LSMLIB: A Level Set Method Software Library for Application Developers

Kevin T. Chu
Department of Mechanical & Aerospace Engineering
Princeton University

Maša Prodanović
The Institute for Computational Engineering and Sciences
University of Texas at Austin

Introduction

The level set method is a mature theoretical and numerical method for using implicitly defined surfaces study moving interface problems [1, 2]. In recent years, there has been a surge of interest in the application of level set methods to a wide range of problems. A *very* small sampling of these applications (that the authors are involved with) includes: shape optimization [3, 4, 5, 6], flow in porous media [7, 8], and simulation of dislocation dynamics [9, 10, 11]. The growing number of applications of level set methods signals a need for high-quality software that makes it easier for non-experts to write level set method programs. The Level Set Method Library (LSMLIB) is an effort to deliver state-of-the-art level set method algorithms to the scientific and engineering communities.

LSMLIB provides a collection of level set method algorithms and numerical kernels which support serial and parallel simulation of dynamic interface problems in two and three space dimensions on structured grids. LSMLIB supports solution of the time-dependent and time-independent level set equations. For the time-dependent level set equations, LSMLIB provides functionality to solve both the vector velocity and normal velocity forms of the equations. Support is also provided for extension/extrapolation of field variables off of the zero level set, reinitialization of level set functions, and, for codimension-two problems, orthogonalization of the gradients of pairs of level set functions [1, 2].

The primary goal of the LSMLIB development effort is to deliver an accessible and versatile software package that allow scientists and engineers to rapidly develop high-performance level set method simulations without requiring them to become experts in the details of the underlying numerics. LSMLIB achieves this goal by defining interfaces that only require the user to supply problem specific information. Moreover, interfaces are provided for several common programming languages (C, C++, MATLAB, and Fortran) to suit the user's programming experience and computational needs. High-performance is achieved by using mixed-language programming techniques that make it possible to implement low-level numerical kernels in a fast language, such as Fortran or C.

Overview of Level Set Methods

The level set method was originally developed by Osher and Sethian in the 1980s [12] and has since been applied to many problems involving dynamic surfaces and curves. The fundamental idea underlying level set methods is that a surface[1] can be represented implicitly as the zero level set of an embedding function, ϕ. That is, a surface is taken to be the set of points in space where ϕ is zero: $\{\mathbf{x} : \phi(\mathbf{x}) = 0\}$. This viewpoint differs from other theoretical and computational methods which rely on explicit representations of the surface, such as marker/string [2] and front-tracking methods [13, 14].

Time-dependent Level Set Equations. For an implicitly represented surface, the dynamics of the surface are determined by the evolution of the embedding function. It is straightforward to derive an evolution equation for the embedding function by considering a particle on the surface. Because the value of ϕ associated with the particle is always zero, the evolution equation for ϕ in a Lagrangian frame moving with the particle is just

$$\frac{d\phi}{dt} = 0. \tag{1}$$

[1] For clarity, we shall focus solely on 2D-surfaces in 3D. However, much of the discussion may be generalized to codimension-one objects in any number of dimensions – in particular, to curves in 2D.

Transforming this equation to an Eulerian frame requires only that we replace the time derivative with a material derivative (or use the chain-rule) to obtain

$$\frac{\partial \phi}{\partial t} + \mathbf{V} \cdot \nabla \phi = 0 \tag{2}$$

where $\mathbf{V}$ is the velocity of the surface, which may be a function of both position and time. This equation gives the vector velocity form of the time-dependent level set equation. To determine the surface at a specified time, one need only solve (2) and locate the points $\mathbf{x}$ where $\phi(\mathbf{x}) = 0$.

For many problems, it is convenient to use a form of the time-dependent level set equation which only involves the normal velocity V_n:

$$\frac{\partial \phi}{\partial t} + V_n \left| \nabla \phi \right| = 0. \tag{3}$$

The normal velocity form of the level set equation follows directly from (2) by writing $\mathbf{V}$ as $\mathbf{V} = V_n \hat{\mathbf{n}} + V_s \hat{\mathbf{s}} + V_t \hat{\mathbf{t}}$, where $\hat{\mathbf{n}}$ is a unit vector normal to the surface, and $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ are an orthonormal basis for the tangent plane. Because $\nabla \phi$ is orthogonal to the surface, $\nabla \phi \cdot \hat{\mathbf{s}}$ and $\nabla \phi \cdot \hat{\mathbf{t}}$ are both zero, leaving only the normal velocity to contribute to the evolution of the embedding function.

Even when a vector velocity for the surface is available, it can be advantageous to cast the problem in normal velocity form because motion of the interface in the tangential direction does not change the shape of the surface. Furthermore, the use of the special extension velocities techniques described in a later section only apply to problems where the normal velocity is specified.

Signed Distance Function. In the above discussion, there are no constraints placed on the embedding function except that the zero level set coincide with the surface of interest. However, it is often convenient to additionally require that

$$\left| \nabla \phi \right| = 1, \tag{4}$$

so that the embedding function is a signed distance function. The main purpose of this restriction on ϕ is to ensure that, near the zero level set, numerical computations of gradients are accurate. Other choices of the embedding function, such as Heaviside functions, have been shown to result in decreased numerical accuracy [15].

