Journal of Computational Physics **170,** 373–394 (2001)
doi:10.1006/jcph.2001.6740, available online at http://www.idealibrary.com on IDEAL®

A Fast Semi-Lagrangian Contouring Method for Moving Interfaces

John Strain[1]

Department of Mathematics, University of California, 970 Evans Hall No. 3840, Berkeley, California 94720-3840
E-mail: strain@math.berkeley.edu

Received August 18, 2000; revised January 29, 2001

General moving-interface problems are solved by a new approach: evaluating an explicit semi-Lagrangian advection formula with efficient geometric algorithms and extracting the moving interface with a fast new contouring technique. The new approach decouples spatial and temporal resolutions, and grid-free adaptive refinement of the interface increases accuracy dramatically. A modular implementation, with a fast new intrinsic geometry module, computes highly accurate solutions to geometric moving-interface problems involving merging, anisotropy, and faceting; with a high-order embedded geometry module, it solves second-order problems involving curvature, dynamic topology, and nonlocal interactions. © 2001 Academic Press

1. INTRODUCTION

A new approach to numerical methods for general moving-interface problems is presented. We contour a semi-Lagrangian advection formula, evaluated with efficient geometric algorithms, to extract the moving interface. A fast new contouring technique controls the interface resolution independent of the time step, and grid-free adaptive refinement increases accuracy by orders of magnitude. Semi-Lagrangian advection merges and breaks complex topology, with stable time stepping independent of the interface resolution, while fast geometric algorithms resolve an N-element interface with optimal $O(N \log N)$ efficiency. Our implementation provides fast new intrinsic first-order and embedded second-order geometry evaluation modules for solving specific moving-interface problems.

The work extends the modular semi-Lagrangian moving-interface methods of [22–25]. The convenient and flexible modular black-box approach decouples into several modules with independently controllable resolution. Exact geometric algorithms are tuned for speed, velocity evaluation and time stepping are decoupled from interface resolution, and the

[1] Research supported by Air Force Office of Scientific Research Grant FDF49620-96-1-0201.

new contouring technique dramatically increases overall accuracy. This efficient adaptive framework combines the high resolution of front tracking [1] with the topological robustness of level sets [16].

We convert the moving-interface problem to a fixed-domain advection equation in Section 1.1 and present our semi-Lagrangian advection formula in Section 1.2. The formula is efficiently evaluated at any desired point by the fast tree-based geometric algorithms of Section 1.3, and the interface is extracted by the fast adaptive contouring technique of Section 2. Evaluation modules for geometric velocities are supplied in Section 3. A detailed two-dimensional implementation is presented in Section 4, where numerical results show that the method computes accurate viscosity solutions to a wide variety of geometric moving-interface problems involving anisotropic faceting, merging, corners, nonlocality, and curvature. Our algorithm extends naturally to three dimensions, and an implementation is in progress.

1.1. Advection of Moving Interfaces

An effective method for general moving interfaces should resolve complex nonsmooth interfaces, merging and evolving on disparate time scales. In Fig. 1, for example, (a) initially circular interfaces $\Gamma(t)$ shear into complex shapes under a passive transport velocity $V = V(x)$, (b) nonsmooth facets develop under a velocity $V = V(N)$ depending on the outward unit normal vector N, (c) circles merge and cusps emerge under unit normal velocity $V = N$, and (d) a complex polygon shrinks on widely varying time scales under curvature flow $V = C$.

Topological changes such as merging can be naturally resolved with an implicit representation of the interface $\Gamma(t)$ via its signed distance function

$$F(x,t) = \pm \min_{y \in \Gamma(t)} \|x - y\|. \tag{1}$$

If the specified velocity functional V on Γ is extended smoothly to a vector field $W(x,t)$ on $\mathbf{R}^d$, then solving the advection equation

$$F_t - W \cdot \nabla F = 0 \tag{2}$$

moves the zero set $\Gamma(t)$ of the initial data F with velocity V. Topological changes in $\Gamma(t)$ are discovered when the zero set is extracted from F by contouring.

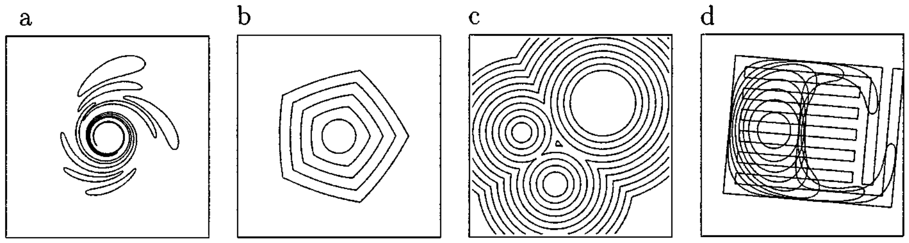

FIG. 1. Challenges in moving interfaces: (a) complexity, (b) faceting, (c) merging, and (d) varying time scales.

We solve the advection equation by the second-order semi-Lagrangian formula presented in Section 1.2. Three independent computational modules then transform the moving interface to an advection problem:

- a fast distancing algorithm similar to [22], which produces the exact signed distance F from a given interface Γ (Section 1.3),
- a fast extension algorithm from [25], which extends interfacial velocities V defined on Γ to global smooth velocities W defined everywhere on $\mathbf{R}^d$ (Section 1.3), and
- the fast adaptive contouring technique of Section 2, which extracts the zero set Γ from the solution F of the advection equation with user-specified accuracy.

1.2. Semi-Lagrangian Advection Formulas

We solve the advection equation (2) with an explicit semi-Lagrangian formula which allows us to extract only the zero set of the solution. Such formulas rely on the observation that Eq. (2) propagates solution values along characteristics $x = s(t)$ satisfying

$$\dot{s}(t) = -W(s(t), t). \tag{3}$$

Thus the solution $F(x, t+k)$ at time $t+k$ can be evaluated by solving the characteristic ODE (3) backwards in time from $x = s(t+k)$ to $s(t)$ and setting $F(x, t+k) = F(s(t), t)$. Semi-Lagrangian schemes solve the characteristic ODE numerically from $x = s(t+k)$ to $s(t)$ and approximate the off-grid value $F(s(t), t)$ by shape-preserving interpolation [14] or monotone advection [18]. (Our approach omits the approximation of F and evaluates the signed distance $F(s(t), t)$ exactly.) Their unconditional stability allows large time steps in linear advection [19, 20] and moving interfaces [23–25], leaving the time step controlled only by accuracy.

Semi-Lagrangian moving-interface methods which solve the advection equation by the first-order Courant–Isaacson–Rees (CIR) formula [3],

$$\tilde{G}(x) = F(\tilde{x}, t) = F(x + kW(x, t), t), \tag{4}$$

were developed and justified in [23, 24]. In [25], we combined the CIR predictor (4) with a second-order trapezoidal corrector formula,

$$G(x) = F(x + k\bar{W}(x, t), t), \qquad \bar{W}(x, t) = \frac{1}{2}W(\tilde{x}, t) + \frac{1}{2}\tilde{W}(x, t+k). \tag{5}$$

The predicted velocity $\tilde{W}$ extends the velocity $\tilde{V}$ determined by the zero set $\tilde{\Gamma}$ of $\tilde{G}$ at time $t+k$. This predictor–corrector pair is second-order accurate in time, explicit, and unconditionally stable in the maximum norm: each new G value is an exact F value, so the maximum of F can never increase.

In the context of a moving interface, semi-Lagrangian advection defines the numerical solution G by a simple explicit functional formula (5) at any desired evaluation point. Eulerian methods, by contrast, advance solution values on a grid and interpolate between grid points. Thus semi-Lagrangian methods find the new interface at $t+k$ by contouring a well-defined function without evaluating the advected solution away from its own zero set. Computational effort is naturally concentrated on the moving interface.

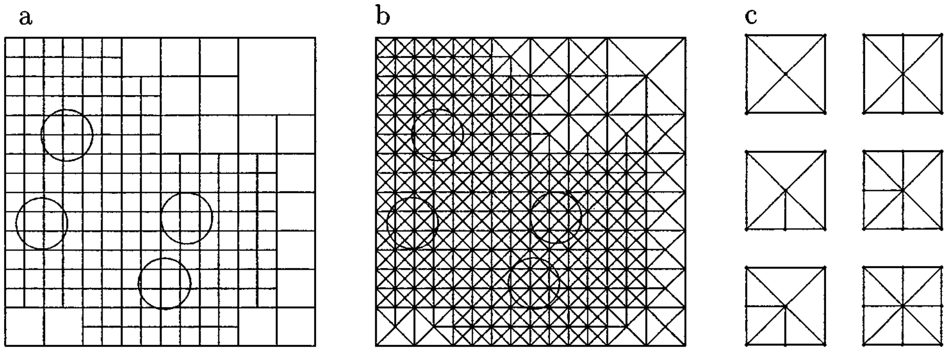

FIG. 2. (a) Four-level distance tree for a simple interface and (b) triangulation of its cell vertices and centers, built from the cell configurations in (c).

