A QUADTREE-BASED
ADAPTIVELY-REFINED
CARTESIAN-GRID ALGORITHM FOR
SOLUTION OF THE EULER
EQUATIONS

by
Darren L. De Zeeuw

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering and Scientific Computing)
in The University of Michigan
1993

Doctoral Committee:
Associate Professor Kenneth G. Powell, Chairperson
Professor Tamas Gombosi
Professor Philip L. Roe
Professor Bram van Leer
I would like to dedicate this dissertation to my loving wife Sue. She has supported me through both the ups and downs of my studies.
I wish to thank my chairperson, Ken Powell, for all of his guidance throughout this research. He has always been available when needed, and has been a pleasure to work with. I would also like to thank the other members of my committee, Bram van Leer, Phil Roe, and Tamas Gombosi, for their contributions to my research.

This work has been funded in part by the National Science Foundation (monitored by Dr. George Lea) and NASA Langley Research Center (monitored by Dr. James Thomas).
TABLE OF CONTENTS

DEDICATION ................................................................. ii
ACKNOWLEDGEMENTS ......................................................... iii
LIST OF FIGURES ........................................................... vii
LIST OF TABLES .............................................................. xiii

I. INTRODUCTION ............................................................. 1

1.1 Structured Grid Methods ................................................. 3
  1.1.1 Grid Generation .................................................... 4
  1.1.2 Grid Adaptation .................................................... 5

1.2 Unstructured Grid Methods ............................................. 5
  1.2.1 Grid Generation .................................................... 6
  1.2.2 Grid Adaptation .................................................... 8

1.3 Cartesian Grid Methods ................................................. 8

1.4 A New Approach - The Adaptive Cartesian Grid Method ............ 10

CHAPTER

II. DATA STRUCTURE AND GRID GENERATION .......................... 13

2.1 Quadtree Data Structure ............................................... 13

2.2 Generating The Initial Grid ........................................... 20

2.3 Geometry Adaptation .................................................. 23

2.4 Solution Adaptation .................................................... 25
  2.4.1 Total-Velocity Difference ......................................... 27
  2.4.2 Curl and Divergence of Velocity ................................ 28
  2.4.3 Comparison of Adaptation Criteria ................................. 29

2.5 Grid “Smoothing” ....................................................... 39

III. FLOW SOLVER ........................................................... 47

3.1 Reconstruction .......................................................... 49
  3.1.1 Path Integral ....................................................... 50
3.1.2 Least Squares ........................................... 52
3.1.3 Limiting ..................................................... 54
3.2 Flux Formulation .............................................. 58
  3.2.1 Euler Equations ........................................ 58
  3.2.2 Roe’s Flux-Difference Splitting ......................... 60
  3.2.3 Van Leer’s Flux Splitting ................................ 62
3.3 Time Stepping ................................................ 62
  3.3.1 Multi-Stage Time Stepping .............................. 62
  3.3.2 Multigrid Convergence Acceleration .................... 64
3.4 Boundary Procedures ......................................... 69
  3.4.1 Far-Field Boundary Conditions ......................... 69
  3.4.2 Body-Cut Boundary Conditions .......................... 70
3.5 Post-Processing .............................................. 71
3.6 Validation .................................................... 73
  3.6.1 Small Cut Cells ......................................... 73
  3.6.2 Multigrid Grid Problems ................................ 74
  3.6.3 Multigrid Convergence Study ........................... 74
  3.6.4 Boundary Condition Study .............................. 79
  3.6.5 Order of Accuracy Study ................................ 82
    3.6.5.1 Subsonic Non-Lifting Airfoil .................... 82
    3.6.5.2 Subsonic Cylinder .................................. 84
    3.6.5.3 Subsonic Ellipses At Angle Of Attack ............ 85

IV. RESULTS ......................................................... 90
  4.1 NACA 0012 Airfoil .......................................... 90
    4.1.1 $M_{\infty} = 0.63, \alpha = 2^\circ$ ....................... 90
    4.1.2 $M_{\infty} = 0.85, \alpha = 1$ .......................... 91
    4.1.3 $M_{\infty} = 0.95, \alpha = 0$ ......................... 94
  4.2 $15^\circ$ Wedge .............................................. 99
  4.3 Axisymmetric Jet ............................................ 99
    4.3.1 $M = 1.25, P_0/\rho_0 = 20$ .......................... 103
    4.3.2 $M = 1.25, P_0/\rho_0 = 5$ .......................... 106
    4.3.3 $M = 1.25, P_0/\rho_0 = 50$ ......................... 106
    4.3.4 $M = 1.25, P_0/\rho_0 = 100$ ....................... 108
  4.4 Multi-Element Airfoils ...................................... 108
    4.4.1 Three-Element Airfoil ............................... 110
    4.4.2 Two-Element Airfoil .................................. 114
    4.4.3 Four-Element Airfoil .................................. 116

V. CONCLUSIONS .................................................... 123
  5.1 Summary .................................................... 123
  5.2 Conclusions .................................................. 124
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Example of Physical to Computational Space Mapping of Structured Grid</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Refinement of a Structured Grid</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>Destructuring of a Structured Grid</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>Grid for Leading Edge of 3-Element Airfoil</td>
<td>9</td>
</tr>
<tr>
<td>1.5</td>
<td>Grid for Leading Edge of 3-Element Airfoil with Curvature Adaptation</td>
<td>10</td>
</tr>
<tr>
<td>2.1</td>
<td>Quadtree Structure</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Tree Paths for Various North Neighbors</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Various Cut-Cell Cell Types</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>Minimum Grid Resolution</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>Example Grid – All-Cell Adaptation</td>
<td>24</td>
</tr>
<tr>
<td>2.6</td>
<td>Example Grid – Cut-Cell Adaptation</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>Example Grid – Curvature-Cell Adaptation</td>
<td>25</td>
</tr>
<tr>
<td>2.8</td>
<td>Thresholds for Adaptation: Refinement and Coarsening</td>
<td>28</td>
</tr>
<tr>
<td>2.9</td>
<td>Initial Grid</td>
<td>30</td>
</tr>
<tr>
<td>2.10</td>
<td>Initial Grid Mach Number Contours</td>
<td>30</td>
</tr>
<tr>
<td>2.11</td>
<td>Cells Flagged for Refinement by Total-Velocity Refinement, $d_i &gt; \sigma$</td>
<td>31</td>
</tr>
<tr>
<td>2.12</td>
<td>Cells Flagged for Refinement by Total-Velocity Adaptation, $d_i &gt; \frac{1}{2} \sigma$</td>
<td>32</td>
</tr>
</tbody>
</table>
2.13 Cells Flagged for Coarsening by Total-Velocity Adaptation, $d_i < \frac{1}{10} \sigma$. 32
2.14 Cells Flagged for Refinement by Curl of Velocity, $\tau_{c_i} > \sigma_c$. 33
2.15 Cells Flagged for Refinement by Divergence of Velocity, $\tau_{d_i} > \sigma_d$. 34
2.16 Cells Flagged for Refinement by Curl and Divergence of Velocity, $\tau_{c_i} > \sigma_c$ or $\tau_{d_i} > \sigma_d$. 34
2.17 Cells Flagged for Coarsening by Curl and Divergence of Velocity, $\tau_{c_i} < \frac{1}{10} \sigma_c$ and $\tau_{d_i} < \frac{1}{10} \sigma_d$. 35
2.18 Final Grid 35
2.19 Initial Grid 36
2.20 Initial Grid Mach Number Contours 37
2.21 Cells Flagged for Refinement by Total-Velocity Refinement, $d_i > \sigma$. 37
2.22 Cells Flagged for Refinement by Curl of Velocity, $\tau_{c_i} > \sigma_c$. 38
2.23 Cells Flagged for Refinement by Divergence of Velocity, $\tau_{d_i} > \sigma_d$. 38
2.24 Cells Flagged for Refinement by Curl and Divergence of Velocity, $\tau_{c_i} > \sigma_c$ or $\tau_{d_i} > \sigma_d$. 39
2.25 Final Grid 40
2.26 Features Which Violate The Data Structure 41
2.27 Features Which Complicate The Data Structure 42
2.28 Features Which Degrade The Solution Accuracy 43
2.29 Grid and Mach Contours With Accuracy And Aesthetic Features Smoothed 44
2.30 Grid and Mach Contours Without Accuracy And Aesthetic Features Smoothed 45
3.1 The Riemann Problem 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Linear Reconstruction for Higher Order $U_L$ and $U_R$</td>
<td>48</td>
</tr>
<tr>
<td>3.3</td>
<td>“Normal” Path</td>
<td>50</td>
</tr>
<tr>
<td>3.4</td>
<td>“Altered” Paths</td>
<td>51</td>
</tr>
<tr>
<td>3.5</td>
<td>Sample Numbering</td>
<td>54</td>
</tr>
<tr>
<td>3.6</td>
<td>Grid Spacing vs. Gradient Errors</td>
<td>55</td>
</tr>
<tr>
<td>3.7</td>
<td>Mach Contours — Transonic Airfoil</td>
<td>57</td>
</tr>
<tr>
<td>3.8</td>
<td>Limiter Values — Transonic Airfoil</td>
<td>57</td>
</tr>
<tr>
<td>3.9</td>
<td>1-Dimensional Multigrid Example</td>
<td>65</td>
</tr>
<tr>
<td>3.10</td>
<td>Multigrid Saw-Tooth Cycle</td>
<td>67</td>
</tr>
<tr>
<td>3.11</td>
<td>Body Face State Reflection</td>
<td>70</td>
</tr>
<tr>
<td>3.12</td>
<td>Example of Periodic Boundary Condition</td>
<td>72</td>
</tr>
<tr>
<td>3.13</td>
<td>Obtaining Post-Processed Nodal Values</td>
<td>72</td>
</tr>
<tr>
<td>3.14</td>
<td>Multigrid Cone of Influence</td>
<td>75</td>
</tr>
<tr>
<td>3.15</td>
<td>Subsonic Convergence History Based on Multigrid Cycles</td>
<td>76</td>
</tr>
<tr>
<td>3.16</td>
<td>Subsonic Convergence History Based on Residual Calculations</td>
<td>77</td>
</tr>
<tr>
<td>3.17</td>
<td>Transonic Convergence History Based on Multigrid Cycles</td>
<td>77</td>
</tr>
<tr>
<td>3.18</td>
<td>Transonic Convergence History Based on Residual Calculations</td>
<td>77</td>
</tr>
<tr>
<td>3.19</td>
<td>Convergence History Based on Multigrid Cycles</td>
<td>78</td>
</tr>
<tr>
<td>3.20</td>
<td>Convergence History Based on Residual Calculations</td>
<td>79</td>
</tr>
<tr>
<td>3.21</td>
<td>Convergence History Based on Run Time</td>
<td>80</td>
</tr>
<tr>
<td>3.22</td>
<td>Lift, Drag, and Number of Cells vs. Boundary Radius</td>
<td>80</td>
</tr>
<tr>
<td>3.23</td>
<td>Grid for Periodic NACA 0012 Case</td>
<td>82</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>4.13</td>
<td>Normal Shock Location for NACA 0012 at $M_\infty = 0.95, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.14</td>
<td>Grid for $15^\circ$ Wedge at $M_\infty = 2.0$</td>
<td></td>
</tr>
<tr>
<td>4.15</td>
<td>Mach Contours for $15^\circ$ Wedge at $M_\infty = 2.0$</td>
<td></td>
</tr>
<tr>
<td>4.16</td>
<td>Pressure Contours for $15^\circ$ Wedge at $M_\infty = 2.0$</td>
<td></td>
</tr>
<tr>
<td>4.17</td>
<td>Wall Pressure for $15^\circ$ Wedge at $M_\infty = 2.0$</td>
<td></td>
</tr>
<tr>
<td>4.18</td>
<td>Schematic of an Under-Expanded Jet with Mach Disk</td>
<td></td>
</tr>
<tr>
<td>4.19</td>
<td>Grid for Jet at $M = 1.25, P_{0j}/P_{0s} = 20$</td>
<td></td>
</tr>
<tr>
<td>4.20</td>
<td>Mach Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 20$</td>
<td></td>
</tr>
<tr>
<td>4.21</td>
<td>Density Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 20$</td>
<td></td>
</tr>
<tr>
<td>4.22</td>
<td>Pressure Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 20$</td>
<td></td>
</tr>
<tr>
<td>4.23</td>
<td>Grid for Jet at $M = 1.25, P_{0j}/P_{0s} = 5$</td>
<td></td>
</tr>
<tr>
<td>4.24</td>
<td>Mach Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 5$</td>
<td></td>
</tr>
<tr>
<td>4.25</td>
<td>Grid for Jet at $M = 1.25, P_{0j}/P_{0s} = 50$</td>
<td></td>
</tr>
<tr>
<td>4.26</td>
<td>Mach Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 50$</td>
<td></td>
</tr>
<tr>
<td>4.27</td>
<td>Grid for Jet at $M = 1.25, P_{0j}/P_{0s} = 100$</td>
<td></td>
</tr>
<tr>
<td>4.28</td>
<td>Mach Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 100$</td>
<td></td>
</tr>
<tr>
<td>4.29</td>
<td>Grid (1) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.30</td>
<td>Grid (2) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.31</td>
<td>Grid (3) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.32</td>
<td>Mach Contours (1) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.33</td>
<td>Mach Contours (2) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.34</td>
<td>Mach Contours (3) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>4.35</td>
<td>Streamlines for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.36</td>
<td>Mach Number on Body for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.37</td>
<td>Pressure Coefficient on Body for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.38</td>
<td>$1 - p_0/p_{\infty}$ on Body for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.39</td>
<td>Grid for 2-Element Airfoil at $M = 0.125, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.40</td>
<td>Mach Contours for 2-Element Airfoil at $M = 0.125, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.41</td>
<td>Pressure Coefficient on Body for 2-Element Airfoil at $M = 0.125, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.42</td>
<td>$1 - p_0/p_{\infty}$ on Body for 2-Element Airfoil at $M = 0.125, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.43</td>
<td>Grid (1) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.44</td>
<td>Grid (2) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.45</td>
<td>Grid (3) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.46</td>
<td>Mach Contours (1) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.47</td>
<td>Mach Contours (2) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.48</td>
<td>Mach Contours (3) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.49</td>
<td>Pressure Coefficient on Body for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$</td>
<td></td>
</tr>
<tr>
<td>4.50</td>
<td>$1 - p_0/p_{\infty}$ on Body for 4-Element Airfoil at $M = 0.125, \alpha = 0^\circ$</td>
<td></td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Strengths and Weaknesses of Various Adaptation Criteria</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Multi-stage coefficients with a first order spatial discretization for optimal damping of the frequency range $\pi/2 \leq k\Delta x \leq \pi$</td>
<td>63</td>
</tr>
<tr>
<td>3.2</td>
<td>Multi-stage coefficients with a second order spatial discretization for optimal damping of the frequency range $\pi/2 \leq k\Delta x \leq \pi$</td>
<td>63</td>
</tr>
<tr>
<td>3.3</td>
<td>Subsonic Cylinder - Entropy And Drag For Three Grid Resolutions</td>
<td>85</td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

Although the variety of complex flows that computational fluid dynamics researchers can analyze continues to increase, the solutions to much more complex flows are desired. Improved computer capacity has and will continue to have a large effect on the quality of solutions obtained. Just as important, however, are recent improvements in the solution algorithms themselves. These new sophisticated algorithms attempt to overcome two main obstacles to obtaining complex flow solutions; the geometric complexity of the solution domain for realistic problems, and the existence of disparate length scales in the solutions.

Geometric complexity can be handled by more sophisticated grid generation schemes. Grid generation is a difficult task for complex geometric configurations. Many techniques still require significant user input to generate a grid for each new configuration. Modern techniques increasingly automate this process.

To overcome the disparate scales, the grid should be allowed to adapt to the solution to ensure that high-gradient regions in the flow are not under-resolved and that low-gradient regions in the flow are not over-resolved [44]. Dominant local length scales can be orders-of-magnitude smaller than the global flow length scale. This disparity in length scales results in computational errors which are much larger at
the dominant features than elsewhere in the flow. Adapting the computational grid allows the grid spacing to respond to the local length scales of the flow. The adaptation scheme must detect the flow features and respond by increasing or decreasing the local resolution of the grid. Unfortunately, adapting the grid forces a coupling of the grid generation and adaptation process with the flow solver. “Off the shelf” flow solvers cannot be used with a grid scheme to which adaptation has been added.

