SIAM J. SCI. COMPUT.
Vol. 19, No. 3, pp. 703–727, May 1998



# THREE-DIMENSIONAL FRONT TRACKING*

JAMES GLIMM†, JOHN W. GROVE†, XIAO LIN LI‡, KEH-MING SHYUE§, YANNI ZENG†, AND QIANG ZHANG†

**Abstract.** We describe a three-dimensional front tracking algorithm, discuss its numerical implementation, and present studies to validate the correctness of this approach. Based on the results of the two-dimensional computations, we expect three-dimensional front tracking to significantly improve computational efficiencies for problems dominated by discontinuities. In some cases, for which the interface computations display considerable numerical sensitivity, we expect a greatly enhanced capability.

**Key words.** front tracking, Riemann problems, nonmanifold geometry

**AMS subject classifications.** 35L65, 35L67, 65M99

**PII.** S1064827595293600

## 1. Introduction.

Front tracking is a numerical method in which surfaces of discontinuity are given explicit computational degrees of freedom; these degrees of freedom are supplemented by degrees of freedom representing continuous solution values at regular grid points. This method is ideal for solutions in which discontinuities are an important feature, and especially where their accurate computation is difficult by other methods. Computational continuum mechanics abounds in such problems, which include phase transition boundaries, flame fronts, material boundaries, slip surfaces, shear bands, and shock waves. The method was initiated by Richtmyer and Morton [55] and was used for high quality aerodynamic computations by Moretti, Grossman, and Marconi [50, 51, 52].

A systematic development of front tracking in two dimensions has been carried out by the authors and their coworkers [20, 12, 21, 22, 19, 10, 29]. See [40, 61, 42, 11, 2, 45, 46] and additional references in the survey [37] for other approaches to front tracking in two dimensions. Special purpose front tracking codes have also been developed, for example, for simulation of the deposition and etching process for the manufacture of semiconductors [31]. Computer-aided design packages for solid geometry use similar concepts, under the terminology of nonmanifold geometry. There are also a number of one-dimensional front tracking codes [41, 33, 60, 9, 45, 56, 34].

The first conclusion to emerge from this body of work is that it is possible to apply front tracking in a systematic fashion to complex shock or wave front interaction problems, including problems with bifurcations, with changes of wave front topology, as occurs after interaction, or crossing of one wave (tracked discontinuity) by another. In other words, the first conclusion is that front tracking is a feasible

*Received by the editors October 16, 1995; accepted for publication (in revised form) June 14, 1996. This work was supported by the Applied Mathematics Subprogram of the U.S. Department of Energy under grant DE-FG02-90ER25084, by the Army Research Office under grant DAAH04-95-10414, and through the Mathematical Sciences Institute of Cornell University under subcontract to the State University of New York at Stony Brook (ARO contract number DAAH-04-95-10414), by the National Science Foundation grants DMS-9500568 and DMS-9057429, and by the U.S. Department of Energy.

http://www.siam.org/journals/sisc/19-3/29360.html

†Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, Stony Brook, NY 11794-3600 (glimm@ams.sunysb.edu, grove@ams.sunysb.edu, zeng@ams.sunysb.edu, zhang@ams.sunysb.edu).

‡Department of Mathematics, Indiana University–Purdue University, Indianapolis, IN 46202.

§Department of Mathematics, National Taiwan University, Taipei, Taiwan.

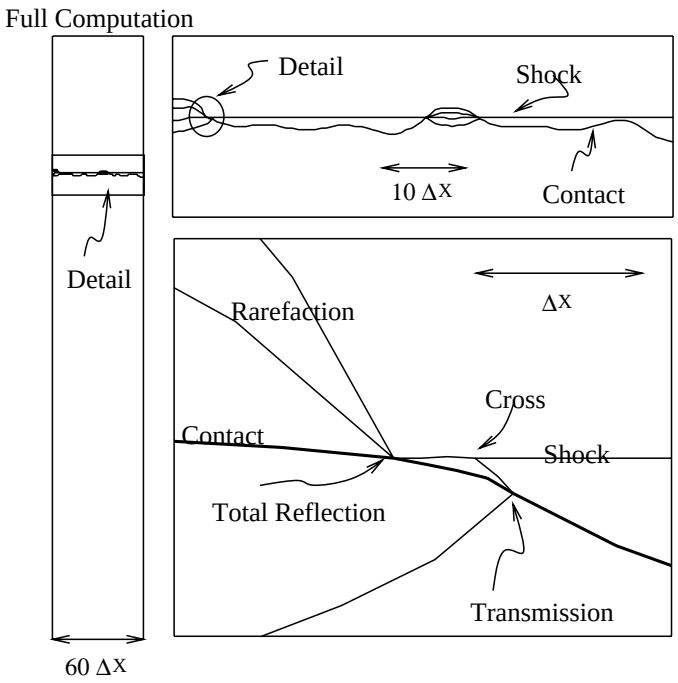



FIG. 1.1. *Tracked waves to represent the passage of a shock wave through a perturbed planar interface separating two compressible fluids of different densities. The three frames show successively zoomed enlargements of a detailed region of shock-contact interaction within a single time step in the interaction.*

method for problems with geometrically complex fronts. The second conclusion is that the front tracking solutions are often (a) better and (b) obtained on significantly coarser grids [26, 36, 30, 4, 10, 6, 7]. Included in the above-cited results are the following: (a) the first simulation of the Rayleigh–Taylor instability which agrees with laboratory data in the incompressible limit and is extensible to the compressible case; (b) the first simulation of the Richtmyer–Meshkov instability to agree with laboratory results for the growth rate of the instability; (c) the best simulation of the inviscid transition between regular reflection and Mach reflection, for the shock on ramp problem.

In Figure 1.1, we show the interaction of a shock wave with a randomly perturbed planar contact discontinuity in two dimensions, representing a density discontinuity layer between two gases. The original computation and two levels of zoomed enlargement are displayed in the three panels. The most enlarged panel occupies only a few mesh blocks and shows a highly resolved and complex set of incident, reflected, and transmitted waves.

The success of two-dimensional front tracking and the intrinsic importance of three-dimensional fluid dynamics provide the motivation for the present paper, whose purpose is to present algorithms and methods for front tracking in three dimensions, and to validate an implementation of these algorithms through demonstration of convergence under mesh refinement for a test problem. The test problem is the small amplitude growth rate for a fluid instability; we take the Rayleigh–Taylor acceleration-driven instability as an example, due to our prior experience with this problem. We also compare our results to those of the level set method.

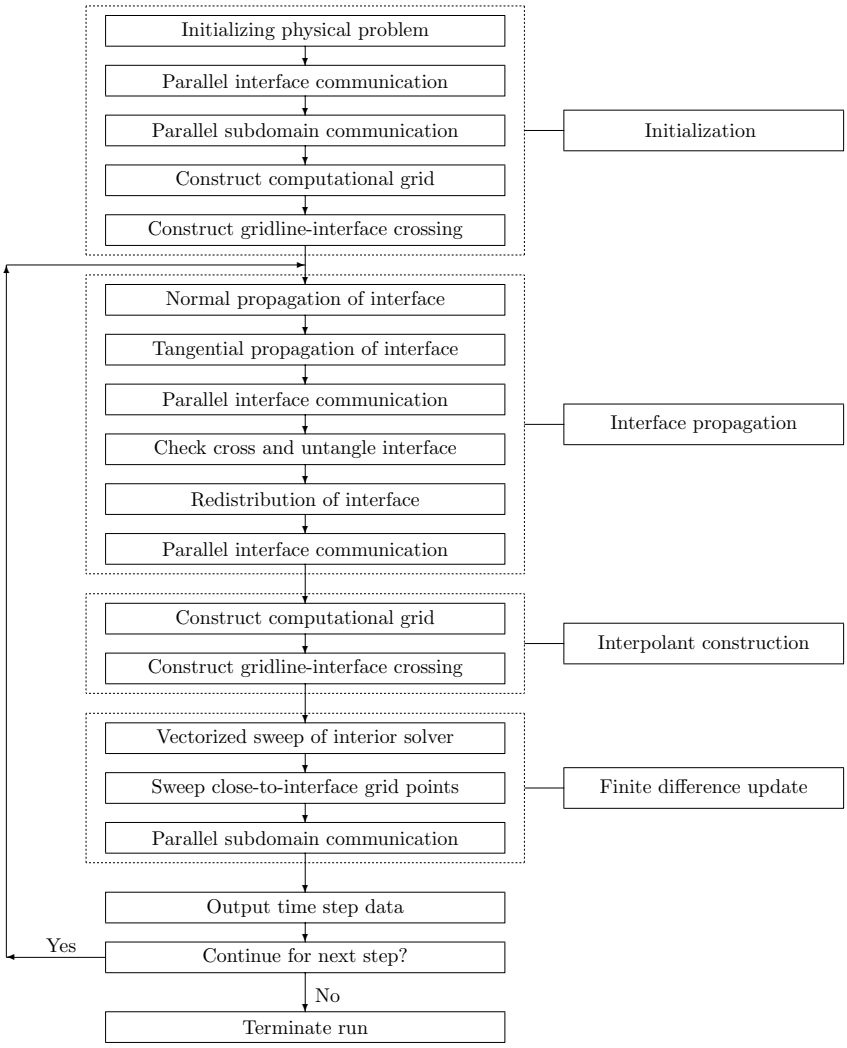



FIG. 1.2. *Flow chart for the front tracking computation. With the exception of the i/o and the sweep and communication of interior points, all solution steps indicated here are specific to the front tracking algorithm itself.*

As a framework for the rest of the paper, we present in Figure 1.2 a flow chart for the front tracking computation. A more detailed account of the method of front tracking can be found in [35, 3].

**2. Modularity and data structures.** The use of modern programming languages and modular organization has been an integral part of our methodology for many years, and is an essential part of the work presented here. The front tracking code is organized into a modular set of data classes that allow the code to be used in a variety of different applications, including compressible gas dynamics, elastoplastic flows, flow in porous media, and resin injection molding. The physical processes in these applications are quite different, but they all share the common property that sharp waves play a critical role. The data structures of the front tracking code are

organized to maximize the amount of code that can be common to these and other applications. Due to the growing interest in object-oriented programming and modular algorithm design, we include a discussion of the data organization aspect of our methods. The discussion is organized by increasing levels of specificity:

1. utilities and software tools;
2. geometrical and topological structures (e.g., grids and interfaces);
3. general equation and problem types and solution methods: hyperbolic, parabolic, elliptic (e.g., Godunov, ADI, conjugate gradient, finite elements, interpolation). As such, routines at this level can copy (bitwise) and allocate storage for dependent variables and pass them as arguments of functions;
4. physical laws: compressible gas dynamics, elastic-plastic flow, etc.;
5. material specification (e.g., equations of state or constitutive laws).

