

# A Conservative Front Tracking Method in N-Dimensions

A Dissertation Presented

by

Jinjie Liu

to

The Graduate School

in Partial fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

August 2006

Copyright by  
Jinjie Liu  
2006

Stony Brook University

The Graduate School

Jinjie Liu

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

---

Xiaolin Li

Advisor

Department of Applied Mathematics and Statistics

---

James Glimm

Chairman

Department of Applied Mathematics and Statistics

---

Yongmin Zhang

Member

Department of Applied Mathematics and Statistics

---

Zhiliang Xu

Outside Member

Brookhaven National Laboratory

Computational Science Center

This dissertation is accepted by the Graduate School.

---

Dean of the Graduate School

**Abstract of the Dissertation**  
**A Conservative Front Tracking Method in  
N-Dimensions**

by

Jinjie Liu

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2006

We propose a fully conservative Front Tracking algorithm for systems of nonlinear conservation laws. The algorithm can be applied uniformly to one, two and three dimensional front tracking method. We report implementation of this algorithm and the tests of fully conservative simulations for all conserved variables.

*Key Words:* conservative, tracking, space-time interface, finite volume scheme.

*To my parents, wife and daughter*

# Table of Contents

<b>List of Figures</b> . . . . .	<b>x</b>
<b>List of Tables</b> . . . . .	<b>xii</b>
<b>Acknowledgements</b> . . . . .	<b>xiii</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
1.1 Hyperbolic Conservation Laws . . . . .	1
1.2 The Front Tracking Method . . . . .	3
1.2.1 Interface Propagation . . . . .	4
1.2.2 Interface Reconstruction . . . . .	5
1.2.3 Interior Solver . . . . .	10
1.3 The Conservative Front Tracking Algorithm . . . . .	13
1.4 Dissertation Organization . . . . .	15
<b>2 The <math>N</math> Dimensional Conservative Front Tracking Algorithm</b>	<b>17</b>
2.1 The Mathematic Formulation . . . . .	17
2.2 The Space-time Control Volume Construction . . . . .	20
2.2.1 Grid Free Interface Propagation . . . . .	22

2.2.2	Grid Based Space-Time Interface Construction . . . . .	22
2.2.3	Space-Time Control Volume Construction . . . . .	35
2.3	Numerical Scheme and Flux Calculation . . . . .	36
2.3.1	Gas Dynamics . . . . .	41
<b>3</b>	<b>Numerical Implementation . . . . .</b>	<b>49</b>
3.1	Overview . . . . .	49
3.2	Front Propagation . . . . .	50
3.3	Space-time Control Volume Construction . . . . .	50
3.3.1	Major Structures . . . . .	50
3.3.2	Major Functions . . . . .	56
3.4	Numerical Solver . . . . .	62
3.4.1	Finite Difference Solver . . . . .	62
3.4.2	Finite Volume Solver . . . . .	67
3.5	Parallelization . . . . .	69
<b>4</b>	<b>Numerical Tests and Computational Results . . . . .</b>	<b>73</b>
4.1	Validity Tests and Code Debugging . . . . .	73
4.1.1	Conservation Check . . . . .	73
4.1.2	Consistency Check . . . . .	75
4.1.3	Convergence Tests . . . . .	75
4.1.4	Code Debugging . . . . .	76
4.2	2D Rayleigh-Taylor instability . . . . .	77
4.3	3D Rayleigh-Taylor instability . . . . .	81
<b>5</b>	<b>Conclusions and Future Work . . . . .</b>	<b>84</b>

Bibliography . . . . . 85

## List of Figures

1.1	A sample of three types of interface representations. Upper-left: grid free (GF) interface; Upper-right: grid based (GB) interface; Lower: locally grid based (LGB) interface. . . . .	8
1.2	Grid based 3D interface. For a two-fluid interface within each grid cell, there are $2^8 = 256$ possible configurations for the crossings of the cell edge by the interface. They can be reduced to 14 isomorphically distinct configurations of block interface. . . . .	9
1.3	Ghost cell method . . . . .	12
2.1	A 2D control volume . . . . .	18
2.2	Construct two <i>TRIANGLES</i> by connecting two <i>BONDS</i> . . . . .	24
2.3	Examples of two 3D space-time interfaces at two sequential time steps. . . . .	25
2.4	Construct three simplices by connecting two <i>TRIANGLES</i> . . . . .	26
2.5	example of 2D space-time interface construction. . . . .	29
2.6	Example of 2D incremental convex hull construction. . . . .	31
2.7	A sample 3D space-time irregular cell. . . . .	33

2.8	Flux calculation on the space-time facet $AB$ which lies on the 2D space-time cell boundary. . . . .	41
2.9	Flux calculation on the space-time facet $CD$ which lies on the 2D space-time interface. . . . .	42
3.1	A sample 3D space-time mesh. . . . .	56
3.2	A sample 2D space-time control volume construction. . . . .	57
3.3	Consistency check for 2D space-time control volume. . . . .	57
3.4	Volume merging for 2D space-time control volume. . . . .	58
3.5	Stencil states for unsplit MUSCL solver. . . . .	63
3.6	Dividing a 2D domain into two subdomains: 1 and 2 . . . . .	70
3.7	Subdomains with periodic boundaries on the left and right. . . . .	72
3.8	Subdomains with reflecting boundaries on the left and right. . . . .	72
4.1	Interface plots of 2D Rayleigh-Taylor instability simulations. The upper row shows the results by the non-conservative tracking method at time = 6.0. The lower row shows the results by the conservative tracking method at the same time. For both rows, from left to right are $10 \times 40$ , $20 \times 80$ , and $40 \times 160$ grids respectively. . . . .	79
4.2	Amplitude for 2D Rayleigh-Taylor instability simulations, as a function of time. The amplitude for a coarse mesh by conservative tracking is in approximate agreement with the amplitude by non-conservative tracking for a mesh four times finer in each spatial directions. . . . .	80

4.3	Interface plot for 3D Rayleigh-Taylor instability simulations at $t=6.0$ . Mesh: $10 \times 10 \times 40$ . Left: non-conservative tracking method; right: conservative tracking method. . . . .	82
4.4	Amplitude plot for 3D Rayleigh-Taylor instability simulations . . . . .	83

## List of Tables

2.1	Incremental convex hull algorithm. . . . .	30
3.1	Definition of the <i>SIMPLEX</i> structure . . . . .	51
3.2	Definition of the <i>ST_INTFC</i> structure . . . . .	51
3.3	Definition of the <i>STVOLUME</i> structure . . . . .	52
3.4	The meanings of eight elements in array <i>*vol_f[8]</i> . <i>*vol_f[i]</i> is the pointer to a <i>VOL_FACET</i> in <i>d</i> direction and on <i>s</i> side of the <i>STVOLUME</i> . . . . .	53
3.5	Definition of the <i>FACET</i> structure . . . . .	54
3.6	Definition of the <i>VOL_FACET</i> structure . . . . .	54
3.7	Definition of the <i>V_FACET</i> structure . . . . .	55
3.8	Sketch of the code for finite difference solver. . . . .	64
3.9	Sketch of the code for finite volume solver. . . . .	68
4.1	Debugging names for conservative front tracking code. . . . .	77
4.2	Conservation error for 2D Rayleigh-Taylor simulations. . . . .	78
4.3	Comparison of the $L_1$ errors. The numerical solution with a finer mesh ( $160 \times 640$ ) is used as the exact solution. . . . .	78

4.4	Comparison of the interface position errors. The interface on a finer mesh ( $160 \times 640$ ) is used as the exact interface. . . . .	81
4.5	Conservation error for 3D Rayleigh-Taylor simulations. . . . .	82

## Acknowledgements

I would like to express my profound gratitude to my advisor, Professor Xiaolin Li, for suggesting this important and exciting thesis topic and for his advice, support and guidance toward my Ph. D. degree. He taught me not only the way to do scientific research, but also the way to become a professional scientist. He is my advisor and a good friend.

I am deeply indebted to the support of Professor James Glimm. His scientific vigor and dedication makes him a lifetime role model for me.

I would like to thank Prof. Dekang Mao, Drs. Yingjie Liu, Zhiliang Xu, Dahai Yu, and Yongming Zhang from whom I have learned many important scientific and mathematical skills.

I would like to thank all my friends during my five-year study as a graduate student at Stony Brook for their friendship and encouragement. In particular, I would like to mention Xicheng Jia, Yuanhua Li, Hyun-Kyung Lim, Dr. Tianshi Lu, Shuqiang Wang and Dr. Yan Yu.

Throughout my academic career, the constant support of my parents and my wife Yuan, have always motivated me to strive forward. Their unconditional love has never been affected by the physical distance between us. My dissertation is dedicated to them.

# Chapter 1

## Introduction

In this chapter, we first review the theory of hyperbolic conservation laws, the front tracking method, the ghost-cell method and the previous work on conservative front tracking method, then we introduce a new conservative front tracking algorithm.

### 1.1 Hyperbolic Conservation Laws

The system of conservation laws in  $N$  spatial dimension in differential form can be written as

$$\frac{\partial U}{\partial t} + \nabla \cdot F(U) = 0, \quad F = (f_1, f_2, \dots, f_N) \quad , \quad (1.1)$$

where  $U \in R^p$  and  $f_j(U) = (f_{1j}(U), \dots, f_{pj}(U))^T \in R^p$  are defined in a spatial domain  $\Omega \subset R^N$ .

We study the *Cauchy problem* for system (1.1): Given an initial condition

$$U(\mathbf{x}, 0) = U_0(\mathbf{x}), \quad \mathbf{x} \in R^N \quad (1.2)$$

where  $U_0(x)$  is piecewise smooth function, to find a solution  $U$  for the system (1.1) which satisfies the initial condition.

The weak solution is defined as

**Definition 1.1.1** *If  $V$  satisfies the following equation (1.3) for all smooth function  $\phi$  with compact support,*

$$\int_0^\infty \int_{R^N} (V \frac{\partial \phi}{\partial t} + \sum_{j=1}^N F_j(U) \frac{\partial \phi}{\partial x_j}) dx dt + \int_{R^N} U_0(\mathbf{x}) \phi(\mathbf{x}, 0) dx = 0. \quad (1.3)$$

$V$  is called a weak solution to initial-value problem (1.1) and (1.2).

If a weak solution  $U$  of (1.3) is a piecewise  $C^1$  function, the values of  $U$  and  $F(U)$  on each side of the discontinuity are linked by a condition which is determined from the following theorem.

**Theorem 1.1.1** [23] *Let  $U: R^N \times [0, +\infty) \rightarrow \Omega$  be a piecewise  $C^1$  function (in the above sense). Then  $U$  is a solution of (1.1) in the sense of distributions on  $R^N \times [0, +\infty)$  if and only if the two following conditions are satisfied:*

- (1)  $U$  is a classical solution of (1.1) in the domains where  $U$  is  $C^1$ ;
- (2)  $U$  satisfies the jump condition

$$(U_+ - U_-)n_t + \sum_{j=1}^N (F_j(U_+) - F_j(U_-))n_{x_j} = 0 \quad (1.4)$$

along the moving space-time surface of discontinuity, where  $n = (n_{x_1}, \dots, n_{x_d}, n_t)^T$  is the normal vector to the discontinuity surface, and  $U_+$  and  $U_-$  are the limiting values of  $U$  on each side of discontinuity surface.

This jump relation (1.4) is known as the Rankine-Hugoniot condition.

## 1.2 The Front Tracking Method

Various numerical schemes have been developed for solving the conservation laws. Commonly used numerical schemes are convergent at higher order only for smooth solutions. At discontinuities, the local truncation errors are first order and the solutions are not convergent in a point-wise sense. For nonlinear discontinuities (shock waves), the width of the solution error region does not grow in time, and is generally about two mesh blocks, but for linear discontinuities, the error region is wider, some five mesh blocks, and is generally growing in time.

Several numerical methods have been developed to solve the solution discontinuity problem, such as the Front Tracking method, the Capturing method [27], and the level set method [11, 12, 36].

The Front tracking method is initiated by Richtmyer and Morton [38] and has been successfully used for high quality aerodynamic computations by Moretti, Grossman, and Marconi [33–35]. A thorough study of the Front Tracking method has been carried out by Glimm and his coworkers [5, 13, 15, 16, 18, 20–22]. See [15] for the references. A robust, validated code called *Frontier* has been developed and used in production simulation of fluid instabilities

[14–17].

The front tracking method tracks a discontinuity interface by using analytical solutions of Riemann problems across the interface, and it applies a finite difference scheme to solve the equation on different sides of the discontinuity interface by using the ghost cell extrapolation method [18, 20].

Two tasks need to be accomplished at one time step in the Front Tracking method. The first is to evolve the front dynamically. The second is to calculate the numerical solutions. We present here an overview of *FronTier* front tracking algorithm.

### 1.2.1 Interface Propagation

An interface is a collection of the geometric objects, such as *POINTS*, *CURVES*, and *SURFACES*, that correspond to zero, one, and two dimensional meshes respectively. The interfaces are represented explicitly as lower dimensional meshes. They divide the computational domain into connected regions.

The 2D interface is a set of *CURVES*, which are discretized into oriented polygonal lines. We call each linear segment of the polygonal lines a *BOND*; *CURVES* are oriented, as are *BONDS*.

The 3D interface is a set of *SURFACES*, which are discretized in terms of *TRIANGLES*. In the *FronTier* code, we simply use structure *TRI* instead of *TRIANGLE*. The linking order of the *TRIANGLES* has no intrinsic relation to the geometry of the surface.

A *POINT* containing an ordered triplet  $(x, y, z)$  of floating point coordinates is a generic description of position in space.

The *BONDS* are defined in terms of *POINTS* and neighbors. The *BOND* consists of a start *POINT* and an end *POINT*.

The *TRIANGLES* are defined in terms of *POINTS* and neighbors. The *TRIANGLE* consists of three *POINTS*.

The interface points are first propagated normally. By computing the solution to the local Riemann problem with initial states being those on either side of the interface point and using the method of characteristics, a wave speed and a new position for the interface point are determined.

The interface states are then updated by a tangential sweep, which uses a chosen interior solver with a stencil centered at the new interface point.

After interface propagation, we get a new interface at new time level. Any tangles in the new interface are need to be resolved. To resolve this problem and reduce the variance in size and aspect ration of the segments (2D) or triangles (3D) making up the interface, the new interface need to be reconstructed.

### 1.2.2 Interface Reconstruction

There are two types of interface representations: grid free (GF) interface representation [13, 15, 16] and grid based (GB) interface representation [30]. A grid free interface is independent of the underlying grid. The interface element (*BOND* for 2D, *TRI* for 3D) size is specified by the users. By reconstruction, all the points on a grid based interface lie on the cell edges. Thus, the grid

based interface elements are constructed from vertices which are the intersections between the interface and the grid lines. The grid based method is more robust, while the grid free method is more accurate. The grid free interface uses the locally grid based (LGB) method [10] to resolve topological bifurcations. The LGB method uses the GF interface except at a region of space and time where interface entities intersect with each other (or with itself). When such a topological bifurcation occurs, we first isolate the troubled region using a minimized but sufficient box to contain the bifurcation. We then use the GB method to reconstruct the interface inside the box. Finally, we relink the interface segment inside the box with the ambient grid free interface. The LGB method combines the robustness of the grid based method with the accuracy of grid free method. It is thus a significant improvement to both of these algorithms.

The grid based reconstruction employs the micro-topology within each rectangular grid block cell on the specified lattice. The grid based reconstruction is divided into three steps [16]:

1. Compute the crossings between the interface and the grid cell edges.
2. Determine components at the grid block corners and eliminate inconsistent crossings.
3. Reconstruct a new interface by using the remaining consistent crossings.

The reconstruction is based on the following three hypotheses:

1. We assume a two-fluid model. At most two fluid components intersect any individual cell in the reconstruction lattice.

2. Each cell edge has at most one interface crossing. So the reconstructed interface cross each cell edge at most once.
  
3. The portion of each cell edge that lie on one side of the reconstructed interface forms a connected set. This implies that if two corners of a cell lie in the same fluid, then the entire edge connecting those corners also remains in the same fluid.

Fig. 1.1 shows examples of these three types of interface representations in 2D. The upper-left frame in Fig. 1.1 shows an example of the grid free interface, which is independent of the underlying grid. The reconstruction process first compute the crossings on every cell edges. There are two crossings  $X$  and  $Y$  on the cell edge  $AB$  and two crossings  $P$  and  $Q$  on cell edge  $CD$ , so they are inconsistent and should be deleted. After deleting  $X$ ,  $Y$ ,  $P$  and  $Q$ , and linking the remaining crossings by line segments (the dashed line segment will be deleted), we obtain the grid based interface as in the upper-right frame of Fig. 1.1. The lower frame in Fig. 1.1 shows the LGB interface representation. The reconstruction is only performed in the box  $CDEF$  where tangle occurs, so compared with the GB interface, it is more accurate.

For 3D grid based two-fluid interface reconstruction, there are totally  $2^8$  cases in each cell and it can be reduced to 14 cases [30], see Fig. 1.2. For 4D case, there are totally  $2^{16}$  cases and can be reduced to 222 cases [1].

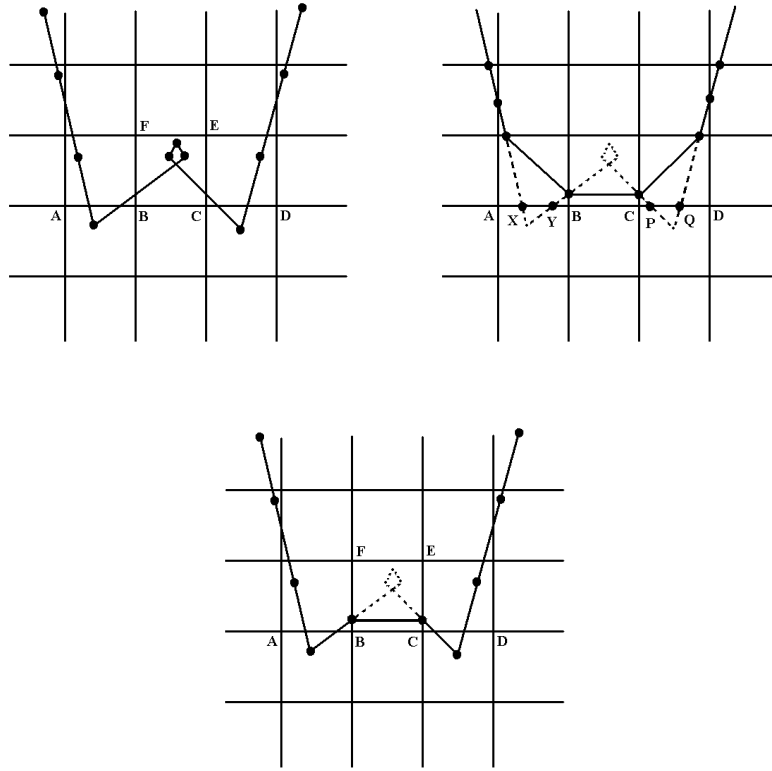


Figure 1.1: A sample of three types of interface representations. Upper-left: grid free (GF) interface; Upper-right: grid based (GB) interface; Lower: locally grid based (LGB) interface.

