Journal of Computational Physics **161,** 512–536 (2000)
doi:10.1006/jcph.2000.6508, available online at http://www.idealibrary.com on IDEAL®

A Fast Modular Semi-Lagrangian 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 June 22, 1999; revised January 14, 2000

A fast modular numerical method for solving general moving interface problems is presented. It simplifies code development by providing a black-box solver which moves a given interface one step with given normal velocity. The method combines an efficiently redistanced level set approach, a problem-independent velocity extension, and a second-order semi-Lagrangian time stepping scheme which reduces numerical error by exact evaluation of the signed distance function. Adaptive quadtree meshes are used to concentrate computational effort on the interface, so the method moves an N-element interface in $O(N \log N)$ work per time step. Efficiency is increased by taking large time steps even for parabolic curvature flows. Numerical results show that the method computes accurate viscosity solutions to a wide variety of difficult geometric moving interface problems involving merging, anisotropy, faceting, nonlocality, and curvature.

1. INTRODUCTION

We present an efficient accurate method for general moving interface problems. Our method functions as a black-box solver, interrogating the interfacial velocity only through a user-supplied module. It also

- merges and breaks complex topology automatically via a level set approach with general velocity extension,
- takes large time steps via stable second-order semi-Lagrangian time stepping schemes, and
- resolves an N-element interface with optimal $O(N \log N)$ work per time step by adaptive quadtree meshing and efficient geometric algorithms.

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

0021-9991/00 $35.00

The work extends the first-order semi-Lagrangian uniform mesh method for geometric moving interfaces of [19], the adaptive quadtree version of [20], and the quadtree-based fast redistancing algorithm of [18]. The present work uses second-order accurate time stepping which greatly reduces numerical dissipation (Section 1.2), fast adaptive quadtree meshing, and a faster redistancing scheme which enhances the semi-Lagrangian approach by exact interpolation (Section 1.3). The fast new quadtree-based velocity extension algorithm of Section 2 allows our method to couple with any user-supplied interfacial velocity, such as the geometric velocity built in Section 3 and used in the computational experiments of Section 4.

1.1. *Overview*

A 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, a curvature C and a 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 shows some typical solutions of geometric moving interface problems, where $V = V(x, t, N, C)$ depends only on the local position and geometry of $\Gamma(t)$.

The main difficulty in moving interfaces is the correct handling of merging, breaking, and other topological changes. This difficulty can be overcome by reformulating the problem on a fixed domain, using the signed distance function to $\Gamma(t)$ defined by

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

If φ is taken positive in $\Omega(t)$ then N, C and V can be computed from φ by [22]

$$N = \nabla\varphi/\|\nabla\varphi\|, \qquad C = -\nabla \cdot N, \qquad V = \varphi_t \nabla\varphi/\|\nabla\varphi\|^2. \tag{2}$$

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

$$\varphi_t - W \cdot \nabla\varphi = 0 \tag{3}$$

moves the zero set $\Gamma(t)$ of the solution φ with velocity V and hence solves the moving interface problem. Topological changes are handled automatically.

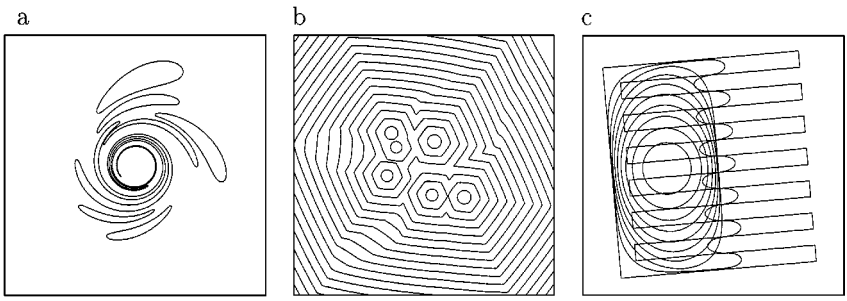


FIG. 1. Sample moving interfaces computed with our method: (a) initially circular bubbles after passive transport in a shearing flow $V = V(x)$, (b) faceted shapes merging and growing under a sixfold anisotropic velocity $V = V(N)$, and (c) a complex polygonal shape shrinking under curvature flow $V = C.N$.

"Level set" methods [11] move $\Gamma(t)$ via the advection equation (3). An initial signed distance function $\varphi(x, 0)$ and an extended velocity field W are built, φ is advected and contoured when $\Gamma(t)$ is required. While these methods handle topology automatically, there are some potential difficulties. They can be expensive since the dimension increases, particularly if uniform meshes are used. One must be careful to obtain the correct "viscosity solution" of the advection equation. Extending the velocity off $\Gamma(t)$ can be difficult. Level set methods are not naturally *modular*: a new code must be written for each new problem to be solved, since the velocity evaluation is intertwined with the moving interface code by velocity extension. Our method removes these barriers: we solve the advection equation on an adaptive quadtree mesh to eliminate the cost of going up a dimension. Correct viscosity solutions are obtained by semi-Lagrangian time stepping with exact evaluation of the signed distance function and frequent efficient redistancing. A general problem-independent velocity extension makes our method modular and easy to apply.

Our numerical method moves interfaces by the second-order semi-Lagrangian time stepping scheme presented in Section 1.2 and three independent computational modules:

- redistancing to produce the signed distance φ from a given interface Γ (Section 1.3),
- contouring to extract the zero set Γ from the solution φ of Eq. (3) (Section 1.3), and
- extension of interfacial velocities V defined on Γ to global smooth velocities W defined everywhere on $\mathbf{R}^d$ (Section 2).

1.2. *Semi-Lagrangian Time Stepping*

The advection equation (3) can be solved by many time stepping schemes. We use semi-Lagrangian schemes which offer some unique advantages: explicit unconditional stability, natural adaptivity, and modularity. These schemes are widely used for modeling linear advection in atmospheric science [13, 15], where unconditional stability eliminates the stringent time-step restriction encountered on small cells by Eulerian schemes [14] and have been applied to moving interfaces in [19, 20]. They rely on the observation that the advection equation propagates solution values along characteristics $x = s(t)$ satisfying

$$\dot{s}(t) = -W(s(t), t). \tag{4}$$

Thus the solution $\psi(x) = \varphi(x, t + k)$ at time $t + k$ can be evaluated by solving the characteristic ODE (4) backward in time from $x = s(t + k)$ to $s(t)$ and setting $\psi(x) = \varphi(s(t), t)$. Standard ODE theory [5] guarantees a unique solution $s(t)$ for any x if W is globally Lipschitz, so backward characteristics do not cross and the procedure is well defined [13]. Semi-Lagrangian methods adopt this two-step approach, solving the characteristic ODE numerically from $x = s(t + k)$ to $s(t)$ and then approximating the off-grid value $\varphi(s(t), t)$ by interpolation [9] or monotone advection [12] to preserve stability. Our moving interface method evaluates the signed distance $\varphi(s(t), t)$ exactly and omits the approximation.

Semi-Lagrangian moving interface methods which use the first-order Courant–Isaacson–Rees (CIR) scheme [1]

$$\tilde{\psi}(x) = \varphi(\tilde{x}, t) = \varphi(x + kW(x, t), t) \tag{5}$$

to solve the advection equation were developed in [19, 20]. Their effectiveness has been heuristically justified and experimentally verified for many moving interface problems involving passive transport, geometry, dynamic topology, faceting, and curvature. The

convergence theory of these methods is straightforward if $V = V(x, t, N)$, because the advection equation is hyperbolic. For curvature flow $V = C$, the advection equation is parabolic and the main barrier to convergence is the Courant–Friedrichs–Lewy (CFL) condition, which restricts the time step of most explicit methods by $k \le O(h^2)$ to ensure information propagates correctly and the computation remains stable. Semi-Lagrangian moving interface methods are explicit yet unconditionally stable and can satisfy the CFL condition by nonlocal velocity evaluation, permitting convergence with large time steps $k = O(h)$ even for parabolic problems [19, 20].

The specific semi-Lagrangian time stepping scheme used in our method combines a CIR predictor (5) with a second-order trapezoidal corrector

$$\psi(x) = \varphi\left(x + \frac{k}{2}W(\tilde{x}, t) + \frac{k}{2}\tilde{W}(x, t+k), t\right): \tag{6}$$

the extended velocity $\tilde{W}$ is evaluated from $\tilde{\psi}$ at time $t+k$. This predictor–corrector pair is second-order accurate in time, explicit, and unconditionally stable. Each new ψ value is an exact φ value so the maximum of φ can never increase. The CIR predictor (5) used as a time-stepping scheme is also unconditionally stable, but only first-order accurate and extremely dissipative [7].

Both predictor and corrector may also incorporate local iteration, where the first approximation $x^* = x + kW(x, t)$ (respectively, $x^* = x + \frac{k}{2}W(\tilde{x}, t) + \frac{k}{2}\tilde{W}(x, t+k)$) to the foot of the characteristic through x is repeatedly replaced by $x + kW(x^*, t)$ (respectively, $x + \frac{k}{2}W(x^*, t) + \frac{k}{2}\tilde{W}(x, t+k)$). Local iteration does not alter the order of accuracy of the time stepping scheme, but reduces numerical dissipation noticeably in some experiments.

Since our advection velocity $W(x, t)$ extends the user-specified velocity functional V defined on the zero set $\Gamma(t)$ of $\varphi(x, t)$, each semi-Lagrangian time step requires several complex global operations. Starting with an interface $\Gamma(t)$, our method carries out the following steps to produce the new interface $\Gamma(t+k)$:

1. Evaluate the signed distance φ from the interface $\Gamma(t)$.
2. Evaluate the interfacial velocity V of $\Gamma(t)$ by a user-supplied module.
3. Extend V to a global advection velocity W.
4. Advance φ via W to the predicted CIR solution $\tilde{\psi}$ defined by Eq. (5).
5. Contour $\tilde{\psi}$ to get the predicted interface $\tilde{\Gamma}$.
6. Evaluate the predicted interfacial velocity $\tilde{V}$ of $\tilde{\Gamma}$.
7. Extend $\tilde{V}$ to a global advection velocity $\tilde{W}$.
8. Advance φ via W and $\tilde{W}$ to the corrected solution ψ defined by Eq. (6).
9. Contour ψ to get $\Gamma(t+k)$.

