $\mathcal{C}^\epsilon$, then the Hausdorff error of $\mathcal{A}$ is also bounded by ϵ.

THEOREM 2 *If all the grid cells satisfy $\mathcal{C}^\epsilon$, then*

$$H(\mathcal{A}, \mathcal{E}) < \epsilon$$

We combine $\mathcal{C}^\epsilon$ with $\mathcal{C}^\square$ and $\mathcal{C}^\bigstar$ to define the following sampling condition:

DEFINITION 7 *A cell C satisfies $\mathcal{C}^{\square\bigstar\epsilon}$ if C satisfies both $\mathcal{C}^{\square\bigstar}$ and $\mathcal{C}^\epsilon$.*

If we apply $\mathcal{C}^{\square\bigstar\epsilon}$ during adaptive subdivision, then we obtain a reconstruction $\mathcal{A}$ with a bounded two-sided Hausdorff error as well as correct topology. See Fig. 11(c).

We compute an upper bound δ on $H(\mathcal{A}_C, \mathcal{P}_{i,C})$, and check if δ is less than ϵ. The diameter of the cell $diam(C)$ serves as a trivial upper bound. Hence a simple subdivision criterion is to subdivide a cell if its diameter is greater than ϵ. Using $diam(C)$ as an upper bound can be overly conservative: It may result in excessive subdivision for small values of ϵ. This is because the diameter of the cell is a loose upper bound on the Hausdorff distance between $\mathcal{A}_C$ and $\mathcal{P}_{i,C}$. It is possible to obtain a tighter upper bound in the case where all the primitives $\mathcal{P}_i$ are polyhedral. Since the output of Marching Cubes $\mathcal{A}_C$ is polygonal, the problem reduces to bounding the Hausdorff distance between two polygonal objects.

Hausdorff distance computation between polygonal objects is a well studied problem. Aspert et al. [71] and Guthe et al. [72] propose efficient techniques for estimating the Hausdorff distance. We provide a brief description of these techniques. Let $\mathcal{P}$ and $\mathcal{Q}$ denote two polygonal objects whose Hausdorff distance needs to be computed. We will focus on computing the forward distance $h(\mathcal{P}, \mathcal{Q})$; the backward distance can be computed similarly. For a fixed point $\mathbf{p} \in \mathcal{P}$, it is relatively easy to calculate the distance $d(\mathbf{p}, \mathcal{Q})$. However, the goal is to obtain the maximum over all points $\mathbf{p} \in \mathcal{P}$, which is difficult in practice. The above techniques resort to sampling in order to estimate $h(\mathcal{P}, \mathcal{Q})$. Each polygon of $\mathcal{P}$ is sampled, and the distance between each sample and $\mathcal{Q}$ is computed. The maximum over these distances provides an estimate of the Hausdorff distance.

We can use similar techniques to compute an upper bound δ on $h(\mathcal{P}, \mathcal{Q})$. We use the error tolerance ϵ to determine the sampling density; we recursively subdivide the polygons in P such that each polygon has a diameter less than or equal to $\epsilon/2$. The vertices of the resulting polygons provide the set S of samples. Let $\tau = \max_{\mathbf{s} \in S} d(\mathbf{s}, \mathcal{Q})$ and $\delta = \tau + \epsilon/2$. We can show that δ is an upper bound on $h(\mathcal{P}, \mathcal{Q})$. The proof is as follows.

Consider any point $\mathbf{p} \in \mathcal{P}$. Let $\mathbf{r}$ be any vertex of the polygon containing $\mathbf{p}$. Since every polygon has a diameter less than or equal to $\epsilon/2$, we have $d(\mathbf{p}, \mathbf{r}) \leq \epsilon/2$. Since $\mathbf{r}$ belongs to the set S of samples used in estimating the Hausdorff distance, we have $d(\mathbf{r}, Q) \leq \tau$. Let $\mathbf{q}$ be the point on $\mathcal{Q}$ that is closest to $\mathbf{r}$; in other words $d(\mathbf{r}, \mathcal{Q}) = d(\mathbf{r}, \mathbf{q})$. Then by triangle inequality, it follows:

$$d(\mathbf{p}, \mathbf{q}) \leq d(\mathbf{p}, \mathbf{r}) + d(\mathbf{r}, \mathbf{q}) \leq \epsilon/2 + \tau = \delta$$

This implies that $d(\mathbf{p}, \mathcal{Q}) \leq \delta$. Since this is true for any arbitrary point $\mathbf{p} \in \mathcal{P}$, it follows that $h(\mathcal{P}, \mathcal{Q}) \leq \delta$.

We can use the above technique to compute upper bounds on both forward and backward Hausdorff distances between $\mathcal{A}_C$ and $\mathcal{P}_{i,C}$. Let δ_1^i and δ_2^i denote the upper bounds on $h(\mathcal{A}_C, \mathcal{P}_{i,C})$ and $h(\mathcal{P}_{i,C}, \mathcal{A}_C)$, respectively. The maximum $\delta^i = \max(\delta_1^i, \delta_2^i)$ is an upper bound on the two-sided distance $H(\mathcal{A}_C, \mathcal{P}_{i,C})$. Let $\delta_C = \max_i \delta^i$. We can use δ_C as a threshold during adaptive subdivision: we check if $\delta_C > \epsilon$ and in that case, subdivide cell C. Using δ_C, rather than the cell diameter, can alleviate the problem of excessive subdivision.

The above technique is applicable only to polygonal primitives. In case of non-linear primitives, it is harder to bound the Hausdorff distance. One possible solution relies on computing *Sleve* to the non-linear primitive [73]. Sleve is a pair of matched triangulations that sandwiches a surface. We can exploit the fact that the two triangulations enclose the non-linear surface to compute an upper bound on the Hausdorff distance. Consider a surface $\mathcal{P}$ whose Sleve consists of triangulations $\mathcal{P}_1^s$ and $\mathcal{P}_2^s$. If both $H(\mathcal{A}, \mathcal{P}_1^s) < \epsilon$ and $H(\mathcal{A}, \mathcal{P}_2^s) < \epsilon$, then we have $H(\mathcal{A}, \mathcal{P}) < \epsilon$. We can compute the distance between $\mathcal{A}$ and the two triangulations using the techniques described for polygonal objects.

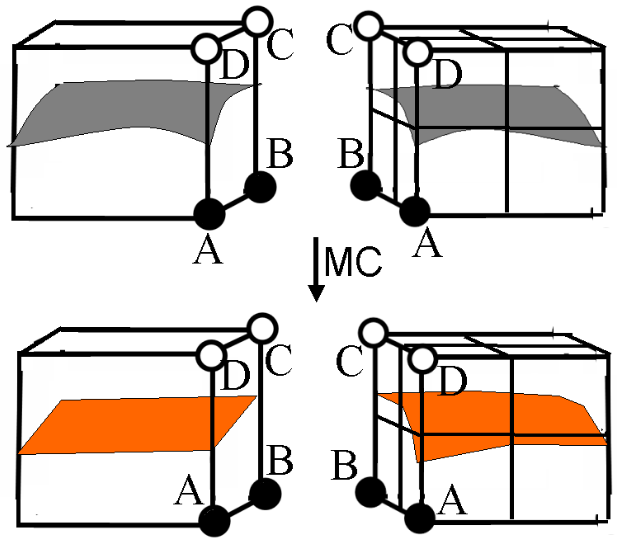


Figure 15: Isosurface Extraction on Adaptive Grids: ***This figure shows a cell at a coarse resolution sharing a face $ABCD$ with 4 cells at a finer resolution. Applying Marching Cubes to these grid cells results in a reconstruction that does not match along the common face $ABCD$.***

8. ISOSURFACE EXTRACTION ON ADAPTIVE GRIDS

The adaptive subdivision algorithm generates an adaptive volumetric grid on which we perform isosurface extraction. Applying the original Marching Cubes algorithm [2] to an adaptive grid can result in cracks in the reconstructed isosurface. A crack occurs when the reconstructed isosurface in two adjacent cells at different resolutions does not match along a shared face (see Fig. 15). This becomes an issue when defining a homeomorphism between $\mathcal{E}$ and $\mathcal{A}$. One solution this problem is to employ crack patching [46]. Crack patching modifies the extracted isosurface within the larger cell to match the extracted surface within the smaller cell. In this way, we ensure that our approximation $\mathcal{A}$ is C^0 continuous. Crack patching maintains the property that $\mathcal{A}$ restricted to the edges, faces, and voxel of the cell is a topological disk. As a result, we can still define a homeomorphism between $\mathcal{E}$ and $\mathcal{A}$, and the topological equivalence result holds.

A better alternative is to use dual methods such as Dual Contouring [4] for isosurface extraction. An important advantage of these methods is that they can handle adaptive grids easily, and can produce a reconstructed isosurface without any cracks. We provide a brief description of the Dual Contouring algorithm. See Fig. 16 for a 2D example. It operates on a grid in two steps. First, for each cell that exhibits a sign change across the edges, this method examines the set of intersection points and generates a vertex (per

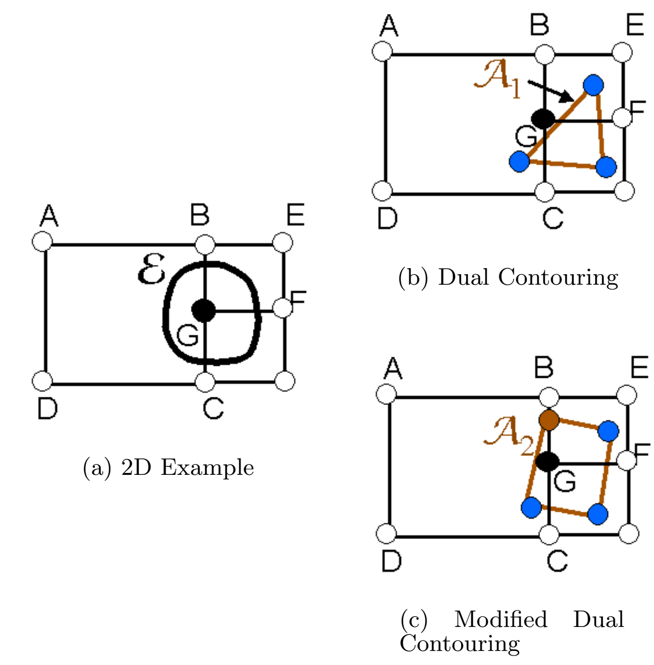


Figure 16: 2D Modified Dual Contouring: ***This figure illustrates the working of the Dual Contouring algorithm on an adaptive grid in 2D. Fig. (a) shows (in 2D) an isosurface $\mathcal{E}$ (black curve). Fig. (b) shows the output $\mathcal{A}_1$ (brown curve) of Dual Contouring algorithm. $\mathcal{A}_1$ violates a requirement of our algorithm – it does not intersect the same set of edges as $\mathcal{E}$. For example, while $\mathcal{E}$ intersects edge BG, $\mathcal{A}_1$ does not. Therefore, we apply a modification to the Dual Contouring algorithm to enforce this requirement. The output $\mathcal{A}_2$ of this algorithm, shown in Fig. (c), satisfies the requirement.***

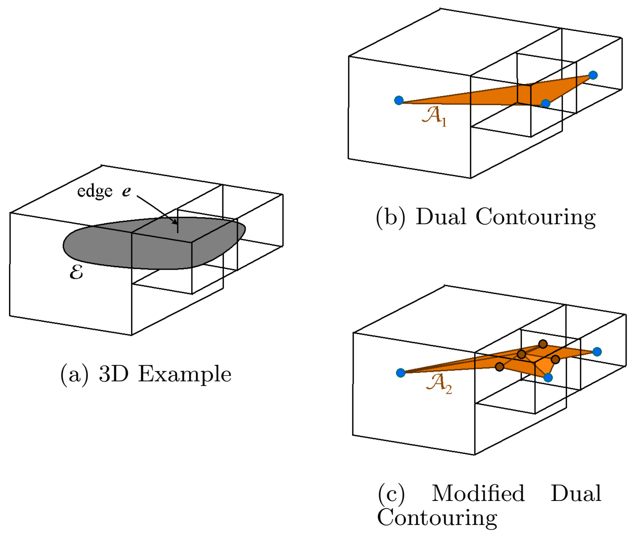


Figure 17: 3D Modified Dual Contouring: ***This figure illustrates the working of the Dual Contouring algorithm on an adaptive grid in 3D. Fig. (a) shows a portion of an isosurface $\mathcal{E}$ intersecting an edge e. Fig. (b) shows the output $\mathcal{A}_1$ of Dual Contouring algorithm. However, $\mathcal{A}_1$ violates a requirement that it should intersect the same set of edges as $\mathcal{E}$. While $\mathcal{E}$ intersects edge e, $\mathcal{A}_1$ does not. Therefore, we use a modified Dual Contouring algorithm that enforces this requirement. The output $\mathcal{A}_2$ of this algorithm, shown in Fig. (c), satisfies the requirement.***

cell) such that a quadratic error function is minimized. We refer to this vertex as the *error-minimizing* vertex. Second, for each edge that exhibits a sign change, the contouring method connects the error-minimizing vertices of the cells sharing the edge. In 2D grids, each edge is shared by two cells; hence the method outputs a line segment connecting the error-minimizing vertices of the two adjacent cells sharing the edge. In 3D uniform grids, each edge is shared by four cells; hence the method outputs a quad for each edge. In 3D adaptive grids, some of the edges in the grid may be shared by three cells; for such edges, the method outputs a triangle.

