COMPUTATION OF INVISCID COMPRESSIBLE FLOWS ABOUT ARBITRARY GEOMETRIES AND MOVING BOUNDARIES

Volume One

by

Sami Alan Bayyuk

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering)
in The University of Michigan
2008

To my father and mother, for all their love and all their sacrifices.

ACKNOWLEDGEMENTS

The research work presented in this dissertation was partially funded by grants from the McDonnel Aircraft Corporation, which were administered and monitored by Dr. August Verhoff. I am grateful for the financial support and interest in the work received from McDonnel Aircraft Corporation, and for the support, encouragement, interest, and help received throughout the work from Dr. August Verhoff.

I sincerely thank all the members of the Doctoral Committee of this dissertation. Each of them contributed to the research work and its documentation by giving valuable guidance and suggestions in the early stages of the research, by reviewing and commenting on the research work and drafts of the dissertation, and by attending and administering the Doctoral Defense Examination of this dissertation. It was a pleasure to work with them, and to benefit from their knowledge and experience. In addition, and more specifically, I thank the following individual members of the Doctoral Committee, for the following specific contributions to the research work or the dissertation:

- Dr. August Verhoff, for his contributions as grant monitor for McDonnel Aircraft Corporation, as cited above, for his personal interest in and support of the research work and the extension and application of the resulting methodology to practical, three-dimensional problems, for his help with research proposals related to the work, and for many valuable suggestions for improvement of the dissertation;

- Prof. Grétar Tryggvason, for many valuable suggestions and comments about the formulation and treatment of moving-boundary problems early on in the research work, and for many helpful comments and suggestions for improvement of the dissertation;

- Prof. Marsha Berger, for many valuable criticisms of the research work and the dissertation, and for demanding several extensions and improvements in the work and the dissertation that made a significant impact on the quality and the value of the research work and the dissertation;

- Prof. Philip Roe, for many valuable criticisms, requests for clarification, and highly precise corrections to an early draft of the dissertation, and for providing encouragement, support, and interest in the work over a period of several years, either informally, or through seminars, related projects, or meetings of the Doctoral Committee; and,

- Profs. Kenneth Powell, Philip Roe, and Bram van Leer for their guidance and mentoring throughout my studies and research work, and for being constant sources of inspiration, knowledge, and deep physical or mathematical understanding over a period of many years, not least through the courses I took with them, through the projects on which I worked with them, and as their Teaching or Research Assistant.

I single out for special thanks, however, the two Co-Chairmen of the Doctoral Committee, Profs. Kenneth Powell, and Bram van Leer. They have been my unwavering allies and supporters, from the beginning, to the end. They have patiently given me all the freedom to explore new research ideas and to learn from my mistakes, while always being ready with ideas, guidance, and help throughout the research work and the many projects on which I worked with them, and despite their busy schedules. I thank them for their many valuable contributions and improvements to the work and the dissertation, and for their extensive reviews and criticisms of the research work, and of drafts of the dissertation. I also thank Prof. Kenneth Powell for kindling my interest in CFD generally, and in methods for Adaptive-Cartesian grids particularly, and his crucial early guidance in my education. It was a pleasure to share a period of my life with them, to experience their influence, and to establish our research collaboration and our friendship.

I also thank the following colleagues or fellow students, for the following (direct and indirect) specific contributions to this work:

- Gregory Ashford, formerly Graduate Student at the Keck CFD Lab, for many valuable discussions on Advancing Front and Delaunay Triangulation techniques, and for providing timing measurements from his unstructured-grid code;

- Eric Charlton, formerly Graduate Student at the Keck CFD Lab, for developing the **xcfd** plotting package which was used to display and visualize data generated with the computer program developed in this work, for many useful discussions about programming techniques, and about his three-dimensional, Octree-based grid-generation and grid-adaptation algorithms, and for helping with the administration of the computer system at the Keck CFD Lab;

- William Coirier, formerly Graduate Student at the Keck CFD Lab and member of the Computational Fluid Dynamics Group, Internal Fluid Mechanics

Division, NASA Lewis Research Center, for many helpful discussions about the accuracy of Cartesian-grid methods, and for very valuable comparisons between our two implementations of Quadtree-related algorithms, and our two approaches to the generation and adaptation of Cartesian grids;

- Darren De Zeeuw, formerly Graduate Student at the Keck CFD Lab, for developing the **dplot** plotting package which was used to display and visualize data generated by the computer program developed in this work and to generate all the contour-line and grid plots shown in this dissertation, for his extensive help with the administration of the computer system at the Keck CFD Lab, and for generously and patiently offering the use of his computing and printing resources, and his disk space, even at the expense of his own inconvenience;

- Usama Fayyad, formerly Graduate Student at The Department of Electrical Engineering and Computer Science at The University of Michigan, for giving me my first few lessons in the "Latex" typesetting language;

- Jeffrey Hittinger, formerly Graduate Student at the Keck CFD Lab, for his help with administration of the computer system at the Keck CFD Lab, especially the printing facilities;

- Jason Hunt, Graduate Student at the Keck CFD Lab, for reviewing the cell-merging chapter in an early draft of this dissertation, and pointing out several typographical and phrasing errors in it;

- Omar Juma, formerly Research Scientist at The Department of Electrical Engineering and Computer Science at The University of Michigan, for many educational discussions on the practices of systematic, professional software development and programming techniques, and for giving me my first few lessons in the use of the “imake” utility and C++ debuggers;

- Robert Lowrie, formerly Graduate Student at the Keck CFD Lab, for many valuable discussions on Boundary Conditions for the Euler and Navier-Stokes Equations, and on the use of the Finite-Element Method in CFD;

- Jens-Dominik Müller, formerly Graduate Student at the Keck CFD Lab, for many valuable discussions on Delaunay Triangulations and the theory of Fluctuation Splitting, and for providing timing measurements from his unstructured-grid code;

- William Rider of Hydrodynamics Methods Group, Applied Theoretical and Computational Physics Division, Los Alamos National Labs, for many valuable discussions on moving-boundary problems and on the VOF method, for suggesting a highly-accurate treatment for boundaries separating two immiscible fluids, for encouragement, enthusiasm, and interest in the work, and for giving the research work reported in this dissertation a new and important context;

- Khaled Shahwan, formerly Graduate Student at The Department of Aerospace Engineering at The University of Michigan, for many valuable discussions and many lessons on the theory and use of the Finite-Element Method in Structural Dynamics, and for many debates about the analogies between solid and fluid mechanics; and,

- Erlendur Steinthorsson, formerly of the Institute for Computational Mechanics in Propulsion, NASA Lewis Research Center, for suggesting the Wankel Engine as a useful test or demonstration case for the methodology developed in this research, and for following up on that test case.

The research work presented in this dissertation was also partially funded by The Department of Aerospace Engineering at The University of Michigan, through several Research-Assistant and Teaching-Assistant appointments. I am grateful for the financial and administrative support received from that department, for the superb research, computing, library, and classroom facilities provided by it, and for its strong promotion of innovation and research. The administrative staff of that department was very helpful throughout my studies and research. In particular, I thank the following individual members of The Department of Aerospace Engineering at The University of Michigan, for the following contributions:

- Margaret Fillion, the Graduate Secretary, for her help with administrative and scheduling matters throughout my studies and work at the department, and for her constant support and personal involvement in the progress and success of the graduate students of the department;

- Caroline Rehberg, Administrative Assistant, for her help with financial and administrative matters throughout my studies; and,

- Bonnie Willey, the Undergraduate Secretary, for her help with administrative and organizational matters.

The research work presented in this dissertation has also benefited from the open and stimulating academic atmosphere, and from the excellent research, learning, computing, library, infra-structural, and support facilities and programs for the faculty and students at The University of Michigan, created largely by the ambition, diligence, and continuous striving for excellence by the administration and staff of The University of Michigan, the College of Engineering, and the Rackham Graduate School. I am particularly grateful for the competent and efficient computer system administration and the help received from the Computer-Aided Engineering Network (CAEN) and its staff: the excellent computing facilities and support of CAEN made possible many of the computationally-intensive calculations performed in the course of the work presented in this dissertation.

The work presented in this dissertation has also benefited from the stimulating academic and research environment created within the CFD Group and the Keck CFD Lab at The Department of Aerospace Engineering at The University of Michigan, by the faculty, students, and researchers of the group, and by others associated with it. More specifically, I have benefited from and enjoyed the many informative or thought-provoking seminars, conference trips, debates, and other mutual learning experiences shared with my colleagues Grenmarie (Gren) Agresar, Mohit Arora, Gregory (Greg) Ashford, Jeffrey (Jeff) Benko, Shawn Brown, Eric Charlton, William (Bill) Coirier, David Darmofal, Darren De Zeeuw, Clinton Groth, Jeffrey (Jeff) Hittinger, Cheolwan Kim, Dawn Kinsey, Dohyung Lee, Timur Linde, Robert (Rob) Lowrie, John Lynn, Ernst Mayer, Karim Mazaheri-Body, Lisa Mesaros, Said Mortazavi, David Mott, Jens-Dominik Müller, Rho Shin Myong, Brian Nguyen, Khaled Shahwan, Nicolas Tonello, Timothy (Tim) Tomaich, and Jeffrey (Jeff) Thomas.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER I

Introduction

This chapter gives an overview of the research work presented in this dissertation and describes the scientific and technological background in which this research work is pursued. The motivation, objectives, and accomplishments of the research are described. The capabilities, the novel contributions, and the distinguishing characteristics of the new computational method developed in this research are described, and they are highlighted by comparing them with those of alternative approaches. This chapter also presents an overview of the contents, scope, and organization of this dissertation.

1.1 General Context, and Overview of the Contribution of the Work Presented in this Dissertation

The research work presented in this dissertation is most appropriately classified as falling within the discipline or field of Computational Fluid Dynamics (CFD). CFD can be described as the science and the techniques of predicting or simulating the behavior or properties of fluid, thermal, or other flows by solving numerically the analytic equations that govern these flows. CFD can be regarded as a sub-discipline of Computational Physics.

As can be deduced from its defining description, CFD is applicable within most branches of traditional engineering; namely, the Aerospace, Mechanical, Marine, Chemical, Electrical, Environmental, and Civil branches, and throughout science. CFD is used to model and study flows and other related physical and transport processes, ranging from those that occur in man-made micro-devices, to those that occur in living organisms, to those that occur in large-scale atmospheric or oceanic, or even galactic or cosmic processes or events.

The purposes of modeling or studying with CFD may range from gaining a better understanding of the fundamental physical phenomena involved (as in fundamental studies of the physics of turbulence, for example), to helping in the creation of better or more efficient designs for manufactured products (as in the optimization of the designs of aircraft or the shapes of mixing vessels, for example).

CFD can be used to study all flow and process "regimes", from the stable, smooth, steady-state flow of heat in a manufactured component with homogeneous isotropic conductivity and fixed boundary temperatures, to the unstable, high-speed, compressible, discontinuous, transient, multi-phase, reacting, electrically-conducting, radiating material flows found in violently exploding stars.

More generally, CFD can be useful in almost any circumstance that involves the phenomena of convection or propagation or diffusion, and in any circumstance in which the physical behavior can be modeled in terms of a transport or flow or propagation or diffusion process. This is because of the power and generality of the methods of CFD for solving general integro-differential governing equations. For example, CFD methods are being increasingly used to solve the Maxwell Equations of Electro-Magnetics, and to study of the electrical behavior of transistors in electric circuits. Such developments extend the applicability of CFD within its parent discipline, Computational Physics.

In the techniques, tools, and thinking processes involved in it, traditional CFD is most usefully viewed as a multi-disciplinary endeavor that draws most heavily on three traditional "pure" disciplines: (i) Fluid Mechanics, as a branch of Engineering and Physics; (ii) Numerical Methods, as a branch of Applied Mathematics; and (iii) Software Engineering and Computer Science. This dissertation reflects this constitution, with some of its parts focusing almost exclusively on Fluid Dynamics, others focusing almost exclusively on numerical-solution methods, and others focusing almost exclusively on algorithms, procedures, and software development.

The research work presented in this dissertation is concerned with devising, developing, and studying a new computational technique for simulating (or numerically predicting) flows involving boundaries that are in relative motion. The new technique is applied to only a specific type of flow: that of an inviscid, compressible fluid. The new technique is developed and demonstrated only for two-dimensional problems, but it is capable of handling boundaries of arbitrarily-complex geometries, and capable of handling arbitrarily-complex motions or deformations of these boundaries. The new technique is also applied and demonstrated within only a specific range of fields: primarily Aerospace and Mechanical Engineering. A typical problem in those fields and flow regimes to which the technique is very suitable is the problem of prediction of the flight path and the dynamic behavior of an axisymmetric, rocket-powered, rigid projectile traveling through the atmosphere.

Even though the work presented in this dissertation is confined largely within the context of the flow regime and fields of application described in the preceding paragraph, the new method is designed and developed to be applicable as widely and generally as possible. This applicability of the new method to other fields has already been demonstrated in two other works that have used variants of it. In the first variant, a methodology was developed to study the motion and adhesion of biological cells [8], where the flow regime is that of incompressible, viscous flow with Low Reynolds Number and micro-scopic length-scales. In the second variant, a methodology was developed to help in the design of waverider airframes [196], where the flow regime involved is that of supersonic three-dimensional steady flow which can be mapped to a mathematically-corresponding two-dimensional unsteady flow.

Even though there are already several computational techniques for computing flows with moving boundaries, as shown in the next section, all of them have shortcomings that inhibit their applicability to the generic moving-boundary problem. The new technique developed here aims to overcome many of these limitations using a new approach that relies heavily on several relatively-recent developments in the field of CFD, especially Cartesian-grid methods and solution-adaptive methods.

Most of this dissertation is concerned with explaining or describing the new method developed here, or with demonstrating or proving its properties.

1.2 Definition, Characterization, and Importance of Moving-Boundary Problems

Throughout Fluid Dynamics, Galilean Transformations are employed to change the inertial frame to one in which relevant boundaries which were originally non-stationary are brought to rest. This is done merely for convenience in the formulation and the solution of problems with originally non-stationary boundaries. However, if there are rotating boundaries, or if there is relative motion between different boundaries or different parts of an individual boundary, the velocity at all points in all the boundaries in the problem cannot be simultaneously eliminated with a Galilean Transformation, and the problem is then classified as a moving-boundary problem.

A **moving-boundary problem** is specifically defined as a problem in which there is relative motion between the bounding surfaces of a flowfield or a phasefield, or a problem in which the shape of any interface (that separates different sub-regions of the entire region of interest) is not fixed in time. An immediate consequence of this definition is that all moving-boundary problems are transient ones.

Moving-Boundary problems occur throughout science and engineering. Instances of these problems arise in all free-surface flows, in the motion of human heart valves in a pulsatile bloodstream, in the deposition of films of solids by the condensation of a vapor in several manufacturing processes, and in the in-flight separation from an aircraft of an external fuel tank after the tank is released. Only in the simplest of problems with moving boundaries is the motion of the boundary decoupled from the solution of the flowfield. Usually, the trajectory of the boundary (or body) and the solution of the flowfield affect each other, as is the case in all four examples just given.

The degree of coupling between the shape and the trajectory of the interface on the one hand, and the flowfield solution on the other hand, can be extremely strong and highly nonlinear in moving-boundary problems. For example, in surface-tension-induced flows, the shape of the boundary, which is typically either a liquid-gas or liquid-liquid interface, is what determines the magnitude and the location of the forces that drive the motion of the flowfield. Another example of such strong coupling is that between the flow and the heat-transfer patterns on the one hand, and the shape and evolution of the solid-liquid interface on the other hand, during a solidification process, such as during the formation of frost from liquid water. Another example of such strong coupling is that between the flow, heat-transfer, and species-concentration fields on the one hand, and the shape and the evolution of the liquid-gas interface on the other hand, during an interface-reaction problem, such as during the combustion of liquid films and spray droplets in a rocket engine. In all three of these examples, the shape of the interface and its evolution strongly influence the behavior of the flow or heat transfer patterns and strongly influence the shape and the evolution of the interface at later times.

As can be seen from some of the examples given in the preceding paragraph, moving-boundary problems are often characterized by interfaces that have geometrically complex and highly-intricate shapes, and that undergo large changes in their characteristic length scales and their topology during their evolution. Another characteristic that can be deduced from the above examples is that the shape and the evolution of the interfaces in a moving-boundary problem can often be determined only as part of the solution of such a problem.

Other, practical, characteristics of moving-boundary problems that can be deduced from the above examples is the physical complexity involved in them, the difficulty or impossibility of solving them analytically, and the value and usefulness of computational techniques for solving them.

From the preceding two paragraphs, it can be deduced that a general computational methodology that is accurate, efficient, and applicable to the solution of the generic moving-boundary problem would have an important impact on the understanding and solution of a wide range of difficult problems that are important from the scientific, technologic, and economic viewpoints.

1.3 Computational Methods for Moving-Boundary Problems

This section presents a brief review and comparison of the alternative techniques that have been devised for computational solution of generic moving-boundary problems, partly in preparation for a description of the defining and distinguishing characteristics of the method developed in this work. Relevant, but not comprehensive, reviews of the most general and most popular computational methods used for moving-boundary problems can be found in, for example, [331], [427], and [269, 323].

Many of the terms used in this section, such as “grid”, “structured grid”, “unstructured grid”, and “Computational Region” are clarified and explicitly defined in Chapter IV, but the exact definitions are not essential for the purposes of this section. The terms “boundary” and “interface” have their obvious meanings, and are also often used synonymously in this section, with the former term being usually preferred in relation to the separation between a solid and a fluid phase, and the latter term being usually preferred in relation to the separation between different fluid phases.

1.3.1 Classification and Characterization of Computational Methods for Moving-Boundary Problems

Perhaps the most useful classification basis for computational methods that have been developed for moving-boundary problems is according to whether the method resolves the interface shape and location using an Eulerian or a Lagrangian formulation.

The Eulerian formulation is characterized by solving throughout the Computational Region one or more transport equations which represent the evolution of properties that are separated by the interface, such as the phase distributions. Moreover, these transport equations must be solved in a stationary inertial frame, and on a stationary grid. The moving interfaces are not explicitly tracked or solved for in an Eulerian formulation. If the shapes or any other geometric properties of interfaces are required, they must be indirectly inferred from the evolution solutions for the properties that vary across the interface. The Lagrangian approach is characterized by tracking the moving interface itself as it evolves, usually using local reference frames or grid nodes that are anchored in the moving interface. These classifications are clarified and illustrated by the examples given below of specific computational techniques for moving-boundary problems.

The classification basis described in the preceding paragraph is also used in the different, but related, context of the way in which the nodes of the grid or the computational cells of the grid are used in solving the field equations. In this different context, the Eulerian and the Lagrangian classifications take on a slightly different meaning, and there is also an additional (third) classification: the Lagrangian-Eulerian, or the "Mixed" classification. The new, different meanings of these classifications are as follows: (i) Eulerian Methods use a grid that remains fixed, no matter how the field solution evolves on that grid; (ii) Lagrangian Methods use a grid in which the grid cells or grid nodes move with the fluid parcels in such a way that no fluid crosses a cell boundary or separates from a grid node; and (iii) Mixed Methods, which are also called Arbitrary Lagrangian-Eulerian Methods, use a grid which moves, but (unlike the case for the pure Lagrangian Methods) not necessarily with the exact local trajectory of the fluid parcels that the cells or grid nodes originally enclose or represent [172, 110, 111].

The next few sub-sub-sections describe and discuss the most important methods that have been developed for computational simulation of moving-boundary problems. Only those methods that are general-purpose, that are acceptably robust and useful, that are physically and computationally well-founded, that have promising prospects, that are applicable in three-dimensional space, and that are applicable to the typical nonlinear problems encountered in CFD are considered. Specifically excluded from this description are methods that are highly specialized or highly restricted in their application. Examples of methods that are excluded from the description given below are the Boundary-Integral Methods [59, 175] and the Phase-Transformation Methods [398].

1.3.1.1 Volume-Tracking Methods

In these methods, the geometry and the dynamics of the interfaces or boundaries separating the different materials or phases in the flowfield are not explicitly updated or tracked. Instead, only the fractional volumes occupied by the different phases are tracked and updated. If the locations of interfaces are required (for example, to calculate the fluxes of the different phases with higher accuracy, or to compute the curvature of the interface to calculate the associated surface-tension forces), then the interface geometries must be "reconstructed" from the volume-fraction data.

In Volume-Tracking Methods, the flowfield is evolved by solving the Equations of Conservation of Mass, Momentum, and Energy, using any appropriate computational scheme. In addition to these equations, the transport and evolution of the phase distributions (and hence implicitly of the interface shape and dynamics) are obtained by solving for the evolution of scalar field variables $f_i = \frac{V_i}{\sum_{i=1}^{N} V_i}$, where $0 \leq f_i \leq 1$ is the volume fraction of phase i, V_i is the local volume (for example, within a computational cell) occupied by phase i, and N is the number of phases. The specific transport equation solved for each f_i, in differential form, is given by

$$\frac{\partial f_i}{\partial t} \quad + \quad \nabla \cdot (f_i \vec{v}) = 0, \tag{1.1}$$

where t is the time, and $\vec{v}$ is the flowfield velocity vector. In practice, it is only necessary to solve for the transport of $N-1$ phases, since the distribution of the remaining phase can be deduced from the distributions of all the others.

It should be emphasized that in Volume-Tracking Methods, only a single set of Mass, Momentum, and Energy Conservation Equations is solved, regardless of the number or the distribution of the different phases present. In addition, in solving these conservation equations, all the fluid, thermal, and transport properties in each computational cell are assumed homogeneously distributed, and these properties are typically computed by volume-averaging using relations of the generic form $\tilde{\phi} = \sum_{i=1}^{N} f_i \phi_i$, where $\tilde{\phi}$ is the required volume-averaged property assigned to a computational cell, ϕ_i is the local property value for phase i, and all other terms are as defined above. An important implication of this homogenization is that no differences are allowed in, for example, the velocities of the different phases that may be present in a cell.

The effects of the distributions of the variables f_i are communicated into the Mass, Momentum, and Energy Conservation Equations through only two means: (i) the evaluation of average fluid, thermal, and transport properties (such as density, conductivity, and viscosity) that are used in the standard conservation equations, as described in the preceding paragraph; and (ii) through the evaluation of fluxes which must take into account the time-averaged and area-averaged values of f_i in any discrete flux contribution. These two means are the manner in which the apparent "passivity" of the transport phenomenon represented in Equation 1.1 is converted into the physically-correct "non-passive" transport.

The coupling between the standard conservation equations and Equation 1.1 can be done either by iteratively solving the latter Equation with the standard conservation equations, or by solving all the equations together in a coupled form. Solving the equations in a coupled form is usually complicated by the fact that Equation 1.1 should be explicitly integrated to minimize the effects of numerical diffusion on the sharpness of the interface representations.

The formulation described above can also be cast as the conservation of mass, momentum, and energy for each of the species or phases involved, and this equivalence is what actually provides the basis for the widely-used formulation described above (in terms of the transport of the field variables f_i).

Different variants of Volume-Tracking Methods differ in the manner in which Equation 1.1 is solved. One variant, as described, for example, in [69], [40], and [351], solves Equation 1.1 using standard MUSCL, TVD, or other such schemes (which are described, for example, in Chapter III). The most common variant, however, is the "VOF" Method, as described in, for example, [173] and [265], and the subsequent improvements of the VOF Method, such as the Piecewise-Linear Interface Construction (PLIC) scheme, as described in, for example, [203], [204], [301], and [302].

The VOF Method and its variants are geometrical approaches that solve Equation 1.1 using a sequence of operations which is effectively equivalent to the following: (i) reconstruction of the individual volume occupied by each of the individual phases in each computational cell, in the form of a polyhedron (that lies within that computational cell); (ii) translation of this polyhedron by a distance $\vec{v}\Delta t$ (where $\vec{v}$ is an appropriate average velocity for the polyhedron over the current time-step, and Δt is that time-step); and (iii) computation of the geometrical overlap of the translated polyhedron with the surrounding computational cells to determine the volumes of the individual phases that are transported to these surrounding cells by the velocity field. The different variants of the VOF Method differ mainly by the accuracy and sophistication of the geometric-reconstruction, volume-advection, and intersection-evaluation steps, as described further in [203], [204], [301], and [302].

The main advantages of Volume-Tracking Methods are as follows: (i) their robustness; (ii) their potential for strict adherence to conservation; and (iii) their ability to readily handle arbitrary topologic transformations in interfaces for any number of phases. The main disadvantages of Volume-Tracking Methods are as follows: (i) their smearing of the interface, to at least the width of a few computational cells in the general case, and even more so with the non-geometrical approach to solving Equation 1.1; (ii) the ad hoc and heuristic basis of their interface-reconstruction procedures; and, for the geometrical variant only, (iii) their cell-by-cell reconstruction of the interface, which leads to discontinuities in the interface across cell edges or faces, which in turn leads to reduced consistence and accuracy for some types of calculations.

While Volume-Tracking Methods are very effective for certain applications (such as modeling complex mold-filling processes, or modeling the impact of liquid droplets with solid or liquid bodies and the subsequent fragmentation of these droplet), these methods can be said to be totally unsatisfactory for modeling the motion of solid objects in fluids. This is because these methods do not preserve the exact geometry of a rigid interface, and because they do not, in their standard form, account for differences in the velocities between the different phases in the same computational cell.

1.3.1.2 Level-Set Methods

In these methods, the geometry and the motion of interfaces is tracked and evolved indirectly, by embedding the interface, which is defined in an n-dimensional Computational Region, as the zero-level-set of a level-set function which is defined in an $(n+1)$-dimensional space [269, 323, 324]. In the generic main variant of these methods, the general functional form of the level-set function is given by $\phi(\vec{x}, t)$, where $\vec{x}$ is the location in the Computational Region, and t is the time, and this function takes the value of the signed normal distance from the interface. As implied in its form, the level-set function is defined at all points in the Computational Region. With these definitions, the motion and evolution of the interface can then be shown to be described using a "speed function", F, defined in conjunction with the level-set function, according to the equation

$$\frac{\partial \phi}{\partial t} \quad + \quad F\,|\nabla \phi| = 0, \tag{1.2}$$

with given initial conditions at, say, $t = 0$, that is, with a given $\phi(\vec{x}, t = 0)$.

The physical dependence of the interface evolution must be wholly expressed in the speed function F, and this can include dependence on the local and the global properties of the interface, as well as on the solution to the flowfield, and even on external parameters or state variables. For example, in surface-tension-driven flows, F depends on the curvature of the interface.

If the function F depends only on position and $\nabla\phi$, then Equation 1.2 reduces to a special case of the Hamilton-Jacobi Equation. If the function F depends only on position, then Equation 1.2 reduces to the Eikonal Equation. If the function F depends only on the flowfield solution in the form $F = \vec{v} \cdot \vec{n}$, where $\vec{v}$ is the flow velocity, and $\vec{n}$ is the interface normal, then Equation 1.2 represents passive transport of the zero level-set and is equivalent to Equation 1.1.

Whatever the form or dependence of F, Equation 1.2 can be solved using any appropriate standard computational scheme for solving Partial-Differential Equations, including higher-order schemes that satisfy, for example, (see Chapter III for the relevant definitions) the TVD property and the necessary entropy-conditions. It should be noted that while the interface still represents a discontinuity in the Computational Region, the level-set function ϕ in which it is embedded is a smooth function that can be evolved accurately using well-understood and standard computational techniques. This is the major accomplishment and benefit of the formulation of the Level-Set Method.

The main advantages of Level-Set Methods are as follows: (i) their robustness; (ii) their ability to handle arbitrary topologic transformations in interfaces for any number of phases; and (iii) their elimination of the entire range of classical, well-known problems associated with advecting discontinuous functions (such as the problems encountered with the advection of the f_i functions in the Volume-Tracking Methods). Another important advantage of the Level-Set formulation that follows directly from the smoothness of the Level-Set function ϕ, is that this function can be readily and accurately differentiated to obtain gradients of ϕ and, for example, curvatures of the interface. For certain applications, this provides a decisive advantage over Volume-Tracking Methods. The main disadvantage of the Level-Set Methods is that they do not conserve mass, momentum, or energy because the integration of ϕ, even by a conservative method, will not necessarily conserve any of the Conserved Variables. Several attempts are currently underway to eliminate this shortcoming, for example, by combining Level-Set Methods with Volume-Tracking Methods to retain the advantages of both [56].

While Level-Set Methods are ideal for a wide range of moving-boundary problems, and for a wide range of problems that are only remotely related to moving-boundary problems [323], they can be considered totally unsatisfactory options for modeling the motion of solid objects in fluids. This is because Level-Set Methods in their standard form do not account for differences in the velocities of the different phases in the same computational cell, because they do not preserve the exact shapes of rigid boundaries, and because they do not even preserve the volumes enclosed by such boundaries.

1.3.1.3 Moving-Grid and Deforming-Grid Methods

In these methods, which are closely related to R-Adaptation Methods (see Chapter IV for more on this), the boundary nodes of the grid covering the Computational Region remain attached to the boundaries of the Computational Region. If the boundaries of the Computational Region move or deform, the boundary nodes of the grid move with these boundaries, and this motion is transmitted to the interior nodes of the grid, causing the interior grid to move and deform as a result. The motion and deformation of the interior grid may be effected by any of a variety of semi-analytical or numerical techniques, such as the application of Transfinite Interpolation to update the positions of the interior nodes from those of the boundary nodes. An important requirement for such techniques is that they must attempt to maintain the "quality" (as defined, for example, in Chapter IV) of the deforming grid.

Moving-Grid and Deforming-Grid Methods may utilize structured grids, as in [365, 250, 331] for example, or unstructured grids, as in [31, 33], [227, 225, 35], [218], [263], or [283], for example.

Moving-Grid and Deforming-Grid Methods are highly effective when the deformations or displacements of boundaries are small relative to the length-scales separating the boundaries or those along the boundaries, as in typical aeroelastic problems, for example. Because of this, the use of these methods in aerodynamic and aeroelastic computations is well established. For problems where the deformation length-scales are relatively large (and a typical example of this is when the boundaries or interfaces are those that separate different fluids in arbitrary motion), the distortion of the grid becomes too severe to maintain accuracy or convergence of the solution, and the computation eventually fails.

The restriction on the extent of the allowed motion or deformation in boundaries tends to be more severe with structured grids than with unstructured grids. For both grid types, however, more flexibility can be gained by allowing the boundary nodes of the grid to slide and to have their positions re-adjusted along the boundaries of the Computational Region to relieve excessive local distortions.

An effective way to avoid catastrophic failure from excessive grid deformation is to stop the computation when the grid quality becomes too poor, and to then regenerate the grid for the updated geometry. The solution is then projected or interpolated from the original grid to the regenerated grid, and the calculation procedure is continued. This grid regeneration procedure (which is also called re-meshing) can be automated, at least for relatively simple geometries. A major detraction of re-meshing is the loss of conservation and solution convergence as a result of the grid-to-grid solution-interpolation step.

Remeshing may be applied on either a local basis to eliminate localized degradations in the quality of the grid, or on a global basis to generate a completely new grid. Remeshing is easier to implement with unstructured grids than with structured grids, largely because unstructured grids are by their construction more adaptable, and because they can be generated in a more automated manner than structured grids. These ideas are discussed in more detail in Chapter IV, and the most common local and global re-meshing techniques are described and discussed in the references cited above for Moving-Grid Methods that utilize unstructured grids. Re-meshing may also be used to extend the applicability of Moving-Grid and Deforming-Grid Methods to situations in which the boundaries undergo topologic transformations, such as merging or tearing, but only at a high cost in terms of computational efficiency, loss of conservation, and loss of solution convergence.

Unlike all the other methods described in this sub-section, Moving- and Deforming-Grid Methods have an intrinsic advantage over most other methods for modeling boundaries separating rigid or solid phases from fluid phases. The reason for this is that the grid-lines in these methods by design coincide with the boundaries or interfaces in the problem, exactly resolving any discontinuities (in any flow or material property) at these boundaries.

A sub-category of Moving-Grid Methods includes the Non-Deforming-Grid Methods, which include the "Overset" or "Chimera" family of methods, and the "Sliding-Grid" Methods. All these methods are very suitable for modeling the motion of rigid objects in fluids.

In the Chimera family of methods, a different, "single-block" grid is generated for each boundary or body in the problem or for each selected part of the Computational Region. Each individual grid is generated independently of the other grids, but each grid is designed and constructed so that it overlaps other grids across a sufficiently-deep layer of computational cells along all its "flow" boundaries [342, 43, 240]. The assembly of the individual overlapping grids in the system must cover the entire Computational Region. The individual grids can remain stationary, for steady-state calculations and for time-dependent calculations with stationary boundaries. For moving-boundary problems, the individual grids move rigidly with the boundaries to which they are attached. The solutions on the different grids in the Chimera Methodology are coupled by interpolating the solution data in the boundary cells of each grid from solution data of the interior cells of the other grids that it overlaps. For moving-boundary problems, typically, there is also one large background grid that covers the entire Computational Region and that overlaps all the other grids to ensure that the solutions in the different grids can remain coupled for arbitrary motions of the individual grids.

Since no grid deformation is involved in them, the Chimera Methods eliminate the need for re-meshing. The main disadvantage of the Overset or Chimera Methods is the loss of conservation and convergence rate due to the interpolation used to transfer solution data from one grid to another across the overlap or interpolation fringes. A major limitation of the Chimera Methods (which is shared with the standard Moving-Grid Methods) is their inability to handle topological changes in boundaries in a satisfactory manner.

1.3.1.4 Front-Tracking Methods

These methods include the generic Marker-and-Cell Method [152, 153, 151, 133], and the Immersed-Boundary Method [279], as well as several other variants, such as that of [370]. These methods solve for the field variables using a fixed grid, but track the interface by tracking the motion of markers that are placed in specific sequences along each interface at the start of the computation. Markers can be added and removed within each sequence (or string), and the markers of different interfaces can be re-connected to represent topologic transformations. In these methods, the interfaces are given a thickness in the form of a distribution function which is used to interpolate properties to and from the stationary grid on which the field solution is computed.

The main advantage of Front-Tracking Methods is their robustness, provided any topological changes in interfaces are of limited complexity. One of the disadvantages of these methods is that the interface is smeared, especially if it moves or if it is required to be stationary in a non-stationary fluid, but the thickness of an interface remains constant, since it is an added, algorithmic artifact, and not caused by numerical diffusion. Another disadvantage is the loss of conservation, especially during topologic transformations. Another disadvantage is that the marker particles cannot be easily immobilized to represent a stationary front in a non-stationary fluid, making these methods unsuitable for modeling the motion of solid objects in fluids.

1.3.1.5 Particle-Based Methods

In these methods (the most notable variant of which is the Method of Smooth Particle Hydrodynamics (SPH)), the fluid is modeled through a collection of representative particles that travel with the fluid, interacting with each other, and carrying with them the representative properties of the fluid, such as its mass, momentum, and energy. Since the particles travel with the fluid parcels they represent, the method is a pure Lagrangian one. Extensive reviews of the main variants of the SPH Method are given, for example, in [251] and [45].

Such methods are ideally suited for modeling the merging or interaction of different fluid masses, especially if there are large relative differences in the length scales involved, as in stellar collisions, for example. The main advantages of the SPH Method are as follows: (i) the sharpness of interfaces does not decrease with the propagation distance or the propagation time, and this is because the particles travel with the fluid; (ii) arbitrary length-scale differences between bodies or across interfaces can be treated, with no adverse effects on the computational accuracy or efficiency; and (iii) no grid is required, except in some variants of the SPH, of which the Particle-in-Cell (PIC) Method [150] is the most notable one. In the PIC Method, the particle properties are averaged in computational cells to obtain field values. The main disadvantage of Particle-Based Methods is their requirement for relatively excessive memory and computational effort. Other than for the constraints of this disadvantage, these methods can, in principle, be effectively used for modeling the motion of solid objects in fluids.

1.3.2 Comparison of the Eulerian and the Lagrangian Methods for Moving-Boundary Problems

In respect of the manner in which the interface is evolved and resolved by the methods described above, the Volume-Tracking Methods and the Level-Set Methods are Eulerian Methods, while the Front-Tracking Methods, the Particle-Based Methods, and the Moving- and Deforming-Grid Methods are Lagrangian Methods.

The main advantage of the Lagrangian approach is that it resolves a boundary or an interface as an exact discontinuity. If a particular Lagrangian method requires a grid or nodal markers in the interface, then the main weakness of that method will be that the grid or the markers will be required to deform or move with the interface. This in turn restricts the complexity of the geometric or topologic interfacial transformations that be handled by the method, and makes the method less robust than an Eulerian approach, especially in three-dimensional space.

The main advantage of the Eulerian approach is its robustness, and its ability to handle arbitrary geometric transformations in boundaries, such as those that occur in large-scale elongations and shearing deformations, and arbitrary topologic transformations in boundaries, such as those that occur during merging, coalescence, tearing, or void creation. The main weakness of the Eulerian approach is its reduced resolution and reduced accuracy in the prediction of the interface evolution and shape, caused by its imposition of diffusive effects on the shape and the motion of interfaces. A second important weakness of the Eulerian approach is its need for additional procedures to reconstruct the location, shape, or any other geometric property of the interface.

The ideal application for an Eulerian approach is one that involves high complexity in terms of the topologic and geometric transformations in the interface, and that also exhibits a weak dependence of the final solution on the exact interface geometry and evolution. The ideal application for a Lagrangian approach is one that involves low complexity in terms of the topologic and geometric transformations in the interface, and that also exhibits a strong dependence of the final solution on the exact interface geometry and evolution, and possibly on the treatment of the interface as an exact discontinuity. Examples of applications that are ideal for each of the Eulerian and the Lagrangian approaches were given in the preceding sub-section, along with the descriptions of the individual methods for computational solution of moving-boundary problems.

1.4 A Brief Description of the New Method

The preceding section discussed the capabilities and the strengths and weaknesses of each of the major, general-purpose methods that have been developed for computational simulation of moving-boundary problems. In particular, the capabilities of each of these methods for physically-correct and accurate modeling of the motion of solid objects in fluids was also specifically assessed. These assessments can be summarized as follows: none of the available Eulerian Methods is suitable for accurate modeling of the motion of solid objects in fluids, and only those Lagrangian Methods which treat the boundary as an impermeable discontinuity at which boundary conditions can be applied are suitable for this purpose.

The aim of the research work presented in this dissertation is to develop a general-purpose method which combines the strengths of both the Lagrangian and the Eulerian approaches, without introducing all the disadvantages of both, and which is capable of physically-correct and accurate modeling of the motion of solid objects in fluids.

The objective described in the preceding paragraph is realized by adopting a Lagrangian scheme to represent the geometry and the motion of boundaries as exact discontinuities, and an Eulerian scheme to evolve the flow variables and any other field variables. Instead of using a moving grid that remains attached to boundaries, as with traditional Lagrangian approaches, such as the Moving-Grid approach, a single stationary grid is used as with traditional Eulerian approaches, but boundaries are still tracked separately on this stationary grid using a Lagrangian approach. Those computational cells which are intersected by boundaries are split into two (and in principle, possibly more than two) sub-cells, each of which may have properties and states that are independent of those of the other sub-cell(s). This splitting of cells is what enables boundaries to be treated as exact discontinuities on the Eulerian grid. The adoption of a Lagrangian formulation for boundary motion in combination with an Eulerian stationary grid requires boundaries to freely move across stationary grid-lines, without introducing new or unacceptable types of numerical errors (such as conservation violations). A major accomplishment of this work is to demonstrate that this can be done, by use of a cell-merging procedure that combines cells in the vicinity of boundaries into composite cells that undergo a change in volume but remain intersected along the same edges during a motion step of the boundaries. This cell-merging procedure is the means by which the difficulties associated with grid-line crossing are eliminated in this work.

In more detail, the new method developed in the work presented in this dissertation is constructed from five main components, each of which contributes to one of the major distinguishing characteristics of the method. These components are as follows:

1. An adaptive, Quadtree-based algorithm for generating stationary Cartesian grids which are used to obtain discrete solutions to the governing equations. This grid generation algorithm forms computational cells by first dividing the Cartesian Square which defines the Computational Region into square cells, and then clipping or splitting each cell that straddles a boundary to form two arbitrary polygonal cells on the two sides of that boundary.

   The grid-generation algorithm is capable of handling boundaries of arbitrarily-complex geometry, and is capable of handling arbitrarily-complex motions of these boundaries. The grid-generation algorithm imposes no intrinsic restrictions on changes in the grid topology, neither between nor along boundaries, in common with the traditional Eulerian approaches. The independent tracking of the motions of boundaries is also used by the grid-generation algorithm to continually adapt the grid to the locations and the geometric features of all the boundaries in a computation.

2. A representation of boundaries as exact, infinitesimally-thin discontinuities separating two arbitrarily-different states or properties, and the adoption of explicit tracking of moving boundaries. The explicit tracking of boundaries, and their representation as exact discontinuities is what provides the Lagrangian character of the new method as far as the modeling of the geometry and the

motion of boundaries is concerned.

The exactness of the boundary representation, and its preservation with no diffusive or dispersive effects while boundaries travel across the grid-lines and the cells of the Cartesian grid that remains stationary is the main distinguishing characteristic of the new technique developed in this work.

The use of flexible data-structures for representing boundaries allows large-scale geometric and topologic transformations to be handled, and allows precise control over the behavior of boundaries during topologic transformations, offering ample opportunity to model the actual physical mechanisms involved.

3. A procedure for dynamically combining selected agglomerations of computational cells from the Cartesian grid into individual composite computational cells which do not change the topology of their intersection with the boundaries during a single motion step of the boundaries.

   The main purpose of this dynamic "re-assembly" or merging of the cells in the vicinity of boundaries is to circumvent all the theoretical and practical complications associated with the crossing of grid lines by moving boundaries. By creating composite cells which do not change their intersection topology, the cell-merging procedure effectively transforms the grid-line crossing problem into a regular problem of deforming cells.

   Cell merging thus provides the means of extending the applicability of stationary Cartesian grids to moving-boundary problems. Cell merging is also used to ensure that the initial and final volumes of intersected composite cells are appropriately bounded, and that the ratios of initial to final volumes in these cells are also appropriately bounded. Bounding these quantities ensures that the timestep size remains sufficiently large, and eliminates the "Small-Cell-Problem" (which is discussed in more detail in Appendix A.1) that arises from the non-boundary-conformality of Cartesian grids.

4. A Finite-Volume, characteristic-based, high-resolution flow-solver which is ideally suited for problems involving high-speed, inviscid, compressible flows. The solution scheme achieves second-order accuracy in space by piecewise-linear reconstruction of the primitive variables, and second-order accuracy in time by a predictor-corrector explicit time-stepping scheme. The discretization of boundaries and their motion on one hand, and of the flux quadratures on the other, are appropriately matched to ensure that the second-order-accurate discretization also satisfies The Geometric Conservation Laws.

   Although the Cartesian grid on which the governing equations are solved is stationary, because boundary motion in this work is effectively treated as localized cell deformation (as described above), the governing equations must be solved using an Arbitrary Lagrangian-Eulerian formulation instead of an Eulerian formulation.

5. An algorithm for adapting the Cartesian grid to the solution. This algorithm enables the resolution of flow features with large gradients (such as discontinuities) in flows with different length-scales, and it also enables automatic tracking of these flow features as they move across the Computational Region.

   The solution adaptation algorithm greatly increases the computational efficiency of the new method, enabling problems with high boundary resolutions to be attempted with relatively modest computing resources compared to the resources required by the corresponding non-adaptive computations.

Each of the five components listed above is described in detail in one of the chapters of this dissertation.

According to the classification definitions given in the preceding section, the scheme developed in this work is classified as follows: in regard to the solution of the flowfield and the governing equations, it is an Arbitrary-Lagrangian-Eulerian method; in regard to the modeling and representation of the boundary geometry and dynamics, it is a Lagrangian method.

In some of its features, the new method developed in this work bears strong resemblances to what an adaptive version of the Marker-and-Cell Method described in the preceding section would be. A key difference, however, is that the lines connecting the objects that correspond in the new method to the "markers" of the Marker-and-Cell Method are effectively represented and treated as grid lines that split any computational cells they divide into two parts, allowing these different parts to have different states and properties, and allowing the appropriate boundary conditions to be imposed on these connecting lines. Because of this, the new method combines the Lagrangian character of "exact separation" exhibited by the Moving-Grid and Deforming-Grid Methods with the Eulerian character of the flexibility and robustness of using a stationary grid in the Marker-and-Cell Method. Also because of this cell splitting and exact separation, the new method does not require the imposition of a thickness on the interface for property interpolation as in most variants of the Marker-and-Cell Method. Also because of this cell splitting and exact separation, and because the motion of those geometrical objects in the new method which correspond to the "markers" in the Marker-and-Cell Method is not derived from the velocity of the flowfield, the new method overcomes the fundamental inability of the Marker-and-Cell Method to accurately model the motion of solid objects in fluids (that is, to accurately model the impermeability condition).

From the inherent characteristics and properties outlined or implied above of the techniques adopted or devised for the new computational method developed in this work, the following can be predicted: unlike it is with any of the methods presented in the preceding section, the new method developed in this work will be ideal for simulating the large-scale motion or deformation of solids in fluids. It can also be predicted, however, that the new method will share one of the weaknesses of the Marker-and-Cell Method for moving-boundary problems with topologic transformations; namely, the violation of conservation during topologic transformations. Another weakness of the new method developed in this work is a consequence of the selection of a Cartesian-grid methodology for the grid generation: because of the spatial isotropy of the cells produced by this grid-generation method, the new method without any further refinements will have a relatively low computational efficiency for the solution of any system of governing equations which has highly anisotropic length scales. An important example of such a system is the Navier-Stokes System of Equations for flows with high Reynolds Number.

Other than for the weaknesses outlined in the preceding paragraph, the method developed in this work is expected to be applicable effectively to the entire range of problems requiring accurate resolution of moving and deforming boundaries, from problems involving wing flutter, to problems involving projectile flight, to problems involving surface-tension-induced flows.

1.5 Objectives, Scope, and Limitations of the Work

The preceding section described the new computational method that was developed through the research work presented in this dissertation, and it emphasized the capabilities, the distinguishing characteristics, and the algorithmic principles of the new method. The purpose of this section is to state the specific requirements and objectives of the work presented in this dissertation, to describe the specific chief contributions of this work, and to describe the scope and the limitations of this work, clarifying the extents to which the development and the exploration of the new method have been pursued in the work.

The specific objectives of the research work presented in this dissertation are as follows:

1. The development of a new, general-purpose method for computational solution of moving-boundary problems. The new method is to be capable of handling boundaries having complex geometric shapes, and is to be capable of handling complex motions of boundaries, as well as topologic transformations in boundaries. The new method is to overcome the major weaknesses of existing methods for the computational solution of moving-boundary problems, especially for accurate and physically-correct modeling of the motion of solid objects in fluids.

   The new method is to be capable of solving steady-flow problems, and unsteady-flow problems with stationary boundaries. Both of these types of problem should be solved as special cases of the general, moving-boundary problem.

   Development of the new method includes development of each of its five main components, which were individually described in the preceding section;

2. The investigation of the capabilities, limitations, and relevant computational properties of the new method in general, and for the chosen fluid-dynamic model, The System of Euler Equations, in particular; and,

3. The extension of the capabilities and the applicability of unstructured Cartesian-Grid Methods to the class of moving-boundary problems.

The intrinsic limitations and weaknesses of the new method developed in the work presented in this dissertation were outlined in the preceding section. The extent to which the research, development, and investigation is carried forward in this work, and the most important limitations of or restrictions on the scope of the work carried out are as follows:

1. The development, demonstration, and investigation of the new method and its related techniques are confined to problems in two-dimensional space;

2. The type of grid used in the new method is confined to the unstructured Cartesian type;

3. The governing equations studied and evaluated with the new method are confined to The System of Euler Equations (which models the flow of inviscid, compressible, ideal gases).

Even though the work presented in this dissertation is confined to two-dimensional space, every effort is made to ensure that the concepts and techniques adopted or developed in this work are applicable in three-dimensional space. Similarly, even though the governing equations are confined to The System of Euler Equations, every effort is made to ensure that the solution methodology can be applied to other categories of governing equations which are frequently encountered in Computational Physics. No restrictions are placed on the types or geometries of boundaries, nor on the types of motion that may be executed by boundaries.

The objectives of the research carried out in this work, within the scope and limitations outlined above, are all achieved, as described further in the appropriate chapters of this dissertation, and as summarized in Chapter IX.

The most important new contributions which result from achieving the objectives of the work presented in this dissertation, in order of decreasing likely long-term impact, and with regard to specific aspects of computational methods, are as follows:

1. With regard to moving-boundary problems: the development of a new, viable, and efficient Lagrangian-Eulerian Method that exhibits the most desirable advantages or features of both the Lagrangian and the Eulerian approaches for moving-boundary problems. These advantages or features are as follows: (i) retaining the interface representation as an exact discontinuity, with a precisely-defined location, separating arbitrarily-different states (the advantages or features of the Lagrangian approach); and (ii) solving the field variables using a non-boundary-conformal, stationary grid, and allowing complex geometries, and large-scale motions, deformations, and topologic transformations in boundaries (the advantages or features of the Eulerian approach).

2. With regard to Cartesian-Grid Methods: the development of the technique of "cell merging" which extends and establishes the applicability of Cartesian-Grid Methods to moving-boundary problems, which re-affirms the applicability of these methods for unsteady-flow problems, and which provides a means of overcoming the difficulties that arise from the non-boundary-conformality of these methods.

3. With regard to solution-adaptive methods for The System of Euler Equations: the demonstration of the viability of a specific scheme for refinement and coars-

ening of cells for solution-adaptive computations for The System of Euler Equations with moving boundaries.

A more detailed review of the most significant contributions and findings of the work presented in this dissertation is given in Chapter IX. Chapter IX also discusses the extension of the new method developed in this work beyond the scope and limitations outlined above.

1.6 Organization and Scope of the Dissertation

1.6.1 Readership and Writing Conventions

The contents and writing of this dissertation are aimed primarily for researchers, for science or engineering practitioners, and for students. The main interest of the reader is assumed to be in understanding the basis or implementation details of part or all of the new computational method presented in this dissertation, or in extending, applying, or evaluating the application of this new method to problems that are similar or related to the ones considered in this work. The contents and writing are aimed most specifically for graduate student researchers starting a new research program. Thus, every part of this dissertation was written in such a way that it can be understood by a student who has completed the first round (or first year) of graduate-level classes that are usually taken before starting the research phase of a Masters or Doctoral Program in Aerospace or Mechanical Engineering, in Applied Mathematics, or in other related fields.

In terms of specific subjects, everything in this dissertation should be understood by anyone who has taken the introductory graduate courses in the following topics: the Theory of Fluid Dynamics, especially for compressible flows; Analysis and the Theory of Multi-Variate Calculus; Discrete Mathematics; the Theory of Vector

Spaces and Linear Operators; the Theory of Partial Differential Equations, including the Theory of Characteristics for Hyperbolic Partial Differential Equations; and Numerical Analysis and Numerical Methods.

Inevitably, a conflict arises in attempting to satisfy the entire range of potential readers falling between the following two extremes: (i) those readers who prefer a direct, very brief, and condensed review of the main methods and contributions of the work, and possibly brief guidelines for re-implementation of certain features of the new method; and (ii) those readers who prefer a detailed description of the work done and the methods used, as well as a comprehensive review of the relevant background of the numerical methods and physical-modeling principles involved. The attempt to resolve this conflict here was made by including much review material, while clearly indicating the sections devoted to such a purpose. In some parts of the dissertation, most notably in Chapter III, these divisions between the review material and the directly-relevant material occur down to the sub-section level and lower. Those chapters or sections that contain a significant review portion may be skipped altogether by readers who are interested only in the new developments explored in this work. To enable this skipping to be readily done, the titles of those chapters or sections have been chosen to clearly reflect their review character.

As described in the preceding section, the work presented in this dissertation is confined to two-dimensional space. However, in view of the importance of extending this work to three-dimensional space, significant effort was devoted to extending the ideas and the formulation to three-dimensional space. For this reason, most of the development for The System of Euler Equations, for the formulation of the Geometric Conservation Laws, and for the formulation of the boundary geometry and the boundary dynamics was carried out for the more general cases of either n-

dimensional or three-dimensional physical space. Whenever concrete examples are given, however, they are strictly in two-dimensional space.

The conjunctions and disjunctions "and" and "or" are used most commonly in the strict inclusive sense, whether they separate pairs of items, or whether they appear in lists. Thus, "or" is used in the non-exclusive sense, that is, "a or b" is equivalent to "only a, only b, or both", while "a and b" means both are required. Any deviations from this convention are made identifiable from the context in which they occur.

1.6.2 Organization and Contents

The contents of this dissertation are organized as follows:

1. **Chapter I:** This chapter introduces and motivates the research work which is presented in this dissertation and describes the resulting new computational method for moving-boundary problems. This chapter also compares the capabilities and features of the new method with those of other, existing alternatives. The scope of the work and its contribution to the field are described. This chapter also outlines the contents and the organization of the dissertation.

2. **Chapter II:** This chapter reviews the derivation, and the physical and mathematical characteristics and properties represented by The System of Euler Equations. It also discusses the practical uses and practical limitations of this system of equations.

3. **Chapter III:** This chapter reviews the alternative methodologies that are available for the discretization, and for the discrete solution of The System of Euler Equations, and presents the specific methods adopted and developed in this work to discretize the governing equations and to obtain discrete solutions to these discretized equations, especially as required for moving-boundary

problems. This chapter discusses several considerations that are central to the contribution of this work, especially the enforcement of The Geometric Conservation Laws and the treatment of boundary conditions for moving-boundary problems.

4. **Chapter IV:** This chapter reviews the alternative methods available for generating grids (to discretize the regions of the space-time continuum in which the governing equations will be solved, and to form the connectivities between these discrete elements). Special emphasis is placed on those methods that are most relevant to moving-boundary problems. This chapter also reviews the alternative methods that are available for adapting grids (to the computational solution, or to the geometry of the boundaries involved).

5. **Chapter V:** This chapter presents the specific methods adopted and developed in this work for defining and representing the geometry of boundaries. The chapter also presents the methods adopted in this work for modeling the motion of (rigid and deformable) boundaries, covering in detail the treatment used for pre-specified boundary trajectories, and for boundary trajectories that are determined by coupling of the boundary dynamics with the flowfield solution.

6. **Chapter VI:** This chapter reviews the Quadtree data-structure and its properties. This chapter also presents the specific methods, which are based on the Quadtree data-structure and the unstructured Cartesian-grid approach, that were adopted for generation and adaptation of grids, emphasizing the features and properties of these methods that are relevant for computational solution of The System of Euler Equations with moving boundaries. The techniques used

in this work to adapt the Quadtree-based grids (wholly by cell-refinement and cell-coarsening operations) to the geometry and the motion of boundaries, and to the flowfield solution are described in detail, with emphasis on the special requirements for adaptation with moving flow-features and moving-boundaries.

7. **Chapter VII:** This chapter presents the new technique of "cell merging", emphasizing the crucial role that it plays in enabling the use of a Lagrangian representation of the geometry and motion of boundaries, and an Eulerian (stationary, Cartesian) grid for solution of the field variables. This chapter also presents the algorithms used to perform the merging operation, together with examples that clarify how these algorithms work in practice. This chapter also presents a theorem which establishes a relation between the local geometric properties of a boundary and the minimum local spatial resolution of the Cartesian grid required to ensure success of the merging algorithm. This chapter also discusses the shortcomings and the computational costs of cell merging.

8. **Chapter VIII:** This chapter presents the verification, validation, and demonstration results that establish the viability, accuracy, and computational efficiency of the new method developed in this work. This chapter presents comparisons of computational results obtained with the new method with the corresponding analytic solutions, and, for a few time-dependent and moving-boundary test-cases, with the corresponding experimental results. This chapter also compares the performance of the new method with other alternative methods for relevant standard test cases. This chapter also presents several computations that demonstrate the most important capabilities and limita-

tions of the new method. For convenience, the computations presented in this chapter are partitioned into categories.

9. **Chapter IX:** This chapter summarizes the work done to develop the new computational method, summarizes the key findings and the main conclusions and contributions of the work, and presents the most promising directions for extension of the work to more general problems and settings, and in directions that appear to be the most promising and useful. This chapter also provides suggestions for improvement of the method as implemented in this work.

The first and last chapters of this dissertation are written in the simplest form, with the aim of providing a comprehensive overview of the work done and its contributions and significance.

The appendices are clearly marked, and include topics and concepts that are repeatedly referenced or invoked in different parts of the dissertation.

The division of the material among Chapters II, III, IV, V, and VI in this dissertation reflects the idealized systematic procedure used to obtain a numerical solution in Computational Physics, which can be summarized as follows:

1. **Formulation, Discretization, and Establishment of a Discrete Solution Scheme for the Dependent Variables:** The governing equations of the problem are formulated, and they and the physical properties that they are to represent are understood and characterized. Then, the appropriate numerical methods which will be used to solve them discretely are selected, in a way that ensures that those numerical methods will faithfully reproduce the key features of the physical behavior being modeled.

   Chapters II and III are devoted to these tasks, with Chapter II devoted to reviewing the derivation, and to reviewing the physical and mathematical features of The System of Euler Equations, and with Chapter III devoted to the selection and the implementation of the computational-solution methods adopted in this work, and to reviewing the feasible alternative methods.

2. **Specification and Discretization of the Independent Variables:** The independent variables (that is, the variables $\vec{x}$ and t that describe the region of the space-time continuum in which the discrete solution is sought) are discretized, generating a set of discrete spatial elements or sub-regions, with each of which a discrete solution vector is eventually associated. The connectivity between those spatial elements is also established simultaneously with this discretization process. This process also clearly involves the geometry of the boundaries of the region in which the discrete solution is sought, and must therefore also generate a discrete representation of those boundaries.

   Chapters IV, V, and VI are devoted to these tasks, with Chapter IV reviewing the relevant options that are available for grid generation and adaptation, with Chapter V presenting the techniques adopted for descritizing and modeling of the boundary geometry and motion, and with Chapter VI describing the specific methods chosen in this work to generate and adapt grids.

   The chosen partitioning of the material among Chapters V and VI reflects the independence of the representation and modeling of the geometry and motion of boundaries from the representation of the grids and the operations in grid generation and adaptation. The partitioning also reflects the very close association for Quadtree-based grids between the grid generation and the grid adaptation processes.

The independence of the representation of the boundary geometry and the representation of the grid is extensively explained in Chapters V and VI, where it is shown that the two representations are maintained and modified completely independently of each other, through separate data-structures and operators, and that the only mechanism through which the boundary geometry and the boundary motion influences the grid is through the geometric adaptation of the grid.

CHAPTER II

Review of the Formulation and Characterization of the Governing Equations

The fluid model adopted in this work is that of inviscid, compressible, adiabatic flow, as described by The System of Euler Equations. This chapter gives a concise description of this system of equations, partly to serve as a reference for other chapters. The System of Euler Equations is derived in various forms from axiomatic physical principles, and the relations and distinctions between these forms are described. The existence and uniqueness of analytic solutions of The System of Euler Equations, and the techniques for obtaining such solutions in special cases are briefly discussed. The fundamental and characterizing mathematical and physical properties of The System of Euler Equations over its entire range of validity are presented, and the extent to and manner in which these properties allow this system to model practically useful flows are described.

A general development of the governing equations is followed throughout, leading to formulations and results that are appropriate for the moving- and deforming-grid discretizations presented in the next chapter. For generality, the physical space (in which the governing equations apply) is assumed to be three-dimensional, but specializations to two- and one-dimensional spaces are also indicated where useful.

2.1 Derivation of The System of Euler Equations

2.1.1 Ontological Origin of The System of Euler Equations

The Mathematical Theory of Non-Relativistic Continuum Hydrodynamics is formulated by taking the principles of conservation of mass, momentum, and energy as axioms. The main aspects of this theory are presented, for example, in [30], [92], or [335] (from a more mathematical point of view), and in [211], [325, 326], or [431] (from a more physical point of view). The System of Euler Equations is a special case of this theory, derived by excluding the effects of shear and dilatation stresses and of heat conduction from the interactions allowed in a fluid. The resulting system is the most general continuum description of steady and unsteady, compressible, ideal (inviscid, non-conducting, non-reacting) flow. The physical consequences and limitations of this idealization are discussed in Sections 2.2 and 2.4.

The Theory of Continuum Hydrodynamics is itself a limiting special case of the more general Kinetic Theory of Hydrodynamics [395]. Indeed, The Euler Equations in a differential form can be be derived from the Boltzmann Equation,

$$\frac{\partial F(t,\vec{r},\vec{v})}{\partial t} + v_i \frac{\partial F(t,\vec{r},\vec{v})}{\partial r_i} + a_i \frac{\partial F(t,\vec{r},\vec{v})}{\partial v_i} = C(t,\vec{r},\vec{v}),$$

where $F(t,\vec{r},\vec{v})$ is the Phase-Space Distribution Function, where r_i, v_i, and a_i respectively refer to the molecular location, velocity, and acceleration in the i^{th} coordinate direction, where t refers to time, and where $C(t,\vec{r},\vec{v})$ is the rate of change of the Distribution Function, $F(t,\vec{r},\vec{v})$, due to molecular collisions. The derivation is accomplished by taking the equilibrium (or continuum) limit of the collision term, to wit, $C(t,\vec{v},\vec{r}) = 0$, and then taking moments of the reduced equation according to the moment vector $\vec{m} = \left(1, v_1, v_2, v_3, e + \frac{1}{2}q^2\right)$, where v_1, v_2, and v_3 are as defined above, e is a function of the molecular internal energy, and $q^2 = v_1^2 + v_2^2 + v_3^2$. With the given ordering of the elements of vector $\vec{m}$, these moments respectively produce the conservation equations for mass, for momentum in the co-ordinate directions 1, 2, and 3, and for energy.

The Euler Equations may also be derived from the Navier-Stokes Equations by imposing the inviscid and non-heat-conducting assumptions.

2.1.2 Derivation of the Integral Form

Following an approach based on the Continuum Hypothesis, the equations that prescribe fluid behavior may be derived in an integral form by analyzing the interaction of a fluid contained in a control volume, Ω, of arbitrary geometry, with its surroundings, subject to the constraints imposed by the conservation axioms. For an ideal fluid, these interactions may only occur through the control surface, $\partial\Omega$, and only by means of pressure on that surface or by means of the convective transport of quantities by flow through that surface.