Each of these steps can be efficiently implemented via the quadtree mesh which we now define.

1.3. *Quadtree Meshes and Fast Algorithms*

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

- A *cell list* of cells C_j, grouped by level l, with the root cell $C_0 = R$ on level $l = 0$.
- A *vertex list* locating cell vertices and centers in $\mathbf{R}^d$.

Each cell C in the cell list contains:

- Its level l and location $(i_1, \ldots, i_d)$ in a uniform d-dimensional mesh with 2^l cells per side.
- The indices in the vertex list of the $2^d + 1$ cell vertices and centers.
- The indices in the cell list of its parent (if there is one), children (if any), and neighbors (if any).

Given an L-level quadtree, many operations related to searching and sorting can be done efficiently [2]. Finding the tree cell which contains a point x, for example, requires $O(L)$ checks of bits in the binary representation of x. An L-level quadtree can be built by recursively splitting cells into 2^d subcells, according to a splitting criterion which characterizes the particular quadtree being built. Applications of quadtrees in Steps 1–9 include velocity extension (Section 2) and:

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

$$d(C, \Gamma) = \min_{x \in C} \min_{\gamma \in \Gamma} \|x - \gamma\| \tag{7}$$

to Γ. Such a tree resolves Γ at optimal $O(N)$ cost, allows fast $O(N \log N)$ evaluation of the signed distance function for Γ, and is used in Step 1 of our method. The following fast redistancing algorithm based on [18] evaluates $\varphi(x)$ at a new vertex x of a level-l cell C in $\lceil 2 + (1/(2 \log 2)) \log d \rceil$ steps in d dimensions:

- **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 of C; replace C by its parent C^*, find and record a nearest element Γ_j^* 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 $\varphi = \pm m^*$ by checking normal vectors of Γ_j^* and its neighbors. Nearest elements forming an acute angle may disagree on the sign of φ, so neighbors must be checked: see Fig. 2a.

This search strategy builds the distance tree in $O(N \log N)$ time and space complexity, because the union of all triples of cells on any level $l \leq L = O(\log N)$ intersects $O(N)$ elements. The search always terminates in a bounded number of steps (see Fig. 2b). The nearest element of Γ which intersects the triple T^* of the parent C^* of a tree cell C may not be the nearest element of Γ overall; nor need it be the nearest element of Γ intersecting the triple T^{**} of the grandparent C^{**}. But in $d \leq 4$ dimensions, any element intersecting T^* beats every element outside the triple T^{***} of the great-grandparent C^{***}. In general dimension d, the number a of levels ascended is determined by the diagonal length of a d-dimensional cube via the requirement $2^a \geq 4\sqrt{d}$ or $a \geq 2 + (1/(2 \log 2)) \log d$.

The contour tree. The unknown zero set of a given function ψ can be efficiently resolved on a quadtree built by splitting each cell whose edge length exceeds the minimum value of $|\psi|$ on the cell. Such quadtrees are built for ψ and $\tilde{\psi}$ in Steps 4 and 8. This simple approach to adaptive meshing takes advantage of the unique functional viewpoint inherent in semi-Lagrangian methods, which define the solution $\psi(x)$ at the new time step by the

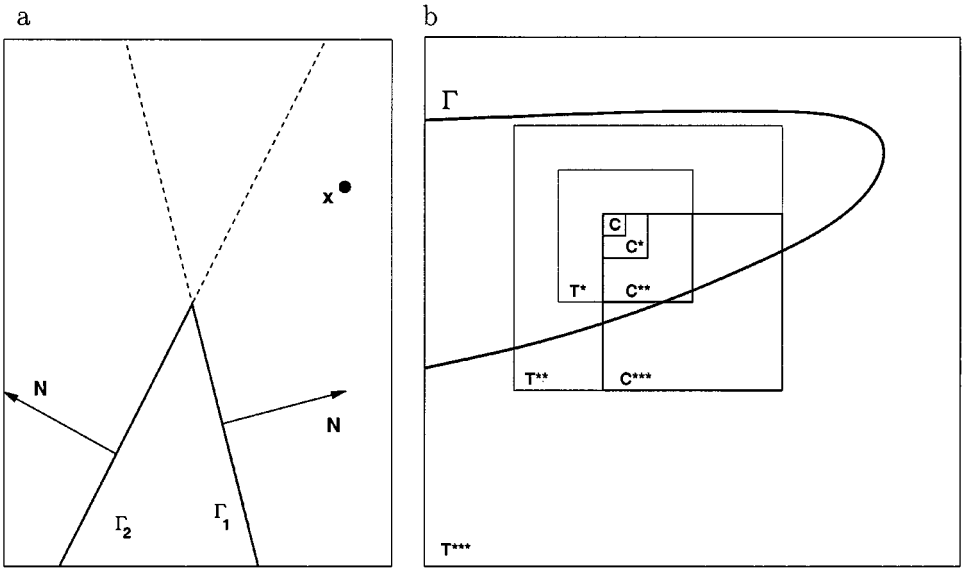


FIG. 2. (a) A pitfall in signing $\varphi(x)$: two nearest elements forming an acute angle may disagree about the sign of $\varphi(x)$. (b) The search strategy always terminates in at most three steps in $d \leq 4$ dimensions: any element intersecting T^* rules out all elements outside T^{***}.

formula (6), which can be evaluated at any desired point. Most adaptive methods use local derivative estimates and must recompute the solution when a trial mesh is refined [8].

Triangulation. For quadtrees in which adjacent cells differ in size by no more than a factor of 2—such as the distance tree—cell vertices and centers can easily be triangulated into conforming meshes [2]. 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 configurations shown in Fig. 4. The distance tree for a simple interface Γ is triangulated in Fig. 3.

Contouring. Given function values $\psi(x)$ at the vertices and centers of a triangulated quadtree mesh, extracting the zero set of the continuous piecewise-linear interpolant on the

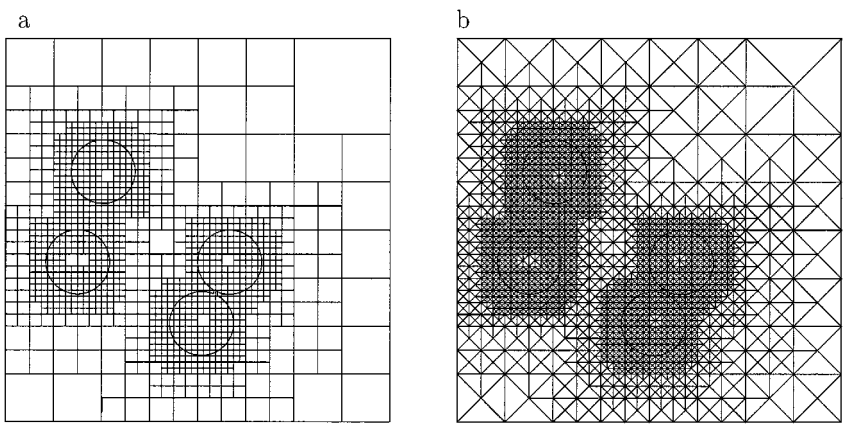


FIG. 3. (a) Six-level distance tree mesh for a simple interface and (b) triangulation of its cell vertices and centers.

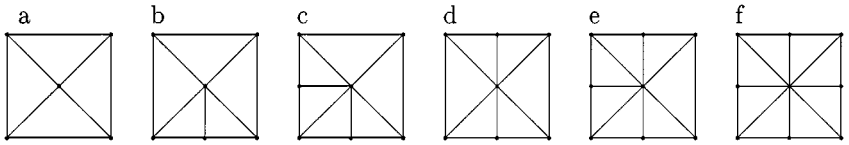


FIG. 4. Possible triangulations of a two-dimensional tree cell.

triangulation is straightforward. In $d = 2$ dimensions, for example, each triangle where ψ changes sign contains a unique line segment where the linear interpolant to ψ vanishes. These line segments form a polygonal curve since the triangulation is conforming and the interpolant is continuous. Following the polygonal zero curve as far as possible in both directions produces an oriented component of $\Gamma(t + k)$ with $\psi > 0$ on its interior. In $d > 2$ dimensions the contouring process is similar but slightly more complicated: d-dimensional simplices replace triangles and $(d - 1)$-dimensional simplices replace line segments. Such a contouring algorithm approximately inverts the redistancing scheme built on a distance tree and is used in Steps 5 and 9 above.

2. VELOCITY EXTENSION

Moving interfaces via the advection equation

$$\varphi_t - W \cdot \nabla\varphi = 0 \tag{8}$$

requires a globally defined velocity W which extends the given velocity functional V smoothly off the interface $\Gamma(t)$. Early level set methods such as [10] built in problem-dependent velocity extensions, such as the natural extensions available for passive transport, geometric problems and interfacial transport in fluid flows. This reduces the usefulness of the level set approach because each new moving interface problem requires a new velocity extension programmed into the level set algorithm. Some recent velocity extensions tailored to specific advection methods and interfacial representations are presented in [11].

We use a fast problem-independent velocity extension to move $\Gamma(t)$ without any information on the $V - \Gamma$ relationship. This permits the solution of a wide variety of moving interface problems with minimal programming effort: given an implementation of our method, a new moving interface problem requires only a new code for evaluating V on $\Gamma(t)$, rather than a completely new moving interface code. Our velocity extension combines the nearest-point extension of Section 2.1 and the distance tree [18] summarized in Section 1.3 into the continuous and efficient "numerical Whitney extension" of Section 2.3.

2.1. *Nearest-Point Extension*

Given any continuous function g on Γ, we can define its nearest-point extension G by

$$G(x) = g(\gamma), \tag{9}$$

where γ is a nearest point on Γ 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. Figure 5 shows an example: G is discontinuous along the "medial axis" [2] consisting of points equidistant from two or more points of Γ. A weighted discrete version of this extension is used in [11].

