Fast Marching Methods and Level Set Methods: An Implementation

by

Jeff Dicker
Honours Student

A THESIS SUBMITTED IN PARTIAL FULLFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
HONOURS IN COMPUTER SCIENCE

in

Irving K. Barber School of Arts and Sciences

THE UNIVERSITY OF BRITISH COLUMBIA
OKANAGAN CAMPUS
March 2006

Abstract

A wide variety of problems can be formulated as an interface propagation. Some examples are burning flames, waves in water and physical boundaries. The fast marching methods and narrow band level set method are useful for finding a solution to these problems.

The fast marching method is associated with the boundary value problem, and as such can only be used for a propagation which strictly expands or contracts. It is in contrast with the narrow band level set method, which is associated with the initial value formulation; it can be used for the propagation of interfaces which both expand and contract.

There is a parallel of both of these problems to general wave equations. Thus, by solving the Hamilton-Jacobi equation with an appropriate flux function, numerical approximation schemes for both propagation methods can be naturally formulated in a way such that the correct solution is obtained.

The final formulation of the two algorithms proves to be both robust and efficient, although they do not produce a very accurate solution using first order approximations.

Contents

1. Introduction

Interface propagation has numerous applications. Water waves, burning fires and physical boundaries all have a form which is easily interpreted as the propagation of an interface. Furthermore, image partitioning problems and even less obvious examples like optimal path planning can be represented this way[1].

This application of this project will be in the field of computer vision; the narrow band level set method and fast marching method can be used when solving the problem of image segmentation.

In order to implement these methods, a numerical approximation scheme for the two principle interface propagation formulations must be constructed. These schemes must propagate the interface using a desired motion, and satisfy certain requirements, such as not introducing any new information when propagating. Using a Hamilton-Jacobi formulation will allow the construction of such schemes, where numerical approximations will be provided by numerical flux functions. With this formulation, the final algorithms can be constructed.

2. Overview of Interface Propagation

In order to understand interface propagation, the interface itself must first be understood . When given an arbitrary region, such as the one in Figure 1, the interface is defined as the curve, or surface, separating the area inside of the region from the area outside of the region. For the purpose of this document, the examined interfaces exist in $\mathbb{R}^2$, and therefore exist as curves on a plane.

The motion of the interface must be understood in order to propagate it properly. Two different types of motion are distinguished: an interface which is strictly expanding or contracting can be described by the boundary value formulation, whereas an interface which arbitrarily expands and contracts must be described using the initial value formulation.

To further define the motion of the interface, a description of the velocity is required. Three different velocity components are formulated: a component local to part of the interface, a component belonging to global properties of the surface, and a component independent of the surface. It is more convenient to ignore the separate directions of these velocity components and simply use a scalar-valued function F to

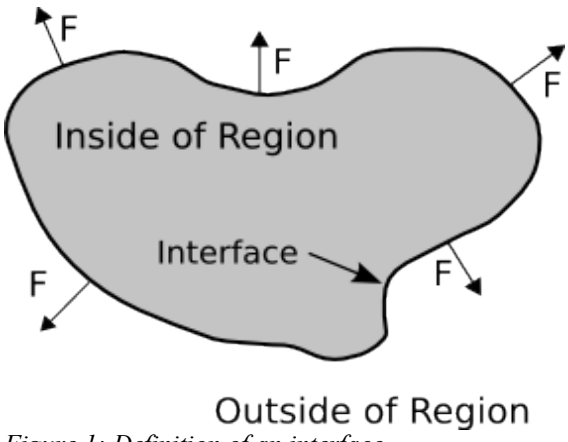

Figure 1: Definition of an interface.

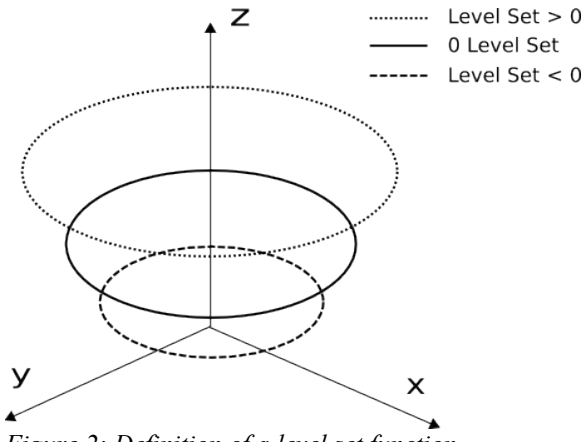

Figure 2: Definition of a level set function.

describe the velocity normal to the interface. Figure 1 shows F. The function F is often referred to as the speed function, and can be formed from the previously mentioned components.

