Fast Marching Methods

J.A. Sethian *

Dept. of Mathematics
University of California, Berkeley 94720

November 11, 1998

Abstract

Fast Marching Methods are numerical schemes for computing solutions to the non-linear Eikonal equation and related static Hamilton-Jacobi equations. Based on entropy-satisfying upwind schemes and fast sorting techniques, they yield consistent, accurate, and highly efficient algorithms. They are optimal in the sense that the computational complexity of the algorithms is $O(N \log N)$, where N is the total number of points in the domain. The schemes are of use in a variety of applications, including problems in shape offsetting, computing distances from complex curves and surfaces, shape-from-shading, photolithographic development, computing first arrivals in seismic travel times, construction of shortest geodesics on surfaces, optimal path planning around obstacles, and visibility and reflection calculations. In this paper, we review the development of these techniques, including the theoretical and numerical underpinnings, provide details of the computational schemes including higher order versions, and demonstrate the techniques in a collection of different areas.

*This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract Number DE-AC03-76SF00098, and the Office of Naval Research under under grant FDN00014-96-1-0381.

1 Introduction to Fast Marching Methods

Fast Marching Methods are computational techniques which approximate the solution to non-linear Eikonal equations of the form

$$|\nabla u(x)| = F(x) \text{ in } \Omega, \qquad F(x) > 0 \tag{1}$$

$$u = g(x) \text{ on } \Gamma,$$

where Ω is a domain in R^2 or R^3.[1] Here, the right-hand side $F(x) > 0$ is typically supplied as known input to the equation, as is the boundary condition that u equal a known function $g(x)$ given along a prescribed curve or surface Γ in Ω.

Equation 1 is part of a broader class of Hamilton-Jacobi equations of the form

$$H(u_x, u_y, u_z, x, y, z) = 0. \tag{2}$$

In the case of the Eikonal equation, the function H reduces to $H = |\nabla u(x)| - F(x)$.

As discussed below, one of the main difficulties in solving these equations is that the solution need not be differentiable, even with smooth boundary data. This non-differentiability is intimately connected to the notion of appropriate weak solutions. Our goal is to devise numerical techniques which naturally account for this non-differentiability by constructing accurate and efficient approximation schemes that admit physically correct non-smooth solutions. The main techniques we will use are Fast Marching Methods, introduced in [27]. These consistent and highly efficient techniques are based on two key components. First, by exploiting upwind "viscosity schemes", they automatically select solutions which include non-differentiability in natural ways. Second, by coupling the causality of these schemes to fast sorting methods borrowed from discrete network problems, they become extremely efficient computationally; the complexity of these algorithms is $O(N \log N)$, where N is the total number of points in the domain Ω.

There is a large collection of applications that require the solution of the non-linear Eikonal equation. To begin, imagine the problem of accurately computing the distance to a curve or surface. By letting the right-hand side F equal unity, the level curves $u = C$ of the Eikonal equation give the set of all points located a distance C from the boundary curve; this can be seen

[1] The theory and Fast Marching Methods algorithms presented in this paper hold for R^n; for notational reasons, we focus on R^2 and R^3.

by noting that the gradient $|\nabla u|$ must be orthogonal to these level curves, and that the gradient has length one. Another way to state this problem is to imagine a disturbance propagating with unit speed away from the initial curve, and then compute the "first arrival time" at each point in the domain Ω. Points where the solution is non-differentiable are locations where two points on the boundary curve are the same distance away. Performing this same calculation on surfaces and tracking backwards orthogonally to equi-arrival curves allows one to compute shortest paths on manifolds.

Extending the problem by changing the right-hand side to a non-constant F corresponds to computing the distance function in a non-uniform metric; here the "arrival times" are slowed or sped-up by the variation in F. Two examples include the calculation of first arrivals in seismic travel times, in which the "slowness" function F corresponds to reciprocal of the velocity that varies depending on the type of rock, and the development process in photolithography, in which the resistive strength of a material is differentially altered through optical processes and the material is then exposed to an etching beam which removes the weaker material.

More drastic alterations of the right-hand side (that is, setting $F = \infty$ in certain subsets of Ω) yield equations for optimal path navigation, in which an infinite value for F corresponds to an impenetrable obstacle which must be circumnavigated. Extending the dimension from pure physical space (from R^2 or R^3) to include rotational effects allows one to include additional degrees of freedom generated by moving arms in the navigation process.

The outline of this paper is as follows. We begin by discussing the ideas of weak solutions and viscosity solutions, followed by numerical approximations and upwind schemes. We then present the Fast Marching Method, first in orthogonal coordinate systems, and then in a triangulated unstructured mesh setting. Here, we discuss in some detail the connection of these techniques to both network path algorithms and level set methods. After extending the Fast Marching Method to higher order, we end with a collection of applications.

Before we start, we point out a slightly confusing issue which is at the heart of the efficiency of Fast Marching Methods. The non-linear Eikonal equation (Eqn. 1) and static Hamilton-Jacobi equation (Eqn. 2) are *boundary value problems*, however the notion of "first arrival times" sounds more like an *initial value problem*, in which information propagates outwards from the boundary/initial data. In fact, while the equations are true boundary value problems, the success of Fast Marching Methods comes from building numerical schemes that allow one to efficiently construct the solution outward from the boundary data.

The majority of this review is taken from the original work on Fast Marching Methods [27], and two recent books on the subject [30, 31]. We refer the interested reader to these resources for many computational schemes which can exploit Fast Marching Methods, as well as many more applications and examples.

2 The Eikonal and Static Hamilton-Jacobi Equations

2.1 Fundamental equations and non-differentiability

We begin by returning to the example of the simple Eikonal equation

$$|\nabla u| = F(x,y),$$

with $F = 1$. One of the subtleties of this equation is that the solution may be non-differentiable, even with smooth initial data. As a simple illustration, consider the case of

$$|\nabla u| = 1 \text{ outside the unit circle} \qquad u = 0 \text{ on the unit circle.} \tag{3}$$

It is readily checked that the function $u(x,y) = (x^2+y^2)^{1/2} - 1$ corresponding to the distance function from the unit circle solves the given Eikonal equation. Indeed, specifying the right-hand side $F(x) = 1$ and providing zero as the boundary condition implies that the solution is just the distance to the initial curve Γ. The solution is shown as a family of concentric circles in Figure 1. It is, of course, easy to check that the solution is everywhere differentiable.

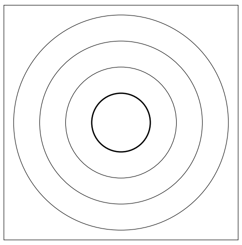
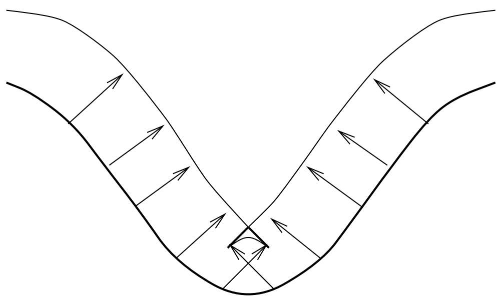


Fig. 1

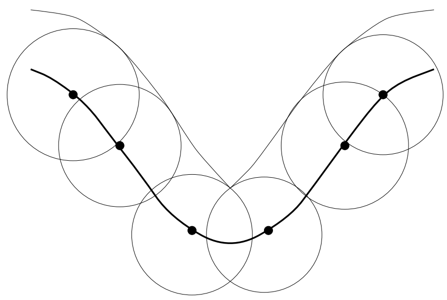
Figure 1: Distance function solution to Eikonal equation $|\nabla u| = 1$

In contrast, now consider boundary data given by a non-convex curve, and suppose we try to find the distance from any point in the domain to this curve. There are two ways we can proceed.

First, suppose we put a particle at each point of the boundary curve and move these particles away from the curve in a normal direction with unit speed. At any time C, the position of these particles gives a set of all points a "distance" C away from the boundary curve. Figure 2a illustrates this approach above a particular non-convex curve.



Points a "local distance" from the Boundary Data
Fig. $2a$



Globally Closest Points to Boundary Data
Fig. $2b$

Figure 2: Possible solutions to Eikonal equation $|\nabla u| = 1$

We note that this construction creates a curve which crosses itself. This "swallowtail" solution is multi-valued; some points are reached by normals emanating from more than one point on the boundary.

A different approach comes from drawing the set of all points located a given distance away. One way to build this solution (shown on the right) is through a Huygens' principle construction. The solution is developed by imagining wavefronts emanating with unit speed from each point of the boundary data. The envelope of these wave fronts always corresponds to the "first arrivals" and automatically produces the solution given in Figure 2b. We note that there will be a vertical ridge along which two points on the boundary curve are the same distance away, as suggested in Figure 2b. Along this ridge, the solution is non-differentiable and the gradient ∇u is not defined.

Both of the above constructions can be thought of as solutions to the Eikonal equation. However, the solution that we want, corresponding to the shortest distance or "first arrival", is the one obtained through the Huygens construction. Another way to obtain this solution is through the notion of an entropy condition. As defined in [23, 24], we imagine the boundary curve as a source for a propagating flame, and the expanding flame satisfies the requirement that once a point in the domain is ignited by the expanding front, it stays burnt. This construction yields the entropy-satisfying Huygens construction given in Fig. 2b.

Yet another way to obtain this entropy-satisfying Huygens construction comes from adding a smoothing term to the equation. Consider the associated "viscous" partial differential equation given by

$$|\nabla u(x)| = F(x) + \epsilon \nabla^2 u \tag{4}$$

It can be shown that the viscous term $\epsilon \nabla^2 u$ acts to smooth out sharp corners in the solution, and guarantees that the solution stays smooth in the entire domain Ω (see [24, 30]. As ϵ goes to zero, the solution converges to the first arrival solution given in Figure 2b.

Thus, our goal is to build numerical methods that automatically extract this viscous limit. Before doing so, we note that the formal way of defining this viscous limit is through the idea of viscosity solutions for time-dependent Hamilton-Jacobi equations. Rather than define the solution as the viscous limit, one instead analyzes the behavior of potential solutions when measured against possible test functions. Crandall and Lions [8] have developed the theory of viscosity solutions for time-dependent Hamilton-Jacobi equation. Briefly, following the definitions in [8], they define a viscosity solution as follows.

Definition: A function u is said to be a *viscosity solution* of Eqn. (2), if, for all smooth test functions v,

1. if $u - v$ has a local maximum at a point (x_o, t_o), then

$$v_t(x_o, t_o) + H(Dv(x_o, t_o), x_o) \leq 0 \tag{5}$$

2. if $u - v$ has a local minimum at a point (x_o, t_o), then

$$v_t(x_o, t_o) + H(Dv(x_o, t_o), x_o) \geq 0. \tag{6}$$

Note that nowhere in this definition is the viscosity solution u differentiated; everything is done in terms of the test function v. This is done so that one can now use the usual trick of integration by parts and move all the derivatives onto the test function in exchange for some boundary conditions.

Using this definition, one can show (i) that if u is a smooth solution of the Hamilton–Jacobi equation, then it is a viscosity solution, (ii) that if a viscosity solution u is differentiable at some point, then it satisfies the Hamilton–Jacobi equation at that point, and (iii) that the viscosity solution is unique, given appropriate initial conditions. Finally, one then proves that the solution produced by taking the limit of the smooth solutions u_ϵ as ϵ vanishes is indeed this viscosity solution.

We shall prove none of these statements here. Precise statements and proofs may be found in [6, 8, 7]. The salient point is that for both time-dependent and static (e.g., the Eikonal equation) Hamilton-Jacobi equations, the viscosity solution can be defined in a way that does not require differentiation, and can be proven to be the unique viscous limit of the smoothed Hamilton–Jacobi equation.

Our goal is to develop numerical approximations which correctly select this viscous limit. As proposed in [25], since the entropy condition is similar to the one for hyperbolic conservation laws, it suggests using the numerical methodologies associated with hyperbolic equation.

2.2 Upwind schemes and numerical approximations

2.2.1 Upwind schemes and numerical quadrature

As motivation for the use of upwind schemes for approximating the gradient operator, consider the one-dimensional Eikonal equation given by

$$\sqrt{u_x^2} = F(x) \qquad\qquad u(0) = 0.$$

Here, the right-hand side $F(x) > 0$ is given, and the goal is to construct $u(x)$ away from the boundary condition that $u(0) = 0$. We note immediately that the solution to this problem is not unique; if $v(x)$ solves the problem, then so does $-v(x)$. Hence, we further restrict ourselves to non-negative solutions u.

We can imagine building the solution "outwards" along the positive and negative x-axis from the origin by solving each problem separately. We consider the ordinary differential equations

$$\begin{array}{lll} \frac{du}{dx} = F(x) & \qquad u(0) = 0 & \quad x \geq 0 \\[2ex] \frac{du}{dx} = -F(x) & \qquad u(0) = 0 & \quad x \leq 0. \end{array}$$

Since the right-hand side is only a function of x, we are essentially performing numerical quadrature. Using the standard finite difference notion that $u_i \approx$

$u(i\Delta x)$, and $F_i = F(i\Delta x)$, we can approximate each of these two solutions using Euler's method, namely

$$\begin{aligned} \tfrac{u_{i+1}-u_i}{\Delta x} &= \ \ F_i \quad i > 0 \\ \tfrac{u_i - u_{i-1}}{\Delta x} &= -F_i \quad i \leq 0 \end{aligned}$$

where $u_0 = 0$. This is an upwind scheme; we are computing derivatives using points "upwind" or towards the boundary condition. In other words, each ordinary differential equation is solved away from the boundary condition.

2.2.2 Upwind schemes and evolving interfaces

Further motivation for approximating the gradient using upwind differences comes from the instructive example of an curve evolving in time whose position can always be described as the graph of a function. Consider an initial front given by the graph of $g(x)$, with g and g' periodic on $[0, 1]$, and suppose that the front (i) propagates with speed S in its normal direction and (ii) remains a function for all time. Let ψ be the height of the propagating function at time t, thus $\psi(x, 0) = g(x)$. One can show (see [25]) that the equation of motion for the changing height is given by

$$\psi_t = S(1 + \psi_x^2)^{1/2}. \tag{7}$$

As an example, with speed $S = 1$, consider the initial value problem

$$\psi_t = (1 + \psi_x^2)^{1/2}, \qquad \psi(x, 0) = g(x) = \left\{ \begin{array}{ll} 1/2 - x & x \leq 1/2 \\ x - 1/2 & x > 1/2 \end{array} \right\}. \tag{8}$$

The initial front is a "V" formed by rays meeting at $(1/2, 0)$. By the entropy condition and Huygens principle construction, the solution at any time t is the set of all points located a distance t from the initial "V". Our goal is to show that the choice of numerical approximations for the gradient term $(1 + \psi_x^2)^{1/2}$ has some subtlety.