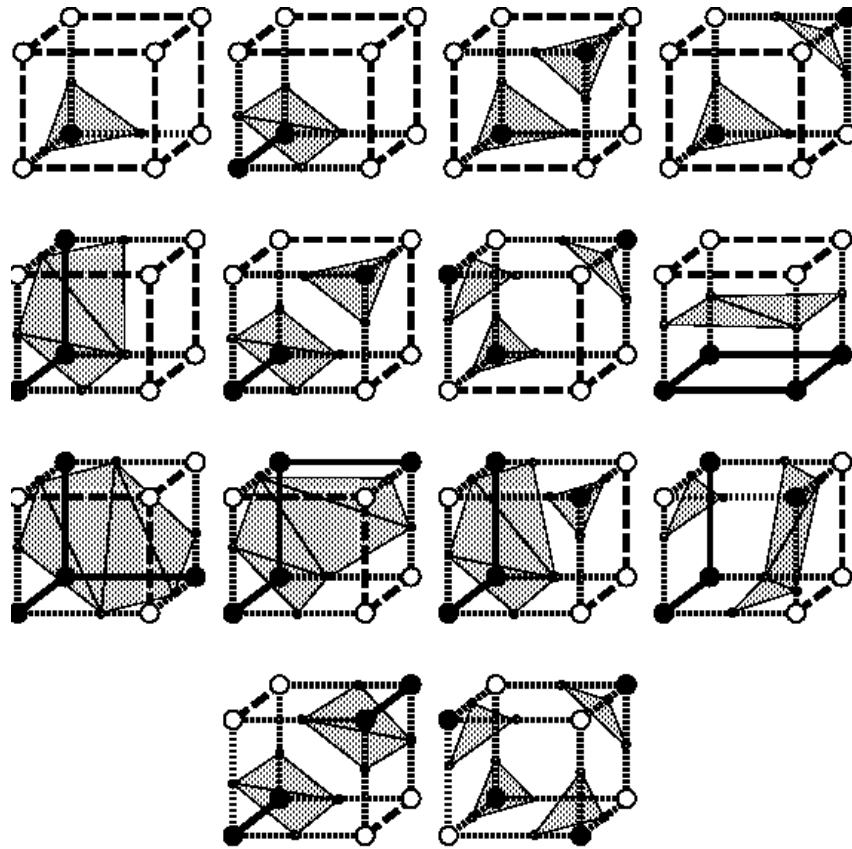


Figure 1.2: Grid based 3D interface. For a two-fluid interface within each grid cell, there are  $2^8 = 256$  possible configurations for the crossings of the cell edge by the interface. They can be reduced to 14 isomorphically distinct configurations of block interface.

### 1.2.3 Interior Solver

A connected region in the domain separated by the interface is represented by a component. Therefore, each grid node is associated with a specific component in addition to the state variables. The interior states are updated by finite difference schemes. In this dissertation, we have used the MUSCL-type schemes.

Computations near the fluid interface use ghost cells [18, 20] to avoid crossing the interface, keeping the different fluid computations entirely separate. The ghost cell method avoids interpolation across the interface. We can either define a ghost cell at every grid node in the computational domain or we can define ghost cells only on certain lattice (on the missing points of the stencil).

Consider a one dimensional conservation law

$$\frac{\partial U}{\partial t} + f_x(U) = 0 . \quad (1.5)$$

We apply the following finite difference scheme

$$U_i^{n+1} = U_i^n - \lambda(F_{i+1/2}^n - F_{i-1/2}^n), \quad (1.6)$$

where

$$U_i^n = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} U(x, t_n) dx,$$

$\lambda = \frac{\Delta t}{\Delta x}$ , and  $F_{i+1/2}^n$  is the numerical flux, defined as a function of several stencil states. For a first-order scheme, usually  $F_{i+1/2}^n = F(U_i^n, U_{i+1}^n)$ . For

example, for the first-order Godunov scheme,  $F_{i+1/2}^n = f(U_{i+1/2}^n)$ , where  $U_{i+1/2}^n$  is the Riemann solution with left state  $U_i^n$  and right state  $U_{i+1}^n$ :  $U_{i+1/2}^n = V_R(U_i^n, U_{i+1}^n)$ . This scheme is in conservation form because the total quantity is conserved. If we uniformly divide the computational domain into  $N$  intervals with  $N$  cell centers  $x_1, x_2, \dots, x_N$ . The total quantity is calculated as

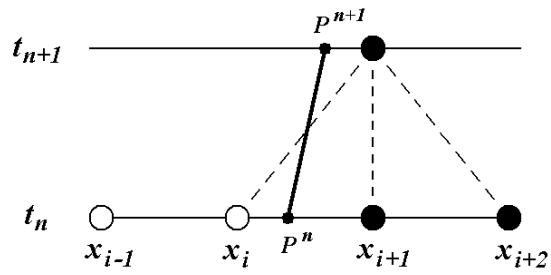
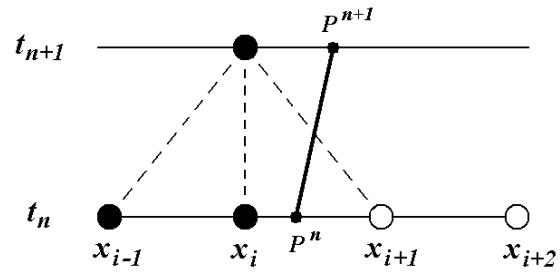
$$\begin{aligned}
Q^{n+1} &= \sum_{i=1}^N U_i^{n+1} \\
&= \sum_{i=1}^N (U_i^n - \lambda(F_{i+1/2}^n - F_{i-1/2}^n)) \\
&= \sum_{i=1}^N U_i^n - \sum_{i=1}^N (\lambda(F_{i+1/2}^n - F_{i-1/2}^n)) \quad (1.7) \\
&= \sum_{i=1}^N U_i^n - \lambda(F_{N+1/2}^n - F_{1/2}^n) \\
&= Q^n - \lambda(F_{N+1/2}^n - F_{1/2}^n).
\end{aligned}$$

So, the total quantity difference  $Q^n - Q^{n+1}$  equals to the total flux through the boundary  $(F_{N+1/2}^n - F_{1/2}^n)$ . Therefore, we say the total quantity is conserved.

If an interface (in 1D, a point) is found between  $x_i$  and  $x_{i+1}$ , the ghost-cell method will use ghost values to calculate  $F_{i+1/2}^n$ . When updating  $U_i^{n+1}$  and  $U_{i+1}^{n+1}$ , the two values of  $F_{i+1/2}^n$  are calculated differently because the ghost values are different. For example, for the first-order Godunov scheme, as in Fig. 1.3, when updating  $U_i^{n+1}$ , we use formula

$$U_i^{n+1} = U_i^n - \lambda(F_{i+1/2}^{ngL} - F_{i-1/2}^n),$$

then  $F_{i+1/2}^{ngL} = f(V_R(U_i^n, U_{i+1}^{ngL}))$ , where  $U_{i+1}^{ngL}$  is the ghost value. When updating



○ ghost value                      ● true value  
 $P^n$  interface at  $t_n$                $P^{n+1}$  interface at  $t_{n+1}$

Figure 1.3: Ghost cell method

$U_{i+1}^{n+1}$ , we use formula

$$U_{i+1}^{n+1} = U_{i+1}^n - \lambda(F_{i+3/2}^n - F_{i+1/2}^{ngR}),$$

where  $F_{i+1/2}^{ngR} = f(V_R(U_i^{ngR}, U_{i+1}^n))$ .  $U_i^{ngR}$  is the ghost value evaluated from the other side of the interface. The  $U_{i+1}^{ngL}$  and  $U_i^{ngR}$  are approximated by using the states on left and right sides of the interface respectively, so  $U_{i+1}^{ngL} \neq U_{i+1}^n$  and  $U_i^{ngR} \neq U_i^n$ . Thus in general,  $F_{i+1/2}^{ngL} \neq F_{i+1/2}^{ngR}$ . The calculation in (1.7) becomes

$$\begin{aligned} Q^{n+1} &= \sum_{j=1}^N U_j^{n+1} \\ &= \sum_{j=1}^N U_j^n - \lambda(F_{N+1/2}^n - F_{i+1/2}^{ngR} + F_{i+1/2}^{ngL} - F_{1/2}^n) \\ &= Q^n - \lambda(F_{N+1/2}^n - F_{1/2}^n) - \lambda(F_{i+1/2}^{ngL} - F_{i+1/2}^{ngR}). \end{aligned} \quad (1.8)$$

Since  $F_{i+1/2}^{ngL} - F_{i+1/2}^{ngR} \neq 0$ , the total quantity loss  $Q^n - Q^{n+1}$  does not equal to the total flux through the boundary. Thus, the total quantity is not conserved for ghost-cell method.

### 1.3 The Conservative Front Tracking Algorithm

It is important that a numerical scheme for solving the nonlinear hyperbolic conservation laws takes the conservative form. In [27], LeVeque showed that the numerical solution to the Burgers' equation could be entirely wrong if a non-conservative scheme is used. When the solution contains discontinuity, the numerical error near the discontinuity is strongly affected by the conservation of the scheme. Conservation also ensures correct propagation speed of

the discontinuity. Harten [25] pointed out that the location of the discontinuity within a grid cell can be recovered by the sub-cell resolution method which requires conservation of the numerical scheme. Lax-Wendroff theorem [26] shows that if a numerical scheme is stable, consistent and conservative, it will converge to a weak solution.

Since the ghost cell method does not preserve conservation at the cells cut by the interface and it has only first order accuracy at discontinuities, several conservative methods have been developed to solve this problem.

Harten and Hyman [24] solve this problem by merging the small cell fragments with larger ones, to maintain the CFL restriction. Their work considered one-dimensional conservation laws.

In Colella and Chern [4], and R. Pember, John Bell and Colella [37], the tracked interface moves through the finite difference mesh. The coupling of the front to the finite difference method is achieved by finite volume differencing in the irregular cells formed by the front and grid intersections. Instead of rearranging the grid, Colella and Chern redistribute the flux difference at the irregular cells algebraically to achieve conservation.

LeVeque and Shyue [28, 29] use a wave propagation method to track discontinuities. The states on irregular cells are updated by calculating the waves propagating in and out of the cells like states inside the regular cells.

Mao's method [31, 32] computes the numerical solution on each side of the discontinuity, then recovers the discontinuity position by enforcing the conservation property of the solution. It has been extended to 2D by tracking the discontinuity curves in a 1D capturing fashion and it is Cartesian grid

based.

The conservative tracking method presented in [19] uses an explicit finite volume integration scheme, derived from a generalized solution of the conservation laws in the integral form. An explicit space-time control volume generation algorithm is developed to treat the moving and deforming interface. The fluxes flowing through the space-time faces is defined explicitly. Marching Cube method [30] is used to construct the space-time interface which is one dimension higher than the spatial interface. This method uses a globally grid based interface representation. So far, the spatially 2D version has been implemented. For spatially 3D problem, to construct the 4D space-time interface, 222 cases [1] need to be handled and analyzed by hand. Thus, the implementation is difficult.

In this dissertation, we propose a new conservative front tracking method which can be applied uniformly in one, two, three and  $N$  dimensions. We use the locally grid based interface representation to improve the accuracy. The space-time interface and control volume construction is done by the convex hull algorithm, which is practical in any dimension. The spatially 1D, 2D and 3D version have been implemented. The work presented in this dissertation can be regarded as the continuation and extension of the work in [24] and [19] to  $N$ -dimensions.

## 1.4 Dissertation Organization

Chapter 1 first gives a brief review of the hyperbolic conservation laws (1.1), the general numerical methods developed for solving (1.1), especially the

Front Tracking method. Then we introduce the Conservative Front Tracking, the background of our work, and related works.

Chapter 2 discusses the Conservative Front Tracking method in  $N$  dimensional space.

Chapter 3 discusses the implementation of the Conservative Front Tracking algorithm into the *FronTier* code.

Chapter 4 contains numerical experiments.

The conclusion and a discussion of remaining open problems are presented in Chapter 5 .

## Chapter 2

# The $N$ Dimensional Conservative Front Tracking Algorithm

The  $N$ -dimensional conservative front tracking algorithm is presented in this chapter.

### 2.1 The Mathematic Formulation

Consider a system of conservation laws in  $N$  spatial dimension in differential form

$$\frac{\partial U}{\partial t} + \nabla \cdot F(U) = 0, \quad F = (f_1, f_2, \dots, f_N) \quad , \quad (2.1)$$

where  $U \in R^p$  and  $f_j(U) = (f_{1j}(U), \dots, f_{pj}(U))^T \in R^p$  are defined in a spatial domain  $\Omega \subset R^N$ .

Integrating (2.1) in a time-space domain  $\mathcal{V} \subset R^{N+1}$ , we obtain the integral form of (2.1),

$$\int_{\mathcal{V}} \left( \frac{\partial U}{\partial t} + \nabla \cdot F(U) \right) d\mathcal{V} = 0 \quad . \quad (2.2)$$

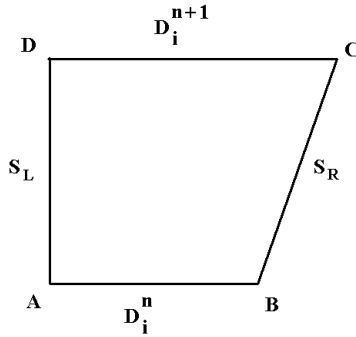


Figure 2.1: A 2D control volume

By the divergence theorem, we have

$$\int_{\partial\mathcal{V}} (U, F(U)) \cdot n dS = 0 . \quad (2.3)$$

The finite difference method presented here is an explicit finite volume integration scheme based on the integral form (2.3).

Assume a space-time discretization  $\{\mathcal{V}_i\}$  which conforms to the space-time interface as  $U$  evolves in one time step from  $t_n$  to  $t_{n+1}$ . We solve (2.3) explicitly in this region. We define each  $\mathcal{V}_i$  as a space-time control volume, where  $\partial\mathcal{V}_i = D_i^n \cup D_i^{n+1} \cup \hat{S}_i$  with  $D_i^n$ ,  $D_i^{n+1}$ , and  $\hat{S}_i$  meeting at most at their boundaries, where  $D_i^n$  and  $D_i^{n+1}$  are the boundary surfaces of  $\mathcal{V}_i$  at time level  $n$  and  $n+1$  respectively, and  $\hat{S}_i$  is the complementary boundary surface of  $\mathcal{V}_i$ . Fig. 2.1 shows a 2D volume with four boundary facets:  $AB, BC, CD,$  and  $DA$ .  $D_i^n = AB$ ,  $D_i^{n+1} = CD$ ,  $S_L = BC$ ,  $S_R = DA$  and  $\hat{S}_i = S_L \cup S_R$ . Dividing the calculation of the integral (2.3) into three parts over  $D_i^n$ ,  $D_i^{n+1}$  and  $\hat{S}_i$

respectively, we have

$$|D_i^{n+1}|\bar{U}|_{t_{n+1}} = |D_i^n|\bar{U}|_{t_n} - \int_{\hat{S}_i} (U, F(U)) \cdot n dS, \quad (2.4)$$

where

$$\bar{U}|_{t_n} = \frac{1}{|D_i^n|} \int_{|D_i^n|} U(x_1, \dots, x_N, t_n) dx_1 \dots dx_N$$

and

$$\bar{U}|_{t_{n+1}} = \frac{1}{|D_i^{n+1}|} \int_{|D_i^{n+1}|} U(x_1, \dots, x_N, t_{n+1}) dx_1 \dots dx_N$$

are cell averages,  $|D_i^n|$  is the face area of  $D_i^n$ ,  $|D_i^{n+1}|$  is the face area of  $D_i^{n+1}$ , and  $n$  is the outward normal of  $\hat{S}_i$ . Therefore,  $\bar{U}|_{t_{n+1}}$ , is the solution to (2.4) at time  $t_{n+1}$ .

To calculate  $\bar{U}|_{t_{n+1}}$ , we first need to determine the space-time control volume  $\{\mathcal{V}_i\}$ . Then we calculate the fluxes defined on the surfaces of  $\mathcal{V}_i$ , so that we can apply (2.4).

When the control volume has a constant face shape  $D_i^{n+1} = D_i^n = D_i$ , and the complementary boundary surface  $\hat{S}_i$  is parallel to the time axis, *i.e.*,  $n = (0, n')$ . Eq. (2.4) degenerates into a static finite volume scheme

$$\bar{U}|_{t_{n+1}} = \bar{U}|_{t_n} - \frac{\Delta t}{|D_i|} \int_{\hat{S}_i'} F(U) \cdot n' dS', \quad (2.5)$$

where  $\hat{S}_i'$  is the projection of  $\hat{S}_i$  on bottom spatial plane.

The conservative front tracking algorithm contains the following major steps:

1. To propagate the spatial interface at time level  $n$  to a new interface at

time level  $n + 1$ . This step will be discussed in Section 2.2.1. See [14–17] for more details.

2. To construct the space-time interface by joining two spatial interfaces at time level  $n$  and  $n + 1$ . This step will be discussed in Section 2.2.2.
3. To construct space-time control volumes by using the intersection points of the space-time interface and the space-time cell edges. Those control volumes with small or no top will be merged with their neighboring volumes to satisfy the CFL condition. This step will be discussed in Section 2.2.3.
4. To use a higher order Godunov type finite volume scheme to update the states. This step will be discussed in Section 2.3.
5. To reconstruct the propagated interface at time level  $n + 1$  to be untangled, using an LGB algorithm. See [10] for details.

