Adaptive Isotopic Approximation of Nonsingular Curves and Surfaces

by

Long Lin

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2011

Professor Chee K. Yap

To my parents.

Acknowledgements

I would like to thank my advisor Professor Chee K. Yap for his guidance during my PhD study, papers writing and thesis work. It has been a great pleasure to work with him and learn from him. I am also grateful to the members of my thesis committee for their time and comments to improve this thesis. My thesis committee members are: Professor Marsha Berger, Professor Gert Vegter, Professor Denis Zorin and Professor David Gu.

Abstract

Consider the problem of computing isotopic approximations of nonsingular curves and surfaces that are implicitly represented by equations of the form $f(X,Y) = 0$ and $f(X,Y,Z) = 0$. This fundamental problem has seen much progress along several fronts, but we will focus on domain subdivision algorithms. Two algorithms in this area are from Snyder (1992) and Plantinga & Vegter (2004). We introduce a family of new algorithms that combines the advantages of these two algorithms: like Snyder, we use the parameterizability criterion for subdivision, and like Plantinga and Vegter, we exploit nonlocal isotopy.

We first apply our approach to curves, resulting in a more efficient algorithm. We then extend our approach to surfaces. The extension is by no means routine, as the correctness arguments and case analysis are more subtle. Also, a new phenomenon arises in which local rules for constructing surfaces are no longer sufficient.

We further extend our algorithms in two important and practical directions: first, we allow subdivision cells to be non squares or non cubes, with arbitrary but bounded aspect ratios: in $2D$, we allow boxes to be split into 2 or 4 children; and in $3D$, we allow boxes to be split into 2, 4 or 8 children. Second, we allow the input region-of-interest (ROI) to have arbitrary geometry represented by an quadtree or octree, as long as the curves or surfaces has no singularities in the ROI and intersects the boundary of ROI transversally.

Our algorithm is numerical because our primitives are based on interval arithmetic and exact BigFloat numbers. It is practical, easy to implement exactly (compared to algebraic approaches) and does not suffer from implementation gaps (compared to geometric approaches). We report some very encouraging experimental results, showing that our algorithms can be much more efficient than the algorithms of Plantinga and Vegter ($2D$ and $3D$) and Snyder ($2D$ only).

Contents

List of Figures

List of Tables

Overview of Thesis

This thesis is about the problem of constructing isotopic meshes for curves and surfaces. We provide a new approach, and a family of corresponding new meshing algorithms, that combines the relative advantages of previous algorithms of Snyder and Plantinga & Vegter.

In chapter 1, we give an overview of the meshing problem, and some of the recent progress in this field. We categorize meshing algorithms into three approaches, namely algebraic, geometric and numeric. Our approach falls under the numeric algorithms.

In chapter 2, we give an introduction of Subdivision Algorithms. In particular, we describe a generic framework for meshing algorithms based on domain subdivision and review some algorithms in this framework: namely, Marching Cube Algorithm, Snyder's Algorithm and Plantinga & Vegter's Algorithm.

In chapters 3 and 4, we introduce our approach for $2D$ curve meshing and $3D$ surface meshing. We describe our algorithms for both problems. We also provide complete proofs for the correctness of our algorithms, as well as encouraging experimental results.

In chapter 5, we give the conclusion and future work.

Acknowledgements: The results in this thesis are joint work with my advisor Professor Chee K. Yap. The $2D$ work has appeared in [23]. I would also like to thank Jihun Yu for his help with rendering the figures.

Chapter 1

What is Meshing?

Approximation of curves and surfaces is a basic problem in many areas such as simulation, computer graphics and geometric modeling. The approximate surface is often a triangulated surface, also known as a mesh. See the recent book [8] for an algorithmic perspective on meshing problems; chapter 5 in particular is a survey of meshing algorithms.

By the $3D$ (resp. $2D$) meshing problem, we mean the problem of meshing surfaces (resp. curves). It is interesting to identify the $1D$ meshing with the problem of real root isolation for a real function $f(X)$. Formally, the **mesh generation problem** (or "meshing problem" for short) is this: given a region $R_0 \subseteq \mathbb{R}^d$ (typically, $d = 2, 3$) of interest, an error bound $\varepsilon > 0$, a smooth curve/surface S implicitly represented by an equation $f(X, Y) = 0/f(X, Y, Z) = 0$, to find a piecewise linear ε-approximation G of $S \cap R_0$. For $2D$ curve meshing, the mesh is just a (planar) straight line graph G (or PSLG, see [33]); for $3D$ surface meshing, the mesh G is a triangulated surface.

1.1 Correctness Criteria

The correctness criteria for G has two parts: **topological correctness** and **geometric accuracy**. Geometric accuracy is typically taken to mean that the Hausdorff distance between G and $S \cap R_0$ is at most ε (this is also known as ε-closeness):

$$d_H(S,G)(mod\ R_0) = \max\{\sup_{x\in S}\inf_{y\in G} d(x,y), \sup_{y\in G}\inf_{x\in S} d(x,y)\} \le \varepsilon$$

In recent years, the topological correctness is understood as that the approximate G should be isotopic to $S \cap R_0$, denoted $G \approx S \cap R_0$. For instance, Figure 1.1(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 1.1(a), where the error bound is $\varepsilon = 0.25$.

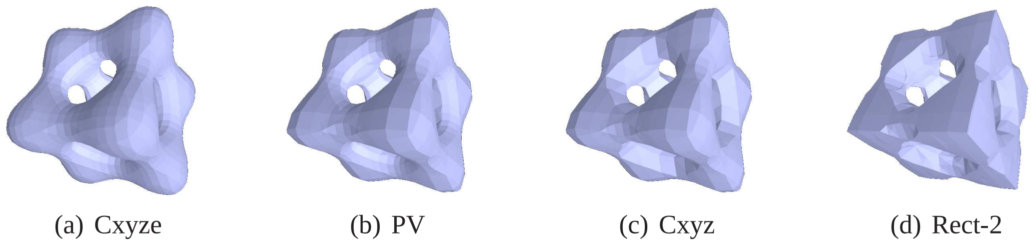


Figure 1.1: Approximation of a tangled cube $f(x,y,z) = x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2 = -10$.

Recall that a function $f : S \to S'$ between two topological spaces T_S and $T_{S'}$ is a homeomorphism if f is a continuous bijection with a continuous inverse f^{-1}. We next introduce the definition of isotopy.

DEFINITION 1. *Two surfaces S and S' is called ambient isotopic to each other if there* *exists a continuous mapping*

$$\gamma : \mathbb{R}^3 \times [0,1] \to \mathbb{R}^3$$

which, for any fixed $t \subseteq [0,1]$*, is a homeomorphism* $\gamma(\cdot, t)$ *from* $\mathbb{R}^3$ *onto itself, and which continuously deforms* S *into the mesh* S' *where* $S' = \gamma(S, 1)$.

DEFINITION 2. *Two surfaces* S *and* S' *is called isotopic to each other if there exists a continuous mapping*

$$\gamma : S \times [0,1] \to \mathbb{R}^3$$

which, for any fixed $t \subseteq [0,1]$*, is a homeomorphism* $\gamma(\cdot, t)$ *from* S *onto its image, and which continuously deforms* S *into the mesh* S' *where* $S' = \gamma(S, 1)$.

Formally, isotopy is weaker than ambient isotopy, however, for our purposes, there is no difference between isotopy and ambient isotopy: the isotopy extension lemma ensures that an isotopy between two smooth surfaces (of class C^1) embedded in $\mathbb{R}^3$ can always be extended to an ambient isotopy (see [19], Theorem 1.3 of Chapter 8, p.180). This does not directly apply to a piecewise linear surface mesh S', but it is easy to show that a piecewise linear surface is ambient isotopic to an approximating smooth surface, to which the theorem applies (see [3]).

A **tubular neighborhood** $\hat{S}$ of a surface S is a thickening of the surface such that within the volume of $\hat{S}$, the projection of a point x to the nearest point $\pi_S(x)$ on S is well-defined. The points x which have the same nearest neighbor $\pi_S(x) = p$ form a line segment through p normal to the surface. These segments are called fibers of the tubular neighborhood, and they form a partition of $\hat{S}$.

LEMMA 1. *(see [37], Theorem 4.1). Let* S *be a compact closed surface of class* C^2 *in*

$\mathbb{R}^3$ *with a tubular neighborhood* $\hat{S}$*. Let* S' *be a closed surface (not necessarily smooth) contained in* $\hat{S}$ *such that every fiber intersects* S' *in exactly one point. Then* $\pi_S : S' \to S$ *induces an ambient isotopy that maps* S' *to* S*.*

The above definitions are similar in the $2D$ case, i.e. for curves. See [3] for further discussion of isotopy. Correspondingly, the meshing problem can be solved in two stages: first we produce an output $\widetilde{G}$ that is isotopic to $S \cap R_0$. Subsequently, we refine $\widetilde{G}$ into a graph G with the requisite geometric accuracy. We may call these the **isolation** and **refinement** stages, following a terminology used for the $1D$ analogue of root approximation. The isolation stage is more challenging and usually draws most of the attention in algorithms literature. Most of our emphasis is also on the isolation stage.

1.2 Classification of Meshing Algorithms

It is helpful to begin with a classification of the meshing algorithms. There are three general approaches to meshing problems: algebraic, geometric or numeric. In practice, some algorithms are best viewed as hybrids of these approaches. All three approaches are exemplified in the survey [3].

Algebraic approaches are based on polynomial operations and algebraic number manipulation. Most algebraic algorithms can be reduced to the powerful tool of cylindrical algebraic decomposition (CAD) [2]. One example is from Mourrain and T$\acute{e}$court [29]. Usually, algebraic approaches work for curves and surfaces with self-intersections, fold lines, or other singularities, but such methods are too inefficient, even on the plane. The construction of efficient specialized algorithms remains a challenge. This has led to much interest in numerical algebraic methods (e.g., [20]). But for special cases such as quadric surfaces [39] or cubic curves [16], efficient algebraic algorithms have been devised.

Geometric approaches exploit geometric properties such as Morse theory [45, 4] or Delaunay triangulations [5, 6, 13, 1, 14]. These geometric properties are encoded into the primitives used by the algorithm. Typical primitives include the orientation predicates or ray shooting operations. Usually, geometric approaches only work for smooth curves and surfaces. However, by introducing constraints on the input, some algorithms also work for non-smooth curves and surfaces. For example, with a new sampling condition, Boissonnat and Oudot's algorithm also works for some non-smooth surfaces provided that the normal deviation is not too large at the singular points (see [7, 3]).

Numeric approaches focus on approximation and numerical primitives such as function evaluation [25, 32, 44, 23, 24, 46, 47]. Such primitives are usually embedded in simple global iterative schemes such as bisection. There is considerable work along this line in the interval arithmetic community (e.g., Martin et al [26]). These algorithms are often called "curve tracing algorithms". See Ratschek and Rokne [35] for references to curve tracing papers. Until recently, numeric approaches were shunned by computational geometers as lacking exactness or complexity analysis. This is unfortunate as practitioners overwhelmingly favor numeric approaches for three simple reasons: *(i) Efficient and easy to implement; (ii) Complexity is more adaptive; (iii) Can restrict to some region of interest.* Our overall goal is to address the above shortcomings of numeric approaches while retaining their advantages.

As suggested above, geometric algorithms are usually described in an abstract computational model that postulates certain geometric primitives (i.e., operations or predicates). These primitives may be implemented either by numerical or algebraic techniques; the algorithm itself is somewhat indifferent to this choice. For the meshing problem, a popular approach is based on sampling points on input surface [13, 5, 1, 14]. The geometric primitive here is ray-shooting; it returns the first point (if it exists) that the ray intersects on the input surface. For algebraic surfaces, this primitive reduces to a special case of real root isolation (namely, finding the smallest positive real root). The sampled points have algebraic number coordinates. In addition, the algorithms typically maintain a Delaunay triangulation of the sampled points, and thus would need orientation predicates on algebraic points. But exact implementation of these primitives requires expensive and nontrivial algebraic number manipulations. This does not seem justified in meshing applications. On the other hand, if we use approximations for sample points, they may no longer lie on the surface. This gives rise to the well-known "implementation gap" concerns of computational geometry [51]: nonrobustness, degeneracies, approximation, etc. In contrast, the subdivision methods studied in this thesis suffers no such implementation gaps. As subdivision methods are important to large communities of practitioners in numerical scientific computation, it behooves us to develop such methods into exact and quantifiable tools for numeric algorithms.

1.3 Recent Progress in Subdivision Algorithms

In this thesis, we focus on algorithms based on domain[1] subdivision methods. We view subdivision algorithms as falling under the numeric approaches (see below for the numerical computational model). The simplest form of domain subdivision uses only axes-parallel boxes (e.g., in bisection searches and Marching Cubes [25]). According to a taxonomy of meshing algorithms in [3], this form is called "cube-based scaffolding". Newman and Yi gave a survey of the development of Marching Cubes Algorithm and

[1] We use the term "domain subdivision" to refer to the subdivision of the underlying space $\mathbb{R}^2$ or $\mathbb{R}^3$ in which the curve or surface lives. Subdivision can also take place in parameter space, as in Bezier surfaces.

its extensions in [30]. The scaffolding provides a global data structure, but the implementation of the primitives must still be reduced to algebraic or numerical operations. E.g., Seidel and Wolpert [40] used algebraic primitives within this scaffolding. Our algorithms will focus on numerical primitives. Note that numerical primitives are not necessarily immune to implementation gaps. For instance, the Morse theory approach to surface meshing in [45] reveals such gaps.

There have been many attempts to extend Marching Cubes [25] from uniform to adaptive grids such as octrees. One example is from Shekhar et al. [42], but it requires crack patching efforts. Some dual approaches are used to eliminate patching problem. Two examples are Dual Contouring [21] and Dual Marching Cubes [38]. [42] and [21] use bottom up "simplification" of the regular grid to form an octree. In contrast, [38] uses a more advantageous top down subdivision scheme to construct the octree. But all of them do not have any topological guarantees. Varadhan et al. [49] introduced an algorithm for constructing a homeomorphic mesh, but their approach is not clear if inputs are functions, and has implementation gaps.

Since numerical methods traditionally do not offer topological guarantees, the key challenge is to devise methods that offer such guarantees. The direct precursors for our work are the subdivision algorithms of Plantinga & Vegter [32, 31] and Snyder [44, 43]. Both algorithms are based on interval arithmetic [28] and the ability to evaluate the exact sign of a function at bigfloat values. For a large class of functions, not necessarily algebraic, these primitives can be easily implemented exactly using a bigfloat number package. Snyder's algorithm is applicable in all dimensions (but it has termination problems as noted below). Currently, the Plantinga & Vegter method is only known in 2 and 3 dimensions. Ben Galehouse [17] has a subdivision algorithm for meshing surfaces in any dimension, but like Snyder, he requires recursive meshing of the boundary. Both

Plantinga & Vegter and Ben Galehouse use surface normal controlling primitives in their algorithms. All these algorithms are also related to the SCCI-hybrid algorithm for curve tracing by Ratschek and Rokne [35].

The problem of approximating curves defined by a bivariate polynomial is the 2-dimensional version of the general problem of approximating the hypersurface defined by a d-variate polynomial. The case $d = 3$ is clearly very important in practice. When $d = 1$, this is the classic root approximation problem. Computing up to isotopy in this case is known as the root isolation problem. Recent progress in this $1D$ problem can be found in [11, 10, 36, 22]. For a nice survey of work on the Descartes-Bernstein methods, including the so-called bit-stream algorithms, see [15]; for results related to the continued fraction method, see [41].

Both Plantinga & Vegter and Snyder assume the input curves and surfaces are non-singular. Recently, numerical subdivision algorithms that can work with singularities and degeneracies have appeared: [52] gave a Bezier curve intersection algorithm that is correct even in the presence of tangential intersection. Subdivision techniques for approximating curves with isolated singularities were given in [9]. The paper also extended the algorithm of Plantinga & Vegter to domains with irregular geometry. [11] introduced the $1D$ versions of the Plantinga & Vegter algorithm, and extended it to treat singularities (i.e., multiple zeros). Another key attraction of subdivision algorithms is their adaptive complexity. [10] introduced continuous and algebraic amortization techniques, resulting in one of the first adaptive analysis of subdivision algorithms.

Chapter 2

Overview of Subdivision Algorithms

To provide intuition for our algorithm, we will recall the work of Snyder and Plantinga & Vegter in $2D$ case. In most of our discussion, we fix a real curve

$$S := f^{-1}(0) = \left\{p \in \mathbb{R}^2 : f(p) = 0\right\}. \tag{2.1}$$

which is specified by a C^1 function $f(X, Y) : \mathbb{R}^2 \to \mathbb{R}$. We assume interval arithmetic and interval versions of functions such as f and its partial derivatives f_x, f_y.

A $2D$ **box** is given by $B = I_x \times I_y \subseteq \mathbb{R}^2$ where I_x, I_y are real intervals. Let $m(I_x)$ and $w(I_x)$ denote the midpoint and width of I_x. For a box $B = I_x \times I_y$, let $w_x(B) := w(I_x)$, $m_x(B) = m(I_x)$; similarly for $w_y(B), m_y(B)$. Then the midpoint, width and diameter of B are (resp.) $m(B) := (m_x(B), m_y(B))$, $w(B) := \min\{w_x(B), w_y(B)\}$ and $d(B) := \max\{w_x(B), w_y(B)\}$. We name the four **edges** of a box B by their relative positions (left, right, top, bottom). See Figure 2.1 for illustration of this terminology. The four **corners** are (resp.) topleft, topright, bottomleft and bottomright. The **sign** of a corner c refers to the sign of $f(c)$. By making an infinitesimal perturbation of f (which will be discussed later), we may assume that every corner c has a positive or a negative sign (never the zero sign). An edge is **monochromatic** if the sign at both of its corners are the same. A box is **monochromatic** if the sign at all of its corners are the same. Since there are only two signs, the negation of monochromatic is **bichromatic**. A **full-split** of B is to subdivide B into four equal subboxes; a **half-split** subdivides B into two equal subboxes. There are two kinds of half-splits: horizontal and vertical. These subboxes are called the **children** of B. If the children of the full split of B are denoted $B_1, \ldots, B_4$ (with B_i in the ith quadrant relative to $m(B)$), then the children in a horizontal (resp., vertical) half-split are B_{12}, B_{34} (resp., B_{14}, B_{23}), where $B_{ij} = B_i \cup B_j$. We use the edge/corner terminology for boxes, but reserve the arc/vertex terminology for the approximation straightline graphs G.

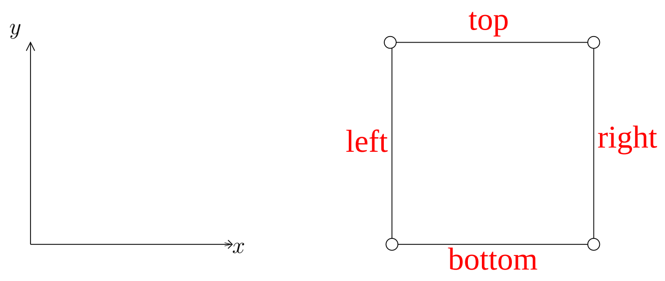


Figure 2.1: Convention and terminology for the edges of a $2D$ box

¶1. Our Computational Model To see why our algorithms are free of implementation gaps, we take a closer look at the computational model we need. Bigfloats or dyadic numbers is the set $\mathbb{F} = \mathbb{Z}[1/2] = \{m2^n : m, n \in \mathbb{Z}\}$. All numerical computations in our algorithms will be reduced to exact ring operations ($\pm, \times$) and comparisons on bigfloat numbers. Bigfloat number packages are efficient and widely available (e.g., GMP, LEDA or Core Library). More generally, $\mathbb{F}$ can be replaced by any "computational ring" [54, 53] satisfying some basic axioms to support exact real approximation. Moreover, machine arithmetic can also be used in place of BigFloats, as long as no overflow or underflow occurs; in most of our examples, this is the case. Even when high precision is needed, machine arithmetic can be exploited as filters.

We also use interval arithmetic [28]. The main tool is inclusion functions ([34]). An inclusion function for $f(X,Y)$ is a function $\square f(I_x, I_y) = \square f(B)$ that takes input intervals and returns an interval that satisfies the inclusion property: $f(B) \subseteq \square f(B)$ where $f(B) = \{f(x,y) : (x,y) \in B\}$. We call $\square f$ a **box function** for f if, in addition, it is **point convergent**, i.e., for any strictly decreasing sequence $B_0 \supset B_1 \supset \cdots$ of boxes that converges to a point p, we have $\square f(B_i) \to f(p)$ as $i \to \infty$. For our computational model, it is assumed that the input arguments to $\square f$ are dyadic boxes, and it returns a dyadic interval. We also need box versions of the derivatives, f_x, f_y.

As in [9], we call f a **PV function** if $f : \mathbb{R}^2 \to \mathbb{R}$ is C^1, and there exist computable box functions $\square f, \square f_x, \square f_y$ and the sign of f at dyadic points $p \in \mathbb{F}^2$ is computable. It will be clear that the algorithms of this thesis can be easy to implement with no numerical errors when the input f is a PV function, and all numerical inputs are dyadic. Therefore, nonrobustness issues are moot. See [9, 34] for additional information.

In contrast to our computational model, the standard model of numerical analysis only supports inexact arithmetic (up to unit round-off error). This leads to the implementation gap issues mentioned in the introduction. Such a model is assumed by Ratschek and Rokne, and even though they have the similar basic approach as ours, they had to discuss rounding errors [35, §2.5]. Moreover, in their model, computing the sign of $f(X,Y)$ at a point $p = (x_0, y_0)$ is problematic.

¶2. Generic Subdivision Algorithm The subdivision algorithms in this thesis have a simple global structure. Each algorithm has a small number of steps called **phases**. Each phase takes an input queue Q and returns some output data structure, Q'. Note that

Q' need not be a queue, but Q is always a queue of boxes. Each phase is a while-loop that extracts a box B from Q, processes B, and possibly re-insert children of B back into Q. The phase ends when Q is empty. If Q' is a queue of boxes, it could be used as input for the next phase. We next describe a generic algorithm with three phases: Subdivision, Refinement and Construction.

For the Subdivision Phase, the input Q_{in} and output Q_{out} are both queues holding boxes. The idea is to keep subdividing boxes until they satisfy certain predicates. The subdivision depends on two box predicates: an **exclusion predicate** $C_{out}(B)$ and an **inclusion predicate** $C_{in}(B)$. For each box B extracted from Q_{in}, we first check if $C_{out}(B)$ holds. If so, B is discarded. Otherwise, if $C_{in}(B)$ holds, then insert B into Q_{out}. Otherwise, we full-split B and insert the children back into Q_{in}. Next, the Refinement Phase takes the output queue from the Subdivision Phase, and further subdivides the boxes to satisfy additional criteria – these refined boxes are put in an output queue Q_{ref}. Finally, the Construction Phase takes Q_{ref} as its input and produces an output structure $G = (V, E)$ representing a planar straight line graph. As we process each box B in the input queue, we insert vertices and arcs into V and E, respectively.

```
GENERIC SUBDIVISION ALGORITHM
Input:   Curve $S$ given by $f(X,Y)=0$, box $B_0 \subseteq \mathbb{R}^2$ and $\varepsilon > 0$
Output:  Graph $G=(V,E)$ as an isotopic $\varepsilon$-approximation of $S \cap B_0$.
    0.   Let $Q_{in} \leftarrow \{B_0\}$ be a queue of boxes.
    1.   $Q_{out} \leftarrow SUBDIVIDE(Q_{in})$
    2.   $Q_{ref} \leftarrow REFINE(Q_{out})$
    3.   $G \leftarrow CONSTRUCT(Q_{ref})$
```

¶3. Example: Crude Marching Cubes Let us instantiate the generic algorithm just described, to produce a crude but still useful algorithm for "curve tracing" (cf. [26]). For the Subdivision Phase, we must specify two box predicates: let the C_{out} predicate be instantiated as

$$C_0(B) : 0 \notin \square f(B) \tag{2.2}$$

If $C_0(B)$ holds, clearly the curve S does not pass through B, and B may be discarded. Let C_{in} predicate be instantiated by $C_\varepsilon(B)$ which states that the edges of B have lengths less than some $\varepsilon > 0$. Thus, all the boxes in output Q_{out} have width $\leqslant \varepsilon$. The current Refinement Phase does nothing (so $Q_{ref} = Q_{out}$). For the Construction Phase, we must specify how to process each box $B \in Q_{ref}$. The goal is to create vertices to be inserted into V, and create arcs (which are straightline segments joining pairs of vertices) to be inserted into E. The output is a straightline graph $G = (V, E)$.

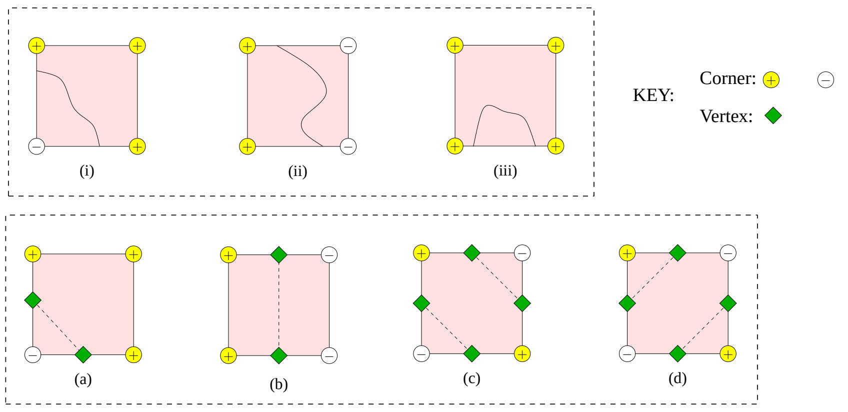


Figure 2.2: Components Types: (A) corner, (B) cut, (C) incursion. Simple Connection Rules: (a,b) corner and cut arc; (c,d) double corner arcs.

We construct G as follows: for each $B \in Q_{ref}$, we evaluate the sign of f at each of the four corners of B. If the endpoints of an edge of B have different signs, we introduce a vertex $v \in V$ at the the mid-point of the edge. Of course, if v has already been created while processing a neighboring box of B, we do not duplicate v. Clearly, B has 0, 2 or 4 vertices on its edges. If B has two vertices, we introduce an arc to connected them (see 2.2(a),(b)). These arcs represent two types of connected components of $S \cap B$: **corner** and **cut components** (respectively) as illustrated in Figure 2.2(i),(ii). A third type of connected component is an **incursion** (or B-incursion) (Figure 2.2(iii)) is not represented, but omission can be justified by isotopy (the reduction step in Figure 3.1(i,ii)). If B has 4 vertices, we introduce two pairs of non-intersecting arcs to connect them (see Figure 2.2(c,d)); there are two ways to do this, but we choose either one arbitrarily. In general, the corners of B may have a zero sign. But henceforth, we give them an arbitrary sign (say, positive). This can be justified by isotopy, as [32].

This completes our description of a crude Marching Cubes algorithm. Other subdivision algorithms to be discussed will be seen as refinements of this crude algorithm. The output graph $G = (V, E)$ is an approximation to $S \cap B_0$, up to "ε resolution". If ε is screen resolution, this is adequate for the purposes of graphical display. Martin et al [26] gave a comparative study of various numerical implementations of the box predicates C_{out}, C_{in}.

Our crude Marching Cubes makes no claims on topological correctness. Until recently, no numerical subdivision algorithms can promise much better. In particular, the ability to handle singularities is regarded as an open problem for numerical methods [3, p. 182]. But many papers assume manifolds in order to avoid singularity. In this thesis, we only assume that *the curve S has no singularities in the region R_0 of interest*. More precisely, $f^2 + f_x^2 + f_y^2$ does not vanish at any point in R_0. Our main issue is to ensure isotopy in such a situation. In domain subdivision, two related approaches have been introduced by Snyder [44] and Plantinga & Vegter [32].

¶4. Snyder's Parametrizability Approach In Snyder's approach, the predicate C_{in} is chosen to be

$$C_{xy}(B) : C_x(B) \vee C_y(B) \tag{2.3}$$

where $C_x(B)$ is the predicate $0 \notin \square f_x(B)$, and similarly for $C_y(B)$ with respect to f_y. A curve S is said to be **parametrizable in the x-direction** (or, x-parametrizable) in a box B if each vertical line intersects $S \cap B$ at most once. Clearly, $C_y(B)$ implies that S is x-parametrizable in B; this is illustrated in Figure 2.3. During the Construction Phase, we isolate the intersections of S with the boundary ∂B of each box $B \in Q_{ref}$ (this amounts to root isolation). With sufficient root refinement, we would be able to correctly construct the isotopy type of $S \cap B$. Note that this isotopy type can be arbitrarily complex, as seen in Figure 2.3.

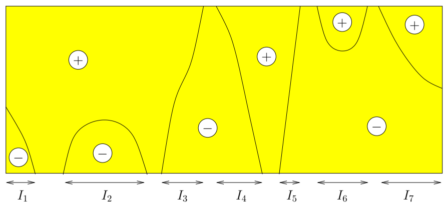


Figure 2.3: The box components of a C_y-box

¶5. Plantinga & Vegter's Small Normal Variation Approach Unfortunately, Snyder's algorithm (assuming that the method is recursively applied to the boundary of B) may not terminate[1] if the curve intersects ∂B tangentially [3, p. 195] (e.g., $f = x^2 + y^2 - 1$, and $B_0 := [(-2,-2),(2,2)]$; Snyder's algorithm would keep subdividing in boxes containing the point $(1,0)$). In view of this, the credit for the first complete subdivision algorithm to achieve isotopic approximation of nonsingular curves and surfaces

[1] In meshing curves, one can handle this problem by some root isolation method that handle multiple roots, but the problem is more serious in meshing surfaces.

belongs to Plantinga & Vegter [32]. In place of $C_{xy}(B)$, the Plantinga & Vegter (or PV) algorithm uses a stronger predicate that we denote by $C_1(B)$:

$$C_1(B) : 0 \notin (\square f_x(B))^2 + (\square f_y(B))^2. \tag{2.4}$$

It is important that the operation $[a,b]^2$ of squaring an interval $[a,b] = \square f_i(B)$ ($i \in \{x,y\}$) in (2.4) is defined as $[\min\{a^2, ab, b^2\}, \max\{a^2, ab, b^2\}]$ and not as $[0, \max\{a^2, b^2\}]$. This predicate is called the "small normal variation" condition in [3]. To see that $C_1(B)$ implies $C_{xy}(B)$, we can follow [32] by rewriting (2.4) as

$$0 \notin \langle \square \nabla f(B), \square \nabla f(B) \rangle$$

where $\nabla f(p) := (f_x(p), f_y(p))$ denotes the gradient at a point p, and $\square \nabla f(B) := (\square f_x(B), \square f_y(B))$, and $\langle \cdot, \cdot \rangle$ is just the scalar product of two vectors. This shows that if $p, q \in B$, then $\langle \nabla f(p), \nabla f(q) \rangle > 0$. Suppose some $p \in B$ has a vertical gradient (there are two choices, up or down). Then no $q \in B$ can have a horizontal gradient (there are two choices, left or right). We conclude that $f^{-1}(0) \cap B$ is parametrizable in the x-direction. There is a symmetric argument in which the roles of horizontal and vertical directions are inter-changed. The PV algorithm has a remarkable **nonlocal isotopy property**:

It does not guarantee isotopy of the approximation G with the curve S within each box B. (2.5)

We view this property favorably because local isotopy in each B is seen as an artifact of the subdivision scheme, and could greatly increase the number of subdivisions. The non-termination of Snyder's algorithm is precisely because it insists on local isotopy.

The processing of C_1-boxes is extremely simple as compared to Snyder's approach. In fact, it is a slight extension of the connection rules in our crude Marching Cubes above (see Figure 3.4). This advantage shows up even more in $3D$, where Snyder's algorithm must recursively solve the $2D$ isotopy problem on the boundary of *each* subdivision box. On the negative side, $C_1(B)$ is a stronger predicate than $C_{xy}(B)$ and may cause more subdivisions than $C_{xy}(B)$. In view of these tradeoffs, it is not immediately clear which approach is more efficient.