One approach is to divide the interval $[0, 1]$ into $2M - 1$ points, and form the central difference approximation to the spatial derivative ψ_x in Eqn. 8, namely

$$\psi_t \approx \frac{\psi_i^{n+1} - \psi_i^n}{\Delta t} = [1 + [\frac{\psi_{i+1}^n - \psi_{i-1}^n}{2\Delta x}]^2]^{1/2} = [1 + [D_i^{0x}\psi]^2]^{1/2} \tag{9}$$

where in the last expression we have used standard notation for the central difference.

Since $x_M = 1/2$, by symmetry we have that $\psi_{M+1} = \psi_{M-1}$, thus the right-hand side is 1. However, for all $x \neq 1/2$, ψ_t is correctly calculated to be $\sqrt{2}$, since the graph is linear on either side of the corner and thus the central difference approximation is exact. Note that this has nothing to do with the size of the space step Δx or the time step Δt. No matter how small we take the numerical parameters, as long as we use an odd number of points, the approximation to ψ_t at $x = 1/2$ gets no better. It is simply due to the way in which the derivative ψ_x is approximated. In Figure 3 we show results using this scheme, with the time derivative ψ_t replaced by a forward difference scheme.

Exact Solution   Central Differences $\Delta t = .005$ Central Differences $\Delta t = .0005$

Figure 3: Central difference approximation to gradient

It is easy to see what has gone wrong. In the exact solution, $\psi_t = \sqrt{2}$ for all $x \neq 1/2$. This should also hold at $x = 1/2$ where the slope is not defined; the Huygens construction sets $\psi_t(x = 1/2, t)$ equal to $\lim_{x \to 1/2} \psi_t$. Unfortunately, the central difference approximation chooses a different (and, for our purpose, wrong) limiting solution. It sets the undefined slope ψ_x equal to the average of the left and right slopes. As the calculation progresses, this miscalculation of the slope propagates outwards from the spike as wild oscillations. Eventually, these oscillations cause blowup in the code. For details, see [30].

2.3 Schemes for viscosity solutions

Continuing with the example of an evolving interface, we now focus on the gradient term $(1 + {\psi_x}^2)$. Consider now the finite difference approximation introduced in [18], namely

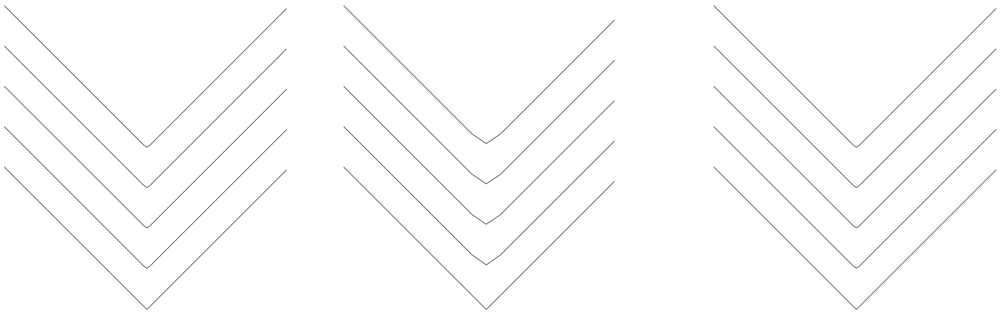
$$ {\psi_x}^2 \approx (\max(D_i^{+x}\psi, 0)^2 + \min(D_i^{-x}\psi, 0)^2) \qquad (10) $$

where again we have used standard finite difference notation that

$$D_i^{-x}\psi = \frac{\psi_i - \psi_{i-1}}{h} \qquad D_i^{+x}\psi = \frac{\psi_{i+1} - \psi_i}{h}. \tag{11}$$

Here, ψ_i is the value of ψ on a grid at the point ih with grid spacing h.

Eqn. 10 is an upwind scheme (see [30]); it chooses grid points in the approximation in terms of the direction of the flow of information. If we consider our propagating "V" curve from the example above, we see that a non-zero value is chosen at the symmetric point. In Figure 4, we show what happens if we use the scheme given in Eqn. 10. The exact answer is shown, together with two simulations. The first uses the entropy-satisfying scheme with only 20 points (Figure 4(b)), the second (Figure 4(c)) with 100 points. In the first approximation, the entropy condition is satisfied, but the corner is somewhat smoothed due to the small number of points used. In the more refined calculation, the corner remains sharp, and the exact solution is very closely approximated. Thus we see that this scheme does a correct job of



(a) Exact Solution (b) Scheme with 20 Points (c) Scheme with 100 Points

Figure 4: Upwind, entropy-satisfying approximations to the gradient

satisfying the entropy condition

It is relatively straightforward to prove that this numerical scheme converges to the correct viscosity solution. Crandall and Lions [8] proved that consistent monotone schemes must converge to the correct viscosity solution; this result parallels similar results for hyperbolic conservation laws (see, for example, [4, 15, 34]). Thus, we need only check that this scheme satisfies the necessary requirements for convergence to viscosity solutions. The operator is easily seen to be consistent, since it uses first order finite difference operators and hence a Taylor series expansion of the error can be shown to go to zero as the time and space step are refined. Proving monotonicity for the most basic first order scheme is also straightforward, and is done by simply checking what happens to monotone data. One also can go back and check that the central difference scheme given earlier does not satisfy these requirements. We refer the interested eader to [8] and [31] for further comment about convergence and additional schemes.

While a vast array of other upwind, entropy-satisfying schemes are available to approximate the gradient, for our purposes, the above approximation (and one small variation) will be sufficient; details on other schemes may be found in [28, 30].

2.4 Approximations to the Eikonal equation

We can now construct appropriate schemes for the Eikonal equation

$$|\nabla u(x, y, z)| = F(x, y, z). \tag{12}$$

Extending these ideas of upwind approximations for the gradient to multiple dimensions, we have the scheme

$$|\nabla u| \approx \left[\begin{array}{c} \max(D^{-x}_{ijk}u, 0)^2 + \min(D^{+x}_{ijk}u, 0)^2 + \\ \max(D^{-y}_{ijk}u, 0)^2 + \min(D^{+y}_{ijk}u, 0)^2 + \\ \max(D^{-z}_{ijk}u, 0)^2 + \min(D^{+z}_{ijk}u, 0)^2 \end{array} \right]^{1/2} = F_{ijk}. \tag{13}$$

The forward and backwards operators D^{-y}, D^{+y}, D^{-z}, and D^{+z} in the other coordinate directions are similar to the one defined earlier for the x direction.

A slightly different upwind scheme due to Godunov, (see [20]), which will turn out to be more convenient, is given by

$$\left[\begin{array}{c} \max(D^{-x}_{ijk}u, -D^{+x}_{ijk}u, 0)^2 + \\ \max(D^{-y}_{ijk}u, -D^{+y}_{ijk}u, 0)^2 + \\ \max(D^{-z}_{ijk}u, -D^{+z}_{ijk}u, 0)^2 \end{array} \right]^{1/2} = F_{ijk}, \tag{14}$$

where we use the same forward and backward operators D^- and D^+ and F_{ijk} is the slowness at the gridpoint ijk.

How might one solve Eqn. 14? One solution, given by Rouy and Tourin in [20], is through iteration. Consider a stencil of a grid point and its six neighbors, as shown in Fig. 5. Observe that Eqn. 14 is a piecewise quadratic equation for u_{ijk}, assuming that the neighboring grid values for u are given. Thus, one solution comes from updating the value of u at each grid point according to this quadratic until a solution is reached:

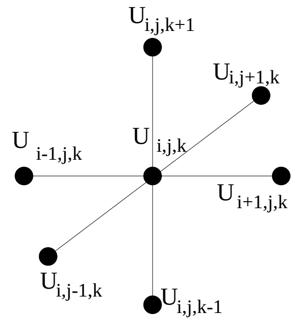


Figure 5: Updating a grid point

For iter=1,n;

For i,j,k=1,Mesh

Solve Quadratic for $u_{ijk}^{\text{iter}+1}$, given

$$u_{i-1,j,k}^{\text{iter}}, u_{i+1,j,k}^{\text{iter}}, u_{i,j-1,k}^{\text{iter}}, u_{i,j+1,k}^{\text{iter}}, u_{i,j,k-1}^{\text{iter}}, u_{i,j,k+1}^{\text{iter}} \tag{15}$$

EndFor

EndFor

If we assume N points in each direction and that it takes roughly N steps to converge, then the operation count for this method is $O(N^4)$.

3 Fast Marching Methods

The key to Fast Marching Methods lies in the observation that the previous iteration contains a very specific causality relationship. In this section, we describe the Fast Marching Method; for details, see [27, 28, 30].

3.1 Causality

The central idea behind the Fast Marching Method is to systematically construct the solution in a "downwind" fashion to produce the solution u. We observe that the upwind difference structure of Equation (14) means that information propagates "one way", that is, from smaller values of u to larger values. Hence, the Fast Marching algorithm rests on "solving" Equation (14) by building the solution outwards from the smallest u value. In fact, this strategy is behind our quadrature view of upwinding described earlier; we can step the solution outwards from the boundary condition in a downwind direction. The algorithm is made fast by confining the "building zone" to a narrow band around the front; this approach is motivated by the narrow band technology introduced by Chopp [3], used in recovering shapes in images by Malladi, Sethian and Vemuri [16], and analyzed extensively by Adalsteinsson and Sethian in [30]. The idea is to sweep the front ahead in a downwind fashion by considering a set of points in 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.

Consider a two-dimensional version of the Eikonal equation, in which the boundary value is known at the origin; this is shown schematically in Figure 6. The black sphere at $u_{0,0}$ signifies a grid point where the value of u is known (in this case, the initial value), and the light grey spheres are grid points where the solution value is unknown.

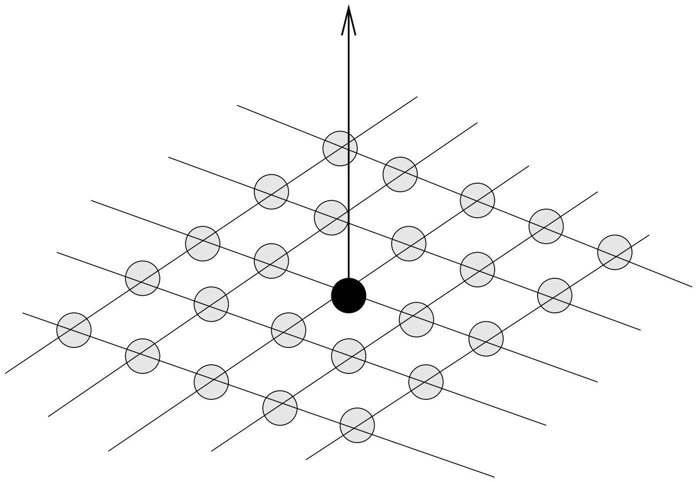
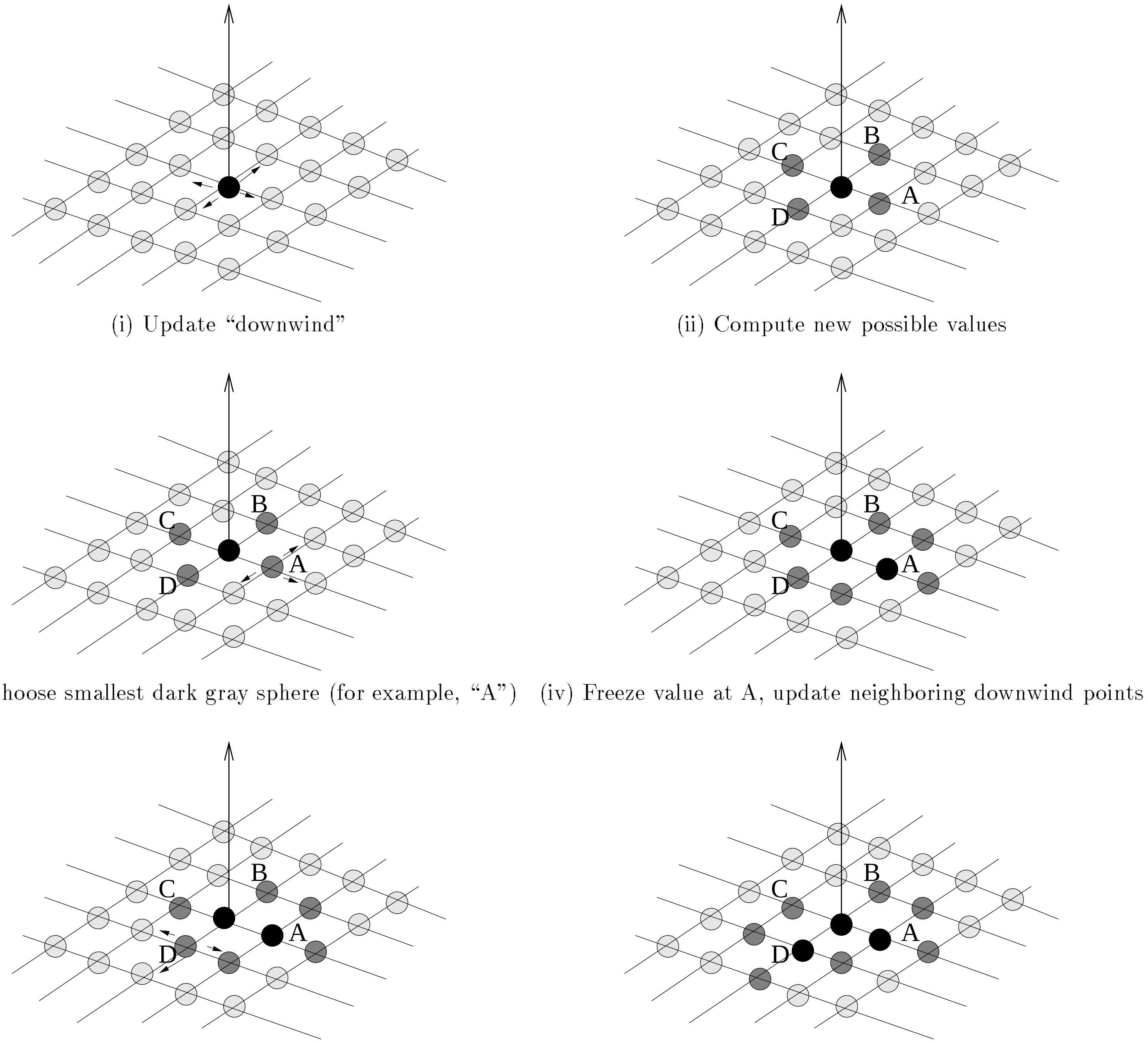


Figure 6: Beginning of Fast Marching Method

We may start the algorithm by marching "downwind" from the known value, computing new values at each of the four neighboring grid points, as shown in Figure 7. This provides possible values for u at each grid point $u_{-1,0}$, $u_{1,0}$, $u_{0,-1}$, $u_{0,1}$, and are shown as dark grey spheres in Figure 7.

