Adaptive Isotopic Approximation of Nonsingular Curves: the Parametrizability and Nonlocal Isotopy Approach[☆]

Long Lin and Chee Yap

Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012 USA

Abstract

We consider domain subdivision algorithms for computing isotopic approximations of nonsingular curves represented implicitly by an equation $f(X,Y) = 0$. Two algorithms in this area are from Snyder (1992) and Plantinga & Vegter (2004). We introduce a new algorithm that combines the advantages of these two algorithms: like Snyder, we use the parametrizability criterion for subdivision, and like Plantinga & Vegter we exploit non-local isotopy. We further extend our algorithm in two important and practical directions: first, we allow subdivision cells to be rectangles with arbitrary but bounded aspect ratios. Second, we extend the input domains to be regions R_0 with arbitrary geometry and which might not be simply connected. Our algorithm halts as long as the curve has no singularities in the region, and intersects the boundary of R_0 transversally. Our algorithm is practical and easy to implement exactly. We report some very encouraging experimental results, showing that our algorithms can be much more efficient than the algorithms of Plantinga & Vegter and Snyder.

Key words: Meshing, Curve Approximation, Isotopy, Parametrizability, Subdivision Algorithms, Topological Correctness, Exact Algorithms.

[☆]This work is supported by NSF Grant CCF-0728977.
Email address: `{llin,yap}@cs.nyu.edu` (Long Lin and Chee Yap)

1. Introduction

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 [5] for an algorithmic perspective on meshing problems. We focus on curves, and in this case the "mesh" is just a (planar) straightline graph G (or PSLG, see [19]). Our problem is this: given a region $R_0 \subseteq \mathbb{R}^2$ of interest, an error bound $\varepsilon > 0$, a curve S implicitly represented by an equation $f(X, Y) = 0$, to find a piecewise linear ε-approximation G of $S \cap R_0$.

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. In recent years, the topological correctness is understood to mean that the approximate curve G should be isotopic to $S \cap R_0$; see [2] 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. The first stage is more challenging and draws most of the attention.

There are three general approaches to meshing problems: algebraic, geometric or numeric. **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) [1] but such methods are too inefficient, even in the plane. This has led to much interest in numerical algebraic methods (e.g., [13]). But for special cases such as quadric surfaces [22] or cubic curves [11], efficient algebraic algorithms have been devised. **Geometric approaches** exploit geometric properties such as Morse theory [25, 3] or Delaunay triangulations [10]. These geometric properties are encoded into the primitives used by the algorithm. Typical primitives include the orientation predicates or ray shooting operations. **Numeric approaches** focus on approximation and numerical primitives such as function evaluation [14, 18]. 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 [15]). These algorithms are often called "curve tracing algorithms". See Ratschek and Rokne [21] 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 two simple reasons: they are efficient and easy to implement. Our overall goal is to address the above shortcomings of numerical approaches while retaining their advantages. Clearly, some algorithms are best viewed as hybrids of these approaches. All three approaches are exemplified in the survey [2].

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 [10, 4, 2]. 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, these may no longer lie on the surface. This gives rise to the well-known "implementation gap" concerns of computational geometry [26]: nonrobustness, degeneracies, approximation, etc. In contrast, the subdivision methods studied in this paper 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 geometric algorithms.

¶*1. Recent Progress in Subdivision Algorithms.* In this paper, we focus on algorithms based on domain[1] subdivisions methods. Figure 1 illustrates the output of four such algorithms on the input curve $f(X, Y) =$

[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.

$X^2(1-X)(1+X)-Y^2+0.01=0$. These output are from our implementation of the algorithms of Snyder and PV (PV), and two new algorithms of this paper (Balanced Cxy, Rectangular Cxy).

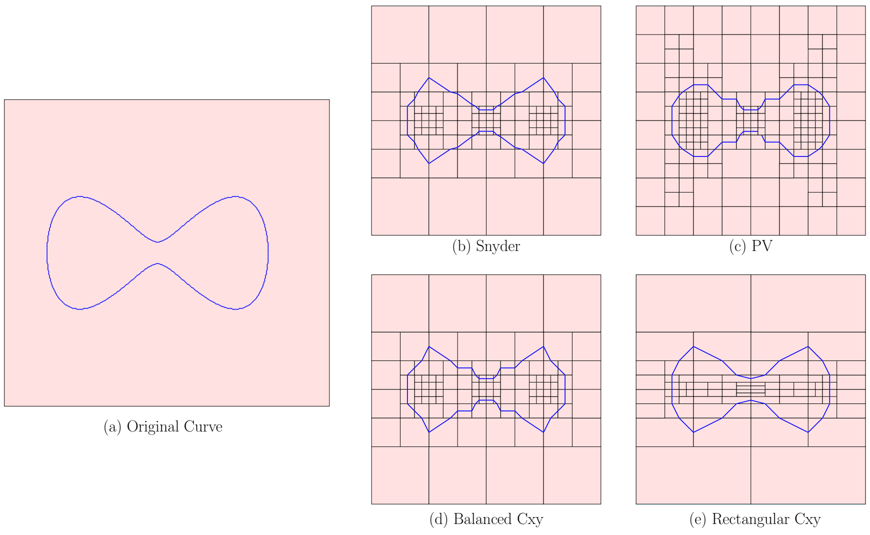


Figure 1: Domain Subdivision Approaches to approximating the curve $f(X,Y)=X^2(1-X)(1+X)-Y^2+0.01=0$: comparison of four algorithms.

We view subdivision algorithms as falling under the numeric approaches (see below for the numerical computational model). The simplest form of domain subdivision use only axes-parallel boxes (e.g., in bisection searches and Marching Cube [14]). According to a taxonomy of meshing algorithms in [2], this form is called "cube-based scaffolding". 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 [23] 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 [25] reveals such gaps.

The direct precursors for our work are the subdivision algorithms of Plantinga & Vegter [18] and Snyder [24]. Both algorithms are based on interval arithmetic [16] and the ability to evaluate the exact sign of function $f(X,Y)$ at bigfloat values. For a large class of functions $f(X,Y)$, 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 [12] has a subdivision algorithm for meshing surfaces in any dimension, but like Snyder, he requires recursive meshing of the boundary. All these algorithms are also related to the SCCI-hybrid algorithm for curve tracing by Ratschek and Rokne [21].

Both Plantinga & Vegter and Snyder assume the input curves and surfaces are nonsingular. Only recently has numerical subdivision algorithms been designed which can work with non-singularities and degeneracies. In [27], we gave a Bezier curve intersection algorithm that is correct even in the presence of tangential intersection. Guaranteed numerical subdivision techniques for approximating curves with isolated singularities were given in [6]. The paper also extended the algorithm of Plantinga & Vegter to domains with irregular geometry. In [8], we studied the 1-D versions of the Plantinga & Vegter algorithm, seen as a new class of numerical (non-algebraic) root isolation algorithms, and extending it to treat singularities (i.e., multiple zeros). A key attraction of subdivision algorithms is their adaptive complexity. But current techniques in algorithmic analysis cannot quantify this adaptivity. In [7], we introduced continuous and algebraic amortization techniques, resulting in one of the first adaptive analysis of subdivision algorithms.

¶2. *Contributions of this Paper.* The present paper represents a more practical contribution to the preceding series of development: we break no new ground in terms of doing something we couldn't do before. On the other hand, we introduce implementable ideas that make subdivision algorithms more useful than ever, while remaining theoretically sound.

Our main contribution is a new approach, and a corresponding new meshing algorithm, that combines the relative advantages of Snyder and Plantinga & Vegter: we retain the weaker C_{xy}-predicate of Snyder, but like Plantinga & Vegter, we do not require local isotopy. Our processing of each box is just as simple as in PV. However, achieving geometric accuracy is somewhat harder with the C_{xy}-predicate. We will address this issue separately.