Each organizational level consists of a set of libraries that contain functions for the computation and manipulation of data objects at that level. The different levels form a hierarchy in which data objects at higher levels can inherit and extend properties of objects at lower levels, while objects at lower levels have no direct knowledge of the existence or properties of the higher level objects. For example, the data structure describing a point is simply a geometric position when viewed from a level 2 library, while the same data structure when extended to level 4 describes a point on a wave surface and carries with it a description of the flow state on either side of the surface. Public data, functions, and structures defined at one level are available to all higher specificity levels but not to lower specificity levels. Indirect access to higher level functions is provided though the use of virtual functions. It is also possible for a particular level to be divided into sublevels. For example, in the front tracking code, level 3 is subdivided into four sublevels corresponding to the propagation of tracked waves, the generation of interpolation grids, the generation of finite difference stencils, and a driver section. It is also possible to define higher levels. For example, the elastoplasticity code is built as a superlevel of gas dynamics.

**2.1. The interface library.** We first discuss the interface library, a level 2 library which describes the geometry and topology of piecewise smooth manifolds with piecewise smooth boundaries, embedded in $R^3$. Boundary and coboundary operators, to map from a manifold to its boundary and to the manifolds which it bounds, are included in this library. The library compiles and runs independently of the levels above it. We begin with a description of the main data structures (whose names are in capital letters) and their interrelationships. At a continuum level, an INTERFACE [23] is a collection of nonintersecting geometric objects, NODEs, CURVEs, and SURFACEs, that correspond to zero-, one-, or two-dimensional manifolds, respectively. The dimension `dim` of an INTERFACE is a run time variable defined as the dimension of the embedding space that contains the interface. The front tracking code currently supports one-, two-, and three-dimensional INTERFACEs. Both CURVEs and SURFACEs are oriented manifolds. NODEs correspond to boundary points of CURVEs, while in $R^3$, CURVEs correspond to the boundaries of SURFACEs. Each geometric object is associated with a corresponding object, HYPERSURFACE or HYPERSURFACE BOUNDARY, depending on its codimension. Thus in $R^2$ a CURVE is associated with a HYPERSURFACE, and a NODE with a HYPERSURFACE BOUNDARY, while in $R^3$, HYPERSURFACEs correspond to SURFACEs, and HYPERSURFACE BOUNDARIES correspond to CURVEs. The geometric and codimension data structures are linked by pointers that map from one object to the other. We designate as COMPONENT some labeling scheme, i.e., equivalence class, for the

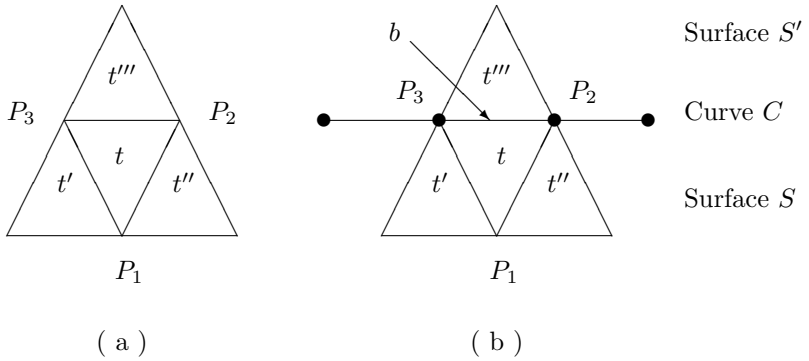



FIG. 2.1. *The neighbor structure of a triangle $t$. The triangle has pointers to the three points $P_1$, $P_2$, $P_3$ at its corners. In case* (a), *the triangle is interior to the surface, and its data structure contains pointers to its three neighboring triangles, $t'$, $t''$, $t'''$. In case* (b), *the triangle is adjacent to a boundary curve $C$ of the surface $S$ it belongs to. The triangle $t$ still has pointers to neighbors $t'$ and $t''$, but now the bond $b$ replaces the neighbor $t'''$. The bond $b$ has pointers to both $t$ and $t'''$.*

connected components produced by the HYPERSURFACEs. Thus several connected components may share a common component label and constitute a single COMPONENT. Each side of a HYPERSURFACE is labeled by the COMPONENT adjacent to that side. The COMPONENT label for a given geometric position is then defined by the component on the side of the nearest HYPERSURFACE to that position. The mapping between manifolds and boundaries consists of pointers from each respective object to its corresponding boundary or coboundary object. Thus NODEs have a list of incoming and outgoing CURVEs at the NODE, CURVEs contain the address of their start and end NODEs, SURFACEs have pointers to their bounding CURVEs, and CURVEs contain pointers to the SURFACEs that they bound. The boundary of a SURFACE may consist of several CURVEs, and each CURVE may bound several SURFACEs (or none at all). In cases where the components on the two sides of the SURFACE are the same, the side of the SURFACE is useful as a local generalization of COMPONENT.

The discretized version of the INTERFACE has the same structure, with a piecewise linear description built from doubly linked lists of simplices of the appropriate dimensions. The CURVEs are composed of BONDs; the linking order corresponds to the natural order of BONDS along the CURVE. Each BOND is a pair of POINTs, and (conceptually) the straight line segment joining them. SURFACEs are discretized in terms of TRIANGLEs. In contrast to the BONDs, their linking order has no intrinsic relation to the geometry of the surface. Figure 2.1 shows the neighbor structure of TRIANGLEs.

In Figure 2.2, we illustrate the geometric data structures used for the front tracking method in three dimensions.

A "next-point" algorithm is provided, which, after a special initialization call, will return, on successive calls, each POINT in succession. This algorithm is elementary in two dimensions and results from traversing the BONDs of each CURVE using the linked list order. As each BOND is considered, its initial POINT (which is also the final POINT of the previous BOND) will already have been considered, so the final POINT of the current BOND is the returned value. This elementary idea fails in three dimensions, as POINTs are redundantly referenced in TRIANGLEs, and the arbitrary nature of the linking order for TRIANGLEs does not allow a simple determination of

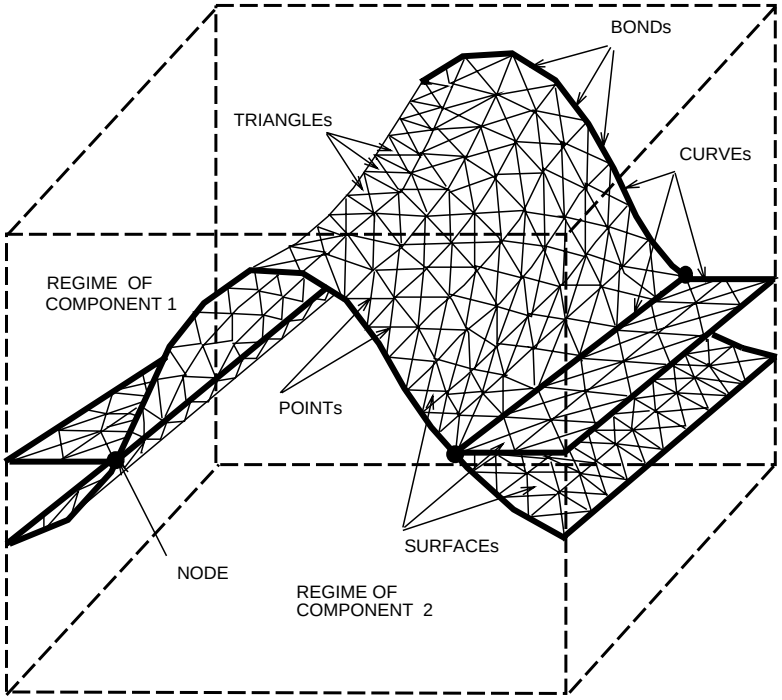



FIG. 2.2. *An illustration of the geometric data structures used for the front tracking method in three dimensions.*

previously considered POINTs of the current TRIANGLE. The initialization call to the next point in three dimensions is used to sort POINTs and TRIANGLEs into an array. Hidden storage with sorting information in the triangle and point structure are used to support the successive calls to the next-point and next-triangle algorithms in three dimensions.

An INTERFACE, in the C language, is a class structure with arrays of pointers to SURFACEs, CURVEs, and NODEs, together with a set of operations for the manipulation of these objects. The SURFACEs, CURVEs, and NODEs are also class structures. They contain arrays of pointers to their bounding and cobounding objects, i.e., arrays of pointers to CURVEs, for the SURFACEs, etc., together with various operators for their manipulation. The SURFACEs and CURVEs also contain reference to the first and last simplices (TRIANGLEs and BONDs) in the linked list that defines them.

The TRIANGLEs and BONDs are data structures defined in terms of POINTs and neighbors (adjacent TRIANGLEs or BONDs). For computational efficiency, they contain additional information, namely, length for BONDs and area and positive unit normal for TRIANGLE. A POINT is also a data structure, with data to represent its coordinate description.

To achieve modularity between dimensions, the coordinates of a point are represented as an array of `dim` $=$ 1, 2, or 3 floating point variables. Since one of the applications of the interface library is to support the efficient interpolation of piecewise smooth functions, it would be beneficial to allow arbitrary positive integer values for the dimension `dim`. In this case, the interface is a generalization of the notion of

a simplicial complex [16] of dimension $\texttt{dim} - 1$ embedded in $R^{\texttt{dim}}$. Multiple INTERFACEs and their uses, even within a single computation, prevent `dim` from being a globally defined variable.

Each of the elementary objects in an INTERFACE (including the INTERFACE itself) has support routines for initializing, copying, printing, reading of printed format, and modifying. These routines are exported and publicly available to the rest of the code. In addition, the INTERFACE has hidden, or private, data and support functions.

