LBL–34893

A Fast Level Set Method for Propagating Interfaces

David Adalsteinsson*
James A. Sethian†

Lawrence Berkeley Laboratory
and
Department of Mathematics
University of California
Berkeley, CA 94720

September 1994

*Supported by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098

†Supported in part by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098 and by the National Science Foundation/ARPA under grant DMS-8919074

A Fast Level Set Method for Propagating Interfaces

David Adalsteinsson* James A. Sethian†

September 1994

Abstract

A method is introduced to decrease the computational labor of the standard level set method for propagating interfaces. The fast approach uses only points close to the curve at every time step. We describe this new algorithm and compare its efficiency and accuracy with the standard level set approach.

1 A Fast Level Set Implementation

The level set technique was introduced in [9] to track moving interfaces in a wide variety of problems. It relies on the relation between propagating interfaces and propagating shocks. The equation for a front propagating with curvature dependent speed is linked to a viscous hyperbolic conservation law for the propagating gradients of the fronts. The central idea is to follow the evolution of a function ϕ whose zero–level set always corresponds to the position of the propagating interface. The motion for this evolving function ϕ is determined from a partial differential equation in one higher dimension which permits cusps, sharp corners, and changes in topology in the zero–level set describing the interface. (For details, see [11].)

Since its introduction, the level set approach has been used to compute and analyze a broad array of physical and mathematical phenomena, including singularities in mean curvature flow [10, 3] motivated by work in [5],

*Supported by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098

†Supported in part by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098 and by the National Science Foundation/ARPA under grant DMS-8919074

crystal growth and dendrite solidification [12], combustion [13], shape recognition [7, 6], minimal surface generation [2], two fluid problems [8] and triple junction problems [1]. In addition it has formed the basis for several theoretical investigations, see [4]. A review of the level set approach may be found in [11]. The generality of this approach makes it very attractive, especially for problems in three space dimensions, problems with sensitive dependence on curvature (such as surface tension problems) and problems with complex changes of topology.

For a one–dimensional interface evolving in two space dimensions, the level set algorithm is an $O(n^2)$ method per time step, where n is the number of points in the spatial direction. One drawback of the technique stems from the expense; by embedding the interface as the zero–level set of a higher dimensional function, a one–dimensional interface problem has been transformed into a two–dimensional problem. In three space dimensions, considerable computational labor ($O(n^3)$) is required per time step.

In this paper we provide a technique to reduce the computational labor involved in the level set technique for two space dimensions. The central idea is to build an adaptive mesh around the propagating interface; that is, a thin band of neighboring level sets, and perform computation only on these grid points. While some programming complexity is introduced, the savings in computational labor are significant and desirable in certain applications.

There is another, more substantial, reason to focus the level set update on a narrow band around the zero–level set. In some problems, the velocity field is only given on the interface, see, for example, the boundary integral crystal growth formulation given in [12]. In such problems, the construction of an appropriate speed function for the entire domain that identifies with the speed function of the zero–level set can be a significant modeling problem; this is known as the "extension problem", see [7, 12]. By performing a narrow band update of the level set, one need only construct this speed function close to the zero–level set.

1.1 The standard level set method and fast tube approach

A brief summary of the level set approach is as follows: Suppose we wish to follow the evolution of a curve γ_0 as it propagates in a direction normal to itself with speed F. We can then match the one parameter family of moving curves γ_t with a one parameter family of moving surfaces in such a way that the zero–level sets always yield the moving front. All that remains is to find an equation of motion for the evolving surface.

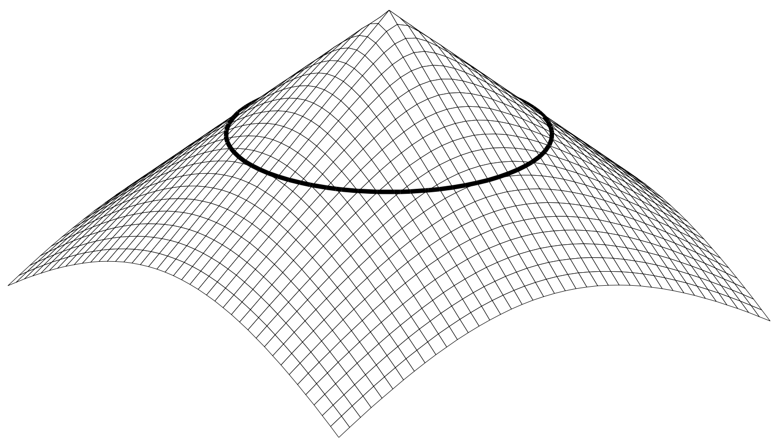


Figure 1: The signed distance function, defined on a rectangle.

Here, we follow the derivation given in [8]. Let γ_0 be a closed, non–intersecting curve. Assume $\phi(\boldsymbol{x}, t)$, $\boldsymbol{x} \in R^2$, is a scalar function such that at time t the zero–level set of $\phi(\boldsymbol{x}, t)$ is the curve γ_t. We further assume $\phi(\boldsymbol{x}, 0) = \pm d(\boldsymbol{x})$, where $d(\boldsymbol{x})$ is the distance from $\boldsymbol{x}$ to the curve γ_0. We use the plus sign if $\boldsymbol{x}$ is inside γ_0 and the minus sign if $\boldsymbol{x}$ is outside. As an example, if the initial front γ_0 is a circle in the (x, y) plane with radius 1, the $z = \phi(x, y, t = 0)$ surface given in figure 1.

