Non-local Isotopic Approximation of Nonsingular Surfaces [1]

Long Lin and Chee Yap and Jihun Yu [2,3]
Courant Institute of Mathematical Sciences
New York University
New York, NY 10012 USA
{llin,yap,jihun}@cs.nyu.edu

Abstract

We consider the problem of approximating nonsingular surfaces which are implicitly represented by equations of the form $f(x, y, z) = 0$. Our correctness criterion is isotopy of the approximate surface to the exact surface. We focus on methods based on domain subdivision using numerical primitives. Such methods are practical and have adaptive and local complexity. Previously, Snyder (1992) and Plantinga-Vegter (2004) have introduced techniques based on parametrizability and non-local isotopy, respectively. In our previous work (SoCG 2009), we synthesized these two techniques into an efficient and practical algorithm for curves. This paper extends our approach to surfaces. The extension is by no means routine: the correctness argument is much more intricate. Unlike the 2-D case, a new phenomenon arises in which local rules for constructing surfaces are no longer sufficient.

We treat an important extension to exploit anisotropic subdivision. Anisotropy means that we allow boxes to be split into 2, 4 or 8 subboxes with arbitrary but bounded aspect ratio. This could greatly improve the adaptivity of the algorithm.

Our algorithms are relatively easy to implement, as the underlying primitives are based on interval arithmetic and exact BigFloat numbers. We report on encouraging preliminary experimental results.

Key words: Mesh Generation, Surface Approximation, Isotopy, Parametrizability, Subdivision Algorithms, Interval Methods, Topological Correctness, Exact Numerical Algorithms.

1. Introduction

A basic problem in areas such as physics simulation, computer graphics and geometric modeling is that of computing approximations of curves and surfaces from implicit definitions. Typically, the surface is represented by an equation, $f(x, y, z) = 0$ as illustrated in Figure 1. We assume the approximation is a triangulated surface, also known as a **mesh**. The recent book edited by Boissonnat and Teillaud [3] provides an algorithmic perspective for this general area; chapter 6 in particular is a survey of meshing algorithms.

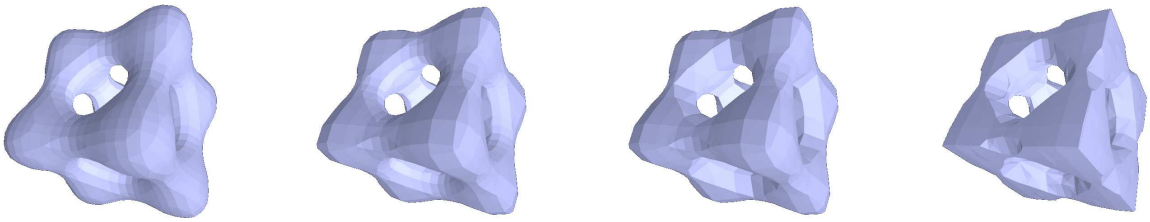
The approximate surface or mesh must satisfy two basic requirements: topological correctness and geometric accuracy. For instance, Figure 1(c) is produced by our algorithm with only topological correctness as stopping criterion. For some applications, this is sufficient. But if one desires geometric accuracy as well, this can be further refined as in Figure 1(a), where the error bound is $\varepsilon = 0.25$.

We formulate the **mesh generation problem** ("meshing problem" for short) thus: *given a region-of-interest (ROI) $B_0 \subseteq \mathbb{R}^3$, an error bound $\varepsilon > 0$, and a surface S implicitly represented by an equation $f(x, y, z) = 0$, to find a piecewise linear ε-approximation G of $S \cap B_0$.*

Here, G is an ε-approximation of $S \cap B_0$ if the Hausdorff distance between G and $S \cap B_0$ is at most ε. Topological correctness means the surface G should be isotopic to S in the interior of B_0, and also on the boundary ∂B_0; we denote this by writing "$G \simeq S$ (**mod** B_0)". Typically, an algorithm would first compute a topologically correct approximation G, and subsequently refine G until it has the desired ε-accuracy.

In order to guarantee topology, researchers traditionally resort to algebraic methods. See [3, Sections 3.6, 3.7] for the typical algebraic approach via projection (resultants). But this paper will emphasize numerical methods because such algorithms have adaptive complexity, making them quite efficient in practice, and they are easier to implement. Also, numerical methods are more general than algebraic ones since they are also applicable to non-algebraic functions such as frequently arise in mathematical anal-

[1] This work is supported by NSF Grant CCF-0917093.
[2] Long is currently at eBay, San Jose, CA.
[3] Jihun is currently with ILM, San Francisco, CA.



(a) Cxyze (b) PV (c) Cxyz (d) Rect-2

Fig. 1. Approximation of tangled cube $f(x, y, z) = x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2 = -10$.

ysis. For instance, although our implementation assumes that $f(x, y, z)$ is a polynomial, it is straightforward to allow functions that are composed using the elementary functions ($\sin x, \tan x, \exp x$, etc) with the usual arithmetic operations. On the negative side, our correctness require nonsingularity of f in the region of interest. Traditionally, numerical algorithms cannot provide topological guarantees: this will be our main challenge.

Throughout this paper, we fix an analytic function $f : \mathbb{R}^3 \to \mathbb{R}$, the surface $S := f^{-1}(0)$ (the zero set of f) and a region-of-interest $B_0 \subseteq \mathbb{R}^3$. The region B_0 is a "nice region" (see below) represented by an octree, and f is nonsingular in B_0, and S intersects the boundary ∂B_0 in a generic way (non-tangentially). Unless otherwise noted, we assume $\epsilon = \infty$ (i.e., we focus on isotopy, with no concern for geometric accuracy). For the algorithms of this paper, it is easy refine to any desired $\epsilon > 0$ once we have the correct isotopy.

1.1. Subdivision Algorithms.

Our main algorithmic paradigm is **(domain) subdivision** where an initial axes-parallel box $B_0 \subseteq \mathbb{R}^3$ is repeatedly subdivided into smaller boxes, forming an octree T rooted at B_0. Each non-leaf of T can have 2, 4 or 8 children, corresponding to half-, quarter- or full-splits of boxes into congruent subboxes. The leaves of T form a partition of B_0 into boxes. To "expand" T means to split its leaves. All algorithms in this paper are viewed as instances of the following:

Generic Subdivision Algorithm:
INPUT: AN OCTREE T REPRESENTING A REGION B_0
I. Subdivision Phase:
Keep subdividing T until some stopping criterion holds.
II. Refinement Phase:
Further subdivide T until some refinement criterion holds.
III. Construction Phase:
Construct the approximation G from the refined tree T.

The conceptual question is: *what kind of stopping and refinement criteria do we need in order to ensure that the Construction Phase has sufficient information to construct an isotopic approximation G?* This question is ill-formed unless we constrain the Construction Phase. The well-known Marching Cubes [17] gives us a clue: for each leaf box B, the Marching Cubes algorithm computes a small surface patch $G_B \subseteq B$ based *only* on the signs of f at the corners of B. This is $O(1)$ work per leaf, and G is defined to be union of all these patches G_B. Such a Construction Phase is said to be **MC-like** ("Marching Cubes like"). But it is well-known that the Marching Cubes could not ensure correct isotopy. We turn to two key papers that address this shortcoming of Marching Cubes: Snyder [26] and [20]. The achievement of Plantinga & Vegter (PV) [20] is that, by using the "small normal variation predicate", they could ensure correct isotopy with a MC-like construction. Theirs is the first topologically correct algorithm for meshing of nonsingular surfaces based on numerical primitives. In contrast, the construction phase in Snyder's algorithm [26] is not MC-like, but requires highly nontrivial processing (e.g., root isolation). In [16,15], we characterize the PV approach as exploiting **non-local isotopy**. We show that the stopping criterion of PV can be weakened to the parametrizability predicate of Snyder, leading to greatly improved efficiency. Our previous result was only for curves; in this paper, we will extend it to surfaces. As we shall see, the extension to surfaces is far from routine, requiring new ideas in the algorithm as well as in its correctness proofs. For instance, a new phenomenon arises in the Construction Phase in which local rules are no longer sufficient.

Our work, though MC-like, is not directly comparable to the standard Marching Cube literature because we assume our input scalar function $f(x, y, z)$ is an analytic function. Many Marching Cube algorithms assume $f(x, y, x)$ is often trilinear or other forms of interpolated data on a given grid. Nevertheless, our approach is also be applied to such kinds of input functions (see remarks after Lemma 1 below).