In the classical Eulerian Formulation, the control surface is assumed stationary. However, the most general formulation is obtained by placing no restrictions on the motion of the control surface, leading to the so-called **Arbitrary Lagrangian-Eulerian (ALE) Formulation**. The only additional analytic complication that this generalization introduces is that the flux of a quantity at any point in $\partial\Omega$ is obtained in terms of the fluid velocity relative to the point instead of the fluid absolute velocity. For a general, volume-specific quantity, $\mathbf{q}$, of arbitrary tensorial order, the net flow rate out of $\partial\Omega$ then becomes

$$\int_{\partial\Omega} \mathbf{q}\,(\vec{v} - \vec{v_s}) \cdot \vec{n} \qquad \text{instead of} \qquad \int_{\partial\Omega} \mathbf{q}\,\vec{v} \cdot \vec{n},$$

where $\vec{n}$ is the (outward-pointing) unit-surface-normal vector at a generic point in the control surface (that is, $\vec{n} = (n_x, n_y, n_z)^T$, where n_α is the component of the vector along the Cartesian coordinate direction α), $\vec{v_s}$ is the velocity vector of the control surface at that point, and $\vec{v}$ is the absolute velocity vector of the fluid at that same point. The velocity vectors are expressed in terms of a Cartesian frame of reference; for example, $\vec{v} = (u, v, w)^T$, where u, v, and w are the three conventionally-ordered Cartesian components of the velocity vector.

If mass is to be conserved, then the rate of change of mass within the control volume, $\frac{\partial}{\partial t} \int_\Omega \rho$, where ρ is the density, must be equal and opposite to the net rate at which mass leaves the control surface, $\int_{\partial\Omega} \rho \left(\vec{v} - \vec{v_s}\right) \cdot \vec{n}$. Here, the convected volume-specific quantity $\mathbf{q}$ is the scalar ρ. After rearrangement, this leads to Equation 2.1.

If linear momentum is to be conserved, assuming no body forces, the net (pressure) force applied to the control surface, $\int_{\partial\Omega} -p\vec{n}$, where p is the pressure, must be equal to the net rate of momentum flow out of the control surface, $\int_{\partial\Omega} \rho\vec{v} \left(\vec{v} - \vec{v_s}\right) \cdot \vec{n}$, together with the rate of change of momentum of the contents of the control volume, $\frac{\partial}{\partial t} \int_\Omega \rho\vec{v}$. Here, the convected volume-specific quantity $\mathbf{q}$ is the vector $\rho\vec{v}$. After rearrangement, this leads to Equation 2.2, which may be viewed as n scalar equations, where $n \in \{1, 2, 3\}$ is the dimension of the vector space of the vector $\vec{v}$ or, equivalently, n is the dimension of the physical space being considered. A similar equation for conservation of angular momentum may be obtained by taking the cross-product of Equation 2.2 with the position vector relative to the point about which the angular momentum is to be evaluated.

The inclusion of body forces would introduce an additional term given by $\int_\Omega \rho\vec{f}$, where $\vec{f}$ is the mass-specific body-force vector, on the right-hand side of Equation 2.2. Body forces are here excluded from the outset because they clutter the momentum and energy conservation equations without leading to any deeper understanding of the fundamental properties of The Euler System. This is the only simplification that will be made here relative to the full Euler System. Body forces are also usually negligible in aerodynamic applications and in compressible industrial gas flows.

If energy is to be conserved, assuming no body forces, then the rate of pressure-work at the control surface, $\int_{\partial\Omega} -p\,\vec{v}\cdot\vec{n}$ (and not $\int_{\partial\Omega} -p\,(\vec{v}-\vec{v_s})\cdot\vec{n}$), must equal the net rate of energy flow out of the control surface, $\int_{\partial\Omega} \rho E\,(\vec{v}-\vec{v_s})\cdot\vec{n}$, together with the rate at which energy changes within the control volume, $\frac{\partial}{\partial t}\int_{\Omega} \rho E$. Energy in this situation can be stored and convected in two possible forms: (i) internal energy; and, (ii) kinetic energy. Therefore, the volume-specific convected quantity here, $\mathbf{q}$, is the scalar ρE, where E is the mass-specific stagnation (or total) energy, $E = e + \frac{1}{2}(\vec{v}\cdot\vec{v})$, where e is the mass-specific internal energy, and the dot product term represents the mass-specific kinetic energy. This leads to Equation 2.3.

$$\frac{\partial}{\partial t}\int_{\Omega}\rho \quad + \quad \int_{\partial\Omega}\rho\,(\vec{v}-\vec{v_s})\cdot\vec{n} \quad = \quad 0. \tag{2.1}$$

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\vec{v} \quad + \quad \int_{\partial\Omega}\rho\vec{v}\,(\vec{v}-\vec{v_s})\cdot\vec{n} \quad = \quad \int_{\partial\Omega} -p\vec{n}. \tag{2.2}$$

$$\frac{\partial}{\partial t}\int_{\Omega}\rho E \quad + \quad \int_{\partial\Omega}\rho E\,(\vec{v}-\vec{v_s})\cdot\vec{n} \quad = \quad \int_{\partial\Omega} -p\,\vec{v}\cdot\vec{n}. \tag{2.3}$$

Equations 2.1 to 2.3 are collectively often called the **The Integral Form** of **The System of Euler Equations** (in honor of Leonhard Euler), although originally the name "Euler Equations" designated the Differential Form of this system, given in the next subsection.

A pre-requisite for the validity of Equations 2.1 to 2.3 is that the indicated surface and volume integrals and time derivatives exist. This is satisfied if any discontinuities in the dependent variables occur only in sets of measure zero within the control volume and on the control surface. The restriction on the existence of time-derivatives arises because the equations were derived starting from the "rate" expression of the conservation axioms, following common practice. Starting from the "difference" expression of the axioms would have led to a more general, but analytically less convenient, form in which no time-derivatives appear, namely:

$$\int_{\Omega_{t_2}} \rho \quad - \quad \int_{\Omega_{t_1}} \rho \quad + \quad \int_{t_1}^{t_2} \int_{\partial\Omega} \rho \, (\vec{v} - \vec{v_s}) \cdot \vec{n} \;=\; 0, \tag{2.4}$$

$$\int_{\Omega_{t_2}} \rho\vec{v} \quad - \quad \int_{\Omega_{t_1}} \rho\vec{v} \quad + \quad \int_{t_1}^{t_2} \int_{\partial\Omega} \rho\vec{v} \, (\vec{v} - \vec{v_s}) \cdot \vec{n} \;=\; \int_{t_1}^{t_2} \int_{\partial\Omega} -p\vec{n}, \quad \text{and} \tag{2.5}$$

$$\int_{\Omega_{t_2}} \rho E \quad - \quad \int_{\Omega_{t_1}} \rho E \quad + \quad \int_{t_1}^{t_2} \int_{\partial\Omega} \rho E \, (\vec{v} - \vec{v_s}) \cdot \vec{n} \;=\; \int_{t_1}^{t_2} \int_{\partial\Omega} -p \, \vec{v} \cdot \vec{n}, \tag{2.6}$$

where t_1 and t_2 represent the initial and final endpoints of the time interval over which the interaction across the control surface occurs, respectively, and Ω_{t_1} and Ω_{t_2} represent the control volume at those two endpoints, respectively.

Regardless of the dimension of the physical space, n, the number of scalar equations still falls short of the number of unknowns. However, the Two-Property Rule for pure substances (which include homogeneous ideal gases) [386] asserts that only two thermodynamic properties may be independently specified for any thermodynamic state. This enables the a-priori selection of two independent state variables and the introduction of one or more algebraic "closing" equations of state to eliminate all other state variables in terms of the two selected ones. For The System of Euler Equations, ρ and e are frequently chosen as the independent state variables, with E eliminated using its defining equation (given above), and p eliminated via the state equation $p = (\gamma - 1)\,\rho e$.

After augmenting equations 2.1 to 2.3 with the necessary equations of state, a system of $2 + n$ equations that are assumed independent is obtained in $2 + n$ scalar variables (namely, two independent thermodynamic state variables and the n components of the velocity vector). Note that the $2 + n$ scalar variables completely specify a flowfield for an ideal fluid.

Alternative presentations of the derivation of the Integral Form may be found in

[431] and [169, 170]. Perhaps the most noteworthy aspect of the derivation is that it was accomplished by quantifying only the effects of the interaction of the fluid with its surroundings through the velocity and the thermodynamic state variables at the control surface. However, because, as demonstrated above, several assumptions and rules were used implicitly or explicitly in addition to the conservation principles, the most rigorous version of the mathematical theory for ideal, compressible flow starts with Equations 2.1 to 2.3 and the chosen equations of state as its axioms.

2.1.3 Derivation of Alternative Forms

Equations 2.1 to 2.3 may be written in the more compact form

$$\frac{\partial}{\partial t}\int_{\Omega}\vec{U} + \int_{\partial\Omega}\vec{\vec{F}}_r \cdot \vec{n} = \int_{\partial\Omega}\vec{\vec{S}} \cdot \vec{n}, \tag{2.7}$$

where for 3-D spaces, the first-order tensor $\vec{U}$ and the second-order tensors $\vec{\vec{F}}_r$ and $\vec{\vec{S}}$ are given by

$$\vec{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}, \quad \vec{\vec{F}}_r = \begin{pmatrix} \rho u_r & \rho v_r & \rho w_r \\ \rho u_r u & \rho v_r u & \rho w_r u \\ \rho u_r v & \rho v_r v & \rho w_r v \\ \rho u_r w & \rho v_r w & \rho w_r w \\ \rho u_r E & \rho v_r E & \rho w_r E \end{pmatrix}, \quad \text{and} \quad \vec{\vec{S}} = \begin{pmatrix} 0 & 0 & 0 \\ -p & 0 & 0 \\ 0 & -p & 0 \\ 0 & 0 & -p \\ -pu & -pv & -pw \end{pmatrix},$$

where u_r, v_r, and w_r are the Cartesian velocity components *relative* to the control surface (that is, $u_r = u - u_s$, $v_r = v - v_s$, and $w_r = w - w_s$), and where all other symbols are as defined above. Note that the $\int_{\partial\Omega}\vec{\vec{S}} \cdot \vec{n}$ term in the right-hand side may also be written in the form $\int_{\partial\Omega}(0, -p\vec{n}, -p\,\vec{v}\cdot\vec{n})^T$, where the second element in the vector integrand is itself an n-dimensional vector. Equation 2.7 is said to be in **Standard Conservation-Law Form**, but clearly this label should also be applicable to the set of Equations 2.1 to 2.3.

The reduction of Equation 2.7 to 2-D spaces may be syntactically accomplished by elimination of, say, the fourth rows in $\vec{U}$, $\vec{\vec{F_r}}$, and $\vec{\vec{S}}$, and the third columns in $\vec{\vec{F_r}}$ and $\vec{\vec{S}}$, effectively eliminating the momentum equation in the z direction and all other effects of the velocity component in the z direction. The velocity component in the z direction must also be eliminated from all independent variables that include it. A similar reduction may again be applied to obtain the corresponding equation in 1-D spaces.

Referring to the terms defined for Equation 2.7, the i^{th} component of $\vec{U}$ can be interpreted as the *volume-specific intensity* of the i^{th} conserved variable, assuming that these variables are arranged in order of mass, the n momenta, and energy. The vector $\vec{U}$ is, however, traditionally, though misleadingly, often called the **Conserved-Variable Vector**. The $(i,j)^{th}$ component of $\vec{\vec{F_r}}$ can be interpreted as the flux of the i^{th} conserved variable in the j^{th} Cartesian coordinate direction. The i^{th} column of $\vec{\vec{F_r}}$ may therefore be regarded as the flux vector of the conserved variables in the i^{th} Cartesian coordinate direction, and the flux tensor $\vec{\vec{F_r}}$ may therefore be expressed in the convenient form

$$\vec{\vec{F_r}} = \left(\vec{F_r} \quad , \qquad \vec{G_r} \quad , \qquad \vec{H_r} \right), \tag{2.8}$$

where the column vectors $\vec{F_r}$, $\vec{G_r}$, and $\vec{H_r}$ refer to the flux vectors in the x, y, and z Cartesian coordinate directions, respectively. The specific form of $\vec{\vec{F_r}}$ given for Equation 2.7 may correctly be called the **Flux Tensor**.

The term $\frac{\partial}{\partial t}\int_\Omega \vec{U}$ in Equation 2.7 represents the rate at which the conserved quantities are changing within the control volume. The term $\int_{\partial\Omega} \vec{\vec{F_r}} \cdot \vec{n}$ represents the rate at which the conserved variables are being removed from the control volume by convective transport through the control surface. The elements of the term $\frac{\partial}{\partial t}\int_\Omega \vec{U}$ are therefore occasionally referred to as the accumulation terms, while the elements of the term $\int_{\partial\Omega} \vec{\vec{F_r}} \cdot \vec{n}$ are often referred to as the convective terms. The term $\int_{\partial\Omega} \vec{\vec{S}} \cdot \vec{n}$ represents the rate at which momenta and energy are being increased by the actions on the control surface of the pressure force and the pressure work, respectively. In this context, the pressure-force and the pressure-work terms are most appropriately viewed as source terms for the momentum and energy conservation equations, respectively. There is no source term for the mass conservation equation.

Equation 2.7 is valid for control volumes of arbitrary geometry and motion, and will be the starting point for the discretization procedure described in the next chapter.

By Liebnitz's Rule

$$\frac{\partial}{\partial t} \int_{\Omega} \vec{U} = \int_{\Omega} \frac{\partial}{\partial t} \vec{U} + \int_{\partial\Omega} (\vec{U} \circ \vec{v_s}^T) \cdot \vec{n},$$

where the operator $\circ$ represents the dyadic product. Note that the partial differentiation in the left-hand term is equivalent to total differentiation with respect to t, since variation of the differentiated integral in the present context is only possible in time.

Let $\vec{\vec{F}}$ be defined from $\vec{\vec{F_r}}$ by replacing every relative velocity component with the corresponding absolute velocity component. In 3-D, $\vec{\vec{F}}$ would then be given by

$$\vec{\vec{F}} = \begin{pmatrix} \rho u & \rho v & \rho w \\ \rho u^2 & \rho v u & \rho w u \\ \rho u v & \rho v^2 & \rho w v \\ \rho u w & \rho v w & \rho w^2 \\ \rho u E & \rho v E & \rho w E \end{pmatrix}.$$

By inspection,

$$\int_{\partial\Omega} \vec{\vec{F}} \cdot \vec{n} - \int_{\partial\Omega} \vec{\vec{F_r}} \cdot \vec{n} = \int_{\partial\Omega} (\vec{U} \circ \vec{v_s}^T) \cdot \vec{n}.$$

Therefore, Equation 2.7 may be re-written in another conservation-law form:

$$\int_\Omega \frac{\partial}{\partial t}\vec{U} + \int_{\partial\Omega} \vec{\vec{F}} \cdot \vec{n} = \int_{\partial\Omega} \vec{\vec{S}} \cdot \vec{n}. \tag{2.9}$$

By comparison of Equation 2.9 with Equation 2.7 from which it was derived, it is evident that the change in the mass, the momentum, and the energy within a control volume due to motion of the control surface is exactly counterbalanced by the change in the mass, momentum, and energy fluxes at the control surface that is caused *only by this motion.* This result holds regardless of the geometry of the control surface or its motion, as can be confirmed by observing that

$$\vec{\vec{F}}_r = \vec{U} \circ \vec{v_r}^T \qquad \text{and} \qquad \vec{\vec{F}} = \vec{U} \circ \vec{v}^T.$$

In analogy with the form in Equation 2.8, the tensor $\vec{\vec{F}}$ may be expressed in the form

$$\vec{\vec{F}} = \left(\begin{array}{ccc} \vec{F} \quad , & \vec{G} \quad , & \vec{H} \end{array} \right), \tag{2.10}$$

where the vectors $\vec{F}$, $\vec{G}$, and $\vec{H}$ respectively have analogous interpretations to the vectors $\vec{F_r}$, $\vec{G_r}$, and $\vec{H_r}$ in Equation 2.8.

The alternative to the Integral Formulation pursued so far is the **Differential Formulation** which may be derived by expressing the conservation constraints for an elemental fluid parcel (that is, a parcel of infinitesimal extent) instead of for a control volume (that is, a volume of possibly finite extent). However, the two forms are not equivalent unless certain conditions on the smoothness in space and time of the dependent variables are satisfied.

From the physical point of view, whenever the Differential and Integral Formulations are not equivalent, the Integral Formulation can be viewed as the more correct one because it reflects more closely the observational basis of the conservation principles being expressed. From the mathematical point of view, the Integral Form is the more general one at least because it admits non-smooth solutions. The distinction between the Integral and Differential Forms is particularly important for The System of Euler Equations and this distinction is propagated to the discretization and discrete solution stages, as discussed in the next chapter. In order to show clearly how the loss of generality arises, the Differential Form will next be derived from the Integral Form.

If $\vec{\vec{F}}$ and p are everywhere differentiable, the Gauss-Ostrogradskii Divergence Theorem may be invoked to assert that

$$\int_{\partial\Omega} \vec{\vec{F}} \cdot \vec{n} = \int_{\Omega} \nabla \cdot \vec{\vec{F}}, \tag{2.11}$$

$$\int_{\partial\Omega} -p\vec{n} = \int_{\Omega} -\nabla p, \qquad \text{and} \tag{2.12}$$

$$\int_{\partial\Omega} -p\,\vec{v} \cdot \vec{n} = \int_{\Omega} -\nabla \cdot (p\vec{v}). \tag{2.13}$$

Substituting these results in Equation 2.9 gives

$$\int_{\Omega} \left(\frac{\partial}{\partial t}\vec{U} + \nabla \cdot \vec{\vec{F}} \right) = \int_{\Omega} \nabla \cdot \vec{\vec{S}}, \tag{2.14}$$

where $\nabla \cdot \vec{\vec{S}} = (0, -\nabla p, \nabla \cdot (p\vec{v}))^T$, where ∇p represents an n-dimensional vector in general.

Since Equation 2.14 must be valid for arbitrary control volumes, it can only be satisfied if the integrands of the left- and right-hand sides are identically equal at every point in space. Therefore, the integrals in Equation 2.14 may be discarded, giving one of the standard Differential Forms:

$$\frac{\partial \vec{U}}{\partial t} + \nabla \cdot \vec{\vec{F}} = \nabla \cdot \vec{\vec{S}}. \tag{2.15}$$

Because of the appearance in it of derivatives, Equation 2.15 is valid only if $\vec{\vec{F}}$ and $\vec{\vec{S}}$ are differentiable. Presumably because of this requirement, the differential form of the mass conservation equation was initially called the continuity equation, although the usage has now spread even to the integral form of that equation. Despite the differentiability requirement, discontinuous solutions of Equation 2.15 can be considered to exist in the sense of distributions. This is the basis of the idea of the **Weak Formulation**, which is obtained by multiplying the Differential Form by an infinitely differentiable test function, $\phi \in C_0^1\left(\mathbf{R^n}\right)$, where C_0^1 is the space of continuously-differentiable functions of compact support, and then performing integration by parts to remove any partial derivatives of possibly discontinuous variables. The resulting equations can be expressed in the form

$$\frac{\partial}{\partial t}\int_\Omega \phi\vec{U} + \int_{\partial\Omega}\phi\vec{\vec{F}}_r \cdot \vec{n} - \int_\Omega \nabla\phi \cdot \vec{\vec{F}}_r = \int_{\partial\Omega}\phi\vec{\vec{S}} \cdot \vec{n} - \int_\Omega \nabla\phi \cdot \vec{\vec{S}}. \tag{2.16}$$

It can be shown [335] that the Weak Formulation, which is clearly an integral formulation, is equivalent to the Integral Formulation of Equation 2.9 in regions were the solution is smooth. In regions where the solution is discontinuous, however, the Weak Formulation gives solutions that satisfy the algebraic jump conditions across discontinuities [335], as described further in Section 2.2. Thus, the Weak Formulation enables the validity of a differential form to be extended to solutions that contain discontinuities on a set of measure zero in the physical space being considered in a manner that circumvents the need to derive the governing equations in integral form.

The Differential Form of Equation 2.15 can be re-written in the form

$$\frac{\partial\vec{U}}{\partial t} + \frac{\partial\vec{\vec{F}}}{\partial x} + \frac{\partial\vec{\vec{G}}}{\partial y} + \frac{\partial\vec{\vec{H}}}{\partial z} = \vec{0}, \tag{2.17}$$

where the vectors $\vec{\vec{F}}$, $\vec{\vec{G}}$, and $\vec{\vec{H}}$ are respectively defined by

$$\vec{\vec{F}} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho uH \end{pmatrix}, \qquad \vec{\vec{G}} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ \rho vH \end{pmatrix}, \qquad \text{and} \qquad \vec{\vec{H}} = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ \rho wH \end{pmatrix},$$

and H is the mass-specific stagnation enthalpy, defined by $H = E + \frac{p}{\rho} = h + \frac{1}{2}(\vec{v} \cdot \vec{v})$. This form may be conveniently obtained by substituting Equation 2.10 into Equation 2.15, followed by absorption of the source terms in Equation 2.15 into the flux vectors. Despite inclusion of the source terms into the vectors $\vec{\vec{F}}$, $\vec{\vec{G}}$, and $\vec{\vec{H}}$, they are still usually called flux vectors in the context of the form of Equation 2.7.

Equation 2.17 may now be re-written in the **Quasi-Linear Form**:

$$\frac{\partial \vec{U}}{\partial t} + \vec{\vec{A}}_{cons}\frac{\partial \vec{U}}{\partial x} + \vec{\vec{B}}_{cons}\frac{\partial \vec{U}}{\partial y} + \vec{\vec{C}}_{cons}\frac{\partial \vec{U}}{\partial z} = \vec{0}, \tag{2.18}$$

where the tensors $\vec{\vec{A}}_{cons}$, $\vec{\vec{B}}_{cons}$, and $\vec{\vec{C}}_{cons}$ are respectively the Jacobians with respect to vector $\vec{U}$ of the vectors $\vec{\vec{F}}$, $\vec{\vec{G}}$, and $\vec{\vec{H}}$:

$$\vec{\vec{A}}_{cons} = \frac{\partial \vec{\vec{F}}}{\partial \vec{U}}, \qquad \vec{\vec{B}}_{cons} = \frac{\partial \vec{\vec{G}}}{\partial \vec{U}}, \qquad \text{and} \qquad \vec{\vec{C}}_{cons} = \frac{\partial \vec{\vec{H}}}{\partial \vec{U}}.$$

Although the temporal and spatial differentiations in Equation 2.18 are applied to $\vec{U}$, the flux vectors are linearized with respect to the elements of $\vec{U}$ as the independent variables.

2.2 Properties of The System of Euler Equations

This section discusses the mathematical properties of The System of Euler Equations, and the physical phenomena that these properties are capable of portraying.

This approach to understanding The Euler System is the consistent and natural viewpoint following the axiomatic construction of the preceding section. The Euler System is most conveniently studied in three parts: (i) general mathematical and physical properties; (ii) mathematical and physical properties in differentiable regions; and, (iii) mathematical and physical properties *across* discontinuities. The discussion in this section is divided accordingly.

2.2.1 General Properties

Several general observations can be made about the Differential Form of The System of Euler Equations derived in section 2.1: (i) there are only first-order derivatives (both in space and in time); (ii) the equations are nonlinear; and, (iii) the equations are coupled. The presence of first-order derivatives and the absence of second-order ones imply that the equations describe phenomena that are dominated by convection rather than by diffusion. The nonlinearity encountered is not only of the "material nonlinearity" type introduced through the equations of state, but also of the "geometric nonlinearity" type introduced through the products of velocities. It is the latter nonlinearity, particularly in the convective terms, which usually causes the most difficulty in obtaining both analytical and numerical solutions of The System of Euler Equations.

The remainder of this section is devoted to the discussion of six specific, fundamental, characteristic properties of The Euler System.

1. The Isenthalpic Property

The energy conservation equation may be extracted from Equation 2.15 in the form

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho u H}{\partial x} + \frac{\partial \rho v H}{\partial y} + \frac{\partial \rho w H}{\partial z} = 0. \tag{2.19}$$

Equation 2.19 can be re-written in the form

$$\frac{\partial \rho H}{\partial t}+\frac{\partial \rho u H}{\partial x}+\frac{\partial \rho v H}{\partial y}+\frac{\partial \rho w H}{\partial z}=\frac{\partial p}{\partial t} \tag{2.20}$$

by replacing E by $H-\frac{p}{\rho}$ in the temporal term. Subtracting from Equation 2.20 the mass conservation equation from the System 2.15 multiplied by H gives

$$\frac{\partial H}{\partial t}+u\frac{\partial H}{\partial x}+v\frac{\partial H}{\partial y}+w\frac{\partial H}{\partial z}=\frac{DH}{Dt}=\frac{1}{\rho}\frac{\partial p}{\partial t}. \tag{2.21}$$

Equation 2.21 shows that the material derivative of the total enthalpy, $\frac{DH}{Dt}$, depends only on the density and the time derivative of pressure, and that for steady flow, the material derivative vanishes, that is,

$$\frac{DH}{Dt}=0, \tag{2.22}$$

implying that the total enthalpy remains constant along streamlines. By its derivation, this result is only valid in differentiable regions. The total enthalpy may jump across shear and contact waves, but since the relative velocity component normal to the discontinuity is zero for these waves, the condition $\frac{DH}{Dt}=0$ is trivially satisfied in steady flow, regardless of the jump in total enthalpy. Shock waves, on the other hand, have a non-vanishing normal velocity component but [431] have a vanishing total enthalpy jump. Therefore, the result of Equation 2.22 is valid for all steady solutions of The Euler Equations, with or without discontinuities. That is, every valid steady solution of The Euler System must exhibit the isenthalpic property. The shock, shear, and contact wave discontinuities mentioned above and the changes in properties across them are explained and discussed in the third sub-section of this Section.

The equation

$$\nabla H\cdot\vec{v}=0 \tag{2.23}$$

is also universally satisfied for steady flows wherever ∇H is defined. For homenthalpic flows, the equation is trivially satisfied. Since any non-vanishing gradients in the total enthalpy field must be perpendicular to the streamlines by Equation 2.22, Equation 2.23 must also be satisfied for non-homenthalpic flows.

2. The Isentropic Property

Subtracting from the x-momentum equation in the System 2.15 the mass conservation equation from the same system multiplied by u gives

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = \frac{Du}{Dt} = -\frac{1}{\rho}\frac{\partial p}{\partial x}. \tag{2.24}$$

Analogous results apply for the other two momentum conservation equations, and the three equations may be recombined into one vector equation (first derived by Euler, and known as **Euler's Equation** or **The Inviscid Momentum Equation**), given by

$$\frac{D\vec{v}}{Dt} = -\frac{1}{\rho}\nabla p. \tag{2.25}$$

Subtracting the energy conservation equation (Equation 2.19) from the mass conservation equation multiplied by E gives

$$\frac{DE}{Dt} = -\frac{1}{\rho}\nabla\cdot p\vec{v}. \tag{2.26}$$

Since $E = e + \frac{1}{2}\vec{v}\cdot\vec{v}$, since $\frac{D\frac{1}{2}\vec{v}\cdot\vec{v}}{Dt} = \vec{v}\cdot\frac{D\vec{v}}{Dt}$, and since $\vec{v}\cdot\frac{D\vec{v}}{Dt} = -\frac{\vec{v}}{\rho}\cdot\nabla p$ by Equation 2.25, Equation 2.26 can be reduced to

$$\frac{De}{Dt} = -\frac{1}{\rho}\nabla\cdot(p\vec{v}) + \frac{\vec{v}}{\rho}\nabla p = -\frac{p}{\rho}\nabla\cdot\vec{v}. \tag{2.27}$$

Since $\nabla\cdot\vec{v} = -\frac{1}{\rho}\frac{D\rho}{Dt}$ from the mass conservation equation, Equation 2.27 can be re-written as

$$\frac{De}{Dt} = \frac{p}{\rho^2}\frac{D\rho}{Dt}. \tag{2.28}$$

Taking the total derivative of the equation of state $p = (\gamma - 1)\rho e$ gives the equation

$$\frac{De}{Dt} = \frac{1}{\gamma - 1}\left(\frac{1}{\rho}\frac{Dp}{Dt} - \frac{p}{\rho^2}\frac{D\rho}{Dt}\right). \tag{2.29}$$

Elimination of the term $\frac{De}{Dt}$ from Equations 2.29 and 2.28 gives

$$\frac{Dp}{Dt} = \left(\frac{\gamma p}{\rho}\right)\frac{D\rho}{Dt}. \tag{2.30}$$

From the equality $s = c_v \ln\left(\frac{p}{\rho^\gamma}\right) + s_0$, the material derivative of the entropy may be expressed by

$$\frac{Ds}{Dt} = \frac{c_v}{p}\left(\frac{Dp}{Dt} - \frac{\gamma p}{\rho}\frac{D\rho}{Dt}\right). \tag{2.31}$$

Inserting Equation 2.30 into Equation 2.31 gives the final result

$$\frac{Ds}{Dt} = 0, \tag{2.32}$$

proving that the entropy along any streamline in a smooth region of the flow is constant. Differentiability is sufficient but not necessary for the isentropic condition (as illustrated by the example of expansion fans, which are everywhere continuous but not everywhere differentiable).

By parallel reasoning to that given for Equation 2.23, the isentropic property implies that the relation

$$\nabla s \cdot \vec{v} = 0 \tag{2.33}$$

is valid for all (that is, homentropic as well as non-homentropic) smooth flows.

3. The Vorticity Preservation Property

From Kelvin's Circulation Theorem, it can be deduced that in isentropic flow, The Euler System can neither create nor destroy vorticity, even in the presence of solid boundaries. However, The Euler System also correctly transports throughout the solution domain any vorticity initially present in the flowfield or supplied at boundaries. The key elements of the modeling of vorticity in The Euler System can be deduced most readily from analyzing the compressible, inviscid vorticity equation, namely

$$\frac{D\vec{\omega}}{Dt} = \frac{\partial\vec{\omega}}{\partial t} + \vec{v}\cdot\nabla\vec{\omega} = \vec{\omega}\cdot\nabla\vec{v} - \vec{\omega}\nabla\cdot\vec{v} + \nabla p \times \nabla\left(\frac{1}{\rho}\right), \tag{2.34}$$

where $\vec{\omega} = \nabla \times \vec{v}$ is the vorticity, and all the remaining symbols have the meanings defined for them above. The term $\nabla p \times \nabla\left(\frac{1}{\rho}\right)$ in Equation 2.34 vanishes for barotropic conditions (including incompressible or homentropic flows), reducing the equation to

$$\frac{D\vec{\omega}}{Dt} = \vec{\omega}\cdot\nabla\vec{v} + \frac{\vec{\omega}}{\rho}\frac{D\rho}{Dt} \tag{2.35}$$

after substituting $\frac{1}{\rho}\frac{D\rho}{Dt}$ for $\nabla\cdot\vec{v}$ (from the mass conservation equation). Equation 2.35 may be re-written in the form

$$\frac{D\left(\frac{\vec{\omega}}{\rho}\right)}{Dt} = \left(\frac{\vec{\omega}}{\rho}\right)\cdot\nabla\vec{v}, \tag{2.36}$$

which shows that if vorticity is initially absent in the flowfield, then its rate of change will remain zero for all time. This result is known as Helmholtz's First Law.

Since a material line, $\vec{\Delta l}$, also obeys an evolution equation of the form

$$\frac{D\vec{\Delta l}}{Dt} = \vec{\Delta l}\cdot\nabla\vec{v}, \tag{2.37}$$

it can be concluded that if a material line and a vortex coincide in location at some time, that is, if

$$\Delta\vec{l} = k\left(\frac{\vec{\omega}}{\rho}\right)$$

at $t = t_0$, then they will remain coincident for all time $t \geq t_0$. This implies that vortex lines are convected with the fluid velocity. This result is known as Helmholtz's Second Law.

For any two-dimensional or any non-swirling, axisymmetric flow, Equation 2.36 reduces to a scalar advection equation for $\frac{\vec{\omega}}{\rho}$, namely

$$\frac{D\left(\frac{\vec{\omega}}{\rho}\right)}{Dt} = 0, \tag{2.38}$$

implying that $\frac{\vec{\omega}}{\rho}$ is convected unchanged along streamlines. If, in addition, the flow is incompressible, then 2.36 reduces to a scalar advection equation for $\vec{\omega}$, namely

$$\frac{D\vec{\omega}}{Dt} = 0, \tag{2.39}$$

implying that $\vec{\omega}$ is convected unchanged (that is, as a conserved variable) along streamlines.

Crocco's Theorem for steady flow, namely

$$\vec{v} \times \vec{\omega} = \nabla H - T\nabla s, \tag{2.40}$$

identifies more precisely the mechanisms by which the vorticity field may be changed in The Euler System. In particular, other than for flows in which the vorticity and velocity vectors are everywhere aligned (Beltrami Flows), vorticity cannot be introduced in any nontrivial flow that is homenthalpic and homentropic. In 2-D flows, the vorticity and velocity vectors cannot be aligned in any non-trivial flow, and therefore, every steady, 2-D, homenthalpic, homentropic solution to The Euler System must be irrotational. For general non-homenthalpic or non-homentropic fields, vorticity must be present unless the non-uniformities precisely cancel each other everywhere in the flowfield.

In non-smooth regions, the entropy may change, for example, across shock waves, and therefore be accompanied by the production of vorticity. Indeed, shock surfaces along which the shock-strength varies (such as curved shocks) are typically sources of vorticity. This is the case for both two- and three-dimensional flows, but for two-dimensional flows, such shock surfaces are the only possible sources of vorticity.

4. The Flux-Homogeneity Property

All components of the Flux Tensor, $\vec{\vec{F}}$, for The Euler System are homogeneous functions of degree one in the variables of the Conserved-Variable Vector, to wit:

$$\vec{\vec{F}}\left(\alpha\vec{U}\right) = \alpha\vec{\vec{F}}\left(\vec{U}\right), \tag{2.41}$$

where α is an arbitrary real number. This property is valuable for proving other properties, and is an important foundation for a certain family of numerical schemes, as described in the next chapter.

Differentiating Equation 2.41 with respect to α and then assigning the value of α to 1 gives the result

$$\vec{\vec{F}}\left(\vec{U}\right) = \frac{\partial\vec{\vec{F}}}{\partial\vec{U}}\vec{U}, \tag{2.42}$$

implying that the flux Jacobian may freely be "moved" inside the divergence operator in Quasi-Linear Forms, such as that of Equation 2.18.

5. Form Invariance to Coordinate Transformations

The Euler Equations in Conservation-Law Form, for example, as in Equation 2.9, remain in Conservation-Law Form under any non-singular coordinate transformation. This result may be expressed in the form

$$\int_{\Omega}\frac{\partial}{\partial t}\vec{U}+\int_{\partial\Omega}\vec{\vec{F}}\cdot\vec{n} = \int_{\partial\Omega}\vec{\vec{S}}\cdot\vec{n} \quad\Longrightarrow\quad \int_{\tilde{\Omega}}\frac{\partial}{\partial t}\left(\tilde{\vec{U}}J_{\Omega}\right)+\int_{\partial\tilde{\Omega}}\left(\tilde{\vec{\vec{F}}}J_{\partial\Omega}\right)\cdot\vec{n} = \int_{\partial\tilde{\Omega}}\left(\tilde{\vec{\vec{S}}}J_{\partial\Omega}\right)\cdot\vec{n}, \tag{2.43}$$

where the symbol $\tilde{(.)}$ denotes the dependent variables in the transformed space, and where J_{Ω} and $J_{\partial\Omega}$ (which must be everywhere nonvanishing) are respectively the volume and surface Jacobians of the transformation.

A compact proof of this property, which is particularly useful for establishing Finite-Difference-Based discrete solution schemes on curvilinear structured grids, is given in [212], and the result can also be generalized to the Arbitrary-Lagrangian-Eulerian form of Equation 2.7.

6. Rotational Invariance of the Flux Tensor

If the flux tensor at a point is assumed to depend only on the state vector at that point, that is, $\vec{\vec{F}} = \vec{\vec{F}}(\vec{U})$, then the rotational invariance property can be expressed by the equation

$$\vec{\vec{F}}(\vec{U}) = \vec{\vec{R}}^{-1}\,\tilde{\vec{\vec{F}}}(\vec{\vec{R}}\vec{U}), \tag{2.44}$$

where the tensor $\tilde{\vec{\vec{F}}}$ refers to the flux tensor in the rotated coordinate system, and $\vec{\vec{R}}$ is the rotation matrix that expresses the coordinate transformation.

For more general cases, including flows with discontinuities, the flux at a point in a surface may be assumed to depend only on the state vectors on each of the two sides of the surface, that is, $\vec{\vec{F}} = \vec{\vec{F}}(\vec{U}_L, \vec{U}_R)$, where the subscripts L and R refer to the "left" and "right" locations relative to the surface. In such cases, the rotational invariance property can be expressed by the equation

$$\vec{\vec{F}}(\vec{U}_L, \vec{U}_R) = \vec{\vec{R}}^{-1}\,\tilde{\vec{\vec{F}}}(\vec{\vec{R}}\vec{U}_L, \vec{\vec{R}}\vec{U}_R), \tag{2.45}$$

where all terms are as defined above. In applications of this property, the rotation is most often restricted to cases in which the rectangular coordinate axes are re-aligned so that one of them falls onto the local normal of the surface, $\vec{n}$.

The above results can be verified by direct computation for arbitrary $\vec{\vec{R}}$ (which is always invertible, since all rotations are invertible), for example, by first re-writing the flux vectors $\vec{F}$, $\vec{G}$, and $\vec{H}$ in Equation 2.7 in terms of the scalar variables $\vec{U}(1)$ to $\vec{U}(5)$.

In 2-D, the rotation matrix is always given by

$$\vec{\vec{R}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where θ is the rotation angle about the z axis. The matrix can also be expressed in terms of n_x and n_y, since $n_x = \cos\theta$, and $n_y = \sin\theta$.

In 3-D, the rotation matrix must be defined from a composition of finite rotations about each of the three coordinate axes. Thus, in the most general case, $\vec{\vec{R}} = \vec{\vec{R}}_x \vec{\vec{R}}_y \vec{\vec{R}}_z$, where the individual rotation matrices are given by

$$\vec{\vec{R}}_x = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta & 0 \\ 0 & 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\vec{\vec{R}}_y = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\phi & 0 & \sin\phi & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

and

$$\vec{\vec{R}}_z = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi & 0 & 0 \\ 0 & \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

where θ, ϕ, and ψ are respectively the rotation angles about the x, y, and z axes. The order of the individual rotations in the composite rotation is non-unique, and in some cases could be arbitrary.

Examination of the expression for $\vec{\vec{R}}$ for the 2-D case and for any composition of the 3-D matrices, $\vec{\vec{R}}_x$, $\vec{\vec{R}}_y$, and $\vec{\vec{R}}_z$ shows that the inverse transformation, $\vec{\vec{R}}^{-1}$, for each of Equations 2.44 and 2.45 is always defined.

The rotational invariance property is of crucial importance in developing discrete solution schemes based on one-dimensional flux functions for multi-dimensional computations, as will be shown in the next chapter.

2.2.2 Properties in Differentiable Regions

As would be expected, the "continuous" regime is most conveniently studied using the Quasi-Linear and Differential Forms of the governing equations. The Quasi-Linear Form is particularly valuable for analytical purposes because it allows much of the mathematical and physical properties of The Euler System to be revealed through analysis of the eigenstructure of the equations in that form, that is, by enabling the analysis of a nonlinear system in a (linear) vector space, as will be shown below. For brevity and generality, only the 3-dimensional case is considered. The reductions to 2- and 1-dimensional physical spaces are indicated towards the end of this subsection.

Direct evaluation of the "flux" Jacobians $\vec{\vec{A}}_{cons}$, $\vec{\vec{B}}_{cons}$, and $\vec{\vec{C}}_{cons}$ in the Quasi-Linear Form of Equation 2.18 gives

$$\vec{\vec{A}}_{cons} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ (\frac{\gamma-1}{2})q^2 - u^2 & (3-\gamma)u & (1-\gamma)v & (1-\gamma)w & (\gamma-1) \\ -uv & v & u & 0 & 0 \\ -uw & w & 0 & u & 0 \\ (\frac{\gamma-1}{2})uq^2 - uH & H-(\gamma-1)u^2 & (1-\gamma)uv & (1-\gamma)uw & \gamma u \end{pmatrix},$$

$$\vec{\vec{B}}_{cons} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -uv & v & u & 0 & 0 \\ (\frac{\gamma-1}{2})q^2 - v^2 & (1-\gamma)u & (3-\gamma)v & (1-\gamma)w & (\gamma-1) \\ -vw & 0 & w & v & 0 \\ (\frac{\gamma-1}{2})vq^2 - vH & (1-\gamma)uv & H-(\gamma-1)v^2 & (1-\gamma)vw & \gamma v \end{pmatrix},$$

and

$$\vec{\vec{C}}_{cons} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ -uw & w & 0 & u & 0 \\ -vw & 0 & w & v & 0 \\ (\frac{\gamma-1}{2})q^2 - w^2 & (1-\gamma)u & (1-\gamma)v & (3-\gamma)w & (\gamma-1) \\ (\frac{\gamma-1}{2})wq^2 - wH & (1-\gamma)uw & (1-\gamma)vw & H-(\gamma-1)w^2 & \gamma w \end{pmatrix},$$

where q^2 is the magnitude of the velocity vector, that is, $q^2 = u^2 + v^2 + w^2$, and all other symbols are as defined above.

Equation 2.17 may also be expressed in terms of another Quasi-Linear Form in which the conservative-variable vector, $\vec{U}$ is replaced by a "primitive"-variable vector, $\vec{P}$, defined by $\vec{P} = (\rho, u, v, w, p)^T$, and so named because its elements are regarded the most "elementary" descriptors of state. This can be accomplished by inserting the transformation T defined by $\vec{\vec{T}} = \frac{\partial \vec{U}}{\partial \vec{P}}$ into Equation 2.18 to give

$$\vec{\vec{T}}\frac{\partial \vec{P}}{\partial t} + \vec{\vec{A}}_{cons}\vec{\vec{T}}\frac{\partial \vec{P}}{\partial x} + \vec{\vec{B}}_{cons}\vec{\vec{T}}\frac{\partial \vec{P}}{\partial y} + \vec{\vec{C}}_{cons}\vec{\vec{T}}\frac{\partial \vec{P}}{\partial z} = \vec{0}. \tag{2.46}$$

By direct evaluation, T is found to be

$$\vec{\vec{T}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ u & \rho & 0 & 0 & 0 \\ v & 0 & \rho & 0 & 0 \\ w & 0 & 0 & \rho & 0 \\ \frac{q^2}{2} & \rho u & \rho v & \rho w & \frac{1}{\gamma-1} \end{pmatrix}.$$

The inverse transformation tensor for $\vec{\vec{T}}$ (which can be shown to always exists since $\det(\vec{\vec{T}}) = \rho^3/(\gamma-1) > 0$ in any region where The Euler System has meaning) is given by

$$\vec{\vec{T}}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{-u}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ \frac{-v}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ \frac{-w}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ (\gamma-1)\frac{q^2}{2} & (1-\gamma)u & (1-\gamma)v & (1-\gamma)w & (\gamma-1) \end{pmatrix}.$$

Applying the inverse transformation $\vec{\vec{T}}^{-1}$ to Equation 2.46 yields The Euler Equations in the **Primitive Quasi-Linear Form**, namely

$$\frac{\partial \vec{P}}{\partial t} + \vec{\vec{A}}_{prim}\frac{\partial \vec{P}}{\partial x} + \vec{\vec{B}}_{prim}\frac{\partial \vec{P}}{\partial y} + \vec{\vec{C}}_{prim}\frac{\partial \vec{P}}{\partial z} = \vec{0}, \tag{2.47}$$

where the Primitive Flux Jacobians are related to their conservative counterparts by the relations

$$\vec{\vec{A}}_{prim} = \vec{\vec{T}}^{-1}\vec{\vec{A}}_{cons}\vec{\vec{T}}, \qquad \vec{\vec{B}}_{prim} = \vec{\vec{T}}^{-1}\vec{\vec{B}}_{cons}\vec{\vec{T}}, \qquad \text{and} \qquad \vec{\vec{C}}_{prim} = \vec{\vec{T}}^{-1}\vec{\vec{C}}_{cons}\vec{\vec{T}},$$

and are given by

$$\vec{\vec{A}}_{prim} = \begin{pmatrix} u & \rho & 0 & 0 & 0 \\ 0 & u & 0 & 0 & \frac{1}{\rho} \\ 0 & 0 & u & 0 & 0 \\ 0 & 0 & 0 & u & 0 \\ 0 & \rho c^2 & 0 & 0 & u \end{pmatrix},$$

$$\vec{\vec{B}}_{prim} = \begin{pmatrix} v & 0 & \rho & 0 & 0 \\ 0 & v & 0 & 0 & 0 \\ 0 & 0 & v & 0 & \frac{1}{\rho} \\ 0 & 0 & 0 & v & 0 \\ 0 & 0 & \rho c^2 & 0 & v \end{pmatrix},$$

and

$$\vec{\vec{C}}_{prim} = \begin{pmatrix} w & 0 & 0 & \rho & 0 \\ 0 & w & 0 & 0 & 0 \\ 0 & 0 & w & 0 & 0 \\ 0 & 0 & 0 & w & \frac{1}{\rho} \\ 0 & 0 & 0 & \rho c^2 & w \end{pmatrix},$$

where c^2 is the square of the linearized sound-speed, defined from $c^2 = \frac{\partial p}{\partial \rho}\big]_s$, where s is the entropy. For an ideal gas, this defining relation can be shown to be equivalent to the relation $c = \sqrt{\gamma R T}$, where R is the Universal Gas Constant, γ is the ratio of specific heat capacities ($\gamma = \frac{c_p}{c_v}$, where c_p is the specific heat capacity at constant pressure, and c_v the specific heat capacity at constant volume), and T is the absolute temperature. The latter relation is the one usually employed to evaluate c.

Although the Flux Jacobians of the Primitive Quasi-Linear Form are simpler than the Jacobians of the Conservative Quasi-Linear Form, the eigenvalues of the two forms must be identical since the Jacobians of the two forms are derivable from each other by a similarity transformation. The Primitive Quasi-Linear Form may also be considered the more general of the two because its direct derivation does not require the use of an equation of state, unlike the situation for the Conservative Quasi-Linear Form. The primary analytical use of the Primitive Quasi-Linear Form is to enable more convenient derivation of the Characteristic Quasi-Linear Form than is possible from the Conservative Quasi-Linear Form. The Characteristic Quasi-Linear Form will be shown over the course of the remainder of this chapter to reveal many important aspects of the mathematical structure of The System of Euler Equations.

The Characteristic Quasi-Linear Form is most conveniently derived from the Primitive-Variable Form by selecting an arbitrary vector in space, $\vec{n} = (n_x, n_y, n_z)^T$, along which to constrain the spatial differentiation of the Primitive Quasi-Linear Form. The flux Jacobian matrix along that vector, $\vec{\vec{D}}$ can then be obtained simply by combining the three flux Jacobians in the Cartesian directions according to this constraint, that is,

$$\vec{\vec{D}}_{char} = n_x \vec{\vec{A}}_{prim} + n_y \vec{\vec{B}}_{prim} + n_z \vec{\vec{C}}_{prim}. \tag{2.48}$$

Evaluation of the above expression gives

$$\vec{\vec{D}}_{char} = \begin{pmatrix} u_\perp & \rho n_x & \rho n_y & \rho n_z & 0 \\ 0 & u_\perp & 0 & 0 & n_x/\rho \\ 0 & 0 & u_\perp & 0 & n_y/\rho \\ 0 & 0 & 0 & u_\perp & n_z/\rho \\ 0 & \rho c^2 n_x & \rho c^2 n_y & \rho c^2 n_z & u_\perp \end{pmatrix},$$

where $u_\perp$ is the projection of the fluid velocity vector along the arbitrary vector $\vec{n}$.

The matrix $\vec{\vec{D}}$ can be shown to have a complete set of real eigenvalues, given by

$$\vec{\lambda} = \left(u_\perp - c, u_\perp, u_\perp, u_\perp, u_\perp + c\right)^T , \tag{2.49}$$

where all symbols are as defined above. For what follows below, it is also convenient to define the eigenvalue matrix, $\vec{\vec{\lambda}}$, given by

$$\vec{\vec{\lambda}} = \begin{pmatrix} u_\perp - c & 0 & 0 & 0 & 0 \\ 0 & u_\perp & 0 & 0 & 0 \\ 0 & 0 & u_\perp & 0 & 0 \\ 0 & 0 & 0 & u_\perp & 0 \\ 0 & 0 & 0 & 0 & u_\perp + c \end{pmatrix} \tag{2.50}$$

The matrix $\vec{\vec{D}}$ also possesses a complete set of linearly independent eigenvectors. Of course, these eigenvectors are not unique, because of the 3-fold multiplicity of the $u_\perp$ eigenvalue, and the particular choice of eigenvectors that will be made here differs from the traditional one, given, for example, in [170]. For the choices made here, the matrix of left Eigenvectors evaluates to

$$\vec{\vec{L}} = \begin{pmatrix} 0 & -n_x & -n_y & -n_z & \frac{1}{\rho c} \\ 1 & 0 & 0 & 0 & \frac{-1}{c^2} \\ 0 & 0 & \frac{-n_z}{n_y^2+n_z^2} & \frac{n_y}{n_y^2+n_z^2} & 0 \\ 0 & -1 & \frac{n_x n_y}{n_y^2+n_z^2} & \frac{n_x n_z}{n_y^2+n_z^2} & 0 \\ 0 & n_x & n_y & n_z & \frac{1}{\rho c} \end{pmatrix} , \tag{2.51}$$

while the matrix of right Eigenvectors evaluates to

$$\vec{\vec{R}} = \begin{pmatrix} \frac{\rho}{2c} & 1 & 0 & 0 & \frac{\rho}{2c} \\ \frac{-n_x}{2} & 0 & 0 & -(n_y^2+n_z^2) & \frac{n_x}{2} \\ \frac{-n_y}{2} & 0 & -n_z & n_x n_y & \frac{n_y}{2} \\ \frac{-n_z}{2} & 0 & n_y & n_x n_z & \frac{n_z}{2} \\ \frac{\rho c}{2} & 0 & 0 & 0 & \frac{\rho c}{2}. \end{pmatrix}. \tag{2.52}$$

The set of eigenvectors given above can be shown by direct verification to be linearly independent. Indeed, for all non-trivial choices of the eigenvectors, the eigenvector set can be shown to be linearly independent, and to satisfy by construction the following two relations between $\vec{\vec{D}}$, $\vec{\vec{L}}$, $\vec{\vec{R}}$, and $\vec{\vec{\lambda}}$:

$$\vec{\vec{L}}\vec{\vec{D}}\vec{\vec{R}} = \vec{\vec{\lambda}}, \tag{2.53}$$

and

$$\vec{\vec{L}}\vec{\vec{R}} = \vec{\vec{I}} = \vec{\vec{R}}\vec{\vec{L}}, \tag{2.54}$$

where $\vec{\vec{I}}$ is the identity transformation matrix.

Since there exists a choice for which the eigenvalues are all real and the set of eigenvectors is linearly independent, the system of equations is purely hyperbolic in time. Furthermore, since the wave-speeds (given by the eigenvalues) are all finite and below the speed of light for the usual practical applications, the system is guaranteed to satisfy the Principle of Causality despite the fact that the principle was not explicitly introduced as an axiom. This fortuitous situation eliminates many of the mathematical and numerical difficulties encountered in other physical models (such as those involving parabolic equations) which do not satisfy the Principle of Causality [205].

The most appealing property of the above choice of eigenvectors is that the characteristic variables can then be expressed in the form

$$\vec{C} = \begin{pmatrix} u_\perp - \frac{p}{\rho c} \\ \rho - \frac{p}{c^2} \\ u_{\|_1} \\ u_{\|_2} \\ u_\perp + \frac{p}{\rho c}. \end{pmatrix}, \tag{2.55}$$

where $u_{\|_1}$ and $u_{\|_2}$ refer to the velocity components in the two directions perpendicular to the arbitrary vector $\vec{n} = (n_x, n_y, n_z)^T$ in an orthogonal coordinate system that are respectively given by $\vec{r}_1 = \left(0, \frac{-n_z}{n_y^2+n_z^2}, \frac{n_y}{n_y^2+n_z^2}\right)^T$, and $\vec{r}_2 = \left(1, \frac{-n_x n_y}{n_y^2+n_z^2}, \frac{-n_x n_z}{n_y^2+n_z^2}\right)^T$, and where all the remaining symbols have the same meanings defined for them above. In this form, the physical meaning of the characteristic variables as conveying an acoustic wave (the element $u_\perp - \frac{p}{\rho c}$ of the vector $\vec{C}$), an entropy wave (the element $\rho - \frac{p}{c^2}$ of the vector $\vec{C}$), two shear waves (the elements $u_{\|_1}$ and $u_{\|_2}$ of the vector $\vec{C}$), and a second acoustic wave (the element $u_\perp + \frac{p}{\rho c}$ of the vector $\vec{C}$) is clearly revealed. This revealing decomposition will be shown to also have a great significance in the analysis of the Riemann Problem, given in the next chapter.

Given that the inverse of the transformation matrix $\vec{\vec{L}}$ can be shown to exist everywhere except in vacua, that is, in every physical situation in which The Euler System applies, the Characteristic Quasi-Linear Form may readily be derived from the Primitive Quasi-Linear Form. This can be accomplished by differentiating Equation 2.47 with respect to $\vec{C}$, substituting $\vec{\vec{L}}^{-1}$ for $\frac{\partial \vec{C}}{\partial \vec{P}}$, and then pre-multiplying by $\vec{\vec{L}}$. This yields the equation

$$\frac{\partial \vec{C}}{\partial t} + \vec{\vec{L}}\vec{\vec{A}}_{prim}\vec{\vec{L}}^{-1}\frac{\partial \vec{C}}{\partial x} + \vec{\vec{L}}\vec{\vec{B}}_{prim}\vec{\vec{L}}^{-1}\frac{\partial \vec{C}}{\partial y} + \vec{\vec{L}}\vec{\vec{C}}_{prim}\vec{\vec{L}}^{-1}\frac{\partial \vec{C}}{\partial z} = \vec{0}. \tag{2.56}$$

Unfortunately, except in the one-dimensional-space case, the Jacobians $\vec{\vec{A}}_{prim}$, $\vec{\vec{B}}_{prim}$, and $\vec{\vec{C}}_{prim}$ are not simultaneously diagonalizable. This is equivalent to stating that the Primitive Jacobians do not have the same eigenvectors. The implication of this situation is that there exists no coordinate direction in which the system of equations can be decoupled. From the practical point of view, this implies that any solution scheme for The Euler System of Equations must be suitable for coupled systems, that is, it must correctly account for the coupling.

The simplification of the above derivations to two- and one-dimensional cases is obvious in most cases. For most of the transformation matrices given above, the two-dimensional specialization can be obtained by elimination of one row and one column and elimination of the w velocity component wherever it appears. The one-dimensional specialization may usually be obtained by elimination of two rows and two columns and the elimination of the w and v velocity components wherever they appear. The multiplicity of the eigenvalues for the shear and entropy waves, together with the physical meaning of the waves represented by the corresponding eigenvectors (as briefly outlined above, and as explained further in [170], for example), lead to the following conclusion: specialization of the system of equations to the two-dimensional case must eliminate one of the shear eigenvalues, while specialization of the system of equations to the one-dimensional case must eliminate the second shear eigenvalue as well, thereby eliminating the multiplicity. The only qualitative change that occurs in the structure of the equations when specializing to lower-dimensional spaces is that the question of simultaneous diagonalizability of the flux Jacobians vanishes in the one-dimensional case.

More detailed derivations of the Quasi-Linear Forms, especially in one- and two-dimensional spaces, as well as a different choice for the eigenvector set for the Characteristic Quasi-Linear Form are given in [170]. References [170] and [426] also discuss the relation between the Characteristic Variables and the Riemann Invariants, and relate the concepts to the Theory of Characteristics.

It was established above in this section that The Euler Equations are purely hyperbolic in time. In space, The Euler Equations may assume three different types, depending on the local flow speed. If the flow is locally subsonic, then the type is partially elliptic (with complex eigenvalues for the acoustic waves, and real eigenvalues for the entropy and shear waves). If the flow is locally sonic, then the type is partially parabolic. If the flow is locally supersonic, then the type is fully hyperbolic. Different types may exist in different parts of the same flow domain. Since different types of PDE require different boundary conditions for well-posedness and require different numerical solution techniques, developing analytic solutions and numerical solution schemes for The Euler Equations has been quite challenging.

2.2.3 Properties Across Discontinuities

An important property of The Euler Equations is that they allow the evolution and propagation of solution discontinuities. The laws governing the propagation of these discontinuities can most rigorously be derived from the integral formulation. Consider a surface discontinuity having a local normal $\vec{n}_d$ and traveling at velocity $\vec{v}_d(t)$, where both $\vec{n}_d$ and $\vec{v}_d(t)$ are functions of location in the surface. Consider a control volume, Ω, enclosing a segment of that surface discontinuity and with a thickness (normal to the discontinuity) that is infinitesimal compared to the two dimensions of the control volume along the discontinuity surface. The jump relations may be derived by directly applying the compact form of Equation 2.7 to the control volume just described.

The first left-hand term of Equation 2.7, $\frac{\partial}{\partial t}\int_\Omega \vec{U}$, is seen to vanish, because it involves a volume integral over an infinitesimal volume. On the other hand, because of the infinitesimal thickness of the control volume, the second left-hand side term of Equation 2.7, the flux integral $\int_{\partial\Omega} \vec{\vec{F}}_r \cdot \vec{n}$, can be re-expressed by $\int_S |[\vec{\vec{F}}_r]| \cdot \vec{n}_d$, where $|[.]|$ represents the jump in quantity (.) across the discontinuity surface, $|[.]| = (.)_i - (.)_f$, where $(.)_i$ is the value of the quantity infront of the discontinuity, and $(.)_f$ is the value behind the discontinuity, and where S is the discontinuity surface segment enclosed within the control volume Ω. Similarly, the right-hand term reduces to $\int_S |[\vec{\vec{S}}]| \cdot \vec{n}_d$, where $\vec{\vec{S}}$ is as defined above and where $|[.]|$ has the same meaning as explained above. Note that in the context of the above, the integral $\int_s \phi$ for quantity ϕ refers to integration of ϕ on the surface of the discontinuity, and does not represent an integral over a closed surface.

Combining the reduced terms yields the general conservation equation across traveling discontinuities:

$$\int_S |[\vec{\vec{F}}_r - \vec{\vec{S}}]| \cdot \vec{n}_d = \vec{0}. \tag{2.57}$$

Since $\vec{\vec{F}}_r = \vec{\vec{F}} - \vec{U} \circ \vec{v}_d^T$, Equation 2.57 may be re-written in the form

$$\int_S |[\vec{\vec{F}} - \vec{\vec{S}} - \vec{U} \circ \vec{v}_d^T]| \cdot \vec{n}_d = \vec{0}. \tag{2.58}$$

This last form could also have been obtained directly by starting from Equation 2.9, as would be expected.

As the above derivation shows, and as would be expected for flow features with infinitesimal geometric dimensions, the jump equations have no "accumulation" terms and must be satisfied instantaneously: they depend only on the instantaneous velocity of the discontinuity, and are independent of its acceleration. The jump relations in a reference frame attached to the moving discontinuity are obtained simply by setting the discontinuity speed $\vec{v}_d$ to zero.

Since Equations 2.57 and 2.58 must be valid for arbitrary control volumes surrounding a discontinuity, the integrals in these equation may be discarded, respectively leaving

$$|[\vec{\vec{F}}_r - \vec{\vec{S}}]| \cdot \vec{n}_d = \vec{0}, \tag{2.59}$$

and

$$|[\vec{\vec{F}} - \vec{\vec{S}}]| \cdot \vec{n}_d - |[\vec{U} \circ \vec{v}_d^T]| \cdot \vec{n}_d = \vec{0}. \tag{2.60}$$

Equations 2.59 and 2.60 may also be re-written in an expanded form; for example, for Equation 2.59, this would be given by

$$\begin{aligned} |[\rho(\vec{v} - \vec{v_d}) \cdot \vec{n_d}]| &= 0, && (2.61) \\ |[\rho\vec{v}(\vec{v} - \vec{v_d}) \cdot \vec{n_d} + p\vec{n_d}]| &= 0, \text{ and} && (2.62) \\ |[\rho E(\vec{v} - \vec{v_d}) \cdot \vec{n_d} + p\vec{v} \cdot \vec{n_d}]| &= 0. && (2.63) \end{aligned}$$

Note that the second of these equations is an n-dimensional vector equation. This set of $2 + n$ scalar equations are called the jump relations, but they are often expressed in different forms using the fact that if any two quantities, a and b, have defined jumps, then $|[a + b]| = |[a]| + |[b]|$.

If all velocities are expressed relative to the discontinuity, the Equations 2.61 to 2.63 simplify to

$$\begin{aligned} |[\rho v_n]| &= 0, && (2.64) \\ |[\rho\vec{v}v_n + p\vec{n}_d]| &= 0, \text{ and} && (2.65) \\ |[\rho E v_n + p v_n]| &= 0, && (2.66) \end{aligned}$$

where v_n is the component of relative velocity directed along the discontinuity normal $\vec{n}_d$. This condition is trivially established for a discontinuity that is stationary in the observation reference frame. The above jump relations are sometimes loosely called the Rankine-Hugoniot Relations, but this label should be distinguished from the original thermodynamic definition of the Rankine-Hugoniot Equation as given, for example, in [431].

The jump relations may also be derived from the Weak Formulation of the conservation laws, as expected from the definition of that formulation. Perhaps the two most noteworthy aspects of the jump-condition derivations are that: (i) the time dependence was eliminated because the volume of the control volume could be made arbitrarily small; and, (ii) the jump relations are expressible purely in algebraic terms. From the physical point of view, the most significant result is that the jump relations establish that all discontinuities of The Euler System possess a hyperbolic character in time and have finite propagation speeds.

Three types of "elemental" discontinuities may exist in The Euler System. If the mass flux through a discontinuity is non-zero (and this implies that the normal velocity components on both sides of the discontinuity are non-zero too), the discontinuity is called a **shock wave**. For both a **normal shock wave** and an **oblique shock wave**, the jump in the tangential velocity, $|[v_t]|$, along the (shock) discontinuity is zero. If the mass flux is zero, then, by Equations 2.61 to 2.63, the only possibility is that the pressure jump is also zero, leaving the possibility of nonzero jumps in only the density and tangential velocity. If the only jump is in the tangential velocity, the discontinuity is called a **shear wave** or a **vortex sheet**. If the only jump is in the density (and hence also in the temperature, since the pressure is invariant), the discontinuity is called a **contact wave** or **contact surface**.