Since our advection velocity $W(x,t)$ extends the user-specified velocity functional V defined on the zero set $\Gamma(t)$ of $F(x,t)$, each semi-Lagrangian time step $\Gamma(t) \to \Gamma(t+k)$ involves the following substeps:

- **Evaluate** the signed distance F from the interface $\Gamma(t)$.
- **Evaluate** the interfacial velocity V of $\Gamma(t)$ by a user-supplied module.
- **Extend** V to a global advection velocity W.
- **Contour** the predicted CIR solution $\tilde{G}$ defined by Eq. (4) to get the predicted interface $\tilde{\Gamma}$.
- **Evaluate** the predicted interfacial velocity $\tilde{V}$ of $\tilde{\Gamma}$.
- **Extend** $\tilde{V}$ to a global advection velocity $\tilde{W}$.
- **Contour** the corrected trapezoidal solution G defined by Eq. (5) to get the corrected interface $\Gamma(t+k)$.

The distancing and extension substeps are efficiently implemented via the quadtree-based algorithms of Section 1.3, while effective algorithms for the contouring substeps are presented in Section 2. Geometric velocity-evaluation modules are described in Section 3.

1.3. Quadtree Meshes: Distancing and Extension

A *quadtree mesh* is composed of square cells organized into an L-level tree structure as in Fig. 2. Our fast algorithms for distancing and velocity extension are based on a specific quadtree developed in [22].

The distance tree. An interface Γ composed of N piecewise-linear elements of similar size can be efficiently resolved on a tree mesh built by splitting any cell C whose edge length exceeds its minimum distance

$$d(C,\Gamma) = \min_{x\in C} \min_{y\in\Gamma} \|x-y\| \tag{6}$$

to Γ. When the elements vary widely in size, computational efficiency benefits from a modified criterion: split cells whose concentric triples[2] intersect more than two adjacent elements. This adaptive resolution copes better with interfaces resolved over widely varying spatial scales. Such a tree allows fast $O(N \log N)$ evaluation of the signed distance function $F(x)$ on each cell C by the fast distancing algorithm from [25]:

[2] If $C = \{|x_i - a_i| \le r\}$, then the concentric triple T of C is defined by $T = \{|x_i - a_i| \le 3r\}$.

- **Start:** Set the current minimum distance m to Γ equal to ∞.
- **Loop:** While $l \geq 0$ and the cube $C(x, m)$ with center x and half-side length m is not completely contained in the concentric triple T of C, replace C by its parent C^*, find and record an element Γ_j^* nearest to x in the element list of C^*, replace m by the minimum distance m^* from x to Γ_j^*, and replace l by $l - 1$.
- **Sign:** Given a nearest element Γ_j^* to x, determine the sign of $F = \pm m^*$ by checking normal vectors of Γ_j^* and its neighbors.

This search strategy builds the distance tree in $O(N \log N)$ time and space complexity and always terminates in a bounded number of steps: in $d \leq 4$ dimensions, any element intersecting T is nearer to x than any element outside the triple T^{***} of the *great*-grandparent C^{***}.

Triangulation. For "balanced" quadtrees in which adjacent cells differ in size by no more than a factor of 2—such as the distance tree shown in Fig. 2a—cell vertices and centers can easily be triangulated into conforming meshes [4]. Each cell in such a tree has 0 to 4 smaller neighbors in $d = 2$ dimensions, so a triangulation can be built from the six possible cell configurations shown in Fig. 2c. The three-dimensional case is similar with more configurations.

Velocity extension. Moving interfaces via the advection equation (2) requires a globally defined velocity W which extends V smoothly off the interface $\Gamma(t)$. Some moving-interface problems suggest a natural velocity extension, but a modular widely applicable black-box solver requires a general problem-independent velocity extension as in [25].

If g is any continuous function on Γ, a nearest-point extension G is defined by $G(x) = g(y)$, where y is a point on Γ nearest to x, chosen arbitrarily if there are several points equidistant from x. The nearest-point extension G is continuous near smooth interfaces Γ but may be discontinuous at points x with several nearest neighbors. The numerical Whitney extension procedure of [25] guarantees continuity by building the continuous piecewise-linear interpolant W of the nearest-point extension G from the triangulated distance tree mesh. Both extensions satisfy a *maximum principle*: The maximum over $\mathbf{R}^d$ of W cannot exceed the maximum over Γ of g. When g is the velocity V of Γ, the maximum principle guarantees that regions of space far from Γ cannot move faster than the interesting nearby regions.

The nearest-point extension can be efficiently evaluated on a triangulated distance tree for Γ. When the tree is built, a pointer from each vertex and center to a nearest element of Γ is stored. $G(x)$ can then be evaluated in $O(1)$ time by finding a nearest point y on a known nearest element and setting $G(x) = g(y)$. Table I verifies the $O(N \log N)$ cost of building the distance tree and evaluating the nearest-point extension shown in Fig. 3.

TABLE I
CPU Seconds for Tree Building, Triangulation, and Extension

K	6	7	8	9	10	11	12
Cells	1421	2957	6029	12,173	24,461	49,037	98,189
Distance tree	0.04	0.11	0.24	0.57	1.24	2.81	6.35
Triangulation	0.02	0.05	0.11	0.24	0.51	1.11	2.32
Extension	0	0.01	0.01	0.02	0.03	0.07	0.14

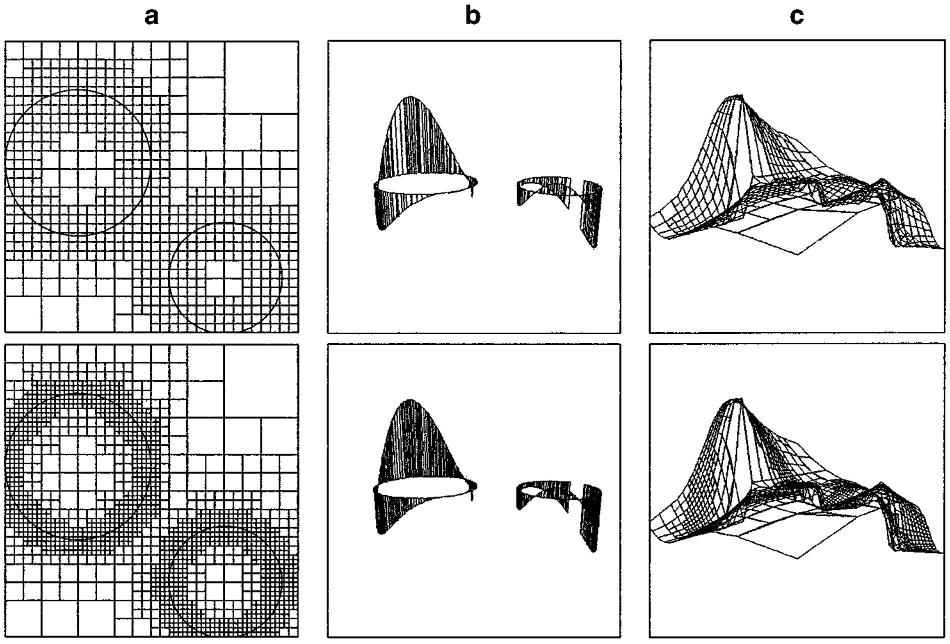

FIG. 3. (a) Interface and K-level distance tree, (b) interfacial velocity x-component V_x, and (c) nearest point extension W_x with $K = 5$ and 6.

At completely arbitrary points $x \in \mathbf{R}^d$, a distance tree does not guarantee efficient evaluation of the nearest-point extension. Points x located in large cells far from Γ may require searching long lists with $O(N)$ elements. However, the distance tree speeds up $G(x)$ evaluation for x near Γ, because such points are contained in small cells with few nearby elements where the search strategy is efficient. Thus we use the nearest-point extension for semi-Lagrangian contouring where effort is focused near the interface. The numerical Whitney extension can be efficiently evaluated everywhere by point location [6, 15] and local interpolation.

2. FAST ADAPTIVE CONTOURING

The *contouring problem*—extracting the zero set of a given continuous function G—naturally separates into two stages: topology resolution and contour approximation [2, 8, 11]. Topology resolution determines a consistent and hopefully correct topology for the set of curves, surfaces, and degenerate objects which make up the zero set. Each component of the zero set is enclosed in a collection of bounding boxes which separates it from the other components. Contour approximation iteratively refines an explicit representation of the approximate zero set for increased accuracy. We describe our topology resolution technique in Section 2.1 and adaptively refine the interface in Section 2.2. The two-dimensional algorithm presented generalizes immediately to three dimensions: triangulation becomes tetrahedralization and segments become triangles.

