LSMLIB: Overview of Algorithm and Software Features

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 (LSM) is a mature theoretical and numerical method for using implicitly defined surfaces to study a wide range of moving interface problems [1, 2]. The Level Set Method Library (LSMLIB) is a collection of state-of-the-art LSM algorithms and numerical kernels which provides support for serial and parallel simulation of codimension-one and codimension-two 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 ($\phi_t + \vec{V} \cdot \nabla\phi = 0$) and normal velocity ($\phi_t + V_n|\nabla\phi| = 0$) 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 LSM 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 a combination of mixed-language programming techniques that make it possible to implement low-level numerical kernels in a fast language, such as Fortran or C, and by adopting advanced LSM algorithmic techniques, such as parallelization and localization [1, 2, 3]. To the best of our knowledge, LSMLIB is the only high-performance LSM library that is freely available to the general public [4].

Algorithmic Features

Time-dependent Level Set Equations. The time-dependent level set equations are solved using standard numerical algorithms for time-dependent Hamilton-Jacobi equations [1, 5, 6]. First, the spatial derivatives are computed using either an essentially non-oscillatory (ENO) [1, 7, 8, 9, 10] or weighted essentially non-oscillatory (WENO) [1, 11, 12, 13] discretization. For the vector velocity form of the equations, upwinding is used to select the appropriate approximation to the spatial derivative for the computation [1, 2, 5, 6]. For the normal velocity form of the equations, the direction of the velocity is no longer available, so Godunov's method is used [1, 2, 5, 6]. The resulting semi-discrete equations are solved using explicit total-variation diminishing (TVD) Runge-Kutta (RK) time integration [1, 8, 14]. The same approach is also used to solve time-dependent PDE-based field extension, reinitialization and orthogonalization equations.

LSMLIB provides support for first-, second-, and third-order ENO and fifth-order WENO discretizations of spatial derivatives. First-, second-, and third-order TVD-RK time integrators are supplied for time integration.

Time-independent Level Set Equations. The time-independent level set equation ($|\nabla\phi| = F(x)$), also known as the Eikonal equation, is solved using the fast marching method, which is a variation of Dijkstra's algorithm for computing the shortest path on a graph with weighted edges [1, 2]. LSMLIB also provides support for computing extension fields via the fast marching method [2, 15]. Currently, only first-order discretizations spatial derivatives are available. Support for second-order discretizations are planned for a future version of the library.

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.

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. All surface and volume integrals are cast in terms of integrals over the entire computational domain through the use of smoothed Heaviside and delta-functions.

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

Software Features

Modular Design. Modularity is a guiding principle in the design of LSMLIB's software architecture. A collection of independent numerical kernels contained in the LSMLIB Toolbox supply the basic building blocks that are used to support LSM calculations throughout the library. To achieve high-performance, these numerical kernels are written in a combination of Fortran and C. High-level application programming interfaces (APIs) based on the LSMLIB Toolbox provide a more user-accessible set of tools to use for writing LSM programs.

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 an LSM simulation as straightforward as possible while allowing sufficient flexibility to explore a wide variety of moving interface problems.

Serial LSMLIB Package. For LSM computations on a single processor, LSMLIB provides a C-interface that contains high-level data structures and algorithmic components. For time-dependent LSM 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 LSM calculations using the fast marching method.

Parallel LSMLIB Package. For parallel LSM calculations, the Parallel LSMLIB Package provides a C++ interface that contains a collection classes which encapsulate the high-level algorithmic components of a parallel LSM 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 an LSM program. The remaining classes in the Parallel LSMLIB Package are available for advanced users who may have special needs.

The parallel capabilities provided by LSMLIB are supported by the Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI) developed and maintained by the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. By leveraging the SAMRAI library, the Parallel LSMLIB Package is able to provide scalable, robust support for parallel LSM 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.

LSMLIB MATLAB Toolbox. For rapid prototyping and moderate sized LSM calculations, LSMLIB provides a MATLAB interface that supplies several MATLAB functions that support LSM 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.

Documentation. LSMLIB is extensively documented. Every function in the library has documentation on its usage contained in the header file in which it is declared. In addition, the user documentation is conveniently available in a browseable HTML version through the use of the `doxygen` software documentation tool. The HTML version of the documentation comes as part of the LSMLIB distribution and is also available online [4].

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). For the Linux systems, both 32- and 64-bit systems have been tested.

The packages that are included as part of an LSMLIB build and installaton 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 (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] 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.

[4] K. T. Chu and M. Prodanović. Level set method library (LSMLIB) software. 2005-2006. URL `http://www.princeton.edu/~ktchu/software/lsmlib/`.

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

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

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

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

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

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

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

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

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

[14] 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.

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