Fast and Accurate Redistancing by Directional Optimization

Matt Elsey Selim Esedoḡlu

September 6, 2012

Abstract

A fast and accurate algorithm for the reinitialization of the signed distance function in two and three spatial dimensions is presented. The algorithm has computational complexity $O(N \log N)$ for the reinitialization of N grid points. The order of accuracy of the reinitialization is demonstrated to depend primarily on the interpolation algorithm used. Bicubic interpolation is demonstrated to result in fourth-order accuracy for smooth interfaces. Simple numerical examples demonstrating the convergence and computational complexity of the reinitialization algorithm in two and three dimensions are presented as verification of the algorithm.

1 Introduction

The *signed distance function* $d_\Sigma(\boldsymbol{x})$ to a set $\Sigma \subset \mathbb{R}^n$ (possibly consisting of multiple connected components) returns the standard Euclidean distance to the boundary $\partial\Sigma$ of Σ with a sign bit indicating whether the point $\boldsymbol{x}$ is inside or outside of Σ:

$$d_\Sigma(\boldsymbol{x}) = \begin{cases} \inf_{\boldsymbol{y}\in\partial\Sigma} |\boldsymbol{x}-\boldsymbol{y}|, & \boldsymbol{x} \in \Sigma \\ -\inf_{\boldsymbol{y}\in\partial\Sigma} |\boldsymbol{x}-\boldsymbol{y}|, & \boldsymbol{x} \notin \Sigma \end{cases} \tag{1}$$

The signed distance function arises in a number of contexts in scientific computing. For example, it is well known that evolving interfaces via the level set method can cause profiles of the level set function to become too steep or too flat, leading to instabilities. Replacing the distorted level set function ϕ by the signed distance function to the zero-level set of ϕ occasionally can alleviate this problem. In addition, there are algorithms such as the signed distance function-based diffusion-generated curvature motion, proposed in [9] and developed in the multiphase setting in [6, 7, 8], that require the efficient and accurate construction of the signed distance function at every time step.

It is of particular interest to be able to accurately and efficiently compute $d_\Sigma(\boldsymbol{x})$ from an arbitrary level set representation $\phi(\boldsymbol{x})$ satisfying

$$\overline{\Sigma} = \{\boldsymbol{x} : \phi(\boldsymbol{x}) \geq 0\}, \tag{2}$$

which is termed the *reinitialization* [3] of $\phi(\boldsymbol{x})$. In this work, we present a fast ($O(N \log N)$ on N grid points) and arbitrarily accurate algorithm for reconstructing the signed distance function from a level set representation of Σ. Our implementation of the algorithm using bicubic interpolation obtains $O(\Delta x^4)$ accuracy for grid spacing Δx given smooth interfaces. The algorithm is closely related to Dijkstra's algorithm [5] for finding shortest paths in a graph, the control theoretic approach of Tsitsiklis [24], and to the fast marching method of Sethian [17]. Indeed, the latter two of these are equivalent in their most basic form. These algorithms are revisited in Section 2 to set the stage for the algorithm proposed here.

Intuitively, the proposed algorithm is based on the following ideas:

1. If a good estimate for the closest point on the interface $\partial\Sigma$ to a given grid point is known, we can exploit the fact that the distance to the interface is the length of a line segment connecting them, and optimize locally the coordinates of the closest point to obtain a very accurate value for the distance, while using interpolation to represent the level set function near the zero level set.
2. A good estimate for the closest point to each grid point can be obtained from an ordered update scheme on the fly (fast marching style), where the update information passed is the closest point on the interface to a neighboring grid cell whose distance to the interface has already been fixed by the algorithm. As usual, information from the *most trustworthy* neighbor is utilized.
3. The algorithm can be made arbitrarily high order simply by swapping the interpolation used in representing the level set function in a neighborhood of its zero level set, without having to deal with high order difference quotients.

Taken individually, ideas similar to these can be found in the literature. An interpolation-based optimization is proposed and implemented in [4], but only utilized near the interface; essentially, as a new way to initialize the standard fast marching algorithm. The idea of propagating closest point information is suggested in [22], but only for a limited class of interfaces: isolated points and piecewise linear interfaces. In this context, proper closest point information is sufficient to compute the *exact* distance to the interface without any need for interpolation or optimization.

The new algorithm proposed in this paper differs from previous algorithms in that the distance to the interface $\partial\Sigma$ is directly computed via *directional optimization*, described fully in Section 3, rather than by solving equations which depend on previously calculated values of d at nearby grid cells. Instead, the local information obtained from neighboring grid cells via the ordered update scheme is used only to provide the directional optimization routine with an initial search direction.

2 Previous Work

The signed distance function (1) arises as a solution to a simplified version of the eikonal equation, which is defined by

$$|\nabla u(\boldsymbol{x})| = g(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{3}$$

subject to the boundary condition $u(\boldsymbol{x}) = \phi(\boldsymbol{x})$, for $\boldsymbol{x} \in \Gamma \subset \Omega$. (1) arises when $g(\boldsymbol{x}) \equiv 1$, $\phi(\boldsymbol{x}) \equiv 0$, and $\Gamma = \partial\Sigma$.

There are two major approaches to reinitialization. First is the PDE-based approach pioneered in [14, 20] and improved in [2, 15], which advocates calculating the signed distance function $d(\boldsymbol{x})$ as the steady-state solution of

$$d_t = \operatorname{sgn}(\phi)(1 - |\nabla d|), \tag{4}$$

with initial condition $d(\boldsymbol{x}, 0) = \phi(\boldsymbol{x})$. Steady states to (4) have $|\nabla d| = 1$ almost everywhere, and maintain the restriction that $d(\boldsymbol{x}) > 0$ iff. $\phi(\boldsymbol{x}) > 0$. Solutions are numerically computed by discretizing (4) and numerically integrating until a steady-state is achieved. These schemes can converge quickly when $\phi \approx d$, as information is preserved far from the interface Γ. A major drawback of these schemes is that the entire function d is updated at each time step. This becomes very computationally demanding when many time steps are needed to achieve a steady state.

