The Fast Construction of Extension Velocities in Level Set Methods

D. Adalsteinsson and J.A. Sethian *

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

Submitted: Dec. 1997: The Journal of Computational Physics

Abstract

Level set techniques are numerical techniques for tracking the evolution of interfaces. They rely on two central embeddings; first the embedding of the interface as the zero level set of a higher dimensional function, and second, the embedding (or extension) of the interface's velocity to this higher dimensional level set function. This paper applies Sethian's Fast Marching Method, which is a very fast technique for solving the Eikonal and related equations, to the problem of building fast and appropriate extension velocities for the neighboring level sets. Our choice and construction of extension velocities serves several purposes. First, it provides a way of building velocities for neighboring level sets in the cases where the velocity is defined only on the front itself. Second, it provides a sub-grid resolution in some cases not present in the standard level set approach. Third, it provides a way to update an interface according to a given velocity field prescribed on the front in such a way that the signed distance function is maintained, and the front is never re-initialized; this is valuable in many complex simulations. In this paper, we describe the details of such implementations, together with speed and convergence tests, and applications to problems in visibility relevant to semi–conductor manufacturing and thin film physics.

1 Introduction

Level set methods, introduced by Osher and Sethian [10], offer highly robust and accurate methods for tracking interfaces moving under complex motions.

*Supported in part by the Applied Mathematics Subprogram of the Office of Energy Research under contract DE–AC03–76SF00098, and the National Science Foundation and DARPA under grant DMS–8919074.

They grew out of the theory of curve and surface evolution developed by Sethian in [13, 14, 16], which constructs the notion of weak solutions and entropy limits for evolving interfaces, and links upwind numerical methodology for hyperbolic conservation laws to front propagation problems. The resulting level set approach works in any number of space dimensions, handles topological merging and breaking naturally, and are easy to program. Details about the theory, implementation, and application of level set methods may be found in [18].

These techniques rely on two central embeddings; first the embedding of the interface as the zero level set of a higher dimensional function, and second, the embedding (or extension) of the interface's velocity to this higher dimensional level set function. More precisely, given a moving closed hypersurface $\Gamma(t)$, that is, $\Gamma(t=0):[0,\infty)\rightarrow R^N$, propagating with a speed F in its normal direction, we wish to produce an Eulerian formulation for the motion of the hypersurface propagating along its normal direction with speed F, where F can be a function of various arguments, including the curvature, normal direction, etc. Let $\pm d$ be the signed distance to the interface. If we embed this propagating interface as the zero level set of a higher dimensional function ϕ, that is, let $\phi(x,t=0)$, where $x\in R^N$ is defined by

$$\phi(x,t=0)=\pm d, \tag{1}$$

then an initial value partial differential equation can be obtained for the evolution of ϕ, namely

$$\phi_t+F|\nabla\phi|=0 \tag{2}$$

$$\phi(x,t=0) \quad \text{given} \tag{3}$$

This is known as the level set equation. As discussed in [13, 14, 16], propagating fronts can develop shocks and rarefactions in the slope, corresponding to corners and fans in the evolving interface, and numerical techniques designed for hyperbolic conservation laws can be exploited to construct upwind schemes which produce the correct, physically reasonable entropy solution.

The above formulation reveals two central embeddings.

1. First, in the initialization step (Eqn. 1), the signed distance function is used to build a function ϕ which corresponds to the interface at the level set $\phi=0$. This step is known as "initialization"; when performed at some later point in the calculation beyond $t=0$, it is referred to as "re–initialization". The need for re–initialization in level set methods was first discussed by Chopp in his work on minimal surfaces, see [8].

2. Second, the construction of the initial value PDE given in Eqn. 2 means that the velocity F is now defined for **all** the level sets, not just the zero level set corresponding to the interface itself. We can be more precise by rewriting the level set equation as

$$\phi_t+F_{ext}|\nabla\phi|=0 \tag{4}$$

where F_{ext} is some velocity field which, at the zero level set, equals the given speed F. In other words,

$$F_{ext} = F \text{ on } \phi = 0$$

This new velocity field F_{ext} is known as the "extension velocity", see Figure 1.

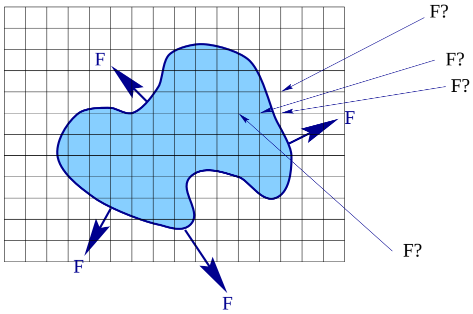


Figure 1: Constructing Extension Velocities

How much freedom does one have in this extension construction? In fact, there is considerable room to maneuver. The extension velocity F_{ext} should, in the limit as one approaches the zero level set, yield the speed F of the zero level set, i.e.

$$\lim_{x \to a} F_{ext(x)} = F(a),$$

where a is a point on the front. Beyond these requirements, considerable opportunities are available.

In this paper, we provide a fast methodology for constructing a particular choice of extension velocity. There are three reasons for doing so.

1. **No natural speed function:**. In some physical problems, the velocity at the interface only has meaning at the front itself. For example, in problems in semi-conductor manufacturing simulations of the etching and deposition process involve determinations of the visibility of the interface with respect to the etching/deposition beam, see [2, 3, 4]. This provides no natural velocity off the front, since it is unclear what is meant by the "visibility" of the other level sets. In this case, an extension velocity must be specifically constructed.

2. **Sub-grid resolution:**. As demonstrated in an example later in this paper, there are problems in which the speed of the interface changes very rapidly or discontinuously as the front moves through the domain. In such cases, the exact location of the interface determines the speed, and constructing a velocity from the position of the interface itself, rather than from the somewhat coarse grid velocities is desirable.

3. **Maintaining a nice level set representation:**. Under some velocities, such as those which arise in fluid mechanics simulations [6, 20], the level set functions have a tendency to either bunch up or spread out, which manifests itself as the ϕ becoming either very steep or flat. The extension velocity discussed here is designed so that an level set function initialized using the signed distance function is essentially maintained as the front moves; this will be demonstrated in the examples section. Thus, the algorithm both avoids re-initialization, which can often perturb the front, and also provides a technique which does not cause the bunching and stretching of neighboring level set lines which has led to mass conservation issues in some level set calculations.

The outline of this paper is as follows. First, we discuss some previous work on building extension velocities. Next, we discuss the Fast Marching Method, which is at the core of our technique. This then is followed by numerical tests and examples.

2 Construction of Extension Velocities: Previous Work

The need to build extension velocities in the context of level set methods has been recognized for some time, and is intertwined with two other algorithmic methodologies in level set methods, namely adaptive Narrow Band methods and level set re-initializations. Here, we review some of that work to set the stage for the current paper.

2.1 Building Extension Velocities

A variety of level set simulations have used extension velocities in one form or another. In many fluid simulations, one can choose to directly use the fluid velocity itself to act as F_{ext}. This is what was done in the two phase flow simulations of Chang, Hou, Merriman and Osher [6] and Sussman, Smereka and Osher [20]. These were incompressible flow calculations in which the velocity are continuous across the boundary. In these detailed and careful simulations, bunching and flattening of the level set function occurs, which is then repaired every time step through a re-initialization process which rebuilds the signed distance function through an iterative process given in [20], based on an observation of Morel.

In cases where there is no available choice for an extension velocity, one approach is to simply extrapolate; standing at each grid point, the value of the speed function at the closest point on the front is used as the extension velocity at that point. This is the approach used in [9].