Time-independent Level Set Equation. For the class of single-signed normal velocity fields (*i.e.,* $V_n > 0$ or $V_n < 0$ everywhere), there is an important alternative way to represent the dynamics of the surface. Rather than choosing the embedding function in such a way that motion of the surface is given by the dynamics of the zero level set, we can choose the embedding function so that its value at a position $\mathbf{x}$ in space is equal to the time when the surface passes through $\mathbf{x}$ from a given surface configuration at the initial time. We shall refer to this embedding function as the arrival function, $T(\mathbf{x})$, in light of its interpretation as the time it takes for the surface to reach each point in space from its initial configuration. Given an arrival function, the surface at a later time (or earlier time if $V_n < 0$), t, is defined as the set of points where the embedding function takes the value t.

The equation for the arrival function is easily derived by recognizing that $|\nabla T|$ is the rate of change of the arrival time with respect to the change in the distance to the surface. In other words, $|\nabla T|$ is equal to the reciprocal of the normal velocity:

$$\left| \nabla T \right| = 1/V_n. \tag{5}$$

If V_n depends only on position and is independent of time, this equation becomes the well-known Eikonal equation [2]. Note that the fact that the normal velocity is single-signed is necessary in order to be able to define the arrival function. If the normal velocity changes sign, then there would be some point in space that the surface passes through more than once. As a result, the arrival function would be an ill-defined function.

Extension Velocities. In the preceding discussion, we have focused on the evolution of the embedding function at the surface. However, the level set equations themselves are partial differential equations that are defined throughout the entire spatial domain. In writing the level set equations, we have implicitly assumed that the velocity field is a defined quantity off of the surface. In order to use level set methods, we must ensure that we have a means for defining the velocity field throughout the spatial domain. This extended velocity field is known as the *extension velocity* [2].

For problems involving the vector velocity form of the time-dependent level set equations, the velocity field is usually defined throughout the entire domain as part of the problem. For example, in fluid flow problems, the fluid velocity is a natural choice for the extension velocity.

For problems where the normal velocity is specified, a good way to define the extension velocity is to extrapolate the velocity off the surface so that it is constant on rays normal to the surface [1, 2]. This goal can be achieved by solving either the time-dependent two-way extrapolation equation [1]

$$\frac{\partial V_n}{\partial t} + \text{sgn}(\phi)\hat{\mathbf{n}} \cdot \nabla V_n = 0 \tag{6}$$

or its steady-state equivalent [2]

$$\nabla V_n \cdot \nabla \phi = 0. \tag{7}$$

The advantage of this choice of extension velocity is that it preserves signed distance functions [2].

Field Extension. For some problems, it may be necessary to a extend field variable, γ, other than ϕ or the V_n off of the zero level set. In these situations, equations (6) and (7) with V_n replaced by γ provide two convenient methods for computing the extension field for γ:

$$\frac{\partial \gamma}{\partial t} + \text{sgn}(\phi)\hat{\mathbf{n}} \cdot \nabla \gamma = 0 \tag{8}$$

$$\nabla \gamma \cdot \nabla \phi = 0. \tag{9}$$

Reinitialization. At the beginning of a time-dependent level set method calculation, there may be a need to generate a signed distance function, ϕ, from an arbitrary embedding function. In addition, as a level set method calculation progresses, numerical errors will often cause ϕ to drift away from being a signed distance function (even if the extension velocity is defined as described in the previous section). To deal with both of these issues, we use a process known as *reinitialization*, which essentially computes a signed distance function from a given embedding function.

As with the basic level set equations and extension velocity equations, there are time-dependent and time-independent formulations of the reinitialization procedure. The goal of both formulations is to solve the signed distance function equation (4). The time-dependent approach solves (4) as the steady solution to the equation

$$\frac{\partial \phi}{\partial t} + \text{sgn}(\phi)\left(|\nabla \phi| - 1\right) = 0. \tag{10}$$

The time-independent approach solves (4) directly by treating it as a time-independent level set equation of the form (5) with a constant normal velocity of 1.

Each approach has its advantages and disadvantages. Equation (10) is convenient because it can be solved using the same machinery developed for solving the time-dependent level set equations and does not require the zero level set to be explicitly computed. Unfortunately, although the mathematical solution of (10) guarantees that the zero level set will remain stationary, errors in the numerical solution can cause the location of the surface drift during the reinitialization process [1, 2]. The time-independent approach does not suffer from the drift problem and can be solved using an elegant algorithm known as the fast marching method (described in the Level Set Method Algorithms section). However, it requires explicit computation of the zero level set and is difficult to parallelize.