Grid adaptation is carried out in one of two ways: grid-point redistribution (r-refinement) or grid embedding (h-refinement). Grid-point redistribution is accomplished by redistributing the currently existing grid points to give a more advantageous distribution of the points in the domain. The grid embedding approach consists of adding or removing grid points in the regions of interest. The utility of these two adaptation strategies depends on the particular structure of the grids.

The biggest difficulty with structured grid methods is grid generation. Grids still cannot be generated for arbitrary complex geometries in a totally automatic way. Unstructured grid methods can provide grids for much more complex geometric configurations, but may not be able to handle arbitrarily complex geometries. These methods do provide an excellent framework for adaptation based on grid embedding, however.

Cartesian grid methods are based on the idea that a body is “cut” out of a “background” grid made up of cells with purely horizontal and vertical faces. The “cut” is determined by examining the interaction of a cell and the body. This procedure allows grids to be automatically generated for arbitrarily complex geometries. In order to allow the grid to adapt to the body, a data structure is needed similar to that used for an unstructured grid. Adaptation allows the grid not only to resolve properly highly curved regions of the bodies, but also to resolve the disparate length scales in
the flow. The existence of small cut cells, however, is still a problem with Cartesian grids. The Adaptive Cartesian grid method presented in this thesis overcomes the difficulties that the small cut cells present. The result is a unique method which uses a Cartesian grid to handle complex geometries automatically and uses adaptation to handle the disparate length scales (both flow-based and geometry-based). The broad range of results presented demonstrates the geometric flexibility of this approach, as well as the accuracy and efficiency attainable by solution-based adaptation.

1.1 Structured Grid Methods

A structured grid is one which can be mapped from physical space to a computational space in which it appears as a rectangle, in two dimensions. An example grid is given in Figure 1.1. A physical point in the grid is represented as grid point \((i, j)\) in the two dimensional array of the computational space.
1.1.1 Grid Generation

The generation of a suitable starting grid using a body-fitted structured grid about arbitrary bodies is a difficult task. This is due to the fact that it is increasingly difficult to maintain the structured cell numbering of the grid points. For example, one common organization of grid points dictates that $i$ represents points around the body while $j$ represents points radial to the body. For one body, this organization works well, but when multiple bodies exist, it becomes increasingly difficult to maintain.

Two standard methods are commonly used for generating these body-fitted structured grids: elliptic and algebraic grid generation. Elliptic grid generation is done by solving Laplace’s or Poisson’s equation, which smoothes the boundary data over the domain [28, 58]. Using the proper boundary conditions, the grid lines represent streamlines and potential lines of potential flow. The quality of the grids generated and the robustness of this method have made it popular for generating grids about simple geometries. An excellent review of PDE-based techniques is given by Eisenman [19]. Algebraic grid generation is based on the idea of a smooth interpolation between points on boundary curves, for example, transfinite interpolation [18]. Algebraic grid generation methods are faster than elliptic ones, but are not as easily automated. However, algebraic methods provide more control over the boundaries of the grid, which in turn allows an easier extension to multiblock methods. These two methods are not viable options for complex geometries, but they can each be used as building blocks for the newer multiblock [16] or patched-grid [46] methods. Multiblocking dictates that the physical domain is carved up into simple subdomains, or blocks, where elliptic grid generation yields satisfactory results. The different blocks are then linked together. The multiblocking is not unique, and how a flow is blocked
can greatly affect the quality of the grid. The work required to generate these grids increases as the complexity of the bodies increase. A way is needed to automate the blocking procedure for structured methods to be viable.

1.1.2 Grid Adaptation

Both grid-point redistribution and grid embedding have been employed on structured grids [15, 19, 57]. Structured grids are especially well suited to grid-point redistribution. The solution to the governing equations is computed on a grid which is composed of a fixed number of points, redistributed from their original positions such that they congregate in the vicinity of flow features. The desired result is the best solution possible for a fixed amount of resources. Possible point moving schemes include a spring analogy [39] and forcing function terms in the grid generation equations [27]. As an example, if the shaded region in part (a) of Figure 1.2 requires refinement, part (b) gives the resulting grid after the grid-point redistribution. Alternatively, the grid in part (c) is the result of grid embedding. Adding a point requires adding the entire line on which the point lies. Unfortunately, many of these new grid points may not be needed. For this reason, grid embedding is generally not practical for simple structured grids. More sophisticated structured grids which both overcome this grid embedding difficulty and also allow more complicated geometries include multi-block [16] and patched-grid [46] methods. Another variation is a tree of structured grids in which a region of the initial structured grid is refined by creating a structured grid with a finer resolution which is a child of the initial grid [6, 45].

1.2 Unstructured Grid Methods

An unstructured grid is one in which there is no mapping from the physical space to a simple computational space. Connectivity information must be stored, relating,
for instance, cells to the vertices that define them. A variety of data structures are possible [31, 51, 52], and the choice of data structure will affect the implementation of the grid generation process. Body-fitted unstructured grids are discussed below.

1.2.1 Grid Generation

The simplest form of body-fitted unstructured grid is one which is based on a structured grid [14, 47, 55]. The grid generation is achieved by destructuring a structured grid, as shown in Figure 1.3. The method works well for simple geometries where a structured grid is readily available, but when arbitrarily complex bodies are needed a more sophisticated method is needed.

One sophisticated grid generation scheme is the Advancing Front method [14, 32, 43]. First, a list of frontal faces is created between boundary nodes. The smallest face typically becomes the start of the front. An ideal third node is created from this face and put in a new list of nodes. Then, all other nodes in the triangulation are sorted by their distance from the new node and added to the list. The first node on the list which creates a triangle without crossing existing faces is used. The front is then updated, and the process repeated until completion. This method requires a lot of sorting and searching, inherently an $O(N^2)$ operation. More sophisticated
data structures used for searching can bring the costs down close to $O(N \log N)$. The grid created is usually smoothed with a Laplacian filter, resulting in a grid with a high degree of regularity.

Another sophisticated grid generation scheme is a Delaunay triangulation [3, 25, 34, 48, 49]. Delaunay alone is not a complete scheme, but only a method of triangulating a given set of points. One way to automate the introduction of points and create a Delaunay triangulation is first to triangulate the boundary nodes [23]. The Delaunay triangulation of these points is taken as the initial grid. Cells with a high skewness are then refined by the insertion of a new node at the circumcenter of that triangle, followed by retriangulation. This procedure is costly, however, since skewness is a difficult value to obtain. The Delaunay triangulation itself begins by dividing the domain into Voronoi regions. The Voronoi region for a given node consists of the part of the plane which is closer to that node than any other. A unique triangulation results when nodes whose Voronoi regions share a common boundary are connected. This triangulation is optimal in the sense that the minimum angle is
maximized in any triangle for all possible choices of diagonals between four convex nodes.

A new development is a combination of Advancing Front and Delaunay methods into a “Frontal-Delaunay” method [37]. Grids with a high directional skewness can be created for use with Navier-Stokes solvers, while some properties of Delaunay grids are maintained.

1.2.2 Grid Adaptation

Grid-point redistribution can also be useful for unstructured grids, but it is difficult to maintain properties of a “good” grid, specifically grid skewness. One popular method of adaptation uses a spring analogy to determine node movement [26, 41].

Unstructured grids are ideally suited for grid embedding. In areas where adaptation is needed, a cell is divided, or replaced by a number of smaller cells. Since the grid is unstructured, the change to the grid is purely local. The main difficulty is to determine when and where to refine. A number of different refinement criterion may be used which in some way use an estimate of the solution error. Then a threshold for refinement is set either by experience or by information obtained from the distribution of the error. Refinement of this type has been carried out on triangular/tetrahedral grids [34, 43, 64] and quadrilateral/hexahedral grids [7, 17, 36, 40, 59, 64] or a mixture of both [38, 47]. This refinement is most often isotropic, but anisotropic (directional) refinement is also a viable option. [1, 14, 24, 55].

1.3 Cartesian Grid Methods

An approach which is becoming more popular is the use of non-body-fitted unstructured grids, specifically Cartesian grids, in which the bodies are “cut” out of the grid. A sample grid is shown for the leading edge of an airfoil in Figure 1.4. This
Figure 1.4: Grid for Leading Edge of 3-Element Airfoil

technique has been commonly used for potential flow calculations [66]. A new and exciting approach to automatic grid generation has been recently presented which makes use of CAD/CAM surface geometry databases [35].

Cartesian grids have a number of advantages including ease of grid generation, simpler flux formulations, simplifications of the data structure, and a fortuitous cancellation of truncation errors in regions where the grid is regular. But, Cartesian grids also have a couple of difficulties to overcome: the difficulty of resolving high curvature regions of a body; and the introduction of cut cells that are a small fraction of the size of uncut cells.

The small cut cell issue is a large problem. The small cut cells can not only lead to inaccuracies in the flow by becoming decoupled from the rest of the flow, but to severe restrictions on the cell’s time-step. A variety of methods have been used to overcome this difficulty, with varying degrees of success. They include: cell merging [5, 11]; rotated differences with large time-steps [8]; non-conservative
extrapolations \[20\]; volume of fluid redistributions \[42\]; and a linear reconstruction with local time-stepping \[13, 17\].

1.4 A New Approach - The Adaptive Cartesian Grid Method

An adaptive Cartesian method combines the best elements of structured, unstructured, and Cartesian grids. One element is a fortuitous cancellation of truncations errors in regions where the grid is regular, as in structured grids. A Cartesian grid can also be easily generated about any arbitrary geometric configuration; at least as easily as an unstructured grid. And, by applying adaptation to the resulting Cartesian grid, both the geometry-based and solution-based length scales can be properly resolved.

The main difficulty lies in creating a flow solver which fits into this approach. It must not only handle the differences in cell level, but also accurately address the cut cells. This includes the somewhat troublesome cut cells which are several orders of
magnitude smaller in area than their uncut neighbor cells.

An early Euler solver based on central-differencing on Cartesian grids was developed by Clarke et al [11]. Body curvature resolution was achieved by clustering the grid lines near the areas of interest. Cut cells with an area less than 25% - 50% of uncut neighbor cells were merged into adjacent cells away from the body. The solutions obtained were poor near highly curved regions of the body and not smooth along cut cells. Another central-differencing method was used by Epstein et al [20]. Here, resolution was achieved by local grid refinement. Cut cells were handled by a non-conservative extrapolation procedure. These solutions resolved body curvature, but did not adequately address cut cell boundary conditions.

The first upwind-differencing method of the Euler equations on an adaptively-refined Cartesian grid was developed by Berger and LeVeque [8]. They addressed unsteady flows by using a rotated difference scheme with large time-steps on small cells. Adaptation was achieved through local uniform fine grid patches, which were recursively nested. This approach works for cut cells with area greater than a few percent of uncut cells, but a special cell merging procedure was need for the very small cells. A related unsteady solution method was developed by Pember et al [42]. Again, adaptation was achieved by refined patches of the grid. Cut cells were addressed by a “volume-of-fluid redistribution” to eliminate time step restrictions. This method has problems with this type of boundary condition on small cut cells.

In the work presented in this thesis, cell adaptation is applied to resolve properly both body curvature and flow features. This adaptation is no longer “block” adaptation, but rather “cell” adaptation, allowing more efficient resolution of features that are not aligned with the grid. Cut cells are handled by a linear reconstruction method designed for unstructured grids combined with local time-stepping. Arbitrarily small
cut cells now require no special treatment.

The method specifically consists of a Cartesian grid with an unstructured cell-based data structure. Initial grid generation is enhanced by geometry-based refinement [38]. Solution adaptation is achieved by solution-based gradient information [40, 64]. The flow solver is based on the MUSCL concept [60], with a linear reconstruction technique [4], and either Roe’s approximate Riemann solver [50] or Van Leer’s flux splitting [30]. The solution is advanced in time by a multi-stage time stepping scheme [54, 62] with multigrid convergence acceleration [9].

The data structure and grid generation are discussed in Chapter II. Each element of the flow solver, including some accuracy studies, are examined in Chapter III. Chapter IV discusses the results for a variety of test cases. Finally, in Chapter V, a summary of the work presented is given, along with some conclusions and directions for future work.
2.1 Quadtree Data Structure

The data structure of a code is the roadmap of the information contained in the code. It shows what information is stored, and how to access it. For a typical structured-grid code, the data structure can be wholly contained in an \((i, j)\) index. Unstructured-grid codes usually have a significantly more complicated data structure. Of primary concern is the connectivity of the grid. Some information must be stored to determine which cells are neighbors of a given cell. A common method of achieving this is a linked list. For each cell, a list is stored of neighboring cells or faces, for example, implemented as array indices in FORTRAN or memory addresses in C. For many unstructured-grid schemes, this is the only option to determine cell connectivity. Some schemes, however, allow the use of a data tree to achieve this goal. Other features of the code will determine whether the linked list or tree structure will be the more efficient choice.

A quadtree-based structure is ideally suited for the Cartesian grid scheme implemented here. Like any true tree structure, it begins with one “root” cell. The root cell is said to be at level 0. When this cell is refined, four children cells of equal size
Figure 2.1: Quadtree Structure

are created at one level higher, level 1. In turn, these cells are refined again until a suitable grid has been created. Each cell has a pointer to its parent and its four children, if they exist. From this basic structure, all other needed information about the relative position of any cell can be determined. This concept is illustrated in Figure 2.1.

This tree structure has many inherent advantages over a linked list. One of the most important advantages is that a tree-based structure is well-suited to multigrid [22]. A multigrid method relies on a sequence of coarser grids. The low-frequency errors on a given grid become higher-frequency errors when passed to a coarser grid, and can then be more easily damped out. With the tree-based structure, these coarser grids already exist, and the communication between levels follows the existing parent/child tree relationship. Multigrid implementation in a linked list requires the generation of multiple grids and special communication to relate the information between the grids. This process is difficult at best using a linked list.

Another very important advantage of a tree-based data structure is the ease
with which cells can be refined or coarsened. For a cell to be refined, the children
 pointers which were previously empty would now point to the four newly created
 children cells. Using a linked list would require an additional check to see if there
 are any neighbors which need their neighbor links updated to match these new cells.
 Coarsening a cell is also a very simple process in a tree structure. The cell simply
 removes the pointers to its four children. This assumes that the children cells do not
 have children themselves. In a linked list, the effect this has on all neighboring cells
 must be determined. Another inherent advantage to a data tree is the amount of
 information which can be obtained simply from the tree. Much of this information
 is obtained by determining the level of a cell, or the number of successive parents to
 the root cell. For example, the area of an uncut cell is simply a function of the root
 cell area and the cell’s level

 \[ A_c = \frac{A_{\text{root}}}{4^{\text{level}}} \].  \hspace{1cm} (2.1) 

 Neighbors in a tree structure are determined from the data tree, rather than
directly stored in a linked list structure. To determine a cell’s neighbor, the parents
 are recursively queried as to whether one of its other children is that neighbor. In the
 best case, finding a cell’s neighbor requires simply querying its parent for the location
 of one of its other children. In the worst case, the tree must be traversed all the way
to the root and back to find the neighbor. In general, finding the neighbor usually
 involves querying the cell’s parent and grand-parent, since the expected number of
 levels asymptotes to

 \[ E\left(n_{lev}\right) = \frac{1}{2} + \frac{1}{2} \left(\frac{2}{2} + \frac{1}{2} \left(\frac{3}{2} \cdots \right) \right) \]

 \[ = \sum \frac{k}{2^k} \]

 \[ \asymp 2 \hspace{1cm} (2.2) \]
for finding neighbors in the four primary directions. Thus, the neighbor is usually a
grandchild of the cell’s grandparent. The expected number of levels asymptotes to
$\frac{8}{3}$ for finding neighbors in all eight directions (horizontal, vertical, and diagonal). As
an example, the pseudo-code for finding the North neighbor of a cell is listed below.

```plaintext
NorthNeighbor(cellA)
    cellB = parent of cellA
    if cellA is SW child of cellB, then
        NorthNeighbor = NW child of cellB
    if cellA is SE child of cellB, then
        NorthNeighbor = NE child of cellB
    if cellA is NE child of cellB, then
        cellC = NorthNeighbor(cellB)
        NorthNeighbor = SE child of cellC
    if cellA is NW child of cellB, then
        cellC = NorthNeighbor(cellB)
        NorthNeighbor = SW child of cellC
```

Figure 2.2 shows the tree paths taken for finding a North neighbor which require
different numbers of levels.