Another is to build a speed function from the front using some other technique. In [19], a numerical simulation of dendritic solidification was performed. In this model, the velocity at the interface depended on a jump condition across the interface, and hence had no meaning for the other "non-physical" level sets. A boundary integral expression was developed for the velocity on the interface, and this boundary integral was evaluated both on and off the front to provide an extension velocity. A later work on crystal growth by Chen, Merriman, Osher, and Smereka [7] worked directly with the partial differential equations (rather than the conversion to a boundary integral), and built an extension velocity by solving an advection equation in each component again coupled to a re-initialization procedure; we refer the interested reader to [7] for a collection of impressive simulations performed using this approach.

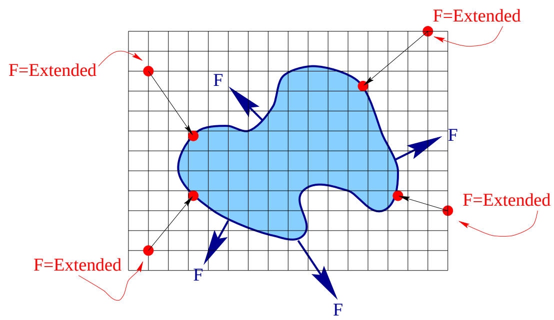


Figure 2: Constructing Extension Velocities by Extrapolation from Front

2.2 Narrow Band Level Set Methods and Re-initialization

The original level set method as introduced by Osher and Sethian in [10] updated all the level sets, not just the zero level set. Adalsteinsson and Sethian introduced Narrow Band level set methods in [1], which confine computation to a narrow band around the interface of interest. The narrow band was of arbitrary size. As the front moved and reached the edge of the narrow band, the calculation was stopped, and a new initial level set function corresponding to the signed distance function was re-built. This was known as "re-initialization". A very large narrow band meant that one was essentially computing everywhere, and this re-initialization was never performed. A very thin narrow band meant that one was computing only very close to the front, and hence re-initializing every time step. The numerical tests reported in [1] indicated that a narrow band of a particular size (around 6-10 grid points each side of the front) seemed to be the correct balance between work spent updating points in the band, and work spent doing re-initialization.

The papers of Zhao, et. al. [21] and Chen, et. al. [7] choose to use a limiting case of our Narrow Band methodology, work with a very small local band, and re-initialize every time step. Rather than characterize the band of points by their physical distance from the front, they opt to characterize those points by their ϕ values. Their narrow band is very small, and hence they re-initialize every time step. These re-initializations are performed in several ways. The most straightforward is to simply formally compute the distance to the zero level set, as was done in Chopp's [8] original work, this, however, is computational inefficient. As mentioned earlier, one approach is to use the iteration technique given in [20]. However, reinitialization every time step can lead to movement of the zero level set and must be performed extremely carefully, otherwise serious difficulties will result. If one's goal is to re-initialize only in a band one or two cells around the front, one might try an iterative technique. For greater distances, and indeed, as a methodology for building distance functions away from curves and surfaces in general, the Fast Marching Method offers a very fast ($O(N \log N)$ approach, where N is the total number of points) technique. This will be discussed in detail below.

2.3 Equations for Extension Velocities

What are desirable properties of an extension velocity? Obviously it should match the given velocity on the front itself. Another desirable feature is that it move the neighboring level sets in such a way that the signed distance function is preserved. Consider for a moment an initial signed distance function $\phi(x, t = 0)$, and suppose one builds an extension velocity of the form

$$\nabla F_{ext} \cdot \nabla \phi = 0, \tag{5}$$

as was discussed in [21]. Then it is straightforward to show that the level set function ϕ remains the signed distance function for all time, assuming that both F and ϕ are smooth. To see that this is so, (see [21]), suppose that initially $|\nabla \phi(x, t = 0)| = 1$, and we move under the level set equation $\phi_t + F_{ext}|\nabla \phi| = 0$; then we note that

$$\frac{d|\nabla \phi|^2}{dt} = \frac{d}{dt}(\nabla \phi \cdot \nabla \phi) = 2\nabla \phi \cdot \frac{d}{dt}\nabla \phi$$

$$= -2\nabla \phi \cdot \nabla F_{ext}|\nabla \phi| - 2\nabla \phi \cdot \nabla |\nabla \phi| F_{ext}.$$

The first term on the right is zero because the way the extension velocity is constructed; the second is zero because $|\phi(x, t = 0| = 1$. Thus, $|\nabla \phi| = 1$ is one solution to this equation; this plus a uniqueness result for this differential equation shows that $|\nabla \phi| = 1$ for all time.

Thus, our strategy is as follows. Given a level set function, in this paper we show how to simultaneously construct a signed distance and an extension velocity F_{ext} very rapidly using the Fast Marching Method. We then use this velocity to update the level set function ϕ. There are several important things to note about our proposed algorithm:

- This construction finds an extension velocity which is then used to update the level set function. One can, of course, use as a high order method as one chooses for the level set update. If one wants to perform this update restricted to a narrow band using the narrow band methodology of [1], one is free to do so. However, this methodology provides a way of doing so in all of space very rapidly, i.e., $O(N \log N)$, where N is the total number of points where one wants to build this extension velocity.

- In this approach, one can choose to never re-initialize the level set function. Our approach is as follows.

  1. Consider a level set function ϕ^n at time step $n\Delta t = 0$
  2. Build the extension velocity by finding simultaneously constructing a temporary signed distance function ϕ^{temp} and an extension velocity such that
  $$\nabla \phi^{temp} \cdot \nabla F_{ext} = 0,$$
  with ϕ^{temp} matching ϕ^n at their zero level sets, and F_{ext} matching the F given on the interface.
  3. Then advance the level set function ϕ^n under the computed extension velocity to produce a new ϕ^{n+1} by solving $\phi_t + F_{ext}|\nabla\phi| = 0$.

  Thus, the proposed algorithm never re-initializes the evolving level set function, yet moves it under a velocity field that maintains the signed distance function. This avoids a large set of problems that have plagued some implementations of level set methods, namely that re-initialization steps can perturb the position of the front corresponding to the zero level set. Better yet, the velocity field itself is quite smooth, and does not suffer from the undesirable bunching and stretching of level sets that have also plagued some level set calculations.

- In our approach, we are going to explicitly find the zero level set corresponding to the interface in order to build the extension velocity. One of the appeals of level set methods is that the front need not be explicitly constructed, and that all of the methodology may be executed on the underlying grid. Our approach is in fact to find the front in both two and three dimensional problems, however, we shall never move or update that representation. In cases of speed functions that depend on factors like visibility, this is completely natural. A central virtue of level set methods lies in the update of the level set function on a discrete mesh to embed the motion of the interface itself, rather than advance a discrete tracked representation of the front. This strategy and philosophy is maintained in our approach.

To summarize, our algorithm allows one to update an interface represented by an initial signed distance function according to a velocity field given on the front in such a way that the signed distance function is maintained, and the front is never re-initialized. If one chooses to use the adaptive methodologies given in the Narrow Band approach, occasional rebuilding of the narrow band may be required, but this is performed only sporadically.

In order to proceed with the algorithm, we now must review some aspects of the Fast Marching Method and show how it can be used to both construct signed distance functions and extension velocities; this is the subject of the next section.