The interface library supports its own storage allocation scheme. Storage allocation is a level 1 module, built as an extension of the Unix routine `malloc`. As a hidden variable, a linked list of all active interfaces is maintained, and for each a separate instance of the level 1 storage allocation scheme is maintained. Storage for an interface is allocated in blocks of a designated size. These are used as needed for the allocation of SURFACEs, etc. In this way, the storage is (more nearly) contiguous in physical memory, and paging inefficiencies for access of computationally related variables are minimized. Deleted objects are not deallocated; rather the knowledge of their addresses is eliminated, so that they are deaddressed. The reason for this choice is that available deaddressed space is highly fragmented. Much of the data consists of pointers, so that compression by recopy of data is incorrect. Upon deletion of the entire INTERFACE, all of its storage is deallocated and returned to the system. The combination of copy INTERFACE to get a new INTERFACE and delete (old) INTERFACE will free deaddressed storage, reset pointers correctly, and accomplish compression. Storage allocation is a private aspect of publicly available functions for initialization of INTERFACE objects.

It is frequently necessary to determine the topology associated with an INTERFACE. INTERFACEs are required to be non-self-intersecting (so that SURFACEs can meet only along their bounding CURVEs, etc.). After each time step in the dynamical evolution, it is necessary to check the propagated INTERFACE for intersections. If intersections arise, signaling a bifurcation of the topology, a call will be made to a physics-specific routine, to modify and reconnect the INTERFACE, with the possible introduction of additional scattered waves, as required by the physics. A second topological requirement is to determine the COMPONENT of a given location in space.

These requirements lead to a private INTERFACE data structure of hashed lists of hypersurface simplices (BONDs or TRIANGLEs) stored according to their intersection with each mesh cell. The mesh used here is unrelated to any other (finite difference) mesh used in the computation; we call it the topological GRID. See Figure 2.3. GRID defines a regular mesh over a brick in space. Its objects are pointers to arrays of `dim` numbers, so that GRID is independent of dimension. The creation (allocation) of a GRID thus requires dynamic storage allocation. The GRID contains the upper and lower boundaries of the brick, the number of mesh points, and the mesh spacing in each coordinate direction. In order to support ghost cells in parallel computing domain decomposition, the GRID also contains upper and lower offset mesh boundaries for the location of the ghost cell boundaries of the brick. In addition to the topological GRID, there is a finite difference GRID used for storage of state values (at cell centers) and for discretization of derivatives. Another GRID, dual to the finite difference GRID, is used for interpolation (section 2.3).

The intersection routine checks all pairs of hypersurface simplices for intersections and returns a hashed list of intersecting hypersurfaces and their intersection

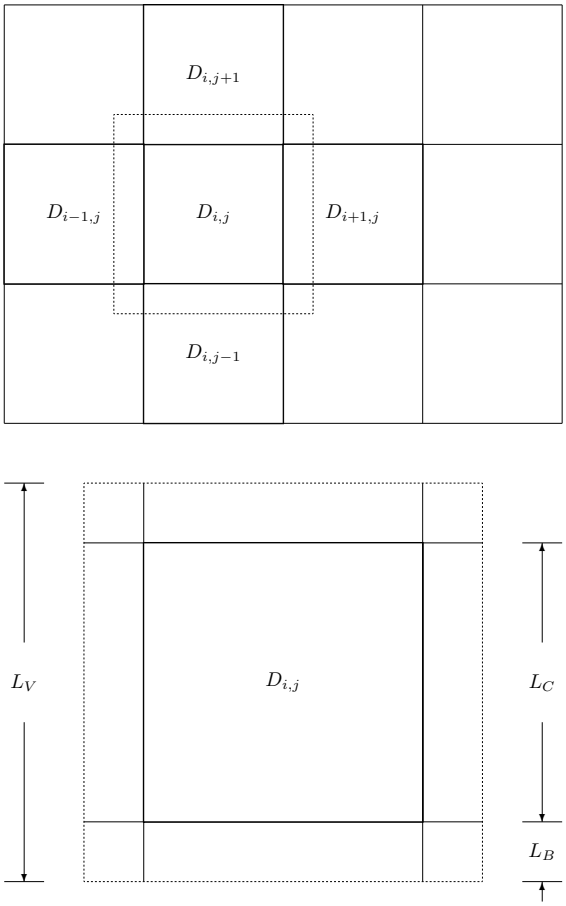



FIG. 2.3. *Illustration of the GRID data structure in two dimensions, for description of single-processor grid information, in a domain decomposition parallel algorithm. The GRID contains mesh index ranges, grid spacing, and (redundantly) absolute spatial coordinates for upper and lower boundaries of the computational domain and of associated ghost cell buffer domains which refer to cells controlled by neighboring processors but needed by the current processor. Here $L_C$ is the length of the computational domain, $L_B$ is the length of the buffers, and $L_V$ is the length of the virtual domain, including both the computational domain and the buffers.*

locations. In three dimensions, the intersections are organized into CURVEs, while in two dimensions, they are isolated NODEs. By use of this hashed list, intersections are tested only for pairs meeting a common mesh block. Since the local density of hypersurface elements is normally bounded, the $O(n^2)$ intersection computation is reduced to $O(n)$ in complexity for typical problems.

In addition, these hashed lists are used to support a projection algorithm, which will find the closest interface point (i.e., the TRIANGLE, side, and the closest point of the TRIANGLE) to a given point in space. To find the COMPONENT associated with a given point of space, it is first located in a topological grid mesh cell. If the cell is regular (i.e., it does not intersect the interface), then precomputed information will give the COMPONENT by table lookup. If the cell is irregular, then the closest side, given by the above projection algorithm, determines the COMPONENT. The

precomputation of COMPONENTs starts with irregular cells. In a neighbor cell which is regular, the projection algorithm will determine the COMPONENT. These values are then extended by transitivity, since adjacent regular cells must share the same COMPONENT.

A general discussion of data structure design and algorithms for sorting, queuing, searching, hashing, matching, merging, splitting, etc., as well as complexity analysis, can be found in [1]. Algorithms in computational geometry involving geometric searching, triangulation, detection of intersections, etc. are discussed in [54].

**2.2. Front.** An INTERFACE, together with generic physics-dependent information and "black box" dynamics, is called a FRONT. The resulting algorithms define a level 3 library. At level 3, a new object, called F_POINT, is introduced that inherits the properties of a POINT and acquires new structure: physical STATEs, associated with each side of the HYPERSURFACE on which the POINT is located. Similarly, F_NODE, F_CURVE, and F_SURFACE are objects that inherit and extend the corresponding INTERFACE object. In practice, we tend to ignore the difference between an object and its extension and denote the entire range of an inherited object by its corresponding INTERFACE identifier. Thus we speak of a POINT or CURVE in the FRONT library, when actually these objects are F_POINTs or F_CURVEs, respectively. At level 3, a STATE is the address of allocated storage of known size. Three basic operations can be performed on STATEs in level 3. A STATE can be erased (all bits in the STATE assigned to zero) or copied, or two STATEs can be interpolated. The latter operation is implemented via virtual functions that must be assigned from a higher level library. FRONT can use the interpolation functions as "black boxes" but has no information on how the interpolation is performed. For POINTs of co-dimension 2, such as where multiple SURFACEs meet along a CURVE, there are several STATEs (one for each HYPERSURFACE side) for each such HYPERSURFACE; this storage is associated with the HYPERSURFACE, rather than with the POINT. The HYPERSURFACEs also acquire new structure: a wave type, which designates physics-specific information about the type of front. At level 4 specificity, these wave types can be read fully, but at level 3, only generic wave types defining boundary conditions (NEUMANN, REFLECTING, DIRICHLET, or PERIODIC) or abstract physics can be read. NEUMANN boundaries correspond to reflecting walls, while REFLECTING boundaries correspond to symmetry axes. For some physics these two boundary types may be equivalent, but their implementation is different. DIRICHLET boundaries come in two types, those with fixed, time independent boundary conditions, and those whose boundary conditions are defined by user-defined functions. Scalar or vector waves are defined as wave fronts corresponding to wave families whose ray cone is either degenerate (scalar) or nondegenerate (vector). For example, in gas dynamics a contact discontinuity or material interface is a scalar wave, while a shock front is a vector wave. Note that the differentiation between scalar and vector is known at level 3, but the notion of shocks and contact discontinuities is a level 4 concept.

Passing to specificity level 4, meaning is attached to the (floating point and integer) data contained in a STATE. Thus the simplest idea of a STATE for three-dimensional compressible fluid flow would be five floating point numbers. It is convenient to have the equation of state addressable through the STATE itself, for application to multicomponent or multiphase flow problems. Thus the address of the

equation of state data base is added to the STATE structure. The equation of state information is opaque at level 4 but can be accessed at level 5.

Apart from support routines for front-associated data structures, the main operations performed within the front library are (a) remeshing of the FRONT, (b) drivers for propagation routines for both regular (codimension one) and irregular (codimension two or higher) POINTs (section 3), and (c) untangling of self-intersecting FRONTs with only scalar degrees of freedom (section 3). Scalar fronts have degenerate ray cones and simple wave interaction laws that can largely be implemented using pure geometry. In contrast, vector fronts, such as shock waves, interact so as to create both reflected and transmitted waves, even in the simplest cases. The FRONT data structure contains parameters to control these operations (a)–(c), including function pointers to level 4 physics functions for propagation details.

Remeshing introduces tangential diffusion through the interpolation of STATE values, and it smoothes the hypersurface shape through convex interpolation of positions, so that it is important not to remesh too often. However, unduly long or short BONDs, which arise during unremeshed multiple time step propagation, can interfere with the accuracy or stability of the computation, so that it is also important not to remesh too infrequently. Remeshing, in two dimensions, is accomplished by dividing the arc length $L$ of the CURVE by a desired BOND length $l'$, to arrive at the number $n$ of BONDs. Since this division must achieve an integer value, we set $n$ equal to the integer roundoff of $L/l'$, and $l = L/n$. New POINTS are inserted along the CURVE, with arclength spacing $l$, and then old POINTs are removed. The remeshed BONDs have nearly uniform length.

In three dimensions, there is no such linear order to the TRIANGLEs of a SURFACE. Remeshing is a local algorithm. Size and aspect ratio criteria are given as input parameters to this algorithm. Individual triangles are tested and, if they fail these criteria, placed on a queue. One of two elementary operations is then applied to each triangle in the queue. These operations either split or combine pairs of triangles having a common edge; the operations are the inverse of each other in the sense that applying both will leave the interface unchanged. The first elementary operation divides a pair of TRIANGLEs with a common edge through bisection of their common edge. The other operation shrinks a pair of TRIANGLEs with a common edge down to a pair of edges. This operation can alternately be described as shrinking the common edge to a single point.