Orthogonalization. For codimension-two problems, it is often desireable to maintain orthogonality between the gradients of the two level set functions whose zero level sets intersect to define the 3D curve. This goal is achieved by alternately solving the orthogonalization equations for the pair of level set functions ϕ and ψ:

$$\frac{\partial \phi}{\partial t} + \text{sgn}(\psi)\frac{\nabla \phi \cdot \nabla \psi}{|\nabla \psi|} = 0 \tag{11}$$

$$\frac{\partial \psi}{\partial t} + \text{sgn}(\phi)\frac{\nabla \psi \cdot \nabla \phi}{|\nabla \phi|} = 0. \tag{12}$$

As with the time-dependent reinitialization equation, these equations can be solved using the same techniques used to solve the time-dependent level set equations. It should be noted that orthogonalization procedures are

still an area of research, so the orthogonalization functionality provided by LSMLIB may lack the robustness of other parts of the library.

Further Reading. For a deeper discussion of level set methods, we refer the reader to *Level Sets Methods and Dynamic Implicit Surfaces* by S. Osher and R. Fedkiw and *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science* by J. A. Sethian. Both books provide an excellent discussion of the theory underlying level set methods and give many interesting example applications.

Level Set Method Algorithms

Numerical Algorithms for Hamilton-Jacobi Equations. For externally generated velocity fields, both the vector velocity and normal velocity forms of the time-dependent level set equations are examples of time-dependent Hamilton-Jacobi equations. To solve these equations, LSMLIB uses well-developed numerical methods for solving this class of equations [1, 16, 17]. The basic approach is to first discretize the spatial derivatives using high-order finite difference schemes and then solve the resulting semi-discrete equations using explicit total-variation diminishing (TVD) Runge-Kutta (RK) time integration.

Spatial derivatives of ϕ in (2) and (3) are computed using high-order essentially non-oscillatory (ENO) [1, 18, 19, 20, 21] or weighted essentially non-oscillatory (WENO) [1, 22, 23, 24] discretizations. The idea underlying both of these schemes is that spatial derivatives should be computed using the smoothest possible polynomial interpolant of the local grid points [1]. The main difference between ENO and WENO schemes is that a WENO scheme make use of *all* of the ϕ values from the set of grid points that are potentially used by a lower-order ENO scheme. The result is that the WENO scheme has a higher-order than its associated ENO scheme in regions where ϕ is smooth. For a more detailed description of ENO and WENO discretizations, we refer the reader to an excellent discussion by Osher and Fedkiw in [1].

In each grid cell, the forward or backward approximation of the spatial derivative can be selected for each coordinate direction. For the vector velocity form of the equation (2), the appropriate approximation for the spatial derivative is selected by using the components of the velocity field to determine the upwind direction [1, 2, 16, 17]. For the normal velocity form of the equation (3), the direction of the velocity is no longer available, so Godunov's method is used to select the spatial derivative [1, 2, 16, 17]. It is worth mentioning that the upwinding approach used for (2) is completely equivalent to Godunov's method [1].

In LSMLIB, the method of lines [25, 26] is used for temporal discretization of the time-dependent level set equations. The semi-discrete equations that arise once the discretization scheme for the spatial derivatives has been chosen are solved using explicit total-variation diminishing (TVD) Runge-Kutta (RK) time integration [1, 19, 27]. The basic idea behind the TVD-RK schemes is that increased accuracy in time can be achieved by taking linear combinations of the results of multiple first-order forward Euler steps with the initial data available at the beginning of the time step. The TVD property is preserved by only taking convex combinations of the forward Euler steps and initial data. As pointed out in [1], the TVD property is not guaranteed when upwinding is used. However, practical experience has shown that the upwind schemes are likely to be total variation bounded (TVB) when using TVD-RK schemes. For an explicit description of TVD Runge-Kutta methods, we refer the reader to an excellent discussion by Osher and Fedkiw in [1].

In LSMLIB, the following spatial discretization schemes are provided for the user: ENO1, ENO2, ENO3, and WENO5. First-, second-, and third-order TVD-RK time integrators are supplied for time integration. The user is free to choose which of these schemes is most appropriate for his/her problem.

Effects of Interdependence Between Velocity Field and ϕ. The numerical methods described are designed specifically for Hamilton-Jacobi equations where the external velocity is independent of the embedding function ϕ. In many applications of interest, however, the velocity field may have a dependence on ϕ. For example, in mean curvature problems, the velocity field depends on ϕ through the curvature of the zero level set. Problems arising in physical systems (*e.g.,* multiphase fluids and dislocation in materials) may have an even more complicated relationship between the velocity field and ϕ. In these situations, we technically cannot treat the time-dependent level set equations as Hamilton-Jacobi equations and should design numerical schemes that are appropriate for the governing systems of equations.

Fortunately, practical experience has shown that assuming that the velocity field is independent of the embedding function and using the algorithms for Hamilton-Jacobi equation is adequate for many problems of interest. The main concern when using this approach is typically that the time step size needs to be reduced

in order to ensure numerical stability. Often, the stable time step size is determined via a combination of experimentation and theoretical intuition.

Field Extension, Reinitialization, and Orthogonalization. Because the field extension, reinitialization, and orthogonalization equations are all Hamilton-Jacobi equations, the same approach described in the previous section are used to solve these equations. As for the time-dependent level set equations, LSMLIB provides the user a choice between several spatial and temporal discretization schemes.