2.1 The Boundary Value Formulation

As stated earlier, the description of an interface which is strictly expanding or contracting is known as the boundary value formulation:

$$|\nabla T| F = 1 \quad ,$$

where T is the arrival function and F is the speed of the interface. This equation is known as the Eikonal equation.

In this case, the arrival function T = T(x, y) is a function in $\mathbb{R}^3$ for a surface in $\mathbb{R}^2$. Given a coordinate pair (x, y), T will give the time at which the interface reaches (x, y). This function, in effect, is what the fast marching method will produce – it describes the time at which the interface will arrive at any given point.

For the boundary value formulation, F > 0 or F < 0 for all time. This implies that an interface described by the boundary value formulation is strictly expanding or contracting. Since the interface can only move in one direction, the efficient Fast Marching Method can be used. The initial value formulation does not impose this restriction.

2.2 The Initial Value Formulation

In the initial value formulation, the motion of an interface that is not strictly expanding or strictly contracting is described. That is, using the initial value formulation, an interface may propagate back to points it has already propagated to. It follows that the speed function F is no longer necessarily strictly greater or strictly less than 0 for all time. In order to facilitate this more general definition, a level set function will be required.

The level set function used is two dimensions higher than the surface: here it will be a function in $\mathbb{R}^4$ for a surface in $\mathbb{R}^2$. With the level set function, the interface at time T can be 'embedded' as the 0 level set, where the other level sets (negative and positive) are at times less than and greater than T, respectively. This is illustrated in Fig 2, where a circle propagating outwards is shown as a zero level set embedded into a level set function.

Now, the initial value formulation can be described by

$$\phi_t + F|\nabla \phi| = 0 \quad ,$$

where $\phi(t)$ is the level set function, F is the speed function, and ϕ_t represents the first derivative of ϕ with respect to *t*.

While the initial value formulation allows for a more general way to move an interface, the Fast Marching Method can no longer be used. Instead, the less efficient narrow band level set method can be used.

2.3 Upwind Values

Some thought must be put into where the data will come from when approximating the next time step values for an interface. When working with interface propagation, information from the next time step is unknown. Thus, only information

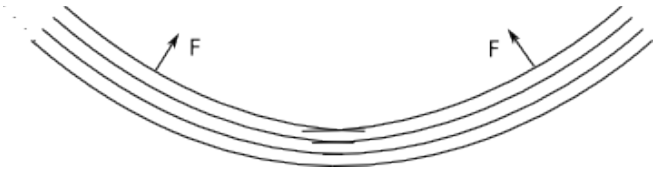

Figure 3: A smooth curve can quickly develop a singular point.

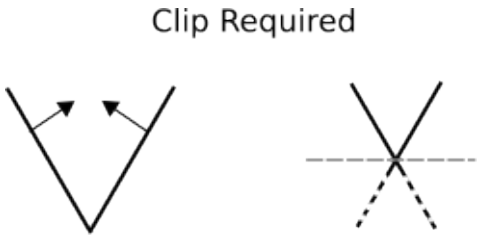

Figure 4: A clip is required to remove the dashed overlapping pieces at the bottom.

from the known time steps – earlier time steps – should be used to produce the next time step's data. In other words, numerical schemes used by algorithms to solve interface propagation problems must use upwind values appropriately.

3. The Entropy Condition

There is no guarantee of any level of smoothness in an interface when it moves at a constant speed. As a matter of fact, even when propagating a C^∞ function (such as a sinusoid) as an interface, a singular point may quickly develop. This fact is illustrated in Figure 3, where the singular point leaves overlapping pieces near the center. The earlier formulation of an interface as a curve which separates two regions may have seemed trivial, but now becomes an important distinction geometrically. The possibility of curve overlap is inevitable, and since the interface should separate two regions, it is clear that leaving overlap pieces is not the desired solution to the propagation. Figure 4 shows that these overlapping pieces will need to be 'clipped'.