In this paper, we give the first complete proof of the global isotopy of the Plantinga & Vegter method. Such a proof, being global, is more subtle than the correctness of Snyder's parametrizability approach (which is entirely local).

When meshing a curve that is almost horizontal in some neighborhood, it is very useful to allow boxes in that neighborhood to be elongated along the horizontal direction. Note that the PV algorithm limits the aspect ratios of boxes to be less than 2. So another contribution is to allow subdivision boxes with variable but bounded aspect ratio. The aspect ratio of a box is the length of the longest side of a box over that of its shortest side. This further improves the adaptivity of our method. Other practical improvements include allowing domains of arbitrary geometry, as in [6]. Thus the input domains need not be connected or simply-connected.

We have implemented our algorithms, and to perform comparisons, we have also implemented the Plantinga & Vegter and Snyder's algorithms. We will provide experimental evidence showing that our new approach can greatly speed up the previous algorithms. See Figure 1 for some impressions of our approach: our Balanced Cxy Algorithm produce fewer boxes than Plantinga & Vegter, but unlike Snyder, we achieved this without having to isolating roots. The Rectangular Cxy Algorithm produces even fewer boxes than Snyder's Algorithm.

2. Overview of Subdivision Algorithms

To provide intuitions for our new results, we will recall the work of Snyder and Plantinga & Vegter. In most of our discussion, we fix a real curve

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

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 **box** is given by $B = I \times J \subseteq \mathbb{R}^2$ where I, J are real intervals. Let $m(I)$ and $w(I)$ denote the midpoint and width of I. For a box $B = I \times J$, let $w_x(B) := w(I)$, $m_x(B) = m(I)$; similarly for $w_y(B), w_y(B)$. 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 **sides** and **corners** of a box B by their compass directions (north, south, east, west and NE, NW, SW, SE). We say B has **uniform sign** if the input function f has the same sign at each of its four corners. If $p, q \in \mathbb{R}$ are the SW and NW corners of B, we may denote $B = [p, q]$. 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 side/corner terminology for boxes, but reserve the edge/vertex terminology for the approximation straightline graphs G (or PSLG [19]).

¶*3. 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" [28] satisfying some basic axioms to support exact real approximation.

We also use interval arithmetic [16] – the main tool being inclusion functions ([20]). An inclusion function for $f(X,Y)$ is a function $\Box f(I,J) = \Box f(B)$ that takes input intervals and returns an interval that satisfies the inclusion property: $f(B) = \{f(x,y) : (x,y) \in B\} \subseteq \Box f(B)$. We call $\Box f$ 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 $\Box f(B_i) \to p$ 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 box. We also need box versions of the derivatives, f_x, f_y.

As in [6], 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 paper can be easy to implement with no numerical errors when the input f are PV functions, and all numerical inputs are dyadic. Therefore, nonrobustness issues are moot. See [6, 20] 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 Ratschek and Rokne, and even though they have the same basic approach as this paper, they had to discuss rounding errors [21, §2.5]. Moreover, in their model, computing the sign of $f(X, Y)$ at a point $p = (x_0, y_0)$ is problematic.

¶*4. Generic Subdivision Algorithm.* The subdivision algorithms in this paper 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, possibly re-inserting 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 subdivision depends on two box predicates, $C_{in}(B)$ and $C_{out}(B)$. 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 into Q_{in}. Next, the Refinement Phase takes the output queue from Subdivision, and further subdivide the boxes to satisfy additional criteria – these refined boxes are put in an output queue Q_{ref}. Strictly speaking, it should be possible to combine refinement with the subdivision phase. 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 edges 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.  $G \leftarrow CONSTRUCT(Q_{ref})$
```

¶*5. Example: Crude Marching Cube.* Let us instantiate the generic algorithm just described, to produce a crude but still useful algorithm for "curve tracing" (cf. [15]). For the Subdivision Phase, we must specify two box predicates: let the C_{out} predicate be instantiated as

$$C_0(B) : 0 \notin \square f(B) \tag{2}$$

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 sides of B have lengths less than some $\varepsilon > 0$. Thus, all the boxes in output Q_{out} have width $\leq \varepsilon$. 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 edges (which are straightline segments joining pairs of vertices) to be inserted into E. The output is a straightline graph $G = (V, E)$.

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 a side of B have different signs, we introduce a vertex $v \in V$ at the the mid-point of the side. (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 sides. If B has two vertices, we introduce an edge to connected them (see 2(a),(b)). These edges represent two types of connected components of $S \cap B$: **corner** and **cut components** (respectively) as illustrated in Figure 2(A),(B). A third type of connected component is an **incursion** (or B-incursion) (Figure 2(I)) is not represented, but omission can be justified by isotopy. If B

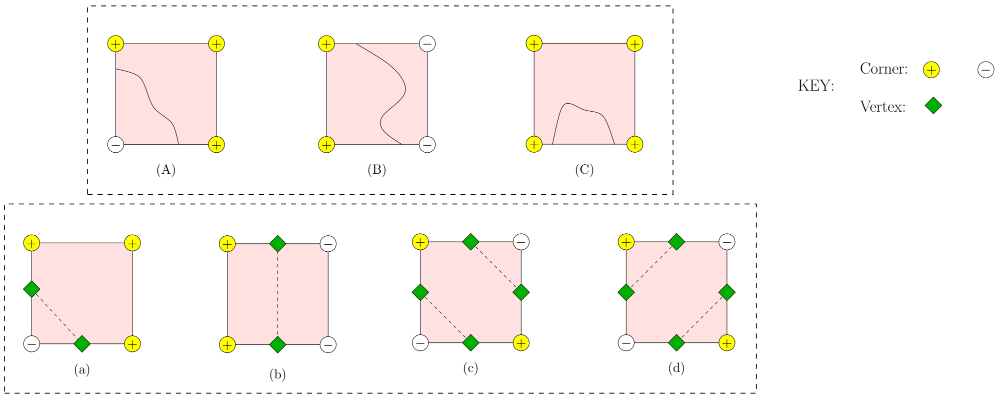


Figure 2: Components Types: (A) corner, (B) cut, (C) incursion. Simple Connection Rules: (a,b) corner and cut edge; (c,d) double corner edges.

has 4 vertices, we introduce two pairs of non-intersecting edges to connect them (see Figure 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 [18].

This completes our description of a crude Marching Cube 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 [15] gave a comparative study of various numerical implementations of the box predicates C_{out}, C_{in}.