The second approach includes well-known algorithms such as Dijkstra's algorithm [5], Tsitsiklis's algorithm [24], and the fast marching method [17]. These approaches attempt to solve the boundary value problem

$$|\nabla d| = 1 \tag{5}$$

subject to the constraint $d(\boldsymbol{x}) = 0$ on Γ by propagating information outwards from the interface via an *ordered update scheme*. The basic structure of this scheme can be described as a three-part procedure:

1. Initialize $F = \emptyset$ to be the set of grid points for which $d(\boldsymbol{x})$ is *fixed*, and $T = \{\text{all grid points}\}$ to be the set of grid points whose value is *tentative*. Set $d(\boldsymbol{x}) = \text{sgn}(\phi(\boldsymbol{x})) \cdot \infty$ for all $\boldsymbol{x} \in T$.
2. For each grid point in a neighborhood of the interface, assign its signed distance $d(\boldsymbol{x})$ via some interpolation scheme designed to leave the interface unmoved. Move all of these points from T to F. Update the values $d(\boldsymbol{x})$ of all first-neighbors $\boldsymbol{x}$ to the interface points in F via `UPDATE`, leaving these points in T.
3. While T is non-empty, move the "minimal" element $\boldsymbol{y} \in T$, defined by $|d(\boldsymbol{y})| \leq |d(\boldsymbol{x})|$ for all $\boldsymbol{x} \in T$, to F, and update the values of all first-neighbors of $\boldsymbol{y}$ via `UPDATE`.

Assuming that `UPDATE` is an $O(1)$ (or even $O(\log N)$) process, the runtime of this class of algorithms is $O(N \log N)$ when a heap structure is used to manage ordering of the elements in T. The difference between Dijkstra's algorithm, the fast marching method, Tsitsiklis's algorithm, and the directional optimization method proposed here is in the details of `UPDATE`.

Dijkstra's algorithm was initially proposed for finding shortest paths in a graph, where edges between vertices are all assigned weights. If first-nearest neighbors on a uniform grid (four in 2D, six in 3D) are considered to have edges between them, with edge weight 1, Dijkstra's algorithm gives the ℓ_1 (taxicab) distance. Providing additional edge connectivity (with appropriate weighting) allows for better approximation of the standard Euclidean metric, albeit at the cost of additional work.

For the basic fast marching method, `UPDATE` consists of solving a quadratic equation related to a first-order accurate upwind discretization of (5). On a uniform grid, this update is exactly the same as that proposed by Tsitsiklis. These methods can be applied to the wider class of problem described by the eikonal equation, with arbitrary "speed" function $f(\boldsymbol{x}) = g^{-1}(\boldsymbol{x})$. A simple piecewise formula for the solution of this quadratic is presented in [25]. Higher-order versions of the fast marching method can be obtained by the use of more accurate approximations to the gradient. Second-order accurate versions are described and demonstrated in [18, 19]. The interpolation-based initialization proposed by Chopp [4] allows for even higher-order versions of the scheme, requiring larger stencils for the high-order accurate discretization of derivatives.

The fast sweeping method [23, 25] uses a similar update to the fast marching method but takes advantage of the fact that there are a limited number of directions in which information can propagate. In this algorithm, one simply "sweeps" over the grid in appropriate patterns a fixed number of times (the number depends on the dimensionality of the problem and, for general Hamilton–Jacobi equations, the speed function $f(\boldsymbol{x})$), resulting in an algorithm which has optimal complexity scaling $O(N)$. Recently, Benamou et al. [1] have obtained a second-order accurate version of fast sweeping while using a simple 9-point stencil. However, the authors state that they do not expect their approach to be generalizable to higher-order methods. We refer the reader to Chapter 7 of the book of Osher and Fedkiw [13] for a more complete history of reinitialization algorithms.

All of the algorithms mentioned here are described in terms of the reinitialization problem, but can be easily extended to more general eikonal equations (3) or even to a class of static Hamilton–Jacobi equations — and, indeed, in many cases were initially proposed for this one of these wider classes of problems. The drawback to such generality is that the critical property of (5), namely, that all characteristics are rays, goes unused in all of these algorithms. In the second class of algorithms, the `UPDATE` at $\boldsymbol{x}$ is always determined by the current values of $d(\boldsymbol{y})$ for $\boldsymbol{y}$ near $\boldsymbol{x}$.

In contrast, the directional optimization algorithm proposed here will search on the interface for the closest point $\boldsymbol{x_0} \in \Gamma$ to $\boldsymbol{x}$ to compute $d(\boldsymbol{x}) = \text{sgn}(\phi(\boldsymbol{x}))|\boldsymbol{x_0} - \boldsymbol{x}|$. The near-interface search property gives two major advantages to this method over the fast marching methods:

First, the directional optimization method described here gives one additional order of accuracy over the standard fast marching method with the same computational stencil. In addition, in the standard fast marching method, the local update causes errors in computed values of $d(\boldsymbol{x})$ to accumulate and affect grid cells whose update is computed from these incorrectly set values. The directional optimization algorithm instead *always* looks back to the grid cells about the interface Γ to calculate the updated approximation of signed distance, preventing such accumulation of error.