Contouring the semi-Lagrangian formula for moving interfaces imposes different requirements than many standard graphical applications of contouring. Our main issues are:

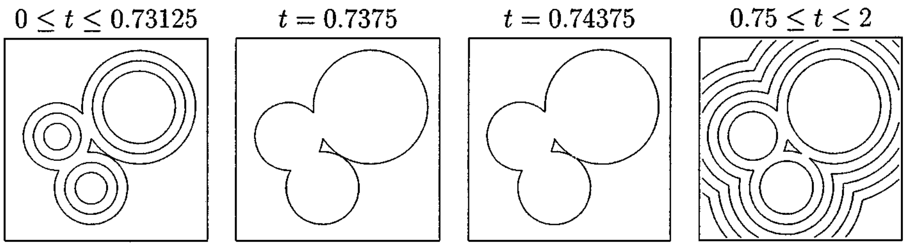

FIG. 4. Island formation at cusps in merging.

Accuracy. We require a contouring technique with high and controllable accuracy. The zero set found at each time step becomes the initial value for evaluating the velocity and moving the interface through the next time step. Therefore contouring errors of size α accumulate into global error $O(\alpha/k)$ per unit time.

Consistency. In the presence of inexact computer arithmetic, high accuracy requires globally correlated decisions to ensure a consistent interface topology—free of loops, crossing, dead ends, and so forth. In the context of moving interfaces where cusps form and pinch off, the exact topology of the numerical solution at time t may be irrelevant to the later evolution of the interface (Fig. 4). Thus consistency may be balanced against efficiency and accuracy.

Efficiency. Since we contour the solution twice per time step, the contouring technique must be efficient: computational effort should concentrate on the zero set. Standard contouring techniques obtain a set of smooth contours to high accuracy with a global uniform Cartesian mesh [13]. This is more than we want, at a price we cannot afford.

Derivatives. Many robust contouring techniques require derivatives of G, assume that G is a piecewise polynomial or spline, or assume that all zeroes of certain nonlinear systems involving G and ∇G can be found exactly [2, 8]. Evaluation of ∇G from a semi-Lagrangian formula requires differentiation of nonsmooth distance functions and extended velocities, so derivative-free contouring techniques are desirable. However, G is within $O(k)$ of the signed distance function F, which has $\|\nabla F\| = 1$ almost everywhere; thus a gradient bound $\|\nabla G(x)\| \le \gamma$ can usually be assumed and plays a key role in resolving a consistent topology.

Transversality. Contouring problems are well posed under the transversality condition that G and ∇G do not vanish simultaneously [2]. For the CIR formula $G(x) = F(x + kW(x))$ and certain special velocities W, this condition can be verified by exact computation. In moving interfaces, we usually assume transversality since we start from a fresh signed distance function with $\|\nabla F\| = 1$ at each time step.

2.1. Topology Resolution

We extract a consistent topology for the interface by building a K-level triangulated quadtree mesh, evaluating G at the mesh vertices, finding the exact zero set of the continuous piecewise-linear interpolant, and refining its vertices by bisection.

Meshing. The zero set Γ of G can be efficiently resolved on a triangulated quadtree mesh built by pretending that G is a signed distance function and splitting each cell C whose edge

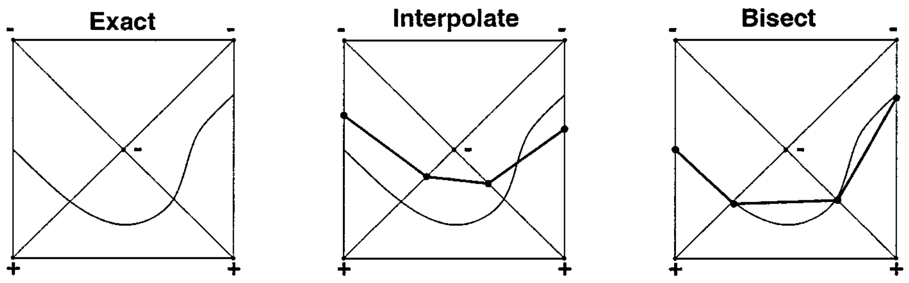

FIG. 5. Linear contouring and bisection of a zero set.

length exceeds the minimum value of $|G|$ on C. Since G is not a signed distance function, the resulting quadtree may be *unbalanced* and difficult to triangulate: neighboring cells vary in size by more than a factor of 2 (Fig. 2).

We can balance the quadtree by brute force [4] or by modifying the cell splitting criterion: fix an estimated gradient bound γ for $\|\nabla G\|$ and split every cell C whose size exceeds the minimum of $|G|/\gamma$ over C. If the bound $\|\nabla G\| \leq \gamma$ holds, then the resulting tree is balanced and easy to triangulate. Otherwise, we double γ and rebuild until satisfied.

Interpolation. Given nonzero function values $G(x)$ at the vertices of a triangulation, extracting the zero set of the continuous piecewise linear interpolant Q on the triangulation is straightforward. In $d = 2$ dimensions, for example, each triangle where G changes sign contains a unique line segment on which Q vanishes. These line segments form a polygonal curve because the triangulation is conforming and Q is continuous. Following each polygonal zero curve as far as possible in both directions produces an oriented component of $\Gamma(t+k)$ with $G > 0$ on its right (Fig. 5). In $d = 3$ dimensions the contouring process is similar: tetrahedra replace triangles and triangular patches replace line segments.

Exact zero vertex values produce ambiguities in extracting the zero set of Q, we choose a small tolerance β such as 10^{-10} and perturb vertices where $|G| \leq \beta$ until all vertex values of $|G|$ exceed β. If repeated perturbations fail, transversality is doubtful and the contouring problem may be ill posed.

Table II verifies the predicted $O(N \log N) = O(K 2^K)$ cost and $O(2^{-2K}) = O(h^2)$ accuracy of linear interpolation on the triangulated quadtree, applied to the interface of Fig. 6. The error reported is the maximum of the exact distance function $|G|$ over segment endpoints and midpoints.

Bisection. Before declaring the interface topology resolved, we move each interface vertex to satisfy $|G| \leq \beta$. Each interface vertex found from the linear interpolant is known

TABLE II
Error and CPU Seconds for Linear Contouring

K	3	4	5	6	7	8	9
Cells	85	301	865	2041	4445	9309	19,005
Segments	47	114	217	436	889	1819	3610
Length	0.48	0.214	0.104	0.0604	0.0305	0.0150	0.00817
Error	0.191	0.121	0.0323	0.0099	0.00296	0.000828	0.000214
CPU	0.01	0.01	0.03	0.07	0.15	0.35	0.71

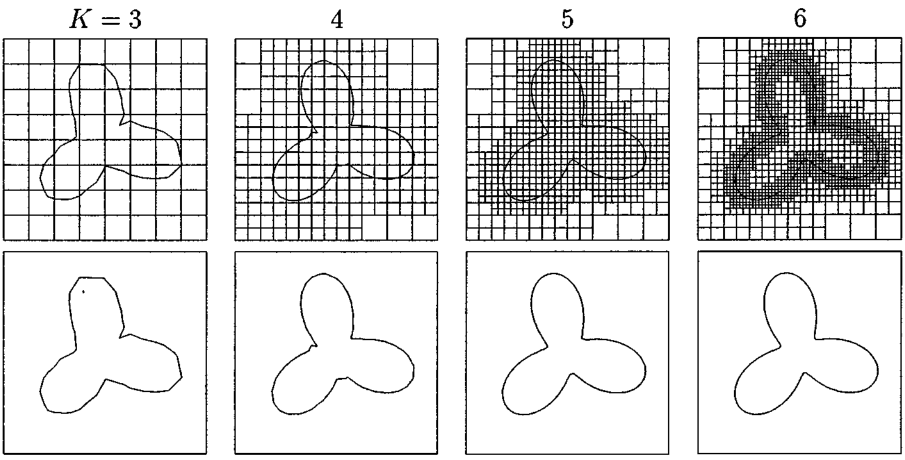

FIG. 6. Contouring a propeller shape on a triangulated K-level tree mesh.

to lie on the edge of some triangle Δ, bracketed by two triangle vertices with G values of opposite sign. A standard one-dimensional bisection algorithm [12] applied along this edge is guaranteed to produce a zero of G on this edge to accuracy β in $O(\log \beta)$ evaluations of G (Fig. 5). In three-dimensional problems, the interface is a collection of triangles in $\mathbf{R}^3$, and each triangle vertex moves along an edge of the tetrahedralization. The resulting interface enjoys $|G|$ values below β at every vertex and shares the topology of the piecewise linear interpolant on the triangulation. In Section 2.2, we adjust the patches which connect these highly accurate vertices, to improve the resolution of the interface without changing the interface topology.