Let each level set of ϕ flow along its gradient field with speed F. This speed function should match the desired speed function for the zero–level set of ϕ. Now consider the motion of some level set $\phi(\boldsymbol{x}, t) = C$. Let $\boldsymbol{x}(t)$ be the trajectory of a particle located on this level set, so

$$\phi(\boldsymbol{x}(t), t) = C.$$

The particle speed $\partial \boldsymbol{x}/\partial t$ in the direction $\boldsymbol{n}$ normal to the level set is given by the speed function F. Thus

$$\frac{\partial \boldsymbol{x}}{\partial t} \cdot \boldsymbol{n} = F,$$

where the normal vector $\boldsymbol{n}$ is given by $\boldsymbol{n} = -\nabla\phi/|\nabla\phi|$. This is a vector pointing outward, given our initialization of ϕ. By the chain rule,

$$\phi_t + \frac{\partial \boldsymbol{x}}{\partial t} \cdot \nabla\phi = 0.$$

Therefore ϕ is the solution to the differential equation

$$\begin{aligned} \phi_t - F|\nabla\phi| &= 0, \\ \phi(\boldsymbol{x}, t=0) &= \pm d(\boldsymbol{x}). \end{aligned}$$

At any time, the moving front γ_t is just the zero–level set of ϕ.

If the speed function F of the front depends on the curvature, the curvature may be expressed in terms of ϕ by

$$F = \frac{\phi_{yy}\phi_x^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{xx}\phi_y^2}{(\phi_x^2 + \phi_y^2)^{3/2}}.$$

This is called an Eulerian formulation for front propagation, because it is written in terms of a fixed coordinate system in the physical domain. There are three advantages to such an approach. First, since the underlying coordinate system is fixed, discrete mesh points do not move and the stability problems that plagued the Lagrangian approximations may be avoided. Second, topological changes are handled naturally, since the zero–level set of ϕ need not be simply connected. Third, the above obviously extends easily to moving surfaces in three dimensions with appropriate expressions for the curvature (such as the mean or Gaussian curvature).

The above initial value partial differential equation may be approximated using spatial and temporal derivatives on a fixed grid. Since the evolution equation admits non–differentiable solutions (that is, corners and cusps in the propagating front), care must be taken to choose an approximation to the gradient which produces a conservative scheme satisfying the entropy solution posed in [10]. Details of such a construction are found in [11, 9].

Here, we modify the level set technique in a way that saves substantial computational expense. We consider points close to the curve at each time. One way to do so is to choose points that lie less than some given distance away from the curve, confining computations to these points, gives a tube–like domain containing the zero–level set; see, for example, the tube surface associated with figure 1 is given in figure 2.

With such a construction, the work is cut down to $O(nk)$, where k is the width of the tube. With careful programming, a commensurate reduction in required memory is possible.

1.2 Building and Evolving the Narrow Band

In this section we present an overview of the fast level set algorithm.

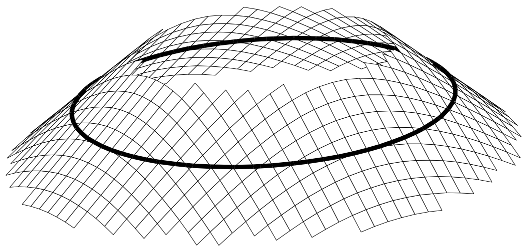


Figure 2: The signed distance function, defined on a tube.

1.2.1 The tube

To execute the fast level set approach, we begin by building the tube where the ϕ function will be defined. We make a tube containing all the points with distance to the curve less than $maxDist$ by calculating the distance function and using that to select the points. Rather than calculate the distance from each grid point to the initial curve (which would require $O(n^3)$ operations), we extend out from the initial curve approximately k grid points, and accurately calculate the distance function only at such points; this requires $O(nk^2)$ operations. The ϕ function is then initialized to be the signed distance function.

As the zero–level set corresponding to the front evolves, we must ensure that it stays within the tube. One way to do so would be to reconstruct a new tube around the curve at each time step. This requires at every time step the time–consuming procedure of determining which points make up the domain, deciding how to take the differentials at the edge points, and deciding how to define the surface on all the points inside the domain.

Instead, we use a given tube for as many iterations as possible; and devise a technique to trigger tube reinitialization when the front is close to the edge of the domain. During the life of a given tube, we can use the same initialization of ϕ, and design a data structure to speed up calculations.

1.2.2 Calculating the derivatives

Particular care must be taken when calculating partial derivatives at the edge points of the tubular domain. We can calculate the first order derivatives with central or one–sided differences at all the points in the domain. We calculate the second order derivatives by standard stencils in the interior and get the values on the edges by linear extrapolation from the newly computed values (see page ??.). To do so, we must smooth the domain to simplify the analysis and to exclude points where one–sided differences for the first order derivatives are not available, or where there is some ambiguity about which direction to use for the extrapolation. Higher order derivatives are evaluated by repeated use of first or second order derivatives.