Now, we would like to march downwind from these values given at the dark grey spheres, but we don't know which one to choose. The answer lies in the observation that the *smallest u value at these dark grey spheres must be correct.* Because of upwinding, no point can be affected by grid points containing larger values of u. Thus, we may freeze the value of u at this smallest dark grey sphere,[2] and proceed ahead with the algorithm; this is shown schematically in Figure 7.

[2]that is, turn it into a black sphere and consider its value known.



(i) Update "downwind"

(ii) Compute new possible values

(iii) Choose smallest dark gray sphere (for example, "A")

(iv) Freeze value at A, update neighboring downwind points

(v) Choose smallest dark gray sphere (for example, "D")

(vi) Freeze value at D, update neighboring downwind points

Figure 7: Update procedure for Fast Marching Method

This algorithm works because the process of recomputing the u values at downwind neighboring points cannot yield a value smaller than any of the accepted points. Thus, we can march the solution outwards, always selecting the narrow band grid point with minimum trial value for u, and readjusting downwind neighbors (see Figure 8).

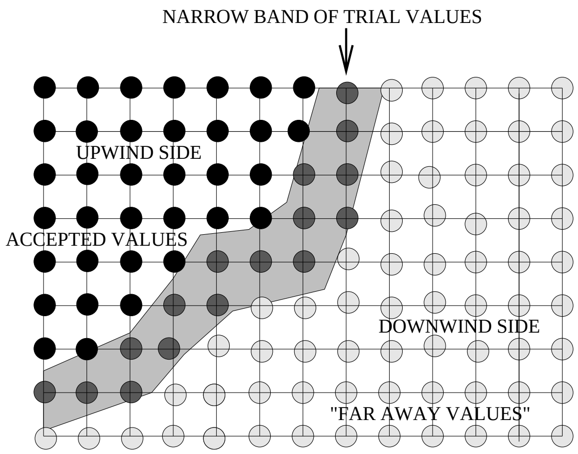


Figure 8: Upwind construction of accepted values

Another way to look at this scheme is that each minimum trial value begins an application of Huygens' 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 $Trial$.

Thus, the Fast Marching Method is as follows: First, tag points in the initial conditions as $Alive$. Then tag as $Close$ all points one grid point away. Finally, 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 of u.

2. Tag as $Close$ all neighbors of $Trial$ that are not $Alive$. If the neighbor is in Far, remove it from that list and add it to the set $Close$.

3. Recompute the values of u at all $Close$ neighbors of $Trial$ by solving the piecewise quadratic equation according to Eqn. 14.

4. Add the point $Trial$ to $Alive$; remove it from $Close$

5. Return to top of Loop;

3.2 Heap sorts and computational efficiency

The key to an efficient version of the above technique lies in a fast way of locating the grid point in the narrow band with the smallest value for u. An efficient scheme for doing this is discussed in detail in [30]; here we follow that discussion.[3]

There are several ways to store the *Trial* elements so that one can easily find the smallest element (see, for example, [5]). In our case, imagine that we have an ordered structure of the elements in *Trial*. When a point is accepted, its neighbors are updated, and their u values may change. Thus, only a small subset of the structure must be re-ordered in order to regain the ordering.

This leads quite naturally to a variation on a heap algorithm (see Sedgewick [22]) with back pointers to store the u values. Specifically, we use a min–heap data structure. In an abstract sense, a min-heap is a "complete binary tree" with a property that the value at any given node is less than or equal to the values at its children. In practice, it is more efficient to represent a heap sequentially as an array by storing a node at location k and its children at locations $2k$ and $2k+1$. From this definition, the parent of a given node at k is located at $k/2$. Therefore, the root which contains the smallest element is stored at location $k = 1$ in the array. Finding the parent or children of a given element are simple array accesses which take $O(1)$ time.

The values of u are stored, together with the indices which give their location in the grid structure. The marching algorithm works by first looking for the smallest element in the *NarrowBand*; this `FindSmallest` operation involves deleting the root and one sweep of `DownHeap` to ensure that the remaining elements satisfy the heap property. The algorithm proceeds by tagging the neighboring points that are not $Alive$. The $FarAway$ neighbors are added to the heap using an `Insert` operation and values at the remaining points are updated using equation (14). `Insert` works by increasing the heap size by one and trickling the new element upward to its correct location using an `UpHeap` operation. Lastly, to ensure that the updated u values do not violate the heap property, we need to perform an `UpHeap` operation starting at that location and proceeding up the tree.

The `DownHeap` and `UpHeap` operations (in the worst case) carry an element all the way from root to bottom or vice versa. Therefore, this takes $O(\log N)$ time assuming there are N elements in the heap. It is important

[3]The work and contributions of Dr. Ravikanth Malladi were invaluable in the design of the initial version of the Fast Marching Method.

to note that the heap, which is a complete binary tree, is always guaranteed to remain balanced. All that remains is the operation of searching for the $NarrowBand$ neighbors of the smallest element in the heap. This can be done in time $O(1)$ by maintaining back pointers from the grid to the heap array. Without the back pointers, the search takes time $O(N)$ in the worst case. As an example, Figure 9 shows a typical heap structure and an `UpHeap` operation after the element at location $(2,7)$ gets updated from 3.1 to 2.0.

Since the total work in changing the value of one element of the heap and bubbling its value upwards is $O(\log M)$, where M is the size of the heap, this produces a total operation count of $M \log M$ for the Fast Marching Method on a grid of total M points. Thus, if we imagine a three-dimensional grid of N points in each direction, the Fast Marching Method reduces the total operation count from N^4 to $N^3 \log N$; essentially, each grid point is visited once to compute its arrival time value. For more details, see [27, 28, 30, 17].

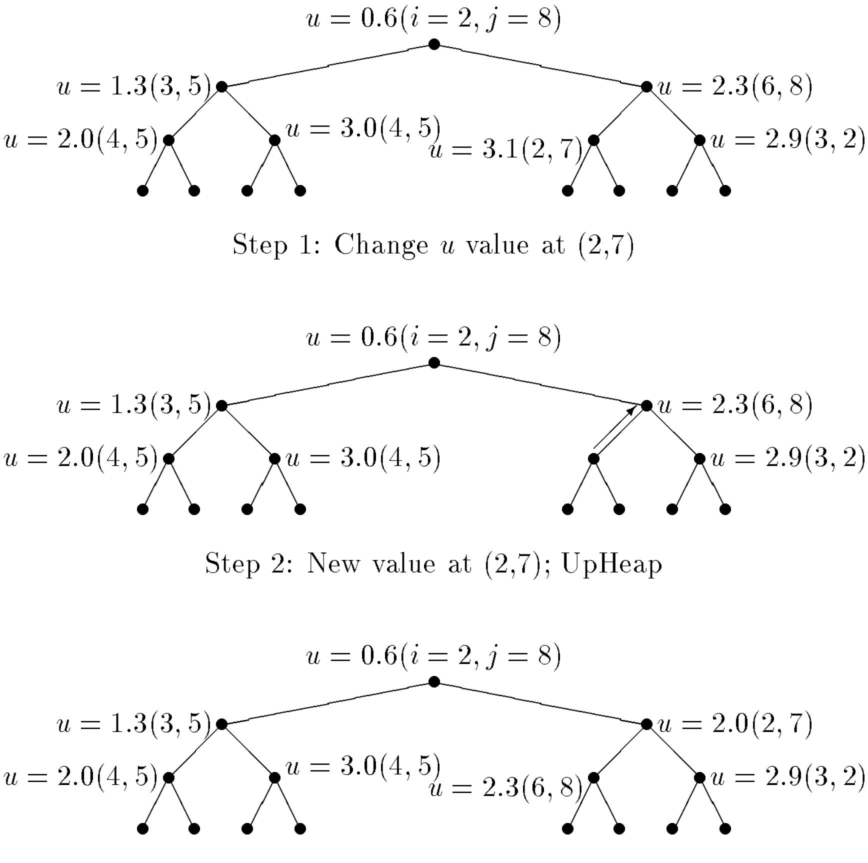


Figure 9: Heap structure and UpHeap+ operation

4 Related Algorithms

4.1 Network path algorithms

The Fast Marching Method is reminiscent of Dijkstra's algorithm [10] (see also [5] and [22]), which is a method for finding the shortest path on a network with prescribed weights between each link. As illustration, imagine one is given a rectangular network with equal unit cost of entering each link (see Figure 10).

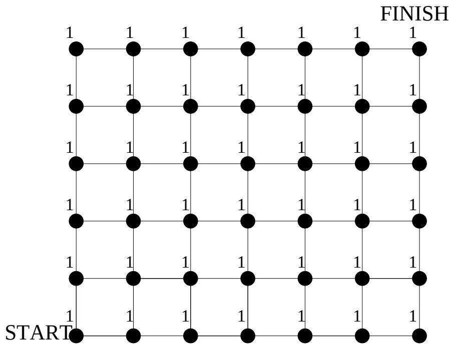


Figure 10: Inconsistent vs. consistent algorithms

Dijkstra's method starts at the "START" point, and expands outwards, visiting each node and keeping a running cost. The "front" is advanced by looking for the node reached with the smallest current cost, and then advancing to neighbors. In this sense, the method is similar to the Fast Marching Method; suitably programmed by means of a heap as described above, the method is an $O(N \log N)$ method of computing the cost of going from point A to point B along the network.

However, if the two points are positioned relative to the network so that the optimal path is a straight line diagonal between the two, such a graph search algorithm cannot distinguish between the various Manhattan graphs of equal cost connecting points A and B. The actual problem one wants to approximate is the solution of the *continuous* problem, rather than the network solution. Thus, such algorithms are said to be *inconsistent* with the underlying continuous problem; refinement of the network will not produce a solution which converges to the correct diagonal shortest path. While it is possible to rectify this problem by adding diagonal links to the graph, the

Fast Marching Method provides an approach which directly approximates the solution of the underlying partial differential equation through consistent numerical approximations, that is, one that selects the correct diagonal with relatively few nodes.

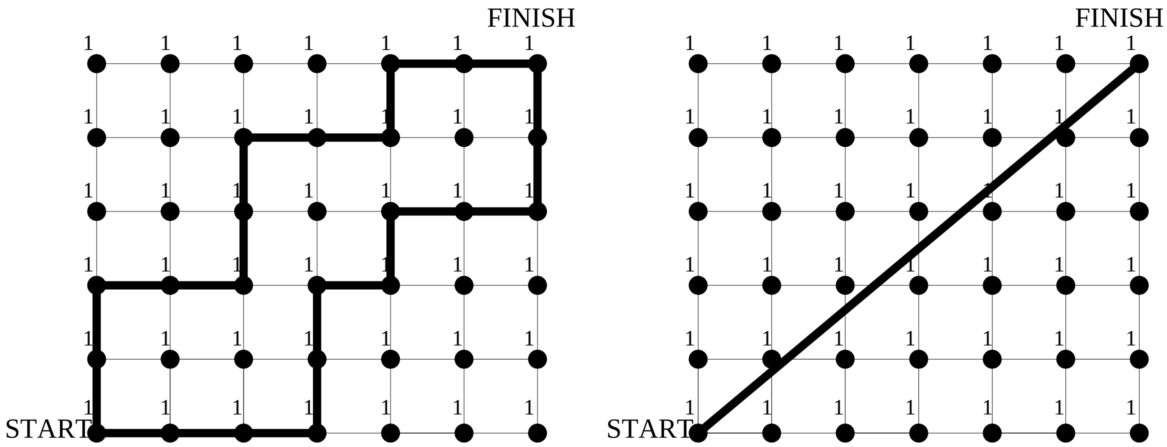


Fig. 11a: Dijkstra's method: Multiple "Shortest paths"

Fig. 11b: Fast Marching Method: Optimal diagonal path

Figure 11: Non-consistent vs. consistent algorithms

4.2 Optimal orderings

A different way to look at the Fast Marching Method is by returning to the iterative loop given in Eqn. 15. The Fast Marching Method is a re-ordering of the points so that the inner loop becomes a backsolve, and hence no outer loop iteration is necessary. The price of computing the correct re-ordering, done while the loop is in progress, is $O(\log N)$, which is what is required to re-order the heap once values are updated.

4.3 Relation to Level Set Methods

A companion technique, which also grew out of the work on theory and numerics of curve/surface evolution developed in [24] is known as the level set method, and also requires the notions of entropy conditions and viscosity solutions in order to produce stable and accurate numerical schemes. Level set methods, described by Osher and Sethian [18], track the motion of interfaces propagating under complex speed laws, and share with Fast Marching

Methods the ability to track fronts which change topology, break and merge. They handle more complex speed motions, including speeds which depend on local curvature and can track fronts that move forward and backwards. However, they are considerably more computationally expensive than Fast Marching Methods.