¶*6. Snyder's Parametrizability Approach.* Our crude Marching Cube 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 [2, p. 182]. But many papers assume manifolds in order to avoid singularity. In the present paper, 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 [24] and Plantinga & Vegter [18]. In Snyder's approach, the predicate C_{in} is chosen to be

$$C_{xy}(B) : C_x(B) \vee C_y(B) \tag{3}$$

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 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 3.

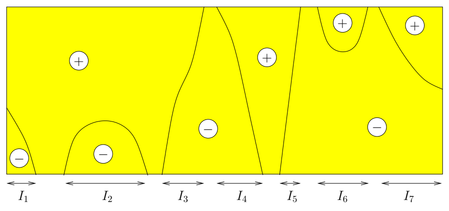


Figure 3: The box components of a C_y-box

¶*7. 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[2] if the curve intersects ∂B tangentially [2, p. 195]. In view of this, the credit for the first complete subdivision algorithm to achieve isotopic approximation of nonsingular curves and surfaces belongs to Plantinga & Vegter [18]. In place of $C_{xy}(B)$, the Plantinga & Vegter (or PV) algorithm uses a stronger predicate that we denote by $C_1(B)$:

$$C_1(B) : 0 \notin (\square f_x(B))^2 + (\square f_y(B))^2. \tag{4}$$

It is called the "small normal variation" condition in [2]. To see that $C_1(B)$ implies $C_{xy}(B)$, we can follow [18] by rewriting the condition 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 $\square f(B) := (\square f_x(B), \square f_y(B))$, and $\langle\cdot,\cdot\rangle$ is just scalar product of a vector. 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 parameterizable 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 $B \in Q_{ref}$. (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 Cube above (see §15 Figure 7). This advantage shows up even more in 3-D, where Snyder's algorithm must recursively solve the 2-D 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.

¶*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 $B(u)$. 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, "in" or "out". So we may speak of the **in-leaves** or **out-leaves** of T. The associated boxes are called **in-boxes** or **out-boxes**. The quadtree T represents a **region** denoted $R(T) \subseteq \mathbb{R}^2$ which is just the union of all the in-boxes. Following [6], we call $R(T)$ a **nice region**. The notion of side/corner is relative to a box B.

A **refinement step** is an operation on a quadtree T in which we split any in-leaf $u \in T$, and tagging the children as in or out. Note that we do not split out-leaves. Although the original tagging is arbitrary, subsequent tagging of new nodes created by refinement must follow a fixed rule, depending on some fixed pair of box predicates $\pi = (\pi_{in}, \pi_{out})$. We tag a node as "out" if it's associated box B satisfies $\pi_{out}(B)$, else it is "in". If $\pi_{in}(B)$ holds but not $\pi_{out}(B)$, we say B is **terminal**. In this paper, π_{out} is always the predicate C_0 above; it ensures that out-boxes can safely be omitted in our approximation of the curve S. So we only focus on π_{in}.

To recap, our algorithm begins with a nice region $R(T)$ in which the tagging of nodes of T are arbitrarily assigned. Subsequently, we refine T using the above rules for tagging new nodes. Thus, there are two types of "out" leaves: original or C_0, and two types of "in" leaves: terminal or non-terminal.

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 three properties of quadtrees T, each obtained by successive refinements:

[2]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.

- $SUBDIVIDE_{\pi_{in}}(T)$ returns a quadtree that **satisfies** the pair $\pi = (\pi_{in}, C_0)$ of box predicates, i.e., each out-box B is either originally tagged as "out" or else $C_0(B)$ holds, and each in-box must satisfy π_{in}, but not C_0.

- $REGULARIZE(T)$ returns a **regular quadtree**, i.e., any two adjacent in-boxes have the same depth. Thus,
$$REGULARIZE(T) \equiv SUBDIVIDE_{\pi_{reg}}(T)$$
where $\pi_{reg}(B) \equiv$ all in-boxes adjacent to B have width$\geq w(B)$. Note that this is more general than Plantinga & Vegter's notion of regularity which requires all the leaves to have the same depth, since the leaves of different connected components of $R(T)$ are allowed to have different depths.

- $BALANCE(T)$ returns a **balanced quadtree**, i.e., one where the depths of any two adjacent in-boxes differ by at most one. Thus,
$$BALANCE(T) \equiv SUBDIVIDE_{\pi_{bal}}(T)$$
where $\pi_{bal}(B) \equiv$ all in-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 a side of a box is subdivided. There are two types of segments: a **boundary segment** e is a side of an in-box of T such that $e \in \partial R(T)$; an **internal segment** e has the form $e = B \cap B'$ where B, B' are adjacent in-boxes of T. Thus each side of a box in T is divided into one or more segments. If T is a regular quadtree, then each side of an in-box of T is also a segment; if T is a balanced quadtree, then each side of an in-box of T is composed of either one or two segments. A **boundary box** is an in-box that has a boundary segment as one of its sides.

For now, assume the above 3 subroutines use only full-splits; the general case where we also allow half-splits is treated in Section 7. Given a quadtree T, we assume a simple subroutine $Q \leftarrow InBox(T)$ that returns a queue Q containing all the in-boxes in T. Thus, the above 3 subroutines can be viewed as "phases" (see ¶4) whose input queues are $InBox(T)$. These subroutines are easily implemented by a simple while-loop as described earlier.

¶*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. 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.

Lemma 1. *For any given f and T, there exists an nice perturbation $\tilde{f}$ of f relative to T.*

From now on, we assume f has been replaced by some nice perturbation (relative to some T).

3. Regular Cxy Algorithm

In this paper, we will describe three increasingly sophisticated subdivision algorithms for curves, all based on the C_{xy} predicate. These will be known as the Regular 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.

Our initial goal is to replace the C_1-predicate in the PV Algorithm by the parametrizability condition of Snyder. As in Plantinga & Vegter [18], we first consider a simplified version in which we regularize the quadtree, i.e., reduce all adjacent in-boxes to the same depth. This is our *Regular Cxy Algorithm.* Our simplified algorithm has this form:

```
Regular 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.   $G \leftarrow CONSTRUCT(T_3)$
```

Initially, ignore Phase 0 (treating the operation $BOUNDARY(T_0)$ as a no-op). Then, the algorithm is just an elaboration of the Crude Marching Cube, 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 Cube because we never have 4 vertices on the sides of an in-box because the condition $C_{xy}(B)$ implies f cannot have alternating signs on the corners of B. Thus, the only connection rules we need are Figure 2(a,b) (i.e., Figure 2(c,d) are excluded).

The naive correctness statement is this: that $S \cap R(T_0)$ is isotopic to G (we will handle geometric accuracy issues later). But the naive algorithm may be incorrect because of "incursions" or "excursions" at boundary segments. More precisely, suppose B is a boundary box and $e \subseteq \partial R(T_0)$ is a side of B. We say S makes an **incursion** (resp., **excursion**) at e if it enters and exits B (resp., exits and re-enters B) at e. Clearly, such incursions/excursions are not captured by our straightline approximation.

¶*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 $\partial R(T_0)$. One way to do this, while still exploiting isotopy at the boundary, is a fairly elaborate method in [6]. But for this paper, we are contented with a simpler solution which was mentioned in [6]: we 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 "check" each of its boundary segment e (there may be two or even three such sides). Checking e amounts to doing the 1-D analogue of C_0 and C_{xy} predicates: let

$$C'_0(e) : 0 \notin \square f(e), \qquad C'_{xy}(e) : 0 \notin \square f_z(e)$$

where $z = x$ if e is horizontal, and $z = y$ if e is vertical. If $C'_0(e)$ or $C'_{xy}(e)$ holds, we discard B. Otherwise we split B, mark these children as in- or out-boxes depending on whether they satisfy C'_0 or not, and place those children that are still boundary boxes back into Q_0. This completes Phase 0.

Note that is the analogue of the EVAL algorithm [8, 7]), but for the boundary of $R(T_0)$. The upshot of this boundary processing is that the curve S intersects every boundary segment of $R(T_0)$ at most once. However, if S intersects a boundary segment tangentially, then the $BOUNDARY(T_0)$ process will not halt.

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

THEOREM 2 (Correctness of Regular Cxy Algorithm). *The algorithm terminates provided S intersects $\partial R(T_0)$ only transversally and f is nonsingular in $R(T_0)$. 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 first two phases have the potential for non-termination. The following lemma provides the condition to guarantee their termination.

LEMMA 3.
(i) If $S = f^{-1}(0)$ intersects the boundary of $R(T_0)$ only transversally, 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 sides, $e_0 \supset e_1 \supset \cdots$, 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:

$$0 \in (\Box f(B_i) \cap \Box f_x(B_i) \cap \Box f_y(B_i)). \tag{6}$$

The boxes B_i must converge[3] to some point $p \in R(T_0)$ as $i \to \infty$. Since $\Box f$ is a box function for f, we conclude that $\Box(B_i) \to f(p)$. Then (6) implies $0 = f(p) = f_x(p) = f_y(p)$. Thus, f is singular in $R(T_0)$. **Q.E.D.**

4. Partial Correctness of Regular Cxy Algorithm

The basic partial correctness technique in Plantinga & Vegter [18] is to apply isotopies which remove any excursion of the curve $f^{-1}(0)$ 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 $S \cap R(T_0) = S \cap R(T_3)$ 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 4(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.

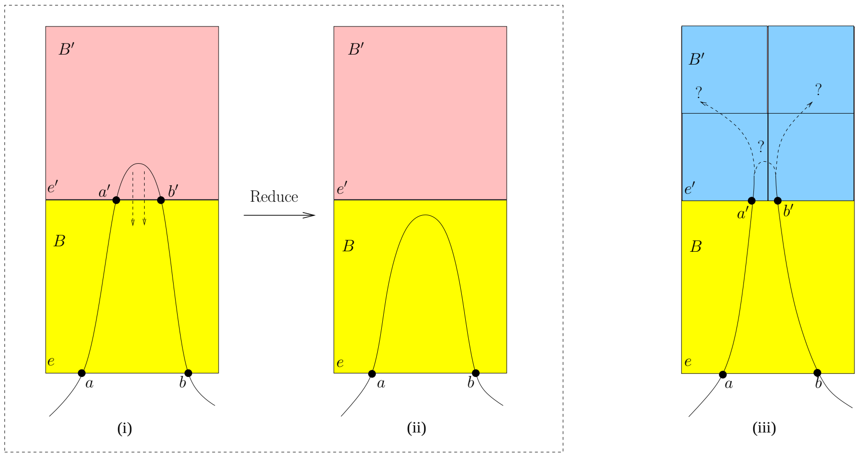


Figure 4: Reduction step with $(a', b') \prec (a, b)$

¶*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 4(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 for $a <_y b$ means $a_y < b_y$. Also, $a \le_x b$ means $a_x \le b_x$.

Let e be a segment, so $e = B \cap B'$ for some in-boxes B and B' (see Figure 4(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 our assumption that S has no vertical or horizontal components, $S \cap e$ is a finite set. In general, S can intersect e at points with multiplicity greater than 1; then, As in [9], we can view $S \cap e$

[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 Cxy Algorithm later.

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 $m \geq 2$. Moreover, if $m \geq 2$ 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 4(i) and (ii).

We have three 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 4. *Let $e = B \cap B'$ be a segment. If B and B' satisfies C_{xy} then every consecutive pair of e is convergent (upward or downward or 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. To define such pairs by the lemma above, we allow either B or B' to be complementary boxes, so the segment $e \subseteq \partial R(T_0)$. The pairs (a, b) in these cases are called **boundary pairs**. 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 4(ii)). But suppose $X_a \neq X_b$, then X_a and X_b are cut components (see Figure 4(i)) satisfying

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

because C_y holds in B. This is illustrated in Figure 4(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 $2k \geq 0$ such intersections, then we can find $k+1$ convergent pairs on $[a', b']$. Assume that B is not a complementary box, that $(a", b") \in P(f)$. Then f (a'', b'') is such a convergent pair, we will define

$$(a'', b'') \prec (a, b). \tag{7}$$

Let $\preceq$ denote the reflexive, transitive closure of the set of binary relations defined as in (7). 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 $C_0^f, C_1^f, C_{xy}^f, CONSTRUCT^f(T)$.

Let T be a quadtree and f, g be PV functions. We say f is **compatible** with T if for each boundary segment e of T, the curve $f^{-1}(0)$ intersects e at most once, and any intersection is transversal. If f and g are both compatible with T, and for all corners u of in-boxes, we have $f(u)g(u) > 0$, then we say f and g are **consistent** on T.