The front spacing is a run time parameter, which sets the overall length scale for the remeshing of front points, as a multiple of the regular grid spacing of the interior solver. For two-dimensional computations, experience with many problems indicates that a front spacing value of 0.75 is satisfactory. For three dimensions, further experiments in a variety of problems will be needed to determine suitable values for the parameters which control the redistribution of front points. For the runs given here, the value of 0.75 was also used.

**2.3. Interpolation.** Three other libraries complete the level 3 code for conservation laws. A hyperbolic library is concerned with states and propagation at regular grid points. A driver library contains the main program initialization, time loop control, and i/o capabilities. Another level 3 library, discussed here, supports interpolation, based on state data from the front and hyperbolic libraries.

The ability to interpolate piecewise smooth functions with arbitrary discontinuities across interfaces is of considerable independent interest, and has been developed

as an isolated capability. It is used to support equation of state tables with phase transitions across the discontinuity [13, 14].

Interpolation is also used for tabulated rarefaction curves, for the rapid solution of Riemann problems starting from a tabular equation of state [58]. With the recent extension of front tracking interpolation to three independent variables, the tabulation of shock curves is also feasible.

To ensure the integrity of the interpolation process, only state values from a single COMPONENT can be interpolated. State data is stored at grid cell centers. For interpolation, we consider the dual grid, with states stored at grid cell corners. For a regular (dual grid) cell, i.e., one which does not intersect the front, bilinear interpolation gives the interpolated state values. For an irregular cell, in two dimensions, we introduce a triangulation that respects the interface and that uses only those points for which the states are already known: the dual grid corners and the (one-sided) front points. We now describe our algorithm for triangulation in two dimensions. First, subdivide each dual grid rectangular mesh cell which meets the interface into polygonal subdomains. The edges of each subdomain are formed by the bonds from the interface and the mesh line edge of the dual grid cell. A subdomain may be multiconnected. Multiconnectedness can be detected by calculating the winding number at a vertex. Each multiconnected subdomain is divided into simply connected subdomains by adding two new edges, if necessary.

Finally we triangulate each simply connected polygon by joining pairs of vertices. Triangulation is based on a divide and conquer algorithm. The idea is to divide the polygon into two subpolygons and determine a triangulation for each subpolygon. The triangulation of the two subpolygons determines the triangulation of the combined polygon. We apply this method recursively until a subpolygon has only three vertices, and is thus a triangle. A detailed description of such an algorithm can be found in [1]. In [1], only convex polygons are considered, but the algorithm can be modified to obtain triangulations for nonconvex polygons as well.

Linear interpolation on each triangle constructed above then completes the definition of the solution function. It respects the discontinuities in the computed solution exactly. Access to individual fields within the state is accomplished by function pointers in the front and hyperbolic libraries to level 4 functions.

The grid construction upon which this interpolation method is based does not generalize to three dimensions. A surface-respecting tetrahedralization of space, starting with a given set of vertices (at which states are defined), is generally not possible. Extra vertices, called Steiner points, may be needed. The problem of deciding whether Steiner points are required for a given surface and set of vertices is NP-hard [57]. Standard tetrahedralizations, such as the Delaunay triangularization, will not, in general, respect a given surface, in the sense that the tetrahedrons formed from vertices on one side of the surface may cross the surface. If the surface is convex, then this cannot happen for the Delaunay triangularization. We cannot assume a convex surface, since convex surfaces are concave when viewed from the opposite side. If the surface triangles are sufficiently refined through the addition of extra points, then the Delaunay tetrahedralization will respect the surface. In contrast to the case of general Steiner points, where data for interpolation would be missing, data at the extra points on the surface triangles can be given by linear interpolation from the vertices of the triangles. This solution to the problem will be expensive and will give poor tetrahedra (with bad aspect ratios). Our solution to the interpolation problem is to allow a separate tetrahedralization of each connected component of $R^3$, as defined by the interface.

The tetrahedralization is determined by the vertices in that component, including the points on the interface facing that component. This tetrahedralization, for a specific component, will in general cross the component boundaries. Thus, when applied to all components, it may be multiple-valued, in that a single point may belong to two tetrahedra associated with two distinct components. For our intended interpolation use, this multiple valuedness is not a disadvantage. The interpolation function has as arguments a point in $R^3$ and the associated component containing that point. The tetrahedralization associated with that component is used for interpolation, and so the solution function is single valued, with sharp discontinuities at the interface, as desired. This construction assumes that the interface, restricted to a dual grid mesh block, separates the mesh block in the sense that each side of each surface faces distinct components. Delaunay triangulation applied to each polynomial component generates a tetrahedral grid with an optimal aspect ratio. It is implemented by an $O(n \ \log n)$ algorithm. Here $n$ is the number of vertices in a given irregular rectangle grid cell. The algorithm first takes four non-coplanar vertices and generates one tetrahedron, and then adds the other vertices one at a time. If the vertex under consideration lies inside an existing tetrahedron, we subdivide the tetrahedron at this newly added vertex. Otherwise, for each triangular surface of a tetrahedron which is completely visible to the newly added vertex, we form a new tetrahedra by using the three vertices of the triangular surface and the newly added vertex. When all vertices have been added, the Delaunay procedure is complete and we have a tetrahedralized grid for all vertices of the same COMPONENT in that particular irregular rectangular grid cell. We apply this procedure for each set of vertices of the same COMPONENT in that cell. For a general discussion of tetrahedralization in three dimensions, see [15].

**3. The time step algorithm.** The solution of conservation laws of the form

$$\vec{U}_t + \vec{\nabla} \cdot \vec{F}(\vec{U}) = \vec{G}(\vec{U})$$

are supported by the general framework discussed in sections 2–3. The nature of the propagation is governed by the codimension of the point. The codimension is that of the maximal (local) space-time manifold containing the point on which the solution is smooth. Within this manifold, the solution is continuous along all curves passing through the point. Normal to this manifold, the solution is discontinuous along all curves passing through the point. Stated more technically, for any family of transverse curves defined throughout a neighborhood of the point, the solution will fail to be jointly continuous in the base point in the manifold and arclength along the curve. In simple examples, the codimension is the number of simple jump discontinuities present simultaneously in the solution at a given point in space and time. For time dependent problems in `dim` space dimensions, $0 \leq$ `codimension` $\leq$ `dim` $+ 1$. The minimum value, 0, occurs for points of continuity of the solution. At a discrete level, regular grid points are treated as having codimension 0. Their time propagation is discussed in section 3.1. The maximum value, `dim` $+ 1$, occurs for points at which the solution is bifurcating in time, as well as being discontinuous in space. Thus for this case there is no space-time curve passing through the point, along which the (space-time) solution function is continuous. Intermediate values of codimension will often be continuous in time, with all discontinuous directions realized in space at a fixed time. However, this simplifying picture need not hold. A simple example is provided by a phase transition. Applying pressure uniformly, in a time dependent manner, can produce a simple jump discontinuity in some physical variable, and this discontinuity can be

spatially uniform, i.e., discontinuous in time only, but not in space coordinates. Even if some physical effect, such as a gravity- or heterogeneity-induced spatial pressure difference, can be found to break the spatial symmetry of this example, the effect can be weak and ill conditioned. Thus numerical methods based on the simplified picture are not always applicable.

The codimension 1 points are located on the front but are otherwise regular. They are points of simple jump discontinuity. They lie on the interior of a surface in three spatial dimensions and on the interior of a curve in two dimensions. We discuss point propagation in order of increasing codimension, and thus of increasing difficulty.

**3.1. Interior states: Codimension 0.** The propagation in time of interior states (when based on dimensionally split methods) uses a one-dimensional regular grid stencil, for a sweep along each coordinate direction, and a choice of finite difference operators for this stencil, such as the higher-order Godunov method, the Lax–Wendroff scheme, etc. Special care is needed only when the stencil is cut by a front; in this case there are missing state values, as the finite difference operator is expected to receive states from a single component only. In this sense, the method takes the idea of weak derivatives seriously and will never compute a finite difference across a tracked front. For reasons of modularity, the assembly of the stencil (a level 3 routine) and the computation of the finite difference (a level 4 routine) are separated. As a result, the introduction of new physics or of a new finite difference algorithm only requires insertion of a new level 4 one-dimensional elementary finite difference step, defined on a single stencil.

The missing points of the stencil, in the case of a front cutting through the stencil, are obtained by extrapolation. Conceptually, the state values are double-valued near the front, with the left component states extending by extrapolation for a small distance into the right component, and vice versa. The computationally efficient implementation of this extrapolation is obtained from precomputed information of front crossings of dual grid block edges, which is a side computation in the interpolation grid data structure referred to in section 2.3. This method is conservative in the interior, i.e., away from the front. Computational experiments have shown very reasonable conservation properties with this method; see Table 4.2.

**3.2. Regular front states: Codimension 1.** The propagation of the front coordinates and states is performed in a single step. Operator splitting, in a rotated coordinate system, allows separate propagation steps in directions normal to and tangent to the front. First consider the normal propagation step. The analysis reduces to the integration of a differential equation in one space dimension (the normal direction), and thus is largely independent of spatial dimension.

The leading order terms in the propagation of a discontinuity, in the direction normal to the front, is given by the solution of a Riemann problem. This is the one-dimensional Cauchy problem, with idealized initial conditions consisting of a single jump discontinuity. The solution will, in general, contain a number of waves. Of these, one is identified with the discontinuity being tracked (through the wave-type variable stored as part of the front contribution to the hypersurface). This could be a bona fide discontinuity in the Riemann solution wave structure, or it could be one of the edges (leading or trailing) of a rarefaction wave. In any case, the Riemann solution will give the wave speed and states immediately ahead of and behind the advancing front. This speed and the states define the new interface position, and thus the lowest-order version of the normal propagation algorithm.

