Journal of Computational Physics **152,** 664–686 (1999)
Article ID jcph.1999.6259, available online at http://www.idealibrary.com on IDEAL®

Fast Tree-Based Redistancing for Level Set Computations

John Strain[1]

Department of Mathematics, University of California, 970 Evans Hall, Number 3840, Berkeley, California 94720-3840
E-mail: strain@math.berkeley.edu

Received August 19, 1998; revised January 19, 1999

Level set methods for moving interface problems require efficient techniques for transforming an interface to a globally defined function whose zero set is the interface, such as the signed distance to the interface. This paper presents efficient algorithms for this "redistancing" problem. The algorithms use quadtrees and triangulation to compute global approximate signed distance functions. A quadtree mesh is built to resolve the interface and the vertex distances are evaluated exactly with a robust search strategy to provide both continuous and discontinuous interpolants. Given a polygonal interface with N elements, our algorithms run in $O(N)$ space and $O(N \log N)$ time. Two-dimensional numerical results show they are highly efficient in practice.

Key Words: moving interfaces; level sets; distance function; data structures; triangulation.

CONTENTS

[1]Research supported by NSF Young Investigator and SCREMS Awards and by Air Force Office of Scientific Research Grant FDF49620-93-1-0053.

0021-9991/99 $30.00

1. INTRODUCTION

Many problems in computational science involve complex interfaces evolving through topological changes, faceting, and singularities. Such problems are difficult to solve numerically. Level set methods, which view the interface as the zero set of a function, form an effective general approach because they handle topological changes automatically.

An essential procedure in level set methods is *redistancing*: Given an interface Γ built of N elements, compute a signed distance function

$$D(x) = \pm \min_{y \in \Gamma} \|x - y\| \tag{1}$$

on some collection of points x. A more general form of redistancing computes a simpler function $\varphi(x)$ with zero set Γ. Many evaluations of φ are required, so efficiency is important.

We present efficient algorithms for solving this general redistancing problem. Our algorithms resolve Γ with a quadtree mesh and evaluate D on the quadtree vertices in $O(N)$ space and $O(N \log N)$ time. Interpolation produces each value $\varphi(x)$ in $O(\log N)$ time and runs extremely fast in practice.

Section 2 of this paper reviews level set methods for moving interfaces and specifies the redistancing problem. Section 3 describes standard properties of quadtrees, Voronoi diagrams, and Delaunay triangulations. Section 4 presents our fast redistancing algorithms and Section 5 demonstrates their efficiency and utility with numerical experiments. Section 6 draws conclusions and discusses extensions and applications.

2. MOVING INTERFACES, LEVEL SETS, AND REDISTANCING

This section reviews moving interface problems, level set methods, and redistancing. Subsection 2.1 describes four moving interface problems, passive transport, unit normal velocity, anisotropic curvature-dependent flow, and crystal growth. Subsection 2.2 converts moving interface problems into level set equations on a fixed domain and reviews their solution by the level set method. Subsection 2.3 discusses initialization and redistancing in the level set method.

2.1. *Moving Interface Problems*

A general moving interface is the boundary $\Gamma(t) = \partial\Omega(t)$ of a set $\Omega(t) \subset \mathbf{R}^d$ depending on time t. If Ω is sufficiently smooth, then $\Gamma(t)$ has an outward unit normal N and a normal velocity V at each point. A *moving interface problem* is a closed system of equations which specifies V as a functional of Γ, possibly in a highly indirect and nonlocal way. Figure 1 displays representative solutions of several of the following moving interface problems.

Passive transport. An interface is transported in an ambient flow which is independent of Γ. Thus a velocity field $F(x, t)$ is given on $\mathbf{R}^d$ and $\Gamma(t)$ moves with normal velocity $V = N \cdot F$.

Unit normal velocity. The simplest geometric flow moves $\Gamma(t)$ along its normal with velocity $V = 1$. Nonconvex initial interfaces moving under this flow produce complex merging and cornering patterns which challenge standard numerical methods.

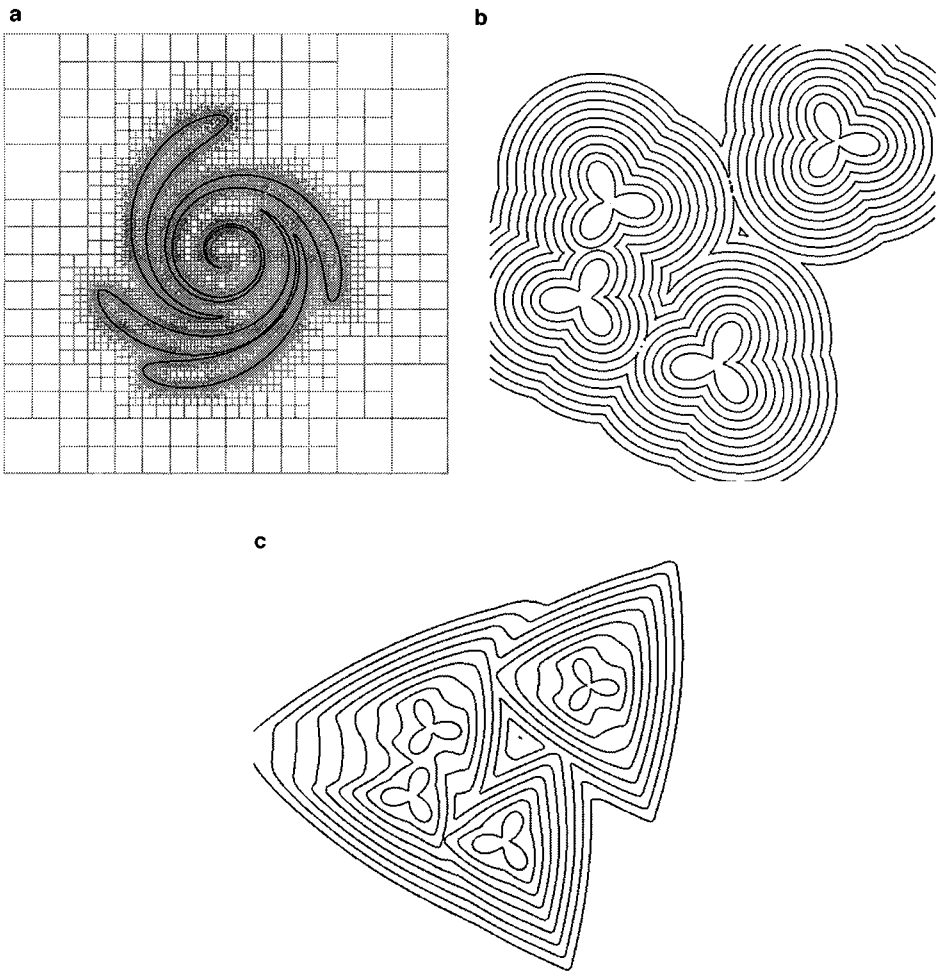

FIG. 1. Representative solutions of moving interface problems: (a) initially circular bubbles after transport in a shearing flow, (b) merging of complex interfaces with unit normal velocity, and (c) crystalline facets developing under a threefold anisotropic curvature-dependent velocity.

Curvature-dependent velocity. An interface moves with anisotropic curvature-dependent normal velocity

$$V(x,t) = R + \epsilon\cos(K\theta + \theta_0) + (R' + \epsilon'\cos(K'\theta + \theta_0'))C, \tag{2}$$

where C is curvature and $\cos\theta = N \cdot e_1$ is the cosine of the angle between the normal vector and the positive x-axis. These velocity fields produce faceted interfaces merging in complex anisotropic patterns and are often used as simplified models in materials science [16].

Crystal growth. Many industrial problems involve moving interfaces separating different phases of a material. The interface between a solidifying crystalline material and its liquid melt, for example, can be modeled by a Stefan-type problem

$$u_t = \Delta u \qquad \text{off } \Gamma(t) \tag{3}$$

$$u = -\epsilon C \qquad \text{on } \Gamma(t). \tag{4}$$

Here the temperature field u is unknown and the interface Γ moves with normal velocity V equal to the jump in the normal derivative of u. Many numerical methods have been developed for this problem [2, 8, 10].

2.2. *Level Set Methods*

Moving interface problems can be reformulated as "level set equations" on a fixed domain, using the *zero set*

$$\Gamma(t) = \{x \in \mathbf{R}^d : \varphi(x, t) = 0\} \tag{5}$$

of an arbitrary function $\varphi : \mathbf{R}^d \times \mathbf{R} \to \mathbf{R}$, such as the signed Euclidean distance

$$D_2(x, t) = \pm \min_{y \in \Gamma(t)} \|x - y\|_2 = \pm \min_{y \in \Gamma} \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \tag{6}$$

or the signed max-norm distance

$$D_\infty(x, t) = \pm \min_{y \in \Gamma(t)} \|x - y\|_\infty = \pm \min_{y \in \Gamma} \max_{1 \le i \le d} |x_i - y_i|. \tag{7}$$

Figure 2 shows a pentagonal interface in $\mathbf{R}^2$ and the corresponding signed Euclidean distance function D_2.