1.2.3 Rebuilding the tube

The above algorithm must detect when the curve is getting too close to the edge of the tube. Detection can not wait until the curve has moved out of the domain because of accuracy degredation near the tube edges. Additionally some forms of constructing the boundary derivatives can result in a slight instability. This can be avoided through a careful monitoring of the evolving front. When a new tube is required, we reinitialize, using the current zero–level set as the initial curve; in the case of the observed instability we must undo the last time step.

Finally, the computational labor may be further decreased by noting that the tube is lying inside a rectangular array. If the front propagates inward, the full square array is unnecessarily large and it is made smaller to decrease both the memory and the computational time. Conversely, if the front is expanding, the matrix size is increased to make sure that the curve always remains in the computational domain. A border around the tube is always kept to ease the calculation of the signed distance for the next reinitialization.

1.3 The algorithm

Given an initial curve,

1. Calculate the signed distance on a tube around the curve. Precalculate as much as possible to decrease the computation for each time step.

2. Calculate the update matrix on the domain and evolve the function ϕ on the tubular domain.

3. If the curve is within a set distance of the tube boundary, or instability is developing on the tube boundary, reinitialize the surface by going to step 1, and resize the rectangular square where we do the calculations.

4. Otherwise go to step 2.

The outline of this paper is as follows. In Section 2 we provide the details about the update of the tube around the front. In Section 3 we present timing results comparing the narrow band approach with the full level set technique.

2 Technical Details

Here we consider two different methods for updating the values inside the tube. The methods differ in their treatment of the boundary values on the edge of the tube. The first fixes the values on the boundary, while the second extends values from the interior to the edges.

In both cases there are several common steps, such as calculating the signed distance, finding when to reinitialize the surface and storage techniques. There are some extra technical difficulties in extending the values to the edges, and they will be described later.

2.1 Calculating the signed distance in a tube

The signed distance function is defined as the distance from the given point to the curve and the sign is chosen to be positive if the point is inside the curve, and negative if outside. In our case we want to calculate the function only on the tubular domain. Outside the domain the value is defined to be $\pm maxDist$ depending on whether the point lies inside or outside the curve. This "far–field" value for the signed distance function is useful when finding the correct signs during next reinitialization.

As mentioned in the introduction, rather than computing the signed distance function for each grid point, we turn the roles around, and go along the curve first and then the evaluate the distance function at those points of the grid that lie close to the curve. This is a technique that can be used in other places, for example in the extension of a speed function from the curve to the tube.

Thus, we keep an array containing the current minimum distances. Initially, all the entries are set equal to $maxDist$. All segments on the curve are then tested by taking a square around each such segment and, for all points in that square, calculating the minimum distance to the curve segment. This yields the same matrix as if we had calculated the distance for all possible points and then truncated values at distance $= maxDist$. In order to construct the initial tube, we proceed as follows. First, we Initialize all points on the grid to be equal to $maxDist^2$. Then, for every curve segment in the curve, we take a box around the curve segment that includes all points that could be closer than $maxDist$ from it, and for every point (i, j) in that box calculate the square of the distance from the curve segment to that point and if it is less than $Grid(i, j)$ put it into $Grid(i, j)$. Finally when all the segments have been treated, we take the square root of each entry to produce the final matrix.

Finally, we must set the signs correctly. When the curve is the zero–level set of a known array, the sign of the signed distance function at (i, j) is the same as the sign of the array at that point. Therefore we define the points outside the domain to be $\pm maxDist$, even though they are not used at each time step. During the first initialization of the ϕ field, we determine the sign by other means, for example, by finding curve intersections.

2.2 Barriers

Next, we design a scheme to detect when the curve is getting closer to the edge than a preset minimum $howClose$.

One obvious technique is to calculate the exact distance between the curve and the edge. Such an approach is expensive and would dominate the time spent in each iteration. Note, however, that there is no real need to know the exact distance; we need only check whether the distance is less than a certain minimum distance or not. Thus, we use the ϕ value of the surface; if a point on the grid has a negative value it lies outside the curve, else it lies inside the curve. If the curve is less than $howClose$ away from the edge, we can find a point less than $howClose$ away from the edge that was initially inside the curve but is now outside or vice versa.

Therefore when we initialize the domain we find the level sets at heights $\pm(maxDist - howClose)$. We round those coordinates to the nearest points on the grid, and use the sign at those points. This constructs a barrier approximately $maxDist - howClose$ away from the initial curve ($howClose$ away from the edge). We store two sets of points, the barrier lying inside (the values should be > 0) and the barrier lying outside. At each time step we check if any of these points changes sign.

We must also be able to detect when there is an instability forming at the edge, i.e., if any of the edge points changes sign. To do this, we have to put edge points into two bins; those lying inside the curve, and those lying outside the curve.

2.3 Storage and data structures

When the curve gets too close to the edge, we must reinitialize and prepare a data structure that will speed up the calculation during the lifetime of that tube. This data structure includes:

- Information on the interior of the domain, used when taking the derivatives.
- Information on the total domain, used when updating the surface.
- The barriers, used for deciding when reinitialization is required.