In principle, the time-dependent field extension, reinitialization, and orthogonalization equations should be solved to steady-state. However, because all of these equations are hyperbolic with characteristics that propagate away from the zero level set, we can reduce the computational cost required to solve these equation by stopping the time integration well before steady-state is reached. In practice, it is often sufficient to use stopping criterion based on the number of time steps taken or an estimated distance travelled by the data front. The distance stopping criterion relies on the fact that the data propagates with a speed of one along the characteristics for (8), (10), (11) and (12).

Computational Cost Analysis. The computational cost of the numerical algorithms for the time-dependent level set, field extension, reinitialization, and orthogonalization equations are dominated by two factors: (1) the size of the computational grid used to solve the problem and (2) the number of time steps taken during the simulation. Because the temporal discretization for the PDEs is based on the method of lines, time integration is the outerloop for the computation of the "right-hand side" of the semi-discrete equations. As a result, the algorithmic analysis for the numerical method may be decomposed into an analysis of the cost for computing the terms that involve only the spatial derivatives and the cost of the time integration scheme.

When computing the spatial derivative terms in any of the time-dependent PDEs that arise in level set method calculations, the same operation is carried out for each grid point in the computational domain. As a result the time and memory cost for computing the spatial derivative terms is $O(N^d)$, where N is the number of grid points in one spatial direction and d is the dimensionality of the problem. In addition to this cost, there is a smaller order cost associated with filling the ghostcells to impose boundary conditions. This cost is $O(N^{d-1})$ because it is proportional to the number of grid points on the boundary of the computational domain. It is tedious to get a more precise cost estimate for the different possible discretizations for the spatial derivatives, but it is relatively easy to *compare* the costs.

For ENO schemes, the number of finite difference tables required is exactly equal to the order of the method. In constrast, WENO discretizations use only one finite difference table regardless of the order the scheme. Because difference tables dominate the memory requirements for spatial derivative calculations, the memory costs for the various choices of the spatial derivative can be placed in the following rough order:

$$ENO1 < WENO5 < ENO2 < ENO3, \tag{13}$$

where the memory cost of the WENO5 scheme is just slightly more than the ENO1 scheme due to the wider ghost cell width required for WENO5 schemes.

The time complexity of of ENO schemes also scales with the order of accuracy for the discretization. More specifically, each additional order of accuracy requires approximately one extra addition, one extra multiplication, and one extra logical comparison per grid point. In comparison, the fifth-order WENO scheme uses no logical comparisons but requires significantly more arithmetic operations than any of the ENO schemes. Thus, the ordering of the different spatial derivative discretizations is roughly:

$$ENO1 < ENO2 < ENO3 < WENO5. \tag{14}$$

Unfortunately, the above analysis for the time complexity of the different spatial discretizations neglects an important contributor to the actual running time for an actual simulation – the impact of cache misses on performance. Because the fifth-order WENO scheme requires data from five different grid indices when it computes the spatial derivative at each grid point, it is likely to be more sensitive to cache misses than the ENO schemes, which only need data from one or two grid indices.

Empircal evidence suggests that the time complexity analysis for ENO schemes is robust. However, when comparing the ENO3 and WENO5 schemes, the running times can range from being comparable to being signficantly slower for the WENO5 scheme. Three-dimensional problems seem to produce results where ENO3 and WENO5 perform similarly, where two-dimensional problems yield a large difference between the

running times. The difference between the running times in two and three space dimensions is still an issue we are investigating.

The computational costs of the time integration schemes are relatively straightforward to estimate. Because number of intermediate steps required for a TVD Runge-Kutta integration schemes is one less than the order of the scheme, the amount memory needed for an n-th order TVD Runge-Kutta scheme is equal to $(n+1)$ times the memory required to store a single field variable on the computational grid plus the memory required for the spatial derivative calculation. The factor of $(n+1)$ is a result of the storage needed for the initial time step, $(n-1)$ intermediate time steps, and the right-hand side of the semi-discrete equations.

The total time required for a level set method calculation is equal to the number of time steps taken multiplied by the time it takes for each time step. For level set calculations with externally supplied velocity fields, the maximum stable time step size is set by the CFL condition [1, 16]:

$$\Delta t = \alpha \Delta x, \tag{15}$$

where α is chosen to be a sufficiently small scale factor that is independent of the grid spacing. From this relation, we see that, if the difference between the final and initial simulation time is held fixed, the total number of time steps required by a simulation is proportional to N, the number of grid points in each coordinate direction. By combining this observation with the result that computation of the spatial derivative terms takes $O(N^d)$ time, we find that the computational time required for a fixed simulation time calculation is $O(N^{d+1})$. Therefore, the cost of level set method calculations increases rapidly with the number of grid points. For problems with stricter stability conditions, the cost of time-dependent level set method calculations increases even more rapidly.

Efficient Time-Dependent Level Set Method Schemes. To reduce time required for time-dependent level set method calculations, LSMLIB provides support for both localization and parallelization of these computations.