If we choose the sign of φ positive in $\Omega(t)$, then the outward unit normal N and normal velocity V of $\Gamma(t)$ are given by standard geometric formulas [17]:

$$N = \nabla\varphi / \|\nabla\varphi\|_2, \tag{8}$$

$$V = \varphi_t / \|\nabla\varphi\|_2. \tag{9}$$

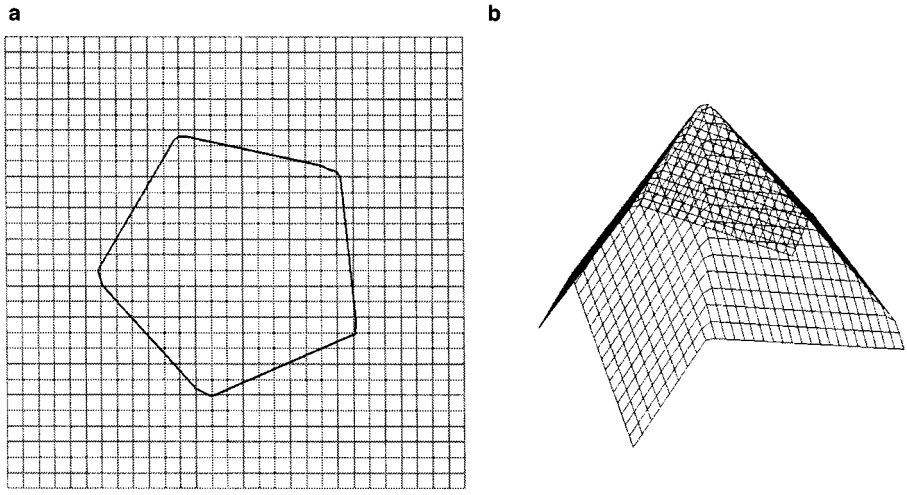

FIG. 2. The correspondence between an interface and a level set function: (a) A pentagonal interface Γ, and (b) the signed Euclidean distance function D_2 to Γ.

Given any extension of the normal velocity vector VN to a function $F(x,t)$ on $\mathbf{R}^d$, Eq. (9) implies the "level set equation" which moves Γ by evolving φ:

$$\varphi_t - F \cdot \nabla\varphi = \varphi_t - (F \cdot N)\|\nabla\varphi\|_2 = 0. \tag{10}$$

Equation (10) moves every level set of φ with the extended velocity F, and in particular moves the zero set $\Gamma(t)$ with the correct velocity VN. Merging, breaking, and other topological changes happen automatically because the topology is embedded implicitly in φ rather than explicitly in $\Gamma(t)$, as illustrated by Fig. 3. The moving interface problems of Subsection 2.1 can be reformulated into level set equations as follows.

Passive transport. For passive transport, F is already defined on $\mathbf{R}^d$ and is a natural extension of VN. The level set equation becomes a linear hyperbolic partial differential equation (PDE)

$$\varphi_t - F(x,t) \cdot \nabla\varphi = 0. \tag{11}$$

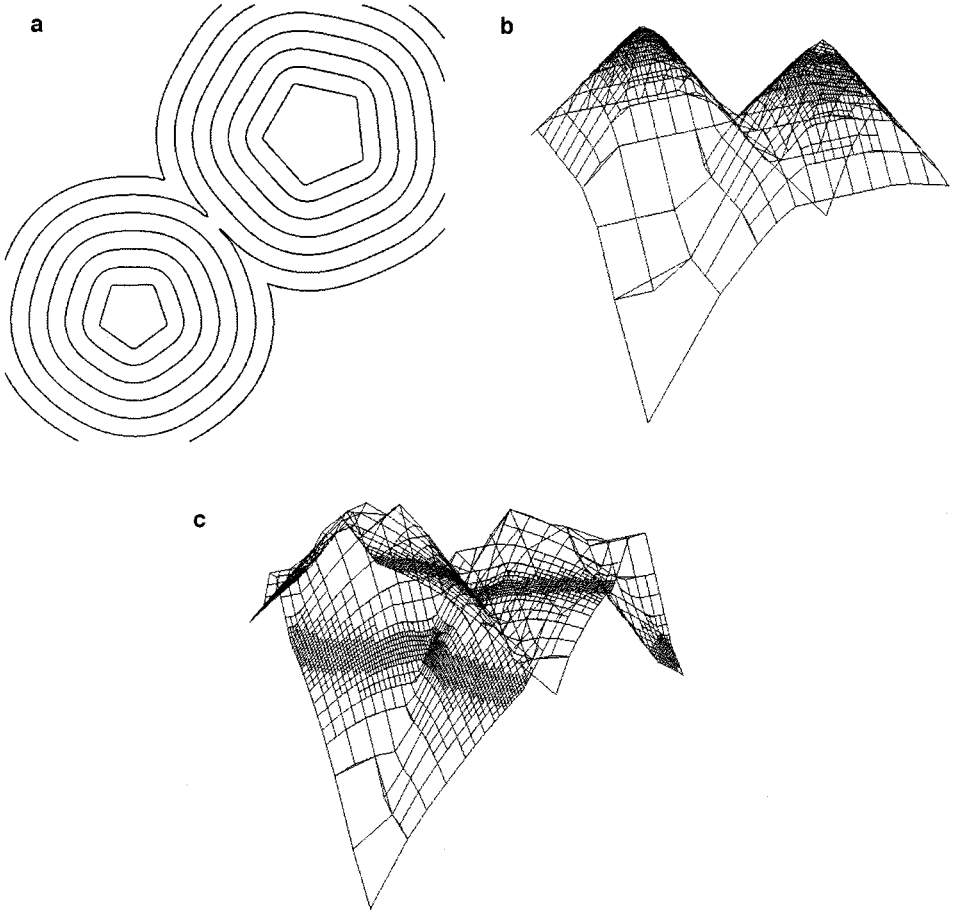

FIG. 3. (a) Two pentagons growing with unit normal velocity merge. The discontinuous piecewise bilinear interpolant to the signed max-norm distance function D_∞ is plotted over a quadtree mesh at (b) initial and (c) final times.

Unit normal velocity. With N extended by Eq. (8), motion with unit normal velocity becomes a nonlinear hyperbolic PDE

$$\varphi_t - \|\nabla\varphi\|_2 = 0. \tag{12}$$

Curvature-dependent velocity. The velocity defined by Eq. (2) yields

$$\varphi_t - (R + \epsilon\cos(K\theta + \theta_0))\|\nabla\varphi\|_2 = (R' + \epsilon'\cos(K'\theta + \theta_0'))\nabla\cdot(\nabla\varphi/\|\nabla\varphi\|_2)\|\nabla\varphi\|_2. \tag{13}$$

Here $\cos\theta = \varphi_x/\|\nabla\varphi\|_2$ and we have used the curvature formula $C = -\nabla\cdot N$ from [17]. Equation (13) is a mixed hyperbolic-parabolic PDE which is singular where $\nabla\varphi$ vanishes.

Crystal growth. The exact velocity of the moving interface in crystal growth is a complicated nonlocal functional of the interface, which can be extended off the interface in many ways. See [2, 10] for representative extensions and numerical results.

Level set methods move $\Gamma(t)$ via the level set equation. An initial level set function $\varphi(x, 0)$ and an extended velocity field F are built, the level set equation (10) is solved numerically, and the solution $\varphi(x, t)$ is contoured when $\Gamma(t)$ is required. The method was introduced in [6], and an extensive recent survey is [9]. It has undergone much development and been applied to many moving interface problems.

2.3. *Redistancing*

Moving interface computations begin with the initial interface Γ, while the level set equation (10) requires globally defined and continuous initial values $\varphi(x)$ satisfying

$$\Gamma = \{x : \varphi(x) = 0\}. \tag{14}$$

The *redistancing problem* consists of the stable specification and efficient evaluation of φ.

Efficiency is important because directly evaluating the signed distance function D_2 from Eq. (6) is prohibitively expensive. Consider a uniform mesh in $\mathbf{R}^d$ with N^d points. A typical interface Γ contains $O(N^{d-1})$ elements, so evaluating D_2 directly on the mesh costs $O(N^{2d-1})$ work. This greatly exceeds the $O(N^d)$ cost of moving Γ one step on the mesh if $d \geq 2$. Thus we need to change φ or compute it efficiently—or both.

Efficiency is even more important for methods and problems where the level set function is redistanced frequently, where we need an algorithm costing less than the computation of one time step. See [2, 9, 13] and Subsection 5.2 for examples.

Several fast schemes for evaluating D_2 approximately on a uniform mesh have been developed: The Eikonal equation (12) is solved to steady state in [15], while [1] applies heapsort techniques. For many applications, however, a uniform mesh wastes computational effort solving the level set equation accurately far from the interface. The ultimate goal of level set methods is to move Γ, and redistancing discards values of φ far from Γ, so efficient level set methods should use an adaptive mesh to concentrate computational effort near the interface. Adaptive level set methods are presented in [7, 9, 13].

The algorithms of this paper evaluate exact signed distance values in any norm at the vertices of a quadtree mesh which resolves Γ accurately at optimal cost. Given these vertex values, many interpolants to the signed distance function can be built. We discuss discontinuous piecewise d-linear interpolation on quadtree cells and continuous piecewise-linear interpolation on various triangulations of the quadtree vertices. All these interpolants are

close to signed distance functions near Γ and give excellent results in the level set method. Our algorithms build φ in $O(N \log N)$ work if Γ contains N elements, cost $O(\log N)$ per φ evaluation, and run extremely fast in practice.

