Bulletin of the Institute of Mathematics
Academia Sinica (New Series)
Vol. **11** (2016), No. 1, pp. 115-130

CONSERVATIVE FRONT TRACKING: THE ALGORITHM, THE RATIONALE AND THE API

R. KAUFMAN[1,a], H. LIM[1,b] AND J. GLIMM[1,c]

[1]Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794-3600, USA.

[a]E-mail: rkaufman@ams.sunysb.edu

[b]E-mail: hyulim@ams.sunysb.edu

[c]E-mail: glimm@ams.sunysb.edu

Abstract

A conservative algorithm for front tracking, previously proposed, is developed here in detail. Conservation follows from general ideas of numerical analysis. Portions of the algorithm are higher order.

Front tracking is well suited to the study of problems in turbulent mixing, in that it controls excess numerical species concentration diffusion, normally present in Eulerian finite difference codes. Conservative tracking is important to prevent a gradual drift away from correct mass balances.

We propose an Application Programming Interface (API) to mediate the insertion of front tracking into an external physics code. The main requirement upon a client code imposed by the API is a front-aware interpolation function, which is used to construct two sided states at each front point. These states are defined by interpolation or extrapolation from interior states on each side of the front. Using these, we define a time integration for front points, and a conservative time integration for interior (grid cell average solution values). We define this first for cells not cut by the front and then for cut cells also. Reference implementations of client functions will be provided for regular grid client codes.

1. Introduction

Front tracking is the technique of storing and dynamically evolving a meshed front that partitions a simulation domain into two or more regions,

Received May 4, 2015 and in revised form July 15, 2015.

AMS Subject Classification: 76F35, 76F65.

Key words and phrases: Front tracking, numerical diffusion.

each representing a different material, or physics model. The motivation for conservative front tracking is rather simple: Front tracking is the unique method presently demonstrated to avoid systematic errors in an important class of problems revolving around turbulent mixing [11, 12, 7]. The benefit to be derived from sharp resolution of interfaces and steep gradients runs through broad classes of problems, including cardiac electrophysiology [15], resin transfer molding (fiber reinforced plastic) [2, 3], primary breakup of a diesel fuel jet [1], deposition and etching in the manufacture of semiconductors [9], the tracking of cloud boundaries in meteorology [16], models of targets for high energy particle accelerators [4], and mixing models for chemically reacting flows [17].

Previous versions of the front tracking algorithm for fluid discontinuities violated conservation due to inconsistent data flows into the region from artificial states constructed near the tracked discontinuity. The primary rationale for concern with conservation arises from the fundamental importance conservation properties play in the design of numerical algorithms. Conservation becomes an important criterion in practice to prevent a possible systematic buildup of errors in problems with many simulation time steps.

Section 2 is devoted to details of the conservative front tracking algorithm. In Section 3 we describe the API, allowing insertion of this algorithm into a physics code. Conclusions are contained in a final Section 4.

2. The Conservative Front Tracking Algorithm

2.1. Overview

We start with conservative equations, typically of fluid dynamics (i.e. the Euler or Navier-Stokes equations). Solutions of these equations may have discontinuities or surface layers with large gradients (i.e., near discontinuities). Shock waves, the most notable of these, are not proposed for tracking, as shock waves are well resolved by commonly used algorithms. Rather, it is the contact waves which present persistent problems in numerical solutions. These waves express jumps in temperature, species concentration, and shear velocity. Such waves, whether discontinuous or representing steep gradients, are usually poorly resolved by Eulerian finite difference algorithms.

The conservative front tracking algorithm [5, 8] follows general principles of finite difference and finite volume numerical analysis. Consider a finite difference or finite volume space-time cell. We call the new time level its top, and the old time level its bottom. The remaining faces of the cell, with variable time, are its sides. The general idea of conservative differencing is to define the integral of the conserved solution values over the top as equal to the same integral over the bottom plus the integrals of the flux of the conserved quantities through the sides (plus any source term relevant to that cell). In this way, the sum of conserved solution variables and their dynamic change over all faces is zero, i.e. a conservation property within the cell. Conservation is achieved when the two flux integrals, evaluated from each of the two directions at a shared side of two adjacent cells, coincide. In this case, no solution variables are created or annihilated in the passage between the cells. In other words, the cells are joined conservatively.