Localization and Narrow Band Methods. To reduce the computational effort required by serial, time-dependent level set methods, LSMLIB provides support for localization and narrow-band methods [28]. In the localization method, only the grid points in a narrow band around the interface are updated during each time step. This procedure is only slightly more complicated to implement than the the original level set method and reduces the computational effort by an order of magnitude by effectively decreasing the dimensionality of the problem by one. As a result, the time needed to compute the spatial derivative terms becomes $O(N^{d-1})$ and the total time required to solve the time-dependent level set equations becomes $O(N^d)$. Localization based on [28] does, however, require an extra $O(N^d)$ of memory to store the narrow band information. In the future, we plan to investigate more memory efficient implementations of localization, such as the dynamic tubular grid [29].

Parallel Computation. An alternative method for reducing the time required for time-dependent level set method calculations is to rely on parallel computation. Because the grid point updates during each time step are independent of each other, parallelization of the time-dependent level set method algorithms is easily achieved by decomposing the computational domain into rectangular blocks and distributing the data on these blocks across multiple processors. Before each time step (or substep for second- and third-order TVD Runge-Kutta schemes), ghost cell data is communicated between processors to ensure that each processor has all of the data it needs to update the grid points is assigned.

While parallelization does not reduce the amount of computational work that is done during the calculation, it (1) reduces the total amount of time required to obtain a solution and (2) makes it possible to run very high-resolution simulations that require more memory than available on a single machine. Ideally, the amount of time required to obtain a solution would scale as $1/p$ where p is the number of processors used for the parallel computation. This scaling, however, assumes that communication time is negligible compared to computation time.

In practice, the actual time required to obtain a solution depends on the trade-off between communication and computation. Using a simplistic model for the cost of communication (that neglects latency and other effects), the communication time for the transfer of ghost cell data between the processors is $O(N^{d-1})$ because it is proportional to the surface area of the blocks. Assuming that the ratio of communication to

computation speed is r, the time required for each time step of a parallel level set method calculation is

$$O\left(\frac{N^d}{p} + rN^{d-1}\right). \tag{16}$$

This bound may be improved if computation and communication can be overlapped. However, it can be difficult to estimate the amount of overlap that can be achieved.

Optimal performance for a given simulation and parallel computer is achieved through a combination of efficient design of the communication pattern for the computation and tuning of the number of processors used for the calculation to the problem size. As a rule of thumb, one should choose the number of processors so that computation and communication times roughly balance: $p \approx (N/r)$. Beyond this point, using more processors will not dramatically speed up the calculation because communication time will dominate the total simulation time in (16).

LSMLIB provides support for parallel computations by building off of the Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI) developed and maintained by the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. SAMRAI handles many of the important issues arise parallel computing is used to solve partial differential equations (*e.g.,* domain decomposition and load balancing). LSMLIB's robustness and scalability for parallel computation is in large part due to the high quality of the SAMRAI library.

Fast Marching Methods. Fast marching methods are an elegant way to solve the time-independent level set and field extension equations [1, 2]. These equations are solved using upwind discretizations of (5) and (9). The resulting discrete equations for the are quadratic for (5) and linear for (9) regardless of the order of accuracy chosen for the spatial discretization. For example, the equations for a first-order accurate discretization of (5) in 3D are [2]:

$$\left[\begin{array}{rl} & \max\left(D_{ijk}^{-x}T, -D_{ijk}^{+x}T, 0\right)^2 \\ + & \max\left(D_{ijk}^{-y}T, -D_{ijk}^{+y}T, 0\right)^2 \\ + & \max\left(D_{ijk}^{-z}T, -D_{ijk}^{+z}T, 0\right)^2 \end{array}\right]^{1/2} = \frac{1}{F_{ijk}}, \tag{17}$$

where $F_{ijk} = V_n(x_i, y_j, z_k)$ and we have used the forward and backward finite difference operator notation, D^+ and D^-, to express the equation more compactly. For the explicit forms of the analogous field extension equations, we refer the reader to the discussion in [2] and [30].

The key idea behind fast marching methods is that the upwind numerical discretizations of (5) and (9) lead to a set of discrete equations where there are no circular data dependencies. Intuitively, the solution can be marched outward from the initial data because the solution at a grid point never depends on the the solution at grid points further away from the initial data.

Using this observation, the fast marching algorithm for (5) can be stated as follows [2]:

(1) Tag all grid points outside of the computational domain as *Outside*.
(2) Initialize the solution, T, for all the points in a small neighborhood of the zero level set. These points supply the boundary values for equation (5). The number of grid points around the zero level set to initialize typically depends on the order of accuracy of the numerical discretization. Tag these points as *Known*.
(3) Estimate the value of T for all points (except for *Known* or *Outside* points) that are immediately adjacent to a *Known* point using only solution values from *Known* points. Tag these points as *Trial*.
(4) Tag all remaining grid points as *Far*.
(5) Execute the following loop until there are no more *Trial* points.
    (a) Let A be the *Trial* point with the smallest value of T. Change the tag of A from *Trial* to *Known*.
    (b) Estimate the value of T for all *Far* points that are immediately adjacent to A using only solution values from *Known* points. Change the tag of these points to *Trial*.
    (c) Recompute value of T for all *Trial* points that are immediately adjacent to A using only solution values from *Known* points. Continue to tag these points as *Trial*.