2.2. Contour Approximation

Given a collection of piecewise linear patches with a consistent topology and all vertices within $O(\beta)$ of the exact interface, we now improve the resolution of the underlying zero set Γ by P passes of the following adaptive refinement algorithm (Fig. 7).

Split. Any patch where $|G|$ exceeds a specified tolerance α at the midpoint is subdivided into patches of half the size. New vertices are interpolated linearly between existing vertices. Since $|G| > \alpha$ at the new vertices, the following steps move them to reduce $|G|$.

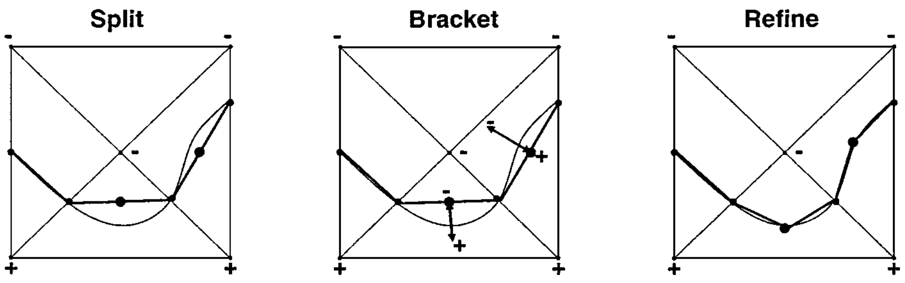

FIG. 7. Adaptive refinement of a zero set.

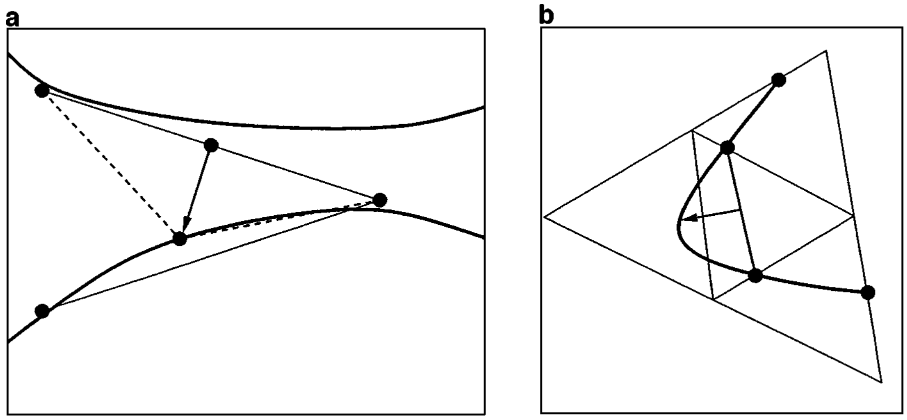

FIG. 8. Safeguards in adaptive refinement: (a) topology preservation and (b) triangle limiting.

Bracket. New interface vertices, unlike the linear interpolant vertices of Section 2.1, are not bracketed between triangulation vertices where G changes sign. Thus bisection requires a preliminary bracketing step which finds a point where G has opposite sign from the new vertex. In order to keep vertices from bunching together and losing resolution, we seek brackets along the normal vector to the interface at the new vertex x at distance equal to the patch radius. The bracketing distance is reduced if necessary to avoid collisions with adjacent segments (Fig. 8).

Bisect. For each bracket found, we run the bisection algorithm to obtain an approximate zero within tolerance β. If more than one approximate zero is found, we accept the one nearest to the new vertex, subject to the following topological safeguards.

Safeguards. We avoid adjacent patch crossings by rejecting any new vertices which would produce a tiny angle $|\theta| \leq 2.56^\circ$ with either neighbor. Such crossings are possible when adaptive refinement attempts to compensate for incorrect topology in the linear interpolant (Fig. 8). The topology of the linear interpolant can be preserved by preventing new vertices from leaving the triangle where they start, but this sacrifices too much accuracy for the sake of topology preservation. Thus we allow new vertices to move to adjacent triangles, but only if the destination is empty (Fig. 8).

Prune. In the final pruning step, we produce a more uniform resolution of the interface by deleting patches with areas less than ρ times smaller than their neighbors. Pruning produces neighboring patches of comparable sizes, yielding more accurate normal vectors when the intrinsic geometry module of Section 3.1 is used.

The $O(\alpha^{-1/2})$ cost and $O(\alpha)$ accuracy of the segment midpoints produced by this algorithm for the smooth interface of Fig. 6 are verified in Table III.

3. LOCAL GEOMETRIC VELOCITIES

Solving any moving-interface problem with our modular approach requires a user-supplied module which evaluates the interfacial velocity functional. We provide two modules for geometric velocities $V = V(x, t, N, C)$ which depend on the local position and geometry of the interface. These velocity functionals pose numerical difficulties because

TABLE III
Error and CPU Seconds for Adaptive Contouring of a Propeller

K/α	$3/10^{-1}$	$4/10^{-2}$	$5/10^{-3}$	$6/10^{-4}$	$7/10^{-5}$	$8/10^{-6}$	$9/10^{-7}$
Cells	85	301	865	2041	4445	9309	19,005
Segments	32	82	245	757	2344	7382	23,683
Length	0.55	0.227	0.101	0.049	0.027	0.011	0.0034
Error	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
CPU	0.02	0.04	0.09	0.26	0.78	2.38	7.41

standard formulas for N and C are complicated and their numerical approximation is sensitive. Thus we evaluate normal vectors by intrinsic methods in two dimensions and curvature by embedded methods in general dimension.

3.1. Intrinsic Geometry

Suppose Γ is a curve in $\mathbf{R}^2$, approximated by a polygon with edges $x_i x_{i+1}$. The exact normal vector and curvature are given by the intrinsic formulas [5]

$$N = \frac{x_s^\perp}{\|x_s\|} = (\cos\theta, \sin\theta), \qquad C = \frac{1}{\|x_s\|}\frac{d}{ds}\theta(s), \qquad (x, y)^\perp = (-y, x),$$

where $x(s)$ is a parametrization of Γ. For any function $g(s)$ on Γ, a natural discretization of the vertex derivative $\frac{dg}{ds}$ is the left–right average difference

$$\Delta g_i = \frac{1}{2}\left(\frac{g_{i+1} - g_i}{\|x_{i+1} - x_i\|} + \frac{g_i - g_{i-1}}{\|x_i - x_{i-1}\|}\right).$$

With $g = x$ this gives an approximate normal

$$N_i = \frac{\Delta x_i^\perp}{\|\Delta x_i\|}. \tag{7}$$

Since the normal angle θ satisfies

$$N^\perp \cdot \frac{dN}{ds} = \theta_s,$$

we can approximate curvature by

$$C_i = \frac{1}{\|\Delta x_i\|} N_i^\perp \cdot \Delta N_i. \tag{8}$$

Equation (7) produces first-order-accurate normal vectors if Γ is smooth, but Eq. (8) often produces inaccurate oscillatory curvature. Thus we present an alternative module based on embedding Γ into a local uniform mesh.

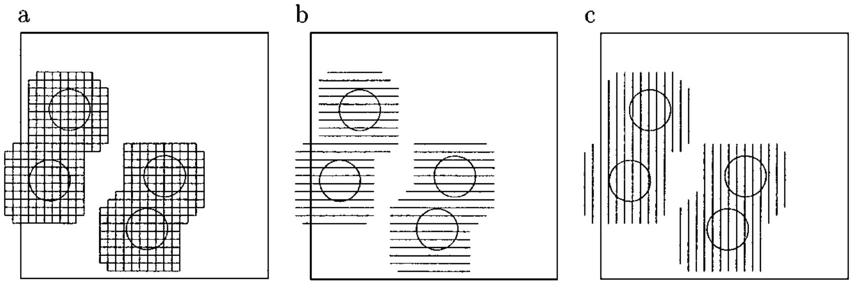

FIG. 9. Local embedded mesh with range $R = 3$ for a simple interface (a), viewed as x-intervals in (b), and as y-intervals in (c).

3.2. Embedded Geometry

Curvature can be accurately computed from the signed distance function F via the formulas [26]

$$N = \frac{\nabla F}{\|\nabla F\|}, \qquad C = -\nabla \cdot N, \tag{9}$$

which embed the interface Γ as a subset of $\mathbf{R}^d$. We generalize the velocity-evaluation method of [25] to build a uniform mesh near Γ, evaluate F exactly at mesh points, apply high-order essentially nonoscillatory (ENO) differentiation formulas [10] to extract smooth accurate approximations to N and C, and interpolate back to the vertices of Γ.

Embedded mesh. First, we build a local uniform mesh near Γ and evaluate F. The simplest technique, marking nearby points of a global uniform mesh, is prohibitively expensive for fine meshes. Thus we employ the efficient sorting and pruning technique of [25].