We base the time step for finite difference stencils which do not cross the front on an assumed client directionally split conservative scheme (e.g. WENO, PPM). For the remaining grid cells (whose stencils contain discontinuities), we construct front states at the current time level (with first order accuracy only), and use these to extrapolate ghost cell states across the front. Using the same interior solver with the expanded ghost cell stencil, we again construct a time propagation to the next time level using the conservative scheme above, but using the extrapolated states for the parts of the stencil which cross the discontinuity. This calculation is conservative only for cells which do not cross the front (as using these artificial states in a cut cell is inherently non-conservative).

Finally, we recompute solution values for space time cut cells, using the cut cell conservative algorithm defined here. We define a cut cell as the portion of a grid cell that lies on a single side of the front. Although the solution may be discontinuous across the front, the flux, in the direction of the front normal, is continuous. Thus the conservation requirement of continuous fluxes across cut cell boundaries is possible to achieve.

2.2. Merger of small cut cells

The conservative flux balance law, is stated schematically as

$$\overline{u(t_{n+1})} = (|\mathrm{T}OP|)^{-1}\left[\overline{u(t_n)}|\mathrm{B}OTTOM| - \int_{\widetilde{S}} F(u)\cdot n dS\right], \qquad (2.1)$$

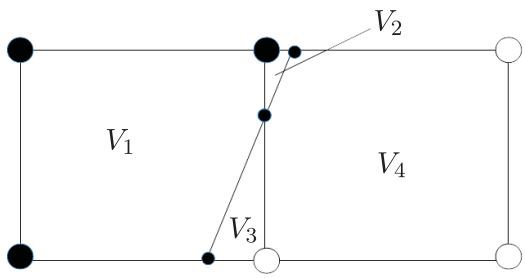

Figure 1: Volume merging of cut cells illustrated in a 2D space time example. Here the volume V_3 with no top is merged with the larger one V_4 and likewise the region V_2 with small top is merged with V_1.

where $\overline{u(t_n)}$ and $\overline{u(t_{n+1})}$ are cell averages, |TOP| and |BOTTOM| are cut cell volumes, and F is the flux as defined by the conservation law. Here $\widetilde{S}$ is the sides of a space-time cell and n is the outward normal of $\widetilde{S}$. Eq. (2.1) defines the new time level solution. We have thus advanced the solution by one time step. However, with the cut cells defined by the front, the top may be small, leading to a numerical instability or severe time step restriction. Even worse, the top may be missing, leading to an undesired constraint on the solution at the already constructed previous time level. There are several strategies to avoid the problems of small tops or no tops at all. We employ the method of merger of small cells into adjacent ones until all merged cut cells have a sufficiently large top, so that unwelcome restrictions on the algorithm CFL number are avoided. This algorithm, previously proposed [13], is illustrated in Figure 1.

2.3. Quadrature points and weights for flux integrals

We propose second order accurate quadrature, although the data is first order only. At this order of accuracy, for planar surfaces, integration over a region W is approximated by the integration area (volume) $|W|$ multiplied by evaluation of the integrand at the centroid of the integration area or volume,

$$\int_W f \cdot dA \approx f(W_{\text{centroid}}) \cdot |W| \ , \tag{2.2}$$

where the integrand f is the flux. The fluxes into the space-time cell via the top and bottom (fixed time faces) are the solution function at each time

level, whereas the flux through the variable time faces is the conservative flux. The time centroid amounts to consideration of the interface at the 1/2 time level. Now, we are in the familiar case of an interface describing a 2D surface in 3D space at a fixed time level. The front propagation to the 1/2 time level is constructed to first order only, leading to front point location errors of the order of Δx^2, because the error in location is: $|\Delta x| = O(\Delta t|\Delta v|)$ i.e. the time step times the error in velocity. This location error for the flux does not affect its stated accuracy. The fluxes at the front are constructed from front states, and these are only first order accurate, constructed by interpolation/extrapolation from 1/2 time level interior states.

