Two-Dimensional Anisotropic Cartesian Mesh Adaptation for the Compressible Euler Equations

by

W. Andrew Keats

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2004

Abstract

Simulating transient compressible flows involving shock waves presents challenges to the CFD practitioner in terms of the mesh quality required to resolve discontinuities and prevent smearing. This thesis discusses a novel two-dimensional Cartesian anisotropic mesh adaptation technique implemented for compressible flow. This technique, developed by Ham, Lien and Strong in [6], is efficient because it refines and coarsens cells using criteria that consider the solution in each of the cardinal directions separately. Originally designed and tested for laminar flow simulations, in this thesis the method will be applied to compressible flow. The procedure shows promise in its ability to deliver good quality solutions while achieving computational savings.

The convection scheme used is the Advective Upstream Splitting Method (Plus) [14], and the refinement/coarsening criteria are based on work done by Ham et al. Transient shock wave diffraction over a backward step and shock reflection over a forward step are considered as test cases because they demonstrate that the quality of the solution can be maintained as the mesh is refined and coarsened in time. The data structure is explained in relation to the computational mesh, and the object-oriented design and implementation of the code is presented. Refinement and coarsening algorithms and their effects on the solution are outlined. The features of the adaptation technique found to have the greatest influence on the quality of the solution are a) the arrangement of cells of different aspect ratios, b) the application of the mesh refinement and coarsening at appropriate intervals throughout the simulation, and c) the refinement criterion used. Computational savings over uniform and isotropic mesh approaches are shown to be significant.

Acknowledgements

Professor Fue-Sang Lien's special style and attitude toward research has inspired me since we first met, and I thank him for accepting me as his student and friend. I am looking forward to studying under his supervision for another three years.

Professors George A. Davidson and Roydon A. Fraser deserve special mention for being among the first to wade through this thesis and provide valuable feedback.

My fellow researchers Johan and Masoud have afforded me with valuable advice and guidance throughout the past two years, for which I am most grateful.

My family have provided me with moral and financial support during this endeavour and were always there for me.

Finally, I feel privileged to acknowledge my fiancée Sonia Singh for her unyielding love and support.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Objectives and Scope . . . 3
- 1.3 Thesis Structure . . . 3

2 The Euler Equations **4**

3 Numerical Methods for the Euler Equations **7**
- 3.1 Finite Volume Discretization . . . 7
- 3.2 Boundary Conditions . . . 9
  - 3.2.1 Walls . . . 9
  - 3.2.2 Inflow and Outflow . . . 9
- 3.3 Survey of Numerical Schemes . . . 10
  - 3.3.1 Flux via Finite Differences . . . 10
  - 3.3.2 Flux Difference Splitting . . . 11
  - 3.3.3 Flux Vector Splitting . . . 11
- 3.4 First-Order AUSM$^+$ Scheme . . . 13
- 3.5 Higher-Order Extension . . . 16
  - 3.5.1 Spatial Extension . . . 17
  - 3.5.2 Gradient Limiters . . . 17
  - 3.5.3 Temporal Extension . . . 19

4 Test Cases **20**
- 4.1 Backward Step . . . 20
  - 4.1.1 Setup of Computational Problem . . . 21
  - 4.1.2 Existing Experimental and Computational Results . . . 22
- 4.2 Forward Step . . . 25
  - 4.2.1 Setup of Computational Problem . . . 26
  - 4.2.2 Existing Computational Results . . . 26

5 Results for Uniform Cartesian Mesh **30**
- 5.1 Backward Step . . . 30
  - 5.1.1 First-Order Results . . . 31
  - 5.1.2 Higher-Order Results . . . 31
  - 5.1.3 Analysis of Results . . . 32
- 5.2 Forward Step . . . 33

5.2.1 First-Order Results 33
5.2.2 Higher-Order Results 33
5.2.3 Analysis of Results 34

6 Cartesian Mesh Adaptation **37**
6.1 Survey of Adaptation Techniques 37
6.1.1 Isotropic 38
6.1.2 Anisotropic 39
6.2 Cartesian Anisotropic Adaptation for Compressible Flow 39
6.2.1 Data Structure 39
6.2.2 Refinement and Coarsening Algorithms 43
6.2.3 Application of the Algorithm to Compressible Flow Simulations 44
6.2.4 Adaptation Criteria 47
6.2.5 Solution Reconstruction via the Gradient 48

7 Results for Anisotropically Adapted Mesh **51**
7.1 Backward Step 51
7.1.1 Analysis of Results 53
7.2 Forward Step 55
7.2.1 Analysis of Results 56

8 Concluding Remarks **61**
8.1 Computational Resource Savings 61
8.2 Future Direction 64

A Derivations **65**
A.1 Least-Squares Gradient Computation 65
A.2 Optimal Cell Size Criterion 66

List of Figures

1.1 Curved shock wave in front of a cylinder . . . 2

3.1 A sample Cartesian grid cell . . . 7
3.2 Wave propagation across boundaries . . . 9
3.3 A sample cell in a uniform mesh . . . 10
3.4 A sample cell in a two-dimensional mesh . . . 14
3.5 Cell-centred ϕ extrapolated to cell faces . . . 17
3.6 Face numbering for gradient limiter . . . 18

4.1 Backward step geometry . . . 21
4.2 Structure of the perturbed region behind a diffracting shock wave . . . 23
4.3 Schlieren image of a refracting shock wave, $M_s = 1.3$ . . . 24
4.4 Schlieren image of a refracting shock wave, $M_s = 2.4$ . . . 24
4.5 Schlieren image of a refracting shock wave, $M_s = 5.09$ . . . 25
4.6 False Schlieren image of a refracting shock wave, $M_s = 5.09$ . . . 25
4.7 Forward step geometry . . . 27
4.8 Backward step: first-order Godunov scheme . . . 27
4.9 Backward step: Second-order MUSCL scheme . . . 28
4.10 Forward step: PPM results . . . 29

5.1 Mach 5.09 shock diffraction, first-order results . . . 31
5.2 Mach 1.3 shock diffraction, higher-order results . . . 32
5.3 Mach 2.4 shock diffraction, higher-order results . . . 33
5.4 Mach 5.09 shock diffraction, higher-order results . . . 34
5.5 Mach 3.0 flow over a forward step, first-order results . . . 35
5.6 Mach 3.0 flow over a forward step, higher-order results . . . 36

6.1 Two Cartesian mesh adaptation techniques . . . 38
6.2 Tree data structure representation of physical mesh . . . 38
6.3 Classes of objects found in the computational mesh . . . 39
6.4 Cell indexing system used in the present code . . . 40
6.5 Example of irregular coarsening . . . 40
6.6 Class aggregation diagram . . . 42
6.7 Cell refinement procedure . . . 42
6.8 Cell coarsening procedure . . . 43
6.9 Gradient extrapolation . . . 49
6.10 Schematic of gradient limiter application . . . 50

7.1 Mach 1.3 shock diffraction, higher-order results 52
7.2 Mach 2.4 shock diffraction, higher-order results 53
7.3 Mach 5.09 shock diffraction, higher-order results 54
7.4 Mach 1.3 shock diffraction, high-resolution results 55
7.5 Mach 2.4 shock diffraction, high-resolution results 56
7.6 Mach 5.09 shock diffraction, high-resolution results 57
7.7 Mach 3.0 flow over a forward step: computational mesh 58
7.8 Mach 3.0 flow over a forward step: density 59
7.9 Mach 3.0 flow over a forward step: pressure 60

8.1 Memory and CPU time savings for the forward step test case 62
8.2 Memory savings over isotropic refinement 63

List of Tables

4.1 Forward step test case: initial conditions 22

6.1 Adaptive code: class definitions 41
6.2 Adaptive code: variable definitions 41

8.1 Allocation of processor time 63

Nomenclature

t	Time
x	Horizontal coordinate direction
y	Vertical coordinate direction

Refinement algorithm-related symbols

(i, j)	`Cell` location indices
(l_i, l_j)	`Cell` refinement level indices
τ	Error tolerance
$\xrightarrow{L}, \xrightarrow{R}$	Pointer to the left- or right-hand `Cell` relative to the `Face`
l	`Face` length

Matrix quantities

$\mathbf{A}$	Jacobian of flux vector: $\partial \mathbf{f} / \partial \mathbf{u}$
$\mathbf{\Lambda}$	Diagonal matrix of eigenvalues of $\mathbf{A}$
$\mathbf{Q}$	Matrix of eigenvectors of $\mathbf{A}$

Scalar quantities

a	Speed of sound
ρ	Density
p	Pressure
u	$x-$component of velocity
v	$y-$component of velocity
e_T	Total internal energy
ϕ	Any flow variable; pressure, density, etc.
M	Mach number

m	Upwinded Mach number
$\mathcal{M}^{\pm}$	Mach splitting function
$\mathcal{P}^{\pm}$	Pressure splitting function
γ	Ratio of specific heats c_p/c_v of the gas ($\gamma_{\text{air}} = 1.4$)
Δt	Length of time between updates of $\mathbf{u}$
Δx	Horizontal cell dimension
Δy	Vertical cell dimension
λ	Maximum wave speed *or* Lagrange multiplier

Vector quantities

$\mathbf{u}$	Vector of conserved variables
$\mathbf{f},\ \mathbf{g}$	Vector of {horizontal, vertical} flux variables
$\mathcal{F}(\mathbf{u})$	Sum of fluxes through all cell faces
$\vec{u}$	Velocity vector
$\mathbf{x}$	Vector of coordinate directions
$\mathbf{e}_i$	Unit normal vector in the i^{th} coordinate direction
$\mathbf{s}$	Vector from cell-centre to another point in the cell

Chapter 1

Introduction

This thesis focuses on the use of the anisotropic Cartesian mesh adaptation technique to improve the resolution of numerical flow simulations governed by the two-dimensional Euler equations. In particular, flows with strong transient shock waves are chosen to test the ability of the refinement algorithm to handle moving discontinuities.

1.1 Background

Governing Equations

The Euler equations provide a model for the inviscid compressible flow of a homogeneous fluid in subsonic and supersonic regimes [12]. They can be used to numerically predict the locations and shapes of shock waves that occur in regions of the flow where the effects of viscosity are negligible.

Shock Waves

Physically, a shock wave is a thin region of flow (thickness of only a few mean free molecular paths) that separates two regions of differing stagnation pressure and density. Because pressure signals travel through a flow field at the speed of sound, some physical mechanism must be present to modify the supersonic flow direction in the presence of obstacles (or changes in density and pressure); this mechanism is the shock wave (demonstrated in figure 1.1).

Shock waves can be visualized experimentally using methods such as Schlieren photography, which uses the index of refraction of the fluid to provide an image of the exponentially-weighted density gradient.

Numerically, a shock wave is represented by a discontinuity in the flow field variables $\{\rho, p, \vec{u}\}$; these discontinuities arise in numerical solutions of the Euler equations for certain geometries and flow parameters. For numerical solution schemes that incorporate gradient information $\{\nabla\rho, \nabla p, \nabla u_i\}$ to improve their order of accuracy, these discontinuities (and others) cause numerical difficulties such as negative density and pressure. These difficulties arise because the numerical reconstruction of the solution gradient does not take the discontinuity into account and thus may use data points

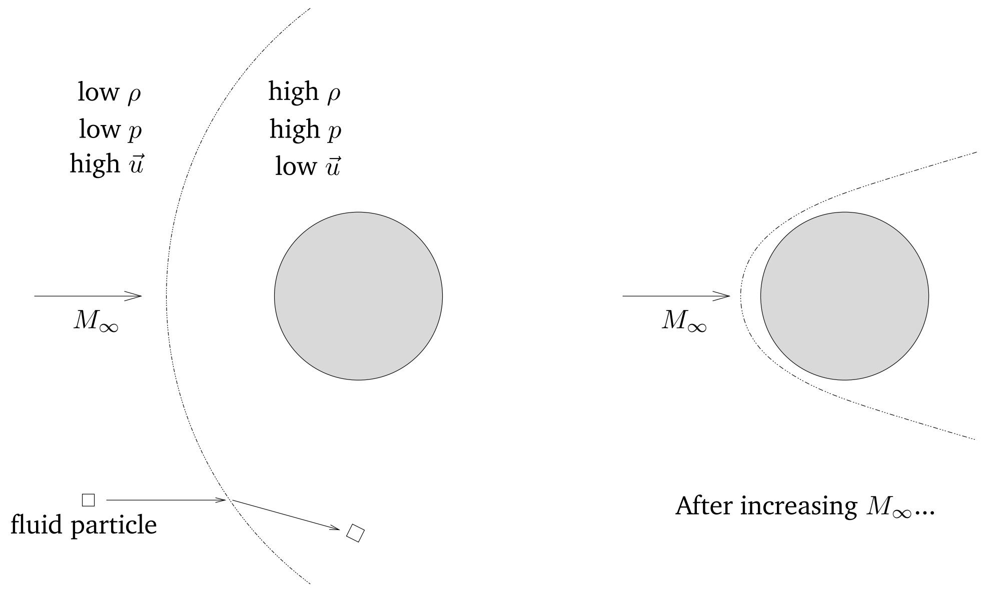


Figure 1.1: Curved shock wave in front of a cylinder placed in a supersonic flow

that lie across a shock wave; of course, mathematically, gradients are either non-unique or invalid at discontinuous points. Gradient reconstructions that identify discontinuities and select their data accordingly have been developed [20] but are computationally very expensive when applied to transient flows.

Mesh Adaptation

Qualitatively speaking, the governing equations are solved over a mesh consisting of a multitude of tiny adjacent finite volumes (otherwise known as *cells*). Each cell maintains corresponding $\{\rho, p, \vec{u}\}$ values, usually located at the centroid of the cell, and the overall solution consists of the field of $\{\rho, p, \vec{u}\}$ encompassed by the mesh. Thus, the resolution of the numerical solution is limited by the distance between adjacent finite volumes.

Mesh adaptation serves to selectively modify the mesh layout so that in regions of the flow where high resolution is required to discern flow features, more cells are added. Likewise, in smooth regions of the flow, cells are removed in order to increase the computational efficiency. The total number of cells in the computational mesh does not necessarily need to remain constant, as long as it does not grow beyond the memory capacity of the computer. *Isotropic* mesh adaptation refines areas of the mesh by subdividing cells into cells of equal aspect ratio. *Anisotropic* mesh adaptation does not necessarily preserve aspect ratio while subdividing and joining cells. The differences between anisotropic and isotropic mesh adaptation are covered in greater detail in chapter 6.

As a quick aside, mesh adaptation is not the only way to improve solution accuracy; one can use an higher-order numerical scheme to obtain a better numerical estimate of the solution. This approach works well until discontinuous regions appear in the flow, at which point the high-order scheme must be reduced to a lower-order one (because there are less valid mesh points to choose from). For example, a fourth-order accurate scheme requires information from four adjacent cells in each dimension in order to estimate the cell-centred value at the next time step. For those cells adjacent to or straddling a discontinuity, different (or less) data must be used, compromising the solution quality at those points.

1.2 Objectives and Scope

This thesis aims to show the following:

1. The anisotropic mesh adaptation technique can be successfully applied to transient compressible flows involving shock waves. Success is measured by comparing the adapted-mesh numerical solution to previously validated experimental and computational results.
2. The anisotropic adaptation technique instead of a uniform mesh leads to significant savings with respect to the following computational resources: processing time and computer memory. Furthermore, memory savings are improved when compared to *equivalent*[1] isotropically refined meshes.

The following limitations apply to the scope of the thesis:

1. All test cases considered are two-dimensional, and the mesh refinement and coarsening algorithms were implemented for a two-dimensional mesh. Three-dimensional equivalents have been implemented for incompressible flow by Ham et al. [6] but have yet to be applied to compressible flow.
2. All outflow boundary conditions are supersonic; subsonic outflow boundary conditions introduce numerical difficulties whose mitigation is complex and not necessarily solvable via mesh refinement. See [10] for a discussion of these difficulties.
3. An ideal gas assumption is made; this precludes test cases involving hypersonic or other chemically reacting flows.

1.3 Thesis Structure

Chapters 2 to 4 provide the reader with the information required to interpret the uniform mesh results presented in chapter 5. These results are provided in order to validate the code implementation of the basic numerical method and to serve as benchmark data against which the adapted mesh solution can be compared.

The transient mesh adaptation algorithms and refinement criteria are presented in chapter 6. Although the method is based on work done by Ham et al. [6], the material presented here is significantly different; the method required modification to be appropriate for compressible flow involving shock waves. To the best of the author's knowledge, the anisotropic Cartesian refinement technique has not yet been applied to *unsteady* compressible flows in any of the literature. Finally, results obtained on adapted meshes are presented and discussed.

[1] *Equivalent* implies that the refinement criterion remains the same.

Chapter 2

The Euler Equations

The Euler equations consist of a formulation of the continuity, momentum and energy equations for inviscid flow. In two space dimensions, they are

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} = 0, \tag{2.1}$$

where

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_T \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ (\rho e_T + p)u \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ (\rho e_T + p)v \end{bmatrix}.$$

In one space dimension, they are

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} = 0, \tag{2.2}$$

where

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho e_T \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ (\rho e_T + p)u \end{bmatrix}, \quad .$$

The variable $\mathbf{u}$ is the vector of conserved variables; mass, momentum and energy, all per unit volume. The pressure p is obtained using an equation of state for ideal gases:

$$p = (\gamma - 1)\left(\rho e_T - \frac{1}{2}\rho(u^2 + v^2)\right). \tag{2.3}$$

The variables $\mathbf{f}$ and $\mathbf{g}$ represent horizontal and vertical mass, momentum and energy fluxes, plus their respective pressure contributions. It is important to note that $\mathbf{f} = \mathbf{f}(\mathbf{u})$ and $\mathbf{g} = \mathbf{g}(\mathbf{u})$; this allows us to construct and analyze flux Jacobian matrices.

The equations (2.1) have two important mathematical properties: hyperbolicity and homogeneity. These properties and their implications are easiest to demonstrate using the one-dimensional Euler equations.

Hyperbolicity

Consider the Jacobian matrix of the flux $\mathbf{f}$:

$$\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \tag{2.4}$$

The system of equations (2.2) is hyperbolic *if and only if* $\mathbf{A}$ is diagonalizable [9]; that is, *iff* $\mathbf{A}$ can be written as

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}, \tag{2.5}$$

where $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$ is a diagonal matrix of the eigenvalues of $\mathbf{A}$.

For the one dimensional Euler equations, we have:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{\gamma-3}{2}u^2 & (3-\gamma)u & \gamma-1 \\ -\gamma u e_T + (\gamma-1)u^3 & \gamma e_T - \frac{3\gamma-3}{2}u^2 & \gamma u \end{bmatrix}, \tag{2.6}$$

$$\mathbf{Q} = \begin{bmatrix} 1 & \frac{\rho}{2a} & -\frac{\rho}{2a} \\ u & \frac{\rho}{2a}(u+a) & -\frac{\rho}{2a}(u-a) \\ \frac{u^2}{2} & \frac{\rho}{2a}\left(\frac{u^2}{2} + \frac{a^2}{\gamma-1} + au\right) & -\frac{\rho}{2a}\left(\frac{u^2}{2} + \frac{a^2}{\gamma-1} - au\right) \end{bmatrix}, \tag{2.7}$$

$$\mathbf{Q}^{-1} = \left(\frac{\gamma-1}{\rho a}\right) \begin{bmatrix} \frac{\rho}{a}\left(\frac{-u^2}{2} + \frac{a^2}{\gamma-1}\right) & \frac{\rho}{a}u & -\frac{\rho}{a} \\ \frac{-u^2}{2} - \frac{au}{\gamma-1} & -u + \frac{a}{\gamma-1} & 1 \\ -\frac{-u^2}{2} - \frac{au}{\gamma-1} & u + \frac{a}{\gamma-1} & -1 \end{bmatrix}, \tag{2.8}$$

$$\mathbf{\Lambda} = \begin{bmatrix} u & 0 & 0 \\ 0 & u+a & 0 \\ 0 & 0 & u-a \end{bmatrix}. \tag{2.9}$$

Notice that the eigenvalues of $\mathbf{A}$ represent the speeds at which information in the form of *waves* (or *characteristics*) propagate in a one-dimensional gasdynamic situation. A parcel of fluid with certain properties moving at speed u accounts for the first eigenvalue; pressure waves travel at the speed of sound away from the parcel in both directions at speeds $u + a$ and $u - a$.

Upwinded finite volume schemes developed for the one-dimensional equations can be directly applied to higher dimensions by considering flux components normal to cell faces; by doing so, they *implicitly* consider waves travelling in directions normal to those cell faces. However, in a real multidimensional situation, waves 'transmitted' by a fluid particle travel in all possible directions away from that fluid particle, similar to the way in which circular waves result when one drops a stone into a pool of water. Therefore, by decomposing waves into those which travel in face-normal directions, a simplification is made.

Homogeneity

The system (2.2) is *homogeneous* because it satisfies

$$\mathbf{f}(\mathbf{u}) = \mathbf{A}(\mathbf{u})\mathbf{u}. \tag{2.10}$$

This property allows for an efficient splitting of the flux vector into positive- and negative-going fluxes via the Jacobian matrix $\mathbf{A}$:

$$\mathbf{f}^{+} = \mathbf{A}^{+}\mathbf{u}\,, \quad \mathbf{f}^{-} = \mathbf{A}^{-}\mathbf{u} \tag{2.11}$$

where

$$\mathbf{A}^{+} = \mathbf{Q}\mathbf{\Lambda}^{+}\mathbf{Q}^{-1}\,, \quad \mathbf{A}^{-} = \mathbf{Q}\mathbf{\Lambda}^{-}\mathbf{Q}^{-1} \tag{2.12}$$

and $\mathbf{\Lambda}^{\pm}$ describe positive- and negative-going waves. This approach allows the flux vector to be effectively upwinded in space, based on wave speeds and directions [29]. The numerical implementation of this upwinding is discussed in detail in section 3.3.

Chapter 3

Numerical Methods for the Euler Equations

3.1 Finite Volume Discretization

In this thesis we consider unstructured two-dimensional anisotropic Cartesian grid cells, such as the one shown in Figure 3.1. The conserved variables $\mathbf{u}$ are stored at the cell centres, and the face fluxes, stored at the faces, are indexed by cardinal direction {N, S, E, W} and position (denoted by subscript {0, 1}).

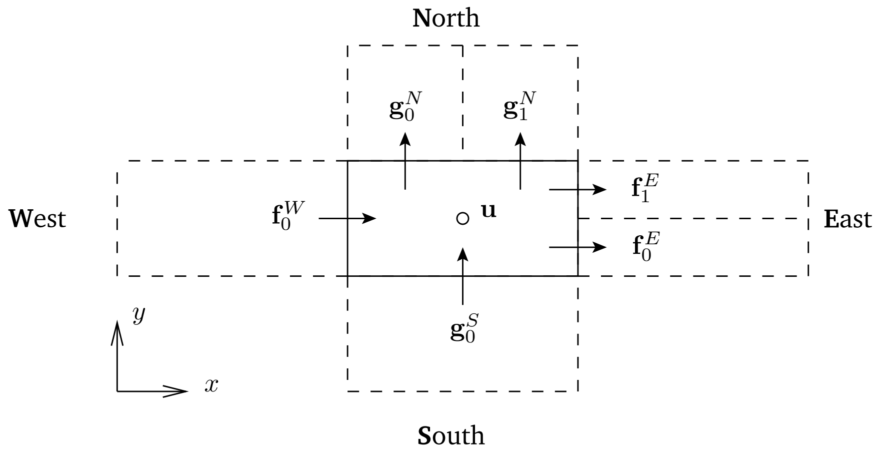


Figure 3.1: A sample Cartesian grid cell with multiple faces and neighbours

Consider the integration of the Euler equations (2.1) over a control volume with area A and perimeter C;

$$\int_A [\mathbf{u}(\mathbf{x}, t^{n+1}) - \mathbf{u}(\mathbf{x}, t^n)]\, dA = -\int_{t^n}^{t^{n+1}} \oint_C [\mathbf{f}\cdot\mathbf{e}_x + \mathbf{g}\cdot\mathbf{e}_y]\, dl\, dt\,, \tag{3.1}$$

where dl is an infinitesimal length of the perimeter, and $\mathbf{e}_x$, $\mathbf{e}_y$ are the $x-$ and $y-$components of the vector normal to the face of the control volume.

In the general case of a Cartesian grid cell with a maximum of two faces[1] in each of the cardinal directions, we obtain the discretized equation:

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t\, \underbrace{\frac{1}{\Delta x \Delta y}\left(\sum_{i=0}^{1}(l_{\text{face}}\mathbf{f})_i^E - \sum_{i=0}^{1}(l_{\text{face}}\mathbf{f})_i^W + \sum_{i=0}^{1}(l_{\text{face}}\mathbf{g})_i^N - \sum_{i=0}^{1}(l_{\text{face}}\mathbf{g})_i^S\right)}_{\text{flux } \mathcal{F}(\mathbf{u})} \tag{3.2}$$

where Δx and Δy are the width and height of the cell, respectively, and l_{face} refers to the length of the face in position i. In the more specialized case of a uniform, structured Cartesian grid, equation (3.2) reduces to

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \frac{\Delta t}{\Delta x}(\mathbf{f}_0^E - \mathbf{f}_0^W) - \frac{\Delta t}{\Delta y}(\mathbf{g}_0^N - \mathbf{g}_0^S)\,. \tag{3.3}$$

Equation 3.3 is first-order accurate in time; higher-order extension is discussed in section 3.5.3.

Courant-Friedrichs-Lewy Criterion

Equation 3.3 is explicit, meaning that values at the next time step depend only on values from the previous time step. Thus, the flow solver is relatively simple, needing only to visit each control volume once per time step and update $\mathbf{u}$ based on the interface fluxes. However, for the solution to remain stable, the solver must satisfy the *Courant-Friedrichs-Lewy* (CFL) condition:

> *The full numerical domain of dependence must contain the physical domain of dependence [9].*

This is equivalent to saying that the time step must be small enough so that the fastest wave speed, $\lambda = u + |a|$, cannot cross the entire length of a cell in the computational domain. If pressure waves are able to propagate beyond the length of a cell within a time step, the solution will diverge in time.