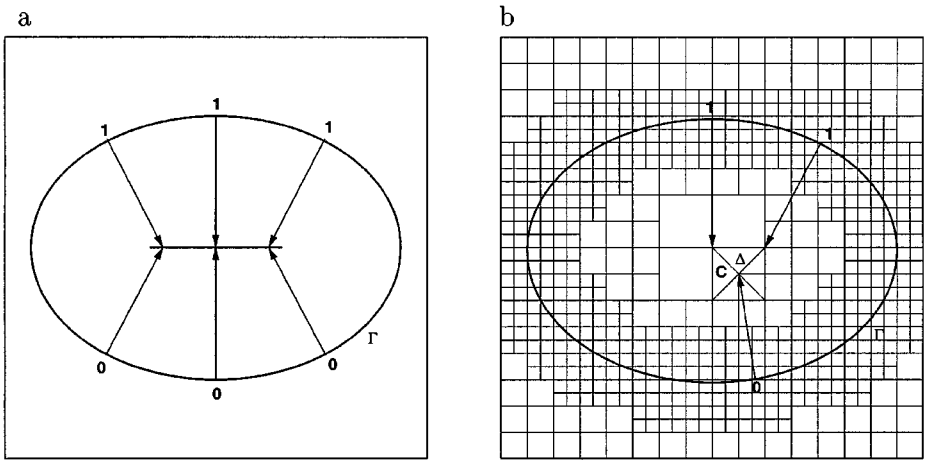


FIG. 5. (a) Discontinuities of the nearest-point extension occur along the medial axis of Γ, where nearest-point values jump from 0 to 1. (b) The numerical Whitney extension maintains continuity by interpolating linearly between the nearest-point values 0 and 1 in the triangle $\Delta \subset C$ in a 5-level distance tree.

The nearest-point extension can be efficiently evaluated at the vertices and centers of a distance tree resolving 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 by finding a nearest point γ on a known nearest element and setting $G(x) = g(\gamma)$. At arbitrary points $x \in \mathbf{R}^d$, however, 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. The distance tree speeds up $G(x)$ evaluation only for x near Γ, because such points are contained in small cells with few nearby elements where the search strategy of Section 1.3 is efficient.

2.2. *Whitney Extension*

The velocity extension technique used in our moving interface method resembles the classical Whitney extension procedure of [23]:

- Build an *infinite* distance tree covering all of R^d with $L = \infty$.
- Evaluate the nearest-point extension $G(x)$ at a random point x in each childless cell.
- Piece these values together into a continuous function with a partition of unity subordinate to the distance tree.

The Whitney procedure produces a continuous extension of g off Γ, can be modified to produce a C^k extension, and is widely used in harmonic analysis [16]. The Whitney extension, unlike the nearest-point extension, is continuous at every $x \in \mathbf{R}^d$, because the cell containing x is never subdivided once its distance from Γ exceeds its size. Thus the Whitney extension is eventually equal to a fixed continuous function on each cell as $L \to \infty$.

2.3. *Numerical Whitney Extension*

Our velocity extension technique differs from the Whitney procedure, by using a *finite* distance tree with $L < \infty$, efficiently evaluating the nearest-point extension $G(x)$ at distance

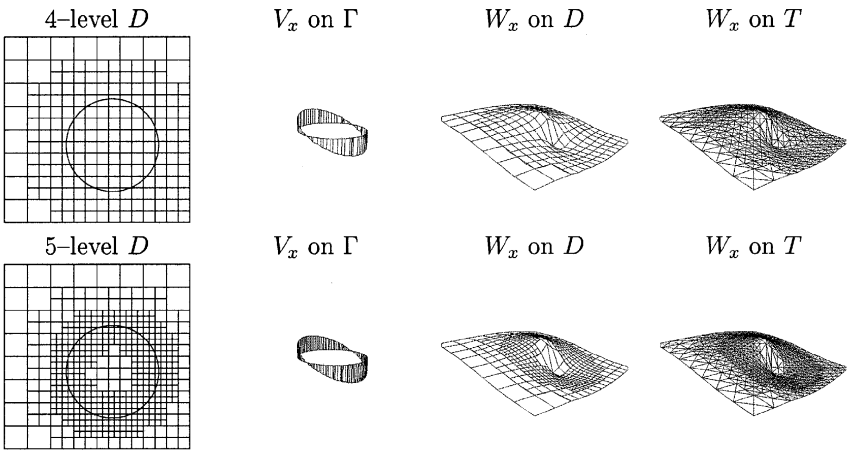


FIG. 6. Distance tree D, interfacial unit normal velocity x-component V_x and Whitney extension W_x over distance tree and triangulation T with $L = 4$ and 5 levels.

tree vertices and centers, and replacing Whitney's partition of unity by continuous piecewise-linear interpolation from the triangulated tree vertices and centers. The procedure evaluates $W(x)$ via the following steps, shown in Figure 5(b):

- Find the distance tree cell C containing x by binary search.
- Find the triangle $\Delta \subset C$ containing x.
- Interpolate the nearest-point extension of g linearly from vertices of Δ to get the numerical Whitney extension $W(x)$.

Unit normal velocity $V = N$ on a circular interface Γ, and its extension W on $\mathbf{R}^2$ are shown in Fig. 6. Timings for building the distance tree and evaluating the nearest-point extension at all cell vertices and centers are given in Table I and exhibit the expected $O(N \log N)$ cost.

The numerical Whitney extension produces a continuous function but not an exact extension of g off Γ, unless level-L childless cells meet Γ at vertices and centers and g is piecewise linear on Γ. This inexactness can be remedied by subdividing Γ elements and tree cells at every intersection point and adding such points to the triangulation.

When g is the velocity V of Γ, the numerical stability of moving interface methods is improved by the *maximum principle* which holds for the numerical Whitney extension: the

TABLE I

Number L of Tree Levels and Number N_c of Tree Cells, versus CPU Seconds T_d for Building the Distance Tree D, T_t for Triangulating the Tree Vertices and Centers, T_e for Evaluating the Nearest-Point Extension on D, and T_s for One Step of Semi-Lagrangian Time Stepping

L	4	5	6	7	8	9	10	11	12
N_c	269	653	1421	2957	6029	12173	24461	49037	98189
T_d	0.01	0.02	0.04	0.11	0.24	0.57	1.24	2.81	6.35
T_t	0.01	0.01	0.02	0.05	0.11	0.24	0.51	1.11	2.32
T_e	0	0	0	0.01	0.01	0.02	0.03	0.07	0.14
T_s	0.02	0.04	0.09	0.2	0.43	0.95	2.04	4.3	9.5

maximum over $\mathbf{R}^d$ of W cannot exceed the maximum over Γ of V. The maximum principle guarantees that regions of space far from Γ cannot move faster than the interesting nearby regions.

3. LOCAL GEOMETRIC VELOCITIES

Computations with our method require a user-supplied module which evaluates the interfacial velocity for a given moving interface problem. In this section, we describe such a module for evaluating velocity functionals $V = V(x, t, N, C)$ depending on the local position and geometry of the interface. These velocities provide important computational tests for the accuracy, efficiency, and modularity of our method and pose numerical difficulties of their own. The computation of a smooth and accurate normal and curvature for a complex polygonal interface Γ with facets and corners is difficult because the standard formulas for curvature are complicated and their numerical approximation is sensitive. Thus we use an indirect technique based on the signed distance function φ and the geometric formulas

$$N = \frac{\nabla\varphi}{\|\nabla\varphi\|}, \quad C = -\nabla \cdot N. \tag{10}$$

A uniform mesh gives convenient robust approximations of these formulas, but requires excessive CPU time and memory. Thus we evaluate N and C on Γ efficiently and accurately by a module which

- builds a local equidistant mesh efficiently near Γ,
- evaluates φ on the local mesh,
- differentiates φ and N accurately on the local mesh, and
- interpolates N and C back to the vertices of Γ.

3.1. *Local Equidistant Meshing*

First, we build a local equidistant mesh near Γ. The simplest technique, marking nearby points of a global mesh, is prohibitively expensive for fine meshes. A more efficient technique employs sorting as follows.

A two-dimensional local mesh with mesh size h 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. Figure 7 shows a local mesh and these two viewpoints.

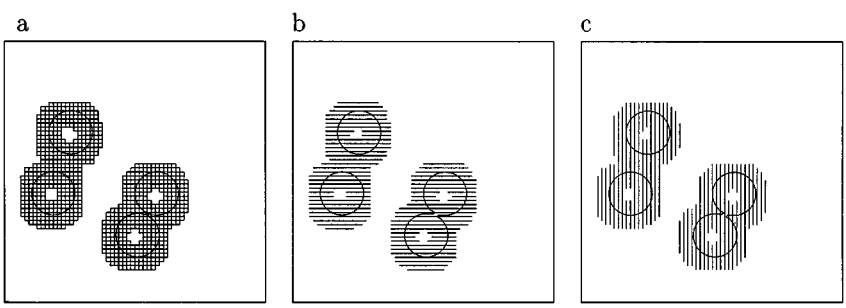


FIG. 7. Local grid for a simple interface (a), viewed as x-intervals in (b) and y-intervals in (c).

We build the local mesh by listing every mesh point within *horizontal* distance Rh of any interface point $\gamma \in \Gamma$, then listing each mesh point within *vertical* distance Rh of some point listed in the horizontal pass. The resulting mesh includes every point necessary to form a two-dimensional difference stencil of half-width R for differentiating or interpolating to any interface point $\gamma \in \Gamma$.

An efficient construction algorithm is ensured by sorting and pruning local mesh points listed more than once. Red–black trees or hashing could also be used to prevent duplication [6].