3 The Fast Marching Method

Here, we briefly review the Fast Marching Method for computing the solution to the Eikonal equation, for details, see [17]. The goal is to solve the equation

$$|\nabla u| = F(x, y). \tag{6}$$

The key idea is to build an approximate to the gradient term which correctly deals with the development of corners and cusps in the evolving solution. It is well–known that the above Eikonal equation becomes non–differentiable, and an appropriate weak solution must be built; this is related to the entropy condition for propagating interfaces introduced in [14]. One of the simplest such upwind entropy–satisfying approximations to the gradient is due to Godunov, and was used, for example, by Rouy and Tourin [12] to solve the Eikonal equation, namely

$$\left[\begin{array}{c} \max(D_{ij}^{-x} u, -D_{ij}^{+x} u, 0)^2 + \\ \max(D_{ij}^{-y} u, -D_{ij}^{+y} u, 0)^2 \end{array} \right]^{1/2} = F_{ij}, \tag{7}$$

Additional schemes for solving Hamilton-Jacobi equations may be found in [11, 5].

The central idea behind the Fast Marching Method is to systematically advance the front in an upwind fashion to produce the solution u. The key idea is the observation that the upwind difference structure of Equation (7) means that information propagates "one way", that is, from smaller values of u to larger values. Hence, the Fast Marching Method rests on "solving" Equation (7) by building the solution outward from the smallest u value. The algorithm is made fast by confining the "building zone" to a narrow band around the front. The idea is to sweep the front ahead in an upwind fashion by considering a set of points in a narrow band around the existing front, and to march this narrow band forward, freezing the values of existing points and bringing new ones into the narrow band structure. The key is in the selection of *which* grid point in the narrow band to update.

The algorithm is as follows: Put the points into three sets, *Far*, *Close* and *Accepted*. We tag points in the initial conditions as *Accepted*. We then tag as *Close* all points less than one grid point away. Finally, we tag as *Far* all other grid points. Then the loop is

1. Begin Loop: Let *Trial* be the point in *Close* with the smallest value for u.
2. Move the point *Trial* into *Accepted*.
3. Move all the neighboring points of *Trial* that are in *Far* into *Close*

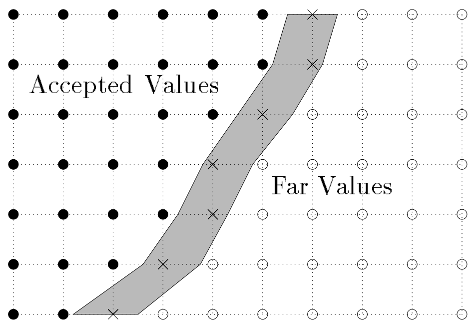


Figure 3: Upwind construction of Accepted Values

4. Recompute the values of u at all neighbors that are in *Close* according to Eqn. 7 by solving the quadratic equation, treating all points in *Close* and *Far* as if they had the value ∞.
5. Return to top of Loop.

This algorithm works because the process of recomputing the u values at upwind neighboring points cannot yield a value smaller than any of the accepted points. Thus, we can march the solution outward, always selecting the narrow band grid point with minimum trial value for u, and readjusting neighbors, (see Figure 3). Another way to look at this is that each minimum trial value begins an application of Huyghen's principle, and the expanding wave front touches and updates all others.

The speed of the algorithm comes from a heapsort technique to efficiently locate the smallest element in the set $Trial$. Let us now perform a quick operation count on the method. Suppose there are a total of N computational points in a domain, and one wants to solve the Eikonal equation away from an initial curve or surface Γ lying in this domain. Imagine that it took no time at all to locate the smallest trial value. Then since each point in the domain is touched only once during the update, the total operation count to construct the solution to the Eikonal equation is $O(N)$. By using heapsort methodology, the smallest such point can be located in $O(\log N)$, and hence the entire method is order $N \log N$. This is a very fast algorithm; in most cases $\log N$ is very small. We point out that if one wants to produce this Eikonal solution only very close to the front, (one or two points away), one might attempt to iterate the solution is done in [20]. However, beyond a small range around the boundary data Γ, this approach is less computationally efficient than the Fast Marching Method. Since we will use our algorithm to construct extension velocities any distance from the interface, the efficiency of Fast Marching Methods is desirable. For more details, see [17, 18].

4 Using the Fast Marching Method to Construct Signed Distances and Extension Velocities

Recall that given a level set function ϕ^n, our goal is to build an extension velocity F_{ext} such that if $|\nabla\phi| = 1$, then updating under this extension velocity maintains this unit gradient. The plan is to solve the equation

$$\nabla\phi^{temp} \cdot \nabla F_{ext} = 0$$

so that ϕ^{temp} is the signed distance function which has the same zero level set as the level set function ϕ^n. We stress that we do not use this computed signed distance to re-initialize the level set function; it is used instead only in the construction of F_{ext}.

4.1 Constructing signed distances

Suppose we are given a level set function ϕ^n, where the superscript n indicates the time step in the usual notation, and suppose that this level set function does not correspond to the signed distance function. We can use the Fast Marching Method to compute the signed distance ϕ^{temp} by solving the Eikonal equation

$$|\nabla T| = 1$$

on either side of the interface, with the boundary condition that $T = 0$ on the zero level set of ϕ. The solution T will then be our temporary signed distance function ϕ^{temp}. The Fast Marching Method is run separately for grid points outside and inside the front (we note whether a grid point is inside or outside is immediately apparent from the given level set function ϕ^n).

The only difficulty is in the initialization stage of the Fast Marching Method, that is, the computation of the approximate distances of the set of *Close* points in order to begin the Fast Marching Method. We now show how to find the initial set of *Close* values for grid points outside of a two dimensional front; points inside the front and points close to a three-dimensional surface are handled similarly.

Begin by initially tagging as *Close* those grid points where one of the neighbors lies inside the front. We must assign values at these points to approximate the distances to the front. While this can be computed exactly for a smooth front, a faster method can be designed which only uses the intersection of the front with the grid lines. This is particularly useful when the front is given as the zero level set of a function defined at the grid points, and a smooth representation is not available.

Up to rotation there are five possible cases that need to be considered, and are shown in Figure 4.

- In Figure 4a, only one of the neighboring points is on the other side of the front. Here we define the value to be the distance s to the intersection point on the line connecting the two grid points. This value is larger than

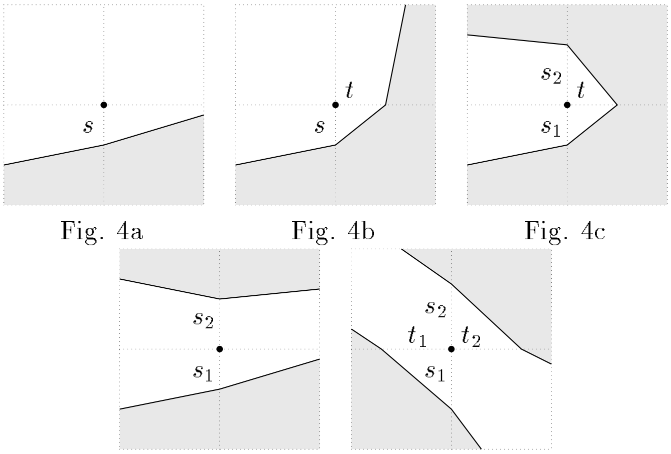


Fig. 4a Fig. 4b Fig. 4c

Fig. 4d Fig. 4e

Figure 4: All cases for the neighborhood of a point

the real distance to the front, but most likely the value at the grid point on the other side is the distance to the same point, so that the zero level set will not have moved after the reinitialization.