¶6. MC-like 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. Marching Cubes [25] gives us a clue: for each 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 box, and G is defined to be union of all these patches G_B. Such a Construction Phase is said to be **MC-like** or "Marching Cubes like" ([49] uses the same terminology for Marching Cube algorithm and its variants). The achievement of Plantinga & Vegter (PV) [32] is that, by using the "small normal variation" predicate, they could ensure correct isotopy with a MC-like construction[2]. In contrast, the construction phase in Snyder's algorithm [44] is not MC-like, but requires highly nontrivial processing (e.g., root isolation).

¶7. Other MC-like Approaches Most MC-like approaches can be formulated using our Generic Subdivision Algorithm. Schaefer and Warren [38] use quadratic error functions (developed in [18]) and some user defined tolerance ϵ as C_{in} predicate. This dual approach produces a crack free, adaptive approximation of the surface that reproduces

[2] The simplicity of the PV algorithm makes it a textbook case study, alongside the Marching Cubes.

sharp features, but it does not have any topological guarantee. Another example is from Varadhan et al. [49]. They use two criteria as the C_{in} predicate: a complex cell criterion and a star-shaped criterion. Similar to Snyder's algorithm, they require the mesh to be homeomorphic to the original surface within each box. They proved that the mesh produced by their approach is homeomorphic to the original surface, and provided the detail for the case that the input is a triangulated model. But if the input is a function, their approach is not so clear (e.g., kernel computation), and might require time consuming computation (the first criterion requires computation of the Max-Norm Distance [48] for each corner, edge and face of each box by solving equation systems). This approach has implementation gaps in many places, and also requires crack patching efforts.

¶8. Quadtrees Instead of queues, we prefer to work with a slightly more elaborate structure: a **quadtree** is a rooted tree T whose nodes u are associated with boxes B_u and if u is an internal node then it either has four or two children whose associated boxes are obtained by full- or half-splitting of B_u. Two nodes u, v are said to be **adjacent** (or **neighbors**) if the interiors of B_u and B_v are disjoint, but their boundary overlap. Overlapping means $B_u \cap B_v$ is a line segment, not just a point or empty. In order for T to represent regions of fairly complex geometry, we assume that each leaf of T is tagged with a Boolean flag, "on" or "off". The associated boxes are called **on-boxes** or **off-boxes**. The quadtree T represents a **region** denoted $R(T) \subseteq \mathbb{R}^2$ which is just the union of all the on-boxes. Following [9], we call $R(T)$ a **nice region**. A nice region is a closed subset of $\mathbb{R}^2$, but it need not be connected. Intuitively, we can substitute the queues with quadtrees T in our generic subdivision algorithm. Each phase accepts an input quadtree T, extend it, and outputs an quadtree T' (except in the last phase, when the output is the combinatorial representation of the surface).

In the subdivision phase, the C_{out} predicate is always the predicate C_0 above; it ensures that out-boxes can safely be omitted in our approximation of the curve S. This exclusion predicate is more or less universal among subdivision methods, so we only focus on C_{in}. Different subdivision methods are distinguished by their approach to inclusion predicates. Snyder's inclusion predicate is given by $C_{xy}(B)$, and Plantinga & Vegter uses a stronger inclusion predicate $C_1(B)$.

We repeatedly extend an quadtree T by splitting its on-boxes. The on-boxes of T are classified into three mutually exclusive categories:

- **Discarded Boxes**: these are on-boxes that satisfy the exclusion predicate C_{out}
- **Candidate Boxes**: these on-boxes do not satisfy the exclusion predicate C_{out}, but satisfy C_{in}.
- **Inconclusive boxes**: these on-boxes do not satisfy C_{out} or C_{in}.

No further processing is done on the discarded boxes (so they remain as leaves of T from now on). In the subdivision phase, we only split the Inconclusive boxes until every on-box is either discarded or candidate. For the refinement phase, no Inconclusive boxes remain. So we only split candidate boxes. A subtle point arises: we would like to assume that

the children of candidate boxes are either candidate or discarded. (2.6)

Property (2.6) would hold if the notion of an "C_{in} box" is **hereditary**, meaning that the children of a C_{in} box will remain C_{in}. Unfortunately, this is not guaranteed because our definition of C_{in} predicates are based on box functions, which is implementation-dependent. If the box function $\square f$ is "isotonic", the hereditary is automatic. To fix this, we redefine the concept of a candidate box: *(i) it does not satisfy C_{out}, and (ii) either its parent is a candidate box or it satisfies C_{in}.*

A **refinement** of T is obtained by a sequence of refinement steps. Note that if T' is a refinement of T, then $R(T') \subseteq R(T)$. We are interested in two properties of quadtrees T, each obtained by successive refinements:

- $REGULARIZE(T)$ returns a **regularized quadtree**, i.e., any two adjacent candidate boxes have the same depth. Thus,

  $$REGULARIZE(T) \equiv SPLIT_{C_{reg}}(T)$$

  where $C_{reg}(B) \equiv$ all candidate boxes adjacent to B have width $\geqslant w(B)$. Note that we must not replace the condition "width $\geqslant w(B)$" by "width $= w(B)$" because this can cause the smallest square to split and possibly lead to non-termination. In contrast to Plantinga & Vegter's notion of regularity which requires all the leaves to have the same depth, ours allow the leaves of different connected components of $R(T)$ to have different depths.

- $BALANCE(T)$ returns a **balanced quadtree**, i.e., one where the depths of any two adjacent candidate boxes differ by at most one. Thus,

  $$BALANCE(T) \equiv SPLIT_{C_{bal}}(T)$$

  where $C_{bal}(B) \equiv$ all candidate boxes adjacent to B have width$\geq \frac{1}{2}w(B)$.

A useful terminology is the notion of "segments" of a quadtree T. Roughly speaking, **segments** are the units into which an edge of a box is subdivided. There are two types of segments: a **boundary segment** e is an edge of an candidate box of T such that

$e \in \partial R(T)$; otherwise, it is an **internal segment**. An internal segment e has the form $e = B \cap B'$ where B, B' are adjacent candidate boxes of T. Thus each edge of a box in T is divided into one or more segments. If T is a regularized quadtree, then each edge of an candidate box of T is also a segment; if T is a balanced quadtree, then each edge of an candidate box of T is composed of either one or two segments. A **boundary box** is a candidate box that contains a boundary segment.

For now, assume the above subroutines use only full-splits; the general case where we also allow half-splits is treated in our Rectangular Algorithms.

¶9. Perturbation The correctness statements of geometric algorithms can be quite involved in the presence of degeneracy. To avoid such complications, and in the spirit of exploiting nonlocal isotopy, we exploit perturbations of f (for more details for treatment of geometric degeneracies, see [50]). We call $\tilde{f} : \mathbb{R}^2 \to \mathbb{R}$ a **nice perturbation** of $f : \mathbb{R}^2 \to \mathbb{R}$ **relative to** T if

i) $\tilde{f}^{-1}(0) \cap Interior(R(T)) \approx \tilde{f}^{-1}(0) \cap R(T)$.

ii) $\forall \epsilon > 0$, $\exists f_\epsilon : \mathbb{R}^2 \to \mathbb{R}$ such that (a) $|f(q) - f_\epsilon(q)| < \epsilon$ for $\forall q \in \mathbb{R}^2$, and (b) $\tilde{f}(p) f_\epsilon(p) > 0$, for any corner p of T.

An intuitive way to get a nice perturbation of f is to slightly shift the $S = f^{-1}(0)$ so that the resulting curve does not pass any of the corners of the boxes in T.

LEMMA 2. *For any given f and T, there exists an nice perturbation $\tilde{f}$ of f relative to T.*

We do not need an explicit $\tilde{f}$ (which depends on T which is being expanded during the algorithm). Instead, each time we evaluate f at a corner p of a subdivision box, if $f(p) = 0$ then we simply declare the sign to be positive. We justify this by saying that we are really using a nice perturbation $\tilde{f}$ instead of f. Of course, we could give f any non-zero sign at each p, as long as the sign is treated consistently for each p. This use of $\tilde{f}$ incurs no additional cost or complexity for our algorithm. For notational simplicity, we simply refer to the some nice perturbation $\tilde{f}$ as f.

Chapter 3

Isotopic Meshing of Curves

In this chapter, we will describe three increasingly sophisticated subdivision algorithms for curves. They all based on the C_{xy} predicate and will be known as the Regularized Cxy, Balanced Cxy and Rectangular Cxy Algorithms. For the first two algorithms, we only perform full-splits of boxes. We now present the first of these three algorithms.

3.1 Regularized Cxy Algorithm

Our initial goal is to replace the C_1-predicate in the PV Algorithm by the parametrizability condition of Snyder. As in Plantinga & Vegter [32], we first consider a simplified version in which we regularize the quadtree, i.e., reduce all adjacent candidate boxes to the same depth. This is our *Regularized Cxy Algorithm*, which has this form:

```
Regularized Cxy Algorithm:
Input:   Nice region given by a quadtree $T_0$ and curve $S = f^{-1}(0)$
Output:  Isotopic approximation $G$ for $S \cap R(T_0)$
0.  $T_1 \leftarrow BOUNDARY(T_0)$
1.  $T_2 \leftarrow SUBDIVIDE_{C_{xy}}(T_1)$
2.  $T_3 \leftarrow REGULARIZE(T_2)$
3.  $G \leftarrow CONSTRUCT(T_3)$
```

Note that there are four phases (Phases 0 to 3) and only Phase 0 remains to be clarified. Suppose we ignore Phase 0 (treating the operation $BOUNDARY(T_0)$ as a no-op). Then the algorithm is just an elaboration of the Crude Marching Cubes, in which we replace its (empty) Refinement Phase by a Regularization Phase, and replace the predicate C_ε by C_{xy}. The Construction Phase here is simpler than in the Crude Marching Cubes because we never have 4 vertices on the edges of an candidate box in view of condition $C_{xy}(B)$. Thus, the only connection rules we need are Figure 2.2(a,b) (i.e., Figure 2.2(c,d) are excluded).

The naive correctness statement is this: "$S \cap R(T_0)$ is isotopic to G". But this naive statement may fail because of "incursions" or "excursions" at boundary boxes. More precisely, suppose B is a boundary box and let $e \subseteq \partial R(T_0)$ be a boundary segment of B. We say S makes an **incursion** (resp., **excursion**) at e if $S \cap B$ (resp., $S \cap (\mathbb{R}^2 \setminus R(T_0))$) has a connected component C with both end points in e (Figure 2.2(iii) shows an example of incursion). Thus, C enters and exits B (resp., exits and re-enters B) at e. Such incursions/excursions are not captured by our output graph G. So a non-trivial Phase 0 is necessary to fix this problem. There is an important situation where boundary processing requires no effort at all: when S is fully contained in $R(T_0)$. Note that this was the assumption in Plantinga & Vegter's algorithm.

¶10. Boundary Processing The role of Phase 0 is to "secure" the original boundary of $R(T_0)$. This basically amounts to isolating all the intersections of S with $\partial R(T_0)$. In principle, we could invoke any exact root isolation algorithm for this purpose. However, we prefer to apply the same subdivision method, albeit applied recursively to one dimension lower. In general, for a d-dimensional subdivision algorithm, we want to recursively use the $(d-1)$-dimensional analogue for processing its boundary. For $1D$, this algorithm is essentially the EVAL algorithm for real root isolation [27, 11, 10].

The basic idea is to keep splitting any boundary box that has a potential incursion or excursion. Initially place all the boundary boxes of T_0 into a queue Q_0, and while Q_0 is non-empty, we remove a boundary box B and "tests" each of its boundary segment e (there may be one to four such segments). It e fails the test, B is split and its boundary children is put back into Q_0. If each boundary segment of B passes the "test", we discard B (i.e., it does not have to be split). But this amounts to doing nothing.

Let us now clarify the "test" on a boundary segment e. The $1D$ analogue of C_0 and C_{xy} predicates are (respectively)

$$C_0'(e) : 0 \notin \Box f(e), \qquad C_{xy}'(e) : 0 \notin \Box f_i(e)$$

where $i = x$ if e is horizontal, and $i = y$ if e is vertical. If $C_0'(e)$ holds, the curve does not intersect e. If $C_{xy}'(e)$ holds then there can be no incursion/excursion curve at e. We say that e fails the test if either $C_0'(e)$ or $C_{xy}'(e)$ does not hold. When Q_0 is empty, we terminate Phase 0. The output from this Phase is a quadtree T_1 that refines the boundary boxes of T_0 so that the curve S intersects each boundary segment of $R(T_1)$ at most once. In this case, we say S intersects the boundary of $R(T_1)$ **cleanly**.

There are still problems: if the curve intersects the boundary of $R(T_0)$ tangentially, this method does not terminate. This problem was addressed by [9], using a weakened correctness statement and a more elaborate algorithm. Also, if the curve has an end point in the interior of $R(T_0)$, our algorithm might not terminate as well. For this thesis, we shall be contented with the above simple method of boundary processing, but we need to make two strong requirements: (1) the input curve S **intersects the boundary of** $R(T_0)$ **generically**, i.e., *any intersection of S with the boundary of $R(T_0)$ is transversal;* and (2) $S \cap R(T_0)$ is compact, and any end point of $S \cap R(T_0)$ lies on the boundary $\partial R(T_0)$. By definition, transversal intersection does not allow the curve to just touching a corner of $R(T_0)$ without entering the interior of $R(T_0)$. From now on, we assume the above two requirements always hold.

¶11. Correctness It is perhaps surprising that this simple algorithm, only a small extension of Crude Marching Cubes, already produces the correct isotopy. Because it is easy to implement, it may have credible practicality.

THEOREM 3 (Correctness of Regularized Cxy Algorithm). *The algorithm terminates provided that S intersects $\partial R(T_0)$ generically and f is nonsingular inside $R(T_0)$. Moreover, the output graph G is isotopic to $S \cap R(T_0)$.*

The proof will be spread over several steps. We first prove termination. Only the boundary and subdivision phases have the potential for non-termination. The following lemma provides the condition to guarantee their termination.

LEMMA 4.
(i) If $S = f^{-1}(0)$ intersects the boundary of $R(T_0)$ generically, then the Boundary Phase will terminate.
(ii) If f has no singularities in $R(T_0)$ then the Subdivision Phase will terminate.

Proof. (i) If the Boundary Phase does not terminate, then there is an infinite decreasing sequence of edges, $e_0 \supset e_1 \supset \cdots$, such that each $C'_0(e_i)$ and $C'_{xy}(e_i)$ fail. Wlog, let e_0 be horizontal and $e_i \to p$ as $i \to \infty$. Then $C'_{xy}(e_i)$ failing means $0 \in \square f_x(e_i)$. Since $\square f_x(e_i) \to f_x(p)$, we conclude that $f_x(p) = 0$. Similarly, $C'_0(e_i)$ failing implies $f(p) = 0$. This shows that $f^{-1}(0)$ intersects e_0 tangentially.

(ii) If the Subdivision Phase does not terminate, then there is an infinite decreasing sequence of boxes $B_0 \supset B_1 \supset \cdots$ such that each $C_0(B_i)$ and $C_{xy}(B_i)$ fail. Thus:

$$0 \in (\square f(B_i) \cap \square f_x(B_i) \cap \square f_y(B_i)). \tag{3.1}$$

The boxes B_i must converge[1] to some point $p \in R(T_0)$ as $i \to \infty$. Since $\square f$ is a box function for f, we conclude that $\square f(B_i) \to f(p)$. Then (3.1) implies $0 = f(p) = f_x(p) = f_y(p)$. Thus, f is singular in $R(T_0)$. **Q.E.D.**

3.2 Partial Correctness of Regularized Cxy Algorithm

The basic partial correctness technique in Plantinga & Vegter [32] is to apply isotopies which remove any excursion of the curve $S = f^{-1}(0)$ from a box B to its neighboring box B'. Such isotopies are not "local" to any single box, but it is nevertheless still fairly local, being restricted to a union $B \cup B'$ of two adjacent boxes. But in our algorithm, an excursion from B can pass through a sequence of boxes, so we need a more global view of how to apply such isotopies.

We next prove partial correctness: if the algorithm terminates, the output G is isotopic to $S \cap R(T_0)$. The key idea in the proof is to use isotopy to transform the curve

[1] The existence of p depends only on the existence of a bound r on the maximum aspect ratio – so this proof applies in the more general setting of Rectangular Cxy Algorithm later.

$S \cap R(T_0) = S \cap R(T_3)$ repeatedly, until we finally obtain a curve S^* that we can show is isotopic to G. Each transformation step removes a pair of intersections between S and the boundary of boxes, as illustrated in Figure 3.1(i,ii): the pair (a', b') is eliminated via the isotopic transformation from (i) to (ii). We say that the pair (a', b') is **reducible**. We will make this precise.

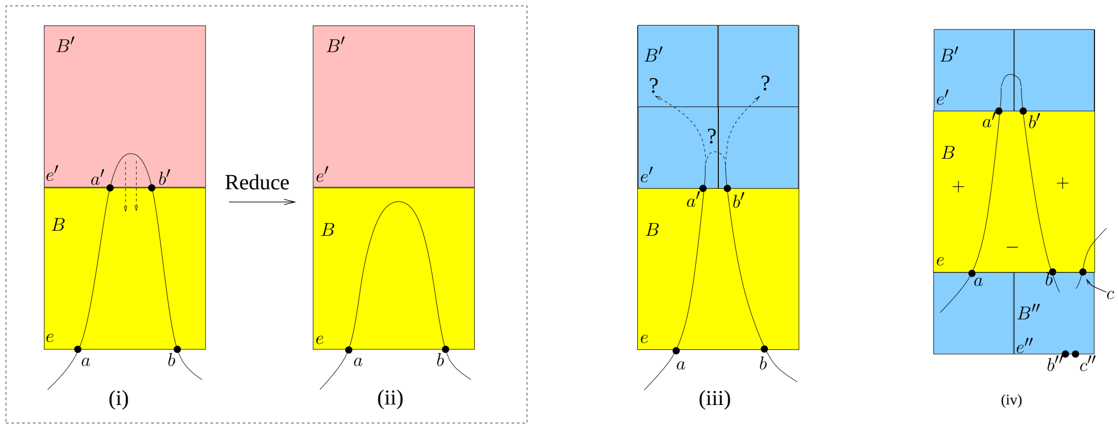


Figure 3.1: Reduction step with $(a', b') \prec (a, b)$

¶12. Partial Ordering of Convergent Pairs To give a structure for our induction, we need a partial ordering on pairs of intersection points, such as (a, b) or (a', b') in Figure 3.1(i,ii). If $a = (a_x, a_y), b = (b_x, b_y)$ are points, it is convenient to write "$a <_x b$" to mean that $a_x < b_x$. Similarly, $a <_y b$ means $a_y < b_y$. Also, $a \leqslant_x b$ means $a_x \leqslant b_x$.

Let e be a segment, so $e = B \cap B'$ for some candidate boxes B and B' (see Figure 3.1(i)). Assume C_{xy} holds at B and B'. By symmetry, assume e is a horizontal segment (the following definitions can be modified if e is vertical).

Consider the set $S \cap e$. By nice perturbation, we can assume that S has no vertical or horizontal components overlapped with edges of each candidate box (i.e., $S \cap e$ is a finite set). In general, S can intersect e at points with multiplicity greater than 1; then, as in [12], we can view $S \cap e$ as a multiset where each point $p \in S \cap e$ has multiplicity 1 or 2, according as S intersects e with odd or even multiplicity. However, we can avoid this complication by simple perturbation arguments (this will be noted in the proof below). Therefore, we assume that S intersects e transversally. Let $S \cap e = \{p_1, \ldots, p_m\}$ where the points are sorted so that $p_1 <_x p_2 <_x \cdots <_x p_m$. A pair of the form (p_i, p_{i+1}) is called a **consecutive pair** of e. Clearly, e contains a consecutive pair iff $m \geqslant 2$. Moreover, if $m \geqslant 2$ and $C_{xy}(B)$ holds, then S must be x-parametrizable in B.

A consecutive pair (a, b) of a horizontal segment e is said to be **upward convergent** if the two portions of the curve S, near a and near b (respectively), are moving closer to each other as the respective curve portions move upward across e. This is equivalent to saying that the slope of the curve S is positive at a and negative at b. This is illustrated in Figure 3.1(i) and (ii).

We have three other related definitions: if (a, b) is a consecutive pair of segment e, we say (a, b) is **downward convergent** if e is a horizontal segment and the slope of f at a is negative, and at b is positive. If e is a vertical segment, we similarly define **left or right convergent**. A key property is:

LEMMA 5. *Let $e = B \cap B'$ be a segment. If B and B' satisfies C_{xy} then every consecutive pair of e is convergent (upward, downward, left or right).*

Proof. Wlog, let e be horizontal and (a, b) be a consecutive pair of e. We must show that e is either upward or downward convergent. Since $C_{xy}(B)$ holds, the fact that $f^{-1}(0)$ intersects e in two distinct points a, b means that, in fact, $C_y(B)$ holds. Wlog, assume $f_y(B) > 0$. There are two possibilities: $f((a + b)/2) > 0$ or $f((a + b)/2) < 0$. In the former case, we have $f_x(a) > 0$ and $f_x(b) < 0$ and so the slope of $f^{-1}(0)$ at a is negative, and the slope at b is positive. This means (a, b) is downward convergent. The latter case will imply (a, b) is upward convergent. **Q.E.D.**

By symmetry, we mainly focus on upward convergent pair (a, b) of a horizontal segment $e = B \cap B'$. Because of the presence of (a, b), the curve S is x-parametrizable in B and B'; so C_y must hold at B and at B'. Wlog, we henceforth assume that $f_y(B) > 0$ and $f_y(B') > 0$.

Let $P = P(f)$ be the set of all upward convergent pairs of segments in the quadtree T_3. Note that none of these pairs lies on a boundary segment because of the Boundary Processing (¶10). Let X_a be the connected component of $B \cap S$ that contains a; similarly for X_b. Let a' be the other endpoint of X_a; similarly for b'. In case $X_a = X_b$, we have $a' = b$ and $b' = a$ and X_a is a B-incursion. Hence we call (a, b) an **incursion pair** (see Figure 3.1(ii)). But suppose $X_a \neq X_b$, then X_a and X_b are cut components (see Figure 3.1(i)) satisfying

$$a <_x a' <_x b' <_x b$$

because C_y holds in B. This is illustrated in Figure 3.1(i).

Also, it is easy to see that $f_x(a') < 0$ and $f_x(b') > 0$. Clearly S intersects the relative interior of the line segment $[a', b']$ an even number of times. If there are $2k \geqslant 0$ such intersections, then we can find $k+1$ convergent pairs on $[a', b']$. Suppose these pairs are

$$(a_0, b_0), (a_1, b_1), \ldots, (a_k, b_k)$$

where $a_0 = a'$ and $b_k = b'$. Then we define

$$(a_i, b_i) \prec (a, b) \tag{3.2}$$

for each $i = 0, \ldots, k$. Let $\preceq$ denote the reflexive, transitive closure of the set of binary relations defined as in (3.2). It is easy to see that $\preceq$ is a partial order on P. For regularized quadtrees, the minimal elements of this partial order are those (a, b) for which $X_a = X_b$ are incursion components or boundary pairs; for balanced quadtrees (next section), this is no longer true.

¶13. Compatibility So far, our box predicates C_0, C_1, C_{xy} and Phases such as $CONSTRUCT(T)$ are implicitly based on some PV function f. In order to explicitly indicate their dependence on f, we put f in the superscript as in C_0^f, C_1^f, C_{xy}^f and $CONSTRUCT^f(T)$.

Let T be a quadtree and f, g be PV functions. If for all corners u of each candidate box, we have $f(u)g(u) > 0$, then we say f and g are **compatible** on T.

Let us review the process of the Regularized Cxy Algorithm. The role of the 0th Phase is to construct a quadtree T such that for each boundary segment e of T, the curve S intersects e at most once (or S intersects $\partial R(T)$ cleanly). The 1st Phase is to make the curve x- or y-parametrizable inside each box of T. Recall that $CONSTRUCT^f(T)$ produces a straightline graph $G = (V, E)$ where, for each segment e of T, we introduce a vertex $v \in V$ iff f has opposite signs at the endpoints of e, and for each candidate box with two vertices u, v on its boundary, we introduce an arc $(u, v) \in E$.

LEMMA 6. *Let f be a PV function, and T be the quadtree after the Regularization phase in our algorithm (i.e., $T = T_3$). If $S = f^{-1}(0)$ intersects the boundary of $R(T)$ cleanly and generically, then the graph $G := CONSTRUCT^f(T)$ is isotopic to $S \cap R(T)$.*

Proof. We will inductively define a sequence $f_0, f_1, f_2, \ldots, f_n$ of C^1 functions such that $f_0 := f$ and each pair f_{i-1}, f_i is compatible over T ($i = 1, \ldots, n$) and $S_i \approx S_{i-1}$ where $S_i := f_i^{-1}(0)$.

We may assume that each S_i intersects the segments of T only transversally, and avoids the corners of candidate boxes. Hence, we can define the partial ordering $P_i =$

$P(f_i)$ of upward convergent pairs (relative to the segments of quadtree T). The transformation from S_i to S_{i-1} is illustrated by the "reduction step" of Figure 3.1(i,ii), and amounts to the removal of an upward convergent pair which is minimal in the partial order P_i. No other convergent pairs of P_{i-1} are affected by this transformation. It is then clear that $S_i \approx S_{i-1}$. Thus, we have the further property that $P_i \subseteq P_{i-1}$ with $|P_i| = |P_{i-1}| - 1 = |P_0| - i$. We stop after $n = |P_0|$ transformations, when $|P_n| = 0$.

We can similarly remove all the downward, left and right convergent pairs, by repeating the preceding process three more times. We finally arrive at a function $\overline{f}$ such that there are no consecutive pairs on any segment. According to Lemma 5, this means the curve $\overline{S} := \overline{f}^{\,-1}(0)$ intersects each segment at most once. Moreover, the curves $\overline{S}$ and $S = f^{-1}(0)$ are isotopic.

It remains to show that $\overline{S} \cap R(T) \simeq G$ where $G = CONSTRUCT^f(T)$. Let B be any candidate box of T. Since $C^f_{xy}(B)$ holds, our construction of G ensures that $|G \cap \partial B| \in \{0, 2\}$. Note that G has a vertex at a segment e iff $|\overline{S} \cap e| = 1$. Since we may assume that $\overline{S}$ does not intersect the corners of B, it follows that $|G \cap \partial B| = |\overline{S} \cap \partial B|$. In other words, $G \cap \partial B$ is isotopic to $\overline{S} \cap \partial B$. Moreover, this can be extended into an isotopy for the entire candidate box: $G \cap B$ is isotopic to $\overline{S} \cap B$.

Q.E.D.

The transformation of the function f_{i-1} into f_i can be made explicit if desired. Suppose the transformation removes the $\preceq$-minimal upward convergent pair (a, b) on segment e. Let $e = B \cap B'$ where B, B' are candidate boxes and B lies top of e. We emphasize that this transformation is local to $B \cup B'$. Let $X_{a,b}$ denote the connected component of $S_{i-1} \cap B$ whose endpoints are a, b. Let $B_{a,b}$ denote the smallest rectangle that contains $X_{a,b}$. Suppose $B_{a,b} = [x_1, x_2] \times [y_1, y_2]$. For $\epsilon > 0$, let $B^{\epsilon}_{a,b} = [x_1 - \epsilon, x_2 + \epsilon] \times [y_1 - \epsilon, y_2 + \epsilon]$. Choose ϵ sufficiently small so that $B^{\epsilon}_{a,b} \cap S_{i-1}$ is comprised of a unique component, denoted $X^{\epsilon}_{a,b}$. Now define $f_i : [x_1-\epsilon, x_2+\epsilon]\times[y_1-\epsilon, y_2+\epsilon] \to \mathbb{R}$ so that f_i is the identity on the boundary of $[x_1-\epsilon, x_2+\epsilon]\times[y_1-\epsilon, y_2+\epsilon]$, but otherwise $f_i(x,y) = f_{i-1}(x, g(x,y))$ where the function $g(x,y)$ has the property that $g(x,\cdot)$ is a piecewise linear shear. Explicit formulas for g can be given if desired. Moreover, $f_i(x,y) = 0$ implies $y < y_1$. In other words, $f_i^{-1}(0) \cap [x_1-\epsilon, x_2+\epsilon]\times[y_1-\epsilon, y_2+\epsilon] = f_i^{-1}(0) \cap [x_1-\epsilon, x_2+\epsilon]\times[y_1-\epsilon, y_1]$. Thus the component $X^{\epsilon}_{a,b}$ has moved out of B into B'. Finally, let extend the function f_i to all of the Euclidean plane by defining $f_i(x,y) = f_{i-1}(x,y)$ for all $(x,y) \notin [x_1-\epsilon, x_2+\epsilon]\times[y_1-\epsilon, y_2+\epsilon]$.

COROLLARY 7. *Let T be a regularized quadtree. If (i)f,g are compatible on T; (ii) $S_f = f^{-1}(0)$ and $S_g = g^{-1}(0)$ intersect $\partial R(T)$ cleanly and generically; and (iii) each box of T satisfies C^f_{xy} and C^g_{xy}, then $f^{-1}(0) \cap R(T) \approx g^{-1}(0) \cap R(T)$.*

Proof. Note that compatibility of f and g implies that $CONSTRUCT^f(T) = CONSTRUCT^g(T)$. By the previous lemma, we also have $f^{-1}(0) \cap R(T) \approx CONSTRUCT^f(T)$ and $g^{-1}(0) \cap R(T) \approx CONSTRUCT^g(T)$. **Q.E.D.**

Conclusion of the Proof of Theorem 3. *Proof.* Termination follows from Lemma 4. We note how each phase of the Regularized Cxy Algorithm provides the necessary properties for correctness: Phase 0 converts T_0 to T_1 which satisfies the boundary condition such that $S = f^{-1}(0)$ intersects $\partial R(T_1)$ cleanly. Phase 1 converts T_1 to T_2 which satisfies the box condition for parametrizability between T_2 and f (the boundary condition is preserved in this transformation). Phase 2 converts T_2 into a regularized quadtree, again preserving the boundary condition. Note that $f^{-1}(0) \cap R(T_0) = f^{-1}(0) \cap R(T_3)$, since the out-boxes introduced by each of these phases satisfy C_0. By Lemma 6, the output G from Phase 3 is isotopic to $f^{-1}(0) \cap R(T_3)$. **Q.E.D.**

3.3 Balanced Cxy Algorithm

The Regularized Cxy Algorithm is non-adaptive because of regularization. The **PV Algorithm** is similar to the Regularized Cxy Algorithm, except that they replace the Regularization Phase by a Balancing Phase, and use C_1 predicate instead of C_{xy}. The connection rules in the Construction Phase will become only slightly more elaborate (see below and [9, 32]).

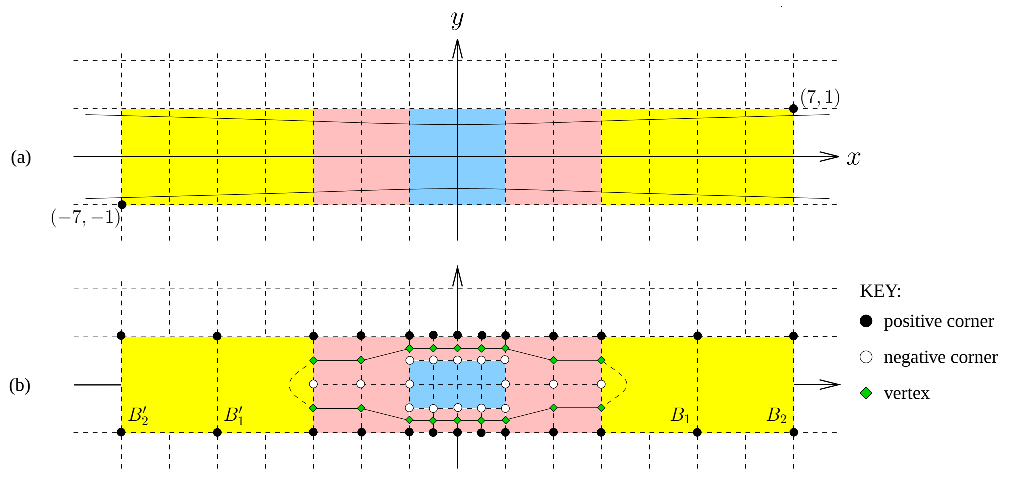


Figure 3.2: (a) Input “flat” hyperbola. (b) Output graph with wrong isotopy type.