To delineate the differences between the two techniques, imagine a closed curve Γ in the plane propagating normal to itself with speed $\bar{F}$. Furthermore, assume that $\bar{F} > 0$, hence the front always moves "outwards". One way to characterize the position of this expanding front is to compute the arrival time $T(x, y)$ of the front as it crosses each point (x, y), as shown in Figure 12. Since the speed $\bar{F}$ is inversely proportional to the gradient, we must have

$$|\nabla T|\bar{F} = 1 \qquad T = 0 \text{ on } \Gamma. \tag{16}$$

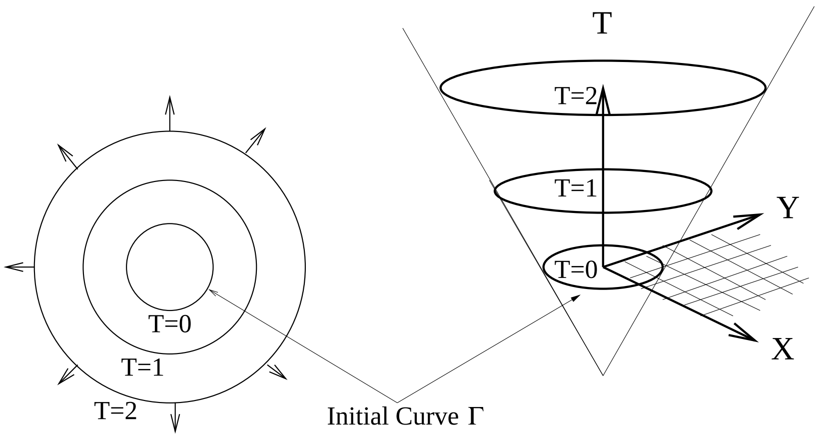


Figure 12: Transformation of front motion into boundary value problem

Thus, the front motion is characterized as the solution to a boundary value problem; if the speed $\bar{F}$ depends only on position, then the equation reduces to our familiar Eikonal equation, with $\bar{F} = 1/F$.

Conversely, suppose we embed the initial position of the front as the zero level set of a function ϕ in one higher dimension. The level set method identifies the evolution of this level set function ϕ with the propagation of the front itself through the time-dependent initial value problem

$$\Phi_t + \bar{F}|\nabla\Phi| = 0. \tag{17}$$

This equation describes the time evolution of the level set function Φ in such a way that the zero level set of this evolving function is always identified with the propagating interface; see Figure 13.

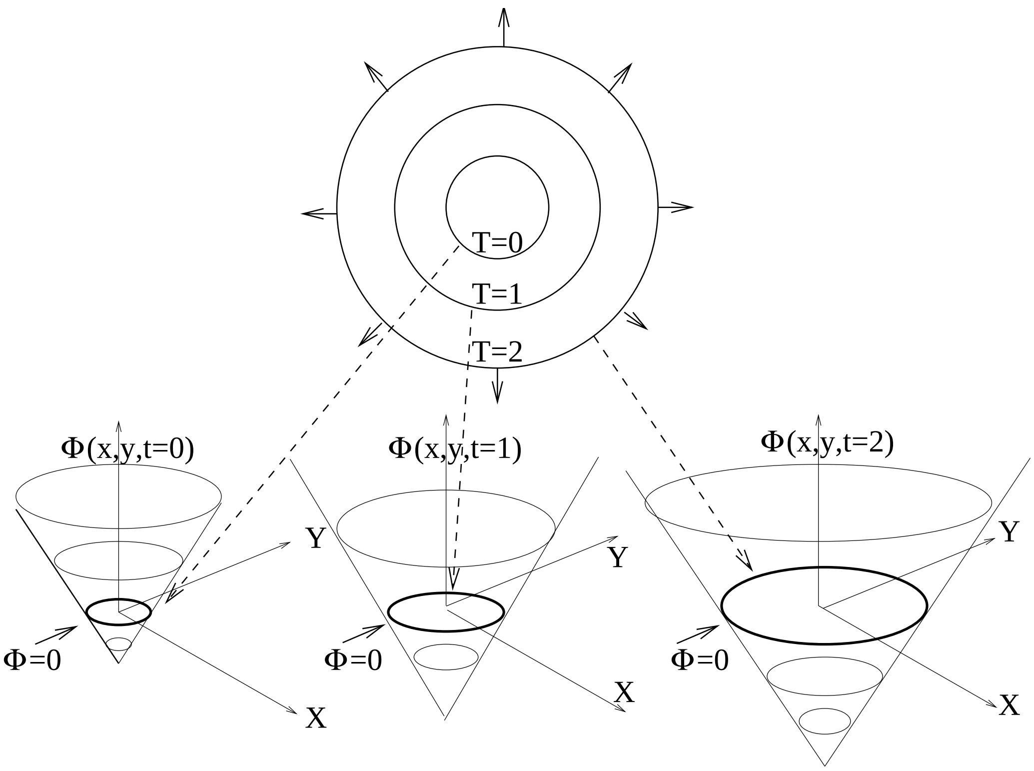


Figure 13: Transformation of front motion into initial value problem

The most obvious distinction between the two views is that the initial value level set formulation allows for both positive and negative speed functions $\bar{F}$; the front may move forwards and backwards as it evolves. The boundary value perspective is restricted to fronts that always move in the same direction, because it requires a single crossing time T at each grid point, and hence a point cannot be revisited.

Thus, we wish to solve

Initial Value Formulation	**Boundary Value Formulation**
$\phi_t + \bar{F}\lvert\nabla\phi\rvert = 0$	$\lvert\nabla T\rvert\bar{F} = 1$
Front= $\Gamma(t) = \{(x,y)\lvert\phi(x,y,t) = 0\}$	Front= $\Gamma(t) = \{(x,y)\lvert T(x,y) = t\}$
Applies for arbitrary $\bar{F}$	Requires $\bar{F} > 0$

(18)

Recall that the Fast Marching Method is an efficient "adaptive" technique which drops the computational labor involved in solving the boundary value formulation from $O(N^4)$ to $O(N^3 \log N)$, assuming N points in each direction of a three-dimensional problem. Similarly, an efficient implementation of the level set method, introduced in [1] and called the *Narrow Band Level Set Method*, is available. The original level set method given in [18] requires $O(N^4)$ work; here we assume roughly N time steps for the front to propagate. In comparison, the Narrow Band Method focuses all the computational labor onto a thin band around the zero level set, thus reducing the labor to $O(N^3 k)$, where k is the width of this narrow band, providing an efficient technique for implementing level set methods.[4]

Schematically, the history of the two methods and their adaptive versions is shown in Figure 14

[4]At first glance, the computational savings in the Fast Marching Method may not be evident on the basis of these operation counts. However, two additional advantages provide the largest computational savings. First, because the level set method is solving a time-dependent problem, time step restrictions in terms of CFL conditions based on the speed $\bar{F}$ determine the number of steps required to evolve a front; in contrast, the Fast Marching Method has no such restriction. The speed $\bar{F}$ of the front is irrelevant to the efficiency of the method. Second, the number of elements in the heap depends on the length of the front; in most cases, this length is small enough that, for all practical purposes, heap lookup is very fast.

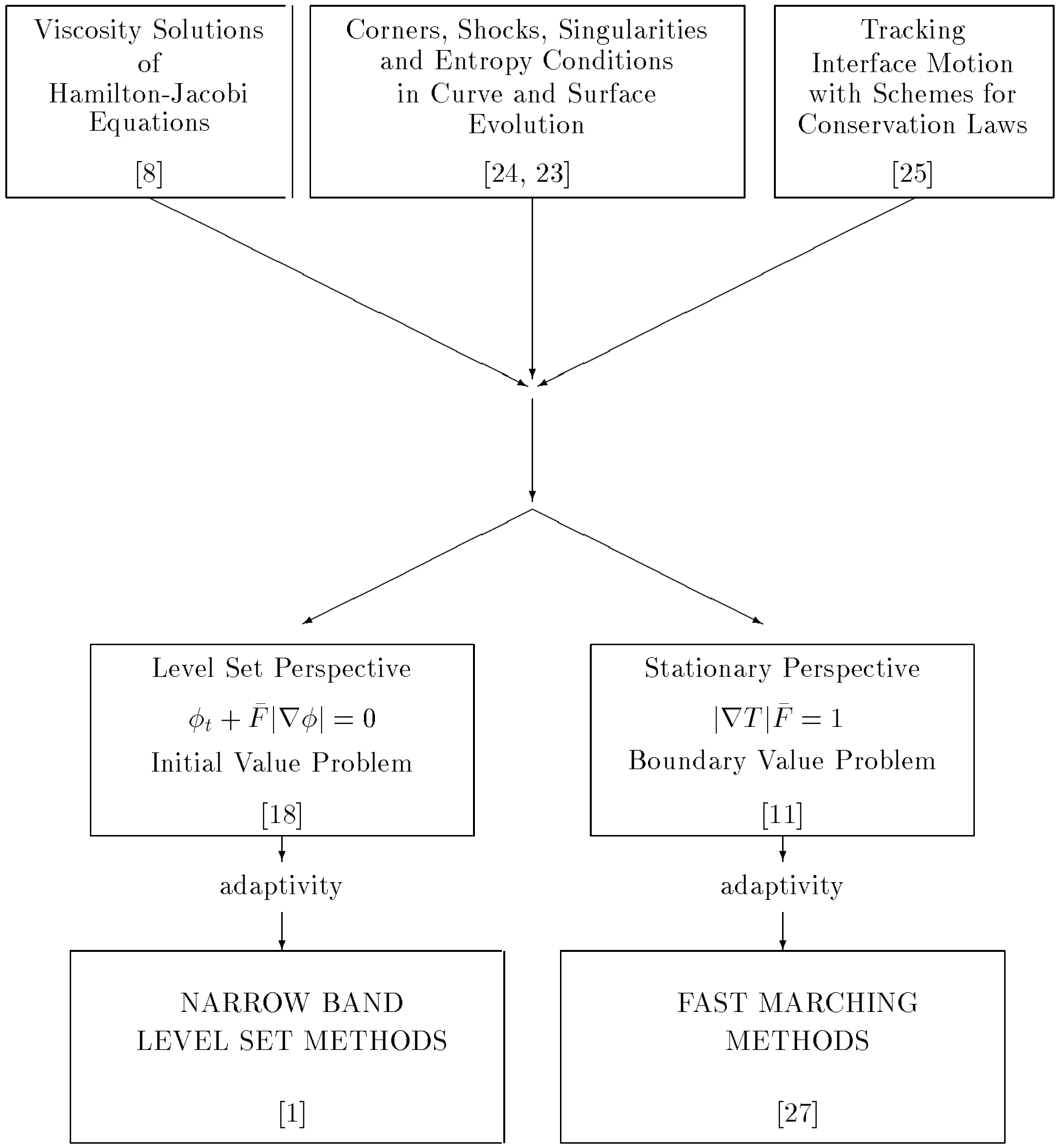


Figure 14: Evolution of theory and algorithms for interface propagations.

5 Extensions of Fast Marching Methods

5.1 Higher accuracy Fast Marching Methods

As presented, the Fast Marching Method is a first order scheme, owing to the use of a first order approximation to the gradient, namely,

$$\left[\begin{array}{c} \max(D^{-x}_{ijk}u, -D^{+x}_{ijk}u, 0)^2 + \\ \max(D^{-y}_{ijk}u, -D^{+y}_{ijk}u, 0)^2 + \\ \max(D^{-z}_{ijk}u, -D^{+z}_{ijk}u, 0)^2 \end{array} \right]^{1/2} = F_{ijk}, \tag{19}$$

We now discuss higher order Fast Marching Methods.

We begin by noting a slightly different implementation of the Fast Marching Method.[5] When values in *Trial* (which are the tentative values in the heap) are being recomputed, use only *Known* values in the computation. Consider the second order backward approximation to the first derivative u_x (see [9]), given by

$$u_x \approx \frac{3u_i - 4u_{i-1} + u_{i-2}}{2\Delta x},$$

which may be compactly written as

$$u_x \approx D^{-x}u + \frac{\Delta x}{2}(D^{-x})^2 u.$$

A similar expression holds for the forward difference, namely

$$u_x \approx D^{+x}u - \frac{\Delta x}{2}(D^{+x})^2 u.$$

Consider now the switch functions defined by (the expressions are similar in y and z)

$$\mathrm{switch}^{-x}_{ijk} = \left[\begin{array}{cc} 1 & \text{if } u_{i-2,j,k} \text{ and } u_{i-1,j,k} \text{ are known and } u_{i-2,j,k} \le u_{i-1,j,k} \\ 0 & \text{otherwise} \end{array} \right],$$

$$\mathrm{switch}^{+x}_{ijk} = \left[\begin{array}{cc} 1 & \text{if } u_{i+2,j,k} \text{ and } u_{i+1,j,k} \text{ are known and } u_{i+2,j,k} \le u_{i+1,j,k} \\ 0 & \text{otherwise} \end{array} \right].$$

We can then use these operators in the Fast Marching Method, namely,

[5] This variation was suggested by D. Adalsteinsson.

$$
\left[\begin{array}{c}
\max\left[\left[D^{-x}_{ijk}u + \mathrm{switch}^{-x}_{ijk}\frac{\Delta x}{2}(D^{-x}_{ijk})^2 u\right], -\left[D^{+x}_{ijk}u - \mathrm{switch}^{+x}_{ijk}\frac{\Delta x}{2}(D^{+x}_{ijk})^2 u\right], 0\right]^2 \\
+ \\
\max\left[\left[D^{-y}_{ijk}u + \mathrm{switch}^{-y}_{ijk}\frac{\Delta y}{2}(D^{-y}_{ijk})^2 u\right], -\left[D^{+y}_{ijk}u - \mathrm{switch}^{+y}_{ijk}\frac{\Delta y}{2}(D^{+y}_{ijk})^2 u\right], 0\right]^2 \\
+ \\
\max\left[\left[D^{-z}_{ijk}u + \mathrm{switch}^{-z}_{ijk}\frac{\Delta z}{2}(D^{-z}_{ijk})^2 u\right], -\left[D^{+z}_{ijk}u - \mathrm{switch}^{+z}_{ijk}\frac{\Delta z}{2}(D^{+z}_{ijk})^2 u\right], 0\right]^2
\end{array}\right]^{1/2}
$$

$$
= F_{ijk}. \tag{20}
$$

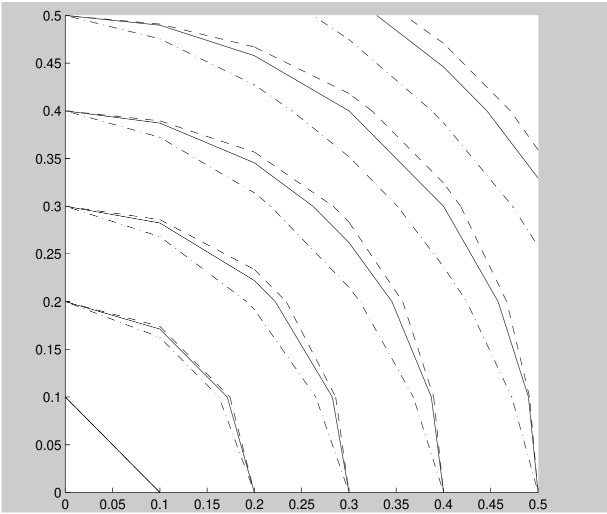
This scheme attempts to use a second order one-sided upwind stencil whenever points are available, but reverts to a first order scheme in the other cases. We make two observations. First, in order to start the scheme, we must use a higher order scheme to produce accurate values in a band around the boundary values. Second, we have chosen a very "conservative" higher order version of our Fast Marching Method; it is possible to devise versions which invoke the first order scheme less often.