## 2.2 The Space-time Control Volume Construction

The discontinuities of the numerical solution form an *INTERFACE* (*POINTS* for 1D, *CURVES* for 2D, and *SURFACES* for 3D), which is propagated from one time level to the next. These interface constituents are described in a piecewise linear manner, and made up of linear elements (simplices), which are defined by their vertices, the interface *POINTS*. The front *POINTS* are propagated by solving the Riemann Problem in the normal direction and a tangential sweep to update the states on the interface. Here and

below, we use the dimension  $D$  of an interface to refer to the ambient space in which it is embedded. Thus  $D$  may be the dimension of space or of space time, and the elements of the  $D$  dimensional interface have intrinsic dimension  $D - 1$  (co-dimension 1) or lower. See [15–17] for details.

By joining the spatial interfaces at time level  $n$  and  $n + 1$ , we construct a space-time manifold which is called the space-time interface. The space-time interface cuts the space-time grid cells into fragments when the grid cell and the interface intersect. These fragments together with the regular space time grid cells which do not meet the interface are the space-time control volumes. We apply the finite volume scheme to them to calculate the numerical solutions.

The conservative front tracking algorithm described in this paper is LGB, in other words, we apply the grid based reconstruction in the tangled region through a boxing method and in complementary regions, we use the grid free method. The LGB (GF when there is no topological bifurcation) propagated interface poses a difficult problem to the construction of irregular control volume. Therefore, to simplify the procedure of constructing space-time control volume  $\{\mathcal{V}_i\}$  in (2.4), we clip the GF space-time interface into a GB space-time interface. As a result, the space-time control volumes are grid based and their vertices are intersections of the space-time interface and regular space-time cell edges. In this section, we give the construction of the grid based space-time control volume.

### 2.2.1 Grid Free Interface Propagation

In [10] we have compared the two algorithms for the interface propagation methods, the mostly Lagrangian propagation (GF and LGB when bifurcation occurs) and the Eulerian reconstruction method (GB). The latter showed a large truncation error due to clipping to the Eulerian grid while the former is very accurate even in regions of large curvature. The error associated with the GB reconstruction, or with any grid based discontinuity description, including untracked ones, is second order. But this error typically has a constant sign, so that when summed over all time steps, the error becomes first order. Since the error is of a surface smoothing nature, we have called it numerical surface tension. In our algorithm, we have preserved the feature of fully Lagrangian propagation at all the time. However, the space-time control volumes are constructed by the grid-based interface. The grid-free interface, or its LGB reconstruction in the case of spatial tangles, is the permanent interface used within the time step propagation loop, while the space-time GB interface is only used for the space-time control volume reconstruction. While the clipping of the grid-free interface to become grid-based introduces a truncation error at each time step, this error is not involved in the interface propagation to subsequent time steps.

### 2.2.2 Grid Based Space-Time Interface Construction

Given two piecewise linear spatial *INTERFACES* which are separated in time by a step  $\Delta t$ , we construct (triangulate) a space-time surface joining

the two spatial *INTERFACES*.

To construct the space-time interface, two major tasks are needed for each space-time cell: (1) Construct a grid-free space-time interface by joining the interface at time step  $n$  and the new propagated grid free interface at time step  $n+1$ ; (2) Reconstruct the control volume by the convex hull method using the vertices of the grid based space-time interface and corners of the regular space-time control volume.

### Construction of the Grid-free Space-time Interface

Interface propagation yields a general (grid free) interface at every time step. The grid free space-time interface is formed by joining the interface  $I_n$  at time step  $n$  and the new grid free interface  $I_{n+1}$  at time step  $n+1$ . The space-time interface is described as a set of  $N$ -D simplices. An  $N$ -D simplex, also called a hyper-tetrahedron [3], is the generalization of a tetrahedral region of space to  $N$  dimensions. For example, a simplex is a line segment (*BOND*) in 2D, a triangle (*TRI*) in 3D, and a tetrahedron in 4D, etc.

For a spatially 2D problem, the space-time interface is described as a set of surfaces, as a 3D interface. The surfaces are represented as a union of 3D simplices, i.e., *TRIANGLES*. These *TRIANGLES* can be constructed by joining the *POINTS* of  $I_n$  and  $I_{n+1}$ . The 2D spatial interface is a set of *CURVES*, which are discretized as oriented *BONDS*. Let the set of *BONDS* on  $I_n$  to be  $S_B^n = \{B_k^n : k = 1, \dots, m\}$ , and every  $B_k^n$  contains start *POINT*  $P_{sk}^n$  and end *POINT*  $P_{ek}^n$ . For each  $B_k^n$  in  $S_B^n$ , we can find a corresponding *BOND*  $B_k^{n+1}$  on  $I_{n+1}$  with start *POINT*  $P_{sk}^{n+1}$  and end *POINT*  $P_{ek}^{n+1}$ , i.e.,  $B_k^{n+1}$  is

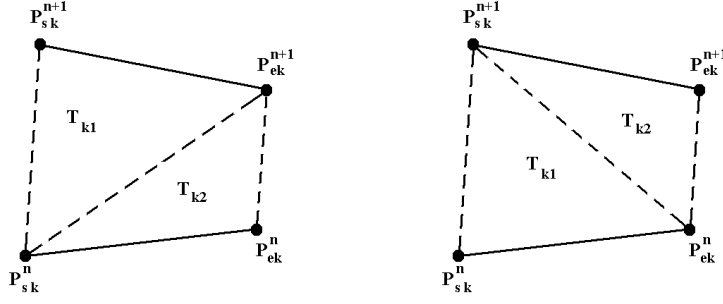


Figure 2.2: Construct two *TRIANGLES* by connecting two *BONDS*.

the propagated  $B_k^n$ .  $P_{sk}^{n+1}$  is the *POINT* of  $I_{n+1}$  into which  $P_{sk}^n$  propagates, or in short  $P_{sk}^{n+1}$  corresponds to  $P_{sk}^n$ . Similarly,  $P_{ek}^{n+1}$  corresponds to  $P_{ek}^n$ . As shown in Fig. 2.2, we construct two *TRIANGLES*  $T_{k1}$  and  $T_{k2}$  as follows:

if  $\|P_{sk}^n P_{ek}^{n+1}\| < \|P_{ek}^n P_{sk}^{n+1}\|$ , then  $T_{k1} = P_{sk}^n P_{ek}^{n+1} P_{sk}^{n+1}$  and  $T_{k2} = P_{sk}^n P_{ek}^n P_{ek}^{n+1}$ ;  
otherwise,  $T_{k1} = P_{sk}^n P_{ek}^n P_{sk}^{n+1}$  and  $T_{k2} = P_{ek}^n P_{ek}^{n+1} P_{sk}^{n+1}$ ,

where  $\|P_{sk}^n P_{ek}^{n+1}\|$  is the distance between two *POINTS*  $P_{sk}^n$  and  $P_{ek}^{n+1}$  and  $\|P_{ek}^n P_{sk}^{n+1}\|$  is the distance between two *POINTS*  $P_{ek}^n$  and  $P_{sk}^{n+1}$ . The union  $U_T = \cup\{T_{k1}, T_{k2}, k = 1, \dots, m\}$  forms the space-time interface. Fig.2.3 shows two triangulated 3D space-time interfaces at two sequential time steps.

For a spatially 3D problem, the space-time interface is a set of 4D hyper-surfaces; each is defined by a union of 4D simplices. The 4D simplex contains four vertices, linearly independent in 4D space. These 4D simplices can be constructed by joining the *POINTS* of  $I_n$  and  $I_{n+1}$  lying on corresponding triangles. The 3D spatial interface is a set of *SURFACES*, which are discretized into a set of *TRIANGLES*. Every *TRIANGLE* consists of three vertices.

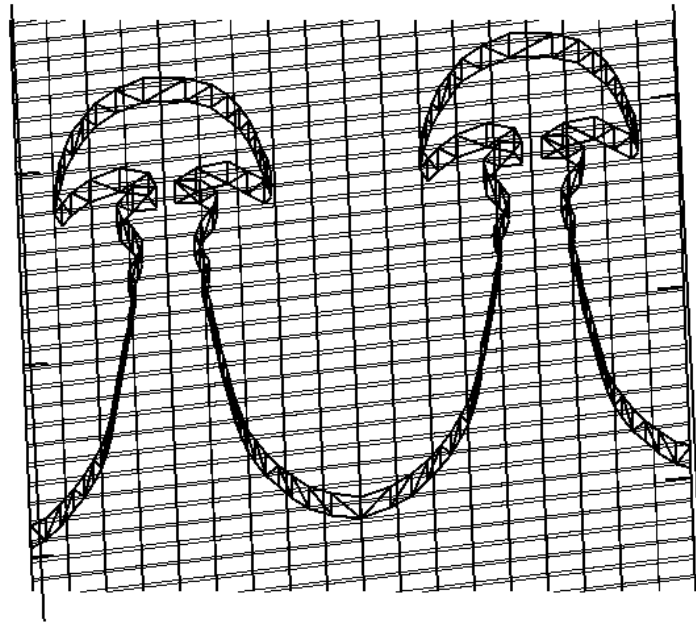
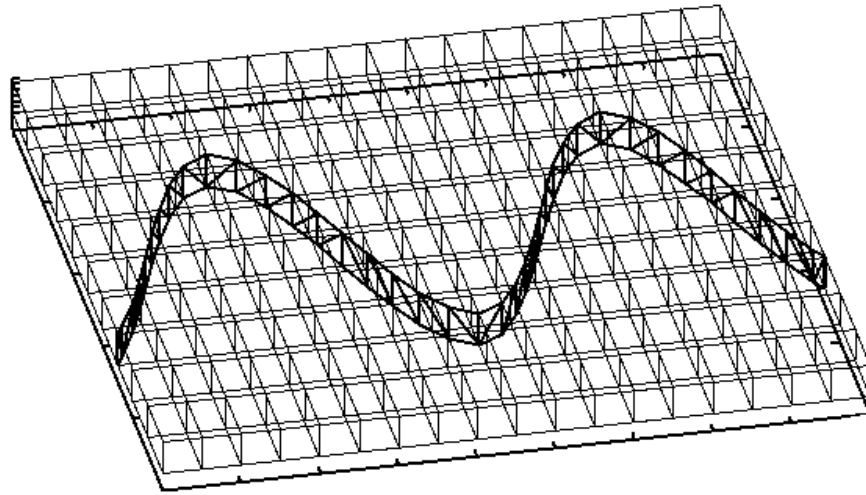


Figure 2.3: Examples of two 3D space-time interfaces at two sequential time steps.

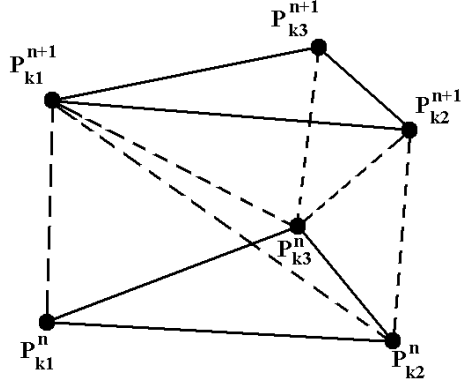


Figure 2.4: Construct three simplices by connecting two *TRIANGLES*.

Let  $T_k^n$  be an arbitrary *TRIANGLE* on  $I_n$ , with three vertices  $P_{k1}^n$ ,  $P_{k2}^n$  and  $P_{k3}^n$ . We can find a corresponding *TRIANGLE*  $T_k^{n+1}$  on  $I_{n+1}$ , with three vertices  $P_{k1}^{n+1}$ ,  $P_{k2}^{n+1}$  and  $P_{k3}^{n+1}$ , i.e.,  $T_k^{n+1}$  is the propagated  $T_k^n$ .  $P_{ki}^{n+1}$  is the corresponding *POINT* to  $P_{ki}^n$ . As shown in Fig. 2.4, we construct three 4D simplices:  $S_{k1}$  with vertices  $P_{k1}^n$ ,  $P_{k2}^n$ ,  $P_{k3}^n$  and  $P_{k1}^{n+1}$ ;  $S_{k2}$  with vertices  $P_{k3}^{n+1}$ ,  $P_{k2}^{n+1}$ ,  $P_{k1}^{n+1}$  and  $P_{k2}^n$ ; and  $S_{k3}$  with vertices  $P_{k1}^{n+1}$ ,  $P_{k2}^n$ ,  $P_{k3}^n$  and  $P_{k2}^{n+1}$ . The space-time hyper-surface joining  $T_k^n$  and  $T_k^{n+1}$  is triangulated into three 4D simplices:  $S_{k1}$ ,  $S_{k2}$  and  $S_{k3}$ . Thus, the union  $U_S = \cup\{S_{k1}, S_{k2}, S_{k3}, k = 1, \dots\}$  forms the space-time interface.

The space-time interface may be tangled. For the purpose of the finite volume update of the interior states, we reconstruct this space-time interface to be grid based to eliminate possible tangles and to simplify geometrical calculations in cut cell fragments.

## Grid Based Reconstruction of the Space-time Interface

The regular  $N$ -dimensional spatial mesh together with time grid lines form an  $(N + 1)$ -dimensional space-time mesh. The grid free space-time interface obtained from previous section will be reconstructed in the space-time mesh to obtain a grid based space-time interface. The major steps of the space-time interface reconstruction are listed below.

1. Compute crossings of the space-time grid edges with the grid free space-time interface.
2. For each space-time cell  $C$ ,
  - (2.1) pick out the crossings in space-time cell  $C$  to form set  $\{X_C\}$ ;
  - (2.2) pick out those vertices in  $C$  which lie on one side of the grid free space-time interface to form set  $\{V_C^S\}$ ;
  - (2.3) lexicographically sort points in  $\{X_C\} \cup \{V_C^S\}$  and construct a canonically triangulated convex hull  $H$  using these sorted points;
  - (2.4) obtain the space-time interface in  $C$ , which is a union of simplices, by removing those simplices of  $H$  which lie on the boundaries of  $C$ .
3. The Union of all the space-time interfaces in every space-time cell forms the space-time interface in the whole domain.

**Step 1. Finding all space-time cell edge crossings.** There are two types of space-time cell edge crossings:

$\{X_S\}$ : crossings lie on the grid edges orthogonal to  $t$  axis;

$\{X_T\}$ : crossings lie on the grid edges parallel to  $t$  axis.

$\{X_S\}$  are the intersections of the spatial interface at time level  $n$  or  $n + 1$  with the spatial grid lines.  $\{X_T\}$  are the intersections of the grid free space-time interface with  $t$  grid lines. For example, in Fig. 2.5,  $\{X_S\} = \{B, D\}$  and  $\{X_T\} = \{A, C\}$ . In Fig. 2.7,  $\{X_S\} = \{P, Q, M, N\}$  and  $\{X_T\} = \phi$ . It is possible to have multiple crossings on a grid cell edge. We apply the grid based reconstruction algorithm to remove those inconsistent crossings to guarantee there is at most one crossing on each grid cell edge. The method to remove the invalid crossings is described in [16].

**Step 2. Construction of the convex hull.** We assume a two material model, so there are at most two kinds of components. We label each space-time cell corner as positive, '+', or negative, '-', depending on the material type. Since the interface separates two materials, two points with different component labels tell us that these two points are on different sides of the interface. A convex hull is the union of several facets, and each facet is a simplex. In a space-time cell  $c$ , let  $V_c^+$  and  $V_c^-$  be the set of vertices with component label + and - respectively, and  $X_c$  be the set of crossings. Let  $P^+(c) = V_c^+ \cup X_c$ . Prior to computing the convex hull of the initial point set  $P^+(c)$ , we sort  $P^+(c)$  lexicographically. A convex hull is represented by a set of boundary facets with normal vector outwards. Each boundary facet is a simplex. Clarkson, et. al. in [6] describe an incremental convex hull algorithm which adds points to the convex hull one at a time. The incremental convex hull algorithm is shown in Table 2.1. This approach assures a canonical triangulation such that triangulation match along the space-time cell boundary.

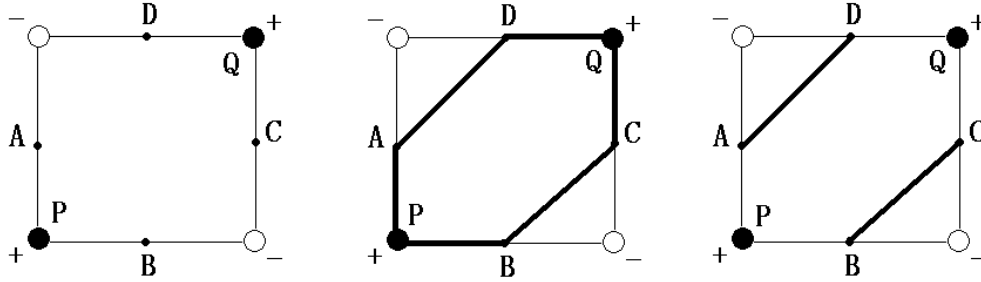


Figure 2.5: example of 2D space-time interface construction.

P. Bhaniramka, R. Wenger, and R. Crawfis [1, 2] give the detailed discussion of the isosurface construction, which is similar to our problem.

**Step 3. Reconstruction of the space-time interface.** If a facet of the convex hull lies on the space-time cell boundary, it will be removed. All the remaining facets are in the interior of the space-time cell, and they form a grid based space-time interface in this space-time cell.

Fig. 2.5 shows an example of the 2D space-time interface construction by the convex hull method. The cell corners with positive components are in black and the corners with negative components are shown in white. At first, we compute the cell edge crossings:  $A, B, C, D$ . Let  $P^+(c)$  be the union of the corners with positive components together with four crossings  $A, B, C, D$ , so

$$P^+(c) = \{A, B, C, D, P, Q\}.$$

Then we lexicographically sort the points in  $P^+(c)$ . We assume  $P$  is the lower corner and  $Q$  is the upper corner, *i.e.*, the  $x$  and  $t$  coordinates of  $P$  is less than

Incremental Convex Hull Algorithm:

1. Initialize the convex hull by the first  $N$  points.
2. For each remaining point  $P$ :
  - 2.1 Determine the set of facets that  $P$  can see. A point  $P$  can see a facet  $F$  (or  $F$  is a visible facet to  $P$ ), if  $\overrightarrow{AP} \cdot n > 0$ , where  $A$  is an arbitrary point on  $F$  and  $n$  is the normal vector of  $F$ .
  - 2.2 Determine the set of horizon facet boundaries to point  $P$ . A facet boundary is defined as the intersection of two facets. A facet boundary is horizon to point  $P$  if one of its neighbor facet is visible to  $P$  while the other one not. The horizon facet boundary separates what point  $P$  can see from what it can't see.
  - 2.3 For each horizon facet boundary, create a new facet (simplex) connecting the boundary to point  $P$ .
  - 2.4 Remove all of the facets that were previously visible to point  $P$ .