Although the data tree is the most important part of this data structure, it is
not the only part. Cell type classification must be provided by the data structure.
It must also provide a means for storing and retrieving information about cells cut
by a body. An integer value is used for each cell which records the cell type. For cut
cells, that type is a two digit number which is encoded with information regarding
which of its faces has been cut by the body. Each face of a cut cell is divided into
two segments, so that the cell is made up of eight segments, numbered clockwise
from the left segment of the top face. Then the first digit in the cell type is the first
face of the cut and the second digit the second face of the cut. The first and second
cuts are determined by their clockwise order. Figure 2.3 shows a few cut cells and
the resulting cell types. For those cut cells, the coordinates of the cut location, or
Figure 2.2: Tree Paths for Various North Neighbors
node, also needs to be stored, as well as a link between the cut cell and the correct cut node number.

Since this data structure requires that some information be computed when needed, it is important to see that computing this information does not take so much time as to eliminate the effectiveness of the data tree approach. Note that how much time a specific piece requires to compute will be highly dependent upon the specific programming implementation. The time to compute different pieces of information is not fixed. A good example is the time to compute the level of a cell. This is a very simple procedure, but the level of a cell is used many, many times. If the cell level is computed when it is needed, that computation takes 10% of the total time. On the other hand, if the cell level is computed once and stored, only one integer of storage per cell is needed. Determination of the neighbors of a cell is another necessary procedure. If a cell’s neighbors are computed when needed, it takes 10-15% of the total time. Storing that information requires up to 10 integers of storage per cell. Based on these kinds of comparisons, the cell level is stored and

Figure 2.3: Various Cut-Cell Cell Types
the cell neighbors are computed when needed.

Memory/time comparisons are not only needed for specific routines, but also for the whole code. The quadtree data structure here allows us to keep the amount of storage to a level comparable to that of a structured grid approach. The integer variables stored per cell total 10 words, as follows:

- 5 words - Pointers to one parent and four children cells
- 1 word - Cell type information
- 1 word - Cell level
- 1 word - Multigrid level
- 2 words - Other temporary values

The real variables stored per cell total 31 words, as follows:

- 4 words - Conserved variables (density, momentum, and energy)
- 4 words - Temporary conserved variables for time stepping scheme
- 4 words - Temporary conserved variables for multigrid
- 4 words - Cell centroid, area, and volume (axisymmetric)
- 8 words - Gradients in X and Y directions
- 4 words - Residuals for updating conserved variables
- 1 word - Time step
- 1 word - Cell pressure
- 1 word - Limiter value for frozen limiter
Since only the integer storage is associated with the data structure, this code requires nearly the same amount of memory as a structured grid code implementing the same flow solver strategy. On the other hand, the data structure imposes a 30% overhead for the total time to run the code, with neighbor finding accounting for half of that overhead. This run-time cost is offset by the fact that this method can produce up to an order of magnitude reduction in the number of cells needed due to adaptation.

2.2 Generating The Initial Grid

In the generation of a suitable grid, the first step is the generation of an initial grid. To generate the initial grid, certain information is needed from the user of the program. In the approach developed in this work, all that is needed is the location of the outer boundaries, the cell aspect ratio, and a desired minimum grid resolution. Then, this information in hand, the generation of a suitable initial grid can proceed.

The outer boundary information used is in the form of an $X$ coordinate at the left and right boundary, $X_L$ and $X_R$, and a $Y$ coordinate at the bottom and top boundary, $Y_B$ and $Y_T$. The cell aspect ratio and minimum grid resolution are specified by giving the number of cells in each direction, $M$ in the $X$ direction and $N$ in the $Y$ direction. Once this information is known, the exact size and location of the root cell can be determined. First, the size of a cell at the given minimum grid resolution is $\Delta X_c$ by $\Delta Y_c$ defined as

\[
\begin{align*}
\Delta X_c &= \frac{X_R - X_L}{M} \\
\Delta Y_c &= \frac{Y_T - Y_B}{N}.
\end{align*}
\]

Next, since the flow is surrounded by a ring of boundary cells at the minimum grid resolution, the left $X$ and bottom $Y$ values of the root cell are set as

\[
\begin{align*}
X_{L,\text{root}} &= X_L - \Delta X_c \\
Y_{B,\text{root}} &= Y_B - \Delta Y_c.
\end{align*}
\]
Now, since successively refining a root cell and its children will give possible uniform grid resolutions of 1, 2, 4, 8, 16, $\ldots$, $2^k$, etc., the smallest value of $k$ that satisfies $2^k \geq \max(M, N) + 2$ must be found. With $k$ known, the right $X$ and top $Y$ values of the root cell become

$$X_{R_{\text{root}}} = X_{L_{\text{root}}} + 2^k \Delta X_c$$

$$Y_{T_{\text{root}}} = Y_{B_{\text{root}}} + 2^k \Delta Y_c.$$  

(2.5)

At this point, all the attributes of the root cell can be set. Its location and size are known, and cell level is set to zero. Now that the first cell is completed, all cells without children which lie within the outer boundaries are recursively refined until the cells reach a level equal to $k$. By just refining cells within the outer boundaries, the number of cells that will never be used because they are outside of the flowfield is reduced. Finally, all cells at level $k$ that are within the box described by

$$X_L - \Delta X_c \leq X \leq X_R + \Delta X_c$$

$$Y_B - \Delta Y_c \leq Y \leq Y_T + \Delta Y_c.$$  

(2.6)

are flagged as cells in the flowfield. All other cells are marked as unused cells which just complete the data tree. Of the cells in the flowfield, cells within

$$X_L \leq X \leq X_R$$

$$Y_B \leq Y \leq Y_T$$  

(2.7)

are specifically marked to be computational cells and the rest of the flowfield cells to be outer boundary ghost cells.

The grid generation for an arbitrary channel flow can serve as an example. The user has specified the following:

$$X_L = 0.0 \quad , \quad X_R = 3.0$$

$$Y_B = 0.0 \quad , \quad Y_T = 1.0$$

$$M = 12 \quad , \quad N = 4.$$  

(2.8)
Therefore, the minimum grid resolution cell size is known to be $\Delta X_c = 0.25$ by $\Delta Y_c = 0.25$ and the number of cell levels, $k$, to have $2^k \geq 14$ is 4. Then the root cell is defined by $X_{L,root} = -0.25$, $X_{R,root} = 3.75$, $Y_{B,root} = -0.25$, and $Y_{T,root} = 3.75$. The grid generated is pictured in Figure 2.4.

For flows with bodies, each body must be “cut” out of the flow. This is done by examining the four corner nodes of each computational cell. Each node is tested to see whether it lies inside, outside, or on a body by examining the intersections of the $x=$constant and $y=$constant lines through the node and the body. Since each body is made up of a combination of parametric splines and linear segments, the intersections are found through a search routine. The cell type is saved when the classification of the four corner nodes has finished. A node that has been found on the body is said to be outside of the body when no other node is inside or on the body, otherwise it is considered to be inside the body. When all four nodes lie outside of the body, the cell becomes an uncut computational cell. When all four nodes lie
inside of the body, the cell becomes an unused cell just filling the data tree. All other combinations imply a cell which has been cut by the body. The cut locations are stored using the $x=$constant and $y=$constant intersection information. From here on out, each new cell is classified and checked for body cuts as soon as it is created.

### 2.3 Geometry Adaptation

Once the initial grid has been generated and the cells all classified, the grid is improved through a series of three geometry-based adaptations: all-cell adaptation; cut-cell adaptation; and curvature-cell adaptation. The amount of each type of adaptation is determined by individual cell length scale parameters given by the user. The cumulative effect of these procedures is a suitable grid for the computation of a flow solution.

In all-cell adaptation, all computational cells are refined until their length scale, the minimum of $\Delta x$ and $\Delta y$, is less than the length scale provided by the user for this type of adaptation. The minimum grid resolution is already known from the generation of the initial grid, but the user may wish to begin flow calculation on a finer grid, particularly for use with the multigrid convergence acceleration described in Section 3.3.

Cut-cell adaptation is similar to all-cell adaptation. The difference is that cells cut by the body will be refined if their length scale is less than the cut-cell length scale. Another difference is that when a cell is flagged for refinement, all of its neighbors are also refined. Refining the neighbors ensures a smoother transition from the finer cells on the body to the coarser cells away from the body.

Once cut-cell adaptation has finished, curvature-cell adaptation begins. This process begins by examining two neighboring cut cells and comparing the slope of the body face on each cell. If the difference in the slopes is above a threshold value, the cells and their neighbors are flagged for refinement. The actual check used to
flag cells for curvature adaptation is

\[ |\theta_{Cell1} - \theta_{Cell2}| \geq 5^\circ . \tag{2.9} \]

Figures 2.5-2.7 represent the stages in the grid generation process for a scramjet inlet. Figure 2.5 shows the grid after the initial grid has been generated and all-cell adaptation has finished. Figure 2.6 highlights the effect of cut-cell adaptation. Finally, the grid in Figure 2.7 demonstrates the effect of curvature adaptation.
2.4 Solution Adaptation

In addition to all-cell, cut-cell, and curvature-cell adaptation, the grid is also modified by solution-based adaptation. The cells in the grid are either refined or coarsened based on the characteristics of the flow. This takes place only after a solution is sufficiently converged. At that point, cells are flagged for refinement or coarsening based on a given adaptation criterion. That criterion should detect any discontinuity in the flow. For inviscid compressible flow, the important discontinuities are shocks, shear layers, and contact surfaces.

Typically, the criterion of choice has been a density, pressure, or total-velocity difference. However, since the density and pressure differences have difficulty detecting shear layers, differences of total velocity have been most often used. Even so, the total-velocity difference has trouble differentiating between features of different strengths, such as a strong shock and a weaker shear. The total-velocity difference criterion will be compared to a new adaptation criterion based on the curl and divergence of velocity. This new criterion effectively detects each discontinuity and minimizes the differences in the relative strengths of different discontinuities. The strengths and weaknesses of various criteria are listed in Table 2.1.
<table>
<thead>
<tr>
<th>Criterion</th>
<th>Strength</th>
<th>Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>\tilde{e}_i</td>
<td>\propto</td>
</tr>
<tr>
<td>$</td>
<td>\tilde{e}_i</td>
<td>\propto</td>
</tr>
<tr>
<td>$</td>
<td>\tilde{e}_i</td>
<td>\propto</td>
</tr>
<tr>
<td>$</td>
<td>\tilde{e}_i</td>
<td>\propto l^2 \left( \frac{\partial^2 \rho}{\partial x^2} + \frac{\partial^2 \rho}{\partial x \partial y} + \frac{\partial^2 \rho}{\partial y^2} \right)$</td>
</tr>
<tr>
<td>$</td>
<td>\tilde{e}_i</td>
<td>\propto l_i</td>
</tr>
<tr>
<td>$</td>
<td>\tilde{e}_i</td>
<td>\propto l_i</td>
</tr>
<tr>
<td>Combination of $</td>
<td>\tilde{e}_i</td>
<td>\propto l_i</td>
</tr>
</tbody>
</table>

Table 2.1: Strengths and Weaknesses of Various Adaptation Criteria
2.4.1 Total-Velocity Difference

The total-velocity difference is most often implemented as a “face based” difference. The difference is computed between the two cells sharing a face. Then if the face difference exceeds the adaptation threshold, both the cells are flagged for refinement. However, this criterion will be formulated as a “cell based” difference to match the curl and divergence of velocity criterion. Then the difference for a cell is just the maximum of its face differences.

The total-velocity criterion is made up of two factors: an undivided difference of total velocity and a length scale weight, which allows adaptation to see other features after one feature has already been resolved [61]. The criterion, \( d_i \), for cell \( i \) is

\[
d_i = l_i^2 \max_{j \text{ faces}} (|\Delta q_i|),
\]

where \( q_i \) is the quantity the difference is applied to, in this case total velocity, and \( \Delta q_i \) the face based difference. \( l_i \) is a length scale for the cell and \( r \) determines the weight that \( l_i \) applies to the difference. This length scale weight is used to find weaker features which are in a coarser grid area of the flow. This allows the weaker features to be refined when the stronger features have been resolved. The best example is when a shock is in the flow. The undivided difference of any quantity in a shock will remain essentially the same as the shock receives more and more refinement. The adaptation criterion will never leave the shock alone and refine other weaker features. The length scale effectively weakens the shock’s undivided differences as it is refined. For all cases presented, \( r \) was taken to be 2. As \( r \) goes to infinity, the difference becomes a simple undivided difference.

In order to set a threshold for the refinement and coarsening, the standard deviation about zero is computed for \( d_i \) for \( n \) cells,

\[
\sigma = \sqrt{\frac{\sum_{i=1}^{n} d_i^2}{n}}.
\]
A cell is flagged for refinement if $d_i > \sigma$ and is flagged for coarsening if $d_i < \frac{1}{10}\sigma$. Figure 2.8 shows this graphically.

### 2.4.2 Curl and Divergence of Velocity

A more effective adaptation criterion is one which is built up from pieces which independently find specific flow features. Specifically, using the curl of velocity is used to find shear layers and the divergence of velocity is used to find shocks [40]. For each cell, weighted forms of the curl, $\tau_c$, and divergence, $\tau_d$, of the velocity are computed as

$$\tau_{ci} = |\nabla \times U| \frac{l_i^{p+1}}{l_i}, \quad \tau_{di} = |\nabla \cdot U| \frac{l_i^{p+1}}{l_i}, \quad (2.12)$$

where $l_i$ is again the length scale for the cell. The extra factor of $l_i$ is needed to make the curl and divergence undivided differences. Once these forms of the curl and divergence have been computed for all $n$ cells, the standard deviations about zero are computed for each,

$$\sigma_c = \sqrt{\frac{\sum_{i=1}^{n} \tau_{ci}^2}{n}}, \quad \sigma_d = \sqrt{\frac{\sum_{i=1}^{n} \tau_{di}^2}{n}}. \quad (2.13)$$

A cell is flagged for refinement or coarsening if one of two possible conditions hold:
1. if either \( \tau_{ci} > \sigma_c \) or \( \tau_{di} > \sigma_d \), the cell is flagged for refinement,

2. if both \( \tau_{ci} < \frac{1}{10} \sigma_c \) and \( \tau_{di} < \frac{1}{10} \sigma_d \), the cell is flagged for coarsening.

For cases where the resolution of one feature is much more important than other features, the weights on the adaptation criterion can be changed. For example, if the resolution of a shear layer is vital while the shocks in the flow are not, cells could be flagged for refinement if \( \tau_{ci} > \frac{1}{2} \sigma_c \) or \( \tau_{di} > 2 \sigma_c \). This flags more cells for refinement based on curl and less cells based on divergence, effectively weighting the shear layer more heavily.

2.4.3 Comparison of Adaptation Criteria

The adaptation criteria described above were tested on a variety of test cases, including a double wedge, a NACA airfoil, and an axisymmetric jet, among others. Of these, an axisymmetric jet will be examined first since it clearly highlights the differences of the adaptation criteria when shocks are present. Then a smooth subsonic NACA airfoil will be examined.

The axisymmetric jet is made up of a nozzle of radius one which exits into the freestream flow at \( X = 0.0 \). This flow has a freestream and jet Mach number of 1.25, a jet-to-stream total temperature ratio of 1.0, and a jet-to-stream total pressure ratio of 35.0. This particular jet flow has the following flow features: 1) an expansion out of the nozzle, 2) a curved shock emanating from the nozzle lip and reflecting at the symmetry axis, 3) a shear separating the jet and the outer stream, and 4) an oblique shock through which the outer stream passes. A more detailed description of the axisymmetric jet is found in Section 4.3. The grid and Mach contours for the converged solution are shown in Figures 2.9 and 2.10.

Figure 2.11 shows the cells that will be flagged for refinement based on total velocity. Notice that both the shear layer and the shocks get flagged for refinement. However, the shocks are not flagged along their entire length. As the shock on
Figure 2.9: Initial Grid

Figure 2.10: Initial Grid Mach Number Contours
the nozzle lip weakens, it does not generate enough total-velocity difference to be flagged. Also, very little of the expansion at the nozzle lip which reflects off the axis of symmetry is flagged.