A two-dimensional local mesh with mesh size h (Fig. 9) can be viewed as a collection of disjoint x-intervals $(i_L : i_R, j) = \{(ih, jh) \mid i_L \le i \le i_R\}$, or as a similar collection of y-intervals. It can be built by listing every mesh point within *horizontal* distance Rh of any interface point $y \in \Gamma$ and then listing each mesh point within *vertical* distance Rh of some point produced in the horizontal pass. Efficient construction algorithms are ensured by sorting and pruning local mesh points listed more than once. The local mesh is stored in a data structure which contains the mesh points (ih, jh), a list of x-intervals $(i_L : i_R, j)$, and so forth and includes every point necessary to form a difference stencil of half-width R for differentiating or interpolating to any interface point $y \in \Gamma$. The three-dimensional case is similar. The signed distance function F can be efficiently evaluated at the local mesh points—which are close to the interface—by fast exact distancing with the distance tree.

Differentiation. Equidistant centered difference formulas oscillate if their stencils cross points where F is not smooth. Thus we compute N and C on the local mesh by ENO methods which shift equidistant stencils to avoid corners in F. We apply S passes of cosine smoothing to reduce oscillations further and satisfy CFL conditions [25] and then interpolate N and C back to the vertices of Γ and apply the user-specified velocity $V(x, t, N, C)$.

4. IMPLEMENTATION AND NUMERICAL RESULTS

We demonstrate the accuracy and efficiency of the semi-Lagrangian contouring approach by computing solutions to a wide variety of moving-interface problems. We describe control

TABLE IV
Control Parameters and Default Values

Name	Default	Explanation
N	$40 \dots 640$	Number of time steps on $[0, a]$
k	a/N	Time step
K	$3 \dots 7$	Depth of contouring tree mesh
P	$0 \dots 4$	Depth of adaptive refinement
α	10^{-K}	Midpoint error tolerance for adaptive refinement
β	10^{-10}	Bisection tolerance
ρ	10^{-1}	Segment pruning tolerance
M	2^{-1-K}	Embedded $M \times M$ mesh size
E	3	Order of ENO differentiation on embedded mesh
S	$0 \dots 3$	Number of cosine smoothing passes on embedded mesh
L	$K + P$	Depth of distance tree

parameters and their default values in Section 4.1, discuss convergence testing in Section 4.2, and outline our implementation in Section 4.3. We measure dissipation for a propeller under passive rotation in Section 4.4. First-order geometric velocities depending only on N and inducing anisotropy and merging are treated in Section 4.5. In Section 4.6, we demonstrate convergence for second-order curvature-dependent velocities, while motion under a second-order nonlocal geometric velocity is computed in Section 4.7.

4.1. Control Parameters

Our experiments vary the initial interface $\Gamma(0)$, the velocity functional V, the spacetime domain $[0, a] \times [-b, b]^2$, and the control parameters summarized in Table IV. Convergence studies proceed by refining N, K, P, and α with other parameters set to default values. The distance tree depth L does not affect the error and is set to $K + P$, so that each distance tree cell encloses a fixed number of segments for efficiency. We set $S = 0$ except for strongly curvature-dependent velocities, where stability requires S to increase logarithmically with embedded mesh size.

4.2. Convergence Studies

We carry out both exact and graphical convergence studies. When the exact signed distance function F is known, we tabulate maximum interfacial errors

$$E_\infty = \max_{x \in \Gamma(t)} |F(x, t)| \tag{10}$$

and CPU seconds per step T. We analyze convergence with 5×5 tables of E_∞ and T where $N \leftarrow 2N$, $K \leftarrow K + 1$, and $\alpha \leftarrow \alpha/10$ as the row increases, while $P \leftarrow P + 1$ as the column increases. Thus each row exhibits first-order spatial convergence until the $O(k^2)$ error takes over, while the $P = 4$ column exhibits second-order temporal convergence: the $O(\alpha/k)$ accumulated spatial error is $o(k^2)$ since $\alpha = o(k^3)$. Figure 10 demonstrates first-order convergence $E_\infty = O(T^{-1})$ for three cases with exact solutions: rigid rotation (Section 4.4), merging circles (Section 4.5.2), and curvature flow (Section 4.6.1).

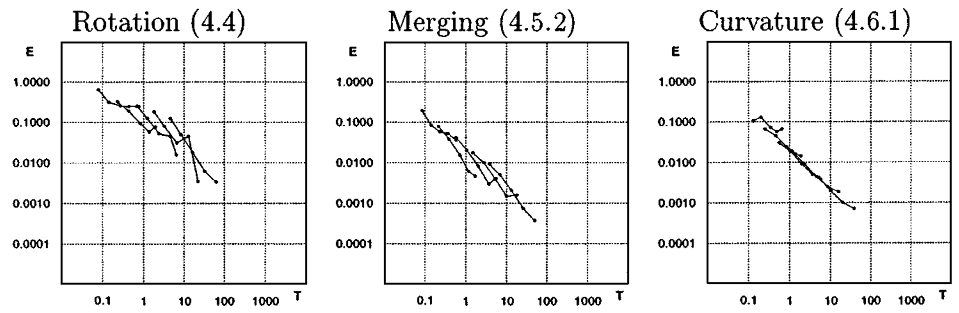

FIG. 10. Errors vs CPU times for exactly solvable examples.

For complex problems where exact solutions are not available, we demonstrate convergence to graphical accuracy by superimposing coarse and fine computations. We label a computation with given values for N, S, K, P, and α as a $N/S/K/P/\alpha$ run for brevity.

The convergence of the numerical solution (modulo the time-stepping error) as P increases with fixed k supports the conclusions of [20, 23, 25]: semi-Lagrangian methods—unlike Eulerian methods—do not suffer from Courant–Friedrichs–Lewy (CFL) stability conditions.

4.3. Implementation

The semi-Lagrangian contouring approach was implemented in Standard C, compiled with the Sun C compiler and the `-fast` optimization flag, and run on one 450-MHz CPU of a Sun Ultra 60 under Solaris 7. Timings reported are preliminary as the code has not been tuned for maximum speed. Each time step $t \to t + k$ of our algorithm begins with $\Gamma = \Gamma(t)$ and produces $\Gamma(t + k)$ by the following steps:

- $[D, F] = \texttt{DistanceTree}(L, \Gamma)$
  [Build an L-level triangulated distance tree D around Γ.]
- $V = \texttt{Velocity}(t, \Gamma)$
  [Call a user-supplied module to evaluate the velocity on the interface.]
- $\tilde{\Gamma} = \texttt{Contour}(K, P, \alpha, \beta, \rho, \tilde{G}(x) = F(x + kW(x, t), t))$
  [Find the zero set $\tilde{\Gamma}$ of the predicted CIR solution $\tilde{G}$.]
- $[\tilde{D}, \tilde{F}] = \texttt{DistanceTree}(L, \tilde{\Gamma})$
- $\tilde{V} = \texttt{Velocity}(t + k, \tilde{\Gamma})$
- $\Gamma(t + k) = \texttt{Contour}(K, P, \alpha, \beta, \rho, G(x) = F(x + k\overline{W}(x, t), t))$

Once the distance tree has been built and nearest points have been stored, the nearest-point extension W of the velocity V requires no additional effort to evaluate. Therefore the extension steps from Section 1.2 can be omitted.

Semi-Lagrangian evaluation. The contouring algorithm requires values of $\tilde{G}(x) = F(\tilde{x}, t)$ constructed by the following sequence of operations (Fig. 11):

- $[y, \sigma] = \texttt{SignClosestPoint}(\Gamma, D, x)$
  [Find the childless distance tree cell C in D containing x; search interface elements in the concentric triples of C and its ancestors C^*, C^{**}, and C^{***}; find a nearest point $y \in \Gamma(t)$ to x and the sign $\sigma = \pm 1$ of F at x.]

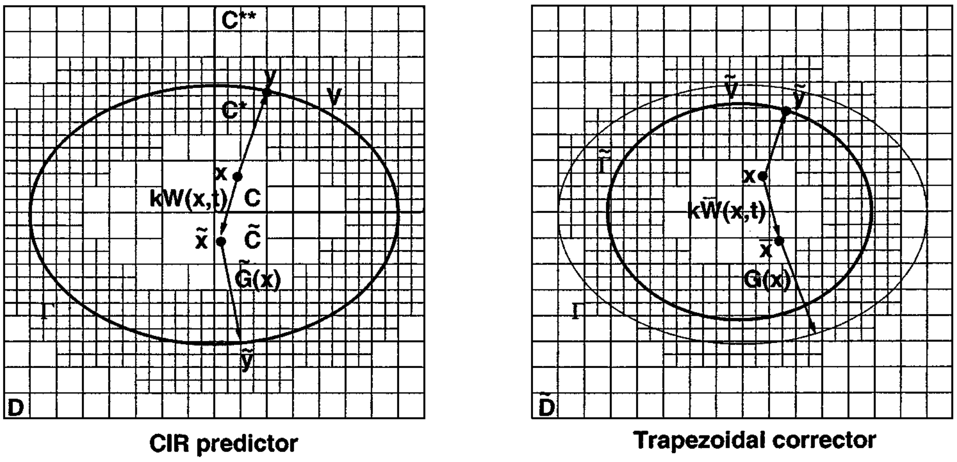

FIG. 11. Evaluation of the CIR predictor and the trapezoidal corrector at a point x.