¶14. Issue of Ambiguous Boxes We now explore the possibility of using the C_{xy} predicate in the PV Algorithm. To indicate the critical issue, consider an horizontally-stretched hyperbola $(cY+X)(cY-X)=1$ for some $c \gg 1$ as in Figure 3.2(a). We run the PV algorithm on this input hyperbola It is conceivable the Subdivision Phase ends up with the squares inside $[(-7,-1),(7,1)]$, as shown in Figure 3.2(b). Moreover, each of the four larger yellow squares (B_1, B_2, B_1', B_2') satisfy C_x, while the pink squares satisfy C_y, and blue squares[2] satisfy C_0. The output graph G obtained by using the

[2] Thanks to Prof. Gert Vegter who pointed out that there is a critical point p in the blue region. So the subdivision phase will subdivide the blue region to produce C_0 boxes that include p.

connection rules of Figure 3.4 is the graph shown in Figure 3.2(b). Since G forms a loop, it is clearly wrong. The error occurred in the boxes B_1 (and by symmetry, in B'_1). If we had split B_1, we would have discovered that there are two, not one components, in $S \cap B_1$. The box B_1 (and B'_1) is said to be "ambiguous". In general, a leaf box B is **ambiguous** if it (i) satisfies C_{xy}; (ii) is monochromatic; and (iii) has exactly two vertices. The ambiguity classification marks B for a full-split. A slightly more elaborate definition can be provided to avoid unnecessary splits[3].

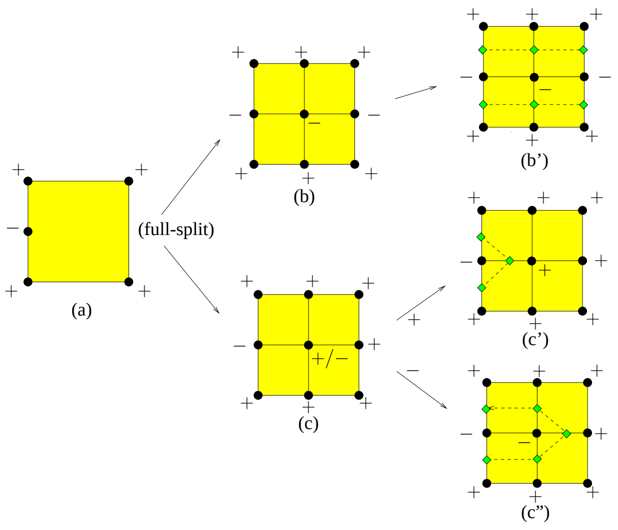


Figure 3.3: Ambiguous box (a) and its resolution (b',c',c")

Figure 3.3(a) shows an ambiguous box B (it satisfies C_x but not C_y). Note that our definition of ambiguity does not depend on whether B's top or bottom edges have been subdivided. If we full-split box B, the situation resolves into one of two possibilities, as in Figure 3.3(b) or 3.3(c). In fact, 3.3(c) has 2 subcases, depending on the sign of the midpoint of the box. In any case, splitting an ambiguous box will "disambiguate" it. In case of Figure 3.3(b), this might further cause the right neighbor of B to become

[3] I.e., we may require an optional condition: (iv) If B satisfies C_y (resp., C_x) and one of its horizontal (resp., vertical) edges need not to be subdivided.

ambiguous. This propagation of ambiguity can be iterated any number of times. But propagation of splitting can be caused also by the need to rebalance boxes. However, *both kinds of propagation will terminate because if a box splits, it is "caused" by a neighboring box of smaller size.* In our hyperbola example in Figure 3.2(b), the splitting of B_1 and B_1' will cause B_2 and B_2' to become ambiguous and be split. The final output graph will now be correct.

¶15. The Algorithm We now present the overall algorithm using our (now familiar) 4 Phases. To propagate and resolve ambiguity, we need a slightly more elaborate Construction Phase, which we call $CONSTRUCT^+$ in the following:

```
Balanced Cxy Algorithm:
Input:   Nice region given by a quadtree $T_0$ and $S = f^{-1}(0)$
Output:  Isotopic approximation $G$ for $S \cap R(T_0)$
0.   $T_1 \leftarrow BOUNDARY(T_0)$
1.   $T_2 \leftarrow SUBDIVIDE_{C_{xy}}(T_1)$
2.   $T_3 \leftarrow BALANCE(T_2)$
3.   $G \leftarrow CONSTRUCT^+(T_3)$
```

The first three phases are now standard. Our goal in the $CONSTRUCT^+(T_3)$ is to do the usual construction of the graph $G = (V, E)$, but also to disambiguate boxes. As usual, the input quadtree T_3 for $CONSTRUCT^+$ provides a queue Q of candidate boxes to be processed. However, the queue is now a priority queue. The **priority** of a box B is given by the inverse of its width (i.e., smaller width boxes have higher priority), and among those boxes with the same width, the ambiguous boxes have higher priority. We may organize this priority queue as a list $Q = (L_1, L_2, \ldots)$ of sublists. Each sublist

L_i contains all the candidate boxes of a given width (boxes in L_i has width half of those in L_{i+1}). In each sublist, the ambiguous boxes appear ahead of the non-ambiguous boxes. Note that some sublists may be empty. It is easy to manipulate these lists: when a box is removed from L_i to be split, its children goes into sublist L_{i+1}. If a box in L_i becomes ambiguous because of insertion of two new vertices on one of its edges, it is moved to the front of its sublist. The top-of-queue is the first element in the first non-empty list L_i.

We need two subroutines called

$$REBALANCE(B), \qquad PROCESS(B).$$

To "rebalance" B, we split any neighbor of B whose width is more than twice that of B, and recursively rebalance the children of its split neighbors. These children are re-inserted into the queue for future processing. More precisely:

```
$REBALANCE(B)$:
    For each candidate box $B'$ that is a neighbor of $B$
        If $w(B') > 2w(B)$,
            Full-split $B'$
            For each child $B''$ of $B'$
                Insert $B''$ into $Q$
                $REBALANCE(B'')$
```

To "process" B, we add vertices to the edges of B (if they were not already added) and connect them according to the following rules: as shown in the next section, B has $0, 2$ or 4 vertices on its boundary. If B has 2 vertices, we connect them as for the crude

Marching Cubes Figure 2.2(a,b), but reproduced in Figure 3.4(a,b). If B has 4 vertices, it turns out that two of them will lie on one edge of B; we connect these two vertices to the other two in such a way that the arcs are non-intersecting (this connection rule is unique, unlike Figure 2.2(c,d)). These rules are summarized in Figure 3.4(a–f).

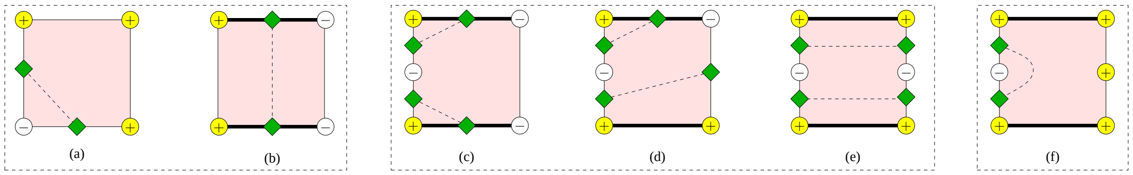


Figure 3.4: Extended Connection Rules: Cases (c–f) treats two vertices lying on one side of a box.

Four new cases arise Figure 3.4(c–f). Case (e) does not arise in the original PV algorithm. Case (f) does arise in PV but it is ambiguous and so will be eliminated by our algorithm through its disambiguation process. Thus, case (f) does not[4] arise in our current algorithm.

It is easy to see that these cases are exhaustive, and they can occur. There is an additional detail: if we add new vertices, we must also update the priority of any candidate box neighbor of B that may become ambiguous as a result. More precisely:

[4] Note that case (f) may arise if our definition of ambiguity includes the optional condition (iv).

```
$PROCESS(B)$:
    For each edge of $B$,
        If it has not been split, and has not yet been processed,
        and has a change in sign at its endpoints
            Add a vertex
            Update the priority of its neighbors (if candidate) across this edge.
    Connect the (at most four) vertices in the edges of $B$
        using the connection rules of Figure 3.4(a-e).
```

The correctness of $PROCESS(B)$ depends on the fact that any smaller boxes has already be processed. Moreover, B itself is terminal (will not be split in the future).

```
$CONSTRUCT^+(T_3)$
    Assume $T_3$ has a priority queue $Q$ containing all of its candidate boxes
    While $Q$ is non-empty
        $B \leftarrow Q.remove()$ $\triangleright$ So $B$ has the current smallest width
        If $B$ is ambiguous
            Split $B$
            For each child $B'$ of $B$
                $PROCESS(B')$
                $REBALANCE(B')$
        Else        $\triangleright$ $B$ is unambiguous
            $PROCESS(B)$
```

3.4 Correctness of Balanced Cxy Algorithm

The statement is similar to that for the Regularized Cxy Algorithm:

THEOREM 8 (Correctness of Balanced Cxy Algorithm). *The algorithm terminates provided S intersects $\partial R(T_0)$ generically and f is nonsingular in $R(T_0)$. Moreover, the output graph G is isotopic to $S \cap R(T_0)$.*

Let us first prove termination: the termination of the Boundary Phase and Subdivision Phases follows from Lemma 4. But we must also be sure that $CONSTRUCT^+(T_3)$ is terminating because of its splitting of ambiguous boxes and rebalancing. To see that this is a finite process, we observe that when a box B is split in $CONSTRUCT^+$, it is "triggered" by an adjacent box B' of smaller width. Thus, the minimum width of boxes in the quadtree is invariant. This implies termination.

The Construction Phase assumes the following property:

LEMMA 9. *Each candidate box has 0, 2 or 4 vertices on its edges. If it has 4 vertices, then two of them will lie on a common edge.*

We omit the proof which amounts to a case analysis. This is similar to the PV Algorithm [32], but we actually have a new possibility: it is possible to have two vertices on the right and two vertices on the left edge of the candidate box as shown Figure 3.4(e).

Next, we must show partial correctness. Let us see why the proof for the Regularized Cxy Algorithm does not work here: in the key lemma there (Lemma 6), we transform the function f_{i-1} to f_i by a reduction step that removes a convergent pair (a, b) that is minimal in the partial order $P(f_{i-1})$. Now, there can be "obstructions" to this reduction: in Figure 3.1(iii), the pair (a', b') is an upward convergent of e'. But in the Balanced Cxy Algorithm, the box B' might be split. Say e' is thereby split into subsegments e'_a and e'_b where $a' \in e'_a$ and $b' \in e'_b$. Thus, (a', b') is no longer a consecutive pair on any segment, and so (a, b) is now the minimal pair in $P(f_{i-1})$. There are two possibilities: (1) We might still be able to reduce the pair (a', b'), but we note that the new f_i is no longer compatible with f_{i-1} relative to T_3. (2) It might also happen that B' was split because the component X'_a of $S \cap B'$ with endpoint a' and the component X'_b with endpoint b' are different, so we cannot do reduction.

In view of the above discussion, we say that an upward convergent $(a, b) \in P(f)$ is **irreducible** if it is minimal in the partial order $P(f)$ but it is not an incursion pair (see Figure 3.1(iii)).The following lemma is critical in the correctness proof. It says that if there exists irreducible minimal pairs, then there exists ambiguous boxes:

LEMMA 10. *Let T be a balanced quadtree in the Construction phase. Let Q_u (resp., Q_d) be the set of all minimal upward (downward) convergent pairs of T. Assume $Q_u \cup Q_d$ is non-empty, and each pair in $Q_u \cup Q_d$ is irreducible.*
(i) If a segment e contains a convergent pair of Q_u, then e is the entire bottom edge of an candidate box.
(ii) One of the candidate boxes of T is ambiguous.

Proof. Let e be a segment containing a pair $(a, b) \in Q_u \cup Q_d$. Wlog, (a, b) is an irreducible upward convergent pair. Assume e lies in the bottom edge of candidate box B. See Figure 3.1(iii).

(i) First, we show that e is the entire bottom edge of B. In other words, the bottom edge of B is not composed of two segments, one of which is e. Since $C_{xy}(B)$ holds and there are two distinct points a, b on the bottom edge of B, it follows that $0 \notin f_y(B)$. As usual, let X_a, X_b be the connected components of $f^{-1}(0) \cap B$ with one endpoint at a, b (resp.). Clearly, $X_a \neq X_b$ since (a, b) is irreducible. If the other endpoints of X_a, X_b are a' and b' (resp.), then a' and b' lie on the top edge (call it e') of B. Moreover, $a <_x a' <_x b' <_x b$ and, by irreducibility of (a, b), we must have a', b' lying in different subsegments of e'. Then the subsegment e'_a containing a' (resp., b') would have $w(e'_a) \leqslant w(e)/2$. If e is not the entire bottom edge of B, then this contradicts the assumption that

T is balanced because $w(B) \geqslant 2w(e) \geqslant 4w(e'_a)$.

(Of course, an analogous statement is true: if e contains a pair of Q_d: in this case, e must be the entire top edge of an candidate box.)

(ii) We next show that B must be ambiguous under the additional assumption that the width $w(e)$ of e is minimum among all such choices of e (i.e., the minimum-width segment which contains a pair of irreducible convergent pair). We now know that e is the entire bottom edge of B (Recall the assumption that e lies in the bottom edge of candidate box B, and the pair (a, b) is an irreducible upward convergent pair). We will use Figure 3.1(iv) to illustrate the following arguments. Note that b lies in the right half of e and a lies in the left half of e.

First, we show that all the corners of B have the same sign under f. Wlog, assume $f_y(B) > 0$ and $f((a+b)/2) < 0$. Then we claim that all the corners must be positive.

Suppose the bottomright corner of B is negative. Then $S = f^{-1}(0)$ must intersect e between b and the bottomright corner. We may choose c so that c is closest to b among all the intersections. We have $a <_x b <_x c$ and (b, c) is a downward convergent pair (since (a, b) is an upward convergent pair). Let (b'', c'') be the minimal convergent pair where $(b, c) \succeq (b'', c'')$ (note that (b'', c'') might be b and c themselves). By assumption, (b'', c'') is irreducible. Say (b'', c'') lies in a segment e''. By part (i), we know that e'' is the complete top edge of an candidate box B''. Let X''_b, X''_c denote the connected components of $S \cap B''$ with endpoints b'', c'' (resp.). By the irreducibility of (b'', c''), the other endpoints of X''_b and X''_c must lie in separate segments. Since b lies in the right half of e and $b <_x b'' <_x c'' <_x c$. This implies $w(e'') \leqslant w(e)/2$. This contradicts our choice of $w(e)$ to be minimal.

Thus we may assume that the bottomleft and bottomright corners of B are both positive. But the assumption that $f_y(B) > 0$ implies that the topleft and topright corners are also positive. Recall that the top edge of B is e' and it is split into two subsegments. Thus B is ambiguous iff the midpoint $m(e')$ of e' has negative sign. Note that $a' <_x m(e') <_x b'$. Note that if there are any incursions of the curve $f^{-1}(0)$ into box B between a' and b', then we would have some c' such that either (a', c') or (c', b') forms an upward convergent pair. This would contradict the minimality of (a, b). But if there are no incursions between a' and b', then the sign of $m(e')$ would be negative (same as $f((a + b)/2)$). This completes our proof. **Q.E.D.**

As corollary, if T has no ambiguous boxes, then there can be no convergent pairs ($Q_u \cup Q_d = \varnothing$).

The following is the analogue of Lemma 6 for the Regularized Cxy Algorithm:

LEMMA 11. *Let T be a balanced quadtree in the Construction phase. If T contains no ambiguous boxes, then the graph $G := CONSTRUCT^f(T)$ is isotopic to $f^{-1}(0) \cap R(T)$.*

Proof. This proceed as in the proof of Lemma 6: we can repeatedly reduce each minimal convergent pair (upward, downward, left or right) by transforming $f_0 = f$ to $f_1, f_2, \ldots$. Let $\overline{f}$ be the final function when we cannot further reduce any minimal pair. According to Lemma 10, this means there are no more convergent pairs (otherwise, there would be ambiguous boxes). This means the curve $\overline{S} = \overline{f}^{-1}(0)$ must intersect each segment e at most once. We conclude that $G = CONSTRUCT^f(T)$ is isotopic to $\overline{S} \cap R(T)$. **Q.E.D.**

Conclusion of the Correctness Proof. *Proof.* The curve $S = f^{-1}(0)$ intersects $\partial R(T_3)$ cleanly and generically. The quadtree T_3 is balanced and S is parametrizable in each candidate box of T_3. When we invoke $CONSTRUCT^+(T_3)$, T_3 is further transformed by splitting of ambiguous boxes and their rebalancing. Let T_4 be the final quadtree. It is clear that the output of $CONSTRUCT^+$ on T_3 is the same as what the original $CONSTRUCT$ would produce on input T_4:

$$CONSTRUCT^+(T_3) = CONSTRUCT(T_4).$$

Clearly, S still intersects $\partial R(T_4)$ cleanly and generically.. By Lemma 11, the straight-line graph $G = CONSTRUCT(T_4)$ is isotopic to $f^{-1}(0) \cap R(T)$. This concludes our proof. **Q.E.D.**

3.5 Rectangular Cxy Algorithm

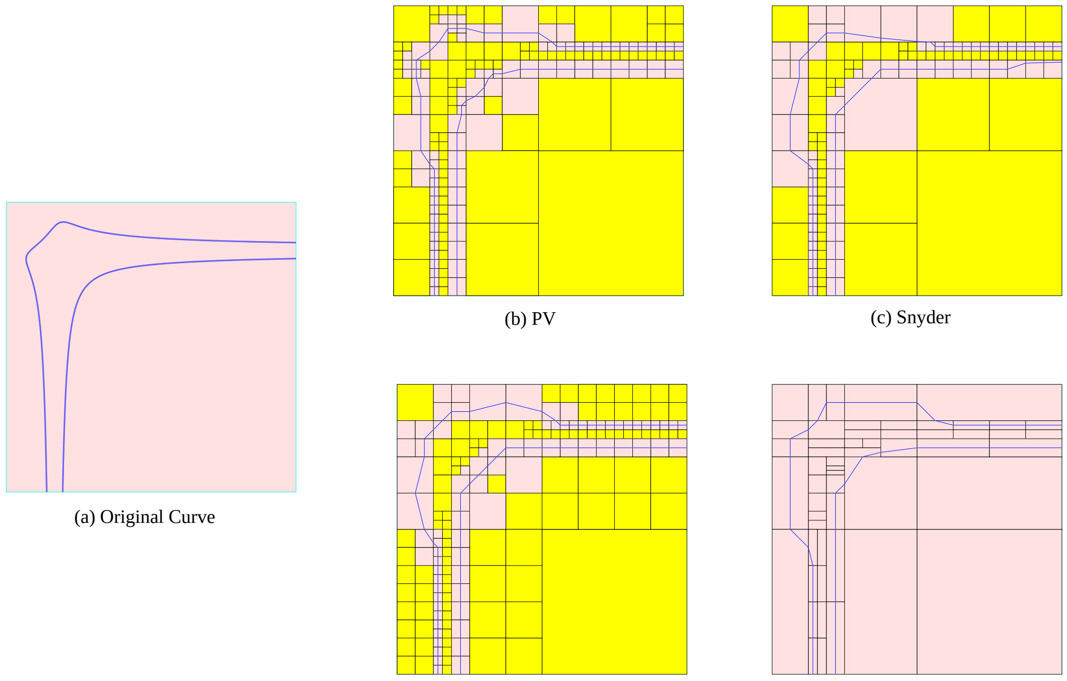

(a) Original Curve
(b) PV
(c) Snyder
(d) Balanced Cxy
(e) Rectangular Cxy

Figure 3.5: Approximation of $f(X,Y) = X^2Y^2 - X + Y - 1 = 0$ inside the box $[(-2,-10),(10,2)]$ using PV, Snyder, Cxy, and Rect.

The recent meshing algorithms [9, 32, 44] all assume full-splits (subdividing a box into four subboxes). We now introduce an Cxy algorithm that can do half-splits. The boxes are no longer squares, and hence the next algorithm is known as the **Rectangular Cxy Algorithm**. This algorithm is even more adaptive than the Balanced Cxy Algorithm, and this can be illustrated with the curve $X^2Y^2 - X + Y = 1$ shown in Figure 3.5. The boxes with yellow color are discarded boxes, and the boxes with pink color are candidate boxes. The curve has preferred directions in the horizontal and vertical directions. Our algorithm can automatically produce rectangles that are elongated along the corresponding directions to adapt to the curve — see Figure 3.5(e). As a result, the number of subdivisions can be drastically reduced as compared to algorithms based on square boxes. The new algorithm differs from balanced Cxy in three major aspects:

First, we need an arbitrary but fixed parameter r called the **aspect ratio bound**. For a box B, let $\alpha(B) := w_y(B)/w_x(B)$. Then its **aspect ratio** is defined as $\rho(B) := \max\left\{\alpha(B), \frac{1}{\alpha(B)}\right\} \geqslant 1$. We require that all boxes in our quadtree satisfy $\rho(B) \leqslant r$. This ensures the termination of our algorithm.

Second, we modify the Subdivision Phase as follows: For each on-box B in the queue, we must decide how to tag it, or how to to split and tag its children. This is accomplished by a new **splitting procedure**, which amounts to checking the following three lists of conditions (in this order):

$$\left.\begin{array}{ll} L_0: & C_0(B), C_{xy}(B) \\ L_{out}: & C_0(B_{12}), C_0(B_{34}), C_0(B_{14}), C_0(B_{23}) \\ L_{in}: & C_{xy}(B_{12}), C_{xy}(B_{34}), C_{xy}(B_{14}), C_{xy}(B_{23}) \end{array}\right\} \tag{3.3}$$

We stop at the first verified condition. If a condition in L_0 is verified, we tag B as an candidate or discarded box, accordingly. If a condition in L_{out} or L_{in} is verified, we do a half-split of B to produce the child that satisfies that condition. That child is tagged as discarded (if an L_{out} condition) or candidate (if an L_{in} condition). The other child is pushed back into the queue. Finally, if no condition is verified, we do a full-split and push the four children into the queue.

Actually, this splitting procedure must be slightly modified in order to respect the aspect ratio bound (this amounts to avoid testing the first half of the conditions in L_{out} and L_{in} if $\alpha(B) < 2/r$, and to avoid testing the second half if $\alpha(B) > r/2$. Note that there is considerable opportunity for sharing, and thus optimization, when implementing the arithmetic operations to check the 10 conditions of (4.8).

Third, we must track the "splitting depth" of a node in the quadtree by a pair of natural numbers, called its x**-depth** and y**-depth**. These count the number of vertical and (respectively) horizontal splits from the root to the given node. A full-split counts as both a vertical as well as a horizontal split. We now say a box B is x**-balanced** if its top and bottom neighbors have x-depth at most 1 away from the x-depth of B; similarly for y**-balanced** with respect to its right and left neighbors. The Balancing Phase is easily modified to only doing half-splits in order to achieve the balance condition for all boxes. One strategy is to first achieve x-balance for all candidate boxes, then to do the same for y-balance. Finally, in the Construction Phase, we modify $CONSTRUCT^{+}(T_3)$ so that ambiguity-based priority queue should distinguish between an y**-ambiguity** (e.g., Figure 3.3(a)-(c')) that must be resolved by a horizontal split, or a x**-ambiguity** that requires a vertical split.

3.6 Ensuring Geometric Accuracy

So far, we have focused on computing the correct isotopy. We now consider the process of **refinement** whose goal is geometric accuracy, i.e., to ensure an approximation G that is ε-close to $S \cap B_0$. The "small normal variation" C_1 predicate is quite strong, so that it is quite easy to use for refinement in the PV algorithm (this is implicit in [32, 31]). To see this explicitly, we claim that it suffices to ensure that for any candidate box B, if it has at least one arc of $G = (V, E)$, then its diameter is $\leqslant \varepsilon/4$. Then any neighbor B' of B has diameter at most $\varepsilon/2$. Thus, each arc e in B is isotopic to a curve component X of $S \cap (B \cup B')$. But the distance between any two points in $B \cup B'$ is $\leqslant \varepsilon\sqrt{(1/2)^2 + (3/4)^2} < \varepsilon$. With our C_{xy} predicate, no such bound on geometric accuracy is possible because our curve could now escape arbitrarily far away from our constructed approximation via undetected excursions. Below, we develop a generalization of the C_1 predicate to capture geometric accuracy bounds for rectangular boxes.

¶16. Extending the Buffer Lemma of Plantinga & Vegter It is noted in Plantinga & Vegter that if B is a square box, and $C_1(B)$ holds, then any "incursion" of the curve S along an edge of B cannot leave B. Thus, B acts as a "buffer" area within which any isotopic variation of the curve S must lie. Their result is still true if B is "almost square", as captured by our next lemma:

LEMMA 12 (Buffer Property). *Let (a, b) be a convergent pair relative to box B. Wlog, assume (a, b) lies on the bottom edge e of B. Let X_a and X_b (resp.) be the connected components of $S \cap B$ with one endpoint at a and b (resp.) If condition $C_1(B)$ holds and $\alpha(B) \geqslant 1/2$, then $X_a = X_b$.*

Proof. Figure 3.6 illustrates our proof. Let H be the upper halfcircle with diameter e. Since $\alpha(B) \geqslant 1/2$, H must lie completely inside the rectangle B. If $X_a \neq X_b$, then the component X_a must leave the interior of the halfcircle H at some first point $a' \in H$; similarly, X_b must leave at some point $b' \in H$. By the mean value theorem, there is

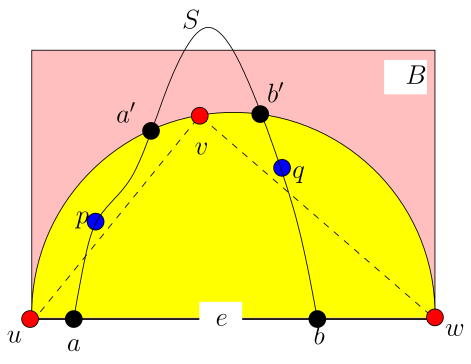


Figure 3.6: Half-circle argument.

a point p (resp., q) on X_a (resp., X_b) whose slope is equal to the slope of the segment $[a, a']$ (resp., $[b, b']$). Let the endpoints of the edge e be u, w and pick any point $v \in H$ between a' and b'. Clearly, the slope at p is more than the slope of $[u, v]$, and the slope at q is more negative than the slope of $[v, w]$. Thus, the angle between the normals at p and q must be greater than the angle between the two normals of the segments $[u, v]$ and $[v, w]$. But the latter angle is exactly 90° (since H is a halfcircle). This contradicts the fact that $C_1(B)$ holds. **Q.E.D.**

We further loose the constraint on B from "almost square" to a rectangle with arbitrary aspect ratio $\alpha(B)$. We also need to do some change on the C_1 predicate.

¶17. Generalized C_1 Predicate We now generalize the C_1 predicate of Plantinga & Vegter so that it guarantees the same buffering effect for *any* rectangle, not just those with aspect ratio $\leqslant 2$.

For any box B, define the linear map

$$T_B : \mathbb{R}^2 \to \mathbb{R}$$

where $T_B(x, y) := (x, y/\alpha(B))$. Note that $B' = T_B(B)$ is a square. Alternatively, the inverse of T_B is $T_B^{-1}(x,y) = (x, \alpha(B)y)$. For any function $f : \mathbb{R}^2 \to \mathbb{R}$, define

$$f^B : \mathbb{R}^2 \to \mathbb{R}$$

where $f^B(p) = f(T_B^{-1}(p))$. It is easy to see that

$$f^B(T_B(p)) = f(T_B^{-1}(T_B(p))) = f(p)$$

and hence $f^B(B') = f(B)$. Let C_1^* denote the "generalized C_1 predicate" which holds at a box B provided

$$C_1^*(B) : 0 \notin (\square f_x^B(B'))^2 + (\square f_y^B(B'))^2.$$

We have the following:

LEMMA 13. *Let (a,b) be an upward convergent pair of a segment e, where e is the bottom edge of a box B. Let X_a and X_b (resp.) be the connected components of $f^{-1}(0) \cap B$ with one endpoint at a and b (resp.) If condition $C_1^*(B)$ holds, then $X_a = X_b$ (i.e., X_a is a B-intrusion).*

Proof. Note that $C_1^*(B)$ means $C_1^g(B')$ holds where $g = f^B$ (see the superscript notation for $C_1^g(B')$ in ¶13). Let $X_{T_B(a)}$ and $X_{T_B(b)}$ be the connected component of $g^{-1}(0) \cap B'$ with one endpoint at $T_B(a)$ and $T_B(b)$ (resp.). From the previous lemma, we know that $X_{T_B(a)} = X_{T_B(b)} = X'$, and X' is completely included inside B'. Since T_B is a bijection that maps B' to B, we can conclude that $X = T_B^{-1}(X') = T_B^{-1}(X_{T_B(a)}) = T_B^{-1}(X_{T_B(b)})$ is completely included inside B, i.e., $X_a = X_b$. **Q.E.D.**

¶18. Refinement based on the Generalized C_1 Predicate We introduce the concept of safety of segments. Intuitively, a segment s is safe if there can be no incursion or excursion along s.

Let T_3 be a quadtree from the Subdivision Phase of our Rectangular Cxy Algorithm. For each (rectangular) box B in T_3, we will classify some of its edges as **safe relative to** B:

- If $C_0(B)$ holds, then each of its edges is safe relative to B.
- If $C_x(B)$ holds, then its top and bottom edges are safe relative to B. Similarly, $C_y(B)$ holds implies its right and left edges are safe.

More generally, a segment s is **safe** (not relative to any box) if there exists s' such that $s \subseteq s'$ and s' is safe relative to some box B'. It is easy to see that we can effectively know whether a segment s is safe from the information derived in constructing the tree T_3. In particular, when we determine that a box satisfies C_{xy}, we actually know whether it satisfies C_x or C_y (or even both).

The safety of some (but not all) segments can be deduced by looking at the presence of vertices along the edges of a box. For instance, in Figure 3.4(a–f), we have indicated by thickening those edges that we know to be safe because of the presence of vertices. Note that we do not have any thick edges for Case (a) even though we know at least two of them must be safe. We could, but need not, exploit such extended notions of safety.

¶19. Exploiting Safe Segments for Refinement

LEMMA 14. *Let s be a safe segment.*

(i) Then the curve $S = f^{-1}(0)$ intersects s at most once, i.e., $|S \cap s| \leqslant 1$.

(ii) $|S \cap s| = 1$ iff f have different signs at the endpoints of s.

Proof. (i) If s is safe, then $s \subseteq s'$ where s' is safe relative to some box B'. If $C_0(B')$ holds, then clearly $|S \cap s| = 0$. If $C_{xy}(B')$ holds such that S is parametrizable in the direction perpendicular to s, then clearly $|S \cap s| \leqslant 1$.
(ii) If f has different signs at the endpoints of e, then $|S \cap e|$ is odd. By part (i), $|S \cap e| = 1$. Conversely, if f have the same sign at the endpoints of e, then $|S \cap e|$ is even. By part (i), $|S \cap e| = 0$. **Q.E.D.**

Let s be a segment. We say that s is **soft** if it is not safe. Suppose B is a terminal box (i.e., satisfies C_{xy} but not C_0) with at least one soft edge. Then the distance from this soft edge to the opposite edge is called the **soft distance** of B. Note that this soft distance is uniquely defined. If B has no soft edge, then the soft distance is 0 by definition. If the soft distance is d, then any incursion into B can be removed by modifying the curve within a Hausdorff distance of d.

