Adaptive grid generation for discretizing implicit complexes

YIWEN JU, Washington University in St. Louis, USA
XINGYI DU, Washington University in St. Louis, USA
QINGNAN ZHOU, Adobe Research, USA
NATHAN CARR, Adobe Research, USA
TAO JU, Washington University in St. Louis, USA

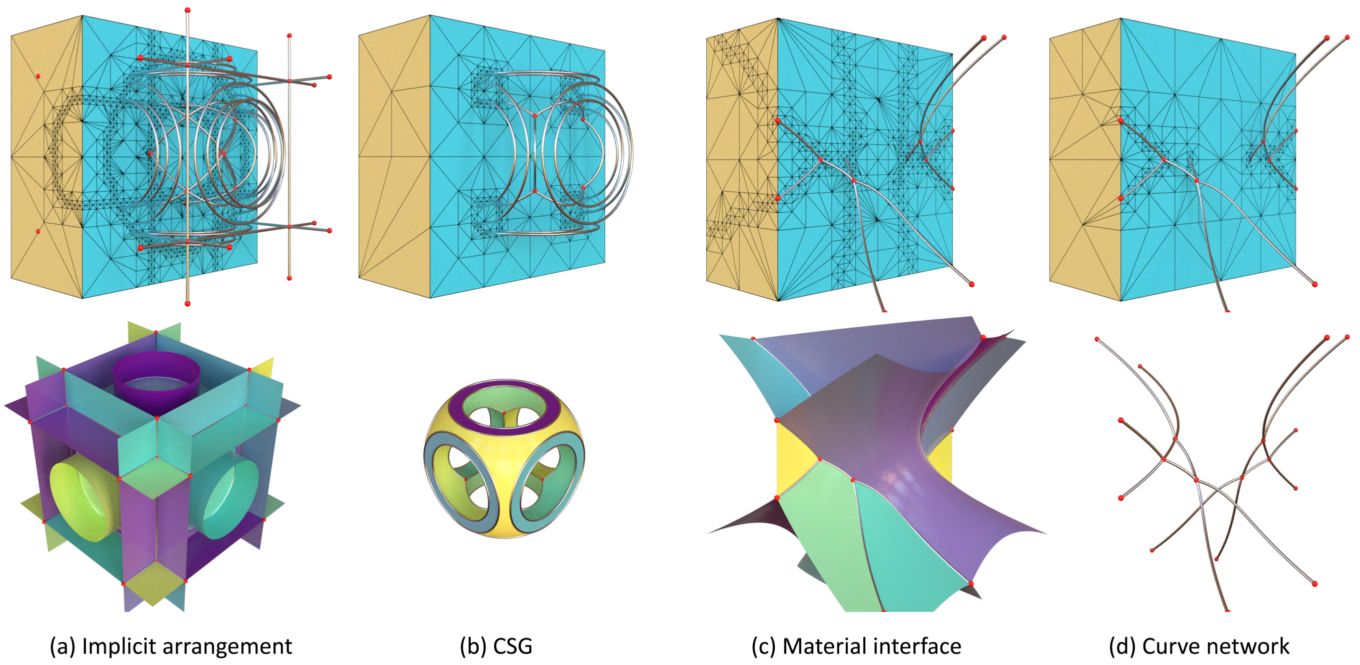

Fig. 1. Our method generates simplicial grids (top) that enable adaptive discretization of a variety of implicit shapes defined by vector functions (bottom), such as arrangements of multiple implicit surfaces (a), CSG shapes (b), non-manifold material interfaces (e), and curve networks (d). The grids are adapted to the geometric complexity of not only the surface components but also their intersections (shown as wires on top).

We present a method for generating a simplicial (e.g., triangular or tetrahedral) grid to enable adaptive discretization of implicit shapes defined by a vector function. Such shapes, which we call implicit complexes, are generalizations of implicit surfaces and useful for representing non-smooth and non-manifold structures. While adaptive grid generation has been extensively studied for polygonizing implicit surfaces, few methods are designed for implicit complexes. Our method can generate adaptive grids for several implicit complexes, including arrangements of implicit surfaces, CSG shapes, material interfaces, and curve networks. Importantly, our method adapts the grid to the geometry of not only the implicit surfaces but also their lower-dimensional intersections. We demonstrate how our method enables efficient and detail-preserving discretization of non-trivial implicit shapes.

CCS Concepts: • **Computing methodologies → Mesh geometry models**.

Additional Key Words and Phrases: implicit surfaces, grid refinement, surface networks

ACM Reference Format:
Yiwen Ju, Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju. 2024. Adaptive grid generation for discretizing implicit complexes. *ACM Trans. Graph.* 43, 4, Article 82 (July 2024), 17 pages. https://doi.org/10.1145/3658215

Authors' addresses: Yiwen Ju, Washington University in St. Louis, USA, yiwen.ju@wustl.edu; Xingyi Du, Washington University in St. Louis, USA, du.xingyi@wustl.edu; Qingnan Zhou, Adobe Research, USA, qzhou@adobe.com; Nathan Carr, Adobe Research, USA, ncarr@adobe.com; Tao Ju, Washington University in St. Louis, USA, taoju@wustl.edu.

1 INTRODUCTION

Implicit representations are widely used in computer graphics. A smooth and manifold surface in 3D can be represented as the level set of a scalar function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, also known as an *implicit surface*. The implicit surface enjoys a number of benefits, including a simple definition, easy modification (in both geometry and topology), and convenience for operations such as offsets and boolean.

Shapes beyond smooth manifolds can also be represented implicitly, by making use of a vector function $f : \mathbb{R}^3 \to \mathbb{R}^m$ where $m > 1$. For example, a CSG shape [Requicha and Voelcker 1977], often used to represent piecewise smooth surfaces, consists of subsets of multiple implicit surfaces, each can be defined by the level set of a scalar component of f. A non-manifold network of surfaces can be represented either as the arrangement of multiple implicit surfaces [Bagley et al. 2016; Guo et al. 2021], or as the interface between regions where one component of f is greater than the rest [Bertram et al. 2005; Zhang et al. 2008]. A vector function can also define lower-dimensional shapes, such as spatial curves defined by the intersection of two implicit surfaces [Burns et al. 2005; Edelsbrunner and Harer 2002; Kohlbrenner et al. 2023]. We refer to a shape implicitly defined by some vector function as an *implicit complex*. Some examples can be found in Figure 1 (bottom).

To be useful for downstream applications, an implicit surface or complex must be discretized. Many discretization methods, such as Marching Cubes [Lorensen and Cline 1987] and Marching Tetrahedra [Bloomenthal 1988], operate on a spatial grid consisting of cubic or simplicial cells (reviewed in Section 2.1). The grid structure plays a key role in the performance of these methods. While a finer grid generally leads to higher discretization accuracy, it comes at the cost of slower grid generation and discretization, a larger memory footprint, and a larger output size. An ideal grid for discretization should be *adaptive* in that the finer grid cells are located *only* where higher accuracy is needed. These locations are typically where the implicit shape has a non-trivial geometry or topology.

While adaptive grid generation has been extensively studied for polygonizing implicit surfaces, works on implicit complexes have been scarce (see Section 2.2). A key challenge in the latter is adapting the grid structure to accurately discretize the *intersection* of multiple implicit surfaces. Such intersections form the lower-dimensional elements in an implicit complex, such as the sharp edges and corners in a CSG shape or the junction graph in a non-manifold surface network. Note that the intersection of implicit surfaces may have a more complex geometry than the surfaces themselves, as shown in the example of Figure 2 (a). As a result, grids adapted only to the implicit surfaces could suffer from either insufficient resolution at their intersections (Figure 2 (b)) or excessive refinement elsewhere (Figure 2 (c)). Existing methods that can capture such intersections either are limited to low-degree polynomials [Dupont et al. 2007; Mourrain et al. 2005; Schömer and Wolpert 2006] or rely on interval analysis [Allgower and Georg 1989; Boissonnat and Wintraecken 2022], and hence they are not suited for general functions where tight intervals can be difficult, if not impossible, to obtain.

This paper presents a new method for generating adaptive simplicial grids for discretizing a variety of implicit complexes (such as those in Figure 1). A key feature of the method is that it adapts the grid structure to the geometry of both the implicit surfaces and their intersections. This allows elements at all dimensions in an implicit complex to be accurately discretized without excessive refinement of the grid (see Figure 2 (d)). Unlike existing intersection-aware methods, our method does not require intervals, and it can be applied to any function whose values and gradients can be queried.

Our main contribution is a set of criteria that decide whether a simplicial cell needs to be refined, given a particular definition of the implicit complex. The core of these criteria are novel tests that (approximately) check if an n-simplex contains the intersection of m implicit surfaces, and if such intersection can be well approximated by linear interpolation. Our tests can be efficiently and robustly implemented in any dimension n and for any $m \le n$, without the need for interval analysis. These tests become the building blocks of the refinement criteria for an implicit complex, which include additional checks to avoid unnecessary refinement along parts of an implicit surface (or intersection) that does not lie on the complex (e.g., the trimmed portion of a primitive in CSG).

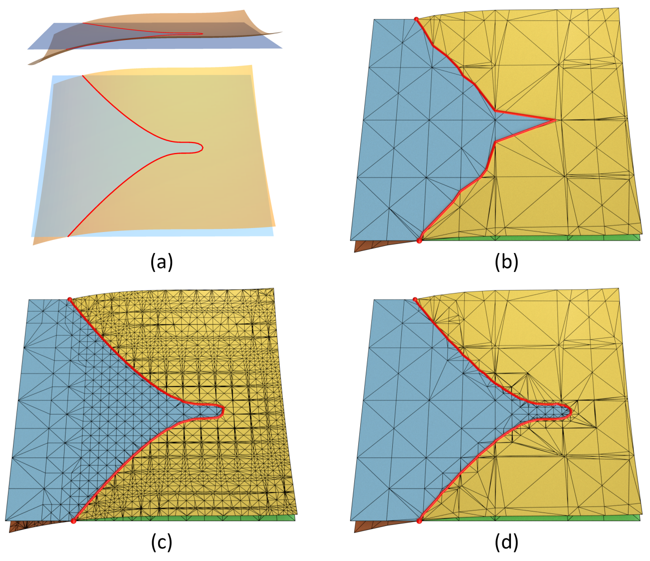

Fig. 2. (a): Two views of a plane (blue, implicit surface of $f(x, y, z) = z$) and a nearly flat surface (yellow, implicit surface of $f(x, y, z) = 0.1 * (x^2 + y^3 - 0.005) - z$), whose intersection curve (red) is far from being flat. (b,c): Discretization on a grid adapted only to the two implicit surfaces either cannot capture well the intersection curve (b) or over-refines the surface (c). (d): Discretization on a grid adapted to both the surfaces and their intersection curve (generated by our method) accurately captures the curve without over-refinement.

Our second contribution is a new method for top-down refinement of a simplicial grid, given some refinement criteria. Our method is a variant of the classic *longest edge bisection* (LEB) method [Plaza and Rivara 2003; Rivara 1991], which bisects a simplex at the midpoint of its longest edge. While LEB generates high quality cells with smoothly varying sizes, the grid can be overly fine for the purpose of discretizing a single implicit shape (see Figure 3 (a)). Our variant produces fewer cells than LEB by allowing cells away from the implicit shape, which are irrelevant to discretization, to have worse shapes and hence higher adaptivity (see Figure 3 (b)).

The rest of the paper is organized as follows. After discussing the previous works in Section 2, we review the implicit representations in Section 3. The technical discussion starts with our tests on the implicit surface intersections in Section 4, followed by our criteria for implicit complexes in Section 5, and it ends with our refinement algorithm in Section 6. We present our experimental results in Section 7 and conclude with a discussion in Section 8.

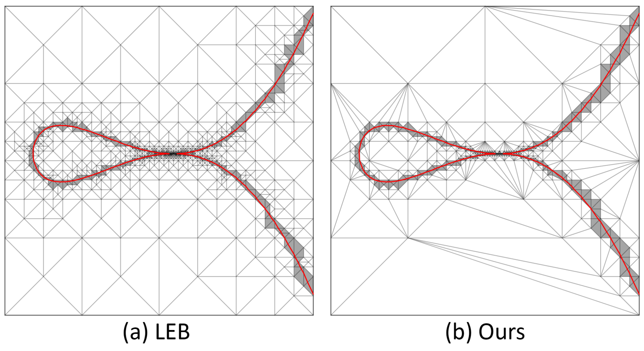

Fig. 3. Refining a triangular grid using the longest edge bisection (LEB) method (a) and our method (b) for discretizing the tear-drop curve [de Figueiredo et al. 2006] (red, implicit curve of $f(x, y) = x^5 + x^4 - 2y^2$). Our method produces much fewer triangles away from the curve, while maintaining the shape and adaptivity of triangles around the curve (gray).

2 PREVIOUS WORK

There is an extensive literature on discretizing implicit shapes. We refer the reader to the excellent and comprehensive survey of De Araújo et al. [2015]. Here we selectively review works that are most relevant to ours, namely methods that discretize the shape using an embedded grid and methods to generate such an embedding. Note that these methods are different from works that produce grids that *conform to* (rather than surround) a given surface [Dey and Slatton 2013; Oudot et al. 2010].

2.1 Grid-based discretization

A common approach to discretize an implicit surface is to divide the domain into *cells* and approximate the surface locally within each cell using polygons [Whitney 1957]. Marching Cubes [Lorensen and Cline 1987] and its many variants adopt uniform cubic cells. The lack of adaptivity of a cubic grid motivates methods that can work on an adaptive octree [Ju et al. 2002; Schaefer and Warren 2004; Varadhan et al. 2004]. Alternatively, adaptivity can be achieved by using simplicial cells (e.g., tetrahedra in 3D), which can tile space with varying sizes. Another advantage of simplicial cells is that the values at the simplex's vertices can be interpolated by a linear function. The linearity enables simple and unambiguous discretization, which is exploited by the Marching Tetrahedra method [Bloomenthal 1988].