A smoothing term ε can be used in the speed function such that

$$F = 1 - \varepsilon\kappa$$

is the speed, where κ is the curvature at a given point on the interface. If F uses a smoothing term according to the curvature in the formulation of the interface motion, the interface will be C^∞ for all time. It follows that there will be no overlap: the interface simply smooths the singularity out into the curve. However, there is a flaw with this solution: the desired solution to the expansion not only must have no overlap, it must be derived only from the original interface information. No new information should be added during the propagation. Thus, this solution will not produce the singularities that are simply a property of the original interface after undergoing expansion.

This condition imposed on the creation of new information is known as the entropy condition. As with the traditional definition of entropy, in this case it describes a constraint on the addition of randomness; no new information can be added during interface propagation.

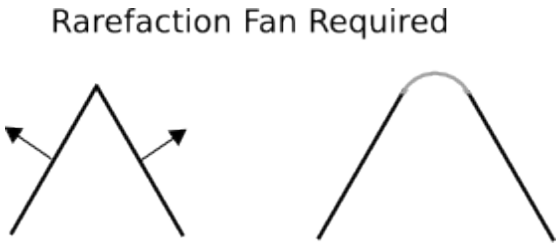

Figure 5: When pieces come apart during propagation, forming a gap, a rarefaction fan must be built.

The most straightforward way of propagating an interface such that it satisfies the entropy condition requires the use of Huygens' principle (see [1] for more detail on this). Waves emanate from the interface and no new information is introduced. Propagating these waves will not produce the overlap while propagating

the interface by its normal does. Sethian showed that this scheme is equivalent to using only grid points which are reached as first arrivals when expanding an interface. In short, if a grid point only allows the interface to traverse it once, then the problem of having overlap is no longer a possibility – the overlap would have to be placed on grid points that have already been traversed. Furthermore, no smoothing is involved and the entropy condition is satisfied.

Conversely to the overlap issue, a gap can form when an interface expands, as shown in Figure 5. This problem can be resolved in the same way – the first arrival solution will give a rarefaction fan that joins the two pieces of the fan.

4. Viscosity Solutions

If a smoothing term was added to the speed function F, the expanded interface clearly would violate the entropy condition. It would smooth out the singular points which would have been produced from the original information. It follows that if the solution to a propagating interface is formulated using a smoothing term according to curvature of the interface, the entropy condition is violated. However, if $X^{\epsilon}_{curvature}(t)$ is the solution using $F = 1 - \epsilon\kappa$, and $X_{constant}(t)$ is the solution using $F = 1$, the limit

$$\forall T, \lim_{\epsilon \to 0} X^{\epsilon}_{curvature}(T) = X_{constant}(T)$$

shows that the speed function with the smoothing term is actually the correct entropy satisfying solution when ε approaches 0. This limit is known as the viscous limit, and this entropy satisfying solution is known as the viscosity solution. As an aside, it is known as the viscosity solution because of the diffusive fluid viscosity term used in hyperbolic conservation laws.

5. Hamilton-Jacobi Equations

Recall both the initial value formulation and the boundary value formulation: if the speed function F depends only on its position in space and the first derivatives of ϕ, then both equations can be reformulated as the general Hamilton-Jacobi equation

$$\alpha u_t + H(Du, x) = 0 \quad ,$$

where H is the Hamiltonian, *Du* represents all the partials of u (i.e. u_x), *x* represents the position, and α is either 0 or 1. For example, substituting the values $\alpha = 0$, and $H = F|\Delta u| - 1$, produce the Eikonal equation: $F|\Delta u| = 1$. Furthermore, if the Hamilton-Jacobi equation was written with a smoothing term, the smoothed version of the interface propagation that was mentioned earlier is achieved:

$$\alpha u_t + H(Du, x) = \epsilon \Delta u \quad .$$

In this case, intuition proves correct – as ε approaches 0, the viscosity solution is reached (for further proof refer to [1] pp 31-32).

In this way, the Hamilton-Jacobi equation can be used to rewrite the original formulations. Indeed, this equation is more desirable for purposes of numerical approximation. Using the Hamilton-Jacobi equations, an entropy-satisfying scheme which properly uses upwind values can be formulated. For these reasons, this equation

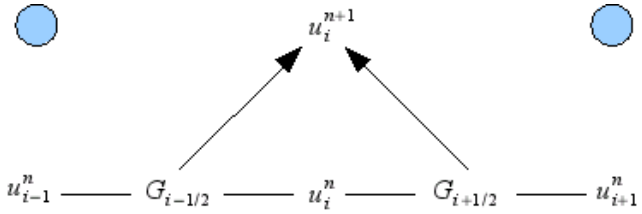

Figure 6: The Flux function can be used to compute the solution to a wave equation at the next time step.

will be used to form the general algorithmic solution to the initial value problem and the boundary value problem.