- In Figure 4b, two of the neighbors are on the other side of the front. In this case the value is defined as the exact distance to the line segment between the two intersection points. If s and t are the distances to the intersection points, the exact distance d satisfies

$$\left(\frac{d}{s}\right)^2 + \left(\frac{d}{t}\right)^2 = 1.$$

The left hand side is an upwind approximation to the gradient of the distance function, since the distance is zero at the intersection points. This suggests what the solution should be for the remaining three cases, and how it should be computed in 3D.

- In Figure 4c, the distance is the positive solution to

$$\left(\frac{d}{\min(s_1, s_2)}\right)^2 + \left(\frac{d}{t}\right)^2 = 1.$$

- In Figure 4d, the distance is

$$d = \min(s_1, s_2).$$

- In Figure 4e, the distance is the positive solution to

$$\left(\frac{d}{\min(s_1, s_2)}\right)^2 + \left(\frac{d}{\min(t_1, t_2)}\right)^2 = 1.$$

4.2 Constructing the Velocity Extensions F_{ext}

Our goal now is to extend a speed function given along an interface to grid points around the front. This extension should extend the speed in a continuous manner, and avoid, if possible, the introduction of any discontinuities in the speed close to the front. Thus, we want to construct a speed function F_{ext} that satisfies the equation

$$\nabla F_{ext} \cdot \nabla \phi^{temp} = 0.$$

The basic idea is march outwards according to the Fast Marching Method perspective, systematically and simultaneously attaching two values to each grid point, the distance from the front and the extended speed value. We first compute the signed distance ϕ^{temp} to the front using the Fast Marching Method as described in the previous section. As the Fast Marching Method constructs the signed distance at each grid point, we simultaneously update the speed value F_{ext} according to Eqn. 5. In the gradient stencil, we only use neighboring points closer to the front to maintain the upwind ordering of the point construction.

In more detail, and similar to the construction of signed distances, we need to first find the speed values for the inital set of *Close* points in order to begin the technique, and second, update the extension value when the distance value is updated according to the above equation.

One technique for building extension velocities near the front would be by copying the speed of the closest grid point, as was described earlier. Instead, we take a weighted average of the speed values at the points which are used in computing the distance, where the weight is proportional to one over the square of the distance; this is equivalent to solving the equation $\nabla F_{ext} \cdot \nabla \phi^{temp} = 0$.

As an example, consider the cases in Figure 4. For simplicity, assume that we are computing the extension value for the point (i, j) in the center.

- For Figure 4a, the extension speed is $f = f(i, j - s)$.
- For Figure 4b, the gradient is given by

$$\left(-\frac{d}{t}, \frac{d}{s} \right).$$

The equation $\nabla F_{ext} \cdot \nabla \phi^{temp} = 0$ is

$$\begin{aligned} 0 &= \left(-\frac{f - f(i+t, j)}{t}, \frac{f - f(i, j-s)}{s} \right) \cdot \left(-\frac{d}{t}, \frac{d}{s} \right) \\ &= d \left[\frac{f - f(i+t, j)}{t^2} + \frac{f - f(i, j-s)}{s^2} \right], \end{aligned}$$

in which case

$$f = \frac{\frac{1}{t^2} f(i+t, j) + \frac{1}{s^2} f(i, j-s)}{\frac{1}{t^2} + \frac{1}{s^2}}.$$

This equation indicates the solution for the remaining cases, and for the three dimensional case. Our expression assumes that the speed of the interface is given at the intersection points of the interface with the grid lines. If the speed is given at other points, one can either use interpolation to get the speed values, or modify the above algorithm.

- For Figure 4c, the equation is

$$f = \frac{\frac{1}{t^2} f(i+t,j) + \frac{1}{s^2} f(i,j+s)}{\frac{1}{t^2} + \frac{1}{s^2}}$$

  where $s = s_1$ if $|s_1| < |s_2|$, otherwise $s = s_2$.

- For Figure 4d, the equation is

$$f = f(i, j+s)$$

  where s is chosen as in the term before.

- For Figure 4e, the equation is

$$f = \frac{\frac{1}{t^2} f(i+t,j) + \frac{1}{s^2} f(i,j+s)}{\frac{1}{t^2} + \frac{1}{s^2}}$$

  where s and t are chosen between the entry from $\{s_1, s_2\}$ and $\{t_1, t_2\}$ which are smaller in absolute value.

Once values for both the signed distance and the extension function are established at *Close* points, we need only update extension values. As the distance value is updated using the Fast Marching Method, a new extension value is chosen such that $\nabla F_{ext} \cdot \nabla \phi^{temp} = 0$, where the gradient of F_{ext} and ϕ^{temp} are calculated using the points that contributed in the update of ϕ. If no points from a grid direction are used, the corresponding component of the gradient is zero.

As an example, consider the case shown in Figure 4b. Here the new distance value at (i,j) is found by solving Eqn. 7. Assuming that $(i+1,j)$ and $(i,j-1)$ are the points that are used in updating the distance, if v is the new extension value, it then has to satisfy an upwind version of Eqn. 5, namely

$$\left(\frac{\phi^{temp}_{i+1,j} - \phi^{temp}_{i,j}}{h}, \frac{\phi^{temp}_{i,j} - \phi^{temp}_{i,j-1}}{h} \right) \cdot \left(\frac{F_{i+1,j} - v}{h}, \frac{v - F_{i,j-1}}{h} \right) = 0.$$

Since $(i+1,j)$ and $(i,j-1)$ have been accepted, F is defined at those points, and this equation can be solved with respect to v to produce

$$v = \frac{F_{i+1,j}(\phi^{temp}_{i,j} - \phi^{temp}_{i+1,j}) + F_{i,j-1}(\phi^{temp}_{i,j} - \phi^{temp}_{i,j-1})}{(\phi^{temp}_{i,j} - \phi^{temp}_{i+1,j}) + (\phi^{temp}_{i,j} - \phi^{temp}_{i,j-1})}$$

This means that, at the end, $\nabla F_{ext} \cdot \nabla \phi^{temp} = 0$, is satisfied for all points on the grid, except the points along the front itself. At those points, the previous construction will make the equation satisfied when the gradient approximation is computed using points on the front as mentioned before.

Finally, we point out that our technique allows one to extend either the normal speed function F or an advective component velocity field (u, v); this will be done in a later section.

5 Numerical Tests

In this section, we perform a series of numerical tests to analyze this extension construction. These tests are design to analyze the following:

- How much time does it take to construct these extension velocities?
- How well do extension velocities constructed by our method maintain the signed distance function?
- Are there any special difficulties in three dimensions, in executing topological change, or in extending speed laws depending on second order effects (such as curvature motion)?
- What is the advantage of using this technique in problems with sub-grid effects?
- Does the algorithm allow adequate construction of off-front speeds in the case where there is no natural extension and the front motion is highly complex, such as those that result from visibility issues, angle-dependent ion-milling flux functions, etc.)?

By way of description, we call the technique in this paper the "extension" method, while defining the "no extension" method to be the one which one uses a velocity defined and available at each particular level set. Also, we shall distinguish between the Narrow Band method which limits computation to the Narrow Band vs. the full matrix method, which operates on the entire computational domain. We now proceed to provide data to answer these questions.