We use Dual Contouring for isosurface extraction. However, we cannot apply Dual Contouring directly – this is because the Dual Contouring does not satisfy Property 1 in Section 5.4.1. Property 1 states that the reconstructed isosurface $\mathcal{A}$ must intersect each edge, face, or voxel that exhibits a sign change. However, when applied to an adaptive grid, Dual Contouring may not satisfy this property. For example, in Fig. 16(b), edge BG exhibits a sign change, but does not intersect $\mathcal{A}$.

We correct this problem using a simple modification to the dual contouring method. We insert an additional intersection point along edge BG and connect it to the error-minimizing vertices of the adjacent cells. This is shown in Fig. 16(c). With this simple modification, both $\mathcal{E}$ and $\mathcal{A}$ exhibit identical sign configurations for every cell.

A similar modification works in 3D. See Fig. 17. It shows an edge e that intersects $\mathcal{E}$. The dual contouring method outputs a triangle that may not satisfy Property 1. We circumvent this problem by inserting an additional vertex v on e and generating a triangle fan around v. See Fig. 17(c). In order to satisfy Property 1, merely inserting v may not be enough: we also insert additional vertices on the faces that are adjacent to e. This is shown in Fig. 17(c).

With the above modification, we ensure that Dual Contouring preserves the sign configuration of the cell. It maintains the property that $\mathcal{A}$ restricted to the edges, faces, and voxel of the cell is a topological disk. As a result, the topological equivalence between $\mathcal{E}$ and $\mathcal{A}$ holds.

The above modification may increase the number of triangles in the reconstructed isosurface. However, the modifications need to be applied only when Property 1 is violated. Therefore, we can first check whether Property 1 fails, and only then apply the modification. Since typically such a violation occurs only in a small number of cells, the increase in the number of triangles is not substantial.

9. SPEEDUP TECHNIQUES

In this section we present two speedup techniques that improve the efficiency of the adaptive subdivision algorithm. Together, they improve the overall performance significantly.

9.1 Cell Culling

Since our objective is to approximate $\mathcal{E}$, it is sufficient to process only those cells that intersect $\mathcal{E}$. Based on this fact, we classify the cells in the grid into two types:

DEFINITION 8 *A cell C is a* **boundary cell** *if $\mathcal{E}_C \neq \emptyset$, and a* **non-boundary cell** *otherwise.*

We need to apply the sampling condition to only the boundary cells. While application of the sampling condition to a non-boundary cell preserves correctness of the algorithm, it is undesirable because it adds an unnecessary overhead to the algorithm. We reduce this overhead by using a technique called *cell culling*. Cell culling enables the adaptive subdivision algorithm to disregard non-boundary cells and improves the overall performance considerably.

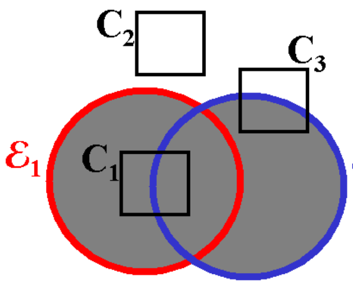


Figure 18: Cell Culling: ***This figure shows a case where $\widetilde{\mathcal{E}}$ is defined as a union of two solids, $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$. The gray shaded region shows the union. Since C_1 is contained within $\widetilde{\mathcal{E}}_1$, it is contained within the union and is a non-boundary cell. Therefore, C_1 can be disregarded. Similarly, C_2 can also be disregarded. On the other hand, cell C_3 is a boundary cell and needs to be taken into account.***

Let $\widetilde{\mathcal{E}}$ denote the solid enclosed by $\mathcal{E}$. We disregard a cell C if it is either completely inside or completely outside $\widetilde{\mathcal{E}}$. This is the main idea behind cell culling. Fig. 18 shows an example of cell culling where $\widetilde{\mathcal{E}}$ is defined as a union of two solids, $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$.

Cell culling is based on the voxel intersection test (Lem. 4). We use the following property: If the signed max-norm distance to $\mathcal{E}$ at the center of a cell C is greater than half the voxel size, then we can disregard C as it is guaranteed not to intersect $\mathcal{E}$. As explained in Sec. 6.1, we do not compute the exact signed distance D^s_∞ to $\mathcal{E}$; it is sufficient to compute a conservative estimate $\widetilde{D}^s_\infty$ of the distance. Lemma 5 ensures that the absolute value of $\widetilde{D}^s_\infty$ is less than the absolute value of D^s_∞. Therefore, we can use $\widetilde{D}^s_\infty$ to perform cell culling while preserving correctness. The condition for cell culling is as follows:

Cell Culling Condition: Let C be a cell with center $\mathbf{o}$ and length l.

$$\text{If} \quad |\widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E})| > l/2 \quad \text{then } C \text{ is a non-boundary cell.}$$

We show an application of this condition to cell C_1 in the union example (Fig. 18). Let d_1 and d_2 denote the signed max-norm distances from $\mathbf{o}$ to $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. Since cell C_1 lies completely within $\widetilde{\mathcal{E}}_1$, we have $d_1 < -l/2$. This implies:

$$\widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}) = \min(d_1, d_2) < -l/2$$
$$\implies |\widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E})| > l/2$$

Therefore C_1 satisfies the condition for cell culling and we can rule out C_1 from further consideration. Similarly, we can employ cell culling for intersection and difference operations. It can also be used for a sequence of Boolean operations.

Cell culling is conservative: While every cell discarded by cell culling is a non-boundary cell, the converse is not true – it may not eliminate all the non-boundary cells. This is because we use a conservative estimate $\widetilde{D}^s_\infty$ of the signed max-norm distance. Due to conservativeness of cell culling, we may unnecessarily process some non-boundary cells; however, this does not affect the correctness of the algorithm.

9.2 Expression Simplification

The adaptive subdivision algorithm processes each cell in the grid independently. Within a cell C, it needs to consider only $\mathcal{E}_C$ – the portion of $\mathcal{E}$ that is contained within C. It may be possible to define $\mathcal{E}_C$ by simplifying the Boolean expression of $\mathcal{E}$. This is the main idea behind *expression simplification.*

$\widetilde{\mathcal{E}}$ denotes the solid enclosed by $\mathcal{E}$. By a slight abuse of notation, we will use $\widetilde{\mathcal{E}}$ to also refer to the Boolean expression associated with the solid $\widetilde{\mathcal{E}}$.

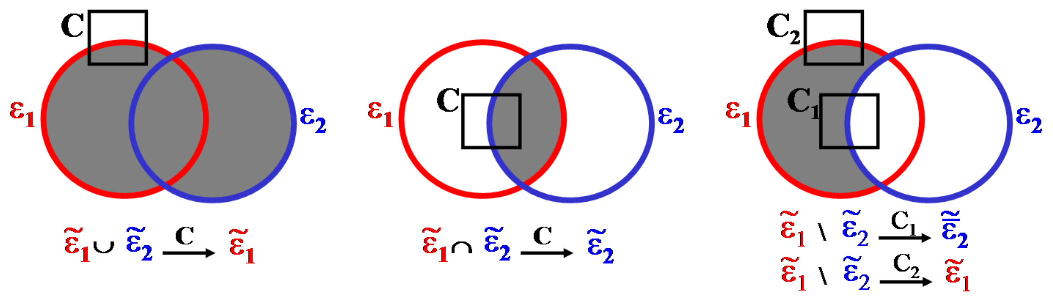
Consider the case of a union $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$, as shown in Fig. 19(a). Because cell C lies completely outside $\widetilde{\mathcal{E}}_2$, C is not influenced by $\widetilde{\mathcal{E}}_2$. Consequently, we can simplify the Boolean expression by getting rid of $\widetilde{\mathcal{E}}_2$ from the expression. This produces a simplified expression $\widetilde{\mathcal{E}}_1$ for cell C. We refer to this step as expression simplification. It can also be used for intersection and difference operations as well as a sequence of Boolean operations. See Figs. 19(b), 19(c).

In order to simplify an expression of $\widetilde{\mathcal{E}}$ w.r.t a cell C, we need to determine if C lies completely outside or completely inside the solids corresponding to the sub-expressions of $\widetilde{\mathcal{E}}$. For example, in the case of a union $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$, we need to test if C is completely outside $\widetilde{\mathcal{E}}_1$ or $\widetilde{\mathcal{E}}_2$. We perform these tests using max-norm distance computation. Given a primitive $\mathcal{P}$ and a cell C, we use the following conditions:

$$\begin{array}{llll} \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{P}) > l/2 & \text{then} & \text{C lies completely outside } \widetilde{\mathcal{P}} \\ \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{P}) < -l/2 & \text{then} & \text{C lies completely inside } \widetilde{\mathcal{P}} \end{array}$$

where $\mathbf{o}$ and l are the center and length of cell C respectively.

Below we provide the conditions for expression simplification. We provide a condition for each Boolean operation: union, intersection, difference, and complement. $\mathcal{A} \xrightarrow{C} \mathcal{B}$ denotes a simplification of a Boolean expression $\mathcal{A}$ into an expression $\mathcal{B}$ in cell C.



(a) Union (b) Intersection (c) Difference

Figure 19: Expression Simplification: ***Figs. (a), (b), & (c) shows examples of expression simplification for union, intersection, and difference operations respectively. In each figure, the gray shaded region shows the final solid obtained by performing the Boolean operation.***

1. **Union:** $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2$
   We can simplify an expression involving union operation when C lies completely outside either $\widetilde{\mathcal{E}}_1$ or $\widetilde{\mathcal{E}}_2$.
   $$\begin{array}{lll} \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_1) > l/2 & \text{then} \quad \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_2 \\ \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_2) > l/2 & \text{then} \quad \widetilde{\mathcal{E}}_1 \cup \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_1 \end{array}$$

2. **Intersection:** $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2$
   We can simplify an expression involving intersection operation when C lies completely inside either $\widetilde{\mathcal{E}}_1$ or $\widetilde{\mathcal{E}}_2$.
   $$\begin{array}{lll} \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_1) < -l/2 & \text{then} \quad \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_2 \\ \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_2) < -l/2 & \text{then} \quad \widetilde{\mathcal{E}}_1 \cap \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_1 \end{array}$$

3. **Difference:** $\widetilde{\mathcal{E}} = \widetilde{\mathcal{E}}_1 \setminus \widetilde{\mathcal{E}}_2$
   We can simplify an expression involving difference operation in either of two cases: (1) C lies completely inside $\widetilde{\mathcal{E}}_1$; (2) C lies completely outside $\widetilde{\mathcal{E}}_2$.
   $$\begin{array}{lll} \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_1) < -l/2 & \text{then} \quad \widetilde{\mathcal{E}}_1 \setminus \widetilde{\mathcal{E}}_2 \xrightarrow{C} \overline{\widetilde{\mathcal{E}}_2} \\ \text{If} & \widetilde{D}^s_\infty(\mathbf{o}, \mathcal{E}_2) > l/2 & \text{then} \quad \widetilde{\mathcal{E}}_1 \setminus \widetilde{\mathcal{E}}_2 \xrightarrow{C} \widetilde{\mathcal{E}}_1 \end{array}$$

4. **Complement:** $\widetilde{\mathcal{E}} = \overline{\widetilde{\mathcal{E}}_1}$
   We can simplify a complement of an expression $\widetilde{\mathcal{E}}_1$ by simplifying $\widetilde{\mathcal{E}}_1$.
   $$\text{If} \quad \widetilde{\mathcal{E}}_1 \xrightarrow{C} \widetilde{\mathcal{E}}_2 \quad \text{then} \quad \overline{\widetilde{\mathcal{E}}_1} \xrightarrow{C} \overline{\widetilde{\mathcal{E}}_2}$$

Fig. 19 shows several examples of expression simplification.