3. QUADTREES AND TRIANGULATION

Our fast redistancing algorithms rely on basic structures of computational geometry such as quadtrees, Voronoi diagrams, and triangulations. In this section, we review the definitions and properties of these structures which suit them to redistancing. We define, build, and triangulate quadtree meshes in Subsection 3.1, discuss the Voronoi approach to redistancing in Subsection 3.2, and provide background on the Delaunay triangulation in Subsection 3.3.

3.1. *Quadtree Meshes*

3.1.1. Definition. A *quadtree mesh* covering the cube $[0, 1]^d$ in $\mathbf{R}^d$ is composed of square cells organized into levels, with each cell on level $l+1$ contained in some level-l cell, and stores the following information:

- The root cell $C_0 = [0, 1]^d$.
- A maximum level $L \geq 0$.
- A *cell list* of cells, grouped by level.
- A *vertex list* of cell vertices, without repetitions.
- Other application-dependent data.

Each cell C in the cell list stores:

- Its level l and corner vertex $(i_1, \ldots, i_d)$: the cell covers the box $2^{-l}[i_1, i_1 + 1] \times \cdots \times [i_d, i_d + 1]$.
- The indices in the vertex list of the 2^d cell vertices.
- The index in the cell list of its parent (if there is one).
- The indices in the cell list of its children (if there are any).
- Other application-dependent data.

Figure 4 shows an example with $L = 2$. Given a general L-level quadtree, many operations related to searching and sorting can be done efficiently: finding the quadtree cell where a point x lies, for example, requires $O(L)$ examinations of bits in the binary representation of x.

3.1.2. Building a quadtree. To build a quadtree, start with a root cell at level $l = 0$. Test whether it requires splitting into 2^d children on level $l + 1$. The *splitting criterion* distinguishes between quadtrees and must be specified to suit the application. If a cell needs splitting, some bookkeeping must be done: create new vertices, adjust familial pointers, and so forth. Then test the children, split if necessary, and repeat the process recursively. Terminate the build when no cell requires splitting.

Many useful quadtrees can be built with some variant of the following splitting criterion:

$$\text{Split any cell whose edge length exceeds its minimum distance to } \Gamma. \tag{15}$$

If the distances from all the vertices to Γ are computed as the quadtree is built, then this criterion is easy to implement. It results in quadtrees with smoothly varying cell sizes which

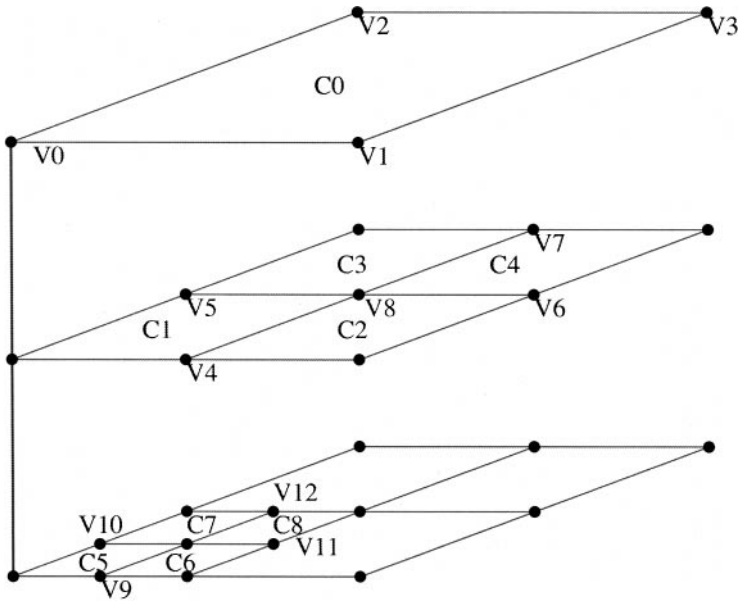

FIG. 4. Levels 0, 1, and 2 of a quadtree with cells C_i and vertices V_i.

are useful in adaptive level set methods [13] and harmonic analysis [12]. These quadtrees represent Γ efficiently: if N childless cells touch Γ, then the entire quadtree meshes $[0, 1]^d$ with only $O(N)$ childless cells. Figure 5 plots a quadtree mesh built with Criterion (15) in the max-norm distance $|D_\infty|$, and a related criterion is used in Section 4.

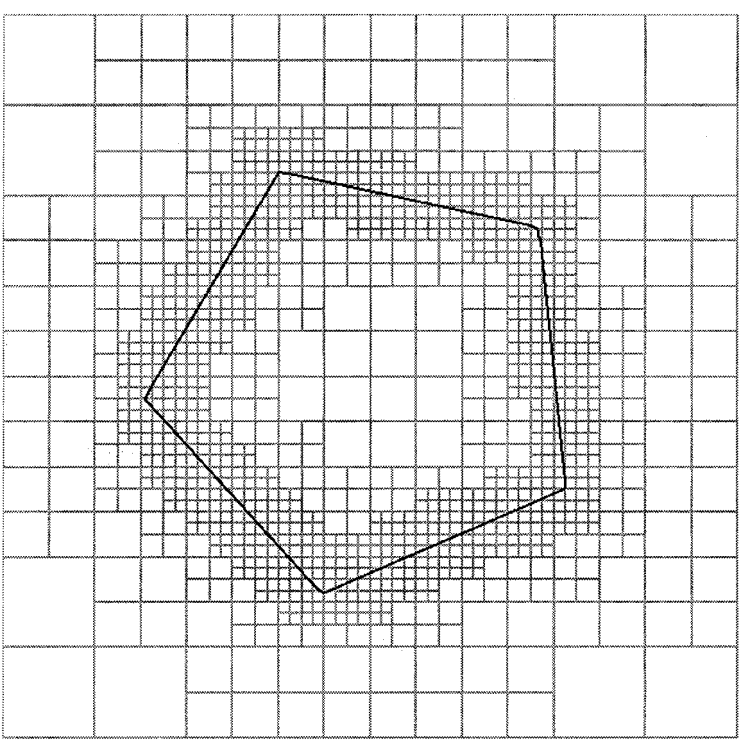

FIG. 5. A six-level quadtree mesh built around the pentagonal zero set of Fig. 2.

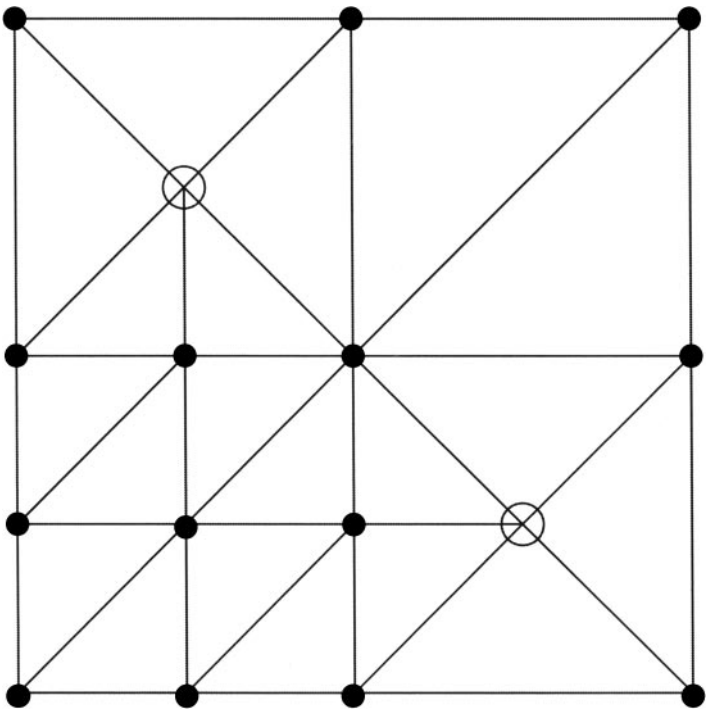

FIG. 6. Triangulating a quadtree mesh. The filled dots are quadtree vertices, open circles are Steiner points.

3.1.3. Triangulating quadtree vertices. Given function values $\varphi(x)$ at the vertices x of the quadtree mesh, there are many ways to define a global interpolant. Multilinear interpolation on each quadtree cell is convenient and efficient, but produces a discontinuity when cells change size.

A continuous interpolant can be built by triangulating the quadtree vertices into a simplicial mesh and linearly interpolating on each simplex (triangle if $d = 2$, tetrahedron if $d = 3$). Any set of vertices can be triangulated with the Delaunay triangulation of Subsection 3.3, but quadtrees built with Criterion (15) provide a more efficient alternative. Adjacent cells differ in size by no more than a factor of 2, so such quadtrees can be triangulated by adding at most one Steiner point per cell and triangulating the result. Figure 6 shows a two-dimensional example: the technique generalizes easily to higher dimensions [3].

Given a triangulation of the tree vertices, level set functions with any desired degree of smoothness can be built by varying the interpolation technique appropriately. A C^k level set function φ can be constructed by Hermite interpolation to the function values and estimated derivatives of order $\leq k$ at triangulation vertices. These techniques are useful in geometric moving interface problems involving φ derivatives such as the normal vector N and curvature C.