We store the local equidistant mesh in a data structure which contains

- The mesh points (ih, jh),
- a list of x-intervals $(i_L : i_R, j)$, sorted into groups with the same y-index j,
- a list of y-intervals $(i, j_L : j_R)$, sorted into groups with the same x-index i,
- pointers to x-intervals with given y-index, and
- pointers to y-intervals with given x-index.

The three-dimensional case is similar with a z-index added.

3.2. *The Local Distance Function*

We evaluate the local distance function φ on the local mesh by injection and interpolation. The local mesh near Γ has mesh size half the size of the level-L cells of the distance tree D, so most local mesh points lie at vertices or centers of the distance tree. At other local mesh points x, we interpolate φ from the triangle containing x.

It is possible to evaluate φ exactly and efficiently as the local mesh is built, by marking each local mesh point with the distance and location of the interface point $\gamma \in \Gamma$ responsible for its creation. However, we expect minimal improvement in accuracy from such an algorithm because injection is already exact near Γ.

3.3. *Differentiation*

Given φ on a local equidistant mesh, we use standard formulas such as

$$\varphi'(x) \approx \frac{1}{2h}(\varphi(x+h) - \varphi(x-h)) \tag{11}$$

to approximate N and C from Eq. (10). Such formulas are based on polynomial interpolation and produce inaccurate oscillatory results if their stencils include points where φ is not smooth. The signed distance function is not smooth when Γ has corners, since $\nabla\varphi$ and C are undefined there.

Thus we compute N and C by essentially nonoscillatory (ENO) differentiation [4]: use standard equidistant formulas, but slide the stencil in the direction that reduces oscillations. For example, we can approximate $\varphi'(x)$ to second order by Eq. (11), or by uncentered formulas

$$\frac{3\varphi(x) - 4\varphi(x-h) + \varphi(x-2h)}{2h} \quad \text{or} \quad \frac{-\varphi(x+2h) + 4\varphi(x+h) - 3\varphi(x)}{2h}. \tag{12}$$

Normally the centered formula in Eq. (11) would give better accuracy, but if x is adjacent to a discontinuity located near $x + h/2$ we would prefer the left-shifted stencil in Eq. (12).

As an automatic algorithm, we choose the stencil to minimize the sum of absolute values of the *second* difference $\varphi(x+h) - 2\varphi(x) + \varphi(x-h)$ over the stencil points, because we expect φ to be Lipschitz continuous with jumps in φ'. Large values of φ'' or the second difference will therefore signal corners in φ, which the stencil should not cross.

One-dimensional ENO differentiation on the local mesh suffices to evaluate the multidimensional derivatives in N and C. We differentiate in x by using the x-interval representation of the local mesh, and vice versa in y.

Accuracy of numerical derivatives is further improved by cosine smoothing: in the x direction, for example, replace $\varphi(x)$ by

$$S_x\varphi(x) = \frac{1}{4}(\varphi(x+h) + 2\varphi(x) + \varphi(x-h)), \tag{13}$$

and similarly in the y direction. Such operations commute, so we apply S_x on each x interval (with simple averaging at the endpoints), then apply S_y on each y interval. This constitutes one pass of cosine smoothing, and improves the accuracy of ENO differentiation noticeably. Previous work on semi-Lagrangian moving interface methods [19, 20] shows that smoothing is essential when the normal velocity V depends strongly on curvature. While the convergence theory of these methods is not yet complete, it appears that smoothing satisfies a CFL accuracy condition and produces convergence.

3.4. *Interpolation*

We complete the evaluation of V on a polygonal interface Γ by interpolating N and C back to the vertices of Γ. Since N and C are known on a local equidistant mesh near Γ, many accurate interpolation schemes are available. We use ENO interpolation on a stencil chosen to minimize the sum of absolute values of *first* differences over the stnecil, because these quantities may have jumps.

4. COMPUTATIONAL VALIDATION

In this section, we demonstrate the accuracy and efficiency of our method by computing solutions to a wide variety of moving interface problems. We describe the implementation of our method in Section 4.1 and the common parameters of our computational experiments in Section 4.2. Our method is tested on complicated smooth and nonsmooth interfaces under rigid and shearing passive rotation velocities in Section 4.3 and on smooth and faceted interfaces moving under geometric velocities including anisotropy, topological complexity, curvature, and nonconvexity in Section 4.4. Motion under a simple nonlocal geometric velocity functional is computed in Section 4.5.

4.1. *Algorithm*

Our method was implemented in Standard C, following the outline below, 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 2.7. The code was not extensively tuned for speed so timings reported are far from optimal. The numbered steps correspond to Steps 1–9 in Section 1.2. We begin with $\Gamma = \Gamma(t)$.

1. $[D, \varphi] = \texttt{DistanceTree}(L, \Gamma)$ [Build an L-level distance tree around Γ and evaluate the signed distance φ from Γ at tree cell vertices.]
2. $V = \texttt{Velocity}(t, \Gamma; D, \varphi)$ [Call a user-supplied module to evaluate the velocity on the interface.]
   $T = \texttt{Triangulation}(D)$
3. $W = \texttt{WhitneyExtension}(V, D, T)$ [Build the Whitney extension of V.]
4. $[\tilde{D}, \tilde{\psi}] = \texttt{ContourTree}\ (k, t, \varphi(x + kW(x, t), t))$ [Build a contour tree $\tilde{D}$ resolving the zero set of the predicted CIR solution $\tilde{\psi}$ defined in Eq. (5).]
   $\tilde{T} = \texttt{Triangulation}\ (\tilde{D})$
5. $\tilde{\Gamma} = \texttt{ZeroSet}\ (\tilde{\psi}, \tilde{T})$ [Find the zero set of $\tilde{\psi}$ on the triangulation $\tilde{T}$.]
6. $\tilde{V} = \texttt{Velocity}(t + k, \tilde{\Gamma}; \tilde{D}, \tilde{\psi})$
7. $\tilde{W} = \texttt{WhitneyExtension}\ (\tilde{V}, \tilde{D}, \tilde{T})$
8. $[D, \psi] = \texttt{ContourTree}(k, t, \varphi(x + \frac{k}{2} W(\tilde{x}, t) + \frac{k}{2} \tilde{W}(x, t), t))$
   $T = \texttt{Triangulation}(D)$
9. $\Gamma(t + k) = \texttt{ZeroSet}\ (\psi, T)$

Each cell splitting in Step 4 requires new values of $\tilde{\psi}(x) = \psi(\tilde{x}, t)$ constructed by the following sequence of operations:

4.1. Find the distance tree cell C and subtriangle Δ of C containing x.
4.2. Interpolate the nearest-point extension of V from vertices of Δ to get the numerical Whitney extension $W(x, t)$.
4.3. Project x backward to the predicted characteristic point $\tilde{x} = x + kW(x, t)$.
4.4. Find the childless distance tree cell $\tilde{C}$ containing $\tilde{x}$.
4.5. Optionally iterate locally to find the velocity and the projected point $\tilde{x}$.
4.6. Search interface elements intersecting the concentric triples of $\tilde{C}$, its parent and grandparent as necessary to find closest point $\gamma \in \Gamma(t)$ to s.
4.7. Set $\tilde{\psi}(x) = \varphi(\tilde{x}, t)$, the exact signed distance from $\tilde{x}$ to $\Gamma(t)$.

Step 8 is similar with $\tilde{x}$ replaced by $x + \frac{k}{2} W(\tilde{x}, t) + \frac{k}{2} \tilde{W}(x, t)$.

4.2. *Parameters and Tests*

Our experiments vary the initial interface $\Gamma(0)$, the velocity functional V, the spacetime domain $[0, a] \times [-b, b]^2$ and the following computational parameters:

L: The number of tree levels in the distance tree and the ψ quadtree. L ranges between 4 and 10, giving spatial resolution equivalent to a uniform mesh with 16^2 to 1024^2 points at much lower cost.
N: The number of time steps from the initial time 0 to the final time a. The time step k is given by $k = a/N$. N ranges from 10 up to 2560, while $k = O(h) = O(2^{-L} b)$ balances spatial and temporal resolution.
S: The number of cosine smoothing passes between φ and $\nabla\varphi$, and between N and C. We took $S = 0$ except for strongly curvature-dependent velocities, where $S = O(L)$.
E: The order of ENO differentiation in local geometric velocities. There is rarely any observable difference between $E = 2$ and $E = 3$, so E is not reported.

Convergence. We refer to a computation with given values for L, N, and S as a $L/N/S$ run for brevity, or as an L/N run if $S = 0$. We carry out several convergence studies comparing each L/N run to the next $(L + 1)/2N$ run, typically superimposing time-exposure

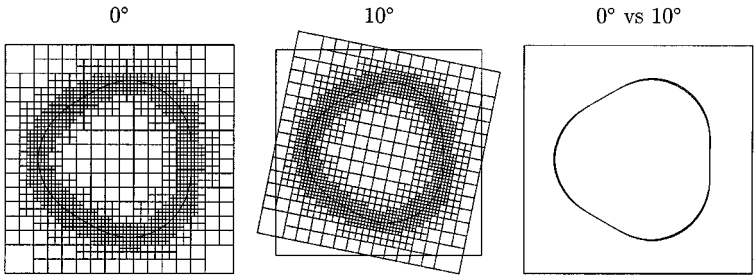


FIG. 8. Testing for grid effects: mesh and solution at 0° and 10°.

plots of the moving interface to demonstrate convergence to graphical accuracy. Interfacial errors

$$\delta = \max_{x \in \tilde{\Gamma}(t)} |\varphi(x, t)|, \tag{14}$$

where φ is the exact signed distance function and $\tilde{\Gamma}(t)$ is the computed interface, are reported for arbitrary interfaces under passive transport and for circles shrinking under curvature.

Grid effects. Many moving interface methods suffer from grid effects which cause anisotropic computational results to depend sensitively on the orientation of the underlying computational grid. Our method is designed to minimize grid effects and maximize isotropy subject to the existence of an underlying square mesh. We often verify the absence of grid effects by computing solutions twice: once with the mesh aligned with an axis of symmetry of the problem and again at 10° to the symmetry axis. The second run is then plotted at -10° and the two runs superimposed to demonstrate the absence of grid effects. Figure 8 shows this process for a threefold anisotropic computation. The small angle 10° typically reveals grid effects well, by distorting propagation directions and speeds for sharp corners propagating under anisotropic velocities.