5.1 Timings

We begin with a straightforward accuracy check (we shall return to a more precise check in the next section). We begin with a pure advection problem of translating a circle with velocity $(u, v) = (2, 3)$. The circle has radius 10, centered at $(20, 20)$ in a box $[0, 60], [0, 60]$ (on a $61x61$ grid). While somewhat unnatural, we recast this problem as one of front moving normal to itself with speed in the normal direction given by $F = \vec{n} \cdot (2, 3)$, where $\vec{n}$ is the normal to the front. We note that at each grid point we need the normal speed of the front, and use a second order upwind scheme to advect the front. For the "no-extension" method, we calculate the normal at every grid point by evaluating $\nabla\phi/|\nabla\phi|$, and the normal speed is then the inner product of the normal and the vector $(2,3)$. For the "extension" method, we begin by finding the points where the front intersects the grid lines. We then calculate the normal at those points using the ϕ function, and take the inner product with $(2,3)$ to find the speed of the front at those points. We then extend the speed onto the grid points.

We first compare the accuracy of the two methods. The two different methods are in close agreement, even for a relatively coarse grid. The differences are so small that comparing the areas of the circles is not a useful measure. In Figure 5 we show the level of agreement by zooming in on a portion of the circle and comparing the two computed solutions to the analytical solution. The numerical solutions almost overlap completely; the accuracy difference between extending or not extending the speed function is far less significant than the numerical error inherent in the space and time discretization of the derivative operators.

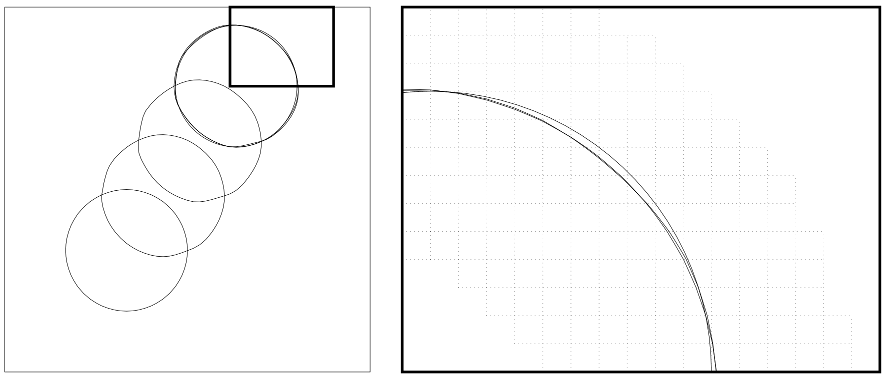


Figure 5: Constructing Extension Velocities

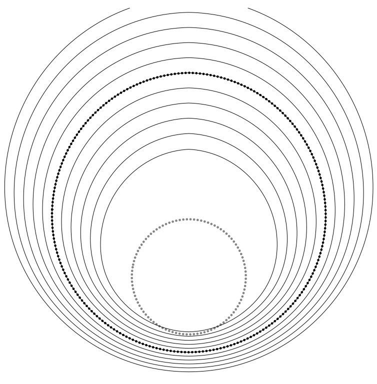
Table 1 compares the execution speeds. We show the total execution time it takes to advect the circle between two corners of the square. We vary the grid size, and compare the execution times both for the narrow band approach and the full level set method. These runs are on Sun Sparc II.

	Narrow band		Full method	
Grid Size	Extension	No extension	Extension	No extension
60	less than 1 sec	less than 1 sec	3s	less than 1 sec
120	5s	2s	22s	12s
240	30	9	180	96
600	450	94	3500	1600

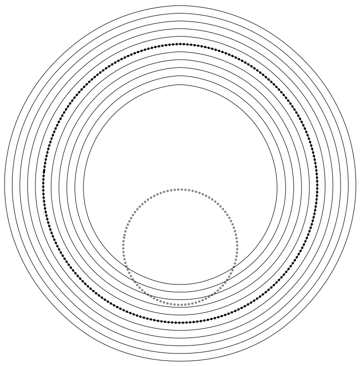
Table 1: Simple advection with velocity $(2,3)$.

These results show that the execution time for the extension problem is roughly three to four times slower than the direct method. It has worse scaling, which is to be expected as the cost of extending the speed to M grid points is $O(M\log(M))$. However, we note an important fact; it costs almost nothing to evaluate the speed function in this example. In problems where calculation of the speed function is **itself** time–consuming (for example, in the evaluation of a boundary integral to determine the speed function), minimizing the number of points at which this must be performed leads to significant speedup. As soon as the cost of computing the speed at the grid points is more than $O(\log(M))$, the extension technique becomes competitive, and in fact can be faster than the direct method.

The above was designed to show a simulation in which the motion is obvious. A more detailed timing run is as follows. We use a circle with center $(10, 6)$, radius 3, in a box with bounding coordinates $[0, 20], [0, 20]$. The speed function is chosen as $F(x, y) = a * y + b$, where a and b are chosen so that $F(x, 2) = 1.0$ and $F(x, 10) = 10.0$. Thus, the motion is one which elongates the circle as propagates outwards. The calculation is performed on a $101x101$ grid. In Figure 6, we show the results of this calculation. On the left, we show the initial front, together with the front at time $T = 0.633$ and the other level sets. The initial front is shown as a dashed line, while the front at a later time is shown as a dark curve. This calculation is performed without using any extension velocities; the speed F is taken directly from the analytic function given above. On the right, the same calculation is repeated using the extension velocity calculated using our algorithm. The figure on the left shows distortion of the neighboring level sets, while the one calculated using our extension velocity algorithm maintains nicely the signed distance function, as seen by the evenly spaced level set function lines.



No extension



Using extension velocity

Figure 6: Motion under elongation speed function: the small dark circle in each figure is the initial front. The larger dark circle and the neighboring level curves are computed at time $t = .633$.

In the previous example, we provided timing results for the extension method vs. the non-extension method.; Here, we provide detailed timing measurements and show how the time required to build our extension velocity depends on the number of grid points where one wants values. In Figure 7, we show timings on a 350MHz - 604e - PowerPC processor. As the grid is refined, there are more points on the front, and hence the extension takes longer. We show results for various widths of the narrow band, and then finally for the "full" method in which we construct extension velocities at all grid points in the full computational domain. As expected, the timings go up as the number of grid points increase; the increase is essentially linear, since the $\log N$ term is almost one for these values.

	$100x100$ grid		$300x300$ grid	
Tube Width	# Points in Domain	Time	# Points in Domain	Time
7 cells	1500	8.8ms	4740	52ms
11 cells	2340	11.7ms	7660	66ms
15 cells	3124	14.8ms	10604	80ms
21 cells	4144	18.5ms	14780	100ms
Full Domain	10000	39ms	90000	475ms

Figure 7: Timings table for elongation flow

5.2 Maintaining the Signed Distance Function

Next, we verify that our algorithm constructs an extension velocity which maintains the signed distance function, assuming one starts from initial data which itself has the property that $|\nabla\phi(x, y, t = 0)| = 1$. We consider two separate problems, each designed to cause significant shearing, stretching, and compression of the neighboring level set functions.

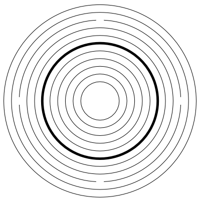
First, we consider a speed function of the form

$$F = (R - (3.0 + 2. * t)) * \sin(4\theta) + 1.$$

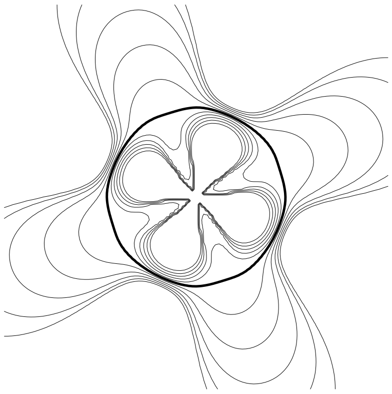
where $R = (x^2 + y^2)^{1/2}$, θ is the angle made between the vector (x, y) and the positive x axis, and t is the time. Here, the initial circle is centered at the origin with radius 3. The calculation is performed on a $101x101$ grid. We show the initial results (Fig. 8a), the results at time $t = 0.8$ using the given velocity field ("no extension") (Fig. 8b), and the results using our extension velocity in (Fig. 8c). The use of the given velocity causes the level sets neighboring the zero level set (shown as a dark line) to wildly diverge, while using the extension velocity causes the level sets to remain equally spaced and conforming with the zero level set. The exact solution to this problem can be seen from the velocity field is a circle with radius given at time t by $(3 + 2t)$, which matches our computed solution.