3 Algorithm

For simplicity, we describe our algorithm on an $n \times n$ computational grid discretizing the domain $[0,1)^2$, with periodic boundary conditions. We choose $\Delta x = \Delta y = 1/n$, and choose $x_j = j\Delta x$, $y_i = i\Delta y$, with $i, j = 0, 1, \ldots, n-1$ and denote the discretized approximation to $d(x_j, y_i)$ by d_{ij}. Index arithmetic is performed mod$n$, so that $(n-1)+1 = 0$ and $0-1 = n-1$, as is standard with periodic boundary conditions. The description here can easily be extended to non-square domains with $\Delta x \neq \Delta y$. Later we will also discuss an extension of this algorithm into three spatial dimensions.

For the `UPDATE` of the directional optimization algorithm, we will also maintain an auxiliary function which will be updated to the *closest point* function. The closest point function $\boldsymbol{C}(\boldsymbol{x}) = \boldsymbol{x_0}$ returns the closest point $\boldsymbol{x_0} \in \Gamma$ on the interface Γ to $\boldsymbol{x}$. $\boldsymbol{C}(\boldsymbol{x})$ captures the magnitude but not the sign of $d(\boldsymbol{x})$, as $d(\boldsymbol{x}) = \text{sgn}(\phi(\boldsymbol{x}))|\boldsymbol{x} - \boldsymbol{C}(\boldsymbol{x})|$. This function is of use in some applications. For example, Ruuth and Merriman [16] utilize it as part of a novel technique for evolving PDEs on surfaces.

As discussed in Section 2, the redistancing algorithm can be divided into three steps: First, fix d_{ij} and $\boldsymbol{C}_{ij}$ within a single grid cell of the interface via some interpolation scheme, and add these grid cells to F. Then, update tentative values of d and $\boldsymbol{C}$ for all first neighbors of all $x_{ij} \in F$ and set the values of all remaining cells to $d(\boldsymbol{x}) = \text{sgn}(\phi(\boldsymbol{x})) \cdot \infty$ and $\boldsymbol{C}(\boldsymbol{x}) = \boldsymbol{\infty}$. Add all these grid cells to T. Finally, while T is non-empty, choose the element $\boldsymbol{y} \in T$ for which $|d(\boldsymbol{y})|$ is minimal, move it to F, and update the tentative values of all first neighbors to $\boldsymbol{y}$.

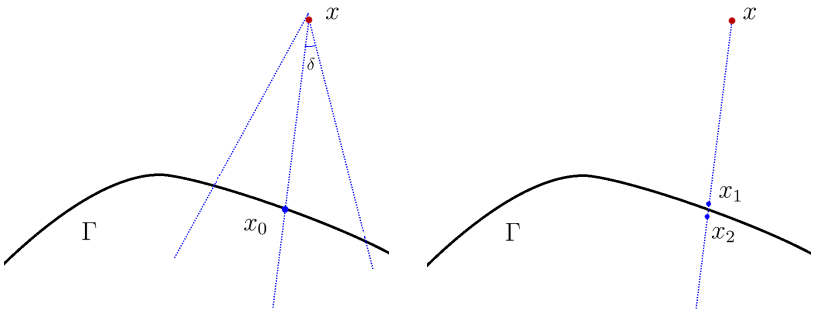
All that remains is to describe how the grid cells adjacent to the interface are detected and how their values are set, and how to update the value $d(\boldsymbol{x})$ for grid location $\boldsymbol{x}$ adjacent to some newly-fixed grid location $\boldsymbol{y}$. These will be carefully described next.

3.1 Updating $d(\boldsymbol{x})$

Here we describe how to update the signed distance function $d(\boldsymbol{x})$ and closest point function $\boldsymbol{C}(\boldsymbol{x})$ which tracks the closest known point on the interface to a given point $\boldsymbol{x}$ given an initial search location $\boldsymbol{x_0}$. In the context of the previous discussion, $\boldsymbol{x}$ is a first neighbor of the element $\boldsymbol{y}$ whose value has just been fixed. The initial search location $\boldsymbol{x_0}$ is chosen to be $\boldsymbol{C}(\boldsymbol{y})$, the closest point to $\boldsymbol{x_0}$ on the interface to $\boldsymbol{y}$. We call our update "directional optimization," as it locates the the closest point to $\boldsymbol{x}$ on the interface near $\boldsymbol{x_0}$ by optimizing the distance to the interface as a function of the angle θ formed by the vector $\boldsymbol{x_0} - \boldsymbol{x}$ with the positive x-axis. Along each direction θ searched, the algorithm performs a line search to locate the intersection of the interface with the half-line $\boldsymbol{x} + r\boldsymbol{\theta}$ for $r > 0$, where $\boldsymbol{\theta} = (\cos\theta, \sin\theta)$. Details of each of the necessary routines (directional search, line search, and interpolation) are provided subsequently.

3.1.1 Directional Search

The directional search algorithm takes as input the point $\boldsymbol{x}$ for which the distance to the interface is to be computed and a point $\boldsymbol{x_0}$ which is expected to lie near the interface. Its objective is to search over directions within an angle δ of the vector $(\boldsymbol{x_0} - \boldsymbol{x})$ emanating from $\boldsymbol{x}$ for the



(a) (b)

Figure 1: (a) The outer directional optimization step performs a directional search near $(\boldsymbol{x_0} - \boldsymbol{x})$ for the closest point $\boldsymbol{y} \in \Gamma$ to $\boldsymbol{x}$. (b) The line search sub-algorithm locates two points near the interface then performs a linear interpolation between them to location the intersection of Γ with the search line. In the case that $|\boldsymbol{x} - \boldsymbol{x_0}| \leq \frac{5}{4}\Delta x$, we use $\boldsymbol{x}$ and $\boldsymbol{x_0}$ for the linear interpolation.

closest point on the interface to $\boldsymbol{x}$ nearby $\boldsymbol{x_0}$. See Figure 1(a). This computed signed distance is compared to the current estimate for $d(\boldsymbol{x})$ and is accepted if smaller in magnitude than the current estimate. The directional search is first described in two dimensions. Later, the extension into three spatial dimensions will be described.