For explicit one-dimensional codes, this condition implies:

$$CFL = \lambda\frac{\Delta t}{\Delta x} < 1.0. \tag{3.4}$$

However, in practise, for two and higher dimensional codes, the practical upper limit on the CFL number for maintaining stability lies somewhere below 1.0. In two dimensions, the fastest wave speed is chosen using:

$$\lambda = max(u + |a|, v + |a|). \tag{3.5}$$

[1]This limit is used in the adaptation method discussed in section 6.2

3.2 Boundary Conditions

3.2.1 Walls

Any cell face may be specified as a *wall* boundary condition, in which case the flux through the face is modified by setting the normal component of the velocity to zero. For horizontal wall-faces, $v = 0$; for vertical walls, $u = 0$. This leaves only a pressure term in the momentum flux; for simplicity, this term is set equal to the cell-centred pressure, so that:

$$p(x_{i+1/2}, y, t) \approx p(x_i, y, t^n) + O(\Delta x) + O(t - t^n) \tag{3.6}$$

is a constant extrapolation of pressure that is first-order accurate in space [9]. Linear and higher-order extrapolation may be used instead.

3.2.2 Inflow and Outflow

Specifying inflow and outflow boundary conditions requires knowledge of the Mach number of the flow. Recall from chapter 2 that the eigenvalues λ_i of $\mathbf{A}$ determine the direction and speed of propagation of wave information, and consider an inflow/outflow boundary that is perpendicular to the direction of flow, as in Figure 3.2. For a *supersonic inflow*, all waves $(u - a, u, u + a)$ enter the domain. Therefore, in order to completely specify the properties of a supersonic inflow boundary condition, all flow variables must be specified at the inlet. For a *subsonic inflow*, one wave travels out of the domain, and attempting to define it will cause the problem to be over-specified. The remaining characteristics may be specified using a combination of either ρ and p or ρ, u, and v.

Outflow specification follows a similar approach; for a *supersonic outflow*, no flow variables need to be specified at the exit; face fluxes can simply be determined from the cell-centred conserved variables $\mathbf{u}$. For a *subsonic outflow*, one wave travels back into the domain, and must be specified correctly. Defining subsonic outflow boundary conditions to be free of acoustic reflections and other spurious phenomena is one of the numerical banes of compressible flow research; in this thesis, no test cases have subsonic outflow boundaries.

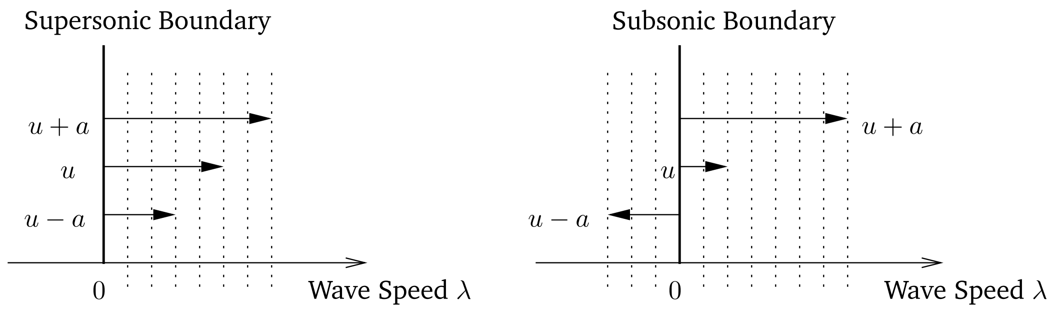


Figure 3.2: Wave propagation across boundaries

3.3 Survey of Numerical Schemes

A multitude of finite-difference, finite-volume and finite-element schemes have been proposed for solving the Euler equations. In order to provide the reader with a representative background of the available methods, this section outlines the three fundamental *finite-volume* paradigms and a few of their associated numerical schemes. For simplicity, we assume a one-dimensional uniform mesh with cells indexed by i in the $x-$direction as in Figure 3.3.

Although not all of the schemes discussed here are derived from finite-volume origins, they can be phrased in the *conservation form* of equation (3.3) by defining $\mathbf{f}_0^W$ and $\mathbf{f}_0^E$ appropriately.

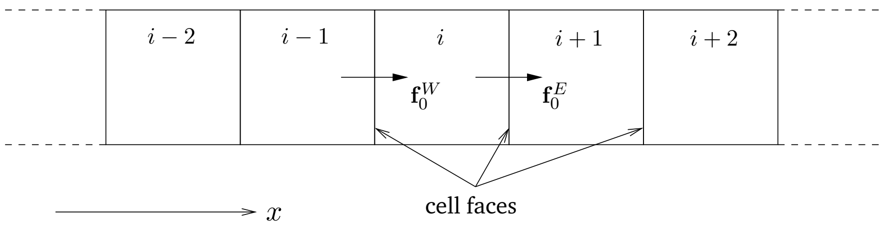


Figure 3.3: A sample cell i in a uniform mesh surrounded by neighbours

3.3.1 Flux via Finite Differences

As the title suggests, the flux at each cell interface is calculated using a given finite-difference approach. These methods do not necessarily use flow information in the discretization (for example, upwinding), and tend to use central-difference and/or predictor-corrector formulations.

MacCormack's Method

This is a *predictor-corrector* method that comes in two versions:

$$\begin{aligned}\tilde{\mathbf{u}}_i &= \mathbf{u}_i^n - \frac{\Delta t}{\Delta x}(\mathbf{f}(\mathbf{u}_{i+1}^n) - \mathbf{f}(\mathbf{u}_i^n))\\ \mathbf{u}_i^{n+1} &= \frac{1}{2}(\mathbf{u}_i^n + \tilde{\mathbf{u}}_i) - \frac{\Delta t}{2\Delta x}(\mathbf{f}(\tilde{\mathbf{u}}_i) - \mathbf{f}(\tilde{\mathbf{u}}_{i-1}))\end{aligned} \tag{3.7}$$

$$\begin{aligned}\tilde{\mathbf{u}}_i &= \mathbf{u}_i^n - \frac{\Delta t}{\Delta x}(\mathbf{f}(\mathbf{u}_i^n) - \mathbf{f}(\mathbf{u}_{i-1}^n))\\ \mathbf{u}_i^{n+1} &= \frac{1}{2}(\mathbf{u}_i^n + \tilde{\mathbf{u}}_i) - \frac{\Delta t}{2\Delta x}(\mathbf{f}(\tilde{\mathbf{u}}_{i+1}) - \mathbf{f}(\tilde{\mathbf{u}}_i))\end{aligned} \tag{3.8}$$

Version (3.7) is better-suited to capturing left-running waves, while (3.8) is better-suited to capturing right-running waves [9]. The two versions can be expressed in conservation form by defining:

$$\begin{aligned}\mathbf{f}_0^E &= \frac{1}{2}\left(\mathbf{f}(\mathbf{u}_{i+1}^n) + \mathbf{f}(\tilde{\mathbf{u}}_i)\right)\\ \mathbf{f}_0^W &= \frac{1}{2}\left(\mathbf{f}(\mathbf{u}_i^n) + \mathbf{f}(\tilde{\mathbf{u}}_{i-1})\right)\end{aligned} \tag{3.9}$$

$$
\begin{aligned}
\mathbf{f}_0^E &= \frac{1}{2}\left(\mathbf{f}(\tilde{\mathbf{u}}_{i+1}) + \mathbf{f}(\mathbf{u}_i^n)\right) \\
\mathbf{f}_0^W &= \frac{1}{2}\left(\mathbf{f}(\tilde{\mathbf{u}}_{i}) + \mathbf{f}(\mathbf{u}_{i-1}^n)\right)
\end{aligned}
\tag{3.10}
$$

which correspond to versions (3.7) and (3.8) respectively.

3.3.2 Flux Difference Splitting

This technique provides a basis for numerically upwinding the Euler equations (or any set of hyperbolic equations, for that matter [12]) based on wave speed and direction. Since, in subsonic flow, waves can travel in opposite directions, upwinding based on only one variable such as fluid velocity (as is commonly used for incompressible flow calculations [32]) does not provide numerical stability.

Recall from chapter 2 that the Jacobian $\mathbf{A}$ satisfies homogeneity and may be split into $\mathbf{A}^\pm$ which acknowledge contributions from right-running and left-running waves by defining $\mathbf{\Lambda}$ appropriately:

$$
\begin{aligned}
\mathbf{\Lambda}^+ &= diag(max(\lambda_i, 0)) \\
\mathbf{\Lambda}^- &= diag(min(\lambda_i, 0))
\end{aligned}
\tag{3.11}
$$

Also consider that the *secant-line* approximation of the *flux difference* across a cell face is:

$$
\mathbf{f}(\mathbf{u}_{i+1}) - \mathbf{f}(\mathbf{u}_i) = \mathbf{A}_{i+1/2}(\mathbf{u}_{i+1} - \mathbf{u}_i) \tag{3.12}
$$

where $\mathbf{A}_{i+1/2}$ is some choice of average of the Jacobian between cells i and $i+1$. Using equation (3.11) to split $\mathbf{A}$, the approximation becomes:

$$
\mathbf{f}(\mathbf{u}_{i+1}) - \mathbf{f}(\mathbf{u}_i) = (\mathbf{A}^+_{i+1/2} + \mathbf{A}^-_{i+1/2})(\mathbf{u}_{i+1} - \mathbf{u}_i) \tag{3.13}
$$

with $\mathbf{A}^\pm$ defined in equation (2.12). Using the fact that contributions related to $\mathbf{A}^+_{i+1/2}$ only affect cell $i+1$ and that contributions related to $\mathbf{A}^-_{i+1/2}$ only affect cell i, then with reference to Figure 3.3, the numerical flux $\mathbf{f}_0^E$ at the interface between cells i and $i+1$ can be expressed using any of the following:

$$
\begin{aligned}
\mathbf{f}_0^E &= \mathbf{f}(\tilde{\mathbf{u}}_i) + \mathbf{A}^-_{i+1/2}(\mathbf{u}_{i+1} - \mathbf{u}_i) \\
\mathbf{f}_0^E &= \mathbf{f}(\tilde{\mathbf{u}}_{i+1}) - \mathbf{A}^+_{i+1/2}(\mathbf{u}_{i+1} - \mathbf{u}_i)
\end{aligned}
\tag{3.14}
$$

The key to constructing a flux difference splitting method lies in the definition of $\mathbf{A}_{i+1/2}$ and its associated splitting $\mathbf{A}^\pm_{i+1/2}$.

3.3.3 Flux Vector Splitting

Rather than splitting the Jacobian matrix $\mathbf{A}$ and computing fluxes based on the ensuing matrix-vector products (equation (3.14)), the flux vector is split directly into right-going and left-going fluxes $\mathbf{f}^+$ and $\mathbf{f}^-$. Since the flux vector must remain consistent,

$$
\mathbf{f}(\mathbf{u}) = \mathbf{f}^+(\mathbf{u}) + \mathbf{f}^-(\mathbf{u}) \ , \tag{3.15}
$$

all flux *vector* splitting methods can be expressed as flux *difference* splitting methods because of the additional requirement placed on the split flux vector:

$$\mathbf{f}^{\pm} = \mathbf{A}^{\pm}\mathbf{u} \, . \tag{3.16}$$

This claim can be verified by using equations (3.15) and (3.16) in (3.13).

Using equation (3.15) in cells i and $i+1$, we obtain the following expressions:

$$\begin{aligned} \mathbf{f}(\mathbf{u}_i) &= \mathbf{f}^{+}(\mathbf{u}_i) + \mathbf{f}^{-}(\mathbf{u}_i) \\ \mathbf{f}(\mathbf{u}_{i+1}) &= \mathbf{f}^{+}(\mathbf{u}_{i+1}) + \mathbf{f}^{-}(\mathbf{u}_{i+1}) \end{aligned} \tag{3.17}$$

which, when substituted into the expression for the interface flux (3.14), yield:

$$\mathbf{f}_0^E = \mathbf{f}^{+}(\mathbf{u}_i) + \mathbf{f}^{-}(\mathbf{u}_{i+1}) \tag{3.18}$$

which is computationally much simpler than its flux difference splitting counterpart.

Steger-Warming Splitting

In [24], Steger and Warming split the wave speeds λ_i using

$$\lambda_i^{+} = \frac{\lambda_i + |\lambda_i|}{2} \, , \quad \lambda_i^{-} = \frac{\lambda_i - |\lambda_i|}{2} \, , \tag{3.19}$$

which are used to constitute the matrices $\mathbf{\Lambda}^{\pm}$. Using equations (2.11) and (2.12), they obtained expressions for the split flux vectors, $\mathbf{f}^{\pm}$, which clearly depend on the wave speeds $\lambda_i^{\pm}$:

$$\begin{aligned} \lambda_1^{+} &= \frac{u+|u|}{2} \, , & \lambda_1^{-} &= \frac{u-|u|}{2} \\ \lambda_2^{+} &= \frac{u+a+|u+a|}{2} \, , & \lambda_2^{-} &= \frac{u+a-|u+a|}{2} \\ \lambda_3^{+} &= \frac{u-a+|u-a|}{2} \, , & \lambda_3^{-} &= \frac{u-a-|u-a|}{2} \end{aligned} \tag{3.20}$$

The split flux vectors are defined by:

$$\mathbf{f}^{\pm} = \frac{\gamma-1}{\gamma}\rho\lambda_1^{\pm} \begin{bmatrix} 1 \\ u \\ \frac{1}{2}u^2 \end{bmatrix} + \frac{\rho}{2\gamma}\lambda_2^{\pm} \begin{bmatrix} 1 \\ u+a \\ \frac{u^2}{2} + \frac{a^2}{\gamma-1} + au \end{bmatrix} + \frac{\rho}{2\gamma}\lambda_3^{\pm} \begin{bmatrix} 1 \\ u-a \\ \frac{u^2}{2} + \frac{a^2}{\gamma-1} - au \end{bmatrix} \tag{3.21}$$

Obviously, if the flow is supersonic, then either $\mathbf{f}^{-}$ or $\mathbf{f}^{+}$ must be zero. For subsonic flow, both $\mathbf{f}^{-}$ and $\mathbf{f}^{+}$ contribute to the interface flux. One problem with the Steger-Warming flux vector splitting is the fact that $\mathbf{f}^{\pm} \notin C^1(\lambda_i = 0)$; the derivatives of the split flux vectors are discontinuous when λ_i change sign. This creates problems in the solution at sonic points [17].

In two space dimensions, the Jacobians $\partial\mathbf{f}/\partial x$ and $\partial\mathbf{g}/\partial y$ must be diagonalized as in chapter 2 in order to find the corresponding $\mathbf{f}^{\pm}$ and $\mathbf{g}^{\pm}$.

Finally, it is important to note that although the Steger-Warming splitting may share the same wave speed splitting as a flux difference splitting method, the proportions of the upwinded flux components will not necessarily be the same, because of the specialized definition of $\mathbf{A}_{i+1/2}$ required.

Van Leer Splitting

In order to overcome the problem of discontinuous flux derivatives, van Leer rewrote the flux vector in terms of the Mach number,

$$\mathbf{f} = \begin{bmatrix} \rho a M \\ \frac{\rho a^2}{\gamma}(\gamma M^2 + 1) \\ \rho a^3 M \left(\frac{1}{2}M^2 + \frac{1}{\gamma-1}\right) \end{bmatrix} , \tag{3.22}$$

and individually split the three flux components according to the following constraints:

1. $\mathbf{f}^{\pm} = \mathbf{f} \quad \text{at} \quad M = \pm 1$
2. $d\mathbf{f}^{\pm}/dM = 0 \quad \text{at} \quad M = \mp 1$
3. $d\mathbf{f}^{\pm}/dM = d\mathbf{f}/dM \quad \text{at} \quad M = \pm 1$
4. $\mathbf{f} = \mathbf{f}^{+} + \mathbf{f}^{-}$
5. $\mathbf{f}^{+}(\mathbf{u}) = -\mathbf{f}^{-}(-\mathbf{u})$

Using the dependence of the flux components on M [9], he went on to develop a continuously differentiable flux splitting [17] by splitting M into quadratic functions for subsonic flow. For $|M| < 1$, the split flux vectors are defined by:

$$\mathbf{f}^{\pm} = \pm\frac{\rho a}{4}(M \pm 1)^2 \begin{bmatrix} 1 \\ \frac{2a}{\gamma}\left(\frac{\gamma-1}{2}M \pm 1\right) \\ \frac{2a^2}{\gamma^2-1}\left(\frac{\gamma-1}{2}M \pm 1\right)^2 \end{bmatrix} \tag{3.23}$$

For $M > 1$, $\mathbf{f}^{+} = \mathbf{f}$, and for $M < 1$, $\mathbf{f}^{-} = \mathbf{f}$, where $\mathbf{f}$ is defined in equation (3.22). It is important to note that no pressure terms are present in this formulation of $\mathbf{f}^{\pm}$; they are implicitly split via the γ, ρ and a terms. If the flux vector were to be formulated using a pressure term in the momentum component, then the *split pressure* $P^{\pm}$ would need to be defined as follows:

$$P^{\pm}(M) = p \begin{cases} \frac{1}{2}(1 \pm sign(M)) & |M| \geq 1, \\ \mathcal{P}^{\pm}(M) & otherwise \end{cases} \tag{3.24}$$

where

$$\mathcal{P}^{\pm} = \frac{1}{4}(M \pm 1)^2(2 \mp M) \; . \tag{3.25}$$

This relation can be derived using the constraints listed above.

3.4 First-Order AUSM^{+} Scheme

The AUSM^{+} scheme of Liou [14] is an extension of the original AUSM (Advection Upstream Splitting Method) scheme of Liou and Steffen [16] in which the flux vector, divided into convective and pressure components, is split based on Mach number. The division, illustrated in equation (3.26), occurs in the momentum component of the flux, but not in the energy component.

$$
\mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 \\ \rho uv \\ (\rho e_T + p)u \end{bmatrix} + \begin{bmatrix} 0 \\ p \\ 0 \\ 0 \end{bmatrix} \equiv \mathbf{f}^{(c)} + \mathbf{f}^{(p)}
$$

$$
\mathbf{g} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 \\ (\rho e_T + p)v \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ p \\ 0 \end{bmatrix} \equiv \mathbf{g}^{(c)} + \mathbf{g}^{(p)} \tag{3.26}
$$

This numerical scheme was used throughout the research for the following reasons:

1. It is a more recent flux vector splitting that overcomes the disadvantages of the alternatives discussed earlier, yet it has already been proven reliable for a range of different flow regimes (see, for example, [18])

2. It is simple to program, compared to flux difference splitting approaches.

3. The order of accuracy of the scheme was found by Ripley [22] to be 1.81 on a uniform Cartesian mesh.

For the sake of completeness, the formulation of the scheme will be discussed as a two-dimensional problem on a uniform Cartesian mesh, as shown in Figure 3.4.

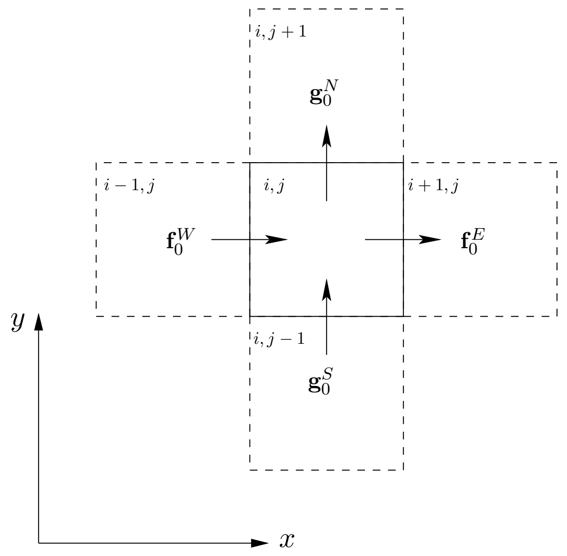


Figure 3.4: A sample cell (i, j) in the computational domain surrounded by neighbours

Liou extended van Leer's requirement of a continuously differentiable flux splitting by introducing Mach and pressure splitting polynomial functions $\mathcal{M}^{\pm}$, $\mathcal{P}^{\pm}$ which are twice continuously differentiable at the points $M = 0, \pm 1$. With these additional constraints, the orders of the polynomial splitting functions become 4 and 5 respectively. Unlike van Leer's scheme, however, the

AUSM$^+$ scheme does not split each component of the flux vector individually; rather, the Mach and pressure splitting functions ensure continuity and consistency.

Using the indexed computational domain of Figure 3.4, the split fluxes at the East and North interfaces are defined as:

$$(\mathbf{f}_0^E)^{\pm} = \frac{1}{2}(m_x^E \pm |m_x^E|)\begin{bmatrix}\rho a\\ \rho u a\\ \rho v a\\ (\rho e_T + p)a\end{bmatrix} + \begin{bmatrix}0\\ \mathcal{P}^{\pm}(M_x)\\ 0\\ 0\end{bmatrix} \tag{3.27}$$

$$(\mathbf{g}_0^N)^{\pm} = \frac{1}{2}(m_y^N \pm |m_y^N|)\begin{bmatrix}\rho a\\ \rho u a\\ \rho v a\\ (\rho e_T + p)a\end{bmatrix} + \begin{bmatrix}0\\ 0\\ \mathcal{P}^{\pm}(M_y)\\ 0\end{bmatrix} \tag{3.28}$$

where

$$\begin{aligned} m_x^E &= \mathcal{M}^+(M_x)_i + \mathcal{M}^-(M_x)_{i+1} \\ m_y^N &= \mathcal{M}^+(M_y)_j + \mathcal{M}^-(M_y)_{j+1} \end{aligned} \tag{3.29}$$

The horizontal and vertical Mach numbers M_x and M_y are defined as u/a and v/a respectively. The split fluxes $(\mathbf{f}_0^W)^{\pm}$ and $(\mathbf{g}_0^S)^{\pm}$ are defined similarly. The *interface speed of sound*, a, is defined (at each interface between cells) using an average of the states in each neighbouring cell. For simplicity, this average can be defined using either $a = \sqrt{a_1 a_2}$, or $a = 0.5(a_1 + a_2)$, where subscripts $\{1, 2\}$ refer to the state on either side of the face. In [14], Liou elaborates on another method of finding the intermediate speed of sound that allows the AUSM$^+$ scheme to exactly locate a stationary shock discontinuity; this technique uses the Prandtl relation in conjunction with the speed of sound based on total enthalpy.

It is important to note that the Mach number is 'double-split'; that is, fluxes are split based on Mach numbers from both the left and right (and upper and lower) cells. The functions $\mathcal{M}^{\pm}$ and $\mathcal{P}^{\pm}$ are fourth and fifth-order polynomials that conform to a set of criteria similar to those used by van Leer. For completeness, they are reproduced from [14] below:

1. The split Mach numbers $\mathcal{M}^{\pm}$ hold the following properties:
   (a) $\mathcal{M}^+(M) + \mathcal{M}^-(M) = M,$ *for consistency.*
   (b) $\mathcal{M}^+(M) \geq 0$ *and* $\mathcal{M}^-(M) \leq 0.$
   (c) $\mathcal{M}^{\pm}$ *are monotone increasing functions of* M.
   (d) $\mathcal{M}^+(M) = -\mathcal{M}^-(-M),$ *i.e., a symmetry property.*
   (e) $\mathcal{M}^+(M) = M$ *as* $M \geq 1$; $\mathcal{M}^-(M) = M$ *as* $M \leq -1.$
   (f) $\mathcal{M}^{\pm}$ *are continuously differentiable.*
2. The split pressures $\mathcal{P}^{\pm}$ hold the following properties:
   (a) $\mathcal{P}^+(M) + \mathcal{P}^-(M) = 1,$ *for consistency.*
   (b) $\mathcal{P}^{\pm}(M) \geq 0$ *as required by the physical constraint that pressure be nonnegative.*

(c) $d\mathcal{P}^+/dM \geq 0$ *and* $d\mathcal{P}^-/dM \leq 0$.

(d) $\mathcal{P}^+(M) = \mathcal{P}^-(-M)$.

(e) $\mathcal{P}^+(M) = 1$ *as* $M > 1$; $\mathcal{P}^-(M) = 1$ *as* $M < -1$.

(f) $\mathcal{M}^\pm$ *are continuously differentiable.*

The split $\mathcal{M}^\pm$ depend on the cell Mach numbers in the following way:

$$\mathcal{M}^\pm(M) = \begin{cases} \frac{1}{2}(M \pm |M|) & |M| \geq 1, \\ \mathcal{M}^\pm_\beta(M) & \textit{otherwise} \end{cases} \tag{3.30}$$

where

$$\mathcal{M}^\pm_\beta(M) = \pm\frac{1}{4}(M \pm 1)^2 \pm \beta(M^2-1)^2, \quad \frac{-1}{16} \leq \beta \leq \frac{1}{2} \tag{3.31}$$

The split pressures $\mathcal{P}^\pm$ are also split based on Mach number:

$$\mathcal{P}^\pm(M) = p\begin{cases} \frac{1}{2}(1 \pm sign(M)) & |M| \geq 1, \\ \mathcal{P}^\pm_\alpha(M) & \textit{otherwise} \end{cases} \tag{3.32}$$

where

$$\mathcal{P}^\pm_\alpha(M) = \frac{1}{4}(M \pm 1)^2(2 \mp M) \pm \alpha M(M^2-1)^2, \quad \frac{-3}{4} \leq \alpha \leq \frac{3}{16} \tag{3.33}$$

In the case of the AUSM$^+$ scheme, additional requirements determine α and β:

$$\begin{aligned} \frac{d^2\mathcal{M}^\pm_\beta}{dM^2}(M=0) = 0, &\quad \Rightarrow \beta = \frac{1}{8} \\ \frac{d^2\mathcal{P}^\pm_\alpha}{dM^2}(M=\pm 1) = 0, &\quad \Rightarrow \alpha = \frac{3}{16}. \end{aligned} \tag{3.34}$$

Note that when α and β are both zero, the splitting functions revert to those chosen by van Leer for the subsonic mass flux.

3.5 Higher-Order Extension

Spatial and temporal extension of the numerical method to second– and higher-order accuracy provides a way of superlinearly improving the accuracy of the solution as cell sizes and time steps are reduced.

In all of the numerical methods discussed previously, cell data are assumed piecewise-constant; the flux at the face is generated by a cell-centred conserved variable $\mathbf{u}$. This lack of variation in $\mathbf{u}$ introduces dissipative errors into the solution (see [10] for a discussion of dissipative and dispersive errors) which result in the unwanted smearing of flow features such as shock and contact discontinuities.