We perform expression simplification in a top-down manner during adaptive subdivision. With every cell C, we maintain the corresponding Boolean expression $\widetilde{\mathcal{E}}_C$. The root cell of subdivision is associated with the original Boolean expression $\widetilde{\mathcal{E}}$. Each time we subdivide a cell C into a set of children cells $C_i, i = 1, \ldots, k$, we simplify $\widetilde{\mathcal{E}}_C$ w.r.t the children cells, i.e.,

$$\widetilde{\mathcal{E}}_C \xrightarrow{C_i} \widetilde{\mathcal{E}}_{C_i}$$

This gives us a Boolean expression $\widetilde{\mathcal{E}}_{C_i}$ for each child cell C_i.

Alg. 1 shows pseudo-code for the complete adaptive subdivison algorithm including expression simplification. Expression simplification can reduce the original Boolean expression considerably. As the adaptive subdivision progresses, the expressions corresponding to the subdivided cells become progressively simpler. Typically, the simplified expression requires only a small number of Boolean operations. This improves the performance of the overall algorithm considerably.

Consider a situation where $\widetilde{\mathcal{E}}$ is expressed as a union of a high number of primitives $\widetilde{\mathcal{E}} = \cup_i \widetilde{\mathcal{P}}_i$. For a cell C we can simplify $\widetilde{\mathcal{E}}$ by eliminating all the primitives that lie completely outside C. Typically the resulting simplified Boolean expression has only a small number of primitives. This type of union operation arises frequently in many other problems such as Minkowski sum and swept volume computation.

10. PERFORMANCE

In this section, we analyze the performance of our algorithm. The total time taken by the algorithm is the sum of the times taken by the sampling and reconstruction steps.

- **Sampling:** This is the dominant step of our algorithm. The total time taken by this step is given by $T_S = \sum_{C \in \mathcal{G}} T(C)$ where $\mathcal{G}$ denotes the volumetric grid and $T(C)$ is the time spent on a single cell C. We will provide a bound on $T(C)$ below.

- **Reconstruction:** MC-like methods spend $O(1)$ time on each cell of the grid. Therefore, the time complexity of this step is $O(N)$ where N is the number of cells in the grid.

We analyze the *cell complexity*, T(C), the time taken to process a single grid cell C. $T(C)$ is the sum of the time taken by the complex cell and star-shaped tests. Below we bound the time taken by each test separately. We begin by analyzing the time complexity of the two tests on a single primitive. We consider two cases – depending on whether the primitive is polyhedral or algebraic.

10.0.1 *Polyhedral Primitive*

Consider a polyhedral primitive with n polygons. We define the size of the primitive to be n.

- **Complex Cell Test:** The complex cell test requires two types of computations:
  - **Sign query:** Determining the sign of a point takes $O(n)$ time.
  - **Cell intersection:** This requires directed distance and max-norm distance computation. Each of them takes $O(n)$ time.
- **Star-shaped test:** For a polyhedral primitive, the star-shaped test reduces to linear programming. We combine the linear constraints defined by each polygon of the primitive, and solve the resulting linear program. This step takes $O(n)$ expected time.

10.0.2 *Algebraic Primitive*

We assume an algebraic primitive with a fixed degree.

- **Complex Cell Test:**
  - **Sign query:** Computing a sign for an algebraic primitive requires evaluating a polynomial function. We consider the time taken by this step to be $O(1)$.
  - **Cell intersection:** We use directed and max-norm distance computation for low degree algebraic primitives and interval arithmetic for higher order primitives. For an algebraic primitive, both directed and max-norm distance computation reduce to solving a univariate polynomial equation. We consider the time taken by this step to be $O(1)$. Performing interval arithmetic on an algebraic primitive reduces to evaluating a polynomial function. We assume this step also takes $O(1)$ time.
- **Star-shaped test:** Our star-shaped test for an algebraic primitive uses a combination of linear programming and interval arithmetic. The linear programming step requires a discretization of the algebraic primitive. We assume that each algebraic primitive $\mathcal{P}$ has an associated discretization (see Sec. 6.4). Let n denote the number of points in the discretization. We define the size of the algebraic primitive to be n. The linear programming step takes an $O(n)$ expected time. We assume the interval arithmetic step takes $O(1)$ time.

Therefore, applying the complex cell and star-shaped tests to a single primitive – polyhedral or algebraic – takes $O(n)$ expected time, where n is the size of the primitive.

Our algorithm performs the complex cell and star-shaped tests on $\mathcal{E}$. Specifically, within a cell C, we perform them on $\mathcal{E}_C$, whose associated Boolean expression $\widetilde{\mathcal{E}}_C$ is obtained by performing expression simplification. Let $\Gamma_C = \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}\}$ denote the set of primitives in the Boolean expression $\widetilde{\mathcal{E}}_C$. In order to perform the complex cell and star-shaped tests on $\mathcal{E}_C$, we need to take into account every primitive in Γ_C. For example, in order to compute the sign of a point w.r.t $\mathcal{E}_C$, we need to compute its sign w.r.t every primitive in Γ_C. Similarly, in case of star-shaped test, we combine the linear constraints derived from the all the primitives in Γ_C. Therefore, the complex cell and star-shaped tests take $\sum_{j=1}^{k} O(n_j)$ time where n_j is the size of $\mathcal{P}_{i_j}$. This means the cell complexity T(C) is given by:

$$T(C) = \sum_{j=1}^{k} O(n_j) = O(n)$$

where $n = \sum_{j=1}^{k} n_j$.

11. IMPLEMENTATION & APPLICATIONS

In this section, we describe the implementation of our algorithm and highlight three different applications: Boolean operations, simplification, and remeshing.

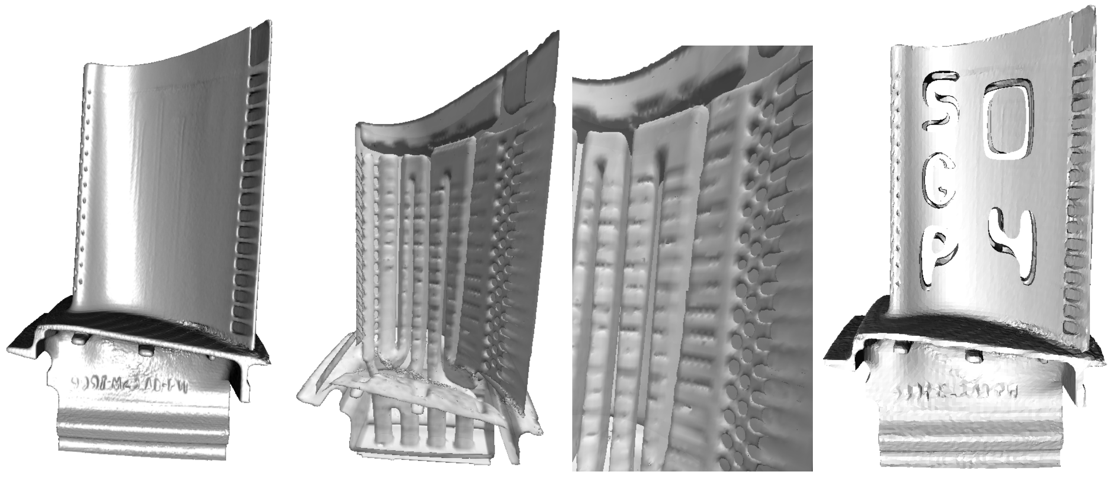
We used C++ programming language with the GNU g++ compiler under Linux operating system. We demonstrate the performance of our algorithm on many complex models. Table 1 highlights the performance of our algorithm on these models. All execution times are on a 2 GHz Pentium IV PC with a GeForce 4 graphics card and 1 GB RAM.

In all our applications, we first generate an adaptive octree grid using our adaptive subdivision algorithm. We then compute a polyhedral approximation to the boundary of the final solid using a modified Dual Contouring algorithm (See 8). The reconstructed surface is a manifold. We used a freely available linear programming package, *QSOPT* [68], to implement the star-shaped test.

11.1 Boolean operations

Figure 21(left) shows the reconstruction of the final surface generated by our algorithm on the dragon model. The solid is defined as a union of two dragons, each with over 870K triangles. It took 95 secs to compute the approximate union. The second example is obtained by performing 5 difference operations on the Turbine Blade model (see rightmost image in Fig. 20). The model has more than 1.7 million triangles. The final surface has multiple components and a higher genus. Our algorithm computes the boundary in 116 secs. Figure 21(right) highlights application of our algorithm to perform Boolean operations on curved primitives. It shows a challenging scenario where we perform 100 difference (Boolean) operations between a polyhedron and 100 ellipsoids. The resulting surface has a complex topology with numerous small holes; it has a genus of 208. Our algorithm took 16 secs to generate an approximation with the correct topology.

The problem of Minkowski sum computation arises in many applications including solid modeling, digital geometry processing, robotics, dynamic simulation and computer animation. The Minkowski sum of two sets P and Q is the set of points $\{\boldsymbol{p} + \boldsymbol{q} \mid \boldsymbol{p} \in P,\ \boldsymbol{q} \in Q\}$. Minkowski sum computation of polyhedral models can be reduced to a union operation. We have applied our isosurface extraction algorithm to approximate the resulting union thereby producing an accurate Minkowski sum approximation. Details of the Minkowski sum approximation algorithm and its application to robot motion planning can be found in [74–76].



(a) Turbine Model (b) Turbine Interior (c) Zoomed View (d) Difference Operations

Figure 20: Simplification and Boolean operations: ***This*** $1.7M$ ***triangle model of a turbine has a high genus and many features in the interior. We highlight the application of our algorithm to simplification and Boolean operations on this complex model. The simplified model of the turbine has*** $511K$ ***triangles and we show a zoomed view in the center-right image. We perform five difference (Boolean) operations on the turbine model and reconstruct the boundary of the final solid. Our algorithm produces a geometrically close and topology preserving approximation to the final solid. Overall, our algorithm can perform such geometric computations on complex models in tens of seconds and give rigorous guarantees in terms of preserving the topology of the final surface.***

11.2 Topology Preserving Volumetric Simplification

Model simplification algorithms produce a lower polygon count approximation of a polygonal object that preserves the shape or appearance of the object. Simplification techniques have been used for fast display and simulation. In order to compute a drastic simplification for interactive visualization, many algorithms do not preserve the surface topology. On the other hand, preserving topology during simplification is important for applications like CAD, medical visualization, and molecular modeling. Volumetric algorithms have been proposed for model simplification [77,78]. These algorithms are fast in practice and are applicable to all kind of models. However, none of these algorithms give rigorous guarantees on preserving the surface topology.

We compute a discrete sampling of the distance field by applying our adaptive subdivision algorithm. Different levels of detail are generated by changing the value of the two-sided Hausdorff error tolerance. The reconstruction algorithm generates an isosurface that has the same topology as the original model. Our metric of Hausdorff error can be easily combined with other metrics such as curvature, quadric error, etc, to guide the simplification.

Figure 22 shows our simplification algorithm applied to a medical dataset, a $650K$ triangle *Hand* model. This model has a number of topological features that need to be preserved in order to maintain the anatomical structure of the hand. Figs. 22(b) ($27K$ triangles) and 22(c) ($58K$ triangles) show a coarse and a fine approximation, respectively, of the original model. The coarse approximation was computed by applying our adaptive subdivision algorithm without imposing any Hausdorff error bound. The resulting grid is shown in Fig. 22(d). Fig. 22(e) shows a closeup view of part of the finer approximation. It took 36 secs to generate the approximation. Figure 20 shows our simplification algorithm applied to the Turbine Blade model. This model has a high genus and many tunnels in the interior. It took 110 secs to generate a simplified model with $511K$ triangles. Note that our method preserves the complex topological features in the simplified model.

Some prior surface simplification methods can be adapted to perform topology preserving simplification [79,80]. However, one limitation of these approaches is that they need to perform global tests to avoid surface self-intersections, which can result in considerable overhead. On the other hand, our algorithm is guaranteed not to produce self intersections.