1. Choose δ to be a nonincreasing function of $|\boldsymbol{x_0} - \boldsymbol{x}|$. We make the choice

$$\delta = \begin{cases} \operatorname{asin}\left(\frac{\Delta x}{|\boldsymbol{x}-\boldsymbol{x_0}|}\right), & \text{if } |\boldsymbol{x} - \boldsymbol{x_0}| > \Delta x \\ \frac{\pi}{2} & \text{otherwise.} \end{cases}$$

2. For M iterations, do the following:
   (a) Define R_δ to be the 2×2 matrix which rotates input vectors by δ radians counter-clockwise, and define $\boldsymbol{x_{\pm\delta}} = \boldsymbol{x} + R_\delta\,(\boldsymbol{x_0} - \boldsymbol{x})$.
   (b) Apply the line search algorithm to obtain $\boldsymbol{y_0}$, and $\boldsymbol{y_\pm}$ from $\boldsymbol{x_0}$ and $\boldsymbol{x_{\pm\delta}}$, respectively.
   (c) While $\min(|\boldsymbol{y_{+\delta}} - \boldsymbol{x}|, |\boldsymbol{y_{-\delta}} - \boldsymbol{x}|) < |\boldsymbol{y_0} - \boldsymbol{x}|$, redefine $\boldsymbol{y_0}$ to be the closer of $\boldsymbol{y_{+\delta}}$ or $\boldsymbol{y_{-\delta}}$ to $\boldsymbol{x}$, and return to step (2a).
   (d) Perform a Newton step to obtain a new search direction:
   - Define $d_{(\cdot)} = |\boldsymbol{y}_{(\cdot)} - \boldsymbol{x}|$ for $(\cdot) \in \{\pm\delta, 0\}$, and compute

$$\hat{\delta} = \frac{-\delta}{2}\frac{d_+ - d_-}{d_+ - 2d_0 + d_-} \tag{6}$$

   - Let $\boldsymbol{x^*} = \boldsymbol{x} + R_{\hat{\delta}}(\boldsymbol{x_0} - \boldsymbol{x})$ and apply line search to obtain $\boldsymbol{y^*}$.
   - If $|\boldsymbol{y^*} - \boldsymbol{y_0}| < .01\Delta x^p$, where p is the order of accuracy of the interpolation routine (e.g. $p = 3$ for bi- or tricubic interpolation), continue to 3.
     Else, if $|\boldsymbol{y^*} - \boldsymbol{x}| < |d_0|$, set $\boldsymbol{x_0} = \boldsymbol{y^*}$.
     Otherwise, set $\boldsymbol{x_0} = \boldsymbol{y_0}$.

   (e) Reduce $\delta \to \delta/2$.

3. Return $\boldsymbol{x_0}$ as the candidate closest point to $\boldsymbol{x}$.

If $|\boldsymbol{x_0} - \boldsymbol{x}| < |\boldsymbol{C}(\boldsymbol{x}) - \boldsymbol{x}|$, set $\boldsymbol{C}(\boldsymbol{x}) = \boldsymbol{x_0}$ and $d(\boldsymbol{x}) = \text{sgn}(\phi(\boldsymbol{x}))|\boldsymbol{x_0} - \boldsymbol{x}|$. We observe that choosing $M = 5$ is sufficient for all of our numerical tests to obtain the expected accuracy.

In three dimensions, the directional optimization step of the algorithm becomes slightly more complicated. However, the computational complexity of the algorithm as a whole remains $O(N \log N)$. We propose to use a five-point search stencil to replace the three-point stencil used in two dimensions. In step (1) of the directional search algorithm described above, we now arbitrarily choose two vectors, $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$, mutually orthogonal to each other and to $\boldsymbol{x_0} - \boldsymbol{x}$, and define $\boldsymbol{x_{\pm\delta,0}}$ and $\boldsymbol{x_{0,\pm\delta}}$ to be the vectors rotated $\pm\delta$ from $\boldsymbol{x_0} - \boldsymbol{x}$ towards $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$, respectively. We neglect the cross derivatives in the three-dimensional version of (6) to limit the number of line searches needed, and obtain new angles $\hat{\delta}_{\boldsymbol{\xi}}$ and $\hat{\delta}_{\boldsymbol{\zeta}}$ independently. The update is chosen as $\boldsymbol{x^*} = \boldsymbol{x} + \frac{1}{2}\left(R_{\hat{\delta}_{\boldsymbol{\xi}}} + R_{\hat{\delta}_{\boldsymbol{\zeta}}}\right)(\boldsymbol{x_0} - \boldsymbol{x})$ to preserve the desired property that $\boldsymbol{x^*} = \boldsymbol{x_0}$ if $\hat{\delta}_{\boldsymbol{\xi}} = \hat{\delta}_{\boldsymbol{\zeta}} = 0$. Numerical tests indicate that this extension to three dimensions maintains the expected performance of the algorithm.