Quadrature for a cut cell planar surface or full cell face follows (2.2). Surface (volume) centroids are obtained by triangulation (tetrahedralization) of the cell in question, with all triangles (tets) on a single side of the front. An elementary calculation yields the centroid of triangles (tets), and a simple average on one side of the front gives the centroid of the integration area. While this algorithm considers a large number of triangles (tets), the calculation in each (of the centroid) is elementary. The more expensive integrand evaluation is performed only once per cut cell face or surface fragment. Generalizing the integration formula (2.2) to the curvilinear surface S defined by the front itself raises a new issue. For each facet (triangle) t_i of the front, with area $|t_i|$ and centroid C_i, totally or partially within the cell, we apply the centroid method to the part lying within the cell. Then

$$\int_S f dA = \sum_i \int_{t_i} f dA \approx \sum_i |t_i| f(C_i) \ . \tag{2.3}$$

For efficiency reasons, we partition triangles into groups, and compute a single centroid and flux evaluation for the group as a whole. With grouped triangles, it is necessary to restrict the grouping size to allow a single valued projection onto a coordinate plane. Except for interfaces which are highly complicated at the cell level, a single coordinate projection will suffice for the entire front surface within a cell.

Let P be the projection of S into R^2, J the Jacobian of P. Then

$$\int_S f dA \approx |P(S)| \times f(C) \times |J(C)| \ , \tag{2.4}$$

where C is the centroid of $P(S)$.

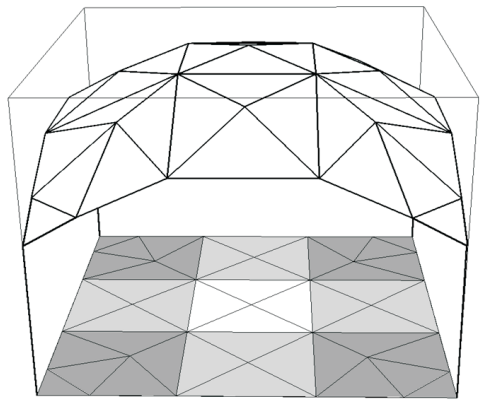

Figure 2: This figure shows the cut cell with curvilinear cut face projected into the plane (shown here in the cell bottom. Shading indicates $|J|$, the magnitude of the Jacobian of the projection. The face flux is calculated once in the plane (at the centroid) and weighted by area (times Jacobian).

2.4. Time step pseudo code

We outline the time step of the conservative front tracking algorithm in terms of high level functions, whose role is suggested by the function names. Detailed properties of the individual routines are explained in Section 3. Time step pseudocode is shown in Figure 3.

We summarize the above with a list of key client functions:

1. Find front state
2. Interior state update via ghost cells for cells not cut by the front
3. Conservative state update for cut cells

2.5. Order of accuracy

The conservative algorithm is second order accurate excluding a neighborhood of the front (due to the first order nature of the front state algorithm). An obstacle to higher order accuracy in the neighborhood of the discontinuity can be seen in trying to define the order of accuracy in a physics neutral (or problem neutral) matter. The definition of order of accuracy assumes a smooth solution of the PDE, its substitution into the difference

```
function timestep()
{
  front.point.propagation()                              /* FTI code*/
  /* to 1/2 time level for flux calculation and to next time level
                                                           as predictor */
  interior.state.update()                                /* client code */
  /* 1/2 time level and full time level for spatial cells not cut
                                                                by front */
  find.front.states.at.half.time.level()                 /* client code */
  /* 1/2 time level front states */
  flux.quadrature.at.half.time.level.cut.cells()         /* client code */
  cut.cell.interior.state.update()                       /* client code */
  /* conservative update new time level */
  find.front.states.at.new.time.level()                  /* client code */
  /* full time level front states */
  front.propagation.corrector.at.new.time.level()        /* FTI code */
  process.new.front()                                    /* FTI code */
  component.update()                                     /* client code */
}
```

Figure 3: Timestep pseudocode

scheme, and an observation of the residual in powers of Δx. Extending this definition to piecewise smooth solutions with jump discontinuities, we could imagine an order of accuracy for the discontinuity surface, and the standard order of accuracy definition applied to solutions smooth on each side of the discontinuity. However, extension of these ideas to smooth solutions with accuracy uniform in the presence of arbitrarily large gradients would require specification of the gradient magnitude in the trial solution, and thus remove the analysis from a physics neutral plane to a highly context specific one; not a desired step. Thus, our approach is less fundamental, and we only discuss orders of accuracy in substeps, and according to conventional notions of accuracy. For example, the flux across the cut cell boundary is continuous and single valued. At the level of the Euler equations, it is also bounded, but for solutions of the Navier-Stokes equation, arbitrarily large flux values can be observed in the initial time steps of resolution of a sharp discontinuity. A second order flux quadrature is not accurate uniformly in the presence of large flux values.