3.2. *Voronoi Diagrams*

The Voronoi diagram is a classic tool of computational geometry which solves the equivalent of the redistancing problem when Γ is a *discrete* set of N points $x_j \in \mathbf{R}^d$. The Euclidean Voronoi diagram $V_2(\Gamma)$ is a collection of N regions

$$V_j = \{x \in \mathbf{R}^d : \|x - x_j\|_2 \leq \|x - x_i\|_2 \text{ for all } i \neq j\}. \tag{16}$$

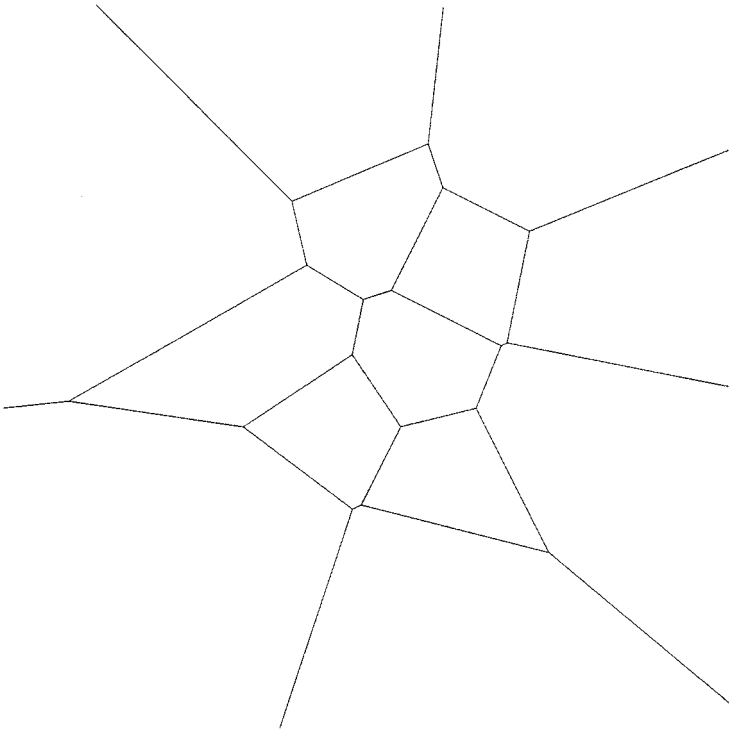

FIG. 7. The Euclidean Voronoi diagram of a small set of points.

The max-norm Voronoi diagram $V_\infty(\Gamma)$ is defined by Eq. (16) with $\| \ \|_2$ replaced by $\| \ \|_\infty$. V_j consists of all points closer to x_j than to any other point of Γ. Figure 7 plots the Euclidean Voronoi diagram of a small point set.

Given $V_2(\Gamma)$, the minimum distance from any x to Γ is simply $\|x - x_j\|_2$ where $x \in V_j$, so redistancing immediately reduces to point location in the partition $V_2(\Gamma)$ of $\mathbf{R}^d$. Many $O(N \log N)$ algorithms for building Voronoi diagrams and for point location have been designed [3, 4], so redistancing from a discrete point set is straightforward.

When Γ is a collection of N computational elements E_j such as segments in the plane or triangles in $\mathbf{R}^3$, the Voronoi regions can be defined by

$$V_j = \left\{ x \in \mathbf{R}^d : \min_{y \in E_j} \|x - y\|_2 \leq \min_{y \in E_i} \|x - y\|_2 \text{ for all } i \neq j \right\}.$$

Thus the Voronoi diagram of Γ reduces the redistancing problem to point location in a subdivision of $\mathbf{R}^d$. Since $V(\Gamma)$ can be built efficiently [19], it provides—in theory—an asymptoticallyoptimal solution to the redistancing problem. However, even when Γ is a set of segments in the plane, there is no practical implementation of any fast algorithm for computing the Voronoi diagram. The Voronoi diagram is expensive to construct because it must identify a single nearest element to every point of space.

A compromise between construction speed and redistancing speed is the compact Voronoi diagram of [5]. This geometric object produces several candidates for the nearest element to any point x, but has a much simpler structure. The boundaries of a compact Voronoi region, for example, are simple polygons, rather than the piecewise algebraic curves of

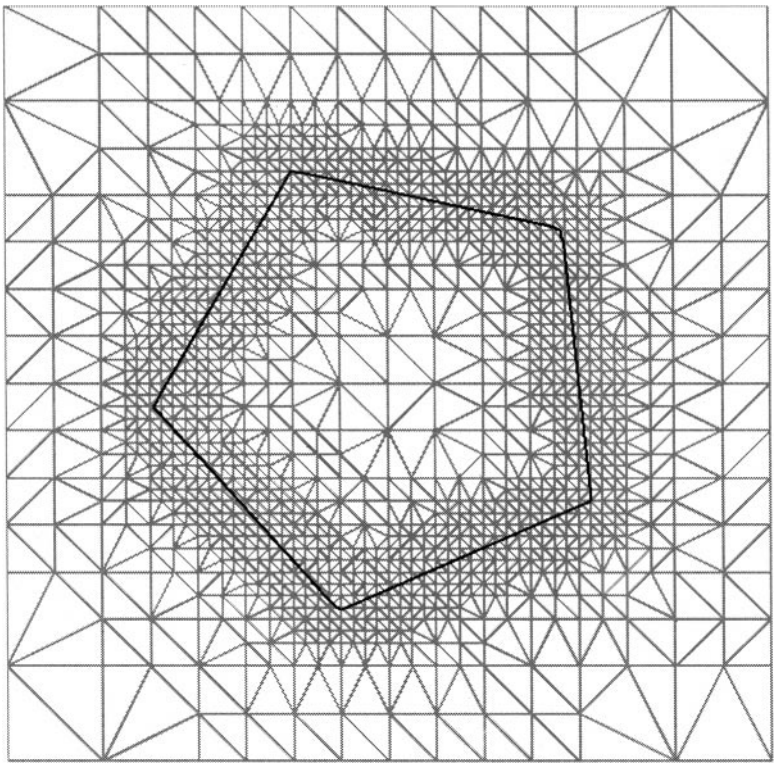

FIG. 8. The Delaunay triangulation of the tree vertices from Fig. 5.

the exact Voronoi diagram for Γ. The compact Voronoi diagram may in future become the redistancing method of choice, but no practical implementation is presently available.

3.3. *Delaunay Triangulation*

The Delaunay triangulation is a geometric object related to the Voronoi diagram, which we use to interpolate φ in Subsection 4.3 and to fix the sign of φ in Subsection 4.4. Given any set of M data points x_j in $\mathbf{R}^d$, there are many ways to connect the points into a simplicial mesh (a triangulation in the plane, a tetrahedralization in space), some better than others. One of the best is the Delaunay triangulation, which is the dual of the Voronoi diagram described in Subsection 3.2. (Figure 8 shows a Delaunay triangulation of the tree vertices from Fig. 5.) It gives optimal error bounds for interpolation [18] and can be built efficiently in time $O(M \log M)$ by many algorithms. We use the standard C code *Triangle* [11]. In three dimensions, the worst-case complexity of triangulating N vertices increases to $O(N^2)$ but Delaunay triangulation remains efficient.

4. EFFICIENT REDISTANCING ALGORITHMS

Efficient tree-based algorithms for redistancing the level set function φ are presented. We define a new data structure, the distance tree, in Subsection 4.1, and build it in Subsection 4.2. In Subsection 4.3, we interpolate on the distance tree to compute various *unsigned* distance functions $|\varphi|$. Subsection 4.4 uses a triangulation of the distance tree vertices to specify a sign for the distance function at each point. Subsection 4.5 achieves the same goal by checking the normal vectors at nearest elements of Γ.

4.1. *Definition of the Distance Tree*

The distance tree is a special quadtree designed to assist in fast redistancing of an interface Γ. In addition to the root cell, cell list, and vertex list defined in Subsection 3.1, it stores vertex distances and cell pointers to nearby elements of Γ.

Suppose Γ is composed of N computational elements E_j, which are segments in the plane, triangles in $\mathbf{R}^3$, or higher-order piecewise polynomial patches. We assume two properties of this decomposition of Γ: first, we can compute the exact minimum distance from any point x to any element E_j in any norm desired; and second, the element sizes all vanish at the same rate as $N \to \infty$. Then the distance tree contains:

- a *vertex distance list* of the minimum max-norm distances $|D_\infty(x)|$ to Γ from each vertex x of the tree.

Each distance tree cell C contains:

- an *element list* of all elements E_j intersecting the *concentric triple* of C. If

$$C = \left\{ x \in \mathbf{R}^d : \|x - c\|_\infty \leq r \right\}$$

has center c and edge length $2r$ then its concentric triple T is given by

$$T = \left\{ x \in \mathbf{R}^d : \|x - c\|_\infty \leq 3r \right\}.$$

For convenience, we will refer to a cell with empty element list as empty. Figure 5 shows the cells in the distance tree for a simple curve.

4.2. *Building the Distance Tree*

We build a distance tree around Γ by

- choosing the root cell,
- specifying a splitting criterion,
- maintaining the element lists, and
- computing the vertex distances from each cell.

The root cell is the smallest square cell enclosing Γ. Its level is 0, and its element list contains every element of Γ. The distances from its 2^d vertices to Γ are computed directly.

The splitting criterion is Criterion (15) from Subsection 3.1.2, specialized to the max-norm distance:

$$\text{Split any cell whose concentric triple intersects } \Gamma. \tag{17}$$

This criterion leads to an efficient search strategy for computing the vertex distances as we build the tree. It can be varied slightly by counting the number of elements intersecting the cell, to avoid over-resolving faceted interfaces Γ with large facets. This variant efficiently resolves adaptively refined interfaces built with elements of varying sizes.