3.5.1 Spatial Extension

In order for a finite-volume numerical method to achieve second-order accuracy, gradient information must be used in the calculation of the fluxes at cell interfaces. In van Leer's MUSCL (Monotone Upstream-centred Scheme for Conservation Laws) approach, piecewise-constant cell-centred data is modified by higher-order components of the Taylor series approximation to $\mathbf{u}$ at the cell faces [29]. Specifically, for a second-order accurate solution, the cell-centred $\mathbf{u}$ must be extrapolated to the cell faces using $\nabla \mathbf{u}$ in order to calculate $\mathbf{f}^{\pm}$ and $\mathbf{g}^{\pm}$. This extension to the numerical method is vital when using irregular meshes, since first-order finite differences lose accuracy when grid spacing is nonuniform. Higher-order accurate solutions are possible using, for example, quadratic approximations of $\mathbf{u}$ at the cell face. The PPM (Piecewise Parabolic Method) scheme of Woodward and Colella [33] uses this approach, as seen in section 4.2.2.

Figure 3.5 demonstrates a typical part of an anisotropic Cartesian mesh, in which cell-centred ϕ values (individual components of $\mathbf{u}$) are linearly extrapolated to face centres via the gradient, $\nabla \phi$.

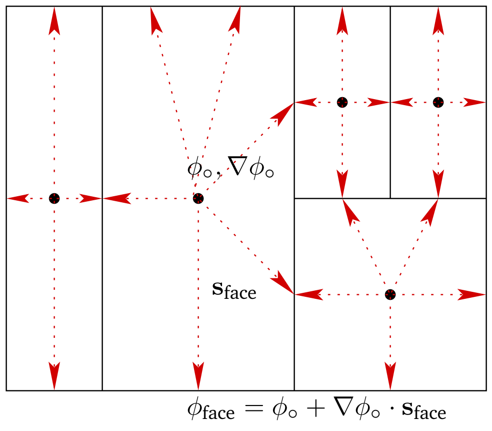

$\phi_\circ, \nabla\phi_\circ$
$\mathbf{s}_{\text{face}}$
$\phi_{\text{face}} = \phi_\circ + \nabla\phi_\circ \cdot \mathbf{s}_{\text{face}}$

Figure 3.5: Cell-centred ϕ are extrapolated to cell faces

3.5.2 Gradient Limiters

First-order numerical schemes are highly dissipative; second and higher-order are much less so. Gradients calculated based on information across flow discontinuities, such as shock and contact waves, are invalid, and may lead to the catastrophic destabilization of the solution. For example, fluxes calculated based on unrealistic gradient information may yield negative pressures and densities at the next time step.

In order to overcome these problems while maintaining second-order accuracy throughout the majority of the domain, the solution gradients must be modified, or *limited* in the presence of discontinuities and other sharp flow features. In theory, the best approach is simply to locate the discontinuous regions of the flow and to apply gradient information selectively there; techniques for accomplishing this exist [20] but are computationally very intensive; they require the inversion of at least a 5×5 matrix for each cell at each time step. In practise, it is easiest to use non-oscillatory gradient information in order to calculate the cell-centred gradient. In the present code, this calculation is performed using either the *minmod* limiter or the *superbee* limiter [29].

The minmod limiter captures non-oscillatory gradient information by taking the minimum modulus of the face-centred gradients and applying it to the cell centre. In other words, if any two gradients are of different sign, the cell-centred gradient is set to zero. Otherwise, the gradient with the minimum absolute value is used. The limiter function itself is:

$$\psi_{minmod}(r) = \begin{cases} 0, & r \leq 0, \\ r, & 0 \leq r \leq 1, \\ 1, & r \geq 1, \end{cases} \tag{3.35}$$

where r is the ratio of the i^{th} component of two interface gradients:

$$r = \frac{\nabla \phi_1 \cdot \mathbf{e}_i}{\nabla \phi_2 \cdot \mathbf{e}_i} \tag{3.36}$$

The subscript $\{1, 2\}$ denote cell interfaces as shown in Figure 3.6.

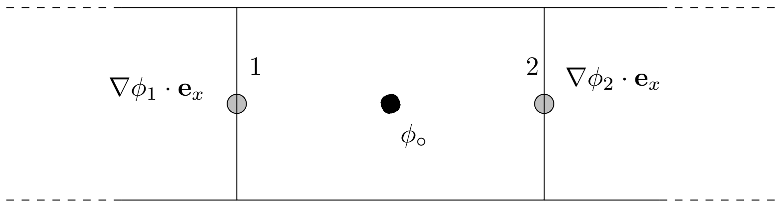

$\nabla \phi_1 \cdot \mathbf{e}_x$
1
$\phi_\circ$
2
$\nabla \phi_2 \cdot \mathbf{e}_x$

Figure 3.6: Face numbering for gradient limiter application

When more than two interfaces are present, the minmod function is applied repeatedly. The superbee limiter is less strict in that it does not necessarily take the gradient with the lowest absolute value:

$$\psi_{superbee}(r) = \begin{cases} 0, & r \leq 0, \\ 2r, & 0 \leq r \leq \frac{1}{2}, \\ 1, & \frac{1}{2} \leq r \leq 1, \\ r, & 1 \leq r \leq 2, \\ 2, & r \geq 2. \end{cases} \tag{3.37}$$

Using either of these limiters, the cell-centred gradient is simply:

$$\nabla \phi_\circ \cdot \mathbf{e}_i = \nabla \phi_2 \cdot \mathbf{e}_i \, \psi(r) \tag{3.38}$$

The use of limiters to smooth gradient information does not guarantee the stability of the solution in two or more dimensions. However, it provides great improvement over unlimited solutions, most of which do not converge (they result in negative pressure and density) at high Mach numbers.

3.5.3 Temporal Extension

In this work, a formally second-order accurate three-stage Runge-Kutta time-stepping scheme was adopted for the following reasons:

1. The first-order explicit Euler technique outlined at the beginning of this chapter is highly sensitive to CFL number when combined with the AUSM$^+$ scheme. Using the 3-stage Runge-Kutta technique allows improved stability at CFL numbers approaching unity.

2. For high Mach numbered test cases (Mach 2 and above), additional dissipation is needed to maintain positivity of density and pressure. By using a combination of first– and second-order spatial accuracy through different time steps, the solution can approach second-order accuracy while remaining stable. The specific choice of spatial order of accuracy is outlined in chapter 5.

This technique, outlined in [10], is reproduced below:

$$\begin{aligned}
\mathbf{u}^{(*)} &= \mathbf{u}^n - \Delta t \mathcal{F}(\mathbf{u}^n) \\
\mathbf{u}^{(**)} &= \frac{1}{2}[\mathbf{u}^n + \mathbf{u}^{(*)}] - \frac{\Delta t}{2}\mathcal{F}(\mathbf{u}^{(*)}) \\
\mathbf{u}^{n+1} &= \frac{1}{2}[\mathbf{u}^n + \mathbf{u}^{(**)}] - \frac{\Delta t}{2}\mathcal{F}(\mathbf{u}^{(**)}) \qquad (3.39)
\end{aligned}$$

where $\mathcal{F}(\mathbf{u})$ is the summation of the fluxes through all cell faces, as in equation 3.2.

Chapter 4

Test Cases

In order to test the effectiveness of the mesh adaptation algorithm, two different geometric configurations were used: the backward-facing and forward-facing steps. These two cases were chosen for the following reasons:

- Both qualitative and quantitative computational and experimental results are readily available.
- The geometry is simple, so the adaptation algorithm can be used independently of cut-cell techniques, which are needed to handle non-rectangular geometry.
- Both cases are transient and therefore test the ability of the mesh to preserve the solution quality while refining and coarsening in time.

The goal of this chapter is to survey the literature related to the chosen test cases and determine useful benchmarks against which the present code can be compared.

4.1 Backward Step

Shock wave diffraction over a backward step is an important test case for validating numerical codes that are designed to solve the Euler equations. First and foremost, experimental results are available for a variety of geometries and Mach numbers (See Skews [23], Schardin [30], Bazhenova et al. [2], and Takayama and Inoue [28] for experimental results). Secondly, the problem has been studied extensively from a computational and mathematical point of view. Lastly, a diffracting shock wave induces secondary physical phenomena in the perturbed region behind the shock [23]; namely, a secondary shock wave, vortex, slipstream, terminator, and an incident sound wave. The resolution of these phenomena using a given numerical code provides insight into its strengths and weaknesses.

4.1.1 Setup of Computational Problem

The backward step is constructed from three equally-sized square cells which are subsequently refined until the desired mesh resolution is achieved. The geometry and boundary conditions are shown in Figure 4.1. No measurements are present in the figure because the cell size is arbitrary; the flow is self-similar for any cell size.

Initially, the fluid states α and β are separated by a single discontinuity at the edge of the step, a shock wave that moves to the right.

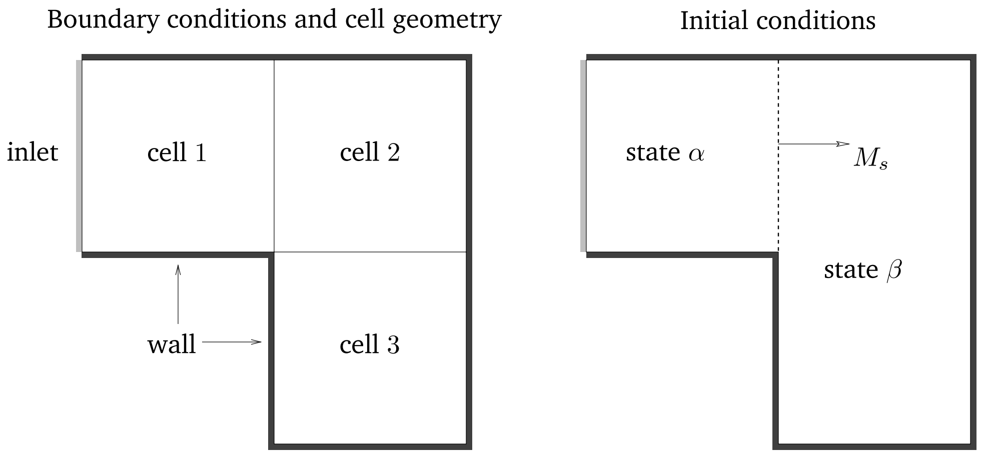


Figure 4.1: Backward step geometry

For the backward step geometry, three shock Mach numbers $M_s = \{1.3, 2.4, 5.09\}$ are investigated in order to verify the position and quality of the secondary phenomena. Each M_s is generated using different pressure and density ratios, and is defined as:

$$M_s = \frac{W}{a_\beta} \tag{4.1}$$

where W is the shock wave speed relative to a fixed frame of reference, and a_β is the speed of sound in the gas in state β.

As the shock wave moves to the right, a velocity is induced in the gas immediately to the left. This induced speed, u_s can be calculated along with the initial pressure and density ratios using the following relations (derived in [1]):

$$\frac{p_\alpha}{p_\beta} = 1 + \frac{2\gamma}{\gamma+1}(M_s^2 - 1) \tag{4.2}$$

$$\frac{\rho_\alpha}{\rho_\beta} = \frac{1 + \frac{\gamma+1}{\gamma-1}\left(\frac{p_\alpha}{p_\beta}\right)}{\frac{\gamma+1}{\gamma-1} + \frac{p_\alpha}{p_\beta}} \tag{4.3}$$

$$u_s = \frac{a_\beta}{\gamma}\left(\frac{p_\alpha}{p_\beta} - 1\right)\sqrt{\frac{\frac{2\gamma}{\gamma+1}}{\frac{p_\alpha}{p_\beta} + \frac{\gamma-1}{\gamma+1}}} \tag{4.4}$$

The initial velocity on the right hand side of the shock is set to zero. The x-component of the initial velocity in state α is set to u_s, and the y-component is set to zero, since the shock travels in the x-direction only. Table 4.1 summarizes the pre- and post-shock fluid initial conditions for the three M_s, determined using equations 4.2, 4.3 and 4.4.

	M_s		
	1.3	2.4	5.09
p_α/p_β	1.805	6.553	30.06
ρ_α/ρ_β	1.516	3.212	5.029
$u_s\ (m/s)$	151.7	432.9	624.1

Table 4.1: Pre- and post-shock fluid initial conditions

4.1.2 Existing Experimental and Computational Results

Experimental Results

The experimental study of diffracting shock waves has revealed a complicated flow structure in the perturbed region behind them. Skews [23] performed experiments for a variety of Mach numbers and convex corner angles, and has outlined the structure of the perturbed region; qualitatively, this structure is shown in Figure 4.2. Skews also determined experimentally and tabulated the following correlations (for several corner angles):

- The slipstream angle variation with the shock Mach number M_s.
- The terminator angle variation with M_s.
- The relationship between M_s and the velocity of the secondary shock.
- The contact surface velocity variation with M_s.
- The variation of the vortex angle and velocity with M_s.

The flow structures shown in Figure 4.2 can be described as follows:

Incident shock: Diffracts in a similar way to a sound wave; its radius of curvature is approximately u_0t.

Reflected sound wave: Propagates upstream and marks the start of the curvature of the incident shock.

Slipstream: Due to separation, it separates high-velocity gas on the upper side from almost stationary gas on the lower side. It represents the outermost characteristic of the Prandtl-Meyer expansion fan.

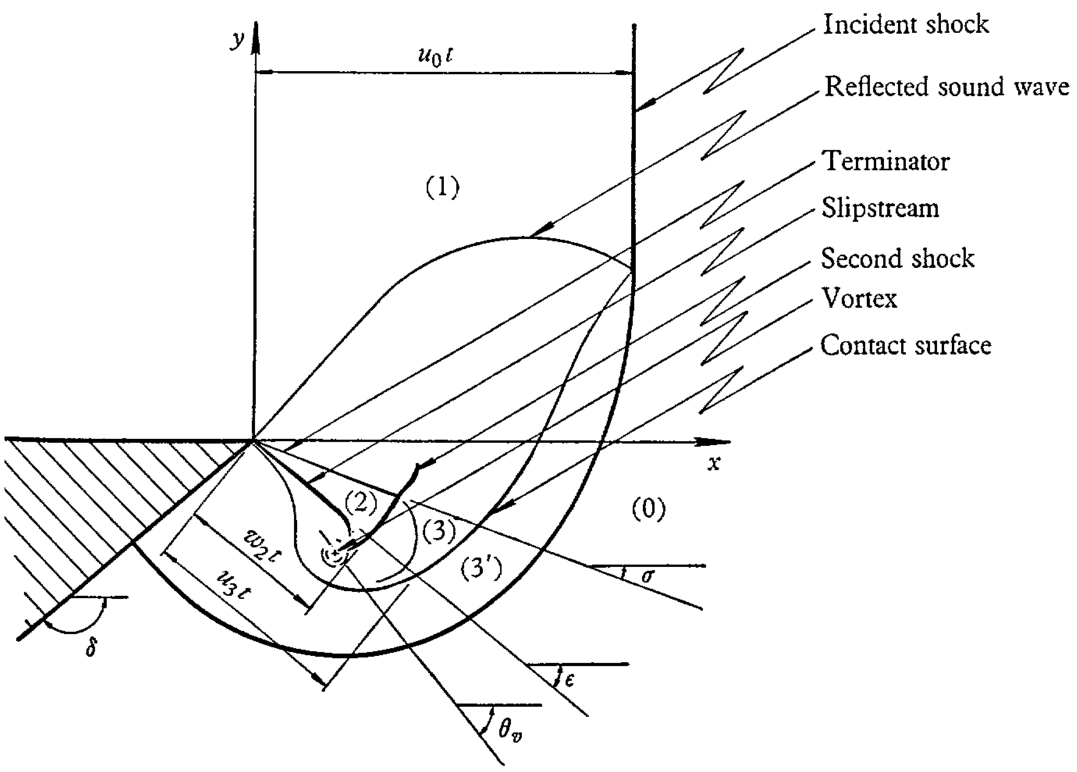


Figure 4.2: Structure of the perturbed region behind a diffracting shock wave (from Skews [23])

Terminator: The first characteristic of the Prandtl-Meyer expansion; the angle separating the terminator from the horizontal increases with rising M_s.

Second shock: The region between the slipstream and the terminator is a uniform flow region parallel to the slipstream [23], and the second shock is a normal shock wave caused when the flow in this region exceeds Mach 1.0 [27].

Vortex: Located just below the slipstream, its location is well defined for $M_s < 1.5$. The angle between the vortex and the slipstream decreases as M_s increases.

Contact surface: Originates at the intersection point of the reflected sound wave and the incident shock, but is highly diffuse in this region. It becomes better-defined as it nears the region containing the rest of the flow structures.

Schardin and Bazhenova et al. produced Schlieren images of density gradients for flows over 90 degree convex edges at Mach numbers of 1.3, 2.4, and 5.09. These are shown in Figures 4.3, 4.4, and 4.5.

Computational Results

Figure 4.6, generated by Quirk [21], is a false Schlieren image of computational results for a diffracting shock wave with $M_s = 5.09$. The resolution of the simulation is such that there are 560

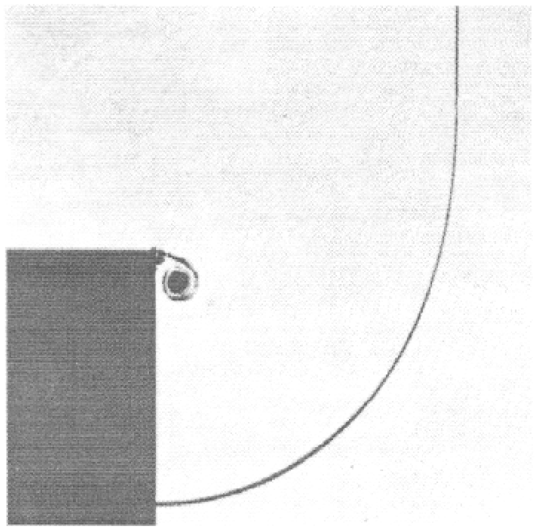


Figure 4.3: Schlieren image of a refracting shock wave, $M_s = 1.3$ (by Schardin [30])

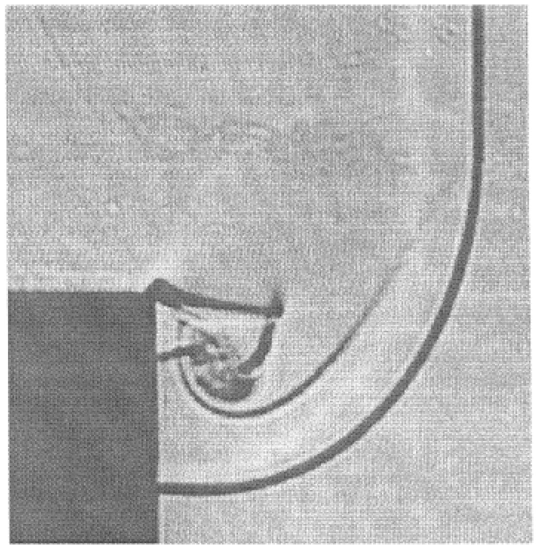


Figure 4.4: Schlieren image of a refracting shock wave, $M_s = 2.4$ (by Schardin [30])

mesh cells from the apex of the corner to the point where the Mach stem meets the wall.

All of the major flow structures named by Skews can be easily identified in Figure 4.6; the terminator, secondary shock, and contact wave can all be seen as sharply-defined density gradients. It is important to note that because of the HLLE switching function used in the simulation, no expansion shock (a discontinuous expansion wave) is present.

Hillier performed the simulation of a diffracting shock over a backward step in 1991 using less cells than Quirk (180 000), but for different Mach numbers [7]. Using an explicit second-order Godunov-type scheme, Hillier achieved computational results matching closely the experimental results of Skews (1967). Specifically, his results reflected that $a)$ the vortex location becomes

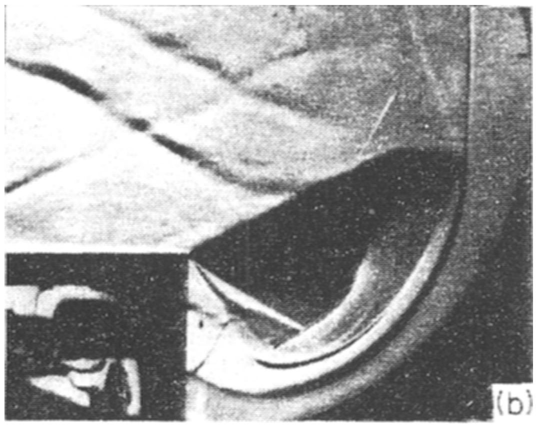


Figure 4.5: Schlieren image of a refracting shock wave, $M_s = 5.09$ (by Bazhenova et al. [2])

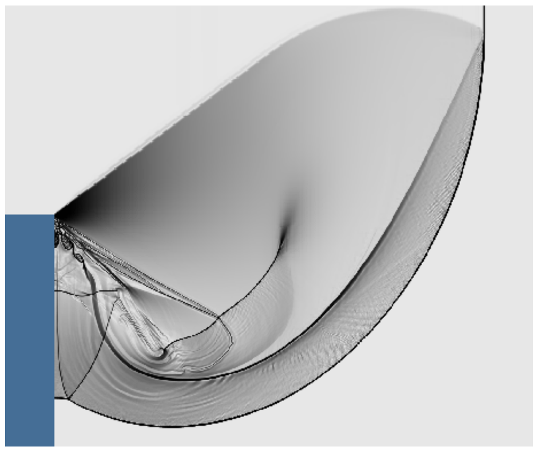


Figure 4.6: False Schlieren image of a refracting shock wave, $M_s = 5.09$ (by Quirk [21])

progressively ill-defined as M_s increases, and $b)$ the shape of the incident shock matches Skews' measurements.

4.2 Forward Step

This test case uses the same geometrical, boundary and flow parameters as the cases studied by Woodward and Colella [33], and van Leer [31]. The case was first proposed by Emery [4] for evaluating finite-difference methods, but his results are not comparable to the later studies owing to their low resolution and parametric differences.

The forward step test case is designed to resolve complex oblique shock reflections, pertinent to supersonic variable-geometry jet engine intakes. Due to computational constraints faced by Emery, the test case was designed to be numerically simple to set up (the initial conditions are uniform throughout the domain, and the outlet boundary condition is supersonic), but is actually difficult to perform experimentally, due to the unrealistic combination of initial and boundary conditions. However, quantitative results are available for a range of numerical methods (see [33]), which makes this a perfect case for validating the capabilities of the present mesh refinement algorithm, independently of the numerical method used.

The artificial nature of the test setup helps to evaluate the robustness of the scheme used when combined with the mesh refinement technique. Because of the strong shock reflection at the lower face of the step during the first few time steps, it is difficult to maintain positivity of pressure and density using various numerical schemes. Furthermore, the edge of the forward step is a singular point of the Prandtl-Meyer expansion fan generated by the flow over the step (John in [8] shows how the continuity and momentum equations can disobey the second law of thermodynamics through an expansion fan). This introduces numerical difficulty in the form of a nonphysical expansion shock, which at high Mach numbers and small cell sizes can yield negative pressure and density in the code.

4.2.1 Setup of Computational Problem

The geometry of the computational domain is identical to that used in [33], and is shown in Figure 4.7. In order for the cells at the lowest refinement level to be square, the domain was divided into cells of dimension $\{0.2 \times 0.2\}$, the greatest common factor of 0.6 and 0.8. The inflow and outflow boundary conditions are both supersonic, so the solver does not have to account for waves leaving the domain through the inflow or entering through the outflow. The wall boundaries, as in the backward step case, are full-slip pressure-reflection boundaries.

Initially, the fluid is at Mach 3 everywhere. Since no physical constants or parameters such as viscosity are involved in the Euler equations, space and time units can be eliminated without the need for a nondimensional constant (such as Reynolds number) [1] and the flow is driven by pressure and density *ratios*. The Mach number is calculated using the usual definition:

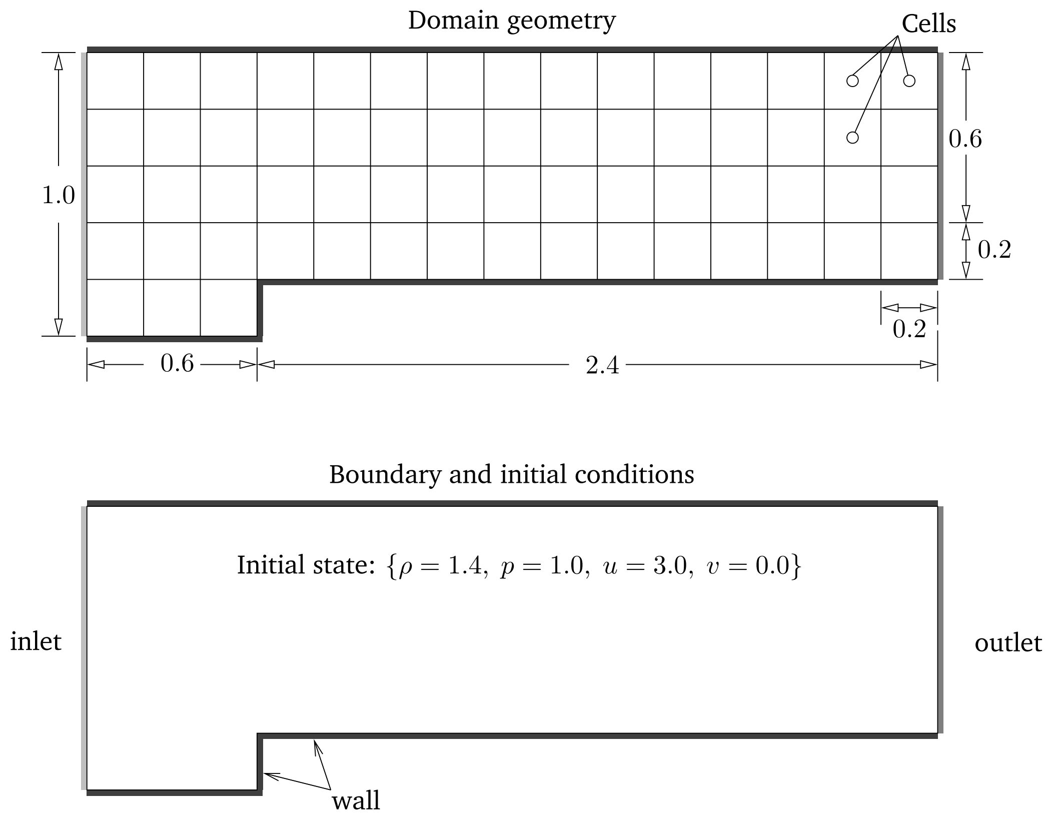
$$M = u\sqrt{\frac{\rho}{\gamma p}}. \tag{4.5}$$

The simulation is run until time $t = 4.0$, and the resulting shock wave pattern is examined. At this stage, the flow is still transient, but the shock waves are moving relatively slowly, which tests the ability of the refinement technique to maintain the solution quality as time progresses.