Elemental discontinuities of different types may coincide in location to give compound discontinuities. There are important differences in the mathematical properties of these discontinuities. For example, while shock waves can evolve from smooth solutions and propagate in a stable manner, vortex sheets are neutrally stable only in the (trivial) limit of vanishing jump in tangential velocity. Otherwise, they are unstable (even linearly so), and their evolution is called a Kelvin-Helmholtz instability. Contact discontinuities are neutrally stable.

All the types of discontinuities described above and their characteristic analytical properties are exhibited in flows of real, compressible, homogeneous gases where viscosity, reaction, or other effects have little influence. Also, all three elemental discontinuities mentioned above play an important role in aerodynamic and compressible flows of practical interest.

An important point with regard to the shock waves admitted by The Euler System is that they allow the flow across the shock to occur in both the direction of the pressure increment (giving a compression shock, with an increase in entropy), and against it (giving an expansion shock, with a decrease in entropy). However, expansion shocks are never observed in practice. Although they would satisfy the mass, momentum, and energy conservation laws discussed above, expansion shocks would violate the Second Law of Thermodynamics [386] which was not made part of the classical theory on which The Euler System is founded, as explained above in this section.

Entropy-violating solutions are removed in analytical work either by inspection of the solution set and discarding any non-physical shocks or, preferably, by reformulating The Euler Equations to reduce to the inviscid limit of the Navier-Stokes Equations. The second approach relies on applying an **Entropy Condition**, which is theoretically equivalent to adding the Second Law of Thermodynamics as another equation in The System of Euler Equations. In particular, the Isentropic Property expressed in Equation 2.32 is effectively replaced by the Second Law of Thermodynamics, which in the above context is most conveniently expressed by

$$\frac{Ds}{Dt} = \epsilon_v \geq 0, \tag{2.67}$$

where ϵ_v is the viscous dissipation term in the Navier-Stokes Equations, taken to the vanishing limit.

It is quite remarkable that many computational schemes for The Euler Equations have the tendency to generate expansion shocks in their numerical solutions. Such entropy-violating solutions are eliminated from computational schemes by applying an Entropy Condition at the numerical level, rather than in the system of equations, as described for some schemes in Section 3.2.

The entropy jump across pure shear waves must always be zero, since the Two-Property Rule and the definition of a shear wave imply that the two states separated by such waves are thermodynamically identical. The entropy jump across contact waves may be arbitrary.

Extensive discussions of Riemann Invariants, the jump relations, and the properties of the discontinuities supported by The System of Euler Equations may be found in [91, 170, 426].

2.3 Analytical Solutions of The System of Euler Equations

While much is precisely known about the local properties and behavior of smooth regions and isolated discontinuities in solutions of the full Euler System (see above), no general procedure is currently available for obtaining closed-form whole solutions for that system, and no general existence proofs are currently known, even for smooth initial data [207]. Furthermore, no general results addressing the uniqueness of solutions have yet been derived for The Euler System; rather, recent computational studies [182] appear to indicate the possibility of a fundamental non-uniqueness.

Several analytical solutions have been derived for special cases of The Euler System, but almost always with simple geometries, or for flows that are essentially two-dimensional. For example, for irrotational and isentropic (that is, smooth) flows, The Euler Equations can be reduced to the compressible, velocity-potential equation, namely:

$$\left(1-\frac{u^2}{c^2}\right)\phi_{xx}-\left(2\frac{uv}{c^2}\right)\phi_{xy}+\left(1-\frac{v^2}{c^2}\right)\phi_{yy}-\left(2\frac{vw}{c^2}\right)\phi_{yz}+\left(1-\frac{w^2}{c^2}\right)\phi_{zz}-\left(2\frac{wu}{c^2}\right)\phi_{zx}=0, \tag{2.68}$$

where c is the speed of sound, as defined above, and u, v, and w are respectively the velocity components in the x, y, and z directions, and are related to ϕ by

$$u=\phi_x, \qquad v=\phi_y, \qquad \text{and} \qquad w=\phi_z.$$

For flows that have a Mach number well below one everywhere, or for flows that are purely supersonic, or for flows that approach the incompressible limit, Equation 2.68 can either be approximated by a linear equation or can be linearized by a similarity transformation, such as the Prandtl-Glauert Transformation [431]. When either of these approximations is feasible, a wide range of techniques becomes available to enable the derivation of analytical solutions, provided the geometry is simple enough. If the flowfield includes both supersonic and subsonic regions, the full nonlinear potential equation cannot be approximated by a linear equation, and no analytic solution will generally be possible. One exception is the Hodograph Transformation [142] which allows the nonlinear potential equation to be linearized by expressing it in terms of the velocities as the independent variables, but at the expense of leaving the geometry to be determined as part of the solution rather than a-priori, as done in Ringleb's Flow solution [75].

For discontinuous flows, a wide range of classical solutions has been derived for interaction of simple shocks, expansion fans, shear waves, and contact surfaces with sharp corners or blunt surfaces having simple geometry [75, 325, 326, 431], and for flows in specially-shaped nozzles or ducts [325, 326].

A major thrust in recent analytical efforts is aimed at promoting the uniqueness of The Euler Equations by the seemingly trivial technique of adding an entropy constraint [155, 213, 352]. Another thrust [393, 392] is based on using asymptotic analysis techniques to obtain two-dimensional steady-state solutions for flows that include shocks and strong vorticity effects. Otherwise, there has been little progress over the past decade or two in developing analytical techniques for solving The Euler Equations or in finding new solutions for general cases.

2.4 Practical Relevance of The System of Euler Equations

The majority of fluid-dynamic problems studied in current industrial and commercial practice either comprise incompressible flows, or flows that are strongly dependent on the effects of viscosity, heat conductivity, mixing of species, or reaction. Since it does not model these physical phenomena, the fundamental System of Euler Equations as described in this chapter is not appropriate for studying such flows. By its assumptions and its construction, The System of Euler Equations is confined to accurate modeling of convection-dominated, adiabatic, compressible flows of real, thermodynamically homogeneous gases, including ideal gases as a special case.

Depending on the application, convective effects are typically considered to dominate the effects of viscous diffusion when the Reynolds Number exceeds $1.0e+06$. In this regime, viscous boundary layers are typically thin compared to the "free-flow" cross-sections. It is also possible to improve on the inviscid predictions by attempting to incorporate the displacement effects of the viscous boundary layer, either by displacing the original solid boundaries, or by applying a "transpiration" flowfield normal to these boundaries [412]. Depending on the application, compressibility effects are typically considered to become important when the Mach Number exceeds 0.3. The Euler Equations are particularly appropriate for transonic and supersonic flows, since the correct modeling of the effects of convection and compressibility in this regime translate into correct predictions of the strengths, speeds, and locations of the shock, shear, or contact discontinuities which are invariably present and important.

As would be expected from their correct modeling of the coupled effects of convection and compressibility, the most appropriate application areas for The Euler Equations involve turbomachinery, blast and detonation waves, and external aerodynamics. Aerodynamic applications also rely heavily on the capacity of The Euler Equations to accurately model a third physical phenomenon: vorticity.

As described in Section 2.2, The Euler Equations correctly convect any vorticity that is initially present in the flowfield or applied at the boundaries. Furthermore, since The Euler Equations can support discontinuities in the velocity (that is, vortex sheets), they can in principle have unique and correct solutions for practical problems involving circulation, lift, or propulsion, such as jet flows, or flows over airfoils or propellers, given appropriate boundary and initial conditions. On the other hand, as also explained in Section 2.2, there is no physical mechanism in The Euler System to generate vorticity in homenthalpic, homentropic flows. In principle, this would imply that a computational solution that is started with homentropic, homenthalpic initial conditions could not be made to converge to the correct flowfield as the infinite-time limit of a transient solution for any problem involving circulation, separation, or lift.

In practice, despite the analytical inability to generate vorticity and circulation in smooth flows, The Euler System is usually found to give accurate computational predictions of flow separations (including the locations and strengths of vortex sheets), provided the separation behavior is dominated by the geometry, rather than by precise viscous effects [325, 326] (that is, provided the separation is of the so-called "inviscid separation" type). This allows correct prediction of the circulation and inviscid lift on objects with sharp geometric features. This unexpected behavior apparently occurs because the artificial viscosity introduced in almost all numerical schemes appears to act in the same manner as physical viscosity and to be normally of sufficient strength to automatically impose the Kutta Condition or another similar analytical device.

A practical contributing factor to the widespread use of The Euler Equations during the past decade or two for aerodynamic applications is probably due to a favorable cost-benefit situation in relation to the relative fidelities and computational requirements of the level of dynamic approximation immediately above and that immediately below The Euler Equations in the hierarchy of models for fluid dynamics. Solutions of The Euler Equations are far more economical than those of the Compressible Navier-Stokes Equations. On the other hand, solutions of The Euler Equations are far more reliable and useful than those of the Compressible Potential Flow Model, which is incapable of modeling vorticity, or any phenomena associated with it. As computing power continues to increase, the popularity of computational solutions for the Navier-Stokes Equations will probably continue to increase, and the role of The Euler Equations will probably become increasingly restricted to preliminary design and parametric studies.

From the academic viewpoint, The Euler Equations remain important as a model system because the treatment of the nonlinear convective terms (which are identical to the corresponding ones in the Navier-Stokes equations) poses the greatest difficulty in obtaining accurate numerical solutions, especially in the computational solution of discontinuities [169, 170]. The Euler Equations also remain important because all satisfactory discretization and solution techniques (including, for example, convergence acceleration methods) for the Navier-Stokes Equations are expected to be able to accurately predict the inviscid limit, and must therefore be founded from techniques that satisfactorily resolve The Euler System.

CHAPTER III

Discretization and Discrete Solution of the Governing Equations

This chapter presents the discretization formulations and the discrete solution schemes adopted or developed in this work to obtain numerical solutions to The System of Euler Equations, and it discusses the chief properties of these formulations and solution schemes. Alternative feasible choices are also reviewed to provide a background for the work. The way in which the crucial and characterizing physical phenomena captured in The Euler Equations are incorporated into the discrete formulation and preserved in the discrete solution is described. The application of sound boundary and initial conditions in the discrete scheme is also discussed. The manner in which the discrete solution algorithm is incorporated with the other major algorithms to create the overall algorithm for the methodology developed in this work is also outlined.

A general development for arbitrarily moving and deforming grids is adopted throughout this chapter, leading directly to the special considerations and solution procedures required for deforming cells, and in the vicinity of moving boundaries. Sufficient detail is provided in the coverage to enable complete reproduction of all the solution schemes and methods used in this work.

3.1 Selection of the Discretization and Discrete-Solution Methodologies

The purpose of a discretization procedure is to establish a valid "projection" of the continuous governing equations over a continuous space (here, a closed subset of the $(\vec{x}, t)$ space) onto a set of discrete equations over a set of discrete spatio-temporal entities (here, the set of spatial subregions in the grid at different time levels).

The purpose of a discrete solution scheme is to numerically solve the set of discrete equations on the set of discrete spatio-temporal subregions derived from the discretization procedure. As would be expected, different discrete solution schemes are often better suited for different discretization procedures.

There is currently a wide variety of general-purpose methods, each with its own strengths and weaknesses, for discretization of The System of Euler Equations (which is the system adopted in this work). Selecting the most appropriate method depends on the type of the grid to be used, on the preferred scheme for discrete solution of the discretized equations, as well as on various other requirements, such as the accuracy and computational effort targets. A brief review of these methods is given in Appendix B.

A Finite-Volume discretization procedure is chosen in this work in preference to all the other feasible alternatives (which are outlined in Appendix B) for the following reasons: (i) its suitability for the Cartesian-Quadtree type of grid used in this work; (ii) its widespread use and advanced state of development; (iii) its allowance of easier implementation of explicit time-integration schemes, and easier and more robust formulation and solution of problems with discontinuities by direct capturing of the discontinuities; and, (iv) its allowance of easier imposition of the appropriate boundary conditions, especially for moving boundaries.

Although the Vertex-Centered family of discretizations is currently the most popular among the three main families of the Finite-Volume Method (which are listed in Appendix B), a Cell-Centered discretization was chosen in this work. Vertex-Centered discretizations are the most popular because for tetrahedral and triangular cells, the ratio of cells to vertices asymptotically lies typically between 5 and 6, and between 2 and 3, respectively, resulting in greater storage-space requirements for Cell-Centered discretizations. In this work, however, the predominant cell shape is the quadrilateral, and for quadrilaterals and hexahedra, the cell-to-vertex ratio is asymptotically close to 1. Another reason for generally favoring Vertex-Centered discretizations is that the numerical interdependence between vertex values is higher than that between the corresponding cell values (by about a factor of 3 - 4 for tetrahedra, and by about a factor of 2 for triangles). Again, the corresponding factor for quadrilaterals and hexahedra is close to 1.

Despite the above apparent advantages of Vertex-Centered schemes for triangles and tetrahedra, no discernible improvement in the accuracy of Vertex-Centered schemes for inviscid flows has so far been demonstrated relative to the increases in memory or computational effort requirements. Indeed, the opposite seems to be more true. Perhaps the only verifiable advantage of Vertex-Centered schemes is that for viscous flows, they allow easier discretization of the second derivatives, typically by central differencing. Since a Vertex-Centered scheme has no advantage for the type of element or cell chosen in this work, the deciding criterion of geometric simplicity favors the choice of a Cell-Centered discretization.

Independently of whether a Finite-Difference, Finite-Element, or Finite-Volume method is chosen for the discretization, another major choice in selecting the discretization procedure is whether to separate or to combine the spatial and temporal discretizations. Separation is relatively the more common of the two approaches and typically results in larger stencils in "space", requires multi-step time-integration schemes for higher-order accuracy, but removes any dependence of a steady-state solution on the time-step in a pseudo-marching calculation. In this work, the spatial and temporal discretizations are developed separately, mostly to ensure the highest degree of flexibility in dealing with arbitrary cell geometries, and to simplify the overall solution algorithm.

Since the technique being developed in this work is applicable to moving-boundary problems, where the geometry of subregions changes with time, a crucial consideration for the chosen discretization scheme is that it must allow satisfaction of the Geometric Conservation Laws. The definitions, implications, and significance of these "laws" are discussed in detail in Section 3.4 below.

Once the discretization procedure has been specified or chosen, the next step is to specify the discrete solution scheme. The major choices to be made are whether the scheme will be implicit or explicit, and the exact procedure that will be followed to solve the discrete equations. The details of the scheme chosen in this work are described below in this chapter. Because the interest in this work is mostly directed toward time-accurate simulations, an explicit time-stepping scheme is chosen in the discrete solution scheme, facilitating further the separation between the spatial and temporal discretizations, as explained in detail below.

3.2 Spatial Discretization: Alternatives and Specific Implementation

The discretization procedure chosen in this work, the **Cell-Centered Finite-Volume Methodology**, requires the Computational Region (see Section 4.3 for the definition) to be divided into a set of non-overlapping subregions (or computational cells) that satisfy all the fundamental requirements discussed in Section 4.3. With such a set of non-overlapping subregions (or computational cells), the volume and surface integrals appearing in the integral formulation of the governing equations are individually approximated for each cell and each cell face respectively, using appropriate quadratures. The integral formulation for The System of Euler Equations is given and discussed in Chapter II generally, and in Section 2.1 particularly. The solution values at cell centers are taken as the primary unknowns, and as being the volume averages over the corresponding computational cells, while the solution values at cell faces are typically obtained by interpolation of cell-centered values of the solution, as will be shown in detail in the remainder of this section.

Consider an arbitrary individual computational cell in the subregion set. The cell may be moving and deforming. A polyhedral example of such a cell (with five faces) is shown in Figure 3.1. Since the cell satisfies the definition of a control volume, Equation 2.7 may be applied to it. Integrating the equation in time from $t = t_i$ to $t = t_f$ yields

$$\int_{\Omega_f} \vec{U} - \int_{\Omega_i} \vec{U} = \int_{t_i}^{t_f} \int_{\partial\Omega} (-\vec{\vec{F}}_r + \vec{\vec{S}}) \cdot \vec{n}, \tag{3.1}$$

where Ω_i and Ω_f are the closures of the interior of the cell at times t_i and t_f, respectively, and all other symbols are as defined in Chapter II.

The terms in the surface integral of Equation 3.1 may be re-written in the form

$$\vec{\vec{F}}_r \cdot \vec{n} = \begin{pmatrix} \rho v_{r_\perp} \\ \rho u v_{r_\perp} \\ \rho v v_{r_\perp} \\ \rho w v_{r_\perp} \\ \rho E v_{r_\perp} \end{pmatrix}, \quad \text{and} \quad \vec{\vec{S}} \cdot \vec{n} = \begin{pmatrix} 0 \\ -p n_x \\ -p n_y \\ -p n_z \\ -p(u n_x + v n_y + w n_z) \end{pmatrix}, \tag{3.2}$$

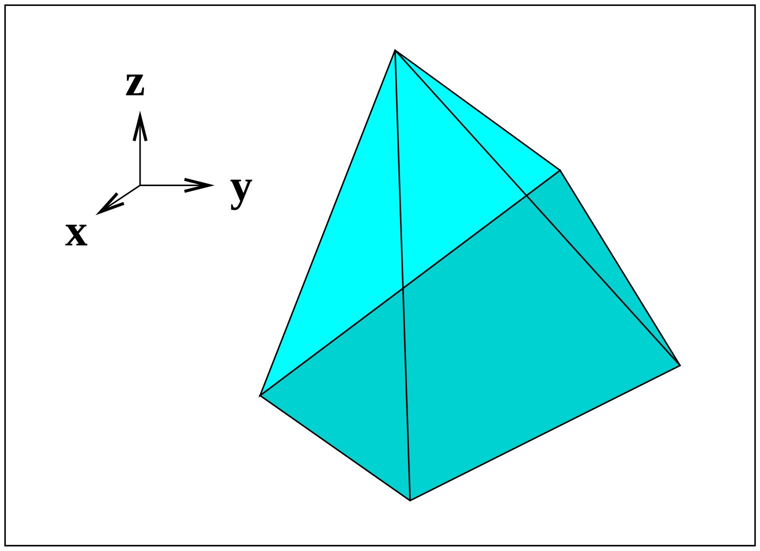

Figure 3.1: A generic three-dimensional polyhedral cell (with five planar polygonal faces).

where $v_{r_\perp}$ is the component of the velocity vector that is aligned with the local outward-pointing normal of the surface of the cell being considered, (that is, $v_{r_\perp} = u_r n_x + v_r n_y + w_r n_z$), and where all the other symbols are as defined above. Now, let $\vec{H}_\perp$ be defined by $(\vec{\vec{F}}_r - \vec{\vec{S}}) \cdot \vec{n}$. Then, it follows that

$$\vec{H}_\perp = \begin{pmatrix} \rho v_{r_\perp} \\ \rho u v_{r_\perp} + p n_x \\ \rho v v_{r_\perp} + p n_y \\ \rho w v_{r_\perp} + p n_z \\ \rho E v_{r_\perp} + p v_\perp \end{pmatrix} . \tag{3.3}$$

The vector $\vec{\vec{F}}_r \cdot \vec{n}$ above may be replaced by $\vec{F_{r_\perp}} = v_{r_\perp} \vec{U}$ and the vector $\vec{\vec{S}} \cdot \vec{n}$ above may be replaced by $\vec{S}_\perp$, where the symbol $\perp$ denotes the same "surface-perpendicularity" intent described above. Note again that the source term vector $\vec{S}_\perp$ is independent of the motion of the surface of the control volume. Note that the fifth element of the vector $\vec{H}_\perp$ may be not be written as $\rho H v_{r_\perp}$, nor as $\rho H v_\perp$, where, as in Section 2.2, $H = E + \frac{p}{\rho}$ is the mass-specific stagnation enthalpy. The vector $\vec{H}_\perp$ is usually called the normal-flux vector, although, as shown above, it incorporates the source terms for the momentum and energy equations.

Defining $\overline{\vec{U}}$ to be the volume-averaged value of the solution vector, to wit,

$$\overline{\vec{U}} = \frac{1}{V} \int_\Omega \vec{U}, \tag{3.4}$$

where V is the volume of the cell, and using the new variable $\vec{H}_\perp$, Equation 3.1 may be re-written in the form

$$\overline{\vec{U}_f} V_f - \overline{\vec{U}_i} V_i = \int_{t_i}^{t_f} \int_{\partial\Omega} -\vec{H}_\perp, \tag{3.5}$$

where the subscripts i and f have the meanings described above. By its derivation, Equation 3.5 is valid regardless of the motion or deformation of the control surface.

Considering only polyhedral cells whose number of faces does not change during the integration interval (that is, topologically-invariant polyhedra), Equation 3.5 can be specialized to the form

$$\overline{\vec{U}_f} V_f - \overline{\vec{U}_i} V_i = \int_{t_i}^{t_f} \sum_{f=1}^{nFaces} \int_{S_f} -\vec{H}_\perp, \tag{3.6}$$

where f is the face number, where S_f represents the surface of the general cell face over which the area integration is carried out, and where $nFaces$ is the number of faces of the polyhedral cell. The specialization to topologically-invariant polyhedra is particularly relevant here since, as described in Chapter VII, the spatial discretization of the Computational Region is designed to generate only cells that satisfy this requirement, regardless of the motion of boundaries.

Since the faces of a polyhedron are all planar polygons, Equation 3.6 may be re-written in the form

$$\overline{\vec{U}_f} V_f - \overline{\vec{U}_i} V_i = \int_{t_i}^{t_f} \sum_{f=1}^{nFaces} -\overline{\vec{H}_{\perp f}} S_f, \tag{3.7}$$

where S_f is the area of the general face, and where $\overline{\vec{H}_{\perp f}} = \overline{\vec{H}(t)_{\perp f}}$ is the *instantaneous* area-averaged value of the normal-flux-vector integral in Equation 3.6, to wit,

$$\overline{\vec{H}_{\perp f}} = \frac{1}{S_f} \int_{S_f} \vec{H}_{\perp}$$

Equation 3.7 is said to be in "semi-discrete" form, since it embodies a complete discretization insofar as the spatial dependence of the unknowns is concerned, but is still in the "continuum" form as far as the temporal dependence is concerned. As briefly mentioned above and as explained in more detail below, in this work, the temporal discretization is separated completely from the spatial discretization (rather than mixed with it), following the most common current practice.

It should be noted that no approximations have yet been committed in the discretization process. Indeed, for topologically-invariant polyhedral cells, Equation 3.7 is equivalent to the generalized integral form of Equation 2.7. Furthermore, since the discrete unknowns in a Cell-Centered discretization are the volume-averaged state-vectors in the cells, and since these averages are updated directly in the Finite-Volume Method, no approximations are required for the volume integrals of the average state vectors. This may be regarded as one of the most appealing features of the Cell-Centered discretization methodology.

In order to continue the spatial discretization beyond Equation 3.7, it is now necessary to express $\vec{H}_{\perp}$ or $\overline{\vec{H}_{\perp f}}$ in an approximate form that depends solely on the discrete solution. It is only in this operation that numerical approximations in the spatial discretization will be committed, and these approximations can therefore be viewed as being confined to estimating the surface-averaged values of the fluxes. The most common alternatives to approximating or computing these fluxes are discussed in the next sub-section.

Regardless of the approximation technique chosen to compute the fluxes, one of the characteristics of the Finite-Volume Method is that the evaluation of the flux vector associated with each cell face, namely, $\vec{H}_{\perp}$, should be unique to that face. That is, the same flux is applied to both of the two cells that share that face. This requirement *alone* guarantees the conservation property, since the fluxes on all cell faces that do not belong to the boundary of the Computational Region could then only cause re-distribution of conserved quantities among the cells in the Computational Region, but cannot contribute to the creation or destruction of conserved quantities. This is the so-called "numerical telescoping" property (that is, the simultaneous satisfaction of the conservation property across arbitrary agglomerations of contiguous cells at the numerical level). The significance of the conservation property for computation of flows with discontinuities is that it can be proved [215] that if a numerical solution converges and is stable, then it will converge to the weak solution of the system, implying that the numerical solution is guaranteed to converge to a solution in which the jump conditions and discontinuity speeds are correctly predicted.

The conservation property can also be enforced in Finite-Difference and Finite-Element Methods, but in contrast to the situation with the Finite-Volume Method, doing so usually involves additional special treatments in the design of the discretization scheme.

3.2.1 Computation of the Flux

The preceding sub-section derived the semi-discrete equation for the generic Cell-Centered Finite-Volume formulation, namely, Equation 3.7, and explained how the numerical approximation in the spatial discretization can be viewed as wholly confined in the discrete evaluation of the area-averaged flux on each cell face, $\overline{\vec{H}_{\perp f}}$ in that equation. This sub-section presents and discusses the different methods of approximating and computing the area-averaged flux vector that were adopted and used in this work, and also briefly discusses other alternatives.

There are currently two main families of approaches to computing fluxes for advection-dominated flows: (i) that associated with the segregated-solution approach (or the so-called "pressure-based" approach); and, (ii) that associated with the coupled-solution approach (or so-called "density-based" approach). These two approaches, their relative strengths and weaknesses, and their implications as far as computing the fluxes are briefly outlined in Appendix C. Since this work is concerned primarily with compressible and strongly-compressible flows, a coupled-solution approach was adopted. However, it should be noted that the advantages of the coupled-solution approach here are limited, and a segregated-solution approach would also have been a feasible alternative. Indeed, a segregated-solution approach has been used in a related work [8] in which the flow is incompressible.

Within the category of coupled-solution approaches (or so-called "density-based" approaches), there are two main approaches to evaluating the fluxes: (i) those based on Centered Approximations (or Centered Discretizations); and (ii) those based on Upwind Approximations (or Upwind Discretizations). There are also several other less-commonly-used approaches, and a great deal of freedom exists in selecting approximations, resulting in significant variations in the robustness, the accuracy, and the computational effort-requirements of the overall scheme. A brief review of Centered Discretizations is given in Appendix D, while Upwind Discretizations are discussed in more detail below, and in appendices cited there. Appendix D also includes a brief comparison between Centered and Upwind Discretizations, which may be better read after the next sub-sub-section.

Although a Centered Discretization would have been a perfectly valid choice for this work, such a discretization was not adopted, and mostly because of a subjective preference for the alternative approach of Upwind Discretization, which, as shown below, has a clearer and more intuitive connection with the wave-propagation behavior embodied in hyperbolic systems of equations.

3.2.1.1 Upwind Flux Functions

Upwind discretizations are based on directional biasing of the differencing, depending on the direction of propagation of physical (and hence numerical) signals. These discretizations are also often called characteristic-based discretizations. The upwind-biasing idea appears to have been systematically introduced first in [90] in an attempt to extend the Method of Characteristics to fixed, regular Cartesian grids. Although successful, the approach was based on a non-conservative Finite-Difference Method and was therefore incapable of satisfying the weak-solution requirements. This meant that it could not correctly "capture" the jumps across discontinuities. Discontinuity-capturing schemes are defined as schemes that automatically satisfy the jump conditions across discontinuities, including prediction of the correct discontinuity speeds, on arbitrary fixed grids, and without invoking techniques that attempt to explicitly locate the discontinuities. The modern, shock-capturing version of Upwind-Differencing was introduced by Godunov in a Finite-Volume framework [170], and then developed further by Van Leer [374, 375, 376, 377, 378], Roe [308, 308, 309, 310], and others.

For a system of equations, including The System of Euler Equations, the basic idea of these Upwind Discretizations is to decompose the total flux into the contributions to it from individual propagating waves and then to perform directionallybiased differencing for each of these waves depending on its direction of propagation. For The Euler System, the waves are inherent in the hyperbolicity of the system, as explained in Chapter II. Apart from the ability of these schemes to capture discontinuities, one of the main reasons for their widespread appeal is the firm physical foundation on which they are based; namely, the Characteristic Theory for hyperbolic systems of Partial Differential Equations. Invariably, the flux in these discretizations, $\vec{H}_\perp$ is evaluated using formulae of the general form

$$\vec{H}_\perp = \vec{H}(\overline{\vec{U}_L}, \overline{\vec{U}_R}) = \frac{1}{2}\left(\vec{H}(\overline{\vec{U}_L}) + \vec{H}(\overline{\vec{U}_R}) - \vec{\mathcal{H}}(\overline{\vec{U}_L} - \overline{\vec{U}_R})\right), \tag{3.8}$$

where $\overline{\vec{U}_L}$ and $\overline{\vec{U}_R}$ are respectively the left and right states across the interface on which the flux $\vec{H}_\perp$ is being evaluated, and where the condition

$$\vec{\mathcal{H}}(\overline{\vec{0}}) = \vec{0}$$

must always be satisfied for a consistent discretization.

Most Upwind Discretization schemes are based on one of three techniques, depending on the manner in which the interface flux is computed, and more specifically, on the manner in which $\vec{\mathcal{H}}(\overline{\vec{U}_L} - \overline{\vec{U}_R})$ is evaluated. These three techniques are as follows: (i) Flux-Vector Splitting; (ii) Exact Riemann Solution; and, (iii) Flux-Difference Splitting. These three techniques are briefly described in the remainder of this section or in appropriate appendices cited there. Throughout the presentation, a general formulation is maintained that is valid for arbitrarily moving and deforming cells.

3.2.1.2 Flux-Vector Splitting

Although Flux-Vector Splitting (FVS) techniques of computing the interface flux typically lead to very robust schemes, these techniques were not used in this work, primarily because of their excessive diffusivity for shear waves compared to the other alternatives discussed below [385]. The FVS techniques are reviewed in Appendix E.

3.2.1.3 Exact Riemann Solvers

The Riemann Problem (described in detail, for example, in [91] and [336]) is the name given to a one-dimensional Initial Value wave-propagation problem that arises when an imaginary membrane separating two possibly different states of ideal gas vanishes instantaneously, allowing the two states to interact. The term is also sometimes used when the gases are non-ideal. In order to remove any time-related restrictions on the evolution of the solution, the two states of gas are assumed to be of infinite extent. If no restrictions are placed on the two initial states, the problem is sometimes called an Arbitrary Riemann Problem.

The Riemann Problem has an analytic self-similar solution expressible in terms of a decomposition into distinct propagating waves separating uniform, equilibrium states. Except in one scenario, there will be three waves separating four uniform states. The exception occurs when the left and right states are moving apart with a sufficiently high velocity difference. In that case, four waves will be created, and they will separate five different states one of which one will be the complete-vacuum state. This exception is of little practical interest since most computational schemes would not allow the appearance of a vacuum.

Because of the assumed infinite extent of the two initial states, the outer states remain unchanged for all time. The two (or three) inner states must be determined as part of the solution. The two outer waves must always be either an expansion fan and a shock wave, or two shock waves, or two expansion fans. If there are only three waves in the solution, the inner wave will be a contact. If there are four waves (that is, if a vacuum state is created), then both of the inner waves will be contacts. In all cases, the contact wave may also coincide with a shear wave, and in 3-D, the shear wave may itself be a compound shear wave. The strength of the shear wave is always zero in a one-dimensional problem, and all inner discontinuities are therefore guaranteed to be pure contact waves. The expansion fan always has a finite thickness and continuous variation of properties within it, while the shock, contact, and shear waves are infinitesimally thick and always have discontinuous property variations across them except in the trivial, zero-strength case. The property variations or jumps that arise across each of these types of discontinuity are discussed in detail in Section 2.2 above.

Use of the solutions of Riemann Problems to obtain the exact flux $\vec{H}_\perp$ in terms of propagating wave effects was first proposed by Godunov [135], and later refined and improved by others. In all such uses, the Riemann interface described above is always supposed to coincide with the cell face that separates the two Finite-Volume cells between which the flux is to be computed. Inevitably, this introduces a non-physical, grid-dependent alignment of wave propagation effects. Because of the exactness of the flux estimate with an Exact Riemann Solver, the quality of solution obtainable with this method is very high. Since the scheme is based on a physical analysis which takes into account the Third Law of Thermodynamics, such a flux computation is guaranteed to preserve the jump conditions at the interface, and also to require no special treatment to enforce the Entropy Condition, which was described in Chapter II.

Exact Riemann Solvers are often recommended for problems where high accuracy is required, or in situations with strong jumps across discontinuities. They are especially useful in situations where vacua or very low pressures or densities are likely to arise, as in the modeling of strong explosions, or where the property of "Positive Conservation" (which is defined and described below) is required. In this work, an Exact Riemann solver was frequently used, as described in Chapter VIII.

The Riemann Problem may be reduced to an implicit algebraic equation and computationally solved using any suitable iterative algorithm as described below, or, for example, in [91], [335], [82], or [139]. The four uniform states of the Riemann problem mentioned above will be denoted by the expressions $\vec{P}_{L_{outer}}$, $\vec{P}_{L_{inner}}$, $\vec{P}_{R_{inner}}$, and $\vec{P}_{R_{outer}}$, where $\vec{P}$ is the primitive-variable vector, given by $\vec{P} = (\rho, u, v, w, p)^T$, and where the subscripts L and R refer to the Left and Right states, respectively. In order to avoid cluttering the equations in the remainder of this sub-sub-section, it will be assumed without loss of generality that the coordinate axes are chosen so that the u component of the velocity is perpendicular to the interface between the two initial gas states, and hence that the v and w components are parallel to the interface. The case in which a vacuum is created will be discussed separately at the end of this sub-sub-section.

Let $\dot{m}_L$ and $\dot{m}_R$ be respectively the mass fluxes "into" the two outer waves (that is, through the shock or expansion fan). For the shock-wave case (which is identified by the condition $\frac{p_{inner}}{p_{L/R_{outer}}} \geq 1$), the mass flux is given by

$$\dot{m} = \rho_{outer} a_{outer} \sqrt{1 + \left(\frac{\gamma+1}{2\gamma}\right)\left(\frac{p_{inner}}{p_{outer}} - 1\right)}.$$

For the expansion-wave case (which is identified by the condition $\frac{p_{inner}}{p_{L/R_{outer}}} < 1$), the mass flux is given by

$$\dot{m} = \rho_{outer} a_{outer} \left(\frac{\gamma-1}{2\gamma}\right)\left(\frac{1 - p_{inner}/p_{outer}}{1 - (p_{inner}/p_{outer})^{(\gamma-1)/(2\gamma)}}\right).$$

Since the inner states may only be separated by contact or shear waves, it follows, as described in Section 2.2, that

$$p_{L_{inner}} = p_{inner} = p_{R_{inner}}, \quad \text{and}$$

$$u_{L_{inner}} = u_{inner} = u_{R_{inner}}.$$

The above two equations may be considered as the defining relations for p_{inner} and u_{inner} (in terms of $p_{L_{inner}}$, $p_{R_{inner}}$, $u_{L_{inner}}$, and $u_{R_{inner}}$). Note that it is not necessary for $v_{L_{inner}}$ and $v_{R_{inner}}$ to be equal, nor is it necessary for $w_{L_{inner}}$ and $w_{R_{inner}}$ to be equal. There is no constraint relating the two tangential components, so the shear wave coinciding with the contact wave may be arbitrary.

Projection of the conservation of momentum vector equation along a direction perpendicular to the Riemann interface gives

$$p_{inner} - p_{L_{outer}} = -\dot{m}_L \left(u_{inner} - u_{L_{outer}} \right), \quad \text{and}$$

$$p_{inner} - p_{R_{outer}} = \dot{m}_R \left(u_{inner} - u_{R_{outer}} \right).$$

Elimination of u_{inner} from these last two equations gives

$$p_{inner} = \frac{\dot{m}_L p_{R_{outer}} + \dot{m}_R p_{L_{outer}} - \dot{m}_L \dot{m}_R \left(u_{R_{outer}} - u_{L_{outer}} \right)}{\dot{m}_L + \dot{m}_R}, \tag{3.9}$$

while elimination of p_{inner} from these equations gives

$$u_{inner} = \frac{\dot{m}_L u_{L_{outer}} + \dot{m}_R u_{R_{outer}} - \left(p_{R_{outer}} - p_{L_{outer}} \right)}{\dot{m}_L + \dot{m}_R}. \tag{3.10}$$

The overall solution algorithm for the Riemann Problem is built around iterative solution of either of Equations 3.9 or 3.10 (with, for example, $\dot{m}_L$ and $\dot{m}_R$ expressed in terms of p_{inner} and p_{outer}, as given above), until convergence to a pre-specified tolerance. In order to guarantee convergence and to minimize the operation count, appropriate lower and upper bounds for p_{inner} or u_{inner} must be chosen. The solution procedure used in this work is based on iteration of the pressure equation (Equation 3.9), using a Newton iteration algorithm. For the pressure equation, a convenient lower bound is clearly provided by

$$p_{inner} = \rho_{outer} a_{outer},$$

while an appropriate upper bound is provided by

$$p_{inner} = \left(\frac{\left(\frac{\gamma-1}{2}\right)(u_L - u_R) + (a_L + a_R)}{a_L / p_{L_{outer}}^{\left(\frac{\gamma-1}{2\gamma}\right)} + a_R / p_{R_{outer}}^{\left(\frac{\gamma-1}{2\gamma}\right)}} \right)^{\left(\frac{2\gamma}{\gamma-1}\right)}.$$

The particular upper bound given above is obtained as the solution to the intersection of the two isentropes passing through the left and right states. As pointed out in [381], this upper bound may be used because the isentropes are always steeper than the Hugoniot curves.

Once p_{inner} has been obtained, $\dot{m}_L$ and $\dot{m}_R$ can be computed (indeed, they are obtained as part of the solution procedure for p_{inner}) and u_{inner} can then be obtained using Equation 3.10. The densities $\rho_{L_{inner}}$ and $\rho_{R_{inner}}$ may then be determined once it becomes known whether the inner state is separated from the corresponding outer one by a shock wave or an expansion fan. Across the shock (which is identified, as above, by $\frac{p_{inner}}{p_{L/R}} \geq 1$), the "inner" density is given by

$$\rho_{inner} = \rho_{outer} \left(\frac{1 + \frac{\gamma+1}{\gamma-1}\left(\frac{p_{inner}}{p_{outer}}\right)}{\frac{\gamma+1}{\gamma-1} + \frac{p_{inner}}{p_{outer}}} \right).$$

Across any expansion present (which is identified, as above, by $\frac{p_{inner}}{p_{L/R}} < 1$), the "inner" density is given by

$$\dot{m} = \rho_{outer} \left(\frac{p_{inner}}{p_{outer}} \right)^{1/\gamma},$$

and the complete solution within the expansion fan can be obtained using the fact that u and a vary linearly through its thickness, leaving ρ, and p to be determined from isentropic flow relations; namely:

$$\frac{\rho_2}{\rho_1} = \left(\frac{c_2}{c_1}\right)^{(2/(\gamma-1))},$$

and

$$\frac{p_2}{p_1} = \left(\frac{c_2}{c_1}\right)^{(2\gamma/(\gamma-1))}.$$

Any differences in the tangential velocity components between the two states remain unaffected and are convected unchanged through any shocks or expansions that are present, introducing a jump in the tangential components only at the contact discontinuity. Thus, the solution for the tangential velocity components is given simply by

$$v_{L_{inner}} = v_{L_{outer}}; \qquad v_{R_{inner}} = v_{R_{outer}},$$

and

$$w_{L_{inner}} = w_{L_{outer}}; \qquad w_{R_{inner}} = w_{R_{outer}}.$$

For two- and one-dimensional specializations of the Exact Riemann Solver, the only simplification introduced is that the corresponding tangential velocity component is eliminated from the solution.

Since the Exact Riemann solver gives the state at any plane in the $(\vec{x}, t)$ space, including the original interface between the two states, the flux vector, $\vec{H}_\perp$ can directly be computed from the solution to $\vec{P}$ on the interface, that is, through

$$\vec{H}_\perp = \vec{H}_\perp\left(\vec{P}\right).$$

Although the solution algorithm for an Exact Riemann solver may seem expensive, in practice, it is found that rarely will more than 5 or 6 iterations be required for convergence of p_{inner} by about 6 orders of magnitude in regions of smooth flow. The situation is only slightly worse near discontinuities. For cases where the component of velocity normal to the interface for both the left and right states is supersonic and of the same sign, the flux can be computed directly from the upwind-state flux, without the need to solve the Riemann Problem. A discussion of the relative performance of several Exact Riemann solvers, and of the optimizations that can be done to eliminate and reduce the computational effort based on the relation between the input states is given in [139].

The iteration procedure for an Exact Riemann solver must also include checks to ensure that the solution remains "above" the vacuum state for all possible inputs. The vacuum case is handled in this work by combining the two contacts that adjoin the vacuum into one, thereby eliminating the vacuum state from the computation altogether, and eliminating the need to handle the possibility of four waves in the solution. This action is necessary since, except perhaps for some specialized applications, the additional computational and algorithmic overhead required to enable proper handling of a state of complete vacuum cannot reasonably be justified in terms of the gains.

The solution quality and the computational cost for the Exact Riemann solver are compared with those of other solution techniques below.

3.2.1.4 Flux-Difference Splitting

In flux-difference splitting (FDS) , the approach taken is to consider an approximate linearized problem that eliminates the need for the iteration procedure described above for the solution of the Exact Riemann problem. This linearization was originally motivated by the need to reduce the computational effort required to solve the Exact Riemann Problem and the recognition that the additional accuracy obtained with such a precise solution was largely unnecessary for a scheme that was originally only first-order-accurate in space [308].

In the Osher FDS scheme, [270], the iterative procedure required to solve the Riemann problem exactly was replaced by an approximate solution in which the shock wave is replaced by an “overturned rarefaction”, resulting in explicit relations for the intermediate state variables separated by each wave. The Osher scheme intrinsically satisfies the entropy condition, and resolves steady-state shock waves exactly. The application of this scheme to multi-dimensional computations is described in detail in [67].

In the Approximate Riemann solver of Roe [308], the linearization is based on the definition of a unique average state at the interface that enables the term $\vec{\mathcal{H}}(\overline{\vec{U}_L} - \overline{\vec{U}_R})$ in Equation 3.8 to be expressed in the form

$$\vec{\mathcal{H}}(\overline{\vec{U}_L} - \overline{\vec{U}_R}) = \left|\hat{D}\right| \left(\overline{\vec{U}_L} - \overline{\vec{U}_R}\right). \tag{3.11}$$

The Flux Function in Roe's scheme can be expressed in the form

$$\vec{H}_\perp\left(\overline{\vec{U}_L}, \overline{\vec{U}_R}\right) = \frac{1}{2}\left(\vec{H}_\perp(\overline{\vec{U}_L}) + \vec{H}_\perp(\overline{\vec{U}_R})\right) - \frac{1}{2}\sum_{k=1}^{5}\left|\hat{\lambda}_k\right| \vec{R}_k \Delta V_k,$$

where the flux vectors $\vec{H}_\perp(\overline{\vec{U}_L})$ and $\vec{H}_\perp(\overline{\vec{U}_R})$ are as defined above, where $\hat{\lambda}_k$ is the k^{th} entry of the vector of eigenvalues, $\vec{\lambda}$, of the linearized flux Jacobian, which is given by

$$\vec{\lambda} = (\hat{u}_\perp - \hat{c}, \hat{u}_\perp, \hat{u}_\perp, \hat{u}_\perp, \hat{u}_\perp + \hat{c})^T,$$

where each vector $\vec{R}_k$ is taken as a column from the tensor $\vec{\vec{R}}$ of the right eigenvectors of the linearized flux Jacobian, which is given by

$$\vec{\vec{R}} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ \hat{u} - \hat{c}n_x & \hat{u} & 0 & \rho(n_y^2 + n_z^2) & \hat{u} + \hat{c}n_x \\ \hat{v} - \hat{c}n_y & \hat{v} & -\rho n_z & -\rho n_x n_y & \hat{v} + \hat{c}n_y \\ \hat{w} - \hat{c}n_z & \hat{w} & -\rho n_y & -\rho n_x n_z & \hat{w} + \hat{c}n_z \\ H - \hat{c}\hat{u}_\perp & \frac{1}{2}\hat{q}^2 & \rho(wn_y - vn_z) & \rho(u - u_\perp n_x) & H + \hat{c}\hat{u}_\perp \end{pmatrix},$$

and where the scalars ΔV_k are the elements of the wave-strength vector, which is given by

$$\Delta\vec{V} = \begin{pmatrix} \frac{\delta\hat{p}}{2\hat{c}^2} - \frac{\hat{\rho}}{2\hat{c}}\delta\hat{u}_\perp \\ \delta\rho - \frac{\delta p}{c^2} \\ \delta\hat{u}_{||_1} \\ \delta\hat{u}_{||_2} \\ \frac{\delta\hat{p}}{2\hat{c}^2} + \frac{\hat{\rho}}{2\hat{c}}\delta\hat{u}_\perp \end{pmatrix}.$$

and which is obtained by projecting the difference $\left(\overline{\vec{U}_L} - \overline{\vec{U}_R}\right)$ onto the right eigenvectors, $\vec{R}_k$.

In the above expressions for λ_k, $\vec{R}_k$, and ΔV_k, the properties which are given the "hat" superscript are averages which must be evaluated in accordance specific formulae. For $\hat{\rho}$, $\hat{u}$, $\hat{v}$, $\hat{w}$, and $\hat{H}$, the formulae are respectively as follows:

$$\hat{\rho} = \sqrt{\rho_L \rho_R}, \tag{3.12}$$

$$\hat{u} = \frac{u_R\sqrt{\rho_R} + u_L\sqrt{\rho_L}}{\sqrt{\rho_R} + \sqrt{\rho_L}}, \tag{3.13}$$

$$\hat{v} = \frac{v_R\sqrt{\rho_R} + v_L\sqrt{\rho_L}}{\sqrt{\rho_R} + \sqrt{\rho_L}}, \tag{3.14}$$

$$\hat{w} = \frac{w_R\sqrt{\rho_R} + w_L\sqrt{\rho_L}}{\sqrt{\rho_R} + \sqrt{\rho_L}}, \tag{3.15}$$

$$\hat{H} = \frac{H_R\sqrt{\rho_R} + H_L\sqrt{\rho_L}}{\sqrt{\rho_R} + \sqrt{\rho_L}}. \tag{3.16}$$

The "hatted" averages of each of the thermodynamic or flow properties that are not among the five listed above *may not* may be obtained by a similar averaging process, but must instead be computed directly (as dependent variables) from the above five quantities. For example, $\hat{c}$ is obtained through

$$\hat{c} = \sqrt{(\gamma - 1)\left[\hat{H} - \frac{1}{2}\hat{q}^2\right]},$$

where the perpendicular and parallel components of the velocity vector relative to the interface are also obtained by decomposing the new "hatted" velocity vector evaluated as given above. The important feature of the averaging procedure just described is that it is the unique one that ensures satisfaction of the "Property U", which is discussed below.

Each of the waves expressed in ΔV_k and $\vec{R}_k$ has a physical interpretation that agrees closely with the corresponding eigenvector of the Characteristic Quasi-Linear Form derived in Section 2.2. For example, the first wave introduces a perturbation of the form $\frac{\delta\hat{p}}{2\hat{c}^2} - \frac{\hat{\rho}}{2\hat{c}}\delta\hat{u}_\perp$ and is therefore an acoustic wave that propagates at the speed of acoustic disturbances. The second wave introduces a perturbation of the form $\delta\rho - \frac{\delta p}{c^2}$, and is an entropy wave that propagates with the convection velocity. The third and fourth waves introduce perturbations of the form $\delta\hat{u}_{\|_1}$ and $\delta\hat{u}_{\|_2}$, respectively, and are shear waves in the two Cartesian directions perpendicular to the interface normal. The fifth wave introduces a perturbation of the form $\frac{\delta\hat{p}}{2\hat{c}^2} + \frac{\hat{\rho}}{2\hat{c}}\delta\hat{u}_\perp$, and is also an acoustic disturbance that also propagates at the speed of acoustic waves. The implication of this correspondence is that the Roe linearization retains the fundamental nonlinear, discontinuity-supporting, and hyperbolic properties of the wave system of the full Euler Equations. This is largely achieved because the linearization and the solution procedure can be interpreted to derive from the Theory of Characteristics (that is, the Characteristic Linearization of The Euler System). This property is also shared by most other Flux-Difference schemes in common use.

The matrix $\hat{D}$ in Equation 3.11 can easily be verified to satisfy the following properties:

1. It has a complete set of eigenvectors with real eigenvalues;
2. $\forall(\overline{\vec{U}_L}, \overline{\vec{U}_R}), \qquad \overline{\vec{U}_L} = \vec{U} = \overline{\vec{U}_R} \quad \longrightarrow \quad \hat{D}\left(\overline{\vec{U}_L}, \overline{\vec{U}_R}\right) = \hat{D}(\vec{U})$; and
3. $\vec{H}_\perp(\overline{\vec{U}_L}) - \vec{H}_\perp(\overline{\vec{U}_R}) = \hat{D}\left(\overline{\vec{U}_L}, \overline{\vec{U}_R}\right)\left(\overline{\vec{U}_L} - \overline{\vec{U}_R}\right)$.

The first property is a natural outcome of the construction principles of Upwind-Differencing schemes. The second property can be interpreted as a consistency requirement, and one that leads explicitly to the preservation of the free-stream state. In order to understand the consequences of the third property, consider any steady state solution for which the left and right states, $\vec{U}_L$ and $\vec{U}_R$ respectively straddle any of the three discontinuities possible in The Euler System. Since the difference between the left and right states must coincide with one of the eigenvectors of the Roe Matrix, then

$$\vec{H}_\perp\left(\vec{U}_L\right) - \vec{H}_\perp\left(\vec{U}_R\right) = \lambda\left(\vec{U}_L - \vec{U}_R\right),$$

where λ is the unique eigenvalue of that eigenvector. As derived in Section 2.2, this condition is precisely the so-called Rankine-Hugoniot relation for a discontinuity separating the two states $\vec{U}_L$ and $\vec{U}_R$ and propagating with speed λ. Since the discontinuity is stationary, that is, since $\lambda = 0$, the flux reduces to

$$\vec{H}_\perp\left(\overline{\vec{U}_L}, \overline{\vec{U}_R}\right) = \frac{1}{2}\left(\vec{H}_\perp(\overline{\vec{U}_L}) + \vec{H}_\perp(\overline{\vec{U}_R})\right) = \vec{H}_\perp(\overline{\vec{U}_R}) = \vec{H}_\perp(\overline{\vec{U}_L}),$$

implying that any steady state discontinuity will remain unchanged and also that it can be resolved in one interior state. If the discontinuity coincides with the interface between the two states $\vec{U}_L$ and $\vec{U}_R$, then it will be resolved in a layer of infinitesimal thickness in principle. As proved in [317] the Roe averages given above are the unique ones that will satisfy this property.

The three properties given above have collectively been called "Property U" [1].

As evident from the formulation described above, the Roe linearization treats any expansion fans present in the solution as having infinitesimal thickness; it does not attempt to resolve their thickness or the variation of properties across them.

Since the eigenvalues of the acoustic waves, namely $\lambda = \hat{u} - \hat{c}$ and $\lambda = \hat{u} + \hat{c}$ vanish at sonic points, destroying any dissipation associated with that wave, and since the Roe Linearization does not incorporate the Second Law of Thermodynamics in its formulation, the Roe flux function often exhibits a tendency to introduce entropy-condition-violating numerical solutions. In extreme cases, this may manifest itself in the appearance of expansion shocks, but more often, it manifests itself in the form of reduced convergence rates for steady-state computations. In order to suppress this tendency, a so-called "entropy-fix" may be introduced into the flux function simply by preventing the vanishing of the dissipation with the vanishing of the acoustic eigenvalues. This can be accomplished most readily by modifying the acoustic eigenvalues to "smooth" their variation near the origin [154]. In particular, the two acoustic wave-speeds, λ_k, for $(k = 1, 5)$ are replaced by smoothed values, λ_k^*, given by

$$\left|\hat{\lambda}_k^*\right| = \begin{cases} \left|\hat{\lambda}_k\right| & \left|\hat{\lambda}_k\right| \geq \frac{1}{2}\delta\lambda_k \\ \frac{\left(\hat{\lambda}_k\right)^2}{\delta\lambda_k} + \frac{1}{4}\delta\lambda_k & \left|\hat{\lambda}_k\right| \leq \frac{1}{2}\delta\lambda_k \end{cases}$$

where

$$\delta\lambda_k = max\left(4\Delta\lambda_k, 0\right), \tag{3.17}$$

[1]The "U" symbol was chosen because the Riemann Solver is **Uniformly** valid for both "strong" and "weak" waves [316].

$$\Delta\lambda_k = \lambda_{k,R} - \lambda_{k,L} \ . \tag{3.18}$$

This and other methods of introducing an entropy-fix into Roe's Scheme are discussed in more detail in [314, 422]. More generally, in [268] the analytic criteria that must be met by a scheme to satisfy an entropy condition have been derived, and schemes that do satisfy this requirement were called **E-Schemes**.

From the theoretical point of view, perhaps the most serious deficiency of the Roe Linearization (and any other linearization) is that it is invalid for some combinations of input states. In [115], a **Positively-Conservative Scheme** is defined as one that is guaranteed to create only Physically-Admissible states (that is, states for which $\rho \geq 0$; and $e \geq 0$) given Physically-Admissible inputs, and the conditions under which *any* linearization of the Riemann Problem (including Roe's) will generate negative energy or density from physically-admissible inputs are derived. The work also proves that in some situations, the failure of the Roe linearization can be traced back to its underestimation of the correct wave-speeds in the Exact Riemann Problem. Although these results are obtained for first-order-accurate, one-dimensional schemes, they are expected to be equally valid for multi-dimensional and higher-order schemes.

Other shortcomings of the Roe linearization as well as most other commonly-used flux functions are cataloged and discussed in [293]. Note that the restriction of Physical Admissibility applies only to the initial and final states during any flux computation; there are no restriction on intermediate states evaluated as part of a computational determination scheme. Indeed, the Osher Scheme, for example, employs non-physical intermediate states as part of the solution algorithm.

The two-dimensional specialization of the Roe Flux-Difference Splitting formulation can be derived in a similar manner to the three-dimensional formulation. As expected, it can also be obtained by elimination of the eigenvectors and eigenvalues that are associated with the spatial multiplicity. The Roe averages are computed using the same formulae, except that the velocity component in the eliminated dimension is removed wherever it appears. The matrix of right eigenvectors is then given by

$$\vec{\vec{R}} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ \hat{u} - \hat{c}n_x & \hat{u} & -n_y & \hat{u} + \hat{c}n_x \\ \hat{v} - \hat{c}n_y & \hat{v} & n_x & \hat{v} + \hat{c}n_y \\ H - \hat{c}\hat{u}_\perp & \frac{\hat{q}^2}{2} & \hat{u}_{||} & H + \hat{c}\hat{u}_\perp \end{pmatrix},$$

the wave-strength vector is then given by

$$\Delta\vec{V} = \begin{pmatrix} \frac{\delta\hat{p}}{2\hat{c}^2} - \frac{\hat{\rho}}{2\hat{c}}\delta\hat{u}_\perp \\ \delta\rho - \frac{\delta p}{c^2} \\ \hat{\rho}\delta\hat{u}_{||} \\ \frac{\delta\hat{p}}{2\hat{c}^2} + \frac{\hat{\rho}}{2\hat{c}}\delta\hat{u}_\perp \end{pmatrix},$$

and the vector of eigenvalues is then given by

$$\vec{\lambda} = (\hat{u}_\perp - \hat{c}, \hat{u}_\perp, \hat{u}_\perp, \hat{u}_\perp + \hat{c})^T.$$

The one-dimensional specialization can be derived similarly to the two-dimensional one; it can also be obtained by elimination of the appropriate rows and columns in the 2-D specialization.

3.2.1.5 The HLL Flux Function

The HLL Flux Function (that is, the Harten-Lax-Van-Leer flux function) [159], and derivatives from it, such as Einfeldt's variant, the **HLLE Flux Function** (that is the Harten-Lax-Leer-Einfeldt flux function) [114], and the **HLLEM Scheme** [115], simplify the linearization of the Riemann Problem by assuming that the solution can be approximated by two traveling waves (and hence by only three uniform states). As would be expected, this scheme strongly diffuses shear and contact waves since it does not attempt to resolve the internal structure between the extreme left and right waves, which are assumed to be infinitesimal in thickness.

In this work, only the HLLE scheme is used from among this family. The HLLE flux function is more dissipative but more robust that the Roe Flux Function and was used in this work in regions of strong shocks or expansions, or in situations where the Roe or Exact Riemann solvers were found to be prone to instability or failure, such as odd-even decoupling during the propagation of strong shocks aligned with the grid lines. The particular formulation used in this work is given by

$$\vec{H}_\perp(\overline{\vec{U}_L}, \overline{\vec{U}_R}) = \frac{\lambda^+ \vec{H}_\perp(\overline{\vec{U}_L}) - \lambda^- \vec{H}_\perp(\overline{\vec{U}_R})}{\lambda^+ - \lambda^-} + \frac{\lambda^+ \lambda^-}{\lambda^+ - \lambda^-} \left(\overline{\vec{U}_R} - \overline{\vec{U}_L} \right), \tag{3.19}$$

where λ^+ and λ^- are the speeds of the fastest forward- and backward-moving waves along the interface normal, namely:

$$\lambda^+ = max\,(\lambda_{max}, 0)\,,$$

and

$$\lambda^- = min\,(\lambda_{min}, 0)\,,$$

and where all other symbols are as defined above.

Considerable choice is allowed [159] in selecting the values for λ_{max} and λ_{min}. However, if these values are respectively chosen to be the maximum and minimum eigenvalues of the Roe Linearization corresponding to the same input states ($\overline{\vec{U}_L}$ and $\overline{\vec{U}_L}$), to wit:

$$\lambda_{max} = max\,(\hat{c}, \hat{u}_\perp + \hat{c})\,, \tag{3.20}$$

and

$$\lambda_{min} = min\,(\hat{c}, \hat{u}_\perp - \hat{c})\,, \tag{3.21}$$

it can be shown [115] that the resulting scheme will be positively conservative.

The relation between the dissipation addition in this scheme relative to the Exact Riemann solution and the magnitude of the eigenvalues is analyzed in [114, 115], and it is shown there that increasing the wave-speeds generally increases the dissipation. Although the proofs are for first-order schemes, the results appear to carry over to multi-dimensional computations. In practice, it is not necessary to increase the dissipation to the level given in Equations 3.20 and 3.21. For the flows studied in this work, experience indicates that it is best to scale the spread between the wave-speeds with the strength of the discontinuity, up to the maximum allowed in Equations 3.20 and 3.21, that is, to use a “strength-adaptive” form of the HLLE. Otherwise, excessive damping forced by the presence of a strong shock may destroy the resolution of important contacts and shears, as often observed with shock-wedge interactions, such as those presented in Chapter VIII. More discussions on appropriate choices for the parameters of HLLE flux functions are given in Chapter VIII.

3.2.1.6 Other Upwind Methods

In addition to the methods used in this work and described in detail above or in an appendix, a wide variety of upwind methods exist that have been used either only for specialized applications, or that have not come into widespread use for one reason or another. A brief review is given in Appendix F of three of the most popular of these methods.

3.2.2 Higher-Order Accuracy

As explained above, the discretization methodology adopted in this work purposely separates the spatial and temporal discretizations, in order to increase the flexibility in the development of the computational scheme and to eliminate the dependence of steady-state solutions on the time-step. This separation is naturally again maintained in the higher-order spatial discretization.

3.2.2.1 Formulation Principles

The general principle underlying the development of all higher-order spatial discretizations from lower-order ones is based on increasing the spatial order of accuracy of the discrete flux integral. Regardless of the form in which it is derived, the flux on any cell face or between any two discrete states, say, $\vec{U}_L$ and $\vec{U}_R$, may be expressed in the generic form

$$\vec{H}_\perp = \sum_{p=1}^{nPoints} \omega_p \vec{H}_\perp(\vec{r}_p), \tag{3.22}$$

where p denotes a quadrature point in space (typically, in the cell face on which the flux is being evaluated), $nPoints$ is the total number of quadrature points, ω_p is the weight assigned to point p, and the term $\vec{H}_\perp(\vec{r}_p)$ refers to the point-value of the flux associated with quadrature point p. Thus, the spatial order of accuracy of the flux integral on a fixed subregion set may be increased by increasing the spatial order of accuracy of the flux computation at the quadrature points and by increasing the order of accuracy of the quadrature computation.

The order of accuracy of the quadrature computation may be increased by increasing the number of quadrature points and by evaluating the locations or the weights assigned to these points more accurately. The most accurate quadrature formula, the Gauss Quadrature Rule, yields an order of accuracy of $2n$ with n quadrature points. This is the quadrature rule used in this work. Since only a second-order-accurate scheme is sought and used in this work, only one quadrature point is required along each edge to compute the flux integral, and this point must be located at the mid-point of the edge. Clearly, computational cells in the vicinity of a moving boundary must have their full geometric definition, including their quadrature points, re-evaluated or re-computed after every motion step. The manner in which this is done is explained in Chapters V and VI. If a third-order-accurate scheme were to be used in this work (instead of a second-order-accurate scheme), at least two quadrature points along each edge would be required (instead of the single quadrature point used currently).

The spatial order of accuracy attained in computing the point fluxes, $\vec{H}_\perp(\vec{r}_p)$, may be increased either directly, by adopting a higher-order flux discretization, or, indirectly, by increasing the spatial order of accuracy of the discretized functional representation, which is typically a polynomial representation, of the discrete unknowns in computational cells. A second-order flux discretization may be expressed in the form

$$\vec{H}_\perp(\vec{r},t) = \vec{H}_\perp(\vec{r}_c,t) + \nabla\vec{H}_\perp(\vec{r}_c,t)\cdot(\vec{r}-\vec{r}_c), \tag{3.23}$$

where $\vec{r}_c$ is the location of the point about which the higher-order expansion is computed, and $\vec{r}$ is the location of the point at which the higher-order evaluation of $\vec{H}_\perp$ is required. Similarly, a second-order-accurate functional representation of a discrete scalar unknown ψ in a cell may be expressed in the form

$$\psi(\vec{r},t) = \psi(\vec{r}_c,t) + \nabla\psi(\vec{r}_c,t)\cdot(\vec{r}-\vec{r}_c), \tag{3.24}$$

where the meanings of the symbols $\vec{r}_c$ and $\vec{r}$ are as given above.