Note that the role of the 0th and 1st Phases of the Regular Cxy Algorithm is to construct a quadtree that is compatible with f. 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 the f has opposite signs at the endpoints of e, and for each in-box with two vertices u, v on its boundary, we introduce an edge $(u, v) \in E$.

Lemma 5. *Let T be a quadtree and f is a PV function. If T is regular and compatible with f, then the graph $G := CONSTRUCT^f(T)$ is isotopic to $f^{-1}(0) \cap R(T)$.*

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_0, f_i are consistent over T $(i = 1, \ldots, n)$ and $S_i(0) \approx S_{i-1}$ where $S_i := f_i^{-1}(0)$.

We may ensure that each S_i intersects the segments of T only transversally, and avoids the corners of in-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 4(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 $|P_n| = 0$.

By repeating this process three more times, we can similarly remove all the downward, left and right convergent pairs. We finally arrive at a function $\overline{f}$ such that there are no consecutive pairs on any segment. According to Lemma 4, this means the curve $\overline{S} := \overline{f}^{-1}(0)$ intersects each segment at most once. Moreover, the $\overline{S} := \overline{f}^{-1}(0)$ is isotopic to $S = f^{-1}(0)$.

It remains to show that $\overline{S} \cap R(T) \simeq G$ where $G = CONSTRUCT^f(T)$. Let B be any in-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 and $|G \cap \partial B| = |\overline{S} \cap \partial B|$. 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 in-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 in-boxes and B lies north 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 g can 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 6. *Let T be a regularized quadtree. If f, g are consistent on T then $f^{-1}(0) \cap R(T) \approx g^{-1}(0) \cap R(T)$.*

Proof. Note that consistency 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 2. *Proof.* Termination follows from Lemma 3. We note how each phase of the Regular Cxy Algorithm provides the necessary properties for correctness: Phase 0 converts T_0 to T_1 which satisfies the Boundary Condition for compatibility between T_1 and f. Phase 1 converts T_1 to T_2 which satisfies the Box Condition for compatibility between T_2 and f (the boundary condition is preserved in this transformation). So T_2 is compatible with f. Phase 2 converts T_2 into a regular quadtree, again preserving compatibility. 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 5, the output G from Phase 3 is isotopic to $f^{-1}(0) \cap R(T_3)$.
Q.E.D.

5. Balanced Cxy Algorithm

The Regular Cxy Algorithm is non-adaptive because of regularization. The **PV Algorithm** is similar to the Regular Cxy Algorithm, except that we 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 [6, 18]).

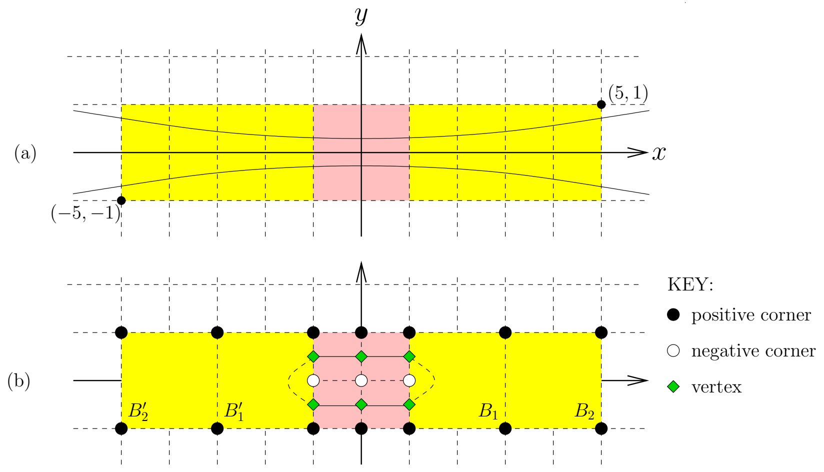


Figure 5: (a) Input "flat" hyperbola. (b) Output graph with wrong isotopy type.

¶*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 5(a). We run the PV algorithm on this input hyperbola and a quadtree T where $R(T) = [(-5,-1),(11,15)]$ which is a 16×16 square. It is conceivable the Subdivision Phase ends up discarding all subboxes except for the 8 shaded squares inside $[(-5,-1),(5,1)]$, as shown in Figure 5(b). Moreover, each of the four larger squares (B_1, B_2, B_1', B_2') satisfy C_x, while the four smaller squares satisfy C_y. The four smaller squares were split from a larger square $[(-1,-1),(1,1)]$ which does not satisfy C_{xy}. The output graph G obtained by using the connection rules of Figure 7 is the 6-vertex graph shown in Figure 5(b). Since G forms a loop, it is clearly wrong. The error occurred in the boxes B_1 (and by symmetry, in B_1') where $G \cap B_1$ has only one connected component while $S \cap B_1$ has two components. 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 (i) it satisfies C_{xy}, (ii) has uniform sign, 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[4].

Figure 6(a) shows an ambiguous box B (it satisfies C_y but not C_x). Note that our definition of ambiguity does not depend on whether B's east or west sides have been subdivided. If we full-split box B, the situation resolves into one of two possibilities, as in Figure 6(b) or 6(c). In fact, 6(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 6(b), this might further cause the southern neighbor of B to become 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 5(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.

[4]I.e., we may require an optional condition: (iv) If B satisfies C_y (resp., C_x) then one of its horizontal (resp., vertical) sides has not yet been subdivided.

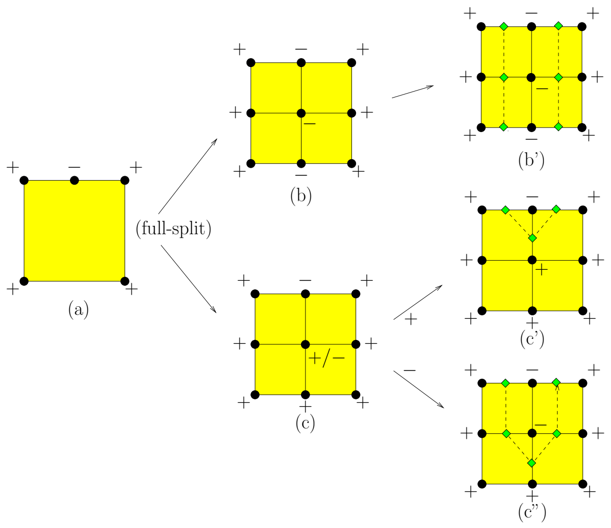


Figure 6: Ambiguous box (a) and its resolution (b',c',c")

¶*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 in-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 in-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 sides, 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 in-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 sides 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 Cube Figure 2(a,b), but reproduced in Figure 7(a,b). If B has 4 vertices, it turns out that two of them will lie on one side of B; we connect these two vertices to the other two in such a way that the edges are non-intersecting (this connection rule is unique, unlike Figure 2(c,d)). These rules are summarized in Figure 7(a–f).

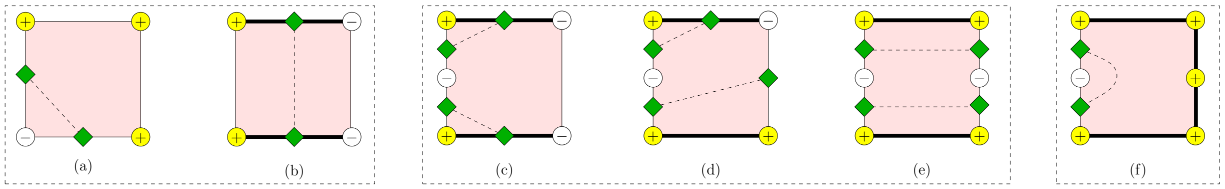


Figure 7: Extended Connection Rules: Cases (c–f) treats two vertices lying on one side of a box.

Four new cases arise Figure 7(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 disambiguating process. Thus, case (f) does not[5] 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 in-box neighbor of B that may become ambiguous as a result. More precisely:

```
PROCESS($B$):
    For each side 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 neighbor (if an in-box) across this side.
    Connect the (at most four) vertices in the sides of $B$
        using the connection rules of Figure 2(a,b) and Figure 7(a-d).
```

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).

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

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

6. Correctness of Balanced Cxy Algorithm

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

THEOREM 7 (Correctness of Balanced Cxy Algorithm). *The algorithm terminates provided S intersects $\partial R(T_0)$ only transversally 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 3. 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 8. *Each in-box has* 0, 2 *or* 4 *vertices on its sides. If it has* 4 *vertices, then two of them will lie on a common side.*

We omit the proof which amounts to a case analysis. This is superficially similar to the PV Algorithm [18], but we actually have a new possibility: it is possible to have two vertices on the east and two vertices on the west side of the in-box as shown Figure 7(e).

Next, we must show partial correctness. Let us see why the proof for the Regular Cxy Algorithm does not work here: in the key lemma there (Lemma 5), 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 4(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 consistent 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 2 (C) and Figure 7 (e)). The following lemma is critical in the correctness proof:

LEMMA 9. *Let T be a balanced quadtree that is compatible with f. 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 an convergent pair of Q_u, then e is the entire south side of an in-box.
(ii) One of the in-boxes of T is ambiguous.

Proof. Let e be 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 south side of in-box B. See Figure 4(iii).

(i) First, we show that e is the entire south side of B. In other words, the south side 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 south side 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', b' lies on the north side (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 $w(e'_a) \leq w(e)/2$. If e is not the entire south side of B, then this contradicts the assumption that T is balanced because $w(B) \geq 2w(e) \geq 4w(e'_a)$.

(Of course, an analogous statement is true: if e contains a pair of Q_d: in this case, e must be the entire north side of an in-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. We now know that e is the entire south side of B.

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 southeast corner of B is negative. Then $S = f^{-1}(0)$ must intersect e at a point c where $a <_x b <_x c$. We may choose c so that (b, c) is a downward convergent pair. If (b, c) is not minimal in the partial order of downward convergent pairs, then $(b, c) \succ (b'', c'')$ for some minimal downward convergent pair (b'', c''). By assumption, (b'', c'') is irreducible. Say (b'', c'') lies in a segment e''. By part (i), we know that e'' is the north side of an in-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 south side of B'' must be split into two subsegments. One of them has to contain the other endpoint of X''_b, and the other subsegment contains the other endpoint of X''_c. But the latter subsegment have width $\leq w(e)/4$ (because since b lies in the right half of e and $b <_x b'' <_x c'' <_x c$). This implies $w(e'') \leq w(e)/2$. This contradicts our choice of $w(e)$ to be minimal.

Thus we may assume that the southwest and southeast corners of B are both positive. But the assumption that $f_y(B) > 0$ implies that the northwest and northeast corners are also positive. Recall that the north side 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 ($Q_u \cup Q_d = \emptyset$).

The following is the analogue of Lemma 5 for the Regular Cxy Algorithm:

Lemma 10. *Let the quadtree T be balanced and compatible with f. 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 5: 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 9, 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.**

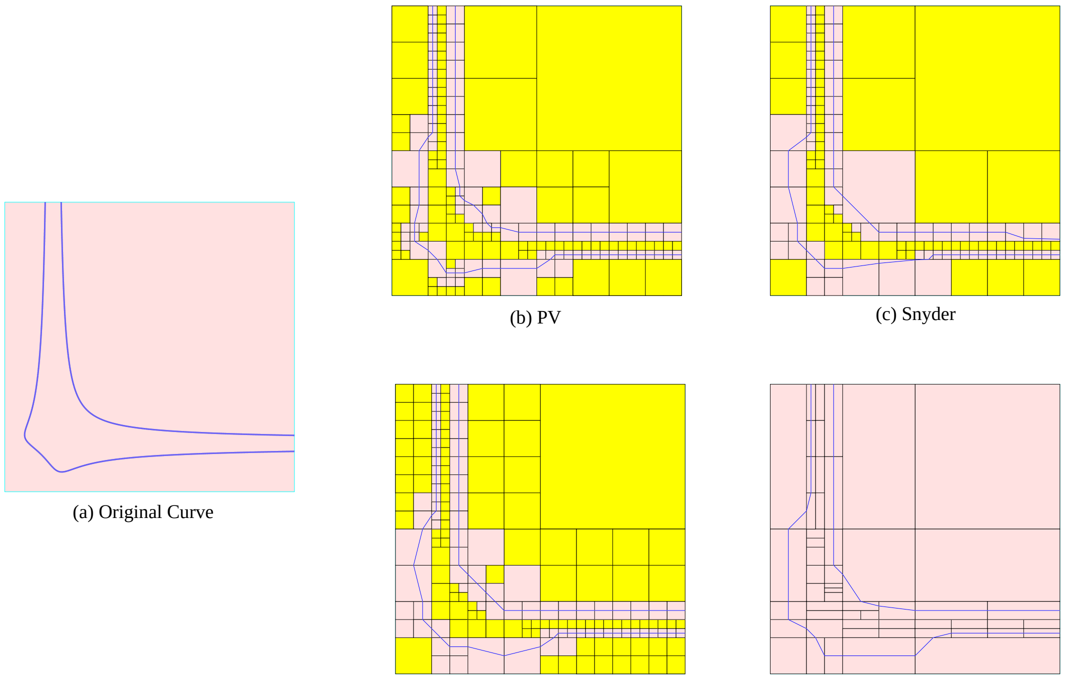
Conclusion of the Correctness Proof. *Proof.* The quadtree T_3 is balanced and compatible with f. When we invoke $CONSTRUCT^+(T_3)$, T_3 is further transformed by splits of ambiguous boxes and their rebalancing. Let T_4 be the final quadtree. It is clear that the output of $CONSTRUCT^+$ on T_3 is the same as what the original $CONSTRUCT$ would produce on input T_4:

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

Clearly, f is still compatible with T_4. By Lemma 10, the straightline graph $G = CONSTRUCT(T_4)$ is isotopic to $f^{-1}(0) \cap R(T)$. This concludes our proof. **Q.E.D.**

7. Rectangular Cxy Algorithm

The recent meshing algorithms [6, 18, 24] 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


(a) Original Curve

(b) PV

(c) Snyder

(d) Balanced Cxy

(e) Rectangular Cxy

Figure 8: 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 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 8. 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 8(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 to 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 $\rho(B) := \max\left\{\alpha(B), \frac{1}{\alpha(B)}\right\} \geq 1$. We require that all boxes in our quadtree satisfy $\rho(B) \leq r$. This ensures the termination of our algorithm.

Second, we modify the Subdivision Phase as follows: For each in-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):

$$\left.\begin{array}{ll} L_0: & C_0(B), C_{xy}(B) \\ L_{out}: & C_0(B_{12}), C_0(B_{34}), C_0(B_{14}), C_0(B_{23}) \\ L_{in}: & C_{xy}(B_{12}), C_{xy}(B_{34}), C_{xy}(B_{14}), C_{xy}(B_{23}) \end{array}\right\} \tag{8}$$