There are three kinds of curve component $C = B \cap S$ in box B as illustrated in Figure 2.2: incursion, cut or corner components. We consider bounds on the dimension of B in order that our straightline approximations to C is within Hausdorff distance $\varepsilon/2$ from C.

(a) Suppose C is an incursion, i.e., both endpoints of C lie on one edge of B. If B has soft distance at most $\varepsilon/2$, then as noted, C can be removed by perturbing the curve by a Hausdorff distance of $\varepsilon/2$.

(b) Suppose C is a cut component, i.e., the endpoints of C lie on opposite edges of B. If s is a edge of B containing an endpoint of C, then we want the length of s to be at most ε. This ensures that our linear approximation is within Hausdorff distance $\varepsilon/2$ from an actual curve component within B.

(c) Suppose C is a corner component, i.e., the endpoints of C lie on adjacent edges of

B. In this case, we want each edge of B to have length at most $\sqrt{2}\varepsilon/3$. Again it ensures that our straightline approximation is within Hausdorff distance $\varepsilon/2$ from an actual curve component within B.

We now sketch how to incorporate ε-refinement into the Rectangular Cxy Algorithm. The idea is to ensure that each terminal box has dimensions bounded as in (a)-(c) above. It is easiest to assume that the original subdivision phase has been carried out (so all boxes are known to satisfy C_0 or C_{xy}). We make another pass through the list of candidate boxes.Recall that such a box B is **monochromatic** if the function f has uniform signs (either all positive or all negative) at the corners on the boundary of B; otherwise it is **bichromatic**. Note that the approximate curve G passes through B iff B is bichromatic. We keep B if the following conditions (a')-(c') hold:

(a') If B is passive and has at least one soft edge, then we check that the generalized predicate $C_1^*(B)$ hold. Under this condition, any undetected entry of the curve into B must represent an incursion. We require the soft distance of B to be at most $\varepsilon/2$.

(b') If B is bichromatic and has sign changes on two opposite edges, then we require the lengths of these edges to be at most $\varepsilon/2$.

(c') If B is bichromatic and has sign changes on two adjacent edges, then we require the lengths of all edges to be at most $\sqrt{2}\varepsilon/3$.

If any of the above conditions fail, we split B and put any child that fails the C_0 predicate back into the queue. This completes our description of the modified subdivision phase. Other phases are unchanged. The correctness follows easily from our discussion.

The above refinement method can also be adapted for the Balanced Cxy algorithm. If we only have square boxes, it amounts to ensuring that each passive box B with at least one soft edge also has width at most $\varepsilon/2$ and satisfies C_1, and each bichromatic box B has width at most $\sqrt{2}\varepsilon/3$.

3.7 Summary of Experimental Results

We report on our experimental results. Our code is developed in `Java` on the Eclipse Platform (SDK Version 3.3.0). The hardware is Dell Laptop Inspiron 6400, with 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)). Note that this implementation is based on machine arithmetic. But since all arithmetic operations use only ring operations and divide by 2, there are no round-off errors except for under/overflows. Our examples below do not reach such limits (except for examples in part (7) of this section, where we use `Core Library` based implementation to avoid under/overflows for high degree curves). The code has been translated into `C++` for distribution with our open source `Core Library`. We implemented five algorithms: PV, Snyder, Balanced Cxy, Balanced Cxy with epsilon precision, and Rectangular Cxy. For Snyder's algorithm, the boundary root isolation is carried out using the $1D$ analogue, namely the EVAL algorithm (see [11, 27, 10]). For brevity, the Balanced Cxy Algorithm and Rectangular Cxy Algorithm will be known as **Cxy** and **Rect**, respectively.

We have not yet implemented two concepts discussed in this thesis: Boundary processing for arbitrary input geometry $R(T_0)$ (Section 3) and exploiting safe segments for geometric accuracy (Section 8). As stated in our introduction, most of our experiments are concerned computing the correct isotopy, ignoring geometric accuracy. But we could easily and cheaply improve geometric accuracy in our approximation graphs by using interpolation: instead of choosing vertices at midpoints of segments, we choose some linearly interpolated point.

We now summarize our main conclusions, based on compare four algorithms: Cxy, Rect, PV and Snyder. We also briefly compare to EXACUS from the Max-Planck Institute of Computer Science.

(1) *Cxy can be significantly faster than PV and Snyder.* Figure 3.7 is gotten by running these algorithms on the curve $f(X, Y) = X^2(1-X)(1+X)-Y^2+0.01 = 0$ inside box $[(-1.5, -1.5), (1.5, 1.5)]$. This example is from [32]. Cxy is twice as fast as PV and Snyder, and Rect is the fastest: the PV produces 196 boxes in 31 milliseconds; Snyder produces 112 boxes in 37 milliseconds; Cxy produces 112 boxes in 16 milliseconds; and Rect produces 76 boxes in 15 milliseconds.

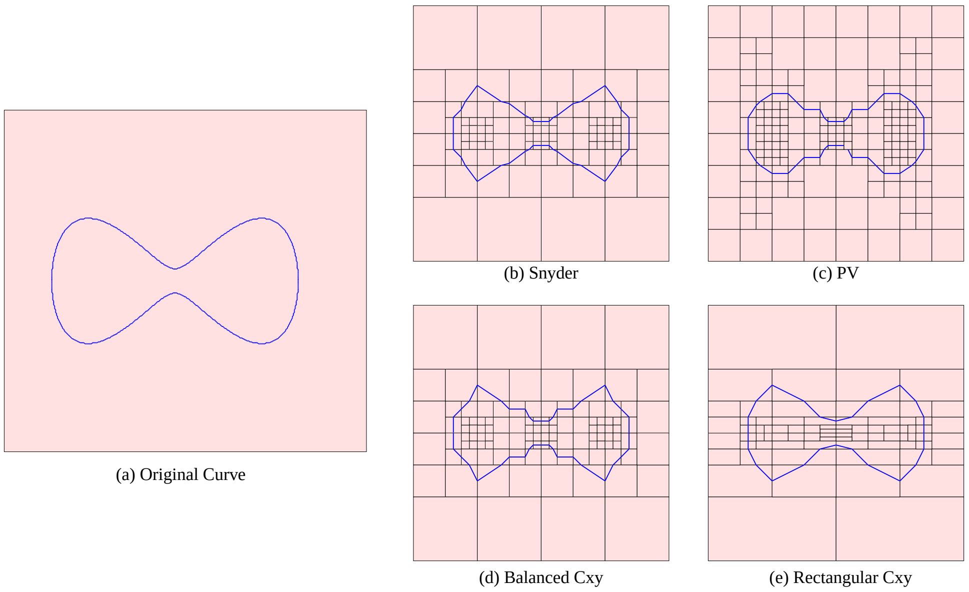


Figure 3.7: Domain Subdivision Approaches to approximating the curve $f(X, Y) = X^2(1 - X)(1 + X) - Y^2 + 0.01 = 0$: comparison of four algorithms.

(2) *When we add refinement, the improvement is minimal.* We currently use a simplistic approach based on the C_1 predicate. We believe this part can be sped up, for

Table 3.1: Rect $>$ Cxy $>$ PV

#Boxes/Time(ms)	$s = 15$	$s = 60$	$s = 100$
PV	5686/157	OME	OME
Cxy	2878/125	45790/2750	OME
Rect	288/31	4470/609	13042/4266

Table 3.2: Rect can exploit larger aspect ratio

#Boxes/Time(ms)	$s = 15$	$s = 60$	$s = 100$
$r = 10$	150/16	2242/265	6540/1109
$r = 20$	82/15	1134/109	3282/406
$r = 40$	48/15	574/62	1656/172
$r = 80$	32/0	296/32	842/78

example, by implementing the method from Section 8. The refined curve, with precision $\epsilon = 0.005$, is shown in Figure 3.7(a). PV produces 8509 boxes in 219 ms, while Cxy produces 8497 boxes in 204 ms.

(3) *Rect can be significantly faster than Cxy.* E.g., Let the aspect ratio bound be $r = 5$. Running the algorithms on the curve $f(X, Y) = X(XY - 1) = 0$ in the box $B_s := [(-s, -s), (s, s)]$ (Figure 3.8(b), (c), (d) and (e) show the cases when $s = 4$. Snyder will not terminate when the curve intersects the edges of the boxes tangentially, so we get Figure 3.8(c) by shifting the initial box a little bit). We get Table 3.1 (OME=OutOfMemoryError):

(4) *Increasing the aspect ratio bounds can speed up the performance of Rect.* Using the same curve and box as (3), we now look at the performance of Rectangular Cxy with variable aspect ratio bounds of $r = 10, 20, 40, 80$. Figure 3.9 shows the case when $r = 15$. Table 3.2 shows a proportional speedup (time$= 0$ means time< 1 ms):

(5) *Sometimes Snyder is faster than Balanced Cxy.* We now show an example in

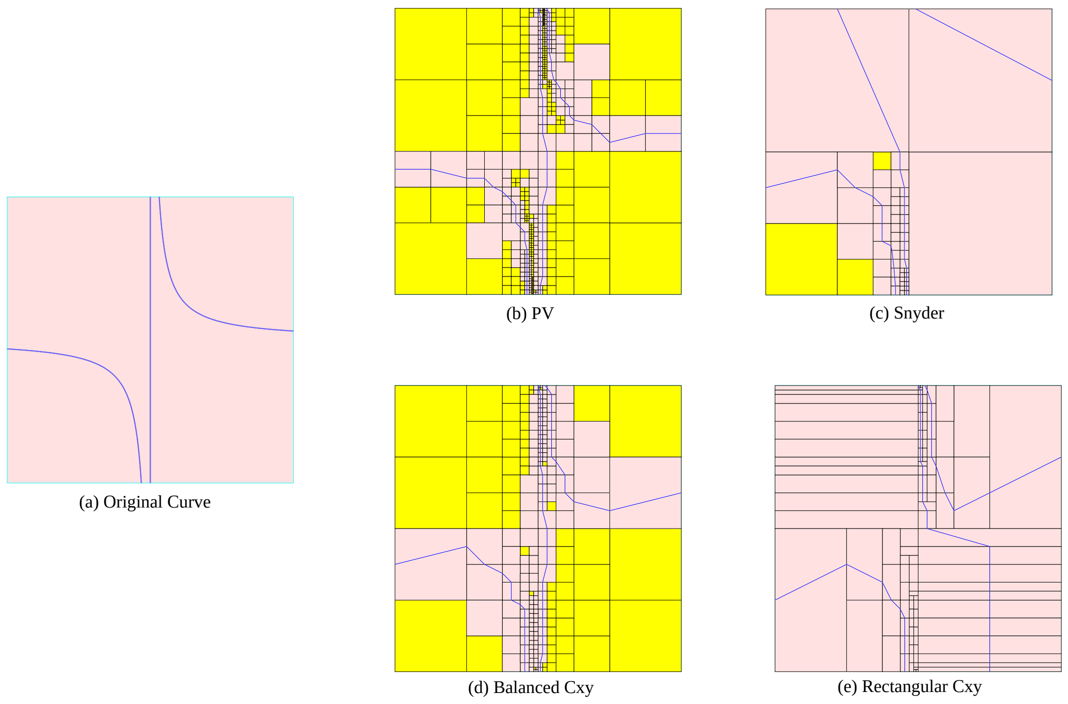


Figure 3.8: Approximation of $f(X,Y) = X(XY - 1) = 0$ inside the box $[(-4,-4),(4,4)]$. Figs. (b),(d),(e) is from PV, Cxy, and Rect. Fig. (c) is from Snyder (inside the box $[(-3.9,-3.9),(4.1,4.1)]$).

which Cxy is slower than Snyder; in turn, Snyder is slower than Rect. When we want to ensure geometric closeness, it is clear that our new approach is considerably faster because Snyder is not forced to subdivide the terminal boxes until their diameters are $\leqslant \varepsilon$. In Table 3.3, we compare PV, Cxy, Rect (with maximum aspect ratio $r = 257$) and Snyder on the curve $f(X,Y) = X^2 + aY^2 - 1 = 0$ in the box $[(-1.4,-1.4),(1.5,1.5)]$ where $a = 10^n$ for $n = 4,\ldots,7$ (Figure 3.10 shows the cases when $n = 2$).

The curve here is a thin and long oval. so the size of the smallest box would be very small. Both Cxy and PV need to do balancing and produce more boxes than Snyder, so they are more time consuming (note that Cxy is significantly (> 50 times) faster than PV when $n = 7$). Rect produces even fewer boxes than Snyder, and Snyder needs to do root isolation; so it is not surprising that Rect is much faster than Snyder.

Table 3.3: Rect $>$ Snyder $>$ Cxy $>$ PV

#Boxes/Time(ms)	n=4	n=5	n=6	n=7
PV	1825/62	6415/234	20806/1219	65926/9219
Snyder	25/16	31/16	34/31	40/31
Cxy	175/15	769/218	694/172	754/172
Rect	17/0	14/0	25/0	29/0

Table 3.4: Rect $>$ Cxy $>$ Snyder $>$ PV

#Boxes/Time(ms)	$n = -1$	$n = 0$	$n = 1$
PV	73/0	4417/516	OME
Snyder	10/15	1306/125	OME
Cxy	13/0	1510/62	OME
Rect	6/0	13/0	255/31

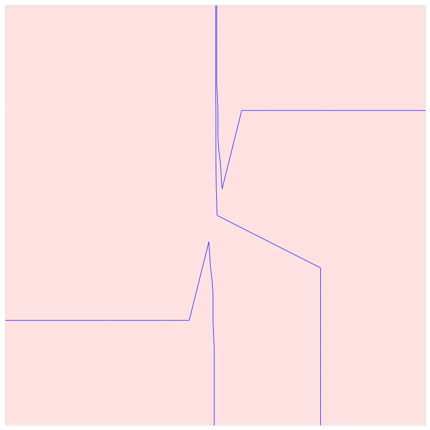
(6) *In general, Cxy and Rect have better performance than Snyder.* We ran Snyder on the curve $f(X, Y) = X(XY - 1) = 0$. Since Snyder will not terminate when the curve intersects the edges of the boxes tangentially, we cannot run this example on the box $B_s := [(-s, -s), (s, s)]$. Instead, we chose the initial box to be $B_n := [(-14 \times 10^n, -14 \times 10^n), (-15 \times 10^n, -15 \times 10^n)]$, where $n = (-1, 0, 1)$. Figure 3.8 (c) shows the case when $B_0 := [(-3.9, -3.9), (4.1, 4.1)]$. We also tested PV, Cxy, and Rect (with maximum aspect ratio $r = 257$) in these examples. The results are shown in Table 3.4.

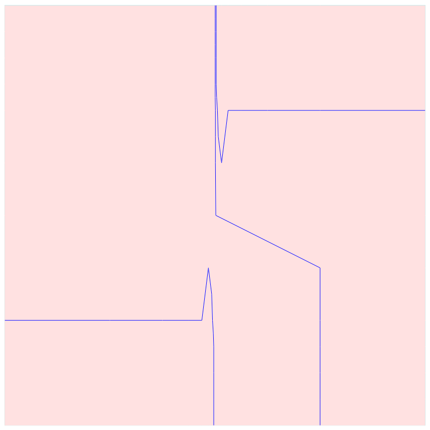
(7) *Cxy can work with high degree curves and sometimes improve on EXACUS.* The EXACUS system has a nice web interface accessible from `http://exacus.mpi-inf.mpg.de/cgi-bin/xalci.cgi`. EXACUS is based on strong algebraic methods and can handle singularities. The following examples show that our algorithm could be much faster than EXACUS. In order to avoid under/overflows, we use the `C++` code in the `Core Library` which supports exact geo-

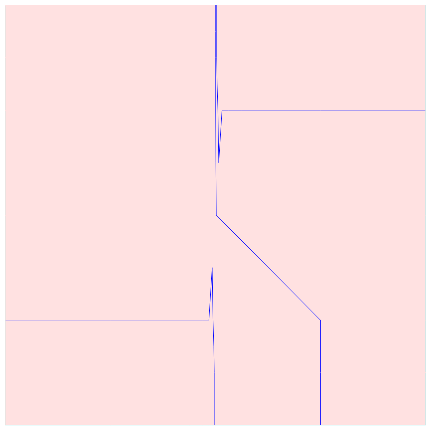
metric computation. The hardware is Apple MacBook Pro, with Intel Core2 Duo CPU 2.40 Ghz and 4.0Gb of RAM.

- Approximating the curve $f(X,Y) = X^{100} + Y^{100} - 1 = 0$ in the box $B_0 := [(-2,-2),(2,2)]$: Cxy takes 701 milliseconds while EXACUS is timed out.
- Approximating the curve: $f(X,Y) = (X^2+Y^2)^k - 4X^2Y^2 - 0.01 = 0$ inside the box $B_0 := [(-1,-1),(1,1)]$. EXACUS is timed out when $k \geq 7$. Cxy takes 1.589 seconds when $k = 7$; 2.312 seconds when $k = 8$; 2.334 seconds when $k = 9$; and 3.439 minutes when $k = 10$.

(8) *Two more examples.* We had already seen Figure 3.5 for the curve $f(X,Y) = X^2Y^2 - X + Y - 1 = 0$ inside the box $[(-2,-10),(10,2)]$. PV produces 211 boxes in 16 milliseconds, Snyder produces 139 boxes in 31 milliseconds, Cxy produces 181 boxes in 15 milliseconds, and Rect produces 54 boxes in < 1 millisecond. Another example in Figure 3.11 shows the approximation of $f(X,Y) = Y^2 - X^2 + X^3 + 0.02 = 0$ inside the box $[(-1.5,-1.5),(1.5,1.5)]$. PV produces 154 boxes in 15 milliseconds, Snyder produces 106 boxes in 31 milliseconds, Cxy produces 106 boxes in 15 milliseconds, and Rect produces 74 boxes in 15 milliseconds.


(a) r=10


(b) r=20


(c) r=40

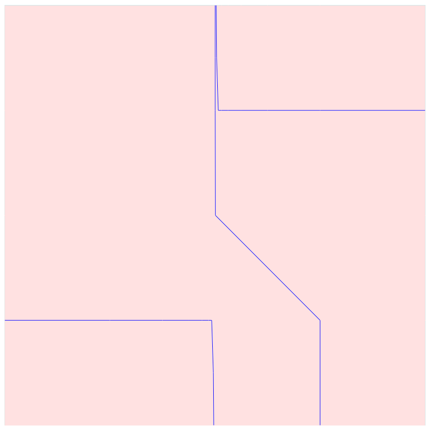
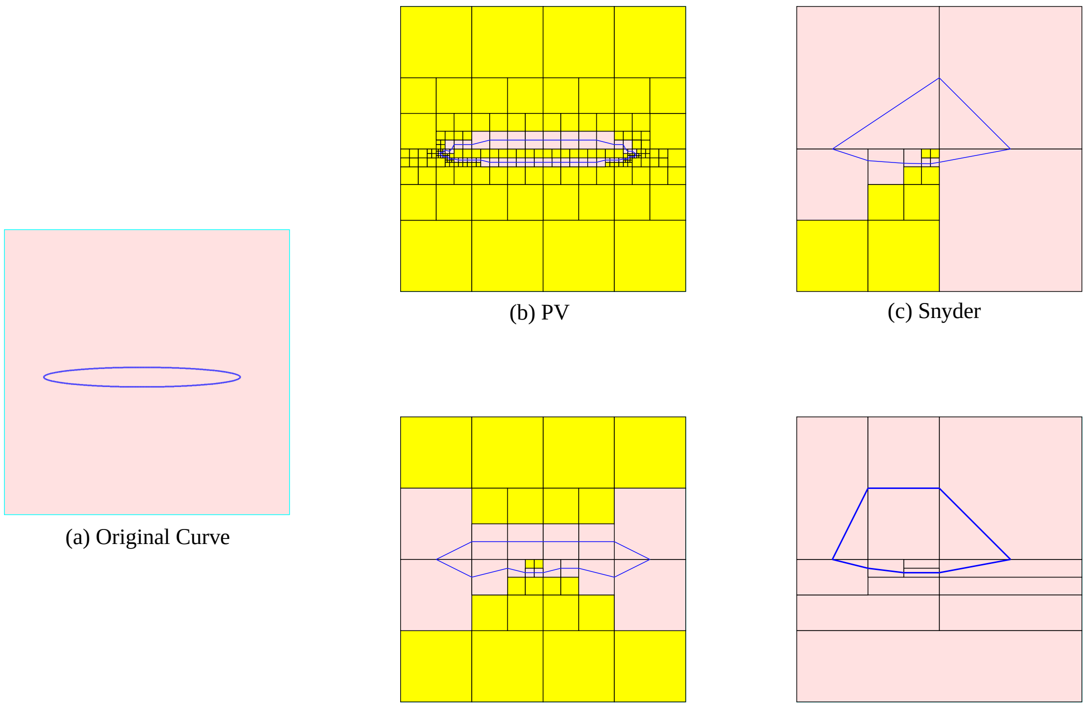

(d) r=80

Figure 3.9: Approximation of $f(X, Y) = X(XY - 1) = 0$ inside the box $[(-15, -15), (15, 15)]$ using Rect with maximum aspect ratios of $10, 20, 40$, and 80


(a) Original Curve

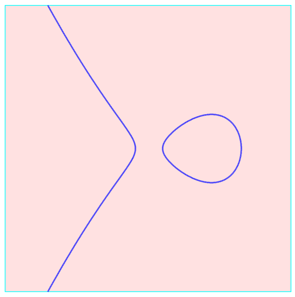
(b) PV

(c) Snyder

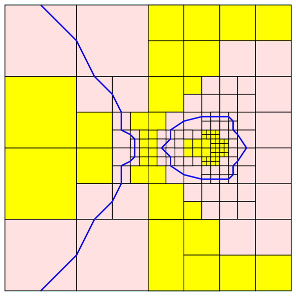
(d) Balanced Cxy

(e) Rectangular Cxy

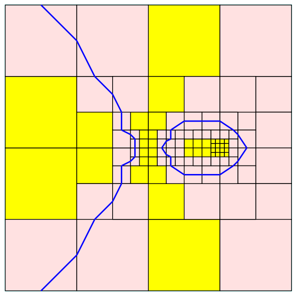
Figure 3.10: Approximation of $f(X, Y) = X^2 + 100Y^2 - 1 = 0$ in the box $[(-1.4, -1.4), (1.5, 1.5)]$ using PV, Snyder, Cxy, and Rect.



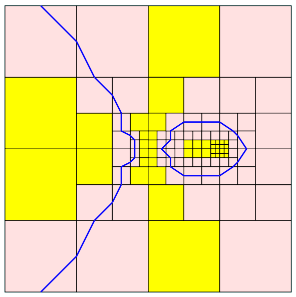
(a) Original Curve



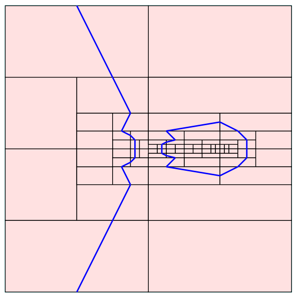
(b) PV



(c) Snyder



(d) Balanced Cxy



(e) Rectangular Cxy

Figure 3.11: Approximation of $f(X,Y) = Y^2 - X^2 + X^3 + 0.02 = 0$ inside the box $[(-1.5,-1.5),(1.5,1.5)]$ using PV, Snyder, Cxy, and Rect.

Chapter 4

Isotopic Meshing of Surfaces

In this chapter, we will extend our $2D$ meshing algorithm to $3D$ (i.e., meshing of surfaces). This extension is by no means routine, as the correctness arguments and case analysis are more subtle. Also, a new phenomenon arise in which local rules for constructing surfaces are no longer sufficient. We will describe three subdivision algorithms for surfaces. They will be known as the Regularized Cxyz, Balanced Cxyz and Rectangular Cxyz Algorithms.

For our $3D$ meshing algorithm, we inherit the terminology and notations used in the $2D$ algorithm, with some generalization and extension. In our discussions, we fix a real surface

$$S := f^{-1}(0) = \left\{p \in \mathbb{R}^3 : f(p) = 0\right\}, \tag{4.1}$$

which is specified by a PV function, $f(X, Y, Z) : \mathbb{R}^3 \to \mathbb{R}$.

By $\square\mathbb{F}$ we mean the set of all closed intervals with endpoints in $\mathbb{F}$. A $3D$ **box** is given by $B = I_x \times I_y \times I_z \subseteq \mathbb{R}^3$ where $I_i \in \square\mathbb{F}$ ($i \in \{x, y, z\}$). For a box B, its midpoint is $m(B) = (m(I_x), m(I_y), m(I_z))$, and it will have three widths, called i-**widths**: $w_i(B) = w(I_i)$ for $i \in \{x, y, z\}$. The width and diameter of B are (resp.)

$w(B) := \min\{w_x(B), w_y(B), w_z(B)\}$ and $d(B) := \max\{w_x(B), w_y(B), w_z(B)\}$. The 0-, 1- and 2-dimensional features of a box are called its **corners**, **edges**, and **faces**. So there are 8 corners, 12 edges and 6 faces in all. We call the faces that are perpendicular to the i-direction ($i \in \{x, y, z\}$) the **i-faces**. Thus there are two i-faces for each i. We will name these faces as follows: The x-face with the smaller x-coordinate is called the **left face**; the other is called the **right face**. Likewise, y-face with the smaller y-coordinate is called the **bottom face**, and the other is the **top face**. The z-face with the smaller z-coordinate is the **front face** and the other is the **back face**. Figure 4.1 illustrates this terminology.

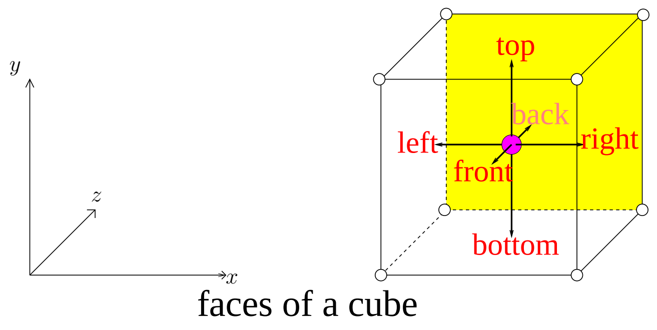


Figure 4.1: Convention and terminology for the faces of a box.

By making an nice perturbation of f, we may assume that every corner c has only positive or negative sign. A $3D$ box is **monochromatic** if the sign at all of its 8 corners are the same. Otherwise, it is **bichromatic**. A **full-split** of B is the operation of subdividing B into eight equal sub-boxes; a **quarter-split** subdivides B into four equal sub-boxes; and a **half-split** subdivides B into two equal sub-boxes. There are three kinds of quarter-splits: x-y **split** (split B by two planes which are perpendicular to x and y directions), y-z **split** and z-x **split**; and three kinds of half-splits: x **split** (split B by a planes which perpendicular to x direction), y **split**, and z **split**. We use the corner/edge/face terminology for boxes, but reserve the vertex/arc/triangle terminology for the triangulated surface G that we shortly introduce to approximate the surface S.

By an **octree** we mean a rooted tree T where each node u is associated with a box B_u, and each non-leaf u has $2, 4$ or 8 children. Moreover, these children are associated with the set of boxes arising from a half-, quarter- or full-split of B_u. Similar to our $2D$ Algorithm, Each leaf of our octrees is labeled as "on" or "off". The union of all these on-boxes is denoted $R(T)$: the **nice region represented by** T, or the **region of interest** (or ROI).

Given an octree T, we can **extend** it by taking any on-box and performing a half-, quarter- or full-split. The newly created children of T will remain on-boxes, thus the ROI is preserved by such extensions. Our algorithms amount to repeated extensions of T.

Similar to the Cxy Algorithm, we also need to discuss the boundary $\partial R(T)$ of $R(T)$. A box B in T is called a **boundary box** if some face of B is contained in $\partial R(T)$; such faces are called **boundary faces**. To avoid extensive discussion of how to process the boundary of the ROI to ensure correctness of our algorithms (such as in [9]), we will make two strong requirements about how S intersect the boundary of $R(T)$: (1) S **intersects the boundary of** $R(T)$ **generically**, which means:

- For each boundary face F, the surface S intersects F transversally, and does not pass through any corner of F.

- The set $S \cap F$ is a finite collection of a finite set of closed loops and/or open curves. By an open curve, we mean one that has two distinct endpoints. The loops lie in the interior of F, and the open curves terminate transversally on the edges of F.

(2) $S \cap R(T)$ is compact, and any end point of $S \cap R(T)$ lies on the boundary $\partial R(T)$. The correctness statement of our algorithm will depend on this assumption. From now

on, we assume the above two requirements always hold.

¶20. Generic Subdivision Algorithm We review a generic framework of subdivision algorithms for computing an isotopic approximation to a given surface $S = f^{-1}(0)$. The following is taken from our Cxy Algorithm using the octree notation:

```
GENERIC SUBDIVISION ALGORITHM
Input:   Surface $S = f^{-1}(0)$, a nice region represented by an octree $T_{in}$, and $\varepsilon > 0$
Output:  Triangulated Surface $G = (V, E, T)$ representing an isotopic $\varepsilon$-approximation
of $S \cap R(T)$
    Phase 1. $T_{out} \leftarrow SUBDIVIDE(T_{in})$
    Phase 2. $T_{ref} \leftarrow REFINE(T_{out})$
    Phase 3. $G \leftarrow CONSTRUCT(T_{ref})$
```

Let us briefly review the subdivision phase: the idea is to keep subdividing boxes until they satisfy certain predicates. Similar to the $2D$ algorithms, we need two box predicates, an **exclusion predicate** $C_{out}(B)$ and an **inclusion predicate** $C_{in}(B)$. If a box satisfies C_{out}, it is discarded, and if it satisfies C_{in}, it is put into the output queue. Otherwise, it is split and its children are placed back into the input queue. The C_{out} predicate is universal:

$$C_0(B) : 0 \notin \square f(B) \tag{4.2}$$

Snyder's inclusion predicate is given by

$$C_{xyz}(B) : C_x(B) \vee C_y(B) \vee C_z(B) \tag{4.3}$$

Note that if $C_i(B)$ holds then f would be i-monotone in B (but the converse need not hold). A surface S is said to be **parametrizable in the x and y directions** (or xy-

parametrizable for short) within a box B if for each pair (x, y), the equation $f(x, y, z) = 0$ has at most one solution z in the box B. Clearly, if f is z-monotone in B, then the surface S is xy-parametrizable in B. We also say B is **monotone** in the z-direction. Similar definition holds for the $2D$ faces of B (i.e., the four faces parallel to the z axis is said to be **monotone** in the z-direction). The Plantinga & Vegter (or PV) algorithm uses $C_1(B)$ as the inclusion predicate:

$$C_1(B) : 0 \notin (\square f_x(B))^2 + (\square f_y(B))^2 + (\square f_z(B))^2. \tag{4.4}$$

4.1 Regularized Cxyz Algorithm

Our basic goal is to replace the C_1 predicate in the PV Algorithm by the parametrizability condition of Snyder. As in Cxy Algorithm, we first consider a simplified version in which we reduce all adjacent candidate boxes to the same depth. We start with an octree T (representing a region $R(T)$) and a non-singular surface $S = f^{-1}(0)$, where $f : \mathbb{R}^3 \to \mathbb{R}$. We full-split the inconclusive boxes until for each leaf box B we have $C_0(B)$ or $C_{xyz}(B)$. Here is the summary of our *Regularized Cxyz Algorithm*.

```
Regularized Cxyz Algorithm:
Input:    Octree $T_0$ and surface $S = f^{-1}(0)$
Output:   Isotopic approximation $G$ for $S \cap R(T_0)$
1.   $T_1 \leftarrow SUBDIVIDE_{C_{xyz}}(T_0)$
2.   $T_2 \leftarrow REGULARIZE(T_1)$
3.   $G \leftarrow CONSTRUCT(T_2)$
```

This algorithm follows our generic subdivision framework. In the subdivision phase, we keep subdividing a box until it satisfies $C_{out} = C_0$ or $C_{in} = C_{xyz}$. Recall that candidate boxes are those who do not satisfy C_{out} but satisfies C_{in} in the hereditary sense. For a boundary box B to be candidate, we further require that its boundary faces satisfy the corresponding $2D$ predicate: more precisely, if F is a boundary i-face, then it must satisfy C_{jk} where $\{i, j, k\} = \{x, y, z\}$. So at the end of the subdivision phase, every on-box is either discarded or candidate. In the regularize phase, we subdivide any candidate box that shares a face with a candidate box of smaller width. The children of the subdivision will satisfy (by hereditary) the C_{xyz} predicate, but we must test if they are C_0. This algorithm is analogous to the Regularized Cxy Algorithm (in $2D$) and the Regularized Plantinga & Vegter Algorithm (in $3D$). We next describe the construction phase.