4.3. *Passive Transport*

We begin validating our method by computing circles and triangles undergoing passive transport by shear velocities

$$F(x, y) = \frac{1 - \max(0, 1 - x^2 - y^2)^4}{8(x^2 + y^2)}(-y, x) \tag{15}$$

and rigid body rotation $F(x, y) = (-y, x)$. The shearing velocity (15) rotates particles around the origin at widely varying speeds, and is often used to test vortex methods for the 2-D Euler equations [17], while rigid body rotation is often used to measure dissipative errors in advection methods [7].

We carried out four computations with smooth and nonsmooth interfaces under these flows:

- (a) eight circles of radii between 0.4 and 1.2 distributed randomly in the domain $[-5, 5]^2$ under shear rotation,
- (b) a single triangle of radius $1/2$ at the origin (0, 0) in the domain $[-2, 2]^2$ under shear rotation,
- (c) a circle of radius $1/2$ located at (2, 2) in the domain $[-4, 4]^2$ under rigid rotation, and

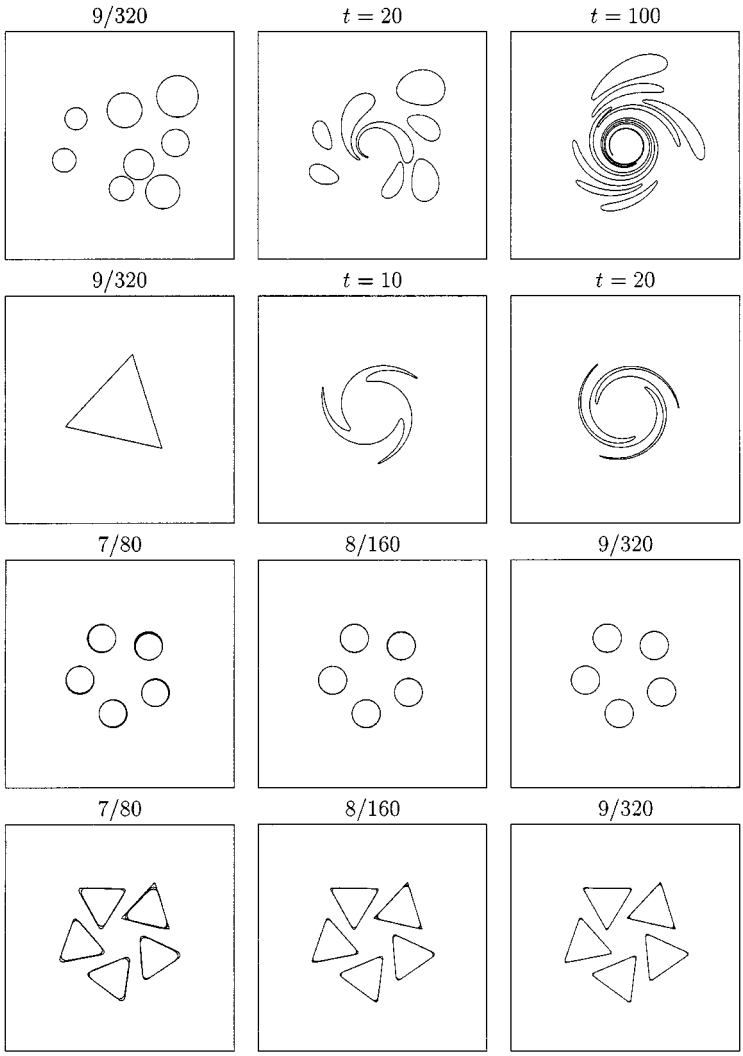


FIG. 9. Shearing and rotating circles and triangles under passive transport.

- (d) a triangle of radius $1/2$ located at $(2, 2)$ in the domain $[-4, 4]^2$ under rigid rotation.

The shearing interfaces (a) and (b) in 9/320 runs resolved to graphical accuracy, plus graphical convergence studies of (c) and (d) after two periods ($t = 4\pi$), are shown in Fig. 9.

Next, we measure the order of accuracy. Passive transport velocities are naturally defined everywhere, but we evaluate them only at the vertices of $\Gamma(t)$: second-order time stepping plus the $O(h^2)$ error in linear interpolation between vertices at each of $O(1/k)$ steps yields a global error of the form $O(k^2) + O(h^2/k)$. The maximum errors in the computed interface at $t = 20$ for shearing and $t = 4\pi$ for rotation are tabulated and plotted in Table II. The observed order of accuracy varies smoothly between 1 for large time steps or the smooth well-resolved interface (c) and $1/2$ for small time steps or the underresolved interfaces (a), (b), and (d). The expected $O(N \log N)$ cost per step is verified by the scaled CPU seconds T/NL reported in Table II.

TABLE II

Maximum Interfacial Errors δ and CPU Seconds T versus Levels L and Time Steps N for Passive Shearing and Rigid Rotation

(a) Shearing circles at $t = 20$ on $[-5, 5]^2$

$L\backslash N$	10	20	40	80	160	320	640	1280	2560	T/NL
6	0.220	.177	.246	.338	.362	—	—	—	—	0.135
7	—	.081	.108	.147	.225	.285	—	—	—	0.309
8	—	—	.032	.052	.082	.119	.165	—	—	0.645
9	—	—	—	.017	.029	.050	.072	.094	—	1.311
10	—	—	—	—	.009	.017	.030	.044	.058	2.625

(b) Shearing triangle at $t = 20$ on $[-2, 2]^2$

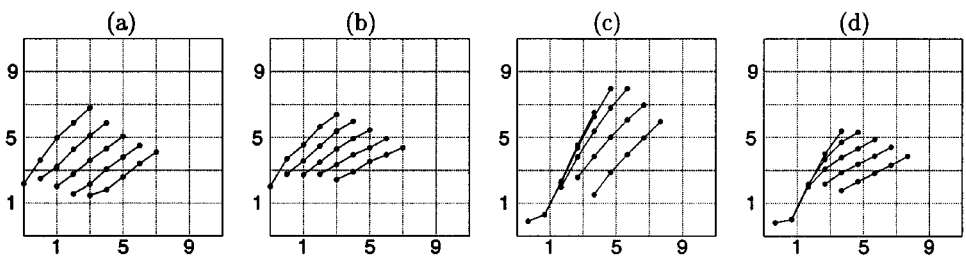
$L\backslash N$	10	20	40	80	160	320	640	1280	2560	T/NL
6	0.247	.147	.151	.147	.185	—	—	—	—	0.050
7	—	.077	.084	.088	.098	.133	—	—	—	0.117
8	—	—	.043	.045	.051	.065	.086	—	—	0.283
9	—	—	—	.020	.024	.033	.048	.065	—	0.651
10	—	—	—	—	.012	.016	.023	.033	.048	1.361

(c) Rotating unit circle at $t = 4\pi$ on $[-4, 4]^2$

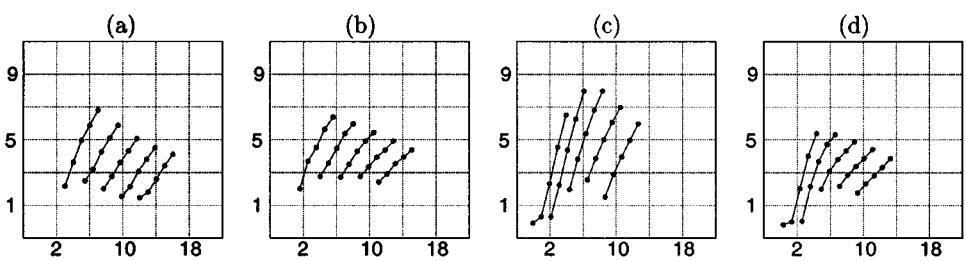
$L\backslash N$	10	20	40	80	160	320	640	1280	2560	T/NL
6	1.060	.806	.254	.168	.347	—	—	—	—	0.016
7	—	.803	.210	.071	.069	.135	—	—	—	0.031
8	—	—	.198	.048	.024	.031	.064	—	—	0.064
9	—	—	—	.043	.013	.009	.015	.032	—	0.128
10	—	—	—	—	.011	.004	.004	.008	.016	0.254

(d) Rotating unit triangle at $t = 4\pi$ on $[-4, 4]^2$

$L\backslash N$	10	20	40	80	160	320	640	1280	2560	T/NL
6	1.130	.970	.249	.222	.291	—	—	—	—	0.021
7	—	.996	.223	.118	.137	.200	—	—	—	0.043
8	—	—	.242	.078	.072	.096	.141	—	—	0.093
9	—	—	—	.062	.038	.050	.068	.100	—	0.190
10	—	—	—	—-	.024	.025	.034	.047	.069	0.388



$-\log_2 \delta$ vs $-\log_2 k$



$-\log_2 \delta$ vs $\log_2 T$

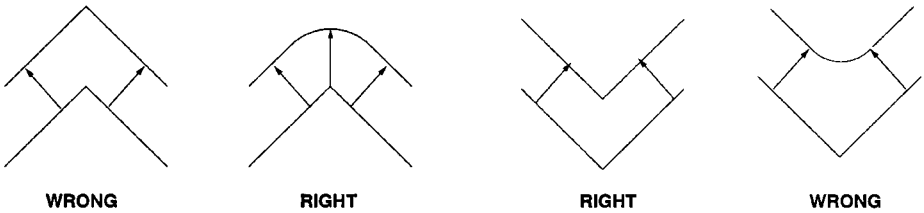


FIG. 10. Right and wrong corner motion with unit normal velocity.