4.2.2 Existing Computational Results

Van Leer in [31] presents results by Woodward, who implemented a MUSCL (Monotonic Upstream-centred Scheme for Conservation Laws) code to compute the forward-facing step test case to both first- and second-order accuracy in space. The MUSCL scheme allows for the extension of the Godunov method to higher-order accuracy by using gradient information to determine fluxes. From these results, shown in Figures 4.8 and 4.9, it is clear that the first-order accurate solution cannot resolve the oblique shock wave reflected from the upper surface of the step.

Woodward and Colella later repeated the test case in order to compare a number of different numerical methods [33]. Their best results were obtained using Colella and Woodward's PPM (Piecewise Parabolic Method) scheme, and are reproduced in Figure 4.10. The bottom contour plot is of the quantity $A = ln(p) - \gamma ln(\rho)$, which is directly related to the entropy. Since entropy remains



Domain geometry
Cells
1.0
0.6
0.2
0.2
0.6
2.4
Boundary and initial conditions
Initial state: $\{\rho = 1.4,\ p = 1.0,\ u = 3.0,\ v = 0.0\}$
inlet
outlet
wall

Figure 4.7: Forward step geometry

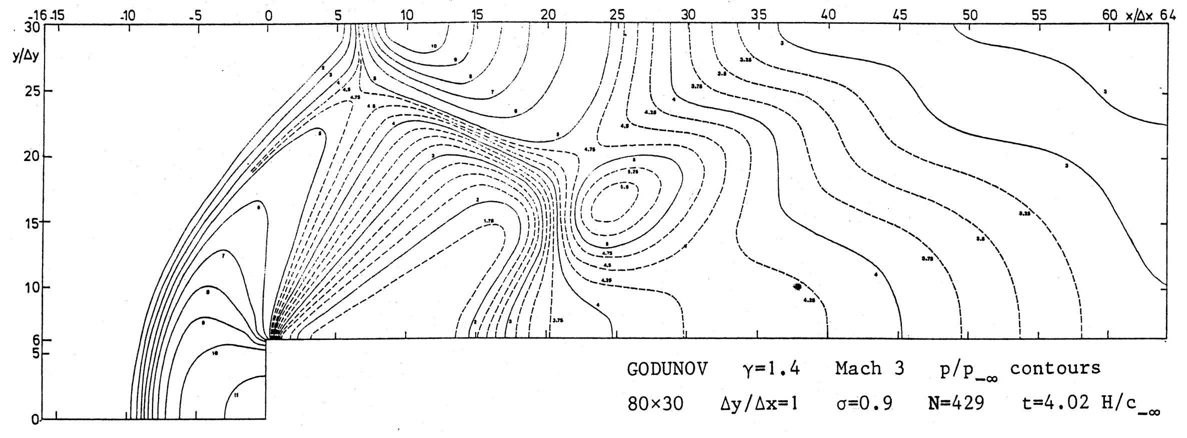


Figure 4.8: Woodward's results from [31]: First-order Godunov scheme

constant along streamlines through isentropic regions of the flow, this plot provides an idea of the instantaneous velocity field.

The PPM results were performed at a much higher resolution than the MUSCL ($\Delta x = \Delta y = 1/80$ vs. $\Delta x = \Delta y = 1/20$), which is the main reason for the improved resolution of the shock and

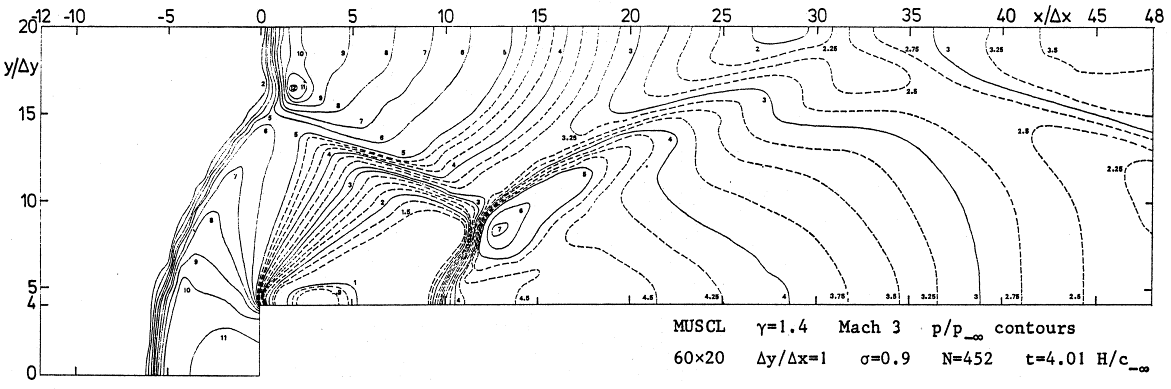


Figure 4.9: Woodward's results from [31]: Second-order MUSCL scheme

contact waves. However, there is a large discrepancy between the location of the reflected oblique shock wave on the upper surface of the step. Using the PPM scheme, the point of reflection clearly occurs near to the point $x = 1.45$; however, the MUSCL results show a lambda-shock instead of an oblique shock reflection. This is because the PPM results are *entropy-corrected* at the step corner. In [33], Woodward and Colella manually reset the density in the first four cells on the upper face of the step corner to make their entropy value the same as that found in the four cells on the lower, vertical face. Without this entropy correction, it is not possible to rid the results of the lambda-shock.

It is apparent from the pressure and density contours of Figure 4.10 that the reflected oblique shock waves generally become weaker (in terms of the density and pressure difference across the shock) the farther downstream they are located. This relative weakness causes low-resolution and low-order numerical methods to smear discontinuities to the point where the reflected shocks are unrecognizable, as in Figure 4.8.

Both sets of computational results provide quantitative contour information as well as insight into the strengths and weaknesses of various numerical schemes, which enables the author to separate the effects of mesh refinement from the attributes of the numerical scheme used.

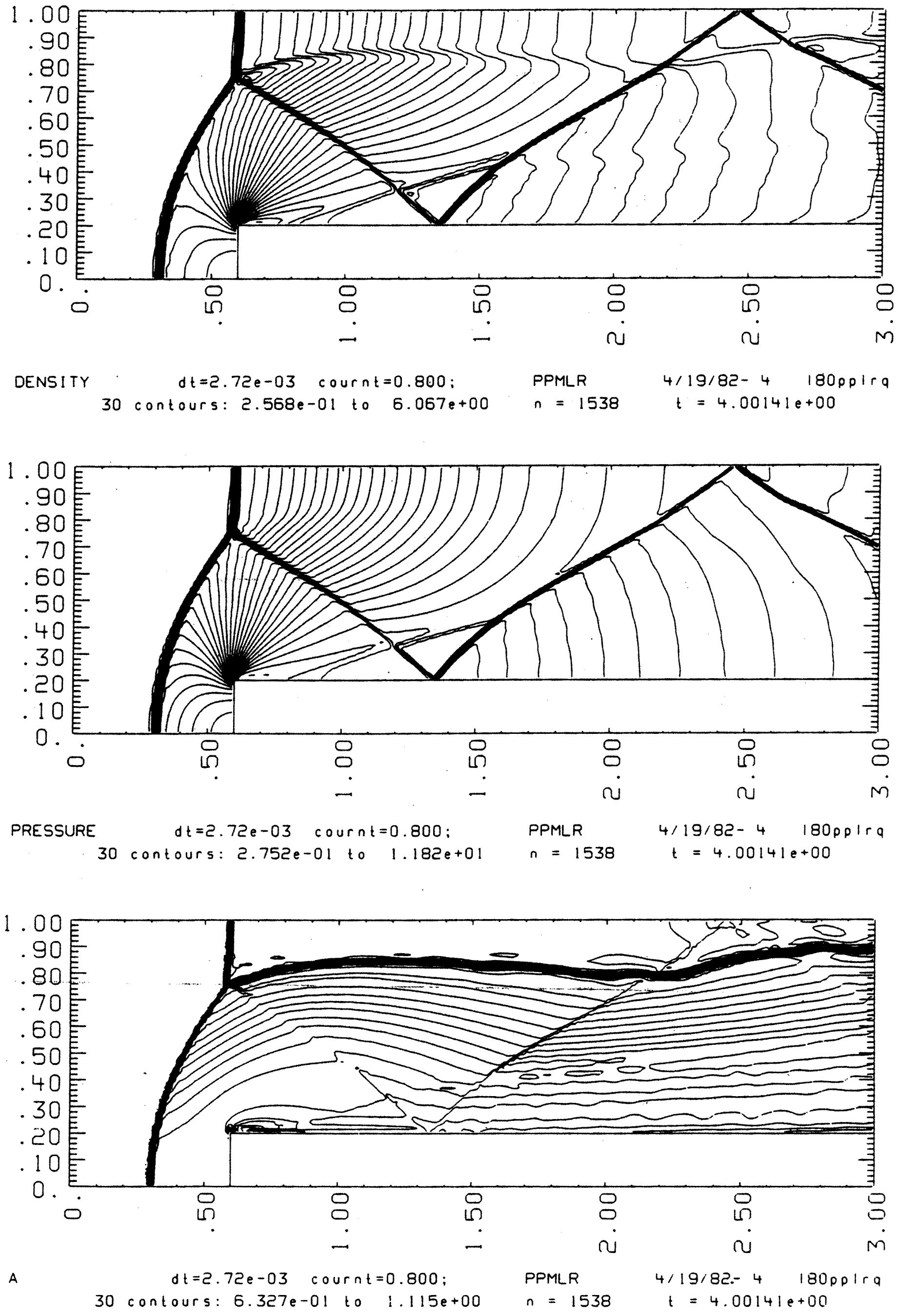


Figure 4.10: Woodward and Colella's PPM results from [33]: Density, pressure, and entropy

Chapter 5

Results for Uniform Cartesian Mesh

Both test cases were solved on uniform Cartesian meshes that were obtained by repeatedly refining the initial meshes shown in Figures 4.1 and 4.7. Uniform mesh results are important for two main reasons:

- They provide a way to validate the numerical method and data structure independently of the mesh refinement method.
- Once validated, they provide benchmark data against which the refined mesh results may be compared.

The 'higher-order' results described later in this chapter are not formally second-order accurate. The high Mach number test cases (the Mach 5.09 backward step and the Mach 3.0 forward step) are virtually impossible to solve[1] using the AUSM$^+$ scheme while incorporating full gradient corrections. Therefore, as suggested in section 3.5.3, the three stages of the Runge-Kutta time stepping scheme use different gradient corrections to calculate interface fluxes. Specifically, the calculation of $\mathbf{u}^{(*)}$ in the first stage uses the first-order extrapolation shown in Figure 6.9, while the other two stages use the second-order extrapolation. For the uniform mesh, the first-order extrapolation is essentially equivalent to using the cell-centred $\mathbf{u}$ in the calculation of the interface fluxes. The degeneration of the first sub-step to first-order accuracy adds numerical dissipation to the overall solution, allowing simulations to remain stable at high Mach numbers.

Time steps are chosen so that the CFL number does not exceed 0.2. This may seem low, and indeed the solution shows little or no degradation for CFL numbers less than 0.8. However, anisotropic refinement introduces numerical noise into the flow field that requires some time to dissipate, and so for purposes of comparison, the low CFL choice was made to be the same as that used to obtain the anisotropic results of chapter 7.

5.1 Backward Step

All backward step results were obtained on a relatively coarse mesh with each cell having dimensions $\Delta x = \Delta y = 40 \div 2^7 = 0.15625m$. The three initial cells shown in Figure 4.1 were all refined

[1]Density and pressure reach negative values.

7 times in each of the $x-$ and $y-$directions in order to construct the uniform mesh consisting of $49\,152$ cells.

5.1.1 First-Order Results

Pressure and density contours of the Mach 5.09 test case are provided in Figure 5.1 for the purpose of qualitatively comparing the ability of the AUSM$^+$ scheme to resolve flow features at different orders of accuracy.

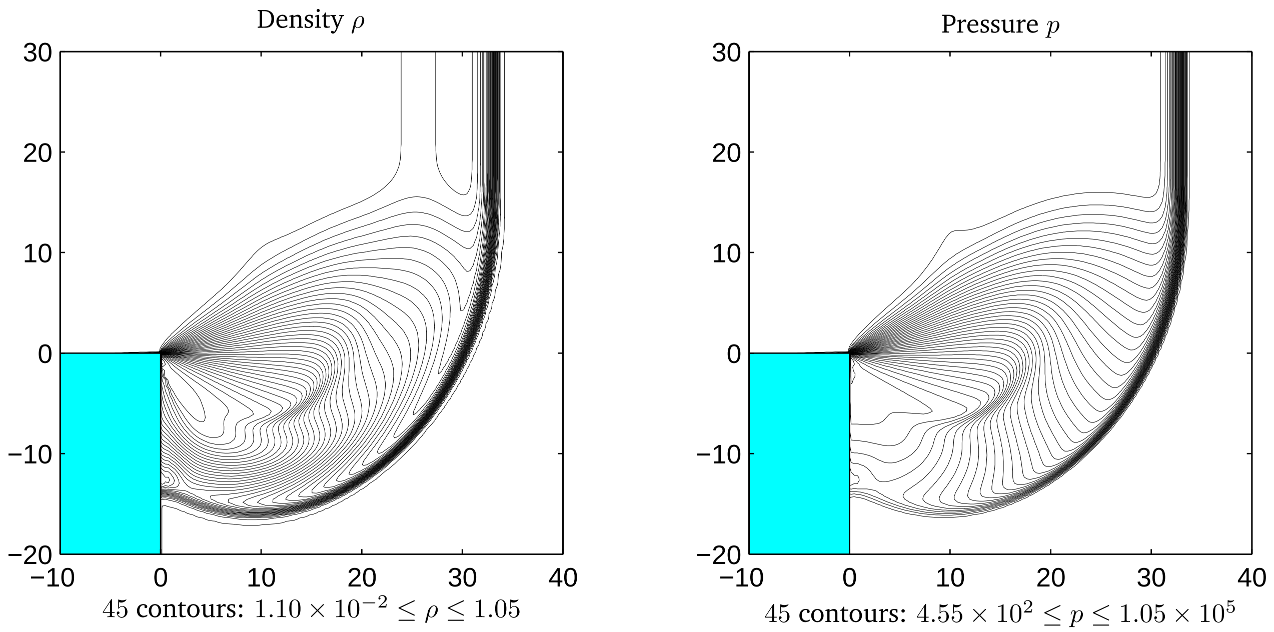
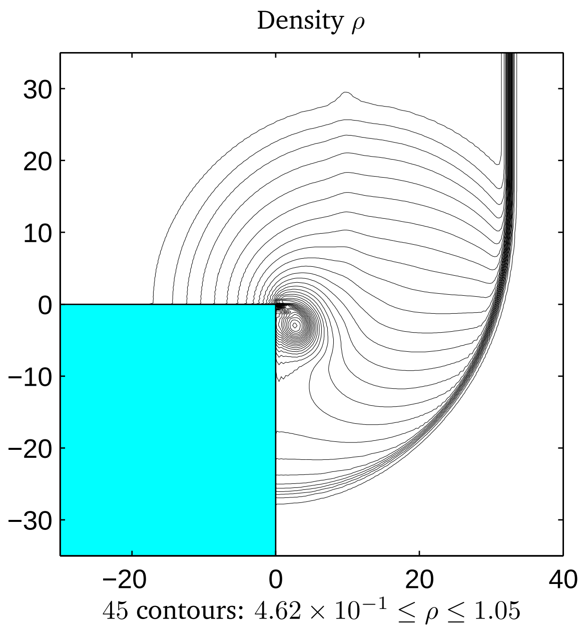


Figure 5.1: Mach 5.09 shock diffraction: pressure and density contours

5.1.2 Higher-Order Results

In each case, the superbee limiter was used in order to maintain positivity of density and pressure. Without limiting the gradient corrections in each cell, the code diverges for the Mach 2.4 and 5.09 flows.

By juxtaposing density and pressure contours in Figures 5.2 to 5.4, the reader is able to gauge the ability of the scheme to detect shock, contact and expansion regions.



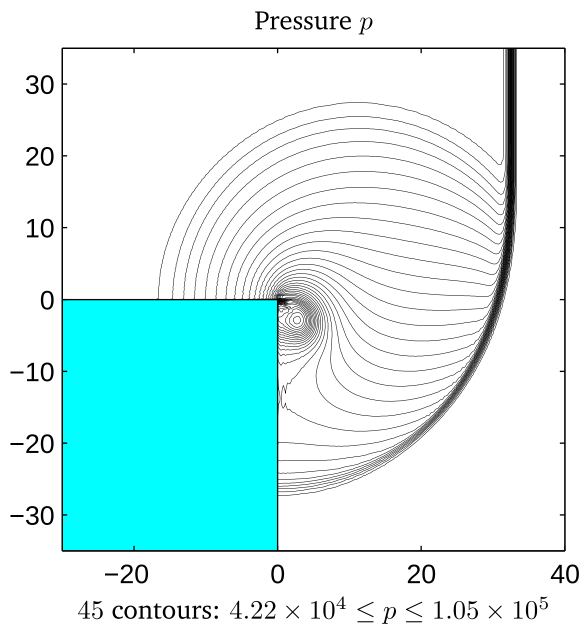


Figure 5.2: Mach 1.3 shock diffraction: pressure and density contours

5.1.3 Analysis of Results

Accuracy

Because of the relatively low resolution of the simulation (compared to Quirk's results, shown in Figure 4.6), the flow features outlined in section 4.1.2 are diffuse, but nonetheless identifiable. The following observations can be made about the validity of the results:

1. The shape of the primary shock wave for each Mach number matches its corresponding experimental result. The kink in the lower $M_s = 5.09$ shock wave is correctly resolved for both the first- and higher-order cases.

2. The contact wave appears very diffuse compared to the experimental and high-resolution numerical results. This is a drawback of all flux vector splitting methods [5]. It is more clearly resolved in the higher-order contour plots.

3. The secondary shock waves are correctly located and sized in the $M_s = 2.4$ and $M_s = 5.09$ plots. As shown by the experimental results, no secondary shock is present for the $M_s = 1.3$ case. Interestingly, Sun and Takayama determined analytically in [27] that the threshold incident shock Mach number M_s required for a secondary shock wave to form is $M_s = 1.346$.

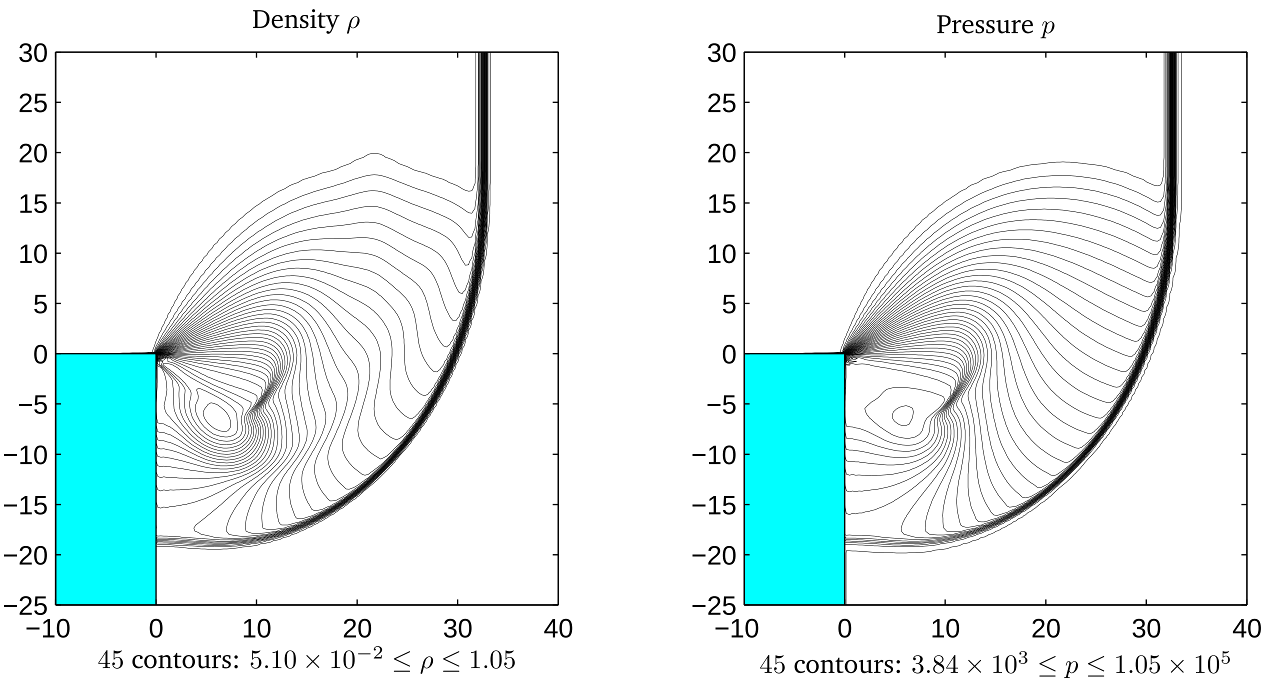


Figure 5.3: Mach 2.4 shock diffraction: pressure and density contours

5.2 Forward Step

Whereas the backward step cases were solved on a relatively coarse mesh, this case was solved using a relatively fine mesh. The backward step test case is meant to convey a qualitative picture of the flow features found behind a diffracting shock wave, and at low resolution the limitations of the numerical scheme are more apparent. For the forward step, however, we are mainly interested in the temporal accuracy of the scheme and its ability to discern slow-moving shock waves.

All of the following results were obtained on a uniform mesh with cell dimensions $\Delta x = \Delta y = 0.2 \div 2^6 = 3.125 \times 10^{-3}$. Based on the initial geometry shown in Figure 4.7, there are a total of $258\,408$ cells in the mesh.

5.2.1 First-Order Results

Density and pressure contours are plotted for the first-order results in Figure 5.5. The contour intervals were selected to match those used by Woodward and Colella as shown in Figure 4.10.

5.2.2 Higher-Order Results

Density and pressure contours are plotted for the higher-order results in Figure 5.6. The contour intervals were selected to match those used by Woodward and Colella as shown in Figure 4.10 in section 4.2.2.

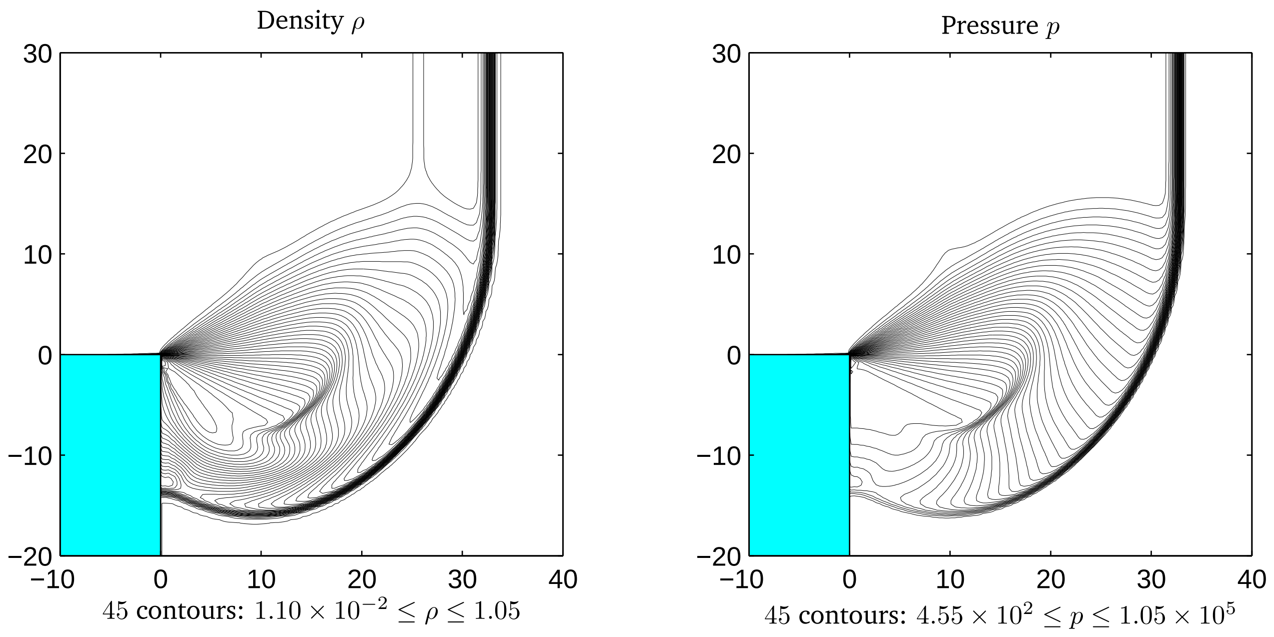


Figure 5.4: Mach 5.09 shock diffraction: pressure and density contours

Using the standard interface speed of sound definition,

$$a_{IF} = \sqrt{a_1 a_2}, \tag{5.1}$$

it was not possible to maintain positivity throughout the simulation while using the superbee limiter and mixed order-of-accuracy time stepping. To remedy this problem, the speed of sound definition was changed to be based on velocity and total enthalpy, as specified by Liou in [15]. This approach increases the numerical dissipation and allows the pressure and density to remain positive in the expansion region on the upper lip of the step.

5.2.3 Analysis of Results

Accuracy

The most obvious discrepancy between the first-order, higher-order and Woodward and Colella's results is that the location of the shock reflection on the lower surface is not well defined in Figures 5.5 and 5.6. Woodward and Colella [33] account for this by using the following arguments:

1. The inability of numerical schemes to avoid entropy-modifying expansion shocks sets up an erroneous numerical boundary layer that interferes with the shock reflection.

2. The expansion shock is coupled with an over-expansion at the step corner (visible in the entropy contours of Figure 4.10), which further exacerbates the accuracy problems at the corner of the flow.