Our subdivision algorithms are practical for two reasons: first, it is based on the easily implementable subdivision paradigm. Second, all our primitives are explicitly numerical. Note that some subdivision algorithm use powerful algebraic primitives (e.g., [25]) with concomitant loss in adaptive complexity and flexibility. To evaluate our primitives, we use: (a) interval methods [19,21], and (b) BigFloats, some software implementation of dyadic numbers. In practice, machine arithmetic can be exploited in two ways: first, it can replace BigFloats when machine precision suffices (taking care to detect overflows which indicate the need for higher precision). In fact all the examples in this paper are run at machine precision. Second, they can be used as filters to speed up BigFloats. See [16] for further discussion.

1.2. Our Contribution and Overview of Paper.

Our general contribution is the development of non-local isotopy techniques. Intuitively, one can construct a globally isotopy I for a surface S (mod B_0) by piecing together a collection of local isotopies I_B for S (mod B) where B ranges over a subdivision of B_0. In the non-local isotopy approach, we compute $\widetilde{I}_B$ that are not necessarily isotopies (mod B), but when they are pieced together, the global map $\widetilde{I}$ is a global isotopy (mod B_0). By not insisting on local isotopies, we can use cheaper predicates and avoid excessive subdivision. Overall, we expect to gain in efficiency. Although such algorithms are harder to prove correct, the actual algorithms remain relatively simple and easy to implement. Our implementation and experiments will bear out the expected speed up.

Our technical contribution comprises three new exact numerical algorithms for isotopic surface approximation. These algorithms, in order of increasing sophistication, are called the **Regular/Balanced/Rectangular Cxyz Algorithms**. Abbreviate them as Reg/Cxyz/Rect, respectively. Note that **Cxyz** is the name of the parametrizability predicate used by all three algorithms, but the Balanced Cxyz Algorithm inherits the "Cxyz" abbreviation. Our main goal is Cxyz and Rect. But Reg has merit of simplicity (easy to implement and useful for simple applications). Moreover, its correctness proof is an important step towards understanding the correctness of the other two. Cxyz is much more efficient than Reg, and Rect has the potential to exploit anisotropic subdivision and achieve great speedups over Cxyz.

Section 2 is a review of the literature, and Section 3 introduces the concepts of subdivision and box predicates. We then describe our three algorithms: Reg (Section 4), Cxyz (Section 5), and Rect (Section 6).

In Section 7, we outline the correctness proof, first for Reg, and then its extension to Cxyz. The further extension to Rect is routine. Section 8 contains our experimental results, and we conclude in Section 9. An Appendix contains additional visualization of our experiments. All proofs and additional experimental data are found in the thesis of Lin [15]. The thesis and the code sources are downloadable from [9].

2. Related Work

We broadly classify approaches to mesh generation into three categories: algebraic, geometric, and numerical. Algebraic approaches [1,24,8,25], exploit tools such as cylindrical algebraic decomposition (CAD), resultants, and manipulation of algebraic numbers ([3, Chapter 3] reviews these technique). These tools are exact, but the algorithms may be slow with non-adaptive complexity. A promising direction to remedy this is to combine symbolic with numeric methods [11]. The geometric approaches [27,4,6,2] postulate some abstract computational model in which geometric primitives such as ray shooting are available, and algorithms based on these primitives are constructed. Implementing these abstract models *exactly* can be an issue. E.g., ray shooting returns points with algebraic coordinates, which may be unsuitable for implementation. The numerical approaches [12,17,20,18,22,28,29] are based on numerical approximations, evaluation and derivatives of function, and interval methods. It is the most pragmatic of the three approaches. Its advantages include having adaptive and local complexity, and relative ease of implementation. Guaranteeing topological correctness is the traditional weakness of this approach. The non-local isotopy idea of this paper can be exploited in other applications: recently we constructed a new subdivision method for complex root isolation [23] that has proved very efficient [13,14]. To motivate the approach of our paper, we review four particular subdivision algorithms: Marching Cubes [17], Snyder's Algorithm [26], Plantinga & Vegter's (PV) Algorithm [20], and our Cxy Algorithm (in 2-D) [16]. We use the framework of the Generic Subdivision Algorithm in the introduction.

2.1. Marching Cubes.

Marching Cubes is one of the most popular subdivision algorithms for surface reconstruction. The stopping criterion for its Subdivision Phase is "box has width $< \epsilon$" for some arbitrary $\epsilon > 0$. In the Construction Phase, we determine the sign of the function f at the corners of each leaf box B of T. Up to rotational symmetry, reflection and interchange of signs, the possible **sign types** are given in Figure 2. Note that Marching Cube has 15 cases ([7, Fig. 1]), but cases 11 and 14 are mirror reflections, corresponding to our Type 4c.

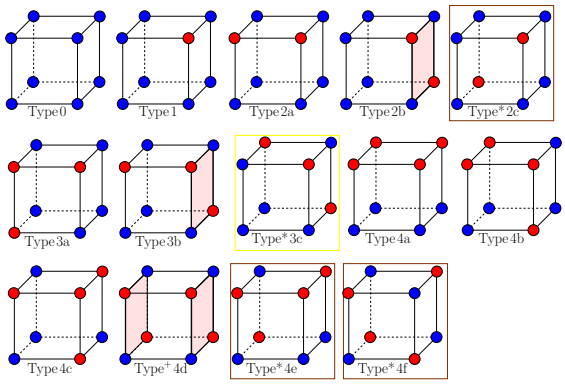


Fig. 2. The 14 Sign Types of f at box corners: only 10 may arise under C_{xyz} Predicate

If an edge of B has different signs at its two corners, we introduce a **vertex** in the middle of the edge. We then connect pairs of vertices on faces of B by **arcs**. Some possibilities for these **arc types** are illustrated in Figure 3 (our figure shows only those types that can arise in our algorithm). Note that Sign Types 2b, 3b and 4d each gives rise to two arc types, and they are topologically distinct. This "ambiguity" will be one of our main correctness concerns. Once the arcs are fixed, we can introduce a triangulated surface

patch G_B in B such that G_B intersects boundary of B with the given arc type. The union $G = \bigcup_B G_B$ of these patches constitutes an approximation of S. We say S intersects box B **cleanly** if (i) it does not pass through a corner of B, (ii) intersects any edge of B at most at one point and this intersection (if any) must be transversal, and (iii) intersects any face of B is an open curve and not in a loop (this includes the case of a degenerate loop with just one tangent point).

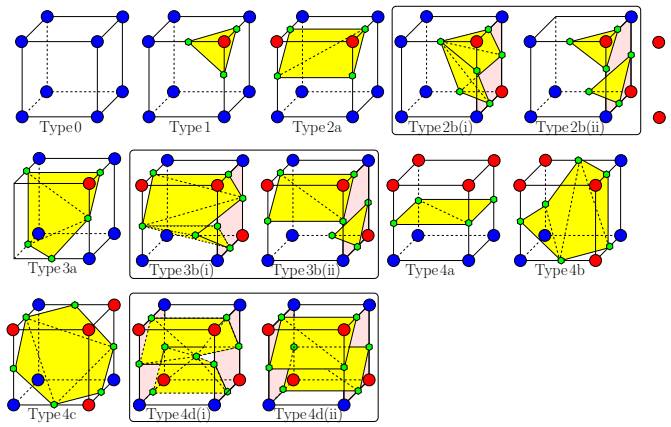


Fig. 3. The 13 Arc Types under C_{xyz} Predicate.

LEMMA 1 *Let the surface S intersect a box B cleanly. Then the intersection of S with the boundary ∂B corresponds to one of the 13 arc types in Figure 3. Moreover, each arc type uniquely determines the isotopy of the surface patch $S \cap B$.*

This lemma follows by case analysis. These 13 cases should be contrasted with Chernyaev's famous 33 cases in his analysis of Marching Cube [7]. It highlights the difference in our approaches: the assumption of clean intersection in this lemma is exploiting isotopy. In general, S does not intersect B cleanly, but there is a "vertex avoiding" isotopy of S which does cleanly intersect B. Eventually, our main result shows that by subdivision, our algorithm only need to form surface patches corresponding to these 13 cases, rather than Chernyaev's 33. We stress that "cleanliness" is a concept used in the correctness analysis.