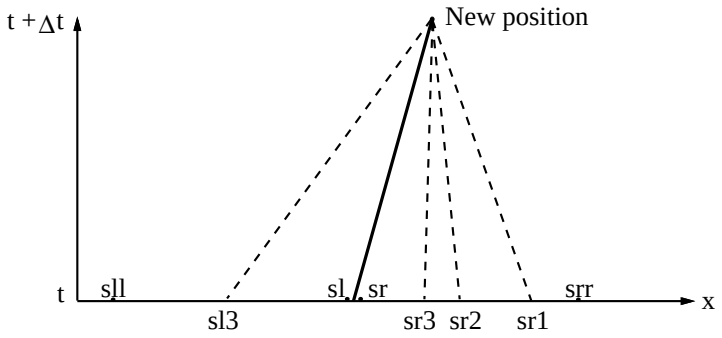



FIG. 3.1. *A schematic picture of the solution of the normal propagation of the front. The front data at the old time step gives a Riemann solution, which is corrected by interior data, using the method of characteristics.*

Corrections are needed to couple the interior variation of the solution states to the front propagation. For this purpose, a *generalized* Riemann problem is solved. By this we mean a Cauchy problem having a single jump discontinuity in the initial data. However, the initial data on each side of the jump discontinuity, rather than being constant, is now allowed to be linear. The linear approximation to the nearby interior solution states is constructed by moving a mesh distance $\Delta s$ away from the interface along the normal on each side of the interface. The resulting point for solution evaluation will not, in general, be a regular grid point, and so the solution at this point is constructed by interpolation, as described above in section 2.3. The solution of the nonlocal Riemann problem is constructed as a finite difference correction to the previously discussed (local) Riemann problem, using the method of characteristics. See Figure 3.1.

Curvature-dependent corrections to the normal propagation are contained implicitly in the tangential sweep. Beyond these, there may be curvature corrections due to self-interaction of the wave form (for a viscous or nonlocalized discontinuity), through its dependence on finite transport coefficients, such as viscosity or chemical reaction rates. For detonation waves, such corrections are known to be important and have been studied quantitatively [8, 39, 47, 48]. It has been argued that there should be a similar but smaller effect for (viscous or nonlocal) discontinuities in general [49]. This effect is not included in the computations at present and is believed to be relatively small for most problems.

The tangential propagation step modifies the interface states but not the points. The tangential motion of the interface is a reparameterization of the interface, and does not contribute to its dynamics. As a convention, the reparameterization is taken to be the identity.

Separate finite difference steps are carried out for the states on each side of the interface. The splitting into normal and tangential directions is locally orthogonal, and for this reason no explicit source terms are introduced into the difference equations by the splitting. While this seems paradoxical, since, e.g., radial expanding flow must decrease as the wave front expands, the decay mechanism is found not in an explicit source term but in the divergence of the velocity field, as seen by the tangential finite difference stencil, after the states are projected onto the plane tangent to the surface.

**3.3. Propagation of codimension 2 points.** The allowed geometry of interacting waves is dependent on the physics, i.e., on the governing equations of motion.

The interaction of fronts is usually of codimension 2. Thus in two space dimensions, the interaction occurs at a NODE, while in three dimensions it takes place along a CURVE. For the well-studied case of compressible gas dynamics in two space dimensions, the geometry defined by interacting waves is moderately well understood. Each type of wave interaction, or node, has its own dynamical algorithm. The basis for these algorithms is the theory of shock polars, which is a graphical method for understanding the equations governing a node. The method yields a succession of equations of elementary waves, for each of the elementary waves meeting at the node, together with an equation stating that traversing each wave in succession, thus tracing a path around the node (curve in three dimensions), yields the starting state of that path. The details of the theory are beyond the scope of this short discussion; see [27, 21].

The propagation of curves in three dimensions will be governed by the same principles, i.e., by the solution of shock polars to describe a sequence of (one-dimensional) Riemann problems while traversing a circle lying in the plane normal to a codimension 2 wave interaction submanifold. Such codimension 2 and higher propagation problems in three-dimensional computations are not presently supported and will be avoided by the choice of initial problems and degrees of tracking within these problems, for which codimension 2 and 3 wave interactions (other than the bifurcations of hypersurfaces of scalar type) do not arise.

**3.4. Propagation of codimension 3 points.** The bifurcation of nodes in two dimensions and the propagation of nodes in three dimensions is a codimension 3 problem. This problem, in which the topology and the nature and number of waves meeting at a node changes, is complicated and less well understood. There appears to be a very large number of possible such bifurcations, not fully classified. A sufficient number of the more commonly occurring bifurcations are understood and supported in the front tracking code in two dimensions so that an interesting range of complicated interaction problems can be handled.

In three dimensions, we avoid this complexity initially by tracking material boundaries only, for which the physics of the interactions is well understood.

Even for the simplest case of scalar waves, such as contact discontinuities, for which the interface is untangled on the basis of geometrical considerations alone [19], the ability to track an interface through a change of topology appears to be unusual, or possibly unique, and allows the computation of complex interfaces. See Figure 3.2. For a study of mesh refinement of a similar unstable interface computation, see [18].

This untangle capability (currently available in two dimensions only) is based on a sharing of coboundary topological data. At a dynamically generated interface self-intersection, topological information, such as components, will be locally inconsistent. It is assumed that the time step leading to the self-intersection is sufficiently small so that the self-intersection is a relatively isolated event. That being the case, the topological information at the self-intersection point can be compared to similar information at other ends (edges) of the intersecting hypersurfaces. These ends are the coboundaries of the intersections. Assuming that the coboundary information did not result from a self-intersection, it must be valid. On this basis, the inconsistent information at the self-intersection point can be evaluated, and unphysical hypersurfaces identified. For further details concerning this algorithm, see [19].

For waves with vector degrees of freedom, such as shock waves, the untangle will produce new waves, in addition to the elimination of unphysical ones. The proper discussion of the algorithm in this case is beyond the scope of this review and is not completely implemented in all cases; see [24, 27, 25, 28, 6, 5].

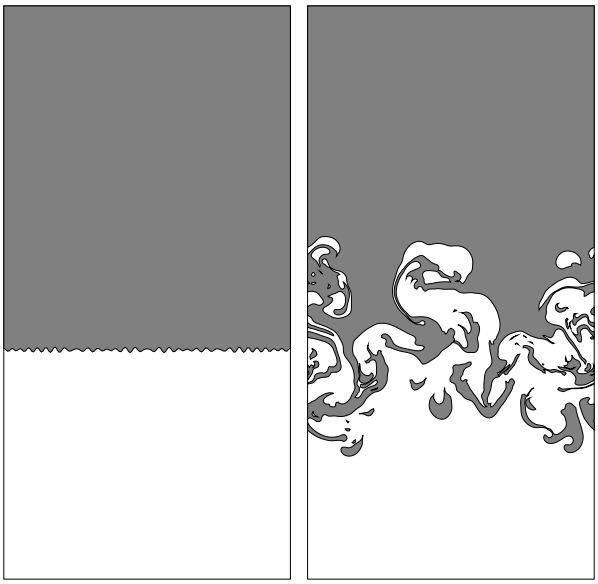
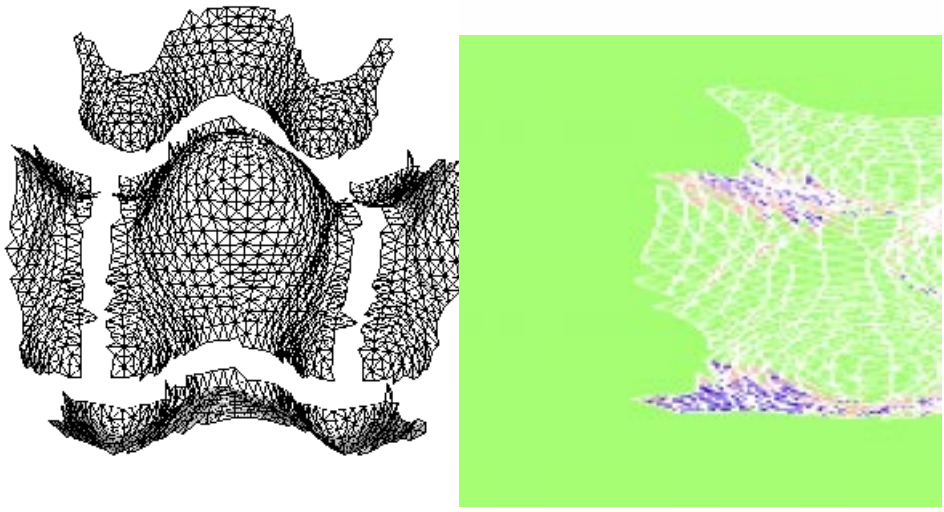


FIG. 3.2. *The frame on the left shows an early time step in the evolution of an interface between fluids of different densities, subject to gravitational acceleration. The light fluid is on the bottom and the heavy fluid is on the top. The acceleration points downwards. The frame on the right shows the complex interface that results, at a later time. Note in particular the formation of droplets of heavy fluid in the mixing region, as a result of successive bifurcations of the interface topology.*

**3.5. Parallel implementation.** Parallel communication of interface data for distributed memory computers poses a special problem, as much of the data consists of addresses of other data, i.e., pointers. Efficient communication requires large message packets, so that all allocated storage is communicated as a bit array, with no attention paid to the logical significance of the data. Upon receipt, this array has to be unbundled and all pointers reset to the addresses the data will have on the new processor. Because of the allocation of interface storage in contiguous chunks of known addresses, this readdressing can be achieved conveniently by simple pointer arithmetic, in terms of the relative addresses of the interface storage chunk on the two processors. Computational efficiency requires setting a relatively large chunk size.

For hyperbolic systems of conservation laws, as considered here, explicit solution of finite difference equations allow domain decomposition methods to be used in a straightforward manner. The entire computational region is divided into an $x, y, z$ rectangular grid of processor domains, with a single domain and single processor assigned uniquely to each other. At a later stage of development, we imagine a hierarchical domain mesh refinement decomposition of the computational region. Buffer zones are created at the edges of each computational domain to allow efficient communication and storage of boundary information from neighboring domains. Synchronization after communication in each coordinate direction allows edge and corner buffer locations to be included within the second and third coordinate direction communications, rather than as distinct operations. In this way, $2d = 6$ rather than $3^d - 1 = 26$ communication steps transmit regular boundary information.