We stop at the first verified condition. If a condition in L_0 is verified, we tag B as an in- or out-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 out (if an L_{out} condition) or in (if an L_{in} condition). The other child is pushed back into the queue. Finally, in 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 $\alpha(B) > r/2$. REMARK: There is considerable opportunity for sharing, and thus optimization, when implementing the arithmetic operations to check the 10 conditions of (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 north and south neighbors have x-depth at most 1 away from the x-depth of B; similarly for y-**balanced** with respect to its east and west 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 in-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 x-**ambiguity** (e.g., Figure 6(a)-(c')) that must be resolved by a vertical split, or a y-**ambiguity** that requires a horizontal split.

8. 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 [18, 17]). To see this explicitly, we claim that it suffices to ensure that for any in-box B, if it has at least one edge of $G = (V, E)$, then its diameter is $\leq \varepsilon/4$. Then any neighbor B' of B has diameter at most $\varepsilon/2$. Thus, each edge 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 $\leq \varepsilon\sqrt{(1/2)^2 + (3/4)^2} < \varepsilon$. 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 a side 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 11 (Buffer Property). *Let (a, b) be a convergent pair relative to box B. Wlog, assume (a, b) lies on the south side 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 $\alpha(B) \geq 1/2$, then $X_a = X_b$.*

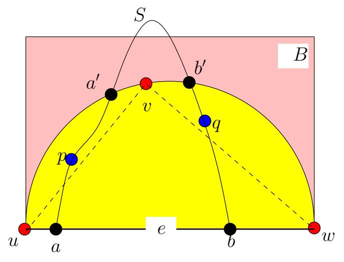


Figure 9: Half-circle argument.

Proof. Figure 9 illustrates our proof. Let H be the upper halfcircle with diameter e. Since $\alpha(B) \geq 1/2$, 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 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 side 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 ≤ 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 12. *Let (a, b) be an upward convergent pair of a segment e, where e is the south side 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 sides as **safe relative to** B:

- If $C_0(B)$ holds, then each of its sides is safe relative to B.
- If $C_x(B)$ holds, then its north and south sides are safe relative to B. Similarly, $C_y(B)$ holds implies its east and west sides 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 as the presence of vertices along the sides of a box. For instance, in Figure 7(a–f), we have indicated by thick edges those sides 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. In Case (f), we can also deduced the eastern side to be "safe", not according to our definition above, but in the extended sense that no incursion or excursion can occur. We could, but need not, exploit such extended notions of safety.

¶*19. Exploiting Safe Segments for Refinement.*

Lemma 13. *Let s be a safe segment.*
(i) Then the curve $S = f^{-1}(0)$ intersects s at most once, i.e., $|S \cap s| \leq 1$.
(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 along the direction of s, the clearly $|S \cap s| \leq 1$. (ii) If f have 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 side. Then the distance from this soft side to the opposite side is called the **soft distance** of B. Note that this soft distance is uniquely defined (for if there is another soft side, then both soft sides are opposite each other). If B has no soft side, then the soft distance is 0 by definition. If the soft distance is $d \geq 0$, 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: 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 side 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 sides of B. If s is a side 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 sides of B. In this case, we want each side 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 terminal boxes. Such a box B is **passive** if the function f has uniform signs (either all positive or all negative) at the corners on the boundary of B; otherwise it is **active**. Note that B contains some edge of the approximate curve G iff B is active. We keep B if the following conditions (a')-(c') hold:

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

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

(c') If B is active and has sign changes on two adjacent sides, then we want the lengths of all sides 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. We remark that this extension of Rectangular Cxy has not been implemented.

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

9. 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). 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. The current code is available on `http://cs.nyu.edu/exact/papers/`. We plan to translate it to `C++` for distribution with our open source

`Core Library`. We implemented four algorithms: PV, Snyder, Balanced Cxy, and Rectangular Cxy. We implemented a version of Snyder in which the boundary root isolation is carried out using the 1-D analogue, the EVAL algorithm (see [8, 7]). For brevity, the Balanced Cxy Algorithm and Rectangular Cxy Algorithm will be known as **Cxy** and **Rect**, respectively.

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 1 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 [18]. 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.

(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 example, by implementing the method from Section 8. The refined curve, with precision $\epsilon = 0.005$, is shown in Figure 1(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 10 (b), (c), (d) and (e) show the cases when $s = 4$. Snyder will not terminate when the curve intersects the sides of the boxes tangentially, so we shift the initial box a little bit). We get the following table (OME=OutOfMemoryError):