- $W(x, t) = V(y, t)$
  [Evaluate the nearest-point extension of V at x.]
- $\tilde{x} = x + kW(x, t)$
  [Project x backward to the predicted characteristic point $\tilde{x}$.]
- $[\tilde{y}, \tilde{\sigma}] = \texttt{SignClosestPoint}(\Gamma, D, \tilde{x})$
- $\tilde{G}(x) = F(\tilde{x}, t) = \tilde{\sigma}\,\|\tilde{x} - \tilde{y}\|$.
  [Evaluate the exact signed distance from $\tilde{x}$ to $\Gamma(t)$.]

Values of $G(x) = F(x + k\overline{W}(x, t), t))$ are found with additional operations involving the predicted interface $\tilde{\Gamma}$ and its velocity $\tilde{V}$ (Fig. 11).

Contouring. Given the ability to evaluate any function G at arbitrary points, the contouring of G proceeds as follows (Figs. 5 and 7):

- $T = \texttt{Tree}(K, G, \gamma)$
  [Build K-level triangulated tree mesh to resolve the zero set of G with gradient bound γ.]
- $\Gamma = \texttt{ZeroSet}(\epsilon, T, G)$
  [Build zero set of linear interpolant to G on triangulation, perturbing triangulation vertices by ϵ as necessary.]
- $\Gamma = \texttt{Bisect}(\Gamma, T, \beta)$
  [Refine zero set vertices by bisection along triangle edges to accuracy β.]
- $\Gamma = \texttt{Adapt}(\Gamma, G, \alpha, \beta, \rho)$
  [Split segments with midpoint $|G|$ values above α, refine new vertices by bracketing and bisection along the normal vector to accuracy β, and delete segments smaller than ρ times their neighbors.]

Velocity evaluation. We evaluate geometric velocities V with the geometry modules of Section 3. The intrinsic module is a single straightforward step, while the embedded module involves the following substeps:

- $U = \texttt{BuildLocalMesh}(M, R, \Gamma)$
  [Build $M \times M$ local uniform mesh U within range $R = E + S$ of Γ.]

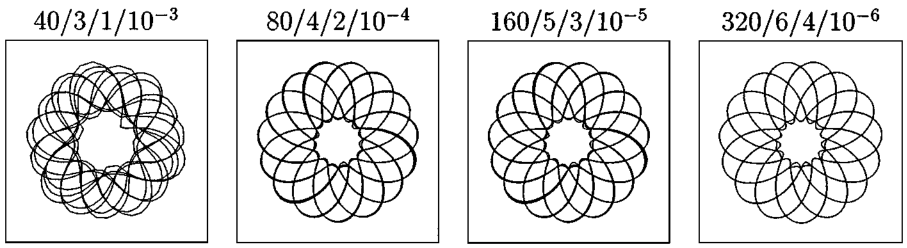

FIG. 12. A propeller shape under rigid rotation: two periods.

- $F = \mathtt{Distance}(U, D)$
  [Evaluate exact signed distance function at mesh points in U with distance tree D.]
- $[n, c] = \mathtt{EmbeddedGeometry}(U, F, E)$
  [Evaluate normal vector and curvature on local uniform mesh with order-E ENO evaluation of the embedded formulas in Eq. 9).]
- $[N, C] = \mathtt{Interpolate}(U, n, c, E, \Gamma)$
  [Interpolate geometric quantities from local uniform mesh U to interface Γ with order-E ENO technique.]
- $V = \mathtt{Velocity}(N, C, \Gamma)$
  [Evaluate local velocity functional on interface Γ with user-supplied module and interpolated N and C.]

We use the intrinsic approach for passive transport (Section 4.4) and first-order geometry involving only the normal direction (Section 4.5). The embedded approach produces smoother and more accurate curvature for the second-order geometric examples of Sections 4.6 and 4.7.

4.4. Passive Transport

We measure the dissipation of our method with a three-bladed propeller under rigid-body rotation $V(x, y) = (-y, x)$ (Fig. 12). This velocity is naturally defined everywhere, but we evaluate V only at the vertices of $\Gamma(t)$ and extend by the general technique of Section 1.3; thus Table V provides a realistic picture of the errors after two rotations at $t = 4\pi$ and supports the theoretical predictions of Section 4.2.

TABLE V
Error E_∞ and CPU Seconds T per Step for a Rotating Propeller

N	S	K	α		$P = 0$	1	2	3	4
40	0	3	10^{-3}	E_∞	0.648	0.320	0.256	0.2462	0.247
				T	0.079	0.142	0.272	0.445	0.770
80	0	4	10^{-4}	E_∞	0.325	0.193	0.0944	0.0580	0.0778
				T	0.232	0.434	0.834	1.39	1.97
160	0	5	10^{-5}	E_∞	0.255	0.126	0.0527	0.0454	0.0158
				T	0.695	1.25	2.41	4.54	6.46
320	0	6	10^{-6}	E_∞	0.183	0.0788	0.0311	0.0455	0.00347
				T	1.83	3.32	6.67	12.9	22.4
640	0	7	10^{-7}	E_∞	0.124	0.0503	0.0180	0.00647	0.00337
				T	4.64	8.36	16.5	33.3	62.5

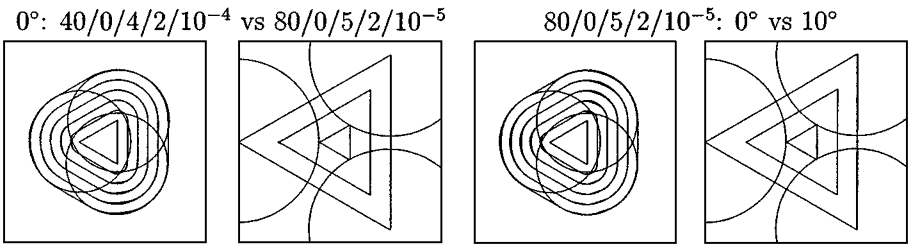

FIG. 13. Triangles growing and shrinking with unit normal velocity: the circles illustrate Huygens' principle.

4.5. First-Order Geometric Velocities

We demonstrate the topological robustness of the semi-Lagrangian contouring approach by computing corners and smooth shapes, growing and merging with unit normal velocity, and regularly faceted shapes developing under anisotropic normal velocities corresponding to nonconvex Hamiltonians.

4.5.1. *Viscosity Solutions with Corners*

Computation of correct "viscosity solutions" for faceted interfaces depends on moving a corner with unit normal velocity. Inward motion should keep corners sharp, while outward motion should produce rounded corners.

The semi-Lagrangian formula satisfies Huygens' principle and naturally computes the correct viscosity solution for a triangle growing and shrinking with unit normal velocity. Figure 13 superimposes $40/0/4/2/10^{-4}$ over $80/0/5/2/10^{-4}$ runs to show convergence to graphical accuracy. Runs at resolution $80/0/5/2/10^{-5}$ and tilted 0° versus 10° show that grid orientation effects are negligible. Huygens' principle is graphically verified.

4.5.2. *Cusps in Merging Circles*

Sharp inward-pointing corners naturally occur when smooth interfaces merge, so we verify correct behavior with the convergence study of merging circles shown in Fig. 14. Cusp singularities and topological changes are computed automatically and accurately. Tiny bubbles naturally form and disappear at the cusps where tangent circles merge, illustrating the importance of consistent rather than correct topology resolution. The errors shown in Table VI and plotted in Fig. 10 surpass theoretical expectations, achieving first-order accuracy in the maximum norm even for this nonsmooth solution.

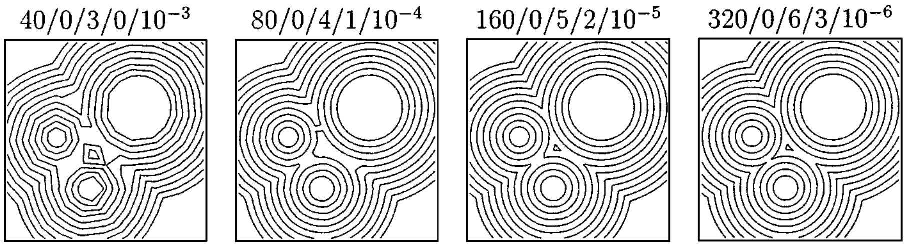

FIG. 14. Circles merging with unit normal velocity.