The majority of the values in the domain are interior points. This part, where most of the calcuation takes place, is essentially contiguous in memory, and should be handled in the same way as in the standard method, that is, as a consecutive list of numbers in an array. However, the lists are different in length, and start and end at different locations. In our implementation, the matrix is stored such that (i, j) and $(i+1, j)$ lay side by side in memory, and for each j coordinate we store the start and end i coordinate of the list. For example for the y coordinate $j = 35$ we might store $\{[5, 30], [40, 80]\}$ meaning that $(5, 35) \cdots (30, 35)$ and $(40, 35) \cdots (80, 35)$ are in the interior. We could also just store the offsets of the beginning and end points of each segment. For the edges and the barriers we store the offsets from the first point in the array (everything is done with pointers to allow the resizing). This is considerably faster than storing the pairs (i, j). Since we know the dimensions of the array it is easy to find where $(i \pm 1, j \pm 1)$ is in memory if we know the location of (i, j).

One of the benefits of the tube method is less memory use. To make implementation easier the surface is stored in a full size grid, but all derived quantities, such as curvature and gradients, only have to be stored at the interior points. This reduces memory consumption considerably.

2.4 Technical details in extensions

If the tube values are fixed, this is straight forward. If they are not fixed, and are obtained by interpolating from the interior, we need to store information how interpolation should be done. In this implementation we use a linear interpolation from the two closest points in some direction. It is not always possible to choose these points to be interior points, and some cases can be ambiguous. For example, let $\times$ be an interior point and $\circ$ be an exterior point in the following drawing:

$$
\begin{array}{ccccccc}
\circ & \circ & \times & \times & \times & \times & \times \\
\times & \times & \circ & \times & \times & \times & \times \\
\times & \times & \times & \ast & \times & \times & \times \\
\times & \times & \times & \times & \circ & \circ & \times \\
\times & \times & \times & \times & \times & \circ & \circ
\end{array}
\qquad
\begin{array}{ccccccc}
\circ & \circ & \circ & \circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \ast & \circ & \circ & \circ \\
\circ & \times & \times & \times & \times & \times & \circ \\
\times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times
\end{array}\,.
$$

It is not clear how to interpolate the surface at the point marked by $\ast$.

However by removing some points from the tube it is possible to choose a direction in which the points are either interior points or edge points that can be interpolated by using only interior points, that is,

$$
\begin{array}{ccccccc}
\circ & \circ & \times & \times & \times & \times & \times \\
\times & \times & \circ & \times & \times & \times & \times \\
\times & \times & \times & \circ & \times & \times & \times \\
\times & \times & \times & \times & \circ & \circ & \times \\
\times & \times & \times & \times & \times & \circ & \circ
\end{array}
\qquad
\begin{array}{ccccccc}
\circ & \circ & \circ & \circ & \circ & \circ & \circ \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ \\
\circ & \times & \times & \times & \times & \times & \circ \\
\times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times
\end{array}\,.
$$

This process is called smoothing.

2.4.1 Smoothing

In this implementation we only need to remove points $\ast$ where the neighborhood contains one of the following four patterns:

$$
\begin{array}{ccc}
 & & \\
\circ & \ast & \circ \\
 & &
\end{array}
\qquad
\begin{array}{ccc}
\circ & & \\
 & \ast & \\
 & & \circ
\end{array}
\qquad
\begin{array}{ccc}
 & \circ & \\
 & \ast & \\
 & \circ &
\end{array}
\qquad
\begin{array}{ccc}
 & & \circ \\
 & \ast & \\
\circ & &
\end{array}\,.
$$

To remove those effectively we construct a byte map of the matrix. The neighborhood of each point can be represented by a 8 bit number where the neighboring points have been assigned different powers of 2. Bit operations can be used to find out how the neighborhood looks like and if it contains a specific pattern. Bit opererations are also used when removing points from the neighborhood.

As an example the number $121 = (01111001)_{\mathrm{bin}}$ might correspond to the neighborhood:[1]

$$
\begin{array}{ccc}
\circ & \times & \circ \\
\times & \ast & \circ \\
\times & \times & \times
\end{array}\,.
$$

[1] By starting at the upper left corner and going in a clockwise direction around $\ast$.

2.4.2 Choosing extension direction

We distinguish between 3 different types of edge points when doing the extension. Every edge point belongs to one of these types.

1. A point is of type 1 if the only exterior point in the neighborhood is a corner point and a diagonal can be exploited using only interior points when doing the extrapolation.
2. A point is of type 2 if a horizontal or vertical direction can be exploited using only interior points when doing the extrapolation.
3. A point is of type 3 if one of the side points is an exterior point and a diagonal can be exploited using interior points or points of the previous types when doing the extrapolation.

When the edge points have been sorted into these 3 types, the extrapolation can be performed by first finding the values at type 1 and type 2 edge points, and then finding the values at type 3 edge points. This then requires 2 sweeps. Each sweep can be done in any order.

In this implemenation we split the edgepoints into 12 bins, depending on the type of the point and direction of the extension. This information is stored along with the domain and barrier information.

2.4.3 Sorting the edgepoints

It is simple to determine what type a given edgepoint is. Assume that the edgepoint has the coordinate (i, j). Without loss of generality we can assume that the neighborhood as one of the three following forms (Here we use the fact that we have excluded some neighborhoods).

$$\begin{array}{ccc} \circ & \times & \times \\ \times & \ast & \times \\ \times & \times & \times \end{array} \quad \begin{array}{ccc} & \circ & \\ \times & \ast & \times \\ & \times & \end{array} \quad \begin{array}{ccc} & \circ & \\ \circ & \ast & \times \\ & \times & \end{array} .$$

In the first case, the point is either of type 1 or type 2, in the second case the point is either of type 2 or type 3, and in the third case the point is of type 3.