3.1.2 Line Search

The line search step takes as input the point $\boldsymbol{x}$ for which distance to the interface is to be computed and a point $\boldsymbol{x_0}$ which is expected to lie near the interface. This sub-algorithm searches for the point $\boldsymbol{y}$ near $\boldsymbol{x_0}$ which lies on the interface along the line defined by $\boldsymbol{x}$ and $\boldsymbol{x_0}$. We consider three cases:

1. If $|d(\boldsymbol{x_0})| > 2\Delta x$, then the input $\boldsymbol{x_0}$ is not near the interface, and $\boldsymbol{y} = \boldsymbol{\infty}$ is returned.
2. If $|\boldsymbol{x} - \boldsymbol{x_0}| < \frac{5}{4}\Delta x$, set $\boldsymbol{y} = \boldsymbol{x} - d(\boldsymbol{x})(\boldsymbol{x_0} - \boldsymbol{x})/(d(\boldsymbol{x_0}) - d(\boldsymbol{x}))$, so that the point $(\boldsymbol{y}, 0)$ lies on the linear interpolation of d along the line defined by $\boldsymbol{x}$ and $\boldsymbol{x_0}$.
3. Otherwise, $\boldsymbol{x}$ is far from the interface, so we locate a point $\boldsymbol{x_1}$ on the opposite side of the interface from $\boldsymbol{x_0}$ (simply by stepping along this line in the appropriate direction), and use the linear interpolation of d obtained from $\boldsymbol{x_0}$ and $\boldsymbol{x_1}$ to obtain $\boldsymbol{y} = d(\boldsymbol{x_1})/(d(\boldsymbol{x_1}) - d(\boldsymbol{x_0}))\boldsymbol{x_0} - d(\boldsymbol{x_0})/(d(\boldsymbol{x_1}) - d(\boldsymbol{x_0}))\boldsymbol{x_1}$.

The interpolation scheme described subsequently is used to evaluate d at the desired locations.

3.1.3 Interpolation

To estimate $d(\boldsymbol{x})$ away from grid points, we must perform some type of interpolation over the values of d_{ij} at nearby grid points. We have implemented biquadratic and bicubic interpolation, obtaining $O(\Delta x^3)$ and $O(\Delta x^4)$ accuracy, respectively, for the computation of the signed distance function to a smooth interface. These numerical results suggest that the interpolation routine sets the order of accuracy for the algorithm as a whole.

3.2 Setting Boundary Values

To initialize the code, we first set $d(x) = \phi(x)$. Given the choice of an $O(\Delta x^p)$ interpolation algorithm, let $I = \{x_{ij} : x_{ij} \text{ is within } p-1 \text{ grid cells of } \Gamma\}$. The grid cells $x_{ij} \in I$ satisfy the property,

$$\text{sgn}(d(x_{ij})) \neq \text{sgn}(d(x_{i+r,j+q}), \tag{7}$$

for some $|r|, |q| \leq p - 1$. Perform the following three steps:

1. Estimate the closest point to $\boldsymbol{x}$ by

$$\boldsymbol{C}(\boldsymbol{x}) = \boldsymbol{x} - \nabla d(\boldsymbol{x}) \frac{d(\boldsymbol{x}))}{|\nabla d(\boldsymbol{x})|}, \tag{8}$$

for each $\boldsymbol{x} \in I$.

2. Perform `UPDATE` for each $\boldsymbol{x} \in I$ to refine the initial guesses for $d(\boldsymbol{x})$ and $\boldsymbol{C}(\boldsymbol{x})$. Store the updated values as $\hat{d}(\boldsymbol{x})$ and $\hat{\boldsymbol{C}}(\boldsymbol{x})$ (so that these updated values do not enter the calculation of `UPDATE` for any other $\boldsymbol{x} \in I$.
3. Batch update all these values at once: Set $\boldsymbol{C}(\boldsymbol{x}) = \hat{\boldsymbol{C}}(\boldsymbol{x})$ and $d(\boldsymbol{x}) = \hat{d}(\boldsymbol{x})$ for all $\boldsymbol{x} \in I$.

3.3 Extension to Eikonal Equations

In this work, we restrict our attention to the signed distance function. It may be possible to extend the algorithm proposed here to the more general eikonal equation (3). The proposed algorithm locates the closest point $\boldsymbol{C}(\boldsymbol{x})$ by optimizing over directions and performing line searches along rays to locate the interface in a given direction. One could in principle use a shooting method-type approach by retaining the optimization over directions and solving initial value problems to locate the interface given $\boldsymbol{x}$ and the initial direction to "shoot in" from $\boldsymbol{x}$. The ordered update scheme would propagate the initial search direction, in analogy to the propagation of closest point information in the current algorithm. Such an extension would be more complicated than the present algorithm, as the line search step would be replaced by the solution of an initial value problem. If $f(x)$ is sufficiently smooth, this problem may be solveable quickly; and the algorithm as a whole may be efficient enough to be of use. This possibility will be the subject of further exploration.

4 Results

We first make a few simple observations about factors which determine the order of accuracy of the proposed algorithm. Limiting factors include the smoothness of the interface Γ, the order of the interpolation routine used, and the profile of the input ϕ nearby and normal to the interface. Then we proceed to show numerical results that support these observations, and demonstrate the accuracy and efficiency of the proposed algorithm. Third-order accuracy is observed for biquadratic or triquadratic interpolation, and fourth-order accuracy for bicubic interpolation in regions for which the closest point on the interface is in a sufficiently smooth region of the interface.

The algorithm is observed to run with $O(N \log N)$ complexity, where N is the total number of grid points for which the redistancing is performed. The constant associated with the computational complexity is about twenty-five times that of the standard, first-order accurate fast marching method in our 2D implementation (one hundred times in 3D); however, in many applications this may be a reasonable price to pay in order to gain higher-order accuracy.

4.1 Order of Accuracy

Three important factors enter into the observed numerical order of accuracy of this algorithm.