¶21. Sign Types on Box Corners There are 14 cases of the signs of f at the corners of a box B (Figure 4.2, up to rotation, mirroring and change of sign). This list is taken from Plantinga & Vegter's paper [32], but we will use a canonical typing scheme: **Sign type** nx (where $n = 0, 1, 2, 3, 4$ and $x = a, b, c, etc$) refers to the sign configuration with exactly n positive corners, and x is some additional identifier (if necessary) to uniquely identify the configuration.

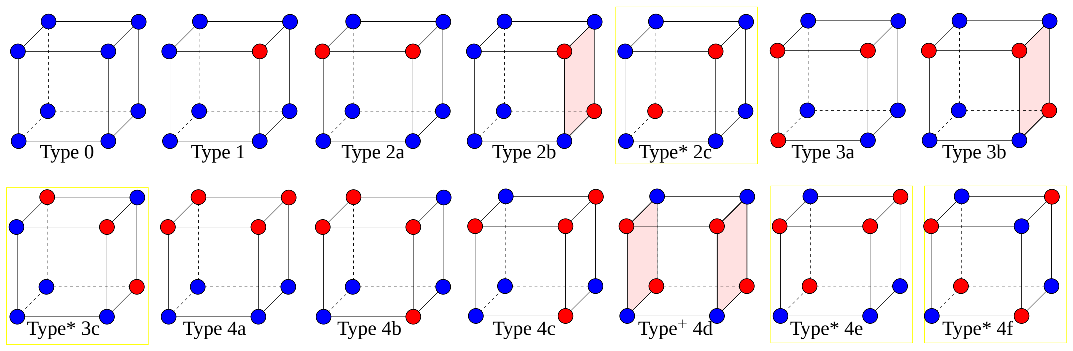


Figure 4.2: 14 Sign Types of f at the corners of a box.

Of the 14 cases in Figure 4.2, only 9 cases can arise under the C_1 predicate. The excluded 5 cases are indicated by a superscript of asterisk or plus: Types $^*2c$, $^*3c$, ^+4d, $^*4e$ and $^*4f$. It is easy to check that the C_{xyz} predicate excludes four of these five cases. The exception is Type^+4d. We use a plus superscript instead of asterisk to indicate this. To summarize, there are a total of 10 sign possibilities under the C_{xyz} predicate – as shown in Figure 4.3.

¶22. Arc Types on Box Faces From the signs types at box corners, we can introduce **vertices** in the middle of those edges whose two end points have opposite signs. Each face of a box can have 0, 2 or 4 vertices. Within the face, we now connect these vertices by line segments[1] which we call **arcs**. Note that these vertices and arcs form the graph $G(V, E)$, where V is the set of vertices and E is the set of arcs (we do not call them "edges" because that is reserved for our box terminology). The arcs are uniquely determined except in the case of 4 vertices. We call a face with 4 vertices an **alternating face** (colored pink) since adjacent corners of such a face must alternate in signs. On an alternating face, we have two distinct ways to introduce a pair of arcs. In $2D$, alternating faces are excluded by the C_{xy} predicate. In $3D$, alternating faces can arise even when a box satisfies the C_{xyz} predicate (e.g., see the right face of Figure 4.3(2b)).

The two possible arc types on alternating faces represent a choice (or ambiguity). This phenomenon was first observed in the Plantinga & Vegter paper. But in the presence of the stronger C_1 predicate, they proved that every choice leads to a correct global surface. For our weaker C_{xyz} predicate, the ambiguity can become an issue – making the wrong choice of arc types can lead to the wrong surface.

By introducing arcs to connect pairs of vertices on each face, we determine the **arc**

[1] Calling them "arcs" is appropriate because in the general case, we may need to introduce non-straight curves – this will arise when we discuss the balanced algorithm.

type for each box. For instance, the Type 2b in Figure 4.2 gives rise to two arc types which we denote as Type 2b(i) and Type 2b(ii) in Figure 4.3. In all, the 10 possible C_{xyz} sign types give rise to 13 arc types as seen in Figure 4.3.

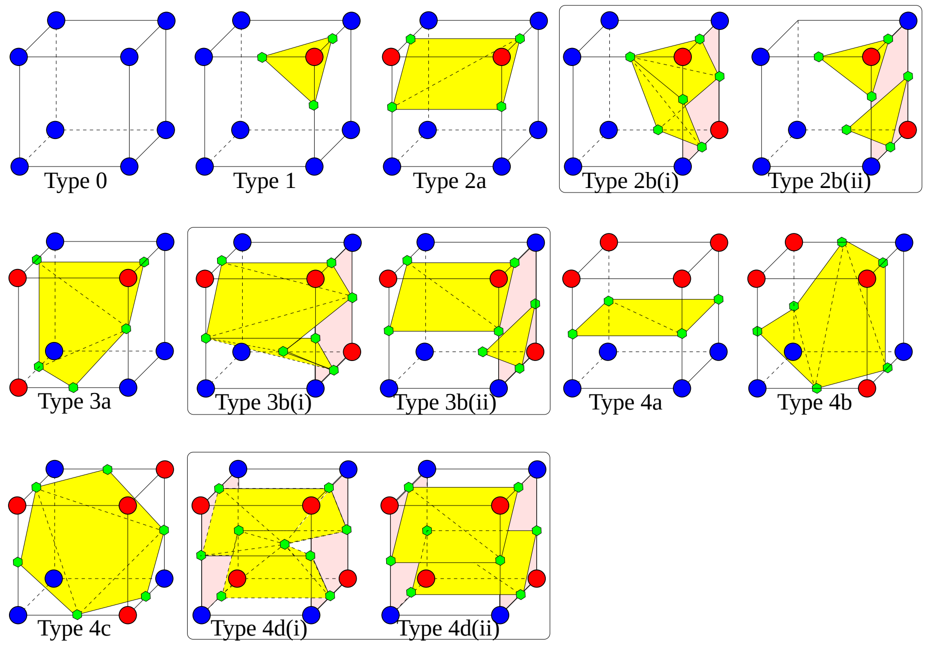


Figure 4.3: The 10 possible C_{xyz} sign types give rise to 13 C_{xyz} arc (hence surface) types.

¶23. Surface Types in Box Interiors After connecting vertices with arcs, we need to construct a triangulated surface in the interior of each box so that the boundary of the surface agrees with the arc type on the faces. Fortunately, this presents no further choices, so the 13 arc types gives rise to 13 **surface types** (colored yellow) as enumerated in Figure 4.3.

A remark about our labeling for these surface types: it refines the typing scheme from Figure 4.2 by adding (if necessary) subtype indications of "(i)" or "(ii)". Moreover, we can always use subtype "(i)" to indicate that the surface in the box has one connected component, and “(ii)” to indicate two connected components.

¶24. Global Analysis of Construction Rules By “construction rules” we refer to the totality of all rules for vertex insertion, arc connection and surface construction. Naively, we can apply these rules to each box without consideration of how the rules are applied to the other boxes. This naive rule turns out to be sufficient in the $2D$ Cxy Algorithm and also the $3D$ Algorithm of Plantinga & Vegter. We now show that in our Regularized Cxyz Algorithm, this is not enough: the counter example is given by Figure 4.4.

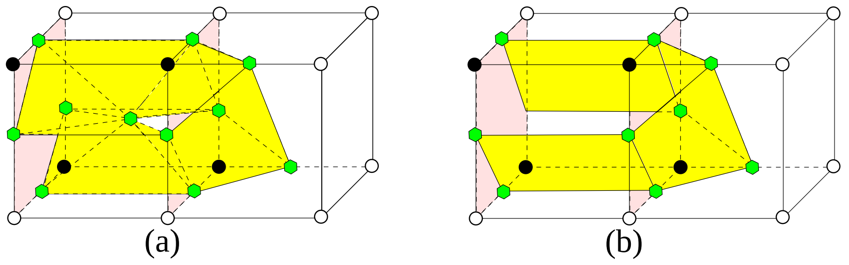


Figure 4.4: Wrong choice of arc types can lead to an impossible connection.

In Figure 4.4(a), the two boxes satisfy C_x, but the triangulated surface determined by the indicated arc connections will violate the C_x condition. On the other hand, the triangulated surface determined by the arc connections Figure 4.4(b) satisfies C_x, and Figure 4.4(a) is topologically different from Figure 4.4(b). In this example, we might be able to locally ensure that these two boxes are connected in a local consistency manner. But the next example in Figure 4.5 shows that local consistency (i.e., consistency between adjacent pairs of boxes) is not enough because of the phenomenon of “blocks”. In Figure 4.5, we have a block of three boxes in which the triangulated surfaces in the first and second boxes are consistent, and the surfaces in the second and third boxes are also consistent. But the surface in the union of three boxes does not have the correct topology because it does not respect the x-monotonicity of f.

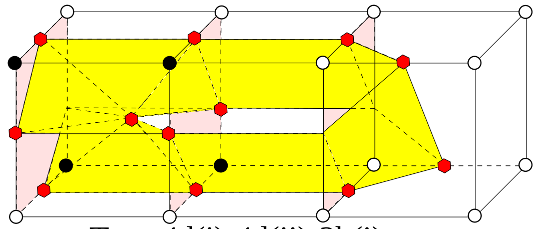


Figure 4.5: Local consistency does not imply the global consistency.

We now introduce the notion of blocks. Two boxes are **alternating neighbors** if they share an alternating face. Note that in a box B, if an x-face is alternating, then no y- or z-face can be alternating because of C_{xyz}. A maximal set of boxes that are connected by this alternative neighbor relation is called an **alternating block**. In particular, if a box has no alternating face, it forms its own alternating block. We call it a **trivial alternating block** (otherwise, **nontrivial alternating block**). If all the alternating faces of boxes in an alternating block are normal to the i-direction ($i = x, y, z$), then we call it an i**-block**. Note that f is i-monotone in the i-block. We say the i-block is **monotone** in i-direction. Let $\mathcal{B}$ be an alternating block, we define the boundary of $\mathcal{B}$: $\partial(\cup\mathcal{B}) = \partial(\cup_{B\in\mathcal{B}}B)$.

LEMMA 15. *Each alternating block is an i-block for some $i = x,\ y$, or z.*

So in a regularized subdivision case, the i-block is just a linear sequence of boxes stacked along the i-direction.

For each alternating face, we will provide global rule for connecting them: the resulting arcs are parallel to one of the three vectors:

$$(1,1,0), (1,0,1), (0,1,1),$$

depending on the orientation of the face. E.g., if an alternating face is perpendicular to x axis, we will connect its four vertices with line segments that are parallel to the vector

$(0, 1, 1)$, as in Type 2b(ii) of Figure 4.3. We refer to this rule for connecting vertices the **Alternating Faces Rule** or **AF Rule** for short. We will show that this choice ensures global consistency and preserves isotopy.

We write "2b(x)" to refer to either 2b(i) or 2b(ii). Note that boxes of Types 2b(x) and 3b(x) in Figure 4.3 have only one alternating face; the boxes of type 4d(x) in Figure 4.3 has two alternating faces that are parallel to each other. Consider how these types can be combined in an alternating block: clearly, the block must begin and end with Types 2b(x) or 3b(x), and the non-end boxes must be Types 4d(x).

Thus each nontrivial alternating block has one of these three patterns:

$$(2b, 4d*, 2b), \quad (2b, 4d*, 3b), \quad (3b, 4d*, 3b)$$

where $4d*$ means a sequence of zero or more Type 4d(x) boxes.

We call '(x)' the **subtype** of Type 2b(x). Similarly for Type 4(x) and Type 3(x). So far, we have not concern ourselves with the subtype of our blocks. Locally, the way for connecting case 4d (Figure 4.3 Type 4d(i) and Type 4d(ii)) will not effect the topological structure. Different ways of connection result in the moving of critical point (e.g., from minimum point to saddle point, or from saddle point to maximum point, as shown in the circled boxes in Figure 4.6). The next lemma shows that this is crucial for blockwise consistency.

LEMMA 16. *In any alternating block, there can have at most one box B whose subtype is (i). Thus the surface type of B is Type 2b(i), Type 3b(i) or Type 4d(i); all the remaining boxes must have Type 2b(ii), Type 3b(ii) or 4d(ii).*

Proof. If we project an i-block to a plane normal to i, we obtain a square s. The projection of the surface in this i-block will be a connected region as illustrated in Fig-

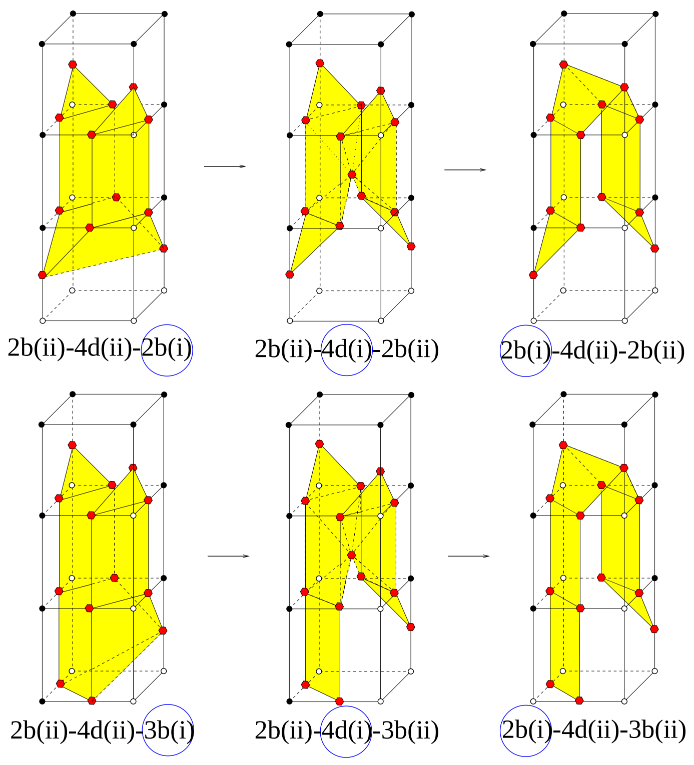


Figure 4.6: Different ways of connection result in the moving of critical point.

ure 4.8. The surface represented in Type 2b(i), Type 3b(i) or Type 4d(i) has only one connected component inside the box. So the projection of the surfaces represented by Type 2b(i), Type 3b(i) and Type 4d(i) must pass thought the center of s (as shown in Figure 4.7 Proj 2b(i), Proj 3b(i) and Proj 4d(i)). If there are two boxes of Type 2b(i), Type 3b(i) or Type 4d(i), the projection of the surface must pass through the center of s more than once, contradicting to the fact that the i-alternating block must be monotone in i-direction.

Q.E.D.

We compare the triangulated surfaces within the combination of $(2b, 4d, 2b)$ (as shown in Figure 4.8 Type 2b (ii)-4d (ii)-2b (i) and Type 2b (ii)-4d (i)-2b (ii)). The different ways of connecting the vertices of Type 4d boxes lead to the same topological

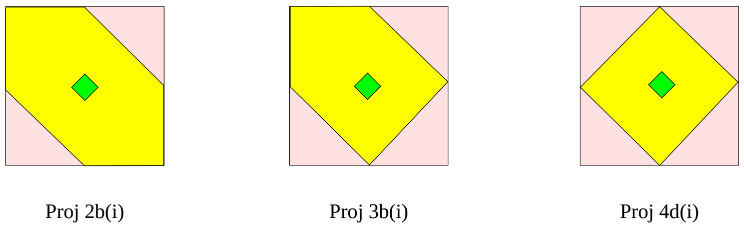


Figure 4.7: Examples of projections of i-blocks (2b(i), 3b(i), 4d(i)).

structure, as long as Lemma 16 is satisfied.

Case 4d (ii) can be viewed as a transitional case, which would not affect the block-wise topology. For example, we compare the combinations of $(2b, 4d, 2b)$ with $(2b, 2b)$. One possible mesh is shown in Figure 4.8 (2b (ii)-4d (ii)-2b (i)) and (2b (ii)-2b (i)). The topology for both meshes are the same. If we connect all the Type 4d boxes with arc Type 4d (ii), then we only need to consider the three basic combinations: $(2b, 2b)$, $(2b, 3b)$ and $(3b, 3b)$ (as shown in Figure 4.8).

For an alternating face, we have two ways to connect the vertices on the edges pair-wise. Both possibilities are shown in Figure 4.8. We claim that both choices lead to the same isotopic approximation. E.g., Figure 4.8 Type 2b (ii)-2b (i) and Type 2b (i)-2b (ii) are meshes constructed by two different connecting methods, and they are isotopic to each other. By applying the AF rule, we have the following lemma:

LEMMA 17. *The reconstructed surface S' in a y-block $\mathcal{B}$ is the graph of a function whose domain is the projection of the block onto the xz-plane. The possible projections of S' onto the xz-plane are shown in Figure 4.82b-2b (p), 2b-3b (p) and 3b-3b (p). Thus, $S' \cap \mathcal{B}$ is a topological disc.*

Proof. Since $\mathcal{B}$ is monotone in y direction, the projection of S' to the xz plane (e.g., Figure 4.82b-2b (p)) has a tubular neighborhood of fibers, and each fiber intersects S'

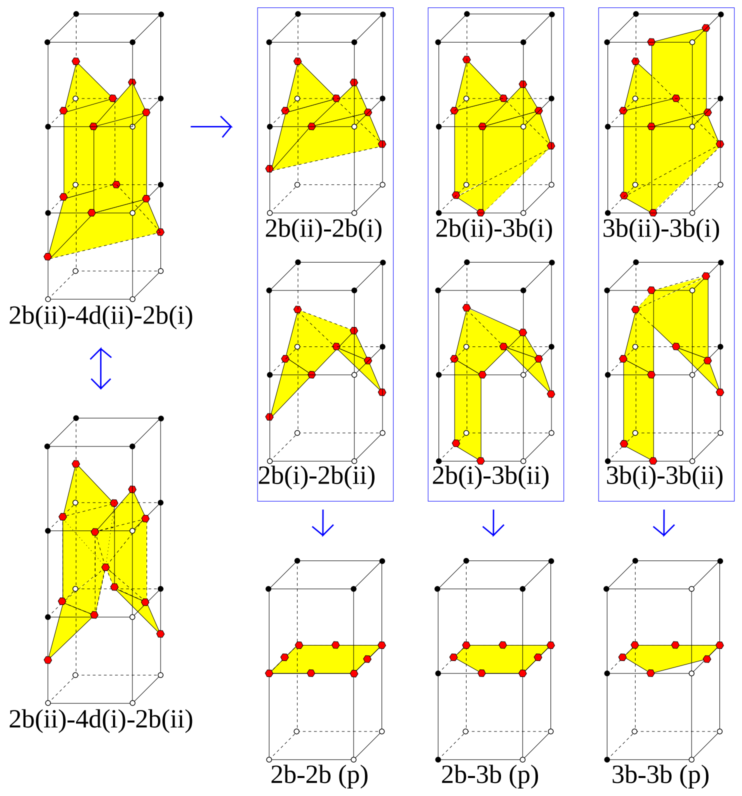


Figure 4.8: The two different triangulations for each of the three alternating block combinations.

in exactly one point[2]. So different connecting methods lead to the isotopic surfaces. We could choose either of them, as long as the triangulations for all the boxes fit together. From the case analysis above, the possible projections of S' onto the xz-plane are Figure 4.82b-2b (p), 2b-3b (p) and 3b-3b (p). So $S' \cap \mathcal{B}$ is a topological disc. **Q.E.D.**

[2]For the surface component on S' which are parallel to the y direction, we view it in the way that it has been infinitesimally slanted, such that each vertical fiber intersect S' in exactly one point.

4.2 Correctness of Regularized Cxyz Algorithm

We address the correctness of the Regularized Cxyz Algorithm. The proof is subtle, and harder than the $2D$ Regularized Cxy Algorithm or the $3D$ Regularized PV Algorithm. Our previous $2D$ proof for Cxy does not seem easy to generalize to $3D$, so we use a different approach. This proof will form the basis for proving the correctness of the Balanced Cxyz Algorithm in the next section.

First, we will prove the termination of the subdivision phase:

LEMMA 18. *If $S = f^{-1}(0)$ intersects the boundary of $R(T_0)$ generically, and if f has no singularities in $R(T_0)$, then the subdivision phase will terminate.*

Proof. If the subdivision phase does not terminate, then there is an infinite decreasing sequence of boxes $B_0 \supset B_1 \supset \cdots$ such that each $C_0(B_i)$ and $C_{xyz}(B_i)$ fail. Thus:

$$0 \in (\square f(B_i) \cap \square f_x(B_i) \cap \square f_y(B_i) \cap \square f_z(B_i)). \tag{4.5}$$

The boxes B_i must converge[3] to some point $p \in R(T_0)$ as $i \to \infty$. Since $\square f$ is a box function for f, we conclude that $\square f(B_i) \to f(p)$. Then (4.5) implies $0 = f(p) = f_x(p) = f_y(p) = f_z(p)$. Thus, f has a singular point in $R(T_0)$. **Q.E.D.**

Note that it is possible for $f_i(p) = 0$ $(i = x, y, z)$ where p lies on the boundary of a box. Figure 4.9 shows a $2D$ example where $f_x = 0$ on the edge of the boxes B_1 and B_2. In this example, $0 \in \square f_x(B_1)$ and $0 \in \square f_x(B_2)$, but $C_y(B_1)$ and $C_y(B_2)$ might still hold.

From now on, let T be the octree at the termination of the Regularized Cxyz Algorithm, and G be the graph constructed by our rules from T. We want to ensure

[3] The existence of p depends only on the existence of a bound r on the maximum aspect ratio – so this proof applies in the more general setting of Rectangular Cxyz Algorithm later.

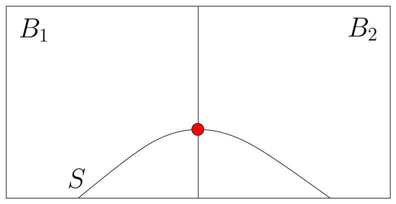
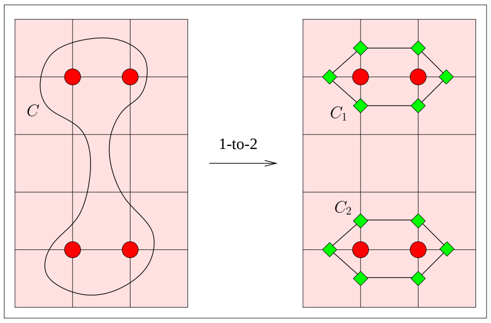


Figure 4.9: $2D$ example where $f_x = 0$ on the edge of the box.

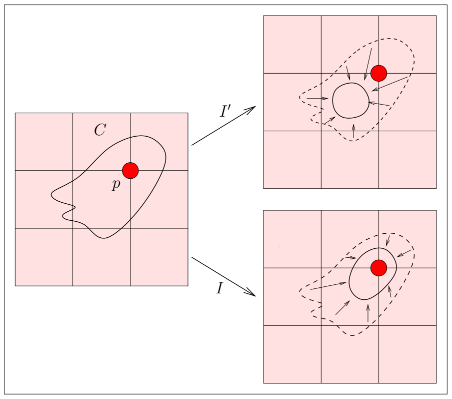
that $G \simeq S$ (**mod** $R(T)$). The outline of our proof is: we first transform S so another surface $\widetilde{S}$ which has some nice properties (e.g., $S \simeq \widetilde{S}$ (**mod** $R(T)$)); then we show that $G \simeq \widetilde{S}$ within each alternating block of T; finally, we can conclude that $G \simeq \widetilde{S} \simeq S$ (**mod** $R(T)$)

¶25. Intuition To understand the proof, it is helpful to be aware of potential issues: (1) We might gain components: See Figure 4.10 which shows that a component C of $S \cap R(T)$ might appear as two components of G. Note that the figure shows a $2D$ illustration, but one must imagine a third z-dimension. This example will not work in $2D$ if we assume that candidate boxes satisfy C_{xy} because the middle square is not monotone in the x or y-directions. But it could arise in $3D$, where the middle square might be monotone in the z-direction.

(2) We might lose a component: consider the isotopy I' at the top of Figure 4.10(b) that transforms a component C to a component C' lying inside a single box. This component C' is "lost" when we reconstruct G. One problem with the isotopy I' is it changes the sign of the function f at the red corner p. One way to prevent this from happening is to require our isotopies to preserve the sign of f at vertices. In our previous proof for Cxy Algorithm, we require that the transformed function f must remain monotone in at least one direction in each candidate box B. This would disallow the isotopy I' (no loop can arise in a box in which f is monotone in at least one direction). This approach



(a)



(b)

Figure 4.10: (a) One component is detected as two, (b) Two isotopic transformations.

seems hard to extend to $3D$, so we introduce the notation of "surface monotonicity" in the next paragraph.

¶26. Monotone Surfaces Let $S \subseteq \mathbb{R}^3$ be a continuous surface, $B \subseteq \mathbb{R}^3$ be a rectangular box and $i \in \{x, y, z\}$. An i-**line** is a straight line that is parallel to the i-axis.

We say S is i-**graph-like** in B if $|S \cap B \cap L| \leqslant 1$ for every i-line L. We say S is i-**monotone** in B if it is i-graph-like and we can assign a plus or negative sign to each connected component of $B \setminus S$ so that adjacent components have different signs and for each i-line L that is directed in the increasing i-direction, the line L never pass from a negative region to a positive region. In $2D$ case, we can similarly define i-**monotone** on the faces F of B. $2D$ examples of graph-like and monotone cases are shown in Figure 4.11. Note that L may keep the same sign as it passes through F/S, or it may change from a positive to a negative region.

Here is an alternative characterization of i-monotone:

LEMMA 19. *Let $B = I_x \times I_y \times I_z$. Then f is z-monotone in B iff there is a continuous function $\phi : I_x \times I_y \to I_z$ such that the graph $gr(\phi) = \{(x, y, \phi(x, y)) : (x, y) \in I_x \times I_y\}$*

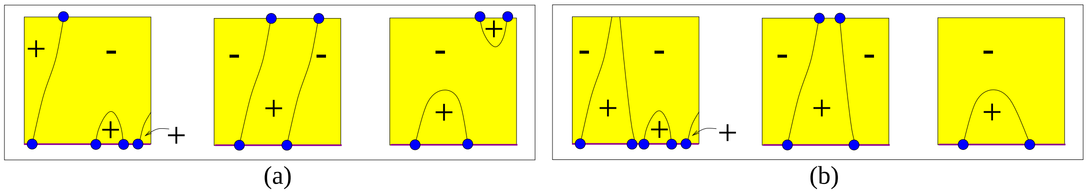


Figure 4.11: (a) $S \cap B$ is graph-like in B but not monotone, (b) $S \cap B$ is monotone.

of ϕ is equal to S in the interior of B, i.e.,

$$gr(\phi) \cap int(B) = S \cap int(B).$$

The easy proof is omitted. Note that if $(x, y) \in I_x \times I_y$ and $(x, y, \phi(x, y)) \notin S$ then $\phi(x, y)$ must be either $\max I_z$ or $\min I_z$. The continuity of the function ϕ is necessary to ensure monotonicity.

We simply say “graph-like” or “monotone” if i is understood from the context. For specificity, we usually let $i = y$ in illustrations. These definitions also make sense in $2D$ where S is a curve and B is a planar rectangle.

LEMMA 20. *Suppose $S = f^{-1}(0)$ where $f : \mathbb{R}^3 \to \mathbb{R}$. For any box B, if $\frac{\partial f}{\partial i}(p) \neq 0$ for all $p \in B$ then S is i-monotone in B.*

This lemma shows the origin of our monotonicity concept, and the proof of it is immediate. Next, suppose T is the octree produced by our regularized Cxyz algorithm on the input function f. Then for each box B in T which is intersected by $S = f^{-1}(0)$, there is a direction $i = i_B \in \{x, y, z\}$ such that S is i-monotone in B. Let $i : T \to \{x, y, z\}$ denote this (canonical) direction. Hence for each candidate box $B \in T$, we have a fixed direction i, where S is i-monotone in B.

S is **monotone** in T if S is i-monotone in each box B in T for some $i \in \{x, y, z\}$. Let S and $\widetilde{S}$ be two surfaces. We say $\widetilde{S}$ preserves the **monotonicity** of S in T if for any candidate box B in T, if S is i-monotone in B, then $\widetilde{S}$ is also i-monotone on B.

In our proof, we will begin with a surface that is monotone in all the candidate boxes in T, and we will repeatedly modify S to some $\widetilde{S}$ which preserves the monotonicity of S in T. What is important is that we can basically "forget" about the original function f as we do this modification, and we do not have to produce a suitable $\widetilde{f}$ with the property that $\widetilde{f}^{-1}(0) = \widetilde{S}$.

Relative to a surface S, an edge E is **dirty** if $|S \cap E| \geqslant 2$ or S intersects E tangentially, and a face F is **dirty** if $S \cap F$ contains a loop (i.e., closed curve) or S intersects F tangentially. The opposite of dirty is **clean**. A surface $\widetilde{S}$ is **clean** if every edge and face of T is clean relative to $\widetilde{S}$.

For the correctness[4] of our algorithm, we must modify our algorithm to do special "boundary processing" so that T is clean relative to S on the boundary faces. This processing amounts doing root isolation on the edges on $\partial R(T)$, followed by the $2D$ Cxy algorithm on the boundary of $R(T)$. These $1D$ and $2D$ processing are performed by splitting boxes in the octree. Boundary processing in the Cxyz Algorithm is similar to the Cxy Algorithm. For the following part, we will assume that the surface S intersects $\partial R(T)$ cleanly.

Note that for a box B, $S \cap B$ might be comprised of several connected components, but one can prove that (in the Regularized Cxyz algorithm) all these components must belong to the same (global) component of $S \cap R(T)$. Note that each component of S can give rise to zero, one, or more components of $S \cap R(T)$.

[4] All our correctness is up to an infinitesimal perturbation of f. It means that our algorithms miss tangential intersections of $S \cap R(T)$, when these components only occur on the boundary of $R(T)$. On the other hand, tangential intersections of $S \cap R(T)$ in the interior of $R(T)$ are excluded by explicit assumption.

¶27. Partial Order on Pairs We fix the usual octree T and f that defines the surface $S = f^{-1}(0)$. Let $\mathcal{P}(S)$ denote the set of all **pairs** of points $\{p, q\}$ such that there is an edge E of T, $\{p, q\} \subseteq E \cap S$ and the segment $[p, q]$ intersects S in an even number of points. *Note that the definition of pair in Cxyz Algorithm is more general than the definition of convergent pair in Cxy Algorithm.* We assume that $\mathcal{P}(S)$ is a finite set. We also regard the empty set $\mathcal{O}$ as a special element of $\mathcal{P}(S)$; all other pairs are called **non-empty pairs**. We say $\mathcal{P}(S)$ is **trivial** if its only member is $\mathcal{O}$.