The adaptivity and linearity of simplicial grids also make them a popular choice for polygonalizing implicit complexes, including implicit arrangements (IA) [Bagley et al. 2016; Guo et al. 2021; Kim et al. 2000], material interfaces (MI) [Bonnell et al. 2003; Dillard et al. 2007; Nielson and Franke 1997; Saye 2015], spatial curves [Kohlbrenner et al. 2023], and intersections of level sets in higher dimensions [Allgower and Georg 1980; Allgower and Schmidt 1985; Boissonnat et al. 2023; Min 2003; Weigle and Banks 1996].

2.2 Grid generation for discretization

Grid generation is important for fields such as finite element analysis (FEA) [Brandts et al. 2020; George et al. 2017] and data visualization [Borgo et al. 2004]. As the criteria of a "good" grid differ by how the grid is used, we focus on methods designed for discretizing implicit shapes. Such methods (ours included) typically take a top-down refinement approach, where an initial coarse grid is iteratively refined where needed. The key questions that need to be answered are (1) which cells need to be refined, and (2) how to refine them. We review existing solutions for each problem.

2.2.1 Refinement criteria. For the purpose of discretization, a cell is generally regarded as refinable if it contains some part of the implicit shape (often known as the *zero-crossing* test), and if that part is insufficiently approximated by the discretization.

Many refinement criteria have been developed for implicit surfaces. Criteria based on interval analysis allow the refinement algorithm to offer strong topological guarantees [Boissonnat et al. 2008; Chattopadhyay et al. 2012; de Figueiredo et al. 2006; Plantinga and Vegter 2006; Stander and Hart 1997] and/or accurately capture the geometric details [de Figueiredo et al. 2006]. However, tight intervals are difficult to obtain for many practical functions, and impossible when the function is provided as a "black-box" (e.g., a deep neural network), while loose intervals may lead to excessive grid refinement [Chattopadhyay et al. 2012]. Criteria that do not rely on intervals resort to approximate means to measure the curvature of the function [Azernikov and Fischer 2005; Bloomenthal 1988; Hui and Jiang 1999; Molino et al. 2003; Schmidt 1993] or the deviation of the discretization from the implicit surface [Hall and Warren 1990; Liang and Zhang 2014; Petersen et al. 1987; Zhang et al. 2005].

In contrast, refinement criteria for implicit complexes are few and far between. Criteria have been developed to exactly discretize the IA of low-degree polynomials [Dupont et al. 2007; Mourrain et al. 2005; Schömer and Wolpert 2006]. Interval-based criteria can offer topological and/or geometric guarantees in discretizing the intersection of multiple level sets [Allgower and Georg 1989; Boissonnat and Wintraecken 2022] or singularities in the projection of surfaces from $\mathbb{R}^4$ [Diatta et al. 2019]. However, these criteria are not suited for general functions or functions without tight intervals. A few other criteria have been proposed for IA [Kim et al. 2000], CSG [de Miras and Feito 2002; Tobler et al. 1995], and MI [Zhang and Qian 2012], but all of them are concerned with only the surface approximation quality. To the best of our knowledge, there are no interval-free refinement criteria for any implicit complex that consider the approximation quality of the low-dimensional elements in the complex (i.e., intersection curves and points).

2.2.2 Refinement method. For discretization purposes, we consider refinement methods that produce a *conformal* simplicial grid, where adjacent cells share complete faces (i.e., no "hanging" nodes). Another desirable feature of a simplicial grid is well-shaped cells. In the context of discretization, poorly shaped (i.e., near-degenerate) cells are more likely to yield poorly shaped triangles in the discretization. Although such elements can be removed in a post-process, for example by relocating grid points [Hall and Warren 1990; Raman and Wenger 2008]) or remeshing [de Figueiredo et al. 2006; Dillard et al. 2007], excessive amount of near-degenerate elements can be challenging to remove.

One way to produce well-shaped simplices adaptively is regularly subdividing each simplex that needs to be refined, followed by a local refinement of adjacent simplices to maintain a conformal grid [Bey 1995; Grosso et al. 1997; Hall and Warren 1990; Hui and Jiang 1999; Kim et al. 2000; Liu and Joe 1995; Zhang 1995; Zhao et al. 2021]. Note that subdividing a simplex regularly creates many new cells. A more economic approach is *bisection*, which splits a simplex into two halves at the midpoint of a chosen edge. A common choice of the bisecting edge is the longest edge [Plaza and Rivara 2003; Rivara 1991], although other choices have also been investigated [Arnold et al. 2000; Bänsch 1991; Belda Ferrín et al. 2022; Kossaczký 1994; Mitchell 2016]. Like regular subdivision, bisecting a simplex is followed by local refinement of surrounding simplices to maintain conformity. When applied to the Kuhn-subdivision of a square or cube, the longest edge bisection (LEB) method creates simplices that belong to 2 or 3 similarity classes [Maubach 1995]. Efficient data structures have also been developed [Gerstner 2003; Weiss and De Floriani 2009, 2011].

The excellent cell quality, combined with improved compactness over regular simplex subdivision, make LEB ideal for applications such as FEA [Anderson et al. 2021] and data visualization [Gerstner and Pajarola 2000; Gregorski et al. 2002; Pascucci 2004; Zhou et al. 1997]. However, the small number of similarity classes of cells limits the adaptivity of LEB for discretizing implicit shapes. This results in unnecessarily high refinement away from the discretization (see Figure 3 (a)), which we try to avoid in our new method.

3 PRELIMINARIES: IMPLICIT REPRESENTATIONS

Before introducing our method, we briefly review several implicit shape definitions that are considered in this paper.

3.1 Level sets and zero sets

The *level set* of a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ at level $s \in \mathbb{R}$, which we denote by $f^{-1}(s)$, is the loci of points where f evaluates to s:

$$f^{-1}(s) = \{x \in \mathbb{R}^n \mid f(x) = s\} \tag{1}$$

For a smooth function f, the level set is generally a smooth $(n-1)$-dimensional manifold. The level set is better known in computer graphics as the *implicit curve* or *implicit surface* for $n = 2, 3$. We call the level set $f^{-1}(0)$ the *zero set*, which we abbreviate as f^{-1}.

The definition in Equation 1 naturally generalizes to the level set of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$, where $m > 1$, as the loci of points where f evaluates to a vector level s. We shall refer to $f^{-1}(s)$ the *vector level set* when $m > 1$ and *scalar level set* when $m = 1$. Note that f can be considered as a set of m scalar functions $\{f_1, \ldots, f_m\}$, which we call the *components* of f. Geometrically, $f^{-1}(s)$ is the *intersection* of the m scalar level sets of its components,

$$f^{-1}(s) = \bigcap_{i=1}^{m} f_i^{-1}(s_i), \tag{2}$$

where $s = \{s_1, \ldots, s_m\}$. As a result, $f^{-1}(s)$ is a $(n-m)$-dimensional manifold, and it generally only exists for $m \le n$. The vector level set is also known as the *implicit manifold* [Allgower and Georg 1989; Kohlbrenner et al. 2023] or *iso-manifold* [Boissonnat and Wintraecken 2022]. For $n = 3$, the vector level set is the intersection curve ($m = 2$) or point ($m = 3$) of m implicit surfaces. We continue to use f^{-1} to denote the *vector zero set*, $f^{-1}(0)$, which is the intersection of m scalar zero sets f_i^{-1}.

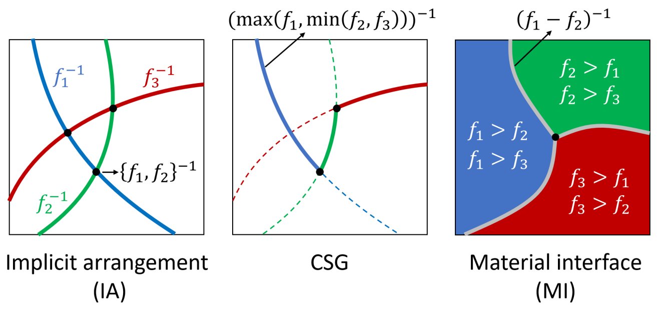

Fig. 4. Illustration of several types of implicit complexes in $\mathbb{R}^2$.

3.2 Implicit complexes

We use the term *implicit complex* to refer generally to any shape composed of zero sets of one or multiple (scalar or vector) functions. Our method is designed for the implicit complexes described below, which have found many applications in computer graphics.

Each of these implicit complexes is defined by a single vector function $f : \mathbb{R}^n \to \mathbb{R}^m$ for some $m > 1$. Specifically, the complex consists of zero sets of either subfunctions of f or their difference bases. We call a function f' a *subfunction* of f, if the components of f' are a subset of components of f. Given a function f' with d components, the *difference basis*, $D(f')$, is a linearly independent set of $d - 1$ pairwise differences between the components of f'.

(1) The *implicit arrangement* (IA) of f is the loci of points where at least one of its component evaluates to zero:

$$IA(f) = \{x \in \mathbb{R}^n \mid 0 \in f(x)\} \tag{3}$$

Geometrically, the IA is the union of zero sets of all subfunctions of f with no more than n components. In $\mathbb{R}^3$, the IA consists of multiple implicit surfaces together with their intersection curves and points. An 2D illustration is given in Figure 4 (right). The IA has been used for representing complex objects with interior partitioning, such as geological structures [Bagley et al. 2016; Guo et al. 2021]. It also includes other implicit complexes as its subsets, such as the ones described below.

(2) *Constructive solid geometry* (CSG) [Requicha and Voelcker 1977], a popular modeling approach, constructs a solid shape from a collection of primitive shapes (e.g., planes, spheres, cylinders, etc.) using set operations (union, intersection, and complement). The boundary of an CSG shape can be implicitly defined by a vector function f, whose components encode the primitives as scalar zero sets, and an CSG expression π consisting of max, min and negation operators that encode the set operations. The CSG boundary is the scalar zero set of the composition $\pi \circ f$:

$$CSG(f) = (\pi \circ f)^{-1} = \{x \in \mathbb{R}^n \mid \pi(f(x)) = 0\} \tag{4}$$

Since the composition $\pi \circ f$ is typically only C^0 continuous, the CSG boundary is a piecewise-smooth $(n-1)$-dimensional manifold. Each smooth $(n-1)$-dimensional piece is a subset of the scalar zero set of a component of f, and adjacent smooth pieces meet at vector zero sets of subfunctions of f. By definition, $CSG(f) \subseteq IA(f)$ (see illustration in Figure 4 middle).

(3) The *material interface* (MI) of f partitions a domain into regions where one component of f *dominates*; that is, some f_i is strictly greater than all other f_j ($j \neq i$) in that region. Equivalently, MI is the loci where multiple components of f attain the maximum,

$$MI(f) = \{x \in \mathbb{R}^n \mid |\arg\max_i f_i(x)| > 1\} \tag{5}$$

The MI is a network of $(n-1)$-dimensional manifolds meeting at non-manifold junctions. Each $(n-1)$-dimensional piece separates regions dominated by two components of f (say f_i, f_j), and it lies on the scalar zero set of their difference ($f_i - f_j$). The junction where more than two regions meet lies on the vector zero set of the difference basis of the subfunction of f whose components dominate those regions (see illustration in Figure 4 left). The MI is a subset of the IA of all pairwise differences of the components of f. MI has been widely used for representing complex objects with interior partitions, such as multi-phase fluids [Kim 2010; Losasso et al. 2006], composite materials [Dillard et al. 2007; Shammaa et al. 2010], and anatomical structures [Bertram et al. 2005; Zhang et al. 2008].

(4) Finally, the lower-dimensional elements of an implicit complex form an implicit complex themselves, which consist exclusively of vector zero sets. This includes the intersections of two or more scalar zero sets in IA, the locations where two or more smooth pieces meet in CSG, and the junction where more than two regions meet in MI. In $\mathbb{R}^3$, the intersection curves of implicit surfaces have been used for visualizing line features in volume data [Burns et al. 2005; Edelsbrunner and Harer 2002] and reconstructing lower-dimensional shapes [Kohlbrenner et al. 2023].

4 REFINEMENT CRITERIA FOR ZERO SETS

We start by presenting our refinement criteria for zero sets of scalar and, more importantly, vector functions, which are the building blocks of implicit complexes. Our criteria answer the following general question: given a function $f : \mathbb{R}^n \to \mathbb{R}^m$ for any dimension n and $m \in [1, n]$ and an n-simplex t, does t need to be refined to better discretize the zero set f^{-1}? In $\mathbb{R}^3$, our criteria consider a single implicit surface ($m = 1$), the intersection curve of two surfaces ($m = 2$), or the intersection point of three surfaces ($m = 3$).

Our criteria follow the same principles as existing ones for implicit surfaces. That is, refinement is not needed if either t does not contain any part of f^{-1}, or if f^{-1} is already close enough to the discretization. Unlike existing criteria, our criteria are formulated so that they are generalizable to vector zero sets and also can be efficiently computed. Specifically, let $\overline{f}$ be the linear approximation of f inside t by barycentric interpolation of values of f at the vertices of t. We consider the zero set of $\overline{f}$, $\overline{f}^{-1}$, as the discretization of f^{-1} in t. We deem t *refinable for* f^{-1} if it passes two tests:

- *Zero-crossing test:*

$$f^{-1} \cap t \neq \emptyset \tag{6}$$

- *Distance test:*

$$d_H(f^{-1} \cap t, \overline{f}^{-1}) > \epsilon, \tag{7}$$

where $d_H(X, Y) = \sup_{x \in X} d(x, Y)$ is the one-sided Hausdorff distance from X to Y, and ϵ is a user-defined threshold.

The distance test checks if the discretization is too far from the zero set. Our specific choices in the test, including the use of the one-sided Hausdorff distance and the complete zero set $\overline{f}^{-1}$ (instead of its portion in t), are made to enable efficient computations, as we shall see below.

While these tests can be performed using interval analysis, intervals on the values and derivatives are not always available for arbitrary functions. Instead, we perform the tests on local approximations of f, called *proxies*, that can be constructed by sampling the functions at spatial locations (Section 4.1). Our design of the proxy leads to efficient implementation of both tests (Sections 4.2 and 4.3); see Equations 9 and 15.