4.4. *Local Geometric Velocities*

We validate our method by computing accurate converged interfaces moving under a variety of local geometric velocities, including

- correct viscosity solutions for corners and complex shapes growing and merging with unit normal velocity,
- regularly faceted shapes growing and shrinking under anisotropic normal velocities,
- irregularly faceted "Wulff shapes" [21] with random angles,
- complex multiply connected faceted growth patterns, and
- simple and complex shapes shrinking under mean curvature and nonlocal volume-preserving mean curvature flows.

4.4.1. Viscosity solutions with corners. Correct computation of "viscosity solutions" for faceted interfaces in geometric problems depends on moving a corner in or out with unit normal velocity. Inward motion should keep corners sharp, while outward motion should produce rounded corners due to Huygens' principle. Even starting from a *smooth* closed curve Γ, inward motion along the normal with unit speed will develop a corner in time at most $1/\max_\Gamma C$. Straightforward moving interface methods can easily produce the incorrect viscosity solutions shown in Fig. 10.

Our method computes the correct viscosity solution for a triangle growing and shrinking with unit normal velocity. We superimpose $5/20$ over $6/40$ runs in Fig. 11, to show

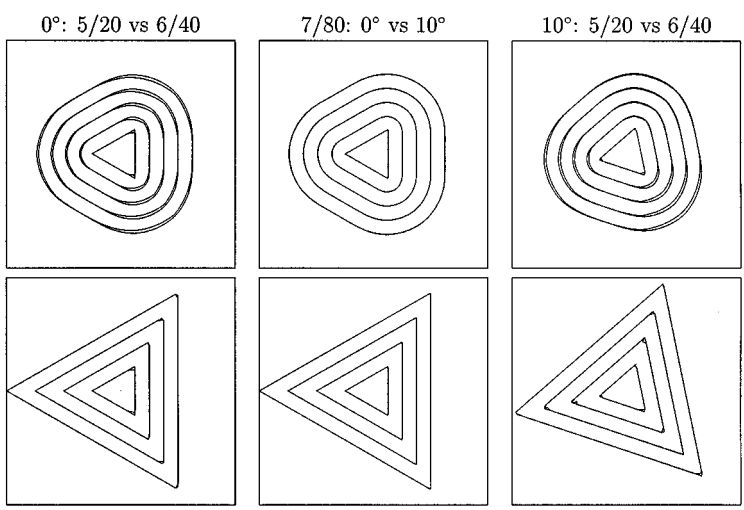


FIG. 11. Triangles growing and shrinking with unit normal velocity: convergence of viscosity solutions and absence of grid effects.

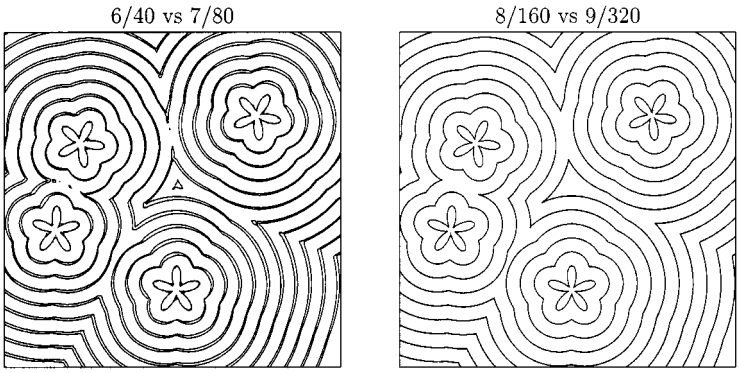


FIG. 12. Clovers merging with unit normal velocity for $0 \le t \le 2.5$ on $[-3, 3]^2$.

convergence to graphical accuracy, and 0° over 10° runs at resolution 7/80, to show absence of grid effects.

Complex interfaces grow and merge correctly in Fig. 12. The manifold corners and changes of topology are computed automatically and easily. In particular, outward-moving inward-pointing corners remain correctly sharp, as the viscosity solution theory requires. The final area enclosed by the computed interface is 35.67937, 35.72377, 35.74304, and 35.75297, respectively, and shows smooth monotone first-order convergence.

4.4.2. Anisotropic normal velocity and the Wulff limit. Anisotropic motion along the normal vector connects moving interfaces to Hamilton–Jacobi equations

$$\varphi_t + H(\nabla\varphi) = 0, \tag{16}$$

which encounter difficulties when the Hamiltonian H is nonconvex. For anisotropic normal velocities

$$V = R + \epsilon\cos(k\theta), \tag{17}$$

the Hamiltonian H is nonconvex—and some Hamilton–Jacobi methods break down—if

$$R + \epsilon(1 - k^2) < 0 < R - |\epsilon|. \tag{18}$$

We evolve an initially unit-circular interface under anisotropic normal velocities (17) producing nonconvex Hamiltonians, with $R = 1$ and $R + \epsilon(1 - k^2) = -4$. Figure 13 compares 7/128 and 8/256 runs for $0 \le t \le 8$ on $[-9, 9]^2$ and tree meshes tilted at 0°, 10°, and $180^\circ/k$ for $k = 3$, 4, 5, and 6. Grid effects are almost invisible and the interface grows rapidly into the regularly faceted Wulff shape with the correct anisotropy. Figure 14 shows similar results for shrinking.

We also test our method on random polygonal Wulff shapes with nonuniform corner angles. These shapes arise from velocities of the form

$$V = R(1 + \epsilon\sin\sigma(\theta)), \tag{19}$$

where σ is the piecewise-smooth function

$$\sigma(\theta) = \pi\frac{\theta - \theta_j}{\theta_{j+1} - \theta_j} \quad \text{for} \quad \theta_j \le \theta \le \theta_{j+1} \tag{20}$$

and $-\pi = \theta_0 < \theta_1 < \cdots < \theta_n = \pi$ are given angles. Motion by Eq. (19) with $R\epsilon > 0$

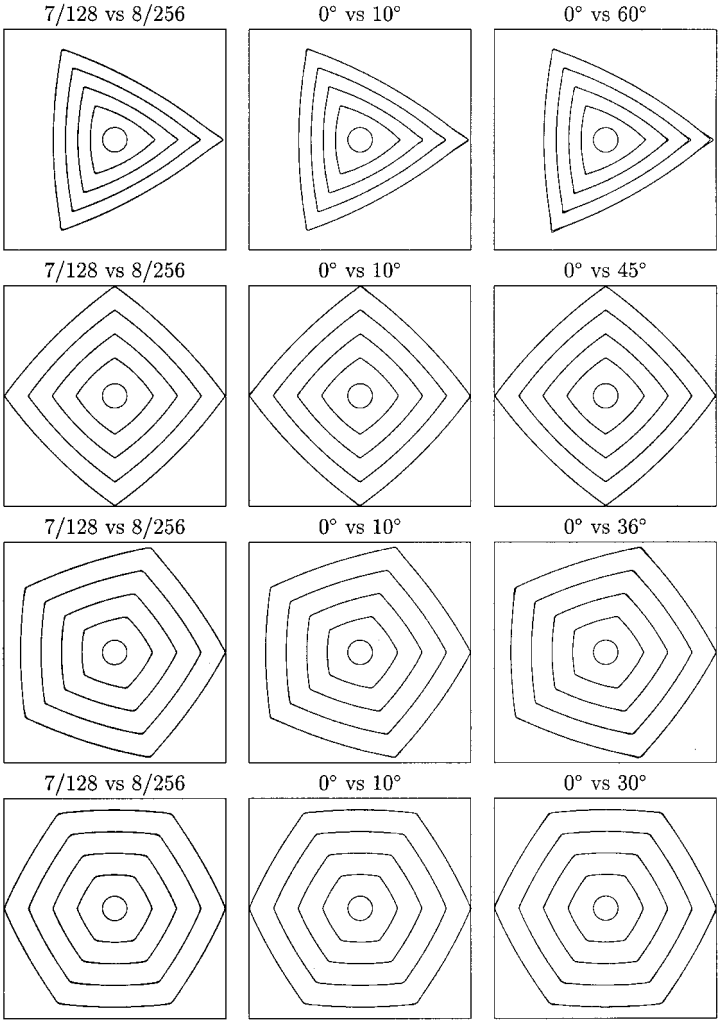


FIG. 13. Circles growing into asymptotic Wulff shapes.

produces facets perpendicular to the angles θ_j, with acute corners rounded. Figure 15 shows polar plots of these velocities with $R = 3/4$ and $\epsilon = 1/2$, the corresponding Wulff shapes, and interfaces moving under Eq. (19) for $0 \le t \le 10$ on $[-12, 12]^2$, and demonstrates excellent agreement between computation and theory [21].

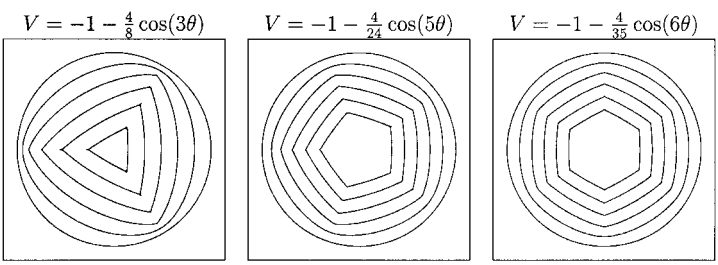


FIG. 14. Radius 3.5 circles shrinking into the asymptotic Wulff shapes: 6/40 vs 7/80 runs for $0 \le t \le 2$ on $[-4, 4]^2$.