#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

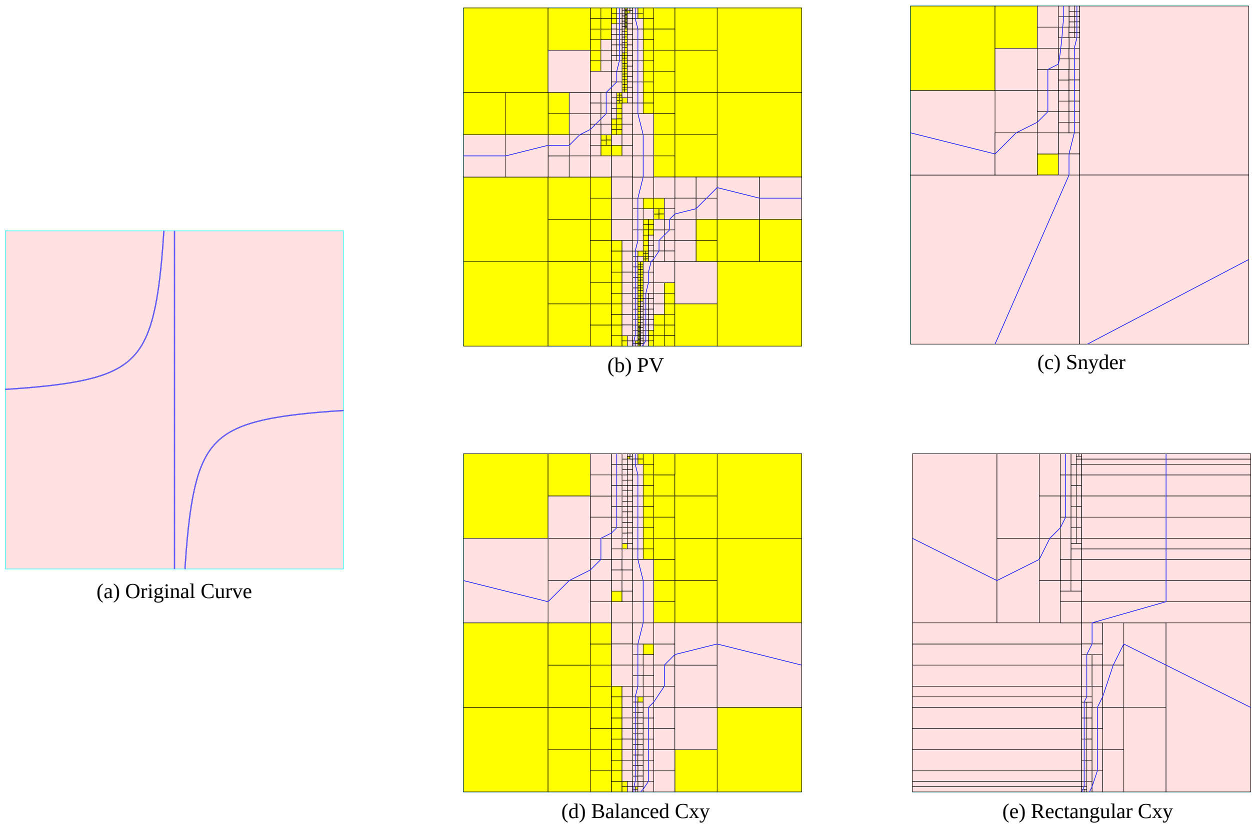


Figure 10: Approximation of $f(X,Y) = X(XY-1) = 0$ inside the box $[(-4,-4),(4,4)]$. Figs. (b),(d),(e) is from PV, Cxy, and Rect, inside the box $[(-3.9,-3.9),(4.1,4.1)]$. Fig. (c) is from Snyder

(4) *Increasing the aspect ratio bounds can speed up the performance of Rect.* Using the same curve and box as before, we now look at the performance of Rectangular Cxy with variable aspect ratio bounds of $r = 10, 20, 40, 80$. Figure 11 shows the case when $r = 15$. The following table shows a proportional speedup (time$= 0$ means time< 1 ms):

#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

(5) *Sometimes Snyder is faster than Balanced Cxy.* We now show an example in 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 $\leq \varepsilon$. 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 12 shows the cases when $n = 2$).

#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

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.

(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 sides 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 10 (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:

#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

(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 can be much faster than EXACUS.

- Approximating the curve $f(X, Y) = X^{100} + Y^{100} - 1 = 0$ in the box $B_0 := [(-2, -2), (2, 2)]$: Cxy takes 9.451 seconds 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 14.235 seconds when $k = 7$; 22.123 seconds when $k = 8$; 27.607 seconds when $k = 9$; and < 3 minutes when $k = 10$.

(8) *Two more examples.* We had already seen Figure 8 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 13 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.

10. Conclusion

This paper introduces a new algorithm for isotopic approximation of implicit curves 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 & Vegter). These ideas are extended to subdivision boxes of bounded aspect ratio. Our experimental results which compare four algorithms (Plantinga & Vegter, Snyder, Balanced Cxy, and Rectangular Cxy) show that our Rect Cxy Algorithm is the best in all tests, and often exhibits great speedup. Future work includes extensions to higher dimensions, implementation of irregular geometries, and exploration of faster techniques for curve refinement. Note that there is no known extension of the PV algorithm to dimension ≥ 4. Another challenge is to produce efficient practical algorithms for higher dimensions. Galehouse [12] recently provided a new subdivision algorithm for meshing surfaces in any dimension, and implemented it in 4 dimensions.

References

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

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

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

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

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

[6] 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.

[7] M. Burr, F. Krahmer, and C. Yap. Integral analysis of evaluation-based real root isolation, Feb. 2009. Submitted, 2009.

[8] M. Burr, V. Sharma, and C. Yap. Evaluation-based root isolation, Feb. 2009. In preparation.

[9] J.-S. Cheng, X.-S. Gao, and C.-K. Yap. Complete numerical isolation of real zeros in zero-dimensional triangular systems. *J. of Symbolic Computation*, 2008. In Press. Special Issue of JSC based on ISSAC 2007. Available online at JSC.

[10] 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.

[11] 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.

[12] B. Galehouse. *Topologically Accurate Meshing Using Spatial Subdivision Techniques*. Ph.D. thesis, New York University, Department of Mathematics, Courant Institute, Apr. 2009 (expected). From `http://cs.nyu.edu/exact/doc/`.

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

[14] 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.

[15] 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.

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

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

[18] 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.

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

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

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

[22] 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.

[23] 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.

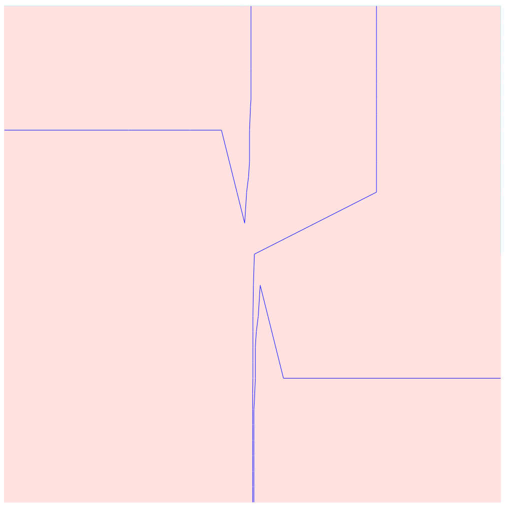
[24] J. M. Snyder. Interval analysis for computer graphics. *SIGGRAPH Comput.Graphics*, 26(2):121–130, 1992.

[25] 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.

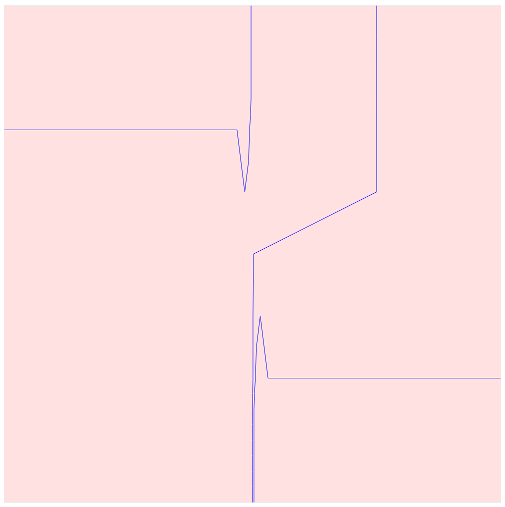
[26] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. Chapman & Hall/CRC, Boca Raton, FL, 2nd edition, 2004.

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

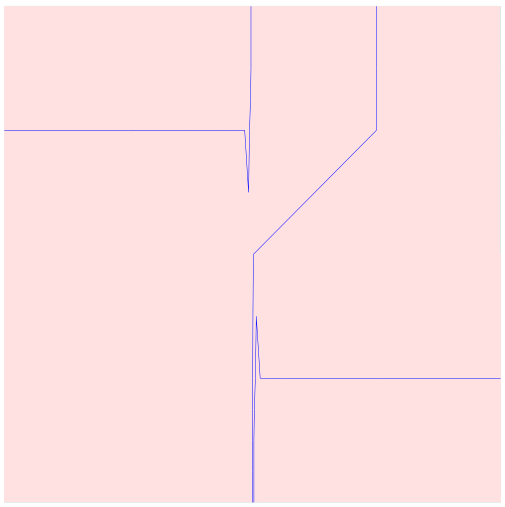
[28] C. K. Yap and J. Yu. Foundations of exact rounding. In S. Das and R. Uehara, editors, *Proc. WALCOM 2009*, volume 5431 of *Lecture Notes in Computer Science*, pages 15–21, Heidelberg, 2009. Springer-Verlag. Invited talk, 3rd Workshop on Algorithms and Computation, Kolkata, India.



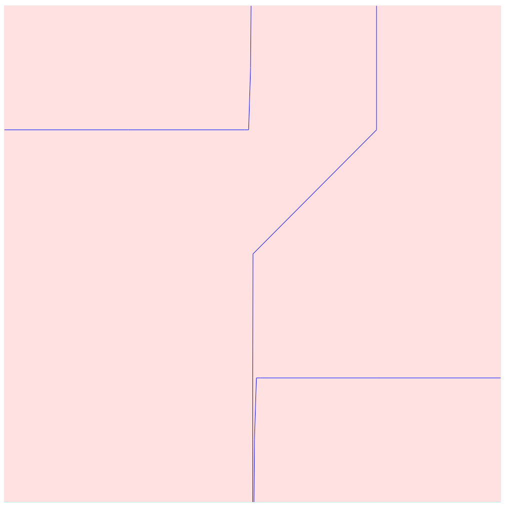
(a) r=10



(b) r=20

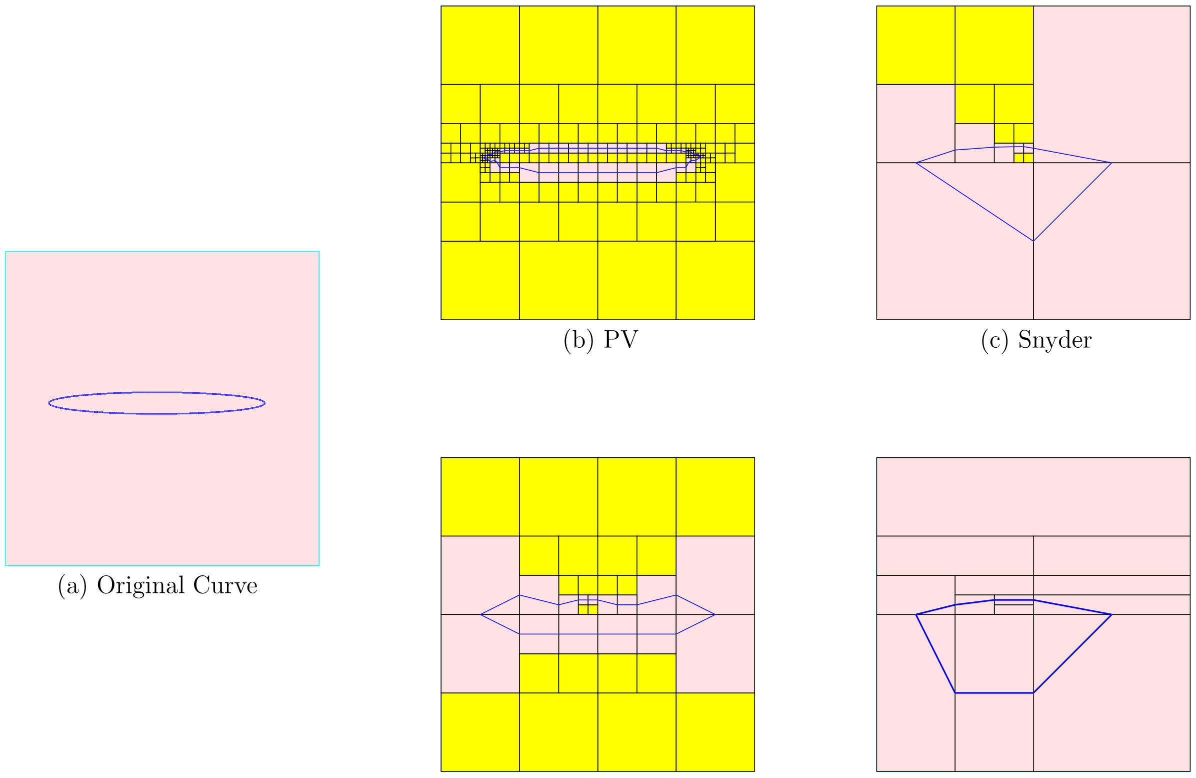


(c) r=40



(d) r=80

Figure 11: 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

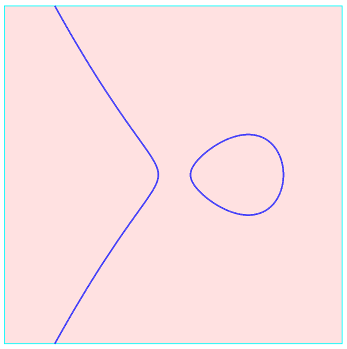
(b) PV

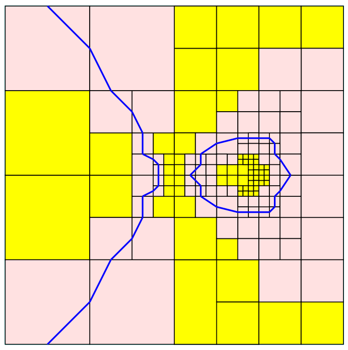
(c) Snyder

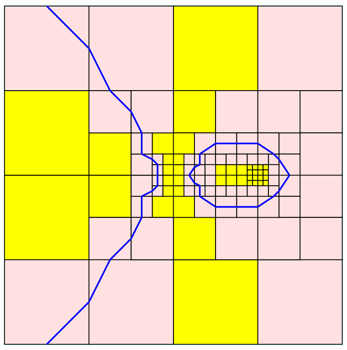
(d) Balanced Cxy

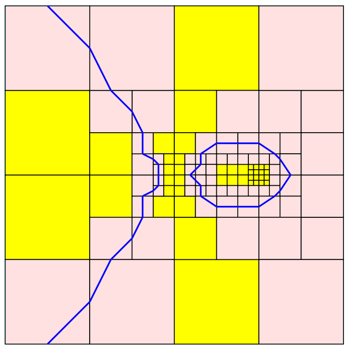
(e) Rectangular Cxy

Figure 12: 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

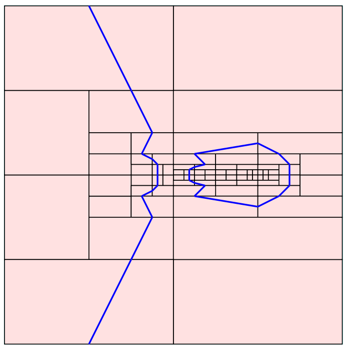

(e) Rectangular Cxy

Figure 13: 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.