Communication primitives are written using modular protocols, with calls to manufacturer's primitives contained in a single file. Until communication primitives and



(a) (b)

FIG. 3.3. *Frame* (a) *shows the portion of the interface associated with a single computational domain, together with the four pieces of extended interface needed to construct the solution over the ghost cells which belong to adjacent computational domains. Note that the domain decomposition is arranged so that there is no z direction communication, and the x direction communication occurs before the y direction communication. Thus the* $x - y$ *corners, in the computational domain of diagonally adjacent processors, are communicated as part of the y direction communication, as explained in the text. Frame* (b) *shows the reconstructed interface after the complete communication step, including both the part of the interface within the computational domain proper, and that part extending into the ghost cells associated with neighboring domains.*

their calling sequences are standardized in programming languages, this method ensures a convenient way to achieve portable and modular parallel communication code and allows an easy upgrade to MPI or PVM protocols.

Rejoining pieces of a triangulated surface along a domain boundary is accomplished by floating point comparison of point positions. Since the points to be compared have been propagated by one time step on distinct processors, a possibility of logical confusion in this operation exists. The need for logical accuracy is very high, due to the large number of such comparisons to be made within any given run and to the lack of tolerance in the interface data structure for logical errors in its data representation. Thus redundancy is built into the identification routine, with a match of three points used to establish logical identity of a single triangle, and nearest neighbor pointers to propagate this identification to neighboring triangles. See Figure 3.3.

**3.6. Memory and CPU time.** The front tracking method actively tracks the evolution of fluid interface geometry and thus maintains a sharp resolution of discontinuous physical quantities. Management of the interface data structure adds certain computational costs to the program. Such costs can be viewed in two parts. One is the memory needed to store and resolve the interface geometry. The other is the CPU time needed to handle the physical evolution of the front. Besides the memory of physical states on a regular computational mesh, front tracking requires additional memory to store states on the interface and to store the associated interface topology. In addition, since the objective of tracking the front is to couple the interaction between the front states and the states on regular mesh, the program has to maintain

TABLE 3.1

*CPU time in seconds consumed by each part of the front tracking program. $T_p$ is the time used for the front propagation including both the propagation of points in the normal direction and the tangential sweep of the front states. $T_g$ is the CPU time for the construction of the unstructured grid near the front. This grid provides the coupling between the interface and the interior states. $T_{fd}$ is the CPU time used by the finite difference scheme on the regular grid. This time is representative of the time for an untracked computation. The CPU time for the regular grid and the front become comparable at an $80 \times 160$ computational mesh. $T$ is the total time for a time step, approximately equal to the sum of the three preceding columns.*

| Grid | $T_p$ | $T_g$ | $T_{fd}$ | $T$ |
|---|---|---|---|---|
| $10 \times 20$ | 0.57 | 1.32 | 0.20 | 2.08 |
| $20 \times 40$ | 0.78 | 2.12 | 0.57 | 3.48 |
| $40 \times 80$ | 1.43 | 4.23 | 2.02 | 8.02 |
| $80 \times 160$ | 2.45 | 8.58 | 7.08 | 18.00 |

an unstructured grid in order to interpolate the states between the regular mesh and the front. The unstructured grid must be updated at each time step.

The percentage of memory storage and CPU time used for tracking the front and computing the states on the regular mesh is dependent on several factors, such as whether the interface is steady or expanding, the refinement of the computational mesh, and the dimension. For a physical problem with a relatively steady front in $D$ dimensions ($D = 1, 2, 3$), we have the following scaling for the memory:

$$M \approx C_f N^{D-1} + C_r N^D,$$

where $N$ is the number of grid points in one dimension. The first term refers to the memory used by the front, and the second term refers to the memory used by the regular mesh. The coefficient $C_f$ is substantially larger than $C_r$. This means that the front takes a large percentage of the memory in the coarse grid. But as the mesh is refined, due to the scaling, the memory for the regular mesh increases faster than that for the front and eventually becomes dominant. Considering the CFL condition, the CPU time is scaled in the following way:

$$T \approx A_f N^D + A_r N^{D+1}.$$

Table 3.1 shows the CPU time for one time step in the two-dimensional simulation of the Rayleigh–Taylor instability. In this table, it is shown that although the CPU time for the front is the leading term in the coarse grid, it becomes comparable with the CPU time of the regular grid with a $80 \times 160$ mesh for one time step. For a comparison of tracked and untracked computations at comparable levels of resolution, an untracked grid finer by a factor of three to five in each spatial dimension should be used, rather than the comparison on identical grids, as shown here. On this basis, the $20 \times 40$ tracked computation is about twice as fast as the $80 \times 160$ untracked computation, and the $40 \times 80$ tracked computation is about comparable to the $40 \times 80$ untracked computation. The three-dimensional front tracking has the same scaling, but since the code requires further optimization, the coefficients of the scaling are not yet meaningful. The scaling law should be modified if the physical problem has either a growing or a shrinking interface. For this reason, the numbers shown here have only qualitative or relative significance.

The CPU time for data communication in two dimensions is negligible, and the efficiency of parallelization is over 95%. In three dimensions, due to relatively large

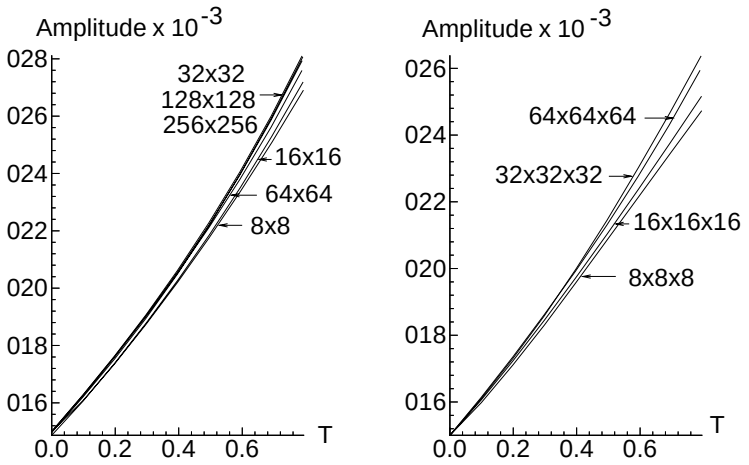



FIG. 4.1. *We show convergence under mesh refinement of the computed front tracking solution. The left frame displays the amplitude as a function of time for a two-dimensional computation; the right frame refers to three dimensions. The amplitude is displayed in dimensionless units, as a fraction of the wave length.*

amount of data to be exchanged in a given time step, the efficiency is decreased to about 75–80%. A careful arrangement of processors and optimized chuck size for each data passage can substantially reduce the CPU time needed for communication.

**4. Numerical results and validation.** The Rayleigh–Taylor instability is a gravity-, or acceleration-, driven instability of an interface separating two fluids of differing densities [59]. For small amplitude, single mode sine wave initial positions of the interface, the two fluid-compressible Euler equations admit a linearization whose solution is still a sine wave in space and an exponential (or for finite domains, a difference of exponentials) in time; cf. [17]. The validity of this solution depends critically on the smallness of the amplitude. The solution is changing very slowly, so that physical effects, which are small, may be masked by numerical errors, including numerical diffusion at the interface.

In Figure 4.1 we display the amplitude (peak to trough) as a function of time, showing convergence under mesh refinement. This plot emphasizes the small amplitude regime, in which the growth rate is nearly exponential. The amplitude, even at the final time step, is approximately one fifth of a mesh block on the coarsest grid and is less than two mesh blocks on the finest three-dimensional grid. Numerical diffusion of the interface, for other methods of computation, would be larger than the perturbation itself, and thus preclude this computation. For this computation, the front spacing was set equal to the interior grid spacing and a CFL number 0.95.

We also study $L^1$ convergence under mesh refinement for the same example. In Table 4.1 we give the $L^1$ errors for different mesh sizes, compared to the $64^3$ mesh grid, when $t = 0.8$. We show relative errors for density and energy and absolute errors for momenta. In this example, the momenta are so small that it is not realistic to measure the relative errors. (The $x$ and $y$ momenta have 45% relative errors for the $8^3$ mesh grid and 19% for the $32^3$ mesh grid, while the $z$ momentum has 98% relative errors for the $8^3$ mesh grid and 24% for the $32^3$ mesh grid.) We see first-order convergence. This is as expected, since the code is second order in the interior and first order in the front propagation; hence the motion of the interface is first-order

TABLE 4.1
*$L^1$ errors for different mesh sizes compared to the $64^3$ mesh grid for $t = 0.8$.*

| Mesh size | Density relative error | Energy relative error | $x$ momentum error | $y$ momentum error | $z$ momentum error |
|---|---|---|---|---|---|
| $8^3$ | 0.47% | 0.37% | 0.0025 | 0.0025 | 0.0061 |
| $16^3$ | 0.27% | 0.18% | 0.0019 | 0.0019 | 0.0032 |
| $32^3$ | 0.15% | 0.07% | 0.0011 | 0.0011 | 0.0015 |

TABLE 4.2
*Errors in conservation properties.*

| Mesh size | Density relative error | Energy relative error | $x$ and $y$ momentum error | $z$ momentum error |
|---|---|---|---|---|
| $8^3$ | 0.093% | 0.066% | $6.9 \times 10^{-6}$ | 0.0059 |
| $16^3$ | 0.041% | 0.054% | $2.8 \times 10^{-6}$ | 0.0025 |
| $32^3$ | 0.014% | 0.014% | $8.3 \times 10^{-5}$ | 0.0009 |
| $64^3$ | 0.009% | 0.001% | $2.3 \times 10^{-5}$ | 0.0004 |

accurate. In Table 4.2 we also show errors in conservation properties. For each mesh size, we compare the conservative quantities at $t = 0$ and at $t = 0.82$. Notice that the integrals of total energy and $z$ momentum are not conserved due to the body force and the boundaries in the $z$ direction. We thus compare these two quantities with solutions to the ordinary differential equations which they satisfy. Again, we see linear convergence. The exception is for $x$ and $y$ momenta, but these are so small that asymptotic decrease of errors has not been achieved on the meshes used.

We also compare numerical results from the front tracking method with results from the level set method using TVD as its interior solver. See Figures 4.2 and 4.3, which show the density and pressure profiles along the $z$ gridline through the spike tip from these two methods at a common time.