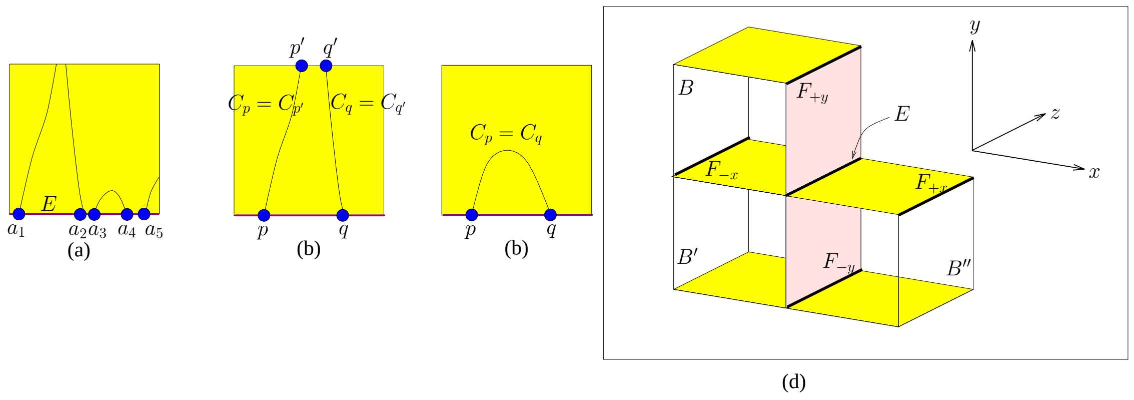


Figure 4.12: (a) Pairs on edge E, (b) $\{p, q\} \succ \{p', q'\}$, (c) $\{p, q\} \succ \mathcal{O}$

Example: Figure 4.12(a) shows an edge E with 5 intersection points with S. There are 6 pairs on E given by

$$\{a_1, a_2\}, \{a_2, a_3\}, \{a_3, a_4\}, \{a_4, a_5\}, \{a_1, a_4\}, \{a_2, a_5\}.$$

In general, an edge with n intersection points with S determines $p(n)$ pairs where $p(0) = 0$ and for $n \geqslant 1$, $p(n) = p(n-1) + \lceil (n-1)/2 \rceil$. So $p(1) = 0, p(2) = 1, p(3) = 2, p(4) = 4, p(5) = 6$.

We define a relationship between pairs of $\mathcal{P}(S)$. For any face F of T, we consider the connected curve components of $F \cap S$. If o is a point in $S \cap \partial F$, let C_o denote the connected component of $F \cap S$ that has o as one endpoint. Given two pairs $\{p, q\}, \{p', q'\}$, we define the relation

$$\{p, q\} \succ \{p', q'\} \, (\textbf{mod}\, F) \tag{4.6}$$

if $d(p, q) > d(p', q')$ and F has two opposite edges, E and E' such that $\{p, q\} \subseteq E$ and $\{p', q'\} \subseteq E'$, and the connected components of $S \cap F$ has this property: $C_p = C_{p'}$ and $C_q = C_{q'}$. Further define

$$\{p, q\} \succ \mathcal{O}(\textbf{mod}\, F) \tag{4.7}$$

if $\{p, q\} \subseteq \partial F$ and $C_p = C_q$. Both the relations (4.6) and (4.7) are illustrated in Figure 4.12(b,c).

For pairs $A, B \in \mathcal{P}(S)$, define the relation $A \succ B$ if there exists a face F such that $A \succ B(\textbf{mod}\, F)$. Let $\succeq$ denote the reflexive transitive closure of $\succ$: $P \succeq Q$ iff $P = Q$ or there is a finite sequence of pairs where $P = P_0 \succ P_1 \succ \cdots \succ P_k = Q$.

LEMMA 21. *The relation $(\mathcal{P}(S), \succeq)$ is a partial ordering on $\mathcal{P}(S)$*

Proof. We check three properties. Let $A, B, C \in \mathcal{P}(S)$. Reflexivity: $A \succeq A$ (by definition). Symmetry: $A \succeq B$ and $B \succeq A$ implies $A = B$. This is true if A or B is equal to $\mathcal{O}$. Otherwise, if $A \neq B$, we see that $A \succeq B$ implies $d(A) > d(B)$. Similarly, $B \succeq A$ implies $d(B) > d(A)$, contradiction. Transitivity: $A \succeq B \succeq C$ implies $A \succeq C$. This follows from the definition of $\succeq$. **Q.E.D.**

If $A \succeq B$, we say B is "smaller" than A and we are interested in minimal elements in this partial order.

Intuitively, $\mathcal{O}$ is the unique minima in $\mathcal{P}(S)$. Towards proving this result, we need a useful property of our octree T:

LEMMA 22. *Let S be a surface which is monotone in T, and E be any non-boundary* *edge of T such that $|S \cap E| \geqslant 2$. Assume (wlog) that E is parallel to the z-axis, and the four faces bounded by E are F_x, F_{-x}, F_y and F_{-y}, as in Figure 4.12(d). Then either S is x-monotone on $F_x \cup F_{-x}$, or S is y-monotone on $F_y \cup F_{-y}$.*

Proof. Suppose S is not x-monotone on F_{-x}. Consider the box B lying above F_{-x}. Since S cannot be z-monotone in B (because E intersects S in more than one point) and it cannot be x-monotone (since S is not x-monotone on F_{-x}), we conclude that S must be y-monotone in B. The same reasoning implies that S must be y-monotone in the box B' below F_{-x}. This concludes that S must be y-monotone on $F_y \cup F_{-y}$. **Q.E.D.**

LEMMA 23. *The empty set $\mathcal{O} \in \mathcal{P}(S)$ is the unique minimal element of $\mathcal{P}(S)$.*

Proof. We must show that for any non-empty pair $\{p, q\}$, there exists another pair $B \in \mathcal{P}(S)$ such that $\{p, q\} \succ B$. That is, either there exists $\{p', q'\}$ with $\{p, q\} \succ \{p', q'\}$ or $\{p, q\} \succ \mathcal{O}$.

Use the notations of the previous lemma, let $p, q \in E$ where E is an edge of T parallel to the z-axis. Wlog, let S be x-monotone on $F_{-x} \cup F_x$. Let C_p/C_q be the connected component of $S \cap (F_{-x} \cup F_x)$ that passes through p/q. If $C_p = C_q$, our lemma is shown, since $\{p, q\} \succ \mathcal{O}$. Otherwise, define the t-distance between C_p and C_q to be the intersection of these curves with the plane $\{x = t\}$. When $\{x = t\}$ contains E, clearly the t-distance is $d(p, q)$. As t increases, the t-distance increases or decreases monotonically. This distance cannot be zero since $C_p \neq C_q$. Moving in the direction where the t-distance decreases, we eventually reach the edge E' of F_{-x} or F_x where the t-distance is minimal. If $C_p \cap E' = p'$ and $C_q \cap E' = q'$, then we see that $\{p', q'\}$ is a pair in $\mathcal{P}(S)$ and $\{p, q\} \succ \{p', q'\}$. **Q.E.D.**

¶28. Cleansing Strategy We are going to transform S to another surface $\widetilde{S}$ that is clean relative to T. We do this by transforming S isotopically to $\widetilde{S}$. A difficult problem in this transformation is that it is very hard to keep track of the nice properties of the original f with respect to T. For instance, we know that each candidate box B of T must satisfy $C^f_{xyz}(B)$. We first overview the cleansing processes:

1. First, we clean all faces. Here we can exploit the original property of f. Because f is monotone in some coordinate direction in each box B, there cannot be loops in two adjacent faces of B. Moreover, the set of all such loops has a natural nesting partial order in each coordinate direction.

2. Next, assuming all the faces are clean, we can clean edges. Actually, we cannot clean an entire edge at once, but we remove pairs from $\mathcal{P}(S)$, one pair at a time. Let $S = S_0$ and we construct a new surface S_{i+1} from S_i by removing one pair. The fact that $\mathcal{P}(S_{i+1})$ is a proper subset of $\mathcal{P}(S_i)$ allows us to preserve the partial order that is induced from the original $\mathcal{P}(S) = \mathcal{P}(S_0)$. We show that each pair removal does not introduce any loop. So, at the end of this process, we have a surface S_k that is clean, and isotopic to S.

We next give details of these cleansing routines.

¶29. Cleaning Faces Consider the set of loops of S in faces of our octree T. Denote this set by $\mathcal{L}(S)$, and as before, introduce an artificial element $\mathcal{O}$ in $\mathcal{L}(S)$. We say $\mathcal{L}(S)$ is **trivial** if its only member is $\mathcal{O}$. We also assume that $\mathcal{L}(S)$ is a finite set.

Let L, L' be two distinct loops of $\mathcal{L}(S)$, and they lie on the boundary of a common box B. Let C_L denote the connected component of $S \cap B$ that is bounded by L. Wlog, let f be y-monotone in B. This implies that L and L' can only lie on y-faces of B. These two y-faces can be distinct or the same. We write $L \succ L'(\textbf{mod}\, B)$ if $C_L = C_{L'}$ and the y-projection of L' is contained in the interior of the y-projection of L (by y-projection, we mean the projection onto the $y = 0$ plane). Note that either $L \succ L'$ or $L' \succ L$ must occur because f is y-monotone in B. This ensures that we have a global partial ordering on $\mathcal{L}(S)$. This global property is derived from our original function f, and is critical for our proof. We must carry some of this information along in the induction, even after we have transformed f. Also, observe that the partial ordering can be naturally partitioned into three subrelations $\mathcal{L}(S) = \mathcal{L}_x(S) \cup \mathcal{L}_y(S) \cup \mathcal{L}_z(S)$, corresponding to the three coordinate directions.

Note that there can be several loops $L^{(i)}$ ($i = 1, 2, \ldots$) such that $L \succ L^{(i)}$. These $L^{(i)}$ can lie in the same face as L or in the opposite face. A fundamental property of this relation is this:

LEMMA 24. *For each loop L', there is at most one L such that $L \succ L'$.*

Proof. Say these loops lie on y-faces. If $L \succ L'(\textbf{mod}\, B)$, then the y-projection of L' is in the interior of the y-projection of L. Moreover, the component $C_L \subseteq B \cap S$ projects into the interior of L. If $L_0 \succ L'$ for some loop L_0, then we see that $C_{L_0} = C_L$ and $L_0 = L$. **Q.E.D.**

In the special case where the boundary of C_L is connected, then we have $\partial C_L = L$. In this case, we write $L \succ \mathcal{O}(\textbf{mod}\, B)$. This produces a partial order on the set of all loops (treating $\mathcal{O}$ as a special loop). Moreover, $\mathcal{O}$ is the unique minimum in this partial order. If $L \succ \mathcal{O}(\textbf{mod}\, B)$, we call $C_L \subseteq B$ a **cap**. Our transformation for loops amounts to repeated removing caps. Initially, let $S_0 = S$. We will define a sequence of surfaces, $S_1, S_2, \ldots$ such that the loops $\mathcal{L}_y(S_{i+1})$ is a proper subset of $\mathcal{L}_y(S_i)$ for each i.

Let $L \succ \mathcal{O}$ in $\mathcal{L}_y(S_i)$ lies in the face F and suppose B' is another box that is bounded by F. We can easily define a $(B \cup B')$-isotopy to transform S_i to S_{i+1} in which L does not occur in $\mathcal{L}_y(S_{i+1})$, but all the other loops of $\mathcal{L}_y(S_i)$ remains. Of course, if $L' \succ L$ in $\mathcal{L}_y(S_i)$, the removal of L may induce the new relation $L' \succ \mathcal{O}$ in $\mathcal{L}_y(S_{i+1})$.

Eventually, $\mathcal{L}_y(S_i)$ becomes trivial and contains only $\mathcal{O}$. We can independently repeat this argument on $\mathcal{L}_x(S_i)$ and $\mathcal{L}_z(S_i)$. All faces are clean when $\mathcal{L}(S)$ is empty.

¶30. Semi-loops and Bases We now have clean faces. To discuss the cleansing of edges, we need some additional concepts. Suppose F is a face and the surface intersects F in a number of curves, including loops (i.e., curve components with no endpoints). A non-loop curve component C whose two endpoints lie on the same edge E of F is called a **semi-loop** (E.g., C on F_{y+} or C' on F_{x+} in Figure 4.13). If p, q are the two endpoints of C, we call the line segment $[p, q] \subseteq E$ the **base** of the semi-loop C. Suppose F' is another face that is bounded by E, and F' has another semi-loop C' sharing the same base as C. Then we say C and C' are **linked** by this base. Suppose C, C' are linked semi-loops, there are two possibilities: they could be coplanar (Figure 4.13, C' and C'') or they may lie on a pair of perpendicular planes (Figure 4.13, C and C'). In general, a base can be shared by up to 4 semi-loops. The next lemma shows that this will not happen.

LEMMA 25 (NO FOURSOMES). *Let S be a surface which is monotone in T. Then at most 3 semi-loops can be linked together.*

Proof. If four semi-loops are linked together (as shown in Figure 4.13), since C and C''' are coplanar linked semi-loops sharing the base b, S can not be y-monotone on both faces of F_{y+} and F_{y-}. Let us assume that S is not y-monotone on F_{y+}. This implies that S must be x-monotone in the two boxes that sharing F_{y+} (note that S can not be z-monotone within the four boxes that sharing b). So S must be x-monotone on the faces F_{x+} and F_{x-}. On the other hand, the fact that C', C'' are coplanar linked semi-loops implies that S can not be x-monotone on both faces of F_{x+} and F_{x-}. This is a contradiction. **Q.E.D.**

REMARK: in subsequent transformation of S, "NO FOURSOMES" property will be preserved (as we will see).

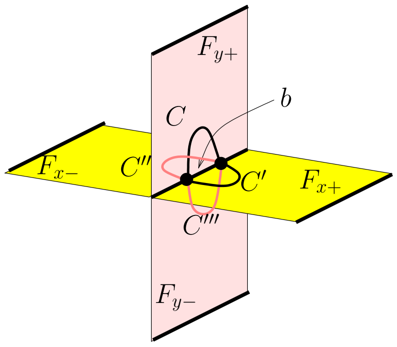


Figure 4.13: Impossibility of 4-linked semi-loops.

LEMMA 26 (NO HOLES). *Let S be the surface after the face cleaning process (note that S is monotone in T). Let $C, C' \subset S$ be linked semi-loops on the boundary of B. Let $P \subseteq S \cap B$ be a surface patch in B (i.e., P is a connected component of $S \cap B$). If $C \cup C' \subseteq \partial P$, then $\partial P = C \cup C'$. In other words, P is topologically a disc.*

Proof. Let B be the box containing C and C' in Figure 4.13. S must be monotone in x or y-direction in B. Wlog, let us assume that S is monotone in y-direction in B. Since P is converging in $y+$ direction, the projection of $P \cap int(B)$ onto F_{x+} must lie within C'. Also, $S \cap B$ contains no loop on the faces of B. So we can conclude that P is a topological disc and $\partial P = C \cup C'$. **Q.E.D.**

In other words, this lemma says that P cannot contain any holes as illustrated in Figure 4.14.

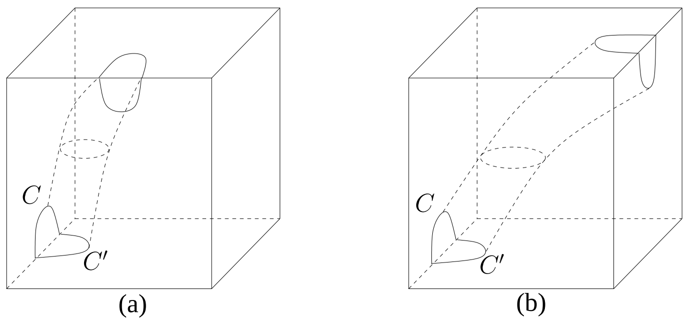


Figure 4.14: Examples of holes.

From the proof of Lemma 26, and the fact that a connected subset of an i-block can be viewed as a rectangular box in which S is monotone in i-direction, it is easy to see that the following lemma is also correct:

LEMMA 27 (NO HOLES 1). *Let $\mathcal{B}$ be a connected subset of an i-block, and S be a surface that is monotone in T which intersects the faces of $B \in \mathcal{B}$ cleanly. Let $C \subseteq S \cap \partial(\cup_{B\in\mathcal{B}} B)$ be a closed curve, and $P \subseteq S \cap \mathcal{B}$ be a connected component. If $C \subseteq \partial P$, then $\partial P = C$. In other words, P is topologically a disc.*

REMARK: in subsequent transformation of S, this property will also be preserved (as we will see).

¶31. Cleaning Edges via Base Removal Operations Let us retain the notations of Figure 4.12 relative to an edge E containing a pair $\{p, q\}$. We call a pair $\{p, q\}$ **penultimate minimum** (or $\{p, q\} \succ_* \mathcal{O}$) if for any pair P, $\{p, q\} \succ P$ implies $P = \mathcal{O}$. If $\{p, q\} \succ_* \mathcal{O}$ and for exactly i of the faces $F \in \{F_x, F_{-x}, F_y, F_{-y}\}$, $\{p, q\} \succ \mathcal{O}(\textbf{mod}\, F)$, then we say $\{p, q\} \succ_i \mathcal{O}$. Note that if $\{p, q\} \succ_i \mathcal{O}$, then $i \geq 1$. In other words, $\{p, q\} \succ_0 \mathcal{O}$ is not possible. We call a base $b = [p, q]$ a **penultimate minimum base** if $\{p, q\}$ is a penultimate minimum pair. Clearly, penultimate minimum base is a base of some semi-loops.

We will remove one penultimate minimum pair in $\mathcal{P}(S)$ each time. Let $S = S_0 = f^{-1}(0)$ and suppose we construct a new surface S_{i+1} from S_i by removing one pair from $\mathcal{P}(S_i)$. The fact that $\mathcal{P}(S_{i+1})$ is a proper subset of $\mathcal{P}(S_i)$ allows us to preserve the partial order that is induced from the original $\mathcal{P}(S) = \mathcal{P}(S_0)$. Our removing of penultimate minimum pairs will not change the partial order in $\mathcal{P}(S)$. In each step $\mathcal{P}(S_i) = \mathcal{P}(S_{i+1}) \cap \{\{p_i, q_i\}\}$ where $\{p_i, q_i\}$ is the penultimate minimum pair which we remove at step i. The removing only creates new relations of the form $\{p, q\} \succ \mathcal{O}$ where $\{p, q\} \succ \{p', q'\}$ in $\mathcal{P}(S_i)$.

The next lemma shows that if a base $b = [p, q]$ is a penultimate minimum base and $\{p, q\} \succ_2 \mathcal{O}$, then the two linked semi-loops must lie on a pair of perpendicular planes:

LEMMA 28. *Let S be a surface that is monotone in T, and $\{p, q\}$ be a pair of $S \cap T$. Consider two distinct faces F_s and F_v in Figure 4.12 where $\{s, v\} \subset \{x, -x, y, -y\}$. If $\{p, q\} \succ_2 \mathcal{O}$ where $\{p, q\} \succ \mathcal{O}(\textbf{mod}\, F_s)$ and $\{p, q\} \succ \mathcal{O}(\textbf{mod}\, F_v)$, then $\{s, v\} \neq \{x, -x\}$ and $\{s, v\} \neq \{y, -y\}$.*

Proof. If $\{p, q\} \succ \mathcal{O}(\textbf{mod}\, F_x)$ and $\succ \mathcal{O}(\textbf{mod}\, F_{-x})$, and curves $C_p, C_q \subseteq S \cap (F_{-y} \cup F_y)$ are the connected components that passes through p and q, then C_p and C_q must be different components in $F_{-y} \cup F_y$. Since $\{p, q\}$ is a penultimate minimum pair, S can not be y-monotone in $F_y \cup F_{-y}$. From Lemma 22, we know that S is x-monotone in $F_x \cup F_{-x}$, which contradicts the fact that $[p, q]$ is the base of two coplanar linked semi-loops on $F_x \cup F_{-x}$. **Q.E.D.**

Suppose $P \succ_i \mathcal{O}$ where P is a pair. We already noted that $i = 0$ is not possible. From Lemma 25, if we can preserve the monotonicity of S during the surface transformation (which will be proven later), then $i = 4$ is also impossible. So the only possibilities for i is $1, 2$ and 3. Because of Lemma 28, a penultimate minimum base b could have three possibilities, as shown in Figure 4.15(I), (II) and (III). Note that if b is not a penultimate minimum base, Figure 4.15(III'') might arise.

Let b be a penultimate minimum base for some semi-loop. To "remove" b means to simultaneously remove all the semi-loops that share the base b. Since there are only three possibilities, so there are three distinct base removal operations. This is shown in Figure 4.15. In Figure 4.15 $(I) \to (I')$, we push down the part of semi-loop component to form a "tunnel" below the edge E. In Figure 4.15 $(II) \to (II')$, we push the topological disc component bounded by the two semi-loops in both $x-$ and $y-$ directions to eliminate it. In Figure 4.15 $(III) \to (III')$, we push down the topological disc component bounded by the three semi-loops to remove the it. Note that these operations are well-defined: this depends on the fact that in each box B that contains a pair of linked semi-loops C and C', the surface patch bounded by $C \cup C'$ is a topological disc (i.e., the "NO HOLES" property in Lemma 26 holds as long as we preserve the monotonicity of the surface during our operations, which will be proven in the following part).

We next describe some properties that our transformation preserves. Let T be an octree and V_T be the set of all corners of the boxes in T. Let S, S' be two surfaces. We say S is **compatible** with S' (respect to T) iff there exist an isotopy $I : \mathbb{R}^3 \times [0, 1] \to \mathbb{R}^3$, s.t. $I(\cdot, 0)$ is the identity; $I(S, 1) = S'$ and $\forall t \in [0, 1]$, $I(S, t) \cap V_T = \varnothing$.

LEMMA 29. *The face cleaning operations and the base removal operations preserve the compatibility of S in T.*

Proof. The correctness of this lemma is based on the nature of our operations: we never transform the surface "across" any corners in T. **Q.E.D.**

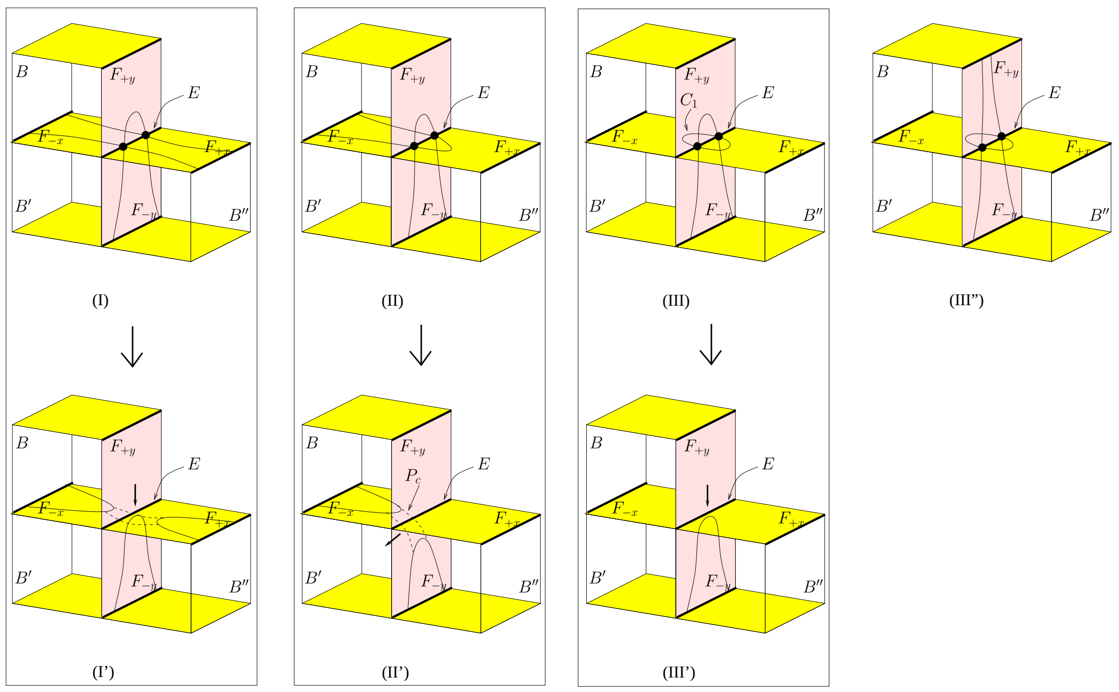


Figure 4.15: Three Base Removal Operations.

LEMMA 30 (Surface Monotonicity Preservation). *Base removal operations preserve the monotonicity of S in T.*

Proof. There are three cases to be considered, corresponding to the three base removal operations. We will analyze each case to show that the monotonicity is preserved within each box. Let S and S' be the surfaces before and after each operation.
Case $(III) \rightarrow (III')$: by symmetry, we only need to consider the monotonicity of the surface in B' (the removal of the topological disc component bounded by the two linked semi-loops does not affect the surface monotonicity in B). There are two possibilities: $S \cap B'$ is monotone in the x direction or $S \cap B'$ is monotone in the y direction. If $S \cap B'$ is monotone in y direction, $S' \cap B'$ is also monotone in y direction iff every y-line L intersects with $int(C_1)$ does not intersect $S \cap B'$. Note that $L \cap S \cap B' \geq 1$ iff S intersects the $int(C_1)$ with a curve C'. C' can not be a loop since we have already cleaned

the faces. So C' must intersect with E, which contradicts the fact that we process the pairs in partial order. If $S \cap B'$ is monotone in x direction, we can see that the operation transforms the surface patch below F_{-x} to form a "cap". By carefully transforming the surface, we can ensure that any x-line L intersect the "cap" at most once. Note that there is a curve on the "caps" in $B' \cap B''$ such that any x-line passes a point on the curve is tangent to S', so the above monotonicity preservation depends on the fact that S can not be x monotone in both B' and B''. We can transform the curve to be contained in the box that is not x monotone.

Case $(II) \rightarrow (II')$: by symmetry, we only need to consider the boxes B and B'. The monotonicity preservation argument in the box B is the same as in the box B' in $(III) \rightarrow (III')$. So we only need to analyze the box B'. Again, by symmetry, we can assume that $S \cap B'$ is monotone in y direction. Let $S \cap B' = S_1 \cup S_2 \cup, \ldots, \cup S_n$, where each $S_i (i = 1, \ldots, n)$ is a connected surface component. Note that $(II) \rightarrow (II')$ connect two surface patches S_u and S_v to form one surface patch in B'. Let P_c be a surface patch in S' which connect S_u and S_v (as shown in Figure 4.15$(II) \rightarrow (II')$). We will show how to construct P_c. We pick a z-line L_1 on F_{-x} such that for all $S_i (i = 1, \ldots, n)$, if S_i does not intersect with F_{-y}, then the distance $dis(S_i, F_{-y})$ of S_i and F_{-y} is larger than $dis(L_z, F_{-y})$. We can similarly pick another z-line L_2 on F_{-y}. The examples of L_1 and L_2 are shown in Figure 4.16 (including all the points' and curves' labels). L_1 intersects C_1 and C_2 at two points (p_1, q_1), and L_2 intersects C_1 and C_2 at two points (p_2, q_2). Let the y-projections of p_1, q_1, p_2 and q_2 onto the bottom face of B' be P_{p_1}, P_{q_1}, P_{p_2} and P_{q_2}. Let S_{c_1} and S_{c_2} be the surface patches bounded by C_1 and C_2. Then S_{c_1} intersects with the rectangle $(p, p_1, P_{p_1}, P_{p_2})$ in a curve I_{c_1}. Similarly, we have I_{c_2}. There exist a surface patch bounded by the lines $[p1, q1]$, $[p2, q2]$ and the curves I_{c_1}, I_{c_2} s.t., it has the some monotonicity as S in B'. We define P_c to be such a surface patch.

Let the y-projection of P_c be P_{P_c}. $S' \cap B'$ is also monotone in y direction iff all the y-line L which intersect with P_{P_c} do not intersect $S \cap B'$. If $L \cap S \cap B' \geq 1$, then $S \cap B'$ must contain a surface patch S_u which intersects with E, which contradicts the fact that we process the pairs in partial order.

Case $(I) \rightarrow (I')$: by symmetry, we only need to analyze the boxes B and B'. The monotonicity preservation argument in the box B is the same as in the box B' in $(III) \rightarrow (III')$, and the monotonicity preservation argument in the box B' is the same as in the box B' in $(II) \rightarrow (II')$. **Q.E.D.**

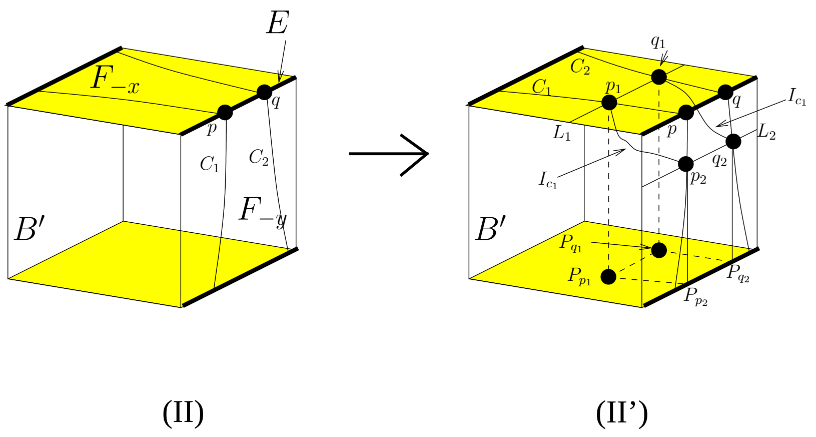


Figure 4.16: Construction of P_c.

The next example shows that if we remove the bases in arbitrary order, we might create holes within the boxes. Let $b1$ be the smallest base in the box B in Figure 4.17(I). Assume S is y-monotone in B, since our operation preserves the monotonicity, we have the length of $b3$ is less than the length of $b4$.. If we remove the bases in arbitrary order, we might remove $b1$ and $b4$ before $b2$ and $b3$, which results in a hole as shown in Figure 4.17(I').

LEMMA 31. *The face cleaning operations do not induce new dirty faces, and the base removal operations do not induce new dirty edges and dirty faces.*

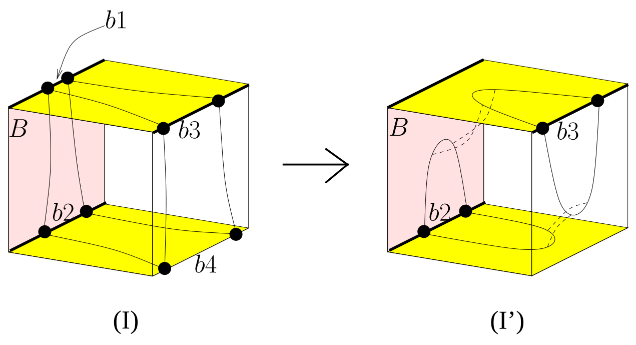


Figure 4.17: Removing bases in arbitrary order might create holes.

Proof. It is clear that the face cleaning operations do not induce new dirty faces, and the base removal operations do not induce new dirty edges. We will show that the base removal operations do not induce new dirty faces. Let R be a base removal operation which removes a penultimate minimum pair b and induces a new loop l on a face F. Then before the operation, l was a semi-loop with the base b. This contradicts the fact that R removed all the semi-loops that share the same base b. **Q.E.D.**

The above base removal process halts only when $\mathcal{P}(S)$ is empty. At this point, all faces and edges are clean relative to T. From the analysis above, we have the following theorem:

THEOREM 32. *Let T be the octree produced by our Regularized Cxyz Algorithm. There $\exists \widetilde{S}$, s.t.*

(1) $\widetilde{S} \simeq S(\mathbf{mod}\, R(T))$.

(2) $\widetilde{S}$ is compatible with S respect to T.