2.2. Parametrizability of Snyder.

A key paper towards ensuring correct topology in subdivision algorithms is Snyder [26]. He introduced interval methods to determine the correct topology within each subdivision box B. Snyder's stopping criterion is "$S \cap B$ is parametrizable". This means that surface patch $S \cap B$ is the graph of some function $g(i, j)$ in two coordinate directions $i, j \in \{x, y, z\}$. This condition can be detected using interval arithmetic: we call this the $C_{xyz}(B)$ predicate below. Snyder is then able to construct a triangulated surface patch $G_B \subseteq B$ with the property $G_B \simeq S$ (**mod** B). His algorithm is recursive in dimension: to construct G_B, recursively solve the 2-D problem of computing the topology of $S \cap F$ on each face F of B. In turn, this requires solving the 1-D problem of root isolation along the edges of F. There are two issues. First, the algorithm may not terminate if S intersects the boundary of B tangentially at isolated points [3, p. 195]. Second, G_B can have arbitrary combinatorial complexity, and thus is not MC-like. This becomes harder to implement beyond 2D.

2.3. Non-local Isotopy of Plantinga & Vegter

The second key paper is from Plantinga & Vegter [20]: instead of parametrizability, they introduce two simple criteria for termination of subdivision: the **exclusion predicate** $C_0(B)$ and the **small normal variation predicate** $C_1(B)$ (see definitions below). The predicate $C_1(B)$ implies that the angle between two gradient vectors of f in B is less than 90 degrees, and in particular it implies that $S \cap B$ is parametrizable. Snyder constructs the **local isotopy** of the surface in each box B. In a radical departure from Snyder, they no longer require that G_B be isotopic to $S \cap B$. Remarkably, this approach also solves the two issues of Snyder. We view non-local isotopy very favorably because enforcing local isotopy is considered wasteful (after all, subdivision boxes are artifacts of the algorithm, not inherent in topology of S).

2.4. Our Synthesis.

Our paper [16] is a synthesis of the parametrizability approach of Snyder with the non-local isotopy of PV. We only treated curves. Basically, we want to run the PV algorithm but replacing the C_1 predicate with parametrizability. It turns out that this is justifiable provided we take care to disambiguate certain configurations by subdivisions. Although we regard C_1 as an overkill for isotopy, it has other uses for refinement and in controlling normal deviation. Experiments confirm our expectation: our synthesis is more efficient than either approach separately.

3. Preliminaries

For any set $S \subseteq \mathbb{R}$, let $\square S$ denote the set of all closed intervals with endpoints in S. We mainly use $S = \mathbb{R}$ and $S = \mathbb{F}$ where $\mathbb{F} := \{m2^n : m, n \in \mathbb{Z}\}$ denote the set of dyadic numbers (BigFloats). A **box** (or d-box) is any element of $\square\mathbb{R}^d$ $(= (\square\mathbb{R})^d)$. Usually, $d = 1, 2, 3$. If $f : \mathbb{R}^d \to \mathbb{R}$ is any function, then a function of the form $\square f : \square\mathbb{F}^d \to \square\mathbb{F}$ is called a **box function** for f if for all $B, B_i \in \square\mathbb{F}$, we have (1) (inclusion) $f(B) \subseteq \square f(B)$, and (2) (convergence) if $\lim_{i\to\infty} B_i = p \in \mathbb{R}^d$, then $\lim_{i\to\infty} \square f(B_i) = f(p)$. Note that using interval arithmetic, it is very easy to construct box functions when f is a polynomial. For a box $B = \prod_{i=1}^d I_i$, let $w(B) = \min_{i=1}^d w(I_i)$ denote the **width** of B, where $w(I)$ denotes

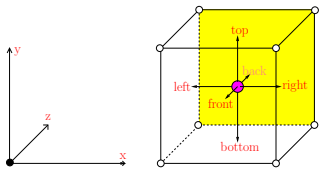


Fig. 4. Box face conventions.

the width of an interval. The 0-, 1- and 2-dimensional features of a box are called its **corners**, **edges**, and **faces**. For $i \in \{x, y, z\}$, an **i-face** is a face that is normal to the i-direction We also name each face of a box as 'front', 'back', 'top', 'bottom', etc, using the convention in Figure 4. Note that the z-direction is the vertical direction.

We may assume that f has positive or negative signs at box corners (never the zero sign); if f is zero, we simply assume it is positive (this trick of [20] amounts to an infinitesimal perturbation of the surface). Viewing signs (+ or −) as colors, we can talk about edges and boxes being **monochromatic** or **bichromatic**. As in Section 2, we introduce **vertices** in the middle of bichromatic edges. In our implemented code, we use linear interpolation to improve the quality of the meshes. On a face, we will introduce **arcs** connecting pairs of vertices (this need not be uniquely determined, as we saw). Finally, for each box B, we introduce a collection of **triangles** to form a triangulated patch G_B such that $G_B \cap \partial B$ is precisely these vertices and arcs. Thus, we use the corner/edge/face terminology for boxes, but reserve the vertex/arc/triangle terminology for the triangulated mesh.

3.1. Octrees.

We assume that each leaf of our octrees is labeled as "in" or "out". A leaf box B is called an **in-box** if it is labeled "in"; similarly for an **out-box**. The set of all the in-boxes of T is called the **box-complex** defined by T. The union of all in-boxes is denoted $R(T)$, the **region represented by** T. Following [5], a set of the form $R(T)$ is called a **nice region**. This extension is very useful: we may know that a subregion contains a singularity, and we want to exclude this subregion from our algorithm (see [5] for such an application). Such regions are closed subsets of $\mathbb{R}^3$, but could be disconnected with holes (like a donut) and voids (like a football). Two boxes of an octree are **neighbors** of each other if they have disjoint interiors but they share an open face (i.e., the relative interior of the face of one of the two boxes). We say they are **edge-neighbors** if they share an open line segment. Note that neighbors are automatically edge-neighbors, but the converse may not hold.

3.2. Box Predicates for Subdivision.

The stopping criterion of the Subdivision Phase (see Introduction) is based on two box predicates: an **exclusion predicate** $C_{out}(B)$ and an **inclusion predicate** $C_{in}(B)$. Subdivision Phase ends when each in-box B satisfies $C_{out}(B)$ or $C_{in}(B)$. The in-boxes of T fall into three mutually exclusive types:

1. Discarded Boxes: these satisfy C_{out}
2. Candidate Boxes: these do not satisfy C_{out}, but an ancestor satisfies C_{in}.
3. Inconclusive boxes: neither Discarded nor Candidate.

If B satisfies C_{in} but not C_{out}, then the above definition implies B is a candidate box (since B is an ancestor of itself). Discarded boxes will no longer be considered. Whenever we split a candidate box, we always check if each subboxes satisfy C_{out}: if so, it is discarded; otherwise it remains a candidate box. After the Subdivision Phase, no inconclusive boxes remain. For the Refinement Phase, we only split candidate boxes. The following list contains various instantiations for C_{out} and C_{in} used in this paper:

$$\left.\begin{array}{ll} C_0(B) & : 0 \notin \square f(B) \quad \text{(Exclusion)} \\ C_x(B) & : 0 \notin \square f_x(B) \quad (x\text{-Monotonicity}) \\ C_{xyz}(B) & : C_x(B) \vee C_y(B) \vee C_z(B) \quad \text{(Parametrizability)} \\ C_1(B) & : 0 \notin (\square f_x(B))^2 + (\square f_y(B))^2 + (\square f_z(B))^2 \\ & \quad \text{(Small Normal Variation)} \end{array}\right\} \quad (1)$$

Note that f_x, f_y, f_z refers to partial derivatives of f. Clearly, if $C_0(B)$ holds, then $S \cap B$ is empty. So we use C_0 as the exclusion predicate C_{out} in all our algorithms. For Snyder's and Cxyz Algorithms, $C_{in} = C_{xyz}$, and for PV Algorithm, $C_{in} = C_1$.

4. Regularized Cxyz Algorithm

An octree is "regular" if every leaf is at the same level, as in Marching Cubes. So the "Regularized Algorithm" amounts to enforcing this regularity during the Refinement Phase. In our Regularized Cxyz Algorithm, we can relax this requirement: we only require that two candidate boxes who are edge-neighbors must have the same width. The correctness of the Regularized Cxyz Algorithm is far from trivial; it also serves as an important intermediate development towards the Balanced Cxyz Algorithm whose correctness proof will be even more intricate. Thus we follow [20,16] in this two-step approach.

The algorithm only perform full-splits, and recall that its inclusion predicate C_{in} is C_{xyz}. This completely defines its Subdivision Phase. The Refinement Phase is defined by the rule that we split a candidate box B if it has an edge-neighbor that is a candidate box of smaller width. At the end of this process, any two edge-neighbors that are both candidates would have the same width. The rest of this section will focus on the Construction Phase, and correctness proof.