Our main conclusion is that if the two fluids have different equations of state (i.e., are different fluids), the level set method does not provide a satisfactory choice of the equation of state in the numerically mixed zone. If the two fluids are identical, the level set method plays no role in the computation and is merely a graphical postprocessor of the solution. It follows that the extra precision in the interface location afforded by front tracking is very important for problems in which the two fluids separated by the interface are qualitatively dissimilar. The level set method is relatively simple in its handling of the fluid interface and, with an enhanced interior solver, can provide a satisfactory solution for interfaces between fluids with similar physical properties [44].

In Figure 4.2, computed by the level set method, the density profile has spread over 4–5 mesh points. The interface position given by the level function at $\psi = 0$ is at the left end of the numerical density mixing zone. A pressure bump is produced near the interface due to the inconsistent use of the equation of state relative to this density profile. Not only does the level set choose a nonoptimal interface location at the lower end of the density mixing zone but, more fundamentally, its use of a sharp interface model for the equation of state, and any type of level set demarcation between the fluids, is inconsistent with a numerically diffused density mixing zone, and so an equation of state error is unavoidable. The specific implementation of the level

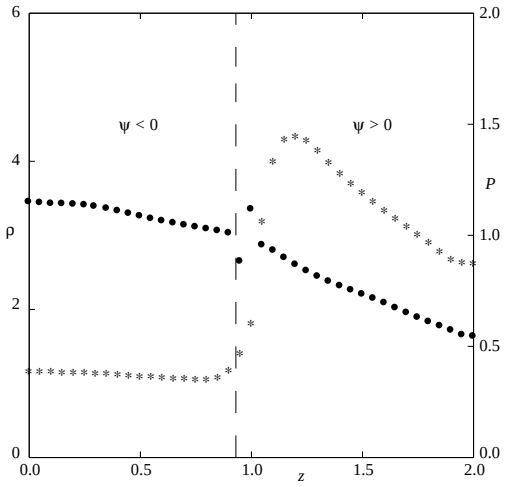



FIG. 4.2. *Density and pressure profiles in the simulation of the three-dimensional Rayleigh–Taylor instability using TVD and the level set method. The vertical dashed line is the position of the unperturbed interface. The gases on the different sides of the interface have different equations of states. The plot illustrates the use of an inappropriate equation of state (inherent to the level set method) within the mixing zone. The incorrect equation of state results in an unphysical pressure spike at the interface, which yields an artificial shock wave. The asterisks represent density (left scale) and the solid dots represent pressure (right scale).*

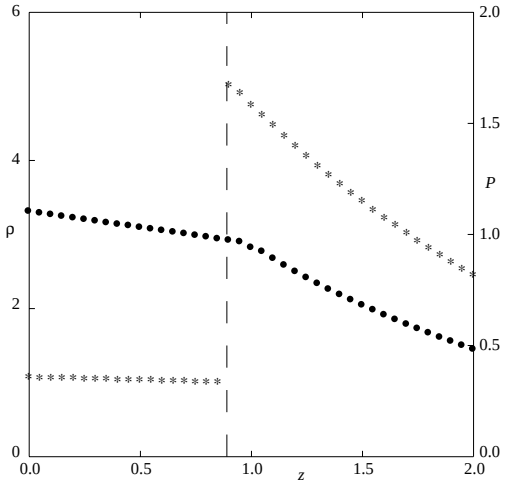



FIG. 4.3. *Density and pressure profiles in the simulation of the three-dimensional Rayleigh–Taylor instability using the front tracking method. The vertical dashed line is the position of the unperturbed interface. The gases on the different sides of the interface have different equations of state, as in Figure* 4.2. *The computation shows a sharp interface, and all equation-of-state computations refer to one of the original fluids; there is no mixing zone and no mixed fluid equation of state.*

set method used here was described in [43, 38] and references cited there. See also [53] for the proposal to use the level set method for the computation of Rayleigh–Taylor instabilities and for additional references. The pressure bump produces an artificial shock which slows down the motion of the heavy fluid. For the front tracking method, however, the two fluid components are explicitly distinguished to ensure the proper

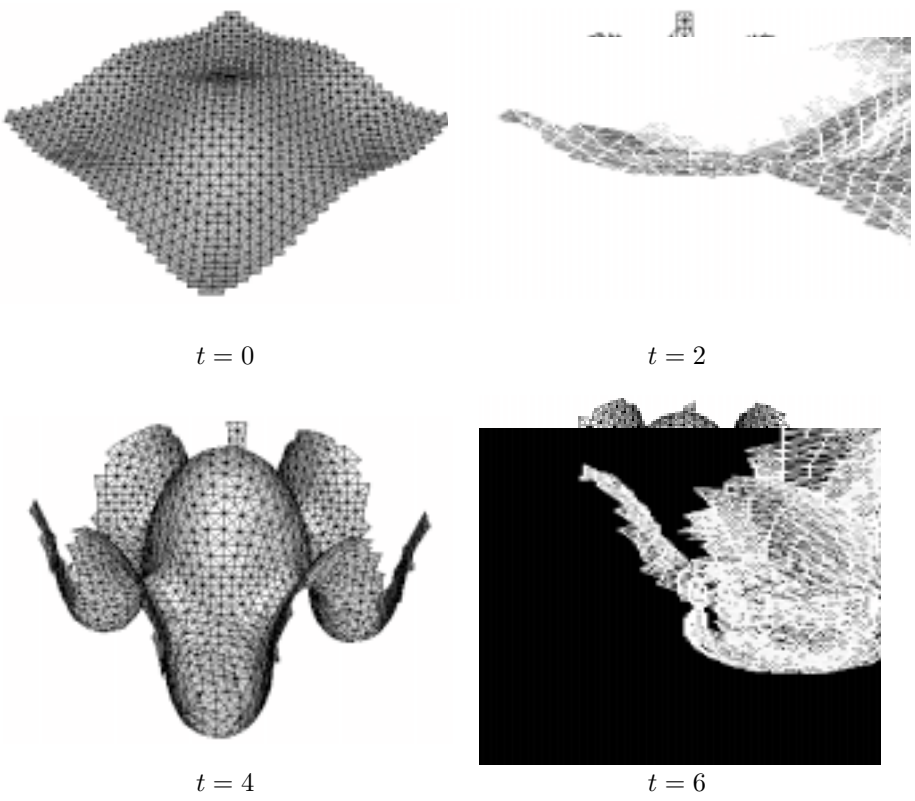



FIG. 4.4. *Evolution of a three-dimensional fluid interface in the Rayleigh–Taylor instability through the front tracking method. The computational mesh in this simulation is* $20 \times 20 \times 40$. *This coarse grid simulation has remarkably good agreement with the growth rate in the linear regime. In this case, the density ratio is* $5:1$ *and the compressibility is* $M^2 = 0.1$.

use of the equation of state. Not only is the density profile sharp but also the pressure profile is smooth, with a discontinuous slope at the interface as required by interface jump conditions.

In these computations, we use the stiffened polytropic equation of state [32]

$$P_i = (\gamma_i - 1)\rho_i e_i - \gamma_i P_{i\,\infty},$$

where $P_i$ denotes the pressure in the fluid $i = 1, 2$, $\rho_i$ is density, and $e_i$ is the specific internal energy. The lower (lighter) fluid has a polytropic equation of state ($P_{1\,\infty} = 0$), with $\gamma_1 = 1.4$. The upper (heavier) fluid has the stiffened polytropic equation of state with $\gamma_2 = 4.0$ and $P_{2\,\infty} = 1.0$, appropriate for compressible liquids such as water. Initially $\rho_1 = 1$, $\rho_2 = 5$ at the interface.

In Figure 4.4, we display a well-developed Rayleigh–Taylor bubble and spike, computed by the front tracking method.

**5. Discussion and conclusions.** We have established feasibility for three-dimensional front tracking. The wide variety of problems with a need for specialized surface or interface computation and the high (distinctive) quality of solutions resulting from two-dimensional front tracking provide the motivation for this effort.

Specific difficulties with front tracking are discussed. Some of these have solutions of possible general interest. An example is the front tracking interpolation, which resolves discontinuities of piecewise smooth functions and (in two dimensions) has given excellent representation of thermodynamic tables with phase transition discontinuities.

Validation of the method is given by mesh refinement studies. We also present comparison to other numerical methods.

Perhaps the biggest promise for front tracking in three space dimensions is to combine it with advanced shock capturing and adaptive mesh refinement. The latter technologies are extremely good at resolving shock fronts and wave interactions but have major deficiencies at contact discontinuities and material interfaces. Front tracking is ideal for material interfaces as long as physical diffusion is insignificant. Thus the three numerical techniques exactly complement each other, and together they have the ability to accurately compute complex flows in three space dimensions.

## REFERENCES


[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structure and Algorithms*, Addison–Wesley, Reading, MA, 1983.

[2] J. B. Bell, P. Colella, and M. L. Welcome, *Conservative Front-Tracking for Inviscid Compressible Flow*, in UCRL-JC-105251, preprint, 1991. Presented at AIAA 10th Computational Fluid Dynamics Conference, Honolulu, HA, 1991.

[3] B. Boston, *Front Tracking of Complex Wave Interactions*, Ph.D. thesis, State Univ. of New York at Stony Brook, Stony Brook, NY, 1995.

[4] B. Boston, J. Glimm, J. W. Grove, R. Holmes, and Q. Zhang, *Multiscale structure for hyperbolic waves*, Stud. Adv. Math., 3 (1997), pp. 1–9.

[5] B. Boston, J. W. Grove, L. F. Henderson, R. Holmes, D. H. Sharp, Y. Yang, and Q. Zhang, *Shock induced surface instabilities and nonlinear wave interactions*, in Proceedings of the Eleventh Army Conference on Applied Mathematics and Computing, 1993, U.S. Army Res. Office, Research Triangle Park, NC.

[6] B. Boston, J. W. Grove, and R. Holmes, *Front tracking simulations of shock refractions and shock induced mixing*, in Shock Waves @ Marseille IV, Proceedings of the 19th International Symposium on Shock Waves, Marseille, France, 1993, R. Brun and L. Z. Dumitrescu, eds., Springer-Verlag, Berlin, Heidelberg, New York, 1995, pp. 217–222.

[7] B. Boston, J. W. Grove, and R. L. Holmes, *Shock induced surface instabilities and nonlinear wave interactions*, Mat. Contemp., 8 (1995), pp. 39–62.