(3) $\widetilde{S}$ intersects T cleanly.

(4) $\widetilde{S}$ preserves the monotonicity of S within each candidate box of T.

Proof. We first clean the faces, then we clean the edges. From Lemma 29, Lemma 30 and Lemma 31, and the fact that each operation is an isotopic transformation, the resulting $\widetilde{S}$ satisfies all the properties in this theorem. **Q.E.D.**

THEOREM 33. *Let G be the mesh we construct by the Regularized Cxyz Algorithm, then $G \simeq S(\textbf{mod}\, R(T))$.*

Proof. Based on the construction phase of our algorithm, for each alternating block $\mathcal{B}$, $\widetilde{S} \cap \partial(\cup\mathcal{B})$ "agrees" with $G \cap \partial(\cup\mathcal{B})$. From Lemma 27, we know that $\widetilde{S}$ is isotopic to G within each block. So $G \simeq \widetilde{S}(\textbf{mod}\, R(T))$. From Theorem 32, we have $G \simeq \widetilde{S} \simeq S(\textbf{mod}\, R(T))$. **Q.E.D.**

4.3 Balanced Cxyz Algorithm

In the previous section we have shown that the Regularized Cxyz Algorithm can be used to create an isotopic approximation of an implicit surface. Now we will describe that how a balanced octree can be used to create an isotopic mesh. The subdivision process is the same as in the regularized case. After the subdivision process, we "balance" the octree. The definition of balance is "edge-balance", and it is given next.

First, note that we regard the boxes of an octree to be closed subsets of $\mathbb{R}^3$. For the purposes of balancing the octree after subdivision, we will define two boxes B, B' as **neighbors** if the interiors of B and B' are disjoint, and their boundaries share an open line segment: $\partial B \cap \partial B'$ contains an open line segment. If they only share a corner, they are not neighbors.

Let $i \in \{x, y, z\}$. An edge of a box is an i-**edge** if it is parallel to the i-axis. An octree is i-**balanced** if for all pairs of candidate boxes B, B' which are neighbors, if $B \cap B'$ contains a open segment of an i-edge of B or B', 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 $i = x, y, z$.

Recall that the width[5] of a box B is defined as $w(B) := \min\{w_x(B), w_y(B), w_z(B)\}$. If all the boxes in T are cubes, then for any box $B \in T$, the i-widths of B are the same for $i \in \{x, y, z\}$. For any edge e of B, any other box that share part of the interior of e must have a width at least half the width of B. Also note that if e is not a boundary edge, then there are between 3 and 6 other boxes that share part of the interior of e.

We will first introduce the Balanced Cxyz Algorithm. We store candidate boxes from each phase into a priority queue, and pass it into the next phase. The comparator for the priority queues is the width of the boxes:

```
Balanced Cxy Algorithm:
Input:    Nice region given by an octree $T_0$ and surface $S = f^{-1}(0)$
Output:   Isotopic approximation $G$ for $S \cap R(T_0)$
1.   $T_1 \leftarrow SUBDIVIDE_{C_{xyz}}(T_0)$
2.   $T_2 \leftarrow BALANCE(T_1)$
3.   $G \leftarrow CONSTRUCT(T_2)$
```

The subdivision phase has been described already. We will next describe the balancing phase. The balancing phase has three sub-phases:

[5] Note that the initial ROI might not be a cube (or cubes). So even if we perform full-split for any box B in T, the i-widths of B might still be different. But the minimum i-width is enough to identify the depth of B in T.

```
$BALANCE(T_1)$:
2.1. $T_1' \leftarrow Split(T_1)$
2.2. For each candidate box in $T_1'$, we introduce vertices in the
middle of bichromatic edges.
2.3. $T_2 \leftarrow Disambiguate(T_1')$
```

The first sub-phase is based on the definition of balancing, where $w(B)$ denotes the width of the box B:

```
$Split(T_1)$:
Assume $T_1$ has an associated priority queue $Q$ containing all of its candidate boxes
Let $Q_1$ be an empty priority queue
While ($Q$ is non-empty)
    $B \leftarrow Q.pop()$
    boolean $BalancedBox \leftarrow true$
    For each candidate box $B'$ that is a neighbor of $B$
        If $w(B') > 2w(B)$,
            $BalancedBox \leftarrow false$
            Full-split $B'$
            For each candidate box $B''$ that is a child of $B'$
                Insert $B''$ into $Q$
    If ($BalancedBox$)
        Insert $B$ into $Q_1$
    Else
        Insert $B$ into $Q$
Return the extended octree $T_1'$ represented by $Q_1$.
```

The third sub-phase is the disambiguation sub-phase. We introduce three ambiguous cases, which will be described in the following paragraph.

¶32. Disambiguation Phase We indicate the issues that arise if we simply replace C_1 by C_{xyz} in the Balanced Algorithm. Consider an horizontally-stretched hyperboloid as in Figure 4.18 (a_1). We run the Balanced Cxyz Algorithm on this hyperboloid. If the subdivision phase ends up with the 10 boxes[6] shown in Figure 4.18 (a_2). Clearly, both of the two larger boxes (B_1 and B_3) satisfy C_x, while the eight smaller boxes satisfy C_{xyz}. The output graph G obtained by using the connection rules (in the regularized algorithm) is the yellow polytope of Figure 4.18 (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 planer surface. If we had split B_1, we would have discovered this error. In this case we say B_1 (resp., B_3) has "$3D$ ambiguity". A very similar problem is seen in Figure 4.18(b_1) and (b_2), corresponding to "$2D$ ambiguity" in each of the boxes B_1, B_3, B_4, B_6.

From the previous analysis, we can define the first two "ambiguous cases" (by symmetry, we may assume that $C_y(B)$ holds):

1. $3D$ **Ambiguity**: The interior of the top or bottom face has four vertices. In Figure 4.18 (a_2), the boxes B_1 and B_3 are both ambiguous by this criterion.

2. $2D$ **Ambiguity:** One or more of its vertical faces is monochromatic, and has exactly two vertices on the same edge. By $C_y(B)$, this edge is not a vertical

[6] In the actual subdivision phase, the boxes after subdivision will not end up with these 10 boxes. The reason is that there exists a critical point p in box B_2, i.e., $f_x(p) = f_y(p) = f_z(p) = 0$. So the subdivision phase will subdivide some children of B_2 at least one more time to produce C_0 boxes that include p. A similar $2D$ example is shown in Figure 3.2. But it is too complicated to draw such an example in $3D$, and Figure 4.18 is enough for us to illustrate the ambiguous cases.

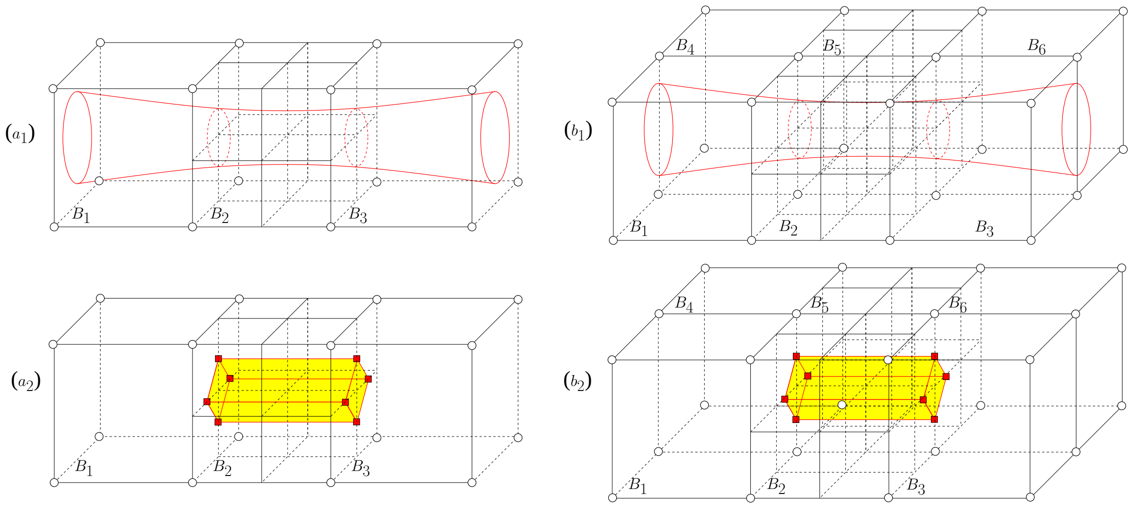


Figure 4.18: Examples of two kinds of ambiguous boxes.

edge. In Figure 4.18 (b_2), the boxes B_1, B_3, B_4 and B_6 are all ambiguous by this criterion.

Unlike the $2D$ case (see ¶14), the definition of the $3D$ ambiguity does not require the box to be monochromatic. Figure 4.19 show an example of the $3D$ ambiguity[7] in a bichromatic box. Also note that our definition of ambiguity is designed to be simple, but it does not prevent unnecessary splitting (e.g., if both the top and bottom faces each have exactly four vertices in their interiors, then there is really no need for splitting).

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. We split B into eight sub-boxes, and put the candidate boxes among the children back

[7] One might be able to develop a more complicated connection rule for connecting the vertices for the $3D$ ambiguity in a bichromatic box B, since we know that the $S \cap B$ will form a cylinder shaped surface patch within B. In our algorithm, we just split B for simplicity.

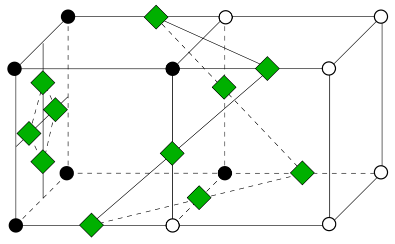


Figure 4.19: Example of the $3D$ ambiguity in a bichromatic box.

into the octree.

LEMMA 34. *If we split an ambiguous box B into 8 children, none of these children will be ambiguous.*

Proof. Let B' be a child of an ambiguous box B. Because its neighboring boxes can not have smaller width than B' (otherwise, the width of the neighboring box is less than half of the width of B). So it is impossible for B' to have two vertices on one edge or have four vertices on the interior of one face. It is also impossible for B' to properly contains any alternating face of its neighbors. **Q.E.D.**

Note that splitting of ambiguous boxes might induce its edge-neighbors to become ambiguous, and also cause the octree to be unbalanced. So we need to re-balance the octree. But this re-balance procedure is very local, and we only need to propagate the "modified" boxes. The following is the disambiguation sub-phase (sub-phase 2.3 of $BALANCE(T_1)$).

```
$Disambiguate(T'_1)$:
Assume $T'_1$ has an associated priority queue $Q$ containing all of its candidate boxes
Let $Q_1$ be an empty priority queue
While ($Q$ is non-empty)
    $B \leftarrow Q.pop()$
    If $B$ is an ambiguous box
        Full-split $B$
        For each candidate box $B'$ that is a child of $B$
            $Rebalance(B')$
            Insert $B'$ into $Q_1$
    Else
        Insert $B$ into $Q_1$
Return the extended octree $T_2$ represented by $Q_1$.
```

The following is the re-balance routine which is used in the disambiguation sub-phase. Note that this re-balancing procedure relies on the fact that the octree T'_1 has already been balanced before.

```
$Rebalance(B_0)$:
    Priority queue $Q$ is initialized to be $\{B_0\}$
    While $Q$ is non-empty:
        $B \leftarrow Q.pop()$
        For each on-box $B'$ that is a neighbor of $B$
            If $w(B') > 2w(B)$
                Full-split $B'$
                For each candidate box $B''$ that is a child of $B'$
                    Insert $B''$ into $Q$
```

We will next describe the construction phase for the Balanced Cxyz Algorithm.

¶33. 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.

By an **arc loop**, we mean a closed curve of arcs on the boundary of a box B. The construction phase also has three sub-phases (3.1-3.3).

```
$CONSTRUCT(T_2)$:
    3.1. $InitialConnect(T_2)$
    3.2. $ArcConnect(T_2)$
    3.3. For each candidate box $B$ in $T_2$, group the arcs on $B$'s boundary into arc loops.
         For each arc loop, form a triangulated surface patch whose boundary is the arc loop.
```

Sub-phase 3.3 is straightforward. In the following, we will describe how to implement sub-phase 3.1 and 3.2. In order to introduce our arc connection rule for active faces, we will first analyze the sign types of the active faces.

¶34. 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 4.20 – there are 13 in number. The sign type of F will uniquely determine the vertices that are introduced into F (as illustrated in Figure 4.20).

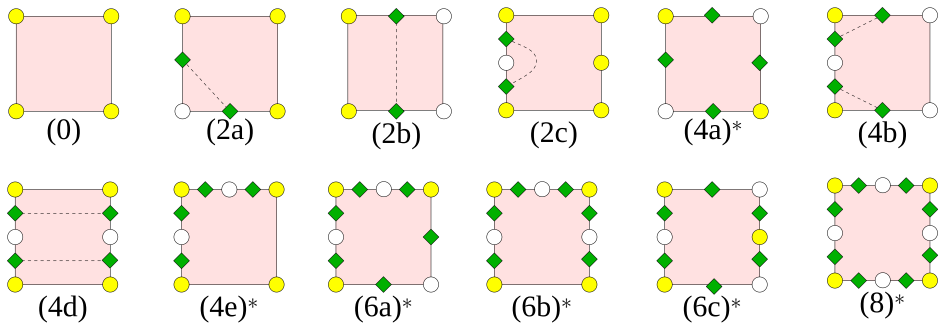


Figure 4.20: Sign Types of active faces. The asterisks indicate the cases that are impossible for the active faces on the boundary of blocks.

¶35. Arc Types of Active Faces The rule for arc connections of active faces depends on whether the faces are (known to be) "parametrizable" or not.

Let F be an active z-face. F is said to be **parametrizable** if $0 \notin f_x(F)$ or $0 \notin f_y(F)$. One problem with this notion is that it is not an effective one – we may not know that a face is parametrizable even though it is. One computationally checkable condition which implies the parametrizability of F is $0 \notin \square f_x(F)$ or $0 \notin \square f_y(F)$. But for our algorithm, we will define the concept of "known parametrizable" faces using the information that is already obtained from our subdivision phase. The definition is based on the fact that each candidate box B satisfies $C_i(B)$ for some $i = \{x, y, z\}$. For every box $B \in T$, we associate a **known monotone direction** (or **monotone direction** for short). Now we define the concept of "known parametrizable faces". Let F be an active face, and suppose F bounds two boxes B and B'. So $F = B \cap B'$. 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**. Examples of known parametrizable faces and not known parametrizable faces are shown in Figure 4.21. Let the known monotone direction of B be y in both Figure 4.21(a) and (b). Then the four vertical faces of B are known parametrizable faces. If the known monotone direction of B' is also y, then F is a not known parametrizable face; otherwise, F is a known parametrizable face, which has the same monotone direction as B'. Clearly, if F is known monotone in some direction, then it is monotone in that direction (converse is not true).

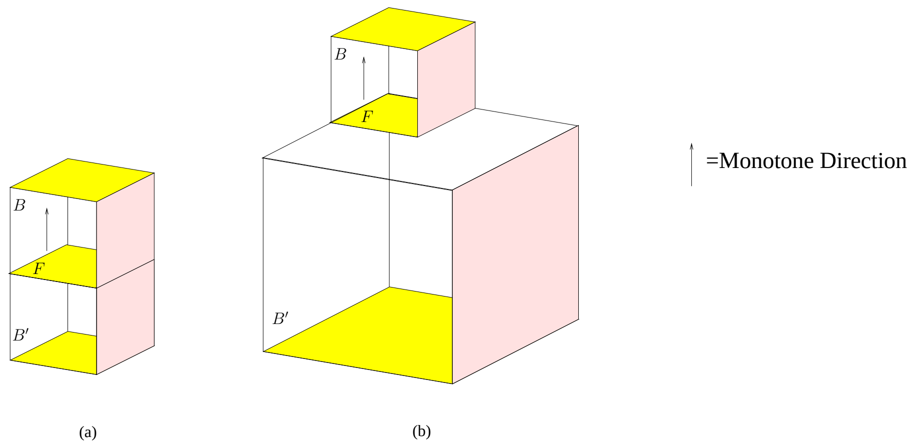


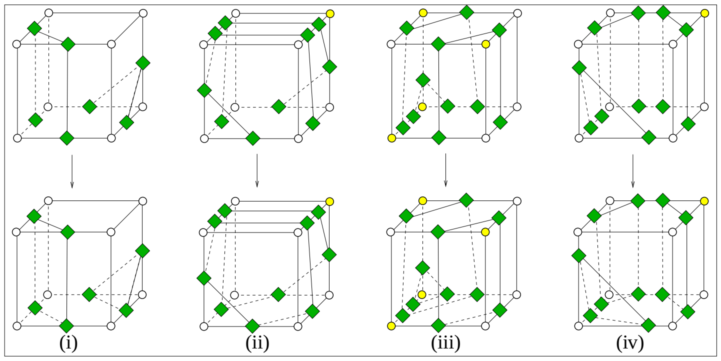
Figure 4.21: Examples of known parametrizable faces and not known parametrizable faces.

¶36. Connection Rule 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 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 rules for any known parametrizable faces can follow the rules given in Figure 3.4. For reference, we call them the **parametrizable face rule**, which is reproduced in Figure 4.20(2a), (2b), (2c), (4b), (4c) and (4d).

It remains to give the connection rule for the case where F is not known parametrizable. We knew that in the regularized algorithm, the arc connections on F may be arbitrary, as long as we ensure a certain block-wise consistency. In the Balanced Cxyz Algorithm, we will need a different approach.

For a box B, let UF_B denotes the number of faces that have not yet been connected. There are at least four known parametrizable faces, which we know how to connect. So we need to connect at most two other faces, i.e., $UF_B \leq 2$. We first introduce the connection rule for boxes where all but one faces have been connected, i.e., $UF_B = 1$. We call this rule the **matching rule**: wlog, let B's monotone direction be y, and the top face of B has been arc connected. Let F be the bottom face of B, and $v_1, v_2, \ldots, v_{2n}$ be the vertices on F. For a vertex $v \in F$, if we follow the arcs starting from v on the vertical and top faces of B, the path must end at another vertex v' on the bottom face F. We say v and v' are **matched**. It is easy to see that this pairwise relationship forms a partition of the set of vertices on F. We connect v_i and v_j iff v_i and v_j are matched. Figure 4.22(i), (ii), (iii) and (iv) show some examples of using matching rule to connect vertices.

We still need the connection rule for the boxes B whose $UF_B = 2$. We previously defined the notion of an "i-block ($i \in \{x, y, z\}$)" for a regular octree. We have a similar definition for the balanced octree T (wlog, let $i = y$): a y**-block** $\mathcal{B}$ is a sequence $B_1, \ldots, B_t$ of candidate boxes of T 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 is y for each B_i; and (3) the block is maximal. Note that this implies that all the boxes in a block have the same width, as in the regular case. The **width** of the block is defined as the width of any B_i. We also



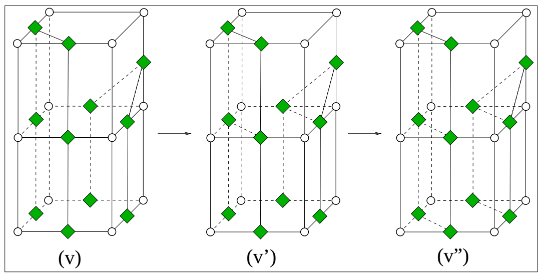


Figure 4.22: Examples of how to use matching rule ((i), (ii), (iii) and (iv)) and propagation rule ((v)→(v')→(v")) to connect vertices.

define the **end boxes** of $\mathcal{B}$ to be B_1 and B_t, and the **end faces** of $\mathcal{B}$ to be the top face of B_1 and bottom face of B_t. We also define the boundary of $\mathcal{B}$ to be: $\partial(\cup\mathcal{B}) = \partial(\cup_{B\in\mathcal{B}} B)$ (i.e., the union of its end faces and all the vertical faces).

Every candidate box $B \in T$ has been assigned a monotone direction. Then this partitions the set of candidate boxes of T into blocks. Let $\mathcal{B}$ be a y-block. We can view $\mathcal{B}$ as a single rectangular box B^r. The surface S is y-monotone within B^r, so S intersects each vertical edge of B^r at most once. The top and bottom faces of B^r are the end faces of $\mathcal{B}$. For any box $B \in \mathcal{B}$, the connection rule for the vertical faces is uniquely defined (the parametrizable face rule). So the only not connected faces on the boundary of $\mathcal{B}$ are the two end faces. The following lemma shows that the connection rule for the active end faces is also uniquely defined.

LEMMA 35. *If F is an end face of a block, and if F is active, then F has at most 4 vertices. The possible sign types for F are shown in Figure 4.20(0), (2a), (2b), (4b), (4c) and (4d), and the connection rule for those cases is uniquely defined.*

Proof. If F is an end face of a y-block $\mathcal{B}$, then F is either (1) the intersection of $\mathcal{B}$ with another block of larger width (recall that the width of the block is defined as the width of any B_i in the block), or (2) the intersection of $\mathcal{B}$ with another x- or z-block of the same width. In the first case, the boxes that share any edge of F have either larger or the same width as F (because of the edge-balance). There is at most one vertex on each edge of F, so F has at most 4 vertices. By the definition of alternating ambiguity, Figure 4.20(4a) is excluded. So the possible cases are Figure 4.20(0), (2a) and (2b). Their connection rule is uniquely defined. In the second case, F is a known parametrizable face. So there are at most four vertices on F. From the analysis in our Cxy Algorithm, we know that the possible cases are Figure 4.20(0), (2a), (2b), (4b), (4c) and (4d), and the connection rule for those cases is also uniquely defined (the parametrizable face rule). Note that Figure 4.20(2c) is impossible in the 2nd case because it is a $2D$ ambiguity. **Q.E.D.**

From the proof of the above lemma, the motivation of the alternating ambiguity is now clear. Let B be a box with known monotone direction in i and F be an i-face of B. It is easy to see that if B's neighboring box B' that shares part of F has a smaller width than B and $F' = B' \cap B$, then F' contains at most two vertices. It is also easy to see that for the end boxes B of a block, $UF_B \leq 1$.

We next describe the connection rule for the boxes B whose $UF_B = 2$: the **propagation rule**. Wlog, let $\mathcal{B}$ be the y-block containing B. We search the boxes in $y+$ direction to find the first box B' such that $UF_{B'} = 1$. Note that B' exists in $\mathcal{B}$ since the end boxes of a block have $UF \leq 1$. We push each box (from B to B') into a stack S_B.

The top of the stack is B', and we can use the matching rule to connect it. After connecting B' (now $UF_{B'} = 0$), we pop it from the stack. Then the top box B'' of the stack has $UF_{B''} = 1$. We keep connecting and popping the boxes until we reach B. Now we have $UF_B = 1$, we can use the matching rule to connect B. Figure 4.22(v)→(v')→(v") shows an example of using propagation rule to connect vertices. Similarly, we can define the arc connection rule for the boxes with known monotone directions in x or z.

Now we are ready to introduce the sub-phase 3.1 and 3.2 in $CONSTRUCT(T_2)$ in ¶33: $InitialConnect(T_2)$ and $ArcConnect(T_2)$.

```
$InitialConnect(T_2)$:
Let $Q$ be a priority queue containing all the candidate boxes in $T_2$
While ($Q$ is non-empty)
    $B \leftarrow Q.pop()$
    $UF_B \leftarrow 2$
    Connect the four faces which are parallel to $B$'s monotone direction
    using the parametrizable face rule. For each of the other two faces $F$
        If $F$ is an inactive face
            Decrease $UF_B$ by 1
        Else if $F = B \cap B'$ and $B'$ has a different monotone direction as $B$
            Connect $F$ using the parametrizable face rule
            Decrease $UF_B$ by 1
        Else if $F$ has less than 4 vertices
            Connect $F$ using the parametrizable face rule
            Decrease $UF_B$ by 1
```

After the $InitialConnect(T_2)$ sub-phase, UF_B should be equal to 0, 1 or 2 for each candidate box B in T_2. We next introduce the $ArcConnect(T_2)$ sub-phase:

```
$ArcConnect(T_2)$:
Let $Q$ be a priority queue containing all the candidate boxes in $T_2$
While ($Q$ is non-empty)
    $B \leftarrow Q.pop()$
    If $UF_B = 0$
        $B$ is fully connected, and there is nothing to do
    If $UF_B = 1$
        Use the matching rule to connect $B$
    If $UF_B = 2$
        Use the propagation rule to connect $B$
```

4.4 Correctness of Balanced Cxyz Algorithm

Let T be the octree produced by our Balanced Cxyz Algorithm. Similar to the correctness proof of the Regularized Cxyz Algorithm, we will first transform the input surface $S = f^{-1}(0)$ to another surface $\widetilde{S}$ which has some nice properties.

In the correctness proof of the Regularized Cxyz Algorithm, we separately defined the partial orders for loops and pairs of S in T. In the Balanced Cxyz Algorithm, we need to define the partial order for the combination of all loops and pairs. The reason is that a loop might be "blocked" by pairs (an example is shown in Figure 4.23(I)), and we need to remove the pairs first in order to remove the loop. Also, a pair might be "blocked" by loops ,as shown in Figure 4.23(II) (we do not have such problem in the Regularized Cxyz Algorithm since the loops are removed before pairs).

We define the new partial order for the set of $\mathcal{P}(S) \cup \mathcal{L}(S)$, where $\mathcal{P}(S)$ is the set of all pairs of $S \cap T$, and $\mathcal{L}(S)$ is the set of all loops of $S \cap T$ (see ¶27 and ¶29). The partial order between loops and between pairs are the same as the partial order defined

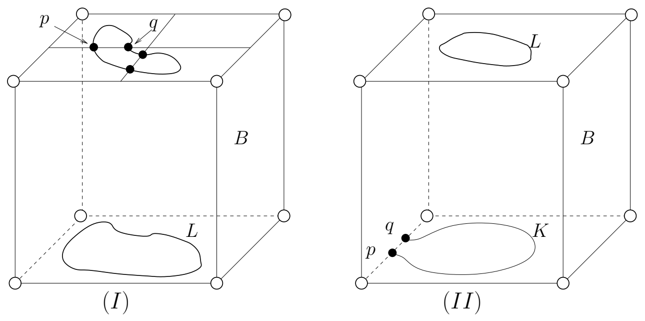


Figure 4.23: Partial order between a loop and a pair.

in the Regularized Cxyz Algorithm: let $\prec_P \subseteq \mathcal{P}(S) \times \mathcal{P}(S)$ be the partial order defined for pairs, and $\prec_L \subseteq \mathcal{L}(S) \times \mathcal{L}(S)$ be the partial order defined for loops. We need to define a partial order on the set $\mathcal{P}(S) \cup \mathcal{L}(S)$.

Let B be a box with monotone direction y. Let L be a loop on the bottom face of B and $\{p, q\}$ be a pair on the top face of B. If the y-projection of $\{p, q\}$ is contained within the y-projection of L, we say $\{p, q\} \prec L$ (as shown in Figure 4.23(I)). In order to remove L, we need to remove $\{p, q\}$ first. We can similarly define such relations in x and z directions. Let $\prec_{PL} \subseteq \mathcal{P}(S) \times \mathcal{L}(S)$ be all the relations so defined. Similarly, we can define $\prec_{LP} \subseteq \mathcal{L}(S) \times \mathcal{P}(S)$: let $\{p, q\}$ be a pair, and K be a semi-loop whose base is $[p, q]$. If there exist a loop L which lies in the same box B as K, and the i-projection of L (for some $i \in \{x, y, z\}$) lies in the interior of the i-projection of K, we say $L \prec \{p, q\}$ (as shown in Figure 4.23(II)).

In the Regularized Cxyz Algorithm, we removed all loops before we remove pairs. But in the Balanced Cxyz Algorithm, we are forced to intermix pair removal with loop removal because of the relations in $\prec_{PL}$ and $\prec_{LP}$. However, if we look at the relation $\prec_P \cup \prec_L \cup \prec_{PL} \cup \prec_{LP}$, we do not obtain a partial order on $\mathcal{P}(S) \cup \mathcal{L}(S)$ (see Figure 4.24: the green points form pairs, and the arrows show the monotone direction of the boxes. It is possible that $L \prec P_{11} \prec \ldots \prec P_1 \prec L$, which forms a loop).

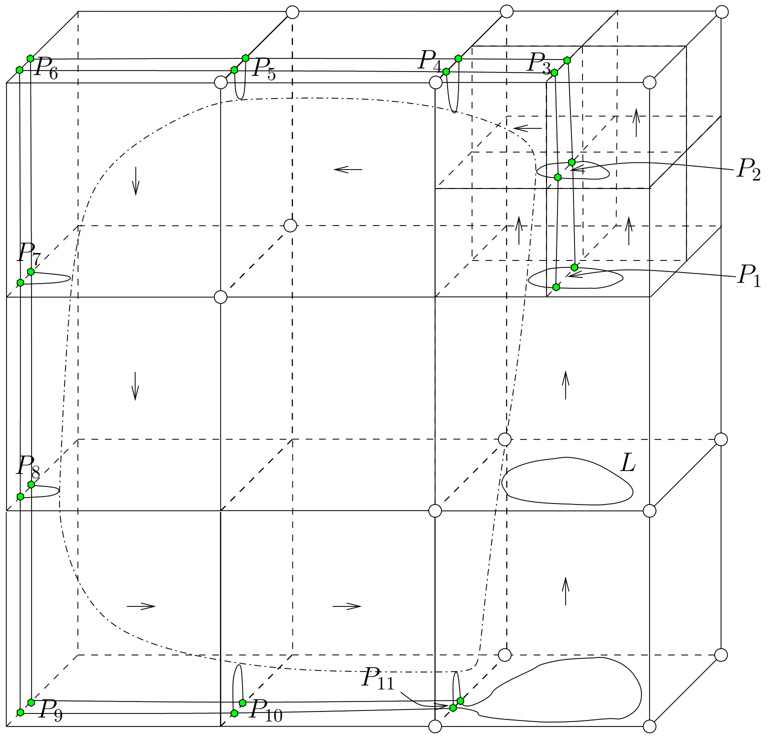


Figure 4.24: Example of a loop in $\prec_P \cup \prec_L \cup \prec_{PL} \cup \prec_{LP}$.

Our solution is to define a partial order based only on $\prec_{Bal} := \prec_P \cup \prec_L \cup \prec_{PL}$. This is clearly a partial order on $\mathcal{P}(S) \cup \mathcal{L}(S)$.