4.1 Proxy construction

We consider the class of functions $\widetilde{f}$ represented as a *convex combination with linear precision* as our proxy. Such a function is defined by *control points* $p_1, \ldots, p_l$ that lie inside or on the boundary of the simplex t, where each p_i is associated with a *control value* (a length-m vector) $b^i = \{b^i_1, \ldots, b^i_m\}$. The function, $\widetilde{f}$, is a weighted average of b^i,

$$\widetilde{f}(x) = \sum_{i=1}^{l} w_i(x) b^i, \tag{8}$$

where the weights $w_i(x)$ satisfy, for all $x \in t$:

(1) Convexity: $w_i(x) \geq 0$ for $i = 1, \ldots, l$, and $\sum_{i=1}^{l} w_i(x) = 1$.
(2) Linear precision: $\sum_{i=1}^{l} w_i(x) p_i = x$.

The convexity of the weights ensures that $\widetilde{f}(x)$ *lies in the convex hull of* $b = \{b^1, \ldots, b^l\}$, where each b^i is treated as a point in $\mathbb{R}^m$. With linear precision, $\widetilde{f}$ can *reproduce the linear interpolation function* $\overline{f}$ by giving b^i the linearly interpolated value $\overline{f}(p_i)$, which we denote as $\overline{b}^i$. These properties will be exploited later in implementing the two tests. An example of such $\widetilde{f}$ is the *Bezier simplex* [Farin 1986], where the control points lie on a regular grid in t and the weights w_i are the Berstein polynomials. One could also place the control points in arbitrary locations on the boundary of t and use one of the *generalized barycentric coordinates* [Floater 2015] as the weights (as long as the coordinates are positive).

The refinement criteria to be introduced in Sections 4.2 and 4.3 apply to any proxy satisfying the properties above. In our implementation, we adopt the *cubic Bezier simplex* as the proxy. The control points p_i in a cubic Bezier simplex consist of all vertices of t, the two trisectors on each edge of t, and the centroid of each triangle face of t (see Figure 5). Although such a proxy can only reproduce cubic polynomials, we have found it to be effective in approximating a variety of non-cubic, and even non-polynomial functions (see Section 7). We obtain the control values b^i, known as the *Bezier ordinates*, by evaluating f and its gradient at the simplex vertices (see details in Appendix A).

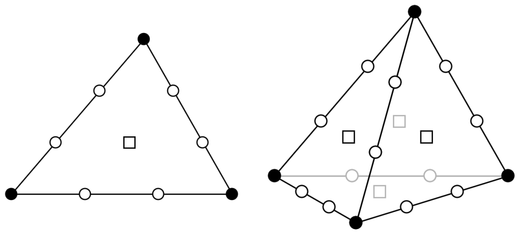

Fig. 5. The control points of a cubic Bezier triangle (left) and tetrahedron (right), consisting of vertices (dots), edge trisectors (circles) and face centroids (squares).

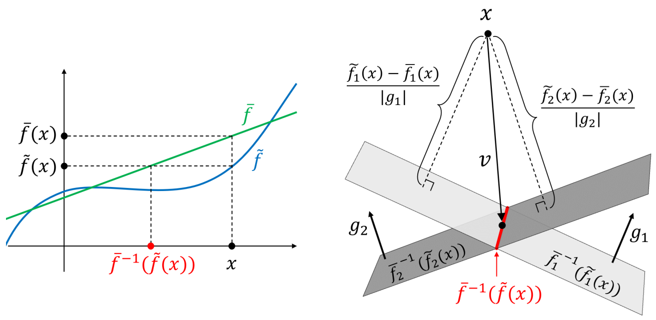
$\bar{f}(x)$
$\tilde{f}(x)$
$\bar{f}$
$\tilde{f}$
$\bar{f}^{-1}(\tilde{f}(x))$
x
x
$\frac{\tilde{f}_1(x) - \bar{f}_1(x)}{|g_1|}$
$\frac{\tilde{f}_2(x) - \bar{f}_2(x)}{|g_2|}$
v
g_2
g_1
$\bar{f}_2^{-1}(\tilde{f}_2(x))$
$\bar{f}_1^{-1}(\tilde{f}_1(x))$
$\bar{f}^{-1}(\tilde{f}(x))$

Fig. 6. Illustration for computing the distance from a point x to the level set $\overline{f}^{-1}(\widetilde{f}(x))$ for a scalar function in 1D (left) and a two-component function in 3D (right).

4.2 Zero-crossing test

We now examine the refinement criteria on a proxy function $\widetilde{f}$. The zero-crossing test checks if $\widetilde{f}^{-1} \cap t \neq \emptyset$. As discussed in the previous section, the value $\widetilde{f}(x)$ at any point $x \in t$ lies in the m-dimensional convex hull of b, the control values of $\widetilde{f}$. As a result, if there exists some point x such that $\widetilde{f}(x) = 0$, then 0 lies in this convex hull. So a necessary condition to pass the zero-crossing test is

$$\boxed{O \in CH(b)} \tag{9}$$

where O is the origin of $\mathbb{R}^m$ and $CH(b)$ is the convex hull of b.

The test in Equation 9 is a generalization of the zero-crossing test for implicit surfaces (i.e., $m = 1$) that has been commonly used in previous works [Hall and Warren 1990; Petersen 1984], which checks if the range of control values b encloses 0. For $m > 1$, the test amounts to checking if O is *not* an *extreme point* in the union set $O \cup b$. The extremity check can be written as a linear programming (LP) problem with the same number of constraints as the number of points (l in our case) [Ottmann et al. 1995], which can be solved in time linear to l if the space dimension (m in our case) is fixed [Megiddo 1984]. For lower dimensions (e.g., $n = 2, 3$), we use a faster, brute-force implementation that will be described in Section 7.4.

4.3 Distance test

We next turn to the distance test on the proxy $\widetilde{f}$, which checks if

$$d_H(\widetilde{f}^{-1} \cap t, \overline{f}^{-1}) > \epsilon, \tag{10}$$

where $\overline{f}$ is the linear approximation of f in t. While the exact (one-sided) Hausdorff distance between the two zero sets can be difficult to compute, we observe that it has the following upper bound:

$$\begin{aligned} d_H(\widetilde{f}^{-1} \cap t, \overline{f}^{-1}) &= \max_{x \in \widetilde{f}^{-1} \cap t} d(x, \overline{f}^{-1}(0)) \\ &= \max_{x \in \widetilde{f}^{-1} \cap t} d(x, \overline{f}^{-1}(\widetilde{f}(x))) \\ &\leq \max_{x \in t} d(x, \overline{f}^{-1}(\widetilde{f}(x))) \end{aligned}$$

(The second equality holds because $\widetilde{f}(x) = 0$ for all $x \in \widetilde{f}^{-1} \cap t$). In other words, the upper bound is the maximal distance from any point $x \in t$ to the *level set* of the linear function $\overline{f}$ at the level $\widetilde{f}(x)$. In the following, we will show that this maximum distance is attained at a control point of $\widetilde{f}$, and hence the test in Equation 10 can be expressed solely by the control values b.

We start by deriving an explicit form for the upper bound. It helps to first consider the simple case of $m = 1$. Since $\overline{f}$ is a scalar function, its level sets are $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$ orthogonal to the gradient $g = \nabla\overline{f}$. The distance from x to the level set of $\overline{f}$ at level $\widetilde{f}(x)$ (i.e., $\overline{f}^{-1}(\widetilde{f}(x))$) is $(\widetilde{f}(x) - \overline{f}(x))/|g|$ (assuming $g \neq 0$). Note that this distance is *signed*, and a positive distance implies that x is on the opposite side of the level set as g. Figure 6 (left) gives an illustration for the 1-dimensional case.

Now consider $m > 1$. The vector level set $\overline{f}^{-1}(\widetilde{f}(x))$ is the intersection of m scalar level sets $\overline{f}_i^{-1}(\widetilde{f}_i(x))$, each being a hyperplane orthogonal to a gradient vector $g_i = \nabla\overline{f}_i$ (see illustration in Figure 6 (right)). Let v be the vector from x to its nearest point on this intersection. Since the signed distance from each scalar level set to x is $(\widetilde{f}_i(x) - \overline{f}_i(x))/|g_i|$, we have the identities[1],

$$\frac{g_i^T \cdot v}{|g_i|} = \frac{\widetilde{f}_i(x) - \overline{f}_i(x)}{|g_i|}, \quad \forall i = 1, \ldots, m. \tag{11}$$

Furthermore, v must lie in the subspace spanned by the basis $g = \{g_1, \ldots, g_m\}$. We can therefore find v as

$$v = M(\widetilde{f}(x) - \overline{f}(x)), \tag{12}$$

where $M = g(g^T g)^{-1}$. The distance $d(x, \overline{f}^{-1}(\widetilde{f}(x)))$ can be written as

$$d(x, \overline{f}^{-1}(\widetilde{f}(x))) = |v| = |M(\widetilde{f}(x) - \overline{f}(x))|. \tag{13}$$

To find the maximal distance, observe that both $\widetilde{f}(x), \overline{f}(x)$ are convex combinations with the same weights. This is because $\overline{f}$ can be reproduced by $\widetilde{f}$ when the control values b are the barycentric interpolants $\overline{b}$, as discussed in Section 4.1. As a result, the difference $\widetilde{f}(x) - \overline{f}(x)$ is a convex combination of values $b - \overline{b}$, and hence it lies inside the convex hull of $b - \overline{b}$ (treated as points in $\mathbb{R}^m$). Since linear transformations (such as M) preserve convexity, v lies in the convex hull of the transformed points $M(b - \overline{b})$. Finally, utilizing

[1]We use the convention in this paper that all vectors, such as v and g_i in Equation 11, are column vectors.

the fact that the maximum distance from the origin to a convex hull is realized at its vertices, we arrive at the bound:

$$d(x, \overline{f}^{-1}(\widetilde{f}(x))) \leq \max_{i=1,\ldots,l} |M(b^i - \overline{b}^i)|. \tag{14}$$

Therefore, a necessary condition to pass the distance test of Equation 10 is

$$\boxed{\max_{i=1,\ldots,l} |M(b^i - \overline{b}^i)| > \epsilon} \tag{15}$$

In the case of scalar zero sets ($m = 1$), the inside part of the max operator in Equation 15 reduces to $|(b^i - \overline{b}^i)|/|g|$, where the b^i and $\overline{b}^i$ are scalar values and g is the gradient vector of $\overline{f}$. This test is similar to tests proposed by other researchers [Gregorski et al. 2002; Petersen et al. 1987; Zhang et al. 2005], who also consider bounding the difference between a function and its linear approximation after a gradient-based normalization. Unlike previous works, our test can be applied generally to any vector zero set ($m > 1$).

One issue that arises in implementing the test in Equation 15 is that it involves inverting matrices that could become singular. For example, computing the gradient g needs to invert a matrix composed of the edge vectors of simplex t, which is singular if t is degenerate (i.e., has no n-dimensional measure). Computing M requires inverting $g^T g$, which is singular if $\text{rank}(g) < m$ (e.g., a component of f has zero gradient, or two components have co-linear gradients, etc.). To address the issue, we can rewrite the test in an inversion-free form (see Appendix B). The re-formulation not only is robust against singular matrices, but also exhibits improved numerical accuracy when the matrices are almost singular.

5 REFINEMENT CRITERIA FOR IMPLICIT COMPLEXES

Building on the refinement criteria for zero sets, we now describe criteria for the implicit complexes discussed in Section 3. Recall that such a complex M is defined by a vector function $f : \mathbb{R}^n \to \mathbb{R}^m$ where m can be any positive integer (possibly greater than n), and it consists of zero sets of either the subfunctions of f or their difference bases. We start by presenting a general structure of the criteria that is shared by all implicit complexes being considered (Section 5.1), and then discuss details specific to each complex (Section 5.2).

5.1 General criteria

Similar to the refinability for zero sets, a simplex t is considered *refinable for M* if it contains some portion of M and if this portion is too far from its discretization (i.e., the implicit complex defined by the linear approximation $\overline{f}$).

A straight-forward way to check refinability is to see if t is refinable for any zero set that makes up M. However, this would lead to over-refinement if M contains *partial* zero sets, as in the cases of CSG and MI. For example, a 3D CSG shape consists of patches of implicit surfaces that are trimmed by other implicit surfaces. To avoid refining around the part of a zero set that does not belong to M, we identify component functions of f that contribute to the portion of M in t as *active components*. The definition of an active component differs by the type of the implicit complex, which will be introduced in the next section. We deem t refinable if *it is refinable for any zero set making up M that involves only active components of f in t.*

To efficiently check refinability, we observe that the zero set of a function h (or its difference basis, $D(h)$) intersects t only if the zero set of *every* subfunction of h also intersects t. We therefore examine the subfunctions of h in the order of *increasing* number of components, and we ignore a function h with d components if t fails the zero-crossing test for any subfunction (or its difference basis) of h with $d - 1$ components. The pseudo-code of our implementation is given in Algorithm 1. Here, gather(F_d) produces a list of $(d + 1)$-component subfunctions h of f such that every d-component subfunction of h is in F_d.

Our criteria accommodate a separate distance threshold ϵ_d for checking zero sets of a d-component function, for each $d \in [1, m]$. This makes it possible to refine the grid only for discretizing the lower-dimensional elements of an implicit complex (e.g., the sharp features in CSG or the junction graph in MI). Specifically, setting $\epsilon_i = \infty$ for all $i < n - d$ will prevent the grid to be refined around zero sets whose dimension is greater than d.