More precisely:

- First case:
    1. If $(i+1, j-1)$ and $(i+2, j-2)$ are interior points, put (i, j) into a type 1 bin.

2. If point $(i+2, j)$ is in the interior we go to the right, else we go down. (Because then $(i, j-2)$ is in the interior). (i, j) is put in a type 2 bin.

- Second case:

  1. If $(i, j-2)$ is an interior point, put (i, j) into a type 2 bin.
  2. If $(i+1, j-2)$ is of the form

$$\begin{array}{ccc} \times & \times & \\ \times & \ast & \times \\ & \times & \end{array},$$

  we can go down to the right. Else we go down to the left. (i, j) is put into a type 3 bin.

- In the third case it is possible to go down to the right, interpolating over points of type 1 or type 2; thus this point is of type 3.

2.4.4 Proof

The proof that this algorithm clasifies all edge points involves taking all the cases seperately, and is somewhat tedious but straightforward. We only show how this is done for one of the cases.

But first we need to introduce a technique that is used in proving all the cases.

To begin, imagine an interior point (that is, a point inside the tube). That point has a well–defined closest point p on the front that was used to make the domain. Take any point in the exterior, and draw the line which is always equidistant to it and the interior point. Then we know that p is on the same side of the line as the interior point. If we take several exterior points, but always the same interior point, we can restrict the area where p can lie. This argument, which we call the "triangle argument", can be used to show that a certain point has to be an interior point.

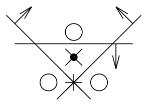
As an example, assume that the neighborhood is given schematically by

$$\begin{array}{ccc} & \times & \\ \bigcirc & \ast & \bigcirc \end{array}.$$

We want to show that the neighborhood in fact must be of the type

$$\begin{array}{ccc} & \times & \\ & \times & \\ \bigcirc & \ast & \bigcirc \end{array}.$$

Assume not. Using the triangle argument on the point directly above the $\ast$, we get the following picture:



Here, the arrows indicate on which side of the line the point p has to lie. The symbol $\divideontimes$ is the point we use in the triangle argument. This picture gives a contradiction, since it indicates that p lies inside the triangle, however we know that p has to lie further than $maxDist$ away from all exterior points.

We now introduce a notation for this argument, and write

$$\begin{array}{ccc} & \times & \\ \circ & \ast & \circ \end{array} \longrightarrow \begin{array}{ccc} & \times & \\ & \divideontimes & \\ \circ & \ast & \circ \end{array}.$$

Using the same point, we can add one more interior point on top. Since we use the same point, we include it on the same picture and the notation becomes

$$\begin{array}{ccc} & \times & \\ \circ & \ast & \circ \end{array} \longrightarrow \begin{array}{ccc} & \times & \\ & \times & \\ & \divideontimes & \\ \circ & \ast & \circ \end{array}.$$

This example creates a triangle in which p has to lie. Note that the shape might be more general, such as a rectangle.

Note that the above assumes that a $\circ$ point is an exterior point. Since we will remove some edge points, this does not necessarily have to be true. In such cases however, two points to each side will be exterior points, which further reduces the area where p can lie.

Proof: Take the first neighborhood. (The others are similar) Either the neighborhood looks like

$$\begin{array}{ccccc} \circ & \times & \times & & \\ \times & \ast & \times & \times & \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \end{array},$$

in which the point is of type 1, or one of the following:

$$\begin{array}{ccc} \circ & \times & \times \\ \times & \ast & \times \\ \times & \times & \times \\ & \circ & \end{array} \quad \begin{array}{cccc} \circ & \times & \times & \\ \times & \ast & \times & \times \\ \times & \times & \times & \\ & \times & \circ & \end{array} \quad \begin{array}{cccc} \circ & \times & \times & \\ \times & \ast & \times & \times \\ \times & \times & \times & \times \\ & \times & \times & \\ & & \circ & \end{array} \quad \begin{array}{ccccc} \circ & \times & \times & & \\ \times & \ast & \times & \times & \\ \times & \times & \times & \times & \times \\ & \times & \times & \circ & \\ & & \times & & \end{array} \quad \begin{array}{ccccc} \circ & \times & \times & & \\ \times & \ast & \times & \times & \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \\ & & \times & \circ & \end{array} \quad \begin{array}{ccccc} \circ & \times & \times & & \\ \times & \ast & \times & \times & \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \circ \end{array},$$

in which it is possible to extrapolate over interior points by going to the right. Therefore the point is of type 2. For the first case, the proof consists of the following diagram:

$$\begin{array}{ccccccccccccccccc}
\bigcirc & \times & \times & & & & \bigcirc & \times & \times & \times & \times & & \bigcirc & \times & \times & \times & \times \\
\times & \ast & \times & \times & & \longrightarrow & \times & \ast & \bullet & \times & & \longrightarrow & \times & \ast & \times & \times & \times \\
\times & \times & \times & \times & \times & & \times & \times & \times & \times & \times & & \times & \times & \times & \bullet & \times \\
 & \times & \times & \times & & & & \times & \times & \times & & & & \times & \times & \times & \\
 & & \times & \bigcirc & & & & & \times & \bigcirc & & & & & \times & \bigcirc &
\end{array} .$$

The other cases are similar.