In an effort to get these features flagged for more efficient refinement, the adaptation threshold can be lowered for refining cells. The plot for division based on \( d_i > \frac{1}{2} \sigma \) is shown in Figure 2.12. Here the complete length of the shocks are flagged, but the expansion fan is still missed. In addition, the number of cells flagged has nearly doubled, with only a little improvement in the cells flagged.

Figure 2.13 shows which cells will be flagged for coarsening. A large portion of the expansion fan, and much of the outer stream flow, has been flagged. From these adaptation plots, it is obvious that in terms of total velocity difference, the shear layer is the strongest feature present, with the shocks a little weaker, and the expansion fan much weaker than the shear.

This variation in the relative strengths is easily taken care of with adaptation based on the curl and divergence of velocity. The curl of the velocity very effectively
Axisymmetric Jet, Total Velocity Adaptation.
2546 Cells Flagged, $d_i > \frac{1}{2} \sigma$.

Figure 2.12: Cells Flagged for Refinement by Total-Velocity Adaptation, $d_i > \frac{1}{2} \sigma$.

Axisymmetric Jet, Total Velocity Adaptation.
2292 Cells Flagged, $d_i < \frac{1}{10} \sigma$.

Figure 2.13: Cells Flagged for Coarsening by Total-Velocity Adaptation, $d_i < \frac{1}{10} \sigma$. 
finds any shear features in the flow, as shown in Figure 2.14. Similarly Figure 2.15 shows that the divergence of velocity effectively finds any shock or expansion features.

When the curl and divergence are combined, as in Figure 2.16, the result is an effective means of flagging cells for refinement. But, effective grid adaptation also requires an efficient flagging of cells for coarsening. Figure 2.17 shows the cells that have been flagged for coarsening by the curl and divergence of velocity. It is clear that the cells flagged appropriately represent the areas in the flow where coarsening cells would not damage the solution accuracy. It is important to note that even though a cell is flagged for coarsening, it may not be coarsened because of the grid “smoothing” discussed in Section 2.5. The final grid for solution adaptation based on the curl and divergence of velocity and grid smoothing is shown in Figure 2.18.

The second case considered is a subsonic NACA 0012 airfoil at a Mach number of 0.63 and an angle of attack of 2 degrees. This flow has an expansion at the leading edge which nearly reaches Mach 1. Since the resolution at the leading edge is not
Figure 2.15: Cells Flagged for Refinement by Divergence of Velocity, $\tau_{d_i} > \sigma_d$.

Figure 2.16: Cells Flagged for Refinement by Curl and Divergence of Velocity, $\tau_{c_i} > \sigma_c$ or $\tau_{d_i} > \sigma_d$. 
Figure 2.17: Cells Flagged for Coarsening by Curl and Divergence of Velocity,
\[ \tau_{ci} < \frac{1}{10} \sigma_c \text{ and } \tau_{di} < \frac{1}{10} \sigma_d. \]

Figure 2.18: Final Grid
quite sufficient, entropy is generated near the stagnation point, and convected over the upper surface of the airfoil. Only the cells which are flagged for refinement will be examined. Since this is an airfoil flow, the cells flagged to coarsen will typically be those cells at a distance of about one or less chords from the airfoil. The grid and Mach contours for this flow are pictured in Figures 2.19 and 2.20.

Total-velocity adaptation will flag for refinement the cells pictured in Figure 2.21. The refinement is restricted to the area around the leading edge and the trailing edge. The entropy layer is not even seen.

When the curl of velocity adaptation criterion is applied, the cells plotted in Figure 2.22 are flagged for refinement. Note that no curl should exist in the flow, but when an entropy layer exists, a small curl is found in that entropy layer. The divergence of velocity, on the other hand, flags the cells which are near the leading and trailing edge stagnation and the expansion about the front portion of the airfoil. The divergence is pictured in Figure 2.23. The reason that this figure may look a little odd, is that the cell length scale weighting is coming into play. Cells with a
Figure 2.20: Initial Grid Mach Number Contours

Figure 2.21: Cells Flagged for Refinement by Total-Velocity Refinement, $d_i > \sigma$
Figure 2.22: Cells Flagged for Refinement by Curl of Velocity, $\tau_{ci} > \sigma_c$

Figure 2.23: Cells Flagged for Refinement by Divergence of Velocity, $\tau_{di} > \sigma_d$
Figure 2.24: Cells Flagged for Refinement by Curl and Divergence of Velocity, \( \tau_{ci} > \sigma_e \) or \( \tau_{di} > \sigma_d \)

little smaller divergence, but with a larger size, will more likely be flagged. The combined effect of these two criteria is shown in Figure 2.24. When this combined effect is implemented, and the grid is smoothed, the grid pictured in Figure 2.25 is obtained.

2.5 Grid “Smoothing”

Any grid that has been generated by the method previously described could have certain “undesirable features.” Some are undesirable in that they would violate the data structure; for others, allowing them would unnecessarily complicate the data structure; still others are undesirable in that computational experience shows that they may degrade the solution somewhat in their vicinity. In order to remove these features, the grid is modified by coarsening or refining individual cells to remove a particular undesirable feature. This process is called “smoothing” the grid. This procedure is recursive, and converges to a grid with no undesirable features. Grid
smoothing is applied immediately following any major modification to the grid and
before the flow solver begins its operation. For example, the grid is smoothed after the
initial grid is created, after the geometry adaptation is done, after solution adaptation
is finished, and before the flow solver is applied.

The features that the grid smoothing routine removes have changed considerably
as the specific implementations of the data structure and flow solver have changed.
A good example of this is the need for removing cell level differences greater than one
between neighboring cells. In an early version of the code, this was done because the
data structure was not general enough to represent grids with jumps in cell level. In
a later version, the data structure was generalized so that it was no longer necessary
to remove cell level differences by smoothing. But, it was then seen that this feature
can lead to a degradation of the solution in its vicinity. Thus the feature was then
returned to the grid smoothing routine for removal. The features discussed below
are those currently “smoothed” from the grid.

The most important smoothing takes place to remove features which would violate
the way the data structure interacts with the flow solver. Two conditions fall in this category. They are listed below and depicted in Figure 2.26.

1. Cells with more than two faces cut, for example cells that have more than one body face cutting them,

2. Cells with shared faces where one cell thinks the face is cut while the other cell thinks the face is uncut.

The main problem with item 1) is that the part of the cell outside of the body is split on two sides of the body, requiring two distinct states, areas, centroids, etc. Item 2) causes problems because length of the shared face is different for the two cells.

Other features are smoothed away because allowing them would unnecessarily complicate the data structure. These features are:

1. Any childless cell which has an immediate neighbor that was smoothed due to a violation of the data structure,

2. Any cell level difference between the cells on the outer boundary and the first row of “ghost cells”.

Figure 2.27 graphically illustrates these features. Item 1) in this figure is flagged
to ensure that a “buffer zone” exists around problem cells. No uncut cell will have an immediate neighbor which is a problem cell. This simply allows much less error checking when finding neighbors and in the flow solver. Item 2) exists simply to facilitate much more general outer boundary conditions. For instance, a boundary condition which was a function of $x$ and $y$ could simply be put in these ghost cells which are at the same level as the cell which uses them, preserving the accuracy of the boundary condition.

Finally, some features are removed simply because computational experience shows that they may degrade the solution somewhat in their vicinity. These features may also be removed simply for aesthetic reasons. The features are diagrammed in Figure 2.28 and listed below.

1. Cell level differences greater than one between neighboring cells,

2. Some cell level differences around cut cells,

3. “Hills” in the grid, for example a cell with children and all of its neighbors on the same level without children,

4. “Holes” in the grid, for example a cell without children and all of its neighbors on the same level with children,
As explained above, item 1) was originally a violation of the data structure. But, when that problem was removed, this feature was again flagged for removal from the grid because it harms the solution accuracy. A more gradual level change of cells helps to remove “grid effects” from the solution. The reason item 2) is smoothed away is similar to item 1). A steep level change of cells away from the body caused a deterioration of the solution in that region. Items 3) and 4) are almost purely aesthetic considerations. They have very little effect on the solution accuracy, but have a very large effect on how good the grid looks.

To show the effect of these accuracy and aesthetic features, a 15 degree ramp in a channel is presented. The grid and Mach contours are shown in Figure 2.29. The grid has no aesthetic “trouble spots” and the Mach contours smoothly flow through the cell level changes. Figure 2.30 shows the grid and Mach contours without the accuracy and aesthetic smoothing. This grid was intentionally modified to show the
Figure 2.29: Grid and Mach Contours With Accuracy And Aesthetic Features Smoothened
type of grid which would be legal without the accuracy and aesthetic smoothing. Nevertheless, the Mach contours given are nearly identical to the smoothed flow, but with a little deviation where neighboring cells are more than one cell level apart.

It is possible for grid smoothing to “undo” some of the changes made by solution adaptation. For instance, solution adaptation may require a particular cell to be coarsened while grid smoothing may want that same cell refined again. When this happens, a small amount of detail may be lost. In order to avoid this problem, some “smoothing logic” has been put into the solution adaptation method. Thus, when solution adaptation flags a cell for coarsening or refinement, and it is obvious that smoothing will “undo” this operation, the cell is left as is. The simplest example may be when a cell with more than two cut faces exists. This particular undesirable feature is removed by refining it and its eight neighbors. If it is determined that
one of these nine cells should be coarsened, the cell is now left alone since that cell
would have been refined again in smoothing, no matter what other cells around it
are refined or coarsened.
CHAPTER III

FLOW SOLVER

The upwind finite-volume flow solver is based on the work of Godunov [21]. In the original approach, the solution was considered to be piecewise constant over each grid cell at a fixed time. The evolution of the flow to the next time step results from the wave interactions originating at adjacent cell boundaries, specifically a Riemann problem. In Riemann’s initial-value problem, a membrane separating a gas at two different states is ruptured, and shock, contact, and expansion waves are emitted when the two states interact. The Riemann problem is pictured in Figure 3.1. This approach leads to first order spatial accuracy.

The Godunov approach was first extended to second-order spatial accuracy by the MUSCL approach developed by Van Leer [60], made up of two decoupled stages. The first stage is the projection stage where the face values $U_L$ and $U_R$ are created. The second stage is the solution to the resulting Riemann problem. Using a linear approximation of the solution on each cell for the projection stage leads to second-order spatial accuracy, while a quadratic representation on each cell leads to third-order spatial accuracy. In this work, a second-order spatial accuracy is achieved by a linear reconstruction procedure on each cell [4]. The effect on $U_L$ and $U_R$ by this higher order reconstruction is pictured in Figure 3.2. Unfortunately, limiting is required to enforce the monotonicity of the reconstruction. Wherever limiting is used, the spatial accuracy of the projection is somewhat reduced, possibly to a first
order spatial accuracy.

The Euler flow solver described here consists of three primary components: a linear reconstruction procedure, for obtaining accurate, limited values of the flow variables at face midpoints; a flux formulation, for computing the flux through cell-faces; and a multi-stage time-stepping scheme using multigrid, for advancing the solution to a steady state. The individual components of these procedures, along with some code accuracy and efficiency considerations, are described below.
3.1 Reconstruction

In order to evaluate the flux through a face, flow quantities are required at both sides of the face. To achieve higher-order accuracy, these flow quantities must be calculated using solution-gradient information. Barth [3, 4] describes a method by which these gradients can be reconstructed to an arbitrary degree of accuracy on unstructured grids. A method specifically tailored to unstructured grids is needed so that the large cell-area differences which occur in the grids will not lead to accuracy or stability problems. A second-order approximation to the gradients can be obtained by either the path integral method described in Section 3.1.1 or by the least squares method discussed in Section 3.1.2. Each of these requires a suitable collection of neighbor cells to determine the gradient.

\( W_k \) represents the quantity being reconstructed; in this work, the primitive variables \( \mathbf{W} = (\rho, u, v, p)^T \) are reconstructed. Once the gradient of \( W_k \) is known in each cell, by either method, the value of \( W_k \) can be found anywhere in the cell from

\[
W_k(x, y) = W_k^c + \nabla W_k \cdot \mathbf{dr}
\]

where \( W_k^c \) is the value of \( W_k \) at the cell centroid, and \( \mathbf{dr} \) is defined as

\[
dr_x = x - x^c, \quad dr_y = y - y^c.
\]

For example, the values at the face midpoints of an uncut cell are simply

\[
\begin{align*}
W_{top}^k &= W_k^c + \frac{\Delta y_{cell}}{2} \nabla_y W_k \\
W_{right}^k &= W_k^c + \frac{\Delta x_{cell}}{2} \nabla_x W_k \\
W_{bottom}^k &= W_k^c - \frac{\Delta y_{cell}}{2} \nabla_y W_k \\
W_{left}^k &= W_k^c - \frac{\Delta x_{cell}}{2} \nabla_x W_k.
\end{align*}
\]
One option in determining the solution-gradient information relies on a suitable path integral about the cell of interest. In this option, the gradient of a quantity $W_k$ in a cell is determined by

$$\nabla W_k = \frac{1}{A_\Omega} \int_{\partial\Omega} W_k \hat{n} dl,$$  \hspace{1cm} (3.4)

where $A_\Omega$ is the area enclosed by the path of integration, $\partial\Omega$. The path for the integration is constructed by connecting the centroid of neighboring cells in a counterclockwise fashion. In most cells, that path is determined by finding the nearest neighbor in each of eight directions (northwest, west, southwest, south, southeast, east, northeast, north). This “normal” path is depicted in Figure 3.3.

An “altered” path is needed for a variety of special geometric configurations. These cases fall into one or both of two general categories. The first occurs when a neighbor in one of the eight directions has children. Here, the appropriate children of the neighbor are used in the path. The second occurs when a neighbor is not a valid cell due to a body or outer boundary. Then, the cell itself is used in the path instead of that neighbor. Near a body, as few as four cells may be included in the path, including the cell itself. Some examples of these “altered” paths are shown in Figure 3.4.
Figure 3.4: “Altered” Paths
These “altered” paths can lead to a less accurate approximation to the gradient than those calculated with “normal” paths. However, the reconstruction scheme described retains the property of an exact reconstruction of a linear function, even in regions where “altered” paths are used.

Once the cells in the path are determined, the path integral is carried out numerically. The area inside the path is calculated by summing the areas of the triangles formed by connecting the centroids in the path to the centroid of the cell for which the gradient is being calculated.

### 3.1.2 Least Squares

Another option for determining the solution-gradient information relies on the solution of a weighted least-squares system [3]. The weighted least-squares system being solved for the gradient of \( u \) is

\[
\mathcal{L} \nabla u = f
\]

\[
\mathcal{L} = \begin{pmatrix}
  w_1 \Delta x_1 & w_1 \Delta y_1 \\
  \vdots & \vdots \\
  w_N \Delta x_N & w_N \Delta y_N
\end{pmatrix}
\]

\[
f = \begin{pmatrix}
  w_1 \Delta u_1 \\
  \vdots \\
  w_N \Delta u_N
\end{pmatrix}
\]

where

\[
\Delta x_i = x_i - x_0
\]

\[
\Delta y_i = y_i - y_0
\]

\[
\Delta u_i = u_i - u_0
\]

and the points are numbered so that 0 is the origin. The solution to the system is

\[
\nabla u = \mathcal{L}^+ f
\]
\[
\mathcal{L}^+ = \frac{1}{\Delta} \left( \begin{pmatrix} L_1^T \left( L_2^T L_1 \right) - L_2^T \left( L_1^T L_2 \right) \\ L_2^T \left( L_1^T L_1 \right) - L_1^T \left( L_1^T L_2 \right) \end{pmatrix} \right)
\] (3.7)

where \( L_1 \) and \( L_2 \) are the first and second columns, respectively, of \( \mathcal{L} \) and

\[
\triangle = \left( L_1^T L_1 \right) \left( L_2^T L_2 \right) - \left( L_1^T L_2 \right)^2 .
\] (3.8)

To solve for the gradient of \( W_k \) for \( N \) neighbors, Equation 3.7 is expanded and split into \( X \) and \( Y \) components as

\[
\nabla_x W_k = \left( \frac{1}{c_1 c_3 - c_2^2} \right) \sum_{i=1}^{N} \left( w_i^2 \left[ c_3(x_i - x_0) - c_2(y_i - y_0) \right] \left( W_i^k - W_0^k \right) \right)
\]

\[
\nabla_y W_k = \left( \frac{1}{c_1 c_3 - c_2^2} \right) \sum_{i=1}^{N} \left( w_i^2 \left[ c_1(y_i - y_0) - c_2(x_i - x_0) \right] \left( W_i^k - W_0^k \right) \right) .
\] (3.9)

where \( c_1, c_2, \) and \( c_3 \) are defined as

\[
c_1 = \left( L_1^T L_1 \right) = \sum_{i=1}^{N} \left( w_i^2 \left( x_i - x_0 \right)^2 \right)
\]

\[
c_2 = \left( L_1^T L_2 \right) = \sum_{i=1}^{N} \left( w_i^2 \left( x_i - x_0 \right) \left( y_i - y_0 \right) \right)
\]

\[
c_3 = \left( L_1^T L_1 \right) = \sum_{i=1}^{N} \left( w_i^2 \left( y_i - y_0 \right)^2 \right)
\] (3.10)

In the above formulas, \( x_0, y_0, \) and \( W_0^k \) denote the cell about which the gradient \( \nabla W_k \) is computed. The weights \( w_i \) provide an added degree of freedom, allowing the enforcement of, for example, an upwind bias in the gradient. For the calculations in this thesis, however, the weights were all set to one.