5.1.1 Accuracy/order of this scheme

The question of whether or not the "second order" Fast Marching Method is really second order depends on how often the switches evaluate to zero, and on how the number of those points where a first order method is invoked changes as the mesh is refined. In many simulations, the number of points where the first order stencil must be chosen is relatively small, and the fraction of such points among the total often decreases as the mesh is refined. In these cases, the error is experimentally observed to be considerably reduced using this approach. The advantages of using second order operators wherever possible can be seen most clearly along diagonals, where the first order approximation to the gradient is significantly worse than the second order version. One can also use third and higher order one-sided differences; the degree to which these yield higher accuracy will again depend on how often the first order scheme is invoked.

5.1.2 Tests of scheme

Next, we examine the accuracy of the Fast Marching Method. The "second order" scheme is defined as the one which uses second order one-sided operators whenever possible. Figure 15 shows sets of points equidistant from the origin, computed using (i) the exact distance, (ii) the first order scheme, and (iii) the "second order" scheme on an extremely crude 6×6 grid. As can be seen, the "second order" method is considerably better. Figure 15 also gives the error associated with computing the distance function from a single point located at the origin.



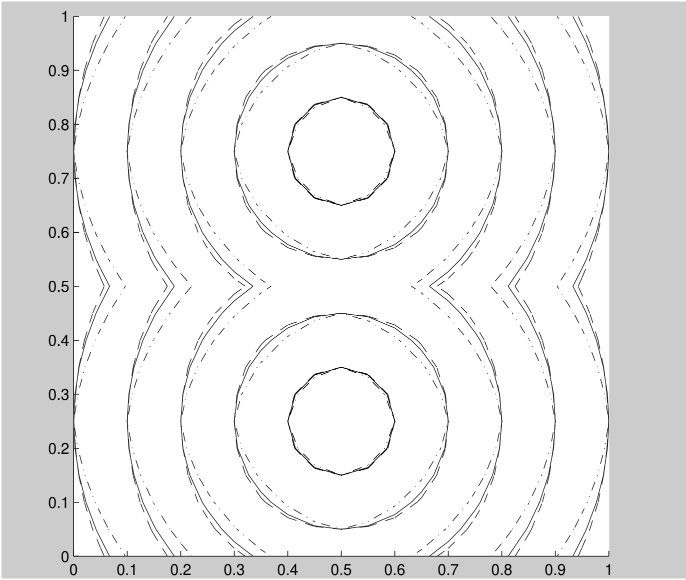
Equidistant points from origin: crude 6x6 Grid
Solid = exact, Short dashed/dotted = 1st order, Long dashed = 2nd order.

Grid	L_2 Error 1st Order	L_2 Error 2nd Order	L_∞ Error 1st Order	L_∞ Error 2nd Order
21^3	0.019790	0.007203	0.034042	0.012921
51^3	0.009409	0.002410	0.017063	0.004325
101^3	0.004610	0.000461	0.008570	0.000734
151^3	0.002289	0.000071	0.004325	0.000205

Figure 15: First and second order computations of distance from origin.

.

Next, we repeat this test, but use the Fast Marching Method to compute the distance from two points, one located at $(0.5, 0.25)$ and the other at $(0.5, 0.75)$. Here, the solution is non-differentiable, and the viscosity solution is chosen. Once again, the "second order" method is considerably more accurate. Figure 16 shows a calculation on a rough 21x21 grid; the table shows the error under mesh refinement.

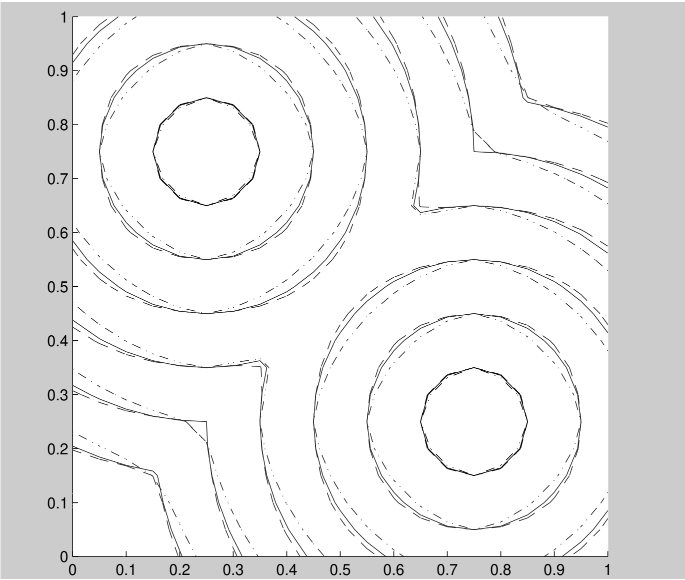


Equidistant points from two points: crude 21x21 Grid
Solid = exact, Short dashed/dotted = 1st order, Long dashed = 2nd order.

Grid	L_2 Error 1st Order	L_2 Error 2nd Order	L_∞ Error 1st Order	L_∞ Error 2nd Order
21^3	0.0096	0.00500	0.0185	0.01047
41^3	0.0051	0.00119	0.0101	0.00230
81^3	0.0028	0.00028	0.0056	0.00055
161^3	0.0014	0.00006	0.0028	0.00014

Figure 16: First and second order computations of distance from two points.

Finally, we repeat this test using a grid rotation and compute the distance from two points, one located at $(0.25, 0.75)$ and the other at $(0.75, 0.25)$. The L_2 error is still second-order convergent for the second order scheme, while the L_∞ error is first order. This is to be expected; first order error occurs along the shock lines, which, due to the grid alignment, occurs between grid points. Figure 17 shows a calculation on a rough 21x21 grid; the table shows the error under mesh refinement.



Equidistant points from two points: crude 21x21 Grid
Solid = exact, Short dashed/dotted = 1st order, Long dashed = 2nd order.

Grid	L_2 Error 1st Order	L_2 Error 2nd Order	L_∞ Error 1st Order	L_∞ Error 2nd Order
21^3	.01410	.005329	.02534	.013800
41^3	.00646	.001832	.01228	.005449
81^3	.00311	.000409	.00607	.002538
161^3	.0015	.000113	.00303	.001346

Figure 17: First and second order computations of distance from two points.

5.2 Non-uniform orthogonal grids

The implementation of Fast Marching Methods on non-uniform orthogonal meshes is straightforward. Non-uniform Cartesian meshes are handled by the difference operators. Cylindrical and spherical coordinate meshes require the standard altered expressions for the gradient in the upwind directions. Since the use of these altered expressions does not affect flow of the Fast Marching Method, the actual algorithm remains unchanged. Only the update formula is changed.

5.3 Triangulated unstructured mesh formulation

There are a variety of reasons to develop a Fast Marching Method for triangulated domains. First, triangulated grids may be constructed to fit material or fluid boundaries. For example, problems in photolithographic development in non-rectangular regions can require a triangulated meshing so that the non-etchable material boundaries are accurately represented. Second, surfaces are often described by triangulated patches, and hence this discretization is natural. In [13], a Fast Marching Method on triangulated unstructured meshes was introduced, using the unstructured mesh methodology for level set methods developed by Barth and Sethian [2], and applied to the problem of constructing geodesic shortest paths on manifolds. Various examples were given there, and we follow that work closely in this presentation. For further details, see [13].

5.3.1 The update procedure

As an introduction, recall the update procedure in the Fast Marching Method for readjusting the values of neighbors which are downwind to known points. Our goal is to solve the Eikonal equation $|\nabla T| = F$. This is the procedure by which new trial values are created for T in the heap of nearby points. Imagine a uniform square grid and suppose that the goal is to update the value of T at the center point (i, j). We label the values of u at the surrounding grid points $T_A = T_{i-1,j}$, $T_B = T_{i+1,j}$, $T_C = T_{i,j-1}$, and $T_D = T_{i,j+1}$ (see Figure 18). Some of the values may be infinite, corresponding to *Far* values.

Standing at the center point, the Fast Marching Method attempts to solve the quadratic equation given by each quadrant. For example, we refer to possible contributors A and C. Without loss of generality, there are two cases:

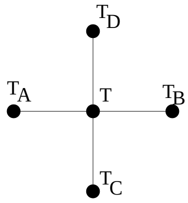


Figure 18: Construction of upwind solution on orthogonal mesh.

1. If only T_A is known, then we find the solution $T_A < T$ to the "quadratic" equation
$$(T - T_A)^2 = h^2 F_{ij}^2,$$
where h is the uniform grid spacing.

2. If T_A and T_C are known, then we take the real solution to the quadratic equation
$$(T - T_A)^2 + (T - T_C)^2 = h^2 F_{ij}^2,$$

For each possible up-down/left-right pair, we construct all possible real solutions; we then accept as the updated point the one that produces the smallest value of T. This is the Fast Marching Method described in [27, 28].

5.3.2 A scheme for a particular triangulated domain

We now to extend this method to triangular domains. In order to do so, we shall build a monotone update procedure on the triangulated mesh. As motivation, we consider the obvious triangulation of a square grid in the plane. Imagine the triangulation given in Figure 19.

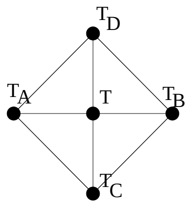


Figure 19: Simple triangulation for building a monotone update operator.

If we consider the known values T_A and T_C, and look at the triangle formed by the points which hold T_A, T_C and T, we can easily write down the equation of the plane determined by the known values T_A and T_C, as well as the unknown value T, namely (with h as the side length)

$$\left[\frac{T - T_A}{h}\right] x + \left[\frac{T - T_C}{h}\right] y + T = z.$$

Computing the gradient, we then want to select a value of T such that

$$\left[\frac{T - T_A}{h}\right]^2 + \left[\frac{T - T_C}{h}\right]^2 = F_{ij}^2.$$

In other words, we are lifting the plane by a value T at the center point i, j in order to have a gradient magnitude equal to $1/F$. We note that this is the exact same construction as the one produced by the "orthogonal" construction. Note also that in this case the gradient vector, with origin at the center point, always points into the triangle from which it is updated. This is not necessarily the case for an arbitrary acute triangle. In order to establish monotonicity we will need to verify this condition.

The inability to solve this quadratic corresponds to an inability to tilt at an appropriate angle, and the requirement that the solution T be greater than the contributors means that the solution is always constructed in an upwind manner. Using the same update rules as before, and the heap structure to maintain a list of *Trial* points, this provides a method for executing the Fast Marching Method on this simple triangulation.

5.3.3 Fast Marching Methods on triangulated domains

Following the construction unstructured mesh upwind approximations to the gradient ([2]), we now to extend this idea to an arbitrary triangulation.

Acute triangulations

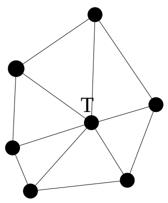


Figure 20: Acute triangulation around center grid point.

We start with an acute triangulation and consider the triangulation around the grid point given in Figure 20. A large number of triangles may share the center vertex. Our procedure, motivated by the simple triangulation in the previous section, is to compute a possible value for T from each triangle that includes the center point as a vertex. Since several triangles can produce admissible values for T, we must select an appropriate value. There are several possibilities. We chose the one that produces the smallest new value for T; this will correspond to an algorithm similar to the one used on the triangular mesh. More elaborate upwind constructions on triangulated meshes are given in [2].

Consider the non-obtuse triangle ABC in which the point to update is C. Assume that $T(B) > T(A)$. We first verify that the update is from within the triangle, i.e., the altitude h should be inside the non-obtuse triangle CBD (see Figure 21). This means that we search for $t = EC$ such that

$$\frac{t-u}{h} = F.$$

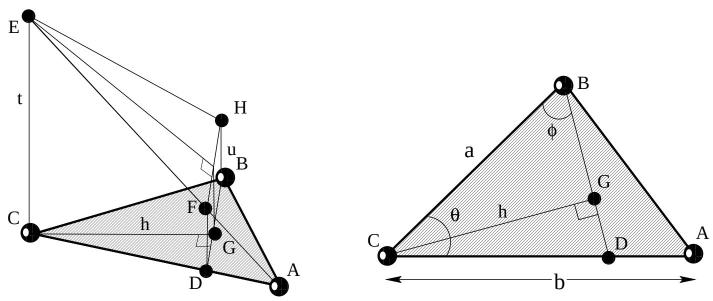


Figure 21: Given the triangle ABC, such that $u = u(B) - u(A)$, find $u(C) = u(A) + t$ such that $(t-u)/h = F$. Left: a perspective view of the triangle stencil supporting the $u()$ values that form a lifted plane with a gradient magnitude equal to F. Right: the trigonometry on the plane defined by the triangle stencil.

Denote $a = BC$ and $b = AC$; we have, by similarity, that $t/b = DF/AD = u/AD$, thus $CD = b - AD = b - bu/t = b(t-u)/u$. Next, by the Law of Cosines: $BD^2 = a^2 + CD^2 - 2a\,CD\cos\theta$, and by the Law of Sines: $\sin\phi = \frac{CD}{BD}\sin\theta$. Now, using the right angle triangle CBG we have

$$h = a\sin\phi = a\frac{CD}{BD}\sin\theta = \frac{a\,CD\sin\theta}{\sqrt{a^2 + CD^2 - 2a\,CD\cos\theta}}.$$

This yields a quadratic equation for t:

$$(a^2 + b^2 - 2ab\cos\theta)t^2 + 2bu(a\cos\theta - b)t + b^2(u^2 - F^2a^2\sin^2\theta) = 0.$$

The solution t must satisfy $u < t$, and should be updated from within the triangle, namely:

$$a \cos \theta < \frac{b(t-u)}{t} < \frac{a}{\cos \theta}. \tag{21}$$

Thus, the update procedure is given as follows:

If $u < t$ and $a \cos \theta < \frac{b(t-u)}{t} < \frac{a}{\cos \theta}$,

then $T(C) = \min\{T(C), t + T(A)\}$;

else $T(C) = \min\{T(C), bF + T(A), cF + T(B)\}$.