Splitting a cell according to this criterion requires element list and vertex distance list maintenance. Element lists are easy to handle: check every element in the parent's element list for intersection with the child's triple and add it to child lists when intersection occurs. Note that triples of child cells do not tile their parent's concentric triple, so a nonempty parent may have empty children.

Vertex distance list. The most important information in the distance tree —the minimum max-norm distance $|D_\infty(x)|$ from a new vertex x of a child cell C to Γ—is computed efficiently by the following three-step search strategy:

1. Search the element list of C, finding the minimum distance m_1 from x to the elements of Γ in the element list of C.

2. If C is empty or m_1 violates the inclusion condition

$$\left\{y \in \mathbf{R}^d : \|x - y\|_\infty \leq m_1\right\} \subset T, \tag{18}$$

where T is the triple of C, compute the minimum distance m_2 from x to elements of Γ in the element list of C's parent C'. The parent must be nonempty since it is being split.

3. If m_2 violates the inclusion condition

$$\left\{y \in \mathbf{R}^d : \|x - y\|_\infty \leq m_2\right\} \subset T', \tag{19}$$

where T' is the triple of C', expand the search to include elements of Γ in the element list of the grandparent C''. The result m_3 is the minimum distance to Γ.

With this search strategy, the total cost of building an L-level distance tree around an N-element interface is $O(NL)$ as $N \to \infty$, because the union of all triples of cells on level $l \leq L$ intersects $O(N)$ elements.

The correctness of this procedure relies on using the max-norm distance function $|D_\infty|$, because our square tree cells are spheres in the max-norm. Figure 9 illustrates why *great*-grandparents never need to be searched. The nearest element of Γ which intersects the triple

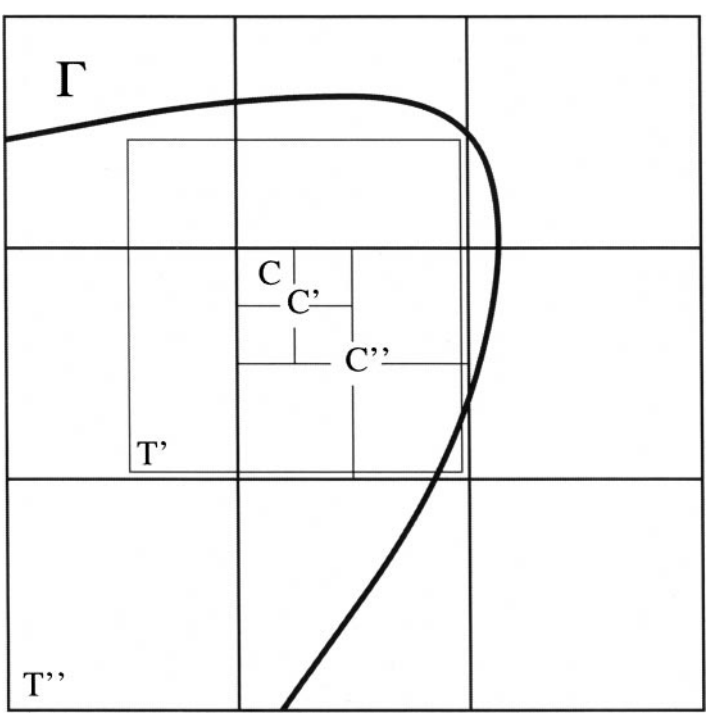

FIG. 9. The nearest element of Γ which intersects the triple T' of the parent C' of a distance tree cell C may not be the nearest element of Γ overall. But if the max-norm is used, the nearest element intersecting T' beats every element outside the triple T'' of the grandparent C''.

T' of the parent C' of a tree cell C may not be the nearest element of Γ overall. But if the max-norm is used, the nearest element intersecting T' beats every element outside the triple T'' of the grandparent C''.

In the Euclidean norm, the distance between opposite corners of a hypercube is $\sqrt{d} > 1$ so we may need to search great-grandparents. However, a slight variant of this search strategy computes all vertex values of D_2 in $O(NL)$ time.

4.3. *Interpolation*

The distance tree gives us exact distance values at $O(N)$ vertices clustered near the interface. A globally defined level set function φ can be built from these distance values in many ways, of which we discuss d-linear cell interpolation and linear simplex interpolation.

4.3.1. Cell interpolation. Let $|\varphi|$ be the d-linear interpolant to the vertex values on each empty childless distance tree cell. In $d = 2$ dimensions, for example, a cell C with edge length h has four vertices $(x_0 + ih, y_0 + jh)$ with distances $|\varphi_{ij}|$ for $0 \le i, j \le 1$. Then $|\varphi|$ is defined at a point $(x = x_0 + \alpha h, y = y_0 + \beta h)$ in C by

$$|\varphi|(x, y) = (1-\alpha)(1-\beta)|\varphi_{00}| + \alpha(1-\beta)|\varphi_{10}| + (1-\alpha)\beta|\varphi_{01}| + \alpha\beta|\varphi_{11}|. \quad (20)$$

On nonempty childless cells, whose concentric triples intersect Γ, $|\varphi|$ is defined to be exactly equal to the distance $|D_\infty|$ to Γ. (See Fig. 10.) The following search strategy evaluates $|\varphi|$ efficiently in nonempty childless cells near Γ: A nonempty childless cell C stores a list of all elements of Γ intersecting its triple T. This allows us to search the element list of C for the closest element to x, yielding a minimum distance m_1. As in the construction of

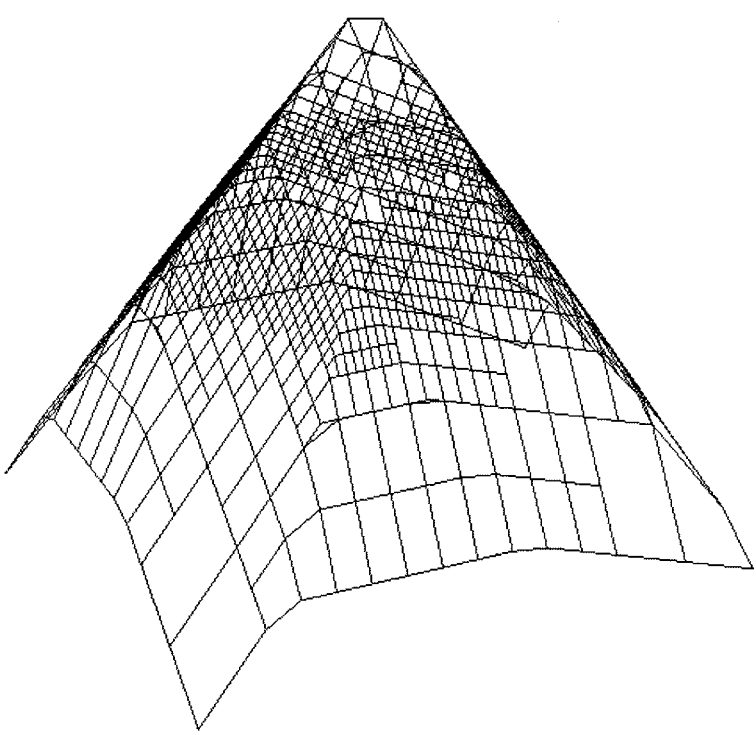

FIG. 10. The discontinuous piecewise bilinear interpolant to the signed max-norm distance function D_∞ on a six-level quadtree mesh built around the pentagonal zero set of Fig. 2.

the distance tree, m_1 may not be the global minimum distance, but searching the element list of the parent C' is guaranteed to find the minimum. There are a bounded number of Γ elements near any nonempty childless cell C as $N \to \infty$, so the cost of each evaluation is dominated by the $O(\log N)$ cost of finding the cell C containing x.

Thus $|\varphi|$ is evaluated by interpolating exact vertex distances on empty childless cells and by searching one or two short lists on nonempty childless cells near the interface. It has the following properties.

- $|\varphi|$ has zero set Γ,
- $|\varphi| = |D_\infty|$ at vertices of the tree and in cells sufficiently near Γ,
- $|\varphi|$ can be evaluated in $O(\log N)$ work at any x, and
- $|\varphi|$ is continuous almost everywhere, with jumps where cells change size decreasing in magnitude as cells approach Γ.

4.3.2. Simplex interpolation. We can construct a continuous level set function φ by triangulating the vertices and piecewise-linearly interpolating to the vertex values. See Fig. 11 for an example. This level set function can be evaluated at any point x by finding the simplex containing x and interpolating from its vertices. Finding the simplex costs $O(\log N)$, so each $|\varphi|$ evaluation costs $O(\log N)$. Building the Delaunay triangulation of the $O(N)$ tree vertices costs $O(N \log N)$, while triangulating the quadtree vertices directly as in Subsection 3.1.3 costs $O(N)$—both asymptotically comparable to building the distance tree.

The simplex interpolant is continuous, but not exactly equal to D near Γ, and thus has zero set slightly different from Γ. If necessary, we can form the constrained Delaunay triangulation including the edges and vertices of the polygonal interface Γ, to get a continuous interpolant with zero set exactly equal to Γ.