6. Flux Functions

A flux function, G, can be used to produce the solution to a hyperbolic conservation law,

$$u_t + (\mathrm{G}(u))_x = 0 \quad ,$$

for the next time step as shown in Figure 6. If a numerical approximation to a flux function produces the entropy satisfying solution, a scheme using that flux function will parallel solving the interface propagation problems initially formulated.

Given two points, u_1 and u_2, the numerical flux function

$$\mathrm{g}_{EO}(u_1, u_2) = (\max(u_1, 0)^2 + \min(u_2, 0)^2) \qquad \text{Equation A}$$

will approximate the flux function G. There are many numerical flux functions that will give a flux function approximation, and this is one such function. It was originally developed by Engquist and Osher[3].

In order to see why this flux function, let alone any other flux function is useful, some more work must be done. A scheme to solve first order wave equations is in conservation form if

$$\mathrm{g}(u_{i-1/2}, u_i) \text{ and } \mathrm{g}(u_i, u_{i+1/2})$$

can be used to approximate

$$\mathrm{G}_{i-1/2} \text{ and } \mathrm{G}_{i+1/2}$$

such that

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{-\mathrm{G}_{i+1/2} - \mathrm{G}_{i-1/2}}{\Delta x} \quad . \qquad \text{Equation B}$$

So, if a scheme is in conservation form, the solution to the wave equation for the next time can be approximated easily. An additional property is required – the scheme to be used for propagating interfaces must satisfy the entropy condition. In order to ensure this, the scheme must be monotonic. That is, a scheme

$$u_i^{n+1} = \mathrm{W}(u_{i-1}^n, u_i^n, u_{i+1}^n)$$

must be a non-decreasing function of all its arguments. The flux function g_{EO} comes from a scheme which is in conservation form, and monotonic. This is why it is particularly desirable. With the formulation of the flux function g_{EO}, a numerical approximation for time values in the interface propagation problem can now be given.

7. A General Algorithm for Convex Speed Functions

Recall that the initial value formulation and boundary value formulation can be written as a general Hamilton-Jacobi equation. Here it is formulated in a slightly different manor:

$$\alpha U_t + \mathrm{H}(U_x) = 0 \quad ,$$

where U is the solution to the Hamilton-Jacobi equation. The problem can be reformulated by letting $u = U_x$, where u is the solution to a hyperbolic conservation law, and then differentiating:

$$u_t + (\mathrm{H}(u))_x = 0 \quad .$$

This equation is the same hyperbolic conservation law that was seen earlier, and it can be approximated by using a numerical flux function. More specifically,

$$\mathrm{H}(U_i^n) \approx \mathrm{g}(u_{i-1/2}, u_{i+1/2})$$

can be used. Now a substitution can be done from Equation B to produce a one-dimensional scheme:

$$\alpha U_i^{n+1} = U_i^n - \Delta t\, \mathrm{g}\left(U_i^n - \frac{U_{i-1}^n}{\Delta x}, U_{i+1}^n - \frac{U_i^n}{\Delta x} \right) \quad ,$$

or in two dimensions,

$$U_{i,j}^{n+1} = U_{i,j}^n - \Delta t\, \mathrm{g}\left(U_{i,j}^n - \frac{U_{i-1,j}^n}{\Delta x}, U_{i+1,j}^n - \frac{U_{i,j}^n}{\Delta x}, U_{i,j}^n - \frac{U_{i,j-1}^n}{\Delta y}, U_{i,j+1}^n - \frac{U_{i,j}^n}{\Delta y} \right) \quad .$$

Using Equation A with the scheme that has been formulated, a basic level set method can be created. In one dimension,

$$\phi_i^{n+1} = \phi_i^n - \Delta t\, (\max(D_i^{-x}, 0)^2 + \min(D_i^{+x}, 0)^2)^{1/2}$$

can be used, where the forward and backward difference operators

$$\text{Backward: } D_i^{-x} = \frac{u_i^n - u_{i-1}^n}{\Delta x}$$

$$\text{Forward: } D_i^{+x} = \frac{u_{i+1}^n - u_i^n}{\Delta x}$$

are substituted for u_1 and u_2 in the numerical flux function. It follows that the first order equation for a convex speed function is