For equation (9), we use the same algorithm (including computing the solution for the arrival function T) but augment it to include estimating and recomputing the solution to the time-independent field extension equation. In essence, the fast marching method is a way to solve the system of discretized equations efficiently by choosing an optimal order in which to solve the equations at each grid point [2]. It is also interesting to note that the fast marching methods are a variation of Dijkstra's algorithm for computing the shortest path on a graph with weighted edges [1, 2].

LSMLIB provides fast marching method support for computing distance functions, computing extension fields, and solving the Eikonal equation [2, 30]. Currently, only first-order discretizations spatial derivatives are available, but support for second-order discretizations are planned for a future version of the library.

Efficient Data Structure and Algorithms. The most computationally expensive component of the fast marching method is determination of the grid point with the smallest value of T. In order to make this operation efficient, we manage the *Trial* points using a heap data structure implemented as a complete binary tree stored in a contiguous block of memory [31]. In LSMLIB, we use the standard approach for implementing the heap data structure except that we use an extra layer of indirection between the nodes in the heap and the actual data associated with the node. This layer of indirection reduces the overhead cost of updating the "pointers" from grid points to the data associated with nodes when the minimum is removed. The "pointers" from grid points to node data are necessary in order to make updating the value of T associated with a node a cheap operation.

Computational Cost Analysis. Because fast marching methods terminate only after all grid points have been removed from the set of *Trial* points, the computational cost of fast marching methods is equal to the total number of points in the computational domain, $O(N^d)$, times the time required to find the *Trial* point with the smallest value of T and update the solution at neighboring grid points. Because updating the neighbors takes a bounded amount of time, the time required to find the minimal *Trial* point dominates the cost of the innermost loop of the fast marching method algorithm. By using a heap data structure, the cost of finding the minimal *Trial* point (including the cost of maintaining all heap properties) is on $O(\log(N^d)) = O(\log N)$. Therefore the total cost of fast marching methods is $O(N^d \log N)$. For single-signed normal velocity fields, fast marching methods have the potential to be much faster than their time-dependent cousins. However, in terms of memory, fast marching methods are not a significant improvement over time-dependent level set methods because they still use an $O(N^d)$ amount of memory to store the field data.

One major difference between time-dependent level set methods and fast marching methods is that the latter are inherently serial algorithms. For time-dependent level set methods, the same calculation is done independently at each grid point during each time step. The independence of the calculations at the grid points makes parallelization very straightforward. In constrast, for fast marching methods, the calculations carried out at a grid point depend on the results of calculations at other grid points making parallelization more difficult.

Arbitrary Computational Domains. Even though all LSMLIB calculations implicitly assume an underlying structured grid, LSMLIB supports computations on arbitrary domains through the use of masking. Masks may be used for both the time-dependent and time-independent forms of the level set equations. Because the logic required to deal with masks can substantially reduce the computational performance of a simulation, LSMLIB provides two versions of several of its core numerical kernels – a version with support for masking and a version that does not use masks.

Geometric Computations. LSMLIB provides support computation of several geometric quantities related to implicitly defined surfaces: unit normal vector (naturally extended off of the zero level set), perimeter/surface area of the zero level set, area/volume of region enclosed by the zero level set, surface integrals over the zero level set, and volume integrals over the region enclosed by the zero level set. For codimension-two problems, support is provided to locate the intersection of the two zero level sets.

Computation of the unit normal vector involves no special algorithmic features. Depending on the user's preferences and needs, the spatial derivatives required to compute the unit normal vector are calculated using the ENO/WENO schemes or standard central finite difference schemes (second- and fourth-order are provided).

Location of the intersection of two zero level sets is also algorithmically straightforward. We need only compute linear approximations to the level set functions and use vector calculus ideas to determine their intersection.

The remainder of the geometric computations provided by LSMLIB involve integration over the surface of the zero level set, $\partial\Omega$, or regions of space separated by the zero level set, Ω^+ and Ω^-. For each of these calculations, we use the Heaviside function, $H(x)$, and Dirac delta function, $\delta(x)$, to recast the integral in terms of an integration over the entire computational domain, Ω:

$$S = \int_{\partial\Omega} f(\mathbf{x}) dA = \int_{\Omega} f(\mathbf{x}) \delta(\phi(\mathbf{x})) |\nabla\phi(\mathbf{x})| d\mathbf{x} \tag{18}$$

$$V^+ = \int_{\Omega^+} g(\mathbf{x}) d\mathbf{x} = \int_{\Omega} g(\mathbf{x}) H(\phi(\mathbf{x})) d\mathbf{x} \tag{19}$$

$$V^- = \int_{\Omega^-} g(\mathbf{x}) d\mathbf{x} = \int_{\Omega} g(\mathbf{x}) \left(1 - H(\phi(\mathbf{x}))\right) d\mathbf{x}, \tag{20}$$

where S is a surface integral over the zero level set and V^+ and V^- are volume integrals over the region where ϕ is positive and negatively, respectively. The surface area and volumes of the regions separated by the zero level set are special cases of these formulas with the integrand set equal to unity.