Our subdivision criterion ensures that the isosurface is a topological disk within each grid cell by satisfying the complex cell and star-shaped criteria. In applications such as simplification and remeshing, a simpler test exists. In these applications, we have a polygonization of the original surface. We can ensure the topological disk property by computing the Euler characteristic and testing if it is equal to 1. However, in case of Boolean operations, we do not have a polygonization of the final surface (in fact, our goal is to compute a polygonization). Consequently, the test based on

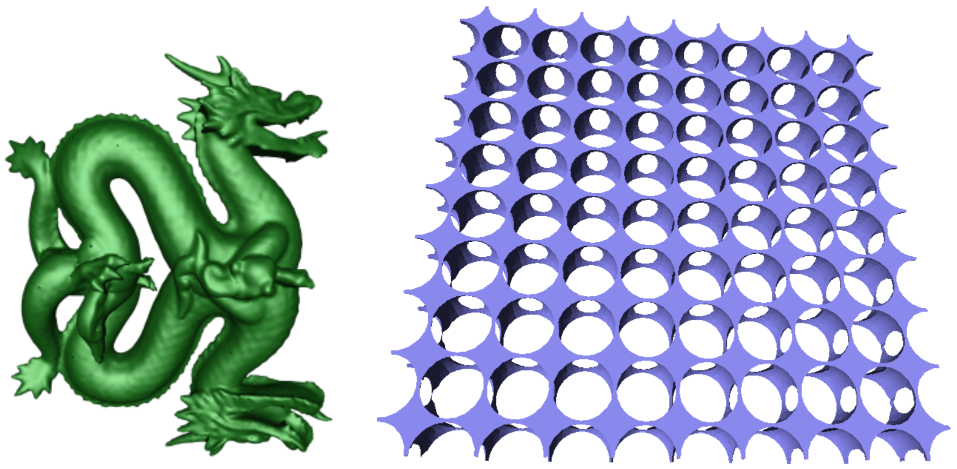


Figure 21: Boolean operations on complex models and curved primitives: ***The left figure shows the result of the union of two dragons. Each dragon is represented using*** $850K$ ***triangles. Our algorithm computes a topology preserving approximation of the final boundary. It took*** 95 ***secs to compute an approximation with*** $118K$ ***triangles at a relative Hausdorff error bound of*** $1/128$***. The relative Hausdorff error is defined as the ratio of the absolute Hausdorff error to the maximum length of a "tight" axis-aligned bounding box around the object. The right figure shows the result of*** 100 ***difference operations between a polyhedron and*** 100 ***ellipsoids. The resulting surface has a complex topology; it has a genus of*** 208***. Our algorithm took*** 16 ***secs to generate an approximation with the correct topology at a relative Hausdorff error bound of*** $1/64$***.***

	Combi. Complexity		Performance (secs)			
Model	Input	Output	Complex	Star-shaped	Subdivision	Total
Hand Simplification (Fig. 22)	654,666	58,966	4.3	8.1	23	36
Turbine Simplification (Fig. 1)	1,765,388	511,182	14.1	31.3	65.8	110
Turbine Blade Boolean (Fig. 1)	1,765,388	319,074	16.8	29.1	70.4	116
Union of Dragons (Fig. 21)	1,714,920	118,214	10.2	21.1	63.6	95
Curved Boolean (Fig. 21)	-	57,286	5.4	5.1	5.5	16
Brake Hub Remeshing (Fig. 23)	14,208	7,056	0.38	0.55	0.90	1.85
CAD model Remeshing (Fig. 23)	41,152	16,524	0.58	0.81	1.41	2.8

Table 1: Performance: ***This table highlights the complexity of our input models and performance of our algorithm. The columns on the left shows the the triangle count of the input and triangle count in our reconstruction. The columns on the right show the cumulative time taken by the complex cell test, star-shaped test and adaptive subdivision over all the grid cells. The rightmost column shows the total execution time.***

Euler characteristic does not work for Boolean operations.

11.3 Topology Preserving Remeshing

Volumetric approaches have been used for remeshing of polygonal models [3, 78]. In many applications, the polygonal models can have triangles with bad-aspect ratios. The goal is to compute a valid manifold representation of the underlying closed solid and ensure that the resulting triangles have a good aspect ratio. The mesh generated after remeshing can be used for multiresolution analysis or simplification.

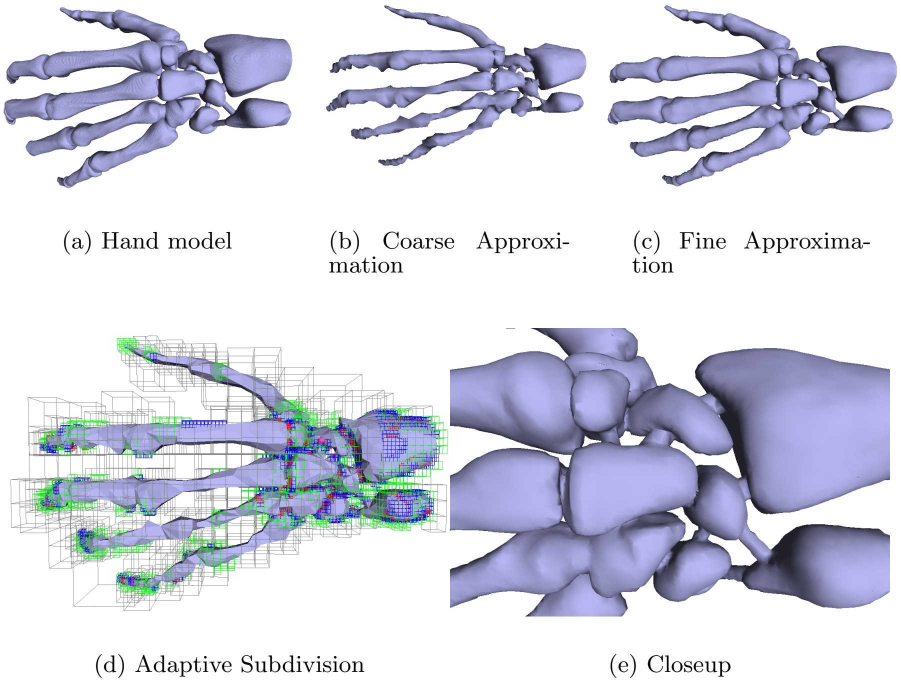
Earlier algorithms generated a volumetric representation by sampling the distance field on a uniform grid [3], or with a simplified topology [78]. However, these methods provide no guarantees on the genus or the number of connected components. We have applied our subdivision algorithm to compute a topology preserving remeshing of CAD models. Fig. 23 shows some of our results. The Euler characteristic test (see Section 11.2) could also be used for remeshing.

11.4 Discussion

Table 1 highlights the performance of our algorithm on these models. It also provides a breakup of the total time spent in performing the complex cell test, star-shaped test, and adaptive subdivision. This corresponds to the time taken to "push" the input triangles down the octree data structure. For each octree cell C, we perform expression simplification to obtain a smaller set of primitives Γ_C that define the isosurface within the cell. We maintain a list T_C of triangles that belongs to the primitives in Γ_C. We maintain the set T_C in cell C. As we subdivide C, we partition it into 8 children C_i and compute their triangle lists T_{C_i} similarly. For large input models, this takes substantial fraction of the total time.

12. ANALYSIS

In this section, we analyze the behavior of our adaptive subdivision algorithm and provide a sufficient condition for its termination. We show that under certain conditions, once a cell becomes smaller than a certain size, it will eventually satisfy both complex cell and star-shaped criteria.


(a) Hand model (b) Coarse Approximation (c) Fine Approximation

(d) Adaptive Subdivision (e) Closeup

Figure 22: Topology-Preserving Simplification: ***Figs (a),(b),(c) show the original model along with a coarse and fine approximation generated using our topology preserving simplification algorithm. The original model has*** $654K$ ***triangles, while the two approximations consist of*** $27K$ ***and*** $58K$ ***triangles respectively at relative Hausdorff error bounds of*** 1 ***and*** $1/128$***. Fig (d) shows the adaptive grid generated for the coarse approximation. The colors, green, blue, and red in that order, indicate the increasing level of subdivision. Fig (e) shows a closeup view of a part of the fine approximation. It highlights the features in the original model and our algorithm is able to reconstruct all these features accurately. It took*** 36 ***secs to generate the approximation.***

Our analysis assumes that $\mathcal{E}$ is a smooth surface – twice-differentiable manifold. This assumption may not hold when $\mathcal{E}$ is defined using Boolean operations. We note that we make this assumption only to simplify the analysis of the algorithm; the algorithm itself does not make this assumption.

We perform the analysis in two stages. In the first stage, we analyze both complex cell and star-shaped criteria in terms of the Gauss map of $\mathcal{E}$ within the cell. The Gauss map $\mathbb{G}$ of a smooth surface $\mathcal{S}$ in $\mathbb{R}^3$ is a set-valued function from $\mathcal{S}$ to the unit sphere $\mathbb{S}^2$, which assigns to each point $\mathbf{p} \in \mathcal{S}$ the outward unit normal to $\mathcal{S}$ at $\mathbf{p}$. We use the Gauss map of $\mathcal{E}$ to provide sufficient conditions for when a cell will satisfy both the complex cell and star-shaped criteria. There are two separate conditions – one for complex cell criterion and another one for star-shaped criterion.

In the second stage, we relate the Gauss map conditions to the notion of *local feature size* (LFS), proposed by Amenta and Bern [81]. The local feature size LFS($\mathbf{p}$) at a point $\mathbf{p}$ on a surface $\mathcal{S}$ is defined as the least distance of $\mathbf{p}$ to the *medial axis* of $\mathcal{S}$ (Fig. 26). The medial axis of $\mathcal{S}$ is the set of points with more than one closest point on $\mathcal{S}$. Amenta and Bern used the LFS to design a sampling condition for topology preserving reconstruction using Delaunay triangulation. They showed that it suffices to choose a set of samples on $\mathcal{S}$ such that every point $\mathbf{p} \in \mathcal{S}$ has a sample at a distance less than a fraction of LFS($\mathbf{p}$).

We extend the above definition to define the LFS of a grid cell. We then show that if a cell is smaller than a certain fraction of its LFS, then it will meet the Gauss map conditions thus satisfying both the complex cell and star-shaped criteria. This provides a bound on the size of a cell relative to its LFS. In the absence of degeneracies, the adaptive subdivision algorithm will terminate once all the cells become smaller than a fraction of their respective LFS. We use the LFS to also characterize some of the degenerate cases of our algorithm. This will be discussed in Sec. 13.

12.1 Preliminaries

We use the following notation in this section. $d(\mathbf{p}, \mathbf{q})$ denotes the Euclidean distance between $\mathbf{p}$ and $\mathbf{q}$. $\|\vec{\mathbf{u}}\|$ denotes the length of a vector $\vec{\mathbf{u}}$. $\angle \mathbf{m}, \mathbf{n}$ denotes the angle between two vectors $\mathbf{m}$ and $\mathbf{n}$.

Let $\mathbf{o}$ be the origin of $\mathbb{R}^d$ where $d = 2, 3$. Let $\mathbf{x}_i^+, \mathbf{x}_i^-, i = 1, \ldots, d$ denote the principal directions of $\mathbb{R}^d$. $\mathbf{x}_i^+$ is a unit vector in $\mathbb{R}^d$ whose ith component is 1 and rest of the components are 0. $\mathbf{x}_i^-$ is equal to $-\mathbf{x}_i^+$. By a *principal hemisphere*, we mean a hemisphere whose axis is along one of the principal directions. For example, a principal hemisphere with

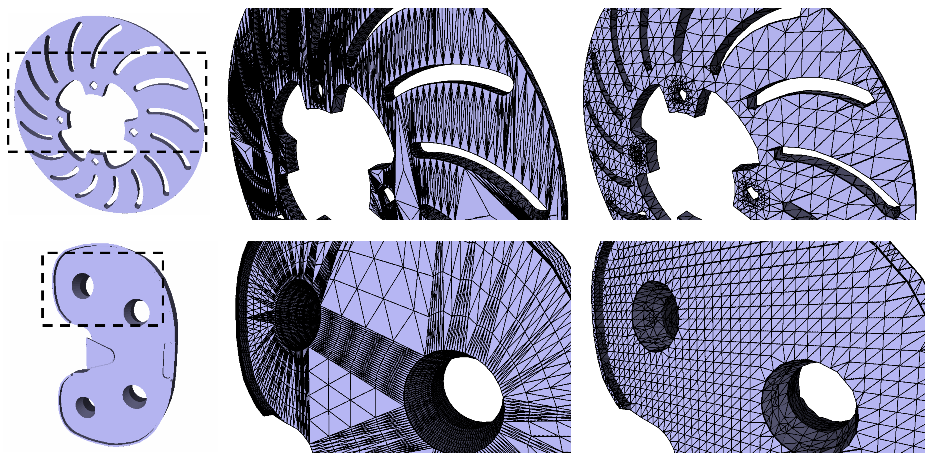


Figure 23: Remeshing of brake hub and a CAD model*: The left column of images shows a brake hub and a CAD model, which have many "skinny" triangles with poor aspect ratios (center column) The right column of images show the triangulation of the remeshed model produces by our algorithm. The original brake hub mod el has* $14K$ *triangles. Our algorithm took* 1.85 *secs to perform remeshing and generate a mesh with* $7K$ *triangles at a Hausdorff error of* $1/32$*. The CAD model has* $41K$ *triangles. We performed remeshing and generated a mesh with* $16K$ *triangles at a relative Hausdorff error of* $1/32$ *in* 2.8 *secs.*

$\mathbf{x}_1+$ axis in $\mathbb{R}^3$ is defined as:

$$\{\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \mid \|\mathbf{ox}\| = 1, x \geq 0\}$$

A right circular cone with an *axis* $\vec{\mathbf{u}}$ and a *half-angle* θ is defined as the set of points $\{\mathbf{x} \in \mathbb{R}^d \mid \angle\vec{\mathbf{u}}, \mathbf{ox} = \theta\}$.