[8] B. Bukiet and J. Jones, *The competition between curvature and chemistry in a spherically expanding detonation*, Appl. Phys. Lett., 52 (1988), pp. 1921–1923.

[9] P. Charrier and B. Tessieras, *On front-tracking methods applied to hyperbolic systems of nonlinear conservation laws*, SIAM J. Numer. Anal., 23 (1986), pp. 461–472.

[10] Y. Chen, Y. Deng, J. Glimm, G. Li, D. H. Sharp, and Q. Zhang, *A renormalization group scaling analysis for compressible two-phase flow*, Phys. Fluids A, 5 (1993), pp. 2929–2937.

[11] I.-L. Chern and P. Colella, *A Conservative Front Tracking Method for Hyperbolic Conservation Laws*, LLNL Rep. No. UCRL-97200, Lawrence Livermore National Laboratory, Livermore, CA, 1987.

[12] I.-L. Chern, J. Glimm, O. McBryan, B. Plohr, and S. Yaniv, *Front tracking for gas dynamics*, J. Comput. Phys., 62 (1985), pp. 83–110.

[13] L. Coulter and J. W. Grove, *The Application of Piecewise Smooth Bivariate Interpolation to Multiphase Tabular Equation of States*, Report No. SUNYSB-AMS-92-11, State Univ. of New York at Stony Brook, Stony Brook, NY, 1992.

[14] L. O. Coulter, *Piecewise Smooth Interpolation and the Efficient Solution of Riemann Problems with Phase Transitions*, Ph.D. thesis, New York Univ., NY, 1991.

[15] H. Edelsbrunner, F. P. Preparata, and D. B. West, *Tetrahedrizing Point Sets in Three Dimensions*, Report UIUCDCS-R-86-1310, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 1986.

[16] S. Eilenberg and N. Steenrod, *Foundations of Algebraic Topology*, Princeton University Press, Princeton, NJ, 1952.

[17] C. L. Gardner, J. Glimm, O. McBryan, R. Menikoff, D. Sharp, and Q. Zhang, *The dynamics of bubble growth for Rayleigh-Taylor unstable interfaces*, Phys. Fluids, 31 (1988), pp. 447–465.
[18] J. Glimm, *Nonlinear waves: Overview and problems*, in Multidimensional Hyperbolic Problems and Computations, J. Glimm and A. Majda, eds., IMA Volumes in Mathematics and Its Applications 29, Springer-Verlag, New York, Heidelberg, Berlin, 1991, pp. 89–106.
[19] J. Glimm, J. Grove, W. B. Lindquist, O. McBryan, and G. Tryggvason, *The bifurcation of tracked scalar waves*, SIAM J. Comput., 9 (1988), pp. 61–79.
[20] J. Glimm, E. Isaacson, D. Marchesin, and O. McBryan, *Front tracking for hyperbolic systems*, Adv. Appl. Math., 2 (1981), pp. 91–119.
[21] J. Glimm, C. Klingenberg, O. McBryan, B. Plohr, D. Sharp, and S. Yaniv, *Front tracking and two-dimensional Riemann problems*, Adv. Appl. Math., 6 (1985), pp. 259–290.
[22] J. Glimm, W. B. Lindquist, O. McBryan, and L. Padmanabhan, *A front tracking reservoir simulator, five-spot validation studies and the water coning problem*, in Frontiers in Applied Mathematics 1, SIAM, Philadelphia, PA, 1983, pp. 107–136.
[23] J. Glimm and O. McBryan, *A computational model for interfaces*, Adv. Appl. Math., 6 (1985), pp. 422–435.
[24] J. Grove, *The interaction of shock waves with fluid interfaces*, Adv. Appl. Math., 10 (1989), pp. 201–227.
[25] J. Grove, *Irregular shock refractions at a material interface*, in Shock Compression of Condensed Matter 1991, S. Schmidt, R. Dick, J. Forbes, and D. Tasker, eds., North–Holland, Amsterdam, 1992, pp. 241–244.
[26] J. Grove, R. Holmes, D. H. Sharp, Y. Yang, and Q. Zhang, *Quantitative theory of Richtmyer-Meshkov instability*, Phys. Rev. Lett., 71 (1993), pp. 3473–3476.
[27] J. Grove and R. Menikoff, *The anomalous reflection of a shock wave at a material interface*, J. Fluid Mech., 219 (1990), pp. 313–336.
[28] J. W. Grove, *A survey of the analysis of irregular shock refractions and its application to front tracking methods*, Mat. Contemp., 3 (1992), pp. 53–66.
[29] J. W. Grove, *Applications of front tracking to the simulation of shock refractions and unstable mixing*, J. Appl. Numer. Math., 14 (1994), pp. 213–237.
[30] J. W. Grove, Y. Yang, Q. Zhang, D. H. Sharp, J. Glimm, B. Boston, and R. Holmes, *The application of front tracking to the simulation of shock refractions and shock accelerated interface mixing*, in Proceedings of the 4th International Workshop on the Physics of Compressible Turbulent Mixing, Cambridge Univ., Cambridge, UK, 1993.
[31] S. Hamaguchi, M. Dalvie, R. T. Farouki, and S. Sethuraman, *A shock-tracking algorithm for surface evolution under reactive-ion etching*, J. Appl. Phys, 74 (1993), pp. 5172–5184.
[32] F. Harlow and A. Amsden, *Fluid Dynamics*, LANL Monograph LA-4700, National Technical Information Service, Springfield, VA, 1971.
[33] L. F. Henderson, *Regions and boundaries for diffracting shock wave systems*, Z. Angew. Math. Mech., 67 (1987), pp. 73–86.
[34] J. Hilditch and P. Colella, *A front tracking method for compressible flames in one dimension*, SIAM J. Comput., 16 (1995), pp. 755–772.
[35] R. L. Holmes, *A Numerical Investigation of the Richtmyer-Meshkov Instability Using Front Tracking*, Ph.D. thesis, State Univ. of New York at Stony Brook, Stony Brook, NY, 1994.
[36] R. L. Holmes, J. W. Grove, and D. H. Sharp, *Numerical investigation of Richtmyer-Meshkov instability using front tracking*, J. Fluid Mech., 301 (1995), pp. 51–64.
[37] J. M. Hyman, *Numerical methods for tracking interfaces*, Phys. D, 12 (1984), pp. 396–407.
[38] B. X. Jin, *An artificial compression method for the computation of contact discontinuity*, Comp. Math., 1 (1993), p. 121. (In Chinese.)
[39] J. Jones, *The spherical detonation*, Adv. Appl. Math., 12 (1991), pp. 147–186.
[40] Y. L. Zhu and B.-M. Chen, *A numerical method with high accuracy for calculating the interactions between discontinuities in three independent variables*, Scientia Sinica, 23 (1980), pp. 1491–1501.
[41] R. J. LeVeque and K.-M. Shyue, *One-dimensional front tracking based on high resolution wave propagation methods*, SIAM J. Comput., 16 (1995), pp. 348–377.
[42] R. J. LeVeque and K.-M. Shyue, *2-dimensional front tracking based on high resolution wave propagation methods*, J. Comput. Phys., 123 (1996), pp. 354–368.
[43] X.-L. Li, *Study of three dimensional Rayleigh-Taylor instability in compressible fluids through level set method and parallel computation*, Phys. Fluids A, 5 (1993), pp. 1904–1913.

[44] X.-L. Li, B. X. Jin, and J. Glimm, *Numerical study for the three dimensional Rayleigh-Taylor instability using the TVD/AC scheme and parallel computation*, J. Comput. Phys., 126 (1996), pp. 343–355.
[45] D.-K. Mao, *A treatment of discontinuities in shock-capturing finite difference methods*, J. Comput. Phys., 92 (1991), pp. 422–455.
[46] D.-K. Mao, *A treatment of discontinuities for finite difference methods in the two-dimensional case*, J. Comput. Phys., 104 (1993), pp. 377–397.
[47] R. Menikoff, *Curvature effect on steady detonation wave*, Appl. Math. Lett., 2 (1989), pp. 147–150.
[48] R. Menikoff, *Determining curvature effect on detonation velocity from rate stick experiment*, Impact Comput. Sci. Engrg., 1 (1989), pp. 168–179.
[49] R. Menikoff, *Errors when shock waves interact due to numerical shock width*, SIAM J. Comput., 15 (1994), pp. 1227–1242.
[50] G. Moretti, *Thoughts and Afterthoughts about Shock Computations*, Rep. No. PIBAL-72-37, Polytechnic Institute of Brooklyn, Brooklyn, NY, 1972.
[51] G. Moretti, *Computations of flows with shocks*, Ann. Rev. Fluid Mech., 19 (1987), pp. 313–337.
[52] G. Moretti, B. Grossman, and F. Marconi, *A Complete Numerical Technique for the Calculation of Three Dimensional Inviscid Supersonic Flow*, Rep. No. 72-192, American Institute for Aeronautics and Astronautics, New York, 1972.
[53] R. Mulder, S. Osher, and J. A. Sethian, *Computing interface motion in compressible gas dynamics*, J. Comput. Phys., 100 (1992), pp. 209–228.
[54] F. P. Preparata and M. L. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1988.
[55] R. Richtmyer and K. Morton, *Difference Methods for Initial Value Problems*, Interscience, New York, 1967.
[56] N. H. Risebro and A. Tveito, *A front tracking method for conservation laws in one dimension*, J. Comput. Phys., 101 (1992), pp. 130–139.
[57] J. Ruppert and R. Seidel, *On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra*, in Proceedings of the 5th Ann. ACM Symposium on Comput. Geom., ACM, New York, 1989, pp. 380–392.
[58] J. Scheuermann, *Efficient Solution of the Riemann Problem Using a Tabular Equation of State*, Ph.D. thesis, New York Univ., New York, 1989.
[59] D. H. Sharp, *An overview of Rayleigh-Taylor instability*, Phys. D, 12 (1984), pp. 3–18.
[60] B. Swartz and B. Wendroff, *Aztec: A front tracking code based on Godunov's method*, Appl. Numer. Math., 2 (1986), pp. 385–397.
[61] Y.-L. Zhu, B.-M. Chen, X.-H. Wu, and Q.-S. Xu, *Some New Developments of the Singularity-Separating Difference Method*, Lecture Notes in Physics 170, Springer-Verlag, Heidelberg, 1982.