$$\phi_{ij}^{n+1} = \phi_{ij}^n - \Delta t \left(\max(F_{ij}, 0) \nabla^+ + \min(F_{ij}, 0) \nabla^- \right) \quad ,$$

given

$$\nabla^+ = \max(D_{ij}^{-x}, 0)^2 + \min(D_{ij}^{+x}, 0)^2 + \max(D_{ij}^{-y}, 0)^2 + \min(D_{ij}^{+y}, 0)^2 \quad ,$$

and

$$\nabla^- = \max(D_{ij}^{+x}, 0)^2 + \min(D_{ij}^{-x}, 0)^2 + \max(D_{ij}^{+y}, 0)^2 + \min(D_{ij}^{-y}, 0)^2 \quad .$$

This equation uses the speed function F, difference operators, and the time step Δt in order to produce the ϕ value for the next time. Higher order schemes can be built in order to increase accuracy, but they will not be discussed here.

7.1 The CFL Condition

The Courant-Friedrichs-Levy condition states that this algorithm will only be stable if there is a restriction placed on the time step with respect to the space step. In this case,

$$\max_{\Omega} F\,\Delta t \leqslant \Delta x$$

shows how the speed and time steps are constrained by the grid spacing over the entire grid. The narrow band level set method can add some flexibility to this constraint, but the principle is the same: the amount of error in the level set method approximation will become very large if care is not taken with respect to time and space steps.

8. Approximating Curvature

If a given interface needs to be propagated according to curvature, the simpler algorithm given before will not suffice. Instead, a new formulation comes next that requires the calculation of curvature of the zero level set is needed. The curvature can be approximated using

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

9. A General Level Set Method

The general level set method described here combines three speed functions: one for the expansion speed desired, one for speed accumulated from curvature and one which moves the interface according to underlying properties of the surface on which the interface lies. This final speed function (in $\mathbb{R}^2$) is given as the vector valued function

$$(u, v) = \vec{U}(x, y, t) \quad ,$$

which gives the underlying push at a given point at a given time. These three speed functions, with the original formulation of a level set method, give

$$\phi_{ij}^{n+1} = \phi_{ij}^{n} + \Delta t\Big(-(max(F_{0\mathrm{ij}}, 0)\nabla^{+} + min(F_{0\mathrm{ij}}, 0)\nabla^{-}) - \Big(max(u_{ij}^{n}, 0)\,D_{ij}^{-x} + min(u_{ij}^{n}, 0)\,D_{ij}^{+x} + max(v_{ij}^{n}, 0)\,D_{ij}^{-y} + min(v_{ij}^{n}, 0)\,D_{ij}^{+y}\Big) + \epsilon\,K_{i,j}^{n}\left((D_{ij}^{0*x})^2 + (D_{ij}^{0*y})^2\right)^{\frac{1}{2}}\Big) \quad ,$$

where $F_{0\mathrm{ij}}$ is the desired expansion speed, u and v come from the underlying speed function and ε describes how much this interface is driven by curvature.

10. Algorithm for the Fast Marching Method

Given the boundary value formulation in $\mathbb{R}^2$, a solution for the Eikonal equation

$$|\nabla \mathrm{T}|\,\mathrm{F}(x, y) = 1$$

must be obtained. By using ∇^{+} , the quadratic

$$\left(\max(D_{ij}^{-x}T, 0)^2 + \min(D_{ij}^{+x}T, 0)^2 + \max(D_{ij}^{-y}T, 0)^2 + \min(D_{ij}^{+y}T, 0)^2\right)^{1/2} = \frac{1}{F_{ij}}$$

is formulated. Sethian notes that a slightly different formulation is more convenient:

$$\left(\max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0)^2 + \max(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0)^2\right)^{1/2} = \frac{1}{F_{ij}} \quad .$$ Equation C

This quadratic is used in the fast marching method to solve for the next time values. It is apparent already that there is no time step involved here – the fast marching method does not need one. However, without a time step a different method of propagation is required. Where the level set method propagates according to time and space step, the fast marching method propagates according to the smallest known time value, some T(x, y).

In order to achieve this, the initial interface points (here, at time $T = 0$) are added to a min-heap structure. During each iteration, the grid point with the earliest time in the min-heap is put into the set of known grid points, and a fringe of trial values adjacent to that grid point receive updated time values and are added to the min-heap. Propagating this way is advantage to algorithm efficiency and will be outlined later in the run time analysis.