Within the framework of an upwind-based, Finite-Volume formulation following a Godunov scheme, increasing the order of accuracy of the functional representation of the discrete variables within each cell is reflected in the flux computation only in the evaluation of the Left and Right states, $\vec{U}_L$ and $\vec{U}_R$, for the Riemann Problem that is solved at each quadrature point on the faces of the computational cells. The numerical flux functions or Riemann solvers described above in this chapter apply quite generally, regardless of the manner in which the Left and Right state inputs are determined, and therefore do not generally require any modification for higher-order schemes if this (indirect) approach to higher-order accuracy is taken. The indirect approach to higher-order accuracy just described (as opposed to the direct approach of using higher-order-accurate flux computations) is currently the more popular one, and is the one followed in this work.

The increase in the order of accuracy of the representation function of the discrete variable in a computational cell is achieved by increasing the degree of the polynomial function onto which the solution average in each computational cell is projected. Typically, this projection step is performed separately, usually based only on the data in the target cell and its surrounding neighbors, after the average value in the cell has been updated. In that case, the process is called "reconstruction". The main constraint in a conservative discretization is that the reconstruction may not change the average value of any conserved variable. The reconstruction approach to increasing the order of accuracy as an improvement on Godunov's original scheme (which was first-order accurate) was pioneered for structured-grids in [378], where it was called the MUSCL approach [2], and later in different forms in [84] and [157]. More recently, the reconstruction approach as a basis for higher-order accuracy was extended in a general manner to unstructured grids in [29], and then in [28, 27] and several related works.

A uniform or constant functional representation within each computational cell in a uniform grid implies a first-order-accurate spatial representation since a constant function over the solution domain can be represented exactly, while the truncation

[2]MUSCL is an acronym for Monotone Upstream-Centered Schemes for Conservation Laws.

error will be linear in the local cell dimension for arbitrary smooth functions. Similarly, a linear or bilinear functional representation within each computational cell on a uniform grid implies a second-order-accurate representation since a linear or bilinear function over the solution domain can be represented exactly (hence the name “1-exact” [27]), while the truncation error will be quadratic in the local cell dimension for arbitrary smooth functions. More generally, a reconstruction that uses polynomial projection functions of degree k is called a k-exact reconstruction [27]. For a uniform grid, it can readily be shown that a k-exact reconstruction leads to a scheme that has a spatial order of accuracy of $k + 1$.

The immediate difficulty with increasing the polynomial degree of the projection function beyond 0 is the creation of spurious oscillations in the solution. This is true even for scalar equations, including linear equations, as well as linear systems. Indeed, Godunov’s Theorem concludes that no linear difference scheme with constant coefficients could be monotone if it has a spatial order-of-accuracy higher than 1. This difficulty is circumvented by making the computational scheme nonlinear [378]. More specifically, the most common way of introducing this nonlinearity with the reconstruction approach is based on **limiting** the slope of whatever variables are reconstructed in individual cells to prevent the reconstruction values from overshooting or undershooting the “support” values, and hence to prevent spurious oscillations in the solution. With limiting, the general formulation for the generic variable ψ with at any location $\vec{r}$ within a computational cell with a second-order scheme becomes

$$\psi(\vec{r}, t) = \psi(\vec{r}_c, t) + \phi \; \nabla\psi(\vec{r}_c, t) \cdot (\vec{r} - \vec{r}_c), \tag{3.25}$$

where $\psi(\vec{r}_c, t)$ is the centroidal value of ψ (or, equivalently, the volume-averaged value of ψ over the entire cell), where $\vec{r}_c$ is the location of the cell controid, where $\nabla\psi(\vec{r}_c, t)$

is the unlimited gradient of ψ at $\vec{r}_c$, and where ϕ is the limiter value associated with that cell. Note that the values of the gradient and limiter are assumed constant over the entire cell in the above expression (although they need not be always so), and that the value of ϕ must satisfy the condition

$$0 \leq \phi \leq 1.$$

The use of limiters whose value varies from cell to cell depending on the local conditions is the basis for construction of so-called High-Resolution schemes [170]. Typically, these schemes allow the value of the limiter to revert as close to 1 as possible in smooth regions, while near discontinuities and extrema, the value of the limiter becomes much smaller, and even zero. The need for non-uniform limiting arises because the reconstruction overshoots (and hence the associated non-physical oscillations) in the unlimited solution typically both have greater amplitudes near high gradients of the solution, and especially around discontinuities. The introduction of variable limiting therefore allows second-order accuracy to be achieved in smooth regions, while forcing a return to a monotone first-order scheme near discontinuities and extrema.

The remainder of this section describes, in more detail than given in the above introduction, two of the three main ingredients that constitute a High-Resolution scheme in general, and the one used in this work in particular: (i) the reconstruction; and, (ii) the limiting. The third ingredient; namely, the evolution operator, is described in Section 3.3. Of particular concern is maintaining the monotonicity and the convergence to weak solutions that is exhibited by first-order schemes as the order of accuracy is increased to 2.

3.2.2.2 Reconstruction Procedures

The meaning of "reconstruction" and the motives for reconstructing the solution variables in computational cells were discussed in the preceding sub-sub-section. This sub-sub-section mostly describes the specific procedures used in this work to perform the reconstruction.

There are two popular approaches to second-order or 1-exact reconstruction on unstructured grids with arbitrary cell geometries: (i) the Green-Gauss reconstruction [29, 27]; and (ii) the Least-Squares reconstruction [26]. Both have been implemented and tested in this work, as described below.

The Green-Gauss approach is founded on the Gauss-Ostrogradski Theorem, which for a scalar ψ can be expressed in the equation

$$\int_{\Omega} \nabla\psi = \int_{\partial\Omega} \psi\vec{n}, \tag{3.26}$$

where all symbols are as defined above. Defining a volume-averaged mean value for $\nabla\psi$ over Ω by

$$\overline{\nabla\psi} = \frac{\int_{\Omega} \nabla\psi}{\int_{\Omega}},$$

Equation 3.26 may be re-written in the form

$$\overline{\nabla\psi} = \frac{1}{V}\int_{\partial\Omega} \psi\vec{n}, \tag{3.27}$$

where $V = \int_{\Omega} 1$ is the volume of the region Ω. Since the gradient in each computational cell is assumed constant, the expression given in Equation 3.27 is exact, provided the surface integral encloses the cell whose gradient it is desired to estimate. The discrete form of Equation 3.27 is given by

$$\overline{\nabla\psi} = \frac{1}{V}\sum_{f=1}^{nFaces} \overline{\psi_f}\vec{S}_f, \tag{3.28}$$

where $\overline{\psi_f}$ is an appropriate average on face f, and $nFaces$ is the number of faces of the polyhedral computational cell. In the three-dimensional case, $\overline{\psi_f}$ is best determined from the Gauss points of the face, or by a suitably-weighted linear combination of the values at the vertices of the face, or any other suitable quadrature formula. In the two-dimensional situation, the value of $\overline{\psi_f}$ may be conveniently determined through the second-order-accurate formula

$$\overline{\psi_f} = \frac{\psi_{v1} + \psi_{v2}}{2},$$

where ψ_{v1} and ψ_{v2} are the values of the unknown at the vertices $v1$ and $v2$ of the edge. For a third-order-accurate scheme, at least a third point is required to evaluate the average to give a sufficiently-accurate discrete contour integral. No matter how accurate it is, the approximation used to evaluate the discrete face average $\overline{\psi_f}$ will clearly compromise the exactness of the corresponding continuous equation, namely, Equation 3.27.

It is also possible to determine an approximation to $\overline{\nabla\psi}$ by performing a contour integration along a contour that passes not through the vertices of the target cell, but through the centroids of a set of selected support cells surrounding the target cell. This approach was also implemented and tested in this work.

The second reconstruction technique implemented and used in this work is the Least-Squares reconstruction, with a formulation closely following that described in [26, 27]. In this formulation, $\nabla\psi$ is computed as the solution to the equation

$$\begin{pmatrix} w_1\triangle x_{n1} & w_1\triangle y_{n1} & w_1\triangle z_{n1} \\ \vdots & \vdots & \vdots \\ w_N\triangle x_{nN} & w_N\triangle y_{nN} & w_N\triangle z_{nN} \end{pmatrix} \begin{pmatrix} \frac{\partial\psi}{\partial x} \\ \frac{\partial\psi}{\partial y} \\ \frac{\partial\psi}{\partial z} \end{pmatrix} = \begin{pmatrix} w_1\triangle\psi_1 \\ \vdots \\ w_N\triangle\psi_N \end{pmatrix} \tag{3.29}$$

where

$$\begin{aligned}
\triangle x_{ni} &= x_{ni} - x_c \\
\triangle y_{ni} &= y_{ni} - y_c \\
\triangle z_{ni} &= y_{ni} - z_c \\
\triangle \psi_{ni} &= \psi_{ni} - \psi_c,
\end{aligned} \tag{3.30}$$

where the subscript c denotes association with the target cell and the subscript ni denotes association with the i^{th} neighbor of cell c. For a problem with no moving boundaries, the elements of the weighted-distance matrix on the left-hand-side of Equation 3.29 may be computed just once, at the initialization stage of a calculation. For problems with moving boundaries, the elements of that matrix require updating only if the geometry of a computational cell or any of its neighbors changes during a calculation. The reduction of Equation 3.29 to the two-dimensional case is obvious.

The weights in Equation 3.29 may be chosen to be 1, or may be chosen to depend on the flow solution or the stencil geometry. In particular, it was found advantageous in this work to use the relation

$$w_i \propto \frac{A}{r},$$

where r is the distance between the centroid of the target cell and that of its i^{th} neighbor, and A is the area of that neighbor. The introduction of A into the evaluation appeared to reduce the effect of biasing of the gradient by the values in the several small cells that may be present on one side of a target cell near refinement boundaries. Other combinations, with a proportionality to only $\frac{1}{r}$ or to only A have also been popularly used, but were not examined in this work.

From the above, it can be seen that the Green-Gauss and the Least-Squares formulations for estimation of the gradient are similar in structure and properties, since both can be written as a weighted linear average of the values in the support cells. Indeed, the gradient evaluation produced by both methods is identical for any linear function. Although the Least-Squares approach is in general the more accurate one [2, 3], especially for stretched or skewed cells, both approaches result in formally second-order accurate spatial reconstruction, and several published works on tests with basic flows indicate minimal differences [101, 86] in the overall results obtained with the two techniques. From the practical point of view, the Least-Squares approach is easier to implement, especially for three-dimensional spaces, and this is largely because, unlike the the case for the Green-Gauss formulation, the Least-Squares formulation does not require the support cells to be ordered in any particular manner.

The stencil of neighbors used in the gradient computation of Equation 3.28, that is, following the Green-Gauss approach, should in principle include all the immediate neighbors of the target cell, that is, all the face-, edge-, and vertex-neighbors of that cell. This is because the surface or contour on which the discrete integration is performed must be closed, and must fully surround the cell for which the gradient is being evaluated. This implies that all the face-, edge-, and vertex-neighbors for the target cell must be used to construct a “super-cell” whose vertices are defined from the centroids of these support cells. In practice, however, this requirement is purposely relaxed, by using a contour integral which includes only the face-neighbors, or only the face- and edge-neighbors, especially when there are sufficient cells surrounding the target cell, sometimes even if the surface or contour on which the integration is performed does not completely enclose the target cell. Using a stencil with fewer members not only reduces the operation count for the gradient evaluation, but also usually increases the accuracy, even if the order of accuracy remains unchanged. Clearly, the above discussion is irrelevant if the contour is constructed from the vertices of the target cell rather than from the centroid of the neighboring support cells.

The stencil of neighbors used in the gradient computation of Equation 3.29, that is, following the Least-Squares approach, explicitly need not include all the immediate neighbors of the target cell. Indeed, the Least-Squares approach is most often implemented with a stencil that includes only the face neighbors.

In this work, however, none of the neighbor groups were excluded in the gradient evaluations, neither when using the Green-Gauss approach, nor when using the Least-Squares approach. The remain reason for doing so was to implement the most extensive form of the approaches, so that any experimentation with different, reduced, stencils can readily be done later.

Near boundaries, the support-cell stencil must be reduced as necessary, with the attendant reduction in accuracy, but without loss in 1-exactness. As mentioned above, in the case of the Green-Gauss formulation, the theoretical basis of the formulation is eroded if the surface or contour on which the integral is evaluated passes through the interior of the target cell. The support cells may be reduced to only a few in convex regions or near boundaries, but the surface or contour integral may not collapse or become degenerate. In the case of the Least-Squares formulation, the cell centroids used for the Least-Squares reconstruction may not become collinear or degenerate. Near boundaries, with the Green-Gauss formulation, the surface or contour integral may be forced to pass through the centroid of the target cell, causing the centroidal value of the target cell to contribute to the gradient evaluation of that cell.

In this work, whichever of the two reconstruction procedures described above is chosen for a computation, that procedure is applied independently to each of the primitive variables ρ, u, v, and p, resulting in an independent gradient vector for each of these primitive variables. Exactly the same procedure is used for all of the primitive variables. Thus, the generic scalar ϕ in the above equations can be taken to represent any of these four primitive variables. Such a unified treatment is in accordance with established common practice.

As explained above, in this work, the reconstruction is applied to the primitive variables, although it would also have been feasible to choose either the conserved or the characteristic variables for this purpose. The appeal of choosing the conserved variables is that the mean value in a cell is automatically preserved (although violation of this mean in the reconstruction would not be reflected in the solution in any manner). In the one-dimensional case, reconstructing the characteristic variables can be shown to be a theoretically more sound option. From a practical point of view, applying the reconstruction to one of the three sets of variables mentioned above instead of another appears to have little effect on the final result or the quality of the solution in multi-dimensional computations. Some additional guidance on the selection of the “category” of variables for reconstruction is given in [378].

Several reconstruction procedures based on other than the pure Least-Squares or Green-Gauss approaches have also appeared in the literature. Most of these techniques, however, are either equivalent to combinations of the Least-Squares and Green-Gauss approaches, or do not take into account the values in all the neighboring cells (by, for example, restricting the set of support cells to only the face-neighbors), or compute extrapolated values on a face by direct interpolation across the cells sharing that cell face, or are restricted to specific cell geometries, for example, as in [223].

3.2.2.3 Limiting Procedures

The first sub-sub-section of this sub-section explained the meaning of "limiting", and the need for it in higher-order schemes. More specifically, that sub-sub-section explained how reconstruction of the solution variables within the computational cells in a Finite-Volume method (by use of the gradient-estimate in those cells, for example) may create new local extrema in those variables, which in turn result in high-frequency oscillations or "wiggles" in the solution. These local extrema and the resulting oscillations are numerical (that is, non-physical) artifacts which not only pollute the local solution, but could also affect the global solution, especially for highly-nonlinear problems, such as those involving reacting flow. That sub-sub-section also explained that in order to counter these spurious numerical effects, the solution gradients in computational cells as originally computed by the reconstruction procedure are limited (or reduced) to levels that would largely eliminate oscillations in the solution. That sub-sub-section also mentioned that the notion of limiting the reconstruction gradients to eliminate spurious oscillations was originally developed in [377, 378], and called the MUSCL approach.

This sub-sub-section mostly presents the specific limiting procedures that are used in this work, and discusses their formulation principles and main properties.

While many limiters have been developed for the "structured" category of grids (which is defined in Chapter IV) [350], only two limiters seem to have attained widespread use for the "unstructured" category of grids (which is also defined in Chapter IV), especially grids that contain arbitrary polygonal cells, like the grids used in this work. These two limiters are the Barth Limiter, and the Venkatakrishnan Limiter. Both of these limiters were implemented and used in this work.

The Barth Limiter and the Venkatakrishnan Limiter both compute the value of the limiter in computational cell c_i, $\phi(c_i)$, using a formula of the form

$$\phi(c_i) = \min_j\{\tilde{\phi}(c_i, c_j)\} \tag{3.31}$$

where the index j traverses all the "support" cells used in the limiting procedure, and $\tilde{\phi}(c_i, c_j)$ is computed differently for the Barth and Venkatakrishnan limiters, as described below. In this work, as in most other implementations, the set of support cells used to compute the limiter function (that is, the set of cells over which the index j traverses in Equation 3.31) is identical to the set of support cells used to compute the gradient. As explained above, in this work, this set of cells was chosen to include all the face-, edge-, and vertex-neighbors of the target cell, c_i.

Since, as explained in the preceding sub-sub-section, the reconstruction in this work is performed on the primitive variables, the limiting must also be performed on (the slopes of) the primitive variables. Thus, while the reconstruction independently computes the slopes of each of the variables ρ, u, v, and p, the limiting independently computes the limiter value for each of those four slopes. Thus, throughout this sub-sub-section, the limiter variable $\phi(c_i)$ for cell c_i refers to the generic limiter variable for that cell, of which 4 specific instances are computed and used, one for each primitive variable. Just as for the reconstruction procedures, the limiting procedure used for all four of the primitive variables is identical, allowing for a completely uniform treatment.

The value of $\tilde{\phi}(c_i, c_j)$ for the Barth Limiter [29] is computed in accordance with the following formula:

$$\tilde{\phi}(c_i, c_j) = \begin{cases} \frac{\Delta u_{c_i}^{max}}{\Delta u_r} & : \quad \Delta u_r > \Delta u_{c_i}^{max} \\ \frac{\Delta u_{c_i}^{min}}{\Delta u_r} & : \quad \Delta u_r < \Delta u_{c_i}^{min} \\ 1 & : \quad \Delta u_{c_i}^{min} \leq \Delta u_r \leq \Delta u_{c_i}^{max} \end{cases} \tag{3.32}$$

where the subscripts c_i and c_j respectively denote association with computational cell c_i and its neighbor cell c_j, where u is the variable being limited, where the symbol Δ symbolizes the generic difference between the variable values associated with the two cells being considered, for example, $\Delta u = u_{c_j} - u_{c_i}$, where the superscripts max and min respectively refer to the largest and smallest differences in the value of the variable u between the target cell and all its other support cells, namely,

$$\Delta u_{c_i}^{max} = \max_j \{(u_{c_j} - u_{c_i})\}, \tag{3.33}$$

and

$$\Delta u_{c_i}^{min} = \min_j \{(u_{c_j} - u_{c_i})\}, \tag{3.34}$$

where u_r refers to the reconstructed value of the variable u in the cell c_i, and where Δu_r refers to the difference between the reconstructed value and the corresponding cell-centroidal value; namely,

$$\Delta u_r = \nabla u_{c_i} \cdot (\vec{r}_{f_{ij}} - \vec{r}_{c_i}),$$

where ∇u_{c_i} is the (unlimited) gradient of u in cell c_i, $\vec{r}_{f_{ij}}$ is the location of the point at which the reconstructed value is being sought (which is typically a point in a cell face shared by cells c_i and c_j), and $\vec{r}_{c_i}$ is the location of the centroid of cell c_i.

As evident from Equations 3.32 and 3.25, no reconstructed value with the Barth Limiter is allowed to exceed the maximum value in the support cells or to fall below the minimum value in the support cells. Therefore, the Barth Limiter enforces strict monotonicity in the reconstructed values (which are used in the flux evaluations, or any other solution-related evaluations). As would be expected, relative to the corresponding unlimited scheme, application of the Barth Limiter leads to more robust solutions, but also to slightly lower orders of spatial accuracy, even in smooth regions.

In practice, in the calculation of steady-state problems, the Barth Limiter strongly inhibits convergence, to the extent of preventing the residuals from dropping more than typically 2 or 3 orders of magnitude. This is largely because in regions of uniform or nearly-uniform flow, the highly-nonlinear form of the limiter, which is introduced through the "min" and "max" functions used in that limiter, cause it to trigger the local limiting on and off (or, equivalently, cause the local limiter value to vary seemingly randomly between 0 and 1) during the iteration cycle [389]. This triggering occurs even when there is little activity in the local solution, and even in response to arithmetic truncation errors [389]. This deficiency of the Barth Limiter does not seem to affect the global solution, and there is no known test case in which it leads to an incorrect overall solution. Since most of the computations performed in this work were for unsteady problems, this deficiency was largely inconsequential; rather, the strict enforcement of monotonicity in the Barth Limiter helped to avoid failures in the Approximate Riemann solvers for many of the test cases that involved strong shocks or low pressures or densities.

Despite the limited effects on the overall solution of the deficiency described in the preceding paragraph of the Barth Limiter, the effects of that deficiency on the convergence behavior remain highly undesirable. This was the major motivation for the development of the Venkatakrishnan Limiter [389], as described in more detail below.

The value of $\tilde{\phi}(c_i, c_j)$ for the Venkatakrishnan Limiter [389] is given by:

$$\tilde{\phi}(c_i, c_j) = \frac{1}{\Delta u_r}\left[\frac{((\Delta u_m)^2 + \epsilon^2)\Delta u_r + 2\Delta u_m(\Delta u_r)^2}{(\Delta u_m)^2 + 2(\Delta u_r)^2 + \Delta u_r \Delta u_m + \epsilon^2}\right] \tag{3.35}$$

where

$$\Delta u_m = \begin{cases} \Delta u_{c_i}^{max} & : \quad \Delta u_r \geq 0 \\ \Delta u_{c_i}^{min} & : \quad \Delta u_r < 0 \end{cases} \tag{3.36}$$

where $\Delta u_{c_i}^{max}$, $\Delta u_{c_i}^{min}$, and Δu_r are all as defined above, and where ϵ is a free parameter whose value is chosen to introduce a given level of "looseness" in the limiting.

One suitable choice for ϵ is given by the formula

$$\epsilon = \epsilon_v \left(\Delta u^{max} - \Delta u^{min} \right) \tag{3.37}$$

where ϵ_v is a "user-specified" parameter, typically chosen in the range $0.01 \leq \epsilon_v \leq 0.20$, and where

$$\Delta u^{max} = \max_{c_i} \{ \Delta u_{c_i}^{max} \}$$

and

$$\Delta u^{min} = \min_{c_i} \{ \Delta u_{c_i}^{min} \}$$

represent the global maximum and minimum differences in the variable values (within all the stencils used for limiting).

Another suitable choice for ϵ [389] is given by the formula

$$\epsilon^2 = K d^3,$$

where d is the local cell dimension, and K is a user-specified parameter, falling in the typical range $0.05 \leq K \leq 0.5$. In this formula, K serves as an explicit threshold that determines the overshoot level that is allowed before the limiter is invoked. Regarding the choice of d, it should be noted that even though this has been done in some implementations, using a global cell dimension instead of a local cell dimension (for d) is clearly not advisable.

As can be seen by examination of Equations 3.31 and 3.35, the Venkatakrishnan Limiter does not strictly enforce monotonicity in the reconstructed values. Instead, as mentioned above, the parameter ϵ introduces a "looseness" or "slack" in limiting the gradient. More specifically, setting ϵ to zero fully recovers the Barth Limiter, as explained in [389], while setting ϵ to a large number effectively eliminates any limiting of the slope, by causing $\tilde{\phi}(c_i, c_j)$ to be one, or nearly so, regardless of the solution variable values. It is also clear from the forms of Equations 3.32 and 3.35 that the Venkatakrishnan Limiter is more differentiable than the Barth Limiter, and hence is inherently less likely to produce "noisy" or seemingly random fluctuations in the limiter value.

Choosing an appropriate formulation for ϵ and an appropriate setting for ϵ_v or K is case-specific and typically requires some trial and error. In general, however, calculations with strong shocks or rapid accelerations of boundaries require lower values of ϵ_v or K, while calculations involving only smooth solutions are best performed with large values of ϵ_v or K.

With an appropriate choice for ϵ, the Venkatakrishnan Limiter is found to eliminate the effects of the noise-induced triggering of the Barth Limiter, especially in regions of uniform flow, and to eliminate the invocation of limiting in smooth non-uniform regions, leading to better convergence rates and slightly higher orders of accuracy for computations of steady-state problems. The higher accuracy order is attained even though the Venkatakrishnan Limiter is formally more diffusive that the Barth Limiter. On the other hand, since the monotonicity of the reconstruction is lost with the Venkatakrishnan Limiter, the overall robustness of calculations with that limiter is also reduced.

3.3 Temporal Discretization and Time-Integration: Alternatives and Specific Implementation

As explained in Section 3.1 above, and for the reasons given there, the spatial and temporal discretizations in this work are intentionally separated. The spatial discretization adopted in this work is described in Section 3.2, while the temporal discretization adopted in this work is described mainly in this section and (with a greater emphasis on the relevant moving-boundary considerations) in Section 3.4.

3.3.1 Choice of Temporal Discretization and Time-Integration Scheme

As explained in Section 3.2 above, with the separation of the spatial and temporal discretizations adopted in this work, an appropriate starting point for the temporal discretization is the semi-discrete Equation 3.7, or equivalently, the derivative in time of that equation, namely:

$$\frac{\partial \overline{\vec{U}} V}{\partial t} = \vec{f}(\vec{U}), \tag{3.38}$$

where $\vec{f}(\vec{U})$, which can be viewed as a lumped "spatial operator", here denotes $\sum_{f=1}^{nFaces} -\overline{\vec{H}_{\perp f}} S_f$ in Equation 3.7.

In the given form, Equation 3.38 can be regarded as an Ordinary Differential Equation in time, which can be discretized (in time), and numerically integrated using any technique from a wide range of suitable ones for equations in this category. The techniques of temporal discretization that are most commonly used for equations in this category (and specifically for the semi-discrete form of The System of Euler Equations) are based either on: (i) The family of Linear Multi-Step Methods [37]; or, (ii) the family of Runge-Kutta Methods [185]. The specific discretization adopted determines the order of accuracy in time of the overall computational scheme. Obviously, the options just described for the time-integration scheme are identical whether the starting point is taken to be Equation 3.38 or its analytically more general variant, Equation 3.7.

Perhaps the most consequential feature of a time-integration scheme is whether it is implicit or explicit. For the work presented here, only explicit schemes are appropriate and attractive choices. The following are the (two) main factors contributing to this situation:

1. As explained in Chapter I, in its most general mode of operation, the solution algorithm in this work must be applicable to problems with moving boundaries. As outlined below in this section, and as described in more detail in Chapter VII, a fundamental constraint of the overall cell-merging algorithm developed in this work (to enable boundaries to move across the stationary grid) is that a boundary may cross no more one computational cell during a time-step.

2. As explained in Chapter I, the solution algorithm in this work utilizes adaptation of the grid to the solution. This implies that flow features must be tracked and retained within cells having a given level of spatial resolution, even as these features move across the grid. This in turn implies that a flow feature may move across no more than one computational cell during a time-step.

The two considerations in the above list imply that the integration time-step must be small enough for any displacements of the boundary or any waves in the solution to propagate no more than one computational cell during a single time-step. These constraints are similar to the time-step limits imposed by the stability constraints of an explicit time-integration scheme. Therefore, the main practical advantage of an implicit time-integration scheme relative to an explicit time-integration scheme, namely, the use of larger time-steps in the time-integration of the semi-discrete equation, cannot be realized in the solution algorithm adopted and developed in this work. Therefore, the additional complexity, computational effort, and storage-space requirements of an implicit time-integration scheme relative to an explicit time-integration scheme cannot be justified for the uniform solution algorithm developed and adopted in this work, and an explicit time-integration scheme is the more appropriate choice here.

3.3.2 Formulation and Properties of the Chosen Temporal Discretization and Time-Integration Scheme

Closely associated with time-integration schemes and procedures is the concept of **residual**. Regardless of the motion or deformation of a computational cell, the residual vector, $Res(\vec{U}V(t))$, in that cell is defined as the instantaneous rate of change of the conserved variables in that cell, that is,

$$Res(\vec{U}V(t)) = \frac{\partial}{\partial t}\int_{\Omega}\vec{U}, \tag{3.39}$$

where V is the volume of the computational cell, and all other symbols are as defined in Section 3.2 above. In exact discrete form, the residual may be expressed by

$$Res(\vec{U}V(t)) = \sum_{f=1}^{nFaces} -\overline{\vec{H}_{\perp f}}(t)S_f(t), \tag{3.40}$$

where all symbols have the same meanings given to them above and in Section 3.2, and where the time-dependence in the left-hand side applies to both $\vec{U}$ and V.

One of the simplest temporal discretizations possible is obtained by approximating the time-derivative in a semi-discrete equation, such as Equation 3.38, by the corresponding first-order finite-difference in time. Applied to the left-hand sides of, say, Equation 3.38 or 3.40, this approximation can be expressed by

$$\frac{\partial\overline{\vec{U}}V}{\partial t} = Res(\vec{U}V(t)) \approx \frac{\overline{\vec{U}_f}V_f - \overline{\vec{U}_i}V_i}{\Delta t} \tag{3.41}$$

where $\Delta t = t_f - t_i$ is the integration time-step (or the integration interval), where the subscript f denotes association with the ending or final time, and the subscript i denotes association with the starting or initial time, and all other symbols have the meanings defined for them above. As implied above, this approximation is first-order accurate in time (or in Δt).

One of the simplest time-integration schemes possible can be formulated from the first-order-accurate approximation of Equation 3.41. The time-integration scheme is obtained by evaluating the residual or the right-hand side of the semi-discrete equation under consideration at the starting time, t_i, of the time-step, and re-arranging the equation to obtain the unknowns at the ending time, t_f, of the time-step, in fully explicit form. This scheme is called the Forward-Euler Scheme, and for Equation 3.38, for example, it can be expressed in the form

$$\overline{\vec{U}_f} V_f = \overline{\vec{U}_i} V_i + \Delta t Res(\vec{U} V(t_i)), \tag{3.42}$$

where all symbols are as defined above, and where the residual is typically evaluated in accordance with Equation 3.40. The implicit variant of this explicit time-integration scheme, the Backward-Euler Scheme, is obtained by evaluating the residual or the right-hand side of the semi-discrete equation under consideration at the ending time, t_f, of the time-step, instead of at the starting time of the time-step.

As indicated in Equation 3.42, the fluxes and geometries for all the terms on the right-hand side of that equation are evaluated at the initial time, t_i. As mentioned above, this scheme is only first-order accurate in time, and for moving and deforming cells, it does not generally satisfy the Geometric Conservation Law requirements, which are described in Section 3.4 below. As explained in Section 3.4, this implies that the scheme cannot generally preserve the free-stream state for a problem with moving boundaries. This scheme is also unstable for any spatial discretization scheme that has an order of accuracy higher than 1. Therefore, other than for code verification calculations with stationary boundaries, this scheme was not used in this work.

The actual "workhorse" scheme used in this work is based on a Predictor-Corrector formulation which is both second-order accurate, and also satisfies the Geometric Conservation Law requirements, which are explained in Section 3.4. The scheme is given by:

$$\begin{aligned}
(\vec{U}V)^{(0)} &= (\vec{U}V)^{n}, \\
(\vec{U}V)^{(1)} &= (\vec{U}V)^{(0)} + \Delta t \left[Res^{*}(\vec{U}V)^{(0)} \right], \\
(\vec{U}V)^{(2)} &= (\vec{U}V)^{(0)} + \frac{\Delta t}{2} \left[Res^{*}(\vec{U}V)^{(0)} + Res^{*}(\vec{U}V)^{(1)} \right] \\
(\vec{U}V)^{(n+1)} &= (\vec{U}V)^{(2)},
\end{aligned} \tag{3.43}$$

where the "starred" residuals have the following meanings:

$$Res^{*}(\vec{U}V)^{(0)} = Res(V^{(a)}\vec{U}^{(0)}) = \sum_{j=1}^{nFaces} -\overline{\vec{H}_{\perp j}}(\vec{U}^{(0)})S_j(t_a),$$

and

$$Res^{*}(\vec{U}V)^{(1)} = Res(V^{(a)}\vec{U}^{(1)}) = \sum_{j=1}^{nFaces} -\overline{\vec{H}_{\perp j}}(\vec{U}^{(1)})S_j(t_a),$$

where $V^{(a)}$, which denotes the time-averaged cell-geometry, is given by

$$V^{(a)} = \frac{V^{(0)} + V^{(1)}}{2},$$

where $V^{(0)}$ denotes the volume (and the geometry for calculating the fluxes) of the computational cell at the starting time, $t = t_i$ of the current time-step, where $V^{(1)}$ denotes the volume (and the geometry for calculating the fluxes) of the computational cell at the ending time, $t = t_f$, of the current time-step, and where t_a denotes the mid-interval time, $t_a = \frac{t_i + t_f}{2}$. It should be noted here that in this work, any motions of boundaries are calculated explicitly and a-priori for any given time-step (from either prescribed functions, or by integrating the forces and moments applied to the boundary at the preceding time-step), and are not adjusted in accordance with the Predictor-Corrector formulation. As a result of this, the relation $V^{(1)} = V^{(2)}$ is always satisfied. The manner in which boundary locations are updated is described in more detail in Chapter V.

In words, what the above Predictor-Corrector scheme means is that while the state-vector values may change from the Predictor to the Corrector steps, the fluxes are evaluated for the same time-averaged geometry for both the Predictor and the Corrector steps. More emphatically, this means that all the edge lengths, Gauss-point locations, and cell areas used in computing the fluxes or residuals (in both of the steps of the Predictor-Corrector scheme) are taken to be the time-averages of the corresponding quantities across the entire time-step.

For stationary boundaries, the time-integration scheme of Equation 3.43 reduces to Heun's Method, and can also be considered as a two-stage Runge-Kutta scheme.

The second-order accuracy cited above for this time-integration scheme can be established by direct Taylor Series Expansion of the third Equation of the Equation System 3.43, and using the relation

$$\vec{f}(\vec{U})\frac{\partial \vec{f}(\vec{U})}{\partial \vec{U}} = \frac{\partial \vec{U}}{\partial t}\frac{\partial \vec{f}(\vec{U})}{\partial \vec{U}} = \frac{\partial \vec{f}(\vec{U})}{\partial t} = \frac{\partial^2 \vec{U}}{\partial t^2}$$

where $\vec{f}(\vec{U})$ denotes the same meaning assigned to it for Equation 3.38, or equivalently, $\vec{f}(\vec{U})$ denotes the residual $Res^*(\vec{U}V)$, or equivalently, it denotes the term $\sum_{f=1}^{nFaces} -\overline{\vec{H}_{\perp f}} S_f$ in the semi-discrete form of Equation 3.7, and where all other symbols are as defined above. The second-order accuracy in time of the scheme of Equation 3.43 can also be established indirectly [169].

The manner in which the use of the time-averaged geometry in Equation 3.43 for computing fluxes ensures that the Geometric Conservation Law requirements are satisfied is explained in detail in Section 3.4.

As evident from the formulation given above, all the discrete unknowns or variables in every discrete equation in the time-integration scheme are simultaneously drawn from the same time-level. This greatly simplifies the implementation of the time-integration scheme. Schemes which require the combination of variables from different time-levels in individual discrete equations are only used in practice when there are strong reasons for doing so, as is the case for some turbomachinery applications, for example [130].

As also evident from the formulation given above, the discrete unknowns (or variables) in the scheme adopted in this work need be simultaneously stored at only two time levels. This is typical of second-order-accurate time-integration schemes. Similarly, all geometric data used in computing the solution update, such as edge lengths and cell areas, need be simultaneously stored or computed in the adopted scheme at only two time-levels. This simplifies the implementation of the overall scheme. It also reduces the storage requirements of the scheme. It also reduces the number of global body-grid intersection evaluations that must be carried out, as explained in Chapter V, to one evaluation for each new time-step, keeping the overall computational cost to the minimum possible.

Despite the adoption of a time-integration scheme which operates using data from only two time-levels simultaneously, the overall software framework developed in this work allows for an arbitrary number of time-levels to be introduced (both in the time-integration scheme, and the geometric intersection configurations). Indeed, a scheme independently implemented within this software framework (for computation of incompressible flows with moving boundaries [8, 9]) used three time-levels (in both the time-integration scheme and the geometry definition).

A key requirement for proper use of any time-integration scheme is the characterization of its stability limits. For the Heun Method adopted in this work, this requirement, in terms of the local (that is, using a cell-by-cell evaluation method) CFL Number, as shown in [169], is given by the equation

$$CFL = \frac{(\sqrt{u^2 + v^2} + c)\Delta t}{\Delta x} \leq 1.0 \tag{3.44}$$

where Δx denotes the dimension of the computational cell, and where all other terms are as defined above.

An important additional feature of the Heun Predictor-Corrector time-integration scheme adopted in this work when used in conjunction with the MUSCL type of second-order spatial discretization also adopted in this work, is that the overall numerical scheme satisfies the **Total Variation Diminishing (TVD)** property, which is defined and discussed in detail in [170]. The latter reference also describes and discusses the manner in which the TVD property is satisfied by a numerical scheme like the one adopted in this work.

As outlined above in this section, for a problem with moving boundaries, there is a constraint on the time-step that is associated with the extent of displacement of boundary points across the grid. This constraint can be viewed as being imposed by the cell-merging algorithm, or even by fundamental geometric-consistency requirements, and is independent of the fundamental stability constraint of Equation 3.44, even though the two constraints roughly have the same limiting effect on the time-step. In particular, the geometric constraint imposes the requirement that computational cells do not "intersect" themselves or form degenerate geometries. A necessary but not sufficient constraint to satisfy this requirement is that the cell volume remains positive, that is, that the sum of the swept volumes on all the faces does not become sufficiently negative to make V_{t_f} non-positive in Equation 3.54 below. For the special case of a moving boundary on a stationary grid (which is the only case applicable in this work), this constraint reduces to

$$max(|\Delta x_b|, |\Delta y_b|) < h_{cell}, \tag{3.45}$$

where $|\Delta x_b|$ and $|\Delta y_b|$ are respectively the greatest local absolute displacements in the $x-$ and $y-$directions of a point in the boundary, and h_{cell} is the dimension of the composite cell [3] in which the boundary moves.

3.4 Satisfaction of The Geometric Conservation Laws

This section presents and discusses the quantitative relations between interdependent geometric entities (such as the coordinates of cell vertices, the volumes or areas of cells, and the speeds of cell vertices and cell faces) that should be preserved within the formulation of a computational scheme, and in the discretization of cell geometries and motions.

Preservation of the relations between the relevant geometric variables of a subregion set at the discrete level will be shown to eliminate a variety of fundamental cell-geometry-related and cell-motion-related sources of error in the solution. While the latter errors apply only if there is cell motion or deformation, the former ones apply whether cells are stationary or moving. As well as ensuring conservation for moving or deforming cells, it will also be shown that an important and desirable consequence of eliminating these fundamental sources of error is that an arbitrary uniform state can be preserved to within the arithmetic precision of the computation for arbitrary grid motions and deformations. Moreover, for a moving-boundary

[3] The definition and role of composite cells is outlined in Chapter I, and described in detail in Chapter VII.

problem with a stationary grid and a uniform, arbitrary free-stream, eliminating these fundamental sources of error will ensure that the free-stream is preserved if all boundaries move with the same uniform velocity as that of the free-stream.

3.4.1 Continuous Form of the Geometric Conservation Laws

The generic control volume, Ω, universally used to derive the integral forms of conservation laws in the Theory of Continuum Hydrodynamics (see Chapter II for the instance of The System of Euler Equations) by construction intrinsically satisfies the equations

$$\int_{\partial\Omega} \vec{n} \quad = \quad \vec{0}, \tag{3.46}$$

and

$$\frac{\partial}{\partial t}\int_{\Omega} 1 \quad = \quad \int_{\partial\Omega} \vec{v}_s \cdot \vec{n}, \tag{3.47}$$

where all symbols are as defined in Chapter II. Equation 3.46 expresses the requirement for "closure upon itself" of the control surface, while Equation 3.47 expresses the requirement for any variation in the volume of Ω to remain arithmetically consistent with the increases or decreases in volume caused by any motion of the control surface. These two elementary geometric identities are together known as [355, 356] **The Geometric Conservation Laws**, and more briefly called, as done hereinbelow, "The GCLs".

A few important consequences of Equations 3.46 and 3.47 on the properties of conservation laws can be distinctively revealed by analyzing Equation 2.7.

Using the result $\vec{\vec{F}}_r = \vec{U} \circ \vec{v}_r^T = \vec{U} \circ (\vec{v} - \vec{v}_s)^T$ (shown in Chapter II), Equation 2.7 may be re-written in the form

$$\frac{\partial}{\partial t}\int_{\Omega} \vec{U} \quad + \quad \int_{\partial\Omega} \vec{U} \circ (\vec{v} - \vec{v}_s)^T \cdot \vec{n} \quad = \quad \int_{\partial\Omega} \vec{\vec{S}} \cdot \vec{n}, \tag{3.48}$$

where all symbols are as defined in Chapter II. For an arbitrary spatially-uniform initial state $\vec{U}$ (and hence a uniform velocity, $\vec{v}$, and a uniform source tensor, $\vec{\vec{S}}$), Equation 3.48 may be re-written in the form

$$\frac{\partial}{\partial t}\int_{\Omega}\vec{U} \quad + \quad \left(\vec{U}\circ\vec{v}^{T}\right)\cdot\int_{\partial\Omega}\vec{n} \quad - \quad \vec{U}\int_{\partial\Omega}\vec{v}_s\cdot\vec{n} \quad = \quad \vec{\vec{S}}\cdot\int_{\partial\Omega}\vec{n}. \tag{3.49}$$

Two independent consequences of Equation 3.46 are clearly apparent in Equation 3.49: (i) the vanishing of the convective term $\left(\vec{U}\circ\vec{v}^{T}\right)\cdot\int_{\partial\Omega}\vec{n}$ (and hence of its contribution to the change in conserved quantities $\int_{\Omega}\vec{U}$); and, (ii) the vanishing of the source term $\vec{\vec{S}}\cdot\int_{\partial\Omega}\vec{n}$ (and hence of its contribution to the change in conserved quantities $\int_{\Omega}\vec{U}$). These results hold for arbitrary motions of the control surface, including the stationary case. Substitution of Equation 3.46 into Equation 3.49 therefore gives

$$\frac{\partial}{\partial t}\int_{\Omega}\vec{U} \quad - \quad \vec{U}\int_{\partial\Omega}\vec{v}_s\cdot\vec{n} \quad = \quad \vec{0}, \tag{3.50}$$

which, since $\vec{U}$ is initially uniform in space, can be re-written in the form

$$\frac{\partial\vec{U}}{\partial t}\int_{\Omega}1 \quad + \quad \vec{U}\frac{\partial}{\partial t}\int_{\Omega}1 \quad - \quad \vec{U}\int_{\partial\Omega}\vec{v}_s\cdot\vec{n} \quad = \quad \vec{0}. \tag{3.51}$$

Substitution of Equation 3.47 into Equation 3.51 (and division by $\int_{\Omega}1$) gives the result

$$\frac{\partial\vec{U}}{\partial t}=\vec{0}. \tag{3.52}$$

Therefore, the implication of Equation 3.47 in Equation 3.51 (which already incorporates Equation 3.46) is that an arbitrary spatially-uniform initial state is preserved for all time throughout a control volume for arbitrary motions of the control surface.

The results established above are obvious from the physical point of view: the purpose of the derivation is more to identify the mathematical mechanism through which The GCLs effect these results, as well as the expected outcomes of failing to satisfy The GCLs in a computational scheme [4].

Although the derivation presented above was applied only to The System of Euler Equations, the results can be seen to apply to any system of equations which can be written in the general form of Equation 2.7, including the Navier-Stokes Equations, as well as a wide range of equations throughout the Continuum Theory of matter. The vanishing of the effect of any diffusive terms that are present is guaranteed solely by satisfaction of Equation 3.46, since, unlike the case for convective terms, the instantaneous differential local contribution of any diffusive terms is influenced only by the geometry, and not the motion of the control surface.

3.4.2 Discrete Form of the Geometric Conservation Laws

The discrete analogs of Equations 3.46 and 3.47 for any topologically-invariant computational cell with polyhedral geometry, are respectively given by

$$\sum_{f=1}^{nFaces} \vec{S}_f = \vec{0}, \tag{3.53}$$

and

$$V_{t_f} - V_{t_i} = \sum_{f=1}^{nFaces} \Delta V_f, \tag{3.54}$$

where $nFaces$ is the number of faces of the computational cell, where $\vec{S}_f$ is the surface-area vector of face f of the computational cell, where V_{t_f} and V_{t_i} respectively denote the volume of the computational cell at times t_f and t_i, where

$$\Delta V_f = \int_{t_i}^{t_f} \int_f \vec{v}_s \cdot \vec{n},$$

[4]This is why it was chosen to start the derivation with Equation 2.7 instead of Equation 2.9: the latter leads to the final result of Equation 3.52 more directly, but because Equation 2.9 already incorporates Leibnitz's Rule, the derivation would not have illustrated the sought implications of Equations 3.46 and 3.47.

where $\int_f$ denotes integration over face f of the computational cell, and where all other symbols are as defined in Chapter II and above in this chapter.

The discrete forms given in Equations 3.53 and 3.54, as well as other more general forms that are valid for non-polyhedral computational cells may collectively be called **The Discrete Geometric Conservation Laws**, and will hereinbelow also be called "The DGCLs".

The interpretations of Equations 3.53 and 3.54 for a specific computational cell are visually illustrated in Figure 3.2 below. The interpretation of Equation 3.53 is that the cell must remain closed *at the level of the discrete representation*, at each point in time or each time-level at which a flux calculation or a flux integration is performed. The interpretation of Equation 3.54 is that the (signed) volumes that are individually swept by each of the moving faces of the cell must sum *at the discrete level* to exactly the net change in the *discrete* volume of the cell. Although Figure 3.2 shows a quadrilateral cell in two-dimensional space, the interpretations given above hold for cells of arbitrary geometry in spaces of any dimension.

The analytical interpretations and consequences of Equations 3.53 and 3.54 in the discrete form of a system of conservation laws [168, 108] are analogous to the analytical interpretations and consequences of their continuous analogs in the continuous form of that system; namely: (i) failure to satisfy Equation 3.53 results in the numerical generation of spurious net convective fluxes (but without violating conservation if a conservative discretization is used); and, (ii) failure to satisfy Equation 3.54 for moving or deforming cells results in the numerical generation of conservation-violating spurious source and sink terms (even if a conservative discretization is used). Thus, violation of either of Equations 3.53 or 3.54 would prevent the preservation of an arbitrary initial free-stream. However, while satisfaction of Equation 3.54 requires

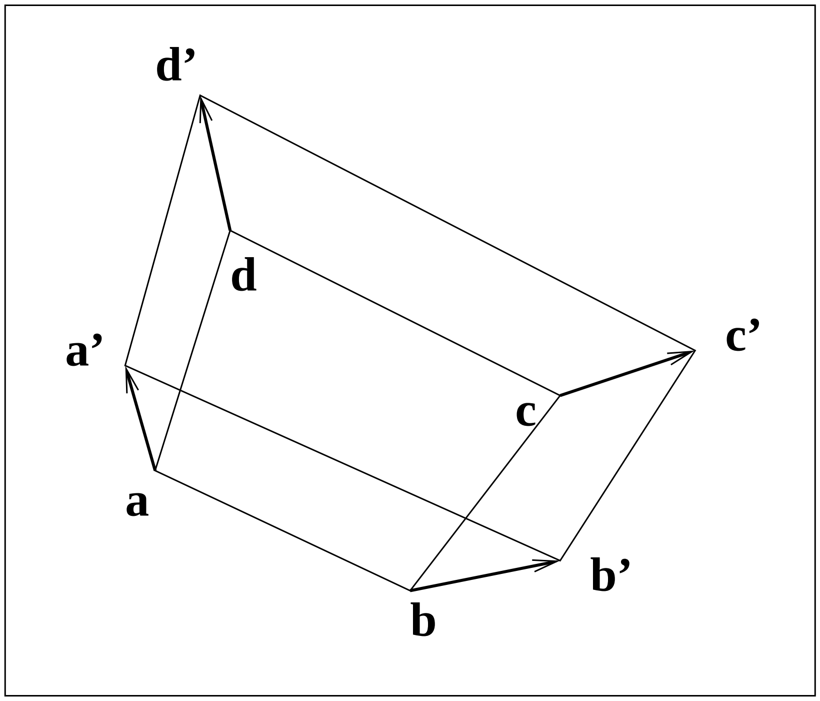

Figure 3.2: Deformation of a two-dimensional computational cell, showing the original and final geometries of the cell, and showing the displacement of each vertex. The initial locations of the vertices are indicated by regular letters, and the corresponding final locations are indicated by the corresponding “primed” letters. The figure clearly indicates the net volume (or area) swept by each face (or edge) of the cell during the deformation.

additional effort only if there is cell motion or deformation, satisfaction of Equation 3.53 is a concern whether or not there is cell motion or deformation.

The DGCLs must be satisfied individually by each computational cell in the subregion set whenever a Finite-Volume, a Finite-Element, or a Finite-Difference formulation is applied. In the Finite-Difference case, The GCLs must be satisfied indirectly [396] since there are no surfaces, volumes, or cells: The GCLs must be incorporated into the computations of the Jacobians, coordinate-transformation metrics, and vertex velocities. The required derivations have been carried out for the standard Finite-Difference formulation [396, 429] as well as for a combined, space-time-continuum formulation [354].

Although, just like their continuous analogs, The DGCLs are intuitively obvious from both a physical and a mathematical point of view, what is not so obvious are two characteristic features of their application: (i) they must be satisfied *exactly, at the discrete level* (for example, like the uniqueness of the flux computation in a conservative discretization); and, (ii) they must be satisfied implicitly [429] as part of the formulation of the calculations of volumes and surface areas, and as part of the time-integration scheme, rather than solved directly as two additional equations. For this reason, The DGCLs have also been called The Implicit Geometric Conservation Laws [429] in the context of their incorporation or implementation in computational schemes.

The DGCLs can be readily satisfied in any computational scheme that commits no inconsistencies in the calculation and use of values of geometric quantities related to volumes and surface areas (which are the quantities in which The GCLs are expressed). General desiderata and guidelines often cited in the literature (for example, in [297], [33, 34], [123, 218], [263], [225], [108], and, for the special case of known grid velocity [109]) include the following:

- Consistency at the discrete level between all geometric computations, including those of primary and derived quantities, and consistency in the use of the values of geometric entities between the geometric algorithms and the "flow solver";

- Consistency of the surface area and swept volume computations between neighboring cells that share a face. This often implies that the area and volume computations should be independent of the traversal order along the vertices of a face. In the two-dimensional case, it is easy to ensure this uniqueness.

The three-dimensional case, however, requires special handling; for example, the arithmetic average of all the possible alternatives can be shown to give a consistent and unique value [429];

- Computation of the fluxes (or residuals) at the same set of time levels, in order to ensure that the surface-area vectors for each computational cell sum to zero; and,

- For each cell, consistency between the surface areas, the velocities, and the swept-volumes of the individual moving faces of the cell on the one hand, and the change in the volume of the cell on the other hand.

A useful result related to the last item in the above list, is that the volume swept by a face may be computed exactly (at the discrete level) by assuming that each vertex of a face has a constant velocity throughout a time-step, and by assuming that the "mean" face geometry corresponds to the geometry at the mid-point of the time-step. Thus, the formula for the volume swept by a two-dimensional moving cell face may be computed from

$$\Delta V_f = \left(\Delta t \left(\frac{\vec{v}_{v1} + \vec{v}_{v2}}{2} \right) \times \vec{l}_{t_{(1/2)}} \right) \cdot \vec{k}, \tag{3.55}$$

where $\vec{v}_{v1}$ and $\vec{v}_{v2}$ are the time-averaged (or constant) velocities of the two vertices, $v1$, and $v2$, of the face f, $\vec{l}_{t_{(1/2)}}$ is the edge vector at the mid-point of the time-step assuming uniform vertex velocities, and $\vec{k}$ is the unit vector along the z-axis. The time-averaged velocities of the vertices can most readily be computed by dividing the net displacement vectors of the vertices over the time-step by the magnitude of the time-step. Using the time-averages of the velocities ensures that the result expressed in Equation 3.55 remains exactly satisfied, regardless of the actual trajectory of the vertices $v1$ and $v2$ during a time-step, and even if the velocities of these vertices varies during the time-step. An example depicting the volume swept by one of the faces of the cell of Figure 3.2 is shown in Figure 3.3 below. The figure shows the relevant vertex displacements and the geometry of the moving face at the mid-point of the time interval of the deformation.

A result similar to that expressed for the two-dimensional case in Equation 3.55 holds for the axisymmetric case [429], and, provided faces remain planar, also for the three-dimensional case [429]. If faces do not remain planar, one possibility is to compute swept volumes by subdividing all cells into tetrahedra (and hence all faces into triangles), in order to ensure that all planar faces will remain planar [429].

The list of desiderata given above is mostly a list of obvious errors that should be avoided. Any consistent and accurate scheme for computing the geometry of the subregion set will naturally satisfy all the items in it. The particular scheme chosen in this work computes all geometric entities used in any part of the computational scheme exactly (to the available arithmetic precision), from the vertex coordinates. Thus, for example, face areas and swept volumes are computed exactly from the locations and the changes in the locations of the related cell vertices, and the mean speeds of cell faces across a time-step are computed exactly from the changes in the positions of the face centroids using the vertex coordinates at the beginning and end of the time-step. For polyhedral cells, the vertex locations as a function of time may be considered as the most primitive geometric variables that uniquely and exactly specify all other geometric entities as functions of time. This uniqueness alone ensures the internal consistency between all derived geometric entities.

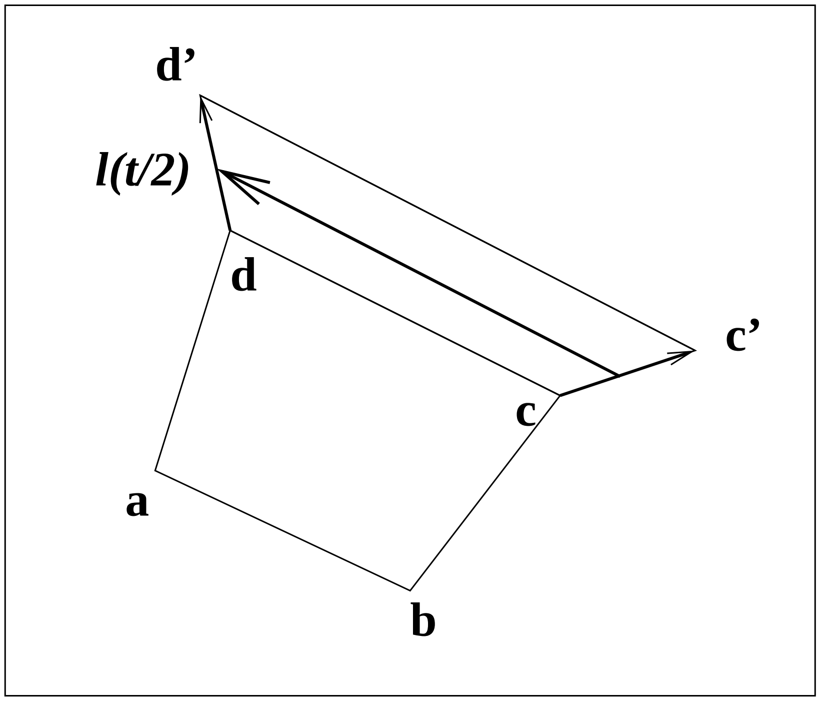

Figure 3.3: Computing the volume (or area) swept by a moving face: the swept volume (or area), here represented by the quadrilateral $c - c' - d' - d$, can be obtained from the cross product of the vector representing the face area (or edge length, here indicated by $l(t/2)$) at the mid-point of the time-interval, with the vector representing the average displacement of the vertices of the face (or edge) across the time-interval.

3.4.3 Requirements for Boundaries that Move on a Stationary Grid

In the computational technique developed and studied in this work, the original cells of the Cartesian-Quadtree grid always remain stationary, even if there are moving boundaries. Any of these cells that is intersected by a boundary is divided into two or more disconnected sub-cells: one that falls “inside” the boundary, and one that falls “outside” the boundary. The faces that separate each of these sub-cells are therefore naturally defined from the geometry of the boundary, and are handled as full-fledged cell faces for each of the two sub-cells that they bound. If any part of a boundary moves, it must travel across the stationary cells through which that part passes, moving the faces that divide these stationary cells in the process. Thus, the only cell faces that may move are ones associated with boundaries.

The case where boundaries move on a stationary grid, as described in the preceding paragraph, and as arises in the technique developed in this work, is clearly a special case of the generalized moving- and deforming-grid case. This sub-section presents the special considerations related to The GCLs and The DGCLs for this special case, and presents the specific treatments used in this work to satisfy The GCLs and The DGCLs for this special case.

As outlined in Section 1.4 and as explained in detail in Chapter VII, the cell-merging procedure used in this work combines and reconfigures the original "geometric" cells of the Cartesian-Quadtree grid in the vicinity of boundaries into "computational" composite cells comprising one or more of the original cells. The cells are combined so that every "internal" boundary is wholly contained within composite cells, and so that every moving boundary remains inside the same set of composite cells over any time-step. Furthermore, the merging procedure ensures that only two general patterns of boundary-motion occur in composite cells: (i) a pattern in which the initial and final boundary intersections occur on the same two opposite faces of a composite cell, as simplifyingly depicted in Figure 3.4; and, (ii) a pattern in which the initial and final boundary intersections occur on the same two adjacent faces of a composite cell, as simplifyingly depicted in Figure 3.5.

Figures 3.4 and 3.5 show only typical convex composite cells, only planar boundaries, and only the "retraction" direction of boundary motion. However, the motion patterns are similar for non-convex composite cells, for boundaries of arbitrary shape, and for both directions of boundary motion, as described in detail in Chapter VII. Since, as explained in Chapter VII, the flow solver operates only on composite cells, the two configurations simplifyingly depicted in Figures 3.4 and 3.5 are the only ones that need to be specifically considered in this work in regard to satisfying The GCLs for moving or deforming cells.

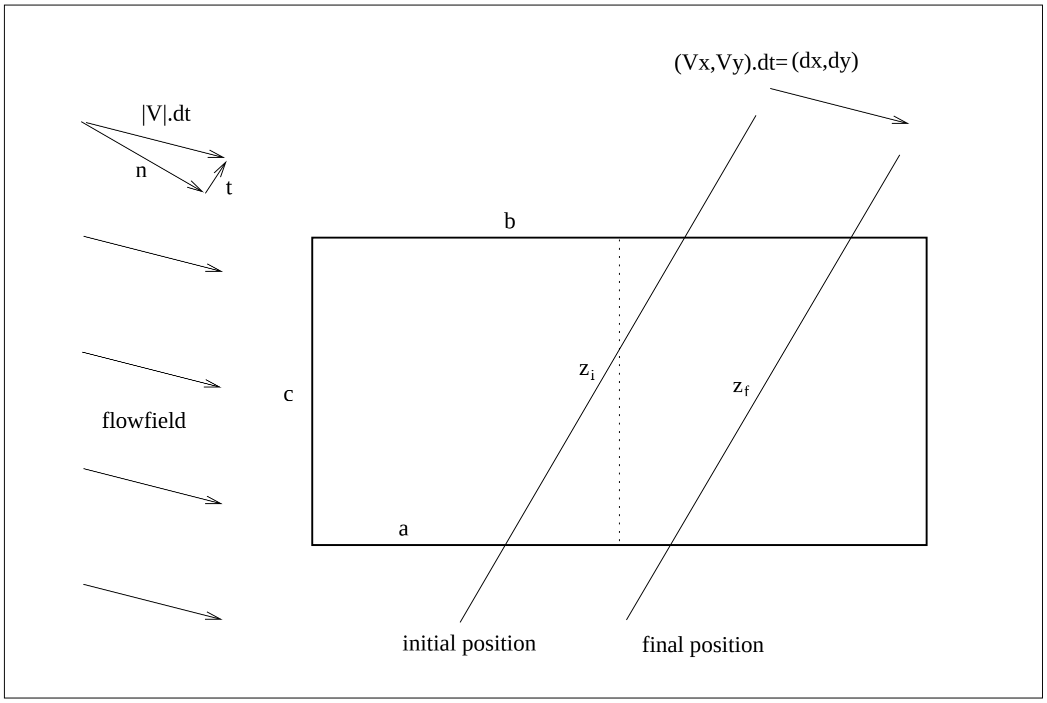

Figure 3.4: Motion Pattern for a composite cell (comprised of two Cartesian cells) intersected on opposite edges by a boundary that remains locally planar.

Figures 3.4 and 3.5 describe specifically only the case of planar boundaries. Figure 3.6 describes the more general case where the boundary is curved and the velocity of the boundary is *not* parallel to one of the Cartesian axes. Even though the velocity is also not parallel to either of the Cartesian axes in the cases represented in both of Figures 3.4 and 3.5, the significance of this non-parallelism is far greater for the case of curved boundaries, as explained in the next two paragraphs.

As shown in Figure 3.6, the difference between the initial and final geometric truncations of the boundary causes the discretized representation of the boundary

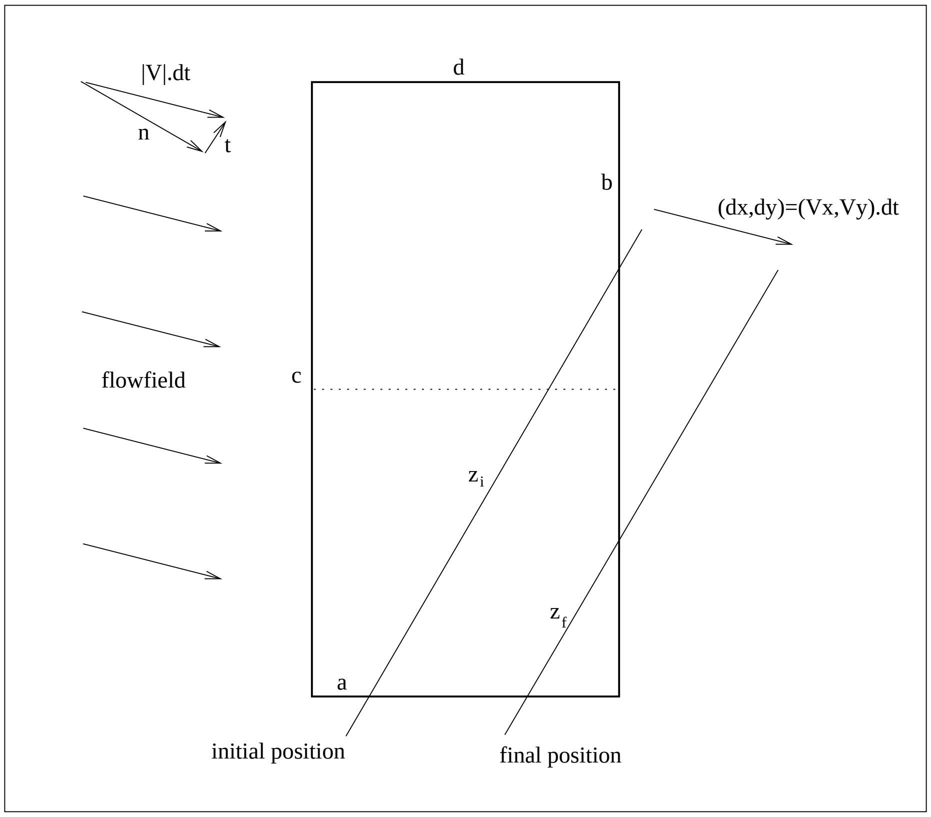

Figure 3.5: Motion pattern for a composite cell (comprised of two Cartesian cells) intersected on adjacent edges by a boundary that remains locally planar.

to move with a velocity that differs from the actual uniform velocity of the boundary (and that of the uniform free-stream). Not only are the average speed and the average direction of motion of the line connecting the intersection points generally different from those of the boundary, but as shown in the figure, an artificial rotational component may be introduced into that line as well. The distortion described above is eliminated only in the special case where the velocity of the curved boundary is parallel to one of the Cartesian axes and for the configuration in which the boundary intersects the composite cell along parallel faces. For all other configurations or directions of motion, the segment of the boundary that is truncated by the composite cell will change in general.