We now repeat this calculation for another example, and provide a quantitative measurement for the error involved. This time, we use a velocity field of the form

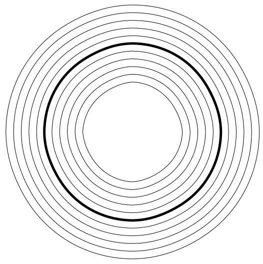
$$F = ((R - 3)^2 + 1) * (2 + sin(4\theta)).$$



Initial Front
Fig. 8a



Level Sets using Given F
Fig. 8b



Level Sets using Extension Velocity
Fig. 8c

Figure 8: Effect of No-extension and Extension Velocities on Neighboring Level Sets: there is no reinitialization of the level set function at any time.

The calculation is performed on a $401x401$ grid. We start with the same initial data as given in Fig. 9a. This velocity field produces significant distortions in the front. In Figure 9, we show the front at time $t = .45$ and $t = .90$ both without the extension and with our extension velocity. It is clear that the extension velocity preserves the initial property of the signed distance velocity, namely $|\nabla\phi| = 1$, as can be seen by the equal spacing of the neighboring level sets.

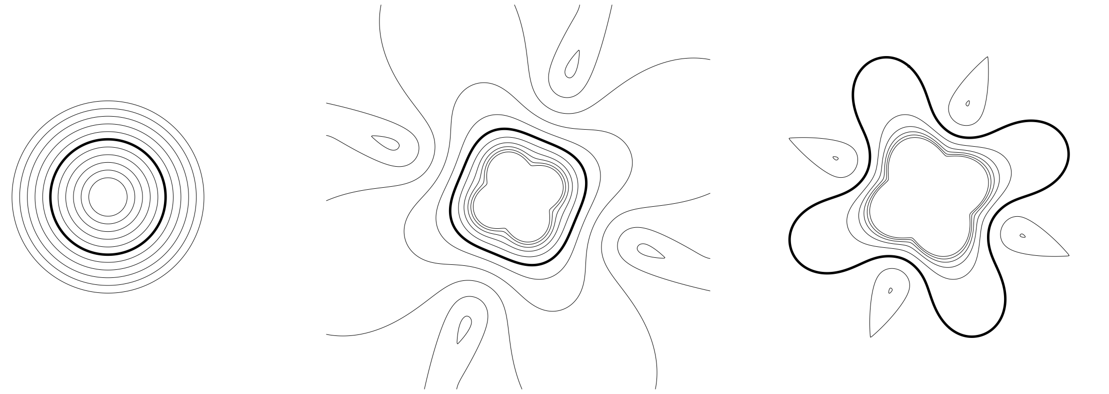


Fig. 9a: Initial shape   Fig. 9b: No Extension, t=.45   Fig. 9c: No Extension, t=.90

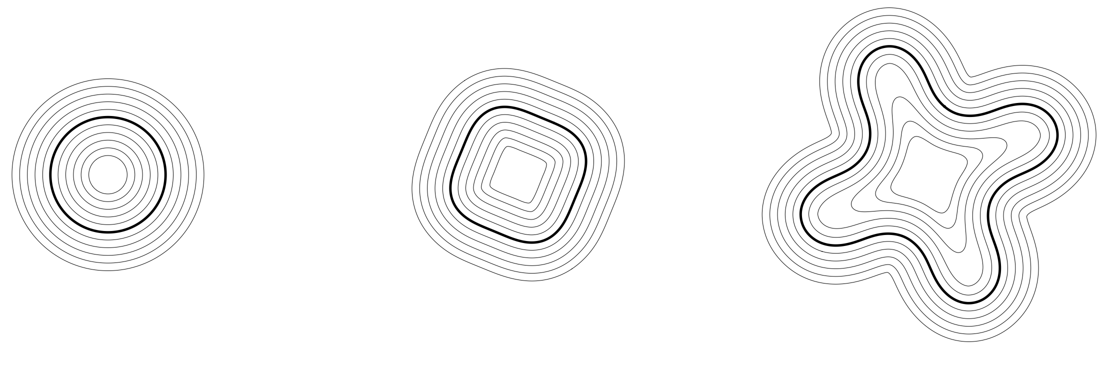


Fig. 9d: Initial shape   Fig. 9e: Extension, t=.45   Fig. 9f: Extension, t=.90

Figure 9: Effect of No-extension and Extension Velocities on Neighboring Level Sets: there is no reinitialization of the level set function at any time.

Once again, the equi-spacing of the contour lines shows that in the extension case, the signed distance function has been maintained.

Next, we analyze the error as a function of mesh size in the computed solution. We do this as follows. We assume that the finest calculation on a $800x800$ grid of the extension velocity is the correct reference solution (for both a first and second order method), and compute the signed distance function from that zero level set at time $t = .90$. We do this as follows. We take the points on the reference path and for each point calculate the minimum distance to the path we want to compare it to. This gives an array of distances which are then used for the norm evaluations for the L_∞, L_1 and L_2 error. The calculation is made on a physical box of size $(20x20)$. The results are shown in Figure 10 for a first and second order spatial method. As expected, the error in the computed solution for the interface using the extension velocity is considerably better than that computed using the no-extension solution.

	First Order						Second Order					
	No extension			Extension			No extension			Extension		
Grid	L_∞	L_1	L_2	L_∞	L_1	L_2	L_∞	L_1	L_2	L_∞	L_1	L_2
50^2	.00818	.00356	.00418	.00732	.00329	.00402	.00365	.00138	.00163	.00279	.00107	.00138
100^2	.00434	.00185	.00219	.00327	.00136	.00171	.00103	.00042	.00049	.00075	.00027	.00036
200^2	0.00211	.00090	.00111	.00137	.00056	.00071	.00039	.00016	.00019	.00019	.00006	.00009
400^2	.0011	.00047	.00059	.00045	.00018	.00023	.00025	.00008	.00011	.00004	0.00001	.00002

Figure 10: Error analysis for No-extension and Extension Calculations

5.3 Three Dimensions and Topological Change

Next, we show how to apply the extension techniques to three-dimensional problems undergoing topological change. A good prototype problem is the collapse of a dumbbell under mean curvature; as shown in [15], level set methods naturally track the topological change that occurs at the narrowing of the handle. Here, we extend the mean curvature-based speed function from the interface itself using the Fast Marching Technique to the neighboring grid points, and then update the level set function. The results are shown in Figure 11. Similar to the above example, we built a contour plotter to construct nodal points at the triangular representation of the zero level set, evaluated the curvature at these nodal points (by bilinear interpolation from the mean curvature evaluated at the mesh grid points using standard level set methodology), and then extended this curvature field to the narrow band. We stress that, in the case of such a geometric speed law, it is far more natural to construct a speed function at each grid point directly from the level set passing through that point, rather than develop the extension construction. However, we include this example to demonstrate the applicability of our technique to sensitive three-dimensional problems involving higher order derivatives (such as curvature), which can often be required in complex examples such as three-dimensional etching/deposition simulations which include surface diffusion, visibility and re-emission/re-deposition.