TABLE VI
Error E_∞ and CPU Seconds per Step T for Merging Circles

N	S	K	α		$P = 0$	1	2	3	4
40	0	3	10^{-3}	E_∞	0.194	0.0854	0.0432	0.0344	0.0219
				T	0.085	0.141	0.230	0.360	0.572
80	0	4	10^{-4}	E_∞	0.0788	0.0386	0.0156	0.00616	0.00470
				T	0.212	0.384	0.710	1.17	1.69
160	0	5	10^{-5}	E_∞	0.0423	0.0207	0.00826	0.00298	0.00133
				T	0.576	1.05	2.00	3.63	5.50
320	0	6	10^{-6}	E_∞	0.0175	0.0101	0.00401	0.00149	0.000831
				T	1.49	2.73	5.22	9.98	17.6
640	0	7	10^{-7}	E_∞	0.00924	0.00509	0.00210	0.000863	0.000378
				T	3.73	6.84	13.0	25.3	49.5

4.5.3. *Anisotropic Normal Velocity and the Wulff Limit*

Anisotropic motion along the normal vector connects moving interfaces to Hamilton–Jacobi equations $F_t + H(\nabla F) = 0$, which encounter difficulties when the Hamiltonian H is nonconvex [7], for example, if

$$V = R + \epsilon \cos(m\theta) \qquad \text{where } R + \epsilon(1 - m^2) < 0 < R - |\epsilon|. \tag{11}$$

Our semi-Lagrangian approach deals effectively with nonconvex Hamiltonians.

We evolve an initially circular interface using Eq. (11) with $R = \pm 1$ and $|R| + \epsilon(1 - m^2) = -4$. Figure 15 shows the development of the faceted interface, superimposing computations tilted at 0°, 10°, and 180°/m for anisotropies $m = 3, 4, 5$, and 6. Grid effects are negligible, convergence is fast, and the interface evolves rapidly into the regularly faceted Wulff shape [27] with the correct anisotropy.

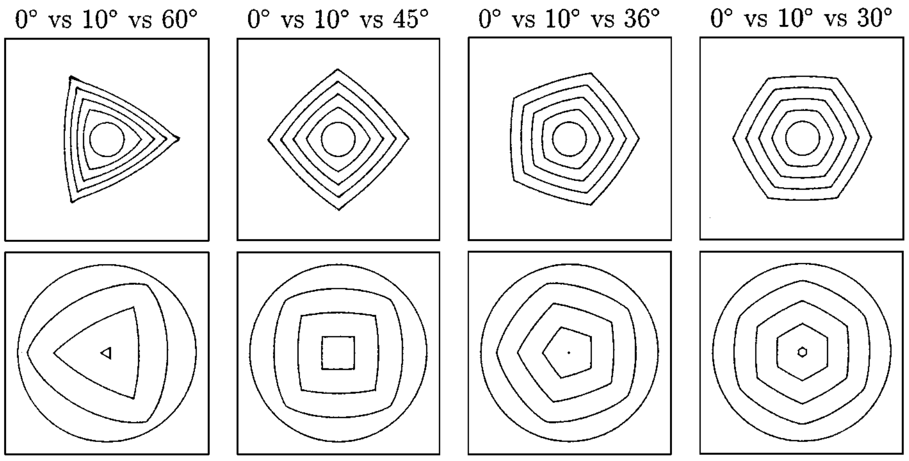

FIG. 15. Circles growing and shrinking into asymptotic Wulff shapes.

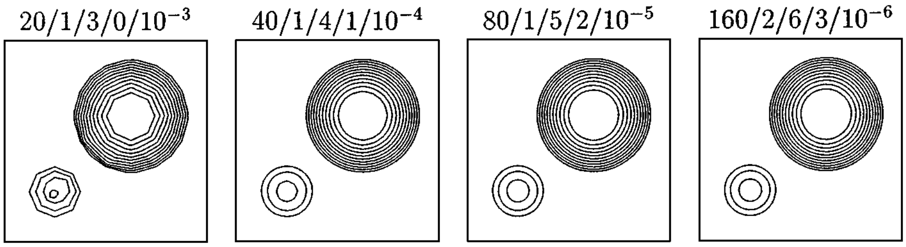

FIG. 16. Circles shrinking under curvature flow.

4.6. Second-Order Geometric Velocities

Geometric velocities involving second-order derivatives, such as curvature generate parabolic advection equations. Semi-Lagrangian formulas are designed for hyperbolic advection and require small time steps or additional smoothing to maintain stability for curvature flows [23, 25]. We compute the curvature with an embedded mesh of size $M = 2^{K+1}$ and S passes of cosine smoothing after each differentiation. S is increased logarithmically as the resolution is refined, to maintain stability without compromising efficiency and accuracy. The stability of the time-stepping scheme depends on the curvature resolution M and smoothing S, but not on the "subgrid" resolution produced by P passes of adaptive interface refinement.

4.6.1. *Circles Shrinking under Curvature*

A classic geometric problem shrinks a plane curve with velocity equal to its curvature and forms a useful second-order test case. A circle shrinking with $V = C$ has exact radius $R(t) = \sqrt{R(0)^2 - 2t}$; thus with $R(0) = \sqrt{5}$, a circle should shrink to radius 1 at time $t = 2$. A smaller circle with $R(0) = 1$ vanishes completely in time $t = 1/2$. Figure 16 shows convergence to graphical accuracy on $0 \leq t \leq 2$. CPU seconds per step and maximum errors in the interface location at $t = 2$ are reported in Table VII and Fig. 10, and they verify the theoretical predictions of Section 4.2.

TABLE VII
Error E_∞ and CPU Seconds per Step T for Curvature Flow

N	S	K	α		$P = 0$	1	2	3	4
20	1	3	10^{-3}	E_∞	0.106	0.130	0.072	0.057	0.057
				T	0.131	0.200	0.346	0.483	0.645
40	1	4	10^{-4}	E_∞	0.0664	0.0454	0.0241	0.0160	0.0142
				T	0.253	0.462	0.854	1.40	1.90
80	1	5	10^{-5}	E_∞	0.0305	0.0186	0.00909	0.00501	0.00376
				T	0.557	1.04	1.99	3.65	5.85
160	2	6	10^{-6}	E_∞	0.0188	0.00894	0.00443	0.00246	0.00186
				T	1.15	2.35	4.59	8.62	15.9
320	3	7	10^{-7}	E_∞	0.00684	0.00419	0.00200	0.00101	0.000711
				T	2.87	5.20	9.97	19.6	38.4

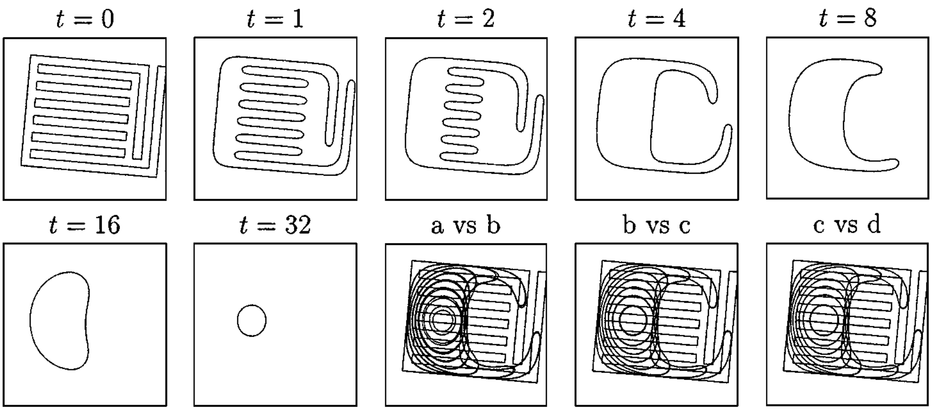

FIG. 17. Tilted polygon shrinking under curvature flow: runs a through d have parameters 320/1/5/4/10^{-3} through 2560/4/8/10^{-6}.

4.6.2. *Nonconvex Interfaces under Curvature*

A geometric theorem [9] predicts that any smooth embedded plane curve should collapse to a round point and vanish in finite time under curvature flow $V = C$. We illustrate the theorem for a complex polygonal shape with the graphical convergence study shown in Fig. 17. The curvature velocity displays fast-moving infinite transients at initial sharp corners, an intermediate regime of smooth motion, and infinite velocity again as the interface vanishes. Our adaptive approach easily converges to graphical accuracy despite the wide range of spatiotemporal scales.