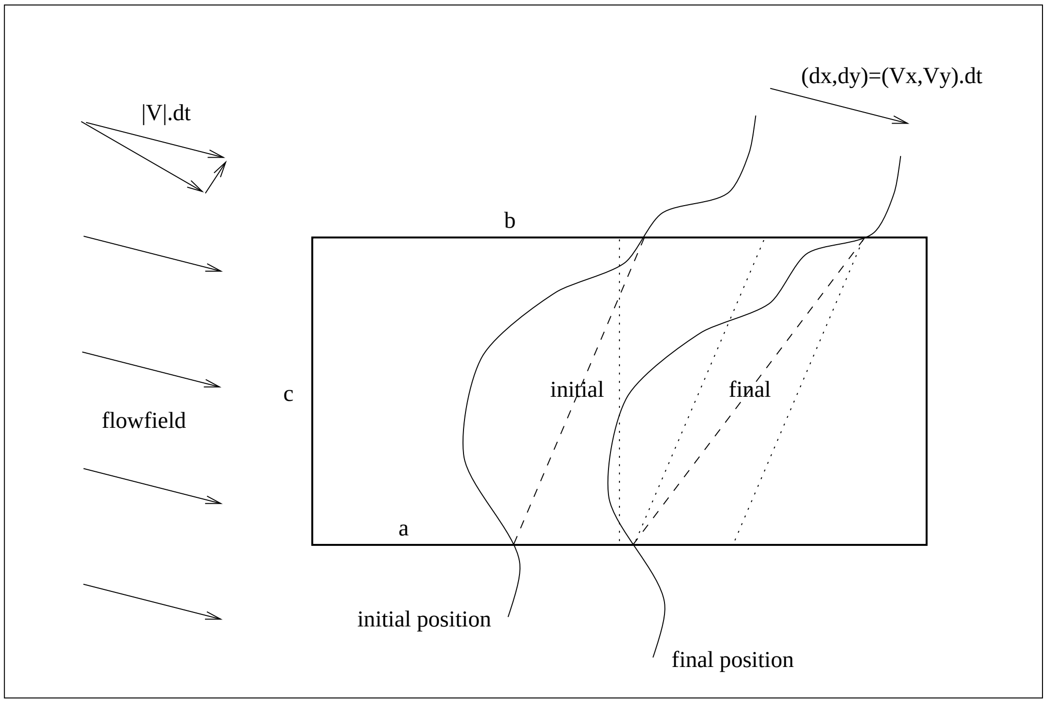

Figure 3.6: Representation of the uniform motion of a curved boundary in a composite cell (comprised of two Cartesian cells), showing truncation-induced distortions in the rectilinear and angular velocities of the discretized boundary geometry.

Figure 3.7 shows another example of the truncation effects that can arise with curved boundaries moving in directions not parallel to either of the coordinate axes. This example is perhaps even more extreme than that of Figure 3.6, because it shows how the truncated discrete representation of the boundary appears to be stationary during the given time-step even though the boundary is moving at a uniform velocity. In this case, the change in volume of the cell will incorrectly evaluate to zero if only the truncated geometry is considered.

As evident from the representation of boundary motion described above and in Figures 3.4, 3.5, 3.6, and 3.7, the most general change in composite cell geometry that must be considered in this work contains the following elements:

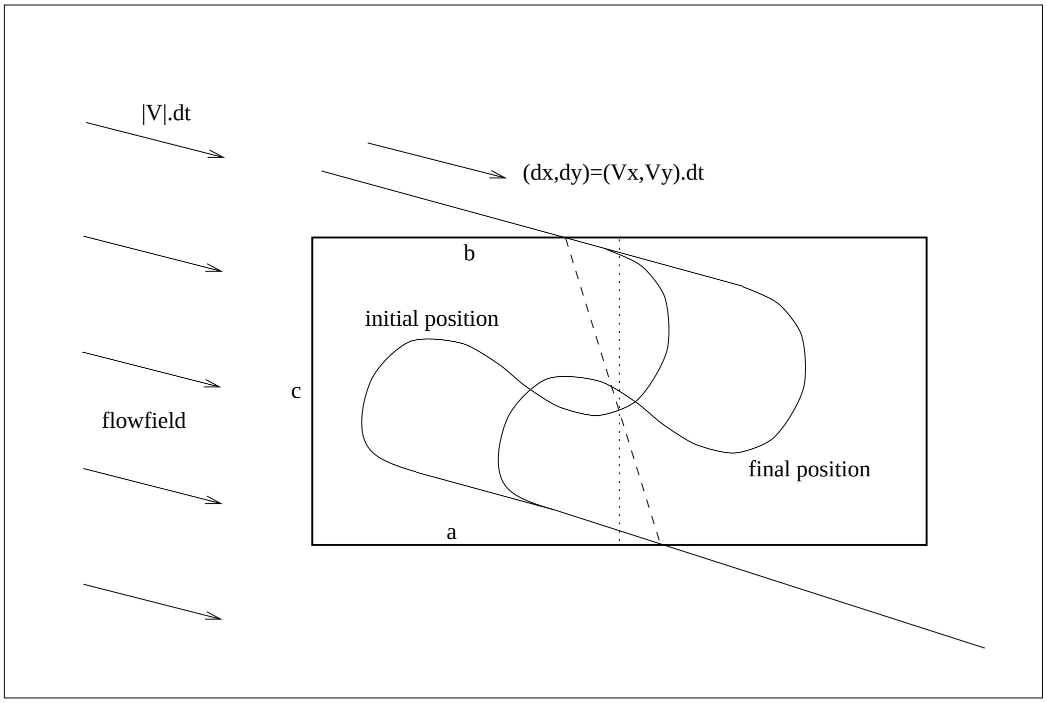

Figure 3.7: Representation of the uniform motion of a curved boundary in a composite cell (comprised of two Cartesian cells), showing a degenerate case in which the discrete representation of the boundary appears to be stationary.

- Motion of cell faces that are associated with boundaries;
- Elongation or contraction of the intersected faces of intersected composite cells (along the original, fixed orientations of these faces); and,
- Change in the truncated geometry of a boundary due to a change in the orientation or location of the boundary within a composite cell.

Despite the restrictions implied in the first two items in the above list that are imposed in this work on the motion or deformation patterns of cells, no reduction in the generality of the formulation or the implementation of The GCLs as described in the preceding two sub-sections is possible. This is because the moving faces may be either permeable or impermeable and thus must be treated in the most general way, and because, as described below (see Figure 3.8, for example), each composite cell may have an arbitrary nonnegative number of moving faces.

The need to additionally account for variations in the truncation of boundary geometries that arise in the technique developed in this work in a manner that ensures satisfaction of The GCLs is an unusual requirement or complication: this need does not arise in "traditional" moving-mesh techniques (for example those of [297], [123, 218], [225], [263], [33, 34], and [108]). Another major difference between the technique developed in this work and the "traditional" moving-mesh methods is the following: the cells that are subject to a change in geometry in the technique developed in this work all lie immediately around any "internal" moving boundaries, reducing the "dimensionality" of the number of cells that require special treatment for satisfaction of The GCLs (relative to the corresponding number for "traditional" moving-mesh methods) by one.

The remainder of this section is concerned with the manner in which the space-integration and time-integration formulations used in this work are coordinated to ensure satisfaction of The GCLs, including proper treatment of the change in truncation geometry due to boundary motion. The issues that are especially relevant for a situation in which a boundary moves on a stationary grid are the following:

- Computing all residuals at the same set of time levels to ensure satisfaction of Equation 3.53;

- Selection of the time levels at which to compute the residuals (or, equivalently, selection of the time-integration scheme for computing the fluxes on faces whose areas are changing with time) so that the overall scheme satisfies the required order of accuracy in time; and,

- Computing cell volumes and face velocities for moving faces in a manner that ensures satisfaction of Equation 3.54, regardless of the change in truncated geometry of a boundary segment.

As explained in its derivation, Equation 3.6 is an exact, semi-discrete expression for the generic conservation law for topologically-invariant polyhedral cells of otherwise arbitrary geometry. For any passively-transported, volume-specific, conserved scalar, $\psi = \psi(\vec{x}, t)$, Equation 3.6 may be specialized and re-written in the form

$$\overline{\psi}_f V_f - \overline{\psi}_i V_i = \int_{t_i}^{t_f} \sum_{f=1}^{nFaces} \int_{S_f} -\psi v_{\perp r} + \int_{t_i}^{t_f} \sum_{f=1}^{nFaces} \int_{S_f} s_\psi, \tag{3.56}$$

where the overbar denotes volume-averaging, where s_ψ is the source term of ψ that is applied only through the control surface, and where all other symbols are as defined above in this chapter.

Whether in its first-order-accurate or second-order-accurate spatial discretization modes, the computational scheme used in this work approximates the flux term (that is, the right-hand-side term) in Equation 3.56 by the expression

$$\begin{aligned} F = \; & - \;\; (t_f - t_i) \sum_{f=1}^{nFaces} \left[(\psi v_{\perp r} S_f)_{t=t_i} + (\psi v_{\perp r} S_f)_{t=t_f} \right] /2 \\ & + \;\; (t_f - t_i) \sum_{f=1}^{nFaces} \left[(s_\psi S_f)_{t=t_i} + (s_\psi S_f)_{t=t_f} \right] /2. \end{aligned} \tag{3.57}$$

The main reasons for selecting this approximation are as follows:

1. It involves geometric entities at only the beginning and end of a time-step (or motion step);

2. It allows formulation of computational schemes that are up to second-order accurate in time; and,

3. It enables The DGCLs to be automatically satisfied for piecewise-planar boundary representations [5].

The advantage of compliance with the first item in the above list is the elimination of any geometric computations beyond the minimum necessary to specify the boundary geometry as a function of time and to perform the grid generation required to construct and maintain a valid grid at all time-levels.

The second-order accuracy in time of the approximation expressed in Equation 3.57 can be established from the fact that the time-integrated area of every face is computed exactly for any planar boundary moving at a constant velocity, that is,

$$\int_{t_i}^{t_f} S_f = (t_f - t_i)\left(\frac{S_{f_f} + S_{f_i}}{2}\right),$$

where all symbols are as defined above. As explained below, this result or conclusion can also be extended to any boundary represented with piecewise planar segments.

In order to establish the third claim in the above list, it is necessary and sufficient to prove the claim for each of the two boundary-motion patterns represented in Figures 3.4 and 3.5. The sufficiency stipulation arises because, as explained at the beginning of this sub-section, the boundary-motion patterns represented in Figures 3.4 and 3.5 (and the corresponding ones with oppositely-directed motions) are the only two possible patterns for the motion of boundaries in composite cells.

For any allowed motion pattern of a planar boundary (as described in Figures 3.4 and 3.5), the approximation in Equation 3.57 can be re-written in the form

$$F = \quad - \quad (t_f - t_i) \sum_{f=1}^{nFaces} \left[(\psi v_{\perp tr} S_f)_{t=t_i} + (\psi v_{\perp tr} S_f)_{t=t_f} \right] /2$$

[5]As explained in Chapter V, any "internal" boundary in this work is represented by a string of connected planar facets that are independent of the intersection of the boundary with the grid, not planar facets that result from connecting the intersection points of an arbitrary boundary shape with the Cartesian grid.

$$\begin{aligned} - \quad & (t_f - t_i) \sum_{f=1}^{nFaces} \left[(\psi v_{\perp nr} S_f)_{t=t_i} + (\psi v_{\perp nr} S_f)_{t=t_f} \right] /2 \\ + \quad & (t_f - t_i) \sum_{f=1}^{nFaces} \left[(s_\psi S_f)_{t=t_i} + (s_\psi S_f)_{t=t_f} \right] /2, \end{aligned} \tag{3.58}$$

where $v_{\perp tr}$ and $v_{\perp nr}$ refer to the velocity components normal to cell face f that are respectively due to the velocity component of the flowfield that is tangential to the moving boundary (or the moving cell face), and to the velocity component of the flowfield that is normal to the moving boundary (or the moving cell face).

For a uniform initial state and with a boundary velocity that is everywhere equal to that of the free-stream, the flux of ψ through the moving boundary will be zero (regardless of the permeability or type of the boundary), and the conditions

$$\sum_{f=1}^{nFaces} (\psi v_{\perp tr} S_f)_{t=t_i} = 0, \tag{3.59}$$

$$\sum_{f=1}^{nFaces} (\psi v_{\perp tr} S_f)_{t=t_f} = 0, \tag{3.60}$$

$$\sum_{f=1}^{nFaces} (s_\psi S_f)_{t=t_i} = 0, \tag{3.61}$$

and

$$\sum_{f=1}^{nFaces} (s_\psi S_f)_{t=t_f} = 0, \tag{3.62}$$

will be independently satisfied as a result of "closure" of the boundary of the composite cell at each of the times t_i and t_f, as explained at the beginning of this section.

For the motion pattern described in Figure 3.4, substituting the uniform flowfield conditions, that is, Equations 3.59, 3.60, 3.61, and 3.62 reduces Equation 3.58 to

$$\begin{aligned} F &= -(t_f - t_i)\psi_i v \left[-\left(\frac{b_i + b_f}{2}\right) \sin\theta + \left(\frac{a_i + a_f}{2}\right) \sin\theta - \left(\frac{c_i + c_f}{2}\right) \cos\theta \right] \\ &= (t_f - t_i)\psi_i v \left(\frac{z_i + z_f}{2}\right) \\ &= \psi_i \left(V_f - V_i\right), \end{aligned} \tag{3.63}$$

where θ is angle between the x axis and the projection of the flowfield velocity vector onto the face-normal of the moving boundary (or, equivalently here, between the moving boundary and the x axis), where v is the magnitude (constant over the time-step) of this projection, and where a, b, c, and z refer to the face areas indicated in Figure 3.4, while the subscripts i and f refer to the values at the beginning and end of a time-step. Note that the equalities

$$\psi_i = \psi_f,$$

$$z_i = z_f,$$

$$b_i \sin\theta + c_i \cos\theta - a_i \sin\theta = z_i,$$

and

$$b_f \sin\theta + c_f \cos\theta - a_f \sin\theta = z_f,$$

which were used to reduce Equation 3.63 to its third equality, always hold for the motion pattern of Figure 3.4.

Substituting Equation 3.63 into Equation 3.56 gives the result

$$\overline{\psi}_f = \psi_i = \overline{\psi}_i, \tag{3.64}$$

proving that the initial uniform free-stream state is preserved for the flux integration scheme expressed in Equation 3.57 for the motion pattern of Figure 3.4.

For the boundary motion pattern of Figure 3.5, substituting Equations 3.59, 3.60, 3.61, and 3.62 into Equation 3.58 yields

$$\begin{aligned} F &= -(t_f - t_i)\psi_i v \left[\left(\frac{a_i + a_f - d_i - d_f}{2} \right) \sin\theta + \left(\frac{b_i + b_f - c_i - c_f}{2} \right) \cos\theta \right] \\ &= (t_f - t_i)\psi_i v \left(\frac{z_i + z_f}{2} \right) \\ &= \psi_i (V_f - V_i). \end{aligned} \tag{3.65}$$

where all symbols are as defined above or as indicated in Figure 3.5.

Substituting Equation 3.65 into Equation 3.56 again gives the result of Equation 3.64, proving that the initial uniform free-stream state is preserved by the flux integration scheme of Equation 3.57 also for the motion pattern of Figure 3.5.

Note that the use of the variable ψ in the above proofs is useful but not necessary, since the invariance of an initially-uniform, passively-transported ψ is an immediate consequence of the result

$$\int_{t_i}^{t_f} \sum_{faces} v_{\perp r} S_f = V_f - V_i \tag{3.66}$$

which is simpler to prove. Also note that the invariance of the state vector $\vec{U}$ in The System of Euler Equations easily follows from the invariance of the generic ψ used in the above proofs, and therefore the satisfaction of The DGLCs proved above also applies to the governing equations used in this work.

While the flux integration scheme given by Equation 3.57 is adequate for the two exclusive boundary motion patterns admissible with cell-merging provided boundaries remain planar, it would not suffice for arbitrarily-curved boundaries moving in arbitrary directions, as described above and shown for the two examples of Figures 3.6 and 3.7. Indeed, these two examples and others like them imply that *it is impossible to satisfy The GCLs with a conservative computational scheme unless the discrete quadratures used to compute or estimate the discrete cell volumes and the discrete time-averages of cell-face areas evaluate to the exact volumes or time-averaged areas.* Thus, for curved boundaries, the integration scheme must be selected to match the functional representation of the local boundary geometry. Such integration schemes can easily be found for both face areas and cell volumes for any polynomial or sinusoidal representation of a boundary, but with a computational effort that increases with the polynomial degree or number of terms in the interpolant.

For computational schemes that have orders of accuracy less than or equal to 2, as is the case in this work, there is little incentive to use a boundary representation of higher degree than a piecewise-planar one [6]. The major attraction of the piecewise-planar representation is that it minimizes the computational effort (required to compute geometric quantities and their time averages). Indeed, because it requires geometric data only at the beginning and end of a time-step to sufficiently satisfy all the accuracy and consistency requirements of a numerical scheme (including The DGCLs), the piecewise-planar representation allows the number of global, whole-grid boundary-cell intersection calculations to be reduced to the absolute minimum of one global calculation per time-step for all time-steps after the first. It also allows the number of intersection configuration sets that require storage to be reduced to the absolute minimum of two sets (for each time-step).

Another consequence of the important conclusion reached in the paragraph immediate before the preceding one is that if a boundary is represented by piecewise-planar segments (that are independent of the intersection of the boundary with the grid, in the sense described above), then the global time-step must be subdivided locally for each composite cell, so as to enable a separate time-averaging calculation to be performed for each piecewise-planar segment that travels across the "Cartesian" faces of the composite cell. This is necessary because the quadrature formulae for time-averaging the face areas depend on the coefficients of the equation of the boundary segment moving through the composite cell. Of course, it is possible that

[6]As mentioned above in this chapter, the piecewise-planar boundary representation is here taken to mean that the boundary is represented by a string of planar segments that are independent of the intersection of the boundary with the Cartesian grid, not the planar segments that result from connecting the intersection points of an arbitrarily-shaped boundary with the Cartesian grid. The specific implementation of the piecewise-planar representation adopted in this work is described in Chapter V.

only a single boundary segment intersects a composite cell during a time-step, and in this case, no subdivision of the time-averaging calculation in that composite cell is needed.

The preceding paragraph described the need for the local subdivision of the time-step if more than one boundary segment passes through the “Cartesian” faces of a composite cell during a global time-step. This paragraph describes this subdivision process, which is hereinbelow called **the sub-stepping process**, for the case of the piecewise-planar boundary representation adopted in this work. The specific process for this case is depicted in Figure 3.8. In this case, the time at which each end-point (or vertex) of a segment reaches the “Cartesian” faces of the composite cell is determined from the velocity of that end-point (or vertex), which is treated as constant over the entire time-step (and each sub-time-step). The calculation can most rapidly be executed by linear interpolation, but higher-order approximations are also possible if derivatives of the vertex locations higher than the first are also available. In the two-dimensional situation, only line-line intersection computations are involved; in the three-dimensional situation, only plane-plane intersection computations are involved. The “crossing” times for the segment end-points are then used to calculate the time-averaged area of the intersected face of the composite cell in accordance with the following formula:

$$\overline{S}_f = \frac{\sum_{s=1}^{nSubSteps} \frac{(S_i+S_f)_s}{2} (\Delta t)_s}{\sum_{s=1}^{nSubSteps} (\Delta t)_s = \Delta t}, \tag{3.67}$$

where the subscript s denotes association with a sub-step, $nSubSteps$ is the total number of sub-steps, $\overline{S}_f$ denotes the time-averaged face area (used in computing the flux residual in Equation 3.43, for example), and $(S_i)_s$ and $(S_f)_s$ refer respectively to the values of the face area at the start and end of sub-step s.

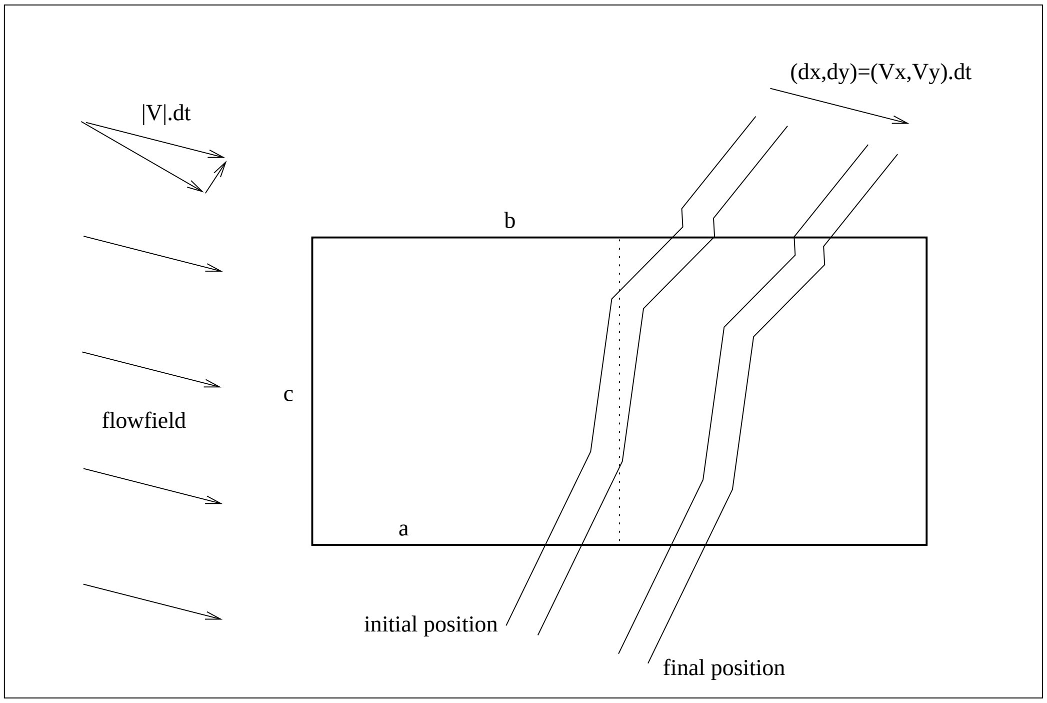

Figure 3.8: Enforcement of The GCLs for a boundary comprised of straight-edge (piecewise-linear) segments, using a sub-stepping procedure.

The formula of Equation 3.67 gives the exact time-averages of the areas of intersected faces of composite cells for piecewise-planar boundaries that are moving at a constant speed. For satisfaction of The GCLs, as explained above, the cell volumes for a piecewise-planar boundary representation must also be computed exactly using the given piecewise-planar representation of the boundary. As explained further in Chapter VI, this volume (or area) calculation is accomplished in this work by subdividing each composite cell into triangles. The formula in Equation 3.67 is therefore appropriate for computational schemes that are up to second-order accurate in time and space for boundaries that are moving with non-constant velocities, and gives exact results for boundaries moving with constant velocity.

For a boundary that consists of any number of piecewise-linear segments, the motion pattern during each sub-step is still either of the type shown in Figure 3.4, or of the type shown in Figure 3.5. Therefore, satisfaction of The GCLs for each sub-step can readily be established for such a boundary in the same manner as that demonstrated above for a single planar boundary. The only additional complication in establishing this result for a multi-segment boundary geometry (compared to the case for a single-segment boundary geometry) is that the normal and tangential field components, $v_{\perp nr}$ and $v_{\perp tr}$ need not be the same on the two "Cartesian" faces of the composite cell that are intersected by the boundary segments (because of differences in the boundary angle at the two intersection points, as shown, for example, in Figure 3.8). Since each sub-step satisfies The GCLs, the overall motion step must also satisfy The GCLs. Results verifying and validating the preservation of the free-stream with piecewise-planar boundary representations are shown in Chapter VIII.

It should be emphasized that the sub-stepping procedure and the time-averaging calculations are carried out on a purely local basis (that is, composite-cell by composite-cell), and that the time-averages of face areas will be identical from both sides of a face (that is, for both of the two cells that share a face). The sub-stepping technique also allows the automatic handling of changes in the total number of "internal" or "non-Cartesian" faces of a composite cell [7].

The piecewise-planar boundary representation is used in the overwhelming majority of current grid-generation implementations. However, the major difference in the use of the representation adopted in this work with Cartesian-Quadtree grids is that the geometric scales of the composite cells are not required to match the geometric scales of the boundary segments that intersect them as they do in almost all other grid-generation techniques. Indeed, as mentioned above, and as explained

[7]These are the faces of a composite cell that are formed exclusively from the boundary geometry "cut" by the composite cell.

further in Chapter V, the piecewise planar segments used to represent a boundary are independent of the intersections of that boundary with the grid.

In the three-dimensional analog of the technique described above, the boundaries would be represented by polygonal facets that are, again, independent of the intersection of the boundary with the computational cells. The polygonal facets should preferably be triangles, so that the facets are guaranteed to remain planar without any additional constraints on the motions of the vertices of the facets. These facets will move in composite cells formed from the cubic cells of the Cartesian-Octree grid, and again only two motion patterns are possible. The GCLs will again be satisfied because the cell volumes and time-averages of face areas will again be computed exactly from the vertex locations of the facets for boundaries moving with constant velocity. For boundaries moving with arbitrary velocity, the computation will again become second-order accurate. A local sub-stepping procedure must again be used in each composite cell whenever a vertex or edge of a boundary facet enters or leaves a composite cell.

3.5 Boundary Conditions and Procedures

3.5.1 The Role of Boundary Conditions and Treatments

The Finite-Difference, Finite-Volume, and Finite-Element Methods all require the Computational Region to be of finite extent and to be enclosed within finite, specified boundary surfaces. Obtaining a solution to The System of Euler Equations in finite domains is analytically and computationally always formulated either as a Boundary-Value Problem or as an Initial-Boundary-Value Problem, as explained further in Section 3.6. The need for special, discretized treatments of boundary conditions in computational schemes for solution of The System of Euler Equations therefore arises naturally in all three discretization methods, in the same way that the need for analytic boundary conditions arises for the corresponding analytic solutions of Boundary-Value and Initial-Boundary-Value Problems in bounded domains.

In general, the required boundary conditions for The System of Euler Equations are imposed either: (i) to simulate the effects on the Computational Region of bounding surfaces that partially or completely obstruct the flow; or, (ii) to enable the transmittal of a-priori-known disturbances or variations in the flowfield to the interior of the Computational Region. The second case includes boundary conditions that are imposed to approximate analytic far-field conditions onto the finite Computational Region. In both cases, the boundary-condition effects are transmitted to the computational scheme operated within the Computational Region.

Assuming the existence and uniqueness of analytical solutions for The System of Euler Equations, the importance of correct treatment of analytic boundary conditions can be appreciated by observing that the only differences between one solution and another on a given domain can arise from differences in the initial and boundary conditions. An analogous situation holds for discrete solutions and discrete boundary conditions.

The specific treatment of boundary conditions for a given application is important because it is a major determinant of the behavior of the numerical solution. Correct treatment of boundary conditions is necessary to maintain the well-posedness, accuracy, and stability of the solution, while correct treatment of boundary conditions together with matching of this treatment to the interior scheme is necessary to extend the accuracy, stability, and convergence rate of the interior scheme to the overall computational scheme.

3.5.2 The Mathematical Bases of Boundary Condition Treatments

As explained in Appendix G, the correct treatment of the boundary conditions for The System of Euler Equations is strongly based on the Theory of Characteristics. More specifically, the Theory of Characteristics provides a complete and rigorous mathematical basis for the linearized treatment of these boundary conditions. As explained in Appendix G, the Theory of Characteristics not only specifies the number of discretized Physical Boundary Conditions and the number of discretized Numerical Boundary Conditions [8] that are required at each point in the boundary (or correspondingly, at each boundary face in a Finite-Volume Method), but it also identifies the formulation of the variables in the boundary treatment (namely, the Riemann Invariants) that must be specified or extracted from the interior solution. As shown in Appendix G, this knowledge allows the identification of the specific boundary conditions and the appropriate treatments for any possible flow state or condition at a boundary.

As explained below and in Appendix G, the specific implementation of the Boundary Condition treatments adopted in this work accords with the fundamental relevant requirements of the Theory of Characteristics.

3.5.3 Techniques of Boundary Condition Treatments

While the Theory of Characteristics provides the theoretical bases and constraints for correct treatment of the Boundary Conditions for The System of Euler Equations, there is considerable flexibility in choosing a specific numerical technique to implement the treatment, and in choosing the variables in which to formulate the Boundary Conditions. The main possible alternatives for these choices are discussed

[8] Appendix G explains and defines the terms Physical Boundary Conditions and Numerical Boundary Conditions.

in Appendix H.

3.5.4 Implementation Details

The implementation technique selected in this work falls in the category of Variable Extrapolation Boundary Condition Techniques or Treatments, which are described in Appendix H. The choice was also made to implement the treatment of inflow, outflow, and symmetry-plane boundaries using a ghost-cell approach, but to implement the treatment of impermeable boundaries using a direct flux-evaluation approach. As evident in the preceding sub-sections of this section and the Appendices referenced there, it is common practice to use boundary conditions of only the Dirichlet type for The Euler Equations, and this common practice was also followed in this work.

Since the grid used in this work is of the stationary Cartesian-Quadtree type (which is defined and described in Chapters IV and VI), the outer boundaries of the Computational Region are always stationary. In this work, each of these outer boundaries is allowed to have the inflow, outflow, impermeable, symmetry, or cyclic condition type. Boundaries that lie within the Quadtree Region (that is, boundaries representing the surfaces of obstacles or bodies) may be stationary or moving, and each of them may have the inflow, outflow, impermeable, or symmetry condition type. The implementation details for these boundary-condition types are given below in the form used for moving boundaries; they specialize trivially to the form used for stationary boundaries.

The specific combination of required Numerical and Physical Boundary Conditions and the specific values of these boundary conditions are determined or evaluated independently for each boundary face or boundary cell of each boundary in the Computational Region. This independent evaluation ensures that any variations in the flow conditions along a boundary, which may be due, for example, to variations arising in the interior solution, are accounted for correctly, and in accordance with the requirements of the type of that boundary. Even more importantly, this independent evaluation ensures that even if the type of boundary condition changes locally, for example, from the inflow to the outflow type, that the change will be correctly reflected in the boundary-condition treatment.

3.5.4.1 Treatment of Inflow and Outflow Boundaries

Following the standard ghost-cell approach for the inflow and outflow boundary condition types, each computational cell that is attached to a boundary of the inflow or outflow type is assigned its own "ghost" state vector. The "ghost" state vector for each such boundary cell is determined by applying the Variable Extrapolation Technique described in Appendix H for each required Numerical Boundary Condition, and by applying direct value-specification for each required Physical Boundary Condition, in accordance with the specific requirements of the particular simulation being computed, as described in more detail below. Once the state in a ghost-cell is determined, the interior scheme is applied in the usual manner to compute the flux between the "ghost" state and the opposing "interior" state, that is, to compute the flux across the corresponding boundary cell-face. This flux computation is all that is required to impose the corresponding boundary condition.

For implementations in which the grid does not change during a computation, the procedure described in the preceding paragraph is most efficiently implemented by requiring all the ghost-cell states to be stored in arrays and to be treated in an identical manner to the interior-cell states, except that the ghost states are computed using boundary procedures rather than being updated by the interior scheme. In this work, a "virtual" ghost cell is used that allows only one ghost state vector to be computed and stored at a time while the boundary flux for a boundary cell is being evaluated. Thus, no global storage is used for simultaneous storage of all the boundary ghost-cell states. This approach is more suitable in this work since the boundary cells may change arbitrarily during a computation (in number, and in shape) due to adaptation of the grid.

The procedure followed for inflow and outflow boundaries is to determine the applicable value of the Primitive-Variable vector in the ghost-cell, and then to compute the corresponding state vector from that, that is, by using the relation $\vec{U} = \vec{U}(\vec{P})$. Once the state vector is available, the boundary flux is obtained by using the same flux function that is applied for interior cell-faces.

The Primitive-Variable vector of a ghost cell for the inflow or outflow boundary types is computed as described below, where the subscript s refers to a specified value (corresponding to a Physical Boundary Condition), and where the subscript e refers to an extrapolated value (corresponding to a Numerical Boundary Condition). Specified boundary-condition values are provided as part of the input which forms the problem specification, while extrapolated values are computed using the formula:

$$\phi_e = \phi_c + (\nabla\phi)_c \cdot (\vec{r}_e - \vec{r}_c), \tag{3.68}$$

where ϕ_c and ϕ_e are respectively the values of the variable ϕ at the centroid of the boundary cell, and at the extrapolation point in the boundary (taken as the centroid of the boundary face being treated), where $\vec{r}_c$ and $\vec{r}_e$ are respectively the position vectors of the centroid of the boundary cell and of the extrapolation point, and where $(\nabla\phi)_c$ is the gradient associated with the current boundary cell of the variable

ϕ. For computations which are first-order-accurate in space, $(\nabla\phi)_c$ is set to zero for all variables. As indicated in Equation 3.68, no gradient limiting is applied in the boundary-value extrapolation for computations that are second-order-accurate in space.

With all flow speeds taken relative to the boundary, the specific formulations used for the various possible inflow and outflow boundary conditions are as follows:

1. Supersonic Inflow Boundary Condition: Whether stationary or moving, the five required physical boundary conditions are specified by fully specifying the primitive variable vector:

$$\vec{P} = \left(\rho_s, u_{\perp s}, u_{\|s}, p_s\right)^T.$$

2. Supersonic Outflow Boundary Condition: Whether stationary or moving, the five required numerical boundary conditions are computed by fully extrapolating the primitive variable vector from the interior of the Computational Region:

$$\vec{P} = \left(\rho_e, u_{\perp e}, u_{\|e}, p_e\right)^T.$$

3. Subsonic Outflow Boundary Condition: The pressure is specified as the required (given) Physical Boundary Condition, and the remaining three primitive variables are computed by extrapolation from the interior of the Computational Region:

$$\vec{P} = \left(\rho_e, u_{\perp e}, u_{\|e}, p_s\right)^T.$$

4. Subsonic Inflow Boundary Condition: The two components of the velocity vector and the density are specified as the Physical Boundary Conditions, while

the pressure is extrapolated from the interior of the Computational Region:

$$\vec{P} = \left(\rho_s, u_{\perp s}, u_{||s}, p_e\right)^T.$$

The three-dimensional analogs of the above formulae are respectively given by the following formulae:

1. Supersonic Inflow Boundary Condition:

$$\vec{P} = \left(\rho_s, u_{\perp s}, u_{||1s}, u_{||2s}, p_s\right)^T.$$

2. Supersonic Outflow Boundary Condition:

$$\vec{P} = \left(\rho_e, u_{\perp e}, u_{||1e}, u_{||2e}, p_e\right)^T.$$

3. Subsonic Outflow Boundary Condition:

$$\vec{P} = \left(\rho_e, u_{\perp e}, u_{||1e}, u_{||2e}, p_s\right)^T.$$

4. Subsonic Inflow Boundary Condition:

$$\vec{P} = \left(\rho_s, u_{\perp s}, u_{||1s}, u_{||2s}, p_e\right)^T.$$

where the subscripts 1 and 2 that are applied to the components of the velocity that are parallel to the local boundary face have the obvious meanings, and all other symbols are as defined above.

3.5.4.2 Treatment of Impermeable Boundaries

A ghost-cell approach is not used at impermeable boundaries. Instead, the pressure is extrapolated from the interior of the Computational Region, and the flux is evaluated explicitly and directly from the formula

$$\vec{\tilde{H}} = \begin{pmatrix} 0 \\ p_e n_x \\ p_e n_y \\ p_e(u_b + v_b) \end{pmatrix}, \tag{3.69}$$

which has the three-dimensional analog

$$\vec{\tilde{H}} = \begin{pmatrix} 0 \\ p_e n_x \\ p_e n_y \\ p_e n_z \\ p_e(u_b + v_b + w_b) \end{pmatrix}, \tag{3.70}$$

where u_b, v_b, and w_b are the standard Cartesian components of the local *absolute* velocity of the impermeable boundary, and all other symbols are as defined above. The pressure is extrapolated following either the regular first-order-accurate evaluation (given by $p_e = p_c$, where p_c is the pressure in the boundary cell), or the second-order accurate evaluation of Equation 3.68, although many improvements upon the latter (for example, as in [94]) can be readily implemented.

The local velocity of the boundary must be computed exactly from the displacement of the boundary over the given time-step in order to ensure that The Geometric Conservation Laws (see Section 3.4 above) are satisfied. The flux formulae given above are applicable even if the boundary is deformable; the only difference in the latter case is that the motion and trajectory of the boundary are affected by the pressure distribution around the boundary, and should strictly be computed implicitly, although this is not done in this work. This issue and others related to it are discussed further in Chapter V.

The flux formulae of Equations 3.69 and 3.70 reflect the fact that an impermeable boundary provides a source term for the Momentum Conservation Equation that does not depend on the speed of the boundary. These formulae also reflect the fact that an impermeable boundary provides a source term for the Energy Conservation Equation only if there is boundary motion, that is, only if the impermeable boundary does pressure work. Whether a solid boundary is stationary or moving, it does not provide any source term for the Mass Conservation Equation, since by definition, no mass may cross an impermeable boundary; otherwise, the boundary becomes an inflow boundary and the pressure work is automatically provided by the absolute velocity of the in-flowing fluid. In accord with the explanations given in Appendix G, the application of the condition $\vec{v}_{\perp r} = \vec{0}$ at an impermeable boundary may be regarded as the Physical Boundary Condition that is imposed in association with the single negative characteristic eigenvalue at an impermeable boundary face.

The choice of treatment of impermeable boundaries made above has the appeal of focusing the associated errors into the prediction of the value of the pressure at such boundaries. Other than for errors or approximations in this prediction, the flux is computed directly using an evaluation that is precisely correct at the discrete level from both the mathematical and physical points of view. The prediction of boundary pressure may be improved by solving the discretization of the projection of the Momentum Conservation Equation in a direction perpendicular to the wall [305] in accordance with the equation

$$\frac{\vec{v}_{||} \cdot \vec{v}_{||}}{R_b} = \frac{1}{\rho}\frac{\partial p}{\partial n},$$

where $\vec{v}_{||}$ is the velocity vector tangential to the boundary, R_b is the local radius of curvature of the boundary, n is the coordinate position along the local normal to the boundary, $\vec{n}$, and all other symbols are as defined above. One important appeal of this correction to the prediction of the pressure value at a boundary is that the resulting discretization corresponds closely to a one-sided discretization of the compatibility relation at the boundary [170].

Since the symmetry-plane boundary condition for inviscid flow is identical to the impermeable boundary condition for inviscid flow, the former is treated in an identical manner to the latter in this work. This is in accord with common convention for schemes for The System of Euler Equations.

Periodic or Cyclic Boundary Conditions do not require boundary condition treatments in the sense described above: they are implemented by data-structural operations that link each cell at one edge of the Quadtree Region with the "cyclically-corresponding" cell on the opposing edge of the Quadtree Region. No analytic requirements apply to this treatment technique. No Periodic or Cyclic Boundary Conditions are implemented for boundaries that lie within the Quadtree Region. In order to ensure precise implementation of the Cyclic Boundary Condition, the grid generation and adaptation procedures are constrained to link all cell refinements and coarsenings on any cyclic boundary with identical operations on the opposing boundary.

The imposition of special boundary conditions, such as the Kutta Condition, as needed with, for example, the Potential Equations, in connection with problems involving lift or separation is usually unnecessary for The System of Euler Equations, as explained in Chapter II.

3.5.4.3 Order of Accuracy

Since the boundary procedures in this work are perfectly matched with the interior scheme, and since the extrapolations used in them have the same order of accuracy as the interior scheme, the boundary condition treatments in this work have the same order of accuracy as the interior scheme for planar boundaries. Specifically, for first-order-accurate computations, the boundary condition treatments are first-order-accurate in space and time; for second-order-accurate computations, the boundary condition treatments for planar boundaries are second-order-accurate in space and time.

3.5.4.4 Placement and Identification of Boundary Conditions

An important implementation detail is how the boundary-condition types are determined during a computation. The most common practice is to assume that throughout a computation, each boundary retains the same boundary-condition type originally assigned to it in the problem definition. However, this may result in an incorrect boundary condition treatment if the flowfield changes in the vicinity of a boundary sufficiently to alter the boundary type. The likelihood of this event is greater with unsteady computations in general. As briefly outlined above, in this work the type of boundary treatment is allowed to vary arbitrarily along each boundary of the Computational Region, and to vary with the evolution of the solution. This is achieved by using the flow state in each boundary cell to determine the boundary treatment applied to the boundary face(s) of that cell on a purely local basis. For example, if the flow state in a boundary cell changes from supersonic inflow to subsonic inflow, the treatment of the boundary face(s) of that cell will change accordingly.

Another important implementation detail is the relative location of boundaries on which far-field conditions are applied, in order to ensure the correctness and accuracy of such boundary conditions. For lifting airfoils, the typical practice is to locate boundaries at least ten chords away from the airfoil, although a circulation condition [144], usually derived from perturbation theory, is usually imposed in addition, into the outermost layer or ring of cells as a farfield boundary-condition correction to improve the analytic approximation. Semi-analytic formulations [391] have also been used with remarkable results. With a Quadtree-Adaptive grid, the additional number of cells that would be required to relocate the boundary from about ten chords away from the airfoil to several hundred chords or so away from the airfoil is relatively small, and this approach is adopted for some of the test cases shown in Chapter VIII, although it is recognized that the effectiveness of filling the increased distance with cells that are comparatively large is probably far smaller than the effectiveness of using a far-field correction technique.

It is often the case with computations of transient problems, especially those involving moving boundaries, that the size of the Computational Region that will be suitable for the entire period of time over which the solution is required cannot be determined a-priori. In order to allow the Computational Region to expand during a computation, an option is implemented in this work to re-root the Quadtree grid as an immediate subnode of a newly created root node, thus making the original Quadtree one of the four immediate subnodes of a new root, and expanding the Computational Region by a factor of four. This procedure is described further in Chapter VI, which also describes the Quadtree-based grid-generation procedure in detail. This expansion may be triggered by a user signal (namely, the insertion of a string in a file that is read at the end of every time-step of the calculation), or by detection of the approach of a refinement zone that would normally contain a discontinuous flow feature (such as a shock wave) toward a boundary of the Quadtree Region.

3.6 Specification of Initial Conditions

3.6.1 Analytical Requirements

Solutions of the steady-state Euler Equations fall in the class of Boundary-Value Problems, and therefore do not analytically require Initial Conditions. Numerical solution algorithms for such problems, however, may require Numerical Initial Conditions to start the solution algorithm. In such cases, the restrictions on the Initial Conditions that may be specified are motivated solely by numerical and practical considerations.

Solutions of the full (or time-dependent) Euler Equations classify either as Initial-Value Problems or as Initial-Boundary-Value Problems, depending on whether the solution domain is respectively unbounded or bounded. For both cases, Initial Conditions are analytically required, but no comprehensive theory exists on the allowable values. Instead, a few universal intrinsic analytical constraints are known, and used for general guidance.

One of these fundamental constraints is that Initial Conditions may not be specified along Characteristic Surfaces or Curves. If Initial Conditions are given along Characteristic Surfaces or Curves, then they must satisfy the corresponding ordinary differential equations along those surfaces or curves. Otherwise, no solution exists with those Initial Conditions. If the Initial Conditions satisfy the corresponding ordinary differential equations along those surfaces or curves, then the solution will be unique on those surfaces or curves but nowhere else, making the problem ill-posed. This constraint is of little practical impact for numerical solution schemes because

Initial Conditions are usually specified by giving the spatial distribution over the entire spatial domain or Computational Region at a single value of time.

Another fundamental general analytical requirement is that the Initial Conditions must not conflict with the governing equations nor with any explicit or implicit constitutive relations. For example, discontinuous initial data may not be specified for analytic solutions if the differential formulation of the governing equations is being used; for analytic solutions with an integral formulation, the differentiability constraint may be replaced by the integrability constraint which only requires the measure of discontinuity points to be zero in the space of the solution domain. A conflict with the governing equations can also arise even if all the smoothness requirements are satisfied. For example, for time-dependent incompressible flows, a velocity field which does not everywhere satisfy the incompressible form of the Mass Conservation Equation, $\nabla \cdot \vec{v} = 0$, cannot be specified as part of a valid Initial Condition.

For Boundary-Value Problems that are solved as the infinite-time-limit of Initial-Value or Initial-Boundary-Value Problems, the restrictions on the allowable Initial Conditions may be relaxed all the way to the restrictions for Boundary-Value Problems.

3.6.2 Physical Requirements

In addition to the analytical restrictions described above for Initial-Value and Initial-Boundary-Value Problems, the physical meaning of the variables in the specific problem being studied usually imposes some additional restrictions on the Initial Conditions that may be validly selected or specified. The usual physical restriction is that all Initial Conditions must be "physically realizable". For The System of

Euler Equations, physical realizability requires the Initial Conditions to be Physically Admissible (as defined in Section 3.2) at every point in the solution domain. Initial Conditions in which, for example, membranes separating different states of fluid "instantaneously" vanish and in which the Initial Conditions are not compatible with the Boundary Conditions are still considered physically realizable. For numerical solution schemes of The Euler Equations, the discrete analog of the restrictions described above is what must be respected; namely, that every discrete state vector must be Physically Admissible.

Physical Admissibility includes the "internal" consistency of the initial data (that is, that all the applicable thermodynamic relations governing the values of interdependent thermodynamic properties must be simultaneously satisfied). For solutions of the steady-state Euler Equations or for solutions of the time-dependent Euler Equations that are computed as the infinite-time-limit of a transient solution, the internal-consistency requirement can in practice be relaxed somewhat provided the solution algorithm converges.

The introduction of inconsistence as described in the preceding paragraph is often a convenient or necessary means of avoiding arithmetic overflow or underflow. An example where such a situation occurs is with computations of flows whose Mach Number approaches infinity. With all symbols as defined above, the density ratio across a shock, given by $\frac{\rho_f}{\rho_i} = \frac{(\gamma+1)M^2}{(\gamma-1)M^2+2}$, remains bounded from above by $\frac{(\gamma+1)}{(\gamma-1)}$ as the Mach Number increases, while the pressure ratio across the shock, given by $\frac{p_f}{p_i} = 1 + \frac{2\gamma}{(\gamma+1)}(M^2 - 1)$ is unbounded, but is in practice set or limited to a high value, such as 1.0e+15, in order to enable the computation to be executed, even though this leads to inconsistence with the density ratio. Such bounding of values was indeed used in this work to handle flows with very high Mach Number or flows that generate vacua, mostly for validating and testing the various Riemann solvers used in this work. It is not clear whether the solution from such a computation can be considered valid, but it appears that provided such internal inconsistencies are quantified, isolated, and treated in a careful manner that avoids the creation of non-physical states, their overall effect on the solution is mostly localized and acceptable, and good results can be obtained for flows that would otherwise be impossible to numerically compute.

3.7 Numerical Properties of the Discretization and Discrete Solution Scheme

A computational scheme cannot be considered well-understood, nor can it be used confidently or reliably without an adequate quantitative characterization of its fundamental properties of **consistency**, **stability**, **convergence**, and **well-posedeness**. A general review of the definitions, meanings, and implications of these properties in relation to discretizations and numerical-solution schemes, and of the important inter-relations between them is beyond the scope of this dissertation. Such reviews are abundantly available in many texts on numerical methods in general, for example [148], [333], or [300], and in many texts on numerical methods for Computational Fluid Dynamics in particular, for example [12], [222], [169], [170], or [134].

Most of the texts just cited, especially [169], and others like them also explain how the order of accuracy of a computational scheme (and hence its consistency) can be determined, and describe in detail the alternative techniques of computationally or analytically establishing some of the stability properties of such a scheme. These texts also explain how the stability analysis for a nonlinear scheme is typically confined to the linearized version of the scheme, and how most theorems connecting the different basic properties of a computational scheme (such as its stability, consistency, and convergence), such as the Lax Equivalence Theorem [300], are strictly valid only for linear schemes, or for the linearized versions of nonlinear schemes.

As far as the consistency (or equivalently, the spatial and temporal orders of accuracy), stability, convergence, and well-posedness of the specific discretization formulation and the specific discrete solution scheme adopted and developed in this work, the most relevant considerations are mentioned or discussed above in this chapter, mainly in Sections 3.2, 3.3, 3.5, and 3.6. The basic numerical properties of the specific computational scheme developed and adopted in this work follow directly from its classification into the family of High-Resolution TVD Schemes for The System of Euler Equations. A review of the properties of such schemes is also beyond the scope of this dissertation, but can be found, for example, in [222], [169], and [170], especially the latter reference, which contains one of the most extensive and up-to-date treatments of the numerical properties of High-Resolution Schemes, especially in the context of the Finite-Volume Method.

The references just cited also define the **monotonicity** property, and the **Total-Variation Diminishing (TVD)** property (mentioned in the preceding paragraph and in Section 3.3), and discuss their role in the construction of "satisfactory" computational schemes for The Euler and Navier-Stokes Equations. They also discuss the main methods of accelerating the convergence of steady-state solutions, including the **pre-conditioning** approach and the **multi-grid** method, that are directly applicable to the scheme developed and adopted in this work.

3.8 Incorporation of the Discrete Solution Algorithm into the Overall Algorithm

As outlined in Chapter I, the overall methodology developed in this work employs five different and largely separate algorithms or modules, namely: (i) a geometric-definition and boundary-dynamics algorithm; (ii) a grid-generation algorithm; (iii) a discrete-solution algorithm; (iv) a solution-adaptation algorithm; and, (v) a cell-merging algorithm.

The main functions of each of these five algorithms were outlined in Chapter I. In addition, each of these algorithms is described in detail in one of the chapters of this dissertation: the geometric-definition and boundary-dynamics algorithm is described in Chapter V; the grid-generation algorithm and the solution-adaptation algorithm are both described in Chapter VI; the cell-merging algorithm is described in Chapter VII; and, the discrete-solution algorithm is described in this chapter. In particular, preceding sections of this chapter describe the individual components of the discrete solution scheme adopted or developed in this work, as well as the most important properties of this scheme, while this section provides an algorithmic overview of this scheme. As explained above in this chapter, especially in Sections 3.1 and 3.3, because the spatial and temporal discretizations are here separated, and because the time-integration scheme is here chosen to be of the explicit type, the discrete solution algorithm in itself is particularly simple in this work.

The specific purposes of this section are as follows: (i) to identify and describe the main steps of the overall solution algorithm developed in this work, including the main steps of the discrete-solution algorithm, and in doing so to further clarify the role of each of the five algorithms of the methodology developed in this work; and, (ii) to identify how the discrete-solution algorithm interacts and integrates with the four other algorithms.

The overall solution algorithm followed for the most general application of the methodology developed in this work (namely, a problem with arbitrarily moving and deforming boundaries), with the steps of the discrete solution scheme marked with the symbol $\star$, is as follows:

1. **Initialize all algorithm parameters.** This includes initializing all global variables; all variables related to timing of process durations and to monitoring and tracking of computing resources; all discrete-solution-scheme parameters; all geometry-related and grid-generation-related tolerances and variables; all solution-adaptation variables and parameters; all geometric-adaptation variables and parameters; all cell-merging parameters; and all parameters and variables that are specific to the test case being simulated. Also included in this step is the appropriate setting and initialization of all input and output devices and files;

2. **Input the initial boundary geometries, and transform them into the internal representations adopted in this work.** The data-structures used for storing the internal representation of every "internal" boundary in the test case being simulated are allocated as part of this procedure or step. Chapter V describes the details of this internal representation, and of the data-structures used in relation to it. This step also includes input of the motion model for every boundary present, and allocation of the data-structures that are used to retain and manipulate these motion models;

3. **Generate a preliminary, uniformly-refined Cartesian grid.** This grid covers the Computational Region, and its intersections with the the initial

geometries of all boundaries present in the simulation is computed as part of this generation process. As outlined in Chapter I, and as explained in detail in Chapter VI, this Cartesian grid is of the Quadtree type. Chapter VI describes in detail how this grid is generated and discusses all its important properties, while Chapter V describes how the intersection of the grid with the boundary geometries is computed;

4. **Adapt the grid generated in the preceding step to the initial geometry of all boundaries that are present in the simulation.** This is done as described in Chapter VI, and results in a grid that has a refinement that varies around boundaries in such a manner as to resolve all the geometric features of those boundaries to a specified accuracy;

5. $\star$ **Initialize the solution variables or the State Vectors.** As explained above in this chapter, these here are chosen to be the Conserved Variable Vectors, and they are initialized to the prescribed initial conditions in all the computational cells (of the Cartesian Quadtree grid);

6. **Repeat the following set of steps, once for each time step (or for each solution-update cycle), until an appropriate termination criterion is satisfied** [9]:

   (a) **Determine the complete geometric definition of all boundaries in the Computational Region at the beginning and end of the current time-step.** The geometric definition at the beginning of the current time-step is either that at the end of the preceding time-step, or

[9]The termination criterion here may either be the attainment of a pre-determined total integration time, or the advancement by a pre-determined number of time steps, or the attainment of a pre-determined convergence criterion for a steady-state computation.

that given by the initial boundary geometry. The geometric definition at the end of the current time-step is determined from the motion during the current time-step. The motion of any boundary or boundary segment during the current time-step is determined in accordance with the specified motion model for that boundary or boundary segment, as described in Chapter V. Depending on the motion model, determining the motion may require computation of the forces and moments applied to the boundaries or boundary segments involved, solution of the equations of motion for any flexible- or rigid-body dynamics problems involved, and computation of the trajectories of those boundaries or boundary segments. The manner in which this is done is described in detail in Chapter V;

(b) **Adapt the Cartesian Quadtree grid to the updated boundary geometry.** This is done so that the resulting grid satisfies the grid-refinement requirements of the boundary geometries at both the beginning and end of the current time-step. This adaptation is performed as described in detail in Chapter VI;

(c) **Determine the complete geometric definition of all the Cartesian Quadtree cells at the beginning and end of the current time-step.** If no boundary motion occurs during the time-step, then the initial and final geometries will be identical for all these cells. The manner in which the cell geometries are determined from the boundary locations (that is, the manner in which the grid-boundary intersection problem is computed) is described in detail in Chapters V and VI;

(d) **Conditionally invoke the solution-adaptation algorithm to visit all Cartesian Quadtree cells, and to either refine, coarsen, or**

leave each of these cells unchanged. This procedure is described in detail in Chapter VI. For steady-state computations, this step is invoked only with a pre-determined frequency of solution-update cycles (and may therefore be skipped for some of the solution-update cycles);

(e) **Invoke the cell-merging algorithm to form an appropriate set of composite cells from the Cartesian Quadtree cells.** As outlined in Chapter I, and as explained in detail in Chapter VII, the composite cells are created by combining one or more of the Cartesian Quadtree cells in such a manner that none of the composite cells changes its topologic type as a moving boundary travels across it, and each of the composite cells can therefore be treated as either a regular deforming cell, or as a regular, fixed-geometry cell. As explained in Chapter VII, the composite cells also eliminate any small cells formed from the intersection of boundaries with Cartesian cells, and as also explained in that chapter, this is an essential function for a grid that is not of the boundary-conformal type. All fluid-dynamic calculations and solution updates are actually performed on the composite cells. The composite cells cover the entire Computational Region, so that every Cartesian Quadtree cell is incorporated into a unique composite cell;

(f) $\star$ **Project the solution variables from each Cartesian Quadtree cell to the composite cell into which the Cartesian cell has been incorporated.** This is done to transfer the solution values at the time corresponding to the beginning of the current time-step from the Cartesian Quadtree cells to the newly-created composite cells;

(g) $\star$ **Perform any required solution reconstruction and gradient lim-**

iting in the composite cells, and compute all flux quadratures on all faces of the composite cells. This is done in accordance with the chosen spatial discretization scheme, with either first-order or second-order spatial accuracy;

(h) ⋆ **Apply the specified boundary conditions.** These boundary conditions are applied to the appropriate faces (or edges) of the composite cells;

(i) ⋆ **Sum the flux quadratures on the faces (or edges) of the composite cells and time-integrate these sums.** The integration in time is performed with either first-order or second-order temporal accuracy, to obtain the updated (or evolved) solution variables in the composite cells. If there are moving boundaries, only the second-order-accurate update procedure is used, taking into account the change in geometry of any deforming composite cells, and in a manner that ensures satisfaction of The DGCLs, as explained in detail in Sections 3.3 and 3.4;

(j) ⋆ **Repeat the preceding three steps if second-order temporal accuracy is being used.** In this case, the first application of these three steps corresponds to the Predictor Step, and the second application of these three steps corresponds to the Corrector Step of the time-integration scheme adopted in this work, as explained in detail in Section 3.3;

(k) ⋆ **Prolong the updated solution variables (or State Vectors) from each composite cell onto the Cartesian Quadtree cells from which the composite cell is composed**. This is done with either first-order or second-order spatial accuracy, in accordance with the chosen spatial discretization scheme;

(l) **Eliminate the composite cells formed for the just-completed solution-update cycle.** As part of this step, all associated data-structures are removed, and their memory "freed";

(m) **Conditionally output all required data.** The typical data output here is related to the solution variables, to the convergence parameters (for steady-state calculations), to the computational-cell-counts, and to the CPU and memory-usage. The output of each category of variables or parameters listed here is invoked with an independently-pre-determined frequency of the solution-update-cycle.

7. **Output all required global parameters and data related to the run, and to the specific test case for which the computation was performed.**

As repeatedly mentioned above in this section, additional details of the operations performed in each of the steps of the overall solution algorithm can be found in an appropriate chapter of this dissertation.

The algorithm as actually implemented in software form differs slightly from the form given above. The main such differences are as follows: (i) in the actual implementation, the solution- and geometry-adaptation steps are each separated into a refinement sweep, and a separate coarsening sweep (rather than having the refinement and coarsening actions combined together and performed during the same single sweep of the cells); (ii) in the actual implementation, the data output steps are distributed over several points in the main loop, rather than being bundled in only two locations as shown in the above form; (iii) in the actual implementation, some of the steps that are given separately in the above form are combined together for greater efficiency; (iv) in the actual implementation, there are additional steps and slight variations from the specific sequence given in the above form, mainly to provide different "running modes" of the solution algorithm, such as the "restart mode" (in which a computation is started from the results of a different computation), and such as the "grid-expansion mode" (in which the Computational Region is expanded to enable continuation of the solution as flow features or moving boundaries approach the outer boundaries of the original Computational Region [10]); and, (v) in the actual implementation, there are additional, minor steps that are not included in the form given above. Despite these differences and other, less important ones, the overall algorithm as given above provides an accurate picture of the corresponding actual software implementation.

As mentioned above, the overall solution algorithm as given above is for the most general type of problem that can be simulated with the methodology developed in this work. The individual steps that can be omitted or bypassed for the special cases of unsteady problems with stationary boundaries, or problems requiring only steady-state computations are self evident. In the actual implementation, either conditional execution or conditional compilation are used for all the steps that can be omitted or bypassed for special cases, in order to keep the run-time for any simulation as low as possible.

[10]The "grid-expansion" algorithm, and its purposes and implementation are explained in more detail in Chapter VI.

CHAPTER IV

Review of Grid Generation and Adaptation: Definitions, Nomenclature, Classification, and Methods

4.1 Overview

This chapter is concerned with grids, and their generation and adaptation. A definition is given of what constitutes a grid, the roles of grid generation and grid adaptation in obtaining a computational solution are identified, and the processes of grid generation and adaptation are described. Alternative methods of generation and adaptation are reviewed, classified, and compared. Special emphasis in the comparison is placed on computational efficiency, on the capacity to discretize complex geometries, and on the "quality" of the resulting discretizations.

4.2 Hardware and Software Influences: an Example

The purpose of this section is to present some relevant properties of software and computer hardware and to describe their influence on the implementation of numerical models. This should set the scene for a discussion of grids and grid generation, and motivate the formulation of the definitions to follow in subsequent sections.

Consider a one-dimensional, incompressible, gravity-driven flow in a straight, open channel of uniform rectangular cross-section. Suppose that the only variables of interest are the height and the velocity of fluid. Suppose that the length of the channel has been discretized into intervals (or cells) of possibly differing length, and suppose that the geometry of the cells is to be represented in terms of the coordinates of their right endpoints along the channel.

Figure 4.1 and figure 4.2 show what are by far the two most common paradigms of storing the data for computational modeling of such a problem. For compactness, the figures show only a small number of cells and only a segment of the available computer memory. In the first storage model, the coordinates of the endpoints of the cells are stored at consecutive locations in memory in order of increasing distance from the left end of the channel. The location of only the first coordinate in this sequence is stored explicitly and, therefore, available directly. A similar arrangement is adopted for the height and velocity data, here assumed uniform in each cell. No algorithmic requirements are placed on the starting location of each of the three sequences of variables relative to each other. In the second model, the geometric and solution data for each cell are grouped together in a prescribed manner, but the data groups (also called **structures** or **records**) for different cells are not necessarily stored in any particular order. The storage location of the structure for any particular cell is stored only in the structure of the previous cell in the list. The only exception to this is the first cell: its location is stored separately, and is directly available.