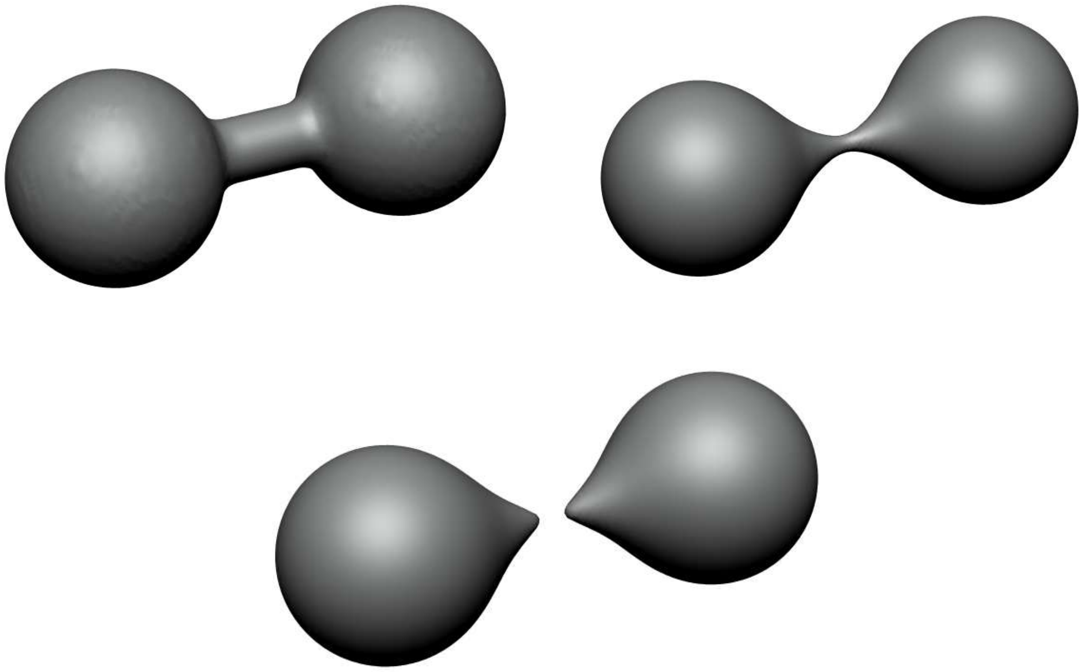


Figure 11: Extension Velocity Solution of Collapse of Dumbbell under Mean Curvature

5.4 Subgrid accuracy

Next, we study sub-grid accuracy. There is a yet another significant class of problems in which there is substantial benefit in using the speed extension rather than the natural speed at the grid points. As illustration, consider a problem in which there is a dramatic range of speed values within a narrow spatial range. We take a domain in which the speed in the normal direction is 1, except in a thin annulus of width two grid cells where the speed is 100. As a practical motivation one might consider an etching problem in which a very thin band of easily etched materials lies embedded within a material with much slower etch rate. We start with an initial front that intersects the circle, and advect the front according to the given speed function. We note the difference between the extension and the direct approach:

- **Extension**: The speed at any point on the front can be calculated precisely by checking if it is inside this annulus or not. This speed is then extended onto the surrounding grid points. This leads a more accurate detection of the front location and the accompanying etch rate; the existence of the thin fast etch region is noted as the front passes through.

- **Direct (No Extension)**: Here, speeds are defined only in terms of the value of the etch function at each grid point. Thus, if grid points near the annulus are used, they may provide speed values which do not accurately reflect the local etch rate at the interface itself.

The results of the simulations for a 30 by 30 grid are shown in Figure 12. On the left, the direct solution method is shown; on the right, the extension technique is used. The underlying grid is indicated by the dotted lines. On the left side, considerable jaggedness occurs, since the effective speed of the front at each point will be an average of the speed at the surrounding grid points. A much smoother and correct evolving profile is obtained using the extension technique shown in the figures on the right. We note that merger is handled easily; as the two fronts come close together the construction of extension velocities works well, and the merger occurs as expected.

We also note that in this example, we choose to extend the individual velocity components, rather than their projection in the normal direction. This may be easily done using our methodology.

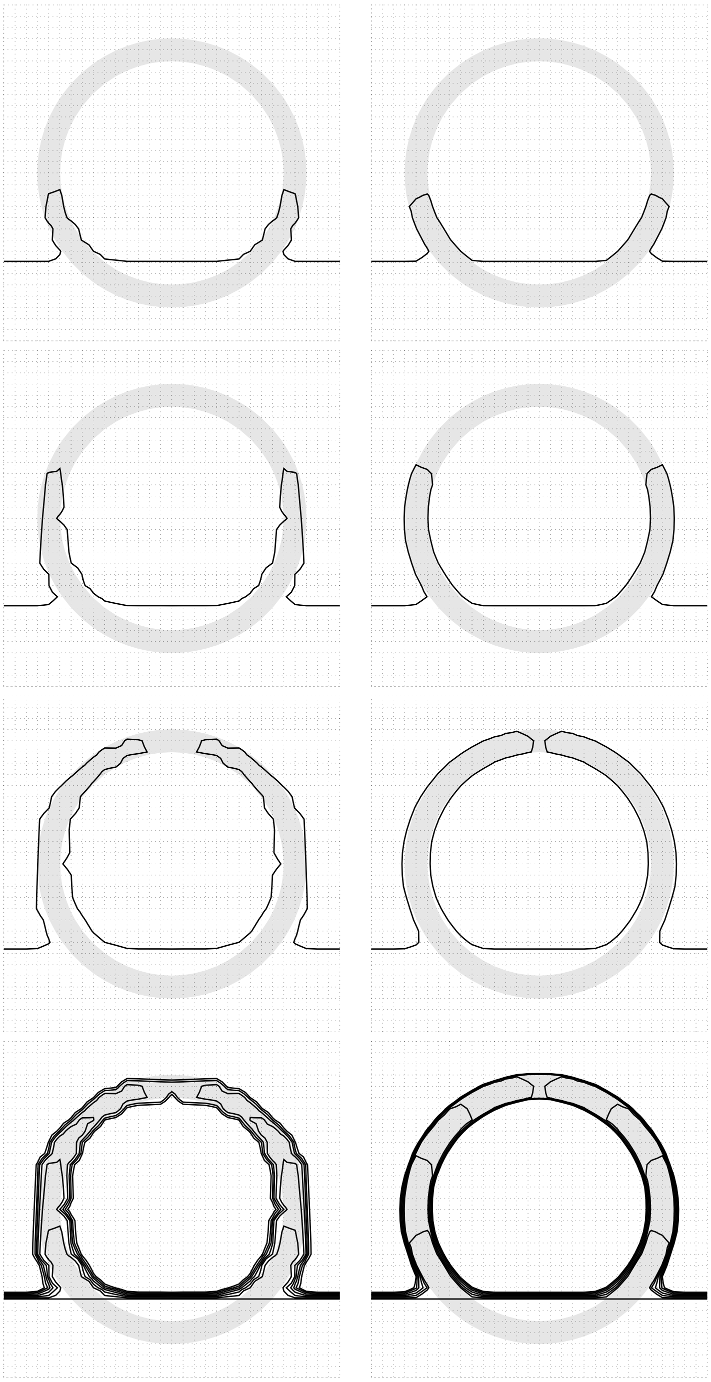


Figure 12: Left=No Extension. Right=Extension: Beginning

5.5 Complex Problems without Natural Speed Definitions

Finally, we apply these extension techniques to problems in which there is no natural definition of the speed off of the evolving interface, and extension is necessary in order to build an appropriate velocity embedding.