Algorithm 1: Refinement criteria for IA, CSG or MI

```
Input: $n$-dimensional simplex $t$
Input: Function $f : \mathbb{R}^n \to \mathbb{R}^m$
Input: Distance thresholds $\epsilon = \{\epsilon_1, \ldots, \epsilon_n\}$
Output: True if $t$ is refinable; False otherwise
/* This color: used only in IA and CSG */;
/* This color: used only in MI */;
$F_1 \leftarrow$ active components of $f$ in $t$;
forall $h \in F_1$ do
    if $t$ passes distance test for $\{h^{-1}, \epsilon_1\}$ then
        return True
    end
end
for $d = 2, \ldots, n$ (or $n + 1$) do
    $F_d \leftarrow \emptyset$;
    forall $h \in gather(F_{d-1})$ do
        if $t$ passes zero-crossing test for $h^{-1}$ (or $D(h)^{-1}$) then
            $F_d \leftarrow F_d \cup \{h\}$;
            if $t$ passes distance test for $\{h^{-1}, \epsilon_d\}$ (or
              $\{D(h)^{-1}, \epsilon_{d-1}\}$) then
                return True
            end
        end
    end
end
return False;
```

5.2 Active components

We now detail, for each type of implicit complex M, the criteria for a component f_i of f to be active in t. To avoid missing geometric details, our criteria are intentionally *conservative*, in that we want to identify all f_i that *possibly* contribute to the portion of M in t. Our criteria will again utilize the proxy function $\widetilde{f}$, as constructed in Section 4.1, and particularly its control values b. Note that, since $\widetilde{f}$ is a convex combination of b, the value of a component $\widetilde{f_i}(x)$ at any point $x \in t$ is bounded within the range of the control values b_i, which we denote by r_i. Our criteria for active components will be expressed by this range.

- *IA:* Since $IA(f)$ (Equation 3) contains only complete zero sets, we deem a component f_i active if its scalar zero set intersects t. Using the zero-crossing test (Equation 15), we only need to check if $0 \in r_i$.
- *CSG:* Recall that $CSG(f)$ (Equation 4) is the scalar zero set of the composition $\pi \circ f$, where π is some CSG expression consisting of min, max, and negation operations on the components of f. We consider a component active if its scalar zero set *might* lie on $(\pi \circ f)^{-1} \cap t$, if it is not empty. Leveraging the hierarchical structure of π, we collect the list of active components in a recursive manner. We first estimate the range of $\pi \circ f$, denoted by $r(\pi)$, recursively on π following [Duff 1992; Hijazi et al. 2010],

$$r(\pi) = \begin{cases} [\triangle(r^l(\pi_1), r^l(\pi_2)), \triangle(r^h(\pi_1), r^h(\pi_2))], & \text{if } \pi = \triangle(\pi_1, \pi_2) \\ [\neg(r^h(\pi')), \neg(r^l(\pi'))], & \text{if } \pi = \neg(\pi') \\ r_i, & \text{if } \pi = f_i \end{cases}$$

  where $\triangle$ can be either min or max, $\neg$ is negation, and r^l, r^h denotes the lower and higher ends of a range r. We then recursively collect the list of active components for an expression π, denoted by $A(\pi)$, as the union of the active lists for the operands of π, if $r(\pi)$ contains zero, and empty otherwise:

$$A(\pi) = \begin{cases} A(\pi_1) \cup A(\pi_2), & \text{if } \pi = \triangle(\pi_1, \pi_2) \text{ and } 0 \in r(\pi) \\ A(\pi'), & \text{if } \pi = \neg(\pi') \text{ and } 0 \in r(\pi) \\ \{f_i\}, & \text{if } \pi = f_i \text{ and } 0 \in r_i \\ \emptyset, & \text{otherwise} \end{cases}$$

- *MI:* Recall that $MI(f)$ (Equation 5) partitions the space into regions within which one component of f dominates the others. We therefore consider a component f_i active if it dominates all other components at *some* point in t. A necessary condition is that there is no range r_j ($j \neq i$) that is *strictly higher* than r_i. Here, we say a range $[a_1, b_1]$ is strictly higher (or lower) than another range $[a_2, b_2]$ if $a_1 > b_2$ (or $b_1 < a_2$). It is easy to verify that the ranges in the set $R = \{r_1, \ldots, r_m\}$ satisfying this condition are those whose higher end is no smaller than the highest lower end among all ranges in R. These ranges, which belong to active components, can be efficiently found by two sweeps over R, first identifying the highest lower end and then checking it against each range.

6 SIMPLICIAL REFINEMENT

We now consider *how* to refine a simplicial grid using the criteria described in the previous sections. We adopt the top-down refinement regime that is common in grid generation. Given an initial grid and some criteria (e.g., Algorithm 1), our goal is to produce a maximally refined grid where no cell is deemed refinable by the criteria. While our implementation is specialized for the practically useful cases of $n = 2, 3$, our methodology is not specific to dimensionality.

Our method is a variation of the classical *longest edge bisection* (LEB) method. Recall from Section 2.2.2 that this scheme refines a simplicial cell by splitting it into two at the mid-point of its longest edge, followed by a local refinement of surrounding cells to restore a conformal grid. To avoid the unnecessary refinement in LEB away from the implicit shape, as seen in Figure 3 (a), our key idea is to relax the requirement that *every* cell has to be split by its longest edge. Instead, for any chosen edge e, we bisect *all* cells incident to e at the mid-point of e, regardless of whether e is the longest edge of that cell (see illustration in Figure 7). Note that the operation maintains the conformity of the grid, which avoids the local refinement in LEB. Our choice of the bisecting edge e is motivated by the need to maintain cell quality. Specifically, we call an edge *refinable* if any of its incident cells is refinable according to the given refinement criteria. At each iteration of the algorithm, we bisect the longest refinable edge in the current grid. We call our method the *longest refinable edge bisection* (LREB) method.

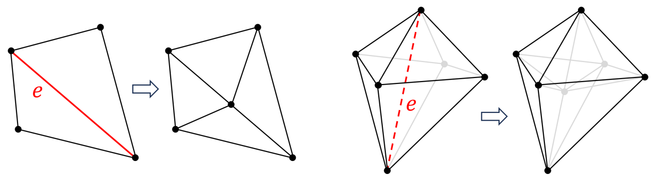

Fig. 7. Bisecting triangles (left) and tetrahedra (right) incident to an edge e.

We have observed that LREB produces more adaptive and compact grids than LEB, as seen in Figure 3 (b). The adaptivity is made possible by larger and more irregularly-shaped cells away from the implicit geometry. Although lacking a theoretical bound, we provide experimental evidence in Section 7.3 that the worst quality among cells enclosing the implicit shape produced by LREB appears to be lower-bounded for a variety of implicit surfaces and complexes.

We give the pseudo-code of our complete refinement algorithm in Algorithm 2. We use a priority queue to maintain all refinable cells, prioritized by the length of each cell's longest edge. Each iteration splits the cell at the head of the queue by its longest edge (which is the longest refinable edge in the grid), as well as all cells (refinable or not) sharing that edge. New cells are checked for refinability and the refinable ones are placed into the queue. The algorithm stops when the queue is empty or some termination criteria is met (e.g., the longest refinable edge is shorter than a given threshold L).

7 RESULTS

We show results of our refinement method on implicit complexes in both 2D and 3D. In all examples, the domain is a unit square (in 2D) or cube (in 3D). To create the initial grid (G_{init} in Algorithm 2), we adopt the Kuhn-subdivision, which splits a square into two triangles and a cube into six tetrahedra, although our algorithm is not specific to this initialization. Unless stated otherwise, the minimum edge length threshold (L in Algorithm 2) is set to 0, and hence the algorithm terminates only when the grid is fully refined (i.e., no cell is deemed refinable). We adopt the discretization method of [Du et al. 2022], which computes the exact IA and MI of the piecewise linear functions in 2D and 3D. CSG is discretized by first computing the IA and extracting the relevant subset.

Algorithm 2: Grid refinement using LREB

```
Input: $n$-dimensional initial simplicial grid $G_{init}$
Input: Function $f : \mathbb{R}^n \to \mathbb{R}^m$
Input: Distance thresholds $\epsilon = \{\epsilon_1, \dots, \epsilon_n\}$
Input: Edge length threshold $L$
Output: A grid for discretizing $f^{-1}$
$G \leftarrow G_{init}$;
$Q \leftarrow \emptyset$ /* priority queue */;
forall cell $t \in G$ do
    if $\{t, f, \epsilon\}$ is refinable by Algorithm 1 then
        $Q.push(t, len(t))$ /* $len(t)$: longest edge length in $t$ */;
    end
end
while $Q \neq \emptyset$ do
    $t \leftarrow Q.pop()$;
    if $len(t) < L$ then
        break;
    end
    $e \leftarrow$ longest edge in $t$;
    $T \leftarrow$ all cells in $G$ incident to $e$;
    forall $s \in T$ do
        $Q.remove(s)$;
        $\{s_1, s_2\} \leftarrow$ split cells of $s$ after bisecting $e$;
        $G \leftarrow G \setminus \{s\} \cup \{s_1, s_2\}$;
        for $i = 1, 2$ do
            if $\{s_i, f, \epsilon\}$ is refinable by Algorithm 1 then
                $Q.push(s_i, len(s_i))$;
            end
        end
    end
end
return $G$;
```

7.1 Zero-crossing and distance tests

We start by evaluating the effectiveness of our two tests, zero-crossing test and distance test (Section 4), on zero sets of functions with different number of components.

We first examine the tests on a scalar zero set. Although they are similar to previous tests in the literature, we include them not only for completeness, but also since their behavior echoes those we shall see on vector zero sets. Figure 8 shows the refined grids for the same implicit curve used in Figure 3, by only performing the zero-crossing test (with some $L > 0$ to avoid infinite refinement) or the distance test. Observe that the zero-crossing test alone results in highly adaptive triangles away from the curve, as desired, but over-refines the region along the curve. On the contrary, the distance test alone creates desirable cell sizes and shapes along the curve, but over-refines the rest of the domain. The latter is because the distance test considers the level sets at all levels (not just the zero set). Combining both tests yields well-shaped and adaptive cells along the implicit curve while avoiding over-refinement away from the curve, as seen in Figure 3 (b).

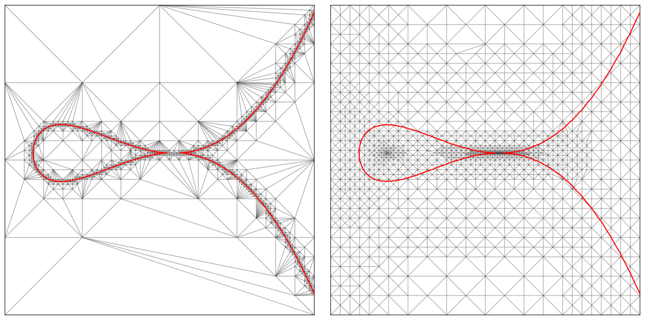

Fig. 8. Refined grids and resulting discretization for the tear-drop curve by performing only the zero-crossing test (a) or only the distance test (b). The result of using both tests is shown in Figure 3 (b).

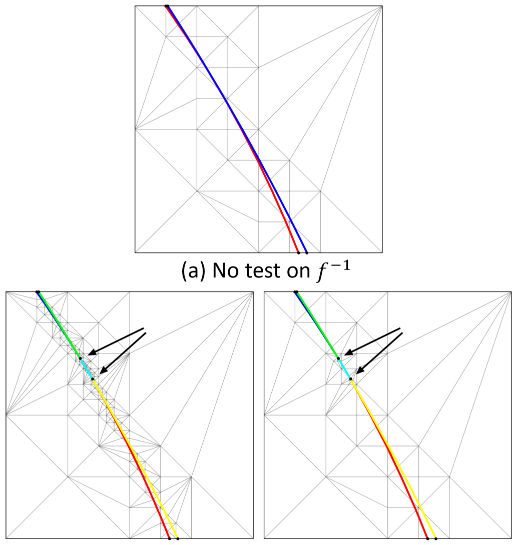

Fig. 9. The refined grids and resulting discretizations of the IA of a 2-component function $f = \{f_1, f_2\}$ in 2D created using different settings: checking only the criteria on each scalar level set f_i^{-1} (a), also performing the distance test on the vector level set f^{-1} (b), and performing both zero-crossing and distance tests on f^{-1} (c). Each curve segment in the IA is assigned a different color, and arrows point to f^{-1}.

We now examine the tests on vector zero sets. We start with a 2-component function $f = \{f_1, f_2\}$ in 2D in Figure 9. The implicit curves f_1^{-1}, f_2^{-1} are circles with large radii and are almost tangent at their intersection points (f^{-1}). As the curves are nearly flat, checking only the criteria on each curve leads to a coarse grid, where the intersection points are missed by the discretization (see (a)). If we additionally perform the distance test on the intersection points (f^{-1}), we obtain a more refined grid that recovers the intersection points (see the arrows in (b)), but the refinement extends to everywhere that the two curves are close by. By also performing the

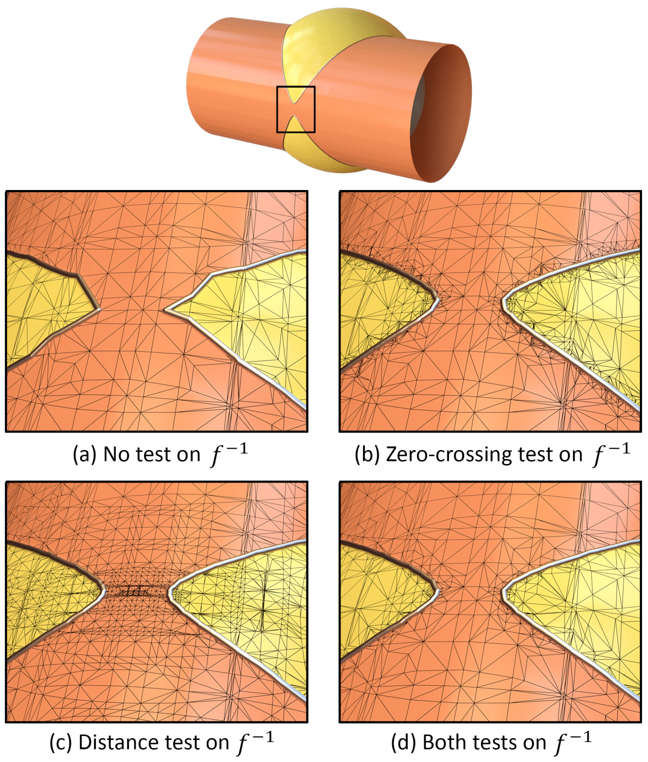
(a) No test on f^{-1}
(b) Zero-crossing test on f^{-1}
(c) Distance test on f^{-1}
(d) Both tests on f^{-1}

Fig. 10. Discretization of the IA of a 2-component function $f = \{f_1, f_2\}$ in 3D (top) on grids refined under three settings (showing zoomed-in views): checking only the criteria on each f_i^{-1} (a), adding only the zero-crossing test on f^{-1} (b), adding only the distance test on f^{-1} (c), and performing both tests on f^{-1} (d). Each patch in the IA is assigned a different color.

zero-crossing test on the intersections, our algorithm localizes the refinement to the vicinity of the intersection points (see (c)).