This gradient calculation requires a cloud of neighboring cells. These cells do not need to be in any particular order. For simplicity, the cloud was made up of the same neighbors used in the path for the Path Integral Method. A sample numbering of an “altered” configuration is shown in Figure 3.5.

There is no appreciable difference between the results of the two gradient calculation schemes. To show this, gradients were calculated from exact cell-centered values and used to create approximate values at the cell corners. The exact value of the
function $f(x, y) = 2 + \cos(\pi x) + \cos(\pi y)$, was given to each cell in the grid. Then, cell gradients of this function were computed and cell corner values created from the gradients. The error was then computed between the value computed and the cell corners and the known exact value. This process was applied to a sequence of finer uniform grids to compare the error to the grid size. Figure 3.6 gives a log-log plot of grid spacing versus the $L_2$ norm of the error. The slope of each method asymptotes to two, as it should for second order accuracy. The limited cases begin with lower errors, but as the grid spacing decreases, all the errors converge to a single value. The limiting procedure is outlined in Section 3.1.3. The path integral method was used for all cases presented in this dissertation.

3.1.3 Limiting

If the full gradient were used in reconstructing the values at face midpoints, these computed values could fall outside the bounds of the values given, for example those used to compute the gradient. To avoid this, the computed gradients of the cell are limited; that is the primitive variables $W = (\rho, u, v, p)^T$ are reconstructed via

$$W(x, y) = W^c + \phi \nabla W \cdot dr$$

(3.11)

where $\phi$ is a limiter, with a value between zero and one. In regions where $\phi = 1$, the reconstruction used is linear, and the truncation error is $O(h^2)$; in regions where
\[
\phi = 0, \text{ the reconstruction used is piecewise constant, and the truncation error is } O(h) [3].
\]

The limiter used is a diffusive limiter of the minmod variety [53], and is defined as

\[
\phi = \min \left\{ \frac{1}{\min_k \left( \frac{|W_k - \max_{\text{path}}(W_k)|}{|W_k - \max_{\text{cell}}(W_k)|} \right)} \right\}. \tag{3.12}
\]

The minimum and maximum over the path are found by examining the values of \( W_k \) used to compute the gradient; the minimum and maximum over the cell are found by using the gradient to reconstruct \( W_k \) at the corners of the cell. Thus, the limiter acts to ensure that the values of \( W_k \) at the nodes of the cell for which the gradient is being calculated are bounded by the values of \( W_k \) that are used in calculating the gradient. This limiting procedure is slightly modified for cut cells. For cut cells, the minimum and maximum over the cell is found using the values at the corners of the cell, excluding those on the body. If the values on the body were included in the
maximum over the cell, the scheme would tend to revert to first order in the cells
on the body. In standard MUSCL-type schemes [60], a separate limiter is typically
used for each variable, and for each grid direction, resulting in eight limiters for each
cell. However, using a single limiter for the gradient of the vector $\mathbf{W}$ was found to
give superior convergence.

Unfortunately, limiting can seriously hamper the convergence to a steady state,
with the nonlinearity of the scheme resulting in limit cycles. The limiter values in
smooth regions of the flow can oscillate. To combat this problem, the limiter values
are “frozen” after a certain point in the convergence, using previously stored values
of the limiter rather than recomputing them at each time step. Freezing the limiters
allows the residuals to converge to machine zero. Even though the limiter is “frozen”,
the values of density and pressure are still checked to be sure that they do not dip
below zero. If they do, a more restrictive limiter is used for the cell for that one
time-step.

A new alternative to limiter freezing was recently proposed by Venkatakrish-
nan [63]. The limiter in Equation 3.12 is modified to be more like the Van Albada
limiter [53]. This new limiter prevents the limiter from acting in smooth regions of
the flow, thus allowing the solution to converge. This limiter is, however, a little
more diffusive, allowing some smearing of the solution near shocks. Also, a tuning
value is needed to tell the limiter how strictly to enforce limiting.

Typical values of the limiter, $\phi(x,y)$, are shown for a transonic airfoil case. Fig-
ure 3.7 gives the Mach contours for this case and Figure 3.8 shows the limiter $\phi$. As
can be seen, a linear reconstruction ($\phi = 1$) is used nearly everywhere. In the imme-
diate vicinity of the shocks and the wake, the limiter reduces the order of accuracy of
the scheme. However, the percentage of cells in which the limiter is less than one is
extremely small. The limiter values on the body are one, allowing the full accuracy
of the scheme there. This is true even for the smallest cut cells.
Figure 3.7: Mach Contours — Transonic Airfoil

Figure 3.8: Limiter Values — Transonic Airfoil
3.2 Flux Formulation

The flux formulation described here corresponds to the solution of the Euler equations in two dimensions. An update to a cell is provided by the summation of the fluxes on the faces making up the cell. After describing the Euler equations in the proper form, two possible methods to compute the flux function will be examined. The first method is Roe’s Approximate Riemann Solver and second is Van Leer’s Flux Vector Splitting. In this code, the former was used for most of the cases considered.

3.2.1 Euler Equations

The finite-volume form of the Euler equations in two dimensions can be written as

$$\frac{dU}{dt} = -\frac{1}{A} \sum_{faces} (F \Delta y - G \Delta x)$$

(3.13)

where $A$ is the area of the cell, $\Delta x$ and $\Delta y$ are the changes of $x$ and $y$ along a face (defined so that the integral is carried out in a counter-clockwise direction), and $U$, $F$ and $G$ are defined as

$$U = \begin{pmatrix} 
\rho \\
\rho u \\
\rho v \\
\rho E 
\end{pmatrix}$$

$$F = \begin{pmatrix} 
\rho u \\
\rho u^2 + p \\
\rho uv \\
\rho uH 
\end{pmatrix}$$
\[
G = \begin{pmatrix}
\rho v \\
\rho v u \\
\rho v^2 + p \\
\rho v H
\end{pmatrix}
\] (3.14)

where \( \rho \) is the density, \( u \) and \( v \) are the components of velocity in the \( x \) and \( y \) directions, \( E \) is the specific total energy, and \( H = E + \frac{\rho}{\rho} \) is the specific total enthalpy.

Defining the face length and the normal and tangential velocities as

\[
\Delta s = \sqrt{(\Delta x)^2 + (\Delta y)^2}
\] (3.15)

and

\[
\begin{align*}
    u_n &= \frac{(u\Delta y - v\Delta x)}{\Delta s} \\
    u_t &= \frac{(u\Delta x + v\Delta y)}{\Delta s},
\end{align*}
\] (3.16)

the flux through a face may be written as

\[
(F \Delta y - G \Delta x) = \begin{pmatrix}
\rho u_n \\
\rho u_n u + p \frac{\Delta y}{\Delta s} \\
\rho u_n v - p \frac{\Delta x}{\Delta s} \\
\rho u_n H
\end{pmatrix} \Delta s \equiv \Phi \Delta s.
\] (3.17)

The function \( \Phi \) is a formulation of the flux through a face based on the reconstruction of flow quantities to the “left” and “right” side of the face.

For each cell, the face fluxes \( \Phi \) are summed to give the residual for the cell,

\[
\text{Res}(U) = - \sum_{faces} \Phi \Delta s.
\] (3.18)

These residuals are then integrated in time, as described in section 3.3.

For axisymmetric flow, the two dimensional Euler equations must include a source term and can be written as

\[
\frac{dU}{dt} = -\frac{1}{A\hat{y}} \sum_{faces} y (F \Delta y - G \Delta x) + S
\] (3.19)
with $S$ defined as

$$S = \begin{pmatrix} 0 \\ 0 \\ \bar{p}/\bar{y} \\ 0 \end{pmatrix}. \quad (3.20)$$

Here, $y$ is the midpoint of the face, $\bar{y}$ is the $y$ centroid of the cell, which takes into account that the cell has been rotated about the axis of symmetry, and $\bar{p}$ is the cell pressure at the cell centroid. Instead of generating a new function $\Phi^*$, our original $\Phi$ from the two dimensional case can be used. Then equation 3.18 becomes

$$\text{Res}(U) = -\left( \frac{1}{\bar{y}} \sum_{faces} y \Phi \Delta s \right) + AS. \quad (3.21)$$

With this change, our function $\Phi$ can now be developed in the next sections without considering the special case of axisymmetric flow.

### 3.2.2 Roe’s Flux-Difference Splitting

The flux through a face is a function of the values at the face midpoint, given by the reconstruction in the cells to the “left” and “right” of the face. Using Roe’s approximate Riemann solver, this flux function $\Phi$ is

$$\Phi(U_L, U_R) = \frac{1}{2}[\Phi(U_L) + \Phi(U_R)] - \frac{1}{2} \sum_{k=1}^{4} |\hat{a}_k|^* \Delta V_k \hat{R}_k \quad (3.22)$$

with

$$\hat{a} = \begin{pmatrix} \hat{u}_n - \hat{c} \\ \hat{u}_n \\ \hat{u}_n \\ \hat{u}_n + \hat{c} \end{pmatrix}$$
\[
\Delta V = \begin{pmatrix}
\frac{\Delta p - \bar{\rho} \Delta u_n}{2e^2} \\
\frac{\bar{\rho} \Delta u_L}{e} \\
\Delta \rho - \frac{\Delta \rho}{e} \\
\frac{\Delta p + \bar{\rho} \Delta u_n}{2e^2}
\end{pmatrix}
\]

\[
\hat{R} = \begin{pmatrix}
1 & 0 & 1 & 1 \\
\hat{u} - \hat{c} \frac{\Delta u}{\Delta s} & \hat{c} \frac{\Delta u}{\Delta s} & \hat{u} + \hat{c} \frac{\Delta u}{\Delta s} \\
\hat{v} + \hat{c} \frac{\Delta u}{\Delta s} & \hat{c} \frac{\Delta u}{\Delta s} & \hat{v} + \hat{c} \frac{\Delta u}{\Delta s} \\
H - \hat{u}_n \hat{c} & \hat{u}_n \hat{c} & \frac{\rho + \rho'}{2} & H + \hat{u}_n \hat{c}
\end{pmatrix}
\]  

(3.23)

and

\[
\hat{\rho} = \sqrt{\rho_L \rho_R}
\]

\[
\hat{u} = \frac{\sqrt{\rho_L} u_L + \sqrt{\rho_R} u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}
\]

\[
\hat{v} = \frac{\sqrt{\rho_L} v_L + \sqrt{\rho_R} v_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}
\]

\[
\hat{H} = \frac{\sqrt{\rho_L} H_L + \sqrt{\rho_R} H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}
\]

(3.24)

where \( \hat{c} \), \( \hat{u}_n \), and \( \hat{u}_t \) are calculated directly from \( \hat{\rho} \), \( \hat{u} \), \( \hat{v} \), and \( \hat{H} \). The flux difference terms (the summation in Equation 3.22) provide the upwind character which stabilizes the scheme.

To prevent expansion shocks, an entropy fix is imposed [61]. The entropy controlling term, \( \left| \hat{a}^{(k)} \right| \), is replaced by a smoothed value, \( \left| \hat{a}^{(k)} \right|^{\text{s}} \), for the two acoustic waves \( (k = 1, k = 4) \). For those two waves,

\[
\left| \hat{a}^{(k)} \right|^{\text{s}} = \begin{cases}
\left| \hat{a}^{(k)} \right| & \left| \hat{a}^{(k)} \right| \geq \frac{1}{2} \delta a^{(k)} \\
\frac{\left( \hat{a}^{(k)} \right)^2}{\delta a^{(k)}} + \frac{1}{4} \delta a^{(k)} & \left| \hat{a}^{(k)} \right| \leq \frac{1}{2} \delta a^{(k)}
\end{cases}
\]

\[
\delta a^{(k)} = \max \left( 4 \Delta a^{(k)}, 0 \right), \Delta a^{(k)} = a^{(k)} - a^{(k)}.
\]

(3.25)
3.2.3 Van Leer’s Flux Splitting

Using Van Leer’s flux vector splitting, the flux function, $\Phi$, is defined as

$$\Phi = T^{-1} \left( H^+ (U_L) + H^- (U_R) \right),$$  \hspace{1cm} (3.26)

where

$$T = \begin{pmatrix} 
1 & 0 & 0 & 0 \\
0 & \frac{\Delta y}{\Delta s} & -\frac{\Delta x}{\Delta s} & 0 \\
0 & \frac{\Delta x}{\Delta s} & \frac{\Delta y}{\Delta s} & 0 \\
0 & 0 & 0 & 1 
\end{pmatrix} \quad \text{and} \quad H = \begin{pmatrix} 
\rho u_n \\
\rho u_n^2 + p \\
\rho u_n u_t \\
\rho u_n H 
\end{pmatrix}.$$  \hspace{1cm} (3.27)

The split fluxes $H^+$ and $H^-$ are defined for various normal Mach numbers as

$\begin{align*}
M_n & \geq 1, \quad H^+ = H, \quad H^- = 0 \\
M_n & \leq -1, \quad H^+ = 0, \quad H^- = H \\
|M_n| & \leq 1, \quad H^+ = \begin{pmatrix} h^+_1 \\
h^+_2 \\
h^+_3 \\
h^+_4 \end{pmatrix} = \begin{pmatrix} \pm \rho c (M_n \pm 1)^2 / 4 \\
h^+_1 c \left[ (\gamma - 1) M_n \pm 2 / \gamma \right] \\
h^+_1 c M_t \\
h^+_1 c^2 \left[ (\gamma - 1) M_n \pm 2 / [2(\gamma^2 - 1)] + M_t^2 / 2 \right] \end{pmatrix}. \hspace{1cm} (3.28)
\end{align*}$

3.3 Time Stepping

The time-stepping scheme used is one of the optimally-smoothing multi-stage schemes developed by Tai [54, 62]. A multigrid convergence method is also used to increase the convergence rate of the solution.