10.1 Solving the Quadratic

After building a min-heap, the final step in writing the fast marching method is creating a system to solve this particular quadratic formulation. There are two ways of thinking of this. This quadratic was formulated from a flux function which is intended to both upwind properly and satisfy the entropy condition. To upwind properly over a discrete grid, for a given point P, intuitively it seems that only known values adjacent to P should be used. This is shown in Figure 7. Trial values may not be correct, and that is exactly why an upwind scheme was formulated in the first place.

After expanding the quadratic equation given in Equation C for $\mathbb{R}^2$,

$$\max\left(t-\mathrm{T}(i-1,j),t-\mathrm{T}(i+1,j),0\right)^2+\max\left(t-\mathrm{T}(i,j-1),t-\mathrm{T}(i,j+1),0\right)^2=\frac{1}{F_{ij}^2}$$

is found, where $t = \mathrm{T}(i, j)$. In order to formulate the correct mathematical solution to this, two pieces of information are required.

First, monotonicity is a key property of this algorithm. That is, when using the fast marching method, whether expanding or contracting, time values are always either increasing or staying the same. Additionally, the initial interface points will be set to $T = 0$. Knowing this, it is clear that time values will always be greater than or equal to 0.

Second, if a grid point's time value is only known to be accurate (aside from approximation inaccuracy) once it is put into the known set. If it is in the fringe of trial points, or outside of the fringe and known set all together, its time value is not known to be accurate and is infinity for all intents and purposes. Keeping the actual time value is

∞	∞	∞	∞	∞
∞	1+√2/2	1	t	∞
∞	1	0	1	∞
∞	∞	1	∞	∞
∞	∞	∞	∞	∞

Figure 7a: Using upwind values to calculate t: the cells containing ∞ are not used, as their values are not yet known.

∞	∞	∞	∞	∞
∞	1+√2/2	1	1+√2/2	∞
∞	1	0	1	∞
∞	∞	1	∞	∞
∞	∞	∞	∞	∞

Figure 7b: After solving the quadratic formulated for Figure 7a. Note the propagation error in the first order scheme – the value for t should have been √2.

important for trial points, as these time values are used in picking the next minimum from the min-heap, but formulating the values as infinity may allow for a simpler approach to solving the quadratic equation.

Using Figure 8 as an example, taking the zero term into account as a non-negative condition imposed on the final outcome, the quadratic is reformulated as

$$\max(t-\infty, t-1)^2+\max(t-1, t-\infty)^2=\frac{1}{F_{ij}^2} \quad .$$

With this formulation, picking the maximum becomes trivial, and

$$(t-1)^2+(t-1)^2=\frac{1}{F_{ij}^2}$$

then gives

$$2F_{ij}^2t^2-4F_{ij}^2t+F_{ij}^2=0 \quad ,$$

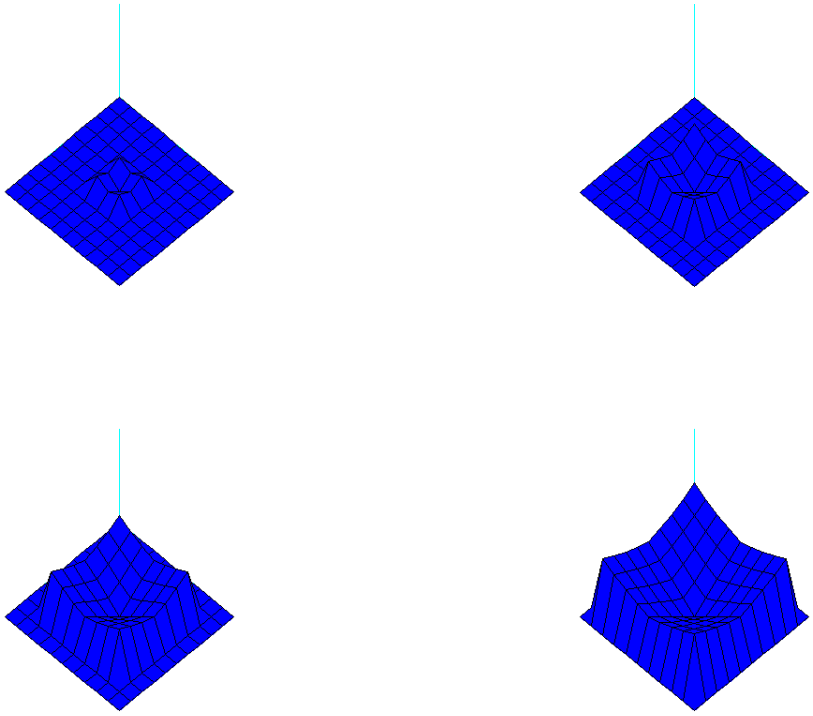

Figure 9: The fast marching method does not propagate accurately using a first order scheme. These illustrations show that the solution does not quite have the desired cone shape.