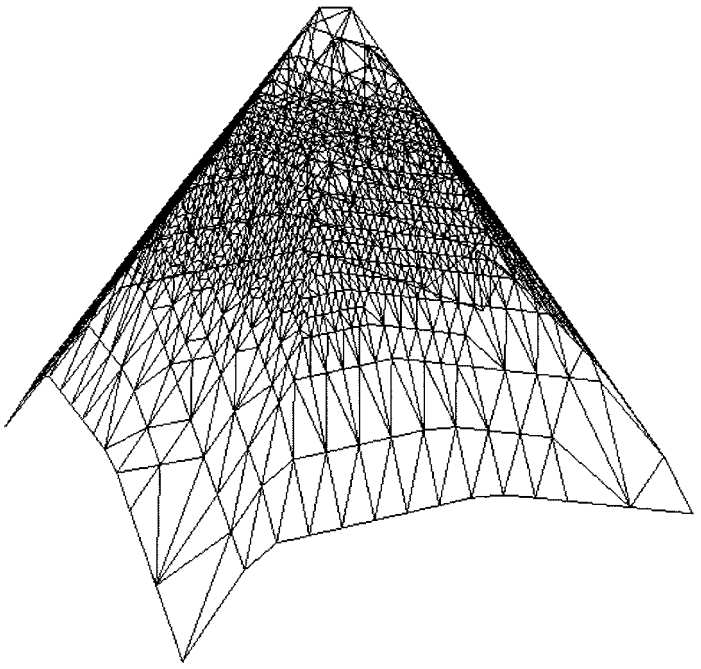

FIG. 11. The continuous piecewise linear interpolant to the signed max-norm distance function D_∞ on the Delaunay triangulation of Fig. 8.

4.4. *Signing φ by Triangulation*

Both cell and simplex interpolation produce an approximate *unsigned* distance function $|\varphi(x)|$ associated with an interface Γ, while the level set method requires the *signed* distance function $\varphi(x) = \pm|\varphi(x)|$. When redistancing φ from its own zero set on a fixed grid, as in Subsection 2.3, we can preserve the sign of each value of $\varphi(x)$. We need to fix the sign, however, when initializing φ from an interface Γ. The following algorithm locates the interface in a triangulation of the quadtree vertices, then propagates the sign inward from the boundary.

Given a triangulation of the distance tree vertices, mark every edge of the triangulation which is crossed by an element of Γ. This requires time proportional to the tree depth times the size of Γ, $O(N\,L) = O(N \log N)$, because marking each edge costs $O(L)$. Set the sign of φ to -1 at one corner vertex of the root cell and propagate the sign along every edge. Before propagating along a marked edge, change the sign from -1 to $+1$ or from $+1$ to -1. If Γ crosses the edge n times, change the sign n times. A robust algorithm must take special care when Γ touches an edge without crossing. After propagation finishes, we have a consistent choice of sign for φ at each distance tree vertex. See Fig. 12 for an example.

4.5. *Signing φ by Normal Vectors*

The sign of φ can be fixed more efficiently if we have an outward unit normal vector on each element of Γ. Then while we are finding a closest point of Γ to any quadtree

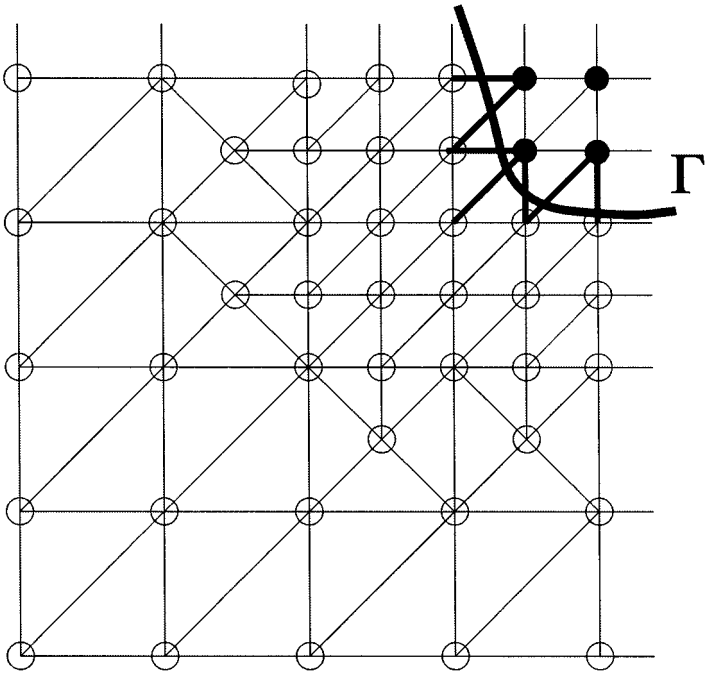

FIG. 12. Fixing the sign of φ with a triangulation of the distance tree vertices. The lower left corner is assigned sign -1; then the sign propagates unchanged along edges not crossing Γ, shown as thin lines, and flips on edges crossing Γ, shown as thick lines. Thus the open circles receive negative φ values while the solid dots receive positive φ values.

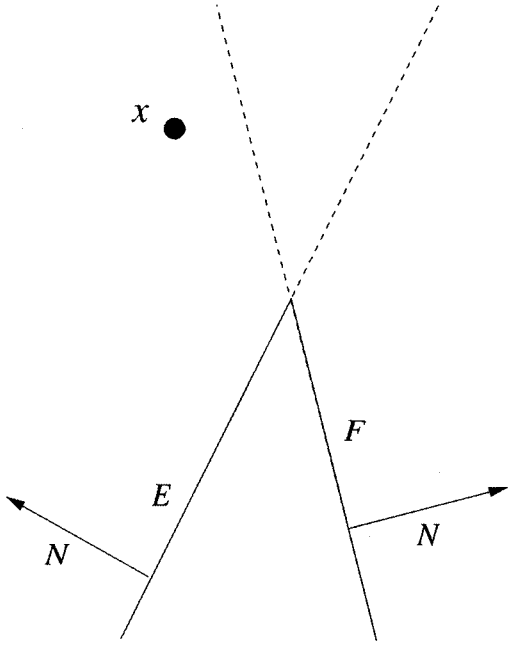

FIG. 13. Pitfalls in fixing the sign of φ. The point x lies outside Γ according to element E, but inside Γ according to element F, and both are nearest elements of Γ to x.

vertex x we can determine the sign of $\varphi(x)$ simultaneously by checking which side of the nearest element x lies on. This procedure costs much less than the triangulated algorithm of Subsection 4.4, but requires some care in implementation. Figure 13 shows one possible pitfall: At acute corners of Γ, *all* elements of Γ touching the corner must be checked to determine the sign of $\varphi(x)$ correctly.

5. NUMERICAL RESULTS

We verify the speed of our algorithms with large-scale runs on simple and complex interfaces in Subsection 5.1 and show a simple application to level set computations in Subsection 5.2. Two-dimensional versions of our algorithms were implemented in ANSI C, compiled with the SunSoft C compiler using the *-fast* flag, and run on one CPU of a Sun 200 MHz Ultra-2 workstation under Solaris 2.6. The codes have not been specially optimized and could probably be speeded up by one to two orders of magnitude by investing additional programming effort.

5.1. *Efficiency*

5.1.1. Direct evaluation. We took the signed Euclidean distance function D_2 of the pentagonal interface from Fig. 2, contoured it on a square $M \times M$ grid to represent Γ with $N = O(M)$ segments, then evaluated $D_\infty(x)$ directly at every vertex of the $M \times M$ grid. Table I(a) records the CPU times T required and the computed area A enclosed by the interface, which indicates the resolution of Γ provided by the $M \times M$ grid. We repeated the experiment with the complex interface of Fig. 14 and report the corresponding times and areas in Table I(b).

TABLE I

Uniform Grid Size M, Number N of Segments in the Interface, Computed Area A, and CPU Seconds T for Direct Redistancing of the Level Set Function ϕ

(a)				(b)			
M	N	A	T	M	N	A	T
32	120	3.61486	0.09	32	302	1.50602	0.19
64	248	3.62906	0.73	64	856	1.71439	2.32
128	496	3.63180	5.84	128	1900	1.73998	20.5
256	1000	3.63253	49.7	256	3920	1.73377	169
512	2004	3.63266	400	512	7936	1.72980	1370

Note. Results for (a) the pentagonal interface of Fig. 2, and (b) the complex interface of Fig. 14.