This is our scheme to extend the Fast Marching Method to acute triangulated domains.

Extension to general triangulations

So far we have required an acute triangulation. This is so that any front entering the side of a triangle will have two points to provide values before the third is computed. In other words, for monotonicity, we restrict the update to come from within the triangle, i.e., the gradient of the solution at a grid point should point into the triangle from which it is updated. The most straightforward way to enforce this requirement is to build a non-obtuse triangulation, thus making sure that the grid captures all incoming fronts.

One approach to handle non-acute triangulations, which we now describe, is to build numerical support locally at obtuse angles by splitting these angles in a special way. An obtuse angle at vertex A can be updated by its neighboring points in a consistent way only at a limited section of upcoming fronts. Connecting the vertex to any point in this section splits the obtuse angle into two acute ones.

The difficulty with this approach is that we need to reach back and use more triangles than simply the one containing the point to be updated. These other triangles might not be coplanar, since the unstructured mesh may lie on a convoluted surface. The idea is to extend this section by recursively unfolding the adjacent triangle(s), until a new vertex B is included in the extended section. Then the vertices are connected by a virtual directional edge from B to A (i.e., A may be updated by B). The length of the edge AB is equal to the distance between A and B on the unfolded triangles plane.

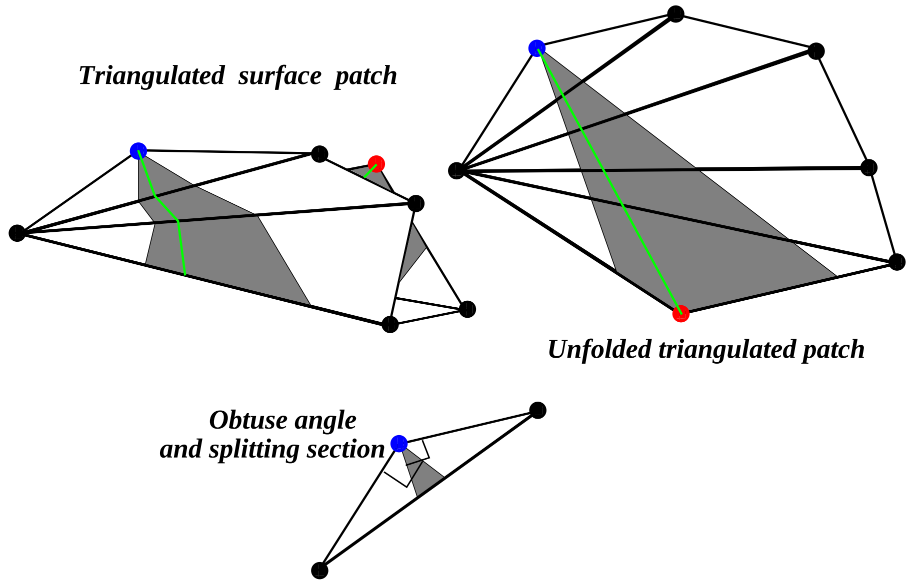


Figure 22: Top: a triangulated surface patch. Bottom: the initialization of the construction for the splitting section. Right: the unfolded patch and the splitting section expansion up to the first vertex B, and the virtual edge connecting the two vertices AB.

Finally, we note that a complexity analysis for this unfolding procedure given in [13] shows that the $O(N \log N)$ operation count is maintained. We refer the interested reader there for further details. Further discussion of triangulated Fast Marching Methods, higher order versions, and extensions to the basic techniques may be found in [33].

6 Applications

In this section, we provide a collection of examples to demonstrate the applicability of Fast Marching Methods.

6.1 Shape-offsetting

The first example, borrowed from [30], is the straightforward problem of shape-offsetting. Given a closed shape in two or three dimensions, one wants to compute the offset, obtained by propagating the boundary in its normal direction with constant unit speed. All that is required is to compute the distance to the boundary and to plot the level curves.

Here, we work directly with the Eikonal equation. Since the speed is $F = 1$, we must then solve

$$|\nabla T| = 1, \tag{22}$$

where the boundary condition is $T = 0$ on and inside the given shape. In Figure 23, the boundary is represented as a dark heavy line and we show the larger shape-offsets, obtained using the Fast Marching Method.

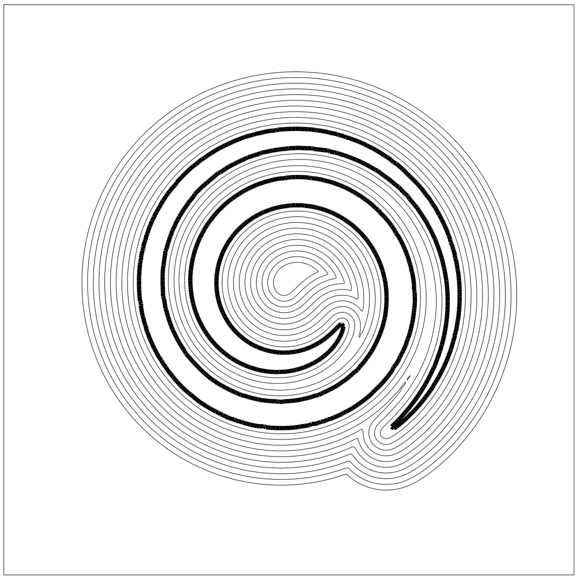
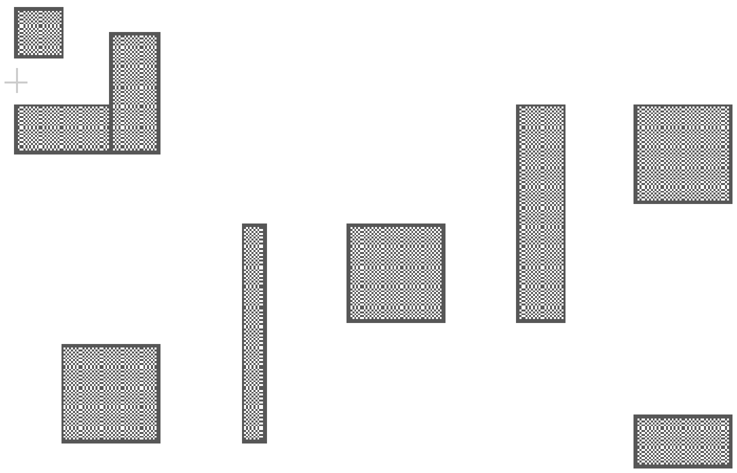


Figure 23: Shape-offsetting of spiral

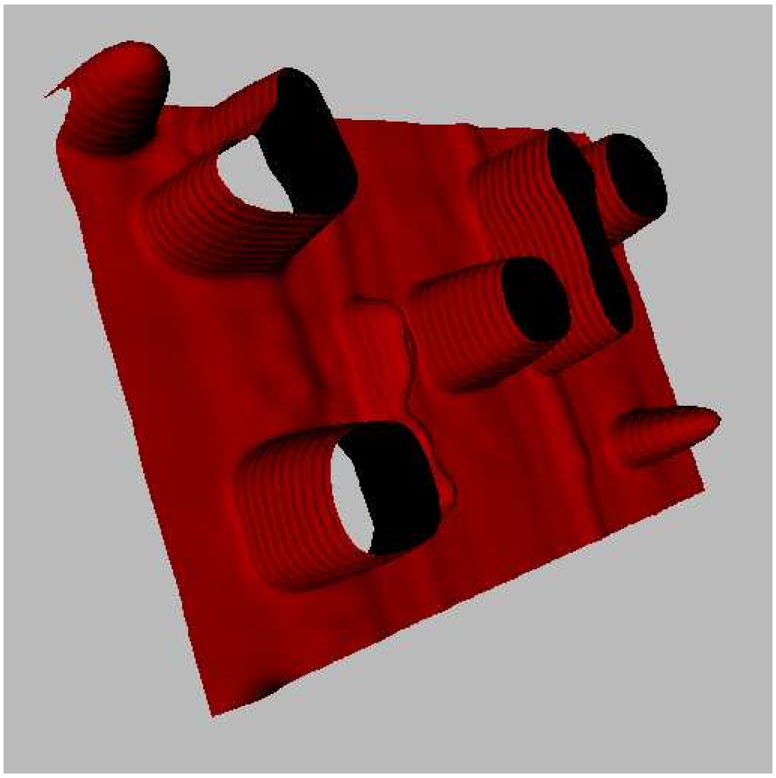
6.2 Photolithographic development

One aspect of the manufacturing of microchips is the process of *lithographic development*. In this process, the resist properties of a material are altered through exposure to a beam that has been partially blocked by a pattern mask. The material is then "developed", which means the material with less resistivity is etched away. The development process reduces to that of following an interface propagating downwards in three dimensions, where the speed in the normal direction is given as a supplied rate function at each point. The speed $F = F(x, y, z)$ depends only on position; however, it may change extremely rapidly. The goal in lithographic development is to track this evolving front. In order to develop realistic structures in three-dimensional development profiles, a grid of size $300 \times 300 \times 100$ is not unreasonable; hence a fast algorithm is of considerable value in the development step.

As illustration (see [30]), a rate function calculated using the three-dimensional exposure and post-exposure bake modules of TMA's Depict 4.0 [36] is coupled to the Fast Marching Method method. Figure 24(a) shows the top view of a mask placed on the board; the dark areas correspond to areas that are exposed to light. The presence of such factors as standing waves in the etching profile depends on issues such as the reflectivity of the surface. In Figure 24(b) a view of the developed profile is shown from underneath; the etching of the holes and the presence of standing waves can easily be seen. For further results, see [29].



(a) Masking pattern



(b) Lithographic development: View from below

Figure 24: Lithographic development using Fast Marching Method

6.3 Seismic traveltimes

Next, we apply Fast Marching Methods to problems involving the imaging of geophysical data sets. In [32], Sethian and Popovici used the Fast Marching Method to rapidly construct first arrival times in seismic analysis, and then coupled this work to prestack migration. Here, we summarize that work. For further details, see [32].

Three-dimensional prestack migration of surface seismic data is used to image the earth's subsurface when complex geological structures and velocity fields are present. The most commonly used imaging techniques applied to 3-D prestack surveys are methods based on the Kirchhoff integral, because of its flexibility in imaging irregularly sampled data and its relative computational efficiency. To perform Kirchhoff migration, one approximately solves the wave equation with a boundary integral method. The reflectivity at every point of the earth's interior is computed by summing the recorded data on multidimensional surfaces; the shapes of the summation surfaces and the summation weights are computed from the Green's functions of the single scattering wave-propagation experiment (see, for example, [21]).

6.3.1 Background equations

In some more detail, the essence of 3-D prestack migration (see, for example, [19]), is expressed by the following integral equation:

$$\text{Image}(\mathbf{x}) = \int \int_{\mathbf{x_s}} \int_{\mathbf{x_r}} G(\mathbf{x_s}, \mathbf{x}, \omega) G(\mathbf{x}, \mathbf{x_r}, \omega) \text{Data}(\mathbf{x_s}, \mathbf{x_r}, \omega) d\mathbf{x_r} d\mathbf{x_s} d\omega,$$

where $\mathbf{x}$ is the image output location, and $\mathbf{x_s}$ and $\mathbf{x_r}$ are the data source and receiver coordinates, and ω is angular frequency. The Green's functions $G(\mathbf{x_s}, \mathbf{x}, \omega)$ and $G(\mathbf{x}, \mathbf{x_r}, \omega)$ parameterize propagation from source to image point and from image point to receiver, respectively. In most implementations, the calculation is often done instead in the time domain, and can be expressed as the summation

$$\text{Image}(\mathbf{x}) = \sum_{\mathbf{x_s}} \sum_{\mathbf{x_r}} A_s A_r \text{Input}(\mathbf{x_s}, \mathbf{x_r}, t_s + t_r),$$

where Input is a filtered version of the input data, and the Green's functions are parameterized by the amplitudes A_s and A_r and traveltimes t_s and t_r.

For 3-D prestack Kirchhoff depth migration, the Green's functions are represented by five-dimensional (5D) tables; these tables are functions of the source/receiver surface locations (x, y) and of the reflector position (x, y, z)

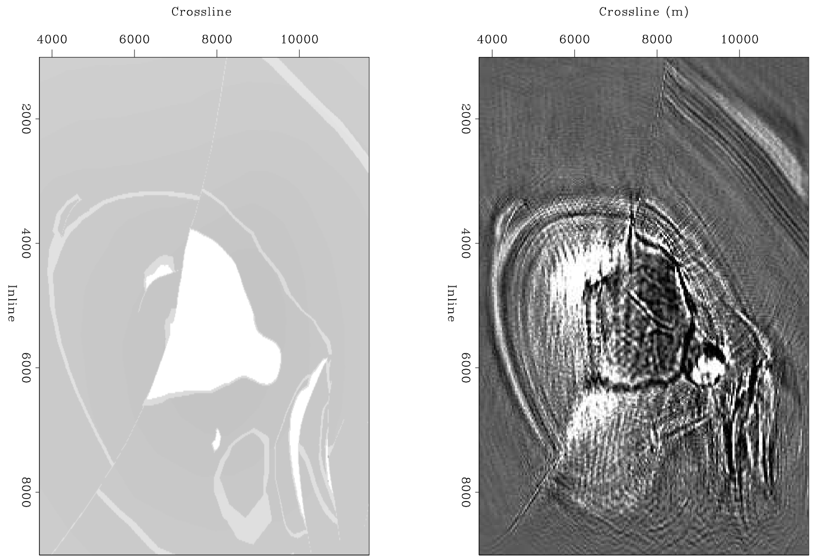
in the earth's interior. This Green's function parameterization is usually based on the assumption of acoustic propagation. This Kirchhoff prestack migration process consists of two stages. First, traveltime tables are computed and stored. Second, the migrated image is formed by convolving the prestack data with migration operators derived from the traveltime tables. Both phases present challenges from the perspective of the geophysical accuracy and of the computer implementation.