At this juncture, we insert a concept that will be useful in subsequent analysis. At the end of the Subdivision Phase, each candidate box B in the octree is known to satisfy $C_i(B)$ for some $i \in \{x, y, z\}$. We arbitrarily pick one of these i's and call it the **known monotone direction** ("monotone direction" for short) for B. In subsequent computation, when we split B, the candidate descendants of B will inherit this monotone direction. This direction is stored with B by our algorithm since some decisions will depend on it.

4.1. Sign Types, Arc Types and Surface Types under the C_{xyz} Predicate

Of the 14 possible sign types of f at box corners shown

in Figure 2, only 10 can arise under the Cxyz predicate. The 4 excluded cases are indicated by asterisks: Types $^*2c$, $^*3c$, $^*4e$, $^*4f$. As usual, we introduce vertices in the middle of bichromatic edges, and connect pairs of vertices on each face by arcs. The 10 sign types give rise to 13 **arc types** in Figure 3. Lemma 1 asserts that these arc types give rise to unique **surface type** within each box, shown in yellow in Figure 3.

4.2. Counter Example to the Neighborly Connection Rule.

In 2-D, we can apply the above method to construct a surface in each box, without consideration of other boxes [16]. But now, there are two choices of arc connections when a face has 4 vertices: we call these **alternating faces**. In Figure 2, these faces are colored pink, as in Types (2b), (3b) and (4d). This implies that constructing surface patches in each box must (at least) be **neighborly**, meaning that two boxes sharing an alternating face must agree on which choice of arcs to make. Alternating faces arise even under the C_1 predicate of PV Algorithm. They showed *any* neighborly choice will lead to a correct surface, which is rather non-intuitive. For our C_{xyz} predicate, neighborly choices alone is insufficient: Figure 5 gives a counter example.

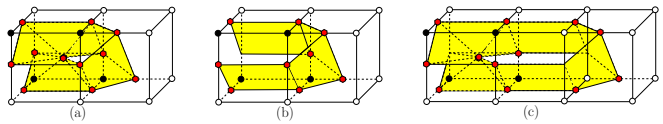


Fig. 5. Neighborly choice of arc patterns is insufficient for correctness.

In Figure 5(a), the arc connections are neighborly. The two boxes satisfy C_x, but the triangulated surface determined by the indicated arc connections violate the C_x condition. Using a different arc connection, we obtain the triangulated surface in Figure 5(b) (this one is consistent with the C_x condition). Extending this example (using the phenomenon of "blocks" below) we see that a choice in one box can force the choice of boxes arbitrarily far away. E.g., Figure 5(c).

4.3. Alternating Faces (AF) Rule.

For alternating faces, we provide the following globally consistent rule for connecting arcs: *RULE: the arcs will be line segments that are parallel to one of the three vectors:* $(1,1,0), (1,0,1), (0,1,1)$, *depending whether the alternating face is an z-, y- or x-face (respectively).* E.g., for an alternating x-face we will connect its four vertices with line segments that are parallel to the vector $(0,1,1)$, as in Type 2b(ii), and not as in Type 2b(i) of Figure 3. Call this the **Alternating Faces Rule** (AF Rule for short). With this rule, we have now completely specified the Regularized Cxyz Algorithm.

5. Balanced Cxyz Algorithm

We now extend the Regularized Cxyz Algorithm to the Balanced Cxyz Algorithm. This extension aims at reducing the number of unnecessary splits. The idea is to allow the widths of edge neighbors to differ by a factor of ≤ 2; this is called "balancing". The tradeoff is that we are faced with more involved connection rules and correctness analysis. The Subdivision Phase is the same as in the regularized case. For the Refinement Phase, we need some notation. Let $i \in \{x, y, z\}$. An edge of a box is an **i-edge** if it is parallel to the i-axis. The **i-width** of a box is the length of its i-edges. An octree is **i-balanced** if for all pairs of candidate boxes B, B' which are edge-neighbors, then the i-widths of B and B' is within a factor of 2 of each other. The octree is **balanced** if it is i-balanced for all $i = x, y, z$. This general definition will be used later for the Rectangular Cxyz Algorithm. For now, we only do full splits and we can use $w(B)$ as the definition of width.

In the rest of the Balanced Cxyz Algorithm, all our queues will be minimum priority queues. The comparison criterion for these queues is $w(B)$ for each box B. The Refinement Phase has three sub-phases:

Refinement Phase:
1. $T_1' \leftarrow \mathit{Balance}(T_1)$
2. For each candidate box in T_1', introduce vertices in the middle of bichromatic edges.
3. $T_2 \leftarrow \mathit{Disambiguate}(T_1')$

The first sub-phase $\mathit{Balance}(T_1)$ amounts to splitting any candidate box B that has an edge-neighbor of width $> 2w(B)$. At the end of this sub-phase, we say the octree is "balanced". The third sub-phase is based on the concept of ambiguity which we next introduce.

5.1. Disambiguation Sub-phase

We want to call certain boxes "ambiguous" if there is not enough information to do a MC-like construction, and this is resolved by splitting the ambiguous box. This may in turn cause new boxes to become ambiguous. In the following we will identify three kinds of ambiguity.

Let us indicate the issues that arise if we simply replace C_1 in the Balanced PV Algorithm by C_{xyz}. Consider an horizontally-stretched hyperboloid as in Figure 6 (a_1). We run the Balanced Algorithm on this hyperboloid, and the Subdivision Phase terminates with the 10 boxes shown in Figure 6 (a_2). Clearly, both of the two larger boxes (B_1 and B_3) satisfy C_x. The output graph obtained by our connection rules (in the Regularized Algorithm) is the yellow polytope G seen in Figure 6(a_2). Since G forms a closed surface, it is clearly wrong. An error occurred in box B_1 (and also B_3) where $S \cap B_1$ is a tube while $G \cap B_1$ is a planar surface. If we had split B_1, we would have discovered this error. We say B_1 (resp., B_3) has "3D ambiguity". A similar problem is seen in Figure 6(b_1), corresponding to "2D

ambiguity" in each of the boxes B_1, B_3, B_4, B_6. Suppose B satisfies C_y. Then we say B has **3D ambiguity** if the interior of its top or bottom faces has four vertices. We say B has **2D ambiguity** if one or more of its vertical faces has exactly two vertices on the same edge. Note that this edge is not a vertical edge because $C_y(B)$ is satisfied.

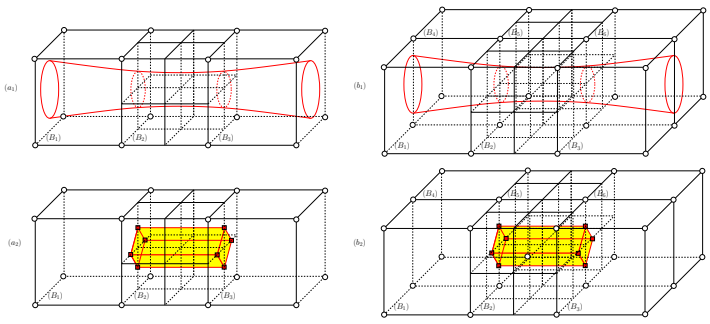


Fig. 6. Examples of 2D and 3D ambiguity.

This definition is modified accordingly if B satisfies C_x or C_z. In Figure 6(a1), the ambiguous boxes satisfies C_x. In Figure 6(b1), the ambiguous boxes might satisfy C_x or C_z.

We now describe the third kind of ambiguity. Its motivation will be become clearer in the Construction Phase below. Let $i \in \{x, y, z\}$ be the monotone direction of a box B. We say B has an **alternating ambiguity** if it properly contains the i-face F of its neighbor, and this F is alternating.

Finally, a box B is said to be **ambiguous** if it is 2D, 3D or alternating ambiguous.

LEMMA 2 *If we split an ambiguous box B into 8 subboxes, none of these subboxes will be ambiguous.*

Nevertheless, splitting of ambiguous boxes might induce its edge-neighbors to become ambiguous and also cause the octree to be unbalanced. The re-balance procedure is very local, we only need to propagate the "modified" boxes. We will next describe the Construction Phase for the Balanced Cxyz Algorithm.