which can be solved by the quadratic formula to give a solution as long as the solution is not less than 0. Also, the solution to the quadratic that is desired is the larger of the two solutions[2]; more precisely, the quadratic is solve by the equation

$$F = \frac{-b + \sqrt{(b^2 - 4ac)}}{2a} \quad .$$

10.2 Higher Order Approximations

While higher order versions of the fast marching method are not discussed here, the first order version clearly shows an incorrect value when completing a unit propagation from a point. This propagation should compute the Euclidean distance transform, but as shown in Figure 8, the values on the diagonal points adjacent to the initial point are $1+\sqrt{2}/2$, where a value of $\sqrt{2}$ is desired. Moreover, Figure 9 shows the result of propagating with unit speed from a single initial point with value $T = 0$. This interface propagation should produce the distance transform: viewed in 3 dimensions it should be a cone. The results are nearly correct, but not entirely accurate. For more information on error bounds and higher order schemes, see [1], chapter 12.

11. The Narrow Band Level Set Method

The narrow band level set method is a general propagation algorithm suitable for any problem using the initial value formulation

$$\phi_t + F|\nabla \phi| = 0 \quad .$$

This method can be thought of as a way to bound the brute force level set method, sometimes known as the “full matrix approach” since the discrete solution to the problem for an entire matrix would be computed for each time step required. While this approach can be made to run in parallel in a fairly straightforward fashion, parallel processing is not always feasible. Thus, for each level set, the problem will bounded by a small 'narrow band' which will surround the interface at any given time. That is, the level set function will only be computed inside of the band, capturing the behavior of the zero level set over time without having to compute the entire matrix of solutions.

There are a few advantages to doing it this way. Certainly there is a speed increase; now only a few of the values are being calculated instead of the entire matrix. However, there are more subtle and arguably much better reasons for using this approach. Using the narrow band level set method, the CFL condition is much easier to deal with. The narrow band can be adjusted according to how many grid or time steps the interface will need to move. This is especially advantageous in a interface motion such as curvature flows, where the speed is based on curvature. Furthermore, calculating a speed function which must take into account more complicated factors can be difficult, and limiting calculation of F to the narrow band can bring further improvement in speed.

When implementing the narrow band level set method, there are a few things to keep in mind. First, Sethian suggests that 6 grid spaces on either side of the zero level set is usually an appropriate width to use for k (see [1], page 85). Next, there must be some iteration condition that will be specific to the problem being solved. As the approach works with time and space, it is most likely that at least one of the two will appear in the condition to end the algorithm (e.g. Stop when a value t = 50 is achieved). Also, despite

advantages mentioned earlier, the CFL condition must still be kept in mind when choosing the band width.

11.1 Runtime Analysis

All of the values for the entire narrow band will need to be calculated for each time step. If the grid the interface is being propagated over is N x N, and the narrow band is of width k, the band will have approximately N grid points in it at any given time. This gives N grid points propagated over for all k; this operation must happen for each time step, which will be considered to be N. Therefore, the worst case runtime will be $O(N^2k)$.

11.2 Pseudocode

```
Populate the matrix M with φ values with initial level set data

k ← width of narrow band
M ← matrix of φ values
t ← initial time
while (iteration condition not met)
      B ← Ø                                  // narrow band list
      for each grid cell where dist(C, zero level set) < k
            insert C into B
      end for
      for each grid cell where dist(C, zero level set) = k
            φ value of C ← infinity          // land mines
      end for
      // Must be cleared up – this would greatly affect run time if
      // it's done in a O(M) search

      mine_hit ← false
      while (mine_hit = false AND iteration condition not met)
            for each grid cell C in B
                  temp ← solution to LSM(C, t + 1)
                  if (temp = 0 AND φ value of C = infinity)
                        mine_hit ← true
                  else
                        φ value of C ← temp
                  end if
                  t ← t + 1
            end for
      end while
end while


LSM(C, time):
```