Table 2.1: Incremental convex hull algorithm.

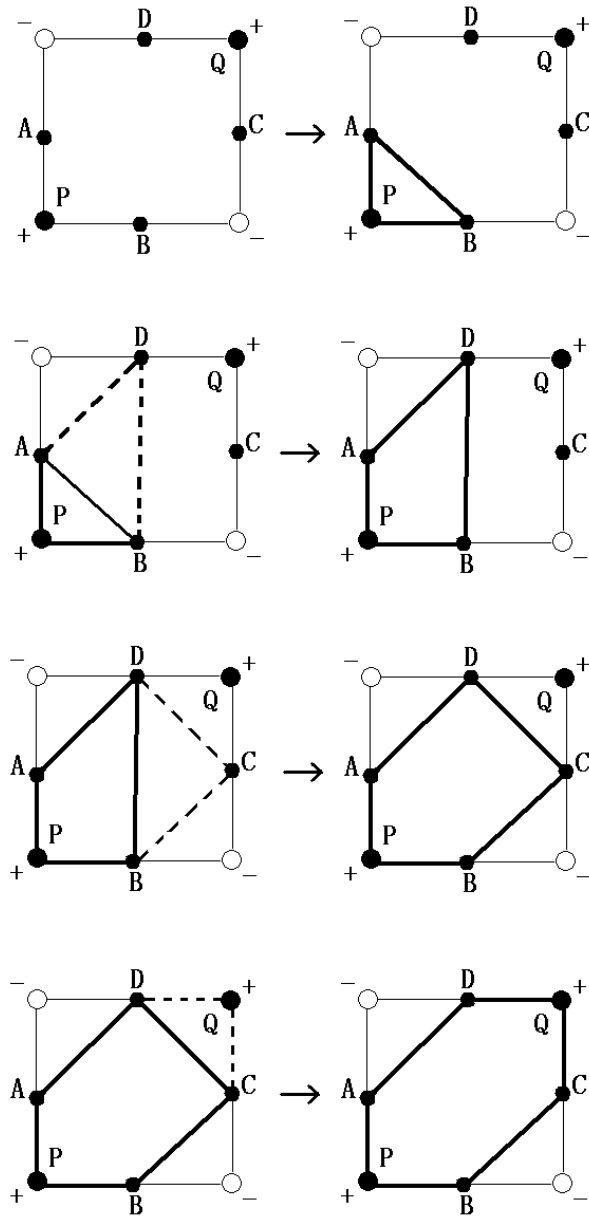


Figure 2.6: Example of 2D incremental convex hull construction.

the x and t coordinates of  $Q$  respectively. We get the new sorted set

$$\hat{P}^+(c) = \{P, A, B, D, C, Q\}.$$

Now we construct a convex hull of  $\hat{P}^+(c)$ . The procedure of convex hull construction is shown in Fig. 2.6.

- (1) Initially, we construct a convex hull by the first three points:  $P, A$  and  $B$ :

$$H_0 = PB \cup BA \cup AP.$$

- (2) Then we add other points one by one:

(2.1) Adding  $D$ : we first find a visible facet to  $P$ :  $BA$ , then we connect  $C$  with the horizon facet boundary (for 2D case, the facet boundary is a point) of  $BA$ . Since  $A$  and  $B$  are horizon, we connect  $D$  with  $A$  and  $B$ . Thus, two new facets (line segments) are added into the convex hull:  $BD$  and  $DA$ . Then we remove the previous visible facet  $BA$ . The new convex hull after this step becomes

$$H_1 = PB \cup BD \cup DA \cup AP.$$

- (2.2) Adding  $C$ : similar to (1), we get the new convex hull

$$H_2 = PB \cup BC \cup CD \cup DA \cup AP.$$



crossings. By the incremental method, we construct a convex hull  $H$  of  $P^-(c)$ . The convex hull  $H$  is a union of simplices and contains ten vertices  $\{ABDEFHPQMN\}$ . All the simplices on polygons  $ABPQD$ ,  $EHMNF$ ,  $AEFB$ ,  $DQMH$ ,  $ADHE$ ,  $BFNP$  will be deleted. The remaining simplices are  $PNM$  and  $PMQ$ , and they form the reconstructed space-time interface  $PNM \cup PMQ$  in the space-time cell.

The reconstructed grid based space-time interface in the whole domain is the union of the space-time interfaces in all the space-time cells.

### **Proof of the Correctness of the Algorithm**

The proof of the correctness for the convex hull reconstruction algorithm is given in [2].

Let  $S^+(c) = cl(\partial conv(P^+(c)) - \partial c)$ , where  $cl$ ,  $\partial$  and  $conv$  are the closure, boundary and convex operators, respectively. Let  $T^+(c)$  be the canonical triangulation of  $S^+(c)$  where the vertices are sorted in lexicographic order. A set  $T$  of simplices defines a simplicial complex if the non-empty intersection of any two or more simplices of  $T$  is a face of each of these simplices.

Theorem 1 and Theorem 2 in [2] show that if  $G$  is a regular grid whose vertices are labeled positive and negative, then the set  $T = \cup_{c \in G} T^+(c)$  is a simplicial complex, and set  $\cup_{c \in G} S^+(c)$  is a manifold with boundary. See [2] for details.

### 2.2.3 Space-Time Control Volume Construction

The reconstructed space-time interface divides each space-time cell it meets into small fragments. Each fragment is an  $N$  dimensional polytope. A polytope [9] is defined inductively as a connected volume whose boundary is a union of polytopes of one lower dimension, i.e., as the general term of the sequence "point, line segment, polygon, polyhedron, ...". If a space-time cell does not intersect with the space-time interface, then itself is a polytope. These polytopes are the space-time control volumes. The boundary of a space-time control volume (polytope) is represented as a set of boundary facets. The boundary facets on time level  $n$  are called bottom facets, the boundary facets on time level  $n + 1$  are called top facets and other facets are called space-time facets. If a space-time facet lies on the space-time interface, we call it interface facet; otherwise, we call it side facet.

As in (2.4),  $\partial\mathcal{V}_i = D_i^n \cup D_i^{n+1} \cup \hat{S}_i$ .  $D_i^n$  is the bottom facet,  $D_i^{n+1}$  is the top facet and  $\hat{S}_i$  is the union of the space-time facets. A 3D example is shown in Fig. 2.7. The space-time interface cuts the space-time cell into two fragments (control volumes):  $V_1$  and  $V_2$ .  $V_1$  has ten vertices  $\{A, B, D, E, F, H, P, Q, M, N\}$ . Let  $\partial V_1 = D_1^n \cup D_1^{n+1} \cup \hat{S}_1$ .  $D_1^n = ABPQD$  is the bottom facet,  $D_1^{n+1} = EHMNF$  is the top facet, and

$$\hat{S}_1 = AEFB \cup DQMH \cup ADHE \cup BFNP \cup PNM \cup PMQ,$$

where  $AEFB$ ,  $DQMH$ ,  $ADHE$ ,  $BFNP$ ,  $PNM$ , and  $PMQ$  are the space-time facets.  $V_2$  has six vertices  $\{Q, P, C, M, N, G\}$ . Let  $\partial V_2 = D_2^n \cup D_2^{n+1} \cup \hat{S}_2$ .

$D_2^n = CQP$  is the bottom facet,  $D_2^{n+1} = GNM$  is the top facet, and  $\hat{S}_2 = MQCG \cup GCPN \cup PMN \cup PQM$ , where  $MQCG$ ,  $GCPN$ ,  $PMN$ , and  $PQM$  are the space-time facets.

The areas and centroids of these boundary facets are calculated by triangulating them. Given an  $N$ -dimensional space-time control volume  $V$ , with  $\partial V = \cup_i \{S_i\}$  and  $S_i$  with outer normal  $n_i$ , the volume of  $V$  is calculated as follows,

$$|V| = \frac{1}{N} \sum_i -\text{sgn}(P, S_i) \text{dist}(P, S_i) |S_i| ,$$

where  $\text{dist}(P, S_i)$  is the distance between  $P$  and  $S_i$ ,  $P$  is an arbitrary point, and

$$\text{sgn}(P, S_i) = \begin{cases} 1 & \text{if } P \text{ on positive side of } S_i \\ -1 & \text{if } P \text{ on negative side of } S_i . \end{cases}$$

If a polytope has no top or no bottom or a small top or a small bottom (less than half of a regular cell area), it will be merged with one of its adjacent polytopes to make the top and bottom area greater than half of the cell area on the same side of the space-time interface. The purpose of volume merging is to satisfy the CFL condition. Numerical evidence, but no theory, seems to support this CFL construction.

## 2.3 Numerical Scheme and Flux Calculation

Let  $\mathcal{V}_i$  be a space-time control volume, and  $\partial \mathcal{V}_i = D_i^n \cup D_i^{n+1} \cup \{\hat{S}_i^j\}$ , where  $D_i^n$  is the bottom side,  $D_i^{n+1}$  is the top side and  $\{\hat{S}_i^j\}$  are other boundary sides with unit outer normal  $\{n_i^j\}$  and centroid  $\{C_i^j\}$ . By discretizing (2.5), we get

the numerical scheme

$$|D_i^{n+1}|\bar{U}^{n+1} = |D_i^n|\bar{U}^n - \sum_j |S_i^j|(\bar{U}_{i,m}^j, f_1(\bar{U}_{i,m}^j), \dots, f_N(\bar{U}_{i,m}^j)) \cdot n_i^j, \quad (2.6)$$

where  $|D_i^n|$  and  $|D_i^{n+1}|$  represents the area of  $D_i^n$  and  $D_i^{n+1}$  respectively,  $|S_i^j|$  represents the area of  $S_i^j$ , and  $\bar{U}_{i,m}^j$  is the state at  $C_i^j$ . We calculate the state  $\bar{U}_{i,m}^j$  and the fluxes through the sides  $\{S_i^j\}$  by a higher-order Godunov type method.

In (2.6), to calculate the numerical flux through those boundary sides  $\{S_i^j\}$ , we use a predictor-corrector formalism followed the idea proposed by Colella [7, 8]. At the predictor step, we construct left and right states  $U_{i,L}^j$  and  $U_{i,R}^j$  for each boundary side  $S_i^j$ . At the corrector step,  $U_{i,L}^j$  and  $U_{i,R}^j$  serve as the initial values of the Riemann problem to be solved, *i.e.*

$$\bar{U}_{i,m}^j = R(U_{i,L}^j, U_{i,R}^j).$$

Then, the numerical flux is  $f_k(\bar{U}_{i,m}^j)$ , where  $k = 1, 2, \dots, N$ .

First, we need to reconstruct left and right states  $U_{i,L}^j$  and  $U_{i,R}^j$  for each boundary side  $S_i^j$  with 2nd order accuracy. By the Taylor expansion, we have:

$$\begin{aligned} U(x_1, \dots, x_N, t) &= U(x_1^0, \dots, x_N^0, t^0) \\ &+ (t - t^0) \frac{\partial U}{\partial t} + \sum_{k=1}^N (x_k - x_k^0) \frac{\partial U}{\partial x_k} \end{aligned} \quad (2.7)$$

Substituting (2.1) into (2.7), we obtain

$$\begin{aligned}
U(x_1, \dots, x_N, t) &= U(x_1^0, \dots, x_N^0, t^0) \\
&\quad - (t - t^0) \sum_{k=1}^N \frac{\partial f_k(U)}{\partial x_k} + \sum_{k=1}^N (x_k - x_k^0) \frac{\partial U}{\partial x_k}.
\end{aligned} \tag{2.8}$$

$\frac{\partial U}{\partial x_k}$ ,  $k = 1, \dots, N$  in (2.8) is approximated by the monotonized central difference  $\frac{\Delta \hat{U}^k}{\Delta x_k}$  and  $\frac{\partial f_k}{\partial x_k}$  is approximated by a difference of Godunov fluxes.

To estimate  $\frac{\Delta \hat{U}^k}{\Delta x_k}$ , we define

$$\begin{aligned}
\frac{1}{2}(U(x_i + \Delta x_i) - U(x_i - \Delta x_i)) &= \sum \alpha_C^{v,i} r^{x_i,v}, \\
2(U(x_i + \Delta x_i) - U(x_i)) &= \sum \alpha_R^{v,i} r^{x_i,v}, \\
2(U(x_i) - U(x_i - \Delta x_i)) &= \sum \alpha_L^{v,i} r^{x_i,v},
\end{aligned} \tag{2.9}$$

where  $U(x_i \pm \Delta x_i) = U(x_1, \dots, x_i \pm \Delta x_i, \dots, x_N)$  and  $l^{x_i,v}, r^{x_i,v}, \lambda^{x_i,v}$  are the eigenvectors and eigenvalues of (2.1) projected in the  $x_i$  direction.  $\alpha_S^{v,i}$  is calculated through  $\alpha_S^{v,i} = l^{x_i,v} \cdot \Delta \hat{U}^i$ ,  $v = 1, \dots, p$ .  $S$  represents  $C$ ,  $R$ , and  $L$  respectively. Then  $\Delta \hat{U}^i$  is:

$$\begin{aligned}
\Delta \hat{U}^i &= \sum \alpha^{v,i} r^{x_i,v}, \\
\alpha^{v,i} &= \begin{cases} \min(|\alpha_C^{v,i}|, |\alpha_R^{v,i}|, |\alpha_L^{v,i}|) \times \text{sgn}(\alpha_C^{v,i}) & \text{if } \alpha_R^{v,i} \alpha_L^{v,i} > 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{2.10}$$

If  $S_i^j$  lies on the space-time cell boundary, the reconstruction is simpler. Assume  $S_i^j$  is an interior space-time facet with center  $C(t + \frac{1}{2}\Delta t, x_1^s, \dots, x_N^s)$ , and normal in  $x_h$  direction lies on the boundary of a space-time cell  $I_{st}$ . Assume the bottom face center of  $I_{st}$  is  $Z(t, x_1^z, \dots, x_N^z)$ , we have  $x_h^s = x_h^z + \frac{1}{2}\Delta x_h$  or

$x_h^s = x_h^z - \frac{1}{2}\Delta x_h$ . By applying (2.8), we get

$$\bar{U}_C = \bar{U}_Z + \left(\pm \frac{\Delta x_h}{2} - \frac{\Delta t}{2} A_h\right) \frac{\partial U}{\partial x_h} - \frac{\Delta t}{2} \sum_{k \neq h} \frac{\partial f_k(U)}{\partial x_k} + \sum_{k \neq h} (x_k^s - x_k^z) \frac{\partial U}{\partial x_k}, \quad (2.11)$$

where  $A_h(U) = \frac{\partial f_h(U)}{\partial x_h}$ ,  $\bar{U}_C = \bar{U}(t + \frac{1}{2}\Delta t, x_1^s, \dots, x_N^s)$  and  $\bar{U}_Z = \bar{U}(t, x_1^z, \dots, x_N^z)$ .

To calculate the partial derivative  $\frac{\partial f_k(U)}{\partial x_k}$  for  $k \neq h$ , we first obtain a Riemann solution  $V_R^l = R(\bar{U}_N^l, \bar{U}_Z)$  and  $V_R^r = R(\bar{U}_Z, \bar{U}_N^r)$  with the initial states  $(U_Z, U_N^l)$  and  $(U_Z, U_N^r)$ , where  $\bar{U}_N^l = \bar{U}(t, x_1^z, \dots, x_k^z - \Delta x_k, \dots, x_N^z)$  and  $\bar{U}_N^r = \bar{U}(t, x_1^z, \dots, x_k^z + \Delta x_k, \dots, x_N^z)$ . Then

$$\frac{\partial f_k(U)}{\partial x_k} = \frac{1}{\Delta x_k} (f_k(V_R^r) - f_k(V_R^l)).$$

However, for nonlinear problems, instead of using equation (2.11) directly, we make two changes according to the suggestion by Colella [8]. The first change is to discard in the  $\frac{\partial U}{\partial x}$  term the components which do not propagate toward the space-time face centroid according to the corresponding characteristics. The second is to introduce arbitrary reference states  $\tilde{U}_L$  and  $\tilde{U}_R$ , by the fact that the characteristic projection operators appearing in both the construction of the left and right states and in the solution of the Riemann problem act on the  $U$  increments.

The reference states are chosen to be:

$$\begin{aligned} \tilde{U}_L &= \bar{U}_i^n + \left(\frac{1}{2} - \max(\lambda^{x_i, p}, 0)\right) \frac{\Delta t}{2\Delta x_i} \Delta U_i^{x_i}, \\ \tilde{U}_R &= \bar{U}_{i+1}^n - \left(\frac{1}{2} + \min(\lambda^{x_i, 1}, 0)\right) \frac{\Delta t}{2\Delta x_i} \Delta U_{i+1}^{x_i}. \end{aligned} \quad (2.12)$$

where  $\bar{U}_{i\pm 1}^n = U(x_1, \dots, x_i \pm \Delta x_i, \dots, x_N)$ . Then we have:

$$\begin{aligned}\hat{U}_S &= \tilde{U}_S + P_s(\bar{U}_{i+k}^n - \tilde{U}_S) + P_s(\pm 1/2 - \frac{\Delta t}{2\Delta x_i} A_i(\bar{U}_{i+k}))(\Delta^{x_i} U)_{i+k}, \\ P_s(w) &= \sum_{v: \pm \lambda^{x_i, v}(U_{i+k}) > 0} (l_{i+k}^{x_i, v} \cdot w) r_{i+k}^{x_i, v},\end{aligned}\quad (2.13)$$

where  $S = L, R$ .