The first model is based on the **array** data-structure while the second is based on the **link-list** data-structure. In the **doubly-linked-list** variant of the second model, each structure except that of the first cell also retains the location of the previous cell in the list, thereby allowing two-way instead of one-way **traversal**. These two storage models are not the only ones possible but they illustrate the most important fundamental differences existing in the entire spectrum of possibilities, comprehensively discussed in [199, 208].

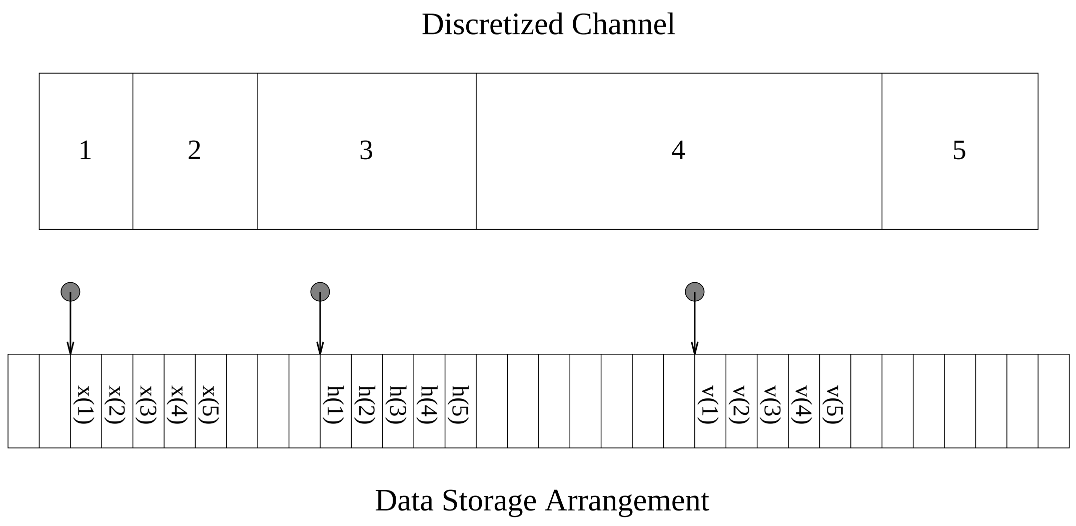

Figure 4.1: Array-based data-storage model.

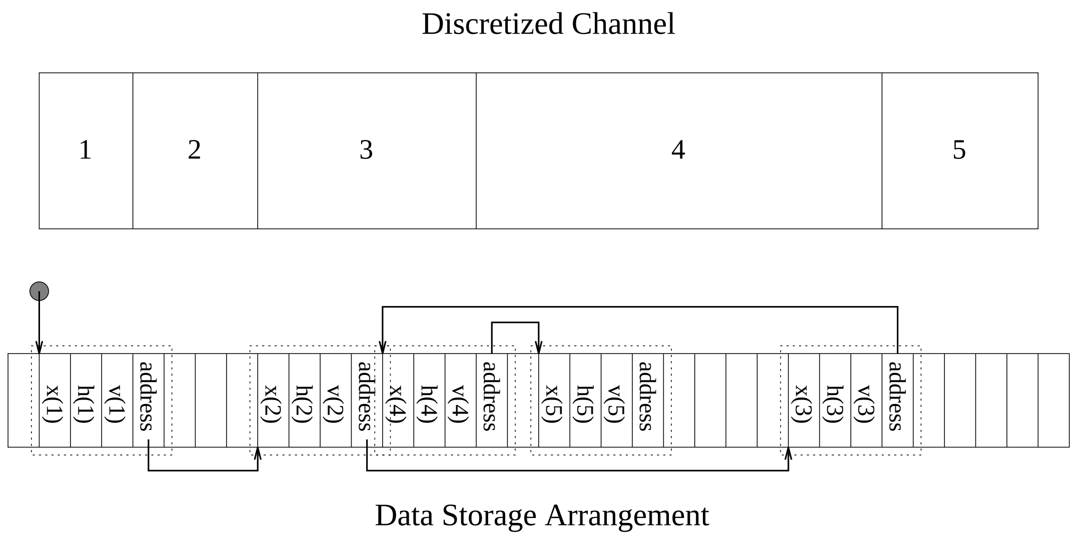

Figure 4.2: Link-list-based data-storage model.

In order to view or modify a datum stored in the memory of a computer, its storage location (identified by an **absolute address**) must first be determined. Suppose, for example, that the address for the datum representing the height in the i^{th} cell along the channel is required. In the first storage model, the address is determined by adding an offset (or **relative address**) to the known starting address (or **base address**) of the height sequence of data. The offset is computed using the fact that heights are stored in consecutive order and the fact that each datum takes up a known storage space. In the second storage model, the situation is somewhat reversed: the offset of each **component** or **field** (such as height) relative to the base address of a structure is uniform and known beforehand. However, the base address for the structure of the i^{th} cell must be determined. In the worst case, this must be done by traversing over the first $(i-1)$ links in the list, hopping from cell to cell. In each storage model, the same memory access procedure is followed for each of the three main data sets (that is, for x , h, and v).

In both storage models described above, entities such as “cell”, “coordinate”, or “height” may exist at the conceptual and software levels but are not represented in the hardware. The data stored in hardware mostly consists of memory addresses, and integer and floating point numbers. Also, no stored datum carries with it any indication of the variable it represents or quantifies; the indices in the figures are only symbolic. Both examples show, though, that by arranging for the storage and access of data to occur in accordance with a prescribed scheme, it is possible to create an association between each variable encoded in the software and the number that quantifies it in the hardware. This association is called the **storage map**. More precisely, the storage map is a function mapping each variable to its storage location in memory. The architecture of the storage map is specified in the software. The software instructions involving memory access must be designed around this architecture, but the detailed construction of the storage map occurs during the compilation and linking of the software.

In a computational model, the number of individual quantities that must be stored and manipulated is almost always so large that it would be quite infeasible to give each quantity a specific name in the software. Even if this were feasible, such an arrangement would require the software to be modified whenever the number of variables changed. Instead, a generic name is used for each group of variables and repetition-loops are used to perform similar operations on all members of a group using that generic name. Repetition loops with generic variable names also execute more efficiently in hardware. In the first storage model a generic variable-name would be supplied for each of the three major data sets. To identify a specific variable in any of these sets, an indexing subscript specifying the order of the variable in the sequence is attached to the corresponding generic name, as shown in figure 4.1. In the second model, only one generic name is required for the basic data-structure and for each of its components. In both models, instead of using explicit names, variables are most often retrieved and manipulated on the basis of some relation they have to other variables. The most frequently used relations are based on concepts such as "adjoining" (between cells) and "solution in" (between solution vectors and their associated cells). Both concepts would be used, for example, if it is desired to access the height and velocity of the two cells adjoining the one currently being processed.

Referencing variables in the indirect manner described above requires a mechanism to identify the storage location of variables using only the storage location of given variables and a specific relation between the sought and given variables. Such a mechanism is called a **connectivity map**. More precisely, let R be a relation from the set of variables X to the set of variables Y. Let $L(X)$ and $L(Y)$ be the set of storage locations of the sets X and Y, respectively. The connectivity map of relation R is an isomorphic relation, $\tilde{R}$, from the set $L(X)$ to the set $L(Y)$

such that $\langle l(x), l(y)\rangle \in \tilde{R} \iff \langle x, y\rangle \in R$, where $x \in X, y \in Y, l(x) \in L(X)$, and $l(y) \in L(Y)$. Although defined in terms of storage locations, connectivity maps are normally implemented on the basis of variable names instead.

Connectivity maps are based on repetitive patterns in the storage arrangement. If the required patterns are non-existent or difficult to specify, connectivity maps must be explicitly stored in the hardware. If the required patterns are simple and highly regular, connectivity maps may be used implicitly, without ever being stored. For example, suppose that the relation for a connectivity map is based on "fluid height in the right neighbor". In the first storage model, an index is always available that references the cell currently being processed. The storage location of the height in the right neighbor of the current cell would be obtained by incrementing this index by one to obtain the offset of the sought location from the base address of the height array. Little more than manipulation of an index is required and the task can be accomplished completely "in" the software. In the second storage model, the required location would be obtained by adding the offset of the height component from the base address of the structure for the right neighbor of the current cell. The base address is in turn given by the "address-of-next-cell-in-the-list" field of the current cell. In this case, part of the connectivity map, the **link**, had to be explicitly stored to enable location of the structure of the right neighbor.

This section attempted to show how the quest for general-purpose, efficiently-produced software influences the types of data used and the memory access methods chosen. In particular, generality and scalability require the use of generic variable names and repetition loops which in turn require most variables to be referenced indirectly using their relation to other variables instead of their explicit names. The important function performed by connectivity maps in allowing this indirect access, and the close dependence of these maps on the storage map were described. The next few sections discuss in more detail the prominent roles of storage and connectivity maps in grid generation.

4.3 Definitions and Fundamental Concepts

4.3.1 Definition of the Computational Region

A discrete solution may be sought in subsets of physical space only if they satisfy certain conditions. In this chapter, grids and grid generation for only subsets that allow use of a flow-solver based on the Finite-Volume, Finite-Element, or Finite-Difference methods are considered. For simplification, the definitions are based on a special class of these subsets: that consisting of all finite single line segments in $\mathbf{R}$, all finite single planar regions in $\mathbf{R^2}$, and all finite single spatial regions in $\mathbf{R^3}$. All other admissible subsets of space may be obtained by bounded continuous transformations of members of this class. A member of this class will hereafter be called a **computational region**, and the precise requirements are as follows:

Definition: Set S is a **computational region** iff for some given $n \in \{1, 2, 3\}$:

1. S is a compact (that is, by the Heine-Borel Theorem, a bounded, closed) subset of $\mathbf{R^n}$;

2. $cls(int(S)) = S$, where $\tilde{S} = int(S)$ is the interior of S in $\mathbf{R^n}$, that is, the union of all open sets in $\mathbf{R^n}$ that are subsets of S; and $cls(\tilde{S})$ is the closure of $\tilde{S}$ in $\mathbf{R^n}$, that is, the complement of the exterior of $\tilde{S}$ in $\mathbf{R^n}$, where the exterior of $\tilde{S}$ in $\mathbf{R^n}$ is the union of all open sets in $\mathbf{R^n}$ that are disjoint from $\tilde{S}$;

3. S is connected in $\mathbf{R^n}$; and

4. $bdr(S)$ is orientable in $\mathbf{R^n}$, where $bdr(S)$ is the complement of the union of the interior in $\mathbf{R^n}$ and the exterior in $\mathbf{R^n}$ of S.

The $\mathbf{R^n}$ in the above definition will hereafter also be called the **space of the computational region**. The Finite-Volume, the Finite-Element, and the Finite-Difference methods, unlike the Boundary-Integral method, all require the boundedness in condition 1.

The second condition excludes sets having any "part" that differs in "dimensionality" from $\mathbf{R^n}$. Unacceptable sets include, for example, a subset of $\mathbf{R^2}$ consisting of a finite planar region with line segments extending from it, or, for example, two cubes in $\mathbf{R^3}$ connected by a line or a surface, or, for example, a curved surface segment in $\mathbf{R^3}$, or, for example, a curved line segment in $\mathbf{R^2}$. As the last two examples demonstrate, one consequence of this condition is that a computational region is required to have nonzero measure in its space, that is, to have finite n-volume in $\mathbf{R^n}$.

The third condition is useful but not mandatory; without it, the computational task would amount to the simultaneous solution of two or more problems that are either independent or at least not fully coupled from the fluid-dynamic point of view. An example of the latter situation is a problem with one or more solid bodies which are allowed to move but which completely isolate different fluid masses. Here, each isolated region may be regarded as a single computational region, and the overall computation may be defined to consist of a number of computations on different computational regions which are coupled together through the motion of the solid bodies. Effectively, the solid bodies provide the coupling between the boundary conditions of the different computational regions.

The fourth condition ensures that the theorems of calculus on manifolds, for example, Gauss's and Stokes's Theorems, apply to computational regions. The definition forces a computational region to have an "interior", an "exterior", and a "boundary", and so allows identification of the locations at which boundary conditions should be applied. The interior of any solid body that lies "within" the outer boundary of a computational region is exterior to the computational region and the boundary of such a body is also a boundary (oppositely oriented) of the computational region. From the above definitions, it may be concluded that a computational region may be fully defined by definition of its boundaries.

4.3.2 Definition of a Grid

Definition: E is a **subregion** of a computational region, CR, iff:

1. E is a closed, simply-connected, subset of CR;
2. $E = cls(int(E))$, where "interior" and "closure" are defined in terms of subsets of the space of CR; and
3. $bdr(E)$ is orientable.

Conditions on E have similar consequences and interpretations to the corresponding ones for a computational region; simple-connectedness is additionally imposed to simplify the implementation of flow-solvers and boundary conditions. Boundedness of E follows from condition 1 and the definition of computational region. Condition 2 implies that any CR has a finite number of subregions. For practical computations, several conditions, usually specifying minimum and maximum bounds on volume and various dimensions, are additionally imposed on E. Although E is generally a subset of $\mathbf{R^n}$, $n \in \{1, 2, 3\}$, it is usually more conveniently specified in terms of another set. If, for example, E is a region in $\mathbf{R^3}$ bounded by planar polygonal faces, or, for example, a polygon in $\mathbf{R^2}$, it may be fully specified in terms of the set of coordinate vectors of its vertices. Subregions are called finite volumes in the Finite-Volume method, and finite elements in the Finite-Element method. Convexity of a subregion is almost always desirable, especially with higher-order methods.

Definition: $\mathcal{E}$ is a **subregion set** of some computational region, CR, iff for some $n \in N$:

1. $\mathcal{E} = \{E_1, ..., E_n\}$, where $\forall i \in \{1, ..., n\}$ E_i is a subregion of CR;

2. $\bigcup_{i=1}^{n} E_i = CR$; and

3. $(\forall i, j \in \{1, ..., n\})[(i \neq j) \implies (E_i \cap E_j$ has measure zero in the space of $CR)]$.

The last condition, necessary for correct implementation of Finite-Volume and Finite-Element methods, specifies that subregions can overlap at no other than points in $\mathbf{R}$; points and edges in $\mathbf{R^2}$; and points, edges, and surfaces in $\mathbf{R^3}$. The second condition ensures that the subregions fully cover CR and nothing else. $\mathcal{E}$ is sometimes called a partition of CR, but a partition requires strict disjointness of its members. The definition given for $\mathcal{E}$ is fully sufficient for proper implementation of a Finite-Volume or Finite-Element solution algorithm.

Two subregions of a subregion set of a computational region CR are **adjoining** iff their intersection in the space of CR is nonempty. Adjoining subregions will hereafter also be called **neighbors**.

Definition: Let $\mathcal{E}$ be a subregion set of some computational region. Let $E_i \in \mathcal{E}$ have m neighbors. NC is a **neighborhood connectivity tuple** for subregion E_i iff it is of the form $\langle i, j_1, ..., j_m \rangle$, where $\forall n \in \{1, ..., m\} E_{j_n} \in \mathcal{E}$ is the n^{th} neighbor of E_i, with the ordering of neighbors given by some convention.

The number of neighbors, m, may vary from subregion to another. One possibility of dealing with this situation is to choose m equal to the largest possible number of neighbors, and for elements lacking a particular neighbor in the ordering, the corresponding index in the neighborhood connectivity tuple is set to some special reference value. Conventions for ordering neighbors are always simple in $\mathbf{R}$ and $\mathbf{R^2}$ but could be elaborate in $\mathbf{R^3}$. In practice, orderings may not always be required. Multiple neighborhood connectivity tuples may also be used instead of one; for example, in $\mathbf{R^3}$ a separate tuple may be used for each type of neighborhood: vertex, edge, and face.

In principle, a neighborhood connectivity map can always be recovered by searching through the geometric attributes of subregions and can therefore be viewed as a redundant entity. In practice, though, such an approach is subject to finite-arithmetic errors and in most cases is not computationally affordable. Therefore, such maps are produced, modified, and used separately from geometric data.

<u>Definition:</u> $\mathcal{NC}$ is the **neighborhood connectivity set** of subregion set $\mathcal{E} = \{E_1, ..., E_n\}$ iff it is of the form $\{NC_1, ..., NC_n\}$, where $\forall i \in \{1, ..., n\}$ NC_i is the neighborhood connectivity tuple for E_i.

Each subregion may have only a unique solution vector.

<u>Definition:</u> Let $\mathcal{E}$ be a subregion set of some computational region. The set $\{\vec{U}_1, ..., \vec{U}_n\}$ of the solution vectors for each subregion in $\mathcal{E}$ is the **solution vector set** of $\mathcal{E}$.

A solution vector set is hereafter assumed to exist whenever a subregion set exists.

<u>Definition:</u> Let $\mathcal{E}$ be a subregion set of some computational region and let SV be its solution vector set. Let $\vec{U}_j \in SV$ be the solution vector for subregion $E_i \in \mathcal{E}$. The tuple $SC = \langle i, j \rangle$ is the **solution connectivity tuple** for subregion E_i.

<u>Definition:</u> Let $\mathcal{E} = \{E_1, ..., E_n\}$ be a subregion set of some computational region. $\mathcal{SC}$ is the **solution connectivity set** for $\mathcal{E}$ iff it is of the form $\{SC_1, ..., SC_n\}$

such that $\forall i \in \{1, ..., n\}$ SC_i is the solution connectivity tuple for subregion E_i.

Definition: $\mathcal{G}$ is a **grid** for computational region CR iff $\mathcal{G}$ is of the form $\langle \mathcal{E}, \mathcal{NC}, \mathcal{SC} \rangle$, where $\mathcal{E}$ is *a* subregion set of CR, $\mathcal{NC}$ is *the* neighborhood connectivity set for $\mathcal{E}$, and $\mathcal{SC}$ is *the* solution connectivity set for $\mathcal{E}$.

In the above definitions, indices were used in connectivity tuples to identify members of sets; for representation in a computer, the indices are replaced through the storage map by memory addresses that identify locations in memory. After this replacement, connectivity sets by definition become relations mapping memory locations in their domains into memory locations in their ranges, as described in the previous section.

The definition of grid given above facilitates avoidance of the epidemic confusion between "grid" and "spatial discretization", that is, between $\mathcal{G}$ and $\mathcal{E}$, respectively.

4.3.3 Definition and Role of Grid Generation

A Grid was defined purely in terms of the geometry of subregions and in terms of connectivity maps. Grid generation may be defined without the introduction of any additional major concept. For example, "generation" of a grid could be defined as its specification and its storage in the memory of a computer. However, such a definition would not reflect the conception most commonly expressed in the literature; to do so, at least the process of spatial discretization of the computational region must be additionally included.

Grid generation is conveniently defined by describing the three main tasks that it performs in computational modeling:

1. **Spatial subdivision of the computational region and storage of the subregion set:** Once it has been specified, the region of physical space in

which a solution is to be computed must be appropriately divided into subregions. This subdivision is performed by following an algorithm that determines the geometric variables defining all the resulting subregions using only the definition of the computational region and possibly some set of control parameters. The geometric variables must then be stored in memory. The subdivision algorithm partly defines the generation process but the implementation details are not relevant.

2. **Creation and storage of the neighborhood connectivity maps:** Neighborhood connectivity maps establish in computer memory the association between each subregion and all its neighboring subregions. In practice, subregions are stored as groups of their intrinsic properties which include at least the geometric definition.

3. **Creation and storage of the solution connectivity maps:** Solution connectivity maps establish the association in computer memory between each subregion and its extrinsic data which at least includes a solution vector.

All solution algorithms require each subregion to allow access on a routine basis to its own intrinsic and extrinsic attributes as well as to those of all its neighbors. The tasks described in items 2 and 3 above fully enable this access. As we saw in the array-based storage scheme in section 4.2, however, it is possible for the connectivity map to be stored implicitly within the instruction set of the computational algorithm, rather than in the form of explicit data.

4.4 Review and Classification of Grids and their Generation Methods

This section establishes, describes, and compares the main categories of grids and grid generation methods. Criteria for the assessment of the suitability and performance of grids and generation algorithms are presented, and the manner in which the characteristic features of each of the above categories result in varying extents of satisfaction of these criteria is described.

4.4.1 Evaluation Criteria

The admissibility constraints for discretizations of a computational region and for connectivity maps that were outlined in section 4.3 are necessary but not sufficient for converged numerical solutions. Beyond those constraints, there are no known general sufficiency conditions or governing fundamental laws. Instead, there have evolved three main categories of loosely defined, and often competing desiderata that may be satisfied to varying extents. These are as follows:

- generality, reliability, robustness, and automatability;
- grid quality; and
- computational efficiency.

The desiderata of the first category relate to the ability of an algorithm to successfully produce satisfactory spatial discretizations for arbitrary computational-region geometries (which include the surfaces of bodies immersed in the flow-field), and with little human intervention (which includes tuning of parameters, whether a-priori or by trial and error).

The desiderata of the second category relate to the extent to which the geometric features of the subregion set promote accurate, reliable, and stable and rapidly convergent computation for the given system of equations [22]. This depends on such factors as accurate resolution of boundaries and interfaces; absence of features associated with increased numerical errors, such as excessive skewedness, or poor orthogonality of grid lines; smoothness of variation of subregion n-volumes; appropriate alignment of the edges or faces of subregions with the local flow direction; proper directional scaling of the dimensions of subregions with the directional length scales in the flow; and a density distribution for subregions that is compatible with the desired resolution distribution. This category, unlike the other two which are motivated primarily by practical considerations, considers the effects of subregion geometries on the solution quality, and it is therefore related to the development of the lacking sufficiency conditions mentioned above.

The third category expresses the desire for optimally low combinations of requirements of processing capacity and memory.

For many classes of problems, a fourth desideratum, adaptability, defined in section 4.5, is also introduced.

4.4.2 The Proliferation of Grid Types and Generation Methods

Computational Fluid Dynamics is used in a wide range of situations. The main types of difference between individual problems are: the governing system(s) of equations, and the physical phenomena they model; the flow regimes encountered; the presence or absence of time dependence; and the complexity of boundary geometries and boundary conditions. The system(s) of equations being modeled and the flow regimes encountered range from 1-D scalar advection flows, to 3-D, compressible, turbulent, reacting, electrically-conducting flows. For some flows, the subregion set may be required to satisfy (albeit rather loosely) certain geometric features to preclude loss of solution accuracy, resolution, or convergence, and in the extreme, to avoid grossly incorrect results or failure of the flow-solver. In such cases, the priority of the second category of desiderata is given an increased weight. The geometry of the computational region may range from simple 2-D shapes to complex 3-D shapes that can only be conveniently represented by thousands or millions of individual curved surface patches. The more complex the geometry, the greater the relative weight that is given to the robustness, reliability, and automatability of the spatial discretization algorithm, that is, to the desiderata in the first category.

In addition to physical differences between problems, exogenous differences are also possible. These are mainly: the chosen method(s) of solution, and the specific numerical algorithm(s); the costs and availability of memory and processing capacity; the purposes of the computation; and, for tasks requiring parametric or sensitivity studies, the number of required repetitions of a computation. The chosen method(s) of solution primarily affect the relative requirements of memory and processing effort and introduce tradeoffs between accuracy and computational resource requirements. The purposes of a computation may range from detailed, highly resolved, investigative studies, where the emphasis is on accuracy and reliability, to optimization tasks, where the emphasis is on accuracy and computational efficiency, to preliminary estimates, where the emphasis is on speed and on reduced preparation effort.

The above lists and examples illustrate how different problems and situations demand different weights to be placed on the various desiderata listed in the previous section. This, together with the absence of strict rules for admissible discretizations, explains and justifies the development and proliferation of a large number of grid types and generation methods. Each method has strengths and weaknesses, and several of them are highly optimized for a specific type of problem and situation.

4.4.3 The Comparison of Grid Types and Generation Methods

The previous section leads to two main conclusions; namely: (i) the performance of a grid generation method must be defined in terms of how well and how efficiently it performs each of the three tasks described in section 4.3 for the given *problem*, for the given *purposes*, and under the given *circumstances*; and (ii) evaluating the superiority of a type of grid or generation method over another is only meaningful when the application and circumstances are specified. The general basis of comparison should therefore be "fitness for purpose".

4.4.4 Classification Bases

Grids were defined partly in terms that are independent of the implementation, that is, in terms of $\mathcal{E}$, and partly in terms whose origins lie in the peculiar requirements of computer hardware and software, that is, in terms of $\mathcal{NC}$ and $\mathcal{SC}$. This anomalous formulation may be disconcerting since the conceptual and implementation viewpoints should ideally remain separate. The justification for this situation is that in computational methods grids are usually regarded as a means, and their definition should be based on aspects having the greatest bearing on the functions they perform. The two aspects which are by far the most important are the geometry of the subregions, and the type of connectivity maps. The geometry of the subregions is important because it largely determines the resolution, accuracy, and reliability of the solution. The neighborhood connectivity map and, to a far lesser extent, the solution connectivity map, and their characteristic features largely determine the computational efficiency, the ability to modify the grid to better suit the solution, the parallelizability and vectorizability of the solution algorithm, and the ease and flexibility with which automated spatial discretization of the computational region can be performed.

In accordance with the above, there are two dominant classification criteria: by type of neighborhood connectivity map; and by method of spatial subdivision of the computational region. The method of spatial subdivision was purposely excluded from the definition of a grid since two grids should be identical if they have identical connectivity maps and subregion sets, even if two different methods were used to derive the subregion sets. The method of spatial subdivision was reintroduced, however, in the definition of grid generation for the reason given therewith which is justified because different methods of generation impart specific geometric characteristics and because each method has its own peculiarities, difficulties, limitations, and computational costs.

The two major bases of classification are not completely independent since some generation methods are tied to a specific type of neighborhood connectivity map. Further complications are introduced by hybrid methods which may use several types of connectivity map and generation method. Nevertheless, classification is useful for the same reason it is in other branches of science; namely, it focuses attention and understanding on underlying characteristic features and allows quick determination and prediction of the properties of classified entities.

4.4.5 Classification by Form of Neighborhood Connectivity Maps

Let CR be a computational region. A subregion of CR is an **interior subregion** iff it is disjoint from the boundary of CR. A subregion of CR is a **boundary**

subregion iff it is a subregion of CR which is not an interior subregion (that is, iff it is a subregion of CR whose intersection with the boundary of CR is nonempty).

The following definition adopts the convention described in section 4.3 for subregion sets having subregions with differing numbers of neighbors.

Definition: Let $\mathcal{G} = \langle \mathcal{E}, \mathcal{NC}, \mathcal{SC} \rangle$ be a grid. Let m be the greatest number of neighbors possessed by any subregion of $\mathcal{E}$. $\mathcal{G}$ is a **structured grid** iff:

1. Each interior subregion has exactly m neighbors; and

2. For some constants $c_1, ..., c_m$, which are all elements of $Z = \{..., -2, -1, 0, 1, 2, ...\}$, all neighborhood connectivity tuples in $\mathcal{NC}$ are of the form $\langle i, i + c_1, ..., i + c_m \rangle$.

The first condition implies that only boundary subregions may have missing neighbors (that is, less than m neighbors). The second condition defines a **partial ordering** on the subregion set. Since every partial ordering on a finite set admits a **linearization**, then there exists a **linear ordering** on $\mathcal{E}$. In particular, if the elements of the subregion set are ordered in memory according to their indices, then after replacing the indices in the neighborhood connectivity tuples by memory locations, the n^{th} neighbor of every subregion having such a neighbor will have the same address relative to its reference subregion. Under this ordering, if the neighborhood connectivity tuples are expressed in relative-address form, they will be identical for all interior subregions. The far-reaching implication of this is that the connectivity tuple for any interior subregion may be used to reconstruct the connectivity tuple for any other subregion as well as the entire connectivity map for the subregion set.

A typical two-dimensional structured grid is represented in figure 4.3. The computational domain and all its subregions are rectangular (although other subregion geometries, such as triangular and hexagonal ones, also permit the definition of structured grids). The indices indicate the ordering of the subregions in memory. The figure shows how the relative indices of corresponding neighbors are identical for all interior subregions. Grid lines and boundaries need not be straight as in the example, and the main ideas illustrated extend readily to 3-D.

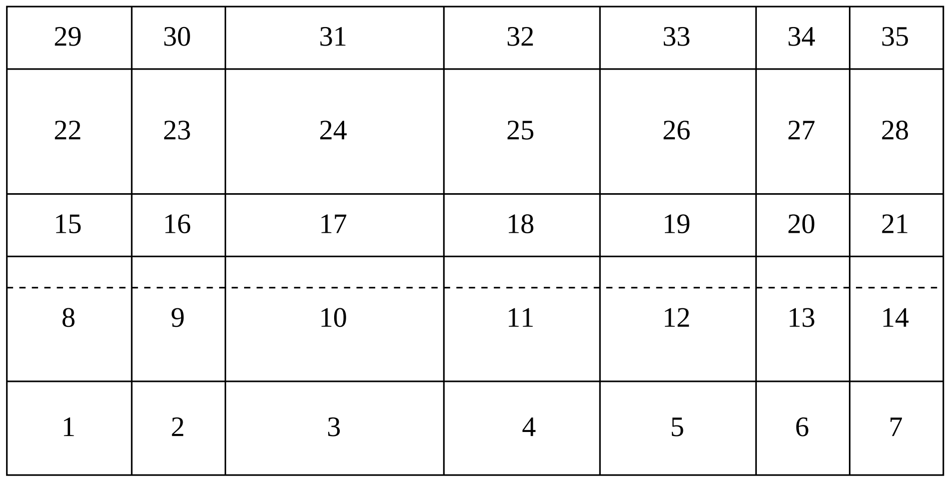

Figure 4.3: A two-dimensional structured grid.

The array-based storage model described in section 4.2 is ideal for structured grids because of the shared requirement for sequential ordering.

The definition given above for structured grids is slightly more general than the one commonly used in the literature but is more correct in regard to stating the conditions necessary to confer the defining characteristics and computational advantages.

Definition: A grid is **unstructured** iff it is not structured.

A typical unstructured grid is shown in figure 4.4. There is no apparent ordering of the subregions in memory that leads to a uniform neighborhood connectivity tuple for interior subregions. Although not sufficient to preclude such an ordering (assuming the adoption of the convention for missing neighbors described in section 4.3), the disruptive effect of non-uniform neighborhood patterns around interior subregions is evident in this regard. Even if such an ordering exists, the search for it is generally an N-P hard problem and, therefore, a discouraging option. Similarly discouraging is the option of locating neighbors by searching through geometric properties. Instead, the neighborhood connectivity tuple for each subregion is explicitly stored.

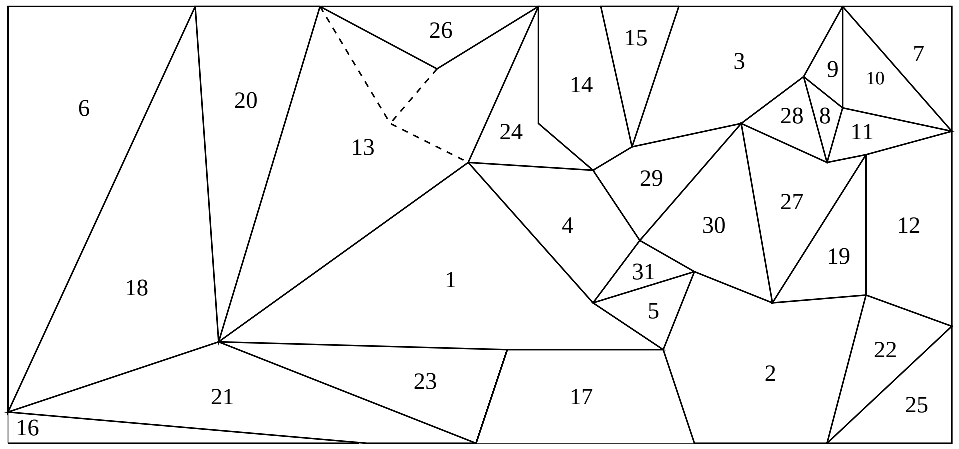

Figure 4.4: A two-dimensional unstructured grid.

The link-list-based storage model described in section 4.2 is ideal for unstructured grids because of the shared absence of a requirement for sequential ordering. The elements in a link-list for this 2-D example will require a link to each of the possible neighbors, and any missing neighbor may be indicated by a link pointing to special, "null", address in memory. Other data structures are possible but only if they admit the characteristic irregularity of ordering.

The definition adopted for unstructured grids groups together all grids whose "regularity" falls short of that of structured grids. The example given above of an unstructured grid shows an extreme case where no regularity at all appears present, but levels of regularity intermediate between the two extremes described so far, although rarely encountered in practice, are also possible.

Consider, for example, the grid represented in figure 4.5. If the cells are stored in memory in order of the numbers given in the figure, the major difference compared to a structured grid is that instead of constants in the neighborhood connectivity tuples, there are now variables, but these are simple to evaluate and depend only on the index of the reference cell. In particular, for the example shown, the formulae for indices of neighbors, where $[x]$ represents the greatest integer in x, are as follows: left face-neighbor: $[i/2]$; left lower vertex-neighbor (possibly also a face-neighbor): $[(i-1)/2]$; left upper vertex-neighbor (possibly also a face-neighbor): $[(i+1)/2]$; lower neighbor: $(i-1)$; upper neighbor: $(i+1)$; right lower face-neighbor: $2i$; right upper face-neighbor: $(2i+1)$; right lower vertex-neighbor: $(2i-1)$; and right upper vertex-neighbor: $(2i+1)$. As long as the neighborhood patterns for all interior subregions remain isomorphic to each other, modifications to the specific order of the structured class can be accomplished with no significant change in the fundamental properties or the computational advantages.

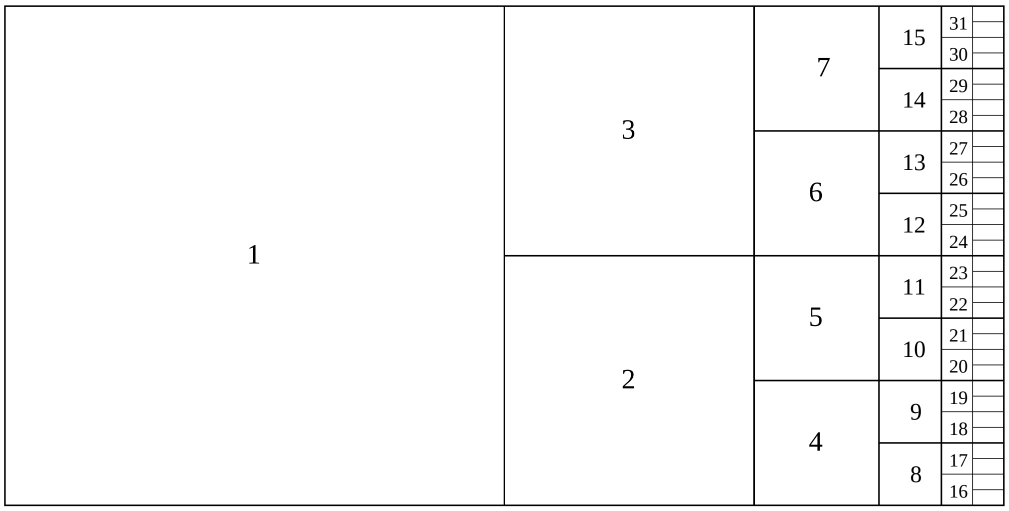

Figure 4.5: A two-dimensional regularly-structured grid.

The definitions which distinguish between structured and unstructured grids are based on the *actual* forms of the neighborhood-connectivity tuples. One implication of this is that all grids that are capable of taking a structured form can actually exists and be implemented as unstructured grids.

4.4.6 Classification by Method of Spatial Subdivision

Although all but the most specialized and rarely used subdivision methods are at least listed or mentioned, the focus in this section is on describing those properties and underlying concepts that are related to the three evaluation criteria mentioned at the beginning of this section. References to detailed properties and implementation algorithms are given abundantly. Recent texts, reviews, and wide-ranging collections include [360, 118, 365, 165, 322, 363, 366, 14, 198]. The methods are described only in a 3-D context, unless the specialization to 2-D is not obvious. Specialization to 1-D is almost always trivial and is therefore ignored throughout.

4.4.6.1 Partial Differential Equation Methods

These methods determine the spatial discretization by solving for the locations of the vertices of subregions (usually quadrilaterals in 2-D or hexahedra in 3-D) as the unknowns in a set of elliptic [364, 365], parabolic [260, 113], hyperbolic [344], or, much less commonly, biharmonic [330] or “equidistributional” [116] partial differential equations. Although applicable to unstructured grids, PDE methods are most frequently used for structured ones.

For elliptic systems, the solution proceeds most commonly by iteration on the local linearization of the governing equations (say, using central differencing or relaxation) starting with an initial distribution of vertices, often provided by a simpler, more economic discretization algorithm. Boundary conditions specifying the distributions of vertices on all boundaries are required. Conditions on vertex mobility along a boundary depend on the boundary conditions: with Dirichlet ones, the vertices are fixed; with Neumann ones, the vertices may be movable. Introduction of Poisson terms in the Laplace operator is necessary for control of the density distribution (especially in the vicinity of convex geometries) and orthogonality, but is effective to the extent of sufficing as a basis for adaptivity [195]. General methods based on the computational region geometry for deriving appropriate Poisson functions and for specifying their parameters in terms of vertex distributions on boundaries have been developed [357, 339, 365] but significant skill is still required for proper choice of these functions for adaptive or specialized applications.

Elliptic methods are characterized by exact conformality with all boundaries, by highly smooth and orthogonal discretizations [58], and by amenability to r-adaptation [58]. Despite the additional computational expense, elliptic methods are the most frequently used of the PDE methods, and also frequently used for smoothing discretizations generated by other methods, possibly in the simplified form of Laplacian filters.

Parabolic systems can be constructed by parabolizing an elliptic system, and Poisson terms can still be used for adaptation and control of orthogonality [264], although less effectively than for elliptic equations. The solution proceeds by marching, usually outward from solid boundaries, typically using implicit finite-differencing on the locally-linearized version of the governing equations. Boundary conditions are required on all boundaries. Since there is no iteration, computational costs are lower than for elliptic systems.

Hyperbolic methods simultaneously solve two equations in 2-D or three ones in 3-D: typically, one equation specifies the desired subregion density distribution while any remaining ones (depending on the spatial dimension) specify orthogonality constraints [344]. The discrete solution methods used are similar to those for parabolic systems, above. Boundary conditions are required only on the surface from which the marching is to occur, and admissible nowhere else. Thus the outer boundary is determined as part of the solution: a limitation, except for exterior flows with "infinity" outer-boundary-conditions and for chimera grids (see below). Although efficient, and somewhat adaptable (by the introduction of second-order forcing functions in the homogeneous equations [197]), these methods are inherently unable to cope (without collision of fronts) with anything other than simple, smooth geometries.

4.4.6.2 Algebraic Methods

In this class of methods, the locations of vertices in the computational region are specified explicitly in terms of mappings from regions with "simpler" geometry. Explicit specification leads to the biggest advantage of these methods: their high speed, compared, say, to elliptic methods. The mappings most commonly used are either based on a conformal mapping [179, 252, 365] or on unidirectional [334, 365] or transfinite [137, 365] interpolation. Algebraic methods are most applicable to structured grids since they presume a homeomorphism between the domain and its image under the mapping. Obviously, conformal mapping methods may also be classified as PDE methods.

In conformal mappings it is only possible to partially specify the boundary vertex locations a-priori; in unidirectional interpolation a-priori locations may be fully specified on only some of the boundaries; in transfinite interpolation, a-priori locations, and possibly face slopes, may be fully specified on all boundaries, as well as on artificial internal boundaries used for control of discretization density, but all boundaries must have matching vertices.

Algebraic methods have several limitations. Conformal mappings are not suitable for complex geometries and do not admit local control over discretization density, although much progress has been made by use and extension of the Schwarz-Christoffel transformation [141] instead of a standard conformal mapping, or by use of multiple conformal mappings, each simpler than the overall composite one [180]. Precise control over grid quality by interpolation methods is difficult, to the extent that there is a tendency for crossing of grid-lines, although that can be somewhat countered by basing the interpolation on normalized edge arc-length.

Conformal and interpolation methods are often used in combination, for example, with other PDE methods, as in [341]. In particular, transfinite interpolation is often used to stretch a discretization generated by a conformal mapping (which is only applicable in 2-D), either into 3-D or into axisymmetric discretizations.

4.4.6.3 Variational Methods

Variational methods, also called optimization and energy methods, are based on minimization of integrals over the entire computational region of field functions providing whatever measure is required of the local “lack of quality”, typically a weighted combination of non-orthogonality and non-smoothness.

One method of performing the minimization is to obtain the Euler-Lagrange equations of the variational problem and to solve these iteratively [58]. Another method is to discretize the variational integrals directly and perform numerical optimization on the resulting summations using a numerical optimization algorithm [63], such as a conjugate-gradient method.

Variational methods produce high-quality discretizations (although features like orthogonality and smoothness are not always sufficient to ensure suitability for the solution) and are well-suited for adaptation, but a major overall detraction is their high cost and complexity, especially in 3-D. These methods are applicable to structured and unstructured grids.

4.4.6.4 Voronoi-Based Methods

For any set of distinct points in a computational region, each point defines a unique convex cell that encloses it and contains every point in the computational region closer to it than to any other point in the set. Such a cell, called a Voronoi cell, has a surface consisting of planar polygonal facets, except possibly at boundaries of the computational region. For any set of points, the set of (by definition, non-overlapping, convex, bounded) Voronoi cells is unique, covers the entire computational region, and is called a Dirichlet tessellation.

A Delaunay triangulation for a computational region is the spatial dual of the Dirichlet tessellation. It is obtained by connecting the focal points of every two Voronoi cells sharing a face by a straight line segment to define the edge of a tetrahedron. In 2-D, such a triangulation is unique if no four Voronoi points are co-circular. Delaunay triangulations have several geometric features [126, 25, 26, 282] that are generally heuristically, and in some cases provably [304], associated with high grid quality, but more so in 2-D than in 3-D; for example, in 2-D the Delaunay triangulation maximizes the minimum angles in the subregion set, and is the closest on average to an equilateral triangulation. These properties have strongly contributed to the appeal of Delaunay triangulations.

In order to avoid the computational expense of algorithms that obtain the triangulation by following the definition, alternative ones, based on Steiner triangulations, that is, incremental insertion of (Steiner) points, are used in practice. A reliable and simple one [403] uses the "circumscribing hypersphere property": starting from an easily generated, initial, coarse triangulation, points are inserted into the computational region and any triangles that consequently fail to satisfy the above property are deleted, leaving an “insertion polyhedron” whose faces are then connected to the inserted point to form new tetrahedra. Other algorithms [57] and variations of them [174] have also been developed.

Voronoi-based methods are applicable almost exclusively to unstructured grids and are in widespread use due to their automatability, ability to discretize arbitrary geometries, and the geometric properties of the resulting discretizations. One detraction of these methods is that the point cloud must be generated prior to the triangulation. Points have been extracted from structured (possibly overlapping) grids [184, 408], by tri-tree branching subdivision [122], by methods involving combination with an advancing-front method [258, 29, 296, 238], and by other methods [23, 404]. Another detraction, and a fundamental one, is that Delaunay triangulations are most suited for isotropic problems, although stretching has been used to extend them to mildly non-isotropic problems with success [237]. Violation of boundary conformality is another shortcoming which is, especially in 3-D, difficult but possible [404] to overcome.

4.4.6.5 Advancing-Front Methods

These methods are applicable with full generality to unstructured grids, and, with restrictions, to structured ones. Recent applications are described in [277, 188] but [128, 359] appear to be the original publications.

Starting from a closed surface of subregion faces (called a “front”), single subregions are incrementally introduced by selecting a face from the front and constructing a subregion sharing only that face into an undiscretized part of the computational region. The front is "advanced" by "replacing" the shared face with the other faces of the new subregion. The selection of faces is usually guided to advance the front mostly in one-element layers. The initial front is usually the surface of a solid boundary, and the geometry of the subregions that are introduced is determined in accordance with length scales specifying size and directional stretching, often chosen to produce the required subregion "quality", and often obtained by interpolation of values specified on (the vertices of) a background, coarser grid.

Contemporaneous advance from multiple fronts is also possible, but requires additional control [368, 238] to ensure adequate grid quality or to avoid failure of the algorithm, especially if initial fronts have greatly differing length-scales. As different fronts or different parts of the same front advance toward each other or toward boundaries, they may be joined together by the addition of shared subregions. The advance may also be halted at an early stage leaving the remainder of the computational region to be discretized by an alternate method. Yet a third possibility, but with poorer control over grid quality, is to allow the fronts to initially overlap and then to eliminate any overlap by multiple contraction mappings, as done with prismatic grids in, for example, [187, 188].

As for the Voronoi-based methods, the connectivity map is built up as subregions are created, but, in contrast, specification of a point cloud is not required, although specification of length scales and directional stretchings still is, and at a slight detriment to full automation. Though the optimality of the discretization in isotropic regions may be inferior to that of, say, a Delaunay triangulation, generality, automatability, and suitability for complex geometries has been demonstrated [231, 232, 275], while amenability to direct and accurate control of the directional scaling, alignment, included angles, and density distribution of subregions [17, 280] has demonstrated excellent suitability for viscous and turbulent flows.

4.4.6.6 Cartesian Methods

These methods accomplish the discretization by splitting the smallest hexahedron enclosing the computational region with three sets of planes. The planes in each set are parallel to the same pair of faces of the hexahedron, and each of the three sets is associated with a different one of the three face pairs. The resulting discretization consists of a cubic assembly of hexahedral subregions (cells), each geometrically similar to the bounding hexahedron. Generalization to parallelepipeds is obvious.

The discretization process is unaltered by the presence or geometry of any internal boundaries. Therefore, unless perfectly aligned with cell boundaries, any internal boundary of the computational region will cut through cells, resulting in a non-conformal grid.

The use of "Cartesian grids" in Computational Fluid Dynamics applications was first demonstrated satisfactorily for potential flows (in the transonic regime) [290] then for Euler flows [79] and [46, 54]. Adaptation was soon introduced for the potential equations [424] and for the Euler equations [46, 101, 85, 246, 4]. Since then, these methods have been successfully extended to a wide range of flow regimes and flow problems, including inviscid and low Reynolds number flows [48, 50, 102, 38, 85, 291, 292, 245, 247], and have also been applied to a wide variety of problems with complex physical processes such as combustion [274], interface phenomena [348], and others [284].

Cartesian methods are ideally suited for generation of structured grids. They are capable of fully automated discretization of arbitrary geometries, are computationally efficient, suitable for adaptivity, and obviate the need for independent generation of surface grids. Their biggest disadvantage is the non-boundary-conformality which leads to poor grid quality near boundaries, resulting in difficulty of applying boundary conditions accurately and in loss of accuracy, stability, and monotonicity, as well as in additional complications in the flow-solver, especially for unsteady flows (see section 4.4.8). These problems can sometimes [102, 103], but not always [85] be easily overcome. The only other major disadvantage is that the surface discretizations are dictated mostly by the intersection pattern with the cells instead of by the geometric properties of the surface.

Adaptation in "Cartesian grids" has been introduced, inefficiently for the general, multi-dimensional case, by line-by-line refinement in 2-D and plane-by-plane refinement in 3-D, and efficiently and convincingly by "nesting" more refined Cartesian "patches" onto parts of the original grid, possibly recursively [46, 291], in a manner similar to the use of overlapping grids (see below).

4.4.6.7 Tree-Based Methods

Tree-based methods [423] start with the smallest hexahedron, denoted the **root** of the tree, that fully bounds the computational region. The root is subdivided into eight (hence the name octree) geometrically similar hexahedra (also called cells), each of which may subsequently be divided in the same way, giving an overall recursive procedure. Each time a cell is subdivided, links are established from it to each of the eight resulting subregions, giving the characteristic "tree" data-structure.

Arbitrary resolution may be obtained at any location in the computational region by appropriately choosing the cells on which to perform the subdivision. The subdivision process is unaltered by the presence of boundaries so unless an interior boundary aligns exactly with cell boundaries it will be "cut" by cells resulting in a "non-boundary-conformal" discretization. The subdivision may be continued if desired until the intersection pattern for any cell is sufficiently "simple". Often this is considered to occur when the boundary is adequately represented by warped or curved facets formed by joining the intersection points in individual cells. The 2-D analog of the above uses rectangles instead of hexahedra and the subdivision is into four (hence the name quadtree) similar quadrilaterals.

The basic procedure described above leads to the Cartesian sub-family of tree-based methods. While the non-tree-based Cartesian methods described above are well suited to structured grids, the tree-based variant is well-suited for unstructured grids. This is the most important difference between the two variants; other differences and similarities between them are obvious.

The tetrahedral sub-family of the tree-based methods goes further than the Cartesian one by subdividing each terminal (that is, undivided or **leaf**) cell in the tree into tetrahedra or triangles, either using heuristic algorithms [21], or systematically, for example, by using a Delaunay triangulation [65]. Tri-trees have also been used to generate triangular discretizations directly [193]. In all types of the tetrahedral sub-family, boundary conformality is almost always ultimately recovered, usually by repositioning the bases of tetrahedra to match with the boundary surfaces, and by applying a smoothing procedure, typically a Laplacian filter, to improve grid quality near boundaries.

Tree-based methods have very similar advantages and disadvantages to the Cartesian methods (see above). The major difference is that tree-based methods are more efficiently adaptable. In the tetrahedral sub-family, the non-boundary conformality is somewhat alleviated but at the expense of recourse to repositioning and smoothing algorithms or other well-founded discretization algorithms, such as a Delaunay-based one, to ensure adequate grid quality.

4.4.6.8 Manual and Semi-Automatic Methods

All the methods presented so far are automatic, that is, the discretization is determined by a machine-executed algorithm. Manual specification, possibly within an interactive program, is also possible. Such programs also often establish the neighborhood and solution connectivity maps automatically, but this is not necessary. Manual and semi-automatic methods are applicable to structured as well as unstructured grids.

4.4.6.9 Methods for Surface Grids

Surface grids are not only subject to the same conditions of grid quality imposed by the flow-solver on the field grid, but are also subject to additional geometric conditions, such as the requirement for vertices to lie on surface-intersection curves.

The typical approach is to parameterize local patches of the surface, perform the discretization on the resulting planar patches in the parametric space, and then transform the discretization back to the surface patch. Either elliptic methods or one of the three main unstructured methods described above are usually used. Yet another advantage of a Delaunay triangulation is that the patch retains a least-energy "optimality" after inversion regardless of the mapping [304]!

Since special attention must be paid to geometric features that are difficult to quantify, there is still strong reliance in surface grid generation on interactive and semi-automated methods [232].

4.4.7 Classification by a Miscellany of Bases

In addition to the two main classification bases described above, there are other, but less frequently used ones, often referring to the way in which an "overall" grid is composed from constitutive, standard grids and generation methods.

Boundary-Conformal vs. Non-Boundary-Conformal: The terms are synonymous with "body-fitted" and "non-body-fitted", respectively. This classification identifies whether subregions align with (or "conform" to) the boundaries which they are closest to. The prime example of a non-boundary-conforming discretization is that generated by Cartesian methods which are discussed above together with the implications of boundary-non-conformality.

Multi-Block vs. Single-Block: Multi-block methods, first introduced in [216], employ multiple grid "blocks", each covering a different "zone" of the computational region. The blocks may either be non-overlapping [405], or, with the advantage of additional flexibility, partially-overlapping [43]. The blocks may be generated independently, but coupling to varying extents is often introduced to obtain smooth transitions (in terms of volumes and face-angles) across zonal interfaces to avoid increases in the truncation error. If there is no overlap of the blocks, special treatment to reduce conservation violation [294] is required only if the subregion edges or faces normal to the interface do not align; in the partial-overlapping case such treatment (for the solution interpolation) across the abutment [47] is always required.

Since each block is required to conform only to the geometry of its zone and since it may be discretized using the most appropriate algorithm for its zonal geometry, the multi-block approach has a strong divide-and-conquer advantage over the single-block one. The process is facilitated by judicious zoning of the computational region, which has remained the most user-time-consuming part of the process, although progress in automating or semi-automating it and automating specification of the block-to-block connectivity and boundary conditions is being made [95, 10, 321, 96]. Multi-block methods introduce complex-geometry capabilities and straightforward parallelizability while, by using mostly structured-grid blocks, largely preserving the advantages of structured grids.

Hybrid vs. "Pure": In hybrid methods, different types of grid are used in different parts of the computational region. The difference may refer to the generation method [191], to the connectivity type [406, 409], or to both. These methods hold great promise since the overall grid can exploit the strengths of different types and methods to best suit local requirements. A prime example where this is useful would be in high Reynolds number flows. The hybrid classification is a generalization of the multi-block classification and both methods share the same major complications and disadvantages: the need to connect different grids and the need for special treatment across component interfaces.

Prismatic vs. Non-Prismatic: A prismatic discretization [191, 397] is a volume discretization obtained by extruding the edges of the subregion set of a discretized closed surface away from that surface to another, possibly imaginary, outer closed surface. The enclosed regions defined by the extrusion are called "prisms", and each subregion of the inner surface therefore becomes associated with a single prism. The collection of prisms may subsequently be cut by one or more closed non-intersecting surfaces lying strictly between the innermost and outermost ones to create layers of volume subregions. Such a set of (one or more) layers is called a prismatic discretization. The connectivity sets for any two layers are isomorphic. If h-refinement (see 4.5) is performed in any subregion, it must also be applied to all the subregions that lie the same prism otherwise, the "prismatic character" will be destroyed. Prismatic grids are usually unstructured since the surface discretization is usually unstructured, but this is not a defining characteristic.

The high degree of control that can be exercised over the stretching in the normal direction and the amenability to h-adaptation (see 4.5) make this method ideally suited for viscous flows, even more so than boundary-conformal structured grids.

By Grid "Topology": This classification is applicable only to structured grids or to the blocks of a multi-block grid: O-, H-, C-, and L-topologies refer to the way the structured grid wraps around internal boundaries and whether and how the outer boundaries of the computational region join each other. Combination topologies such as C-H are also possible. The topologies in most common use in 3-D are described in [121].

Chimera or Overlapping vs. Non-Overlapping: In the chimera approach [43], different boundaries or different parts of the same boundary are individually "fitted" by single independently-generated grids which may overlap each other to arbitrary extents. Since complex geometries can always be split into simpler, more easily discretized components, this approach simplifies the grid generation task, and to the extent that only structured grids are needed. However, data duplication and the necessity for continual interpolation of the solution data across overlap zones reduce computational efficiency so the overlap must be restricted and therefore the approach is only feasible for moderately complex geometries. Interpolation also introduces the major detraction of these methods; namely, the continual introduction of conservation-violating errors into the solution. Although such error sources are arguably tolerable for steady and smooth flows, they make this method unattractive for unsteady flows with discontinuities.

Note that subregion overlap does not violate the definition of a grid since no overlap is allowed within individual grids and each grid and the corresponding solution are treated individually.

Semi-Structured vs. Structured or Unstructured: This classification is often used to distinguish between fully unstructured grids and "more regularly structured" ones. For example, if in an advancing front method, the advance is strictly layer by layer, the term "advancing layer" may be used and the method classified as a semi-structured approach. In this example, the connectivity is typically implemented as it would be for an unstructured grid although the grid is actually "structurable".

Tree-Based or Hierarchical vs. Non-Hierarchical: These terms have been defined above, and the meaning of this classification is self-evident.

Multi-Grid vs. Single-Grid: Multi-grid methods use a sequence of overlapping grids of successively increasing resolution to cover the same computational region. Each grid has a representation of the solution associated with it and these solutions are cyclically transferred between successive members of the sequence with the goal of accelerating convergence.

Traveling or Moving vs. Stationary: If any part of the computational region or its discretization changes location in the global reference frame used in the flow-solver during the computation, the grid is considered to be of the moving type. Any grid in which there is r-adaptation (see 4.5) is a moving grid.

"Gridless" vs. "Grid-Requiring": Pioneered in [32], gridless methods perform computations on a "cloud" of points instead of on subregion sets, and without explicitly establishing neighborhood connectivity maps between the points. One consequence of this is that neighborhood connectivity must be re-evaluated during every iteration or solution step, a procedure described in section 4.3 above as computationally inadvisable, with complexity at best $log(n)$ (that is, worse than the complexity of the solution algorithm for an explicit method). Although sometimes claimed to do so, this approach does not obviate the need to enforce grid quality since the relative locations of points will be subject to the almost the same set of rules that apply to subregion shapes, alignments, and density distributions.

4.4.8 Structured vs. Unstructured Grids

4.4.8.1 Computational Efficiency

Structured grids have three main efficiency advantages over unstructured grids:

(I) Lower memory requirements: Since the neighborhood connectivity tuples for a structured grid may be cast in a form that makes them identical for all interior subregions, storage is necessary of only one neighborhood connectivity tuple for all the interior subregions. The connectivity tuple for each subregion in an unstructured grid must, in the worst (and also the most common) case, be explicitly stored.

(II) Lower execution overhead: Since the repeated connectivity tuple for interior subregions of a structured grid can be reconstructed by simple arithmetic, that tuple can be encoded in the program instructions instead of being explicitly stored, using the constants $c_1, ..., c_m$ in the definition. This reduces the data transfer to and from main memory compared to unstructured grids and so increases execution speed. Since the resulting "connectivity instructions" are very simple, the length of the instruction set for loops dependent on connectivity data is not significantly increased and can easily be designed to fit in the instruction cache. By better exploitation of the data-cache, structured grids also allow better exploitation of devices used to improve processor utilization, especially instruction pipelining. Improved management of the data-cache is possible for structured grids because always at least some of the subregions that are geometrically adjacent are stored adjacently in memory. By contrast, unstructured grid implementations are characterized by frequent cache-misses.

(III) Greater Vectorizability: This follows directly from the sequential storage arrangement for structured grids compared to the more random arrangement for unstructured grids. However, the inferior parallelizability of structured grids compared to unstructured ones is more likely to offset this advantage, especially as the parallelization scales increase with time.

The relative advantages described above are hardware-dependent, and are not significantly diminished by the complications introduced by the need to exceptionally handle the boundary subregions in structured grids.

Except in the situation described in the next paragraph, the computational advantages of structured grids carry over with little loss to regularly-structured grids. In the example represented in figure 4.5, introduction of an operation for computing memory addresses of neighbors is necessary, but its simplicity and length result in little adverse consequences.

In addition to the above basic advantages, the ordering properties of structured grids offer two main composite advantages over unstructured grids: (i) in any situation involving large-scale matrix or vector operations on the subregion or solution vector sets (e.g. inversion of the implicit operator for an implicit factorization scheme), matrices can often be expressed in banded form, leading to significant gains in computational efficiency; and (ii) although several successful attempts to devise sweep paths on unstructured grids [164], line-sweep solution algorithms are more difficult to apply with unstructured grids generally, and require additional memory and processing overhead.

Structured grids enable highly efficient utilization of processing capacity, and the *best* possible utilization of memory. These grids can therefore play the additional role of presenting a benchmark for computational efficiency.

4.4.8.2 Negotiation of Complex Geometries

By introduction of appropriate branch cuts, any computational region, no matter how multiply connected can be made homeomorphic to a square in 2-D or a cube in 3-D. This requires different parts of the boundary of the cube or square to be assigned the boundary conditions corresponding to those of the pre-image boundaries and branch cuts. Since a structured grid can always be generated for a square or a cube, the inverse mapping (which always exists for a homeomorphism, and is also a homeomorphism) is guaranteed to produce a structured grid for the original geometry. Therefore, a structured grid exists for *any* computational region. However, the major obstacle to widespread use of structured grids for complex geometries is that homeomorphisms having the above properties must be expressed in a functional form before they can be used. This is difficult, the more so the more "complex" the geometry of the domain. Even more difficult is to find and express homeomorphisms that result in acceptable grid quality in their domain, that is, the computational region. An easier alternative, but with obvious penalties, to finding a homeomorphism that satisfies the above requirements fully, is to find one that additionally maps part or all of some unwanted regions (such as interiors of solid bodies) and then to apply appropriate "interior" boundary conditions in the transformed region [88].

By eliminating the constraint of isomorphism of the neighborhood tuples for interior subregions of structured grids, unstructured methods discretize the computational region directly rather than a transformed region. By doing so, they introduce additional flexibility in the spatial discretization and admit more precise and direct control over grid quality. This makes unstructured grids more suitable for complex geometries. Since "unstructured methods" require no mapping to be found, they are also more readily automated, and as described in section 4.5 more amenable to h-adaptation. These advantages turn out to be decisive for 3-D complex geometries, and they were the main motivation for the development of unstructured grids. The multi-block approach greatly facilitates the application of structured grid approaches for complex geometries, and therefore remains a viable alternative to unstructured grids for general, semi-automated discretization of complex geometries.

4.4.8.3 Influences on the Solution Quality

The solution quality, especially the accuracy, reliability, and extent of convergence, obtainable by use of structured grids is often claimed to be superior to that obtainable with unstructured grids of similar resolution. The five (geometric) features allegedly conferring this superiority and their effects on the solution quality are as follows:

- Alignment: Greater alignment of subregion edges or faces with the local flow direction reduces numerical diffusion, and is therefore an important consideration for viscous and turbulent flow computations. Alignment becomes increasingly important with increasing speed or energy of the flow. Although alignment can improve accuracy, dependence on it (that is, on directional biasing) to do so can reduce the reliability of the solution.

- Orthogonality: Greater orthogonality between subregion edges and faces decreases the truncation error in general, and some computations, for example, those involving algebraic turbulence models, appear highly sensitive to it. At solid surfaces, orthogonality allows more accurate application of boundary conditions.

- Directional scaling: Greater compatibility between the directional length scales of subregions and the directional length scales of the local flow improves accuracy and utilization of computational resources.

- Smoothness (in variation of subregion volumes): Greater smoothness improves accuracy by decreasing the truncation error.
- Boundary conformality: Loss of accuracy, stability, or reliability are often attributed to non-conformality. However, the causality is connected more analytically with the deterioration that accompanies non-conformality in one or more of the above four measures [85].

Structured grids are claimed to achieve higher quality because they inherently lead to greater satisfaction of the above features. Closer examination of these claims, however, reveals confusion between *geometric choices* and *intrinsic properties*: generation methods for unstructured grids most commonly produce triangles in 2-D and tetrahedra in 3-D. These shapes are preferable (mostly because they more readily allow control over subregion volumes) but not mandatory; quadrilaterals and hexahedra are also possible. The lack of alignment, orthogonality, and directional scaling “presumed” for unstructured grids should more correctly be associated with the geometric shapes of triangles and tetrahedra than with intrinsic properties of unstructured grids.