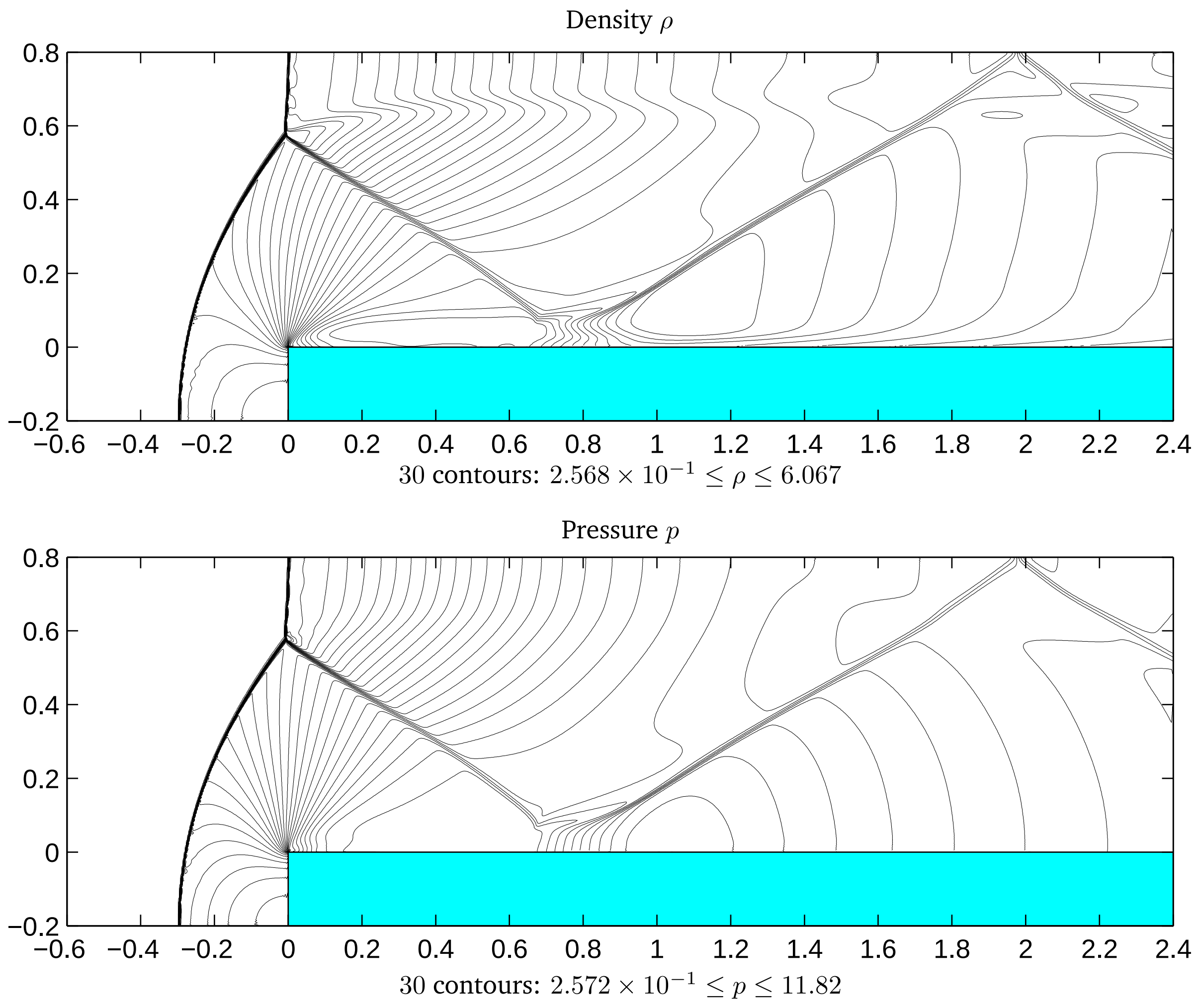
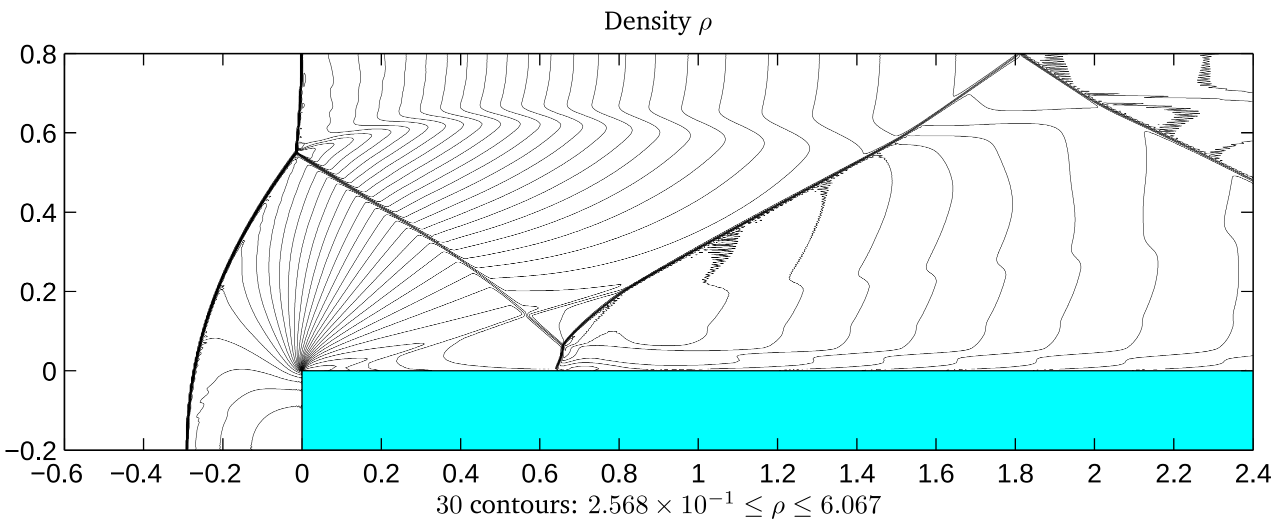


Figure 5.5: Mach 3.0 flow over a forward step: pressure and density contours

Another discrepancy between all sets of results is the location of the reflected shock wave on the upper surface of the channel. Since the location of this reflection is directly related to the location of the lower reflection, it is not surprising that all of the results are different. Woodward and Colella concede in [33] that there is no straightforward way to quantify numerical effects on this reflected shock location.

When the higher-order simulation is repeated one refinement level lower (ie. using cells twice as large), the speed of sound modification is no longer necessary for maintaining positivity, and the original a_{IF} definition (equation 5.1) can be used. This results in a thinning of the boundary layer on the lower surface and improvement in the location of the reflected shock.

Apart from the aforementioned discrepancies, the numerical agreement in the region of flow immediately following the primary shock wave is very good between the present results and those



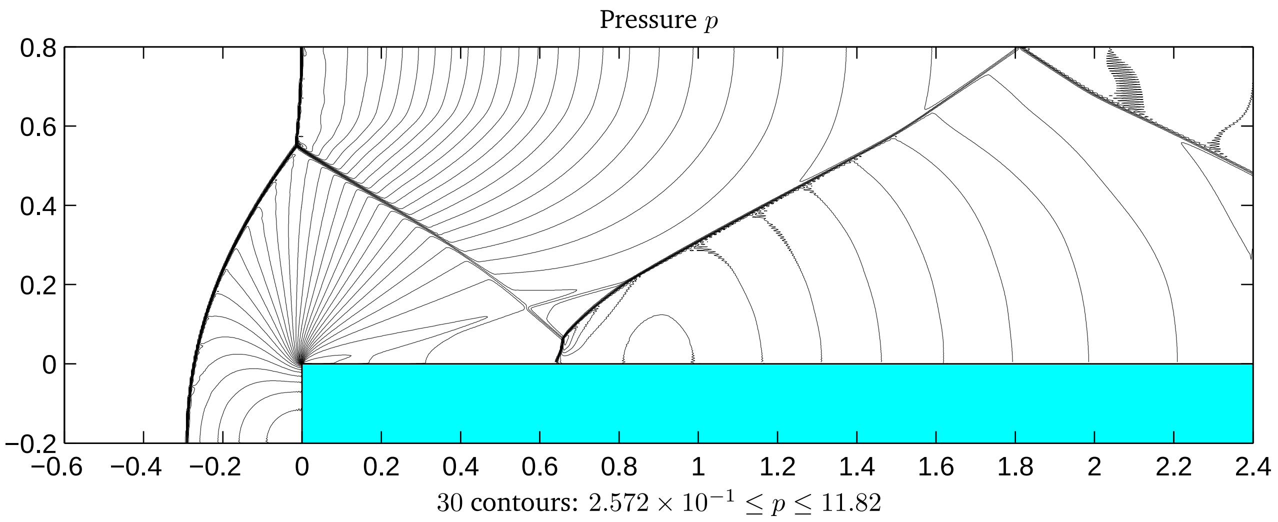


Figure 5.6: Mach 3.0 flow over a forward step: pressure and density contours

of Woodward and Colella. The shock locations and contour lines match closely[2]. This indicates that the AUSM$^+$ scheme combined with the Runge-Kutta time stepping is accurate within the limits of its abilities, and the present high-resolution results may be used as a benchmark for comparing the adapted mesh results of chapter 7.

[2]Note that the origins of the axes are located differently in Figures 5.6 and 4.10.

Chapter 6

Cartesian Mesh Adaptation

Mesh adaptation is the process of increasing and decreasing the sizes of computational cells according to a solution-related criterion. In most cases, the criterion dictates that cell sizes be reduced in regions of high solution gradient, and enlarged in regions of low solution gradient. For a first-order numerical scheme, the solution gradient is proportional to the Taylor series truncation error; for a second-order scheme, the Hessian matrix is a more accurate indicator of the truncation error. Many different criteria have been proposed in the literature, some of which use combinations of gradients and Hessians; these will be discussed in section 6.2.4.

The motivation for performing mesh adaptation arises from the practical computational expense of performing fluid flow simulations. For each cell in the mesh, several variables that define the state of the fluid must be stored, and new quantities (gradients, fluxes, etc...) must be calculated at each time step or iteration. Therefore, reducing the *total* number of cells in the mesh reduces the amount of memory needed for each simulation, as well as the time required by the computer to go through each cell and perform the necessary calculations. Alternatively, given a specific number of cells (and in turn a specific memory allocation), the resolution of the simulation can be improved by judiciously reassigning 'cell memory' to regions of greater interest in the flow. This approach is widely used in incompressible flow simulations by employing grid stretching to resolve high solution gradients in boundary layers.

Since the numerical solutions to the fluid flow problems discussed in this thesis arrive in the form of cell-centred $\mathbf{u}$ values, it is important to have greater numbers of fine-grained cells in areas of high solution interest. For compressible flow, these usually correspond to shock, contact, and expansion waves and their associated regions of interaction.

6.1 Survey of Adaptation Techniques

Techniques for adapting Cartesian meshes can presently be divided into isotropic and anisotropic approaches. In isotropic approaches, the aspect ratio of computational cells remains the same as they are subdivided and conjoined. In anisotropic approaches, cells are subdivided and reconnected in only one coordinate direction at a time. In two dimensions, isotropically and anisotropically adapted meshes can be represented as in Figure 6.1.

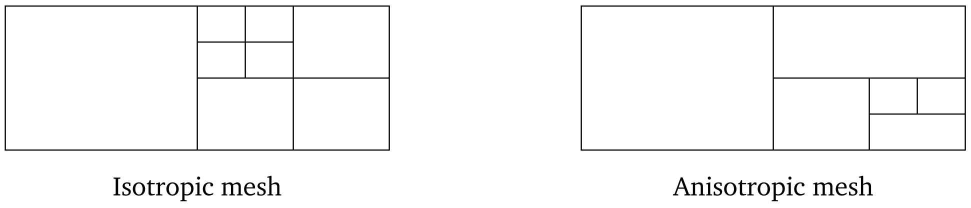


Figure 6.1: Two Cartesian mesh adaptation techniques

6.1.1 Isotropic

In [35], de Zeeuw and Powell present an isotropic mesh adaptation technique that is applied to steady compressible flow. The data structure used is tree-based, which means that in order to calculate fluxes, the tree must be traversed to obtain neighbouring cell information. Fortunately, the adaptation technique permits a maximum of only two neighbouring cells in any direction, which means that only one branch traversal is needed to locate each neighbouring cell. The arrangement of cells within the tree data structure is shown in Figure 6.2, with cell indices referenced as $\{$*refinement level, array index*$\}$.

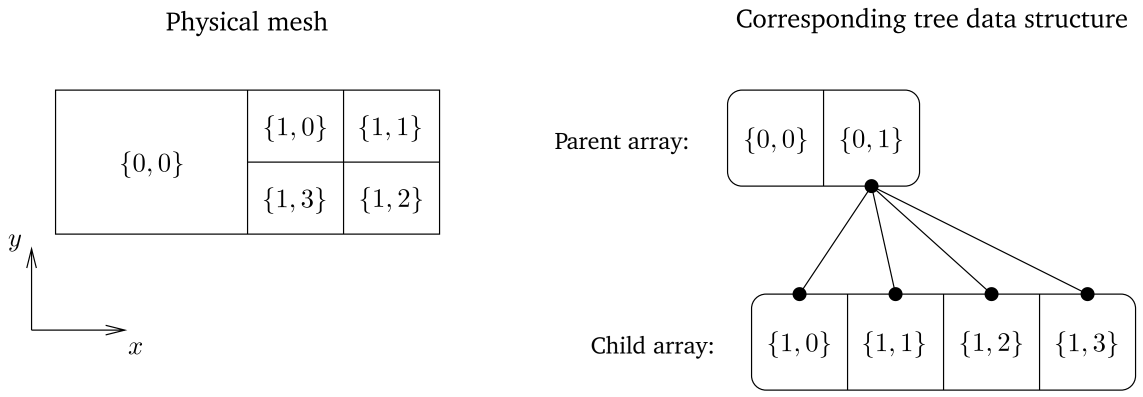


Figure 6.2: Tree data structure representation of physical mesh

Since de Zeeuw and Powell's technique is applied to steady flow, calculations begin on a coarse mesh which is strategically refined as the simulation progresses. Therefore, the method does not provide for the re-coarsening of cells in areas which should become smooth.

Sun also uses a tree-based isotropic adaptation technique [26] but applies it to unsteady flow, and therefore uses a coarsening procedure to join cells together when the solution is sufficiently smooth. Sun adopts many of the 'mesh quality' criteria used by de Zeeuw, such as the limit on the number of neighbouring cells.

6.1.2 Anisotropic

The grid adaptation method upon which this thesis is based was originally developed by Ham, Lien and Strong [6], and is an unstructured face-based method that can transiently refine and coarsen grid cells. Ham et al. applied the method to two- and three-dimensional incompressible flow at Reynolds numbers below 150. The main advantage of using anisotropic grid refinement over isotropic refinement is the fact that memory and computational savings can be realized when the flow gradients are anisotropic in the x- and/or y-directions. Even if anisotropy is present away from these cardinal directions, the method still yields computational savings over an isotropic technique.

6.2 Cartesian Anisotropic Adaptation for Compressible Flow

The adaptation method and criteria described in this section are independent of the choice of numerical method, provided that the method has a compact stencil and can be written in conservation form, or in other words, as the updating of cell-centred $\mathbf{u}$ based on interface $\mathbf{f}$ and $\mathbf{g}$. The data structure and algorithm are similar to those presented in [6], and are described here for completeness. The adaptation criterion used is different; the choice will be explained further in section 6.2.4.

6.2.1 Data Structure

Object and Class Definitions

The present code is object-oriented, with individual classes defined for computational cells and [inter]faces, as well as their encompassing mesh, as shown in Figure 6.3.

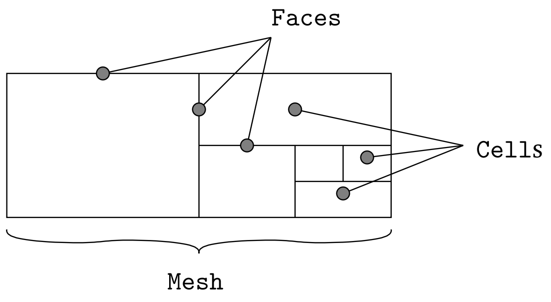


Figure 6.3: Classes of objects found in the computational mesh

In [6], Ham et al. use an (i, j) indexing scheme to keep track of cell (x, y) locations. The present code maintains (x, y) locations explicitly, but also retains the (i, j) indexing scheme in order to simplify the mesh adaptation process. Using (i, j) indices in order to isolate pairs of cells that may be conjoined turns out to be simpler to code than using (x, y) locations with cell refinement levels (l_i, l_j) to accomplish the same task.

Figure 6.4 illustrates the nature of cell indexing. The (i, j) indices correspond to the indices that would be needed if the mesh were structured, with each cell at the same refinement level. It is important to note that whereas the original indexing system of [6] was based around a single origin cell of indices $(0, 0)$ and refinement level $(0, 0)$, the system has been modified to permit multiple initial cells (which greatly aids in the formation of backward and forward steps) with refinement levels of $(0, 0)$, but nonzero indices.

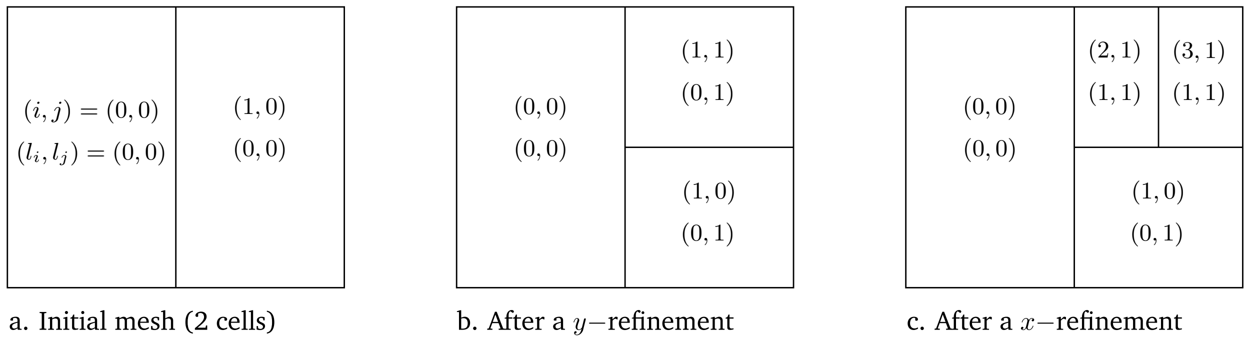


Figure 6.4: Cell indexing system used in the present code

The cell coarsening algorithm uses the (i, j) indices to determine suitable partner cells; in doing so, it helps to maintain the smoothness of the mesh by avoiding 'brick wall' formations, illustrated in Figure 6.5. Such irregular meshes were found by the author to cause serious degradation in the solution quality.

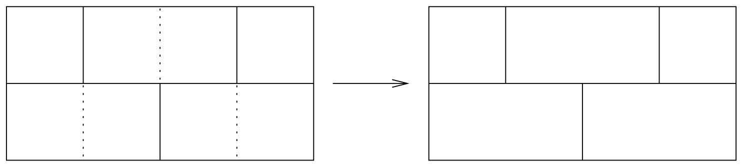


Figure 6.5: The (i, j) indexing system allows the algorithm to avoid such irregular coarsening

In terms of variable storage, `Cell` and `Face` objects are required to keep track of their respective conserved variables and fluxes, as well as standard geometric parameters required in an unstructured, adaptive mesh. Tables 6.1 and 6.2 summarize the necessary storage requirements for each class.

Flags must also be stored for the refinement and coarsening processes; however, they can be implemented in many different ways and are thus left to the developer. A possible method of storage for these flags is provided implicitly in the algorithm definitions of section 6.2.2.

Object Model

The relationships between `Cell`, `Face` and `Mesh` objects are shown in the UML (Unified Modelling Language) aggregation diagram of Figure 6.6 (see, for example, [3] or [11] for an introduction to UML and examples of aggregation diagrams). As described by Figure 6.3, the `Mesh` is composed of

`Cell`	`Face`	`Mesh`
$\mathbf{u}$	$\mathbf{f}$	CFL_{max}
$\nabla\phi,\ \forall i : \phi = \{\mathbf{u}\}_i$	(x, y)	$(l_i, l_j)_{max}$
(x, y)	l	$(l_i, l_j)_{min}$
$(\Delta x, \Delta y)$	*bcFlags[]*	*dimension[]*
(i, j)		
(l_i, l_j)		

Table 6.1: Class definitions

Variable name	**Definition**
$\mathbf{u}$	*cell-centred conserved variable*
$\nabla\phi,\ \forall i : \phi = \{\mathbf{u}\}_i$	*gradients of* $\mathbf{u}$ *components*
(x, y)	*cell/face-centred coordinates*
$(\Delta x, \Delta y)$	*cell size*
(i, j)	*cell index*
(l_i, l_j)	*cell refinement level*
$\mathbf{f}$	*face-centred flux*
l	*face length*
bcFlags[]	*boundary condition flags*
CFL_{max}	*maximum CFL number within the mesh*
$(l_i, l_j)_{max}$	*maximum allowable cell refinement levels*
$(l_i, l_j)_{min}$	*minimum allowable cell refinement levels*
dimension[]	*mesh layout parameters*

Table 6.2: Variable definitions

linked lists of individual `Cell` and `Face` objects. Linked lists are the most efficient data structures to use when objects must be dynamically added to and deleted from the list [25].

Required Object Facilities

Without discussing fine details of the code, the classes must implement the functionality required by the refinement and coarsening processes:

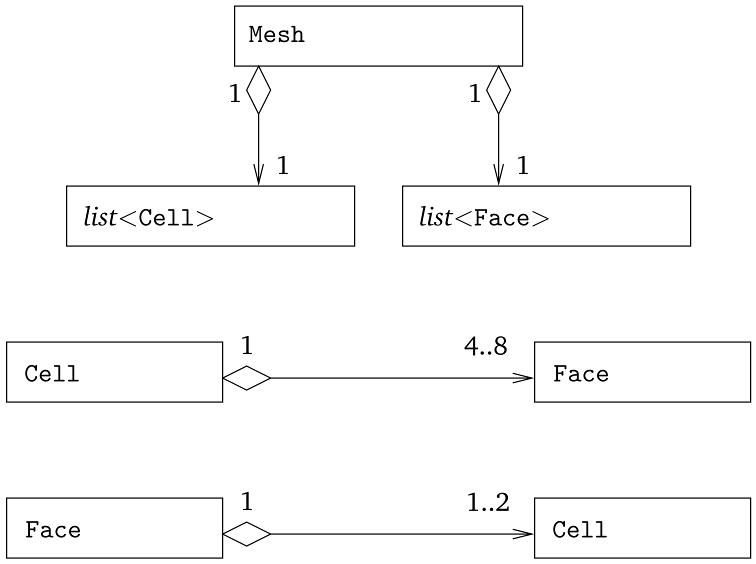


Figure 6.6: Class aggregation diagram

Cell Refinement: This process is outlined in Figure 6.7, and requires the following to be performed:

1. Create a new `Cell` and extrapolate $\phi_\circ$ to $\phi'_\circ$ and $\phi''_\circ$ using $\nabla\phi_\circ$.
2. Modify (x, y), $(\Delta x, \Delta y)$, (i, j), and (l_i, l_j) of the refined `Cells` appropriately.
3. Create a new `Face` between the split `Cells`, and remap neighbouring `Cells`' faces appropriately. For example, in Figure 6.7, the `Cell` containing ϕ_c will need another `Face` to be added to its collection of North `Faces`.
4. Modify (x, y), l and *bcFlags[]* of any `Faces` that were shortened; supply appropriate (x, y), l and *bcFlags[]* to any newly created `Faces`.

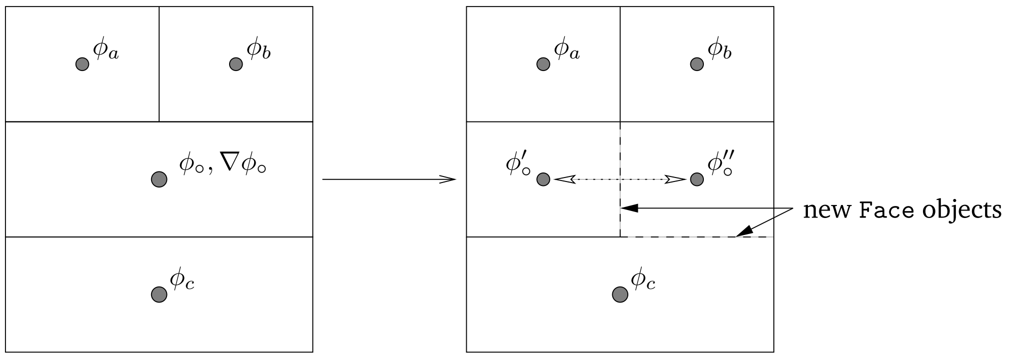


Figure 6.7: Cell refinement procedure

Cell Coarsening: This process is outlined in Figure 6.8, and requires the algorithm to:

1. Choose two `Cells` of the same dimensions and appropriate indices (i, j); remove the upper (right) `Cell` and extend the lower (left) `Cell`.
2. Average $\phi'_\circ$ and $\phi''_\circ$ to obtain $\phi_\circ$.
3. Delete the upper `Cell` and intermediate `Face` from the `Mesh`'s list of `Cells` and `Faces`.
4. Modify (x, y), $(\Delta x, \Delta y)$, (i, j), and (l_i, l_j) of the coarsened `Cell` appropriately.
5. Modify (x, y) and l of any `Faces` that must be lengthened; delete any `Faces` that were made redundant through the process of coarsening. For example, in Figure 6.8, the `Faces` beneath and to the left of $\phi''_\circ$ must be deleted.

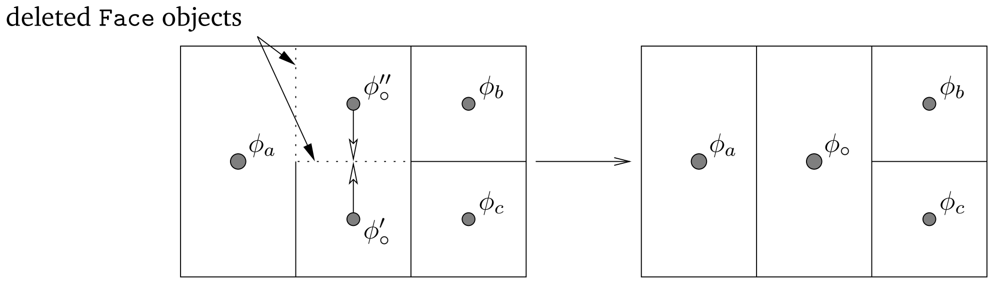


Figure 6.8: Cell coarsening procedure

6.2.2 Refinement and Coarsening Algorithms

Algorithms 6.1 – 6.4 reproduce in pseudo-code the $x-$refinement and $x-$coarsening algorithms as they are implemented in the present code. The $y-$refinement and $y-$coarsening algorithms are defined similarly. These algorithms differ from those of Ham et al. [6] in that additional restrictions are placed on the refinement levels of adjacent cells, as well as the fact that refinement proceeds from the largest to the smallest cells.