- The interface Γ itself must possess sufficient smoothness. If Γ is C^k in a neighborhood of a point $\boldsymbol{y} \in \Gamma$, then the algorithm will have at best order $k+1$ accuracy at any point $\boldsymbol{x}$ for which $\boldsymbol{C}(\boldsymbol{x}) \approx \boldsymbol{y}$. In particular, any interface which contains a sharp corner will limit the method to first-order accuracy in the fan-shaped region for which the corner point is the closest point on the interface. This is because the location of such a corner can be located with at best first-order accuracy by any smooth interpolation method.
- The order of accuracy of the interpolation routine used also gives an upper bound on the accuracy of the algorithm as a whole. Numerical tests suggest that this upper bound is obtained for sufficiently smooth interfaces. The drawback of using high-order interpolation routines is that a wider stencil must be used to perform the interpolation. This in

n	Circle 1	Circle 2	Circle 3	Ellipse	Heptagon	Polar
32	2.69×10^{-5}	3.24×10^{-6}	6.72×10^{-5}	3.20×10^{-6}	1.94×10^{-3}	5.98×10^{-3}
64	2.92×10^{-6}	2.07×10^{-7}	1.87×10^{-5}	2.12×10^{-7}	9.20×10^{-4}	4.03×10^{-4}
128	3.89×10^{-7}	1.29×10^{-8}	4.84×10^{-6}	1.36×10^{-8}	4.22×10^{-4}	6.93×10^{-5}
256	4.63×10^{-8}	8.14×10^{-10}	1.18×10^{-6}	8.58×10^{-10}	2.19×10^{-4}	7.26×10^{-6}
512	6.01×10^{-9}	5.06×10^{-11}	3.09×10^{-7}	5.40×10^{-11}	9.53×10^{-5}	7.72×10^{-7}
1024	7.60×10^{-10}	3.15×10^{-12}	7.76×10^{-8}	3.38×10^{-12}	5.42×10^{-5}	5.72×10^{-8}
2048	9.33×10^{-11}	1.98×10^{-13}	1.95×10^{-8}	2.12×10^{-13}	2.13×10^{-5}	3.87×10^{-9}
4096	1.15×10^{-11}	1.24×10^{-14}	4.87×10^{-9}	1.35×10^{-14}	1.28×10^{-5}	2.47×10^{-10}

Table 1: L^2 error in reinitialized signed distance function for various numerical tests. n is the number of grid points in each spatial dimension. L^2 error is calculated as $\sqrt{\sum_{i,j} \left(d_{ij} - D\left(x_j, y_i\right)\right)^2}/n$.

turn means that the region around the interface whose signed distance and closest point functions are set in the first step of the algorithm must be widened appropriately.

- The accuracy with which the signed distance and closest point functions can be set near the interface is in turn limited by the accuracy of the input ϕ. Namely, if

$$\phi(\boldsymbol{x} = \boldsymbol{x_0} + c\nabla\phi|_{\boldsymbol{x_0}}) = K|\boldsymbol{x} - \boldsymbol{x_0}| + O\left(|\boldsymbol{x} - \boldsymbol{x_0}|^m\right), \tag{9}$$

that is, if the profile of ϕ normal to the interface is linear up to an order m correction for any $\boldsymbol{x_0} \in \Gamma$, then the algorithm can have as most m^{th} order accuracy.

4.2 Numerical Results

We present numerical results for this algorithm in a variety of cases. We begin with examples for which exact solutions are known: a circle, an ellipse, and a regular heptagon. These examples allow us to demonstrate the convergence properties of the algorithm for various initial data and interpolation routines. We describe some results on more complicated geometries. For all tests, we discretize the unit square on many uniform grids, with n grid points in each direction, where n ranges from 32 to 4096.

All numerical tests were performed on an Intel i3-2100 processor running at 3.10 GHz. The code was implemented in serial C and compiled as a MEX function in MATLAB, using the gcc complier, version 4.4.6.

4.2.1 Circle

We perform three tests of the reinitialization algorithm on a circle with radius $0.24 + \varepsilon_r$, and center $(0.55 + \varepsilon_x, 0.50 + \varepsilon_y)$ and $|\varepsilon.| < 0.01$ to mitigate any potential grid effects. The exact signed distance function is denoted as $D(\boldsymbol{x})$. We initialize the first two tests with $\phi(\boldsymbol{x}) = 2D(\boldsymbol{x})$, verifying that the algorithm handles constant scaling properly. The first test is performed using biquadratic interpolation, while the second uses bicubic interpolation. As expected, third-order accuracy is observed in the first test, and fourth-order accuracy in the second test. See Table 1 for calculated errors and Table 2 for convergence rates.

Timing information is presented in Table 3. The runtime is observed to be proportional to $N \log N$, where $N = n^2$ is the total number of grid points used. The cost of using bicubic interpolation over biquadratic interpolation is negligible, as can be observed by comparing the

n	Circle 1	Circle 2	Circle 3	Ellipse	Heptagon	Polar
64	3.21	3.97	1.85	3.91	1.08	3.89
128	2.91	4.00	1.95	3.97	1.13	2.54
256	3.07	3.99	2.04	3.98	0.95	3.25
512	2.94	4.01	1.93	3.99	1.20	3.23
1024	2.98	4.00	1.99	4.00	0.81	3.75
2048	3.03	3.99	2.00	4.00	1.35	3.89
4096	3.02	4.00	2.00	3.97	0.74	3.97

Table 2: Convergence rates for the numerical tests are calculated as $\log_2(E(n)/E(n/2))$, where $E(n)$ is the L^2 error reported in Table 1.

n	Circle 1	Circle 2	Circle 3	Ellipse	Heptagon	Polar
32	0.0105	0.00907	0.0105	0.00729	0.0101	0.00987
64	0.0277	0.0287	0.0391	0.0324	0.0409	0.0403
128	0.102	0.120	0.165	0.137	0.136	0.175
256	0.390	0.465	0.688	0.571	0.571	0.736
512	1.54	1.82	2.88	2.32	2.19	3.07
1024	6.11	6.79	11.9	9.29	8.63	12.7
2048	24.3	26.3	48.4	37.3	33.7	52.8
4096	100	105	197	123	144	194

Table 3: Runtime for the two-dimensional numerical tests (in seconds).

runtimes for the Circle 1 and Circle 2 experiments. At $n = 4096$, applying the bicubic interpolation gives an additional cost of only 5% over the biquadratic interpolation but gives an additional three digits of accuracy.