Given a cell C, we define a Gauss map of restrictions of $\mathcal{E}$ to the voxel and faces of C.

- For a voxel ϑ of C, define $\mathbb{G}_\vartheta$ as a set valued function from $\mathcal{E}_\vartheta$ to the unit sphere $\mathbb{S}^2$, which assigns to each point $\mathbf{p} \in \mathcal{E}_\vartheta$ the outward unit normal to $\mathcal{E}_\vartheta$ at $\mathbf{p}$.
- For a face f of a cell, we consider the restriction of $\mathcal{E}$ to the plane Π_f containing f. $\mathcal{E}_f$ is treated like a curve in $\mathbb{R}^2$. We use a 2D definition of Gauss map. Define $\mathbb{G}_f$ as a set valued function from $\mathcal{E}_f$ to the unit circle $\mathbb{S}^1$, which assigns to each point $\mathbf{p} \in \mathcal{E}_f$ the outward unit normal (defined in 2D) to $\mathcal{E}_f$ at $\mathbf{p}$.

We use the term Gauss map to also refer to the image of the Gauss map.

12.2 Gauss Map Condition for Complex Cell Criterion

In Sec. 5, we had defined a cell to be complex if it has a complex voxel, a complex face, a complex edge, or an ambiguous sign configuration. There is some redundancy in this definition: a complex voxel is allowed to have complex faces or complex edges. We now provide an equivalent definition without the redundancy. We refer to a voxel (face) as *strongly complex* if it is complex and none of its faces (edges) intersect $\mathcal{E}$. Unlike a complex voxel, a strongly complex voxel cannot have a complex face or a complex edge. A strongly complex voxel always contains a closed component of $\mathcal{E}$ in its interior. We define a cell to be complex if it has a *strongly complex* voxel, a *strongly complex* face, a complex edge, or an ambiguous sign configuration. This definition of a complex cell is equivalent to the one presented in Sec. 5.

We now present a Gauss map condition for when a cell satisfies the complex cell criterion. First, we introduce a definition.

DEFINITION 9 *Let c be a voxel/face of a cell. Let $\vec{\mathbf{u}}$ be a unit vector and θ be a value such that $0 < \theta \leq \pi/2$.*

1. *We say c* is normal-bounded w.r.t $(\vec{\mathbf{u}}, \theta)$ *if*
   (a) $\mathcal{E}_c = \emptyset$ *or*
   (b) $\angle\vec{\mathbf{u}}, \mathbf{n} < \theta \ \forall \mathbf{n} \in \mathbb{G}_c(\mathcal{E}_c)$

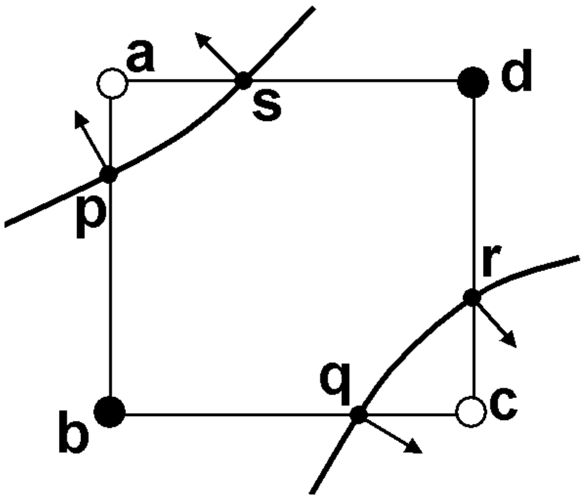
   This means that the Gauss map $\mathbb{G}_c(\mathcal{E}_c)$ lies within a right circular cone whose axis is $\vec{\mathbf{u}}$ and whose half-angle is θ.

2. *We say c* is normal-bounded by θ *if*
   (a) $\mathcal{E}_c = \emptyset$ *or*
   (b) *For any two vectors $\mathbf{n}_1, \mathbf{n}_2 \in \mathbb{G}_c(\mathcal{E}_c)$, we have $\angle\mathbf{n}_1, \mathbf{n}_2 < \theta$.*

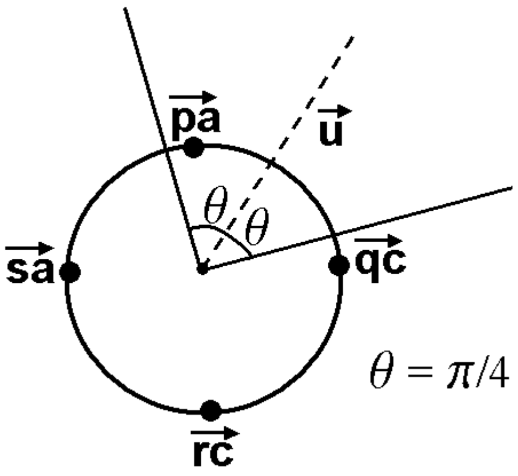
THEOREM 3 *Consider a cell C. If the following conditions hold:*

1. *The voxel of C is normal-bounded w.r.t $(\vec{\mathbf{u}}, arcsin(1/\sqrt{(}3)))$ for some unit vector $\vec{\mathbf{u}} \in \mathbb{S}^2$.*
2. *Every face of C is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/4)$ for some unit vector $\vec{\mathbf{u}} \in \mathbb{S}^1$.*
3. *No edge e of C is complex.*

then C is not complex.



(a) Topological Ambiguity



(b) Supports Proof of Theorem 3

Figure 24: ***This figure supports the proof of Theorem 3. Fig. (a) shows a face with ambiguity. In Fig. (b), the unit vectors along* pa*,* sa*,* qc *and* rc *have been mapped onto the the unit circle. The figure shows a cone with an axis* $\vec{\mathbf{u}}$ *and half-angle* θ*. We show that it is not possible for the Gauss map of the curve to lie within this cone.***

Proof: Let c be a voxel/face of C satisfying (1) or (2). We show it cannot be strongly complex and cannot have an ambiguity. Therefore, C cannot be complex.

Consider a face f satisfying (2). If f is strongly complex, then there exists a closed component of $\mathcal{E}_f$ within f. Therefore $\mathbb{G}_f(\mathcal{E}_f)$ would span the entire circle $\mathbb{S}^1$. This contradicts the fact that f is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/4)$. Therefore, f cannot be strongly complex.

We now prove that f cannot have a face ambiguity. Let the vertices of f be **a**, **b**, **c**, and **d**, as shown in Fig. 24(a). Suppose f has an ambiguity. Without loss of generality, assume that **a** and **c** have a positive sign, while **b** and **d** have a negative sign. Each of the four edges intersect $\mathcal{E}$. Let the intersection points on **ab**, **bc**, **cd**, **da** be **p**, **q**, **r**, **s** respectively. If an edge has more than one intersection point, then choose the one that is closest to the positive endpoint of the edge. Let $\mathbf{n_x}$ denote the unit normal to $\mathcal{E}_f$ at $\mathbf{x} \in \mathcal{E}_f$. Assume that the unit normal points "outwards", i.e., towards a point with a positive sign.

Since f is ambiguous, it has two boundary curves. Without loss of generality, assume **p** and **s** belong to one of the boundary curves, while **q** and **r** belong to the other. See Fig. 24(a).

Because **a** and **c** have a positive sign and the normals to $\mathcal{E}_f$ point outwards, the normals have to lie within a range specified by the following:

$$\begin{aligned} \angle \mathbf{n_p}, \vec{\mathbf{pa}} < \pi/2 \quad &\wedge \quad \angle \mathbf{n_s}, \vec{\mathbf{sa}} < \pi/2 \\ \angle \mathbf{n_q}, \vec{\mathbf{qc}} < \pi/2 \quad &\wedge \quad \angle \mathbf{n_r}, \vec{\mathbf{rc}} < \pi/2 \end{aligned} \tag{5}$$

where $\vec{\mathbf{pa}}, \vec{\mathbf{sa}}, \vec{\mathbf{qc}}, \vec{\mathbf{rc}}$ denote the unit vectors along **pa**, **sa**, **qc**, **rc** respectively. Under the Gauss map $\mathbb{G}_f(\mathcal{E}_f)$, these unit vectors will map to *extremal points* in $\mathbb{S}^1$, i.e. points belonging to the set $\{(1,0),(0,1),(-1,0),(0,-1)\}$. See Fig. 24(b).

Since f is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/4)$, $\mathbb{G}_f(\mathcal{E}_f)$ lies within a right circular cone C centered at the origin and with a half-angle $\pi/4$. The portion of $\mathbb{S}^1$ that lies within such a cone is always a subset of a principal hemisphere. Let $\mathbb{H}_1$ denote such a principal hemisphere. Then we have $\mathbb{G}_f(\mathcal{E}_f) \subseteq \mathbb{H}_1$. Furthermore, $\mathbb{H}_2 = \mathbb{S}^1 \setminus \mathbb{H}_1$ is another principal hemisphere. The apex of $\mathbb{H}_2$ is an extremal point in $\mathbb{S}^1$ ($\vec{\mathbf{rc}}$ in Fig. 24(b)). This extremal point corresponds to a unit vector. Let us denote it as $\vec{\mathbf{v}}$. Since $\mathbb{G}_f(\mathcal{E}_f) \subseteq \mathbb{H}_1$, the angle between $\vec{\mathbf{v}}$ and any vector in $\mathbb{G}_f(\mathcal{E}_f)$ is greater than $\pi/2$. In particular, the angle between $\vec{\mathbf{v}}$ and each of $\mathbf{n_p}, \mathbf{n_q}, \mathbf{n_r}, \mathbf{n_s}$ is greater than $\pi/2$. This contradicts Equation 5.

We can use a similar argument to show that the voxel ϑ of C is not strongly complex and does not have an ambiguity. We can show that if ϑ is normal-bounded w.r.t $(\vec{\mathbf{u}}, arcsin(1/\sqrt(3)))$, then $\mathbb{G}_\vartheta(\mathcal{E}_\vartheta)$ is always a subset of a principal hemisphere. This fact can then be used to prove the result.

□

12.3 Gauss Map Condition for Star-shaped Criterion

We now present a Gauss map condition for when $\mathcal{E}$ is star-shaped w.r.t a voxel. We show that a voxel ϑ will satisfy the star-shaped criterion if there exists a unit vector $\vec{\mathbf{u}}$ such that ϑ is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/2)$. A similar 2D result holds for the faces of the cell.

Before proving the result, we introduce a definition.

DEFINITION 10 *Consider a line segment* **pq** *that intersects* $\mathcal{E}$*.*

- *We call an intersection point* $\mathbf{r} \in \mathcal{E}_{\mathbf{pq}}$ *a* **tangential intersection point** *if* $\mathbf{pq} \cdot \mathbf{n_r} = 0$*. Otherwise we*

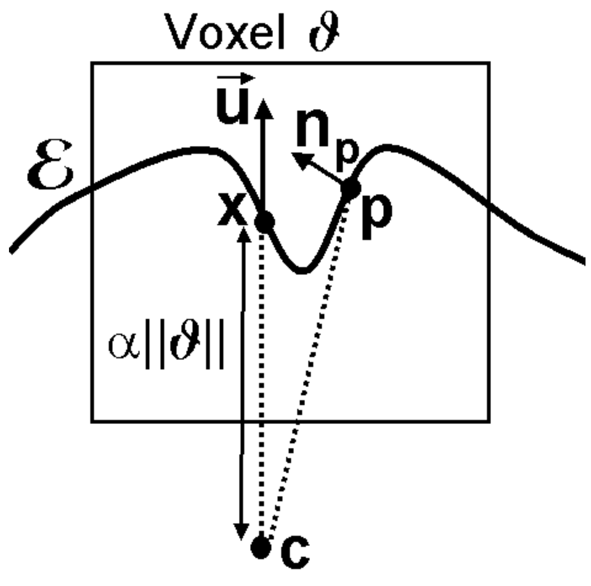


Figure 25: Condition for Star-shaped Criterion: ***This figure supports the proof of Theorem 4. If there exists a unit vector $\vec{\mathbf{u}}$ such that angle between $\vec{\mathbf{u}}$ and the normal at any point in $\mathcal{E}_\vartheta$ is less than $\pi/2$, then $\mathcal{E}_\vartheta$ is star-shaped.***

say **r** *is a* **transversal intersection point**. *We call* **r** *an* **entry point** *if* $\mathbf{pq} \cdot \mathbf{n_r} < 0$*, an* **exit point** *if* $\mathbf{pq} \cdot \mathbf{n_r} > 0$.

- *If all the intersection points are transversal then we say* **pq** *intersects* $\mathcal{E}$ **transversally**.