The key element of 3-D prestack Kirchhoff depth migration is the calculation of traveltime tables used to parameterize the asymptotic Green's functions. An efficient traveltime calculation method is required to generate the 5-D traveltime tables needed for 3-D Kirchhoff migration.[6] Also, since depth migration problems are generally applied in areas of complex velocity structure, the traveltime calculation method must be robust. Computing 3-D Green's function tables over a 100×100 kilometer area (about 430 marine blocks), with sources positioned every 200 meters, requires 1 terabyte of traveltime volumes. Thus, speed is an important issue. Finite difference approximations to traveltime computations include the work of Vidale [38] and van Trier and Symes [37]. Because of their speed, robustness, and the fact that they are unconditionally stable, Fast Marching Methods offer attractive methods of computing travel times.

6.3.2 Migration using the Fast Marching Method

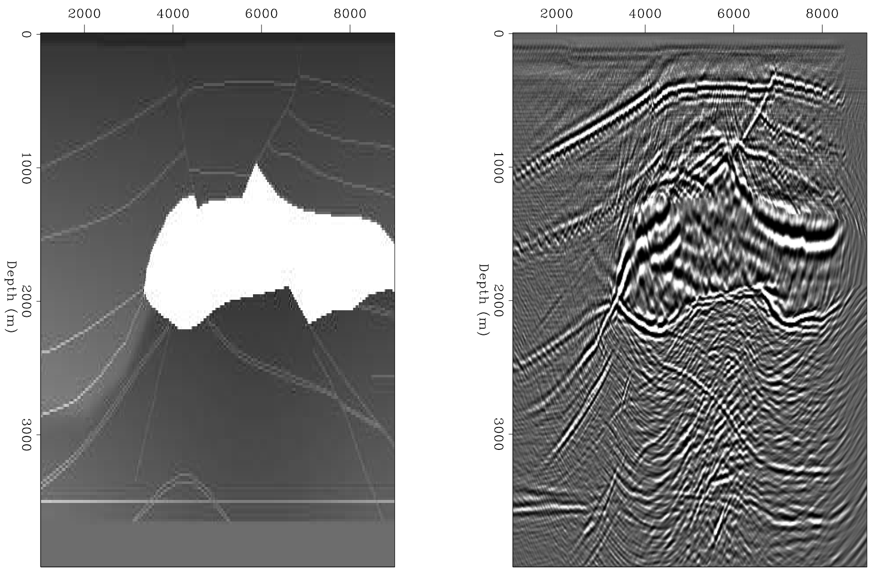
Figures 25 and 26 show slices through the three-dimensional velocity and corresponding structural images obtained from migration on prestack data obtained from a given data set. On the left, Figure 25 shows a depth slice through a velocity cube at a depth of 1220 meters; on the right, the corresponding migrated image slice is shown. The salt/sediment interface and the semicircular fault cutting through the salt body are imaged with high resolution. Figure 26 compares the velocity model on the left with the corresponding migrated line on the right for a different slice. The sediment images are imaged at the correct locations, together with the salt body borders. The areas with lesser quality are under the salt, most probably as a result of multiple reflected arrivals at this spot from the water bottom and intra-salt reflections, and close to the left side of the top of the salt, most probably because of the use of first arrivals in the Fast Marching Method. For further results and discussion of other issues, see [32] and [35].

[6] The Green's function can be reconstructed from traveltime tables that describe traveltimes from all surface points (x, y) to all subsurface locations (x, y, z); thus the tables are five-dimensional.



A. SEG/EAGE velocity, 1220 m depth   B. SEG/EAGE C3, first arrivals, 1220 m

Figure 25: Velocity model and and migrated image.



A. SEG/EAGE velocity, X=6820 m   B. SEG/EAGE C3, first arrivals, X=6820 m

Figure 26: Velocity model and migrated image.

6.4 An optimal algorithm for computing geodesic paths on surfaces

Suppose we try to compute the shortest path on a given surface between two points. This minimal geodesic can be obtained from a problem in front propagation as follows. As discussed in [13], one uses the triangulated Fast Marching Method to solve the straightforward Eikonal equation $|\nabla T| = 1$ on a triangulated approximation to the manifold. One can then backtrace on the surface itself, solving the ordinary differential equation

$$\frac{dX(s)}{ds} = -\nabla T,$$

where $X(s)$ traces out the geodesic path. We use a second order Heun's integration method on the triangulated surface with a switch to a first order scheme at sonic points in the gradient.

Here, we show two examples taken from [13]. Figure 27 presents a perspective view of the triangulation and shortest paths for two surfaces. Figure 27a shows shortest paths on regular triangulation of the surface given by the function $z(x, y) = 0.45 \sin(2\pi x) \sin(2\pi y)$ on $[0, 1] \times [0, 1]$, for grid size of 50×50. The minimal geodesics are painted on the triangulated surface and projected to the $x - y$ plane. Figure 27b gives a polyhedron example, in which a different speed function F was assigned to each side, causing a Snell's law effect along the edges. The speed F is 2 at the close side with the start point, 1 at the second top side, and 4 at the side of the destination point.

Next, we show a computation which illustrates the performance on manifolds with an underlying non-acute (obtuse) triangulation. Figure 28 shows the computation of minimal geodesics on a torus, in which some shortest paths in fact cut through the middle. The equi-distance curves are shown, as well as the shortest paths. We note that the computational complexity is unchanged; we have given an $O(N \log N)$ algorithm for constructing geodesic paths on triangulated surfaces.[7] For details, see [13].

[7] It is important to be clear about what is being claimed. We are *not* finding the exact shortest path for a given triangulation of a surface. The solution path using the above algorithm is the shortest path within an error which depends on the size of the triangulation. It is an *approximate* shortest path, constructed in $O(N \log N)$ steps, where N is the number of triangles used to tessellate the surface. As the triangulation is refined, this path converges to the exact shortest path.

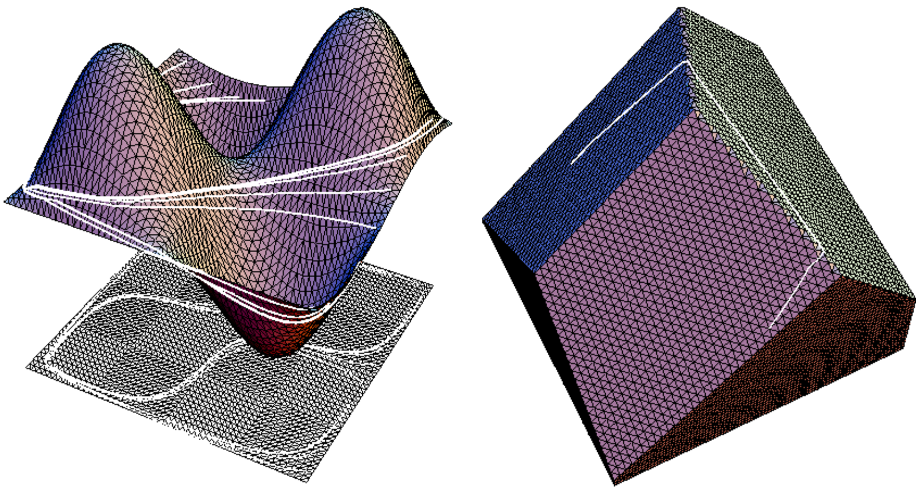
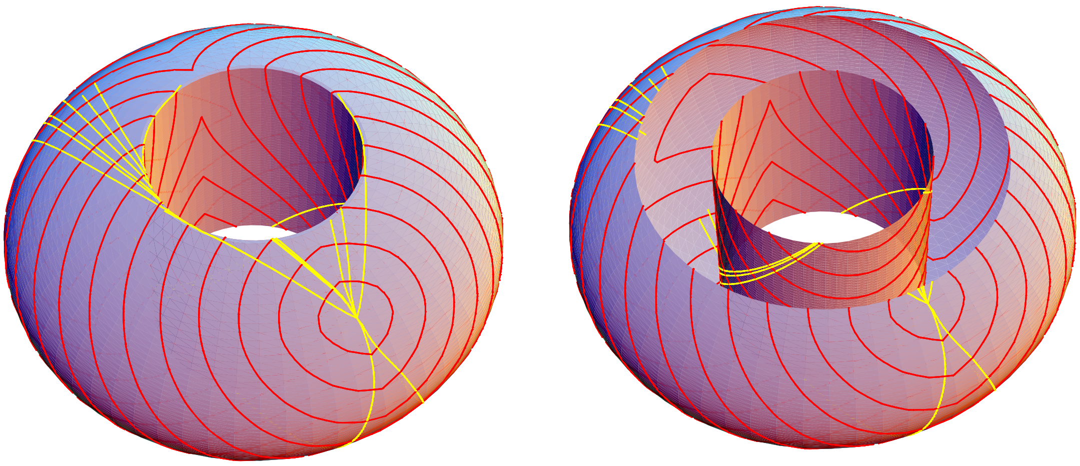


Figure 27: Computing minimal weighted geodesics on triangulated surfaces.



Full view of shortest paths

Cutaway view

Figure 28: Shortest paths on a bead (a genus one 2D manifold).

6.5 Optimal path planning

6.5.1 Statement of problem

The application of Fast Marching Methods to problems in path planning was first developed in [12]; here we summarize some of those results.

Given a cost function $F(x_1, x_2, .., x_n)$, and a starting point A in R^n, one goal in path planning is to construct the path $\gamma(\tau) : [0, \infty) \to R^n$ from A to any point B in R^n which minimizes the integral

$$\int_{A=\gamma(0)}^{B=\gamma(L)} F(\gamma(\tau))d\tau, \tag{23}$$

where τ is the arclength parameterization of γ; namely $|\gamma_\tau| = 1$, and L is the total length of γ.

More specifically, in two dimensions, suppose we are given the cost function $F(x, y)$ and a starting point A. Let $T(x, y)$ be the minimal cost required to travel from A to the point (x, y), that is

$$T(x, y) = \min_{\gamma} \int_{A}^{(x,y)} F(\gamma(\tau))d\tau, \tag{24}$$

with $|\gamma_\tau| = 1$. The level set $T(x, y) = C$ is the set of all points in R^2 that can be reached with minimal cost C, and the minimal cost paths are orthogonal to the level curves, hence we have

$$|\nabla T| = F(x, y). \tag{25}$$

Obviously, the Fast Marching Method can be used to solve this Eikonal equation to produce $T(x, y)$ in all of R^2. Then, given a point B in R^2, explicit construction of the shortest path comes through back propagation from B to A via the solution of the ordinary differential equation

$$X_t = -\nabla T \qquad \text{given} \qquad X(0) = B, \tag{26}$$

until we reach the starting point A.

6.5.2 Results

As illustration of these techniques, consider the problem of navigation with constraints and rotation. Instead of a robot as a single point, we imagine a two-dimensional rectangle with a given width and length. Thus, the initial position A of the robot in configuration space is specified by the position of the center of the rectangle, plus an angle θ between 0 and 2π. The final configuration B is similarly specified, and the goal is to construct the optimal path from A to B.

In the absence of obstacles, a completely straightforward application of the Fast Marching Method is possible: one discretizes the configuration space into a three-dimensional grid consisting of a discretized mesh for both R^2 and θ employing periodic boundary conditions for θ. Thus, we solve the Eikonal equation

$$\left[u_x^2 + u_y^2 + u_\theta^2\right]^{1/2} = 1. \tag{27}$$

In the presence of obstacles, we take the following approach; see [14]. Rather than maneuver an oddly shaped robot, we instead consider the robot as a point, and, for every discretized angle θ_i, alter the shape of the obstacles corresponding to that angle. To do this for all obstacles at each angle requires morphological shape operations consisting of dilations and translations; these too may be done using the Fast Marching Method, see [12]. In Figure 29 we show several examples of a two-dimensional robot with rotational angle navigating in two-dimensional space around obstacles. For further details about applications of Fast Marching Methods to path planning and robotic navigation, see [12] and [31].

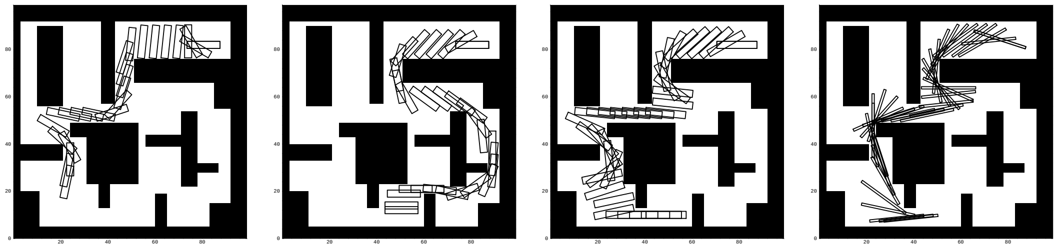


Figure 29: Two-dimensional navigation around obstacles with rotation

6.6 Visibility calculations and reflected bounces

Our last application is the evaluation of visibility. By visibility, we mean the determination of whether two points in a domain have an unobstructed straight line view of each other. Such issues are important in several areas:

- *Semiconductor manufacturing*: One factor which influences the etching and deposition process is whether or not a point on an evolving profile is visible from other points on the surface. As an example, imagine an etching beam which dislodges particles as it strikes the surface; these dislodged particles leave the interface and are then re-deposited somewhere else on the profile. Thus, the total flux of particles into any point on the front depends on computing the amount received from all other points on the front. There are various models which describe how these dislodged particles are ejected, including specular reflection (the angle of incidence equals the angle of refraction) and luminescent reflection (the angle at which a dislodged particle leaves the surface is uniformly distributed among all possible angles). In most cases, visibility comes into play; the amount of material received at a point from dislodged particles emanating from another point on the surface is zero if there is no direct line of sight between the two points, meaning that some other part of the profile blocks the incoming flux.

- *Ray-tracing and scene rendering*: In computer graphics, ray-tracing is often used to accurately render a complex scene. A scene can have many obstacles, corners, and occluded objects; the fundamental algorithm requires tracing a beam from a source to a final location, determining which points are directly visible.

- *Shadowing*: In other computer graphics and optics problems, given a source location, one wants to quickly locate the set of all points shadowed by given collection of obstacles.

Fast Marching Methods can aid in computing the visibility in these situations. Specifically, we give algorithms for the following:

1. Profile Visibility: Given a profile (that is, a curve in two dimensions or surface in three dimension), we can determine whether the straight line path between two points is blocked by the profile itself.

2. Scene visibility: Given a domain filled with obstacles in either two or three dimensions, we can determine if two points in the domain can directly see each other.