We show a 3D example of 2-component function in Figure 10. The implicit surfaces f_1^{-1}, f_2^{-1} consist of a sphere and a cylinder, whose intersection curve (f^{-1}) has a region of high curvature that is poorly discretized if we only check the refinement criteria on each surface. Adding the zero-crossing test on the intersection curve (with some $L > 0$) results in densely refined cells everywhere along the curve, whereas performing only the distance test can adapt the level of refinement to the curvature of the curve but at the cost of over-refining nearby regions where the two surfaces are close. By combining both tests on the intersection curve, our algorithm captures the intersection curve without unnecessary refinement.

We next consider a 3-component function example in Figure 11. The implicit surfaces $f_1^{-1}, f_2^{-1}, f_3^{-1}$ are three intersecting cylinders whose axes are almost parallel. These cylinders intersect at several near-coinciding curves (zero sets of 2-component subfunctions), which in turn intersect at two points f^{-1} (red dots in the top view). However, if we only check the refinement criteria on the surfaces and their intersection curves, the intersection points are missed on the discretization (see (a)). This is because both the surfaces and the intersection curves are nearly "flat" and hence a coarse grid resolution is sufficient to approximate them well. The two

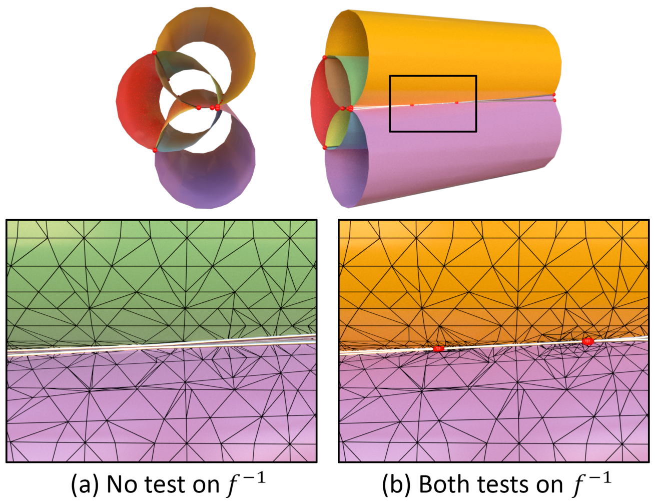

Fig. 11. Discretization of the IA of a 3-component function f in 3D (top; showing two views) on grids refined without checking the refinement criteria on the 3-vector zero set f^{-1} (but checking all scalar and 2-vector zero sets) (a), and with the criteria turned on (b).

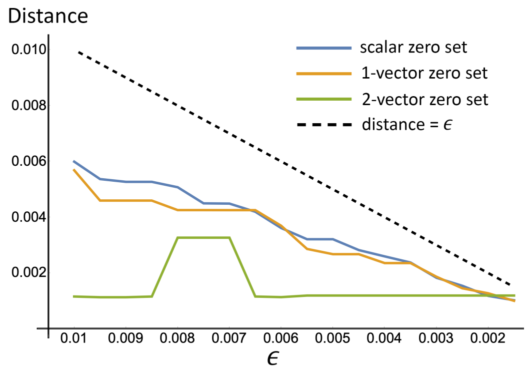

Fig. 12. The two-sided Hausdorff distance between the analytical and discrete zero sets of a scalar function (a sphere of radius 0.3), a 2-component function (Figure 10), and a 3-component function (Figure 11), over decreasing refinement threshold ϵ.

intersection points are recovered only after we additionally perform the tests on them, which refines the grid in their vicinity (see (b)).

To further validate our distance test, we measure the Hausdorff distance between a zero set and its discretization on a refined grid. We consider zero sets with analytical forms, including the zero set of a scalar function (a sphere with radius 0.3), a 2-component function (the intersection curve in Figure 10), and a 3-component function (the two intersection points in Figure 11). Although our distance test is based on the one-sided Hausdorff distance, we measure the two-sided (symmetric) Hausdorff distance for validation, and we do so at a range of threshold values. As shown in Figure 12, the distance between the actual zero set and the discretization remains consistently lower than the threshold value used.

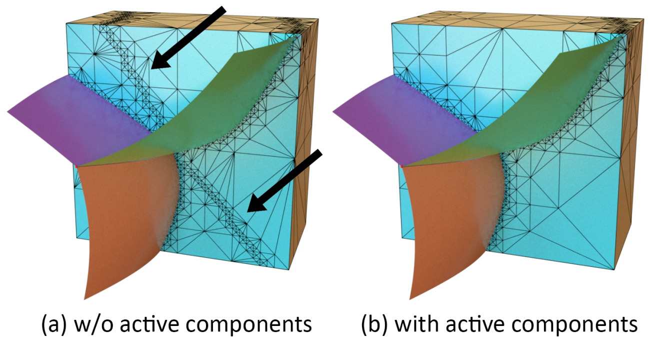

Fig. 13. Comparing discretization of an MI (the Voronoi diagram of three spheres) on a grid refined by applying the refinement criteria to all components (a) and only active components (b). Observe the over-refinement in (a), pointed by arrows.

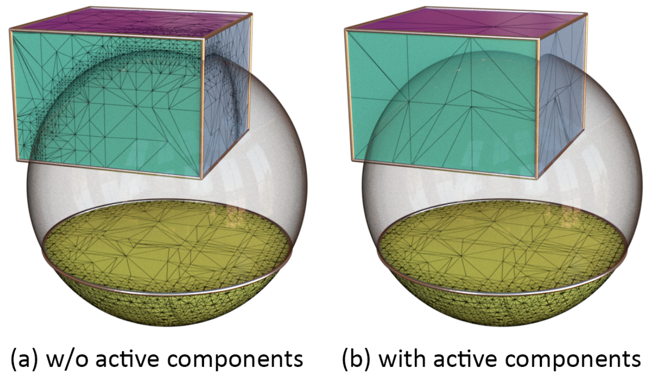

Fig. 14. Comparing discretization of a CSG shape, consisting of a cube and a truncated sphere (the complete sphere is shown in transparency), on a grid refined by only avoiding empty cells (a) and by considering only active components in each cell (right). Observe the over-refinement in (a) where the sphere intersects the cube.

7.2 Active components

We evaluate the effectiveness of active components, which are used in our refinement criteria to avoid unnecessary refinement for implicit complexes that contain partial zero sets, such as CSG and MI (Section 5.2).

Figure 13 shows a simple MI in 3D, as the Voronoi diagram of three spheres with different radii (also known as the Apollonius Diagram). Without restricting the refinement criteria to only active components (i.e., by treating every component of f as active in every cell), the grid is refined both over the MI patches and along their extensions (see arrows in (a)). These extensions lie on the zero set of the difference between a pair of components, but they are dominated (i.e., having lower values than) by another component. The over-refinement is avoided by considering only active components.

An alternative approach for avoiding unnecessary refinement in discretizing CSG is to ignore *empty* grid cells that do not intersect with the CSG surface [Duff 1992; Hijazi et al. 2010]. However, this approach still considers *all* components of f in a non-empty cell, even though they do not all contribute to the shape. This can result in over-refinement where a primitive (e.g., the cube faces in Figure 14 (a)) intersects the trimmed-away part of another primitive (e.g., the sphere in Figure 14 (a)). In contrast, our refinement criteria can reduce such interference by considering only active components in a non-empty cell, as shown in Figure 14 (b).

7.3 Simplicial refinement

We compare the compactness of the grid between our refinement method (LREB) and the classical longest edge bisection (LEB) method (as implemented in the MFEM package [Anderson et al. 2021]) on an implicit surface example in Figure 15. Observe that LREB consistently produces around a third of the total number of cells produced by LEB at the same threshold.

The compactness of LREB comes at the cost of reduced cell quality, particularly those away from the implicit shape. To evaluate the quality of cells around the shape, we took all 3D shapes that we have used so far in the paper and measured the worst quality of the cells around the shape as the distance threshold ϵ decreases to a very small amount (0.01% of the bounding box size). Here the quality of

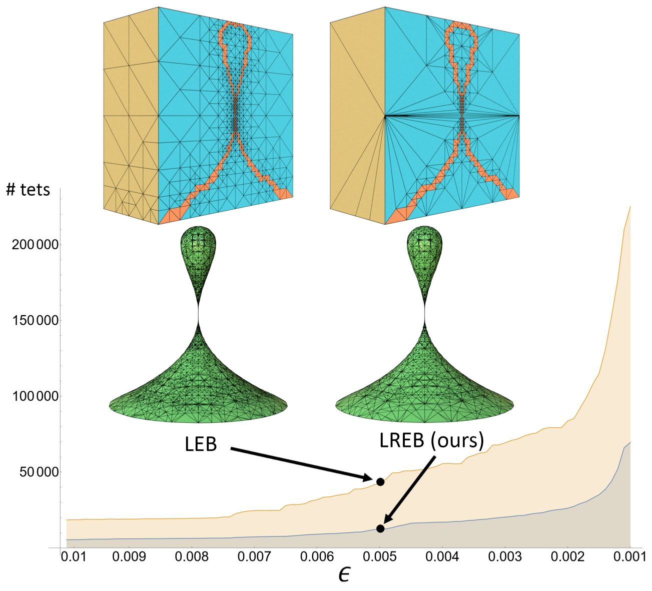

Fig. 15. Comparing the longest edge bisection (LEB) method and our longest refinable edge bisection (LREB) method on the tear-drop surface (a surface of revolution whose profile is defined by the tear-drop curve in Figure 3), showing a cross-section of the grid refined at $\epsilon = 0.005$ (top; cells containing the geometry are highlighted), the implicit surface extracted from the grid (middle), and the graph of number of cells as a function of the distance threshold ϵ (bottom).

a tetrahedron is measured as the ratio between its in-radius over its out-radius, normalized by the maximum of such ratio (1/3). As shown in Figure 16, the worst quality across all thresholds in all examples is around 0.03. Observe that the graphs for many examples (except for CSG) oscillate rather than trending downward, which suggests the possible existence of a lower bound.

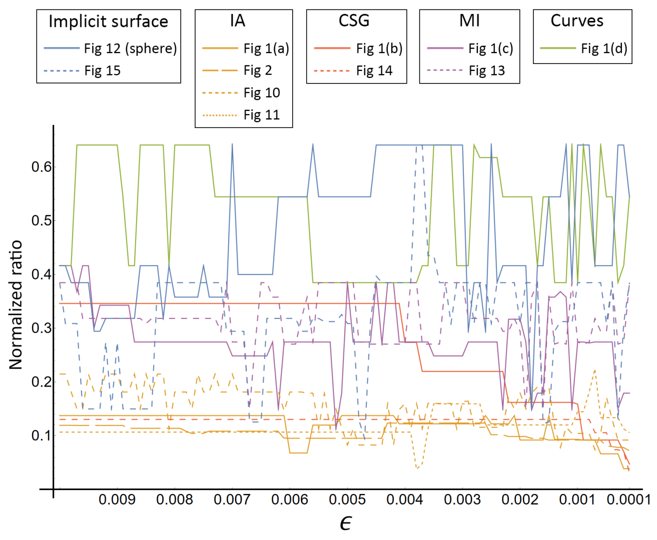

Fig. 16. Worst quality (measured by normalized ratio between in-radius and out-radius) among tetrahedra containing the implicit shape generated by our LREB method as a function of decreasing ϵ, for all 3D examples shown so far in the paper.

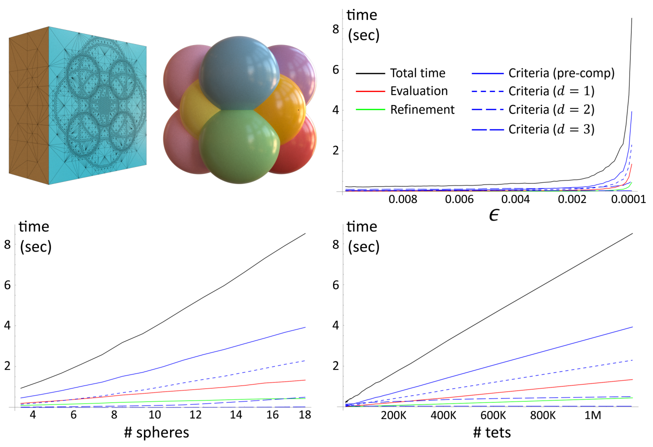

Fig. 17. Timing of our algorithm on an IA consisting of intersecting spheres (top-left; using $\epsilon = 0.0001$), plotted against decreasing threshold ϵ (top-right; for 18 spheres), increasing number of spheres (bottom-left; at $\epsilon = 0.0001$), and increasing number of tetrahedra (bottom-right; for 18 spheres by varying ϵ). The timing is broken down into evaluation of function values and gradients (red), cell splitting (green), and checking the refinement criteria, including computing the proxy and quantities needed for robust evaluation (Appendix B; solid blue), and performing the zero-crossing and distance tests for subfunctions with different number of components d (dashed blue).

7.4 Implementation and performance

Our algorithm is implemented in C++ for $n = 2, 3$ dimensions without using any parallelization or third-party packages. We use an alternative approach for checking extremity in our zero-crossing test (Equation 9), which we found to be faster than LP due to the small problem size. Recall that the zero-crossing test checks whether the origin of $\mathbb{R}^m$ is an extremal point in a set of $l + 1$ points, where $m \leq n$ and l is the number of control points in the cubic Bezier simplex (10 for $n = 2$ and 20 for $n = 3$). We enumerate all m-tuples of points. For each tuple, we check if the origin lies on a different side of the line ($m = 2$) or plane ($m = 3$) formed by the tuple from the remaining points that are not in the tuple. If so, the test returns false. The test returns true when the enumeration successfully completes. All timings are recorded on a PC with a 10-core Intel i9-10900X 3.70Hz CPU and 64GB of RAM.