Therefore,  $\hat{U}_L$  and  $\hat{U}_R$  can be written in explicit form:

$$\begin{aligned}\hat{U}_{i,L} &= \tilde{U}_L + \frac{\Delta t}{2\Delta x_i} \sum_{v: \pm \lambda^{x_i, v}(U_{i,j}) > 0} (\lambda_i^{x_i, p} - \lambda_i^{x_i, v}) \alpha_i^{x_i, v} r_i^{x_i, v}, \\ \hat{U}_{i,R} &= \tilde{U}_R + \frac{\Delta t}{2\Delta x_i} \sum_{v: \pm \lambda^{x_i, v}(U_{i+1}) < 0} (\lambda_{i+1}^{x_i, 1} - \lambda_{i+1}^{x_i, v}) \alpha_{i+1}^{x_i, v} r_{i+1}^{x_i, v}.\end{aligned}\quad (2.14)$$

If  $S_i^j$  is on the tracked space-time interface, we first use the Riemann solver to obtain the left and the right side states  $\bar{U}_{i,l}^j$  and  $\bar{U}_{i,r}^j$  on the tracked wave, and the wave speed  $\nu_i^j$ . Then  $\bar{U}_{i,m}^j$  in (2.6) can be replaced by either  $\bar{U}_{i,l}^j$  or  $\bar{U}_{i,r}^j$ , depending on whether  $l$  or  $r$  is located within the space-time control volume  $\mathcal{V}_i$  or not. Also the  $n_i^j$  in (2.6) should be replaced by  $\tilde{n}_i^j / |\tilde{n}_i^j|$ , where  $\tilde{n}_i^j = (-\nu_i^j \sqrt{n_{ij1}^2 + \dots + n_{ijN}^2}, n_{ij1}, \dots, n_{ijN})$ ,  $n_i^j = (n_{ijt}, n_{ij1}, \dots, n_{ijN})$ . Note that  $\tilde{n}_i^j$  is the normal direction of the tracked space time wave from the Riemann solver; therefore this modification ensures that the Rankine-Hugoniot condition is satisfied.

For a spatially 1D problem, Fig. 2.8 and Fig. 2.9 show the flux calculation on the space-time facet lying on the space-time cell boundary and on the space-time interface respectively. In both figures, the left state  $\hat{U}_L$  and the right state  $\hat{U}_R$  are reconstructed by second order linear reconstruction. The reconstruction of each left or right state needs three stencil states. The reconstructed state

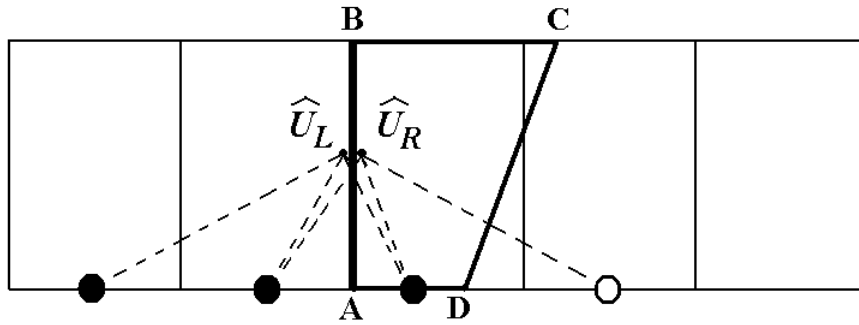


Figure 2.8: Flux calculation on the space-time facet  $AB$  which lies on the 2D space-time cell boundary.

and its corresponding stencil states are linked by dashed line. In Fig. 2.9, the upper and lower frame show the stencils used for the calculation of  $\hat{U}_L$  and  $\hat{U}_R$  respectively. The middle state on the facet is calculated as the Riemann solution  $U_m = V_R(\hat{U}_L, \hat{U}_R, )$  with left initial state  $\hat{U}_L$  and right initial state  $\hat{U}_R$ .

The finite volume scheme calculates the flux through the space-time boundary of space-time control volume. Since the boundary facet of the space-time control volume is identical when viewed from the other side, the numerical scheme is conserved.

### 2.3.1 Gas Dynamics

In this section, we give formulas for the implementation of the predictor-step for Euler equations of compressible fluid dynamics in two and three dimensions. Implementation in lower dimensions is similar.

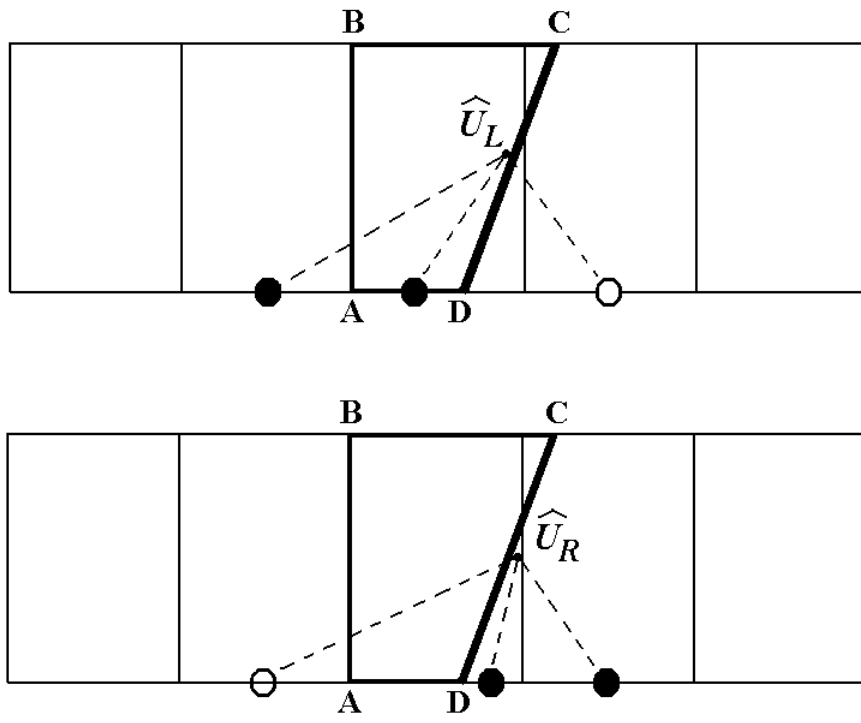


Figure 2.9: Flux calculation on the space-time facet  $CD$  which lies on the 2D space-time interface.

## 2D Gas Dynamics

The Euler equations in two dimensions are:

$$\begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}_t + \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (E + p)u \end{pmatrix}_x + \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (E + p)v \end{pmatrix}_y = 0 \quad (2.15)$$

where  $t$  is the time,  $x$  and  $y$  are the spatial dimensions,  $\rho$  is the density,  $u$  and  $v$  are the velocities,  $E$  is the total energy per unit volume, and  $p$  is the pressure. The total energy  $E$  is the sum of the internal energy and the kinetic energy,

$$E = \rho e + \frac{\rho(u^2 + v^2)}{2} \quad (2.16)$$

where  $e$  is the internal energy per unit mass. The equation of state  $p = p(\rho, e)$  completes the equation.

The eigenvalues and eigenvectors for the Jacobian matrix  $F^x(U)$  are as following. The eigenvalues are

$$\lambda^{x,1} = u - c, \quad \lambda^{x,2} = \lambda^{x,3} = u, \quad \lambda^{x,4} = u + c. \quad (2.17)$$

The eigenvectors are

$$\begin{aligned}
l^{x,1} &= \left( \frac{b_2}{2} + \frac{u}{2c}, -\frac{b_1u}{2} - \frac{1}{2c}, -\frac{b_1v}{2}, \frac{b_1}{2} \right) \\
l^{x,2} &= (1 - b_2, b_1u, b_1v, -b_1) \\
l^{x,3} &= (v, 0, -1, 0) \\
l^{x,4} &= \left( \frac{b_2}{2} - \frac{u}{2c}, -\frac{b_1u}{2} + \frac{1}{2c}, -\frac{b_1v}{2}, \frac{b_1}{2} \right)
\end{aligned} \tag{2.18}$$

$$\begin{aligned}
r^{x,1} &= (1, u - c, v, H - uc)^T \\
r^{x,2} &= \left( 1, u, v, H - \frac{1}{b_1} \right)^T \\
r^{x,3} &= (0, 0, -1, -v)^T \\
r^{x,4} &= (1, u + c, v, H + uc)^T
\end{aligned} \tag{2.19}$$

The eigenvalues and eigenvectors for the Jacobian matrix  $F^y(U)$  are as followings. The eigenvalues are

$$\lambda^{y,1} = v - c, \quad \lambda^{y,2} = \lambda^{y,3} = v, \quad \lambda^{y,4} = v + c. \tag{2.20}$$

The eigenvectors are

$$\begin{aligned}
l^{y,1} &= \left( \frac{b_2}{2} + \frac{v}{2c}, -\frac{b_1u}{2}, -\frac{b_1v}{2} - \frac{1}{2c}, \frac{b_1}{2} \right) \\
l^{y,2} &= (1 - b_2, b_1u, b_1v, -b_1) \\
l^{y,3} &= (-u, 1, 0, 0) \\
l^{y,4} &= \left( \frac{b_2}{2} - \frac{v}{2c}, -\frac{b_1u}{2}, -\frac{b_1v}{2} + \frac{1}{2c}, \frac{b_1}{2} \right)
\end{aligned} \tag{2.21}$$

$$\begin{aligned}
r^{y,1} &= (1, u, v - c, H - vc)^T \\
r^{y,2} &= (1, u, v, H - \frac{1}{b_1})^T \\
r^{y,3} &= (0, 1, 0, -u)^T \\
r^{y,4} &= (1, u, v + c, H + vc)^T
\end{aligned} \tag{2.22}$$

where

$$\begin{aligned}
q^2 &= u^2 + v^2, \quad \Gamma = \frac{p_e}{\rho}, \quad c = \sqrt{p_\rho + \frac{\Gamma p}{\rho}} \\
H &= \frac{E+p}{\rho}, \quad b_1 = \frac{\Gamma}{c^2}, \quad b_2 = 1 + b_1 q^2 - b_1 H.
\end{aligned} \tag{2.23}$$

$c$  is the sound speed,  $H$  is the specific enthalpy per unit mass.

### 3D Gas Dynamics

The Euler equations in three dimensions are:

$$\begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{pmatrix}_t + \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E+p)u \end{pmatrix}_x + \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ (E+p)v \end{pmatrix}_y + \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (E+p)w \end{pmatrix}_z = 0 \tag{2.24}$$

where  $t$  is time,  $x, y$  and  $z$  are the spatial coordinates,  $\rho$  is the density,  $u, v$  and  $w$  are the velocities,  $E$  is the total energy per unit volume, and  $p$  is the pressure. The total energy  $E$  is the sum of the internal energy and the kinetic energy,

$$E = \rho e + \frac{\rho(u^2 + v^2 + w^2)}{2} \tag{2.25}$$

where  $e$  is the internal energy per unit mass. The equation of state  $p = p(\rho, e)$  completes the equation.

The eigenvalues and eigenvectors for the Jacobian matrix  $F^x(U)$  are as follows. The eigenvalues are

$$\lambda^{x,1} = u - c, \quad \lambda^{x,2} = \lambda^{x,3} = \lambda^{x,4} = u, \quad \lambda^{x,5} = u + c. \quad (2.26)$$

The eigenvectors are

$$\begin{aligned} l^{x,0} &= \left( \frac{\Gamma e_k + cu}{2c^2}, \frac{-\Gamma u - c}{2c^2}, \frac{-\Gamma v}{2c^2}, \frac{-\Gamma w}{2c^2}, \frac{\Gamma}{2c^2} \right) \\ l^{x,1} &= \left( 1 - \frac{\Gamma e_k}{c^2}, \frac{\Gamma u}{c^2}, \frac{\Gamma v}{c^2}, \frac{\Gamma w - \Gamma}{c^2} \right) \\ l^{x,2} &= (v, 0, -1, 0, 0) \\ l^{x,3} &= (-w, 0, 0, 1, 0) \\ l^{x,4} &= \left( \frac{\Gamma e_k - cu}{2c^2}, \frac{-\Gamma u + c}{2c^2}, \frac{-\Gamma v}{2c^2}, \frac{-\Gamma w}{2c^2}, \frac{\Gamma}{2c^2} \right) \end{aligned} \quad (2.27)$$

$$\begin{aligned} r^{x,1} &= (1, u - c, v, w, H - uc)^T \\ r^{x,2} &= (1, u, v, w, e_k)^T \\ r^{x,3} &= (0, 0, -1, 0, -v)^T \\ r^{x,4} &= (0, 0, 0, 1, w)^T \\ r^{x,5} &= (1, u + c, v, w, H + uc)^T \end{aligned} \quad (2.28)$$

The eigenvalues and eigenvectors for the Jacobian matrix  $F^y(U)$  are as followings. The eigenvalues are

$$\lambda^{y,1} = v - c, \quad \lambda^{y,2} = \lambda^{y,3} = \lambda^{y,4} = v, \quad \lambda^{y,5} = v + c. \quad (2.29)$$

The eigenvectors are

$$\begin{aligned}
l^{x,0} &= \left( \frac{\Gamma e_k + cv}{2c^2}, \frac{-\Gamma u}{2c^2}, \frac{-\Gamma v - c}{2c^2}, \frac{-\Gamma w}{2c^2}, \frac{\Gamma}{2c^2} \right) \\
l^{x,1} &= \left( 1 - \frac{\Gamma e_k}{c^2}, \frac{\Gamma u}{c^2}, \frac{\Gamma v}{c^2}, \frac{\Gamma w - \Gamma}{c^2} \right) \\
l^{x,2} &= (-u, 1, 0, 0, 0) \\
l^{x,3} &= (w, 0, 0, -1, 0) \\
l^{x,4} &= \left( \frac{\Gamma e_k - cv}{2c^2}, \frac{-\Gamma u}{2c^2}, \frac{-\Gamma v + c}{2c^2}, \frac{-\Gamma w}{2c^2}, \frac{\Gamma}{2c^2} \right)
\end{aligned} \tag{2.30}$$

$$\begin{aligned}
r^{x,1} &= (1, u, v - c, w, H - vc)^T \\
r^{x,2} &= (1, u, v, w, e_k)^T \\
r^{x,3} &= (0, 1, 0, 0, u)^T \\
r^{x,4} &= (0, 0, 0, -1, -w)^T \\
r^{x,5} &= (1, u, v + c, w, H + vc)^T
\end{aligned} \tag{2.31}$$

The eigenvalues and eigenvectors for the Jacobian matrix  $F^z(U)$  are as followings. The eigenvalues are

$$\lambda^{z,1} = w - c, \quad \lambda^{z,2} = \lambda^{z,3} = \lambda^{z,4} = w, \quad \lambda^{z,5} = w + c. \tag{2.32}$$

The eigenvectors are

$$\begin{aligned}
l^{x,0} &= \left( \frac{\Gamma e_k + cw}{2c^2}, \frac{-\Gamma u}{2c^2}, \frac{-\Gamma v}{2c^2}, \frac{-\Gamma w - c}{2c^2}, \frac{\Gamma}{2c^2} \right) \\
l^{x,1} &= \left( 1 - \frac{\Gamma e_k}{c^2}, \frac{\Gamma u}{c^2}, \frac{\Gamma v}{c^2}, \frac{\Gamma w - \Gamma}{c^2} \right) \\
l^{x,2} &= (u, -1, 0, 0, 0) \\
l^{x,3} &= (-v, 0, 1, 0, 0) \\
l^{x,4} &= \left( \frac{\Gamma e_k - cw}{2c^2}, \frac{-\Gamma u}{2c^2}, \frac{-\Gamma v}{2c^2}, \frac{-\Gamma w + c}{2c^2}, \frac{\Gamma}{2c^2} \right)
\end{aligned} \tag{2.33}$$

$$\begin{aligned}
r^{x,1} &= (1, u, v, w - c, H - wc)^T \\
r^{x,2} &= (1, u, v, w, e_k)^T \\
r^{x,3} &= (0, -1, 0, 0, -u)^T \\
r^{x,4} &= (0, 0, 1, 0, v)^T \\
r^{x,5} &= (1, u, v, w + c, H + wc)^T
\end{aligned} \tag{2.34}$$

where

$$\begin{aligned}
e_k &= u^2 + v^2 + w^2, \quad \Gamma = \frac{pe}{\rho}, \quad c = \sqrt{p_\rho + \frac{\Gamma p}{\rho}} \\
H &= \frac{E+p}{\rho}.
\end{aligned} \tag{2.35}$$

$c$  is the sound speed,  $H$  is the specific enthalpy per unit mass.

## Chapter 3

# Numerical Implementation

In this chapter, we discuss the implementation of the conservative front tracking algorithm in *FronTier* front tracking code.

### 3.1 Overview

The main driver is: *hyp\_unsplit\_consv\_driver()* and it contains the following three major routines:

1. *advance\_front()*: routine for interface propagation.
2. *construct\_space\_time\_intf\_volume()*: routine for space-time interface and control volume construction.
3. *hyp\_consv\_vector()*: routine for updating the numerical solutions.

We will discuss these routines in the following sections.

## 3.2 Front Propagation

The routine *advance\_front()* has been installed into the *FronTier* code by Glimm and his coworkers [14–17]. In this routine, the interface is first propagated in the normal direction by solving Riemann problem on every point of the interface, then in the tangential direction. After propagation, interface redistribution or reconstruction will be performed to get a new untangled interface.

## 3.3 Space-time Control Volume Construction

The space-time control volume construction is one of the major routines for the conservative tracking algorithm. In this section, we first discuss some of the important data structures, then discuss the function structures.

### 3.3.1 Major Structures

In this section, we introduce some new structures for the space-time interface and control volume construction.

First, we introduce the structure *SIMPLEX*, the definition is given in Table 3.1. In *SIMPLEX* structure,  $dim = N$  is the dimension. An  $N$ -D *SIMPLEX* contains  $N$  *POINTS*:  $\_pts[0], \dots, \_pts[N-1]$ . For example, when  $dim = 2$ , two *POINTS* are defined:  $\_pts[0]$  and  $\_pts[1]$ , so the *SIMPLEX* is equivalent to a *BOND* (line segment). Similarly, when  $dim = 3$ , three *POINTS* are defined:  $\_pts[0]$ ,  $\_pts[1]$  and  $\_pts[2]$ , so the *SIMPLEX* is equivalent to a *TRIANGLE*.

```

struct _SIMPLEX
{
    int      dim;
    POINT   *_pts[MAXDIM];
};
typedef struct _SIMPLEX SIMPLEX;

```

Table 3.1: Definition of the *SIMPLEX* structure

```

struct _ST_INTFC
{
    int      dim;
    SIMPLEX **s;
    int      num_s;
};
typedef struct _ST_INTFC ST_INTFC;

```

Table 3.2: Definition of the *ST\_INTFC* structure

The space-time interface *ST\_INTFC* is represented by a set of simplices; its definition is given in Table 3.2. In this structure, *dim* is the dimension, 'num\_s' is the number of *SIMPLEX* on the *ST\_INTFC*, and double pointer *\*\*s* is an array of *num\_s* pointers to the *SIMPLEX* pointers.