To evaluate these integrals numerically, we use use a simple midpoint quadrature rule with smoothed approximations of the Heaviside and Dirac delta functions [1]:

$$H(x) = \begin{cases} 0 & x < -\epsilon \\ \frac{1}{2} + \frac{x}{2\epsilon} + \frac{1}{2\pi} \sin\left(\frac{\pi x}{\epsilon}\right) & -\epsilon \leq x \leq \epsilon \\ 1 & x > \epsilon \end{cases} \tag{21}$$

$$\delta(x) = \begin{cases} 0 & |x| > \epsilon \\ \frac{1}{2\epsilon} + \frac{1}{2\epsilon} \cos\left(\frac{\pi x}{\epsilon}\right) & |x| \leq \epsilon \end{cases}, \tag{22}$$

where ϵ is a tuneable parameter that determines the width of the smearing functions. Typically, ϵ is set equal to one to two times the grid spacing. While these approximations are only first-order accurate [1], they are sufficient for many purposes.

Utility Functions. In addition to the main level set method algorithms, LSMLIB provides several utility functions that support level set method calculations. Functions are available for: initializing level set functions for simple geometries, computing the maximum time step size for advection dominated problems, imposing common level set method boundary conditions, managing the computational grid, and comparing of difference between two field variables.

Further Reading. For further discussion of time-dependent level set methods algorithms, we refer the reader to *Level Sets Methods and Dynamic Implicit Surfaces* by S. Osher and R. Fedkiw. For more information about fast marching methods, we suggest that the reader consult *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science* by J. A. Sethian and *The Fast Construction of Extension Velocities in Level Set Methods* by D. Adalsteinsson and J. A. Sethian.

Special Software Features

High-Performance Through Mixed-Language Programming. Mixed-language programming is a well-established method for simultaneously achieving high computational and high programmer performance. It is commonly used in high-performance code at many of the national laboratories in the United States. In LSMLIB, high computation performance is achieved by implementing all numercical kernels in Fortran 77 or C. Numerical kernels involving computations on structured grids are written in Fortran 77, which allows it to be compiled to very fast object code. The fast marching method kernels (which require an implementation of the heap data structure) are programmed in C, which also produces relatively fast object code. These numerical kernels are collected together in the LSMLIB Toolbox and form the basic building blocks for level set method calculations throughout the library. High programmer performance is achieved by providing APIs that encapsulate the high-level components of level set method calculations. These APIs are discussed in detail in the following section.

Multiple Application Programming Interfaces. Because LSMLIB users have different computational needs, resources, and experience, LSMLIB provides multiple software interfaces. Each of these interfaces has been designed to make writing a level set method simulation as straightforward as possible while allowing sufficient flexibility to explore a wide variety of moving interface problems. High-level application programming interfaces (APIs) in C, C++, and MATLAB provide users an accessible set of tools to use for writing level set method programs.

Serial LSMLIB Package. For level set method computations on a single processor, LSMLIB provides a C-interface that contains high-level data structures and algorithmic components. For time-dependent level set method calculations, the Serial LSMLIB Package provides support for managing the computational grid, initializing level set functions, imposing boundary conditions, and using narrow-band algorithms. The Serial LSMLIB Package also provides support for time-independent level set method calculations using fast marching methods.

Parallel LSMLIB Package. For parallel level set method calculations, the Parallel LSMLIB Package provides a C++ interface that contains a collection classes which encapsulate the high-level algorithmic components of a parallel level set method computation. To reduce software complexity for the user, the C++ interface has been designed so that the user only needs to interact with four or five classes to write a level set method program. The remaining classes in the Parallel LSMLIB Package are available for advanced users who may have special needs.

As mentioned earlier, LSMLIB's the parallel capabilities are provided by the SAMRAI library. By leveraging the SAMRAI library, the Parallel LSMLIB Package is able to deliver scalable, robust support for parallel level set method computations and other functionality (*e.g.,* input and restart capabilities). In the future, we plan to add support for adaptive mesh refinement (AMR) by leveraging SAMRAI's AMR capabilities.

Because the Parallel LSMLIB Package is based on SAMRAI, LSMLIB provides parallel support for distributed memory systems based on the message passing interface (MPI) standard. No special support is available for shared memory architectures beyond any features that are provided by the MPI library that a parallel LSMLIB program is linked against.

LSMLIB MATLAB Toolbox. For rapid prototyping and moderate sized level set method calculations, LSMLIB provides a MATLAB interface that supplies several MATLAB functions that support level set method calculations. The MATLAB interface allows a user to take advantage of MATLAB's powerful data analysis and visualization capabilities to obtain results quickly and easily. High-performance of these functions is achieved by using MATLAB MEX-files to interface to LSMLIB's numerical kernels.