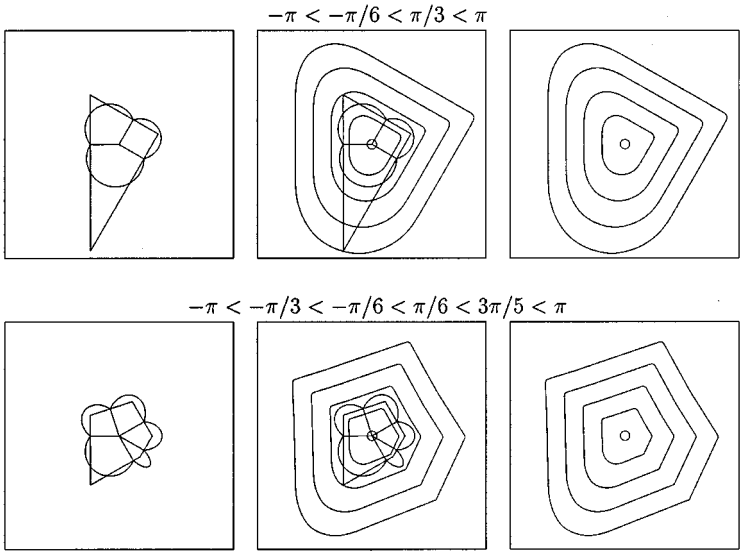


FIG. 15. Circles growing into irregular Wulff shapes: polar plots of $V(\theta)$, 7/128 runs for growing interfaces, and comparisons.

4.4.3. Merging under anisotropy. We validate the topological robustness of the numerical method by starting from a collection of randomly placed and sized circles and moving the interface along its normal with an anisotropic speed $V = 2 + \cos(5\theta + 0.1)$. Figure 16 shows the mechanism which transforms this highly nonconvex initial interface into the asymptotic pentagonal Wulff shape as $t \to \infty$. The 6-level tree meshes at times 0, 40, and 80 are shown, to emphasize the resolution obtained by building the quadtree root cell to enclose the current interface. When an initially small interface grows larger by an order of magnitude, the resolution improves by an order of magnitude as well. Thus, the 8/320 run

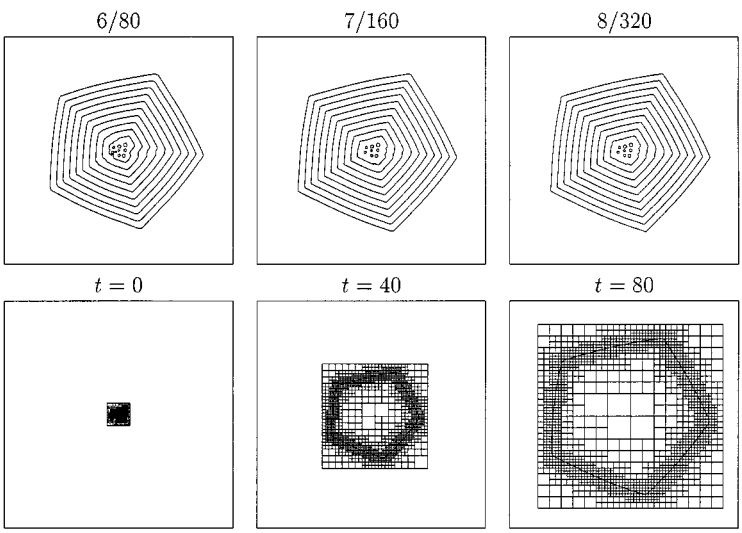


FIG. 16. Bubbles merging into the asymptotic Wulff shape.

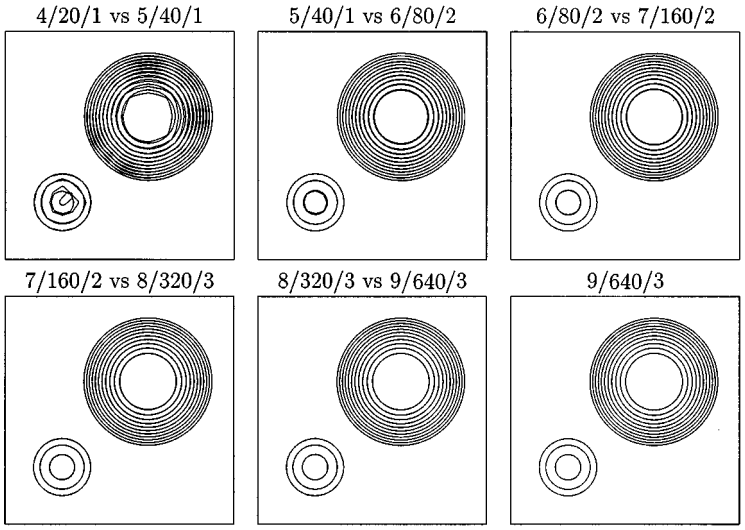


FIG. 17. Circles shrinking with $V = C$ for $0 \le t \le 2$ in $[-4, 4]^2$.

shown—which required 21 min CPU time—would have required 1.5 days of CPU time with a fixed domain sized to fit the final interface.

4.4.4. Circles shrinking under curvature. A classic geometric problem shrinks a plane curve with velocity equal to its curvature, and forms a useful test case for curvature-dependent velocity. A circle shrinking with $V = C$ has exact radius

$$R(t) = \sqrt{R(0)^2 - 2t},$$

so 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 17 shows convergence to graphical accuracy, computed with 20 through 640 time steps on quadtrees with 4 through 9 levels, and plotted every 0.2 time units from $t = 0$ to $t = 2$ on the domain $[-4, 4]^2$. CPU seconds per step and maximum errors in the large circle location at $R = 1$, $t = 2$ are reported and plotted in Table III and display clear first-order convergence along diagonals $k = O(h)$. Along vertical columns the error is dominated by the $O(h^2/k)$ term due to polygonal interface approximation at $O(1/k)$ steps.

4.4.5. Nonconvex interfaces under curvature. A geometric theorem [3] predicts that any smooth embedded plane curve should collapse to a round point and vanish in finite time under curvature flow $V = C$. We verify that our method behaves correctly for two complex polygonal shapes, with the convergence studies shown in Figs. 18 and 19.

4.5. *A Nonlocal Geometric Velocity*

Many important moving interface problems are *nonlocal*—the normal velocity at each point γ depends on all of $\Gamma(t)$ and even on its history $\{\Gamma(s) \mid 0 \le s \le t\}$. In this section, we test our method on the simplest nonlocal geometric velocity

$$F = \left(C - \frac{\int_{\Gamma(t)} C \, ds}{\int_{\Gamma(t)} 1 \, ds} \right) N. \tag{21}$$

TABLE III

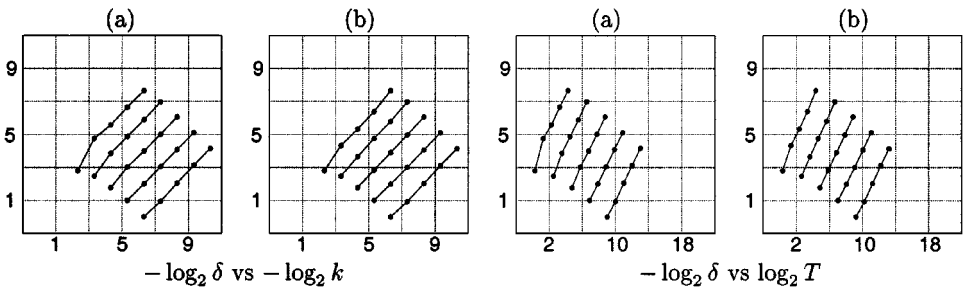
Maximum Error δ at $R = 1$, $t = 2$ and CPU Seconds T per Step versus Number N of Time Steps and Number L of Tree Levels in Two Circles Shrinking under Curvature for $0 \le t \le 2$ on $[-4, 4]^2$

(a) Minimal smoothing

L/S	$N=10$	20	40	80	160	320	640	1280	2560	T/NL
4/1	.143	.179	.290	.493	.991	—	—	—	—	.020
5/1	—	.037	.069	.122	.248	.518	—	—	—	.041
6/2	—	—	.021	.034	.063	.121	.239	—	—	.085
7/2	—	—	—	.010	.017	.031	.059	.114	—	.164
8/3	—	—	—	—	.005	.008	.015	.029	.056	.324

(b) Natural smoothing

L/S	$N=10$	20	40	80	160	320	640	1280	2560	T/NL
4/1	.143	.179	.290	.493	.991	—	—	—	—	.021
5/2	—	.050	.080	.139	.250	.523	—	—	—	.044
6/3	—	—	.025	.037	.067	.124	.244	—	—	.089
7/4	—	—	—	.012	.018	.032	.060	.115	—	.180
8/5	—	—	—	—	.005	.008	.015	.029	.057	.363



This velocity smooths the moving interface by curvature while preserving the area inside the interface, so arbitrary shapes become round points but the interface does not vanish. Small isolated pieces disappear and their area is transferred to large pieces.

We study a tilted square spiral unwinding under this velocity in Fig. 20, where 7/512/1 and 8/1024/2 runs converge to graphical accuracy. The 7/512/1 run took 18 min of CPU time and conserved area to within 0.9% of its initial value. Each successive refinement quadruples the CPU time and halves the area error, confirming the expected $O(N \log N)$ cost per time step and first-order accuracy. The interface is shown at geometrically increasing times $t = 0, 0.005, 0.01, 0.02, \ldots, 2.56$, because its motion slows dramatically as curvature variation decreases toward its final steady state.

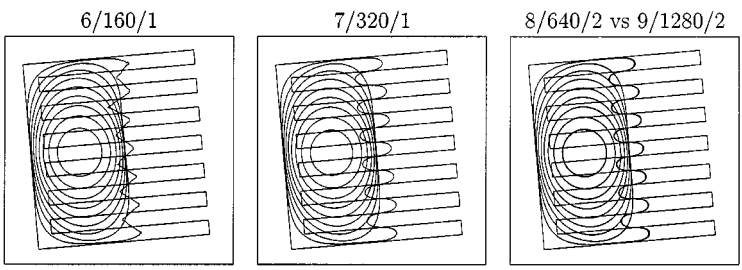


FIG. 18. Tilted polygon shrinking under curvature flow.