In the third test, we investigate the limit that initial data puts on the algorithm. We choose $\phi(\boldsymbol{x}) = \frac{3}{2}D(\boldsymbol{x}) + D(\boldsymbol{x})|D(\boldsymbol{x})|$, so that $O(\Delta x^2)$ error is introduced in the initial data near the interface. In this case, when the biquadratic interpolation version of the algorithm was applied, only second-order accuracy was found due to the error in the initial data.

4.2.2 Ellipse

To demonstrate that the rapid convergence of the algorithm is not dependent on the extreme symmetry of the circle, we test the algorithm on an ellipse. The ellipse is still smooth, but the characteristics of the eikonal equation collide along a line segment along the major axis rather than all meeting at a single point. For this test, we choose the lengths of the major and minor axes to be $0.33 + \varepsilon_a$ and $0.25 + \varepsilon_b$, respectively. We apply the bicubic interpolation and again obtain fourth-order accuracy.

4.2.3 Heptagon

Globally, we cannot expect to obtain greater than first-order accuracy when applying this algorithm to an initial curve which has sharp corners, because smooth methods cannot locate the corner with better than linear accuracy. Here, we consider a regular heptagon (7-gon) such that each corner is a distance $0.2 + \varepsilon_c$ from the center of the grid, with an arbitrary rotation applied

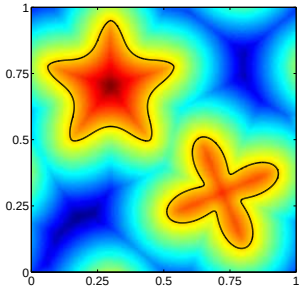


Figure 2: The almost-exact signed distance function $D(\boldsymbol{x})$ for the multiple interfaces is displayed, with the interface contour overlaid.

so that corners do not lie on the grid. The biquadratic interpolation version of the algorithm is applied as only first-order accuracy is expected, and found. Visualization of the error indicates that the first-order errors are committed in the fan-shaped region emanating from each corner for which that corner is the closest point, as expected. Along smooth parts of the interface, higher-order accuracy is still observed.

4.2.4 Multiple disjoint polar curves

In order to test the reinitialization on multiple smooth, disjoint interfaces, we define the parametric family of polar curves

$$\boldsymbol{x}(\theta) = \boldsymbol{x_0} + (\alpha + \beta \sin(\gamma\theta))\boldsymbol{\theta}, \tag{10}$$

where $\boldsymbol{\theta} = (\cos\theta, \sin\theta)$, and describe the interface Γ_i as a member of this family, for $i = 1, 2$. We utilize Newton iterations on the parameter θ to compute nearly-exact signed distance functions $D_i(\boldsymbol{x})$ to the interfaces, and obtain $D(\boldsymbol{x}) = \max(D_1(\boldsymbol{x}), D_2(\boldsymbol{x}))$. See Figure 2 for a visualization of the interfaces and the nearly-exact signed distance function. Bicubic interpolation is applied, and the expected fourth-order convergence is observed once the grid is fine enough to sufficiently resolve the fine-scale details of the interfaces.

4.2.5 Three Dimensions

To demonstrate the efficacy of our algorithm in three dimensions, we reconstruct the signed distance function to a prolate ellipsoid with minor axis length 0.2 and major axis length 0.3. We use triquadratic interpolation and obtain the expected $O(\Delta x^3)$ convergence, see Table 4. There we also present running times for the three-dimensional algorithm. These times clearly reflect the $O(N \log N)$ complexity of the algorithm.

5 Conclusions

We present a novel "directional optimization" algorithm for the reinitialization of the signed distance function from an input level set function ϕ. This algorithm is most similar to the well-known Dijkstra's algorithm [5], Tsitsiklis's algorithm [24], and fast marching method [17], however it differs significantly in the update step. The update step of those algorithms depends only on reinitialized values of the signed distance function local to the grid point $\boldsymbol{x}$ being updated, whereas our algorithm directly interrogates the interface to locate the closest point $\boldsymbol{C}(\boldsymbol{x})$. Directional optimization allows for significantly greater accuracy with the same computational

n	$E(n)$	C.R.	Time	Time / $N \log N$
8	3.78×10^{-3}	—	0.0282	8.81×10^{-6}
16	4.27×10^{-4}	3.14	0.170	4.98×10^{-6}
32	5.07×10^{-5}	3.08	1.52	4.47×10^{-6}
64	6.23×10^{-6}	3.02	13.2	4.02×10^{-6}
128	7.72×10^{-7}	3.01	113	3.69×10^{-6}

Table 4: Numerical results for reinitialization of the ellipsoid. The L^2 error $E(n)$ is measured as $\left(\sum_{i,j,k} \left(d(x_{ijk}) - D(x_j, y_i, z_k)\right)^2\right)^{1/2} / n^{3/2}$. Convergence rate is computed as $\log_2(E(n)/E(n/2))$. The runtime is presented (in seconds), and shown to be proportional to $N \log N$, where $N = n^3$.

complexity as comparable algorithms. As a second advantage, the closest point function $\boldsymbol{C}(\boldsymbol{x})$ is obtained "for free" by the proposed algorithm. The closest point function is the subject of significant recent interest [10, 11, 12, 16, 21], and methods to quickly and accurately construct the closest point function are timely.