The greater alignment achievable with structured grids is actually alignment with boundaries and not flows; it is only advantageous when flows are strongly aligned with surfaces. Although this usually holds whenever alignment is important, e.g. for high-speed viscous flows, the following should be recognized regarding this advantage: (i) its coincidental origin; (ii) its failure for non-boundary-aligned features; and (iii) that arbitrary alignment can better be accomplished by r-adaptation based on *the solution*, not *the geometry*, and that r-adaptation is equally applicable to both classes of neighborhood-connectivity grid type. The best combination of grid quality, including alignment, orthogonality, and directional scaling, can be achieved by combined r- and h-adaptation and this is more applicable to unstructured than to structured grids. An argument similar to the above applies to the directional scaling feature, and, to a lesser extent (since some types of structured grid automatically maximize it as part of the generation process), to the orthogonality feature. Therefore, unstructured grids have the greater potential to controllably and precisely satisfy the above features to arbitrary extents.

Any advantages of structured grids related to solution-quality only hold when the comparison is made with incompletely-developed or inappropriate unstructured techniques. These apparent advantages can only decrease as unstructured techniques develop and improve.

4.4.8.4 The Verdict

The above comparison describes the relative strengths and weaknesses of the two connectivity-classified grid types. Although this comparison alone leads to a decisive choice for certain types of problems, the most general principle is that the selection for each problem must be independently evaluated as described in sections 4.4.2 and 4.4.3.

4.5 Grid Adaptation

Extensive reviews of this subject appear in [361, 328, 366, 235].

4.5.1 Definitions

Adaptation of a grid is defined as its modification to make it more "suitable" for the solution or for changes in (the geometry of) the Computational Region. In **manual adaptation**, all the required modifications are externally specified by changing the data or the algorithm during interruptions of the computation. In **automatic adaptation**, all modifications are internally determined by a fixed machine-executed algorithm that performs the entire computation. Adaptations in practice sometimes fall between these two extremes.

<u>Definition:</u> Grid $\mathcal{G} = \langle \mathcal{E}, \mathcal{NC}, \mathcal{SC} \rangle$ is **solution adaptive** iff $\mathcal{E}$ (and hence $\mathcal{G}$) may change during a computation in at least partial dependence on the solution.

By definition, $\mathcal{E}$ is changed by the introduction, deletion, deformation, or translation of any subregion in it. Introduction or deletion of subregions requires modification of $\mathcal{NC}$ and $\mathcal{SC}$, but this is not mandatory for deformations or translations.

<u>Definition:</u> Grid $\mathcal{G} = \langle \mathcal{E}, \mathcal{NC}, \mathcal{SC} \rangle$ for Computational Region CR is **geometry adaptive** iff CR may change during a computation.

A change in CR by definition requires a change in its grid.

In the first definition above, the required dependence is on the *values* of the solution; dependence on changes in the values of the solution, due to either convergence or evolution in time, follows implicitly. In the second definition, the required dependence is on *change* in (the geometry of) the Computational Region; modification of a grid in response to stationary geometric features is more appropriately considered as part of grid generation, even if actually implemented within an adaptive framework. Any problem in which the Computational Region varies with time (that is, any problem in which the boundaries of the Computational Region vary with time, including the boundaries of any solid objects immersed in the flow-field), including any problem with coupling between interfaces and flows, requires geometric adaptation.

4.5.2 The Role and Benefits of Adaptation

The role and benefits of geometric adaptation are evident from the relevant definitions in the preceding section. Therefore, except towards its end, this sub-section mostly focuses on the role and benefits of solution adaptation.

The main roles and benefits of successful solution adaptation are the reduction of the storage-space requirements, and the reduction of the overall computational-effort requirements for a computational solution which is required to meet certain local or global targets related to accuracy, resolution, fidelity, or some other desideratum. A typical such target is the satisfaction of a minimum local solution accuracy everywhere in the Computational Region. As may be expected from the above description, solution adaptation is often alternatively described in terms of an optimization or minimization procedure.

The reductions described in the preceding paragraph can be achieved because the minimum intensities of spatial or temporal resolutions required to meet the targets or objectives of a computation often vary over a Computational Region. Economic considerations in such cases imply that the density distribution of subregions in the Computational Region, and hence the distribution of computational-resource allocation, should be optimized for the targets or objectives of the computation, whatever these may be. Solution adaptation provides a means for effectively accomplishing this optimization.

A concrete example of the need for the optimization described in the preceding paragraph is a situation where the purpose of a hydrodynamic computation is to study a specific localized flow-feature with high accuracy, with little interest in the remainder of the flowfield. In this case, it would be desirable to resolve the far-field region only to the extent that the solution error in it affects the solution error in the region of main interest beyond an acceptable level. Another concrete example, and one that applies to all computations, is related to bounding the local solution error, and ultimately, the global error. For example, suppose that a computation is to satisfy a certain local truncation error requirement throughout the Computational Region. Since the truncation error in a computation depends on the local solution and on the local subregion or element dimension, it would be best to have local control over the element size in order to achieve the objectives of the computation. Without such local control, the subregion set will, for example, have to be uniformly refined throughout the Computational Region in order to meet the desired accuracy objective, causing unnecessary consumption of computational resources.

The reductions in computational effort that are achievable by solution adaptation for large-scale gasdynamic computations are typically estimated at one [38] or one to two [228] orders of magnitude compared to the case where a uniformly-refined adequate grid is used. It should be noted that, depending on the manner in which the adaptation is accomplished, such reductions are often achieved with computations that are even an order of magnitude slower on an element-by-element basis than their fastest non-adaptive variants. The actual reduction achieved with a specific computation is strongly dependent on the solution features, on the solution technique used, on the number of elements in the subregion set, and on the adaptation techniques used. In computations with large differences in length scales, or where the overall accuracy, convergence, or reliability of the solution depends strongly on resolution of the smaller length scales, the proportional savings from the circumvention of a roughly uniform resolution at the finest required length scale can be several orders of magnitude. In any situation, such reductions or savings can expand the envelope of problems computable with the available hardware capabilities.

The generic benefit of solution adaptation described above may be viewed from different angles, depending on the specific application. The remainder of this subsection is mostly devoted to a description of some of these angles for specific examples, and to a description of some of special requirements of solution adaptation for these examples.

For computations of steady-state problems, the minimal resolution distribution required within the Computational Region to meet the objectives of a computation cannot generally be predicted a-priori and, even if it could, may not easily be specified within or produced by the grid generation algorithm. Solution adaptation enables the subregion density distribution to be modified to better suit the solution as it converges. In this context, solution adaptation may be viewed as aiding the task of generating an appropriate initial grid, or as an extension to the grid generation algorithm. For steady-state problems, manual adaptation, wherein the solution algorithm is occasionally restarted with the grid modified in accordance with the latest solution, and possibly the judgment of the user, is often also feasible.

For computations of transient problems, the minimal resolution distribution required within the Computational Region to meet the objectives of a computation changes as the solution proceeds. Solution adaptation allows the subregion density distribution to continuously vary with the solution. Here, the manual alternative is less feasible, and if chosen, except for simple problems, usually results in waste of resources due to unnecessary resolution that must be introduced to compensate for inability to accurately predict the evolution of the solution.

In addition to modification of subregion density distributions, solution adaptation can also be used to improve grid quality by localized modification of subregion shapes, as mentioned in the preceding sub-section. For example, the aspect ratio, the alignment, or the precise location of cell faces may be modified to better suit the solution. The different means of modifying the grid by solution adaptation are discussed further below in this section.

For problems with moving boundaries, the primary role of adaptation, and in particular, geometric adaptation, is to *enable* a computation to be performed. Therefore, for this category of problems, geometric adaptation expands the envelope of computable problems qualitatively rather than quantitatively.

Solution-adaptation and geometric-adaptation may be combined, allowing the economic solution of physically-complex problems such as those involving fluid-structure interactions with multiple disparate flow-related length scales.

Whether for steady-state or transient problems, and whether for simple or complex problems, the advantages of automatic adaptation of the initial grid are clear, but risks are also introduced.

4.5.3 The Constituents and Workings of A Solution-Adaptive Method

In order to better design or analyze a solution-adaptive scheme, or to better predict its properties or behavior, or in order to enable more useful comparisons between different adaptive schemes, it is necessary to divide such a scheme into smaller components that can be more readily understood and analyzed. It is now generally recognized [228] that a solution-adaptive computational scheme can be conveniently separated into the following three main components:

1. **The Adaptation Indicators**: These are functions that compute pre-specified parameters, such as the local solution error, that in turn imply desired param-

eter values of the local subregion set, such as the optimal local length-scale;

2. **The Grid Optimization Criteria**: These are functions that determine what adaptation action should be performed based on the values of the adaptation indicators; and,

3. **The Adaptation Operators**: These are function that effect the desired adaptation in the grid.

Each of the three components listed above is described in detail further below in this sub-section, with emphasis on its alternative possible forms and functions.

The specific form and function of each of the three components described above cannot be chosen in isolation: the three components must be compatible, and must be chosen or designed to work well with each other.

In addition to the three components identified above, a solution-adaptive methodology requires a solution algorithm that allows for changes in the grid during the solution process, at least between iterations or time-steps.

For manual adaptation, visual evaluation of a solution is sometimes used instead of automated evaluation of adaptation indicators and grid optimization criteria. This approach has the obvious disadvantages.

4.5.3.1 Adaptation Indicators

These are functions that measure the intensity or magnitude of parameters that are to be used to drive the adaptation process. These parameters may depend on the solution values, on the temporal rates of change of the solution values, on the spatial gradients of the solution values, on measures of grid quality (such as measures of skewedness, or orthogonality), or on any other variable, whether local or global, which is desired to have an influence on the adaptation of the grid.

Ideally, an adaptation indicator should be clearly-related to the specific property against which the use of computational resources is to be optimized. Ideally, an adaptation indicator should also be a scalar or field function that can be evaluated in definite, closed form exclusively from readily-available quantities that depend only on the discrete solution or other known quantities. An adaptation indicator that meets the ideals described above could be used to guide the grid adaptation process robustly and accurately, to achieve the desired optimization of the use of computational resources.

From the practical point of view, other desirable features that should be possessed by an adaptation indicator are as follows [229]:

1. Computational economy;
2. Non-dimensionality;
3. Boundedness in value; and,
4. Ability to detect not only strong and dominant features, but also important but weak features.

The most valuable and the most commonly sought adaptation indicators are ones that attempt to measure the solution error, especially ones that do so using only the discrete solution data. This holds even if these adaptation indicators do not fully satisfy the remaining ideals described in the preceding paragraph. Measures of the solution error include, for example, the local truncation, local dispersion, or local amplification error. Because estimates of the solution error are the most common parameters used in adaptation indicators, the latter are typically also called error indicators or error detectors.

Having an accurate estimate of the error in a discrete solution is valuable because such an estimate is a key component in any computational methodology that provides the user of the computational solution with a numerical criterion to assess the utility of the solution, and to assess the faith that can be placed in it for its end uses. The practical value of an adaptation indicator derived from the solution error is therefore very high.

Unfortunately, the solution error for most computations in applied practice is very difficult to determine, and therefore very difficult to incorporate into an adaptation indicator. Accurate measures of the solution error can typically only be developed for simple, scalar, linear problems. For computational solution of systems of nonlinear equations using a Finite-Volume Scheme, there are no general methods that can be applied in all circumstances to accurately obtain a-priori estimates of the solution error or even the truncation error. However, there are several methods of limited validity that can give reasonable estimates in a wide variety of cases.

Among the most popular methods of estimating the truncation error, even for systems of Partial Differential Equations, are those based on the Taylor Series Expansion of the discrete equations solved (as shown in a brief example in the next sub-sub-section), including methods based on the Richardson Extrapolation Technique [307]. Methods based on the Richardson Extrapolation Technique estimate the error by grid convergence studies, or by comparing the convergence rates with different stencils [307]. The family of Richardson Extrapolation Techniques, however, is sensitive to local length-scale variations, and loses its validity, accuracy, and reliability in regions of high gradient, which is where accuracy in estimating the error is usually needed the most. These techniques are also more difficult to use accurately for computations of transient problems, or for computations with mixed orders of accuracy.

Because of the difficulty of computing the error in discrete solutions, many adaptation indicators revert to the simple assumption that regions of high gradient in the discrete solution are associated with regions of high local error in the solution. This point is re-iterated and discussed further in the next sub-sub-section.

The System of Euler Equations is an example of a coupled, nonlinear system which presents many of the difficulties described in the preceding three paragraphs for the development of an ideal adaptation indicator based on the solution error. The most suitable and the most commonly-used adaptation indicators for computations with The System of Euler Equations are discussed in detail in Section 6.5.

4.5.3.2 Grid Optimization Criteria

These are functions which determine, from the values of the adaptation indicators, the following: (i) whether the grid requires adaptation in a certain region; and, (ii) the type of adaptation or modification that should be performed. The type of adaptation that may be performed depends on the adaptive scheme: it could be a change in the local length scale, such as in local refinement or local coarsening, it could be a relocation in the Computational Region of a set of cells or vertices, it could be a re-alignment or localized deformation of one or more cells, or it could even be a change in the local solution scheme. These possibilities are discussed in more detail in the next sub-section.

The grid optimization criteria typically operate by computing the optimal value for some key parameters, say, of the local subregion geometry, such as the local length scale, from the values of the adaptation indicators, and then comparing this optimal value with the corresponding actual value. Based on the comparison, the grid optimization criteria determine the necessary action to be performed in the relevant subregion set.

Grid optimization criteria are often expressed in terms of interval tests, tolerance checks, or limit checks, and they must incorporate in them the complete manner in which the optimality of the grid is defined in terms of the adaptation indicators.

Perhaps the most commonly used grid optimization criteria are ones motivated by the **Principle of Equidistribution of Error** [365], which requires that the spatial resolution should be distributed over the Computational Region so as to give a uniform distribution of the local error throughout the Computational Region.

As an example of the construction of a grid optimization criterion in general, and from the "equidistribution" principle in particular, consider the special case of a smooth scalar field function ϕ, which is approximated by the discrete solution $\tilde{\phi}$. In the limit of grid resolution for a scheme that is first-order-accurate in space, the local error, ϵ, which given by $\epsilon = \phi - \tilde{\phi}$, is accurately approximated by the truncation error, and hence from the corresponding Taylor Expansion, via the equation

$$\epsilon = |\nabla\phi| \simeq \left|\nabla\tilde{\phi}\right| \tag{4.1}$$

where all terms are as defined above. In this case, an appropriate choice for the adaptation indicator is clearly the local error ϵ in Equation 4.1. The equidistribution principle here can be expressed as a relation between the local error and the optimum local length-scale. Since the numerical scheme considered in this example is first-order-accurate in space, the relation will be of the specific form

$$\epsilon d_{opt} = k \tag{4.2}$$

where ϵ is as in Equation 4.1, d_{opt} is the optimum local length scale, given, for example, by the optimum length-scale or dimension of the subregion or the computational cell being analyzed, k is a global constant for the Computational Region, and all the other terms are as defined above. Equation 4.2 again follows from the Taylor Expansion for the special case being considered here. A very appropriate grid optimization criterion in this case would be one that is based on the evaluation of the equation

$$d_{opt} = \frac{k}{\left|\nabla\tilde{\phi}\right|}$$

where all terms are as defined above. Thus, d_{opt} is the optimum local length scale with which the actual local length scale would be compared to determine the action to be taken for the relevant subregion. The grid optimization criterion in this case may be specifically formulated in the following conditional form, where d is the actual local length scale: (i) if $d \geq \alpha d_{opt}$, where $\alpha > 1$ is some numerical constant, say, equal to 2.5, then refine the local computational cell; otherwise, (ii) if $d \leq \beta d_{opt}$, where $\beta < 1$ is another numerical constant, say, equal to 0.4, then coarsen the local computational cell.

As can be predicted from its definition, the "equidistribution" principle is difficult to implement for complex systems of equations, mostly because of the difficulty of approximating the solution error with sufficient accuracy for such systems (that is, mostly because of the difficulty of formulating suitable adaptation indicators for such systems, as discussed in the preceding sub-sub-section). For example, error estimates based on a Taylor Expansion, such as that of Equation 4.1 can usually only be derived if the governing equations and the solution algorithm are simple and well understood, and if the solution is smooth. In non-smooth regions (such as across discontinuities, which in the limit introduce an infinite gradient), the basis for using errors derived from a Taylor Expansion is lost, and the values of the approximation to the gradient, for example, $\left|\nabla\tilde{\phi}\right|$ in Equation 4.1, become a highly misleading indicator of the error.

Similar difficulties arise with nonlinear and coupled systems of equations. The use of techniques based on the Richardson Extrapolation within an adaptation indicator to estimate the error, and the limitations of such techniques, were briefly discussed in the preceding sub-sub-section.

In addition to the difficulty of estimating the error for correct use of the "equidistribution" principle, another obstacle to the widespread use of the principle is that its basis is not completely sound because the relation between local and global error is not straightforward, especially for nonlinear or hyperbolic problems. Furthermore, in many situations, the "equidistribution" principle is not the most appropriate because the objective of the computation is better met by a non-uniform distribution of error. Still, the main aim of the equidistribution principle, reduction of the local error in regions where it is believed high, remains the basis underlying most of the adaptation schemes in common use for hydrodynamic computations.

Because of the difficulty of formulating adaptation indicators that accurately measure the solution error, most adaptive schemes used in hydrodynamic computations are founded on a widely-accepted, but approximate, general principle relating the truncation component of the error to the local element dimension. This principle is that, depending on the order of accuracy of a scheme, the local truncation error is generally proportional to the spatial or temporal first- or higher-derivatives of the solution. If accurate expressions for the truncation error can be formulated, the principle of equidistribution of error can, if desired, be applied directly as the grid optimization criterion, especially if the solution is smooth. Otherwise, the grid optimization criterion may simply be constructed to require a higher resolution (that is, a smaller element dimension) in regions where the magnitudes of the first- or higher-gradients of some chosen combination of solution-dependent variables are greater. In other words, a general guiding principle is to assume an error estimate of the form of Equation 4.1, despite the limitations of this form as discussed above. If the latter approach is adopted, the choice of variables whose gradient is being assessed must reflect the physical phenomena and the system of equations involved. This approach is often characterized by the use of ad hoc physical arguments and the adoption of an ad hoc functional relationship associating the optimal element dimension to the magnitude of the gradient. These ad hoc choices invariably limit the generality and the effectiveness of the overall adaptive scheme compared to the ideal, optimal adaptive scheme.

For adaptive methods that use local refinement and coarsening (that is, h-adaptation, as defined and described in detail in the next sub-section), four different forms of the ad hoc types of grid optimization criteria described above are commonly used in current practice. Because of the direct relevance of such methods to the work presented in this dissertation, and because of their relative success in generating acceptably robust and efficient solution-adaptive techniques, each of these four methods is briefly outlined below.

The first, and possibly most popular, grid optimization criterion for h-adaptation, is a natural and intuitive methodology, apparently first published in Reference [189]. The principle of the methodology is as follows: if any of the adaptation indicators for any given cell exceeds a threshold associated with that indicator, then the cell is determined to be under-refined, and is targeted for refinement. For correct behavior, the threshold should also be partly determined by the dimension of the cell. Similarly, if all of the adaptation indicators fall below their corresponding thresholds for the dimension of that cell, then the cell is determined to be over-refined, and is targeted for coarsening. The grid refinement criterion of this methodology may be expressed through the following test on the value of the left-hand side

$$|\epsilon_{\phi c}| \geq \mu_\phi + k_\phi^r \sigma_\phi \tag{4.3}$$

while the grid coarsening criterion may be expressed through the following test on the value of the left-hand side

$$|\epsilon_{\phi c}| < \mu_\phi - k_\phi^c \sigma_\phi \tag{4.4}$$

where, for both of the above expressions, $|\epsilon_\phi|$ represents the value of the adaptation indicator for the variable ϕ, $|\epsilon_{\phi c}|$ represents the value of that adaptation indicator in cell c, μ_ϕ is the arithmetic average of the adaptation indicator value for variable ϕ across all the cells in the grid, $0 \leq k_\phi^r$ and $0 \leq k_\phi^c$ are some constants that respectively scale the refinement and coarsening adaptation criteria for the variable ϕ and for the given cell size, and σ_ϕ is the statistical standard deviation of the adaptation indicator for the variable ϕ across all the cells in the grid. The effect of the cell dimension may either be accounted for in the values of k_ϕ^r and k_ϕ^c by formulating these terms to include a divisor by, say, the length scale of the cell raised to some power greater than one, or, alternatively, by formulating the adaptation indicator to include a multiplier of the length scale of the cell raised to some power greater than one.

The quantities μ_ϕ and σ_ϕ in Equations 4.3 and 4.4 are computed using the respective standard discrete definitions; namely:

$$\mu_\phi = \frac{1}{n_c} \sum_{c=1}^{n_c} |\epsilon_{\phi c}|$$

and

$$\sigma_\phi = \sqrt{\frac{1}{n_c} \sum_{c=1}^{n_c} \left(|\epsilon_{\phi c}| - \mu_\phi\right)^2}$$

where n_c is the total number of cells in the grid, and all other terms are as defined above. Since the k_ϕ^r and k_ϕ^c may take independent values for different variables ϕ, the adaptive methodology allows for selective emphasis on the adaptation of different features.

A special case of the grid optimization criterion just described is when μ_ϕ is not computed and not considered, and this case is equivalent to setting $\mu_\phi = 0$ in Equations 4.3 and 4.4.

The second grid optimization criterion is based on testing the following condition for the left-hand side:

$$\frac{|\epsilon_{\phi c}| - \epsilon_{\phi min}}{\epsilon_{\phi max} - \epsilon_{\phi min}} \geq \kappa_\phi \tag{4.5}$$

where $\epsilon_{\phi min} = min\{|\epsilon_{\phi 1}|, ..., |\epsilon_{\phi n_c}|\}$ is the minimum value of the adaptation indicator for the variable ϕ among all the cells in the grid, $\epsilon_{\phi max} = max\{|\epsilon_{\phi 1}|, ..., |\epsilon_{\phi n_c}|\}$ is the maximum value of the adaptation indicator for the variable ϕ among all the cells in the grid, κ_ϕ is effectively the adaptation threshold, and n_c is as defined above.

The third grid optimization criterion is based on testing the value of the following expression:

$$\frac{l_c^2 \, |\nabla^2 \epsilon_{\phi c}|}{|\nabla \epsilon_{\phi c}| + \kappa \, |\tilde{\epsilon_{\phi c}}|} \tag{4.6}$$

where l_c is the cell dimension, $\tilde{\epsilon_{\phi c}}$ is the arithmetic mean of the value of $\epsilon_{\phi c}$ in cell c and all its immediate neighbors, $\kappa \leq 1$ is a constant used to tune the optimization criterion, and all other terms are as defined above. Including the first derivative in the denominator prevents the increasing of the attraction of refinement by discontinuities as they become more resolved, while including the mean in the denominator prevents maxima from excessively attracting refinement. A typical value for κ is 0.1.

The fourth grid optimization criterion is based on evaluating the fluxes at the cell faces using a method whose order of accuracy is higher than that of the method being used in the computation, and then comparing the higher-order-accurate and lower-order-accurate flux values: if the difference exceeds a given threshold, then refinement is indicated; otherwise, refinement is not indicated. Using this technique is only feasible if a higher-order-accurate approximation for the fluxes is available. In this method, the adaptation indicator is the difference between the lower-order-accurate and the higher-order-accurate flux values, while the grid optimization criterion is the set of rules that compare the value of the adaptation indicators with the values of the appropriate thresholds, and then determine whether to refine or to coarsen a cell.

4.5.3.3 Adaptation Operators

These are the operators that perform the required modifications to the grid to increase its optimality in accordance with the requirements of the chosen adaptation indicators and the chosen grid optimization criteria. In addition to performing the required refinement, coarsening, relocation or other adaptation modification, these operators must perform all the required subsidiary operations in the adaptive scheme. Examples of such subsidiary operations include the execution of any necessary modifications in the data-structures related to the grid, or in the values of global variables.

4.5.4 Classification and Comparison of Adaptation Methods

Once the process of determining the desired local length scales in terms of the solution or geometry variables has been established, in accordance with whatever adaptation indicators and grid optimization criteria have been selected, the next step is to adapt the grid in response to differences between the extant and the "recommended" or "desired" scales, possibly in accordance with pre-specified thresholds. There are four distinct methods, possibly usable in combination, for doing so:

1. r-adaptation;

2. h-adaptation;

3. p-adaptation; and

4. adaptation by grid regeneration (or "remeshing").

In r-adaptation (relocation- or redistribution-adaptation) variations in subregion density are accomplished by drawing subregions away from regions where lower resolution is required into regions where higher resolution is required [166, 229]. Only translation and deformation are used, typically guided by an optimization or a weighted Laplacian function [58], or by a "spring analogy" formulation [31, 259, 117].

This method is most suited for steady flows. For unsteady flows, application of r-adaptation is made more involved by the need to additionally solve grid motion equations, and to explicitly enforce geometric conservation laws, and because grid distortion, except for simple flows, usually destroys grid quality, especially near boundaries, even if the functions controlling grid motion are designed to preserve grid quality. Since the total available resolution remains fixed, the method is especially disadvantageous for unsteady flows in which multiple additional flow-features are introduced, but has nevertheless been demonstrated for such applications [177]. R-adaptation is particularly appropriate for structured grids since this technique does not require any modification in $\mathcal{NC}$ or $\mathcal{SC}$ of $\mathcal{G}$, but the deformations must be controlled to prevent the disappearance of interfaces between subregions. R-adaptation readily allows alignment, orientation, and directional scaling of subregions to suit local flow-features, making it especially applicable to shock-fitting.

In h-adaptation (where h refers to the local subregion length scale) variations in subregion density are accomplished by subdivision or consolidation of subregions. Therefore, modification of $\mathcal{NC}$ and $\mathcal{SC}$ is mandatory. In some applications, the subdivision is implemented by overlaying subregions with finer discretizations which effectively displace the overlayed subregions in the subregion set [48, 291] (even through a "solution" may be retained for the original subregions for implementation purposes.

A clear advantage of h-adaptation is the possibility of spatial localization of the modifications within the subregion set. For simple, steady, flows about simple geometries, this is not a decisive advantage over r-adaptation [97]. For unsteady flows, however, this localization, and the possibility of using stationary grids (where conservation and grid quality can easily be maintained) make h-adaptation generally more suitable. For unsteady flows that develop multiple additional flow-features a third and possibly decisive advantage of h-adaptation is that the total resolution of the grid is unrestricted. H-adaptation is readily implemented for unstructured grids. The introduction of two new subregions by subdivision of subregion number 13 in figure 4.4 is illustrated by the dashed lines. The neighborhood connectivity tuples for subregions 20, 26, 14, 24, 4, 1, and 13 must all be modified to reflect the new neighborhood patterns, and additional neighborhood-connectivity tuples must be introduced for the two new subregions. All these changes are, however, easy to perform since they mostly require reassignment of address fields in the data-structures.

Alignment of subregion edges or faces and directional scaling of subregions is much more difficult to implement with h- than it is with r-adaptation. H-adaptation cannot be applied in an efficient manner to structured grids. In particular, the only way to implement h-adaptation without destroying the isomorphism of neighborhood patterns is by line-refinement: the introduction of a complete grid line (see the dashed line in figure 4.3) and a reordering of all affected cells in memory. The disadvantages of this is some loss of local control and reduced gains in efficiency. Approaches described as h-adaptations of structured grids without using line-refinement [24] actually disrupt the isomorphism and therefore, despite successfully retaining most of the advantages of structured grids, cannot strictly be classified as structured-grid methods.

In p-adaptation, the order of the function representing variables in the subregion is adjusted without any change in the geometry or connectivity sets. This type of adaptation does not comply with the definitions given above and the appropriate changes can be carried out fully in the flow-solver; it is included here only for completeness. P-adaptation is often combined with r- and h-adaptation.

Combinations of r- and h-adaptation have also been developed [235] and shown to preserve the advantages of both methods for unstructured grids. Particularly useful is augmenting all the advantages of h-adaptation with the re-alignment and directional scaling capabilities of r-adaptation. Given that the ultimate aim of adaptation is to provide the optimum cell size, location, and alignment, these combined approaches are probably the most promising ones in the long-run.

Remeshing [277] involves regeneration of part (local remeshing) or all (global remeshing) of the grid in accordance with guidance from measures derived from the solution on the original grid. Global remeshing is computationally expensive, and is therefore more appropriate for steady state computations where only few remeshings may be required. Local remeshing can be fully automated and is well-suited to unsteady flows, including those with moving boundaries [297]. Since the solution must be interpolated between the old subregion set and its replacement, remeshing (whether local or global) has the disadvantages of conservation violations and introduction of dissipation.

Instead of the adaptation indicators being fixed throughout at least part of the computation, it is possible to have the indicators themselves adapt to the solution or to other aspects of the computation. The “intelligence” of the behavior of an algorithm incorporating a feature so simple in concept and in software implementation is quite astonishing: computations can be performed that provide the “best” solution, according to the given indicators, using a pre-specified number of subregions, or a pre-specified execution time. Adaptive determination of adaptation criteria reduces the need for manual intervention and could be another classification basis for adaptive schemes.

4.5.5 Limitations, Shortcomings, and Pitfalls of Adaptation

As described above, there are no reliable a-priori or a-posteriori error estimates for the typical nonlinear or hyperbolic problems encountered in Computational Fluid Dynamics. Therefore, no good estimates of the optimal resolution distribution in a Computational Region are possible, and so no sound basis for adaptation indicators exists at present. The selection of adaptation indicators therefore still relies heavily on intuition, insight, and a good understanding of the physical phenomena being adapted. Errors in judgment often adversely affect a computation or cause it to fail.

For roughly uniform resolution distributions, grid refinement studies can confidently be carried out; for adaptive grids, such studies may well be misleading [402]. In the absence of good error estimators, a good guiding principle to avoid this danger is to ensure that the adaptation scheme refines all parts of the Computational Region as the number of subregions increases and, particularly, that the local length scale at any point in the Computational Region, not just in its most highly resolved parts, goes to zero as the number of subregions goes to infinity.

The additional potential complications, dangers, and problem-specific user-specified parameters (especially those for the adaptation indicators), that accompany adaptive computations demand greater skill from the user. This is viewed negatively since a shift in the opposite direction is practically desired.

4.6 Retrospective Remarks

This chapter presented a review of grids, grid generation (which includes the discretization of the computational region), and grid adaptation, and described their roles in computational simulation. All the important issues were discussed, and various comparisons were made between alternatives. Although, for brevity, the presentation was in the explicit context of only the Finite-Volume and Finite-Element methods, extension to the Finite-Difference method is implicit and equally covered: the required set of points can be selected from appropriate vertices or internal points of subregions.

The envelope of computable problems in Computational Fluid Dynamics was described as having three main types of theoretical “front”: (i) geometric complexity; (ii) complexity of the physical phenomena and governing equations reliably and accurately solvable; and (iii) the usefully-employable hardware capabilities. It was shown how the quest to expand this envelope along fronts of the first two types introduces the theme of computational efficiency pervasively, and that expansion on any one type of front often affects performance on another. It was shown how all the families of grid forms and generation techniques, each most suited to a certain class of problem and purpose, actually reflect certain choices of compromise within the envelope. It was shown how the strengths and weaknesses of each of these families follow from their intrinsic properties, and in particular, it was shown how choices at the level of hardware representation, in the form of connectivity and storage maps, introduce permanent, intrinsic characteristics observable at higher levels of abstraction.

This chapter described the scaffolding of guiding principles which must be observed in the selection and development of a grid type, and the corresponding generation and adaptation algorithms for a specialized application. In this work, the application involves the solution of the system of Euler equations with moving boundaries and complex geometries, and the grid type is the Quadtree-based Adaptive Cartesian type. The construction proceeds in Chapter VI.

CHAPTER V

Representation and Modeling of the Geometry and Motion of Boundaries

This chapter describes and discusses the specific methods used in this work to define and represent the geometries of (stationary and moving) boundaries, and to perform topological transformations in boundaries. The manner in which the geometries of "cut" computational cells are defined from the intersections between the corresponding Cartesian cells and the boundaries that "cut" them is also discussed, with emphasis on the special simplifications possible with axis-aligned grid-lines, and the special requirements of and treatments for cut cells traversed by moving boundaries. This chapter also describes and discusses the three different methods of specifying or effecting the motion or deformation of boundaries in this work: (i) a method for prescribed-displacement motion; (ii) a method for prescribed-velocity motion; and, (iii) a method for fluid-coupled motion. The formulation and implementation of each of these three methods for each of the two types of boundary allowed in this work; namely, the rigid type, and the deformable type, are described in detail, including the treatment adopted for the solution of the Equations of Motion for the fluid-coupled motion method.

5.1 Categorization of Boundaries and Boundary Representations

Each surface or edge that is identified as a boundary surface or edge in this work, that is, each surface or edge on which boundary conditions are applied, falls into one of two categories: (i) the category of **exterior boundaries**; and, (ii) the category of **interior boundaries**. These two categories and their corresponding data-structural and geometric representations and treatments are described in the next two paragraphs, and the presentations given and the distinctions made there may be more readily understood by reference to, for example, Figure 8.3, Figure 8.21, Figure 8.97, Figure 8.99, or any of the many other relevant figures in Chapter VIII that show a full Computational Region with at least one body or flow obstacle in it.

The category of **exterior boundaries** is the one to which only the four edges or sides of the Cartesian Square, which always represents the Computational Region in this work, belong. The formation of the Cartesian Square that represents the Computational Region and the manner in which the grid is generated within it are outlined in Chapter I and explained in detail in Chapter VI. The four exterior boundaries do not have any special, explicit, separate representation in this work. That is, there are no separate data-structures that exclusively store or represent the four sides of the Cartesian Square or any of their geometric or other properties. Instead, the individual cell faces or edges that make up each of these four boundaries are represented as part of the grid, in the same way with which the interior cell faces and edges of the grid are represented [1]. The four exterior boundaries are implicitly

[1] Of course, the cell faces or edges which belong to the exterior boundaries have a special status as the cell faces or edges on which boundary conditions are applied, but this treatment is done in an implicit manner, through the use of ghost cells, as explained in more detail in Chapter III.

created at the start of a computation, during the definition of the Computational Region and during the grid generation process, and they remain stationary for all time. No models are therefore needed to represent their motion or transformation. This holds true even if the Computational Region (and hence the Cartesian Square) is expanded by the "re-rooting" process outlined in Chapter III and explained in detail in Chapter VI.

The category of **interior boundaries** is the one to which belong all boundaries that are not one of the four exterior boundaries. The interior boundaries must all lie within the Computational Region (that is, within the Cartesian Square in which the grid is constructed). A typical interior boundary is one that represents the surface of a body about which a flow is to be computed. Unlike the exterior boundaries, the interior boundaries in this work may be geometrically complex, may deform or move anywhere within the Computational Region, may undergo topologic transformations, and may even interact with each other. Therefore, a sufficiently complex system is needed to represent them and to model their transformations and motions. The remaining sections of this chapter are mostly concerned with the data-structural and geometric representation and treatment of interior boundaries, and with the modeling of their motions, deformations, and topologic transformations.

The preceding two paragraphs make it clear that the representation and treatment of interior boundaries is completely independent of and distinct from the representation and treatment of exterior boundaries.

5.2 Definition and Representation of the Geometry of Interior Boundaries

In this work, each boundary in the interior of the Computational Region (that is, each interior boundary) is represented in the form of a **Composite Parametric Cubic Spline Curve** (CPCSC) [2], which consists of a set of at least two individual cubic spline segments, connected end to end to form a closed loop. In this work, the closed loop is required to be of genus 0 (that is, the closed loop is required to have no self-intersections, and thus to separate $\mathbf{R^2}$ into exactly two regions, each of which is topologically connected). The single topologically-connected region enclosed by the closed loop is called the **interior of the boundary**, while the single topologically-connected region enclosed between the closed loop and the outer boundary of the Computational Region is called the **exterior of the boundary**. Each of the two topologically-connected regions separated by an interior boundary may be occupied by either a solid or a fluid body, giving rise to four different possible configurations. The construction and parameterization for the individual splines or spline segments in the Composite Parametric Cubic Spline Curve is described in detail below.

The Composite Parametric Cubic Spline Curve representation for each interior boundary present in a computation is determined from an input that consists of a finite ordered sequence of elements in which each element consists of one coordinate pair in $\mathbf{R^2}$ and one integer. The coordinate pairs specify the locations of the **control points** of the Composite Parametric Cubic Spline Curve, and these are the points at which the successive individual cubic splines of the composite spline must terminate. The integers specify the smoothness required at the corresponding control points by specifying the indices i of the continuity levels C^i that are to be achieved at

[2]Such a representation is often abbreviated to "Composite Parametric Cubic Spline" (CPCS).

those points. The value of i must satisfy the condition $0 \leq i \leq 2$, corresponding to continuity levels C^0, C^1, or C^2 at each control point. The continuity level within each individual cubic spline segment is always C^2.

In principle, the control points may lie anywhere in $\mathbf{R^2}$. In this work, however, each individual cubic spline curve in each composite cubic spline curve is required to lie wholly within the Computational Region (that is, every point in every individual cubic spline curve must lie in the Computational Region, implying that the control points must also lie in the Computational Region). This constraint is imposed to enable a more uniform treatment of various topological and geometric evaluations, such as point-set-membership and intersection-point calculations, as described, for example, in Section 5.3.

As implied above, the representation of each interior boundary present in a computation is constructed from the corresponding ordered sequence of elements described above by interpolating each successive pair of control points of that boundary with a parameterized cubic spline that satisfies the required continuity level at each of those control points. The smoothness of the Composite Parametric Cubic Spline Curve at each of its control points is therefore effectively specified in terms of the matching of the local slopes and, if the continuity level is C^2, the local curvatures as well, of the two individual spline curves that are joined at that control point.

The construction procedure described above for the Composite Parametric Cubic Spline Curve representation of an interior boundary reduces to the following requirements for the individual splines in that representation: each spline must terminate at the two control points that it interpolates and must satisfy a tangency constraint at each of its two end points (or its two control points). These four constraints together fully specify the cubic spline between any two successive control points, in accordance with the Hermite Interpolation Formula, which expresses the spline equation in terms of the two end-point position vectors and the two end-point tangent vectors; namely,

$$\vec{P}(u) = f_0(u)\vec{P}_0 + f_1(u)\vec{P}_1 + g_0(u)\vec{T}_0 + g_1(u)\vec{T}_1 \tag{5.1}$$

where f_0, f_1, g_0, and g_1 are the (cubic) Hermite Basis Functions of the interpolant, where the vector $\vec{P}(u) = (x(u), y(u))$ refers to the coordinate position of a point in the spline as a function of the parameterization variable u (which is defined in more detail below), where the vectors $\vec{P}_0$ and $\vec{P}_1$ refer to the positions of end-points 0 and 1, respectively, at which the spline terminates, and the vectors $\vec{T}_0 = \frac{d\vec{P}(u)}{du}|_{\vec{P}_0}$ and $\vec{T}_1 = \frac{d\vec{P}(u)}{du}|_{\vec{P}_1}$ refer to the tangent vectors at the end-points 0 and 1, respectively.

After the application of appropriate constraints on the basis functions to give the correct end-point conditions [254], the formula of Equation 5.1 can be re-written in the matrix form

$$\vec{P}(u) = (\vec{P}_0, \vec{P}_1, l\vec{T}_0, l\vec{T}_1) \begin{pmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} u^3 \\ u^2 \\ u \\ 1 \end{pmatrix},$$

where l is the chord-length between the end-points 0 and 1, and where all other symbols denote the meanings given to them above. The most convenient choice for the parameterization variable u is the distance (whether normalized or not) along the chord connecting the points $\vec{P}_0$ and $\vec{P}_1$.

The matrix form given above leads directly to explicit equations for the spline interpolant between any two successive control points which, in scalar form, may be expressed through

$$x(u) = A_x u^3 + B_x u^2 + C_x u + D_x \tag{5.2}$$

$$y(u) = A_y u^3 + B_y u^2 + C_y u + D_y \tag{5.3}$$

where here u, the parameterization variable or "parameterization coordinate", is the normalized displacement along the chord of the spline, where $x(u)$ and $y(u)$ represent the x- and y-coordinates of the generic point in the spline curve, and where A_x, B_x, C_x, D_x, A_y, B_y, C_y, and D_y are the coefficients of the cubic polynomials, as indicated in Equations 5.2 and 5.3.

In terms of the position vectors and tangent vectors at the end-points, the coefficients of the interpolating cubics evaluate to

$$A_x = 2x_0 - 2x_1 + d_0 x_0^{'} + d_0 x_1^{'}$$

$$B_x = -3x_0 + 3x_1 - 2d_0 x_0^{'} - d_0 x_1^{'}$$

$$C_x = d_0 x_0^{'}$$

$$D_x = x_0$$

$$A_y = 2y_0 - 2y_1 + d_0 y_0^{'} + d_0 y_1^{'}$$

$$B_y = -3y_0 + 3y_1 - 2d_0 y_0^{'} - d_0 y_1^{'}$$

$$C_y = d_0 y_0^{'}$$

$$D_y = y_0$$

where x_0 and y_0 denote, respectively, the x- and y-coordinates of end-point 0, x_1 and y_1 denote, respectively, the x- and y-coordinates of end-point 1, $x_0^{'}$ and $y_0^{'}$ denote, respectively, $\frac{dx}{du}$ and $\frac{dy}{du}$ at end-point 0, $x_1^{'}$ and $y_1^{'}$ denote, respectively, $\frac{dx}{du}$ and $\frac{dy}{du}$ at end-point 1, $d_0 = l$, and all other terms are as defined above.

As evident in the explicit forms of Equations 5.2 and 5.3, given the end-point coordinates, the only remaining unknowns required to establish the values of the coefficients of the individual spline interpolants are the slopes at the end-points. Since specification of the smoothness at the end points only imposes constraints on the matching of the slopes at the end-points, rather than on their exact values, the specific values of the slopes must be determined by solving a global equation that includes all the control points. The slopes are determined by assembling the individual equations for all the splines in the Composite Parametric Cubic Spline Curve system into a single matrix equation of the form

$$\vec{\vec{S}}\vec{\tau}_\phi = \vec{C}_\phi \tag{5.4}$$

where the matrix $\vec{\vec{S}}$ is given by

$$\vec{\vec{S}} = \begin{pmatrix} \gamma_0 & \beta_1 & 0 & \dots & 0 & 0 & 0 \\ \beta_1 & \gamma_1 & \beta_2 & \dots & 0 & 0 & 0 \\ 0 & \beta_2 & \gamma_2 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \gamma_{n-4} & \beta_{n-3} & 0 \\ 0 & 0 & 0 & \dots & \beta_{n-3} & \gamma_{n-3} & \beta_{n-2} \\ 0 & 0 & 0 & \dots & 0 & \beta_{n-2} & \gamma_{n-2} \end{pmatrix}$$

where β_i is the inverse of the chord length joining control points i and $i+1$ (that is, $\beta_i = \frac{1}{l_i}$), where $\gamma_i = 2(\beta_i + \beta_{i+1})$, where the vector $\vec{\tau}_\phi$ is given by

$$\vec{\tau}_\phi = \left(\vec{T}_1^\phi, \vec{T}_2^\phi, ..., \vec{T}_{n-1}^\phi\right)^T,$$

where the subscript and super-script ϕ is either x or y to indicate, respectively, either the x- or the y-component of $\vec{T}_j$ (which, as explained above is the value of $\vec{T}$ at the

j^{th} control point), where the vector $\vec{C}_\phi$ is given by

$$\vec{C}_\phi = \begin{pmatrix} 3(\beta_0^2 l_0 + \beta_1^2 l_1) - \beta_0 \vec{T}_0^\phi \\ 3(\beta_1^2 l_1 + \beta_2^2 l_2) \\ \vdots \\ 3(\beta_{n-3}^2 l_{n-3} + \beta_{n-2}^2 l_{n-2}) \\ 3(\beta_{n-2}^2 l_{n-2} + \beta_1^2 l_{n-1}) - \beta_{n-1} \vec{T}_n^\phi \end{pmatrix}$$

and where all other terms are as defined above.

As implied above, note again that Equation 5.4 represents two different systems of equations, one for the x- and one for the y-values of the slopes.

The matrix equation given above is obtained by differentiation and manipulation of the Hermite Interpolation Formula, and is valid for the case of zero second derivatives at the first and last control points (which must always be coincident in this work, in order to generate closed loops) of the composite spline system. However, different boundary conditions may also be applied at the first and last control points to give slight variants of the above matrix equation, and more specifically, slight variants on the elements of $\vec{\vec{S}}$ and $\vec{C}_\phi$ as given above. In all cases, however, the inverse of the matrix is guaranteed to exist, and the solution is guaranteed to be unique. In this work, that matrix equation, Equation 5.4, is solved for the slopes using a standard tridiagonal solver based on the Thomas Algorithm.

If C^0 is specified for any two successive control points in the sequence, then the interpolating spline connecting these two control points will degenerate to an "exactly-linear" segment. The spline between any two successive control points may also degenerate to an "exactly-quadratic" segment. Such degeneracies, however, are not allowed to occur fully in this work. Instead, when they are detected, the magnitude of the cubic term is maintained above a negligibly-small number, a number so small that it prevents the degeneracy from fully occurring but only with a negligible amplitude of distortion relative to the fully-degenerate shape of the spline curve. This intentional prevention of degeneracies ensures that in all cases, a non-zero coefficient is retained for the cubic term, enabling the adoption of a single, uniform, and efficient treatment throughout the geometric computations that involve splines, even in cases of formal degeneracy to lower-order splines.

As explained above, the consecutive individual cubic splines that connect the control points of each interior boundary are combined together to form a single system that is called a Composite Parametric Cubic Spline Curve. The Composite Parametric Cubic Spline Curve representation is complex enough to allow smoothness of up to C^2 everywhere along an interior boundary, but simple enough to allow fast computation of the intersections of such a boundary with the axis-aligned grid-lines of a Cartesian grid. This combination makes the Composite Parametric Cubic Spline Curve representation a highly suitable choice for the computational methodology developed in this work.

One draw-back of the choice of Composite Parametric Cubic Spline Curve representation of the geometries of interior boundaries is the global dependence (which is also called the **global propagation of change** [254]) of the shape of every boundary on the locations of its individual control points. In particular, the location (or motion) of any control point in an interior boundary may influence the shape (and hence the location or motion) of spline segments far away in the sequence, and possibly significantly. However, this effect may be eliminated by the introduction of auxiliary control points at which the continuity is specified to be C^0: a continuity level of C^0 at any control point effectively disconnects the slopes of the two splines emanating from the control point and prevents any global propagation of change across it. In this work, this draw-back is largely inconsequential since whenever there is boundary motion, all control points are chosen to have a C^0 continuity level. As explained in Chapter III, this is done because it is highly desirable to discretize moving boundaries using piecewise linear segments, in order to allow full satisfaction of the Geometric Conservation Laws.

An important property of the representation implied in Equations 5.2 and 5.3 is that the shape of any individual spline curve is invariant to translations and rotations. This is evident from the parameterization of the individual spline curves in terms of the displacements along the chords of these spline curves. An immediate consequence of this invariance is that the boundaries of rigid objects defined in terms of a composition of such spline curves can be arbitrarily rotated and translated without any change in their shape. More fundamentally, this invariance is a necessary requirement for ensuring that the geometric representation does not violate the fundamental physical requirement of invariance of the geometry and the computational solution of the system of equations being solved to translations and rotations of the coordinate axes.

General reviews of the theory of interpolation by cubic splines, as well as discussions of various practical aspects of cubic spline interpolation, including the construction and use of Composite Parametric Cubic Splines, may be found in [347] or in [254].

The Composite Parametric Cubic Spline Curve representation of each interior boundary is stored in this work in a link-list data-structure [199], as illustrated in Figure 5.1. Each node in the link-list stores the internal representation of an individual spline segment of the composite system for that boundary, including all the required identification data and attribute data of the spline segment and its control points, as described in more detail in the next section. The link-list data-structure naturally preserves the sequential ordering in the composite system of the splines and their associated control points, and readily allows the insertion of new spline segments and the removal of existing ones.

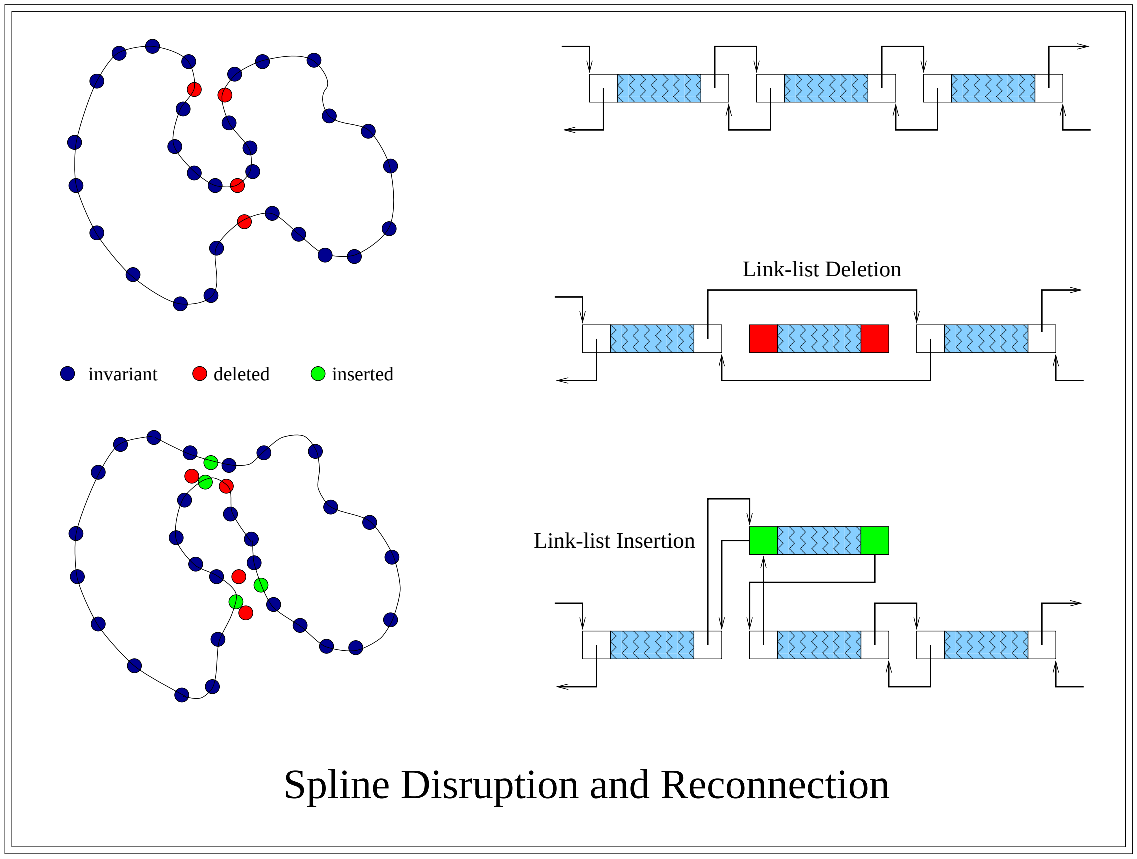

Figure 5.1: The link-list storage of the representation of an interior boundary in terms of a sequence of Parametric Cubic Spline segments. The figure also depicts the treatment of topologic transformations, such as merging and break-up of boundaries, by disruption and reconnection of links between appropriate nodes in the link-list.

New splines are inserted into the link-list representation by allocating and appropriately initializing a new node, and then re-connecting the links between the new node and the assigned predecessor and successor nodes in the link-list, as illustrated in Figure 5.1. Existing splines are removed from the link-list representation by appropriately re-connecting the links of the predecessor and successor nodes in the link-list, as illustrated in Figure 5.1, and then deleting the node. Insertion of new spline segments is necessary, for example, when boundaries elongate and become more curved, while removal of existing spline segments is desirable, for example, when boundaries with low curvature undergo large increases in length.

The link-list data-structure and the ability to easily remove and add spline segments also allows the implementation of topologic transformations in interior boundaries by allowing arbitrary disconnections and reconnections between control points, as depicted, for example, in Figure 5.1. As shown in Figure 5.1, the reconnection of splines from different parts of the same interior boundary or from different interior boundaries represents a boundary-merging operation, while the disconnection of splines within an interior boundary represents a boundary-disintegration operation.

The link-list of the cubic spline segments composing each interior boundary is considered and treated as the complete representation of the boundary, and is only accessible through another link-list, the link-list of interior boundaries. Each node in the latter link-list specifies the key parameters of the corresponding interior boundary, such as the number of spline segments in it, and the extremal coordinates of the bounding box enclosing the boundary. The link-list of interior boundaries may be dynamically incremented or decremented to allow the number of interior boundaries, $b \geq 0$, in a computation to be arbitrarily specified at the start of the computation, and to allow the number of interior boundaries to be arbitrarily changed during the computation.

This section was exclusively devoted to the representation and treatment of interior boundaries, with no discussion of the representation or treatment of exterior boundaries. As explained in more detail in Section 5.1, the latter boundaries do not require a special or distinct representation or treatment in this work.

5.3 Intersection of the Interior Boundaries with the Cartesian Grid

The geometric representation of each boundary in the interior of the Computational Region (that is, each interior boundary) and the process used to construct that representation are both independent of the grid, and are both solely determined by the coordinate values and the continuity-level-index values of the control points of the boundary, as described in the preceding section. Once the geometric representations of the interior boundaries have been constructed, the representations are intersected with the Cartesian grid (that is, with the grid-lines) to determine, for each Cartesian cell in the grid, whether the cell is cut by an interior boundary, whether the cell lies wholly within an interior boundary, or whether the cell is exterior to all the interior boundaries. For cells that are cut by one or more interior boundaries, the geometries of the two or more "sub-cells" (which in general are arbitrary polygons) that result from this "cutting" are determined and (if desired) stored.

To determine whether a Cartesian cell lies wholly within an interior boundary, whether is it cut by one or more interior boundaries, or whether it lies wholly outside all the interior boundaries present in a computation, a standard "point-inclusion", "point-location", or "point-set-membership" algorithm is used [282]. The algorithm is based on extending a "semi-infinite" ray (which in this work is always chosen to be aligned with one of the Cartesian axes, to reduce the computational cost) from a vertex of the Cartesian cell, intersecting that ray with all the interior boundaries present in a computation, and counting the number of intersections with each of those interior boundaries. If the number of intersections of a ray with a given interior boundary is even, then the vertex of the Cartesian cell is outside that boundary; if that number is odd, then the vertex is inside that boundary. This algorithm is applied to each of the four vertices of the Cartesian cell. If all the vertices of the Cartesian cell are inside a given interior boundary, and if none of the four edges of the cell are (multiply) intersected by that boundary, then the Cartesian cell is taken to be inside that boundary. Similarly, if all the vertices of the Cartesian cell are outside a given interior boundary, and if none of the four edges of the cell are (multiply) intersected by that boundary, then the Cartesian cell is taken to be outside that boundary. If at least one of the vertices of the Cartesian cell is outside the boundary, *and* at least one of the vertices of the Cartesian cell is inside the boundary, then the cell is taken to be "cut" by that boundary, regardless of whether any of the edges of the Cartesian cell is multiply intersected by that boundary.

If any of the edges of a Cartesian cell is multiply-intersected by a single interior boundary, then, depending on the specific intersection pattern (of the whole cell with the whole interior boundary), the multiple intersection is eliminated either by refining the Cartesian cell, or by locally truncating the intersected geometry of the boundary (within the Cartesian cell, but not in the geometric representation of the boundary). The algorithm used to evaluate the intersection pattern to determine how to eliminate multiple intersections is quite elaborate. Once all the multiple intersections with a single boundary that were originally present have been eliminated from all edges of the Cartesian cell, the cell (or all of its four sub-cells that were obtained by refinement) will satisfy exactly one of the three classification conditions defined in the preceding paragraph.

As mentioned above, if a Cartesian cell is cut by an interior boundary, then the geometries of the resulting "sub-cell" fragments are determined and (if desired) stored. For gasdynamic computations, the most relevant geometric data related to cell-cutting are the lengths and centroids of the faces (or edges) of the resulting sub-cells and the centroids of these sub-cells, since these data are the ones used in reconstructing and limiting the slopes of the gasdynamic variables and in computing the fluxes between neighboring cells.

Chapter VI describes the "point-location" and "cell-cutting" procedures outlined above from the view-point of grid-generation and geometric-adaptation, and explains in detail the interdependence between these four processes.

As implied above, intersecting the geometric representations of all the interior boundaries with the Cartesian grid effectively requires evaluating the intersection of every individual cubic spline segment in every interior boundary with every Cartesian cell in the grid. The computational expense of such a process is in principle very high, since it scales with the product nm, where n is the number of Cartesian cells in the grid, and m is the total number of individual cubic spline segments in all the interior boundaries. In practice, several techniques can be individually or jointly used to lower this cost, often by orders of magnitude. The most important such technique used in this work relies on evaluating the overlap of the bounding box of a whole interior boundary with suitably-sized quadrants or sub-quadrants of the Cartesian Square. If the bounding box and a quadrant or sub-quadrant have no overlap, then all the Cartesian cells in the quadrant or sub-quadrant can be excluded from further intersection evaluations, and can be classified as being outside that interior boundary. Similarly, any Cartesian cell or straight-line segment that does not overlap the bounding box of an individual cubic spline segment need not have its intersection with that spline segment evaluated any further. Other such techniques that are used in this work are described in Chapter VI.

As implied above in this section and in the preceding section, the evaluation of all intersections between the representations of interior boundaries and the Cartesian cells of the grid reduce to one specific type: the evaluation of the intersection of a cubic spline segment with an axis-aligned, straight-line segment (which could either be a semi-infinite ray, or the finite edge of a Cartesian cell), and can therefore be simplified and optimized to a large degree. This is one of the benefits of using a Cartesian grid (which always has axis-aligned cell faces or edges).

As mentioned above, the intersection between a straight-line segment and a cubic spline segment is evaluated in this work by first evaluating the overlap of the straight-line segment with the bounding box of the cubic spline segment. This evaluation has a relatively very low operation count. If the straight-line segment and the bounding box are found to have no overlap, then the straight-line and cubic spline segments are taken to be non-intersecting. If the straight-line segment and the bounding box overlap, then the appropriate spline equation must be solved. The specific equation to be solved depends on whether the edge or straight-line segment is parallel to the y-axis or parallel to the x-axis. In the former case, the edge or line is called an x-edge or an x-line, while in the latter case, the edge or line is called a y-edge or a y-line. The equations to be solved with an x-line and a y-line are, respectively, as follows:

$$A_x u^3 + B_x u^2 + C_x u + D_x - X = 0 \tag{5.5}$$

and

$$A_y u^3 + B_y u^2 + C_y u + D_y - Y = 0 \tag{5.6}$$

where X is the intercept of the x-line, and Y is the intercept of the y-line, and where all other symbols are as defined in the preceding section.

Once Equation 5.5 or Equation 5.6 has been solved for all real roots, u_s, satisfying

$0 \leq u_s \leq 1$, the coordinates of the corresponding intersection point(s) along the straight-line segment can be determined by substitution of the root(s) u_s into the corresponding spline equation. The solution to either of Equations 5.5 or 5.6 in this work is obtained using a Newton-Raphson solver that is fortified with a bisection technique to ensure robustness. Any possible solutions u_s in the complex plane are discarded a-priori, and any solutions u_s outside the closed interval $[0, 1]$ in the real line are ignored for the purpose of identifying or counting intersections between the two given segments.

Since the Newton-Raphson solution technique requires the derivatives of the spline functions, the coefficients and the determinant of the derivative of the spline equation for each individual cubic spline segment are pre-computed and stored before the intersection computations are performed. Because of the analogous form of the Equations 5.5 and 5.6 for the X and Y intercepts, a single algorithm and software implementation (with an appropriate switching toggle) is used for both cases.

Several precautions and checks are needed to ensure that the intersection-calculation procedure outlined above works reliably and robustly in practice, especially with finite arithmetic calculations. These precautions and checks enable the handling of degeneracies such as tangencies, end-point intersections, multiple or coincident intersections, and apparent topological inconsistencies. The most important of these precautions and checks ensures that no intersection occurs between a spline curve and the vertex of a cell edge (or equivalently, the vertex of a cell). If an intersection would occur in a sufficiently close vicinity to a cell vertex, then the positions of the closest control points in the corresponding Composite Parametric Cubic Spline Curve are repeatedly perturbed by a small displacement in a random direction until the condition is eliminated. The same perturbation procedure is used when tangencies are detected. This approach is similar to the alternative approach of handling typical degeneracies by displacing the control points so that the cubic splines intersect cell edges at only a finite number of pre-determined locations along those edges.

In order to decrease the computational effort for the evaluation of the intersections of splines with cell edges (or cell faces) and to facilitate the process, and for fast proximity evaluations, each node in the link-list of splines that is described in the preceding section stores the representation of the corresponding spline using 34 real numbers. These 34 real numbers fully define the geometry of the spline segment and describe all the primitive, and several of the derived properties of the segment. The first 22 of these real numbers are as follows: 1: x_0; 2: y_0; 3: x_1; 4: y_1; 5: $x_0^{'}$; 6: $y_0^{'}$; 7: $x_1^{'}$; 8: $y_1^{'}$; 9: t_0; 10: t_1; 11: A_x; 12: A_y; 13: B_x; 14: B_y; 15: C_x; 16: C_y; 17: D_x; 18: D_y; 19: $Det(x^{'})/4$; 20: $Det(y^{'})/4$; 21: $\sqrt{Det(x^{'})/4}$ (assuming $Det(x^{'}) >= 0$); and, 22: $\sqrt{Det(y^{'})/4}$ (assuming $Det(y^{'}) >= 0$), where the subscripts 0 and 1 denote the two end-points of the spline, x and y denote the $x-$ and $y-$coordinates, t denotes the cumulative chord length along the CPCSC, the superscript $'$ denotes the spatial derivative with respect to t (which is identical to the spatial derivative with respect to u), and where A, B, C, and D denote the polynomial coefficients of the cubic spline as indicated in Equations 5.2 and 5.3. The factor of four appears in the last four quantities because it eliminates much multiplication in the "boxing" and "intersection computing" operations. The elements $23-26$ of the 34 real numbers used to define the geometry of each spline segment store the coordinates of the corners of the bounding rectangle for the spline segment, as follows: 23: x_{min}; 24: y_{min}; 25: x_{max}; and, 26: y_{max}, where these symbols denote the obvious values associated with a "bounding rectangle". The elements $27-30$ of the 34 real numbers used to define the geometry of each spline segment store commonly-recurring multiples of A_x, A_y, B_x, and B_y for faster computation of, say, the derivative function, as follows: 27: $3A_x$; 28: $3A_y$; 29: $2B_x$; and, 30: $2B_y$. The elements $31 - 34$ of the 34 real numbers used to define the geometry of each spline segment store the coordinates of the corners of the "proximity-space" rectangle for the spline segment, as follows: 31: $xMin$; 32: $yMin$; 33: $xMax$; and, 34: $yMax$, where these symbols denote the obvious values associated with a "proximity rectangle". The proximity rectangle in this work is always obtained by expanding the bounding-box rectangle of the spline segment by an appropriate value based on the size of the smallest Cartesian cell in the grid.