5.2. Construction Phase

Let F be a face of some box B. Our first goal is to connect the vertices on F by arcs. Let B' be a neighbor of B that shares part of F as a common face. There are two possibilities: If $B' \cap B = F$, then B' has width at least that of B. This is the case we are interested in: call F **active** in this case. Otherwise, F is **inactive**; this means B' must have width that is half that of B. We are not interested in inactive F because we would have processed the faces of B' before B, and in particular, any vertex in F would have been processed. Henceforth, we will only focus on arc connections for active faces.

Recall that at the end of the Refinement Phase, we have an octree T_2 in which all the bichromatic edges have a vertex in its middle. Our goal is to connect pairs of these vertices into arcs. Define an **arc loop** to be a closed curve comprising of such arcs on the boundary of a box B. The Construction Phase has three steps:

> **Construction Phase**:
> Let Q be a priority queue of the candidate boxes in T_2.
> While (Q is non-empty)
> Remove a box B from Q
> 1. Arc connect the vertices on the active faces of B
> 2. Group the arcs on B's boundary into arc loops
> 3. Triangulate the arc loops on the boundary of B

Steps 2 and 3 are straightforward. In the following, we will describe how to implement Step 1.

5.3. Sign Types of Active Faces

Note that each edge of an active face can have at most two vertices. There might be a neighbor B' of B that shares an edge with an active F. If B' has smaller width than B, then a corner of B' would be the midpoint of an edge of F. Therefore, in considering sign types of F, we need to consider signs of such midpoints. There can be up to 8 signs on the boundary of F. The possible **Sign Types** of such faces are enumerated in Figure 7 – there are 13 in number. The sign type of F will uniquely determine the vertices that are introduced into F (as illustrated in Figure 7).

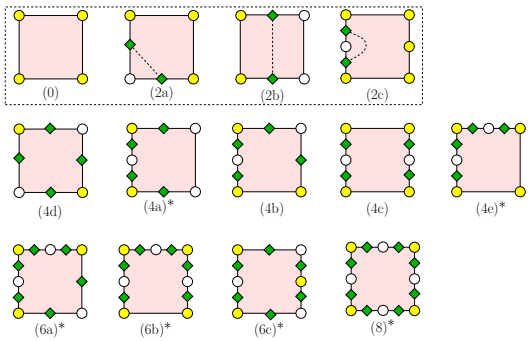


Fig. 7. Sign Types of active faces.

5.4. Arc Types of Active Faces

Let F be an active face, and suppose F bounds two boxes B and B', i.e., $F = B \cap B'$. The rule for arc connection in F depends on whether F is (known to be) "parametrizable" or not. Let us define this concept. We say F is **known parametrizable** if F is parallel to the monotone direction of B or B'. Otherwise, F is said to be **not known parametrizable**.

Assume B is a C_y box. Then the four faces of B which are parallel to the y-direction are clearly known parametrizable faces. It follows from our analysis for curves [16] that each of these faces can have at most 4 vertices. So B can have at most 16 vertices on its edges. Indeed, it is easy to see that 16 vertices can arise. Our connection rule for the known parametrizable faces can follow the rules given in [16]. For

reference, call this the **parametrizable face rule** which is reproduced in Figure 8.

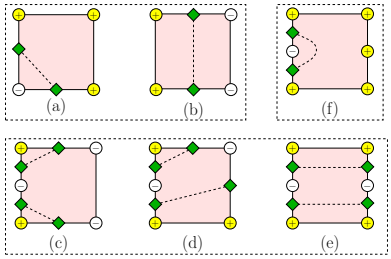


Fig. 8. Parametrizable Face Rules.

It remains to give the connection rule for the case where F is not known parametrizable. In the Regularized Algorithm, the arc connections on F may be arbitrary, as long as we ensure block-wise consistency. But the Balanced Cxyz Algorithm needs a new approach.

We define the term i-block ($i \in \{x, y, z\}$) for the balanced octree T_2. For definiteness, let $i = y$. A **y-block** $\mathcal{B}$ is a sequence $B_1, \ldots, B_t$ of candidate boxes of T_2 such that (1) the bottom face of B_j is the top face of B_{j+1} for $j = 1, \ldots, t-1$; (2) the monotone direction for each B_i is y; and (3) the block is maximal. Note that this implies that all the boxes in a block have the same width. The **width** of the block is defined as the width of any B_i. Also the **end faces** of $\mathcal{B}$ refers to the top face of B_1 and bottom face of B_t.

Recall that every candidate box in our octree T_2 has been assigned or inherited a monotone direction from the Subdivision Phase. This partitions the set of candidate boxes of T_2 into blocks as defined above. All the boundary faces of a block can be connected using the above Parametrizable Face Rule, except for the end faces which is addressed in the next lemma.

LEMMA 3 *Let F be an active end face of a block.*
(a) If F is not known parametrizable, then it has at most 2 vertices.
(b) If F is known parametrizable, then F has at most 4 vertices. When there are 4 vertices, the sign types are one of Figure 7(4b), (4c) and (4d). These can be connected using the Parametrizable Face Rule.

The correctness of above lemma depends on the fact that we have resolved alternating ambiguities in the Refinement Phase. The only faces whose connection rule remains undecided after the above discussion are those in the interior of blocks. We know from previous counter examples that there is a need for global consistency, but it cannot be solved using a simple fixed rule like the AF Rule. Our solution is as follows:
(1) if all but one face remains unconnected, we can connect this face in a safe way (i.e., one which will not lead to contradiction). This connection rule will be known as the "Matching Rule", to be given shortly.
(2) in any candidate box, at most two opposite faces cannot be connected by the Parametrizable Face Rule.

To "process" a box B in the present context means to connect all the vertices on the faces of B. We can now process B as follows: if (1) holds, we can process B by using the Matching Rule to connect its remaining unconnected face. Otherwise (2) holds, and we search in any one of the two directions of the block containing B, looking at neighboring boxes $B_1, B_2, \ldots$ until we find a box B_k that satisfies (1). Then we apply the Matching Rule to B_i for $i = k, k-1, \ldots, 1$. Thus each B_i is processed, and B can now be processed using the Matching Rule.

Let us now define the **Matching Rule** for a candidate box B with parametrization direction y. Assume that B's top face, as well as the other four faces parallel to y-direction, have been connected. Then the Matching Rule tells us how to connect the bottom face F. Let $v_1, v_2, \ldots, v_{2m}$ be the vertices on the boundary of F. Note that $m \leq 4$. The Matching Rule tells us to introduce the arc (v_i, v_j) if there exists a path of arcs on the boundary of B from v_i to v_j. Note that this rule yields a unique way to connect all the vertices on F. Figure 9 illustrates this Matching Rule. The correctness proof of the Balanced Cxyz Algorithm follows the same structure as that of the Regularized Cxyz Algorithm:

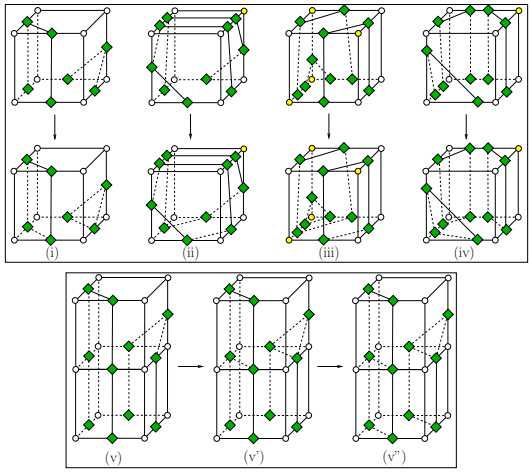


Fig. 9. Examples of matching rules ((i), (ii), (iii) and (iv)) and propagation rules ((v)→(v')→(v")) to connect vertices.