Algorithm Notation

According to the `Face` and `Cell` class aggregation diagram of Figure 6.6, objects of one type carry specific references to neighbouring objects of another type [1]. In the algorithms that follow, this relationship is described using the right arrow, '$\rightarrow$' . For example,

$$\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \text{flaggedRefineX}$$

refers to `Face` `f`'s left-hand `Cell` `c`'s flag 'flaggedRefineX'.

An arrow followed by an expression in braces means that the expression acts on the variable belonging to the object. For example,

$$\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \{l_i = l_i - 1\}$$

[1]e.g. in C++ a `Face` references its neighbouring `Cells` through pointer variables

means that `Face f`'s left-hand `Cell c`'s variable l_i must be decremented.

Loops and if-statements are terminated when the indent level of the pseudo-code block returns to the same level as the originating expression: 'end for' and 'end if' are implied. Comments are provided using slanted text enclosed by parentheses[2].

Algorithm 6.1 Flag and refine `Cells` in the x-direction

```
Require: all refinement/coarsening flags set to $false$
1: for all Cells c do
2:   {clear refinement & coarsening flags}
3:   c $\rightarrow$ flaggedRefineX $= false$
4:   c $\rightarrow$ flaggedCoarsenX $= false$
5:   Calculate least-squares c $\rightarrow \nabla\phi$
6:   Calculate c $\rightarrow \Delta x_{target}$
7: for all Faces f do
8:   Calculate interface gradients
9:   if $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \Delta x_{target} \leq \Delta x_L\right]$ OR $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow \Delta x_{target} \leq \Delta x_R\right]$ then
10:    $\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow$ flaggedRefineX $= true$
11:    $\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow$ flaggedRefineX $= true$
12: for all Cells c do
13:   Apply limiter to c $\rightarrow$ Faces $\rightarrow \nabla\phi$ to generate new c $\rightarrow \nabla\phi$
14: {go through Cells and refine level by level}
15: for $i = l_{i,min}$ to $l_{i,max} - 1$ do
16:   for all Cells c do
17:     if [c $\rightarrow$ flaggedRefineX $\equiv true$] AND [c $\rightarrow l_i \equiv i$] then
18:       call refineX(c) {see algorithm 6.2}
```

6.2.3 Application of the Algorithm to Compressible Flow Simulations

The refinement and coarsening algorithms are not exactly the same as described in [6]; the present code does not perform more than one iteration of cell coarsening for each refinement level. It was found that additional coarsening introduces numerical instability in areas of high gradient, while the memory savings are insignificant. Rather than calling the coarsen routine for each cell until all cells are at the largest size permitted by the mesh refinement criteria, the code makes $(l_{i,max} - l_{i,min})$ iterations through the code, coarsening the largest cells at first, and proceeding to the finest. Coarsening the largest cells first ensures that for a sweep of coarsening, any cell is only joined to another cell once, which helps to maintain the solution quality.

Another difference exists between the present and original versions of algorithms 6.2 and 6.4: the present versions contain additional restrictions on the refinement levels allowed between neighbouring cells. Specifically, lines 5–8 of refineX() and lines 13–19 of coarsenX() specify that East and West neighbouring cells cannot be smaller than half the width of the centre cell. This limit on the change in anisotropy over a given area yields a smoother solution for compressible flow, possibly because of the simulation's sensitivity to the gradient reconstruction. On a highly anisotropic mesh

[2]{*For example, this is a comment.*}

Algorithm 6.2 *boolean* refineX(`Cell c`)

```
1:  for all neighbouring Cells $c_{nb} \in$ Faces $\{N, S\}$ do
2:    {refine Cells that are preventing c's refinement}
3:    if [$c_{nb} \rightarrow l_i \equiv l_i - 1$] AND [refineX($c_{nb}$) $\equiv false$] then {recursive refinement}
4:      return $false$
5:  {refine cells to maintain a maximum $l_i$ difference of 1 between Cells }
6:  for all neighbouring Cells $c_{nb} \in$ Faces $\{W, E\}$ do
7:    if [$c_{nb} \rightarrow l_i < l_i$] AND [refineX($c_{nb}$) $\equiv false$] then {recursive refinement}
8:      return $false$
9:  Create new cell $c_\circ$, while c becomes the refined cell to the West
10: $c\rightarrow \{l_i = l_i + 1\}$
11: $c\rightarrow \{i = 2i\}$
12: $c_\circ \rightarrow l_i = c\rightarrow l_i$
13: $c_\circ \rightarrow i = c\rightarrow i + 1$
14: Add faces and modify geometric data according to Figure 6.7
15: Interpolate cell-centred $\phi$ using limited $\nabla\phi$
16: set $c\rightarrow$ wasXRefined $= true$
17: set $c_\circ \rightarrow$ wasXRefined $= true$
18: return $true$
```

Algorithm 6.3 Flag and coarsen two `Cells` in the x-direction

```
Require: all coarsening flags set to $false$
1:  for all Cells c do
2:    Calculate least-squares $c\rightarrow \nabla\phi$
3:    Calculate $c\rightarrow \Delta x_{target}$
4:  for all Faces f do
5:    if $\left[f \xrightarrow{L} c \rightarrow \Delta x_{target} \geq 2\Delta x_L\right]$ OR $\left[f \xrightarrow{R} c \rightarrow \Delta x_{target} \geq 2\Delta x_R\right]$ then
6:      $f \rightarrow$ doCoarsen $= true$
7:      $f \xrightarrow{L} c \rightarrow$ flaggedCoarsenX $= true$
8:      $f \xrightarrow{R} c \rightarrow$ flaggedCoarsenX $= true$
9:  {go through Faces and coarsen level by level}
10: for $i = l_{i,min} + 1$ to $l_{i,max}$ do
11:   for all Faces f do
12:     if [$f \rightarrow$ doCoarsen $\equiv true$] AND [$f \rightarrow$ wasXCoarsened $\equiv false$] then
13:       call coarsenX(f, $i$) {see algorithm 6.4}
```

Algorithm 6.4 coarsenX(`Face f`, level l)

```
1:  {make sure we are coarsening cells at the correct refinement level}
2:  if $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_i \neq l\right]$ AND $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_i \neq l\right]$ then
3:      return $false$
4:  {don't coarsen cells that are/were flagged for refinement}
5:  if $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \text{flaggedRefineX} = true\right]$ AND $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow \text{flaggedRefineX} = true\right]$ then
6:      return $false$
7:  {only cells of the same dimensions can be coarsened}
8:  if $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_i \neq \texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_i\right]$ OR $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_j \neq \texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_j\right]$ then
9:      return $false$
10: {prevent brick-wall coarsening shown in Figure 6.5}
11: if $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow i \div 2 \neq \texttt{f} \xrightarrow{R} \texttt{c} \rightarrow i \div 2\right]$ then {the $\div$ sign denotes integer division; e.g. $3 \div 2 = 1$}
12:     return $false$
13: {do not coarsen if the neighbouring cells are finer}
14: for all Faces $\texttt{f} \xrightarrow{L} \texttt{c} \xrightarrow{W} \texttt{f}$ do
15:     if $\left[\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow l_i > l\right]$ then
16:         return $false$
17: for all Faces $\texttt{f} \xrightarrow{R} \texttt{c} \xrightarrow{E} \texttt{f}$ do
18:     if $\left[\texttt{f} \xrightarrow{R} \texttt{c} \rightarrow l_i > l\right]$ then
19:         return $false$
20: {enforce the limit of $\leq$ 2 neighbouring cells}
21: for all Cells $\texttt{f} \xrightarrow{R,L} \texttt{c}$ do
22:     if [c has $> 1$ Face f $\in$ Faces $\{N, S\}$] then
23:         return $false$
24: Modify face connectivity and geometry according to Figure 6.8
25: Modify cell geometry for $\texttt{f} \xrightarrow{W} \texttt{c}$; average cell-centred $\mathbf{u}$ values
26: $\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \{l_i = l_i - 1\}$
27: $\texttt{f} \xrightarrow{L} \texttt{c} \rightarrow \{i = i \div 2\}$
28: Remove $\texttt{f} \xrightarrow{R} \texttt{c}$ from the list of Cells
29: Remove f from the list of Faces
30: return $true$
```

consisting of high aspect-ratio cells, it is difficult to achieve a smooth and accurate reconstruction using the least-squares method.

Performing a sweep of the entire mesh to coarsen and refine the cells is not necessary at every time step. The present code operates at low CFL numbers (in the range of 0.1 to 0.2), which implies that the solution requires 5 to 10 time steps to change significantly within a cell. The refinement algorithm is therefore run only every 5 time steps, and the coarsening algorithm every 20. Coarsening regularly does not serve to preserve solution accuracy and is computationally more intensive than refinement, hence its relatively infrequent application. Infrequent coarsening was also found to yield overall memory savings similar to those yielded by frequent coarsening; this was a promising result since it implied that cells were not alternating between coarsening and refinement.

6.2.4 Adaptation Criteria

An adaptation criterion uses a function that is sensitive to some error measure or flow features to specify which cells need to be refined and coarsened. Often, this function requires the user to supply parameters for the refinement process to proceed correctly. This either allows finer control of the solution quality, or else it complicates the problem by introducing several unknowns that CFD practitioners must 'feel' their way around.

There are many adaptation criteria in the literature; however, most fall under one of the following categories:

1. *Manually-tuned gradient sensors*; for example, the criterion used for steady compressible flow by Ripley [22], based on an implementation by Lien [13]. This criterion flags cells for refinement based on the absolute value of the difference between the density in the centre cell and its neighbours. It is a simple criterion that is easy to implement and consumes little computational time. However, it is not sensitive to changes in the second derivative, and requires the user to specify a tolerance for the gradient, which depends on the flow being simulated.

2. *Combination gradient/higher order derivative sensors*. In [26], Sun uses a criterion which is based on the ratio of the second-order Taylor series truncation error term to the first order one, which, when discretized, approximates a sensor function $f(\phi)$ based on the gradients ϕ_x at the cell centre and a neighbouring face:

$$f(\phi) = \frac{|\phi_{x,\text{face}} - \phi_{x,\circ}|}{\alpha\phi_\circ/\Delta x + |\phi_{x,\circ}|} \approx \left| \frac{\phi_{xx}\Delta x}{2\phi_x} \right| , \tag{6.1}$$

   where α is a small user-defined parameter used to prevent division by zero in the denominator. Sun used central finite differences to estimate the gradients at cell faces, and a least-squares technique to reconstruct the cell-centred gradient. Flagging cells for refinement and coarsening involves comparing the sensor function to user-specified upper and lower bounds on the ratio, ϵ_r and ϵ_c:

$$\begin{aligned} &\text{if } f(\phi) > \epsilon_r \Rightarrow \text{ refine cells on either side of face.} \\ &\text{if } f(\phi) < \epsilon_c \Rightarrow \text{ join cells on either side of face.} \end{aligned}$$

   This approach yielded good results for Sun's isotropically adapted mesh, but the user-supplied parameters were flow-dependent, and the method did not perform as well on the present anisotropic mesh, due to the noisy nature of the second derivative.

3. *Optimal cell dimensions based on error criteria.* For example, consider the lowest-order term of the Taylor series truncation error (which is dependent on the order of the scheme used) integrated over the cell area, as presented in [6]. For a second-order accurate scheme, this error tolerance is a function of the cell dimensions Δx, Δy, and the second derivatives $\nabla^2\phi$. Ham et al. in [6] obtained quasi-optimal expressions for $\{\Delta x, \Delta y\}_{\text{target}}$, and used those to determine the required levels of coarsening and refinement in the adaptation algorithm.

In terms of the amount of user knowledge and intervention required to operate the CFD code, method 3 is the most foolproof since no assumptions about the solution need to be made. The L_∞ error norm provides a worst-case estimate of the error bound, and while there is no guarantee that this matches the actual error in the solution, it is the best that can be achieved using a Taylor remainder approximation.

For the present code, method 3 was adopted. However, instead of using the second-order term in the Taylor series, the first-order term was used to generate optimal $\{\Delta x, \Delta y\}_{\text{target}}$. Two arguments can be used to justify this choice of error indicator:

1. Although the numerical method used is formally second-order accurate, the use of gradient limiters and a mixture of quasi-first-order accurate corrections (seen in section 6.2.5) reduces the code to first-order accuracy in the presence of discontinuities and sharp/oscillatory flow features.

2. The second derivative is noisy and computationally expensive to reconstruct on a nonuniform grid, which causes the corresponding adaptation to be haphazard and lose accuracy in key locations.

Using $\nabla\phi$ as the functional of interest (the present code takes $\nabla\phi = |\nabla\rho|$) and the first-order term of the Taylor remainder to be the error term, the optimal $\{\Delta x, \Delta y\}_{\text{target}}$ were found to be:

$$\Delta x^* = \left(\frac{2\tau\phi_y}{\phi_x^2}\right)^{\frac{1}{3}} \tag{6.2}$$

$$\Delta y^* = \left(\frac{2\tau\phi_x}{\phi_y^2}\right)^{\frac{1}{3}} \tag{6.3}$$

where τ is the user-specified error tolerance. These expressions are derived in appendix A.2.

In smooth flow regions, one or both of ϕ_x and ϕ_y may be zero; hence, to avoid division by zero in the code, practical expressions are:

$$\Delta x^* = \left(\frac{2\tau(\phi_y+\alpha)}{(\phi_x+\alpha)^2}\right)^{\frac{1}{3}} \tag{6.4}$$

$$\Delta y^* = \left(\frac{2\tau(\phi_x+\alpha)}{(\phi_y+\alpha)^2}\right)^{\frac{1}{3}} \tag{6.5}$$

where α is some small nonzero constant. In the code, $\alpha = 0.01\tau$ was chosen arbitrarily Ideally, α should be much smaller than τ but larger than machine ϵ in order to avoid catastrophic cancellation.

6.2.5 Solution Reconstruction via the Gradient

As described in section 3.5.1, the MUSCL approach is used to improve the accuracy of the numerical estimate of the cell interface flux. This involves extrapolation of the solution $\mathbf{u}$ from the centre of the cell to another point in the cell. For a formally second-order accurate numerical method on a uniform Cartesian mesh, this extrapolation takes $\mathbf{u}$ to the face of the cell. For the corresponding first-order accurate method, no extrapolation is performed, and interface fluxes are based on cell-centred $\mathbf{u}$.

On the adapted Cartesian meshes presented in this thesis, using cell-centred $\mathbf{u}$ to calculate interface fluxes results in severe degradation of the solution quality. Only by using gradient corrections is the solution rendered reasonably smooth as compared to uniform mesh results. However, for difficult test cases involving high Mach numbers and rapid expansions, even gradient-limited second-order solutions can yield negative pressures and densities. For this reason, it is sometimes necessary to have a first-order scheme that will work because of its dissipative nature.

A quasi-first-order accurate scheme can be constructed by using an auxiliary node approach: on both sides of a given interface (where the flux must be calculated), cell-centred $\mathbf{u}$ are extrapolated to points equidistant from the interface. These points lie on a line that is perpendicular to the interface and crosses its centre. The distance of the points from the interface is the minimum of the perpendicular distances between each of the cell centres and the interface. By moving the locations of the extrapolated $\mathbf{u}$ toward the interface, the scheme is extended from first to second-order accuracy. Figure 6.9 demonstrates the extrapolation graphically.

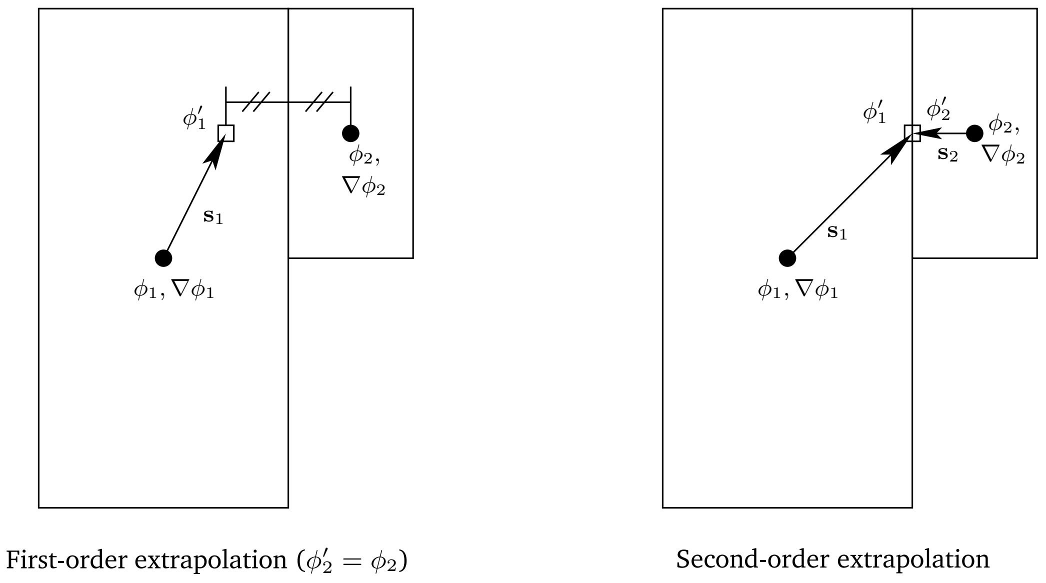


Figure 6.9: First- and second-order gradient extrapolation on an anisotropic Cartesian mesh; $\phi' = \phi + \nabla\phi \cdot \mathbf{s}$

The least-squares method was used to estimate solution gradients; the process is described in Appendix A.1. In the specific case of a uniform mesh, it can be shown that the least-squares method approximate the gradient via two-point central differences.

Application of Gradient Limiters

In addition to using $\nabla\phi$ to increase the order of accuracy of the method, the code requires gradient information during the refinement procedure to determine new cell-centred ϕ values (see Figure 6.7). As described in section 3.5.2, the use of unlimited gradient information can result in negative pressure and density in some cells, which dooms the entire flow simulation. The implementation in code of the minmod and superbee limiters, as they limit x-gradients, is as follows:

Using Figure 6.10 as a reference, both limiters must decide which [combination] of the following gradients to use as the cell-centred value:

$$\left.\frac{\partial\phi}{\partial x}\right|_{ab}, \quad \left.\frac{\partial\phi}{\partial x}\right|_{ac}, \quad \left.\frac{\partial\phi}{\partial x}\right|_{ad} \tag{6.6}$$

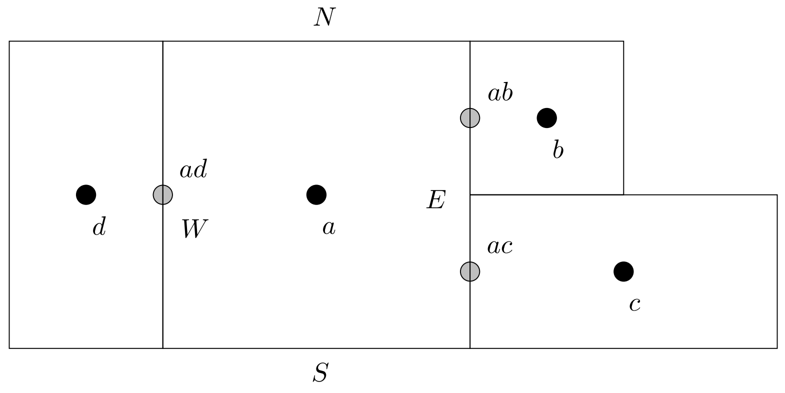


Figure 6.10: Mesh used for gradient limiter application in the x-direction

The face gradients are approximated as in Sun's implementation [26]:

$$\left.\frac{\partial\phi}{\partial x}\right|_{ab} \approx \frac{\phi_b - \phi_a}{\frac{\Delta x_a}{2} + \frac{\Delta x_b}{2}} \tag{6.7}$$

For each cardinal direction, $\{N, S, E, W\}$, if the cell has two faces in a single direction, the face whose gradient has the lowest absolute value is chosen for the ensuing minmod or superbee function evaluation. For example, for the East faces of cell a in Figure 6.10:

$$\left.\frac{\partial\phi}{\partial x}\right|_{E} = minmod\left(\left.\frac{\partial\phi}{\partial x}\right|_{ab}, \left.\frac{\partial\phi}{\partial x}\right|_{ac}\right) \tag{6.8}$$

The chosen limiter function is then applied exactly as described in section 3.5.2, using the ratio of the $\{E, W\}$ pair of gradients as input arguments.

Chapter 7

Results for Anisotropically Adapted Mesh

The results presented in this chapter were obtained on anisotropically refined meshes. Two types of solutions are presented:

1. Results obtained on meshes where the minimum cell dimension is equal to that of the uniform mesh; these are provided in order to validate the adaptation technique.
2. Results obtained on meshes whose minimum cell dimensions provide yet higher resolution than the uniform mesh result, while requiring a similar amount of computer memory.

The core numerical methods used to obtain the adapted mesh results are the same as those used for the uniform mesh results of chapter 5. Refinement sweeps are made once every 5 time steps, and coarsening sweeps are made once every 20. The CFL number of 0.2 ensures that the refinement process can keep track of the change in the solution as time progresses[1].

7.1 Backward Step

Results of the first type are presented in Figures 7.1, 7.2 and 7.3. The mesh was adapted according to the target cell sizes defined in equations 6.4 and 6.5 using $\tau = 10^{-4}$ and $\nabla\phi \cdot \mathbf{e}_i = |\nabla\rho \cdot \mathbf{e}_i|$. Density was chosen as the sensor variable because it is the conservative variable that best indicates changes across shock, contact and expansion waves.

The contours of target cell size are arranged in a geometric sequence as indicated by the caption. This information was obtained from the uniform mesh results via the calculation of $(\Delta x^*, \Delta y^*)$ for every cell. Since the refinement algorithm is relatively conservative (it refines some cells in order to maintain the smoothness of the mesh), the actual refined mesh has small cells in places that the

[1]It does this by ensuring that refinement is performed at least as often as the fastest-travelling wave in the domain can travel the length of one of the finest cells.

$(\Delta x^*, \Delta y^*)$ information does not predict. In general, however, the qualitative agreement between the actual refined mesh and the predicted cell sizes is good.

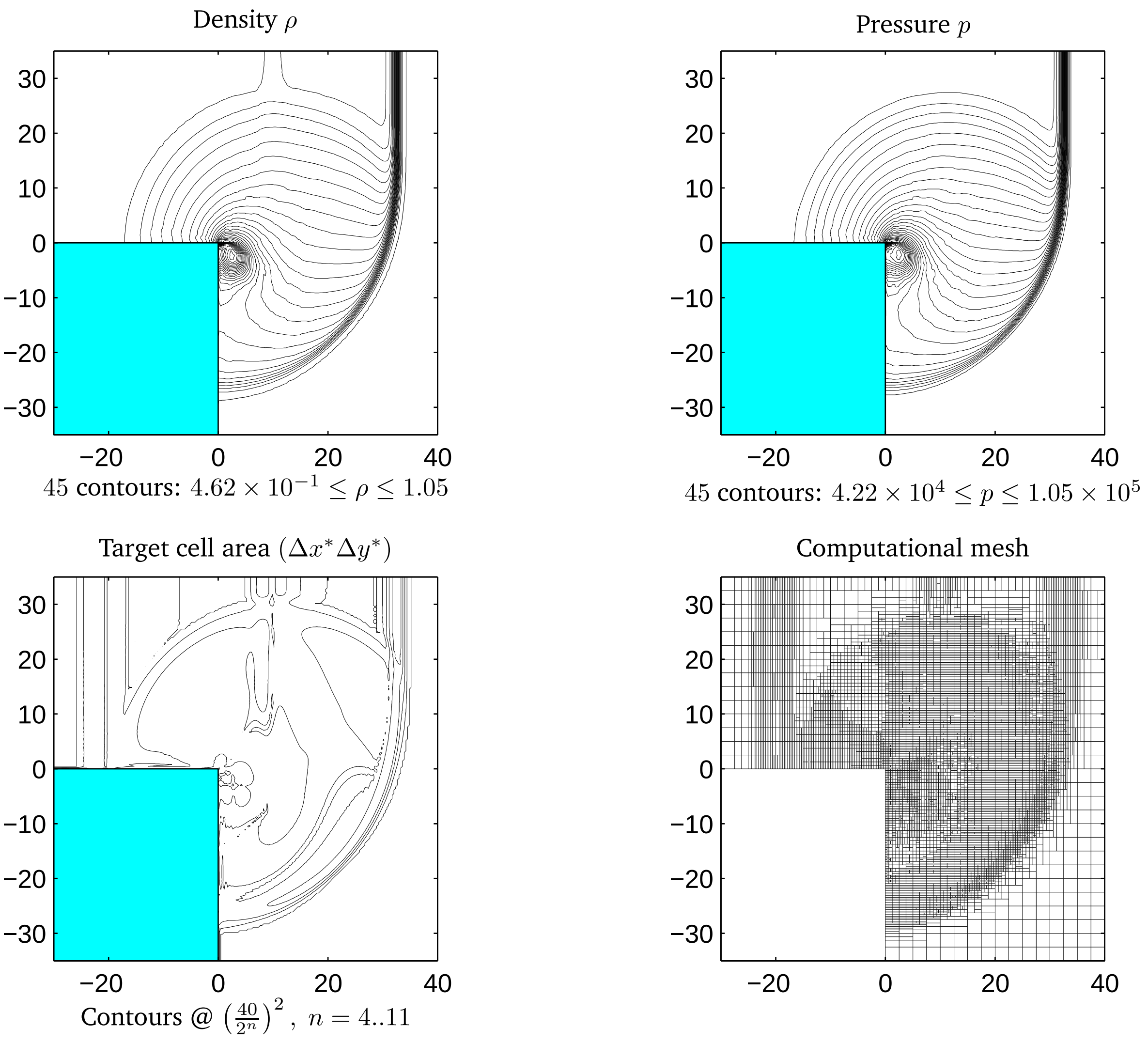


Figure 7.1: Mach 1.3 shock diffraction: pressure and density contours

Note that the smallest target cell areas correspond to areas where the shock wave or density gradient is at a 45° angle to the mesh; as mentioned earlier, this is the case in which anisotropic refinement holds no advantage over isotropic.

The results shown in Figures 7.4 – 7.6 are of the second type: the minimum cell dimensions have been reduced to $\Delta x = \Delta y = 40 \div 2^9 = 7.8125 \times 10^{-2}m$ in order to show the flow features more clearly. The refinement sensor and value of τ remain the same. In addition to providing density contours, a false Schlieren image has been created from the density field using a technique outlined by Quirk [21].