Two natural directions of future work are suggested. As discussed more fully in Section 3.3, it may be possible to extend the directional optimization algorithm to the more general eikonal equation by propagating "interface direction" information rather than the closest point information currently passed and integrating along search directions to find the cost along characteristic curves. Anotheer natural extension is to modify this algorithm for the reinitialization of the closest point function from some approximation to $\boldsymbol{C}(\boldsymbol{x})$. The closest point function can represent a wider class of surfaces than the level set function, which is (in its usual form) restricted to closed surfaces of codimension one. The closest point function can represent both open and closed surfaces of arbitrary codimension. In its current form, this reinitialization algorithm relies on the notion of "inside" and "outside" to locate the interface in the line search subalgorithm. Furthermore, it is unclear how the directional optimization substep should handle surfaces of codimension greater than one. This extension to the reinitialization method proposed here is the subject of ongoing work.

6 Acknowledgments

This work was supported, in part, by grants from the National Science Foundation: DMS–0748333, DMS–0713767, and OISE–0967140, and by the Rackham Predoctoral Fellowship.

References

[1] Benamou, J.-D., Luo, S., and Zhao, H. A compact upwind second order scheme for the Eikonal equation. *J. Comput. Math. 28*, 4 (2010), 489–516.

[2] Cheng, L.-T., and Tsai, Y.-H. Redistancing by flow of time dependent Eikonal equation. *J. Comput. Phys. 227* (2008), 4002–4017.

[3] Chopp, D. L. Computing minimial surfaces via level set curvature flow. *J. Comput. Phys. 106* (1993), 77–91.

[4] Chopp, D. L. Some improvements of the fast marching method. *SIAM J. Sci. Comput. 23*, 1 (2001), 230–244.

[5] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numer. Math. 1* (1959), 269–271.

[6] Elsey, M., Esedoḡlu, S., and Smereka, P. Diffusion generated motion for grain growth in two and three dimensions. *J. Comp. Phys. 228*, 21 (2009), 8015–8033.

[7] Elsey, M., Esedoḡlu, S., and Smereka, P. Large scale simulation of normal grain growth via diffusion generated motion. *Proc. R. Soc. Lond. A 467* (2011), 381–401.

[8] Elsey, M., Esedoḡlu, S., and Smereka, P. Large-scale simulations and parameter study for a simple recrystallization model. *Phil. Mag. 91*, 11 (2011), 1607–1642.

[9] Esedoḡlu, S., Ruuth, S., and Tsai, R. Diffusion generated motion using signed distance functions. *J. Comp. Phys. 229*, 4 (2010), 1017–1042.

[10] Macdonald, C. B., Brandman, J., and Ruuth, S. J. Solving eigenvalue problems on curved surfaces using the Closest Point Method. *J. Comput. Phys. 230*, 22 (2011), 7944–7956. doi:10.1016/j.jcp.2011.06.021.

[11] Macdonald, C. B., and Ruuth, S. J. Level set equations on surfaces via the Closest Point Method. *J. Sci. Comput. 35*, 2–3 (June 2008), 219–240. doi:10.1007/s10915-008-9196-6.

[12] Macdonald, C. B., and Ruuth, S. J. The implicit Closest Point Method for the numerical solution of partial differential equations on surfaces. *SIAM J. Sci. Comput. 31*, 6 (2009), 4330–4350. doi:10.1137/080740003.

[13] Osher, S., and Fedkiw, R. *Level set methods and dynamic implicit surfaces*, vol. 153. Springer, New York, 2003.

[14] Rouy, E., and Tourin, A. A viscosity solutions approach to shape-from-shading. *SIAM J. Numer. Anal. 29*, 3 (1992), 867–884.

[15] Russo, G., and Smereka, P. A remark on computing distance functions. *J. Comput. Phys. 163* (2000), 51–67.

[16] Ruuth, S. J., and Merriman, B. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys. 227* (2008), 1943–1961.

[17] Sethian, J. A. A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci. 93* (1996), 1591–1595.

[18] Sethian, J. A. *Level set methods and fast marching methods*, 2 ed. Cambridge University Press, Cambridge, 1999.

[19] Sethian, J. A., and Vladimirsky, A. Fast methods for the Eikonal and related Hamilton–Jacobi equations on unstructured meshes. *Proc. Nat. Acad. Sci. 97*, 11 (2000), 5699–5703.

[20] Sussman, M., Smereka, P., and Osher, S. A level set approach for computing solutions to incompressible two-phase flow. *J. Comput. Phys. 114* (1994), 146–159.

[21] Tian, L., Macdonald, C. B., and Ruuth, S. J. Segmentation on surfaces with the Closest Point Method. In *Proc. ICIP09, 16th IEEE International Conference on Image Processing* (Cairo, Egypt, 2009), pp. 3009–3012. doi:10.1109/ICIP.2009.5414447.

[22] Tsai, Y.-H. R. Rapid and accurate computation of the distance function using grids. *J. Comput. Phys. 178* (2002), 175–195.

[23] Tsai, Y.-H. R., Cheng, L.-T., Osher, S., and Zhao, H.-K. Fast sweeping algorithms for a class of Hamilton–Jacobi equations. *SIAM J. Numer. Anal. 41*, 2 (2003), 673–694.

[24] Tsitsiklis, J. N. Efficient algorithms for globally optimal trajectories. *IEEE Trans. Autom. Control 40*, 9 (1995), 1528–1538.

[25] Zhao, H.-K. A fast sweeping method for Eikonal equations. *Math. Comput. 74*, 250 (2004), 603–627.