3.3.1 Multi-Stage Time Stepping

The general $m$-stage scheme is defined as

$$U^{(0)} = U^n$$
Table 3.1: Multi-stage coefficients with a first order spatial discretization for optimal damping of the frequency range $\pi/2 \leq k \Delta x \leq \pi$

<table>
<thead>
<tr>
<th></th>
<th>Stages</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nu$</td>
<td>1.0</td>
<td>1.5</td>
<td>2.0</td>
<td>2.5</td>
<td>3.0</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>0.3333</td>
<td>0.1481</td>
<td>0.0833</td>
<td>0.0533</td>
<td>0.0370</td>
</tr>
<tr>
<td>$\alpha_2$</td>
<td>1.0000</td>
<td>0.4000</td>
<td>0.2069</td>
<td>0.1263</td>
<td>0.0851</td>
</tr>
<tr>
<td>$\alpha_3$</td>
<td>1.0000</td>
<td>0.4265</td>
<td>0.2375</td>
<td>0.1521</td>
<td></td>
</tr>
<tr>
<td>$\alpha_4$</td>
<td>1.0000</td>
<td>0.4414</td>
<td>0.2562</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\alpha_5$</td>
<td>1.0000</td>
<td>0.4512</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\alpha_6$</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Multi-stage coefficients with a second order spatial discretization for optimal damping of the frequency range $\pi/2 \leq k \Delta x \leq \pi$

<table>
<thead>
<tr>
<th></th>
<th>Stages</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nu$</td>
<td>0.4693</td>
<td>0.6936</td>
<td>0.9214</td>
<td>1.1508</td>
<td>1.3805</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>0.4242</td>
<td>0.1918</td>
<td>0.1084</td>
<td>0.0695</td>
<td>0.0482</td>
</tr>
<tr>
<td>$\alpha_2$</td>
<td>1.0000</td>
<td>0.4929</td>
<td>0.2602</td>
<td>0.1602</td>
<td>0.1085</td>
</tr>
<tr>
<td>$\alpha_3$</td>
<td>1.0000</td>
<td>0.5052</td>
<td>0.2898</td>
<td>0.1885</td>
<td></td>
</tr>
<tr>
<td>$\alpha_4$</td>
<td>1.0000</td>
<td>0.5060</td>
<td>0.3050</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\alpha_5$</td>
<td>1.0000</td>
<td>0.5063</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\alpha_6$</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where the $\nu$ and $\alpha_k$ are chosen to damp optimally a chosen range of Fourier modes. Multi-stage coefficients for optimal damping of Fourier modes in the range $\pi/2 \leq k \Delta x \leq \pi$ with first and second order spatial schemes are listed in Tables 3.1 and 3.2. Only the high frequency Fourier modes were chosen to be damped since the lower frequency modes are damped by the multigrid procedure. For the results in this thesis, the 5-stage scheme are used, with the coefficients for a second order spatial scheme (Table 3.2) chosen.
Since steady solutions of the Euler equations are computed, local time-stepping is used, and indeed is necessary, since the time-stepping is dependent on the cell size, and the grids generated have extremely large differences in area from cell to cell, due to the cut cells.

3.3.2 Multigrid Convergence Acceleration

The improved convergence of multigrid is due in part to the damping of low frequency errors on the finer grids by the use of coarser grids, the parents of the fine grid cells. Because these low frequency errors appear on the coarser grids as high frequencies, they are efficiently damped before being sent back to the finer grids. Another factor in the improved convergence is the fact that waves move farther in one time-step on the coarser grids. Both damping and convection contribute to the multigrid speed-up.

Since the Quadtree data structure inherently provides inter-level cell communication, a sequence of multigrid levels can easily be constructed which use the tree to communicate. Thus most of the interface between various multigrid levels has already been done. Most multigrid algorithms are applied to a succession of coarser uniform grids. This leaves each cell in the tree with a unique place in the multigrid cycle. For this unstructured-grid approach, not only are the cells on the finest grid not of the same cell level in the tree, but the cells on subsequent multigrid levels are not of the same cell level in the tree. This will cause a little complication to the multigrid algorithm, and will eventually lead to implementation of a “saw-tooth” multigrid cycle instead of the more conventional “V” cycle.

A cell’s level in the tree is defined as the number of successive parents to the root cell. In contrast, the multigrid level is defined as the collection of cells used in a given level of the multigrid algorithm. For \( M \) multigrid levels, the cells on multigrid level \( M \) correspond to the finest level, including all the computational cells
without children, i.e. the leaves of the quadtree. If \( N \) is the maximum cell level in multigrid level \( M \). The grid corresponding to multigrid level \( M - 1 \) is defined as all the computational cells on cell level \( N - 1 \) and all computational cells with a cell level less than \( N - 1 \) having no children. Multigrid level \( M - 2 \) down to multigrid level 1 are constructed similar to level \( M - 1 \). It should be noted, however, that some cells are included in more than one multigrid level, specifically cells without children which are at a cell level less than \( N \). To visualize what is happening here, a one-dimensional case is given in Figure 3.9. The one-dimensional grid is drawn so that the coarser parent cells can be seen. If cell 1 is the root cell, then the maximum cell level in the tree, \( N \), is three. If three multigrid levels, \( M \), are used, cell number 5 is used on two multigrid levels and cell number 3 is used on three multigrid levels. Special care will need to be taken in these cells.

Since the multigrid algorithm makes use of cells with children, accurate state values are needed for these cells. The values for these cells depend on the values of their children. The conserved variables are transferred from the children (grid level \( L + 1 \)) to the parents (grid level \( L \)) by an area weighted collection operator, \( I_{L+1}^L \),
which maintains conservation.

\[ I^L_{L+1} (U_{L+1}) = \frac{\sum A U_{L+1}}{\sum A} \quad (3.30) \]

The residual collection operator, \( I^L_{L+1} \), does not use area weighting, but rather just sums the residuals of its children.

\[ I^L_{L+1} (\text{Res}_{L+1}) = \sum \text{Res}_{L+1} \quad (3.31) \]

These two collection operators make up a process often called the \emph{restriction} of flow variables from one grid to a coarser grid.

A forcing function, \( \text{FF}_L \), is defined to drive the coarser grid by the accuracy of the finer grids. All cells on multigrid level \( M \) have \( \text{FF}_L \) set to zero. For cells on a multigrid level less than \( M \), two different forcing functions may be used. Cells which have children have \( \text{FF}_L \) defined as

\[
\text{FF}_L = I^L_{L+1} (\text{Res} (U_{L+1}) + \text{FF}_{L+1}) - \text{Res} (U_L) \quad (3.32)
\]

and cells without children have \( \text{FF}_L \) defined as

\[
\text{FF}_L = (\text{Res} (U_{L+1}) + \text{FF}_{L+1}) - \text{Res} (U_L) \quad . \quad (3.33)
\]

This forcing function gives the desired property of preserving a conserved solution on the finest grid.

The time-stepping scheme from equation 3.29 then becomes

\[
\begin{align*}
U^{(0)} & = U^n \\
U^{(k)} & = U^{(0)} + \nu \frac{\alpha_k \Delta t}{A} \text{Res} (U^{(k-1)} + \text{FF}) , k = 1 \ldots m \\
U^{n+1} & = U^{(m)} \quad (3.34)
\end{align*}
\]

for each multigrid level. Regardless of whether the scheme on multigrid level \( M \) has a first or second order spatial accuracy, each update on multigrid levels less than \( M \)
uses a first order spatial operator (ie no linear reconstruction). This allows a greater smoothing of the high frequency errors on the coarser grid while still maintaining the accuracy on the fine grid. Once the coarse grid state vectors, $U$, have been updated, the corrections need to be sent back to the finer grids. The correction to $U_{L+1}$ is defined as

$$U_{L+1}^{n+1} = U_{L+1}^{n+1} + \nabla \left( U_{L+1}^{n+1} - U_{L}^{n} \right) \cdot dr$$  \hspace{1cm} (3.35)$$

where the gradient of the difference $U_{L+1}^{n+1} - U_{L}^{n}$ is computed in the same way that the gradients of the state vector are computed, a path integral reconstruction. This correction process is often referred to as the prolongation of the the error correction.

The multigrid algorithm constructed from these operators is depicted in Figure 3.10. The multigrid algorithm begins by updating $U_M$ on multigrid level $M$ (Equation 3.29). Then on multigrid level $M - 1$, accurate state values, $U_{M-1}$, are created (Equation 3.30), the forcing function, $FF_{M-1}$, is computed (Equation 3.31), and $U_{M-1}$ updated in time (Equation 3.34). This process proceeds by level down to multigrid level 1. When multigrid level 1 is reached, the correction is sent to
the state vector, $U$, up to multigrid level 2 (Equation 3.35). At this point either a “saw-tooth” or “V” cycle must be chosen. The saw-tooth procedure dictates just passing the correction level by level back to multigrid level $M$, or the finest grid, before starting the procedure over (again Equation 3.35). On the other hand, the $V$ cycle requires that $U$ be updated in time (Equation 3.34) before the correction is passed up to the next level. However, the update requires knowing the forcing function used for that cell on the way down. This can be seen by again examining Figure 3.9. Cell number 3 is used in three different multigrid levels. The forcing function on multigrid level 3 is zero, but the forcing functions on multigrid levels 2 and 3 are not zero, and are unique values. Two different forcing functions need to be stored. It is common in many cases for a few cells to be used on all of the multigrid levels, requiring a different forcing function on each level. Storing each of these forcing functions can lead to a huge increase in the storage needs of the multigrid algorithm. The $V$ cycle uses the forcing function for each cell twice, once in the restriction stage and once in the prolongation stage. However, a saw-tooth cycle only uses the forcing function in the restriction stage. When a cell is included in two multigrid levels, the first forcing function for the cell is computed normally. When the next multigrid level needs to create a forcing function for that cell, the value already stored is used in computing the next value, which will be stored in its place. Thus, by using a saw-tooth multigrid algorithm, no increase in storage is needed for these special case cells. This choice does not, however, mean that the convergence acceleration of the multigrid algorithm is substantially reduced, since the $V$ cycle requires more work to compute. A comparison of saw-tooth and $V$ cycle results is included in the multigrid validation study, Section 3.6.3. The saw-tooth multigrid method was implemented for the cases in this thesis.
3.4 Boundary Procedures

There are two kinds of boundary conditions which must be implemented, from a data structure point of view. The first is the “far-field” boundary condition. The flow solver needs to know how to handle flow through the outer domain of a flowfield when no body exists on that boundary. The second type of boundary condition is needed for cells which are cut by a body. The types of conditions which can be set are very similar for each of these, although the implementation is quite different.

3.4.1 Far-Field Boundary Conditions

The computational domain is surrounded by a ring of “ghost” cells used to implement the far-field boundary conditions. The generation of these ghost cells and how they relate to the computational cells was discussed in Section 2.2. The state values stored in the ghost cells may or may not be used at all, depending on the type of boundary condition imposed there. Note that ghost cells are not used in the gradient calculation for cells just inside the computational flow field. The gradient calculation is the same no matter what type of boundary condition exists, either in the far-field or on the body.

A number of types of far-field boundary conditions can be implemented. The most common is simply imposing a freestream condition which completely surrounds the flow. A more general variety of this requires specifying a function of \( x \) and \( y \) in the ghost cells. When a computational cell on the outer boundary requires a flux for the face on the outer boundary, the outside state is specified by the state in the ghost cell neighbor. The grid smoothing routine makes sure that the cell level of ghost cells will always equal the cell level of its neighbor in the computational flow. Thus the function representing the outer boundary condition is given to the resolution of the grid at that point. The boundary conditions may also change with time. The values stored in the ghost cells can be updated each iteration.
Other boundary conditions available for the ghost cells include an inviscid reflection and a periodic condition, both of which will be discussed in more detail with the body-cut boundary conditions.

3.4.2 Body-Cut Boundary Conditions

The second type of boundary condition is required for cells which are cut by a body. When a cell is cut, a face is created on the body which cut it. Unlike the ghost cells, there is no unique cell directly on the other side of this face with a state value stored. Instead, the right state required by the flux routine is generated from either the cell itself or a condition associated with the body which cuts the particular cell. For instance, the most common boundary condition on cut cells is simply an inviscid reflection. The flux for the body face may now be computed in two ways. First, the velocity normal to the face may be assumed to be zero and only the pressure flux computed. This condition is only actually true when the flow has reached a steady state, although significant body curvature may still allow a non-zero normal velocity. A more robust method for computing the flux on the body face requires simply “reflecting” the velocities about the face and using the same pressure and density to compute the flux as any normal face with two valid states. This reflection is depicted in Figure 3.11. The present code uses this second form of body face flux.
The body face may also have a fixed right state similar to the outer boundary condition. The difference is that the value used is stored with the body which cuts the cell. Each body may have a different type of boundary condition associated with it. If different conditions are needed for a single body, the body can be made up of any number of sub-bodies, each with its own boundary condition. For example, this condition is useful for mass injection through a surface.

A final variety of boundary condition is the periodic condition. Here a period is specified by a change in the \( X \) and/or \( Y \) directions. Then, when the right state is required for a body face, the \( x \) and \( y \) offsets are added to the face midpoint, the cell containing this point is found, and the right state is computed. In general, the cell containing this offset point will be found on a body identical to the first, only offset by the period. Cases may be constructed in which this is not true however. The only requirement is that the offset point be found within a valid computational cell. Figure 3.12 gives a simple example.

3.5 Post-Processing

Post-processing requires transferring the known cell-centered values to nodal values for plotting as accurately as possible. The same limited cell gradients used in the flow solver are used to extrapolate to each cell’s nodes as shown in Figure 3.13. The function as reconstructed at the nodes is multivalued; there is one value resulting from the representation in each cell sharing that node. Also, nodes on a reflecting body may have the reflection of the sent value as well. These multiple values at each node are averaged to yield a single accurate, bounded value. This yields a state vector at each node, allowing any desired quantity to be computed and used for plotting.
Figure 3.12: Example of Periodic Boundary Condition

Figure 3.13: Obtaining Post-Processed Nodal Values
3.6 Validation

In order to validate the method described in this thesis, a careful examination of the convergence and accuracy will be given. First, the convergence acceleration of multigrid will be examined, along with comparisons of the saw-tooth and V cycle algorithms. Then the accuracy of some special boundary conditions will be given. Finally, a detailed order of accuracy study will show that the scheme is second order accurate, and that the magnitude of the error is acceptable.

3.6.1 Small Cut Cells

A common problem with Cartesian grid methods is that very small cells can exist in the grid due to the somewhat arbitrary way that the grid intersects the body. In fact, cells are commonly found at an area on the order of $10^{-4}$ times that of a neighboring uncut cell. Since the code is designed for steady-state solutions, local time-stepping may be used to remove the stiffness due to these small cells. A difference in time-steps is not the only issue raised by the small cells, however. Depending on the spatial discretization used in the update schemes, the values in the small cells can decouple from those in their neighbors, leading to “spikes” in the results in the vicinity of the small cells. Even with very sophisticated schemes, such as that proposed by LeVeque [8], cells smaller in area than 1% of that of a neighbor can lead to problems. The solution to this problem adopted in this work is to use a spatial discretization suited to unstructured grids, specifically the reconstruction approach of Barth [3]. This combination of the gradient reconstruction and the local time-stepping allowed the solutions to converge as accurately as if the small cells did not exist. Although this work consists primarily of steady flow cases, if unsteady cases need study, a uniform time stepping could be used with some type of cell merging to allow the solution to converge in a reasonable time [5].
3.6.2 Multigrid Grid Problems

A troubling grid feature which leads to some difficulty is the existence of cells which would violate the data structure, specifically cells which are cut more than once. A description of these cells is given in the write-up of the smoothing procedure in Section 2.5. The grid smoothing removes the feature from the finest grid. However, multigrid makes use of the parents of these cells in creating a coarser grid to damp out the low frequency errors. In order to make use of the multigrid convergence acceleration for geometries with this grid problem, a “cone of influence” approach was used around these bad cells. First, each cell on the direct parental succession from the original problem cell to the root was flagged as a problem cell. Then any neighbor to one of these problem cells is handled in a special way when a multigrid update is needed for that cell. Specifically, these cells receive an accurate restriction of state values from their children, but they are not advanced in time. This is because no state value is available one of the cell’s neighbors, thus no flux can be computed on the face shared with that neighbor.

These cells handled specially make up the “cone of influence” of the original problem cell. Figure 3.14 gives an example of such a grid. Part (a) of this figure shows a portion of a grid and data tree in which one cell violates the data structure. Smoothing of the grid results in the grid and data tree given in part (b). Part (c) again gives the data tree, but with each problem cell and “cone of influence” cell identified. Note that no cells without children are flagged. This process only affects the coarser grids used for multigrid.

3.6.3 Multigrid Convergence Study

In examining the convergence acceleration benefits of the multigrid algorithm, it is first important to determine the differences in convergence characteristics between a saw-tooth and V cycle algorithm. When that is completed, the multigrid convergence
Figure 3.14: Multigrid Cone of Influence
can be studied in greater detail.

To see how the saw-tooth and V cycles compare, a NACA 0012 airfoil was run at both subsonic ($M = 0.2$ and $\alpha = 0$) and transonic ($M = 0.85$ and $\alpha = 1$) speeds using five multigrid levels. Figure 3.15 shows the convergence history for the subsonic case. The plot includes convergence for: a 5-stage V cycle, a 5-stage saw-tooth cycle, and no multigrid. The V cycle shows a slightly better convergence rate than the saw-tooth cycle. However, since the V cycle is doing more work, it is not a fair comparison. The convergence history based on the number of residual calculations is plotted in Figure 3.16. Here, the saw-tooth cycle actually has a better convergence rate.