If **pq** intersects $\mathcal{E}$ transversally, then we can classify each intersection point as an entry point or an exit point. We can sort the intersection points in the order of increasing distance from **p**. In the sorted order, the entry and exit points will alternate each other. This is because $\mathcal{E}$ is an oriented closed manifold.

THEOREM 4 *Let ϑ be a boundary voxel, i.e. $\mathcal{E}_\vartheta \neq \emptyset$. If there exists a unit vector $\vec{\mathbf{u}}$ such that $\mathcal{E}_\vartheta$ is normal-bounded w.r.t $(\vec{\mathbf{u}}, \theta)$ for any θ, $0 < \theta < \pi/2$, then*

1. *$\mathcal{E}$ is star-shaped w.r.t ϑ.*
2. *Any point belonging to the set*

$$\{(\mathbf{x} - \alpha||\vartheta||\vec{\mathbf{u}}) \quad | \quad \mathbf{x} \in \vartheta, \ \alpha > 1/\cos(\theta)\}$$

is a guard of $\mathcal{E}_\vartheta$.

Proof: (2) implies (1). Hence we prove (2). Choose any point $\mathbf{x} \in \vartheta$. Let $\mathbf{c} = \mathbf{x} - \alpha||\vartheta||\vec{\mathbf{u}}$. Consider any point $\mathbf{p} \in \mathcal{E}_\vartheta$. We first show that $\mathbf{cp} \cdot \mathbf{n_p} > 0$.

See Fig. 25. We have

$$\begin{aligned}
\mathbf{cp} &= \mathbf{cx} + \mathbf{xp} \\
&= \alpha||\vartheta||\vec{\mathbf{u}} + \mathbf{xp} \\
\mathbf{cp} \cdot \mathbf{n_p} &= \alpha||\vartheta||\vec{\mathbf{u}} \cdot \mathbf{n_p} + \mathbf{xp} \cdot \mathbf{n_p} \\
&> \alpha||\vartheta||\cos(\theta) + \mathbf{xp} \cdot \mathbf{n_p} \quad \text{because } \angle\vec{\mathbf{u}}, \mathbf{n_p} < \theta \\
&> ||\vartheta|| + \mathbf{xp} \cdot \mathbf{n_p} \\
&> 0 \quad \text{because } ||\mathbf{xp}|| < ||\vartheta|| \text{ and } \mathbf{n_p} \text{ is a unit vector}
\end{aligned}$$

We now show that $\mathbf{cp} \cap \mathcal{E} = \{\mathbf{p}\}$. We first note that **cp** intersects $\mathcal{E}$ transversally because otherwise it will contradict the fact that $\mathbf{cp} \cdot \mathbf{n_p} > 0$ for all $\mathbf{p} \in \mathcal{E}_\vartheta$.

Suppose **cp** intersects $\mathcal{E}$ at a point other than **p**. Let **r** be a point belonging to $\mathcal{E} \cap (\mathbf{cp} \setminus \{\mathbf{p}\})$ that is closest to **p**. Since **r** and **p** are "consecutive" intersection points, one of them is an entry point and the other is an exit point. This means either $\mathbf{cp} \cdot \mathbf{n_r} < 0$ or $\mathbf{cp} \cdot \mathbf{n_p} < 0$ which is a contradiction. □

We note that the condition in the above theorem is sufficient, but not necessary. The above theorem can also be rephrased as follows: If the Gauss map $\mathbb{G}_\vartheta(\mathcal{E}_\vartheta)$ is a strict subset of a hemisphere of $\mathbb{S}^2$, then $\mathcal{E}_\vartheta$ is star-shaped. The unit vector $\vec{\mathbf{u}}$ will point towards the apex of such a hemisphere. Similar Gauss map conditions have been widely used in the boundary evaluation literature [82].

12.3.1 Conservative Star-shaped Test

In the case where $\mathcal{E}$ is non-linear, we use a conservative technique to answer the star-shaped query (Sec. 6.4). The conservative technique enumerates a set of samples on $\mathcal{E}$ to estimate a candidate point for the guard and verifies whether $\mathcal{E}$ is actually star-shaped w.r.t the candidate point. Due to a poor estimate of the candidate point, it is possible that a voxel satisfies the condition in Theorem 4, but still fails the star-shaped criterion as per the conservative technique. However, under a more restrictive condition than the one used in Theorem 4, we can show that a voxel will satisfy the star-shaped criterion even as per the conservative technique. This restrictive condition requires that a voxel be normal-bounded by $\pi/2$.

COROLLARY 1 *Consider a voxel ϑ that is normal-bounded by θ, $0 < \theta < \pi/2$. Then*

1. *$\mathcal{E}$ is star-shaped w.r.t ϑ.*
2. *Any point belonging to the set*

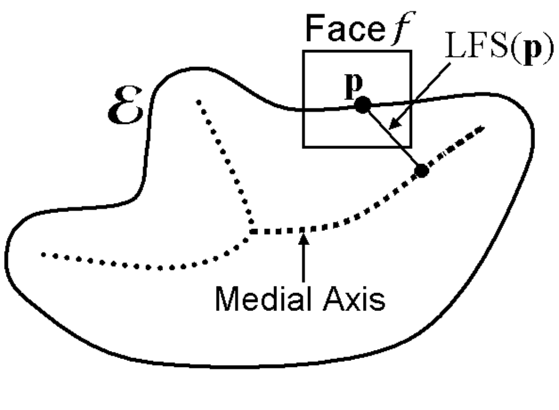
$$\{(\mathbf{p} - \alpha||\vartheta||\mathbf{n_p}) \quad | \quad \mathbf{p} \in \mathcal{E}_\vartheta, \ \alpha > 1/\cos(\theta)\}$$

is a guard of $\mathcal{E}_\vartheta$.

This corollary follows directly from Theorem 4 by choosing $\vec{\mathbf{u}} = \mathbf{n_p}$ for any point $\mathbf{p} \in \mathcal{E}_\vartheta$. The corollary provides a sufficient condition when a voxel will satisfy the star-shaped criterion as per the conservative technique. It suffices to choose merely one sample (say $\mathbf{p} \in \mathcal{E}_\vartheta$) within the voxel. Then any point belonging to the set

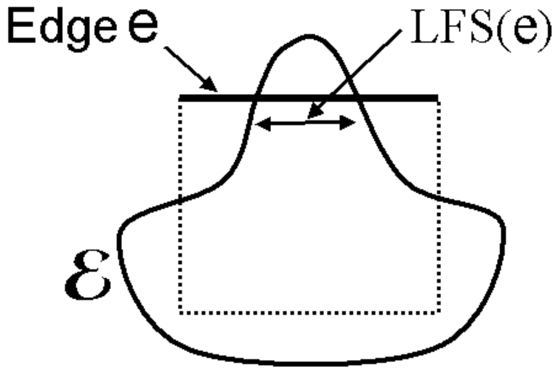
$$\{(\mathbf{p} - \alpha||\vartheta||\mathbf{n_p}) \mid \alpha > 1/\cos(\theta)\}$$

can be chosen as a guard. In practice, we enumerate more than one sample point to obtain a faster convergence. This



(a)

Figure 26: Local Feature Size (LFS): ***The LFS of a point* p *w.r.t* $\mathcal{E}$ *is defined as the distance between* p *and the medial axis of* $\mathcal{E}$*. We extend this definition to a face. The LFS of a face* f *is defined as the minimum of the LFS of all points in* $\mathcal{E}_f$*.***



(a)

Figure 27: LFS of an edge: ***This figure shows the LFS of an edge* e *that is intersected by* $\mathcal{E}$ *at two points. In this case, the LFS of the edge is equal to the distance between the two intersection points.***

enables us to verify the star-shaped criterion even in the case where the voxel is not normal-bounded by $\pi/2$.

12.4 Local Feature Size Condition

In this subsection, we derive a conservative lower bound on the size of the grid cells during adaptive subdivision. This provides a sufficient condition for the termination of the algorithm.

We reduce the Gauss map conditions for both the complex cell and star-shaped criteria to a common condition based on *local feature size* (LFS). We start by defining the LFS of a cell. Then we show that both the Gauss map conditions are met if the grid cells are smaller than a certain fraction of their LFS. This yields a lower bound on the cell size and in turn a sufficient condition for termination of the algorithm.

We define the LFS of a cell, which in turn is defined in terms of the LFS of its voxel, faces and edges. The LFS of a voxel is defined as the minimum of the LFS of all the points on $\mathcal{E}$ that belong to the voxel. The LFS of a face/edge is defined similarly by considering the restriction of $\mathcal{E}$ to the face/edge. See Figs. 26 and 27.

LFS of a Voxel: Let LFS : $\mathcal{E} \to \mathbb{R}$ denote the LFS of $\mathcal{E}$. Recall that LFS(**p**) at a point **p** on $\mathcal{E}$ is defined as the least distance of **p** to the *medial axis* of $\mathcal{E}$. Then the LFS of a voxel ϑ is defined as:

$$\begin{aligned}\text{LFS}(\vartheta) &= \min\{\text{LFS}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{E}_\vartheta\} \quad \text{if } \mathcal{E}_\vartheta \neq \emptyset \\ &= \infty \quad \text{otherwise}\end{aligned}$$

LFS of a Face: Let Π_f be the plane containing a face f. Consider the restriction $\mathcal{E}_\Pi$ of $\mathcal{E}$ to Π. We can treat $\mathcal{E}_\Pi$ as a curve in $\mathbb{R}^2$; hence we can use a 2D definition of LFS for $\mathcal{E}_\Pi$. Let the LFS be defined by the function $\text{LFS}_{\Pi_f} : \mathcal{E}_\Pi \to \mathbb{R}$. Then the LFS of f is defined as:

$$\begin{aligned}\text{LFS}(f) &= \min\{\text{LFS}_{\Pi_f}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{E}_f\} \quad \text{if } \mathcal{E}_f \neq \emptyset \\ &= \infty \quad \text{otherwise}\end{aligned}$$

See Fig. 26.

LFS of an Edge: Let l_e be the line containing an edge e. Consider the restriction $\mathcal{E}_{l_e}$ of $\mathcal{E}$ to l_e. We define a one-dimensional LFS_{l_e} for points on $\mathcal{E}_{l_e}$ in terms of three cases:

1. If $\mathcal{E}_{l_e} = \emptyset$, then LFS_{l_e} is always infinity.
2. Suppose $\mathcal{E}$ intersects l_e at one point. If this intersection is tangential, then LFS_{l_e} is always zero. Otherwise it is infinity.
3. $\mathcal{E}$ intersects l_e at multiple points. Then LFS_{l_e} is defined as follows:

$$\begin{aligned}\text{LFS}_{l_e}(\mathbf{p}) &= 0 \quad \text{if } \mathbf{p} \text{ is a tangential intersection point} \\ &= \inf\{d(\mathbf{p},\mathbf{q}) \mid \mathbf{q} \in \mathcal{E}_{l_e},\ \mathbf{q} \neq \mathbf{p}\} \quad \text{otherwise}\end{aligned}$$

The LFS of edge e is defined as follows:

$$\begin{aligned}\text{LFS}(e) &= \min\{\text{LFS}_{l_e}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{E}_e\} \quad \text{if } \mathcal{E}_e \neq \emptyset \\ &= \infty \quad \text{otherwise}\end{aligned}$$

See Fig. 27.

LFS of a Cell: The LFS of a cell is defined as the minimum of the LFS of its edges, faces, and the voxel.

Intuitively, our goal is to show that a cell will satisfy the complex cell and star-shaped criteria if it is "sufficiently small". We make the previous statement precise using the following definition.

DEFINITION 11 *Let c be an edge/face/voxel of a cell C.*

1. *c is LFS-small if $\|c\| < \rho$ LFS(c) for any $\rho < \beta/(1 + 3\beta)$ where $\beta = arcsin(1/\sqrt(3))$.*
2. *C is LFS-small if every edge/face/voxel of C is LFS-small.*

The choice of the value of ρ is determined by Theorem 5 (see below). We show that a LFS-small cell satisfies the complex cell and star-shaped criteria. Our proof relies on the following lemma presented by Amenta and Bern [81]. It basically states that the normals at two closeby points on the surface are close to each other. The surface $\mathcal{E}$ is assumed to be a twice-differentiable manifold.