6. Rectangular Cxyz Algorithm – Exploiting Anisotropy

The ability to have partial splits (i.e., half-splits or quarter-splits) can be highly advantageous. We design such an algorithm, known as the **Rectangular Cxyz Algorithm**. A technique from the Rectangular Cxy Algorithm [16] can be applied. The implementation details are more complicated, involving four changes:
1. To ensure termination, we must fix some arbitrary upper bound $\rho > 1$ on the aspect ratio of any inconclusive box. The **aspect ratio** of a box is the ratio of the lengths of the longest edge to shortest edge.
2. For the Subdivision Phase, we test each box B as follows. We go through the following list (2),(3) of predicates (in this order):

$$
\left.
\begin{array}{l}
L_0: \\
\quad C_{out}: C_0(B) \\
\quad C_{in}: C_{xyz}(B) \\
L_1: \\
\quad C_{out}: C_0(B_{1234}), C_0(B_{5678}), C_0(B_{1278}), C_0(B_{3456}), \\
\qquad C_0(B_{1458}), C_0(B_{2367}) \\
\quad C_{in}: C_{xyz}(B_{1234}), C_{xyz}(B_{5678}), C_{xyz}(B_{1278}), \\
\qquad C_{xyz}(B_{3456}), C_{xyz}(B_{1458}), C_{xyz}(B_{2367})
\end{array}
\right\} \quad (2)
$$

$$
\left.
\begin{array}{l}
L_2: \\
\quad C_{out}: C_0(B_{12}), C_0(B_{34}), C_0(B_{56}), C_0(B_{78}), C_0(B_{14}), C_0(B_{23}), \\
\qquad C_0(B_{67}), C_0(B_{58}), C_0(B_{18}), C_0(B_{27}), C_0(B_{36}), C_0(B_{45}) \\
\quad C_{in}: C_{xyz}(B_{12}), C_{xyz}(B_{34}), C_{xyz}(B_{56}), C_{xyz}(B_{78}), \\
\qquad C_{xyz}(B_{14}), C_{xyz}(B_{23}), C_{xyz}(B_{67}), C_{xyz}(B_{58}), \\
\qquad C_{xyz}(B_{18}), C_{xyz}(B_{27}), C_{xyz}(B_{36}), C_{xyz}(B_{45})
\end{array}
\right\} \quad (3)
$$

This list amounts to checking C_0 or C_{xyz} on the whole, on half-, or quarter-parts of B. The subboxes of B are denoted B_{ij} or B_{ijk} or $B_{ijk\ell}$ $(i, j, k, \ell \in \{1, \ldots, 8\})$ using some [4] fixed convention. We use the gray code to label successive orthants, starting from $1 = 000, 2 = 001, 3 = 011, 4 = 010, 5 = 110, 6 = 111, 7 = 101, 8 = 100$. for labeling the 8 orthants of the coordinate system. This list has three sublists (L_0, L_1, L_2). If a condition in L_0 is verified we tag B as an in- or out-box, accordingly. If a condition in L_1 (L_2) is verified, we half- (quarter-) split to produce a child that satisfies that condition, and tag that child accordingly. If no condition is verified, we do a full-split.

3. For balancing, we balance in the x-, y- and z-directions independently. This could create pairs (B, B') of neighboring boxes where $B \cap B' = F$ but F is a proper subface of B and of B'. We half-split either B or B' to make F a face of a subbox. Now, F would be active, and this allows our former analysis to work.

4. The Disambiguation Sub-phase and Construction Phase are unchanged.

7. Correctness Proof

The complete proof of the correctness of our algorithms is found in the thesis of Lin [15]. Here, we give a brief overview. Recall that correctness means that the output graph G is isotopic to S in the input region $R(T_0)$, denoted "$G \simeq S$ (**mod** $R(T_0)$)". We must assume that S intersects the boundary of $R(T_0)$ non-tangentially (in a collection of topological ovals).

Let T be the final octree produced by the algorithm. The face of a leaf box of T is called a **boundary face** if it is contained in the boundary $\partial R(T_0)$ of $R(T_0)$. We assume that these boundary faces have been connected by arcs as in the planar Cxy algorithm: this leads to an isotopic approximation of $S \cap \partial R(T_0)$. The remaining faces are connected as described in our algorithm above, and finally we introduce surface patches inside each box.

Why is this construction correct? The argument consists of two major steps. First we show the existence of a surface $\widetilde{S}$ that is isotopic to S via an isotopy that respects the vertices of T (i.e., the isotopy does not move the surface past any vertex). We denote this relation by "$S \simeq \widetilde{S}$ (**mod** T)". Moreover, this surface $\widetilde{S}$ has some nice properties relative to T, namely, $\widetilde{S}$ should intersect all the segments and faces of T "cleanly". Here, "segment" means any edge of a box that does not have a corner in its interior. The intersection is clean if for any face F, $\widetilde{S} \cap F$ has no loops, and for any segment s, $|\widetilde{S} \cap s| \leq 1$. The existence of $\widetilde{S}$ is shown using a (conceptual) process to remove loops and to remove pairs of intersections on segments. To ensure termination, we define a partial order on loops and on pairs, and show that we can remove minimal elements of this partial order repeatedly. When this partial order is empty, the surface is clean.

To define this partial order, we need to maintain some monotonicity property of the surface (not the underlying function that defines the surface). Here we see a new complication: in the Regularized case, we could remove all the loops before the pairs, and so we can define two separate partial orders, on loops and on pairs. In the Balanced case, we are forced to define a single partial order on their union.

The second major step involves the notion of **alternating block**: this is a maximal set of adjacent boxes that share alternating faces. We show that $G \simeq \widetilde{S}$ within each alternating block of T. Finally, we can conclude that $G \simeq \widetilde{S} \simeq S$ (**mod** $R(T)$). This completes are proof for the Balanced Cxyz. But it is clear that the proof applies *mutatis mutandis* to the Rectangular Cxyz.

8. Experimental Results

Our algorithms are implemented in `Java` on the Eclipse Platform. All examples are run on an Intel Core2 Duo Mobile Processor T2500 (2.0Ghz, 667FSB, 2MB shared L2 Cache) and 2.0Gb of RAM. We use the default `Java` heap memory 256MB (some runs result in OutOfMemoryError (OME)). We plan to convert the `Java` codes to `C++` for distribution with our open source `Core Library`. We implemented four algorithms: PV, Balanced Cxyz, Balanced Cxyz with epsilon precision, and Rectangular Cxyz. These are abbreviated as PV, Cxyz, Cxyze, and Rect-n (where n is the maximum aspect ratio). We did not compare to Snyder as his 3D algorithm is non-trivial to implement, and it is expected to be less efficient, based on experience with the planar case [16]. Table 1 lists 11 examples of our tests. Figure 10 visualizes the surfaces of Eg2, Eg3, Eg6 and Eg7. Table 2 compares the number of boxes and timings (in ms) among Cxyz, PV, and Rect-n ($n = 2, 4, 8, 16, 32$). The percentages represent the relative number of boxes and the

[4] Unlike the 2-D case, there seems to be no universally accepted convention for this. See, e.g., http://godplaysdice.blogspot.com/2007/09/convention-for-quadrantoctantorthant.html.

relative timing, with Cxyz as 100%.

We emphasize that our speed gains over PV is, in general, at the expense of losing geometric accuracy. This is not a bad trade-off because we believe that the optimal way to achieve geometric accuracy is not by subdivision (although it is possible). Rather, it should be achieved as a post-construction phase using Newton-type iteration.

(1) Cxyz is at least as good as PV, and is significantly faster than PV in most examples. In Eg8b(4), Cxyz is 7.5 times faster than PV. In Eg8b(6), Cxyz spends 1.3 seconds to construct the mesh, compared to PV which spends more than 70 seconds and runs out of memory. Rect is the fastest in both Eg8b(4) and Eg8b(6): Rect-2 spends 141 ms for Eg8b(4), and 172 ms for Eg8b(6). The only exception is Eg8a where Cxyz and PV produce the same number of boxes, and spend the same amount of time. In Eg8b(2), we use the same function as Eg8a, but with an asymmetric original box. Cxyz is twice as fast as PV. Also note that in the Eg3, Cxyz and PV also produce the same number of boxes, but Cxyz is faster than PV because the predicate C_1 is more expensive than C_{xyz}.

(2) Rect can be significantly faster than Cxyz, but the performance of Rect is inconsistent. In Eg3, Rect-32 takes 11.8% of Cxyz's time; and in Eg8b(6), Rect-2 takes 12.8% of Cxyz's time. The input surface for these examples are very long and thin, allowing Rect to take advantage of larger aspect ratios. The results show that although Rect produces fewer boxes than Cxyz in all examples but Eg8b(2), nevertheless, the running time of Rect is not always faster than Cxyz (as in Eg2 with a "squarish" input surface). This is because Rect must spend more time checking splitting criteria, and processing boxes in 3 directions.

(3) Increasing the maximum aspect ratio n in Rect does not necessarily improve the performance of the algorithm. In Eg3, increasing the maximum aspect ratio directly improves the performance of Rect; but in Eg8b(6), it has an opposite effect. This is because increasing the maximum aspect ratio might cause the boxes to "over split" in one direction, which is also the reason for the inconsistency of Rect. Another example for over-splitting in Rect is Eg2, where Rect-n spends more time than Cxyz.