We evaluate the performance of our algorithm on the IA of 18 implicit spheres with the same radius that are located in a regular pattern (see Figure 17 top-left). Figure 17 (top-right) plots the running time of each part of our algorithm as the distance threshold ϵ decreases. Observe that the running time is dominated by evaluating the function values and gradients at the grid points (red) and checking the criteria (blue), and the latter in turn is dominated by computing the proxy (solid line) and performing the tests on implicit surfaces (short dashes). Note that while the tests on intersection curves and points (medium and long dashes) are more expensive to perform than those on surfaces, those tests are only invoked for cells where there are more than one (two for MI) active components, which are a small fraction of total cells. Our algorithm scales roughly linearly with both the number of function components and the total number of cells, as shown in the two plots in Figure 17 bottom.

7.5 More examples

Figure 1 shows examples of all four types of implicit complexes that are considered in this paper, including an IA of 10 implicit surfaces (a), a CSG shape defined by the same set of surfaces (b), an MI as the Voronoi diagram of five lines (c), and the network of junction curves of the MI (d). Observe from the visualization of the grids that our method can effectively adapt the grid resolution to the geometric complexity in each example without unnecessary refinement.

We next show a few challenging examples of discretizing implicit surfaces and complexes to further demonstrate the benefit of our algorithm, particularly its adaptivity and refinement around intersections.

An adaptive grid is particularly useful when the function is expensive to evaluate. As the runtime of grid generation is dominated by function evaluations, reducing the number of grid points (where evaluation takes place) is the key to improving efficiency. Figure 18 gives an example of such a function, which is an Hermite RBF defined by around 1000 points sampled from the Max Planck model (obtained from [Huang et al. 2019]). Note that the evaluation time of a Hermite RBF scales linearly with the number of points. Evaluating this function on a uniform cubic grid of size 100^3 (over 1 million evaluations) took 27.76 seconds, whereas our algorithm took 1.19 seconds to generated a grid with only 21, 286 points (at $\epsilon = 0.00125$). Despite a much more compact grid, the implicit surface computed on our adaptive grid better captures geometric details, such as eyes and ears, than that computed from the uniform grid.

Another scenario that demands adaptivity is when the implicit shape contains features at different scales. While a uniform grid would need an excessive number of cells to capture the smallest feature, an adaptive grid would need much fewer cells as it can

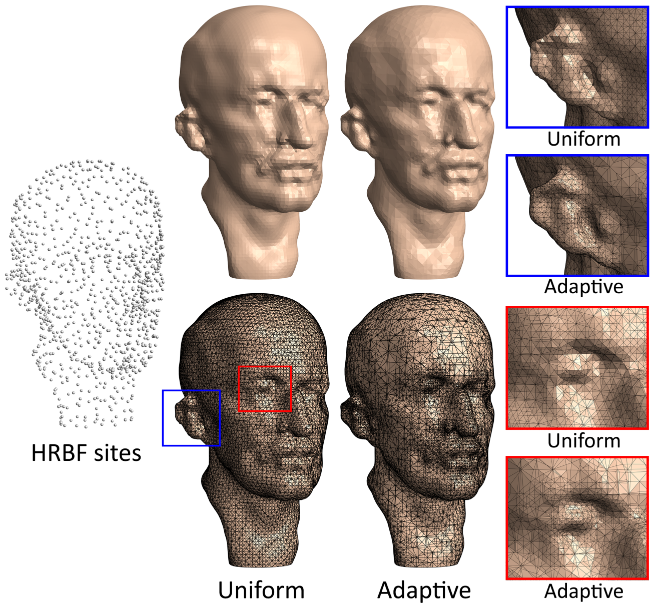

Fig. 18. Comparing the implicit surfaces of a Hermite RBF function with 1000 sites (left) extracted on a uniform tetrahedral grid (tessellation of a 100^3 cubic grid) and on our adaptively refined grid ($\epsilon = 0.00125$).

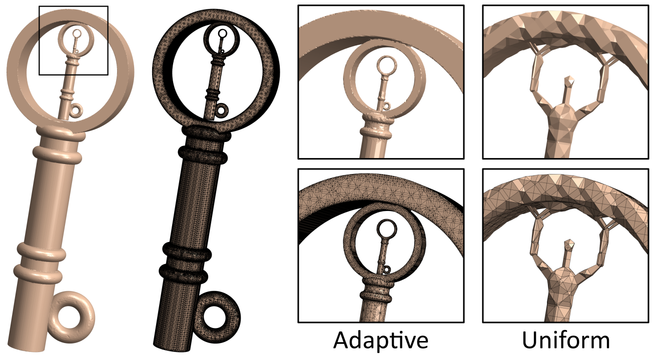

Fig. 19. Comparing the implicit surface of an SDF extracted on our adaptively refined grid ($\epsilon = 0.0005$) and on a uniform tetrahedral grid (tessellation of a 100^3 cubic grid). The SDF is created by Shadertoy user Flopine [2020], released under CC BY-NC-SA 3.0 License.

adapt the cell size to the feature's scale. To this end, we created a stress test in Figure 19 where the input function is the SDF (signed distance function) from the surface of three nested "key" shapes at decreasing sizes. Our algorithm at $\epsilon = 0.0005$ creates a grid of 2, 225, 796 cells that enables the discretization to capture all three keys (observe the variable triangle sizes on different part of the keys in the zoom-in view). In comparison, the surface extracted from a uniform tetrahedral grid with 5×10^6 cells (obtained from a 100^3 cubic grid) can barely reconstruct the largest key.

Finally, we demonstrate the intersection-awareness of our algorithm on several CSG examples. The shape in Figure 20 is created by subtracting three cylinders from a torus. Since the cylinders are

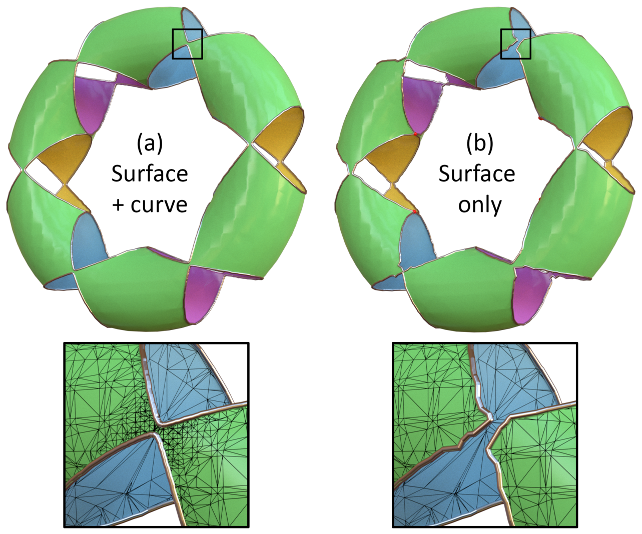

Fig. 20. A CSG shape defined by subtracting three cylinders from a torus, discretized on a grid that is adapted to both the surfaces and their intersection curves (a) or adapted only to surfaces (b).

almost tangent to the torus, the shape contains a few very narrow spots (e.g., in the boxed region). As our algorithm adapts the grid to the geometry of both the surface patches and their intersections, it can faithfully capture the shape of each narrow band (see the zoomed-in view in (a)). In contrast, adapting the grid to only the surfaces (i.e., only checking the refinement criteria for scalar zero sets) leads to artifacts at the narrow spots (see the zoomed-in view in (b)). Similar observations can be made in Figure 21, which takes the union of 10 randomly located tori and subtracts it from the union of another 10 tori, and Figure 22, which performs subtraction (a) and union (b) of two copies of the same dog head surface (each defined by an Hermite RBF function) that are slightly shifted from each other. By adapting the grid to intersection curves in addition to the surfaces, our algorithm better captures fine features and produces smoother intersection curves.

8 DISCUSSIONS

We presented a new method for generating adaptively refined simplicial grids for discretizing implicit surfaces and complexes. Our method adapts the grid resolution to not only surfaces but also their lower-dimensional intersections, thus enabling more accurate discretizations without excessive refinement. Our method does not require interval analysis, and hence can be applied to arbitrary functions, and it can produce grids for several types of commonly used implicit complexes.

The effectiveness of our method largely depends on how well the input function can be approximated by the proxy within each simplex. Our choice of the proxy, the cubic Bezier simplex, may fail to adequately capture the variation of non-trivial functions and thus lead to either insufficient or redundant refinement of the grid. We demonstrate these two failure modes in Figures 23 and 24. In Figure 23, we discretize a high-order algebraic curve in $\mathbb{R}^2$ (the *rose*

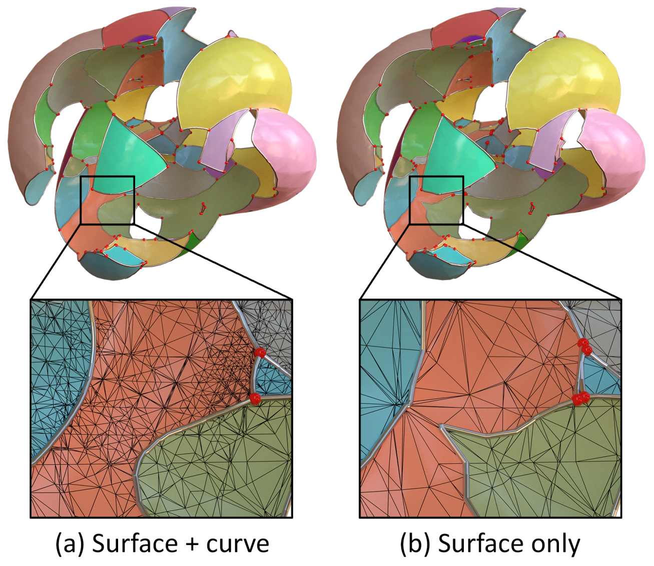

Fig. 21. (a): A CSG shape defined by random subtraction and addition of 20 tori, discretized on a grid that is adapted to both the surfaces and their intersection curves (a) or adapted only to surfaces (b).

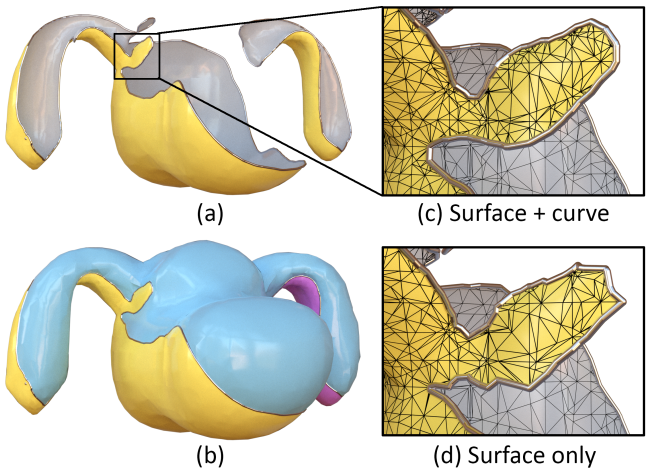

Fig. 22. (a,b): The result of subtraction (a) or addition (b) between two slightly shifted dog head surfaces (each defined by an Hermite RBF function), discretized on our adaptively refined grid. (c,d): Comparing discretizations with or without adaptive refinement for the intersection curves.

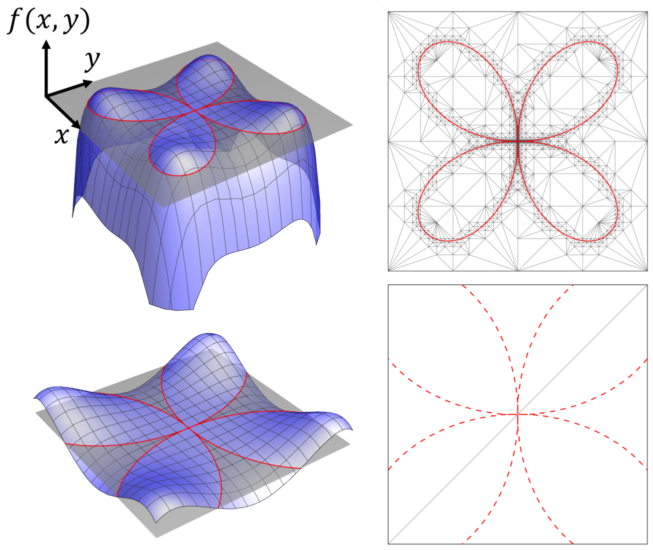

Fig. 23. Discretizing the 4-pedal rose curve (zero set of $f(x, y) = 4x^2y^2 - (x^2 + y^2)^3$) using a larger domain (top) and a smaller domain (bottom). The curves are visualized on the left as the intersection (red) between the height surface of $f(x, y)$ (blue) and the zero plane (gray). The dotted curve in lower-right indicates where the rose curve should be, but missed by the discretization, due to an under-refined grid (made up of two triangles).

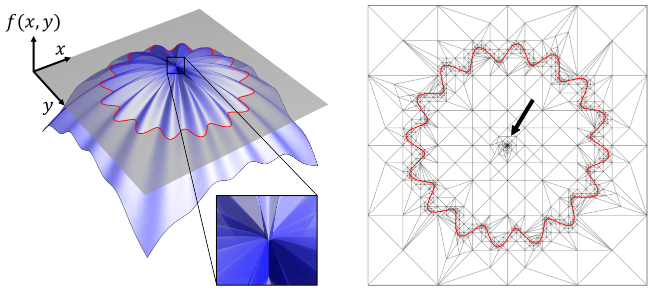

Fig. 24. Discretizing a wavy circle (zero set of $f(r, \theta) = 0.2r^{0.1}\sin(16\theta) + r^2 - 1$, where r, θ are polar coordinates). Left: the curve visualized as the intersection (red) between the height surface of the function (blue) and the zero plane (gray). The zoom-in shows the region around the origin. Right: the refined grid and discretization (red). The arrow points to an over-refined region at the origin, far away from the curve.

curve) using two domains with different sizes. Our method produces an adequately refined grid in the larger domain (top) but fails to refine beyond the initial grid in the smaller domain (bottom). This is because the proxies in both triangles of the initial grid do not pass the zero-crossing test. In Figure 24, our method over-refines the grid in a region far from the zero-set (near the center). This is because the function there has a high amount of oscillation (see the insert), which causes our proxy to over-estimate the function range. Using a higher-order proxy may alleviate these issues, but ultimately an interval analysis may be necessary.