Figure 3.17 shows the convergence history for the transonic case. The plot includes convergence for: a 5-stage V cycle, a 5-stage saw-tooth cycle, and no multigrid. The V cycle shows a slightly better convergence rate than the saw-tooth method based on cycles. But when convergence is based on the number of residual calculations, the saw-tooth cycle actually has a better convergence rate, given in Figure 3.18.
Figure 3.16: Subsonic Convergence History Based on Residual Calculations

Figure 3.17: Transonic Convergence History Based on Multigrid Cycles
Figure 3.18: Transonic Convergence History Based on Residual Calculations

Not all cases will necessarily converge more quickly with a saw-tooth cycle, but it is at least very near to the convergence of a more conventional V cycle. Because the saw-tooth cycle is also much more memory efficient to implement in this code, a saw-tooth multigrid cycle will be used for the rest of the thesis.

An important variable yet to be determined is the number of multigrid levels to use. Any number can be used, provided that many cell levels in the tree exist. At some point the improvement in convergence becomes negligible and the extra work required hurts the time to converge a solution. To see how the number of multigrid levels affect convergence, a NACA 0012 airfoil at \( M = 0.85 \) and \( \alpha = 1 \) was run using between one and seven multigrid levels. Figure 3.19 shows the convergence history based on the number of multigrid cycles.

As can be seen, the convergence rates are nearly identical for five through seven multigrid levels. But, since the work required increases with the number of multigrid levels, the solutions will require increasing amounts of time. The convergence history based on the number of residual calculations is plotted in Figure 3.20. Note that
the optimal number of multigrid levels for this case is four. If more levels are used, the increased cost outweighs the benefits. The number of residual calculations is very closely related to the time the program requires to run. The optimal number of multigrid levels is again four on the convergence history plot based on run time in Figure 3.21.

3.6.4 Boundary Condition Study

Two special types of boundary conditions are examined to determine their accuracy. The first is the simplest far-field boundary condition for a wing, without using a circulation condition. To show the effect of this outer boundary condition on the solution, the steady-state flow was computed about a NACA 0012 airfoil at $M_\infty = 0.85$ and $\alpha = 1^\circ$. This case was run for outer boundaries ranging from a four-chord to a 2048-chord radius. Geometry-based refinement was carried out so as to ensure an equivalent grid in the vicinity of the airfoil for each case. Three levels of solution-based refinement were also done for each case. As can be seen in
Figure 3.20: Convergence History Based on Residual Calculations

Figure 3.21: Convergence History Based on Run Time
Figure 3.22: Lift, Drag, and Number of Cells vs. Boundary Radius

Figure 3.22, this wide range of outer boundary radius resulted in only a small change in the total number of computational cells. Even though only free-stream flow was used on the outer boundary, the lift and drag are converged by the 512 chord case. Thus, by use of grid refinement, simple outer boundary conditions may be enforced, at a radius far from the body, at very little computational cost.

Next, the periodic boundary condition is examined. To show the effect of this condition, consider a sequence of NACA 0012 airfoils at a distance of $1\frac{1}{2}$ chords apart vertically. Since the solution would seem periodic with inviscid walls at $\frac{3}{4}$ chord above and below the airfoil, the periodic condition is imposed at at one chord above and $\frac{1}{2}$ chord below the airfoil. For this case, the freestream Mach number is 2.0 and the angle of attack is 0°. Figures 3.23 and 3.24 show the grid and Mach contours for the flow respectively. Note that since the boundary condition is not imposed at the same distance from the airfoil, the grid is not symmetric about the airfoil. This will lead to solutions which are not perfectly symmetric about the airfoil, but are close. The contours also pass smoothly through one boundary and continue out the other.
boundary. A Mach number cross-section plot at $y = 0.6$ and $y = 0.9$ is given in Figure 3.25. These two lines both represent a distance of 0.6 chords from the airfoil in each direction. A good agreement is shown, verifying the accuracy of the periodic condition.

3.6.5 Order of Accuracy Study

A careful accuracy assessment study is needed to verify that the flow solution procedure described behaves as accurately as expected [13]. To do this, three subsonic test cases have been chosen which will show the accuracy of the scheme and its body boundary conditions. These cases will show both the order and the magnitude of the error.

3.6.5.1 Subsonic Non-Lifting Airfoil

The first case consists of a NACA 0012 airfoil at $M_\infty = 0.2$ and $\alpha = 0^\circ$. Since the flow remains subsonic throughout, the value of the limiter is likewise one throughout. To examine the overall order of the scheme, this case was run on increasingly finer
Figure 3.24: Mach Contours for Periodic NACA 0012 Case

Figure 3.25: Mach Number Cross-Sections for Periodic NACA 0012 Case
grids to examine the effect on the drag (which should be zero). Figure 3.26 is a log-log plot of the drag as a function of the size of an uncut cell of the grid. The slope of this curve, for all but the coarsest grid (which has only eight points on the airfoil), is two, demonstrating the second-order global accuracy of the scheme.

3.6.5.2 Subsonic Cylinder

The second case which highlights the magnitude of the error is a subsonic cylinder at $M_\infty = 0.38$. The lift and drag should both be zero and no entropy should be produced at all. The lift does remain zero for all grid resolutions, but drag and entropy are non-zero. The entropy contours for the first of three grids are shown in Figure 3.27. The grid has a spacing of $dh = 0.125$. The entropy contours for grids with $dh = 0.0625$ and $dh = 0.03125$ are given in Figures 3.28 and 3.29. The magnitude of the drag and the maximum entropy drop to an acceptable level for the finest grid. This grid has 124 cells on the body, and fewer than this would not normally be used in practice for a final solution. The entropy and drag for the three
Table 3.3: Subsonic Cylinder - Entropy And Drag For Three Grid Resolutions

<table>
<thead>
<tr>
<th>Body Cells</th>
<th>dh</th>
<th>Max Entropy</th>
<th>Drag</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>0.125</td>
<td>0.0325</td>
<td>0.2097</td>
</tr>
<tr>
<td>60</td>
<td>0.0625</td>
<td>0.0115</td>
<td>0.0261</td>
</tr>
<tr>
<td>124</td>
<td>0.03125</td>
<td>0.0035</td>
<td>0.0111</td>
</tr>
</tbody>
</table>

Figure 3.27: Entropy generation for Subsonic Cylinder – Grid 1

grid resolutions are given in Table 3.3.

3.6.5.3 Subsonic Ellipses At Angle Of Attack

The final case consists of a variety of ellipses at $M_{\infty} = 0.2$ and $\alpha = 5^\circ$ [65]. This case is challenging because many solution techniques produce lift, where none should exist. The degree of symmetry of the stagnation points determine whether the solution will produce the desired zero lift solution. But, determining the proper location of the stagnation points becomes more difficult when the body has a higher degree of curvature. A series of ellipses will be examined with an increasing degree of eccentricity. Specifically, a 1:1, 2:1, 4:1, and a 6:1 ellipse will be examined.
Figure 3.28: Entropy generation for Subsonic Cylinder – Grid 2

Figure 3.29: Entropy generation for Subsonic Cylinder – Grid 3
Figure 3.30: Pressure Coefficient on the Body for Subsonic Ellipses

A 1:1 ellipse, or cylinder, generates no lift by virtue of its perfect symmetry. This will be the base case for comparison. The 2:1 ellipse generates a $C_L$ of 0.0321. A good measure of the symmetry of the solution is the pressure coefficient on the body. This plot should be perfectly symmetric. The pressure coefficient plots for each ellipse will be combined and plotted together in Figure 3.30. The 4:1 ellipse, with its higher curvature, generates a higher $C_L$ of 0.1028. Finally, the 6:1 ellipse, by virtue of its high body curvature, generates the largest $C_L$, 0.1393. Each of these cases have converged, and the $C_L$ does not change with increased refinement.

The grid and pressure contours for the 6:1 ellipse are given in Figures 3.31 and 3.32. The asymmetry is apparent in the contours surrounding the stagnation points. One method for resolving this problem is to impose a Kutta condition manually. This can be done in a variety of ways, one of which is to put a “splitter plate” on the body which will force a stagnation point location. One attempt at placing the splitter plate in the correct location resulted in a reduced $C_L$ of 0.00939. The change to the pressure coefficient on the body is given in Figure 3.33.
Figure 3.31: Grid for Subsonic 6:1 Ellipse

Figure 3.32: Pressure Contours for 6:1 Subsonic Ellipse
Figure 3.33: New Pressure Coefficient on the Body for 6:1 Subsonic Ellipse
CHAPTER IV

RESULTS

A variety of test cases have been carefully chosen to show the effectiveness of the method presented. The cases span a wide range of geometric complexity and flow conditions.

4.1 NACA 0012 Airfoil

The first cases considered are flows over a NACA 0012 airfoil. These NACA airfoil flows were chosen since they are the most common code validation cases for steady, inviscid, compressible flows. The function describing the shape of the airfoil has been discretized to 128 points, which were then splined by the code. These points straddle the $x = 0$ axis with an airfoil chord of one. The outer boundary for each of the cases presented below was set at 128 chords from the airfoil in each direction.

4.1.1 $M_\infty = 0.63, \alpha = 2^\circ$

The first condition imposed on the NACA 0012 airfoil is $M_\infty = 0.63$ and $\alpha = 2^\circ$. This flow is purely subsonic, and no limiting was used. Three solution refinements were applied to the initial grid resulting in the grid pictured in Figure 4.1. The Mach contours are given in Figure 4.2. The maximum Mach number obtained in the flow was 0.978. The values of the pressure coefficient, $C_p$, are shown in Figure 4.3. Note that the curve is very smooth, despite the area variation of the cells cut by the body.
This flow solution has 494 nodes on the body and a resulting lift, $C_L$, of 0.3289 and a drag, $C_D$, of 0.0004. This grid has 10694 cells, so an equivalent structured grid would be 494 by 21 with the same number of cells on the body, or more realistically, 146 by 73. A good structured grid solution produced a $C_L$ of 0.3335, and a $C_D$ of 0.00003 for a 257 by 65 grid [56]. Contours of total pressure loss, $1 - p_0/p_\infty$, are given in Figure 4.4, with the values on the body given in Figure 4.5.

### 4.1.2 $M_\infty = 0.85, \alpha = 1^\circ$

Next the flow solution is obtained for a NACA 0012 at $M_\infty = 0.85$ and $\alpha = 1^\circ$. For these conditions, shocks of comparable strength exist on the upper and lower surfaces of the airfoil, with a shear layer emanating from the trailing edge. Five solution adaptation steps were taken. The grid and Mach contours for this case are pictured in Figures 4.6 and 4.7. The shocks are both well resolved and a crisp shear layer is computed. It should be noted that the contours pass smoothly through grid level changes; neither the flow solver nor the post-processing procedure causes
Figure 4.2: Mach Contours for NACA 0012 at $M_\infty = 0.63, \alpha = 2^\circ$

Figure 4.3: Pressure Coefficient on Body of NACA 0012 at $M_\infty = 0.63, \alpha = 2^\circ$
Figure 4.4: $1 - \frac{p_0}{p_∞}$ Contours for NACA 0012 at $M_∞ = 0.63, \alpha = 2^°$

Figure 4.5: $1 - \frac{p_0}{p_∞}$ on Body of NACA 0012 at $M_∞ = 0.63, \alpha = 2^°$
Figure 4.6: Grid for NACA 0012 at $M_\infty = 0.85$, $\alpha = 1^\circ$

Jumps in solution values at places in the grid where cell level differences occur. The pressure coefficient on the body shown in Figure 4.8 results in a $C_L$ of 0.3682 and a $C_D$ of 0.0599 for 239 nodes on the body. Since the grid has 1351.5 cells, an equivalent structured grid would be 239 by 56 cells with the same number of cells on the body. The finest grid in the AGARD [2] test cases was a 320 by 64 cell grid which produced $C_L = 0.3584$ and $C_D = 0.0580$. Another good structured grid solution yielded a $C_L = 0.3793$ and $C_D = 0.0576$ [56]. The $C_L$ for the solution presented falls between these two results, and also produced sharper shocks and shear.

4.1.3 $M_\infty = 0.95$, $\alpha = 0$

The final NACA solution given was computed at $M_\infty = 0.95$ and $\alpha = 0^\circ$ [64]. The flow has an oblique shock emanating from the trailing edge with a weak normal shock in the wake as shown in the Mach contours in Figure 4.9. The grid for this solution is plotted in Figure 4.10. There are 13230 cells in this grid.

The location of the downstream normal shock is very sensitive to the angle of the
Figure 4.7: Mach Contours for NACA 0012 at $M_\infty = 0.85, \alpha = 1^\circ$

Figure 4.8: Pressure Coefficient on Body of NACA 0012 at $M_\infty = 0.85, \alpha = 1^\circ$
Figure 4.9: Mach Contours for NACA 0012 at $M_\infty = 0.95, \alpha = 0^\circ$

Figure 4.10: Grid for NACA 0012 at $M_\infty = 0.95, \alpha = 0^\circ$
oblique shock. In turn, the angle of the oblique shock is very sensitive to the Mach number on the body just upstream of the trailing edge. Figures 4.11 and 4.12 show a blowup of the Mach contours and grid in the trailing edge region of the flow.

To guarantee an accurate value of Mach number on the body before the oblique shock, the expansion region must have sufficient grid resolution. The cell length-scale weighting described in Section 2.4 ensures that this is the case by refining not just the shocks, but also this expansion region. The position of the normal shock improves with each solution refinement applied, since both the normal shock and the expansion area on the airfoil are refined. A plot of the normal shock location is given in Figure 4.13 as a Mach number cut at $y = 0$ for each of five different solution levels. The shock location converges to a location 3.41 chords behind the trailing edge of the airfoil. A “best” solution was obtained on a 2049 by 765 O-type structured grid which resulted in a shock location 3.35 chords downstream of the trailing edge [64].
Figure 4.12: Detail Grid for NACA 0012 at $M_{\infty} = 0.95, \alpha = 0^\circ$

Figure 4.13: Normal Shock Location for NACA 0012 at $M_{\infty} = 0.95, \alpha = 0^\circ$
4.2 15° Wedge

In this case, the steady-state flow was computed in a channel with a 15° compression corner at \( x = 0.5 \), followed by a 15° expansion corner at \( x = 1.0 \) [29]. The free-stream Mach number is \( M_\infty = 2.0 \). There is an attached shock at the compression corner, which reflects from the top wall, forming a small Mach stem. The shock reflects from the bottom as well, before exiting the channel. The expansion corner acts to weaken the reflected shock. There is also a slip line, emanating from the triple point near the upper wall.

The final grid is shown in Figure 4.14. This plot includes a blowup of the grid in the Mach stem area. Figures 4.15 and 4.16 give the Mach number and pressure contours, again including a blowup of the Mach stem area. The shear that emanates from the triple point of the Mach stem is carried cleanly out through the flow, although weakened by the expansion. The pressure contours pass smoothly through the shear, as they should. The pressure on the upper and lower walls is given in Figure 4.17. Note that the pressure is nearly constant on the incline of the ramp. Cell areas on the incline vary as much as six orders of magnitude from cell to cell without having a detrimental effect on the solution.

This flow can be easily computed using a structured grid, but a very fine grid is needed in order to resolve the Mach stem to the same resolution given. The grid would need to be 256 by 768 cells, for 196,608 cells, an unreasonable number of cells for a simple flow. Only 14,273 cells are used in the grid given.

4.3 Axisymmetric Jet

The cases considered here consist of axisymmetric under-expanded jet flows. These flows can be characterized by: the stream Mach number; the jet Mach number; the jet-to-stream total pressure ratio; and the jet-to-stream total temperature ratio. A total temperature ratio of one was used for each flow considered here. Also, the
Figure 4.14: Grid for 15° Wedge at $M_\infty = 2.0$
Figure 4.15: Mach Contours for 15° Wedge at $M_\infty = 2.0$
Figure 4.16: Pressure Contours for 15° Wedge at $M_\infty = 2.0$
jet Mach number was set equal to the stream Mach number for each case. A single Mach number will be considered here. The primary effect of Mach number, when the jet and stream Mach number are equal, is the strength and scale of the flow. Thus a flow with larger Mach number would have stronger features which would occur farther downstream of the end of the nozzle. A schematic of an under-expanded jet with a Mach disk is pictured in Figure 4.18. These cases highlight the solution-based adaptation of the code, since a wide range of feature strengths are evident in these flows.