The space time control volume *STVOLUME* is a polytope and is represented as an union of boundary facets, as described in Section 2.2.3. The space-time control volume structure *STVOLUME* is defined in Table 3.3. *dim* is the dimension, *volume* is the volume and *center\_comp* is the component label. There are four flags: *exist\_sign*, *is\_boundary\_vol*, *is\_regular\_vol* and *is\_FV\_vol*. Flag *exist\_sign* indicates whether the *STVOLUME* has been constructed or not. If the *STVOLUME* reaches the boundary, *is\_boundary\_vol* =

```

struct _STVOLUME
{
    int          dim;
    bool         exist_sign;
    bool         is_boundary_vol;
    bool         is_regular_vol;
    bool         is_FV_vol;
    int          num_intf_f;
    FACET       **intf_f;
    bool         vol_f_exist[8];
    VOL_FACET   *vol_f[8];
    float        volume;
    COMPONENT   center_comp;
};
typedef struct _STVOLUME STVOLUME;

```

Table 3.3: Definition of the *STVOLUME* structure

*YES* . Here, a *STVOLUME* 'reaches the boundary' means at least one of its neighbor *STVOLUME* does not exist. *is\_regular\_vol* indicates if this *STVOLUME* is degenerated to a space-time cell or not. The flag *is\_FV\_vol* indicates if we need to apply the finite volume scheme on this *STVOLUME* or not.

In Section 2.2.3, we describe the boundary facets of a space-time control volume as the union of bottom facet, top facet, side facets and interface facets. We can also divide the boundary facets of *STVOLUME* into two groups, depending on whether the facet lies on the space-time interface or not. The boundary facets lying on the space-time interface are the union of some *FACET*s. Double pointer *\*\*intf\_f* points to an array of *FACET* pointers and *num\_intf\_f* is the size of the this array, *i.e.*, how many *FACET*s.

pointer <i>*vol_f</i> [ <i>i</i> ]	direction <i>d</i>	side <i>s</i>
<i>*vol_f</i> [0]	<i>t</i>	left (bottom)
<i>*vol_f</i> [1]	<i>t</i>	right (top)
<i>*vol_f</i> [2]	<i>x</i>	left
<i>*vol_f</i> [3]	<i>x</i>	right
<i>*vol_f</i> [4]	<i>y</i>	left
<i>*vol_f</i> [5]	<i>y</i>	right
<i>*vol_f</i> [6]	<i>z</i>	left
<i>*vol_f</i> [7]	<i>z</i>	right

Table 3.4: The meanings of eight elements in array *\*vol\_f*[8]. *\*vol\_f*[*i*] is the pointer to a *VOL\_FACET* in *d* direction and on *s* side of the *STVOLUME*.

If a boundary facet does not lie on the space-time interface, it is called a *VOL\_FACET*. A *VOL\_FACET* can be either a bottom facet, a top facet or a side facet, so it is orthogonal to the grid lines in a particular direction. For example, the bottom and top facets are orthogonal to *t* grid lines. We call a *VOL\_FACET* in *x<sub>i</sub>* or *t* direction, if it is orthogonal to *x<sub>i</sub>* or *t* grid lines. For a *VOL\_FACET* in *x<sub>i</sub>* or *t* direction, it can be either on the left hand side or on the right hand side of the *STVOLUME*. There are *N* possible directions in *N*-D space, and in each direction we have two possible sides, so we have totally  $2N$  possible *VOL\_FACET*s for each *STVOLUME*. Pointer array *\*vol\_f*[8] has 8 elements since we have at most 8 *VOL\_FACET* for a *STVOLUME* with dimension less than or equal to 4. The meanings of 8 elements are shown in Table 3.4.

The interface facet is represented by the structure *FACET*, see Table 3.5 for declaration. A *FACET* is a simplex with more information, such as the normal, the area, and the centroid. The *FACET* structure contains the dimen-

```

struct _FACET
{
    int          dim;
    int          id;
    POINT        *pt[MAXDIM];
    float        nor[MAXDIM];
    float        area;
    float        centroid[MAXDIM];
    Locstate     state;
    bool         state_found;
    COMPONENT    neg_comp, pos_comp;
    struct _STVOLUME *nbhd_vol[2];
};
typedef struct _FACET FACET;

```

Table 3.5: Definition of the *FACET* structure

sion *dim*, the index *id*, an array to the vertices *\*pt[]*, the normal *nor*, the area *area*, the coordinates of the centroid *centroid[]*, two components on both sides of the *FACET*: *pos\_comp* and *neg\_comp*, and two neighbor *STVOLUME*s: *\*nbhd\_vol[0]* and *\*nbhd\_vol[1]*. *state* is the pointer to the numerical solution, and the flag *state\_found* indicates if the *state* has been computed or not.

The *VOL\_FACET* structure is a linked list as shown in Table 3.6.

```

struct _VOL_FACET
{
    V_FACET      *f;
    float        total_area;
    struct _VOL_FACET *next;
};
typedef struct _VOL_FACET VOL_FACET;

```

Table 3.6: Definition of the *VOL\_FACET* structure

```

struct _V_FACET
{
    int          dim;
    int          id;
    int          num_pt;
    POINT       **pt;
    float        area;
    float        centroid[MAXDIM];
    bool         state_found;
    Locstate     state;
    int          fg_id;
    struct _STVOLUME *nbhd_vol[2];
};
typedef struct _V_FACET V_FACET;

```

Table 3.7: Definition of the *V\_FACET* structure

*V\_FACET* contains the information of the current facet, *total\_area* is the total area of all the facets on the linked list, pointer *next* points to the next *VOL\_FACET*. This linked list structure for *VOL\_FACET* simplifies the control volume merging process. When we merge two control volumes, the corresponding *VOL\_FACET* need to be merged, *i.e.*, the *VOL\_FACET* with same index need to be merged. By using the linked list structure, two *VOL\_FACET*s can be easily merged by just link one at the end of the other one, and removing the overlapped *VOL\_FACET*s.

The structure *V\_FACET* is defined in Table 3.7. A *V\_FACET* is a polygon for a 3D *STVOLUME*, and a polyhedron for 4D *STVOLUME*. It contains *num\_pt* *POINT*s, and they are stored in the double pointer *\*\*pt*. The meaning of *dim*, *id*, *area*, *centroid[]*, *state*, *state\_found* are the same as the *FACET* structure. *V\_FACET* also has two neighbor *STVOLUME*s:

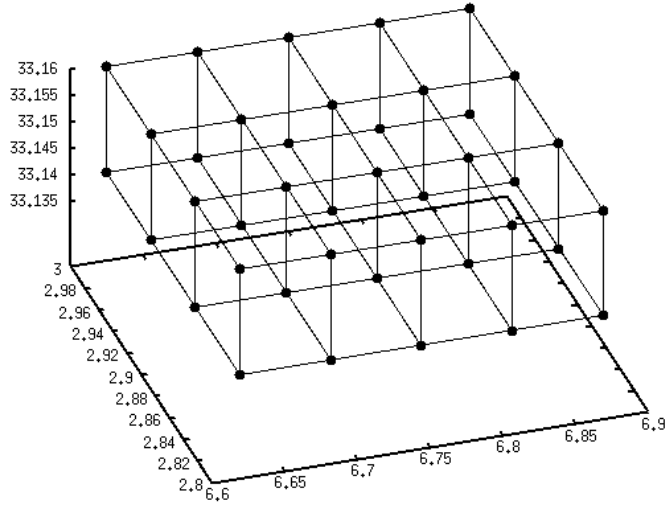


Figure 3.1: A sample 3D space-time mesh.

$nbhd\_vol[0]$  and  $nbhd\_vol[1]$ .  $fg\_id$  is only used when the  $V\_FACET$  is on a bottom or top  $VOL\_FACET$ . The spatial interface cuts the grid cell into fragments. Each fragment is a  $V\_FACET$ , and  $fg\_id$  is the index of this  $V\_FACET$  in the corresponding grid cell.

### 3.3.2 Major Functions

The main driver for the space-time interface and control volume construction is  $construct\_space\_time\_intfc\_volume()$ . For spatially  $N$ -D problem, the major job is done by the subroutine  $construct\_space\_time\_volume\_Nd()$ . It contains the following several major subroutines:

1.  $init\_space\_time\_mesh\_Nd()$ : to initialize the space time mesh for a spatial  $N$ -D problem. For a spatially  $N$ -D problem, the space time mesh is in  $(N + 1)$ -D. Let  $M = N + 1$ . The space-time mesh constructed by

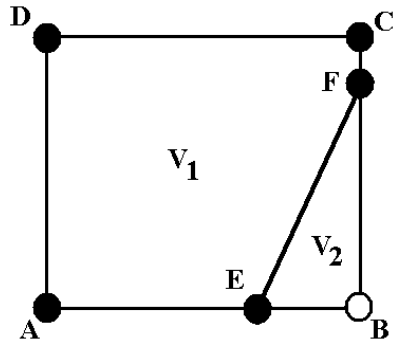


Figure 3.2: A sample 2D space-time control volume construction.

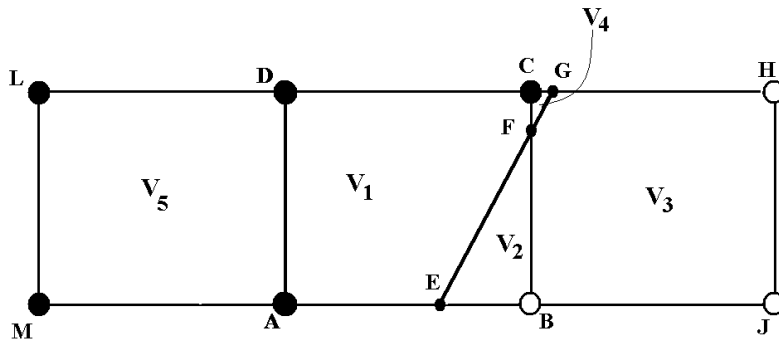


Figure 3.3: Consistency check for 2D space-time control volume.

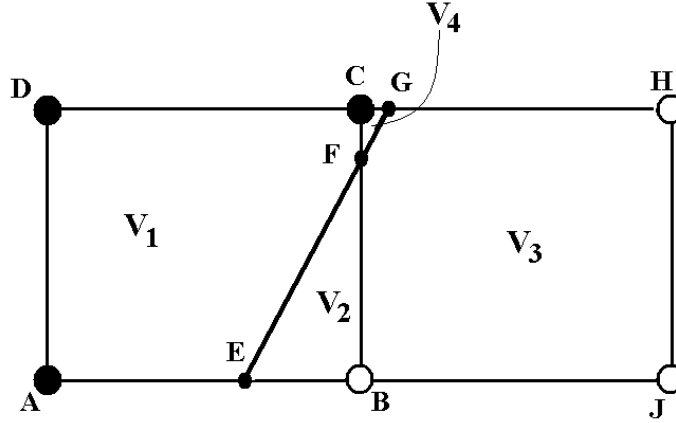


Figure 3.4: Volume merging for 2D space-time control volume.

this routine is a  $M$ -D mesh. Fig. 3.1 shows a 3D space-time mesh with mesh size  $4 \times 3 \times 1$ . The  $M$ -D space time mesh is represented by an  $M$  dimensional array of  $M$ -D blocks. A  $M$ -D block  $BLK_{MD}$  contains  $2^M$  corner points,  $M * 2^{M-1}$  edges, and each edge contains at most one crossing under the grid based assumption.

2. *construct\_Md\_st\_intf()*: to construct a grid free  $M$ -D space-time interface by connecting two spatial interfaces at time step  $n$  and  $n + 1$ . Before calling *advance\_front()*, every point on the interface is given an index, and different points have different indices. After propagation, every point has a target point on the new interface with the same index. Each simplex on the interface also has a target simplex which can be found by identifying the vertices. By connecting an  $N$ -D simplex and its target simplex, we can construct an  $(N + 1)$ -D prism, which can then

be decomprised into a union of  $(N + 1)$ -D simplices. The union of all the  $(N + 1)$ -D simplices is the space-time interface. A sample has been shown in Fig. 2.3.

3. routines for inserting crossings:

3.1 *insert\_crossings\_into\_space\_time\_Md\_mesh()*: to compute the crossings on the top or bottom grid lines of the space-time mesh. The crossings on the bottom grid lines are calculated by finding the intersections of the spatial grid lines with the spatial interface at time step  $n$ . Similarly, the crossings on the top are the intersections of the spatial grid lines with the spatial interface at time step  $n + 1$ . The major subroutines for this function are *insert\_grid\_crossingsNd()* and *remove\_unphysical\_crossingsNd()*.

3.2 *insert\_t\_crx\_into\_st\_Md\_mesh()*: to compute crossings on time grid lines. Before compute the  $t$ -crossings, *make\_st\_intf\_simplex\_list()* is called. Here *simplex* can be *TRI* (3D) or *TETRA* (4D) depending on the dimension of the problem. For each space-time block, this function computes the number of simplices related to it and stores these simplices as a list. So, by using the simplex list, when computing the crossings on each space-time block, we do not need to search all the simplices on the space-time interface.

4. For each space time block, to reconstruct space-time interface and to construct the space-time control volumes.

4.1 *construct\_Md\_rect\_volume()*: to construct regular control vol-

ume. A control volume is regular if it does not intersect with the space-time interface, otherwise, it is irregular. A regular control volume is a space-time cell. An  $N$ -D regular control volume has a bottom facet, a top facet and  $2(N - 1)$  side facets. It does not have interface facets and the volume equals to the volume of the space-time cell. A boundary facet is rectangle or cuboid, so the area and centroid can be calculated easily.

4.2 *construct\_space\_time\_intf\_and\_vol\_Nd()*: to construct irregular control volumes. It contains three major functions:

4.2.1 *divide\_MD\_stblk\_points\_in\_groups()*: to divide all the points (including corners and edge crossings) in the space-time block into several groups. In each group, all the corner components are the same. For example, in Fig. 3.2, the points are divided into two groups:  $G_1 = \{A, C, D, E, F\}$  and  $G_2 = \{B, E, F\}$

4.2.2 *construct\_Md\_hull\_volume\_intf()*: to construct the convex hull for every point group with component label  $\alpha$ . Each convex hull can be used to construct a control volume. The convex hull structure *STHULL* contains a set of *FACET*s. Those *FACET*s lying on the space-time cell boundaries are used to construct the bottom, top and side facets (*VOL\_FACET*s), and others are used to construct the interface facet. *calculate\_premerge\_volume\_facet\_areas()* is the routine to calculate the boundary facet areas. In Fig. 3.2, if  $\alpha$  is represented in black, then this function constructs the space-time interface *FE* and the control volume *ACDFE*.

4.2.3 *construct\_other\_Md\_vol\_in\_Md\_stblk()*: to construct those control volumes with component label  $\beta$ . Each boundary facet of the control volume with component  $\beta$  is constructed by subtraction of the facets constructed in previous step from the space-time cell boundary facet. *calculate\_extra\_volume\_facet\_areas()* is the routine to calculate facet areas of the complementary control volumes. For example, this step constructs the control volume *BFE* in Fig. 3.2.

4.3 *reset\_block\_volume\_facet()*: to check the consistency of the control volume and to delete the overlapped facets. After the previous two steps, all the control volumes in the current space-time block have been constructed. For a given control volume, this routine searches its neighboring control volumes on the same side of the space-time interface in every direction. If a neighbor is found, it checks whether the common facets of two adjacent control volumes are equal to each other. If so, the overlapped facets should be deleted so that two adjacent control volume share the same common facets. For example, in Fig. 3.3,  $V_1$  has a right neighbor  $V_4$  and a left neighbor  $V_5$  on the same side of the space-time interface. This routine checks whether the right boundary of  $V_1$  matches with the left boundary of  $V_4$  and whether the left boundary of  $V_1$  matches with the right boundary of  $V_5$ .

5. *reconstruct\_st\_volumes()*: volume merging. For every space-time control volume, if its bottom and the top are greater than half regular spatial cell area, function *merge\_side\_neighbor\_volumes()* merges neighboring control volumes with small or no top or bottom with it. The routine to

merge two control volumes is *merge\_two\_st\_volumes()*. In Fig. 3.4, the control volume  $V_4$  has a small top  $CG$ . It should be merged with its neighboring control volume  $V_1$ . After merging  $V_4$  with  $V_1$ , the interface facets of  $V_1$  becomes  $EF \cup FG$ ; the top facet of  $V_1$  becomes  $DC \cup CG$ ; and the side facet  $CF$  and  $V_4$  should be deleted. Similarly,  $V_2$  does not have top, so it should be merged with  $V_3$ . After merging,  $V_3$  has two interface facets  $EF$  and  $FG$ ; the bottom of  $V_3$  becomes two pieces:  $BJ$  and  $EB$ ; and  $BF$  and  $V_2$  are deleted.

### 3.4 Numerical Solver

The driver of the numerical solver is *hyp\_consv\_vector()*. It includes an unsplit finite difference solver and a finite volume solver. To improve the efficiency, the finite volume scheme is only performed near the front.

#### 3.4.1 Finite Difference Solver

The finite difference solver implemented here is a second order unsplit Godunov-type MUSCL scheme, using a predictor-corrector, following ideas proposed by Colella [7,8]. The detailed mathematical formulation has been discussed in Chapter 2. For an  $N$ -D problem, we do  $N$  directional sweeps. In each sweep, we calculate the flux in a direction. After all the sweeps, the flux in all directions are added, to complete the calculation. This technique is different from the splitting scheme because the  $N$  directional sweeps are independent in the unsplit scheme, while they depend on the result of the previous sweep

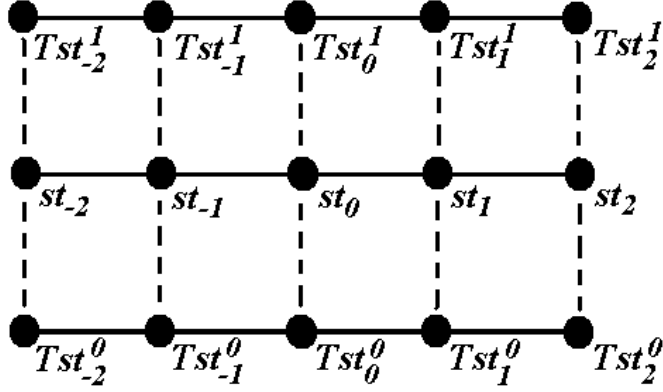


Figure 3.5: Stencil states for unsplit MUSCL solver.

in the splitting scheme. The scheme implemented here is a five-point scheme. It needs five stencil states in every sweep directions, and stencil states in the tangential directions are also needed. Fig. 3.5 shows an example of the stencils for a 2D problem. The stencil states in sweep direction are  $\{st_j, j = -2, \dots, 2\}$ . The stencil states in tangential directions are  $\{Tst_j^k, k = 0, 1, j = -2, \dots, 2\}$ . If all the stencil points do not cross the interface, the state is called interior state, otherwise, it is called close-to-front state.