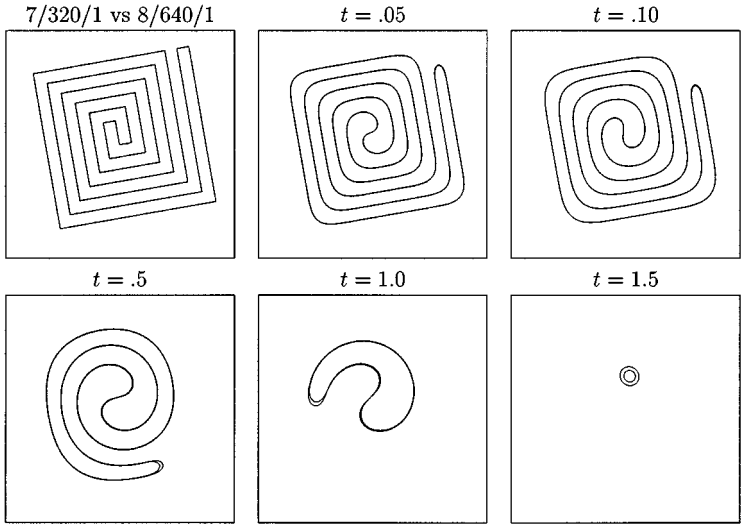


FIG. 19. Tilted square spiral unwinding under curvature flow.

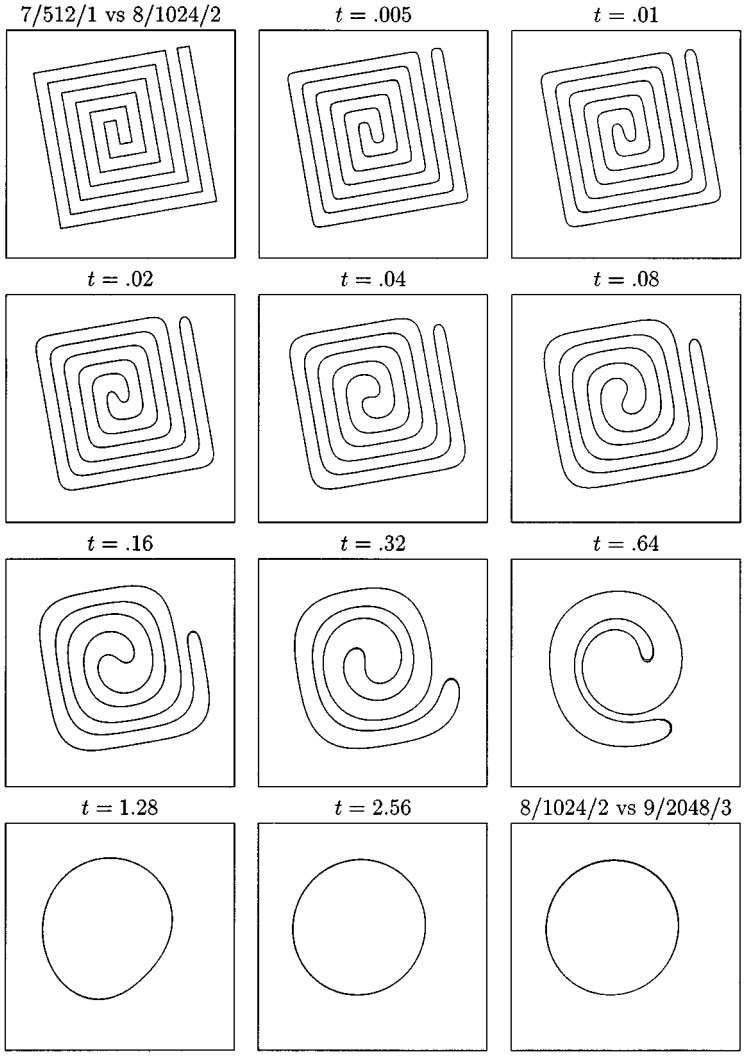


FIG. 20. Spiral unwinding under volume-preserving flow by curvature.

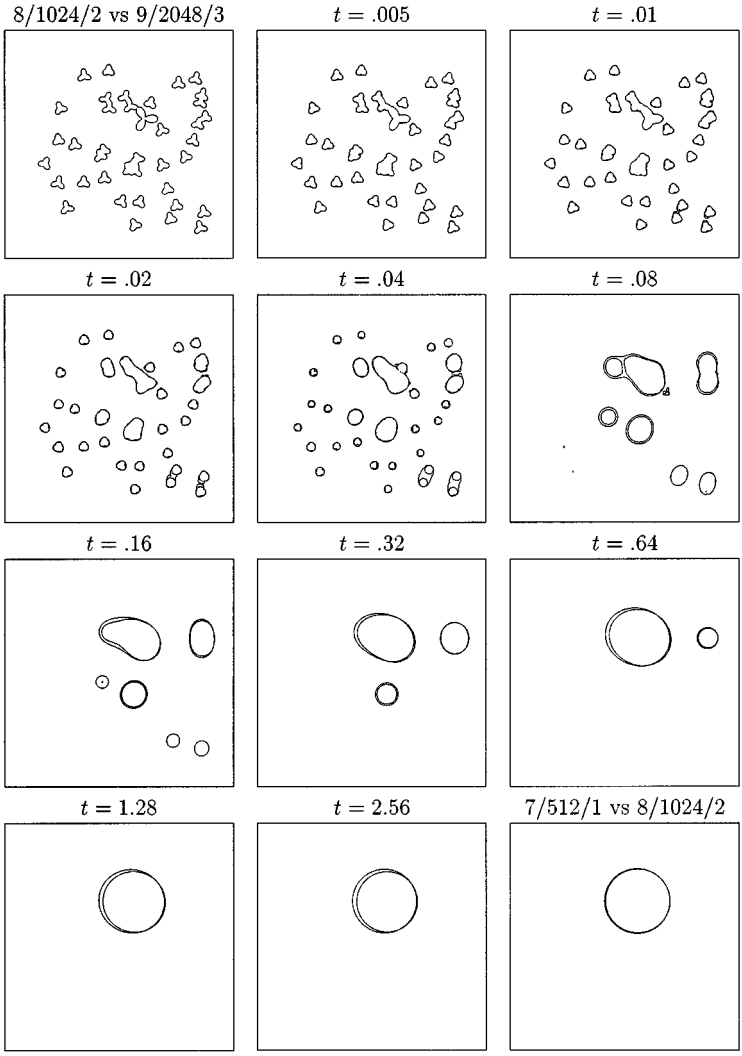


FIG. 21. Trefoils merging under volume-preserving flow by curvature.

Figure 21 superimposes 8/1024/2 and 9/2048/3 runs for a collection of trefoil-shaped bubbles moving under Eq. (21). Convergence to graphical accuracy is evident. The 8/1024/2 computation took 31 min CPU time and conserved the initial area within 8% accuracy. The 9/2048/3 computation took 140 min CPU time and lost 4% of its area by $t = 2.56$.

ACKNOWLEDGMENTS

The author thanks the referees for many helpful suggestions.

REFERENCES

1. 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).
2. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications* (Springer-Verlag, Berlin, 1997).

3. M. A. Grayson, The heat equation shrinks embedded plane curves to round points, *J. Diff. Geom.* **26**, 285 (1987).
4. A. Harten, S. Osher, B. Engquist, and S. R. Chakravarthy, Uniformly high order accurate essentially non-oscillatory schemes, *J. Comput. Phys.* **71**, 231 (1987).
5. M. Hirsch and S. Smale, *Differential Equations, Dynamical Systems and Linear Algebra* (Academic Press, New York, 1974).
6. D. E. Knuth, *Sorting and Searching*, Vol. 3 of *The Art of Computer Programming* (Addison-Wesley, Reading, MA, second edition, 1998).
7. J. D. McCalpin, A quantitative analysis of the dissipation inherent in semi-Lagrangian advection, *Mon. Weather Rev.* **116**, 2330 (1988).
8. W. F. Mitchell, A comparison of adaptive refinement techniques for elliptic problems, *ACM Trans. Math. Software* **15**, 326 (1989).
9. P. J. Rasch and D. L. Williamson, On shape-preserving interpolation and semi-Lagrangian transport, *SIAM J. Sci. Stat. Comput.* **11**, 656 (1990).
10. J. D. Sethian and J. Strain, Crystal growth and dendritic solidification, *J. Comput. Phys.* **98**, 231 (1992).
11. J. D. Sethian, *Level Set Methods and Fast Marching Methods* (Cambridge University Press, Cambridge, UK, 1999).
12. P. K. Smolarkiewicz and W. W. Grabowski, The multidimensional positive definite advection transport algorithm: Nonoscillatory option, *J. Comput. Phys.* **86**, 355 (1990).
13. P. K. Smolarkiewicz and J. Pudykiewicz, A class of semi-Lagrangian approximations for fluids, *J. Atmos. Sci.* **49**, 2082 (1992).
14. P. K. Smolarkiewicz and P. J. Rasch, Monotone advection on the sphere—An Eulerian versus semi-Lagrangian approach, *J. Atmos. Sci.* **48**, 793 (1991).
15. A. Staniforth and J. Côté, Semi-Lagrangian schemes for atmospheric models—A review, *Mon. Weather Rev.* **119**, 2206 (1991).
16. E. Stein, *Singular Integrals and Differentiability Properties of Functions* (Princeton University Press, Princeton, NJ, 1970).
17. J. Strain, Fast adaptive 2D vortex methods, *J. Comput. Phys.* **132**, 108 (1997).
18. J. Strain, Fast tree-based redistancing for level set computations, *J. Comput. Phys.* **152**, 664 (1999).
19. J. Strain, Semi-Lagrangian methods for level set equations, *J. Comput. Phys.* **151**, 498 (1999).
20. J. Strain, Tree methods for moving interfaces, *J. Comput. Phys.* **151**, 616 (1999).
21. J. Taylor, J. W. Cahn, and C. A. Handwerker, Geometric models of crystal growth, *Acta Met. Mat.* **40**, 1443 (1992).
22. C. Truesdell and R. A. Toupin, The classical field theories, in *Handbuch der Physik III/1*, edited by S. Flügge (Springer-Verlag, Berlin, 1960).
23. H. Whitney, Analytic extensions of differentiable functions defined in closed sets, *Trans. Am. Math. Soc.* **36**, 63 (1934).