(4) We also ran our algorithm on the high order polynomial $f(x, y, z) = x^{300} + y^{300} + z^{300} - 1 = 0$. To construct a correct mesh, Cxyz uses 188 ms; PV uses 219 ms; Rect-2 uses 296 ms and Rect-4 uses 375 ms. On the other hand, starting from Rect-8, there are overflow/underflow errors. This problem can be resolved if we use a library like our `Core Library`.

9. Conclusion

This paper introduces new algorithms for the isotopic approximation of implicit surfaces. Our algorithms are relative simple, efficient and easy to implement. A main idea is to exploit parametrizability (as in Snyder) and nonlocal isotopy (as in Plantinga & Vegter), and we further extend this idea to anisotropic subdivision. Our comparison using three algorithms (PV, Balanced Cxyz, and Rectangular Cxyz) show that our Cxyz Algorithm is consistently more efficient than PV and the Rectangular Cxyz Algorithm can exhibit significant speedup. But the precise way to exploit anisotropy remains a research problem.

Three interesting areas of research are (i) to extend this work to higher dimensions (cf. [10]); (ii) effective treatment of singularity using numerical methods (cf. [5]); and (iii) complexity analysis of such algorithms for 2 or higher dimensions (cf. [23]).

References

[1] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics. Springer, 2003.

[2] J-D. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005.

[3] J.-D. Boissonnat and M. Teillaud, editors. *Effective Computational Geometry for Curves and Surfaces*. Springer, 2006.

[4] Jean-Daniel Boissonnat, David Cohen-Steiner, and Gert Vegter. Isotopic implicit surfaces meshing. In *ACM Symp. Theory of Comput.*, pages 301–309, 2004.

[5] M. Burr, S.W. Choi, B. Galehouse, and C. Yap. Complete subdivision algorithms, II: Isotopic meshing of singular algebraic curves. In *33th Int'l Symp. Symbolic and Alge. Comp. (ISSAC) (ISSAC'08)*, pages 87–94, 2008. Hagenberg, Austria. Jul 20-23, 2008. In Special Issue of JSC, vol 47, No.2, pp.131-152, 2012. Also, in arXiv:1102.5463.

[6] S.-W. Cheng, T.K. Dey, E.A. Ramos, and T. Ray. Sampling and meshing a surface with guaranteed topology and geometry. In *Proc. 20th ACM Symp. on Comp. Geom.*, pages 280–289, 2004.

[7] Evgeni V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical report, Institute for High Energy Physics, 142284, Protvino, Moscow Region, Russia, 1995.

[8] Arno Eigenwillig, Lutz Kettner, Elmar Schömer, and Nicola Wolpert. Complete, exact, and efficient computations with cubic curves. In *20th ACM Symp. on Comp. Geom.*, pages 409 – 418, 2004. Brooklyn, New York, USA, Jun 08 – 11.

[9] Exact Geometric Computation homepage, Since 1996. FAQs, downloads, documentation and links from URL `http://cs.nyu.edu/exact/`.

[10] Benjamin Galehouse. *Topologically Accurate Meshing Using Spatial Subdivision Techniques*. Ph.D. thesis, New York University, Department of Mathematics, Courant Institute, May 2009. From `http://cs.nyu.edu/exact/doc/`.

[11] H. Hong. An efficient method for analyzing the topology of plane real algebraic curves. *Mathematics and Computers in Simulation*, 42:571–582, 1996.

[12] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Trans. on Graph.*, 21(3):339–346, 2002.

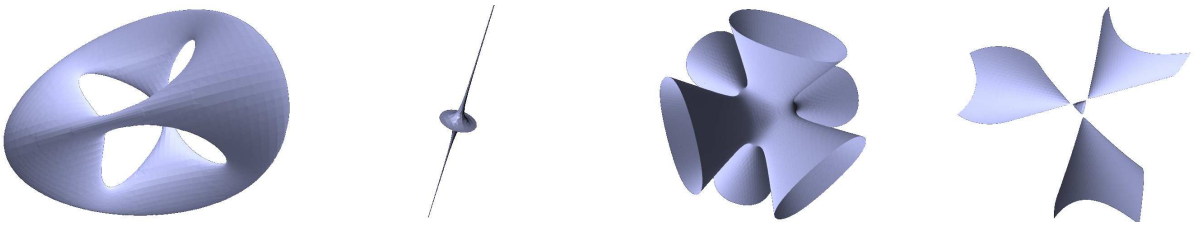
[13] Narayan Kamath. Subdivision algorithms for complex root isolation: Empirical comparisons. Msc thesis, Oxford University, Oxford Computing Laboratory, August 2010.

[14] Narayan Kamath, Irina Voiculescu, and Chee Yap. Empirical study of an evaluation-based subdivision algorithm for complex root isolation. In *4th Intl. Workshop on Symbolic-Numeric Computation (SNC)*, pages 155–164, 2011.

[15] Long Lin. *Adaptive Isotopic Approximation of Nonsingular Curves and Surfaces*. Ph.D. thesis, New York University, September 2011.

#	Surface Name	Equation $f(x,y,z)=0$	Bounding Box (ROI)
Eg1	tangle cube	$x^4-5x^2+y^4-5y^2+z^4-5z^2+10$	$[(-8,-8,-8),(8,8,8)]$
Eg2	chair	$(x^2+y^2+z^2-23.75)^2-0.8((z-5)^2-2x^2)((z+5)^2-2y^2)$	$[(-8,-8,-8),(8,8,8)]$
Eg3	quartic cylinder	$y^2x^2+y^2z^2+0.01x^2+0.01z^2-0.01$	$[(-8,-8,-8),(8,8,8)]$
Eg4	quartic cylinder	$y^2(x-1)^2+y^2(z-1)^2+0.01(x-1)^2+0.01(z-1)^2-0.2002$	$[(-5,-5,-5),(7,7,7)]$
Eg5	quartic cylinder	$y^2(x-1)^2+y^2(z-1)^2+0.01(x-1)^2+0.01(z-1)^2-1.0002$	$[(-12,-12,-12),(14,14,14)]$
Eg6	shrek	$-x^4-y^4-z^4+4(x^2+y^2z^2+y^2+z^2x^2+z^2+x^2y^2)-20.7846xyz-10$	$[(-8,-8,-8),(8,8,8)]$
Eg7	tritrumpet	$8z^2+6xy^2-2x^3+3x^2+3y^2-0.9$	$[(-8,-8,-8),(8,8,8)]$
Eg8a	eclipse	$x^2+10^2y^2+10^2z^2-1$	$[(-8,-8,-8),(8,8,8)]$
Eg8b(n) ($n=2,4,6$)	eclipse	$x^2+10^ny^2+10^nz^2-1$	$[(-7,-7,-7),(8,8,8)]$

Table 1
Equations and bounding boxes of examples



(a) Eg2:chair (b) Eg3:quartic cylinder (c) Eg6:shrek (d) Eg7:tritrumpet

Fig. 10. Approximation of various examples in Table 1.

Equation	Cxyz	PV	Rect-2	Rect-4	Rect-8	Rect-16	Rect-32
Eg1	2584 / 391	198% / 184%	42% / 148%	50% / 168%	66% / 200%	81% / 236%	103% / 288%
Eg2	26104 / 4516	406% / 349%	51% / 163%	76% / 236%	98% / 302%	118% / 372%	141% / 451%
Eg3	35792 / 3437	100% / 112%	33% / 82%	18% / 47%	9% / 28%	6% / 17%	3% / 12%
Eg4	80662 / 10282	$OME_{>90sec.}$	54% / 174%	41% / 129%	34% / 105%	36% / 115%	33% / 103%
Eg5	134163 / 17187	$OME_{>90sec.}$	48% / 205%	28% / 86%	23% / 71%	21% / 65%	20% / 61%
Eg6	31144 / 4046	319% / 296%	44% / 134%	52% / 171%	62% / 208%	70% / 255%	77% / 283%
Eg7	1688 / 328	172% / 128%	47% / 109%	50% / 119%	61% / 129%	74% / 138%	98% / 176%
Eg8a	400 / 94	100% / 100%	44% / 133%	50% / 149%	58% / 166%	68% / 166%	80% / 183%
Eg8b(2)	274 / 125	789% / 200%	54% / 87%	56% / 87%	72% / 100%	82% / 112%	102% / 112%
Eg8b(4)	1247 / 203	1774% / 754%	28% / 69%	34% / 69%	39% / 77%	44% / 85%	53% / 100%
Eg8b(6)	15226 / 1343	$OME_{>70sec.}$	5% / 13%	5% / 14%	6% / 15%	6% / 15%	7% / 16%

Table 2
Cxyz vs. PV vs. Rect-n

[16] Long Lin and Chee Yap. Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. *Discrete and Comp. Geom.*, 45(4):760–795, 2011. Special Conference Issue based on 25th ACM Symp. on Comp.Geom, 2009.