LEMMA 36 (DAG). *The partial order relationship $\prec_{Bal}$ forms a DAG G_p where the pairs and loops are the nodes of G_p and the partial order relations are the (directed) edges of G_p.*

Why is this a solution? As usual, we plan to inductively remove elements from $\mathcal{P}(S) \cup \mathcal{L}(S)$, which are minimal relative to $\prec_{Bal}$. The possible complication arises when we want to remove a pair $\{p, q\}$ where $L \prec_{LP} \{p, q\}$ for some loop L. It turns out, we can remove $\{p, q\}$ without first removing L provided that we generalize our previous base removal operation as follows: to remove a pair $\{p, q\}$, we will remove all semi-loops K whose base is $[p, q]$. There are two possible situations: (A) If there

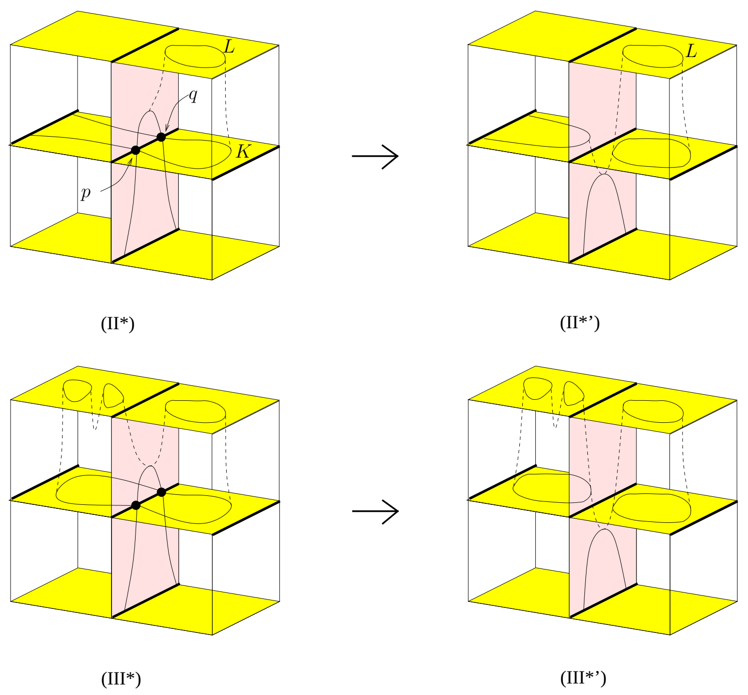


Figure 4.25: Universal Base Removal Operations.

is a loop L s.t. $L \prec_{LP} \{p,q\}$, then we know that $[p,q]$ is the base of a semi-loop K where the i-projection of L (for some $i \in \{x,y,z\}$) lies in the interior of K. In this case, we transform the surface S so that $\{p,q\}$ is removed from $\mathcal{P}(S)$, and a new loop K' appears in $\mathcal{L}(S)$. And moreover, $L \prec K' \in \prec_L$. See Figure 4.25 $(II*) \rightarrow (II*')$ and $(III*) \rightarrow (III*')$ for the illustration of this operation. Note that there might be more than one such loops L. (B) If no such loop L exists, then the operation is defined as in the Regularized Cxyz Algorithm. Similar to the proof of Lemma 30, we can prove that those two generalized operations also preserve the surface monotonicity of S in T. Based on the correctness analysis in the Regularized Cxyz Algorithm, we have the following (similar) theorem for the Balanced Cxyz Algorithm:

THEOREM 37. *Let T be the octree produced by our Balanced Cxyz Algorithm. There $\exists \widetilde{S}$, s.t.*

(1) $\widetilde{S} \simeq S(\textbf{mod}\, R(T))$.

(2) $\widetilde{S}$ *is compatible with* S *respect to* T.

(3) $\widetilde{S}$ *intersects* T *cleanly.*

(4) $\widetilde{S}$ *preserves the monotonicity of* S *within each candidate box of* T.

Proof. The correctness of this theorem follows from the analysis of the face cleaning and edge cleaning processes. **Q.E.D.**

In the Regularized Cxyz Algorithm, we proved Lemma 27. We have a similar result in the balanced algorithm:

LEMMA 38 (NO HOLES 2). *Let* $\widetilde{S}$ *be the surface described in Theorem 37 and* $\mathcal{B}$ *be a connected subset of an* i*-block. Let* C *be a closed curve which is the intersection of* $\widetilde{S}$ *with* $\partial(\cup\mathcal{B}_{B\in\mathcal{B}})$*. Let* $P \subseteq \widetilde{S} \cap \mathcal{B}$ *be a surface patch in* $\mathcal{B}$ *(i.e.,* P *is a connected component of* $\widetilde{S} \cap \mathcal{B}$*). If* $C \subseteq \partial P$*, then* $\partial P = C$*. In other words,* P *is topologically a disc.*

Proof. The correctness of this lemma follows from the facts that $\widetilde{S} \cap \mathcal{B}$ is monotone in $\mathcal{B}$, and $\widetilde{S}$ intersects $\mathcal{B}$ cleanly. The proof is similar to the proof of Lemma 26. **Q.E.D.**

From Lemma 38, it is easy to see that $\widetilde{S} \cap B$ is a set of topological discs for each candidate box B.

THEOREM 39. *The mesh* G *constructed by our Balanced Cxyz Algorithm is isotopic to* $\widetilde{S}$ *within each* i*-block* $\mathcal{B}$ *of* T*. In other words,* $G \simeq \widetilde{S} \simeq S(\textbf{mod}\, R(T))$.

Proof. From Theorem 37, it is easy to see that $\widetilde{S}$ intersects the boundary of $\mathcal{B}$ cleanly. Our construction rule guarantees that $G \cap \partial(\cup\mathcal{B})$ “agrees” with $\widetilde{S} \cap \partial(\cup\mathcal{B})$. And each connected component of $G \cap \mathcal{B}$ is a topological disc. So based on Lemma 38, we have $G \cap \mathcal{B} \simeq \widetilde{S} \cap \mathcal{B}$. **Q.E.D.**

4.5 Rectangular Cxyz Algorithm

As in the Cxy algorithm, the ability to have partial splits can be highly advantageous. In $3D$, this means a box can be half- or quarter-split. Our subdivision boxes will now have various aspect ratios, where the **aspect ratio** of a box is defined to be the ratio of the length of the longest edge to the length of the shortest. In order to prove that such an algorithm will halt, it is necessary to assume some priori bound $\rho \geqslant 1$ on the aspect ratio of any subdivision box. In particular, we are not allowed to do those partial splits that will produce a child with aspect ratio $> \rho$. Our method for deciding how to do partial splits is a straightforward generalization of the $2D$ case. We will assume some fixed convention[8] for labeling the 8 orthants of the coordinate system.

We modify the subdivision phase as follows: For each on-box B in the queue, we must decide how to tag it, or how to to split and tag its children. This is accomplished by a new subdivision phase, which amounts to checking the following three levels of

[8] Unlike the $2D$ case, there seems to be no universally accepted convention for this. See, e.g., http://godplaysdice.blogspot.com/2007/09/convention-for-quadrantoctantorthant.html. We will use the gray code to label successive orthants, starting from $1 = 000, 2 = 001, 3 = 011, 4 = 010, 5 = 110, 6 = 111, 7 = 101, 8 = 100$.

conditions (in this order):

$$\left.\begin{aligned}
&L_0 : \\
&C_{out} : C_0(B) \\
&C_{in} : C_{xyz}(B) \\
&L_1 : \\
&C_{out} : C_0(B_{1234}), C_0(B_{5678}), C_0(B_{1278}), C_0(B_{3456}), C_0(B_{1458}), C_0(B_{2367}) \\
&C_{in} : C_{xyz}(B_{1234}), C_{xyz}(B_{5678}), C_{xyz}(B_{1278}), C_{xyz}(B_{3456}), C_{xyz}(B_{1458}), C_{xyz}(B_{2367}) \\
&L_2 : \\
&C_{out} : C_0(B_{12}), C_0(B_{34}), C_0(B_{56}), C_0(B_{78}), C_0(B_{14}), C_0(B_{23}), \\
&\qquad C_0(B_{67}), C_0(B_{58}), C_0(B_{18}), C_0(B_{27}), C_0(B_{36}), C_0(B_{45}) \\
&C_{in} : C_{xyz}(B_{12}), C_{xyz}(B_{34}), C_{xyz}(B_{56}), C_{xyz}(B_{78}), C_{xyz}(B_{14}), C_{xyz}(B_{23}), \\
&\qquad C_{xyz}(B_{67}), C_{xyz}(B_{58}), C_{xyz}(B_{18}), C_{xyz}(B_{27}), C_{xyz}(B_{36}), C_{xyz}(B_{45})
\end{aligned}\right\} \tag{4.8}$$

We stop at the first verified condition. If a condition in L_0 is verified, we tag B as an candidate or discarded box, accordingly. If a condition in L_1 (L_2) is verified, we do a half-split (quarter-split) of B to produce the child that satisfies that condition. That child is tagged as discarded or candidate. The other children are pushed back into the queue. Finally, if no condition is verified, we do a full-split and push the children into the queue.

The balancing phase of the Rectangular Cxyz algorithm is slightly different from the Balanced Cxyz algorithm. In the splitting sub-phase, we do x-balance first, then balance along the y- and z-direction accordingly (see the definition of i-balance in 4.3):

```
$Split(T_1)$:
    $T_{1.1}$=$Split_X(T_1)$
    $T_{1.2}$=$Split_Y(T_{1.1})$
    $T_1'$=$Split_Z(T_{1.2})$
```

We define $w_i(B)$ to be the i-width of B ($i \in \{x, y, z\}$), and $r_x(B) = (max(w_y(B), w_z(B)))/w_x(B)$ (similarly for $r_y(B)$ and $r_z(B)$). Note that $r_i(B)(i = x, y, z)$ might exceed the bounding aspect ratio ρ in the balancing phase. Also note that we only use ρ to guarantee the termination of the subdivision phase. The termination of the balancing phase is guaranteed by the fact that we never induce a box B with $w_i(B)(i = x, y, z)$ smaller than the minimum i-width in T_1.

After the splitting sub-phase, there are still problems preventing us from adding vertices correctly: Let A be a box, and B be one of A's right neighbors. If $w_y(A) = 2 * w_y(B)$ and $w_z(B) = 2 * w_z(A)$ (as shown in Figure 4.26), and if point p_3 and p_2 have different signs, there is no edge to add a vertex at their midpoint. A vertex will be added at p_2 - the midpoint of (p_1, p_3) when we process the box A or D, and the vertex is at the corner of the box C. There is a simple way to resolve this problem: if we find this kind of situation, we half-split B (or A). We have an additional sub-phase for adjusting such boxes (let Q_{xy} be a priority queue which sorts the boxes by their x-width and then the y-width):

```
$Adjust(T_1')$:
    $T_{1.1}'$=$Adjust_X(T_1')$
    $T_{1.2}'$=$Adjust_Y(T_{1.1}')$
    $T_1''$=$Adjust_Z(T_{1.2}')$
```

Where $Adjust_i(i \in \{x, y, z\})$ is defined as the following procedure (wlog, $i = Z$):

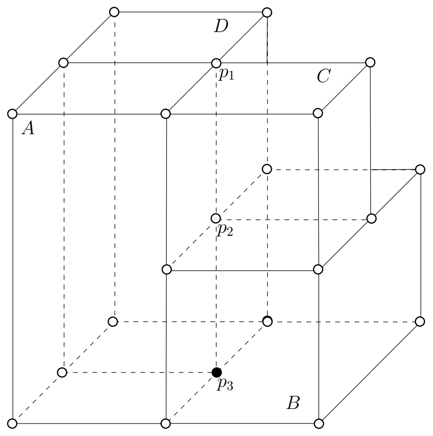


Figure 4.26: Problem of the balancing phase in Rectangular Cxyz algorithm.

```
$Adjust_Z(T'_{1.2})$:
    $Q_{xy}$ is a priority queue containing all the candidate boxes in $T'_1$,
    while $Q_{xy}$ is non-empty:
        $B \leftarrow Q_{xy}.remove()$
        For each candidate box $B'$ that is a $z$-neighbor of $B$
            If $w_x(B') > w_x(B)$ and $w_y(B') < w_y(B)$,
                $x$ split $B'$
              For each candidate box $B''$ that is a child of $B'$
                Insert $B''$ into $Q_{xy}$
    Return the extended octree $T''_1$
```

The adjust sub-phase might introduce new unbalanced cases, so we need to loop over the adjust and split sub-phases until there is no further split:

```
$Split\&Adjust(T_1)$:
    $T_1'$=Split($T_1$)
    Do
        $T_1''$=Adjust($T_1'$)
        $T_1'$=Split($T_1''$)
    While there are splits
    Return $T_2 = T_1'$
```

The the complexity of the split and adjust sub-phase might seem overwhelming. But in the experimental result, the number of splits reduces very fast, and the whole sub-phase can finish faster than the Balanced Cxyz algorithm. On other words, the number of boxes after Split&Adjust sub-phase can be less than the number of boxes after the split sub-phase in the Balanced Cxyz algorithm.

The disambiguation sub-phase is slightly different from the Balanced Cxyz Algorithm too. We need to ensure that the disambiguation phase does not produce boxes with smaller i-width than the minimum i-width in the octree T. For the $2D$ ambiguous box, we do a half split of the box to separate the two vertices which cause the ambiguity. For the $3D$ ambiguous boxes, we do a quarterly split of the box which splits the face (whose interior contains four vertices) into four children. For the alternating ambiguous box B, let $F = B \cap B'$ be the face that causes the ambiguity (wlog, let F be a x-face). We split B to "fit" F (i.e., the children of B will have the same y- and z-width as B'). Note that we might do half- or quarter-split on B in the directions which are perpendicular to F.

The construction phase of the Rectangular Cxyz Algorithm is similar to the Balanced Cxyz Algorithm, and so does the correctness proof.

4.6 Implementation and Software

Our algorithms are implemented in `Java` on the Eclipse Platform. See 3.7 for the hardware configuration. The code for $2D$ meshing is available for download at `http://cs.nyu.edu/exact/papers/cxy/`, and the code for $3D$ meshing is available for download at `http://cs.nyu.edu/exact/papers/cxyz/`.

Note that this implementation is based on machine arithmetic. Our implementation is exact (in particular, there is no numerical rounding error) as long as there is no underflow or overflow. This is because the only arithmetic operations we use are ring operations and divide by 2. The limitation of machine precision is that, for high degree polynomials, the code might fail because of under/overflows. Cxy algorithm has been transformed to `C++` based exact computational library `Core Library` by Shuxing Lu, and improved by Narayan Kamath. We plan to convert other `Java` codes to `C++` for distribution with our open source `Core Library`.

We use the default `Java` heap memory 256MB (some runs result in OutOfMemoryError (OME)). 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 implement Snyder's algorithm in $3D$ since it is relatively complicated.

4.7 Experimental Results

We report some encouraging experimental results. Table 4.1 lists 11 examples of our tests. Table 4.2 compares the number of boxes and the running time among Cxyz, PV, and Rect-n $(n = 2, 4, 8, 16, 32)$. The percentages represents the relative running times, using Cxyz as 100%. Figure 1.1, Figure 4.28, Figure 4.29, Figure 4.30, Figure 4.31, Figure 4.32 and Figure 4.33 illustrates the meshes for Eg.1 to Eg.7 in Table 1 respectively, using Cxyze, PV, Cxyz and Rect-n, where n is selected in a way that Rect-n is the fastest among all Rect algorithms.

Table 4.1: Equations and input boxes of examples

#	Curve name	Equation $f(x, y, z) = 0$	Original Box
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)	eclipse	$x^2 + 10^ny^2 + 10^nz^2 - 1$	$[(-7, -7, -7), (8, 8, 8)]$

(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 milliseconds for Eg8b(4), and 172 milliseconds for Eg8b(6). Note that the only exception is Eg8a, 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 computational cost for the C_1 predicate is bigger than the C_{xyz} predicate.

(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 running time; and in Eg8b(6), Rect-2 takes 12.8% of Cxyz's running time. The input surface for these examples are very long and thin, in which Rect algorithm can take advantage of various aspect ratios. The results also show that although Rect produces less boxes than Cxyz in all examples but Eg8b(2), the running time of Rect is not always faster than the Cxyz (especially when

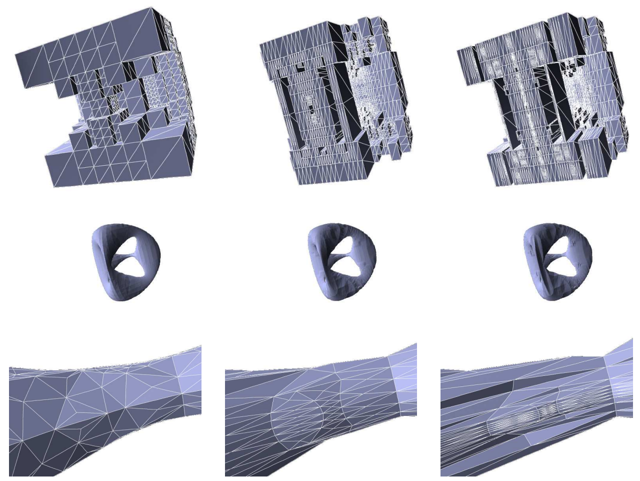
the input surface is squarish, like Eg2). This is because Rect needs to spend more time to check the criteria before splitting a box, and needs to process each box in three directions in Rect.

(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 causes 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. Figure 4.27 shows the resulting boxes, meshes, and details by running Cxyz, Rect-8, and Rect-32 on Eg2.

(4) Figure 4.34 illustrates an example that our algorithms preserve the topology: the first row of Figure 4.34 shows the approximations of Eg4 using Rect-n $(n = 2, 4, 8, 16, 32)$ algorithm. It is not clear that the topology of the resulting meshes is the same by looking at the squared area. By zooming in the squared area (see the second row of Figure 4.34), We could see that the topology is preserved in the squared area of the meshes.

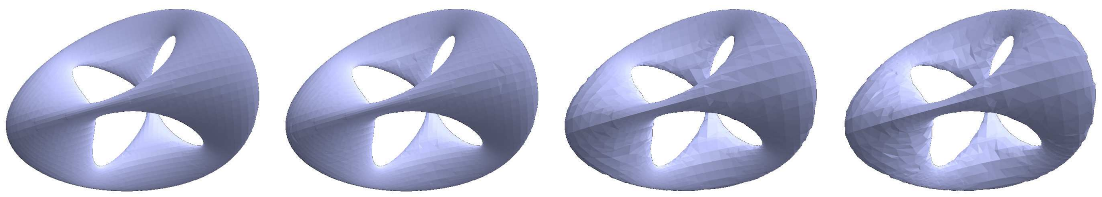
Table 4.2: Cxyz vs. PV vs. Rect-n

Box/Time (ms)/%	Cxyz	PV	Rect-2	Rect-4	Rect-8	Rect-16	Rect-32
Eg1	2584/391	5104/718/184%	1096/579/148%	1304/656/168%	1710/781/200%	2081/922/236%	2653/1125/288%
Eg2	26104/4516	106072/15765/349%	13400/7360/163%	19847/10672/236%	25513/13656/302%	30880/16797/372%	36931/20360/451%
Eg3	35792/3437	35792/3843/112%	12056/2812/82%	6264/1625/47%	3328/953/28%	2000/578/17%	1088/407/12%
Eg4	80662/10282	$OME_{>90sec.}$	43977/17875/174%	32836/13313/129%	27577/10766/105%	29143/11797/115%	26700/10594/103%
Eg5	134163/17187	$OME_{>90sec.}$	64617/35156/205%	37237/14703/86%	30730/12188/71%	27612/11187/65%	26221/10532/61%
Eg6	31144/4046	99436/11985/296%	13688/5421/134%	16348/6922/171%	19332/8422/208%	21698/10328/255%	23827/11469/283%
Eg7	1688/328	2920/421/128%	796/359/109%	836/390/119%	1028/422/129%	1244/453/138%	1652/578/176%
Eg8a	400/94	400/94/100%	176/125/133%	200/140/149%	232/156/166%	272/156/166%	320/172/183%
Eg8b(2)	274/125	2164/250/200%	149/109/87%	154/109/87%	197/125/100%	225/140/112%	279/140/112%
Eg8b(4)	1247/203	22121/1531/754%	345/141/69%	418/141/69%	484/156/77%	551/172/85%	658/203/100%
Eg8b(6)	15226/1343	$OME_{>70sec.}$	696/172/13%	733/187/14%	886/203/15%	952/203/15%	1129/219/16%



(a) Cxyz (b) Rect-8 (c) Rect-32

Figure 4.27: Boxes, meshes, and details of Eg2 using Cxyz, Rect-8 and Rect-32. Note that the triangles are elongated as the maximum aspect ratio increases.

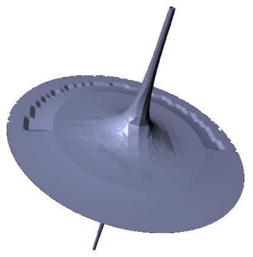


(a) Cxyze (b) PV (c) Cxyz (d) Rect-2

Figure 4.28: Approximation of Eg2: chair $f(x, y, z) = (x^2 + y^2 + z^2 - 23.75)^2 - 0.8((z-5)^2 - 2x^2)((z+5)^2 - 2y^2) = 0$.

(a) Cxyze (b) PV (c) Cxyz (d) Rect-32

Figure 4.29: Approximation of Eg3: quartic cylinder $f(x, y, z) = y^2x^2 + y^2z^2 + 0.01x^2 + 0.01z^2 - 0.01 = 0$.


(a) Cxyz

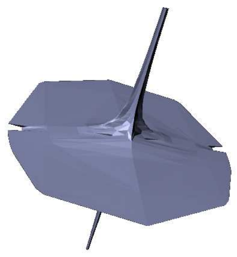
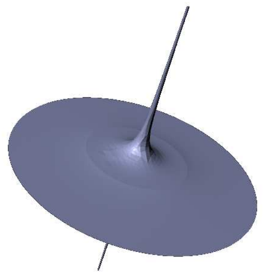

(b) Rect-32

Figure 4.30: Approximation of Eg4: quartic cylinder 1 $f(x,y,z) = y^2(x-1)^2 + y^2(z-1)^2 + 0.01(x-1)^2 + 0.01(z-1)^2 - 0.2002 = 0$.


(a) Cxyz

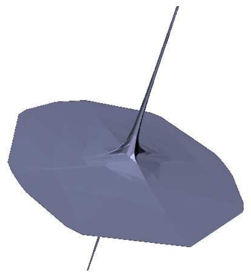
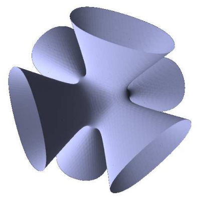
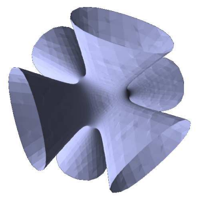
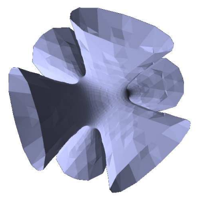

(b) Rect-32

Figure 4.31: Approximation of Eg5: quartic cylinder 2 $f(x,y,z) = y^2(x-1)^2 + y^2(z-1)^2 + 0.01(x-1)^2 + 0.01(z-1)^2 - 0.1002 = 0$.


(a) Cxyze


(b) PV


(c) Cxyz

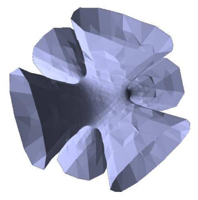

(d) Rect-2

Figure 4.32: Approximation of Eg6: shrek $f(x,y,z) = -x^4 - y^4 - z^4 + 4(x^2 + y^2z^2 + y^2 + z^2x^2 + z^2 + x^2y^2) - 20.7846xyz - 10 = 0$.

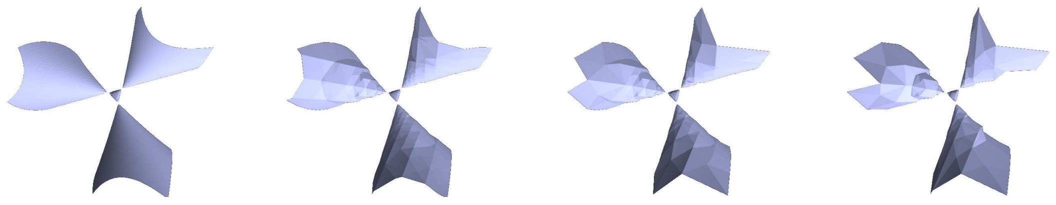
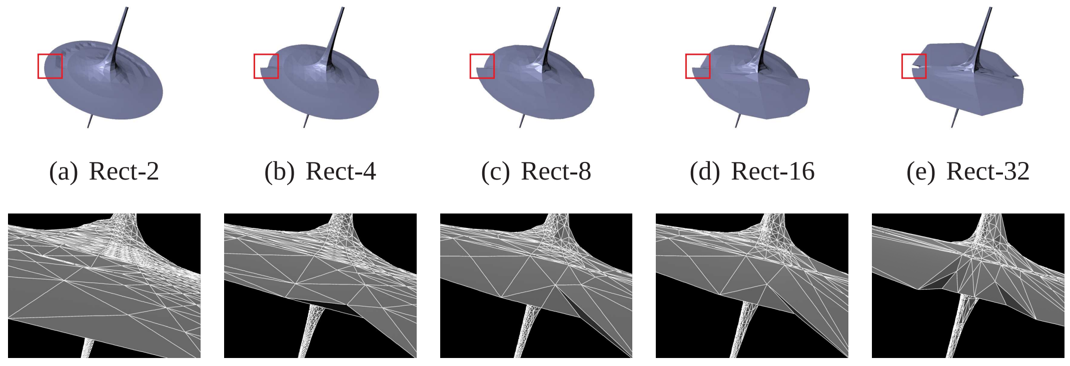

(a) Cxyze (b) PV (c) Cxyz (d) Rect-2

Figure 4.33: Approximation of Eg7: tritrumpet $f(x, y, z) = 8z^2 + 6xy^2 - 2x^3 + 3x^2 + 3y^2 - 0.9 = 0$.


(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

Figure 4.34: First row(a)-(e): Approximations of a quartic cylinder 1 $f(x, y, z) = y^2(x-1)^2 + y^2(z-1)^2 + 0.01(x-1)^2 + 0.01(z-1)^2 - 0.2002 = 0$ using Rect-n $(n = 2, 4, 8, 16, 32)$. Second row(a)-(e): Topology preservation in the squared area of the approximations.

Chapter 5

Conclusion and Future Works

This thesis introduces a new family of algorithms for isotopic approximation of implicit curves and surfaces that is provably correct, simple, efficient, and easy to implement exactly. The basic idea is to exploit parametrizability (like Snyder) and nonlocal isotopy (like Plantinga and Vegter). We also extend these ideas to subdivision boxes of bounded aspect ratio, and mesh construction within irregular geometries. In $2D$, our experimental results which compare four algorithms (PV, Snyder, Balanced Cxy, and Rectangular Cxy) show that our Balanced Cxy Algorithm is faster than Snyder and PV most of the time, and Rectangular Cxy Algorithm is the best in all tests and often exhibits great speedup. In $3D$, our experimental results which compare three algorithms (PV, Balanced Cxyz, and Rectangular Cxyz) show that our Balanced 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.

Future work includes extensions to higher dimensions, effective treatment of singularity using numerical methods, more efficient algorithm to achieve geometric accuracy (by exploiting parametrizability and boundary information of each box), complexity analysis of subdivision algorithms, and converting `Java` codes of Cxyz algorithms to `C++` for distribution with our open source `Core Library`.

Bibliography

[1] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. In *SCG '98 Proceedings of the fourteenth annual symposium on Computational geometry*, pages 39–48, 1998.

[2] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics. Springer, 2003.

[3] J.-D. Boissonnat, D. Cohen-Steiner, B. Mourrain, G. Rote, and G. Vegter. Meshing of surfaces. In Boissonnat and Teillaud [8]. Chapter 5.

[4] J.-D. Boissonnat, D. Cohen-Steiner, and G. Vegter. Isotopic implicit surfaces meshing. In *ACM Symp. Theory of Comput.*, pages 301–309, 2004.

[5] J.-D. Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *SGP '03 Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 9–18, 2003.

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

[7] J.-D. Boissonnat and S. Oudot. Provably good sampling and meshing of Lipschitz surfaces. In *Proc. 22nd ACM Symp. on Comp. Geometry*, pages 337–346, 2006. Sedona, Arizona.

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

[9] M. Burr, S. Choi, B. Galehouse, and C. Yap. Complete subdivision algorithms, II: Isotopic meshing of singular algebraic curves. In *Proc. Int'l Symp. Symbolic and Algebraic Computation (ISSAC'08)*, pages 87–94, 2008. Hagenberg, Austria. Jul 20-23, 2008. Accepted for Special Issue of ISSAC 2008 in JSC. Also, in arXiv:1102.5463.

[10] M. Burr, F. Krahmer, and C. Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. *Electronic Colloquium on Computational Complexity (ECCC)*, TR09(136), December 2009.

[11] M. Burr, V. Sharma, and C. Yap. Evaluation-based root isolation, 2011. In preparation.

[12] J.-S. Cheng, X.-S. Gao, and C.-K. Yap. Complete numerical isolation of real zeros in zero-dimensional triangular systems. *J. Symbolic Computation*, 44(7):768–785, 2009. Special Issue of JSC based on ISSAC 2007. Available online at JSC.

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

[14] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proc. 9th ACM Symp. on Comp. Geometry*, pages 274–280, 1993. San Diego, California, United States.

[15] A. Eigenwillig. *Real Root Isolation for Exact and Approximate Polynomials Using Descartes Rule of Signs*. Ph.D. thesis, University of Saarland, Saarbruecken, Germany, May 2008.

[16] A. Eigenwillig, L. Kettner, E. Schmer, and N. Wolpert. Complete, exact, and efficient computations with cubic curves. In *20th ACM Symp. on Comp. Geometry*, pages 409 – 418, 2004. Brooklyn, New York, USA, Jun 08 – 11.

[17] B. 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/`.

[18] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[19] M. W. Hirsch. *Differential Topology*. Springer-Verlag, 1976.

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

[21] T. Ju, F. Losasso, S. Schaefer, and J.Warren. Dual contouring of hermite data. In *SIGGRAPH '02 Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM, 2002.

[22] N. Kamath, I. Voiculescu, and C. 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.

[23] L. Lin and C. Yap. Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. *Discrete & Computational Geometry*, 45(4):760–795, 2011. Special Issue: 25th Annual Symposium on Computational Geometry SOCG '09.

[24] L. Lin, C. Yap, and J. Yu. Adaptive isotopic approximation of nonsingular surfaces, 2011. In preparation.

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

[26] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang. Comparison of interval methods for plotting algebraic curves. *Computer Aided Geometric Design*, 19(7):553–587, 2002.

[27] D. P. Mitchell. Robust ray intersection with interval arithmetic. In *Graphics Interface'90*, pages 68–74, 1990.

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

[29] B. Mourrain and J.-P. Técourt. Isotopic meshing of a real algebraic surface. Technical Report RR-5508, INRIA, Sophia-Antipolis, France, Feb. 2005. Also, electronic proceedings, MEGA 2005.

[30] T. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879, 2006.

[31] S. Plantinga. *Certified Algorithms for Implicit Surfaces*. Ph.D. thesis, Groningen University, Institute for Mathematics and Computing Science, Groningen , Netherlands, Dec. 2006.

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

[33] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[34] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Horwood Publishing Limited, Chichester, West Sussex, UK, 1984.

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

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

[37] T. Sakkalis and T. J. Peters. Ambient isotopic approximations for surface reconstruction and interval solids. In *SM '03 Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 176–184. ACM, 2003.

[38] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. In *PG '04 Proc. 12th Pacific Conf. Computer Graphics and Applications*, pages 70–76, 2004.

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

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

[41] V. Sharma. Complexity of real root isolation using continued fractions. *Theor. Computer Science*, 409(2), 2008. Also: proceedings ISSAC'07.

[42] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based decimation of marching cubes surfaces. In *IEEE Visualization '96*, pages 335–344, 1996.

[43] J. M. Snyder. *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design using Interval Analysis*. Academic Press, 1992.

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

[45] B. T. Stander and J. 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.

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

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

[48] G. Varadhan, S. Krishnan, Y. J. Kim, S. Diggavi, and D. Manocha. Efficient max-norm distance computation and reliable voxelization. In *SGP '03 Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 116–126, 2003.

[49] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *SGP '04 Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 235–244, 2004.

[50] C. K. Yap. Symbolic treatment of geometric degeneracies. *J. Symbolic Computation*, 10:349–370, 1990.

[51] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. Chapman & Hall/CRC, Boca Raton, FL, 2nd edition, 2004.

[52] C. K. Yap. Complete subdivision algorithms, I: Intersection of Bezier curves. In *22nd ACM Symp. on Comp. Geometry*, pages 217–226, July 2006.

[53] C. K. Yap. In praise of numerical computation. In S. Albers, H. Alt, and S. Näher, editors, *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 308–407. Springer-Verlag, 2009. Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday.

[54] C. K. Yap and J. Yu. Foundations of exact rounding. In S. Das and R. Uehara, editors, *Proc. WALCOM 2009*, volume 5431 of *Lecture Notes in Computer Science*,

pages 15–31, Heidelberg, 2009. Springer-Verlag. Invited talk, 3rd Workshop on Algorithms and Computation, Kolkata, India.