3. FTI: The Front Tracking Interface

FTI (Front Tracking Interface) is the name of the API which implements the standard front tracking components discussed here. The idea of an API is to abstract from a software package those features which a user (client) must interact with. For the API to be useful, this list should be short. FTI version 0.1 is the reconstruction of our own code into a client-server model linked with an API. The programmer's manual for FTI Version 0.1 can be found at `http://www.ams.sunysb.edu/fti`

We are in the process of extending this version to the HEDP code FLASH. FTI Version 1.0 is the conservative version, as discussed here. The API is written in C++, but we expect it to interoperate with programming languages that can be combined with C++. For example, the FLASH code is written in Fortran. It is parallelied using MPI.

FTI routines are divided between client and server routines. The client must write the client routines, while calling the server routines. This design allows the front tracking method to be applied regardless of client data structures or physics, while common functionality (for front dynamics) comes packaged in the server. For client routines, we also offer a reference implementation based on a uniform mesh for solution data. Extensions to AMR box based grids will be similar.

3.1. Client routines

The key client routines are

1. find front states
2. interior states via ghost states
3. conservative cut cell state update

Since FTI cannot know or enforce conditions on the client data model, it is the task of the client to generate front states. The front states are the crux of the coupling between the interior dynamics and the front dynamics.

Although the FTI server can calculate the component at any point, this call is expensive. For the purpose of storing components as well as identifying cut cells we impose two data structures on the client

1. the crossing grid, and

2. the component grid

The crossing grid will store the front elements which cut a cell in each cell (empty for uncut cells). The component grid stores the component indicator for each cell. These can be updated efficiently by the set component (§3.1.1) function, but the details of implementing of the data structure are left open.

The client API routines, needed in the dynamic update are developed in the following sections:

3.1.1. Set components

Input: Front at current time level

Output: Component label set for each cell not crossing the front. Label ONFRONT for cut cells.

Synopsis: Assuming the front is closed, that is, assuming that there is no outer edge to the front surface other than on the outer surface of the computational region, the front divides the computational domain into connected components. See Figure 4. A cell's component can be determined based on which side of the front it is on. This can be determined by the nearest front element, though this lookup is expensive, and so we perform it only once per component region. The remaining cells have their components determined as follows: We store the list of all triangles from the front which lie in each mesh block. Starting from a cell with a known component (via nearest element), we employ the method of marching front method. This algorithm is inverse to the Marching Cubes [14] algorithm, in that it takes the front location data and uses this to mark each cell. The marching front propagates component information of a cell to its neighbors until a front is reached.

3.1.2. Find front state

Input: Component labels and state variables defined at every cell for the current time level. For cut cells, the component is set to a special label,

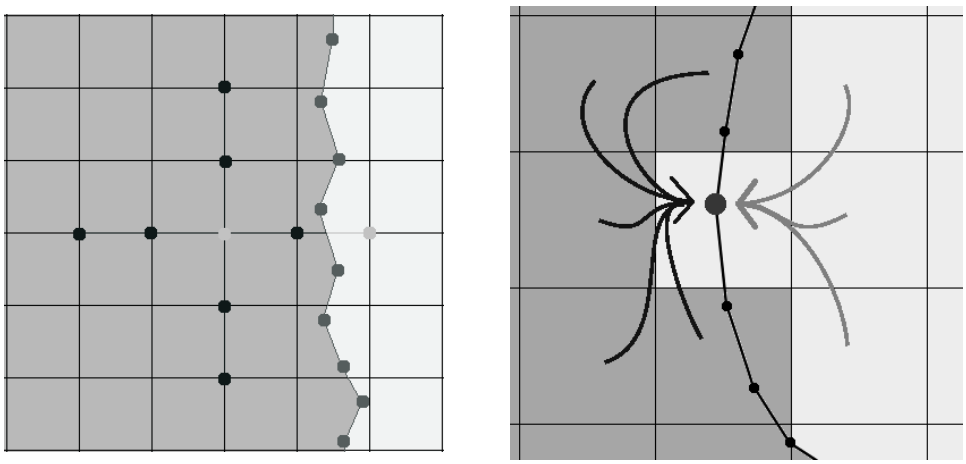

Figure 4: Illustrations of a component grid. Cells are shaded based on component. Left shows a front crossing solution stencil on the dual to the simulation grid, so component and state information is defined at cell corners. Right shows front-aware interpolation on a simulation grid with states and components on cell centers. Only cells of like component are used. Each component meeting at the front gives one of the two front states defined on the front.