Standard Build Procedure. The build procedure for LSMLIB is straightforward on UNIX-based platforms because it follows the "`configure; make; make install`" procedure that is standard for UNIX software. The configure script is automatically generated using the `autoconf` tool to carry out tests to ensure that the system is compatible with LSMLIB and set the build options (*e.g.,* debug vs. optimized mode). A sample of the systems on which LSMLIB has been successfully deployed are: Linux (GCC 4.0.2), Linux (Intel compilers 8.0 & 9.0), Mac OS X (GCC 3.3, IBM XL Fortran compiler), Mac OS X (GCC 3.3, g77), and BlueGene/L (IBM XL compilers).

The hardware and software requirements for LSMLIB are flexible. Building the Parallel LSMLIB Package is optional and may be disabled using the appropriate configure option. The same is true for the LSMLIB MATLAB Toolbox.

Future Development Plans

LSMLIB is still under active development. We have plans to add several algorithmic and software features so that LSMLIB can deliver even more functionality to application developers. A few capabilities that we intend to add in the near future include: support for patch-based AMR (Parallel LSMLIB Package), improved memory usage for the narrow band method using advanced data structures [29] (Serial LSMLIB Package), and improved support for development on the Windows platform.

References

[1] S. Osher and R. Fedkiw. *Level Sets Methods and Dynamic Implicit Surfaces*. Springer-Verlag, New York, NY, 2003.

[2] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

[3] S. J. Osher and F. Santosa. Level set methods for optimization problems involving geometry and constraints I. Frequencies of a two-density inhomogeneous drum. *J. Comp. Phys.*, 171:272–288, 2001.

[4] O. Alexandrov and F. Santosa. A topology-preserving level set method for shape optimization. *J. Comp. Phys.*, 204:121–130, 2005.

[5] G. Allaire, F. Jouve, and A.-M. Toader. Structural optimization using sensitivity analysis and a level-set method. *J. Comp. Phys.*, 194:363–393, 2004.

[6] Y. Jung, K. T. Chu, and S. Torquato. A variational level set approach for surface area minimization of triply periodic surfaces. *J. Comp. Phys.*, 2006. In press.

[7] M. Prodanović and S. L. Bryant. Investigating pore scale configurations of two immiscible fluids via the level set method. In *Computational Methods in Water Resources XVI Conference*, Copenhagen, Denmark, 2006.

[8] M. Prodanović and S. L. Bryant. A level set method for determining critical curvatures for drainage and imbibition. *J. Colloid Interface Sci.*, 2006. In Press.

[9] Y. Xiang, L.-T. Cheng, D. J. Srolovitz, and W. E. A level set method for dislocation dynamics. *Acta Mater.*, 51:5499–5518, 2003.

[10] Y. Xiang, D. J. Srolovitz, L.-T. Cheng, and W. E. Level set simulations of dislocation-particle bypass mechanisms. *Acta Mater.*, 52:1745–1760, 2004.

[11] S. S. Quek, Y. Xiang, Y. W. Zhang, D. J. Srolovitz, and C. Lu. Level set simulation of dislocation dynamics in thin films. *Acta Mater.*, 54:2371–2381, 2006.

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

[13] S. O. Unverdi and G. Tryggvason. A front-tracking method for viscous, incompressible, multi-fluid flows. *J. Comp. Phys.*, 100:25–37, 1992.

[14] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A front-tracking method for the computations of multiphase flow. *J. Comp. Phys.*, 169:708–759, 2001.

[15] W. Mulder, S. Osher, and J. A. Sethian. Computing interface motion in compressible gas dynamics. *J. Comp. Phys.*, 100:209–228, 1992.

[16] R. J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhaüser, 1992.

[17] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

[18] A. Harten, B. Engquist, S. Osher, and S. Chakravarthy. Uniformly high order essentially non-oscillatory schemes, III. *J. Comp. Phys.*, 71:231–303, 1987.

[19] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *J. Comp. Phys.*, 77:439–471, 1988.

[20] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes II. *J. Comp. Phys.*, 83:32–78, 1989.

[21] S. Osher and C.-W. Shu. High order essentially non-oscillatory schemes for hamilton-jacobi equations. *SIAM J. Numer. Anal.*, 28:902–921, 1991.

[22] X.-D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *J. Comp. Phys.*, 126:202–212, 1996.

[23] G.-S. Jiang and C.-W. Shu. Efficient implementation of weighted eno schemes. *J. Comp. Phys.*, 126:202–228, 1996.

[24] G.-S. Jiang and D. Peng. Weighted eno schemes for hamilton jacobi equations. *SIAM J. Sci. Comput.*, 21:2126–2143, 2000.

[25] B. Gustafsson, H.-O. Kreiss, and Oliger. *Time Dependent Problems and Difference Methods*. Wiley-Interscience, 1996.

[26] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 1996.

[27] R. J. Spiteri and S. J. Ruuth. A new class of optimal high-order strong-stability-preserving time discretization methods. *SIAM J. Numer. Anal.*, 40:469–491, 2002.

[28] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE-based fast local level set method. *J. Comp. Phys.*, 155:410–438, 1999.

[29] M. B. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *J. Sci. Comput.*, 26:261–299, 2006.

[30] D. Adalsteinsson and J. A. Sethian. The fast construction of extension velocities in level set methods. *J. Comp. Phys.*, 148:2–22, 1999.

[31] T. .H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.