4.7. A Nonlocal Geometric Velocity

Moving-interface problems, such as crystal growth [17, 21] are *nonlocal*—the normal velocity at each point depends on all of $\Gamma(t)$ and even on its history $\{\Gamma(s) \mid 0 \leq s \leq t\}$. The

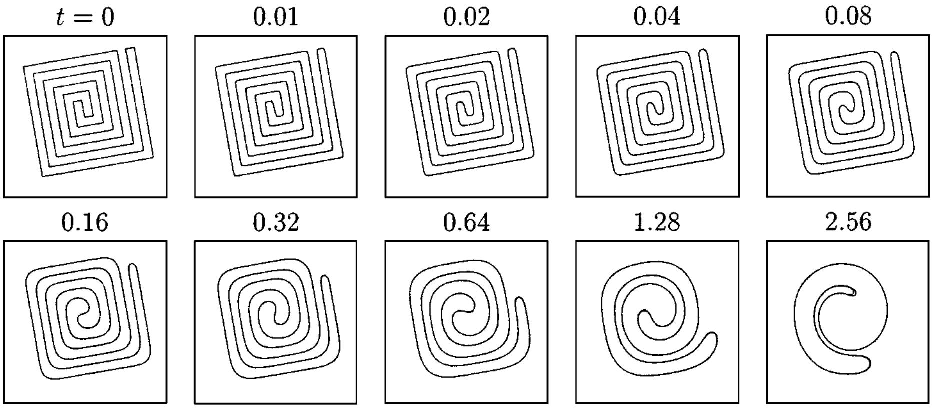

FIG. 18. Spiral unwinding under nonlocal volume-preserving curvature flow.

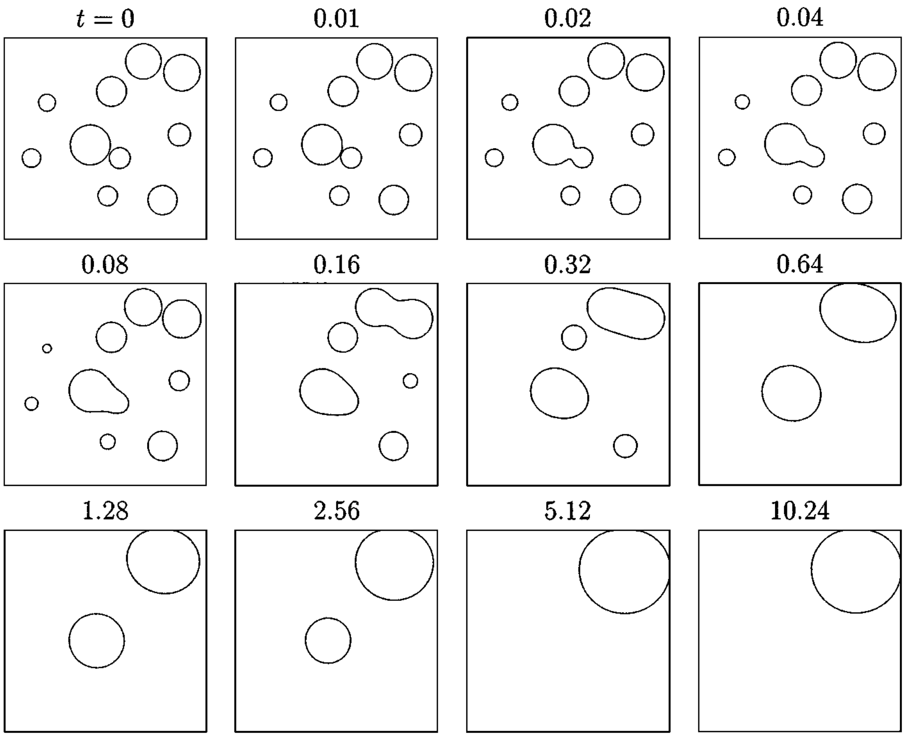

FIG. 19. Bubbles merging under nonlocal volume-preserving curvature flow.

simplest nonlocal geometric velocity

$$V = \left(C - \frac{\int_{\Gamma(t)} C ds}{\int_{\Gamma(t)} 1 ds}\right) N \tag{12}$$

smooths the moving interface by curvature while preserving the area inside the interface, so arbitrary shapes become round but the interface does not vanish. Small components disappear as their area is transferred to large ones.

We study a tilted square spiral unwinding under velocity (12) in Fig. 18, where 512/2/7/3/10^{-3} and 1024/3/8/3/10^{-4} runs converge to graphical accuracy. The interface is shown at geometrically increasing times $t = 0, 0.01, 0.02, \ldots, 2.56$: its motion slows dramatically as curvature variation decreases toward its final steady state. Figure 19 shows complex dynamic merging of initially circular bubbles under velocity (12), with parameters 2048/3/7/4/10^{-4}. Interfaces merge and disappear accurately and stably even under this second-order nonlocal velocity.

ACKNOWLEDGMENTS

The author thanks the referees for many helpful suggestions.

REFERENCES

1. G. Agresar, J. J. Linderman, G. Tryggvason, and K. G. Powell, An adaptive, Cartesian, front-tracking method for the motion, deformation and adhesion of circulating cells, *J. Comput. Phys.* **143**, 346 (1998).

2. H. Burger and R. Schaback, A parallel multistage method for surface/surface intersection, *Comput. Aided Geom. Des.* **10**, 277 (1993).
3. R. Courant, E. Isaacson, and M. Rees, On the solution of nonlinear hyperbolic differential equations by finite differences, *Comm. Pure Appl. Math.* **5**, 243 (1952).
4. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications* (Springer-Verlag, Berlin, 1997).
5. M. P. do Carmo, *Differential Geometry of Curves and Surfaces* (Prentice–Hall, New York, 1976).
6. H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *Commun. Assoc. Comput. Mech.* **29**, 669 (1986).
7. L. C. Evans, *Partial Differential Equations* (Amer. Math. Soc., Providence, RI, 1995).
8. T. A. Grandine and F. W. Klein, IV, A new approach to the surface intersection problem, *Comput. Aided Geom. Des.* **14**, 111 (1997).
9. M. A. Grayson, The heat equation shrinks embedded plane curves to round points, *J. Differential Geom.* **26**, 285 (1987).
10. A. Harten, S. Osher, B. Engquist, and S. R. Chakravarthy, Uniformly high order accurate essentially non-oscillatory scheme, *J. Comput. Phys.* **71**, 231 (1987).
11. J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design* (AK Peters Wellesley, MA, 1993).
12. D. E. Knuth, *The Art of Computer Programming* (Addison–Wesley, Reading, MA, 1998), 2nd ed., Vol. 3, *Sorting and Searching*.
13. A. Preusser. Efficient formulation of a bivariate nonic C^2-Hermite polynomial on triangles, *ACM Trans. Math. Software* **16**, 246 (1990).
14. P. J. Rasch and D. L. Williamson, On shape-preserving interpolation and semi-Lagrangian transport, *SIAM J. Sci. Stat. Comput.* **11**, 656 (1990).
15. N. Sarnak and R. E. Tarjan, Planar point location using persistent search trees, *Commun. Assoc. Comput. Mech.* **29**, 669 (1986).
16. J. A. Sethian, *Level Set Methods and Fast Marching Methods* (Cambridge Univ. Press, Cambridge, UK, 1999).
17. J. A. Sethian and J. Strain, Crystal growth and dendritic solidification, *J. Comput. Phys.* **98**, 231 (1992).
18. P. K. Smolarkiewicz and W. W. Grabowski, The multidimensional positive definite advection transport algorithm: Nonoscillatory option, *J. Comput. Phys.* **86**, 355 (1996).
19. P. K. Smolarkiewicz and J. Pudykiewicz, A class of semi-Lagrangian approximations for fluids, *J. Atmos. Sci.* **49**, 2082 (1992).
20. A. Staniforth and J. Côte, Semi-Lagrangian schemes for atmospheric models—A review, *Mon. Weather Rev.* **119**, 2206 (1991).
21. J. Strain, A boundary integral approach to unstable solidification, *J. Comput. Phys.* **85**, 342 (1989).
22. J. Strain, Fast tree-based redistancing for level set computations, *J. Comput. Phys.* **152**, 648 (1999).
23. J. Strain, Semi-Lagrangian methods for level set equations, *J. Comput. Phys.* **151**, 498 (1999).
24. J. Strain, Tree methods for moving interfaces, *J. Comput. Phys.* **151**, 616 (1999).
25. J. Strain, A fast modular semi-Lagrangian method for moving interfaces, *J. Comput. Phys.* **161**, 512 (2000).
26. C. Truesdell and R. A. Toupin, The classical field theories, in *Handbuch der Physik III/1*, edited by S. Flügge (Springer-Verlag, Berlin, 1960).
27. G. Wulff, Zur Frage der Geschwindigkeit des Wachstums und der Auflösung der Krystallflachen, *Z. Krystall. Min.* **34**, 449 (1901).