4.3.1 $M = 1.25, \frac{P_0}{P_0} = 20$

An under-expanded axisymmetric jet at Mach 1.25 and jet-to-stream total pressure ratio of 20 is considered first. The grid and Mach contours are given in Figures 4.19 and 4.20. The refinement criterion has detected the external shock, the intercepting shock and its reflection, the jet boundary, and the rapid expansion at the nozzle exit. Few of these flow features are visible for plots of density or pressure, since the expansion at the nozzle exit dominates. The contours of density and pressure are shown in Figures 4.21 and 4.22.
Axisymmetric Jet At $M = 1.25, P_{0j}/P_0 = 20$

27045 Computational Cells.

Figure 4.19: Grid for Jet at $M = 1.25, P_{0j}/P_0 = 20$

Axisymmetric Jet At $M = 1.25, P_{0j}/P_0 = 20$
Mach Contours

Figure 4.20: Mach Contours for Jet at $M = 1.25, P_{0j}/P_0 = 20$
Figure 4.21: Density Contours for Jet at $M = 1.25, P_{0j}/P_{0i} = 20$

Figure 4.22: Pressure Contours for Jet at $M = 1.25, P_{0j}/P_{0i} = 20$
4.3.2 $M = 1.25, P_{0j}/P_{0s} = 5$

This case shows the effect of decreasing the jet-to-stream total pressure ratio to 5 from 20. The flow is similar to the previous case, although many of the features are considerably weaker and closer to the nozzle exit. The grid and Mach contours for this case are given in Figures 4.23 and 4.24. Again the flow features are well resolved, despite the disparate strengths of the features.

4.3.3 $M = 1.25, P_{0j}/P_{0s} = 50$

When the jet-to-stream total pressure ratio is increased to 50, the intercepting shock no longer simply reflects off the axis of symmetry; instead, a large Mach disk is formed. The slip line emanating from the Mach triple point is as apparent as the jet boundary. Also, the jet boundary is more highly curved due to the lower pressure of the stream. The grid for this case is pictured in Figure 4.25. Figure 4.26 shows the Mach contours.
Figure 4.24: Mach Contours for Jet at $M = 1.25, \frac{P_{0j}}{P_{0s}} = 5$

Figure 4.25: Grid for Jet at $M = 1.25, \frac{P_{0j}}{P_{0s}} = 50$
Figure 4.26: Mach Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 50$

4.3.4 $M = 1.25, P_{0j}/P_{0s} = 100$

When the jet-to-stream total pressure ratio is increased further to 100, the Mach disk becomes larger, and moves farther downstream of the nozzle lip. The flow is, in all other respects, similar to the previous Mach disk case. Figures 4.27 and 4.28 plot the grid and Mach contours, respectively.

4.4 Multi-Element Airfoils

Three different multi-element airfoils will be examined, each at purely subsonic speeds. These three cases have a higher degree of geometric complexity than the cases already considered, but the grid generation for these configurations is no more complicated than for the simplest of geometries. The initial grids generated show proper resolution of high curvature portions of the bodies, and amply resolve the flow features.
Figure 4.27: Grid for Jet at $M = 1.25, P_{0j}/P_{0s} = 100$

Figure 4.28: Mach Contours for Jet at $M = 1.25, P_{0j}/P_{0s} = 100$
Figure 4.29: Grid (1) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$

### 4.4.1 Three-Element Airfoil

The first multi-element airfoil considered consists of three elements. This configuration was chosen since solution values from Boeing’s TRANAIR code, which solves the full potential equation, were available. The freestream Mach number is 0.2 at $0^\circ$ angle of attack. Due to the high effective camber caused by the flaps, a strong expansion exists at the leading edge of the main element. The flow nearly reaches a Mach number of 1.0 before dropping over the upper surface of the airfoil. Figures 4.29, 4.30, and 4.31 show three different grid plots of the body. This grid has 13933 cells. An equivalent structured grid would be 167 by 83 cells. Mach contours at these three locations are given in Figures 4.32, 4.33, and 4.34. Figure 4.35 shows a few selected streamlines to illustrate further the flowfield.

Figure 4.36 shows the Mach number on the body. This plot includes body values provided by Boeing. The overall agreement is obvious. The main difference is the location of the stagnation point on the forward portion of the main body element.
Figure 4.30: Grid (2) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$

Figure 4.31: Grid (3) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$
Figure 4.32: Mach Contours (1) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$

Figure 4.33: Mach Contours (2) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$
Figure 4.34: Mach Contours (3) for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$

Figure 4.35: Streamlines for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$
The stagnation point produced by this code was slightly farther ahead, resulting in a smaller leading edge expansion and a larger Mach number on the lower surface of the main element. The effect of this difference can also be seen on the pressure coefficient on the body as shown in Figure 4.37. This code produced a slightly smaller pressure coefficient peak at the leading edge. The two flaps, however, agree extremely well with the Boeing results. The computed $C_L$ for this configuration is 5.162; the $C_D$ computed is 0.178.

Again, despite large variations in the cell size on the body (the area of the smallest cut cell is $10^6$ smaller than that of its uncut neighbor) the solution is smooth. The total pressure loss, $1 - \rho_0/\rho_\infty$, is shown in Figure 4.38. The loss is near 0.2%, except for a spike at the trailing edge of each body.

4.4.2 Two-Element Airfoil

This case consists of a two-element Karman-Trefftz airfoil and flap at $M_\infty = 0.125$ and $\alpha = 0^\circ$ [33]. An exact incompressible potential solution exists for this case that
Figure 4.37: Pressure Coefficient on Body for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$

Figure 4.38: $1 - \frac{p_0}{p_0\infty}$ on Body for 3-Element Airfoil at $M = 0.2, \alpha = 0^\circ$
may be used for comparison. At this Mach number, the compressibility effects are very small. The grid for this flow in pictured in Figure 4.39. This grid has 10827 cells. An equivalent structured grid would be 147 by 73 cells. The Mach contours are given in Figure 4.40. The exact surface pressure coefficient is plotted on top of the computed values in Figure 4.41. The solutions show good agreement, with the exception of a stagnation point on the main element slightly forward, resulting in a slightly smaller expansion on the leading edge. The agreement on the rear element is excellent. The computed $C_L$ for this configuration is 1.936, while $C_D$ is 0.0237. The total pressure loss is shown in Figure 4.42. The loss is near zero, except for a spikes at the leading and trailing edges of each body. Significant additional refinement of the leading-edge region does not improve the solution.

4.4.3 Four-Element Airfoil

This case is similar to the previous case in that an exact incompressible solution is known. The freestream Mach number is 0.125 for this four-element airfoil. The grid
Figure 4.40: Mach Contours for 2-Element Airfoil at $M = 0.125, \alpha = 0^\circ$

Figure 4.41: Pressure Coefficient on Body for 2-Element Airfoil at $M = 0.125, \alpha = 0^\circ$
for this airfoil is given in Figure 4.43. More detailed portions of the grid are shown in Figures 4.44 and 4.45. This grid has 14204 cells. An equivalent structured grid would be 168 by 84 cells. The Mach contours corresponding to these grid plots are given in Figures 4.46, 4.47, and 4.48. The known exact solution for the pressure coefficient on the body is again plotted on top of the computed values shown in Figure 4.49. Again, the solutions show good agreement except for the slight difference in the leading edge expansions. The lower surfaces and the trailing edge flaps show excellent agreement, however. Finally, the total pressure loss is shown in Figure 4.50. The loss is around 0.1%, except for the leading and trailing edge spikes on each body.
Figure 4.43: Grid (1) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$

Figure 4.44: Grid (2) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$
Figure 4.45: Grid (3) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$

Figure 4.46: Mach Contours (1) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$
Figure 4.47: Mach Contours (2) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$

Figure 4.48: Mach Contours (3) for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$
Figure 4.49: Pressure Coefficient on Body for 4-Element Airfoil at $M = 0.01, \alpha = 0^\circ$

Figure 4.50: $1 - \frac{p}{p_\infty}$ on Body for 4-Element Airfoil at $M = 0.125, \alpha = 0^\circ$
CHAPTER V

CONCLUSIONS

This thesis is brought to a close with a summary of the contributions of this work as well as some suggestions for ways in which the work could be usefully extended.

5.1 Summary

A method has been developed by which the solution of the steady-state 2-D Euler equations can be obtained using solution-based refinement for flows around bodies with arbitrary geometric complexity.

A quadtree cell-based data structure is used to control the unstructured grid. The data tree is made up of a list of cells with pointers to a cell’s parent and four children (if they exist). Connectivity information is obtained from the data tree via this parent/child relationship.

The initial grid is generated with a minimum of user input for any complex configuration. The user need only specify a set of points defining the bodies, the base grid resolution, and cell size thresholds for the initial grid, cut-cell refinement, and curvature refinement. With proper thresholds, the grid is automatically adapted to the curvature of a body, providing the resolution required to resolve that body adequately.

The grid is also improved through the use of solution-based refinement. Once a solution has converged to a steady state, solution-based gradient information (specif-
ically a blend of $|\nabla \times \mathbf{u}|$ and $|\nabla \cdot \mathbf{u}|$ is used to flag cells for dividing or coarsening. This process ensures that the grid resolution matches the length scales of the local flow field.

Small cut cells are created by the arbitrary way that the Cartesian grid cuts through the body. This problem is easily overcome, however, by using local time-stepping, coupled with a linear reconstruction method. This allows the solution to converge as accurately as if the small cells did not exist. The reconstruction method was designed specifically for unstructured grids, and provides an accurate gradient of cell values as long as at least three non-collinear neighbor cells are available.

The solution is advanced in time with a multi-stage time stepping scheme. Multigrid convergence acceleration was also employed. The tree-based method is ideal for multigrid since coarser grid levels already exist, as well as all of the needed grid-level communication.

Test cases include internal, external, and axisymmetric flows. For the NACA and multi-element airfoils, the outer boundary could be placed at great distances from the airfoil without significantly increasing the total number of cells, thus eliminating the need for sophisticated outer boundary conditions. The wedge case showed that very fine features can be resolved without a large number of cells throughout the domain as in structured grids methods.

5.2 Conclusions

One pressing need for CFD is the ability to generate, automatically and robustly, a grid for complex three-dimensional configurations, and to compute stable and accurate flow solutions about that configuration. Many approaches are being used at the present, but no one method has emerged as the best option for grid generation and flow solution about arbitrarily complex geometries.

A Cartesian grid approach shows great promise as a method to overcome these
problems. But, Cartesian grids do have their share of difficulties to overcome. Any arbitrarily complex geometry can be cut out of a Cartesian grid, but the resulting grid is a very poor representation of the body. A uniform Cartesian grid fine enough to resolve a body adequately would require many more cells than were needed away from the body, thereby requiring unreasonable resources to obtain a solution. The solution to this problem is adaptation. A Cartesian grid which can be automatically and robustly adapted to a body to ensure that each of its features is adequately resolved is needed. Indeed, methods are being developed which are coming close to this ideal for fully arbitrary three-dimensional configurations. Especially promising is the reduction in total number of cells needed to create these grids. Cells can be added only where resolution of the body requires them. This is not only important in terms of physical memory usage to store the information, but also in terms of the real time required to generate the grid and compute flow solutions.

Automatic Cartesian grid generation is not the whole picture, however. The flow solution is also required. Here again, Cartesian grids have some problems. The somewhat arbitrary way that bodies are cut out of a Cartesian grid can create a huge disparity in the sizes of neighboring cells. This issue has been addressed to some degree in two dimensions in this thesis, but questions remain about how well these methods will carry over to three dimensions. Flow solution methods for unstructured grids is currently a widely researched topic, and new innovations may have a large effect on the effectiveness of flow solution on Cartesian grids. Another important issue to be resolved is how solution to the full Navier-Stokes equations, or any other set of conservation equations, can fit into the Cartesian framework. Will new methods need to be developed, or can existing methods be modified to provide flow solutions? One huge benefit to flow solution using a Cartesian grid is that grid adaptation has already been developed in creating the grid. Using additional grid adaptation by solution-based information will provide more accurate solutions for
less cost.

The results of the method presented in this thesis show that this Cartesian grid approach is viable. Second-order global accuracy is obtained for the given computed solutions. These solutions compare very favorably not only with the GAMM and AGARD benchmark solutions, but also with exact incompressible solutions. The number of cells required to obtain these solutions is greatly reduced due to adaptation. In fact, comparable structured grid solutions require from two to ten times the number of cells needed for the Cartesian method. However, compared to a structured grid approach, memory overhead is around 25%, and run-time overhead is around 30%. The run-time scales linearly with the number of cells, due to multigrid. Overall, the flow solutions computed with this Cartesian approach with \( \frac{1}{5} \) the number of cells would take around \( \frac{1}{3} \) of the time to compute, with the same solution accuracy.

Which technologies will still be in use ten years from now cannot be answered. But Cartesian grid methods with adaptation show great promise to be at the forefront. Cartesian grids will one day be used in CFD for automatic, robust, and efficient grid generation and flow solution for complex three-dimensional configurations.

### 5.3 Future Work

There are a number of areas of research where this work could be naturally extended. The first extension is unsteady flow. The most obvious approach is to merge cut cells. Any cell which is cut by the body would be merged with one of its uncut neighbor cells. This results in a slight accuracy loss on the body, but the time-step from the neighbor cell is used rather than the small time-step from the cut cell. This does not address the disparate time-steps from cells on different grid levels, where more sophisticated time-step schemes are needed [10].

Another likely extension of this work is to compute flow solutions of the Navier-Stokes equations. The main difficulty here is the viscous layer should have cells with
a large aspect ratio and are aligned with the body. Allowing these “non-Cartesian”
cuts requires a much more sophisticated data-structure. In this case, a binary tree
may be more useful. Discretization of the viscous terms is also not a trivial task, and
the smoothness of the grid becomes much more important [12]. In fact, the work in
this thesis could be extended to any set of conservation equations, including flows
with chemical reactions or magnetic fields.

The most obvious extension to this work comes in extending the algorithm to
three dimensions. The appropriate data structure in the three-dimensional case is
an octree, with each parent cell being divided into eight children cells. The body
definition is one obvious difficulty to overcome. Defining a body surface from a set
of points and incorporating that information into a code is a difficult task. Memory
usage and computer speed become much more important as well.
BIBLIOGRAPHY
BIBLIOGRAPHY


A QUADTREE-BASED ADAPTIVELY-REFINED CARTESIAN-GRID ALGORITHM FOR SOLUTION OF THE EULER EQUATIONS

by
Darren L. De Zeeuw

Chairperson: Kenneth G. Powell

A method for solution of the steady two-dimensional Euler equations is presented. The scheme is designed to overcome the difficulties associated with geometric complexity and the existence of disparate length scales in the computed flow-fields.

An adaptively-refined Cartesian grid defined by a tree-based data structure is used. Connectivity information is obtained from the data tree via the parent/children relationships of the cells. Initial grid generation is enhanced by geometry-based cell adaptation. The solution is converged to a steady state using a linear reconstruction and an approximate Riemann solver. Multi-stage time stepping and multigrid convergence acceleration are used to advance the solution in time. Solution adaptation is achieved through the use of solution-based gradient information. This enables the grid resolution to match more closely the local length scales of the flow.

The initial grid is generated with a minimum of user input for any complex
configuration. The user need only specify a set of points defining the bodies, the base grid resolution, and cell size thresholds for the geometry-based adaptation. With proper thresholds, the grid is automatically adapted to the curvature of a body, providing the resolution required to resolve that body adequately. The grid is then improved through the use of solution-based adaptation.

The difficulties associated with the small cut cells created by the arbitrary way that the Cartesian grid cuts through the body are overcome by using local time-stepping, coupled with a linear reconstruction method designed specifically for unstructured grids.

The solutions obtained show the second-order global accuracy of the scheme. Results are presented for airfoils at subsonic, transonic, and supersonic speeds. The results compare favorably with benchmark solutions on structured grids with substantially more cells. Also included are a channel flow, several axisymmetric jet flows, and several multi-element airfoil flows. In all cases, the small cut cells generated by the intersection of the body with the Cartesian grid have no adverse effect on the smoothness of the solution. The broad range of results presented demonstrates the geometric flexibility of this approach, as well as the accuracy and efficiency attainable by solution-based adaptation.