The major routines for finite difference sweep is presented in Table 3.8. and the usage of these functions are listed as follows.

1. *sweep\_3l\_comp\_segments()*: to update interior states. In a particular direction, this function sets up the sweep limits and does the vectorized

```

sweep_3l_comp_segments()
    ghyp_unsplit_vec()
        unsplit_MUSCL_in_dir()

hyp_unsplit_npt()
    unsplit_update_reg_grid_state()
        set_unsplit_stencil_comps_and_crosses()
        find_outer_states()
        find_out_states_in_other_dir()
        unsplit_npt_solver()
            unsplit_MUSCL_in_dir()

```

Table 3.8: Sketch of the code for finite difference solver.

sweep by calling the following subroutine:

*unsplit\_vec\_solver()*. This function sets up the stencil states and calls the unsplit MUSCL solver: *unsplit\_MUSCL\_in\_dir()*.

2. *hyp\_unsplit\_npt()*: to update close-to-front states. After setting up the sweep limits, for each cell center point near the front, it calls the following subroutine to update the numerical solution at the current point.

*unsplit\_update\_reg\_grid\_st()*: The following three subroutines are used to set stencils and the last subroutine is the finite difference solver.

*set\_unsplit\_stencil\_comps\_and\_crosses()*: to the components and crossings for stencil points.

*find\_outer\_states()*: to find the stencil states in sweep direction.

*find\_out\_states\_in\_other\_dir()*: to find the stencil states in the tangential directions.

*unsplit\_npt\_solver()*: to update state. After the stencil states have been set up, this function is called to calculate the states at the new time step. The kernel of this function is also the unsplit MUSCL solver *unsplit\_MUSCL\_in\_dir()*.

The unsplit MUSCL solver *unsplit\_MUSCL\_in\_dir()* is similar to the split MUSCL solver *oned\_MUSCL()* in *FronTier*. It first sets up the structure *Vec\_Muscl* which contains all the necessary information for the MUSCL solver. Then we calculate the eigenvalues and eigenvectors. After calculating the viscosity if necessary, we reconstruct linearly the left and right states near the

cell boundaries. Then a Riemann solver is called to calculate the cell boundary state and flux. Finally, by adding up all the flux into the old state at time level  $n$ , we get the updated new state at time level  $n + 1$ . Major subroutines of the unsplit MUSCL solver is listed below.

- 1 *g\_load\_unsplit\_muscl\_state\_data()*: to load the stencil states and all necessary information into *Vec\_Muscl* structure.
- 2 *g\_load\_newucon\_state\_data()*: to set up the old state: *pucon*.
- 3 *g\_compute\_Ndunsplit\_eigens()*: to compute the eigenvalues and eigenvectors.
- 4 *g\_compute\_unsplit\_art\_visc\_coefs()*: to calculate the viscosity parameters.
- 5 *g\_unsplit\_bct\_linear\_reconstructor()*: to compute the linear reconstruction of the state variables.
- 6 *g\_unsplit\_N\_half\_step()*: to calculate the left and right states for half step.
- 7 *g\_unsplit\_T\_half\_step()*: to add the contribution in the tangential directions for the left and right states calculation.
- 8 *g\_unsplit\_muscl\_exact\_solver()*: Riemann solver. Given left and right states calculated previously, the middle state is found by this solver and the flux is also calculated.
- 9 *g\_unsplit\_add\_art\_visc()*: to add viscosity.

10 *g\_unsplit\_cons\_src()*: to compute the source terms.

### 3.4.2 Finite Volume Solver

The major routines for the finite volume solver is presented in Table 3.9. *hyp\_cons\_v\_FV\_npt()* is the main driver. The usage of the functions in Table 3.9 is described as follows.

1. *set\_vol\_btm\_states()*: to set the bottom states for all the control volumes.
2. *find\_st\_volume\_facet\_center\_states()*:

2.1 *g\_intf\_facet\_state()*: to calculate the state on the interface facets. Some major subroutines are

2.1.1 *set\_ff\_stencil()*: to set up the stencils. An interface facet has two sets of stencils on different sides of the interface and with different components.

2.1.2 *vol\_intf\_facet\_mid\_state()*: to calculate the interface facet state. First, *ff\_lr\_state\_reconstruction()* does the second order linear reconstruction of the left and right states of the interface facet. Then a Riemann problem is solved by using the left and right states to compute the interface facet center state.

2.2 *g\_side\_facet\_state()*: to calculate the state on the side facets. Some major subroutines are

2.1.1 *set\_vf\_stencil()*: to set up the stencils. A *VOL\_FACET* needs four stencil points, two on the left and two on the right and stencil

```

hyp_consv_FV_npt()
    set_vol_btm_states()

    find_st_volume_facet_center_states()
        g_intf_facet_state()
            set_ff_stencil()
            vol_intf_facet_mid_state()

        g_side_facet_state()
            set_vf_stencil()
            vol_facet_FD()

    calculate_total_bottom_side_facet_flux():

    add_total_intf_facet_flux():

    updating the top state and adding source terms.

    update_new_wave_fg_states():

```

Table 3.9: Sketch of the code for finite volume solver.

points on tangential directions too.

2.1.2 *vol\_facet\_FD()*: to calculate side (space-time) facet center state. By using the stencil states, we compute the left and right states by linear reconstruction, then we solve a Riemann problem to get the state on the side facet.

2.3 *calculate\_total\_bottom\_side\_facet\_flux()*: to calculate the total flux through the bottom and the side facets.

2.4 *add\_total\_intf\_facet\_flux()*: to calculate the total flux through the interface facets.

2.5 Routines for updating the top state and adding source terms.

3. *update\_new\_wave\_fg\_states()*: to copy the fragment states on top facets of the control volumes into structure *newwave* and to update the regular spatial cell center states if necessary.

### 3.5 Parallelization

The conservative front tracking method is implemented for both serial and parallel machines. The parallel implementation uses domain decomposition. The computational domain is divided into subdomains, and the numerical solution in each subdomain is solved by the corresponding processor. The boundaries between adjacent subdomains are called subdomain boundaries. For the most part, the code is the same as the serial case, except to communicate information near the subdomain boundaries. For example, in Fig. 3.6,

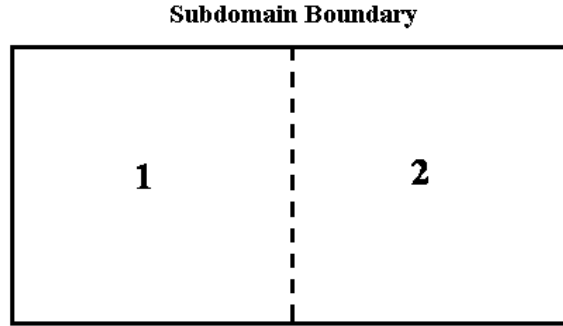


Figure 3.6: Dividing a 2D domain into two subdomains: 1 and 2

the domain is divided into two subdomains. Since we use second order scheme, which is a five-point scheme, each subdomain must have two buffer states in every directions. A scheme with different stencil size would require different buffer zones. The buffer zone is also needed for the normal propagation of the front.

Only the information (including interface information and the numerical solutions) inside the domain will be updated, the information in buffer zone will not be updated. If a buffer zone across the subdomain boundary, the information in this buffer zone should come from the adjacent processors by parallel communication. Besides the interface data and the numerical solutions at rectangular grid cell centers, the numerical solutions on the triangulated mesh also need parallel communication. There are three major functions for parallel communication:

1. *scatter\_front()* does the parallel communication of interface data;
2. *scatter\_states()* is the driver for parallel communication of the rectan-

gular cell center states;

3. *scatter\_fg\_states()* is the routine for parallel communication of the fragment states, which is defined on the triangulated mesh. The fragment state represents the average state on the bottom or top boundary facet of the space-time control volume.

This parallel communication is also used for other boundaries, such as periodic boundary, reflecting boundary, etc. For example, if the boundary in  $x$ -direction is periodic (Fig. 3.7), then subdomain 1 would be adjacent to subdomain 2 on both the left and the right. So the left boundary of subdomain 1 is periodic boundary and the right boundary is subdomain boundary. Thus, the information in buffer zone  $B_{1L}$  should be copied from  $A_{2R}$ , and information in buffer zone  $B_{1R}$  should be copied from  $A_{2L}$ . Similarly, the left boundary of subdomain 2 is subdomain boundary and the right boundary of subdomain 2 is periodic boundary. The information in buffer zone  $B_{2L}$  should be copied from  $A_{1R}$ , and information in buffer zone  $B_{2R}$  should be copied from  $A_{1L}$ . If the boundary is reflecting (Fig. 3.8), the left boundary of subdomain 1 is reflecting boundary and the right boundary is still subdomain boundary. The information in buffer zone  $B_{1L}$  should be formed by reflecting  $A_{1L}$ , and information in buffer zone  $B_{1R}$  should be copied from  $A_{2L}$ . Similarly, the left boundary of subdomain 2 is subdomain boundary and the right boundary of subdomain 2 is reflecting. The information in buffer zone  $B_{2L}$  should be copied from  $A_{1R}$ , and information in buffer zone  $B_{2R}$  should be formed by reflecting  $A_{2R}$ .

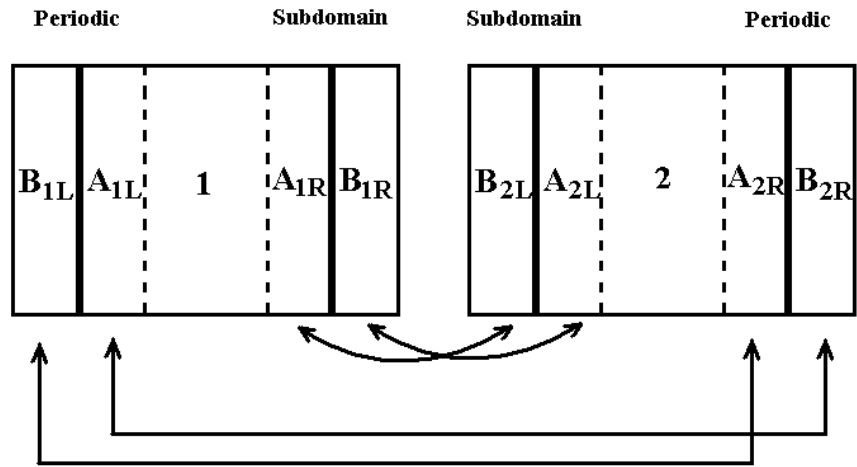


Figure 3.7: Subdomains with periodic boundaries on the left and right.

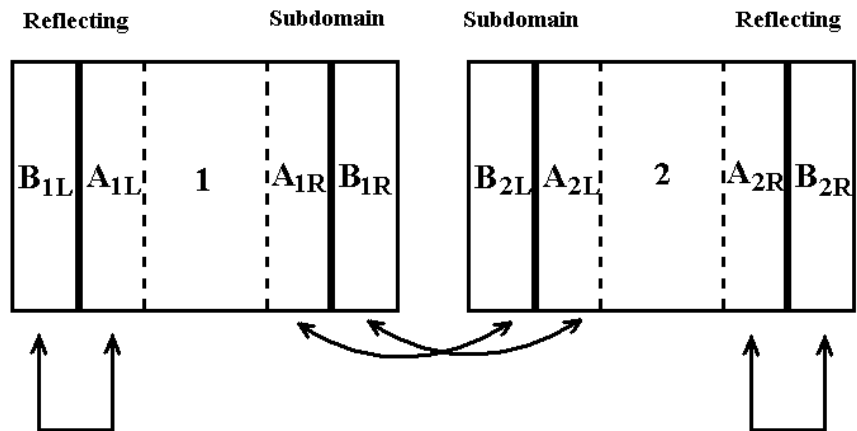


Figure 3.8: Subdomains with reflecting boundaries on the left and right.

## Chapter 4

### Numerical Tests and Computational Results

In this chapter, we conduct several numerical experiments in 2D and 3D to show that the conservative tracking algorithm improves conservation and accuracy. Validity tests demonstrating conservation, consistency and convergence will be presented. Some code debugging techniques will also be discussed.

#### 4.1 Validity Tests and Code Debugging

In this section, we first introduce the techniques for checking the conservation, the consistency and the convergence of the numerical solution, then we discuss some techniques used for code debugging.

##### 4.1.1 Conservation Check

For the two methods (conservative tracking and non-conservative tracking), we compare the relative error in total mass conservation, which is defined

as

$$(\text{final mass} - \text{initial mass} + \text{boundary mass flux}) / (\text{initial mass}), \quad (4.1)$$

with similar definitions for other conserved quantities. First, we add all the quantities at time level  $n$  and  $n + 1$ . If a space-time cell contains some control volumes, the quantities at time level  $n$  are the bottom states and quantities at time level  $n + 1$  are the top states. If a space-time cell does not contain any control volume, *i.e.*, it is away from the interface, we add the quantities at cell center which are the regular interior states. This job is done by function

*g\_volume\_conservation\_check()*.

The boundary flux is calculated when performing the numerical sweeps. If a boundary is periodic boundary or subdomain boundary, we do not need to add the flux because the total flux on these boundaries are zero. If a facet is on any other boundary, after its flux has been calculated, we add it into a global variable. This boundary flux is calculated in the finite difference sweep, because there is no space-time volume near the boundary. In the RT test, the boundaries are periodic except the vertical boundaries, thus we only calculate the flux through the vertical boundaries. By giving the debugging name *check\_conservation* in the input file, after each step, the total boundary flux and the differences of the total quantities will be printed out. By comparing them using a post process program, we can determine if the solution is conserved or not.

### 4.1.2 Consistency Check

The consistency check includes the check of reconstructed space-time interface and the space-time control volumes. This test is performed right after the space-time interface and control volume have been constructed. In the space-time interface and control volume construction routine, after each control volume has been constructed, we perform a consistency check. For every two neighboring control volumes, we check if the adjacent facets for these two control volumes are exactly the same or not. The consistency check of two space-time reconstructed interface pieces in the neighboring space-time cells is checked in the meanwhile. The common boundaries of two neighboring  $N$ -D space-time interface pieces are  $(N - 1)$ -D linear elements (simplices). They are the boundaries of the space-time boundary facets of the control volumes. So, if the boundary facets of two control volumes match, then their boundaries also match.

### 4.1.3 Convergence Tests

We compare the relative  $L_1$  error of numerical solutions, which is defined as

$$\frac{1}{M} \sum_P |U_c(P) - U_r(P)|, \quad (4.2)$$

where  $P$  runs over the coarse grid cell centers,  $M$  is total number of the cell centers on the computational domain,  $U_c(P)$  is the solution on coarse mesh and  $U_r(P)$  is the piecewise linearly interpolated fine grid solution at  $P$ .

We also compare the relative  $L_1$  error of the interface position at early

time, which is defined as

$$\frac{1}{N} \sum_{P \in I_c} \text{dist}(P, I_f) , \quad (4.3)$$

where  $N$  is the number of points on the interface for coarse mesh,  $I_c$  and  $I_f$  are the interfaces for coarse mesh and finer mesh respectively. The reason we do only compare the results at early time is because the interface will not converge after the kink forms at late time.

The convergence tests are done by post processing. The post process first reads the numerical solution or the interface point position from the output data files and then applies the formula to calculate the  $L_1$  errors for the numerical solution or the interface position.

#### 4.1.4 Code Debugging

The most commonly used debugging technique in the conservative front tracking code is the *FronTier* function *debugging()*. By giving a particular debugging name in the input file, more information will be printed in the output file. For example, debugging name *check\_conservation* enables the output of the conservation check. All necessary data will be printed out, such as the total quantities and the boundary fluxes. Table 4.1 gives a list of debugging names used in the conservative front tracking code and their meanings.

In the conservative tracking code, some macros are defined for additional tests. For example, the macro *DEBUG\_STVOLUME* is defined to print out more information if any inconsistency is detected during the space-time control volume construction.

debugging name	meaning
<i>check_conservation</i>	print out information for conservation check
<i>stvolume_construction</i>	to print out space-time control volumes
<i>cs_mem</i>	print out memory allocation information
<i>gp_st_intf</i>	generate gnuplot data file for space-time interface

Table 4.1: Debugging names for conservative front tracking code.

Besides these debugging techniques, some other *FronTier* functions are also used. If a numerical test stops and reports errors, *CLEAN\_UP* information will be printed, such as the name of recent functions entered and left. By reading this information, it is easy to find which function reports the error.

## 4.2 2D Rayleigh-Taylor instability

The 2D numerical experiment was performed on a rectangular  $1 \times 4$  domain. The lower and upper boundaries have reflecting Neumann boundary conditions and periodic conditions for the side boundaries.

The initial configuration consists of an interface separating two polytropic gases (both have polytropic exponent  $\gamma = 1.40$ ). The contact density ratio is 5 : 1. The interface is sinusoidally perturbed with wavelength 1.0 and amplitude 0.1.

Table 4.2 shows the comparison of the conservation errors. It should be pointed out that when bifurcation occurs, the conservation is not exact. This is recorded in the  $40 \times 160$  run at late time ( $t = 5$  and  $t = 6$ ). A conservative treatment of bifurcation is a problem which still remains to be solved. Table 4.4 shows the comparison of the interface position at early time

Conservation Error (Mass)				
	non-conservative tracking		conservative tracking	
time	mesh $20 \times 80$	mesh $40 \times 160$	mesh $20 \times 80$	mesh $40 \times 160$
1	0.065%	0.050%	0.0	0.0
2	0.263%	0.070%	0.0	0.0
3	0.158%	0.110%	0.0	0.0
4	0.507%	0.538%	0.0	0.0
5	0.664%	0.667%	0.0	0.0029%
6	1.532%	1.116%	0.0	0.0012%

Table 4.2: Conservation error for 2D Rayleigh-Taylor simulations.

$L_1$ Error (Mass)				
	non-conservative tracking		conservative tracking	
mesh	$L_1$ error	order	$L_1$ error	order
$10 \times 40$	0.0156403	-	0.0093947	-
$20 \times 80$	0.0197132	-0.33	0.0102038	-0.11
$40 \times 160$	0.0067377	1.54	0.0036500	1.48
$80 \times 320$	0.0040666	0.72	0.0026848	0.44

Table 4.3: Comparison of the  $L_1$  errors. The numerical solution with a finer mesh ( $160 \times 640$ ) is used as the exact solution.