ONFRONT, and the state represents the solution associated with the component to which that the cell center belongs.

Output: Two sided states and a single valued front normal velocity are computed at every front point. Assuming that the physics is a contact discontinuity with zero surface tension or a concentration/thermal/shear velocity iso-surface, the pressure and the normal velocity are identical for these two states.

Synopsis: The interpolation/extrapolation is illustrated conceptually in Figure 4. One sided interpolation defines a pair of outer front states. See Figure 5, where the construction is illustrated in 2D. These are input to a Riemann solver, which returns the midstates associate with the contact discontinuity. We call these mid states the (inner) front states. For the purpose of finding a front normal velocity, the outer front states are defined by linear or constant extrapolation, depending on the number of nearby grid cell centers with the desired component value. The front normal n is constructed at the front point, using higher order accuracy ideas from the

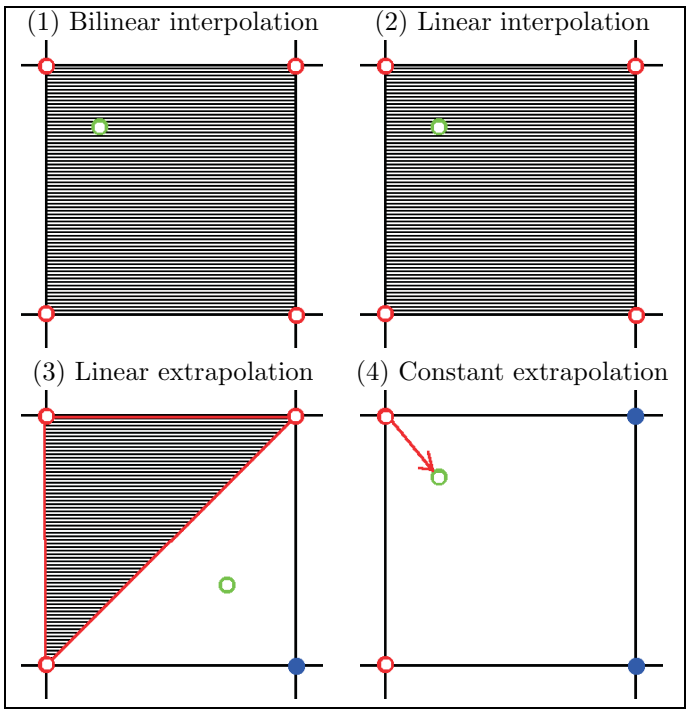

Figure 5: Cases of the component aware interpolation model in 2D. The shaded area shows the interpolation or extrapolation element. Case 1 shows a bilinear interpolation case. Case 2 shows a linear interpolation case. Case 3 shows a linear extrapolation case. Case 4 shows a constant extrapolation case.

interface geometry program of Jiao and students [10] and is provided by the FTI server.

3.1.3. Interior state update

Input: A directionally split 1D solver, with specified stencil size and states defined for all grid cell centers. The front at the current time level is also required.

Output: States defined at the new time level for all grid cell centers. Except for cut cells, this definition is conservative. Except near the front, the update has the order of accuracy of the client solver.

Synopsis: We begin with the identification of all locations where a line through grid cell centers crosses the front. We construct two sided front

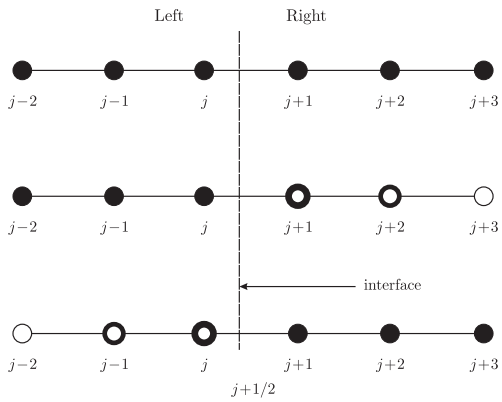

Figure 6: Ghost cell extrapolation. Above, original stencil, crossing the interface. Middle: Two right states are replaced by left side extrapolated states to define a stencil consisting of only left states. Below: a similar construction for the right sided stencil.

states at these crossing points (the front state algorithm reduces to constant extrapolation along the grid cell center line in this case) to define outer front states. The outer front states are input to a Riemann solution, whose mid states define the (inner) front states. Any stencil crossing the front is extended on the other side of the front by constant extrapolation from the associated front state (ghost cell extrapolation) [6]. See Figure 6. Using these modified stencils, the client solver returns a time advanced state for the new time level.