[17] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[18] Ralph Martin, Huahao Shou, Irina Voiculescu, Adrian Bowyer, and Guojin Wang. Comparison of interval methods for plotting algebraic curves. *Computer Aided Geometric Design*, 19(7):553–587, 2002.

[19] Ramon E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.

[20] Simon Plantinga and Gert Vegter. Isotopic approximation of implicit curves and surfaces. In *Proc. Eurographics Symposium on Geometry Processing*, pages 245–254, New York, 2004. ACM Press.

[21] Helmut Ratschek and Jon Rokne. *Computer Methods for the Range of Functions*. Horwood Publishing Limited, Chichester, West Sussex, UK, 1984.

[22] Helmut Ratschek and Jon G. Rokne. SCCI-hybrid methods for 2d curve tracing. *Int'l J. Image Graphics*, 5(3):447–480, 2005.

[23] Michael Sagraloff and Chee K. Yap. A simple but exact and efficient algorithm for complex root isolation. In Ioannis Z. Emiris, editor, *36th Int'l Symp. Symbolic and Alge. Comp. (ISSAC)*, pages 353–360, 2011. June 8-11, San Jose, California.

[24] Elmar Schoemer and Nicola Wolpert. An exact and efficient approach for computing a cell in an arrangement of quadrics. *Comput. Geometry: Theory and Appl.*, 33:65–97, 2006.

[25] Raimund Seidel and Nicola Wolpert. On the exact computation of the topology of real algebraic curves. In *Proc. 21st ACM Symp. on Comp. Geom.*, pages 107–116, 2005. Pisa, Italy.

[26] J. M. Snyder. Interval analysis for computer graphics. *SIGGRAPH Comput.Graphics*, 26(2):121–130, 1992.

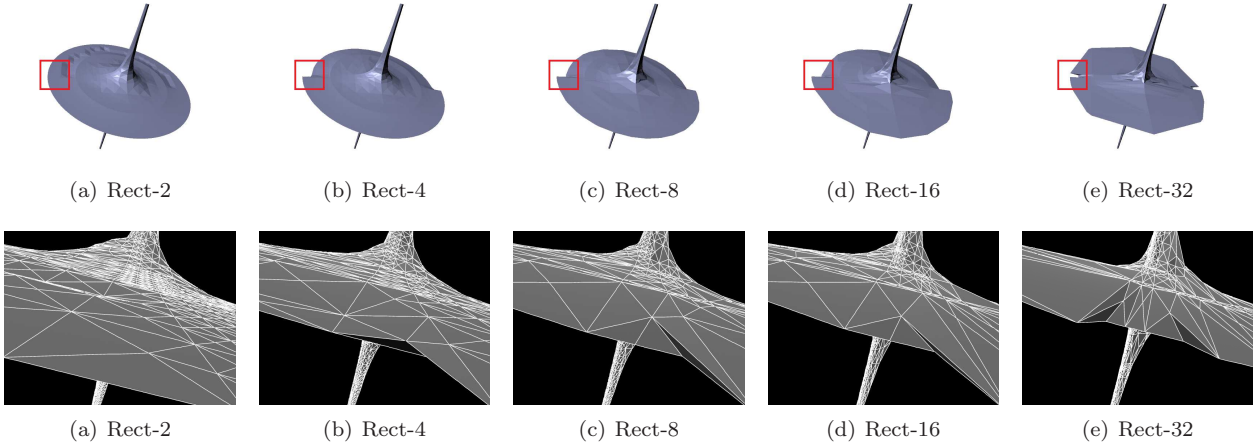
[27] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonalization for interactive meshing. In *Proc. 24th Computer Graphics and Interactive Techniques*, pages 279–286, 1997.

[28] Gabriel Taubin. Distance approximations for rasterizing implicit curves. *ACM Transactions on Graphics*, 13(1):3–42, 1994.

[29] Gabriel Taubin. Rasterizing algebraic curves and surfaces. *IEEE Computer Graphics and Applications*, 14(2):14–23, 1994.

APPENDIX

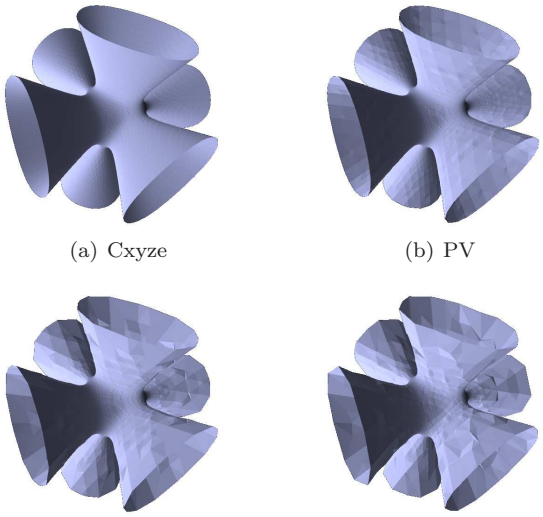
This appendix illustrates some of the surfaces from Table 1 using Cxyze, PV, Cxyz and Rect-n. Here, n is selected so that Rect-n is the fastest among various Rect algorithms.



(a) Rect-2 (b) Rect-4 (c) Rect-8 (d) Rect-16 (e) Rect-32

(a) Rect-2 (b) Rect-4 (c) Rect-8 (d) Rect-16 (e) Rect-32

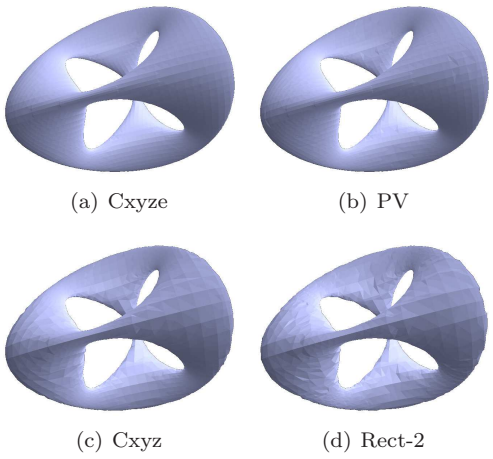
Fig. 11. (a)-(e): Quartic cylinder1 $y^2(x-1)^2 + y^2(z-1)^2 + 0.01(x-1)^2 + 0.01(z-1)^2 - 0.2002$ using Rect-n ($n = 2, 4, 8, 16, 32$). (f)-(j): Despite the visual discontinuity, topology is preserved in the highlighted (red) area of the approximations.



(a) Cxyze (b) PV

(c) Cxyz (d) Rect-2

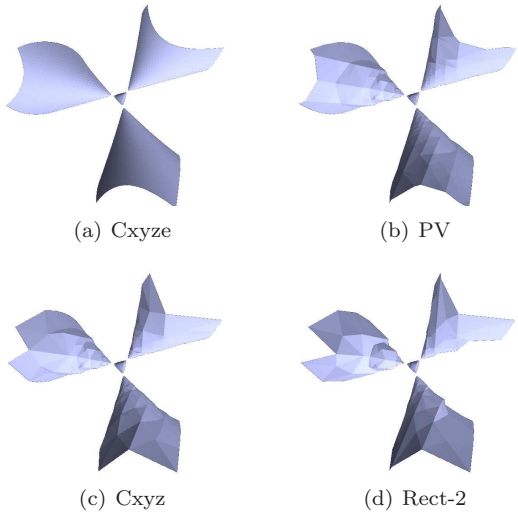
Fig. 12. Eg6: $-x^4-y^4-z^4+4(x^2+y^2z^2+y^2+z^2x^2+z^2+x^2y^2)-20.7846xyz-10$.



(a) Cxyze (b) PV

(c) Cxyz (d) Rect-2

Fig. 13. Eg2: $(x^2+y^2+z^2-23.75)^2-0.8((z-5)^2-2x^2)((z+5)^2-2y^2)$.



(a) Cxyze (b) PV

(c) Cxyz (d) Rect-2

Fig. 14. Eg7: $8z^2 + 6xy^2 - 2x^3 + 3x^2 + 3y^2 - 0.9$.