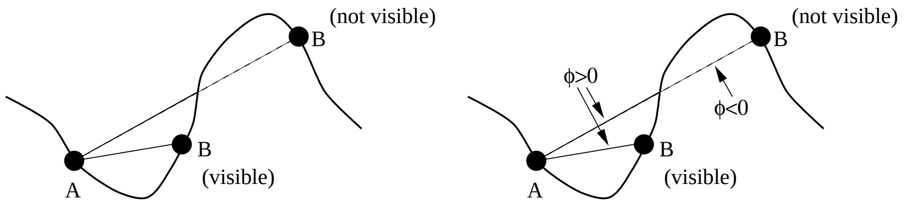
3. Shadowing: Given a source and a domain filled with obstacles, we can provide a fast algorithm to determine those points which are shadowed by the obstacles.

We now discuss these algorithms in some detail.

6.6.1 Profile visibility

Suppose we are given a profile and two points A and B; we wish to determine if there is a direct line of sight between A and B (see Figure 30a).

A brute force approach, while easy to program, is costly. Suppose that there are N points on the curve. By checking all segments between the two points, it costs $O(N)$ to determine if A can see B. Thus, checking whether each pair of points is mutually visible is $O(N^3)$. Of course, clever programming can make this much faster, including quadtree representations and judicious choices of how to proceed through the list of segments.



Can A see B?
Figure 30a

Using implicit representation test
Figure 30b

Figure 30: Visibility test between A and B.

A fast algorithm comes from exploiting an implicit representation of the interface. Given the profile, we use the Fast Marching Method to determine the signed distance T from the profile. Then we need only check to see if $T(x) > 0$ for all points x on the segment between A and B. If T is always positive, then the two points are visible, since the front cannot intersect the segment. We can perform this test of whether $T(x) > 0$ along the segment as follows. First, pick the midpoint of the segment and evaluate $T(x)$ by interpolation from the underlying grid; the value indicates the distance to the closest point on the front. One can now move that distance along the line segment in both directions and then query again. Repeating this process until one either reaches both A and B (in which case the two points were mutually visible) or reaches a point where $\phi < 0$ (in which case the two points were not mutually visible) terminates the algorithm; see Figure 30b.

6.6.2 Scene visibility

Imagine now a domain filled with obstacles, as in Figure 31. The goal is to determine whether or not two points A and B are visible to each other.

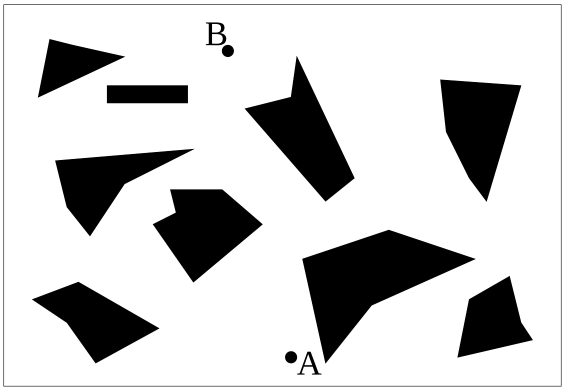


Figure 31: Scene visibility

We may proceed in a manner similar to the above and use the Fast Marching Method to determine scene visibility. First, we compute the distance function $T(x, y)$ from all the obstacles. We then can test the visibility of any two points in the scene by the above algorithm of querying whether $T(x, y) < 0$ at any point on the line connecting A and B.

How does this approach compare to a more direct algorithm which, given the line between two points, checks to see if it intersects any of the obstacles? The answer depends on the relative density of obstacles in the domain. If a large fraction of the domain is filled with obstacles, then the Fast Marching method computes the distance function in the empty spaces quickly. On the other hand, the direct testing approach of checking all obstacles against a given segment connecting two points can be time-consuming. In this case, the Fast Marching approach is viable. Conversely, if there are only a few, small obstacles in the domain, then the price of gridding the domain to construct the solution to the Eikonal equation may not be competitive with a direct testing approach.

6.6.3 Shadowing

Finally, we give a fast algorithm for approximately computing the shadow region behind obstacles created from a point source. Thus, given a collection of obstacles, we wish to find all points in a domain directly visible from a source at A. For a small number of obstacles, there are standard computer graphics techniques that typically perform comparison with each obstacle. As the number of obstacles increases, so can the complexity of such algorithms.

A different approach is as follows. Given a source at A, imagine the solution to two separate Eikonal equations:

$$|\nabla T_{\text{no-obstacles}}| = 1 \qquad |\nabla T_{\text{obstacles}}| = F(x, y), \tag{28}$$

where the right-hand side in the *obstacles* case is set to ∞ inside the obstacles and 1 otherwise. We solve each of these problems separately, using the Fast Marching Method, and then compare the solutions. If the obstacle blocks the source A from the point (x, y), then the first arrival time should be larger in the *obstacles* case. Thus, if $T_{\text{no-obstacles}}(x, y) = T_{\text{obstacles}}(x, y)$, then the point (x, y) is visible from the source; if, on the other hand, $T_{\text{no-obstacles}}(x, y) < T_{\text{obstacles}}(x, y)$, then the point is not visible. In practice, due to numerical error in the scheme, we need to use a threshold on the difference; that is, we check that

$$T_{\text{no-obstacles}}(x, y) + threshold < T_{\text{obstacles}}(x, y).$$

The threshold is chosen as a function of the mesh size h.

We note two advantages with this approach. First, the computational speed is independent of the number of obstacles and second, the technique is unchanged in three dimensions and higher. In Figure 32, we show the shadow zone created around six obstacles for two different placements of the sources.

The technique may be easily extended to multiple sources by running each source separately and then comparing the separate arrival times (see Figure 33). For further discussion of visibility and related issues, see [31].

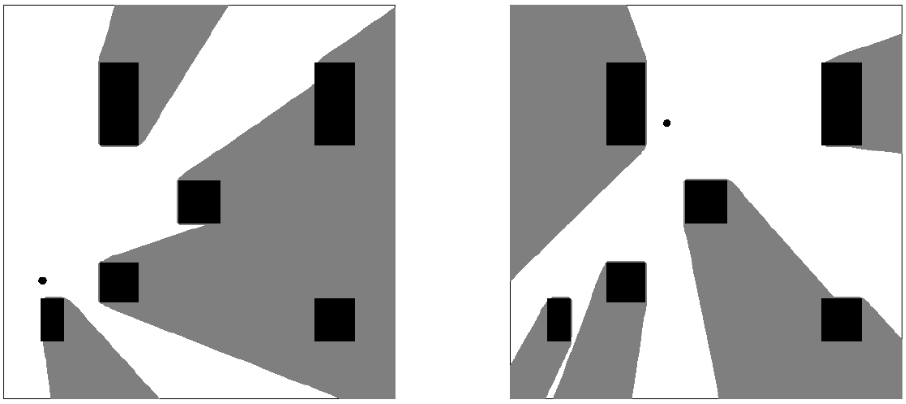


Figure 32: Shadow zone created by obstacles.

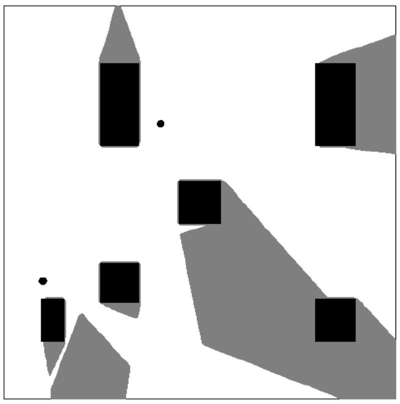


Figure 33: Shadow zone for two sources.

7 Additional Results

The range of applications for Fast Marching Methods is large and currently a field of active interest. These methods are in use in such areas as computer graphics, medical imaging, antenna and wave problems, crack dynamics, and flame propagation. They provide very fast ways to compute extension velocities which are critical to the efficient implementation of level set methods, and have found their way into morphological operations, Boolean operations on shapes, and implicit surface CAD/CAM descriptors. We refer the interested reader to [31] for a much more detailed discussion, as well as many additional applications.

Acknowledgments All calculations were performed at the University of California, Berkeley, and the Lawrence Berkeley National Laboratory. We would like to thank D. Adalsteinsson, R. Kimmel, R. Malladi, and M. Popovici for many useful discussions. The author may be reached at the address sethian@math.berkeley.edu. A web page devoted to Fast Marching Methods may be found at http://math.berkeley.edu/~sethian/level_set.html

References

[1] Adalsteinsson, D., and Sethian, J.A., *A Fast Level Set Method for Propagating Interfaces*, J. Comp. Phys., 118, 2, pp. 269–277, 1995.

[2] Barth, T.J., and Sethian, J.A., *Numerical Schemes for the Hamilton-Jacobi and Level Set Equations on Triangulated Domains*, J. Comp. Phys., 145, 1, pp. 1–40, 1998.

[3] Chopp, D.L., *Computing Minimal Surfaces via Level Set Curvature Flow*, Jour. of Comp. Phys., 106, pp. 77–91, 1993.

[4] Colella, P., and Puckett, E.G., *Modern Numerical Methods for Fluid Flow*, Lecture Notes, Department of Mechanical Engineering, University of California, Berkeley, CA, 1994.

[5] Cormen, T., Leiserson C., and Rivest, R., *Introduction to algorithms*, McGraw-Hill, New York, 1990.

[6] Crandall, M.G., Evans, L.C., and Lions, P-L., *Some Properties of Viscosity Solutions of Hamilton–Jacobi Equations*, Tran. AMS, 282, pp. 487–502, 1984.

[7] Crandall, M.G., Ishii, H., and Lions, P-L., *User's Guide to Viscosity Solutions of Second Order Partial Differential Equations*, Bull. AMS, 27/1, pp. 1–67, 1992.

[8] Crandall, M.G., and Lions, P-L., *Viscosity Solutions of Hamilton–Jacobi Equations*, Tran. AMS, 277, pp. 1–43, 1983.

[9] Dahquist, G. and Bjork, A., *Numerical Methods*, Prentice-Hall, New Jersey, 1974.

[10] Dijkstra, E.W., *A Note on Two Problems in Connection with Graphs*, Numerische Mathematic, 1:269–271, 1959.

[11] Garabedian, P., Partial Differential Equations, Wiley, New York, 1964.

[12] Kimmel, R., and Sethian, J.A., *Fast Marching Methods for Robotic Navigation with Constraints*, Center for Pure and Applied Mathematics Report, Univ. of California, Berkeley, May 1996, submitted for publication, Int. Journal Robotics Research, 1998.

[13] Kimmel, R., and Sethian, J.A., *Fast Marching Methods on Triangulated Domains*, Proc. Nat. Acad. Sci., 95, pp. 8341-8435, 1998.

[14] Latombe, J.C., *Robot motion planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[15] LeVeque, R.J., *Numerical Methods for Conservation Laws*, Birkhauser, Basel, 1992.

[16] Malladi, R., Sethian, J.A., and Vemuri, B.C., *Evolutionary Fronts for Topology-independent Shape Modeling and Recovery*, in Proceedings of Third European Conference on Computer Vision, Stockholm, Sweden, Lecture Notes in Computer Science, 800:3–13, 1994.

[17] Malladi, R., and Sethian, J.A., *An $O(N \log N)$ Algorithm for Shape Modeling*, Proc. Nat. Acad. Sci., Vol. 93, 1996.

[18] Osher, S., and Sethian, J.A., *Fronts Propagating with Curvature Dependent speed: Algorithms Based on Hamilton-Jacobi Formulation*, J. Comp. Phys., 79:12-49, 1988.

[19] Popovici, M., *Prestack migration by split-step DSR*, Geophysics, 61, 5, pp. 1412-16, 1996.

[20] Rouy, E. and Tourin, A., *A Viscosity Solutions Approach to Shape-From-Shading*, SIAM J. Num. Anal, 29, 3, pp. 867–884, 1992.

[21] Schneider, W.A. Jr., *Robust and efficient upwind finite-difference traveltime calculations in three dimensions*, Geophysics, 60, pp. 1108–1117, 1995.

[22] Sedgewick, R., *Algorithms*, Addison-Wesley, Reading, MA, 1988.

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

[24] Sethian, J.A., *Curvature and the Evolution of Fronts*, Comm. Math. Phys., 101:487–499, 1985.

[25] Sethian, J.A., *Numerical Methods for Propagating Fronts*, in Variational Methods for Free Surface Interfaces, Eds. P. Concus and R. Finn, Springer-Verlag, NY, 1987.

[26] Sethian, J.A., *Numerical Algorithms for Propagating Interfaces: Hamilton–Jacobi Equations and Conservation Laws*, Journal of Differential Geometry, 31, pp. 131–161, 1990.

[27] Sethian, J.A., *A Marching Level Set Method for Monotonically Advancing Fronts*, Proc. Nat. Acad. Sci., 93(4): 1591–1595, 1996.

[28] Sethian, J.A., *Applications of Level Set Methods for Propagating Interfaces*, Acta Numerica, 1996.

[29] Sethian, J.A., *Fast Marching Level Set Methods for Three-Dimensional Photolithography Development*, Proceedings, SPIE 1996 International Symposium on Microlithography, Santa Clara, California, March, 1996.

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

[31] Sethian, J.A., *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision and Materials Sciences*, Cambridge University Press, 1998.

[32] Sethian, J.A., and Popovici, A.M., *Three dimensional traveltimes computation using the Fast Marching Method*, in press, Geophysics, 1998.

[33] Sethian, J.A,. and Vladimirsky, A., *Extensions to Triangulated Fast Marching Methods*, to be submitted for publication, 1998.

[34] Sod, G.A., *Numerical Methods in Fluid Dynamics*, Cambridge University Press, 1985.

[35] 3DGeo Corporation, *Computing and Imaging using Fast Marching Methods*, 3DGeo Corporation, Internal Report, June, 1998.

[36] Technology Modeling Associates, Three-Dimensional Photolithography Simulation with Depict 4.0", Technology Modeling Associates, Internal Documentation, January 1996.

[37] van Trier, J., and Symes, W.W., *Upwind Finite-difference Calculations of Traveltimes*, Geophysics, 56, 6, pp. 812–821, 1991.

[38] Vidale, J., *Finite-Difference Calculation of Travel Times*, Bull. of Seism. Soc. of Amer., 78, 6, pp. 2062–2076, 1988.

[39] Vidale, J., *Finite-difference calculation of traveltimes in three dimensions, Geophysics,* **55**, *521–526.*, 1990.