3.1.4. Conservative update for cut cells

Input: States defined for all cell centers and front points for the current and 1/2 time level, and the propagated front (predictor only) defined at the current, 1/2 time and new time levels.

Output: Conservative values for cut cell state averages at the new time level.

Synopsis: The main ideas derive from general conservation principles as explained in Sec. 2.3, Eq. (2.1). The algorithm makes use of the interior (non-cut cell) states and the nonconservative interior cut cell states, both at the 1/2 time level from Sec. to define front states via Sec. 3.1.2. The

flux is computed from the conservation law, with quadrature as discussed in Sec. 2.3. The algorithm for merging small cells is discussed in Sec. 2.2.

Outline of steps:

1. Merge Small Tops
2. Triangulate Cut Space-Time Cell Faces
3. Find Centroids
4. Surface Projection onto Face
5. Face Flux Calculation

3.2. FTI routines

3.2.1. Point propagate predictor

Input: A front point, Two sided states at this point and the time step size Δt.

Output: The front point position at the new time level.

Synopsis: We construct the normal n to the front at this point and advance the front in time using the first order Euler algorithm, to the new position incremented by $\Delta t(n \cdot v)$.

3.2.2. Point propagate corrector

Input: A predictor front position at the new time level, a pair of front states at this location, the time step size and the predictor velocity and position at the old time level.

Output: A corrected front position at the new time level.

Synopsis: The corrector front position is the average of the front (predictor) positions derived from the velocities at the old and new time levels. Together with the point propagate predictor, this makes a second order Runge-Kutta update to the front position.

3.2.3. Process front

Input: A propagated front at a new time level.

Output: Corrections to the propagated front.

Synopsis: Error checking, error correcting, topological changes, and mesh smoothing operations are performed. Tests for large, small or bad aspect ratio triangles are performed, and these are divided or merged with neighbors to improve the mesh quality. Additionally, the front points are moved on the surface to further improve the mesh quality. Self intersections of the front are a propagation error which is corrected a posteriori. The test is by straightforward determination of intersections for pairs of triangles. The order of the algorithm is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ by testing only pairs of triangles intersecting a common mesh block. Resolution of the self intersections is physics model dependent. Implemented is an algorithm causing merger of separate tracked regions upon overlap or contact of distinct part of the interface. The basic idea is to remove a small box containing the overlap or intersecting region, and to retriangulate this using only the component information at grid cell corners (grid based reconstruction). The gap between the box and the front totally outside of the box is then filled by a sequential construction, as discussed in [1].

3.2.3. Utility routines

Additionally, the FTI offers a few client utility routines, for example to initialize and to print.

4. Conclusions

The value of front tracking to enhance the quality of Eulerian computations with free surfaces or "steep gradient" iso-concentration, iso-thermal or iso-shear surfaces is generally recognized. The algorithm belongs to the ALE (Arbitrary Lagrangian Eulerian) family in addressing interface problems, and it is in a sense the ultimate ALE algorithm, with the Lagrangian portion reduced to a dynamic surface. For this reason, it is better able to survive late time surface distortions. The front tracking method grants an

improvement in solution quality by maintaining sharpness without loss of accuracy or conservation.

Implementation of front tracking into physics codes is a daunting endeavor, and the purpose of the present paper is to address just this implementation issue. We codify the interface between front tracking and a client code. Only three key client steps are required, and for these reference implementations are provided in the case of regular grids. These are (a) construction of front states from the existing interior states at a fixed time level, (b) construction of ghost cell states to allow conservative time integration for cells near the front but not crossing it and (c) a conservative algorithm for time integration of cells cut by the front.

Acknowledgments

This work is supported in part by Leland Stanford Junior University (subaward with DOE as prime sponsor), the U.S. Department of Energy via Los Alamos National Laboratory contract number 228022, the Army Research Organization grant number W911NF1310249 and the National Science Foundation grant number 0926190.