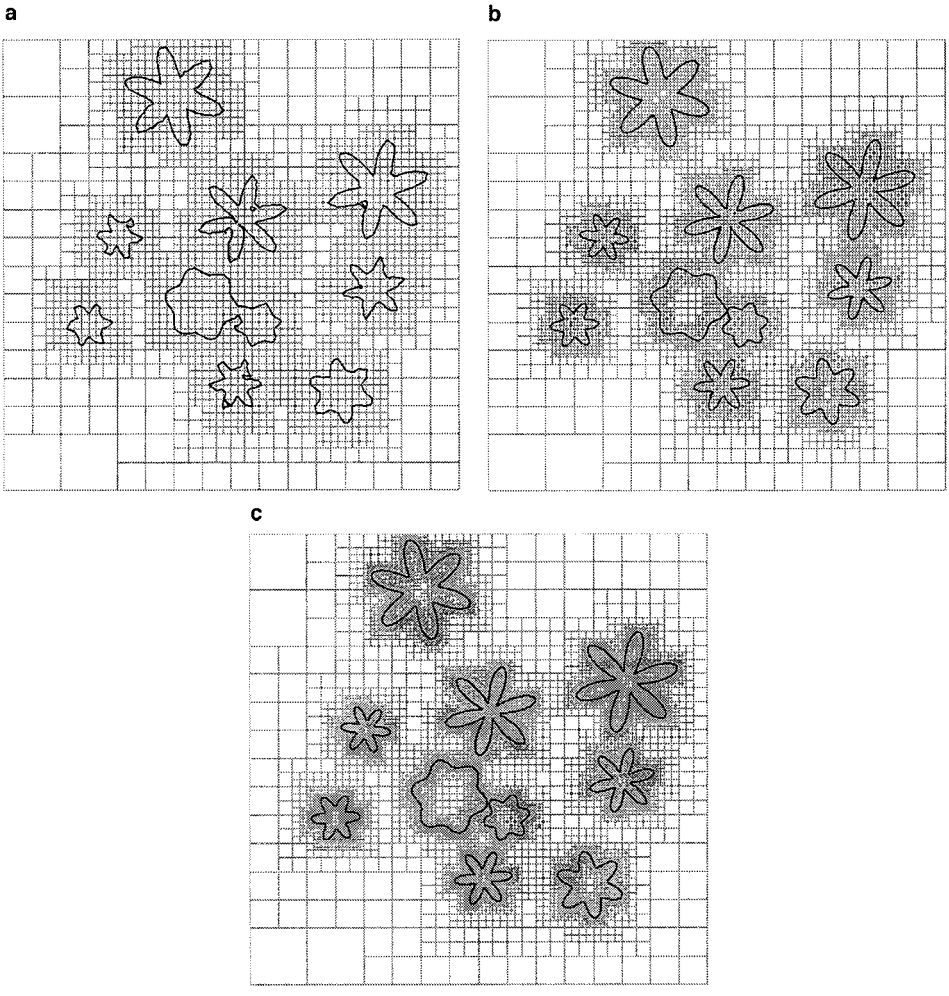

FIG. 14. The complex interface used in Tables I and II, plotted over a distance tree mesh with (a) 6, (b) 7, and (c) 8 levels.

These experimental results agree with theoretical expectations: the direct method requires $O(M^2N) = O(M^3)$ CPU time. Dividing T by M^2N gives an almost constant result, close to 10^{-6} seconds per segment per evaluation. Therefore $T \approx 10^{-6}M^2N$ seconds, so each direct evaluation costs $O(N)$ time. For large-scale computation, these timings are prohibitive. Three-digit accuracy in the area of a complex interface requires almost three minutes per redistancing: adequate accuracy in geometric quantities such as normal vectors and curvature would require greater computational effort and make fast algorithms even more attractive.

5.1.2. Fast algorithms. We redistanced the interfaces shown in Figs. 2 and 14 with our fast algorithms to verify their efficiency. Here we built the L-level distance tree and evaluated all vertex distances for the same interface, contoured by bilinear interpolation on tree cells into N segments. This gives resolution on Γ equivalent to the direct method on a uniform $M \times M$ mesh with $M = 2^L$, at far less cost. Table II reports the CPU times T for our fast algorithms, together with the number C of adaptive tree cells, the CPU time D for Delaunay triangulation of the $O(N)$ tree vertices by *Triangle* [11], and the CPU time S required to move the interface one step with the adaptive tree methods of [13].

5.1.3. Comparisons. We draw the following conclusions from Tables I and II:

1. The total number of quadtree cells required to resolve an interface Γ with the accuracy of a uniform $M \times M$ mesh is asymptotically $O(M)$—an optimal result with bilinear contouring.

TABLE II

Number of Tree Mesh Levels *L* and Cells *C*, Number *N* of Segments in the Interface Γ, Computed Area *A*, and CPU Seconds *T* for Fast Redistancing of the Level Set Function φ

L	$M = 2^L$	N	C	A	T	D	S
				(a)			
5	32	124	729	3.6111	0.03	0.14	0.16
6	64	252	1601	3.62792	0.09	0.24	0.28
7	128	502	3365	3.63145	0.20	0.44	0.50
8	256	1006	6913	3.63238	0.46	0.83	0.96
9	512	2010	14017	3.63264	1.07	1.76	2.01
10	1024	4020	28253	3.63269	2.36	3.56	4.15
11	2048	8040	56673	3.63271	5.25	7.23	8.44
12	4096	16080	113541	3.63271	11.6	14.8	17.4
				(b)			
5	32	322	989	1.41287	0.05	0.17	0.19
6	64	878	2897	1.68656	0.20	0.37	0.42
7	128	1918	7641	1.72989	0.56	0.89	1.03
8	256	3906	19701	1.73062	1.43	2.28	2.62
9	512	7918	46677	1.72947	3.44	5.79	6.66
10	1024	15912	101973	1.72873	8.03	13.2	15.2
11	2048	31849	213561	1.72849	18.5	28.3	33.2
12	4096	63705	437421	1.72844	41.5	60.7	69.8

Note. CPU seconds D for Delaunay triangulation of the tree vertices and S for moving the interface one step. (a) The pentagonal interface of Fig. 2, and (b) the complex interface of Fig. 14.

2. The fast algorithms produce a globally defined level set function φ for an N-element interface in $O(N \log N)$ CPU time, much faster than the $O(M^3)$ required for direct evaluation on a uniform mesh and the $O(N^2)$ required for direct evaluation at the tree vertices.

3. With 8000 segments resolving the area of a complex interface to four-digit accuracy, the equivalent of a 512×512 uniform mesh, our fast algorithms are 400 times faster than direct evaluation on a uniform mesh and 20 times faster than direct evaluation at the tree vertices. At our highest resolution, equivalent to a 4096×4096 uniform mesh, the fast algorithms are 160 times faster than direct evaluation at the tree vertices.

4. Our fast redistancing algorithms are highly practical; they cost less than Delaunay triangulation of the tree vertices and less than moving the interface one step, permitting frequent redistancing at negligible cost.

5.2. *Application to Level Sets*

The following example from [1] demonstrates the usefulness of frequent redistancing. A circle is passively transported with unit radial velocity in the velocity field (see Fig. 15)

$$F(x, y, t) = ((r - (1 + t)) \cos(5\theta + \theta_0) + 1)(x, y)/r. \tag{21}$$

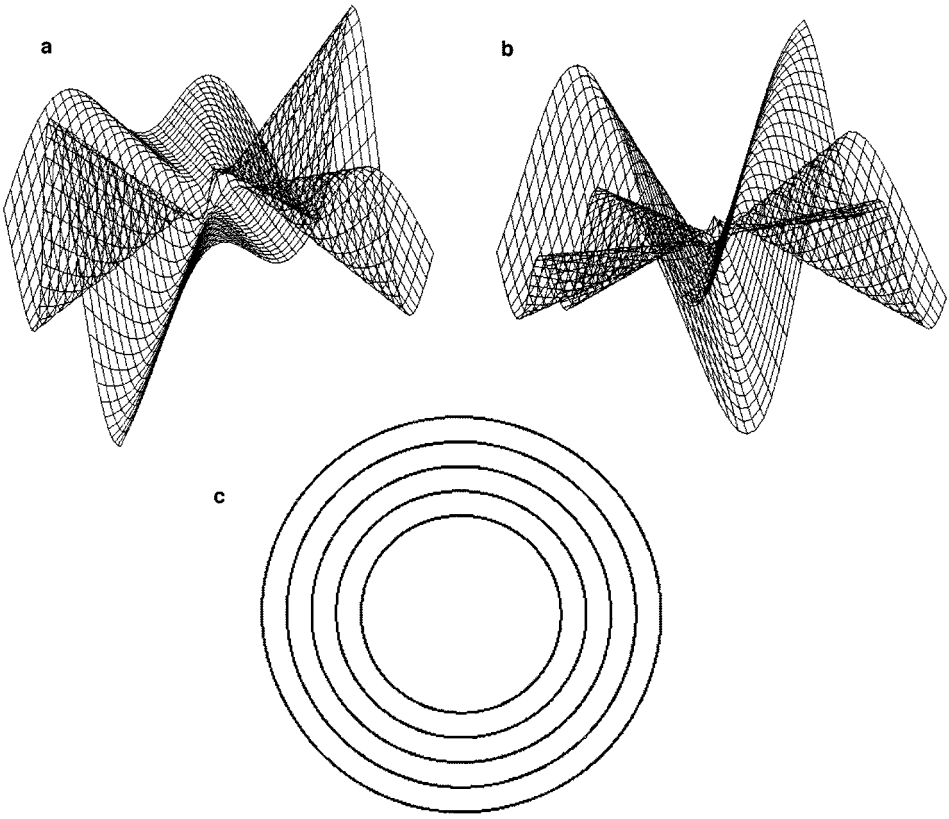

FIG. 15. The velocity of Eq. (21), with X and Y components shown in (a) and (b). The interfaces at equal times are shown in (c).