$$\phi_{ij}^{n+1} = \phi_{ij}^{n} + \Delta t \left(-(max(F_{0ij}, 0)\nabla^{+} + min(F_{0ij}, 0)\nabla^{-}) - \left(max(u_{ij}^{n}, 0) D_{ij}^{-x} + min(u_{ij}^{n}, 0) D_{ij}^{+x} + max(v_{ij}^{n}, 0) D_{ij}^{-y} + min(v_{ij}^{n}, 0) D_{ij}^{+y} \right) + \epsilon K_{i,j}^{n} \left((D_{ij}^{0*x})^2 + (D_{ij}^{0*y})^2 \right)^{\frac{1}{2}} \right),$$

```
where
F0ij is the propagation speed at ij
F = -ek is the dependence of propagation speed on curvature
```

12. The Fast Marching Method

The fast marching method is appropriate only for the boundary value problem. The basic principle underlying this method is that the boundary value problem

$$|\nabla T| F = 1$$

contains a front that is always expanding or contracting. As such, the interface is always propagated to the next smallest T value in the grid, since the interface will always reach the unknown node nearest to the known fringe next. Following directly from this idea, a min-heap structure can be used as the basic building block of the fast marching method.

Like the narrow band level set method, there must be some iteration condition that will be specific to the problem being solved. This will most likely be when a certain time value or location is reached. Also, for efficiency sake, the min-heap should keep not only the time values, but a pointer to which cell that T value came from.

12.1 Runtime Analysis

If the grid is N x N, N^2 time value computations will be made, with log(N) heap operations to get the grid cells for each N^2 time values being computed. Therefore, a runtime of $O(N^2 \log(N))$ is computed.

12.2 Comparison with the Narrow Band Level Set Method

When comparing against the narrow band level set method, it must be noted that there is no time step in the fast marching method. Thus, the fast marching method doesn't require the satisfaction of the CFL condition, whereas the narrow band level set method does, and may be much slower for fronts with a large value of F. Additionally, the amount of heap operations per time calculation are only log(N) in the worst case, and for most practical problems will probably be closer to O(1).

12.3 Pseudocode

```
Initialize all nodes to infinity
Known ← all grid point with known solutions
Trial ← all unknown nodes adjacent to Known nodes
Insert all nodes in Trial into the min-heap

while (iteration condition not met)
      node A ← root of the min-heap
      add A to Known
      remove A from Trial
      perform down heap
      for each neighbour B of A
            if (Time(B) is infinity)
                  Add B to Trial
                  Perform up heap
                  Time(B) = Quadratic2d(B)
            end if
      end for
end while

Quadratic2d (A):
F ← Speed function from the boundary value problem
```

```
A ← Node to solve for

UseRow ← (Left(A) is in Known) OR (Right(A) is in Known)
UseCol ← (Above(A) is in Known) OR (Below(A) is in Known)
RowTerm ← min(Left(A), Right(A))
ColTerm ← min(Above(A), Below(A))

if ((UseRow AND NOT(UseCol)) OR (UseCol AND NOT(UseRow)))
      Term ← max(RowTerm, ColTerm)
      a ← 1
      b ← 2 * Term
      c ← Term * Term - (1 / (F*F));
else
      a ← 2;
      b ← (2*ColTerm) + (2*RowTerm);
      c ← RowTerm * RowTerm + ColTerm * ColTerm - (1 / (F*F));
end if

return max(QuadraticSolver(a, b, c))
```

13. Conclusions and Future Work

The complexities involved with understanding the workings of the level set method and fast marching method, and the robustness of the methods themselves do not translate into inefficient algorithms. When dealing with a propagation problem, a properly formulated combination of the fast marching method and narrow band level set method will most likely produce an appropriate solution.

In the following year, these methods will be used to solve image segmentation problems by propagating an initial interface according to properties of an image. The specific algorithms outlined here may not be accurate enough for other problems, but will probably be accurate enough for image segmentation problems.

References

[1] Sethian, J.A., *Level Set Methods and Fast Marching Methods*, Cambridge University Press, 1999.

[2] Enhancement, Extraction, and Visualization of 3D Volume Data, PhD thesis, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, Apr. 2003. Dissertations. No. 824 http://www.isy.liu.se/~qingfen/thesis

[3] Engquist, B., and Osher, S.J., Stable and Entropy-Satisfying Approximations for Transonic Flow Calculations, Math. Comp., 34, 45, 1980.