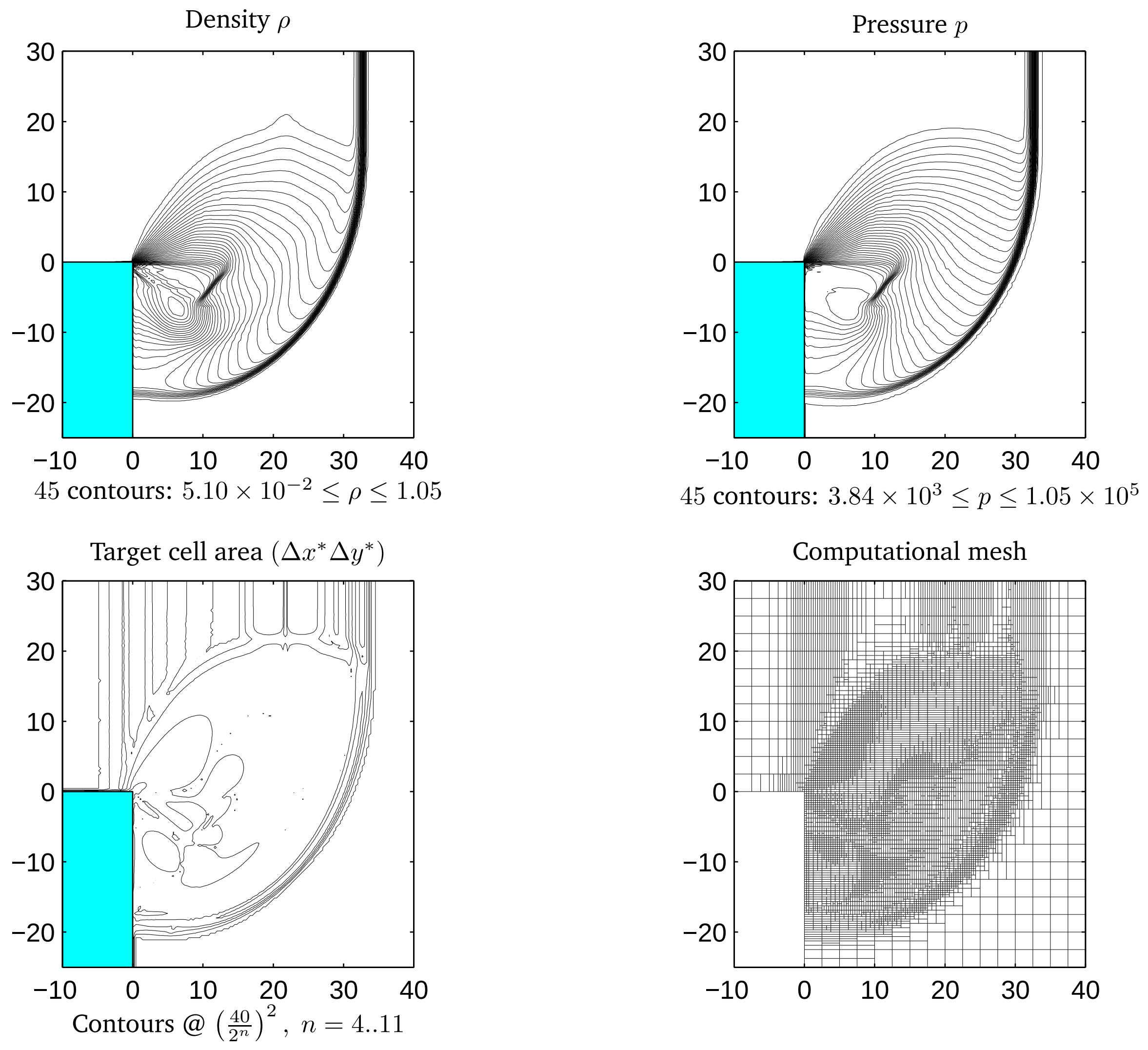


Figure 7.2: Mach 2.4 shock diffraction: pressure and density contours

7.1.1 Analysis of Results

Quality

The contour plots of density and pressure shown in Figures 7.1 – 7.3 show rougher contours than those of the uniform mesh. This is because the data structure provides the contour routine with regularly-spaced data by using a combination of the irregular cell-centred data and gradient extrapolation. The present code outputs flow field data as a set of regularly-spaced values superimposed on a uniform mesh whose cell size corresponds to the finest permissible cells of the adapted mesh. Thus, the contours are subject to deficiencies associated with the gradient extrapolation and limiter techniques.

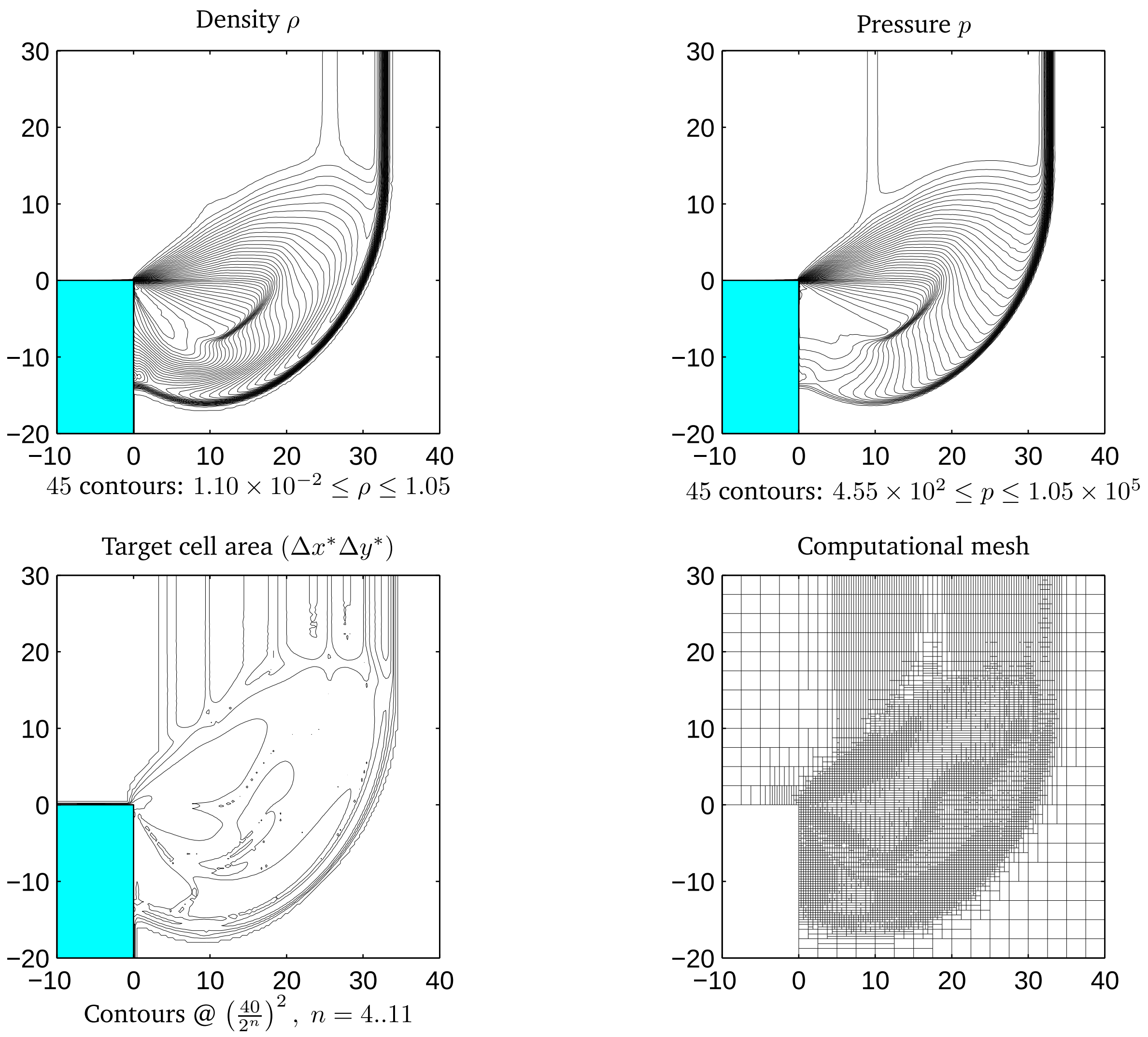


Figure 7.3: Mach 5.09 shock diffraction: pressure and density contours

Accuracy

In general, the contours shown in the low-resolution anisotropic mesh results of Figures 7.1 – 7.3 closely match their uniform mesh counterparts of Figures 5.2 – 5.4. The main deficiency of the adapted mesh results is that the flow features are less well-resolved than those of the uniform mesh. Specifically, the shock and contact waves are slightly more smeared in space. This, however, is to be expected, since the refinement criterion minimizes only an upper bound on the Taylor series truncation error.

Another difference between the two sets of results is the fact that the adapted mesh results show a bunching of contours emanating horizontally from the step corner, out to the top end of the secondary shock wave for the Mach 2.4 and 5.09 flows. This, shown by a dark horizontal line in the

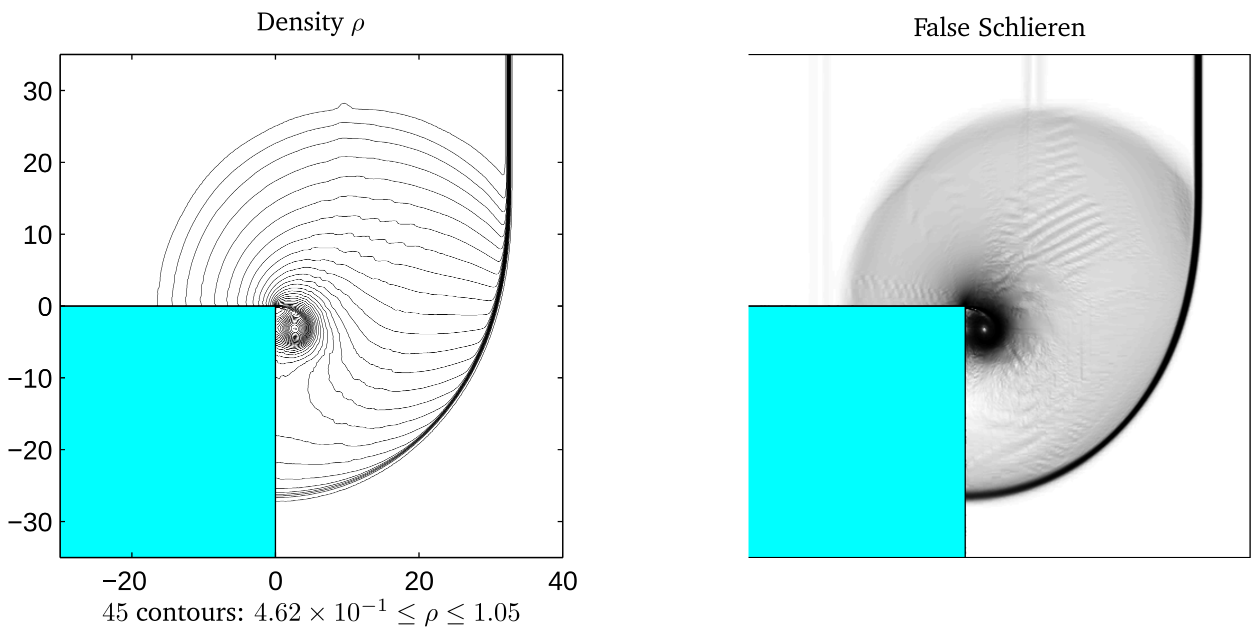


Figure 7.4: Mach 1.3 shock diffraction: density contours and false Schlieren

false Schlieren images of Figures 7.5 and 7.6, is a numerical expansion shock. Wu and Li theorize in [34] that waves travelling across multiple interfaces from coarse to fine cells experience negative numerical dissipation, which in turn can lead to nonphysical expansion shocks and negative density and pressure. Close examination of the computational mesh shows that this juxtaposition of cells does indeed exist along the length of the expansion shock.

The reader may observe that the high-resolution results are significantly more oscillatory than the low-resolution ones; this is an effect of the gradient extrapolations used in the higher-order scheme, combined with the superbee limiter, which is less restrictive than the minmod limiter. Uniform mesh results at an equivalent resolution would have the same oscillatory features. These oscillations serve to illustrate how using a mesh refinement criterion based on the second derivative (rather than the first) would be less effective, since finite difference-based derivatives tend to amplify high-frequency disturbances in the solution.

7.2 Forward Step

The adaptive mesh simulation encounters negative density and pressure if cells at a refinement level higher than $l_{max} = 5$ are used, even with the modified a_{IF} definition described in section 5.2.3. If a manual entropy fix were to be implemented at the step corner as in [33], this would not be the case.

As in the backward step test case, the mesh was adapted using density as the indicator variable for obtaining the target cell sizes (defined in equations 6.4 and 6.5). The stricter value of $\tau = 10^{-5}$ was chosen for the error criterion because of the higher resolution of this simulation over the

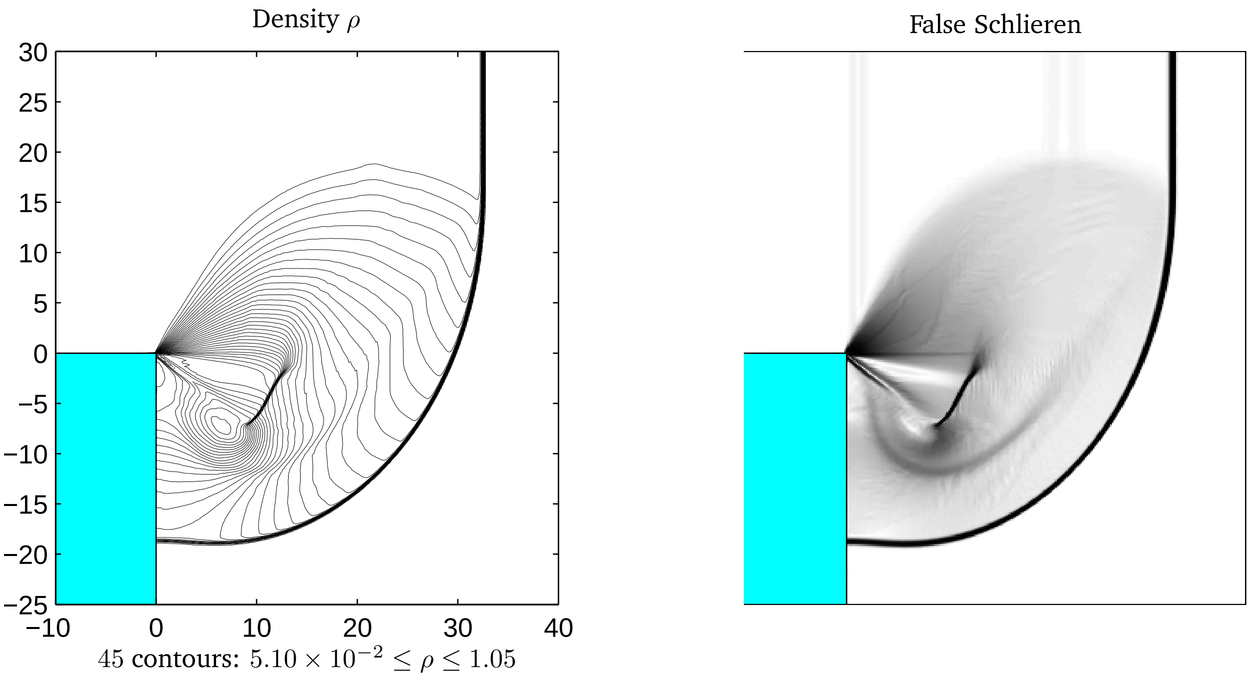


Figure 7.5: Mach 2.4 shock diffraction: density contours and false Schlieren

backward step.

Figure 7.7 shows the computational mesh at different points in time. Most of the shock wave structure is set up in the first two time units of the simulation. Afterwards, the shock waves move relatively slowly toward their position at the end of the simulation.

Density and pressure fields are shown in Figures 7.8 and 7.9, where they are directly compared to those obtained using the uniform mesh. The pressure and density contours are spaced linearly, whereas the difference contours $\{\Delta\rho, \Delta p\}$ are spaced logarithmically.

7.2.1 Analysis of Results

Quality

As in the backward step test case, contours in the adapted mesh solution are not as smooth as in the uniform mesh results. This can only be partially explained by numerical noise in the solution gradients used to extrapolate data onto a uniform mesh. The fact that the simulation does not maintain positivity at higher mesh refinement levels, even though the uniform mesh simulation shown in Figure 5.6 succeeded with $l_{max} = 6$, is an unrelated conundrum. This fact indicates that some combination of the adaptation procedure and the method used to extrapolate cell-centred variables to non-aligned faces causes numerical problems (such as a sudden decrease in artificial dissipation) for the adapted mesh test cases. The exact cause of these numerical problems is unknown; however, when the simulation is repeated using a more dissipative numerical method,

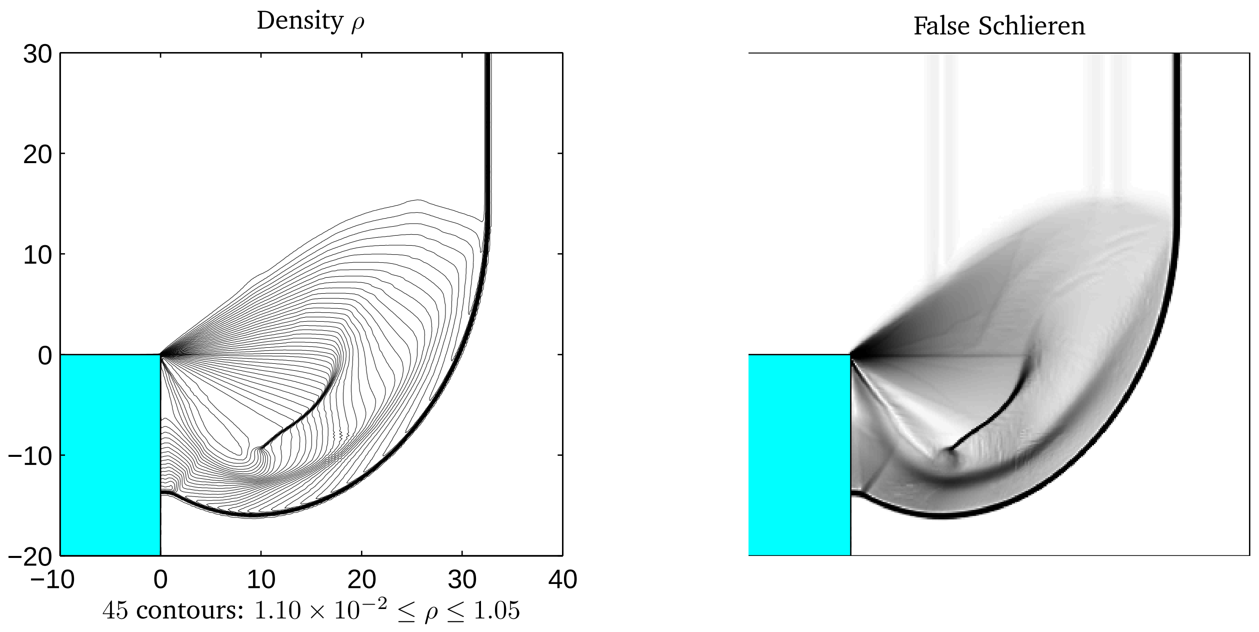


Figure 7.6: Mach 5.09 shock diffraction: density contours and false Schlieren

positivity is maintained. Wu and Li observe in [34] that numerical dissipation has a destabilizing effect when waves move from coarse to fine cells, so numerical problems in the fine cells at the step corner could be exacerbated by the fact that coarse cells exist upstream.

Accuracy

It is apparent from Figures 7.8 and 7.9 that the uniform and adapted mesh solutions differ by at most $\{\Delta\rho,\ \Delta p\} = 1 \times 10^{-1}$ in the majority of the regions outside the shock waves. Although the contours within the shock regions are very tightly packed and correspond closely between the uniform and adapted mesh results, any slight perturbation to one side or the other may be responsible for causing the comparatively large solution differences on the order of $\Delta\rho \approx 10$.

As an indication of the sensitivity of this test case to the numerical method, observe that the location of the shock reflection at the top of the domain (at $x \approx 1.8$) is sightly different between the uniform and adapted mesh results. At this point, however, the shock is relatively weak (see Colella's plot of entropy in Figure 4.10) and therefore so is the solution difference.

One interesting implication of the results of Figure 7.8 is the fact that there are large areas of the solution where the differences $\Delta\rho$ exceed $\tau = 10^{-5}$. This is because the 'optimal' target cell sizes optimize cell size based on only one single, relatively simple indicator of the solution error, whereas in reality the composition of the solution error is more complicated than the gradient term of the Taylor series expansion. It is already apparent from its oscillatory behaviour and its difficulty with the step corner expansion that the uniform mesh result is non-ideal. In addition, the very act of anisotropic mesh refinement introduces errors into the solution through averaging and gradient extrapolation – errors that are partially indicated by rougher density and pressure contours.

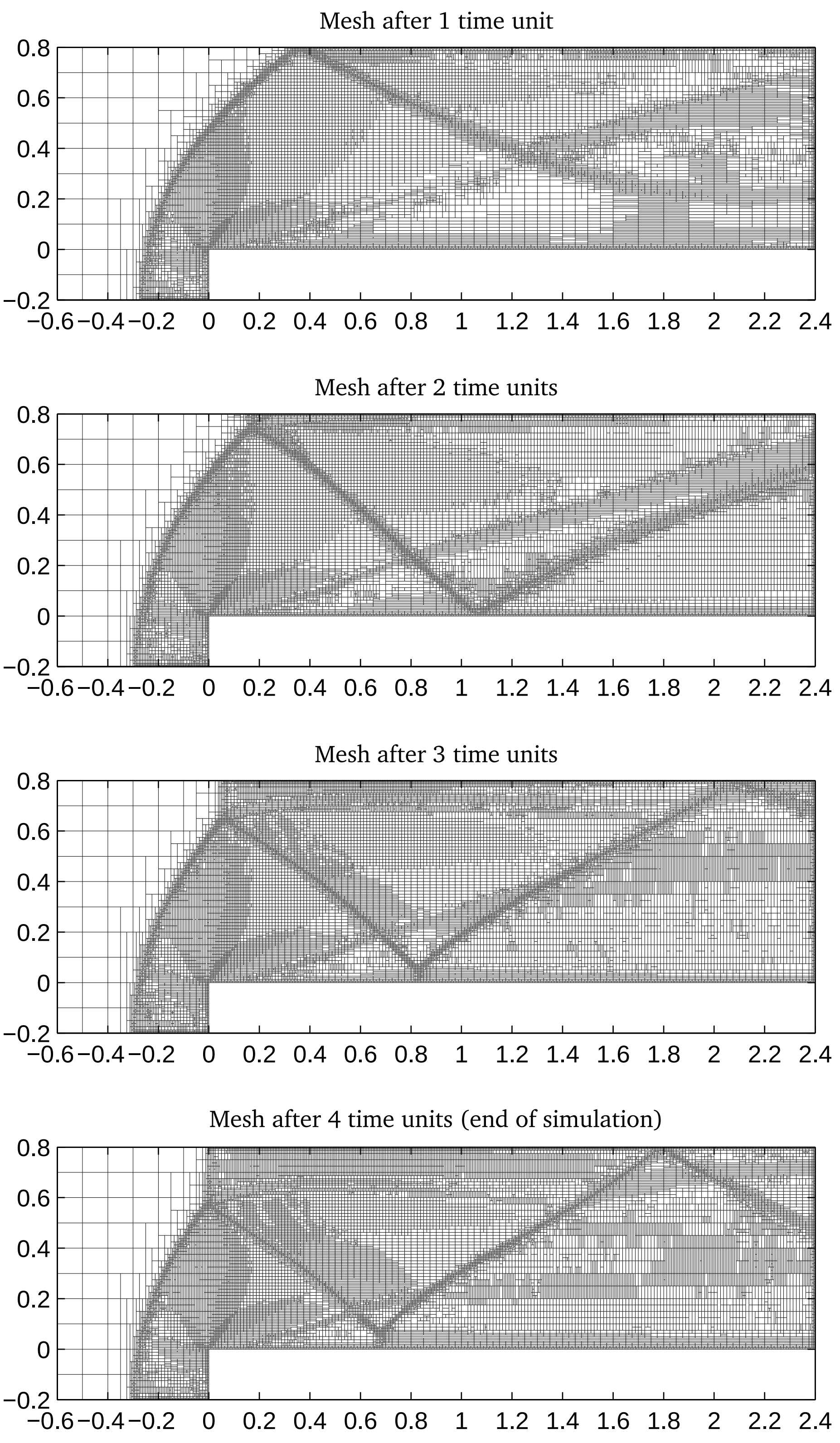


Figure 7.7: Mach 3.0 flow over a forward step: computational mesh

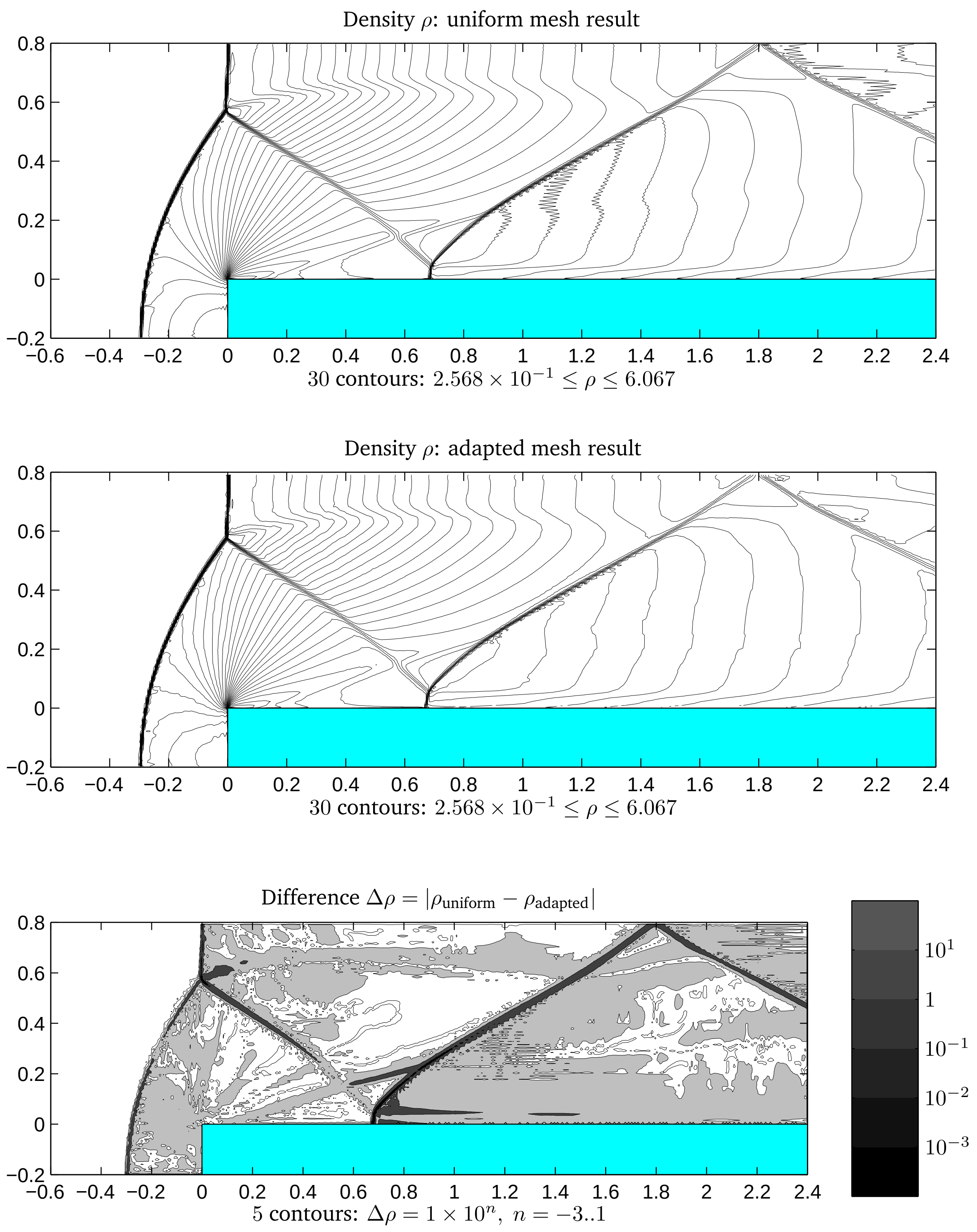


Figure 7.8: Mach 3.0 flow over a forward step: comparison of density field between uniform and adapted mesh results