LEMMA 6 *For any two points $\mathbf{p}$ and $\mathbf{q}$ on $\mathcal{E}$ with $d(\mathbf{p}, \mathbf{q}) \leq \rho \min\{\text{LFS}(p), \text{LFS}(q)\}$, for any $\rho < 1/3$, the angle between the normals to $\mathcal{E}$ at $\mathbf{p}$ and $\mathbf{q}$ is at most $\rho/(1 - 3\rho)$ [81].*

THEOREM 5 *Let C be a LFS-small cell. Then,*

1. *C is not complex.*
2. *$\mathcal{E}$ is star-shaped w.r.t C.*

Proof: Let c be a face/voxel of C. Because c is LFS-small, any two points in $\mathcal{E}_c$ are within distance $\|c\| < \rho$ LFS(c) where $\rho < \beta/(1 + 3\beta)$. Then according to Lemma 6, the angle between the normals to any two points in $\mathcal{E}_c$ is at most $\rho/(1 - 3\rho) = \beta = arcsin(1/\sqrt(3))$. Hence c is normal-bounded by β. We now use Theorem 3 and Corollary 1 to prove the result.

To apply Theorem 3, we choose $\vec{\mathbf{u}}$ to be the normal at any point in $\mathcal{E}_c$. Thus c is normal-bounded w.r.t $(\vec{\mathbf{u}}, \pi/4)$. Furthermore, by definition, an LFS-small edge is not complex. Therefore C cannot have complex edges. Theorem 3 implies that C is not complex.

Since c is normal-bounded by $\pi/4$, Corollary 1 ensures that $\mathcal{E}$ is star-shaped w.r.t both voxel as well as faces of C. □

12.5 Termination

Theorem 5 provides a lower bound on the size of the grid cells relative to the LFS. During adaptive subdivision, once the size of a cell C is less than ρLFS(C), then it is LFS-small, and satisfies both the complex cell and star-shaped criteria. The algorithm will terminate provided there exist a lower bound on the LFS of every grid cell. Suppose there exists such a lower bound τ. Assuming a cell halves its size at each subdivision step, this implies a lower bound of $\rho\tau/2$ on the size of every cell. We use this fact to provide the following sufficient condition for the termination of the subdivision algorithm:

COROLLARY 2 *If there exists an $\tau > 0$, such that during adaptive subdivision, the LFS of every grid cell is greater than τ, then the subdivision algorithm will terminate and the grid cells will be of size greater than $\rho\tau/2$.*

In general, it is difficult to enforce the above condition during adaptive subdivision. During the subdivision process, as new voxels, faces and edges are created, the LFS of the newly created voxels/faces/edges change. While it is true that the LFS of a voxel of a child cell is always greater than or equal to the LFS of the voxel of its parent cell, a similar property does not hold for the faces and edges of the children cells. The LFS for the newly created faces and edges depends on how they intersect $\mathcal{E}$. If these faces and edges intersect $\mathcal{E}$ in a "near grazing" manner, then their LFS can be arbitrarily small. For example, if $\mathcal{E}$ has large flat regions that are parallel to the axis-aligned planes of the coordinate system (XY, YZ, or ZX), then the LFS of some of the faces and edges can be zero. Such problems may sometimes be alleviated by choosing a different set of co-ordinate axes.

We conclude the analysis with a few remarks. We note that LFS-small condition is sufficient, but not necessary. Furthermore, the lower bound $(\rho\tau/2)$ is an overly conservative lower bound on the cell size. In practice, we have observed that the algorithm produces much larger cells. Finally, even though our analysis is restricted to only smooth surfaces, our algorithm is applicable to surfaces with sharp features.

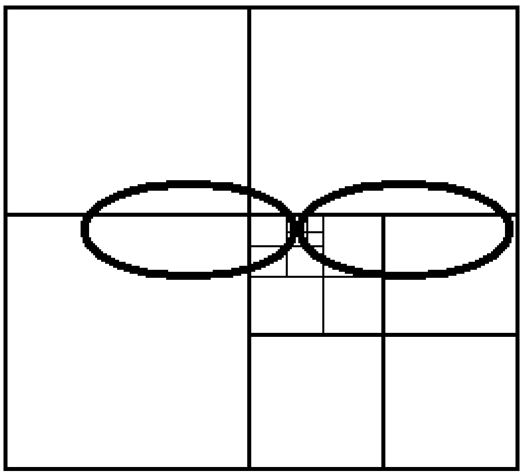


Figure 28: Tangential Contact: ***This figure shows a case where two primitives touch each other at a point. We call such a contact a*** **tangential contact*****. It is a degenerate case for our algorithm.***

13. DEGENERACIES

There are two types of degenerate cases for our algorithm. The first type of degeneracy occurs when $\mathcal{E}$ has a *tangential contact*. See Fig. 28. This can occur when two input primitives touch each other. Different types of tangential contacts are possible: the contact region may be a point, curve, or a surface. Our algorithm cannot handle such cases. No matter how much subdivision is performed in the vicinity of the tangential contact, the complex cell and star-shaped properties are never satisfied.

Another kind of degeneracy occurs when the $\mathcal{E}$ grazes an edge or a face of a cell. The contact region may be a point, curve, or a surface. We refer to these types of situations as *grazing contacts*. Fig. 5 shows a few examples. In the vicinity of a grazing contact, the sampling condition is never met; an edge (face) with grazing contact will not satisfy the complex edge (complex face) criterion. One way of reducing the likelihood of grazing contacts is to perform the adaptive subdivision randomly, i.e., choose random points to split the voxel, faces, and edges of the cell. For example, we can randomly select a point in the interior of the voxel, and subdivide the voxel into tetrahedral regions with the chosen point as an apex. This type of subdivision would generate a tetrahedral grid. Isosurface extraction can then be performed using an MC-like algorithm such as Marching Tetrahedra [83, 84].

Both tangential contact and grazing contact can be characterized in terms of the LFS. At a tangential contact, the LFS of the voxel containing the contact is zero. Similarly, at a grazing contact along a face or an edge, the LFS of the corresponding face or edge is zero.

Detecting the occurence of either type of degeneracy is difficult. This is because we do not have an explicit representation of $\mathcal{E}$. Note that the portion of $\mathcal{E}$ involved in the contact may belong to either a single primitive or the intersection curve between multiple primitives. While it may be possible to detect the first case, the second case is much harder as this requires intersection curve computation.

Handling degeneracies is a challenging problem that has been extensively studied in solid modeling [85–87]. Many approaches have been proposed to handle them. One option is to perform "special case handling": enumerating all the possible types of degeneracies and adding code to explicitly detect and resolve them. While this approach can be useful in many situations, it leads to more complexity in the underlying algorithms and representations, e.g., these algorithms need to work with non-manifold representations. Moreover, these algorithms will not be compatible with MC-like reconstruction methods: We cannot use special-purpose algorithms in cells containing degeneracies and MC-like methods in the remaining cells as this may lead to cracks in the output.

Another approach to handling degeneracies is to use perturbation methods. This approach applies a perturbation to the input to eliminate the degeneracy. The perturbation may be done either symbolically [85] or numerically [87]. It may be possible to use perturbation methods to resolve grazing and tangential contacts. The input primitives defining $\mathcal{E}$ can be numerically perturbed. This defines a perturbed surface $\mathcal{E}'$. If the perturbation is chosen randomly, it is likely that $\mathcal{E}'$ is not degenerate. We can then apply our adaptive subdivision approach to the perturbed input.

The main advantage of perturbation methods is that we can apply the adaptive subdivision algorithm directly without having to explicitly handle degeneracies. However, there are a few issues with using numerical perturbation. It modifies the input data, and hence the output will be topologically equivalent to the perturbed surface $\widetilde{\mathcal{E}}$ and not the original surface $\mathcal{E}$. Moreover, adding a numerical perturbation may not necessarily eliminate the degeneracy, or even worse, it may create another. Therefore, this method requires a robust test for detecting degeneracy. If the applied perturbation does not resolve the degeneracy, another perturbation is needed.

14. LIMITATIONS

In this section, we discuss the limitations of our algorithm. As discussed earlier, our algorithm does not terminate in certain degenerate cases. Our algorithm can only generate manifold boundaries and is not applicable to the cases where the exact boundary is non-manifold.

We do not provide a bound on the time complexity of our algorithm as a function of the combinatorial complexity of the input primitives. This is because of we are unable to give an absolute lower bound on the size of the grid cells generated during adaptive subdivision. Sec. 12 provides a lower bound on the cell size relative to the LFS of a cell. However, to obtain an absolute bound, the LFS needs to be expressed as a function of the combinatorial complexity of the input. This requires further analysis.

Our algorithm may perform conservative subdivision. Within a cell, we require that all the primitives should be star-shaped with respect to a common guard. This is a conservative condition. The isosurface defined by the Boolean expression over the primitives can be star-shaped within the cell even though this condition may not be satisfied. This can result in additional subdivision and lead to higher polygon counts in the approximation.

Our topology preserving simplification algorithm cannot perform drastic simplifications. This is due to the conservative subdivision and also the fact that volumetric approaches can not produce drastic simplifications [88]. Moreover, for a fixed polygon budget, approaches based on surface decimation operations like edge collapses or vertex removal [79] will generate a higher quality simplification.

15. SUMMARY

We have described a novel approach to compute topology preserving isosurfaces that arise in a variety of geometric processing applications. We have presented a sufficient sampling condition based on the *complex cell* and *star-shaped* criteria so that the reconstruction maintains the topology of the original isosurface. We have described a simple extension to the sampling condition to also bound the two-sided Hausdorff error of the reconstruction. We have also described an adaptive subdivision algorithm which is efficient in practice and easy to implement. We have demonstrated the application of our algorithm to Boolean operations, topology preserving simplification, and remeshing on a number of complex examples.

16. FUTURE WORK

There are many avenues for future work. Our sampling criteria – complex cell and star-shaped criteria – are geared towards Marching Cubes reconstruction. We would like to develop better reconstruction algorithms so that we could make the sampling criteria less conservative and yet preserve topology.

The star-shaped criterion ensures that the surface within every cell is star-shaped. A useful property of a star-shaped surface is that it has a spherical parametrization. The fact that every point on the surface is visible to the guard can be used to map the surface onto a portion of the unit sphere. Then a tessellation of this portion of the sphere yields a polygonization of the star-shaped surface. It would be useful to develop a reconstruction algorithm that exploits this property.

Current algorithms for kernel computation on curved primitives involve solving non-linear equations and can be slow. For the special case of rational freeform surfaces, kernel computation can be reformulated as computing the zero sets of polynomial equations [89]. Solving such equations for each grid cell can be rather expensive in practice. Moreover, no good algorithms are known for kernel computation on freeform solids defined using subdivision surfaces. We would like to develop efficient algorithms for kernel computation on curved solids.

In applications such as laser scanning, the input data often contains topological noise due to inaccuracies in the scanning and merging process. We would like to investigate whether our results can be combined with the algorithms presented in [56, 57] and used to perform topological reasoning for noise removal.

Our current implementation supports polyhedral and low order algebraic primitives. We would like to apply our algorithm to higher order NURBS and subdivision surfaces. Finally, we plan to use our algorithm for other surface extraction problems such as swept volume computation.

17. REFERENCES

[1] G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha, "Topology preserving surface extraction using adaptive subdivision," in *Eurographics Symposium on Geometry Processing*, 2004.

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, 1987, pp. 163–169.

[3] L. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel, "Feature-sensitive surface extraction from volume data," in *Proc. of ACM SIGGRAPH*, 2001, pp. 57–66.

[4] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 21, no. 3, 2002.

[5] G. Varadhan, S. Krishnan, Y. Kim, and D. Manocha, "Feature-sensitive subdivision and isosurface reconstruction," *Proc. of IEEE Visualization*, 2003.

[6] B. Wyvill, C. McPheeters, and G. Wyvill, "Animating soft objects," *The Visual Computer*, vol. 2, no. 4, pp. 235–242, 1986.

[7] J. Bloomenthal, "Polygonization of implicit surfaces," *Comput. Aided Geom. Design*, vol. 5, no. 4, pp. 341–355, 1988.

[8] B. Wyvill and K. van Overveld, "Polygonization of Implicit Surfaces with Constructive Solid Geometry," *Journal of Shape Modelling*, vol. 2, no. 4, pp. 257–274, 1996.

[9] D. Breen, S. Mauch, and R. Whitaker, "3d scan conversion of csg models into distance, closest-point and color volumes," *Proc. of Volume Graphics*, pp. 135–158, 2000.

[10] R. Perry and S. Frisken, "Kizamu: A system for sculpting digital characters," in *Proc. of ACM SIGGRAPH*, 2001, pp. 47–56.

[11] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel, "Multi-level partition of unity implicits," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 463–470, 2003.

[12] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. New Jersey, NJ: Prentice-Hall Inc, 1997.

[13] S. Frisken, R. Perry, A. Rockwood, and R. Jones, "Adaptively sampled distance fields: A general representation of shapes for computer graphics," in *Proc. of ACM SIGGRAPH*, 2000, pp. 249–254.