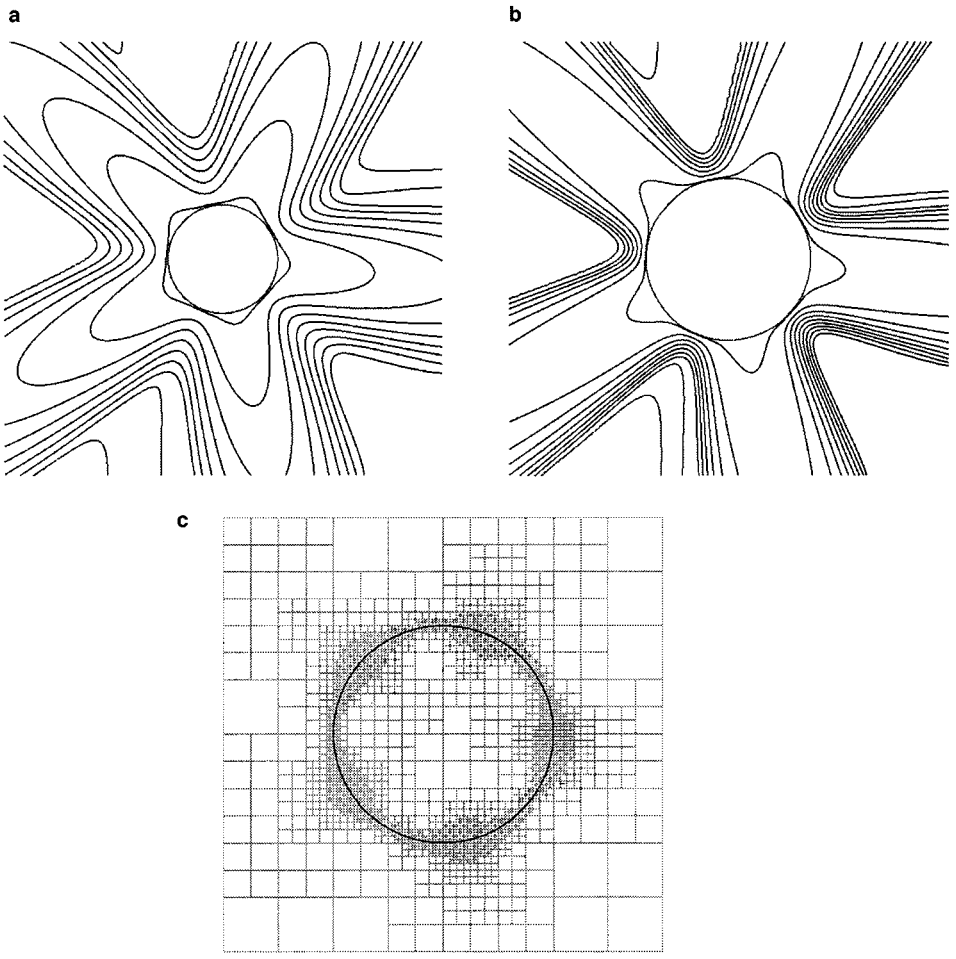

FIG. 16. A circular interface expanding with the velocity of Eq. (21), computed by solving the level set equation with no redistancing: (a) and (b) contours of the level set function at $t = 1$ and 2, (c) the quadtree mesh from the method of [13] at $t = 2$.

While the circle expands at unit speed, other level sets are wildly distorted by the fivefold anisotropic field F (see Fig. 16), producing large solution gradients which reduce the accuracy of any numerical method. These computations used the tree methods of [13], and the large gradients distort the mesh because φ is used as a mesh refinement indicator.

When the solution φ is redistanced every ten steps, the large gradients disappear and mesh distortion is eliminated (see Fig. 17). Solution regularity is controlled by the regularity of the interface rather than the far-field values of F when frequent redistancing is applied.

6. CONCLUSION

We have reviewed the role of redistancing in level set methods for moving interfaces and presented fast redistancing algorithms based on trees, triangulation, interpolation, and a robust search strategy. Our algorithms are fast, straightforward, and extremely useful in adaptive level set methods for moving interface problems. We are currently applying

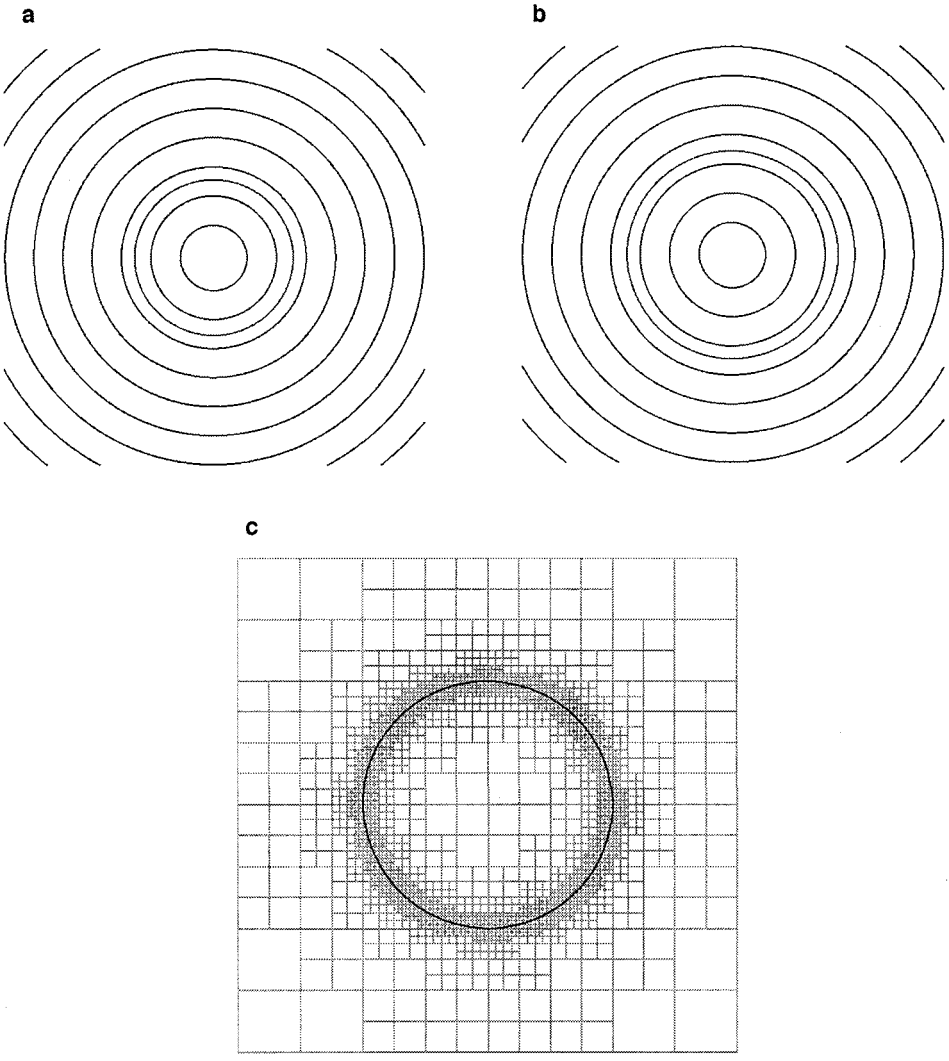

FIG. 17. A circular interface expanding with the velocity of Eq. (21), with redistancing applied every ten steps: (a) and (b) contours of the level set function at $t = 1$ and 2, (c) the quadtree mesh from the method of [13] at $t = 2$.

these algorithms to compute general velocity extensions, permitting the construction of completely modular level set methods [14] for solving general moving interface problems.

REFERENCES

1. D. A. Adalsteinsson and J. A. Sethian, *The Fast Construction of Extension Velocities in Level Set Methods*, Technical Report PAM-738, Center for Pure and Applied Mathematics, University of California, Berkeley, 1997.
2. S. Chen, B. Merriman, S. Osher, and P. Smereka, A simple level set method for solving Stefan problems, *J. Comput. Phys.* **135**, 8 (1997).

3. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications* (Springer-Verlag, Berlin, 1997).
4. H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *Comm. ACM* **29**, 669 (1986).
5. M. McAllister, D. Kirkpatrick, and J. Snoeyink, A compact piecewise-linear Voronoi diagram for convex sites in the plane, *Discrete Comput. Geom.* **15**, 73 (1996).
6. S. J. Osher and J. A. Sethian, Front propagation with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *J. Comput. Phys.* **79**, 12 (1988).
7. D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, *A PDE Based Fast Local Level Set Method*, CAM Report 98-25, Program in Computational and Applied Mathematics, University of California, Los Angeles, 1998.
8. A. Schmidt, Computation of three dimensional dendrites with finite elements, *J. Comput. Phys.* **125**, 293 (1996).
9. J. Sethian, *Level Set Methods* (Cambridge Univ. Press, Cambridge, UK, 1996).
10. J. A. Sethian and J. Strain, Crystal growth and dendritic solidification, *J. Comput. Phys.* **98**, 231 (1992).
11. J. R. Shewchuk, *Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator*, C code, School of Computer Science, Carnegie Mellon University, 1995.
12. E. Stein, *Singular Integrals and Differentiability Properties of Functions* (Princeton Univ. Press, Princeton, NJ, 1970).
13. J. Strain, Tree methods for moving interfaces, *J. Comput. Phys.*, in press.
14. J. Strain, Modular methods for moving interfaces, *J. Comput. Phys.*, in press.
15. M. Sussman, P. Smereka, and S. Osher, A level set approach for computing solutions to incompressible two-phase flow, *J. Comput. Phys.* **114**, 146 (1994).
16. J. Taylor, J. W. Cahn, and C. A. Handwerker, Geometric models of crystal growth, *Acta Met. Mat.* **40**, 1443 (1992).
17. C. Truesdell and R. A. Toupin, The classical field theories, in *Handbuch der Physik III/1*, edited by S. Flügge (Springer-Verlag, Berlin, 1960).
18. F. W. Wilson, R. K. Goodrich, and W. Spratte, Lawson's algorithm is nearly optimal for controlling error bounds, *SIAM J. Numer. Anal.* **27**, 190 (1990).
19. C. K. Yap, An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments, *Discrete Comput. Geom.* **2**, 365 (1987).