References

1. W. Bo, X. Liu, J. Glimm, and X. Li, A robust front tracking method: Verification and application to simulation of the primary breakup of a liquid jet, *SIAM J. Sci. Comput.*, **33** (2011), 1505-1524.

2. W. K. Chui, J. Glimm, F. M. Tangerman, A. P. Jardine and J. S. Madsen, Modeling of resin transfer molding, in *Proceedings of the First Regional Symposium on Manufacturing Science and Technology*, Stony Brook, New York, 1995.

3. W. K. Chui, J. Glimm, F. M. Tangerman, A. P. Jardine, J. S. Madsen, T. M. Donnellan and R. Leek, Process modeling in resin transfer molding as a method to enhance product quality, *SIAM Review*, **39** (1997), 714-727.

4. J. Glimm, H. Kirk, X. L. Li, J. Pinezich, R. Samulyak and N. Simos, Simulation of 3D fluid jets with application to the Muon Collider target design, in *Advances in Fluid Mechanics III*, M. Rahman and C. A. Brebbia, eds., vol. 26, WIT Press, Southampton, Boston, 2000, 191-200.

5. J. Glimm, X.-L. Li, Y.-J. Liu, Z. L. Xu and N. Zhao, Conservative front tracking with improved accuracy, *SIAMJNA*, **41** (2003), 1926-1947.

6. J. Glimm, D. Marchesin and O. McCryan, Statistical Fluid Dynamics: Unstable Fingers, *Comm. Math. Phys.*, **74** (1980), 1-13.

7. J. Glimm, D. H. Sharp, T. Kaman and H. Lim, New directions for Rayleigh- Taylor mixing, *Phil. Trans. R. Soc. A*, **371**(2013), p.20120183. Los Alamos National Laboratory Preprint LA-UR 11-00423 and Stony Brook University Preprint SUNYSB-AMS-11-01.

8. J. Glimm, D. H. Sharp, H. Lim, R. Kaufman, and W. Hu, Euler equation existence, non-uniqueness and mesh converged statistics, *Phil. Trans. R. Soc. A*, **373** (2015), p.20140282. Los Alamos National Laboratory Preprint LA-UR-14-29521 and Stony Brook University Preprint SUNYSB-AMS-14-04.

9. J. Glimm, S. R. Simanca, D. C. Tan, F. M. Tangerman and G. VanDerWoude, Front tracking simulations of ion deposition and resputtering, *SIAM J. Sci. Comput.*, **20** (1999), 1905-1920.

10. X. Jiao and D. Wang, Reconstructing high-order surfaces for meshing, *Engineering with Computers*, **28** (2012), 361-373.

11. H. Lim, J. Iwerks, J. Glimm and D. H. Sharp, Nonideal Rayleigh-Taylor mixing, *Proc. Nat. Acad. Sci.*, **107** (2010), No. 29, 12786-12792. Stony Brook University Preprint SUNYSB-AMS-09-05 and Los Alamos National Laboratory Preprint LA-UR 09-06333.

12. H. Lim, J. Iwerks, Y. Yu, J. Glimm, and D. H. Sharp, Verification and validation of a method for the simulation of turbulent mixing, *Physica Scripta*, **T142** (2010), p.014014. Stony Brook University Preprint SUNYSB-AMS-09-07 and Los Alamos National Laboratory Preprint LA-UR 09-07240.

13. J. Liu, H.-K. Lim, J. Glimm and X. Li, A conservative front tracking method in N-dimensions, *J. Sci. Comp.*, **31** (2007), 213-236. Stony Brook University preprint number SUNYSB-AMS-06-04.

14. W. E. Lorensen and H. E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, *Computer Graphics*, **21** (1987), 163-169.

15. S. Xue, A sharp boundary model for electrocardiac simulations, PhD thesis, State Univ. of New York at Stony Brook, 2015.

16. Ya-Ting Huang, A stochastic formulation of short term cloud cover forecasts, PhD thesis, state university of New York at Stony Brook, 2015. in preparation.

17. Y. Zhou, N. Ray, H. Lim, S. Wang, V. F. de Almeida, J. Glimm, X.-L. Li and X. Jiao, Development of a front tracking method for two-phase micromixing of incompressible viscous uids with interfacial tension in solvent extraction, Technical Report ORNL/TM-2012/28, Oak Ridge National Laboratory, 2012.