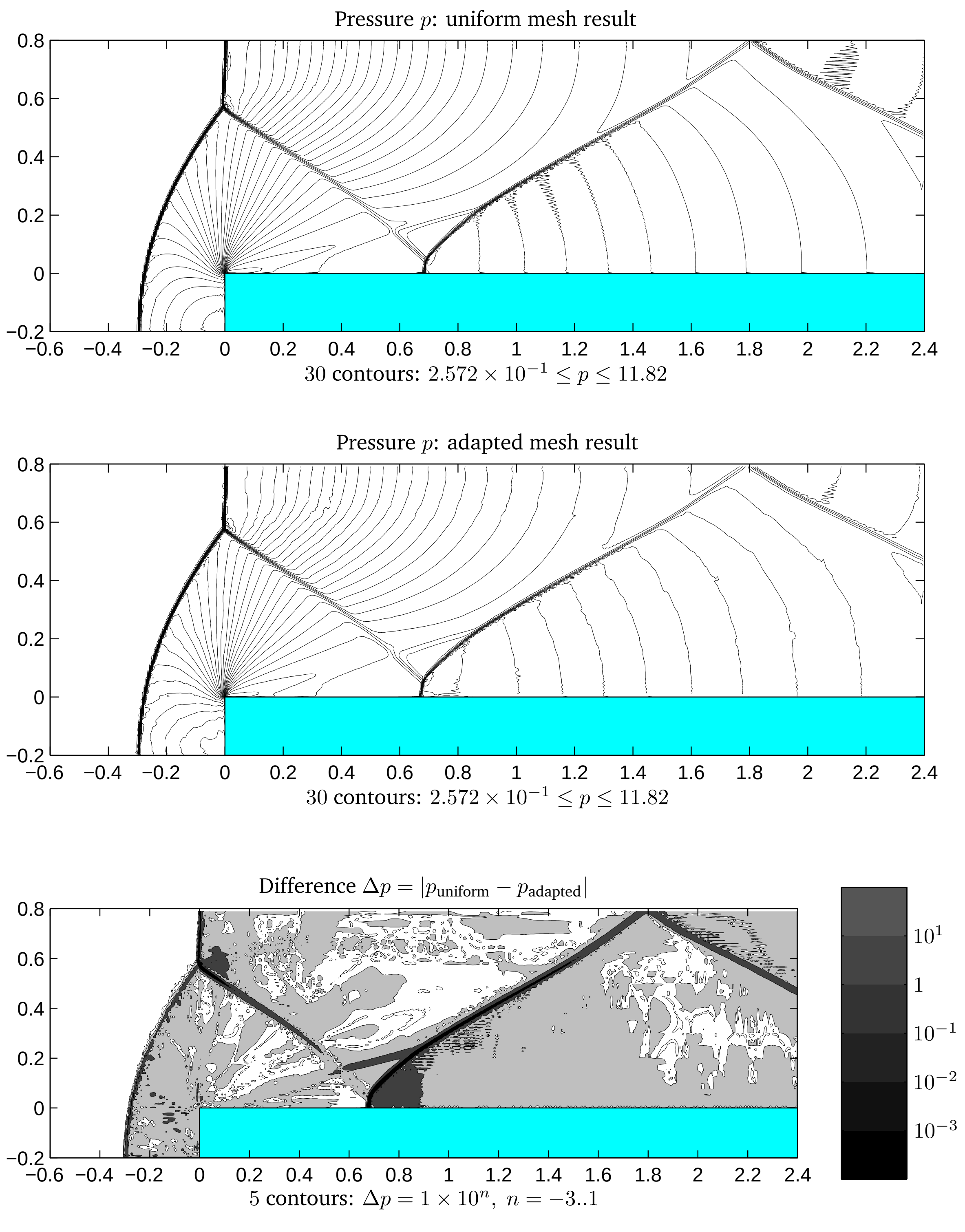


Figure 7.9: Mach 3.0 flow over a forward step: comparison of pressure field between uniform and adapted mesh results

Chapter 8

Concluding Remarks

8.1 Computational Resource Savings

One aim of using the anisotropic mesh adaptation technique was to yield computational resource savings over uniform and equivalent isotropically adapted meshes. Since the forward step test case generally consumes the most resources, and its major flow features lie at angles offset from the cardinal directions, it was chosen as a worst-case scenario for the calculation of resource savings. Figure 8.1 quantifies the time and memory savings achieved over a uniform mesh, while Figure 8.2 quantifies the memory savings achieved over an equivalent isotropically adapted mesh. Processor time savings were tabulated and plotted according to the following expression:

$$\frac{P_{\text{anisotropic}}}{P_{\text{uniform}}} = \frac{\Delta t_{n,\text{ anisotropic}}}{\Delta t_{n,\text{ uniform}}} \tag{8.1}$$

where Δt_n is the real-world time required to reach time step n within the simulation. Note that there is no valid execution time data at time zero. Memory savings were calculated using a similar expression:

$$\frac{M_{\text{anisotropic}}}{M_{\text{uniform}}} = \frac{m_{n,\text{ anisotropic}}}{m_{n,\text{ uniform}}} \tag{8.2}$$

where $m_n = (\text{no. faces} \times m_{\text{face}}) + (\text{no. cells} \times m_{\text{cell}})$. For the present code, $m_{\text{face}} = 236$ bytes, and $m_{\text{cell}} = 352$ bytes. These quantities represent the amount of memory required by `Face` and `Cell` objects, and depend on what extra storage is used for convenience within those classes. Obviously, the memory savings would change if different values were used for m_{cell} and m_{face}.

One interesting feature of Figure 8.1 is the fact that the processing time savings are not proportional to the memory savings. For a uniform mesh, the required processing time is directly proportional to the memory used, since the same number of operations must be performed on each `Cell` and `Face` object. However, there are two key differences between adaptive and uniform meshes that explain the difference in the processing time to memory usage ratio:

1. The ratio of the number of cells to the number of faces is not necessarily the same for adapted

and uniform meshes. For the forward step test cases, the ratios were found to be:

$$\left(\frac{\text{no. faces}}{\text{no. cells}}\right)_{\text{uniform}} \approx 2.007, \quad \left(\frac{\text{no. faces}}{\text{no. cells}}\right)_{\text{adapted}} \approx 2.165 \tag{8.3}$$

Without going into the details of the code implementation, this discrepancy between `Cell`/`Face` ratios affects the computational time since different operations are performed on `Cells` and `Faces`.

2. The act of mesh refinement generally increases the amount of processor time required. The uniform mesh obviously does not require this functionality.

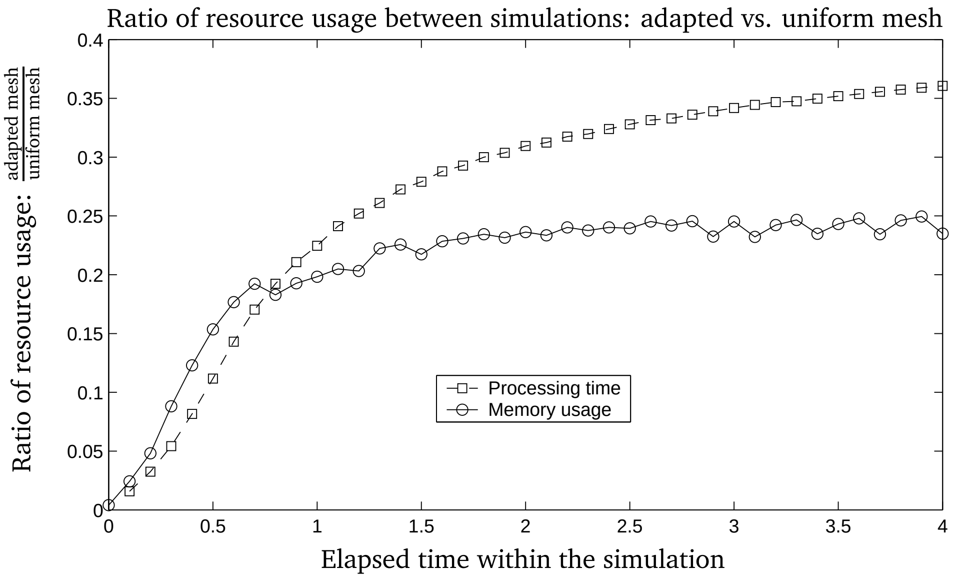


Figure 8.1: Memory and CPU time savings provided by anisotropic mesh adaptation for the forward step test case

For the memory usage comparison of Figure 8.2, the isotropic mesh memory usage was calculated based on the assumption that all of the anisotropic cells in the mesh would be subdivided in the appropriate direction (either $x-$refined or $y-$refined) until their aspect ratios were equal to the equivalent isotropically refined cells. Surprisingly for a flow whose major features lie at an angle to both of the cardinal directions, the anisotropic mesh offers significant memory savings over the equivalent isotropic mesh.

Although the anisotropic code has the ability to refine isotropically, it is not instructive to compare the processor time required for the anisotropic vs. isotropic meshes. The isotropic mesh data structures defined in [35] are operated upon differently than the list used in the present code, and have even been vectorized (see [26]) with the probable result that they are computationally more efficient. It is difficult, however, to quantify this efficiency without setting up a test case and running the code on the same platform that was used for the present anisotropic code.

Table 8.1 outlines the percentage of processor time spent on the main computational tasks. The

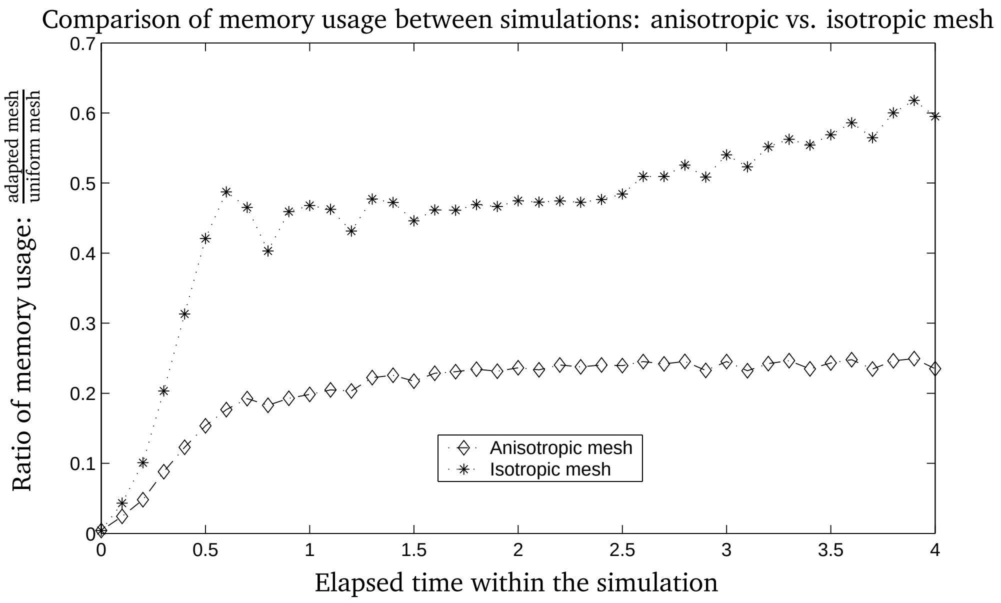


Figure 8.2: Memory savings provided by anisotropic over isotropic adaptation for the forward step test case

	Code	
Task	Uniform	Adaptive
Refine cells	0%	23%
Coarsen cells	0%	2%
Calculate AUSM$^+$ fluxes	25%	15%
Step forward in time	19%	26%
Calculate $\{\rho, p, \vec{u}\}$ from $\mathbf{u}$	3%	2%
Other tasks	53%	32%

Table 8.1: Division of processor time among computational tasks: forward step test case

category 'other tasks' is primarily composed of: a) applying the gradient limiter; b) calculating cell-centred and face-based gradients and c) solution file output. However, some of these tasks are used by the refinement and coarsening procedures, and so to avoid 'double-accounting', only the major tasks are shown. It is interesting to note that the adaptive simulation consumes more processing time when stepping forward in time. Code execution time profiles indicate that this is due to the requirement that the each `Cell` search lists of neighbouring `Face` objects in order to add their flux contribution.

8.2 Future Direction

In order for the anisotropic Cartesian adaptation technique to be effective for a wider range of supersonic flows and geometries, the areas outlined below should be examined in further detail and implemented in code wherever possible.

Cut-cell method: This straightforward improvement has already been implemented in similar codes by de Zeeuw and Powell [35] and Sun [26]. The cut-cell method allows for improved resolution of arbitrary boundaries by approximating them with a line (or plane) that 'cuts' the computational cells. Boundary conditions are implemented by setting velocity normal to the 'cut' equal to zero and interpolating the boundary pressure in the same way as described in section 3.2.1.

Extension to three dimensions: The data structure and algorithms were extended by Ham et al. in [6] to three dimensions. Therefore, extension of the present code is technically feasible. In three dimensions, however, keeping track of adjacent faces in the code becomes complicated, and therefore it is worth examining the present code for possible simplifications before taking on this task.

Solution quality improvement: It is well documented that waves travelling between cells of different aspect ratios experience distortion ([10], [34]) and so the algorithms in this thesis are stricter than the original ones of Ham et al. in that they place greater limits on the differences in aspect ratio between adjacent cells. Wu and Li have proposed that such differences can adversely affect the level of numerical dissipation present, which would suggest that the numerical methods should take neighbouring cells' aspect ratios into account when determining fluxes.

Improvement in the quality of the solution would also serve to provide a better estimate of higher-order derivatives, opening up the possibility of using them in alternate mesh refinement criteria.

Appendix A

Derivations

A.1 Least-Squares Gradient Computation

This technique relies on the minimization of weighted residuals generated by a finite-difference approximation to the solution gradient, $[\tilde{\phi_x}\ ,\ \tilde{\phi_y}]^T$. The residuals associated with cell i can be expressed as:

$$r_i(\tilde{\phi_x}\ ,\ \tilde{\phi_y}) = \sum_{all\ m} w_m \left(\begin{bmatrix} (\tilde{\phi_x})_i \\ (\tilde{\phi_y})_i \end{bmatrix} \cdot \mathbf{s}_m - [(\phi)_m - (\phi)_i] \right)^2 \quad , \tag{A.1}$$

where s_m is the vector from the centroid of cell i to the centroid of neighbouring cell m. In this work, the weight w_m is taken as unity, but could be chosen differently, depending on the nature of the solution.

By setting the partial derivatives of the residual equal to zero, we obtain a system of equations that must be solved for the unknowns $[(\tilde{\phi_x})_i\ ,\ (\tilde{\phi_y})_i]^T$.

Assigning for convenience

$$\begin{aligned}
\Delta x &= x_m - x_i \\
\Delta y &= y_m - y_i \\
\Delta \phi &= \phi_m - \phi_i \\
WXX &= \sum_m w_m \Delta x \Delta x \\
WXY &= \sum_m w_m \Delta x \Delta y \\
WYY &= \sum_m w_m \Delta y \Delta y \\
WX\phi &= \sum_m w_m \Delta x \Delta \phi \\
WY\phi &= \sum_m w_m \Delta y \Delta \phi,
\end{aligned}$$

the system becomes:

$$\begin{bmatrix} WXX & WXY \\ WXY & WYY \end{bmatrix} \begin{bmatrix} (\tilde{\phi_x})_i \\ (\tilde{\phi_y})_i \end{bmatrix} = \begin{bmatrix} WX\phi \\ WY\phi \end{bmatrix} \tag{A.2}$$

The closed-form solution to the system of equations is

$$(\tilde{\phi_x})_i = \frac{(WY\phi)(WXY) - (WX\phi)(WYY)}{(WXY)(WXY) - (WXX)(WYY)} \tag{A.3}$$

$$(\tilde{\phi_y})_i = \frac{(WX\phi)(WXY) - (WY\phi)(WXX)}{(WXY)(WXY) - (WXX)(WYY)} \tag{A.4}$$

A.2 Optimal Cell Size Criterion

Consider the Taylor series expansion of $\phi(\mathbf{x})$ about $\mathbf{x}_\circ$, the cell centre:

$$\phi(\mathbf{x}) = \phi(\mathbf{x}_\circ) + [\nabla\phi(\mathbf{x})]^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T[\nabla^2\phi(\mathbf{x})]\mathbf{x} + O(\mathbf{x}^3) \tag{A.5}$$

For a first-order accurate numerical scheme, the leading error term is $[\nabla\phi(\mathbf{x})]^T\mathbf{x}$. By assuming that the maximum absolute error occurs at the farthest point from the cell centre, in two dimensions the expression becomes $\frac{\Delta x}{2}\phi_x + \frac{\Delta y}{2}\phi_y$.

Integrating this term over the cell provides an expression for the L_∞ error:

$$\epsilon_{\phi,L_\infty} = \int_{-\frac{\Delta x}{2}}^{\frac{\Delta x}{2}} \int_{-\frac{\Delta y}{2}}^{\frac{\Delta y}{2}} \frac{\Delta x}{2}\phi_x + \frac{\Delta y}{2}\phi_y \; dy \; dx = \frac{\Delta x^2 \Delta y}{4}\phi_x + \frac{\Delta x \Delta y^2}{4}\phi_y \tag{A.6}$$

We are now in a position to formulate the optimization problem:

$$\begin{aligned} &\text{maximize} && \Delta x \Delta y && \text{(A.7)} \\ &\text{subject to} && \epsilon_{\phi,L_\infty} \leq \tau && \text{(A.8)} \\ &&& \Delta x \geq 0 && \text{(A.9)} \\ &&& \Delta y \geq 0, && \text{(A.10)} \end{aligned}$$

where τ is a user-specified error tolerance. For ϕ a single variable such as density, this problem can be solved analytically using Lagrange multipliers. Following the theory of optimization using Lagrange multipliers as outlined in [19], we formulate and minimize the following Lagrange function:

$$\mathcal{L}(\Delta x, \Delta y, \lambda) = -\Delta x \Delta y - \lambda\left(\tau - \frac{\Delta x^2 \Delta y}{4}\phi_x - \frac{\Delta x \Delta y^2}{4}\phi_y\right) \tag{A.11}$$

Setting the derivative of $\mathcal{L}$ with respect to $\mathbf{x} = [\Delta x, \Delta y]^T$ equal to zero yields the system of equations:

$$\nabla_{\mathbf{x}}\mathcal{L} = \begin{bmatrix} -\Delta y^* + \lambda^* \frac{\Delta x^* \Delta y^*}{2}\phi_x + \lambda^* \frac{\Delta y^{*2}}{4}\phi_y \\ -\Delta x^* + \lambda^* \frac{\Delta x^{*2}}{4}\phi_x + \lambda^* \frac{\Delta x^* \Delta y^*}{2}\phi_y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} , \tag{A.12}$$

where Δx^*, Δy^* and λ^* are optimal variable values. This system can be reduced to:

$$\lambda^* \Delta x^* \phi_x = \lambda^* \Delta y^* \phi_y \tag{A.13}$$

It is safe to assume that the *strict complementarity* condition applies; in other words, constraint (A.8) is active (equality holds) at the optimal solution, $\mathbf{x} = [\Delta x^*, \Delta y^*]^T$; thus, $\lambda^* \neq 0$ and it can be divided out of the equation. The remaining expression,

$$\Delta x^* \phi_x = \Delta y^* \phi_y \ , \tag{A.14}$$

provides the value of Δx^* in terms of Δy^*; by using this in the *equality* constraint corresponding to (A.8), we obtain the following expressions for the optimal target cell dimensions:

$$\Delta x^* \ = \left(\tfrac{2\tau\phi_y}{\phi_x^2}\right)^{\frac{1}{3}} \tag{A.15}$$

$$\Delta y^* \ = \left(\tfrac{2\tau\phi_x}{\phi_y^2}\right)^{\frac{1}{3}} \tag{A.16}$$

Bibliography

[1] J. D. Anderson. *Modern Compressible Flow*. McGraw-Hill, New York, 1990.

[2] T. V. Bazhenova, L. G. Gvozdeva, and M. A. Nettleton. Unsteady interactions of shock waves. *Prog. Aerospace Sci.*, 21:249–331, 1984.

[3] R. Johnson E. Gamma, R. Helm and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Longman, Reading, Massachusetts, 1995.

[4] A. F. Emery. An evaluation of several differencing methods for inviscid fluid flow problems. *Journal of Computational Physics*, 2:306–331, 1968.

[5] J. Gressier, P. Villedieu, and J-M. Moschetta. Positivity of flux vector splitting schemes. *Journal of Computational Physics*, 155:199–220, 1999.

[6] F. Ham, F. S. Lien, and A. B. Strong. A Cartesian grid method with transient anisotropic adaptation. *Journal of Computational Physics*, 179:469–494, 2002.

[7] R. Hillier. Computation of shock wave diffraction at a ninety degrees convex edge. *Shock Waves*, 1:89–98, 1991.

[8] J. E. A. John. *Gas Dynamics*. Prentice-Hall, Upper Saddle River, 1984.

[9] C. B. Laney. *Computational Gasdynamics*. Cambridge University Press, Cambridge, 1998.

[10] N. J. Larsson. *Computational Aero Acoustics for Vehicle Applications*. Licentiate thesis, Chalmers University of Technology, 2002.

[11] R. C. Lee and W. M. Tepfenhart. *UML and C++: A Practical Guide to Object-Oriented Software Development*. Prentice-Hall, Upper Saddle River, 1997.

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

[13] F-S. Lien. A pressure-based unstructured-grid method for all-speed flows. *International Journal for Numerical Methods in Fluids*, 33:355–374, 2000.

[14] M-S. Liou. A Sequel to AUSM: AUSM$^+$. *Journal of Computational Physics*, 129:364–382, 1996.

[15] M-S. Liou. A Further Development of the AUSM$^+$ Scheme Towards Robust and Accurate Solutions for All Speeds. In *Proceedings of the 16th AIAA Computational Fluid Dynamics Conference*. AIAA Paper 2003-4116, 2003.

[16] M-S. Liou and C.J. Steffen. A new flux splitting scheme. *Journal of Computational Physics*, 107:23–29, 1993.

[17] M-S. Liou, B. van Leer, and J-S. Shuen. Splitting of inviscid fluxes for real gases. *Journal of Computational Physics*, 87:1–24, 1990.

[18] M. Meinke, W. Schröder, E. Krause, and Th. Rister. A comparison of second- and sixth-order methods for large-eddy simulations. *Computers and Fluids*, 31:695–718, 2001.

[19] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[20] C. F. Ollivier-Gooch. Quasi-ENO schemes for unstructured meshes based on unlimited data-dependent least-squares reconstruction. *Journal of Computational Physics*, 133(1):6–17, 1997.

[21] J. J. Quirk. A contribution to the great Riemann solver debate. *International Journal for Numerical Methods in Fluids,* 18:555–574, 1994.

[22] R. C. Ripley. Unstructured-grid methods for numerical simulation of shock wave interactions with bodies and boundary layers. Master's thesis, University of Waterloo, 2002.

[23] B. W. Skews. The perturbed region behind a diffracting shock wave. *J. Fluid Mech.*, 29(4):705–719, 1967.

[24] J. L. Steger and R. F. Warming. Flux vector splitting of the inviscid gasdynamic equations with application to finite-difference methods. *Journal of Computational Physics,* 40:263–293, 1981.

[25] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Boston, 3rd edition, 1997.

[26] M. Sun. *Numerical and Experimental Studies of Shock Wave Interaction with Bodies*. PhD thesis, Tohoku University, 1998.

[27] M. Sun and K. Takayama. The formation of a secondary shock wave behind a shock wave diffracting at a convex corner. *Shock Waves*, 7:287–295, 1997.

[28] K. Takayama and O. Inoue. Shock wave diffraction over a 90 degree sharp corner. *Shock Waves*, 1:301–312, 1991.

[29] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer-Verlag, Berlin, 1999.

[30] M. van Dyke. *An Album of Fluid Motion*. Parabolic Press, Stanford, 1997.

[31] B. van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to Godunov's method. *Journal of Computational Physics,* 32:101–136, 1979.

[32] H. K. Versteeg and W. Malalasekera. *Introduction to Computational Fluid Dynamics*. Prentice-Hall, address, 1995.

[33] P. Woodward and P. Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics*, 54:115–173, 1984.

[34] Z-N. Wu and K. Li. Anisotropic Cartesian grid method for steady inviscid shocked flow computation. *International Journal for Numerical Methods in Fluids*, 41:1053–1084, 2003.

[35] D. De Zeeuw and K. G. Powell. An adaptively refined Cartesian mesh solver for the Euler equations. *Journal of Computational Physics*, 104:56–68, 1993.