[14] P. Sutton, C. Hansen, H. Shen, and D. Schikore, "A case study of isosurface extraction algorithm performance," 2000. [Online]. Available: citeseer.ist.psu.edu/sutton00case.html

[15] H. Carr, "Topological manipulation of isosurfaces," Ph.D. dissertation, The University of British Columbia, 2004.

[16] H. Fuchs, Z. M. Kedem, and S. P. Uselton, "Optimal surface reconstruction from planar contours," *Commun. ACM*, vol. 20, no. 10, pp. 693–702, 1977.

[17] A. Koide, A. Doi, and K. Kajioka, "Polyhedral approximation approach to molecular orbital graphics," *J. Mol. Graph.*, vol. 4, no. 3, pp. 149–155, 1986.

[18] B. A. Payne and A. W. Toga, "Medical imaging: Surface mapping brain function on 3d models," *IEEE Comput. Graph. Appl.*, vol. 10, no. 5, pp. 33–41, 1990.

[19] W. E. Lorensen, "Marching through the visible man," in *VIS '95: Proceedings of the 6th conference on Visualization '95*. Washington, DC, USA: IEEE Computer Society, 1995, p. 368.

[20] P. D. Heermann, "Production visualization for the asci one teraflops machine," in *VIS '98: Proceedings of the conference on Visualization '98*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1998, pp. 459–462.

[21] C. R. F. Monks, P. J. Crossno, G. Davidson, C. Pavlakos, A. Kupfer, C. Silva, and B. Wylie, "Three dimensional visualization of proteins in cellular interactions," in *VIS '96: Proceedings of the 7th conference on Visualization '96*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996, pp. 363–ff.

[22] A. Ricci, "A constructive geometry for computer graphics," *Computer Journal*, vol. 16, no. 2, pp. 157–160, May 1973.

[23] J. F. Blinn, "A generalization of algebraic surface drawing," *ACM Transactions on Graphics*, vol. 1, no. 3, pp. 235–256, July 1982.

[24] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko, "Function representation in geometric modeling: concepts, implementation and applications," *The Visual Computer*, vol. 11, no. 8, pp. 429–446, 1995.

[25] J. Bloomenthal, Ed., *Introduction to Implicit Surfaces*. Morgan-Kaufmann, 1997, vol. 391.

[26] S. Wang and A. Kaufman, "Volume-sampled 3d modeling," *IEEE Computer Graphics and Applications*, vol. 14, no. 5, pp. 26–32, 1994.

[27] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, July 1992, pp. 71–78.

[28] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *SIGGRAPH 96 Conference Proceedings*, ser. Annual Conference Series, H. Rushmeier, Ed., ACM SIGGRAPH. Addison Wesley, Aug. 1996, pp. 303–312, held in New Orleans, Louisiana, 04-09 August 1996.

[29] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, "Reconstruction and representation of 3d objects with radial basis functions," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2001, pp. 67–76.

[30] Z. Wood, H. Hoppe, M. Desbrun, and P. Schroder, "Iso-surface topology simplification," Microsoft Research, MSR-TR-2002-28, Tech. Rep., 2002.

[31] W. J. Schroeder, W. E. Lorensen, and S. Linthicum, "Implicit modeling of swept surfaces and volumes," in *VIS '94: Proceedings of the conference on Visualization '94*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 40–45.

[32] M. Hall and J. Warren, "Adaptive polygonalization of implicitly defined surfaces," *IEEE Comput. Graph. Appl.*, vol. 10, no. 6, pp. 33–42, 1990.

[33] L. Velho, "Adaptive polygonization of implicit surfaces

using simplicial decomposition and boundary constraints," in *Eurographics '90*, C. E. Vandoni and D. A. Duce, Eds. North-Holland, Sept. 1990, pp. 125–136.

[34] P. Ning and J. Bloomenthal, "An evaluation of implicit surface tilers," *IEEE Comput. Graph. Appl.*, vol. 13, no. 6, pp. 33–41, 1993.

[35] A. Bottino, W. Nuij, and K. van Overveld, "How to shrinkwrap through a critical point: An algorithm for the adaptive tesselation of iso-surfaces with arbitrary topology," *Implicit Surfaces*, pp. 53–72, 1996.

[36] B. T. Stander and J. C. Hart, "Guaranteeing the topology of an implicit surface polygonization for interactive modeling," in *Proc. of ACM SIGGRAPH*, 1997, pp. 279–286.

[37] Y. Ohtake, A. G. Belyaev, and A. Pasko, "Dynamic meshes for accurate polygonanization of implicit surfaces with sharp features," *Prof. of Shape Modeling International*, pp. 135–158, 2001.

[38] N. Zhang, W. Hong, and A. Kaufman, "Dual contouring with topology-preserving simplification using enhanced cell representation," in *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 505–512.

[39] J.-D. Boissonnat, D. Cohen-Steiner, and G. Vegter, "Isotopic implicit surface meshing," *STOC*, 2004.

[40] G. M. Nielson, "Dual marching cubes," in *VIS '04: Proceedings of the conference on Visualization '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 489–496.

[41] S. Schaefer and J. Warren, "Dual marching cubes: Primal contouring of dual grids," in *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 70–76.

[42] J. Wilhelms and A. V. Gelder, "Octrees for faster isosurface generation extended abstract," in *Computer Graphics (San Diego Workshop on Volume Visualization)*, vol. 24, no. 5, Nov. 1990, pp. 57–62.

[43] C. L. Bajaj, V. Pascucci, and D. R. Schikore, "Fast isocontouring for improved interactivity," in *1996 Volume Visualization Symposium*. IEEE, Oct. 1996, pp. 39–46, iSBN 0-89791-741-3.

[44] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno, "Optimal isosurface extraction from irregular volume data," in *1996 Volume Visualization Symposium*. IEEE, Oct. 1996, pp. 31–38, iSBN 0-89791-741-3.

[45] Y. Livnat, H.-W. Shen, and C. R. Johnson, "A near optimal isosurface extraction algorithm using the span space," *IEEE Trans. Visualizat. Comput. Graph.*, vol. 2, pp. 73–84, 1996.

[46] R. Shekhar, E. Fayyad, R. Yagel, and F. Cornhill, "Octree-based decimation of marching cubes surfaces," *Proc. of IEEE Visualization*, pp. 335–342, 1996.

[47] R. Westermann, L. Kobbelt, and T. Ertl, "Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces," *The Visual Computer*, vol. 2, pp. 100–111, 1999.

[48] T. Gerstner and R. Pajarola, "Topology preserving and controlled topology simplifying multi-resolution isosurface extraction," *Proc. of IEEE Visualization*, pp. 259–266, 2000.

[49] C. Montani, R. Scateni, and R. Scopigno, "Discretized marching cubes," *Proc. of IEEE Visualization*, pp. 353–355, 1994.

[50] M.J.Durst, "Letters: Additional reference to marching cubes," *ACM Computer Graphics*, vol. 22, no. 4, pp. 72–73, 1988.

[51] J. Wilhelms and A. V. Gelder, "Topological considerations in isosurface generation extended abstract," *Computer Graphics*, vol. 24, no. 5, pp. 79–86, 1990.

[52] G. M. Nielson and B. Hamann, "The asymptotic decider: Removing the ambiguity in marching cubes," in *Visualization '91*, 1991, pp. 83–91.

[53] B. K. Natarajan, "On generating topologically consistent isosurfaces from uniform samples," *Vis. Comput.*, vol. 11, no. 1, pp. 52–62, 1994.

[54] P. Cignoni, F. Ganovelli, C. Montani, and R. Scopigno, "Reconstruction of topologically correct and adaptive trilinear isosurfaces," *Computers & Graphics*, vol. 24, no. 3, pp. 399–418, 2000.

[55] A. Lopes and K. Brodlie, "Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 1, pp. 16–29, 2003.

[56] I. Guskov and Z. Wood, "Topological noise removal," *Proc. of Graphics Interface*, 2001.

[57] S. Bischoff and L. Kobbelt, "Isosurface reconstruction with topology control," *Proc. of Pacific Graphics*, pp. 246–255, 2002.

[58] J. M. Snyder, "Interval analysis for computer graphics," in *Computer Graphics (SIGGRAPH '92 Proceedings)*, E. E. Catmull, Ed., vol. 26, July 1992, pp. 121–130.

[59] S. Plantinga and G. Vegter, "Isotopic approximation of implicit curves and surfaces," in *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. New York, NY, USA: ACM Press, 2004, pp. 245–254.

[60] H. Samet, *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods*. Addison Wesley, 1989.

[61] J. Munkres, *Topology: A First Course*. Prentice-Hall, 1975.

[62] H. Edelsbrunner and N. R. Shah, "Triangulating topological spaces," in *ACM Symposium on Computational Geometry*, 1994, pp. 285–292.

[63] R. E. Moore, *Interval Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1966.

[64] G. Varadhan, S. Krishnan, Y. Kim, S. Diggavi, and D. Manocha, "Efficient max-norm computation and reliable voxelization," *Proc. of ACM SIGGRAPH/Eurographics Symposium on Geometry Processing*, pp. 116–126, 2003.

[65] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed. Berlin, Germany: Springer-Verlag, 2000.

[66] A. Schrijver, *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[67] GLPK, "Gnu linear programming kit, url:http://www.gnu.org/software/glpk/glpk.html," 2003. [Online]. Available: \url{http://www.gnu.org/software/glpk/glpk.html}

[68] QSOPT, "Qsopt linear programming solver, url:http://www.isye.gatech.edu/ wcook/qsopt/index.html," 2005.

[69] J.-K. Seong, G. Elber, J. Johnstone, and M.-S. Kim, "The convex hull and kernel of freeform surfaces," in *UAB Technical Report (UABCIS-TR-2004-120104-02)*, 2003.

[70] D. Manocha and J. F. Canny, "Implicit representation of rational parametric surfaces," *J. Symb. Comput.*, vol. 13, no. 5, pp. 485–510, 1992.

[71] N. Aspert, D. Santa-Cruz, and T. Ebrahimi, "Mesh: Measuring error between surfaces using the hausdorff distance,," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, 2002, pp. 705–708.

[72] M. Guthe, P. Borodin, and R. Klein, "Fast and accurate hausdorff distance calculation between meshes," vol. 13, no. 2, February 2005, pp. 41–48.

[73] J. Peters, "Efficient one-sided linearization of spline geometry," *10th IMA Conference on Mathematics of Surfaces*, pp. 297–319, 2003.

[74] G. Varadhan and D. Manocha, "Accurate minkowski sum approximation of polyhedral models." in *Pacific Conference on Computer Graphics and Applications*, 2004,

pp. 392–401.

[75] G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha, “A simple algorithm for complete motion planning of translating polyhedral robots,” in *Workshop on Algorithmic Foundations of Robotics*, 2004.

[76] G. Varadhan and D. Manocha, “Star-shaped roadmaps - a deterministic sampling approach for complete motion planning,” in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[77] T. He, L. Hong, A. Varshney, and S. Wang, “Controlled topology simplification,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 2, pp. 171–184, 1996.

[78] F. S. Nooruddin and G. Turk, “Simplification and repair of polygonal models using volumetric techniques,” *IEEE Trans. on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 191–205, 2003.

[79] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, “Simplification envelopes,” in *Proc. of ACM Siggraph’96*, 1996, pp. 119–128.

[80] S. Zelinka and M. Garland, “Permission grids: Practical, error-bounded simplification,” *ACM Trans. on Graphics*, 2002.

[81] N. Amenta and M. Bern, “Surface reconsruction by Voronoi filtering,” in *ACM Symposium on Computational Geometry*, 1998, pp. 39–48.

[82] M. E. Hohmeyer, “A surface intersection algorithm based on loop detection,” vol. 1, no. 4, pp. 473–490, 1991.

[83] B. A. Payne and A. W. Toga, “Surface mapping brain function on 3D models,” *IEEE Computer Graphics and Applications*, vol. 10, no. 5, pp. 33–41, Sept. 1990.

[84] A. Gueziec and R. Hummel, “Exploiting triangulated surface extraction using tetrahedral decomposition,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 4, pp. 328–342, Dec. 1995, iSSN 1077-2626.

[85] R. Seidel, “The nature and meaning of perturbations in geometric computing, Manuscript,” 1994.

[86] C. Hoffmann, “Robustness in geometric computations,” *Journal of Computing and Information Science in Engineering*, vol. 1, pp. 143–156, 2001.

[87] K. Ouchi and J. Keyser, “Handling degeneracies in exact boundary evaluation,” in *Proceedings of 9th ACM Symposium on Solid Modeling and Applications*, 2004, pp. 321–326.

[88] J. El-Sana and A. Varshney, “Controlled simplification of genus for polygonal models,” *Proc. of IEEE Visualization*, pp. 403–410, 1997.

[89] J. K. Seong, G. Elber, J. Johnston, and M. S. Kim, “The convex hull of freeform surfaces,” *Computing*, 2004.