3 Results

3.1 Order of the method

In the standard level set method each step costs at least $O(n^2)$ operations. Each step of the tube method costs $O(nk)$ operations, where k is the width of the tube. This is because at each time step we have to calculate the derivatives at $O(nk)$ points, and then update the array at only $O(nk)$ points.

Reinitializing the domain costs $O(nk^2)$, and the bulk of the work is in computing the signed distance on the tube. Sorting the points into the different edges and barriers costs $O(nk)$. The calculation of the signed distance is a fairly simple calculation so the constant in front of nk^2 is fairly small. The signed distance also has to be calculated infrequently. In practice an insignificant amount of time is used to calculate the signed distance compared to evolving the surface.

3.2 Topological changes

In figures 3 and 4 we run tests to show that the ability to change topology is preserved. There are approximately 50 timesteps between each plotted curve. We reinitialized roughly 3 to 5 times between each plotted curve. The tube radius is 12 cells in each of these examples, and the gridsize is approximately 200 by 200.

3.3 Movement by curvature

We start with a circle of radius 1 and let it shrink according to its curvature. The exact solution shows that the curve vanishes at $t = 0.5$. We compare the exact solution with the computed solution at two times $t = 0.1$ and $t = 0.3$ and by also compare extinction times. To estimate the distance from the exact solutions, we calculate the area of the path and get an average radius. We run the test runs for dx equal to 0.04, 0.02, 0.01. For each of these step sizes we will compute with tube width 6 and 12 cells and where the tube width is always 0.24 units. For comparison we also run the same test using the standard level set technique on a rectangular array. Results are given in Tables 1–4.

3.4 Movement by constant speed

We consider an initial circle of radius 1 propagating outward with speed 1 and perform the same tests as before. The time step is fixed to be 0.0008 in all tests. Results are given in Tables 1–4.

3.5 Three alternate methods

We analyze three variations of the method:

Approach 1: Calculate the derivatives only in the interior. Calculate the update matrix by using the derivatives in the interior, and extend the result to the edges to get the update matrix.

Approach 2: Use the extension to determine the values of the derivatives on all of the domain. Use those derivatives to produce the update matrix on all of the domain.

Approach 3: Fix the values on the edge, and only calculate the update matrix on the interior.

3.6 Description of terms in the tables

Type (r, b) gives the tube radius r in in cells, and how many cells the path is allowed to move b before it is reinitialized.

Timings (T_1, T_2) is the execution time in seconds. The first number, T_1, is the time it took to run up to $t = 0.1$, the second, T_2, is the time required to run up to $t = 0.3$.

Dimensions This is the dimension of the square matrix. This is not applicable for the tube method, since the number of points changes at each reinitialization.

Stops This is the error in the stopping time. The circle should vanish at t=0.50s, but the numerical curve will not necessarily vanish at exactly that time. This is how many extra time steps it took to vanish. Therefore the absolute time would be $0.50 + dt*$Time steps.

Total This is the time it took the curve to vanish, measured in seconds.

Reinit This is the number of times the domain had to be reinitialized. The first number of times we reinitialize because the curve gets too close to the edge of the domain. The second one is the number of reinitializations due to instability on the edge.

Error This is the difference between the exact radius and the calculated radius. Measured radius = Exact radius + Error.

3.7 Conclusions

In comparing the three different methods, we see that keeping the values fixed on the edges is disastrous when evolving with the curvature. If we calculate the update matrix by using the derivatives in the interior, and using the extension to get the values at the edges, we get no instability on the edges. If we calculate the derivatives (using extensions for the second order derivatives) on all the domain we get instability on the edges.

Therefore the best approach is to calculate the derivatives in the interior by standard stencils, use that to compute the update in the interior, and then extend the values onto the edges.

dx	dt	Timings	Dimensions	Stops	Total	Error
0.04	8e-4	(5.0,15.1)	(75,75)	1	25	(2.1e-04,3.6e-04)
0.02	2e-4	(78.5,235.8)	(149,149)	1	390	(5.5e-05,8.7e-05)
0.01	5e-5	(1300,3902)	(301,301)	1	6500	(1.3e-05,2.1e-05)