As illustration, we consider aspects of etching and deposition in semi–conductor manufacturing, see [4] for further discussion. Briefly, a typical problem involves a front which is under bombardment from particles, some of which stick and the rest bounce off the front and hit other parts of the surface. These re-emitted particles again have a certain probability of sticking; the fraction β of particles that attach themselves during any particular collision is known as the sticking coefficient. If I^0 is the material strength that arrives at the front, the strength at which material sticks, I_S, is given by the following integral equation

$$I_S(x) = \beta I^0(x) + (1-\beta)\int Flux(x,y)I_S(y)M(x,y)dy,$$

see [4], where, $Flux$ depends on the distance between x and y, and the mask function $M(x,y)$ will be 0 or 1 depending on whether x and y are visible from one another. Discretizing this equation leads to the matrix equation

$$I_S = \beta I^0 + (1-\beta)\Omega I_S,$$

where I_S and I^0 are the total and incoming strength at points on the front, and Ω is a square matrix.

Our exact technique to construct these extension velocities is as follows. First, in either two or three space dimensions, we use a contour finder to find points where the front corresponding to the zero level set intersects grid lines. This produces a set of nodal points, each representing an area (either a line segment in two dimensions or a triangular element in three dimensions). The values of the flux at each of these nodal points are unknowns in a matrix version of the above integral equation. This matrix equation is solved through an iterative technique with an accompanying recursion relation, see [4], which provides the velocity at each nodal point. We then use these velocities as the basis for starting the Fast Marching Method, and then construct extension velocities in the rest of the Narrow Band.

It is awkward to find a natural meaning for this velocity expression at each level set, since solving the front is both time–consuming and physically meaningless. Since the majority of the time is spend setting up and solving the matrix equation; the time it takes to extend the speed to neighboring grid points using the Fast Marching Method is of little consequence.

In Figure 13 the incoming light distribution is proportional to $\cos^4(\theta)$ where θ is the angle from vertical. The sticking coefficient is 0.5, and the grid was 160 by 100. The execution time was about one minute. The deposition on top of the cave like structure is due solely to the effects of the redeposition.

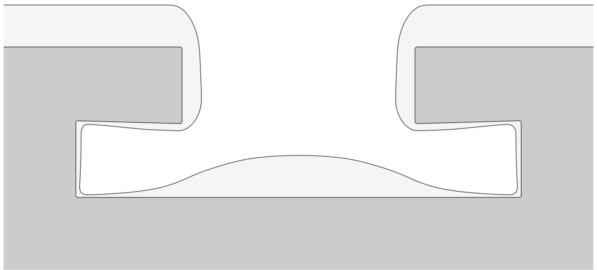


Figure 13: Upwind construction of Accepted Values

5.6 Summary

We have described an algorithm based on the Fast Marching Method for constructing extension velocities for use in level set calculations. The approach constructs an extension velocity in $O(N \log N)$ time, where N is the total number of points where one wants the extension. This extension velocity moves the zero level set with a velocity which is given on the front, preserves the signed distance function without need for re-initialization, and provides sub-grid accuracy in some certain cases.

Acknowledgments: All calculations were performed at the University of California at Berkeley and the Lawrence Berkeley Laboratory. We would like to thank D. Chopp for helpful discussions.

References

[1] Adalsteinsson, D., and Sethian, J.A., *A fast level set method for propagating interfaces*, Jour. Comp. Phys., Vol. 118, pp. 269–277, 1995.

[2] Adalsteinsson, D., and Sethian, J.A., *A Level Set Approach to a Unified Model for Etching, Deposition, and Lithography I: Two–Dimensional Simulations*, Vol. 120, No. 1, pp. 128–144, 1995.

[3] Adalsteinsson, D., and Sethian, J.A., *A Level Set Approach to a Unified Model for Etching, Deposition, and Lithography II: Three–Dimensional Simulations*, Vol. 122, No. 2, pp. 348–366, 1995.

[4] Adalsteinsson, D., and Sethian, J.A., *A Level Set Approach to a Unified Model for Etching, Deposition, and Lithography III: Re–deposition, Re–emission, Surface Diffusion, and Complex Simulations*, 138, 1, pp. 193-223, 1997.

[5] Barth, T.J., and Sethian, J.A., *Numerical Schemes for the Hamilton-Jacobi and Level Set Equations on Triangulated Domains*, to appear, J. Comp. Physics, August, 1998.

[6] Chang, Y.C., Hou, T.Y., Merriman, B., and Osher, S.J., *A Level Set Formulation of Eulerian Interface Capturing Methods for Incompressible Fluid Flows*, Jour. Comp. Phys., 124, pp. 449-464, 1996.

[7] Chen, S., Merriman, B., Osher, S., and Smereka, P., *A Simple Level Set Method for Solving Stefan Problems*, J. Comp. Phys., 138, pp. 8–29, 1997.

[8] Chopp, D. L., *Computing minimal surfaces via level set curvature flow*, Journal of Computational Physics, Vol. 106, pp. 77–91, 1993.

[9] Malladi, R., Sethian, J. A., and Vemuri, B. C., *Shape modeling with front propagation: A level set approach*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 17, No. 2., 1995.

[10] Osher, S., and Sethian, J. A., *Fronts propagating with curvature dependent speed: Algorithms based on Hamilton–Jacobi formulation*, Jour. Comp. Phys., Vol. 79, pp. 12–49, 1988.

[11] Osher, S., and Shu, C., *High-Order Nonoscillatory Schemes for Hamilton–Jacobi Equations*, Jour. Comp. Phys., 28, pp. 907–922, 1991.

[12] Rouy, E., and Tourin, A., *A Viscosity Solutions Approach to Shape–from–shading*, SIAM. J. Numer. Anal., 29(3):867–884, 1992.

[13] Sethian, J.A., *An Analysis of Flame Propagation*, Ph.D. Dissertation, Mathematics, University of California, Berkeley, 1982.

[14] Sethian, J.A., *Curvature and the evolution of fronts*, Commun. in Math. Physics, Vol. 101, pp. 487–499, 1985.

[15] Sethian, J.A., *Numerical algorithms for propagating interfaces: Hamilton–Jacobi equations and conservation laws*, Jour. of Diff. Geom., Vol. 31, pp. 131–161, 1990.

[16] Sethian, J.A., *Numerical methods for propagating fronts*, in Variational methods for free surface interfaces, edited by P. Concus and R. Finn, (Springer–Verlag, New Work, 1987).

[17] Sethian, J.A., *A Fast Marching Level Set Method for Monotonically Advancing Fronts*, Proc. Nat. Acad. Sci., 93, 4, 1996.

[18] Sethian, J.A., *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Material Science*, Cambridge University Press, 1996.

[19] Sethian, J.A. and Strain, J.D., *Crystal Growth and Dendritic Solidification* J. Comp. Phys., Vol. 98, pp. 231–253, (1992).

[20] Sussman, M., Smereka, P. and Osher, S.J., *A Level Set Method for Computing Solutions to Incompressible Two–Phase Flow*, J. Comp. Phys. 114, pp. 146–159, 1994.

[21] Zhao, H-K., Chan, T., Merriman, B., and Osher, S., *A Variational Level Set Approach to Multiphase Motion*, Jour. Comp. Phys., 127, pp. 179–195, (1996).