While our refinement method (LREB) produces more compact grids than the classical LEB method, it comes at the cost of significantly worsened simplex quality away from the implicit geometry. This makes LREB unsuited for applications that require well-shaped simplices over the entire domain, such as FEA and visualization. Another drawback of LREB is that it needs to explicitly store the vertex coordinates and simplex connectivity. In contrast, the regularity of simplices produced by LEB enables space-efficient data structures that can implicitly encode the geometry and connectivity of the grid (see review in [Weiss and De Floriani 2011]).

There are several venues of future work that we wish to pursue. Our current construction of the proxy function requires evaluating gradients. This can be avoided by first constructing the triangular Lagrange interpolant from values of f at the control points [Farin 1986] and then converting to the Bezier form. We would like to explore refinement criteria for other implicitly defined shapes, such as F-reps [Pasko et al. 1995], extremal features (e.g., ridges and valleys) of unsigned functions, and topological features in scalar or vector functions (e.g., separatrices). Another interesting direction is to extend our method to higher-order grids [Jiang et al. 2021; Khanteimouri and Campen 2023]. The extension could leverage the common representation of a typical curved simplex and our proxy function - both as a Bezier simplex.

ACKNOWLEDGMENTS

This work is supported by NIH grant U2C CA233303, NSF grant EF-1921728, and a gift from Adobe Systems. We would like to thank Nicholas Sharp, Marcos Slomp and Inigo Quilez for helpful discussions, and the anonymous reviewers for their suggestions.

REFERENCES

Eugene Allgower and Kurt Georg. 1980. Simplicial and Continuation Methods for Approximating Fixed Points and Solutions to Systems of Equations. *SIAM Rev.* 22, 1 (jan 1980), 28–85. https://doi.org/10.1137/1022003

Eugene L. Allgower and Kurt Georg. 1989. Estimates for piecewise linear approximations of implicitly defined manifolds. *Applied Mathematics Letters* 2 (1989), 111–115. https://api.semanticscholar.org/CorpusID:123564143

Eugene L. Allgower and Phillip H. Schmidt. 1985. An Algorithm for Piecewise-Linear Approximation of an Implicitly Defined Manifold. *SIAM J. Numer. Anal.* 22, 2 (1985), 322–346. https://doi.org/10.1137/0722020 arXiv:https://doi.org/10.1137/0722020

Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, et al. 2021. MFEM: A Modular Finite Element Methods Library. *Computers & Mathematics with Applications* 81 (2021), 42–74. https://doi.org/10.1016/j.camwa.2020.06.009

Douglas N. Arnold, Arup K. Mukherjee, and Luc Pouly. 2000. Locally Adapted Tetrahedral Meshes Using Bisection. *SIAM J. Sci. Comput.* 22 (2000), 431–448. https://api.semanticscholar.org/CorpusID:13167330

Sergei Azernikov and Anath Fischer. 2005. Anisotropic Meshing of Implicit Surfaces. In *Proceedings of the International Conference on Shape Modeling and Applications 2005 (SMI '05)*. IEEE Computer Society, USA, 94–103. https://doi.org/10.1109/SMI.2005.5

Brigham Bagley, Shankar Sastry, and Ross Whitaker. 2016. A Marching-tetrahedra Algorithm for Feature-preserving Meshing of Piecewise-smooth Implicit Surfaces. *Procedia Engineering* 163 (12 2016), 162–174.

Eberhard Bänsch. 1991. Local mesh refinement in 2 and 3 dimensions. *IMPACT Comput. Sci. Eng.* 3 (1991), 181–191. https://api.semanticscholar.org/CorpusID:33479075

Guillem Belda Ferrín, Eloi Ruiz Gironès, and Francisco Javier Roca Navarro. 2022. Bisecting with Optimal Similarity Bound on 3D Unstructured Conformal Meshes. In *Proceedings of the 2022 SIAM International Meshing Roundtable*. Zenodo, USA, 86–87. https://doi.org/10.5281/zenodo.6562417

Martin Bertram, Gerd Reis, Rolf Hendrik van Lengen, Sascha Köhn, and Hans Hagen. 2005. Non-manifold mesh extraction from time-varying segmented volumes used for modeling a human heart.. In *EuroVis*. Citeseer, 199–206.

Jürgen Bey. 1995. Tetrahedral grid refinement. *Computing* 55 (1995), 355–378. https://api.semanticscholar.org/CorpusID:20829446

Jules Bloomenthal. 1988. Polygonization of implicit surfaces. *Comput. Aided Geom. Des.* 5 (1988), 341–355. https://api.semanticscholar.org/CorpusID:16474404

Jean-Daniel Boissonnat, David Cohen-Steiner, and Gert Vegter. 2008. Isotopic implicit surface meshing. *Discrete & Computational Geometry* 39, 1-3 (2008), 138–157. https://doi.org/10.1007/s00454-007-9011-4

Jean-Daniel Boissonnat, Siargey Kachanovich, and Mathijs Wintraecken. 2023. Tracing Isomanifolds in $\mathbb{R}$ d in Time Polynomial in d using Coxeter–Freudenthal–Kuhn Triangulations. *SIAM J. Comput.* 52, 2 (2023), 452–486. https://doi.org/10.1137/21M1412918 arXiv:https://doi.org/10.1137/21M1412918

Jean-Daniel Boissonnat and Mathijs Wintraecken. 2022. The Topological Correctness of PL Approximations of Isomanifolds. *Found. Comput. Math.* 22, 4 (aug 2022), 967–1012. https://doi.org/10.1007/s10208-021-09520-0

Kathleen Sue Bonnell, Mark A Duchaineau, Daniel R Schikore, Bernd Hamann, and Kenneth I Joy. 2003. Material interface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 9, 4 (2003), 500–511.

Rita Borgo, Paolo Cignoni, and Roberto Scopigno. 2004. Simplicial-based Multiresolution Volume Datasets Management: An Overview. In *Geometric Modeling for Scientific Visualization*. Springer Verlag, Berlin, Heidelberg, 309–327. https://api.semanticscholar.org/CorpusID:54075053

Jan Brandts, Sergey Korotov, Michal Křížek, et al. 2020. *Simplicial partitions with applications to the finite element method*. Springer.

Michael Burns, Janek Klawe, Szymon Rusinkiewicz, Adam Finkelstein, and Doug DeCarlo. 2005. Line drawings from volume data. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 512–518.

Amit Chattopadhyay, Simon Plantinga, and Gert Vegter. 2012. Certified Meshing of Radial Basis Function Based Isosurfaces. *Vis. Comput.* 28, 5 (may 2012), 445–462. https://doi.org/10.1007/s00371-011-0627-2

Bruno Rodrigues De Araújo, Daniel S Lopes, Pauline Jepp, Joaquim A Jorge, and Brian Wyvill. 2015. A survey on implicit surface polygonization. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–39.

L. de Figueiredo, A. Paiva, T. Lewiner, and H. Lopes. 2006. Robust adaptive meshes for implicit surfaces. In *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images*. IEEE Computer Society, Los Alamitos, CA, USA, 205–212. https://doi.org/10.1109/SIBGRAPI.2006.40

J Ruiz de Miras and Francisco R Feito. 2002. Direct and robust voxelization and polygonization of free-form CSG solids. In *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission*. 352–355. https://doi.org/10.1109/TDPVT.2002.1024082

Tamal Krishna Dey and Andrew G. Slatton. 2013. Localized Delaunay Refinement for Piecewise-Smooth Complexes. In *Proceedings of the Twenty-Ninth Annual Symposium on Computational Geometry* (Rio de Janeiro, Brazil) *(SoCG '13)*. Association for Computing Machinery, New York, NY, USA, 47–56. https://doi.org/10.1145/2462356.2462376

Sény Diatta, Guillaume Moroz, and Marc Pouget. 2019. Reliable Computation of the Singularities of the Projection in R3 of a Generic Surface of R4. In *MACIS 2019 - Mathematical Aspects of Computer and Information Sciences*. Gebze-Istanbul, Turkey. https://inria.hal.science/hal-02406758

Scott Dillard, John Bingert, Dan Thoma, and Bernd Hamann. 2007. Construction of Simplified Boundary Surfaces from Serial-sectioned Metal Micrographs. *Visualization and Computer Graphics, IEEE Transactions on* 13 (12 2007), 1528–1535.

Xingyi Du, Qingnan Zhou, Nathan A. Carr, and Tao Ju. 2022. Robust computation of implicit surface networks for piecewise linear functions. *ACM Transactions on Graphics (TOG)* 41 (2022), 1 – 16. https://api.semanticscholar.org/CorpusID:249917144

Tom Duff. 1992. Interval arithmetic recursive subdivision for implicit functions and constructive solid geometry. *ACM SIGGRAPH computer graphics* 26, 2 (1992), 131–138.

Laurent Dupont, Michael Hemmer, Sylvain Petitjean, and Elmar Schömer. 2007. Complete, exact and efficient implementation for computing the adjacency graph of an arrangement of quadrics. In *Proceedings of the 15th Annual European Conference on Algorithms* (Eilat, Israel) *(ESA'07)*. Springer-Verlag, Berlin, Heidelberg, 633–644.

Herbert Edelsbrunner and John Harer. 2002. Jacobi sets of multiple Morse functions. *Foundations of Computational Mathematics, Minneapolis* (2002), 37–57.

Gerald Farin. 1986. Triangular bernstein-bézier patches. *Computer Aided Geometric Design* 3, 2 (1986), 83–127.

Michael S Floater. 2015. Generalized barycentric coordinates and applications. *Acta Numerica* 24 (2015), 161–214.

Flopine. 2020. ShATI - Clé. https://www.shadertoy.com/view/wssBDf.

Paul Louis George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille, D. Marcum, and L. Maréchal. 2017. *Mesh Generation and Mesh Adaptivity: Theory and Techniques*. John Wiley & Sons, Ltd, USA, 1–51. https://doi.org/10.1002/9781119176817.ecm2012 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119176817.ecm2012

T. Gerstner. 2003. *Top-down view-dependent terrain triangulation using the octagon metric*. Technical Report. University of Bonn.

Thomas Gerstner and Renato Pajarola. 2000. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *Proceedings of the Conference on Visualization '00* (Salt Lake City, Utah, USA) *(VIS '00)*. IEEE Computer Society Press, Washington, DC, USA, 259–266.

Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy. 2002. Interactive view-dependent rendering of large isosurfaces. In *Proceedings of the Conference on Visualization '02* (Boston, Massachusetts) *(VIS '02)*. IEEE Computer Society, USA, 475–484.

Roberto Grosso, Christoph Lürig, and Thomas Ertl. 1997. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *Proceedings of the 8th Conference on Visualization '97* (Phoenix, Arizona, USA) *(VIS '97)*. IEEE Computer Society Press, Washington, DC, USA, 387–ff.

Jiateng Guo, Xulei Wang, Jiangmei Wang, Xinwei Dai, Lixin Wu, Chaoling Li, Fengdan Li, Shanjun Liu, and Mark Walter Jessell. 2021. Three-dimensional geological modeling and spatial analysis from geotechnical borehole data using an implicit surface and marching tetrahedra algorithm. *Engineering Geology* 284 (2021), 106047.

Mark Hall and Joe Warren. 1990. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics and Applications* 10, 6 (1990), 33–42. https://doi.org/10.1109/38.62694

Younis Hijazi, Aaron Knoll, Mathias Schott, Andrew Kensler, and Charles Hansen. 2010. Csg operations of arbitrary primitives with interval arithmetic and real-time ray casting. In *Dagstuhl Follow-Ups*, Vol. 1. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Zhiyang Huang, Nathan Carr, and Tao Ju. 2019. Variational implicit point set surfaces. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–13.

Kin Chuen Hui and Z.H. Jiang. 1999. Tetrahedra Based Adaptive Polygonization of Implicit Surface Patches. *Computer Graphics Forum* 18, 1 (1999), 57–68. https://doi.org/10.1111/1467-8659.00302 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00302

Zhongshi Jiang, Ziyi Zhang, Yixin Hu, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2021. Bijective and coarse high-order tetrahedral meshes. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–16.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.

Payam Khanteimouri and Marcel Campen. 2023. 3D Bézier Guarding: Boundary-Conforming Curved Tetrahedral Meshing. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–19.

Byungmoon Kim. 2010. Multi-phase fluid simulations using regional level sets. *ACM Transactions on Graphics (TOG)* 29, 6 (2010), 1–8.

Dae-Hyun Kim, Ulf Doring, and Beat Bruderlin. 2000. Polygonization of Non-manifolds With the Aid of Interval Operators. *ACM Eurographics Workshop on Implicit Surfaces* (2000), 145–151. https://api.semanticscholar.org/CorpusID:14381080

Maximilian Kohlbrenner, Singchun Lee, Marc Alexa, and Misha Kazhdan. 2023. Poisson Manifold Reconstruction — Beyond Co-dimension One. *Computer Graphics Forum* 42 (08 2023). https://doi.org/10.1111/cgf.14907

Igor Kossaczký. 1994. A recursive approach to local mesh refinement in two and three dimensions. *J. Comput. Appl. Math.* 55 (1994), 275–288. https://api.semanticscholar.org/CorpusID:123381194

Xinghua Liang and Yongjie Jessica Zhang. 2014. An octree-based dual contouring method for triangular and tetrahedral mesh generation with guaranteed angle range. *Engineering with Computers* 30 (2014), 211–222. https://api.semanticscholar.org/CorpusID:14861190

Anwei Liu and Barry Joe. 1995. Quality Local Refinement of Tetrahedral Meshes Based on Bisection. *SIAM Journal on Scientific Computing* 16, 6 (1995), 1269–1291. https://doi.org/10.1137/0916074 arXiv:https://doi.org/10.1137/0916074

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm.. In *SIGGRAPH*, Maureen C. Stone (Ed.). ACM, 163–169.

Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. 2006. Multiple Interacting Liquids. *ACM Trans. Graph.* 25, 3 (jul 2006), 812–819.

Joseph M. Maubach. 1995. Local Bisection Refinement for N-Simplicial Grids Generated by Reflection. *SIAM Journal on Scientific Computing* 16, 1 (1995), 210–227. https://doi.org/10.1137/0916014 arXiv:https://doi.org/10.1137/0916014

Nimrod Megiddo. 1984. Linear programming in linear time when the dimension is fixed. *Journal of the ACM (JACM)* 31, 1 (1984), 114–127.

Chohong Min. 2003. Simplicial isosurfacing in arbitrary dimension and codimension. *J. Comput. Phys.* 190, 1 (2003), 295–310. https://doi.org/10.1016/S0021-9991(03)00275-4

William F Mitchell. 2016. 30 years of newest vertex bisection. In *AIP Conference Proceedings*, Vol. 1738. AIP Publishing.

Neil Molino, Robert Bridson, Joseph Teran, and Ronald Fedkiw. 2003. A Crystalline, Red Green Strategy for Meshing Highly Deformable Objects with Tetrahedra. *12th Int. Meshing Roundtable*, 103–114.

Bernard Mourrain, Jean-Pierre Técourt, and Monique Teillaud. 2005. On the computation of an arrangement of quadrics in 3d. *Computational Geometry* 30, 2 (2005), 145–164.

Gregory Nielson and R. Franke. 1997. Computing the separating surface for segmented data. 229–233.

Th Ottmann, Sven Schuierer, and Subbiah Soundaralakshmi. 1995. Enumerating extreme points in higher dimensions. In *STACS 95*, Ernst W. Mayr and Claude Puech (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 562–570.

Steve Oudot, Laurent Rineau, and Mariette Yvinec. 2010. Meshing volumes with curved boundaries. *Engineering with Computers* 26, 3 (2010), 265–279. https://doi.org/10.1007/s00366-009-0166-x

Valerio Pascucci. 2004. Isosurface computation made simple: hardware acceleration, adaptive refinement and tetrahedral stripping. In *Proceedings of the Sixth Joint Eurographics - IEEE TCVG Conference on Visualization* (Konstanz, Germany) *(VISSYM'04)*. Eurographics Association, Goslar, DEU, 293–300.

Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir Savchenko. 1995. Function representation in geometric modeling: concepts, implementation and applications. *The visual computer* 11 (1995), 429–446.

Carl S. Petersen. 1984. Adaptive contouring of three-dimensional surfaces. *Computer Aided Geometric Design* 1, 1 (1984), 61–74. https://doi.org/10.1016/0167-8396(84)90004-9

Carl S. Petersen, Bruce R. Piper, and Andrew J. Worsey. 1987. Adaptive contouring of a trivariate interpolant. *Geometric Modeling: Algorithms and New Trends* (1987), 385–395.

Simon Plantinga and Gert Vegter. 2006. Isotopic meshing of implicit surfaces. *The Visual Computer* 23 (2006), 45–58. https://api.semanticscholar.org/CorpusID:751474

Angel Plaza and Maria-Cecilia Rivara. 2003. Mesh Refinement Based on the 8-Tetrahedra Longest- Edge Partition.. In *Proceedings of the 12th International Meshing Roundtable*. 67–78.

Sundaresan Raman and Rephael Wenger. 2008. Quality Isosurface Mesh Generation Using an Extended Marching Cubes Lookup Table. *Computer Graphics Forum* 27, 3 (2008), 791–798. https://doi.org/10.1111/j.1467-8659.2008.01209.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2008.01209.x

Aristides AG Requicha and Herbert B Voelcker. 1977. Constructive solid geometry. (1977).

María Cecilia Rivara. 1991. Local modification of meshes for adaptive and/or multigrid finite-element methods. *J. Comput. Appl. Math.* 36 (1991), 79–89. https://api.semanticscholar.org/CorpusID:120011902

RI Saye. 2015. An algorithm to mesh interconnected surfaces via the Voronoi interface. *Engineering with Computers* 31, 1 (2015), 123–139.

Scott Schaefer and Joe Warren. 2004. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* IEEE, 70–76.

Michael F. W. Schmidt. 1993. Cutting cubesvisualizing implicit surfaces by adaptive polygonization. *The Visual Computer* 10 (1993), 101–115. https://api.semanticscholar.org/CorpusID:35201048

Elmar Schömer and Nicola Wolpert. 2006. An exact and efficient approach for computing a cell in an arrangement of quadrics. *Computational Geometry* 33, 1-2 (2006), 65–97.

M Haitham Shammaa, Yutaka Ohtake, and Hiromasa Suzuki. 2010. Segmentation of multi-material CT data of mechanical parts for extracting boundary surfaces. *Computer-Aided Design* 42, 2 (2010), 118–128.

Barton T. Stander and John C. Hart. 1997. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 279–286. https://doi.org/10.1145/258734.258868

Robert F. Tobler, Thomas Galla, and Werner Purgathofer. 1995. *ACSGM – An adaptive CSG meshing algorithm.* Technical Report TR-186-2-95-13. Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria. https://www.cg.tuwien.ac.at/research/publications/1995/Tobler-1995-AAC/ human contact: technical-report@cg.tuwien.ac.at.

Gokul Varadhan, Shankar Krishnan, TVN Sriram, and Dinesh Manocha. 2004. Topology preserving surface extraction using adaptive subdivision. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 235–244.

Chris Weigle and David C. Banks. 1996. Complex-valued contour meshing. In *Proceedings of the 7th Conference on Visualization '96* (San Francisco, California, USA) *(VIS '96)*. IEEE Computer Society Press, Washington, DC, USA, 173–ff.

Kenneth Weiss and Leila De Floriani. 2009. Supercubes: A high-level primitive for diamond hierarchies. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (2009), 1603–1610.

Kenneth Weiss and Leila De Floriani. 2011. Simplex and Diamond Hierarchies: Models and Applications. *Computer Graphics Forum* 30, 8 (2011), 2127–2155. https://doi.org/10.1111/j.1467-8659.2011.01853.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01853.x

Hassler Whitney. 1957. *Geometric Integration Theory.* Princeton University Press.

Shangyou Zhang. 1995. Successive subdivisions of tetrahedra and multigrid methods on tetrahedral meshes. *Houston J. Math* 21, 3 (1995), 541–556.

Yongjie Zhang, Chandrajit Bajaj, and Bong-Soo Sohn. 2005. 3D finite element meshing from imaging data. *Computer Methods in Applied Mechanics and Engineering* 194, 48 (2005), 5083–5106. https://doi.org/10.1016/j.cma.2004.11.026 Unstructured Mesh Generation.

Yongjie Zhang, Thomas JR Hughes, and Chandrajit L Bajaj. 2008. Automatic 3d mesh generation for a domain with multiple materials. In *Proceedings of the 16th international meshing roundtable*. Springer, 367–386.

Yongjie Jessica Zhang and Jin Qian. 2012. Resolving topology ambiguity for multiple-material domains. *Computer Methods in Applied Mechanics and Engineering* 247 (2012), 166–178.

Tong Zhao, Pierre Alliez, Tamy Boubekeur, Laurent Busé, and Jean-Marc Thiery. 2021. Progressive Discrete Domains for Implicit Surface Reconstruction. *Computer Graphics Forum* 40, 5 (2021), 143–156. https://doi.org/10.1111/cgf.14363 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14363

Yong Zhou, Baoquan Chen, and Arie Kaufman. 1997. Multiresolution tetrahedral framework for visualizing regular volume data. In *Proceedings of the 8th Conference* *on Visualization '97* (Phoenix, Arizona, USA) *(VIS '97)*. IEEE Computer Society Press, Washington, DC, USA, 135–ff.

A CUBIC BEZIER SIMPLEX

We give details of the cubic Bezier simplex and its construction from sampled values and gradients at the simplex vertices. A thorough discussion can be found in [Farin 1986].

As mentioned in Section 4.1, the control points $p_1, \ldots, p_l$ in an n-dimensional cubic Bezier simplex t include the vertices, edge trisectors, and face centroids of t. The barycentric coordinates (w.r.t. to the $n+1$ vertices of t) at each control point p_i thus have the form $\{\lambda_1^i, \ldots, \lambda_{n+1}^i\}/3$, where each $\lambda_j^i \in \{0, 1, 2, 3\}$ and $\sum_{j=1}^{n+1} \lambda_j^i = 3$. The Bernstein polynomials $w_i(x)$ $(i = 1, \ldots, l)$ for any $x \in t$ has the form:

$$w_i(x) = \frac{3!}{\lambda_1^i! \cdots \lambda_{n+1}^i!} \beta_1(x)^{\lambda_1^i} \cdots \beta_{n+1}(x)^{\lambda_{n+1}^i}$$

where $\beta_j(x)$ is the j-th barycentric coordinate of x.

We obtain the control values (or the Bezier ordinates) $b^1, \ldots, b^l$ following the "nine parameter interpolant" method described in [Farin 1986], which interpolates given values and gradient vectors at each vertex of t. Specifically, let $f(q)$ and $\nabla f(q)$ be the value and gradient at a vertex q,

- If p_i is a vertex of t, then $b^i = f(p_i)$.
- If p_i is a trisector of edge $\overline{p_j p_k}$ and closer to p_j, then

$$b^i = b^j + \frac{1}{3} \nabla f(p_j) \cdot (p_k - p_j)$$

- If p_i is the centroid of a triangle with vertices V and edge trisectors E, then

$$b^i = \frac{1}{4} \sum_{p_j \in E} b^j - \frac{1}{6} \sum_{p_j \in V} b^j$$

Note that the control values b^i at the triangle centroids do not affect the interpolation of the given values and gradients. The choices made above have the property that the resulting interpolant reproduces all polynomial f up to quadratics.

B INVERSION-FREE DISTANCE TEST

We show how the distance test of Equation 15 can be re-formulated without matrix inversions. To further reduce numerical errors when implementing on floating-point numbers, we also remove divisions and square-roots in the calculations.

Let $p_1, \ldots, p_{n+1}$ be the vertices of an n-dimensional simplex t, v the n-by-n matrix whose columns are the edge vectors $p_i - p_1$ $(i = 2, \ldots, n+1)$, and h the n-by-m matrix whose columns are the differences of function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ along each edge, $f(p_i) - f(p_1)$. The gradients $g = \nabla f = \{\nabla f_1, \ldots, \nabla f_m\}$ thus satisfy $g^T \cdot v = h$, and hence can be found by

$$g^T = h \cdot v^{-1} = h \cdot \frac{\text{adj}(v)}{\det(v)}$$

where $\text{adj}(v)$ and $\det(v)$ are the adjugate and determinant of the square matrix v. Note that computing both $\text{adj}(v)$ and $\det(v)$ involves only additions and multiplications (no divisions).

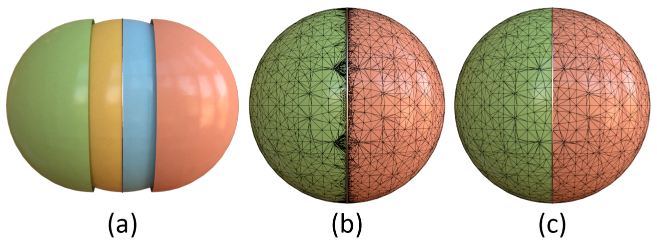
Fig. 25. IA of a two-component function, whose components are slighted shifted spherical distance functions, discretized on a grid refined by the original distance test in Equation 15 (b) and the reformulated test in Equation 16 (c). An exploded view of the IA is shown in (a). .

To construct the matrix $M = g(g^T g)^{-1}$ used in the test (Equation 15), we first re-write $g^T g$ as

$$g^T g = \frac{w^T w}{\det(v)^2},$$

where $w = \text{adj}(v)^T h^T$ is an m-by-m matrix. Let $u = w^T w$, we can express the inverse $(g^T g)^{-1}$ as

$$(g^T g)^{-1} = \det(v)^2 \frac{\text{adj}(u)}{\det(u)}.$$

Finally, we can write M as

$$M = g(g^T g)^{-1} = \frac{w}{\det(v)} \det(v)^2 \frac{\text{adj}(u)}{\det(u)} = \frac{\det(v)}{\det(u)} w \, \text{adj}(u).$$

Again, note that computing the quantities $w, u, \text{adj}(u), \det(u)$ does not involve divisions or matrix inversions.

We now substitute the above expression of M into Equation 15, which yields the inequality

$$\max_{i=1,\ldots,l} \left| \frac{\det(v)}{\det(u)} w \, \text{adj}(u)(b^i - \bar{b}^i) \right| > \epsilon,$$

where $b^i, \bar{b}^i$ are length-m column vectors. After multiplying both sides by $|\det(u)|$ and then squaring both sides (to avoid computing the vector norm $|\cdot|$, which involves a square root), we arrive at an equivalent formulation of Equation 15 that is free of matrix inversions, divisions, and square-roots:

$$\boxed{\max_{i=1,\ldots,l} \det(v)^2 r^t r > \epsilon^2 \det(u)^2,} \tag{16}$$

where $r = w \, \text{adj}(u)(b^i - \bar{b}^i)$.

To demonstrate the improved numerical accuracy of the reformulated test (Equation 16) over the original (Equation 15), we compared both in discretizing the IA of a two-component function in Figure 25. The components f_1, f_2 are identical spherical functions slightly shifted from each other, and hence their gradients g_1, g_2 are almost identical (and hence $g^T g$ is almost singular) along the circle where the two spheres intersect. Compared to the reformulated test (shown in (c)), the original test results in significantly more refinement along the intersection curve (see (b)). This indicates that the original test over-estimates the distance error (left-hand side of Equation 15), which we attribute to inverting the almost-singular $g^T g$.