Table 1: A circle shrinking by curvature on a square

Type	dx	Timings	Reinit	Stops	Total	Error
Case 1: Smoothing the update						
(6,3)	0.04	(2.0,5.3)	(9,0)	-2	7.3	(4.6e-4,1.6e-3)
(6,3)	0.02	(17,43)	(19,0)	-5	58	(1.8e-4,7.1e-4)
(6,3)	0.01	(133,347)	(38,0)	-11	460	(7.0e-5,3.2e-4)
(12,5)	0.04	(3.6,10.3)	(3,0)	0	14	(2.1e-4,6.1e-4)
(12,5)	0.02	(30,80)	(7,0)	-1	110	(5.5e-5,2.7e-4)
(12,5)	0.01	(235,624)	(15,0)	-4	840	(3.3e-5,1.6e-4)
(6,3)	0.04	(2.0,5.3)	(9,0)	-2	7.2	(4.6e-4,1.6e-3)
(12,6)	0.02	(30,80)	(8,0)	-1	110	(5.5e-5,2.9e-4)
(24,12)	0.01	(450,1200)	(8,0)	-1	1600	(1.3e-5,7.0e-5)
Case 2: Smoothing the derivatives						
(6,3)	0.04	(1.9,5.2)	(6,4)	-4	7.0	(4.3e-4,3.9e-3)
(6,3)	0.02	(16.2,42)	(6,18)	-0	57	(8.2e-5,-6.4e-4)
(6,3)	0.01	(130,341)	(18,39)	-16	456	(1.8e-4,6.5e-4)
(12,5)	0.04	(3.5,9.9)	(2,2)	-1	14	(2.5e-4,1.2e-3)
(12,5)	0.02	(29,79)	(1,13)	-2	106	(1.4e-4,5.0e-4)
(12,5)	0.01	(231,611)	(2,40)	-8	824	(7.8e-5,3.1e-4)
(6,3)	0.04	(2.0,5.3)	(6,4)	-4	7	(4.3e-4,3.9e-3)
(12,6)	0.02	(29,78)	(1,13)	-3	106	(1.4e-4,5.0e-4)
(24,12)	0.01	(438,1163)	(2,34)	-7	1580	(5.5e-5,2.6e-4)
Case 3: Fixed boundary						
(6,3)	0.04	(1.9,5.1)	(9,0)	-58	6.5	(9.0e-3,5.0e-2)
(6,3)	0.02	(16,42)	(18,0)	-136	55	(3.3e-3,1.8e-2)
(6,3)	0.01	(131,345)	(40,0)	380	480	(-6.2e-3,-3.2e-2)
(12,5)	0.04	(3.6,10.2)	(3,0)	-5	14	(1.2e-4,3.0e-3)
(12,5)	0.02	(30,78)	(7,0)	-167	100	(7.0e-3,4.4e-2)
(12,5)	0.01	(232,607)	(15,0)	-892	760	(8.7e-3,4.8e-2)
(6,3)	0.04	(2.0,5.2)	(9,0)	-58	6.5	(9.0e-3,5.0e-2)
(12,6)	0.02	(30,78)	(8,0)	-64	100	(5.2e-3,2.0e-2)
(24,12)	0.01	(445,1190)	(8,0)	85	1600	(-8.8e-4,-5.9e-3)

Table 2: A circle shrinking by curvature on a tube.

dx	dt	Timings	Dimensions	Error
0.04	8e-4	(3.3,9.7)	(75,75)	(9.7e-4,2.4e-3)
0.02	8e-4	(13,38)	(149,149)	(4.4e-4,1.2e-3)
0.01	8e-4	(51,154)	(301,301)	(2.1e-4,5.7e-4)

Table 3: Expanding with speed 1 on a square

Type	dx	Timings	Reinits	Error
Case 1: Smoothing the update				
(6,3)	0.04	(1.2,4.2)	(3,0)	(9.7e-4,2.6e-3)
(6,3)	0.02	(2.9,10.3)	(7,0)	(4.8e-4,1.3e-3)
(6,3)	0.01	(7.9,29)	(16,0)	(2.4e-4,6.6e-4)
(12,5)	0.04	(2.2,7.3)	(1,0)	(9.7e-4,2.5e-3)
(12,5)	0.02	(4.8,17)	(3,0)	(4.4e-4,1.2e-3)
(12,5)	0.01	(11.3,39)	(6,0)	(2.2e-4,6.0e-4)
(6,3)	0.04	(1.1,4.3)	(3,0)	(9.7e-4,2.6e-3)
(12,6)	0.02	(4.8,17)	(3,0)	(4.4e-4,1.2e-3)
(24,12)	0.01	(18,68)	(3,0)	(2.1e-4,5.8e-4)
Case 2: Smoothing the derivatives				
(6,3)	0.04	(1.2,4.2)	(3,0)	(9.7e-4,2.6e-3)
(6,3)	0.02	(2.9,10.3)	(7,0)	(4.8e-4,1.3e-3)
(6,3)	0.01	(7.8,29)	(16,0)	(2.4e-4,6.6e-4)
(12,5)	0.04	(2.3,7.4)	(1,0)	(9.7e-4,2.5e-3)
(12,5)	0.02	(4.7,17)	(3,0)	(4.4e-4,1.2e-3)
(12,5)	0.01	(11,39)	(6,0)	(2.2e-4,6.0e-4)
(6,3)	0.04	(1.1,4.1)	(3,0)	(9.7e-4,2.6e-3)
(12,6)	0.02	(4.7,17)	(3,0)	(4.4e-4,1.2e-3)
(24,12)	0.01	(18,67)	(3,0)	(2.1e-4,5.8e-4)
Case 3: Fixed boundary				
(6,3)	0.04	(1.1,4.0)	(3,0)	(1.0e-3,2.8e-3)
(6,3)	0.02	(2.9,10.2)	(7,0)	(5.2e-4,1.4e-3)
(6,3)	0.01	(7.8,29)	(16,0)	(2.6e-4,7.3e-4)
(12,5)	0.04	(2.2,7.2)	(1,0)	(9.7e-4,2.5e-3)
(12,5)	0.02	(4.7,16.6)	(3,0)	(4.4e-4,1.2e-3)
(12,5)	0.01	(11.1,38)	(6,0)	(2.2e-4,6.0e-4)
(6,3)	0.04	(1.1,4.2)	(3,0)	(1.0e-3,2.8e-3)
(12,6)	0.02	(4.7,17)	(3,0)	(4.4e-4,1.2e-3)
(24,12)	0.01	(18,67)	(3,0)	(2.1e-4,5.8e-4)

Table 4: Expanding with speed 1 on a tube.