$t = 1$ . Table 4.3 shows the comparison of the  $L_1$  truncation errors. The conservative tracking algorithm gives smaller  $L_1$  errors. Fig. 4.1 shows the interface at time 6.0 by the two methods Fig. 4.2 shows the amplitude plot. In summary, the conservative solution appears to require  $2 \times$  less mesh per linear dimension ( $2^3 \times$  fewer space-time mesh cells) for comparable accuracy.

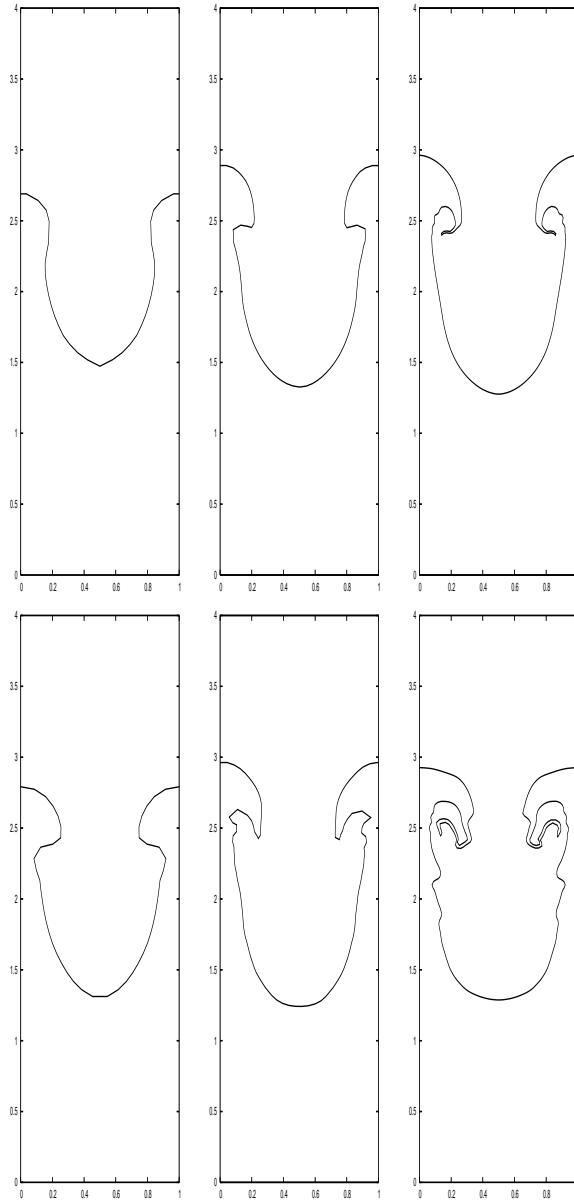


Figure 4.1: Interface plots of 2D Rayleigh-Taylor instability simulations. The upper row shows the results by the non-conservative tracking method at time  $= 6.0$ . The lower row shows the results by the conservative tracking method at the same time. For both rows, from left to right are  $10 \times 40$ ,  $20 \times 80$ , and  $40 \times 160$  grids respectively.

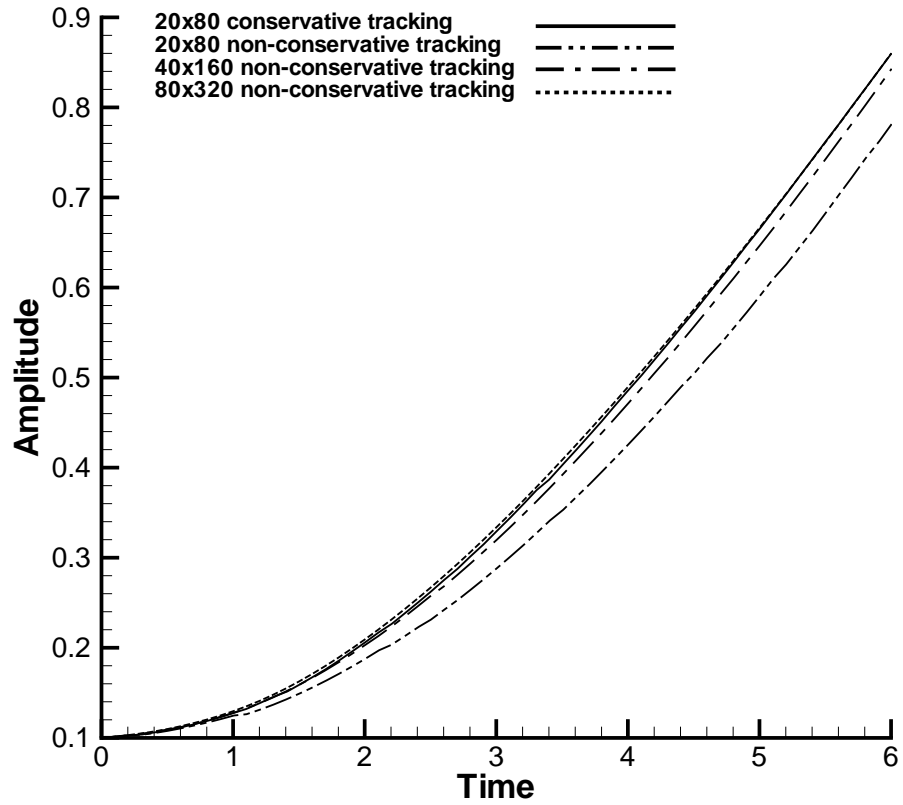


Figure 4.2: Amplitude for 2D Rayleigh-Taylor instability simulations, as a function of time. The amplitude for a coarse mesh by conservative tracking is in approximate agreement with the amplitude by non-conservative tracking for a mesh four times finer in each spatial directions.

Interface Error				
	non-conservative tracking		conservative tracking	
mesh	$L_1$ error	order	$L_1$ error	order
$10 \times 40$	0.000282170090629	-	0.000288084477476	-
$20 \times 80$	0.000319550505349	-0.18	0.000098966932254	1.54
$40 \times 160$	0.000266374694898	0.26	0.000066457271637	0.57
$80 \times 320$	0.000246855366283	0.11	0.000028234852828	1.23

Table 4.4: Comparison of the interface position errors. The interface on a finer mesh ( $160 \times 640$ ) is used as the exact interface.

### 4.3 3D Rayleigh-Taylor instability

The 3D numerical experiment was performed on a rectangular  $1 \times 4$  domain, with a  $10 \times 10 \times 40$  grid with boundary conditions as above.

The initial configuration consists of an interface separating two polytropic gases (both have polytropic exponent  $\gamma = 1.40$ ). The contact density ratio is  $5 : 1$ . The interface is sinusoidally perturbed with wavelength 1.0 and amplitude 0.05. We compared the numerical solutions by non-conservative tracking method and conservative tracking method. Fig. 4.3 shows the interface at time 6.0 by the two methods and Fig. 4.4 shows the amplitude plot. Table 4.2 shows the comparison of the conservation errors.

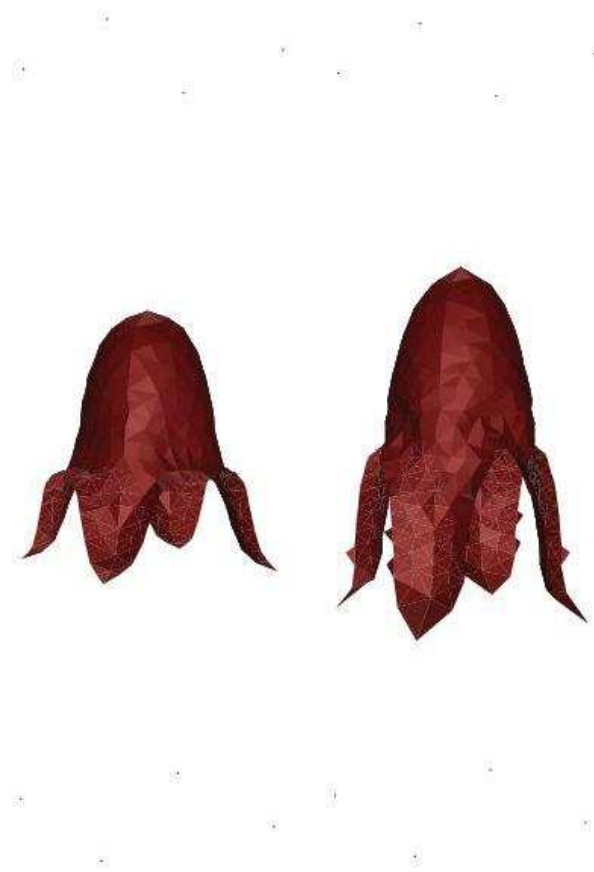


Figure 4.3: Interface plot for 3D Rayleigh-Taylor instability simulations at  $t=6.0$ . Mesh:  $10 \times 10 \times 40$ . Left: non-conservative tracking method; right: conservative tracking method.

Conservation Error (Mass)		
	non-conservative tracking	conservative tracking
time	mesh $10 \times 10 \times 40$	mesh $10 \times 10 \times 40$
1	0.069%	0.0
2	0.216%	0.0
3	0.991%	0.0
4	0.941%	0.0002%
5	2.727%	0.0009%
6	3.856%	0.0013%

Table 4.5: Conservation error for 3D Rayleigh-Taylor simulations.

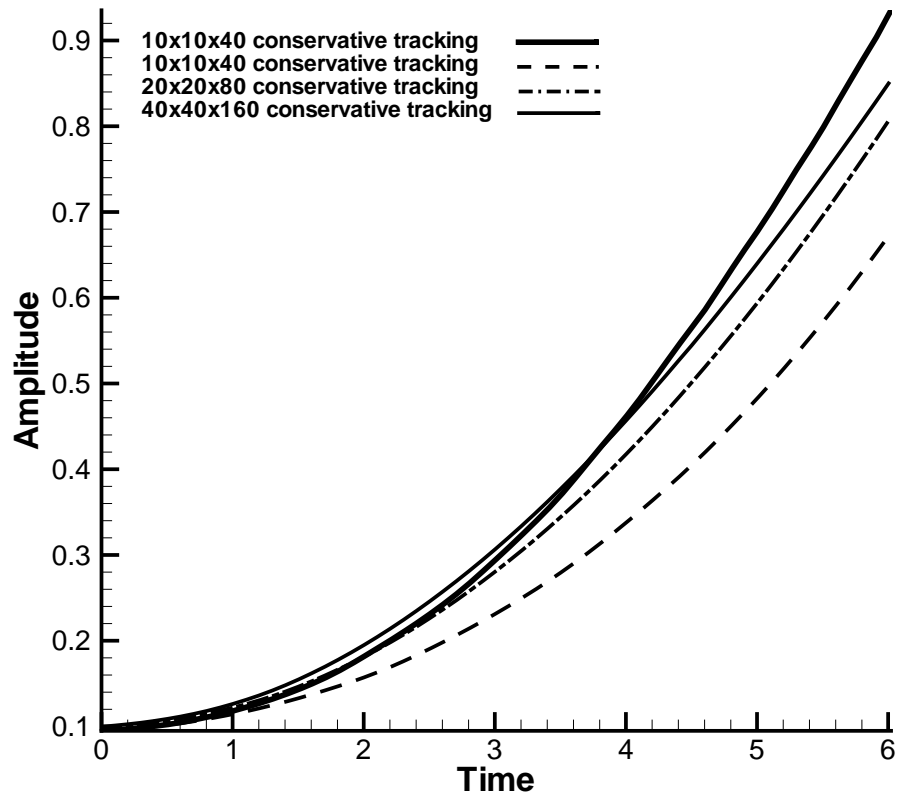


Figure 4.4: Amplitude plot for 3D Rayleigh-Taylor instability simulations

## Chapter 5

### Conclusions and Future Work

We have proposed a fully conservative front tracking algorithm using a grid free (locally grid based in case of bifurcation) with a control volume reconstruction method based on a grid based space-time interface. The algorithm is derived from an integral formulation of the *PDEs*. This algorithm is formulated for the solution of conservation laws in all dimensions. Numerical tests in 2D and 3D are carried out and except for minor effects due to bifurcation, full conservation is achieved. The conservative tracking tends to give more accurate solution while on a coarser grid. Comparable accuracy between the two methods appears to require a  $2\times$  or  $4\times$  less refinement per linear dimension for the conservative algorithm. This algorithm still requires tuning to improve its accuracy and efficiency.

## Bibliography

- [1] P. Bhaniramka, R. Wenge, and R. Crawfis. Isosurfacing in higher dimensions. IEEE Computer Society Press, Salt Lake City, Utah, 2000.
- [2] P. Bhaniramka, R. Wenge, and R. Crawfis. Isosurface construction in any dimension using convex hulls. *IEEE Transactions on Visualization and Computer Graphics*, 10:130–141, 2004.
- [3] F. Buekenhout and M. Parker. The number of nets of the regular convex polytopes in dimension  $\leq 4$ . *Disc. Math.*, 186:69–94, 1998.
- [4] I-L. Chern and P. Colella. A conservative front tracking method for hyperbolic conservation laws. LLNL Rep. No. UCRL-97200, Lawrence Livermore National Laboratory, 1987.
- [5] I-L. Chern, J. Glimm, O. McBryan, B. Plohr, and S. Yaniv. Front tracking for gas dynamics. *J. Comp. Phys.*, 62:83–110, 1986.
- [6] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom. Theory Appl.*, 3:185–212, 1993.
- [7] P. Colella. A direct Eulerian MUSCL scheme for gas dynamics. *SIAM Journal on Computing*, 6(1):104–117, 1985.
- [8] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *J. Comp. Phys.*, 87:171–200, 1990.
- [9] H. S. M. Coxeter. *Introduction to Geometry*. Wiley, New York, 2nd edition, 1969.
- [10] Jian Du, Brian Fix, James Glimm, Xicheng Jia, Xiaolin Li, Yunhua Li, and Lingling Wu. A simple package for front tracking. *J. Comp. Phys.*, 213:613–628, 2006. Stony Brook University preprint SUNYSB-AMS-05-02.

- [11] R. P. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comp. Phys.*, 152:457–492, 1999.
- [12] Ronald P. Fedkiw and Xu-Dong Liu. The ghost fluid method for viscous flows. In *Progress in Numerical Solutions of Partial Differential Equations, France*. Arachon, 1998.
- [13] J. Glimm, M. J. Graham, J. W. Grove, X.-L. Li, T. M. Smith, D. Tan, F. Tangerman, and Q. Zhang. Front tracking in two and three dimensions. *Comput. Math. Appl.*, 35(7):1–11, 1998.
- [14] J. Glimm, J. W. Grove, X. L. Li, W. Oh, and D. H. Sharp. A critical analysis of Rayleigh-Taylor growth rates. *J. Comp. Phys.*, 169:652–677, 2001.
- [15] J. Glimm, J. W. Grove, X.-L. Li, K.-M. Shyue, Q. Zhang, and Y. Zeng. Three dimensional front tracking. *SIAM J. Sci. Comp.*, 19:703–727, 1998.
- [16] J. Glimm, J. W. Grove, X.-L. Li, and D. C. Tan. Robust computational algorithms for dynamic interface tracking in three dimensions. *SIAM J. Sci. Comp.*, 21:2240–2256, 2000.
- [17] J. Glimm, J. W. Grove, X.-L. Li, and N. Zhao. Simple front tracking. In G.-Q. Chen and E. DiBenedetto, editors, *Contemporary Mathematics*, volume 238, pages 133–149. Amer. Math. Soc., Providence, RI, 1999.
- [18] J. Glimm, E. Isaacson, D. Marchesin, and O. McBryan. Front tracking for hyperbolic systems. *Adv. Appl. Math.*, 2:91–119, 1981.
- [19] J. Glimm, X.-L. Li, Y.-J. Liu, Z. L. Xu, and N. Zhao. Conservative front tracking with improved accuracy. *SIAM J. Numerical Analysis*, 41:1926–1947, 2003.
- [20] J. Glimm, D. Marchesin, and O. McBryan. Subgrid resolution of fluid discontinuities II. *J. Comp. Phys.*, 37:336–354, 1980.
- [21] J. Glimm, D. Marchesin, and O. McBryan. A numerical method for two phase flow with an unstable interface. *J. Comp. Phys.*, 39:179–200, 1981.
- [22] J. Glimm and O. McBryan. A computational model for interfaces. *Adv. Appl. Math.*, 6:422–435, 1985.

- [23] E. Godlewski and P. A. Raviart. *Numerical Approximation of Hyperbolic Systems of Conservation Laws*. Springer Verlag, New York, 1991.
- [24] A. Harten. High resolution scheme for hyperbolic conservation laws. *J. Comp. Phys.*, 49:357, 1983.
- [25] A. Harten. ENO schemes with subcell resolution. *J. Comp. Phys.*, 83:148–184, 1989.
- [26] P. Lax and B. Wendroff. Systems of conservation laws. *Comm. Pure Appl. Math.*, 13:217, 1960.
- [27] R. LeVeque. *Numerical Methods for Conservation Laws*. Birkhäuser Verlag, Basel–Boston–Berlin, 1992.
- [28] R. J. LeVeque and K.-M. Shyue. One-dimensional front tracking based on high resolution wave propagation methods. *SIAM Journal on Computing*, 16:348–377, 1995.
- [29] R. J. LeVeque and K.-M. Shyue. 2-dimensional front tracking based on high resolution wave propagation methods. *J. Comp. Phys.*, 123(2):354–368, 1996.
- [30] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.
- [31] D.-K. Mao. A treatment of discontinuities in shock-capturing finite difference methods. *J. Comp. Phys.*, 92:422–455, 1991.
- [32] D.-K. Mao. A treatment of discontinuities for finite difference methods in the two-dimensional case. *J. Comp. Phys.*, 104:377–397, 1993.
- [33] G. Moretti. Thoughts and afterthoughts about shock computations. Rep. No. PIBAL-72-37, Polytechnic Institute of Brooklyn, 1972.
- [34] G. Moretti. Computations of flows with shocks. *Ann. Rev. Fluid Mech.*, 19:313–337, 1987.
- [35] 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, 1972.

- [36] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi equations. *Jour. Comp. Phys*, 79:12–49, 1988.
- [37] R. Pember, John Bell, Phillip Colella, William Cruchfield, and Michael Welcome. An adaptive cartesian grid method for unsteady compressible flow in irregular regions. *J. Computational Phys.*, 120:278–304, 1995.
- [38] R. Richtmyer and K. Morton. *Difference Methods for Initial Value Problems*. Interscience, New York, second edition, 1967.