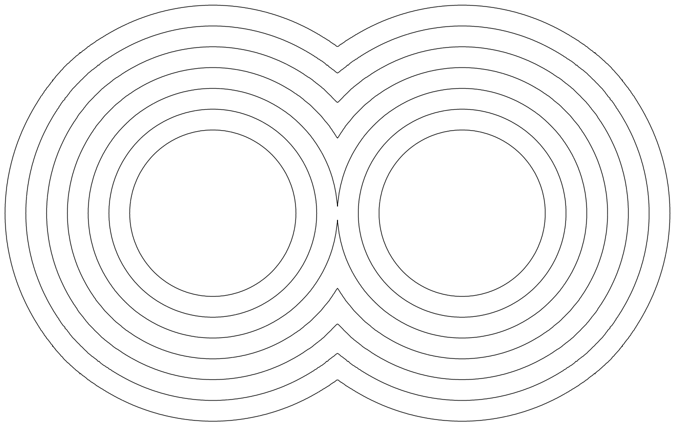


Figure 3: Two circles expanding with constant speed.

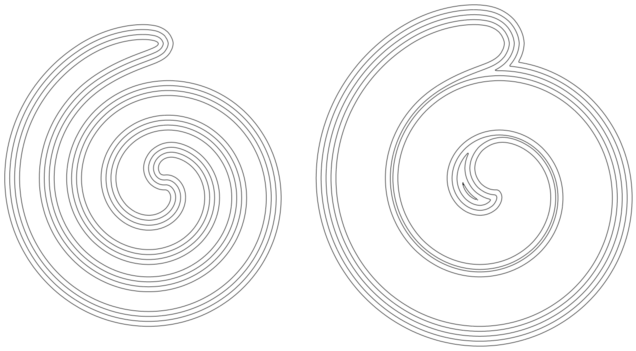


Figure 4: Spiral growing with constant speed.

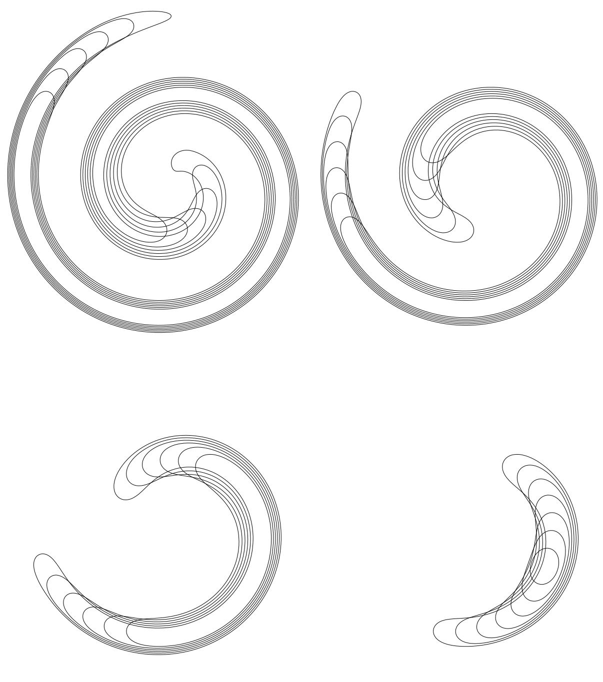


Figure 5: Spiral shrinking under curvature.

References

[1] Bence J., Merriman B., Osher S.J. Motion of multiple triple junctions: A level set approach. *To appear J. Comp. Phys.*, 1994.

[2] Chopp D. Computing minimal surfaces via level set curvature flow. *J. Comp. Phys.*, 106(1):77–91, May 1993.

[3] Chopp D., Sethian J.A. Curvature flow and singularity development. *Submitted for publication J. Experimental Mathematics.*, 1993.

[4] Evans L.C., Spruck J. Motion of level sets by mean curvature. *J. Diff. Geom.*, 33:635–681, 1988.

[5] Grayson M. The heat equation shrinks embedded plane curves to round points. *J. Diff. Geom.*, 26:285, 1987.

[6] Kimmel R., Kiryati N., Bruckstein A. Sub–pixel distance maps and weighted distance transforms. *To appear JMIV.*, 1994.

[7] Malladi R, Sethian J.A., Vemuri B. A shape detection scheme using level sets. *To appear IEEE Journal on Image Analysis.*, 1993.

[8] Mulder W., Osher S., Sethian J.A. Computing interface motion in compressible gas dynamics. *J. Comp. Phys.*, 100(2):209–228, Jun 1992.

[9] Osher S.J., Sethian J.A. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations. *J. Comp. Phys.*, 79:12–49, 1988.

[10] Sethian J.A. Curvature and the evolution of fronts. *Comm. Math. Phys.*, 101:487, 1985.

[11] Sethian J.A. A review of recent numerical algorithms for hypersurfaces moving with curvature dependent speed. *J. of Diff. Geom.*, 31:131–161, 1989.

[12] Sethian J.A., Strain J. Crystal growth and dentrite solidification. *J. Comp. Phys.*, 98(2):231–253, Feb 1992.

[13] Zhu J., Ronney P. Simulation of front propagation at large non–dimensional flow disturbance intensities. *To appear at Combustion Science and Technology.*, 1994.