This section mostly described the algorithms and processes used to determine the point-set membership of the Cartesian cells of the Quadtree grid, and to evaluate the intersections of the cubic spline segments of the interior boundaries with the Cartesian cells of the grid, and to determine the geometries of cut cells. The manner in which the intersection computations are used during the grid generation process to create the "geometry-adapted" grid, and the detailed manner in which "cut" computational cells are defined from the intersection computations are both described in Chapter VI generally, and in Section 6.3 specifically.

5.4 Modeling of the Motion and Deformation of Interior Boundaries

As mentioned in Chapter I and as explained in Section 5.1, in this work, every boundary in the interior of the Computational Region (that is, every interior boundary) may deform or move anywhere within the Computational Region. In addition, every interior boundary may undergo topological transformations of the type associated with the break-up or the disintegration of an interior boundary, or associated with the merging or the coalescence of one interior boundary with another or with itself. As also explained in Section 5.1, the exterior boundaries in this work remain topologically invariant and stationary for all time.

The remainder of this section is devoted mostly to describing and discussing the specific formulations and numerical algorithms used in this work to model and compute the trajectories of moving or deforming interior boundaries and their control points, and to describing and discussing how such motions and deformations interact with the flow-solution, cell-merging, and grid-adaptation algorithms.

5.4.1 Overview of the Modeling of Boundary Motion and Deformation

The motion or deformation of any interior boundary in this work must be specified or formulated in terms of only one of the three available generic motion models: (i) a **prescribed-displacement motion model**; (ii) a **prescribed-velocity motion model**; and, (iii) a **fluid-coupled motion model**. The prescribed-displacement motion model and the prescribed-velocity motion model may individually or collectively be called the **prescribed-motion model**. Every moving interior boundary must also be identified as either a rigid boundary, or a deformable boundary.

With either of the two prescribed-motion models, each control point in a boundary follows a pre-determined trajectory in time, regardless of the flowfield solution (or the flowfield states surrounding the boundary). If the boundary is rigid, then the individual trajectories of all the control points in the boundary must be implicitly constrained in a manner that prevents the introduction of any distortion in the geometry of the boundary.

With the fluid-coupled motion model, the trajectory of each control point in a boundary depends on whether the boundary is rigid or deformable. If the boundary is rigid, then the trajectory of the boundary as a whole is determined through integration of the resultant force and the resultant moment applied to the entire boundary, and the trajectories of the individual control points of the boundary are determined from the trajectory of the whole boundary. If the boundary is deformable, then the trajectory of each control point in the boundary is independently determined through integration of the resultant force applied at or assigned to that control point.

With all three motion model, for deformable boundaries, the relative motion of the control points must either explicitly or implicitly be constrained to prevent violation of the relevant fundamental geometric requirements (which are described in Section 5.2) that must be met by all interior boundaries at beginning and end of each time-step. For example, the control points of a deformable boundary that has not explicitly been allowed to undergo topological transformations may not move in a manner that would cause the boundary to intersect with itself.

With both of the prescribed-motion models, the coupling between the motion or deformation of an interior boundary on the one hand, and the fluid-dynamic solution on the other hand is of the “one-way” type: the motion or deformation of the boundary affects the fluid-dynamic solution through the corresponding boundary conditions that are imposed on the fluid-dynamic solution (which reflect the geometry and, either implicitly or explicitly, the velocity and acceleration of the boundary), but the fluid-dynamic solution has no effect on the motion or deformation of the boundary. In contrast, with the fluid-coupled motion model, the coupling between the motion or deformation of an interior boundary and the fluid-dynamic solution is of the “two-way” type: the motion or deformation of the boundary affects the fluid-dynamic solution through the corresponding boundary conditions that are imposed on the fluid-dynamic solution (which reflect the geometry and, either implicitly or explicitly, the velocity and acceleration of the boundary), and the fluid-dynamic solution affects the motion or deformation of the boundary through the fluid-dynamic forces that the fluid applies on the boundary.

The motion model and the “rigidity type” for each moving interior boundary in a computation can be specified independently of the motion models and the rigidity types for all the other moving interior boundaries in the computation. Thus, different interior boundaries in the same computation may have arbitrarily different motion models and arbitrarily different rigidity types.

5.4.2 Formulation and Implementation Details of the Motion Modeling

In the **prescribed-displacement motion model**, the motion of each control point, p, in an interior boundary is specified by directly specifying $\vec{d_p}(t)$, its displacement vector as a function of time, in the form

$$\vec{d_p}(t) = \vec{f_p}(t) \tag{5.7}$$

where t is the total integration time since the start of the computation, and $\vec{f_p}(t) = (fx_p(t), fy_p(t))$ is a generic vector function of t satisfying $\vec{f_p}(0) = \vec{0}$. More precisely, $\vec{d_p}(t) = (dx_p(t), dy_p(t))$ is the displacement of point p, with $dx_p(t)$ and $dy_p(t)$ specifying, respectively, the components of the displacement of point p along the x and y axes in terms of t, relative to the position of p at $t = 0$.

If the functions $\vec{d_p}(t)$ are identical for all p in a boundary, then rigid translation will result; otherwise, the boundary will either deform in time or will undergo some rotational motion. Since any displacement of a rigid body (or boundary) in three-dimensional space may be fully specified by specifying the translational displacement of a given point in the body and specifying a rotational displacement about some axis passing through that point, the functions $\vec{d_p}(t)$ for general displacement of a rigid body (or boundary) may be expressed in the form

$$\vec{d}_p(t) = \vec{d}_c(t) + \vec{\vec{\theta}}(t)\vec{r}_p \tag{5.8}$$

where $\vec{d}_c(t)$ is the displacement of the chosen reference point c in the body as a function of time, $\vec{\vec{\theta}}(t)$ is the rotation (or angular displacement) matrix of the rigid body (or boundary) about the reference point c as a function of time, and $\vec{r}_p = \vec{d}_p(0) - \vec{d}_c(0)$ denotes the position vector of the point p relative to the reference point c at time $t = 0$. In two-dimensional spaces, the axis of rotation is always perpendicular to the plane in which the translation occurs.

In the **prescribed-velocity motion model**, the motion of each control point, p, in an interior boundary is specified by directly specifying $\vec{v}_p(t)$, its velocity vector as a function of time, in the form

$$\vec{v}_p(t) = \vec{g}_p(t) \tag{5.9}$$

where $\vec{g}_p(t) = (gx_p(t), gy_p(t))$ is a generic vector function of t providing the instantaneous velocity of point p in terms of elapsed time since the start of the computation, and all other symbols and subscripts denote the meanings assigned to them in relation to Equation 5.7.

The displacement of control point p is then computed from the relation

$$\vec{d}_p(t) = \int_0^t \vec{v}_p(t) \tag{5.10}$$

where $\vec{d}_p(t)$ is as defined above.

In order to be able to handle any function $\vec{v}_p(t)$ (or $\vec{g}_p(t)$) in a generic manner in this work, the integration in Equation 5.10 is discretely evaluated in only a second-order-accurate form, which is given by

$$\Delta\vec{d}_p(t) = \frac{\Delta t}{2}(\vec{v}_p(t) + \vec{v}_p(t + \Delta t)) \tag{5.11}$$

where $\vec{v}_p(t)$ is as defined above, Δt is the size of the integration time-step (or the current increment in the total integration time, t, since the start of the computation), and $\Delta\vec{d}_p(t)$ is the current increment in displacement over the current time increment Δt. The total displacement, and hence the present location of control point p is obtained by accumulating the all the individual displacement increments $\Delta\vec{d}_p$ over the individual time steps since the start of the computation.

If the functions $\vec{v}_p(t)$ are identical for all p in a boundary, then rigid translation will result; otherwise, the boundary will either deform in time or will undergo some rotational motion. Again, since any motion of a rigid body (or boundary) in three-dimensional space may be fully specified by specifying the translational velocity of a given point in the body and specifying a rotational velocity about some axis passing through that point, the functions $\vec{v}_p$ for general motion of a rigid body (or boundary) may be expressed in the form

$$\vec{v}_p(t) = \vec{v}_c(t) + \vec{\omega}(t) \times \vec{r}_p(t) \tag{5.12}$$

where $\vec{v}_c(t)$ is the velocity of the chosen reference point c in the body as a function of time, $\vec{\omega}(t)$ is the angular velocity of the rigid body about the reference point c (and also about every other point in the body) as a function of time, and $\vec{r}_p(t) = \vec{d}_p(t) - \vec{d}_c(t)$ denotes the instantaneous position vector of the point p relative to the reference point c at time t. Again, in two-dimensional spaces, the axis of rotation is always perpendicular to the plane in which the translation occurs.

In general, the exact final location or shape of a boundary in the prescribed-velocity formulation will depend partly on the time-step sizes taken in the discrete integration of Equation 5.11, and the exact displacements for the given velocity functions $\vec{v}_p(t)$ will only be achieved in the limit as $\Delta t \to 0$. In addition, for any rigid boundary, discrete integration of the general “split” motion formulation given in Equation 5.12 by use of a discrete integration scheme analogous to that implied in Equation 5.11 will introduce a distortion in the shape of the boundary, since any discretization error in approximating the instantaneous value of $\vec{r}_p(t)$ (in Equation 5.12) may introduce a distortion in the geometry of the boundary. Therefore, for a rigid boundary undergoing rotation, the translational and rotational velocities should be integrated separately to yield the translational and angular displacements separately. These two displacements can then be used to update the position of every control point, as described above for the prescribed-displacement case, without shape distortion. Unfortunately, such a procedure cannot be applied for a deforming body undergoing rotation, and the shape-distorting effect of the discretization error in the time-integration of the velocity functions cannot be avoided in that case.

In the **fluid-coupled motion model**, the accelerations of an interior boundary or its individual control points are determined from the applicable Equations of Motion, under the action of the total forces and, if applicable, the total moments, that are applied to the whole boundary or its individual control points. The velocities and displacements of the whole boundary or its control points are then determined by successive time-integrations of the relevant accelerations. The fluid-dynamic portion of the total forces and moments applied to the boundary or its individual control points are thus the means through which the motion of the boundary and the fluid-dynamic solution are coupled to each other.

For each control point, p, in an interior boundary with fluid-coupled motion, the total applied force as a function of time, $\vec{F}_p(t)$, is partitioned in accordance with the convenient formulation

$$\vec{F}_p(t) = \vec{P}_p(t) + \vec{E}_p(t) \tag{5.13}$$

where $\vec{P}_p(t)$ is the contribution to the total applied force of the fluid-dynamic forces (which for inviscid flows are confined to the pressure forces), and $\vec{E}_p(t)$ is the contribution from all non-fluid-dynamic forces, including external forces and forces due to the elasticity or the stiffness of the boundary or the corresponding body.

The inviscid fluid-dynamic force assigned to each control point p, $\vec{P}_p(t)$, is computed in this work by integrating the surface pressure along the arcs of the two splines emanating from the control point p, up to the point along each arc that corresponds to the mid-point of the chord of the corresponding spline, that is,

$$\vec{P}_p(t) = -\int_{s_-}^{s_+} \tilde{p}(t)\vec{s} \tag{5.14}$$

where the integrand $\tilde{p}(t)$ is the instantaneous surface pressure along the arc, $\vec{s}$ is the outward-pointing arc-length vector along the arc curve, $s_- = s(\frac{t_p+t_{p-1}}{2})$ (denoting the point corresponding to the mid-chord position for the spline connecting control point p with its predecessor control point $p-1$), $s_+ = s(\frac{t_p+t_{p+1}}{2})$ (denoting the point corresponding to the mid-chord position for the spline connecting control point p with its successor control point $p+1$), and t_p, t_{p-1}, and t_{p+1} refer respectively to the cumulative chord length at the control points p, $p-1$, and $p+1$.

Although partly arbitrary, the choice just described for the assignment of pressure forces is first-order accurate in space, and therefore exact in the limit of arc-length refinement. The integration of the pressure force along each of the half-arcs as just described is computed by summing the pressure forces applied to each of the sub-segments of the half-arc that cut the individual computational cells that are intersected by the half-arc, and not by summing or averaging the pressure forces along some selected set of points along the half-arc. The specific formula used for this discrete spatial integration is given by

$$\int_{s_-}^{s_+} \tilde{p}(t)\vec{s} = \sum_{f=1}^{nF} \tilde{p}_f(t) A_f \vec{n}_f \tag{5.15}$$

where f denotes the generic face in the list of "cutting" cell faces defined from the intersection of the half-arcs with the Cartesian cells, nF is the total number of faces in the discretized geometry of the half-arcs (that is, in the list of "cutting" cell faces defined from the intersection of the half-arcs with the Cartesian cells), $\tilde{p}_f(t)$ is the extrapolated instantaneous pressure at the centroid of face f, A_f is the n-area of face f (which here reduces to the length of face f), and $\vec{n}_f$ is the outward-pointing normal of face f. Using the same "cut-cell" geometries that are used to apply the boundary conditions in the flow-solver ensures that the interfacial force is "conservative", that is, that the individual discrete forces applied by the fluid on the interior boundary are equal in magnitude and opposite in direction to the individual discrete forces applied by the interior boundary on the fluid, and hence that the total forces and moments that the fluid and the interior boundary apply on each other are equal in magnitude and opposite in direction.

As implied above, the non-fluid-dynamic forces applied to each control point p, $\vec{E}_p(t)$, may include arbitrary unmodeled forces that are applied "externally" to the system, or may include modeled forces that are internal to the body within the interior boundary (to which control point p belongs). Forces internal to bodies may depend on the distance between the control point p and one or more neighboring control points in the same interior boundary, on the distance to control points on an opposite side of the same interior boundary, on the local curvature of the interior boundary, or on any of a variety of variables, including the absolute velocity or acceleration of the control point, or the velocity or acceleration of the control point relative to other control points in the same boundary. New models for external forces and other non-fluid-dynamic forces may readily be introduced in the current framework of the algorithm.

Once the sum of all the forces $\vec{F}_p(t) = \vec{P}_p(t) + \vec{E}_p(t)$ applied to each control point p of an interior boundary is determined, the procedure followed to predict the dynamic response of the boundary depends on whether the boundary is identified as rigid or deformable, as described below.

For a rigid boundary, the total forces $\vec{F}_p(t)$ that are applied to each of the control points p in the boundary are summed to give the resultant force on the whole boundary. In addition, the moments of the total forces that are applied to each of the control points in the boundary are summed to give the resultant moment on the whole boundary. The trajectory of the boundary (and that of its corresponding body) is then predicted by performing a Forward-Euler integration of the applicable Equations of Motion; namely, (i) the Equation of Conservation of Linear Momentum; and, (ii) the Equation of Conservation of Angular Momentum, which are, respectively, given by

$$\vec{F}(t) = m\frac{d\vec{v}}{dt} \tag{5.16}$$

and

$$\vec{M}(t) = I\frac{d\vec{\omega}}{dt} \tag{5.17}$$

where $\vec{F}(t)$ is the instantaneous resultant force applied to the boundary or body, m is the total mass associated with the boundary or body, $\vec{v}$ is the velocity of the center of mass of the boundary or body, $\vec{M}(t)$ is the instantaneous resultant moment applied to the boundary or body about some point, c_m, in the plane of the Computational Region, I is the moment of inertia of the boundary or body about the point c_m, $\vec{\omega}$ is the angular velocity vector of the boundary or body (about the point c_m, and also about every other point in the plane), and t is the time. For uniformity, convenience, and greater accuracy in computing or determining the moment of inertia, the point c_m is usually chosen to be the center of mass of the boundary or body being considered.

The resultant moment $\vec{M}(t)$ is computed from the individual forces applied to the boundary or body using the discrete formula

$$\vec{M}(t) = \sum_{n=1}^{nf} \vec{r}_n \times \vec{f}_n(t)$$

where $\vec{M}(t)$ is as defined above, nf is the total number of individual forces applied to the boundary or body, $\vec{f}_n(t)$ is the instantaneous force vector associated with the n^{th} individual force, and $\vec{r}_n$ is the displacement vector of the point of application of the force $\vec{f}_n(t)$ relative to the point c_m about which the resultant moment, $\vec{M}(t)$, is being computed.

The time-integrations of the two Equations of Motion 5.16 and 5.17 are performed respectively using the following Forward-Euler (that is, explicit, first-order-accurate) discrete forms:

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \frac{\Delta t}{m}\vec{F}(t) \tag{5.18}$$

and

$$\vec{\omega}(t + \Delta t) = \vec{\omega}(t) + \frac{\Delta t}{I}\vec{M}(t) \tag{5.19}$$

where t is the time at the start of the current time-step, Δt is the time-interval of the current time-step, which in this work is always identical to the time-interval of the flow-solver, and all other symbols are as defined above.

Once the updated value of the translational velocity of the center of mass (of the boundary or body), $\vec{v}(t + \Delta t)$, and the updated value of the rotational velocity about the center of mass (of the boundary or body), $\vec{\omega}(t + \Delta t)$, have been computed through Equations 5.18 and 5.19 respectively, the incremental translational displacement of the center of mass (of the boundary or body) and the incremental rotational displacement about the center of mass (of the boundary or body) can readily be obtained. This is done by a second time-integration, using the same procedure described above for the treatment of prescribed-velocity motion, that is, using Equation 5.11 for the translational displacement, and an analogous equation for the rotational displacement. The incremental translational displacement of the center of mass is then used to update the location of all the control points in the boundary, and the incremental angular displacement about the center of mass is then used to relocate all the control points about the updated center of the mass of the body, again as described above for the prescribed-velocity motion treatment. The translational and rotational velocities are thus integrated separately in order to provide the translational and angular displacements separately, as described above for the prescribed-velocity motion model.

For a deformable boundary with the fluid-coupled motion model, each control point in the boundary is assigned a pre-specified mass and is allowed to have an independent motion as a dimensionless particle. Any stiffness, damping, or other mechanical property of a boundary or its corresponding body that constrains the independent motions of the individual control points is incorporated in the non-fluid-dynamic forces, $\vec{E}(t)$, that are applied to each of the control points, as explained above. The motion of each control point in a deformable boundary is computed again through a Forward-Euler integration of the single applicable Equation of Motion for the point: the Equation of Conservation of Linear Momentum, Equation 5.16, where now m denotes the mass assigned to the control point, not the mass of the whole boundary or body, where $\vec{F}(t)$ now denotes the net force applied at or assigned to the control point, not the net force on the whole boundary or body, and where $\vec{v}$

now denotes the velocity of the control point, not the velocity of the center of mass of the boundary or body. The Equation of Conservation of Angular Momentum is not solved for a deformable boundary since it is irrelevant for the motion of the individual dimensionless particles that are used in representing the motion of such a boundary or body. The velocities and displacements of the individual control points are obtained from the accelerations of the control points using the same procedures and formulations described above for the center of mass of a rigid boundary or body.

Explicit kinematic or dynamic constraints on the motion or the degrees of freedom for rigid or deformable boundaries with the fluid-coupled motion model may also readily be implemented in the present algorithm and data-structure.

As implied in the above explanation, the forces (and if applicable, also the moments) for the fluid-coupled motion treatment, whether for rigid or deformable boundaries, are all explicitly computed at the starting time of the integration step, leading to a first-order-accurate formulation for the overall dynamic behavior of a boundary in that treatment. This can be clearly seen through the forms of Equations 5.18 and 5.19. The extension to second-order accuracy is computationally expensive to accomplish in an overall explicit computational algorithm because it would require a determination of the forces at least at one other time within the integration time-step. This would require the use of an iterative procedure to couple the solution for the fluid-dynamic states with the solution for the boundary motion. Alternatively, a "fully-coupled" formulation which simultaneously computes the fluid-dynamic and boundary-motion updates may be used. While more expensive, such approaches would have the great advantage of providing a "fully-coupled" solution for the locations or shapes of boundaries at the end of a time-step, instead of the current "loosely-coupled" solution.

As implied above, regardless of the motion model chosen for a moving interior boundary, and regardless of whether the boundary is rigid or deformable, the explicit total displacement of the control points of the boundary (relative to their initial positions, which are given as input at the start of the computation) must always be computed for every time-step.

The need to explicitly determine the total displacement of the control points of every moving interior boundary at the end of every motion step (or time-step) arises because regardless of type of the interior boundary or the motion model selected for it, the treatment of the update to the boundary geometry is the same: once the updated location of every control point in a boundary has been explicitly computed at the end of a motion step, the control points are re-splined at their updated locations using the same CPCSC routines that are used to create the boundary representation from the initial geometric specification of the boundary, as described in detail in Section 5.2. Once the re-splining has been performed, the new intersection pattern of the interior boundary with the grid is then computed from the updated CPCSC representation of the boundary, as described in detail in Section 5.3 and in Chapter VI.

In the actual software implementation in this work, only the initial position (that is, the position at $t = 0$) and the cumulative displacement of each control point (relative to its initial position) are stored, and the updated position at any time is determined by adding the current cumulative displacement to the initial position. This arrangement largely eliminates the build-up of round-off errors in the geometric representation and allows the exact preservation of the initial geometries and shapes of rigid boundaries. This exact preservation is useful for operations like restarting a calculation without the introduction of any arithmetic-truncation errors.

It should be emphasized that no topologic changes in any boundaries are allowed between the beginning and end of any time-step. Any change in the number of interior boundaries in a computation, or any change involving the insertion, removal, or re-connection of control points along an interior boundary is applied "instantaneously" between the end of a time-step and the beginning of the next one, and therefore is purely internal to the boundary representation. In other words, topologic transformations are not modeled as physical processes occurring during finite time intervals, but rather as non-physical, instantaneous changes, confined to the representations of the boundaries involved.

5.4.3 The Interaction Between Boundary Motion and Grid Generation, Cell Merging, and Flow Solution

The motion or deformation of interior boundaries may intrinsically be specified completely independently of the grid or the grid generation process, and any convenient method for specifying this motion or deformation may be used in principle, including methods other than those implemented in this work (and described above). There are, however, extrinsic constraints in this work on the magnitude of the local displacement of a boundary during a single time-step. The first of these constraints is imposed by the cell-merging process, which is described in detail in Chapter VII. This process requires the local displacement during a single time-step of every point in every moving boundary to not exceed the corresponding local cell dimension. As explained in more detail in Section 3.3, this limit is analogous to the CFL-Number limit for an explicit scheme. The second of these constraints is imposed by the geometric-adaptation process, which is described in detail in Chapter VI. This process requires a similar constraint to that of cell merging if all the geometric features of a moving boundary are to remain adequately resolved as the boundary arbitrarily deforms or moves across the Computational Region. The third of these constraints is related to the accuracy of the computational solution: given that an explicit scheme is used to update the gasdynamic states in this work, it would not be appropriate to allow a boundary to move across more than a single whole computational cell anywhere along the boundary during a single time-step, even if this were otherwise possible.

The influence of the motion or deformation of interior boundaries on the grid generation process, and on the resulting grids at different time steps is communicated solely by the successive instantaneous positions or locations assumed by the interior boundaries at the beginnings and ends of individual time steps, and by the intersections of the boundaries at these different instantaneous locations with the Cartesian grid lines. These instantaneous locations affect the grid generation process through the mechanism of geometric-adaptation, in which the grid is locally refined or coarsened, between the end of one time-step and the beginning of the next time-step, to fully resolve the geometric features of all interior boundaries at their corresponding current positions. By repeatedly resolving the geometries of moving or deforming interior boundaries every time the shapes or locations of these boundaries are updated (that is, after every motion step), the geometric-adaptation process ensures that such boundaries remain resolved throughout a computation, even as these boundaries move or deform, as explained in more detail in Chapter VI.

As far as the grid-generation algorithm and the gasdynamic-update algorithm are concerned, local changes in the displacement of a moving or deforming interior boundary across successive time-steps, and hence approximations to the local velocities and accelerations of a boundary during any time-step *must* all be determined by computing differences in the positions of local boundary cuts at the beginnings and ends of successive time-steps, and not directly from the nominal local velocities or accelerations of the boundary. Otherwise, the Geometric Conservation Laws may be violated, as explained in Section 3.4.

As implied in Section 3.8, the gasdynamic update for a time-step is always performed after the boundary-motion update has been determined for that time-step. More specifically, the gasdynamic update for a time-step is performed only after the initial and final geometries of all boundaries for that time-step have been determined, and only after the initial and final intersection patterns between all boundaries and all Cartesian cells for that time-step have been determined. This ordering of the geometric and the gasdynamic update operations is necessary because the explicit scheme adopted in this work for the gasdynamic update procedure requires the full geometry of every computational cell (including all cut cells) to be known at both the beginning and the end of each time-step, as explained further in Chapter III and in the preceding paragraph.

The presentation given above in this section of the modeling of boundary motion and deformation makes it clear that the initial and final geometries across any time-step of any moving interior boundaries, and the corresponding initial and final intersection patterns of any such boundaries with the Cartesian cells can readily be determined at the *start* of the time-step for both the prescribed-displacement and the prescribed-velocity motion models. This is because the motion of boundaries in these models is evaluated independently, with no influence on it from the gasdynamic states or the grid, for example. That presentation also explains that the boundary motion update for the fluid-coupled motion model only requires the gasdynamic states at the start of a time-step to compute the boundary-motion update across that time-step, implying that for this motion model too, the boundary geometry and the corresponding intersection patterns at the end of the time-step can also be determined from the data at the *start* of the time-step. Thus, all three motion models used in this work are seen to be able to satisfy the ordering requirement that was explained in the preceding paragraph (for sequencing the geometric and gasdynamic update operations).

For each time-step, the initial (or "old") and the final (or "new") geometries and intersection configurations (that is, the geometry and intersection configuration at the starting time t of the time-step, and the geometry and intersection configuration at the ending time $t + \Delta t$ of the time-step) of every interior boundary remain available in memory. If the time-integration scheme of the flowfield solution algorithm requires more than two boundary positions during a time-step, then the additional boundary positions and the corresponding intersection configurations will also be automatically pre-computed and stored. The advantage of storing the required positions and intersection configurations at both the beginning and end of a time-step is that this minimizes the repetition of the expensive geometric calculations that are required, for example, in the highly-frequent computations of cell-face fluxes and cell-merging evaluations. After all the update operations for a time-step have been performed (including the gasdynamic update), the "new" boundary geometry and intersection configuration are re-assigned to the "old" data for the next time-step. Then, the updated boundary geometry and intersection configuration for the next time-step are computed and stored in the "new" data location for the next time-step. In this way, only a single update to the geometry and the intersection configuration is computed for each new time-step after the first. For stationary boundaries, the boundary-position update step and the intersection-configuration update step may both be eliminated, as implied in Section 3.8.

CHAPTER VI

Quadtree-Based Cartesian Grid Generation and Adaptation

This chapter describes the grid-generation and grid-adaptation techniques adopted and developed in this work to create Quadtree-based, Adaptive Cartesian grids. The chapter explains how the resulting grids are used for arbitrarily-moving and stationary geometries of arbitrary complexity, and explains how the grid-generation process for moving or deforming boundaries is effected through adapting the grid so that the grid simultaneously resolves the initial, final, and any intermediate locations of the boundaries across a motion step. The major individual algorithms that are used in the grid-generation and grid-adaptation processes, such as those for cell refinement and coarsening, for traversal of the Quadtree, and for determination of cell-neighbors, are described, and their fundamental properties and their computational performance are derived and discussed. The manner in which the motion and deformation of boundaries is treated in the grid-generation process is discussed, including the treatment of topologic transformations in boundaries. The properties of the Quadtree data-structure and the associated grid that are most relevant to computational solution procedures by the Finite-Volume Method for the System of Euler Equations are described and discussed, with emphasis on their advantages and disadvantages. The adaptation of the grid to the geometric features of boundaries and to changes in their position are described, and so is the manner in which the grid is adapted to the computational solution. The special requirements for solution-adaptation for time-accurate simulations and for moving boundaries are described. Brief comparisons are made wherever appropriate to alternative techniques of generating and adapting Cartesian grids. The considerations that figure in the most important implementation choices are also briefly discussed.

6.1 The Quadtree Data-Structure

This section introduces and discusses some fundamental terms and concepts that are used in describing the properties of the Quadtree data-structure and its role in the grid generation approach used in this work. All these terms and concepts are related to the components or the organization of Abstract Data Types. Definitions, descriptions, and discussions of data types or data-structures may be found, for example, in [208], while rigorous definitions and an extensive development of many of the concepts and terms discussed here may be found, for example, in [199]. An extensive review of the properties, applications, and development history of the many variants of the Quadtree and Octree data-structures may be found in [318] and [319].

As indicated in the references mentioned in the above paragraph, since their introduction, apparently first in [125], Quadtrees and Octrees have been used throughout Computer Science, in disciplines ranging from Image Processing, to Geographic Information Systems, to Grid Generation, most commonly for solution of the Range-Query Problem.

6.1.1 Fundamental Definitions

Definition: A **Node:** is the basic "atomic" component in an Abstract Data

Type, representing an entity which is to be stored or manipulated in the computer representation of the data type. The computer representation of a node usually retains at least some of the relevant intrinsic attributes of the entity that the node represents. As a fundamental component of Abstract Data Types, it is not necessary (and indeed, actually not possible) to define a node more concretely, in the same way that it is not possible to define completely the terms "element of" and "set" in Set Theory, for example. A node may represent, for example, an individual component in a database of mechanical components, and in that case the node may incorporate details such as weight, dimensions, part numbers or other descriptions of the component it represents. Sets of nodes may be combined and associated to create complex data types or data-structures.

Definition: A **Link:** is a explicit association between two nodes, which not need not be distinct. Two nodes are said to be linked if they are associated by a link. If a link has directionality, then it may be used as the basis for imposing a hierarchical organization on the nodes it links. In such a case, depending on the direction of the link, the two nodes associated by it may be be termed **superior** and **inferior** nodes.

The concept of a link may be extended to include associations between single nodes and sets of nodes, and even between two sets of nodes, as follows: Let DS_1 and DS_2 be two non-empty sets of linked nodes (that is, two multi-node data-structures), with $DS_1 \cap DS_2 = \phi$. If a link is established between a node $n_1 \in DS_1$ and a node $n_2 \in DS_2$, then the link may be said to link n_1 and n_2, and also to link n_1 and DS_2, and n_2 and DS_1, and also to link DS_1 and DS_2. The concept of inferiority and superiority with respect to directional links may be invoked even for links between individual nodes and sets of nodes, and even for links between sets of nodes.

Definition: A **Quadtree:** is a hierarchical (hence the sub-term tree) data-structure consisting of a finite number of $n \geq 0$ nodes such that each node is linked to either exactly 4 or exactly 0 inferior nodes, *and* either exactly one or exactly zero superior nodes. Figure 6.1 illustrates an example of a Quadtree data-structure, indicating the hierarchy, the nodes (shown as spheres), and the links (shown as arrowed lines). The tree shown in Figure 6.1 is said to have three **levels**, or, equivalently, is said to have a **depth** of 3.

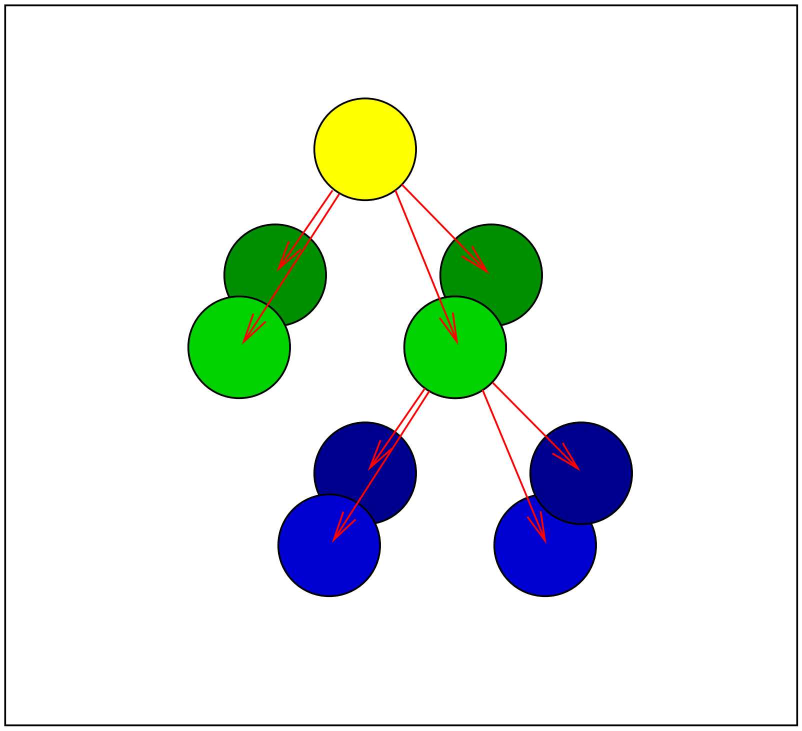

Figure 6.1: A Quadtree data-structure, showing the nodes, the links, and the hierarchical arrangement.

Although only one specific type of tree (the Quadtree) was defined in this subsection, the generalization is evident. A tree may have n branches instead of 4, and the number of inferior nodes need not be fixed. Also, there are no restrictions on differences between the nodes or their types. Useful classifications for trees and the special status of binary trees are discussed in [199].

Depending on its location within a tree, a node may be given a special name.

For example, a **root node** is the highest ranking node in the tree, a **leaf node** is a node that has no subordinate nodes, an **internal node** is any node that is not a leaf node, a **penultimate node** is the super-node of a leaf node, and a **non-penultimate node** is any node that is not a penultimate node.

6.1.2 General Properties

The definition given above for a Quadtree is relatively easy to visualize, but other equivalent definitions, in terms of graphs or lists, are also possible [199]. However, perhaps the most useful equivalent definition, using the extended concept of link, is that a Quadtree consists of a finite number $n \geq 0$ of nodes such that each node is linked to exactly 4 or exactly 0 unique (inferior) Quadtrees, and each inferior Quadtree is linked to exactly 1 or exactly 0 superior nodes. The overwhelming advantage of this second definition is that it establishes a strong correspondence between tree data-structures and recursive functions. The importance of this correspondence arises because recursion dominates the operations on, properties of, and proof techniques for trees.

The organization and hierarchy of a Quadtree also exhibit a strong correspondence to the hierarchy in the recursive spatial subdivision of a square or rectangle in $\mathbf{R^2}$. This correspondence is the basis of the use of Quadtrees in many applications in Computational Geometry, and representation of space and spatial distributions [319]. The correspondence has also given rise to the use of the term “sub-division” to indicate that a node in a Quadtree has inferior nodes. The analogy between Quadtrees and spatial subdivision of spaces in $\mathbf{R^2}$ can be extended to Euclidean spaces of arbitrary dimension, giving, for example, a binary tree in 1-D, an Octree in 3-D, and, more generally, a 2^n-ary tree in $\mathbf{R^n}$.

Starting from a Quadtree with 1 node (that is, a root node), every subdivision adds one interior node, and three leaf nodes. Therefore, after n subdivisions, the total number of leaf nodes will be $(3n+1)$, while the total number of interior nodes will be n. Thus, the ratio of the number of leaf nodes to the number of interior nodes will be $(3n+1)/n$, while the ratio of the number of leaf nodes to the total number of nodes will be $(3n+1)/(4n+1)$. The corresponding ratios for an Octree are $(7n+1)/n$ and $(7n+1)/(8n+1)$, respectively. Similarly, the construction time of the Quadtree is proportional to the total number of nodes in the tree. A tree that has n nodes will have $(n-1)$ links from superior to inferior nodes. The significance of these ratios and properties when using tree-type data-structures for representing and storing grids will be discussed below.

6.2 Fundamental Operations in the Quadtree, I

6.2.1 Traversal

For a data-structure comprised of a non-empty set of inter-connected nodes, a natural requirement exists for an operation that marches along the links in the data-structure in such a way that every node in the structure is “visited”. Visiting a node means “passing” through a node, and gaining immediate access to the node and its attribute data. Visiting a node can also mean performing an operation on that node, and such an operation may modify the data attributes, or even the connectivity of the node. The **Traversal** of a data-structure is most commonly defined as the process of visiting every node in the data-structure exactly once. However, because of the multiplicity of meanings for the term “visit”, a more precise definition, specialized for tree data-structures, is given below.

<u>**Definition:**</u> The **Traversal** of a tree is the tracing of a deterministic path along the links in the tree such that every node in the tree is encountered and every node in the tree has an operation performed on it exactly once during the tracing. The operation performed on the nodes could be the *null* operation. It should be noted that while every node must be operated on exactly once, the trajectory of the traversal path may still pass through a node more than once. The procedure followed in tracing the path through the tree and selecting the nodes for execution of the operation is called a **Traversal Algorithm**. The traversal path must start at the root node of the tree and must advance from one node to the next using only the links that connect the nodes in the tree. However, the nodes need not be operated on in the order they are encountered in the path, as described in more detail below. Although confining the start to occur at the root node of a tree is not necessary in principle, the imposition of this requirement enables the definition of traversal algorithms in convenient, recursive form, as described below.

The definition of traversal allows for the possibility that different traversal algorithms will operate on the nodes in a tree in different orders. The specification of a traversal algorithm may therefore be regarded as the equivalent of the formation of a Linear Ordering on the set of nodes in a tree. For a countable set of nodes, the specification of a traversal algorithm is also equivalent to the selection of an enumeration on the set. In more detail for a tree with a finite number of nodes, let $N = \{n_1, \ldots, n_m\}$ be the set of m nodes in the tree, where n_1, $\ldots$, n_m represent these individual nodes. Let $I = \{1, \ldots, m\}$ be the set of the first m non-zero integers. Let T be the set of all functions $t : I \mapsto N$, which are one-to-one and onto such that $N = \{t(1), \ldots, t(m)\}$. Then the specification of a traversal algorithm is equivalent to the selection of some $t \in T$.

For a tree that changes its composition or form during the traversal procedure, the above definitions and equivalences must be formulated in terms of the nodes in the final composition and form of the tree.

The traversal algorithm for a recursive structure like the Quadtree is most conveniently expressed also in recursive form, as in the following example:

1. Set the current target node of the traversal algorithm to be the root node of the current Quadtree.

2. Perform the (possibly *null*) nodal operation specified in the traversal algorithm at the current target node.

3. Determine if the current target node has any sub-trees. If it does not, terminate the traversal algorithm for the current node. Otherwise, descend to the root node of each sub-tree of the current node, in accordance with a pre-specified order for choosing the sub-trees, and re-apply the traversal algorithm to the current sub-tree (by setting the current Quadtree in Step 1 above to the current sub-tree).

The only remaining parameter to be specified in the traversal algorithm in the above example is the ordering of the sub-trees mentioned in Step 3 of the algorithm. In some situations, the order does not affect the final result, and in this work, steps are taken to ensure that for most operations it does not do so. The specific ordering of the sub-nodes chosen in this work follows the clockwise convention, starting from the SE sub-node; that is, the ordering is given by the (ordered) sequence: SE, SW, NW, and NE, looking down from “above” the tree. The definition of these “directions” is given in detail in the first sub-section of Section 6.6.

As would be expected from the definition of a traversal algorithm, the characteristic of the specific Quadtree traversal algorithm given above that distinguishes it most strongly from other Quadtree traversal algorithms is the implied order in which the nodes are targeted for execution of the nodal operation. This can be seen in more detail by considering the following variant of the traversal algorithm described above:

1. Set the current target node of the traversal algorithm to be the root node of the current Quadtree.

2. Determine if the current target node has any sub-trees. If it does, descend to the root node of each sub-tree of the current node, in accordance with a pre-specified order for choosing the sub-trees, and re-apply the traversal algorithm to the current sub-tree (by setting the current Quadtree in Step 1 above to the current sub-tree).

3. Perform the (possibly *null*) nodal operation specified in the traversal algorithm at the current target node, and terminate the traversal algorithm for the current node.

In the first example algorithm, the root node is the first node in the tree on which the nodal operation is performed; in the second example, the root node is the last node in the tree on which the nodal operation is performed. Other variants are also possible. The specific variant chosen in this work is that described in the first example. For binary trees, three different variants are in most common use; namely, the **preorder**, the **inorder**, and the **postorder** variants, where the name describes the order in which the nodal operation is executed on the root node from the three-element set consisting of (i) the root node; (ii) the left sub-tree of the root node; and, (iii) the right sub-tree of the root node, as described further in, for example, [199].

Traversal algorithms are important for trees and other multi-linked data-structures because the individual nodes in such structures are not accessible, or even identifiable, except through the links in the structure, and traversal algorithms provide one of the few systematic, efficient, and elegant means for performing global operations on the elements of the data-structure.

There are very few restrictions on the type and scope of the nodal operation that is executed during a traversal procedure. The nodal operation may depend on the position of the node in the tree, on the positions of other related nodes, or on the attribute data of the node or other related nodes. The nodal operation need not modify the node on which it operates, nor the attribute data of that node, but may instead operate on other data objects. The nodal operation can also result in the addition or removal of sub-nodes, thereby modifying the structure of the tree as the tree is being traversed. The case where sub-trees are added during the traversal procedure allows the Quadtree to be constructed into its final form during the traversal process, and as described below and in the next section, this enables the grid represented by the Quadtree to be generated with a very compact algorithm.

6.2.2 Counting, Classification, Partitioning, and Searching

At the end of the preceding sub-section, it was noted that the nodal operator executed during a traversal need not modify the node on which it operates, nor the attribute data of that node. The global operations of **counting**, **classification**, **partitioning**, and **searching** can all be carried out with nodal operators that have this non-node-modifying characteristic.

In one of the simplest examples of non-node-modifying operations, the nodal operator accumulates a global counter (which is independent of the data-structure to which the operator is applied) by the integer 1 every time it operates on a node. This operator can be used to count the number of nodes in a tree. Variants of this operator accumulate a global counter only for certain types of nodes; for example, only for leaf nodes, or only for nodes having specific values for some of their attribute data. The test for the required conditions (on node type, or attribute properties, for example) has to be invoked as part of the nodal operation. These variants can provide a count of the number of nodes in the tree that meet the conditions evaluated in the test.

The nodal operator may determine the class to which a node belongs under a certain classification system. The operator may also "mark" the nodes depending on their class. Marking a node is defined as initializing a special identification symbol within the data attributes of the node. For example, nodes representing colored objects could be classified by their color, and a symbol chosen to represent color may be stored within the attribute data of each node. Again, the testing required to determine color, for example, from the other attributes of the node, or from the position of the node in the tree or in the traversal path, must be invoked as part of the nodal operation.

If instead of marking classified nodes, new links to these nodes are stored in external data-structures, then the nodal operator would be performing a partitioning operation. The result of the partitioning is that the nodes become accessible wholly according to their class, and from data-structures that are independent of the data-structure in which the original traversal was performed. Returning to the colored-object example given above, the nodal operator could partition the nodes, for example, by creating and filling link-lists such that each link-list comprises links to all the nodes representing objects with a certain color. The link-lists could also store some or all of the attribute data of the partitioned nodes.

An important nodal operator is one that identifies (and possibly returns) either the first node encountered in the traversal path that meets a certain criterion (which could be a composite, and arbitrarily complex criterion), or all the nodes in the tree that meet the criterion. The traversal operation with such a nodal operator is called searching. Unlike operators that are required to perform global updates on all the nodes, one of the most important consideration in search algorithms is their efficiency, and the key technique for increasing efficiency in a tree data-structure is elimination of the need for exhaustively testing every node for the required condition. This is best accomplished by the selective elimination of entire sub-trees from the search based on the examination of the data in the root nodes of these sub-trees. This not only requires the development of appropriate tests for attribute data, but also usually requires the nodes to be organized in an optimal manner in the tree. Under optimal conditions, the operation count to find a node meeting a certain criterion can be made proportional to $\vartheta(\log n)$, where n is the number of nodes in the tree, whereas the operation count for an exhaustive search is proportional to $\vartheta(n)$.

Extensive use is made throughout the Quadtree-based grid generation procedure used in this work, and the flow-solution procedure used in this work, of the operations of counting, classification, partitioning, and searching, as described more specifically below.

6.2.3 Refinement and Coarsening

Two fundamental nodal operations that modify the nodes on which they operate, and also modify the composition and interconnectivity of any tree data-structure that contains those nodes, are the operations of **refinement** and **coarsening**.

<u>Definition</u>: Refinement of a Node in a Tree: This is the basic or "elemental" operation that results in the introduction of one or more new nodes in a tree, as sub-nodes of the target node (and hence as sub-trees of the target tree). Because of the way in which a tree data-structure is defined, this addition of new nodes must be accompanied by establishment of all the necessary links between the newly-added node(s) and the pre-existing target node. In a Quadtree, nodes must always be added in sets of four in order to preserve the defining structural characteristic of the Quadtree. The operation of refinement of a node in a Quadtree is implemented in this work as follows: (i) four new nodes are created (or allocated); (ii) each of these four nodes is configured as the root node of a new Quadtree data-structure; (iii) each of these four nodes is configured as a sub-node of their common super-node (by assignment of a directional link from each new sub-node to the common super-node, and assignment of a directional link from the super-node to each sub-node); and, (iv) all the necessary allocations for the attribute data of the sub-nodes are performed, and the attribute data for each sub-node is initialized to the appropriate values, possibly using the attribute data of the super-node. An immediate consequence of the action of the refinement operation is that it should only be applied to leaf nodes, and a refinement nodal operator must therefore always include a test to determine whether a node is a leaf node.

<u>Definition</u>: Coarsening of a Node in a Tree: This is the basic or "elemental" operation that results in removal or deletion of the pre-existing sub-nodes of the pre-existing target node (and hence of the pre-existing sub-trees of the target tree), and consolidation of the attribute data of the sub-nodes into that of their super-node. The coarsening operation can therefore be viewed as the inverse of the refinement operation. Because of the way in which a tree data-structure is defined, this deletion of pre-existing nodes must be accompanied by appropriate elimination and redefinition of the links between the deleted nodes and the remaining nodes. In a Quadtree, nodes must always be removed in sets of four in order to preserve the defining structural characteristic of the Quadtree. The coarsening operation is implemented in this work as follows: (i) any attribute data in the four sub-nodes nodes of the target node is consolidated into that of the target node; (ii) the attribute data objects of all the sub-nodes are discarded (that is, de-allocated); (iii) the sub-nodes are discarded (that is, de-allocated from memory); and, (iv) the links from the super-node node to the sub-nodes are reset to point to the *null* node, and the super-node is re-configured as a leaf node, and as the root node of a new Quadtree. Any other external links to the deleted sub-nodes must be reconfigured to reflect the deletion of these nodes, and any external links to the super-node must be reconfigured or updated if necessary to reflect the change of status of the node. A coarsening operation should only be applied to a penultimate node, and, correspondingly, the nodes that are discarded may only be leaf nodes. Therefore, a coarsening nodal operator must always include a test that determines whether the target node is a penultimate node.

The operations of refinement and coarsening have been described as the basic operations for addition or removal of nodes, and are as such the fundamental operators that are used to "build-up", or "prune-down" tree data-structures. As with all other nodal operators, when invoked globally within a traversal algorithm, refinement and coarsening operations are also considered as tree operators, instead of as operators on individual nodes. The precise manner in which these operators are used in the Quadtree data-structure in this work as the fundamental grid generation and grid adaptation operators is described in detail in Section 6.3. That section also makes evident the reasons for the choice of the terms "refinement" and "coarsening" to describe, respectively, the node addition and deletion processes, by showing the close association between tree data-structures and the sub-division of spatial regions.

6.3 The Quadtree Algorithm for Grid-Generation and Adaptation

In this section, the procedure adopted in this work to generate Cartesian Adaptive Grids and to adapt them, using the Quadtree data-structure and the Quadtree Spatial Subdivision Algorithm, is described. The implementation techniques used, and the constitution and organization of the data that encodes the grid representation are also explained. The parallelism between the data-structural operations and the geometric operations they represent is emphasized. The most important computational and algorithmic properties of the Quadtree Spatial Subdivision Algorithm are also described in this section, as well as in the next section.

6.3.1 The Basic Grid-Generation Algorithm

The skeleton of the Quadtree Spatial Subdivision Algorithm followed in this work to generate a grid for boundaries with arbitrary geometries is as follows:

1. Allocate and initialize the root node of the Quadtree. This Quadtree is the tree data-structure that is to be "grown" into a representation of the grid. The corresponding geometric operation is the formation or definition of a square that completely encloses the entire Computational Region. This correspondence is effected by storing the geometric and topologic properties of this all-enclosing square into the attribute data of the root node. In particular, the four real numbers representing the minimum and maximum x-coordinates and the minimum and maximum y-coordinates of the (axis-aligned) square are stored as attribute data of the root node, and this explicitly associates the root node of

the Quadtree data-structure with the square. This all-enclosing square may therefore be called the **root square**, or the **root cell**. The definitions and functions of all the individual elements of the attribute data are described in detail below.

2. Recursively refine the (single-node) Quadtree generated in Step 1 above until a pre-specified termination condition for the refinement operator is attained at every leaf node in the final Quadtree. The corresponding geometric operation to refinement of a node in the Quadtree data-structure is the splitting of a square into four squares by the two bisectors connecting the mid-point of each edge of the original square with the mid-point of its opposite edge. The correspondence between the nodes in the Quadtree and the square Cartesian cells they represent is automatically maintained during refinement by appropriate calculation of the appropriate elements of the sub-node attribute data from the corresponding elements of the super-node attribute data, as described below. The termination condition for the refinement operator is formulated in terms of the resolution of the geometric or solution features within the square corresponding to the node, as described more specifically in Sub-Section 6.3.4 below. The node refinement procedure is described in detail in Section 6.2. For the specific application of grid generation, the recursive refinement procedure can be summarized as follows: the termination condition is evaluated for the root node. If the condition is satisfied, then the grid generation procedure terminates, leaving a grid with a single computational cell. If the termination condition is not satisfied, the root node is refined, forming four sub-nodes, which are also the root nodes of four sub-trees, say, trees A, B, C, and D. Each of these four sub-trees is then visited in turn and the procedure that was

applied to the parent Quadtree is now applied to each of these sub-trees. Thus, starting with the first sub-tree, sub-tree A, the refinement-termination condition is evaluated at its root node. If the termination condition is not satisfied, this leaf node is refined and the four sub-trees that result from the refinement of the root node of subtree A are recursively visited in turn and evaluated for refinement, and so on, until sub-tree A satisfies the refinement-termination condition. The recursive refinement procedure is then applied to sub-tree B, then sub-tree C, and then sub-tree D.

The remainder of this sub-section is devoted to explanations of the most important sub-steps in the above algorithm, and to descriptions of implementation details and the manipulation and use of the attribute data of the nodes. Additional details about the transfer of data between sub-nodes and super-nodes (and vice-versa) during refinement or coarsening of nodes is presented in the next section. The refinement-termination condition, which clearly plays a crucial role in the grid-generation process, is described separately in Sub-Section 6.3.4.

The root square described in Step 1 above must enclose the entire region in which a discrete solution is sought (and this implies that all the boundaries about which the flowfield is to be computed must be wholly contained within this square). In this work, the size of this root square is determined as follows:

- For every i satisfying $1 \leq i \leq ns_j$ and every j satisfying $0 \leq j \leq n_b$, find $x^{i,j}_{min}$, $x^{i,j}_{max}$, $y^{i,j}_{min}$, and $y^{i,j}_{max}$, which are, respectively, the minimum and maximum x-coordinates and the minimum and maximum y-coordinates of the bounding box for the i^{th} spline of the j^{th} boundary in the computational model, where ns_j is the number of splines in the j^{th} boundary, and n_b is the number of boundaries (or bodies) in the computational model.

- Compute $x^g_{min} = \min_{i,j}(x^{i,j}_{min})$, $x^g_{max} = \max_{i,j}(x^{i,j}_{max})$, $y^g_{min} = \min_{i,j}(y^{i,j}_{min})$, and $y^g_{max} = \max_{i,j}(y^{i,j}_{max})$, where x^g_{min}, x^g_{max}, y^g_{min}, and y^g_{max} may be regarded as specifying the extreme coordinates for a bounding box that encloses all the splines in the computational model, that is, as specifying the extreme coordinates for a bounding box that encloses all bodies or boundary geometries in the computational model.

- Compute $x_{max} = \max(|x^g_{min}|, |x^g_{max}|)$, and $y_{max} = \max(|y^g_{min}|, |y^g_{max}|)$.

- Compute $l_{max} = \max(x_{max}, y_{max})$, and multiply l_{max} by a scaling factor set by the user, and typically ranging from 2 to 20 to obtain the absolute value of all four extreme coordinate values of the root square, $\tilde{l}_{max}$. Thus, the root square is defined to be the square that is centered at the origin and that has side length equal to $2\tilde{l}_{max}$.

The first step in finding the root square uses the bounding boxes of the splines, and not the coordinates of the spline end-points. This is because individual points in a curved spline may be located outside the bounding box that encloses the end-points, and only the spline bounding box is guaranteed to enclose every point in the spline, including the end-points.

Initialization of a node refers to allocation of the memory required to store the attribute data of the node, and calculation and assignment of this attribute data to its correct values. The attribute data plays a crucial role in the definition of the grid: although the grid-generation procedure described above will create all the square Cartesian cells which will be used to construct the grid, and although it also generates all the required connectivity data that links these Cartesian cells, the computational cells on which the gasdynamic computations are performed still have to be formed from the square Cartesian cells by intersecting the square Cartesian cells with any boundaries that are present in the Computational Region. The intersection analysis is used to obtain the exact geometry of the computational cells, and the attribute data is what stores this geometry. The attribute data is also what stores the connectivity between the computational cells and the properties in those cells, such as the gasdynamic states and other solution data.

The geometry of a cell is defined depending on the type of the cell. Two different types of cell may be identified: (i) cells that are not cut by any boundary (that is, cells that lie either wholly outside or wholly inside closed boundaries); and, (ii) cells that are cut by boundaries. For the first type of cell, the computational cell is given exactly the same geometry as the corresponding square Cartesian cell. For the second type of cell, the geometry of the two or more connected sub-regions that are formed from the intersection of the square Cartesian cell with a boundary is used to define two or more computational cells from the original square cell. In general, each cut cell defines two or more arbitrary polygons, each with a number of sides, n, satisfying $n \geq 3$. The computational cells in the grid therefore form a set of cells in which each member is either a regular (uncut) Cartesian cell, or a non-degenerate polyhedral cell with at least three (distinct) faces.

The procedure used to determine the intersection geometries of the axis-aligned edges of a square cell with all the splines of all the intersecting boundaries was described in detail in Chapter V. As explained below, the exact geometry of the computational cells need not be stored. For uncut Cartesian cells, the geometry of the edges can be determined directly from the geometry of the corresponding square Cartesian cells. For cut cells, the lengths and centroids of all edges (whether they represent interior faces, or boundary faces) can either be determined at the start of each time-step and stored, or can be computed whenever they are needed, as explained further below. For calculations with no moving boundaries, it is usually advantageous to calculate the geometry and store it once, at the beginning of the computation.

An implied function of the intersection analysis is also to identify which part of a cut cell falls inside a boundary, and which part falls outside the boundary, since this is what determines which part of an edge is occluded, and which part is an interior edge of the grid. The determination of "in" or "out" for a point is done in this work by counting the number of times that an infinite ray from that point cuts the boundary that is being considered. If the number of cuts is even, then the point is outside the closed boundary. If the number of cuts is odd, the point falls inside the closed boundary being considered. Special attention must be used in this algorithm in deciding how to count the number of intersections when the ray passes through a vertex of the discretized boundary geometry. If this occurs, the local neighborhood of the vertex and the infinite line must be analyzed further to determine the counting increment. If two points on the infinite line in the immediate proximity of the intersection point are both outside the body or are both inside the body, the the number of intersections is incremented by two (representing a tangency event). If only one of the points is outside the body and the other is inside the body, then the number of intersections is incremented by one (representing a "piercing" event).

In addition to storing the geometric definition of the computational cells, the attribute data stores a variety of other topologic and connectivity data that encodes the complete grid connectivity as defined in Chapter IV. For some of this data, the actual values stored depends on the number of boundary positions n_p that are incorporated in the solution procedure during each time-step in the computation (varying from 1 for computations with stationary boundaries, up to an arbitrary number, typically less than 4, for computations with moving boundaries). In this work, the attribute data stored for each node in the Quadtree is as follows:

1. The minimum and maximum values of the x-coordinate in the cell corresponding to the tree node, and the minimum and maximum values of the y-coordinate in the cell corresponding to the tree node. These four real numbers are stored for immediate access even though they can be computed whenever they are needed, from the geometry of the root cell and the position of the node in the tree traversal path;

2. A pointer to the (possibly non-existent) super-node of the current node;

3. A pointer to the four (possibly non-existent) sub-nodes of the current node;

4. The refinement level (that is, the depth) of the node in the tree;

5. The n-volume (that is, here the area) of the cell corresponding to the node for each of the boundary positions that are stored during a time-step (stored in an array of real numbers of dimension n_p);

6. The x- and y-coordinates of the cell centroid for each of the boundary positions that are stored during a time-step (stored in two arrays of real numbers, each of dimension n_p);

7. A pointer to the (possibly non-existent) *cell* data-structure. The cell data-structure retains the attribute data of the computational cell that is associated

with the current node. It should be noted that a distinction is made between a *node* and a *cell* in this work. The node is the elemental member of the tree data-structure; in addition to storing all the data required for establishing individual membership and connectivity in the tree, it represent a square spatial region as explained above, and therefore stores the geometric attributes of the spatial region that it represents, and the manner in which this region is intersected by boundaries. The *cell* data-structure is used to encapsulate the more specialized representation associated with the node, which depends on the ultimate application of the Quadtree. For example, if the algorithm is used for storing images, the cell could store attributes such as intensity, color, etc. In this work, the cell encapsulates the fluid-dynamic and other solution data associated with the computational cell that is associated with the node. Thus, the cell data-structure here stores the state vector, the gradient tensor, and the local time-step in the corresponding computational cell, as well as other data that is associated with the specific representation of computational cells. The cell data-structure is allocated and activated only for actual computational cells, that is, only for leaf nodes in the Quadtree, since interior nodes in the tree do not represent computational cells;

8. A pointer to the (possibly non-existent) computational group to which the node belongs. This computational group is only activated for actual computational cells, that is, only for leaf nodes in the tree, and is dynamically allocated and de-allocated as the boundary moves, causing different computational cells to be grouped into different merging groups as part of the solution procedure, as explained in detail in Chapter VII;

9. A pointer to the (possibly non-existent) *intersection-configuration* data-structure. This is a data-structure that stores a detailed description of the manner in which the Cartesian square corresponding to a node is intersected by the boundaries in the computation, for each of the n_p boundary positions that are stored during a time-step. This description includes the following data: (i) the intersection type of the node (from among the exhaustive and mutually exclusive possibilities of "interior", "intersected", or "exterior"); (ii) the *sub-region numbers*, specifying the identification numbers of all boundaries whose interiors are at least partially overlapped by the node, and this data is needed and used even for non-leaf nodes, such as the root node (which by definition intersects every boundary in the Computational Region and overlaps every fluid and solid sub-region of the Computational Region); and, (iii) the integers specifying (if applicable) the identification numbers of the edges of the Cartesian square that are intersected by boundaries, the identification numbers of those intersecting boundaries, the identification numbers of the individual intersecting splines in those intersecting boundaries, the real numbers specifying the coordinates of the intersection points, and the real numbers specifying the coordinates of the Gauss Points at which are evaluated the "left" and "right" state vectors that are supplied to the numerical flux function that computes the gasdynamic face flux. The intersection-configuration data-structure is allocated and used only when it is desired to minimize the computational effort for the calculation at the expense of the required memory;

10. A pointer to the *image* of the node in a special link-list data-structure. This link-list data-structure is periodically constructed from the Quadtree, then used for post-processing and to output the solution and the restart data to data-files,

and then destroyed;

11. Integer parameters specifying the selections for algorithms and operators to be used for performing standard operations on the nodes. These parameters are entirely optional and are used only to over-ride default selections.

The node data-structure can easily be modified to store additional attribute data, and to store links (or pointers) to other, new or pre-existing data-structures. As usual, the choice of whether to store a certain datum or to repeatedly re-calculate it reflects a compromise or trade-off between the computational effort requirements and the memory requirements, and the modularity and flexibility of the various individual data-structures used in this work allows this trade-off to be easily changed within the same algorithm.

Finally, it should be noted that although recursion is a central feature of the algorithm developed in this work for construction of the Quadtree, its use in this work reflects an algorithmic or implementation choice, and not a mandatory or fundamental requirement. The absence of a mandatory need for recursion in generating the Quadtree can be deduced from the fact that any recursive algorithm which is of the *tail-recursion* or *head-recursion* type (as is the case here) can be transformed into an equivalent non-recursive algorithm. The motive for choosing a recursive implementation is that it is simpler, more elegant, and more compact than the equivalent non-recursive alternatives.

6.3.2 The Basic Adaptation Framework

Adaptation in a Quadtree-based Cartesian grid is accomplished most readily and efficiently by local refinement and local coarsening, that is, using the pure h-type of adaptation (as defined, for example, in Section 4.5). This suitability of h-adaptation can be considered a characterizing feature of Quadtree-based Cartesian grids, to the extent that a complete framework for adapting a Quadtree-based, Cartesian grid can be established merely by assembling algorithms for executing and controlling local refinement and local coarsening operations that are driven either by geometric or solution features, or by changes in these features. If these local refinement and coarsening operations are driven by geometric features or by changes in these features, the process is identified as geometric adaptation; if these local refinement and coarsening operations are driven by solution features, the process is identified as solution adaptation. The manner in which these two specific processes are developed and implemented in this work are respectively discussed in Sections 6.4 and 6.5. As emphasized in several places in this chapter and elsewhere, the same refinement and coarsening operators are used for geometric adaptation and for solution adaptation (and even for grid generation), and these operators function in an identical manner regardless of the cause for which they are invoked.

The suitability of pure h-adaptation for Quadtree-based Cartesian grids follows from the fact that, as explained in Sub-Section 6.3.1, the Quadtree-based Cartesian grid is constructed by repeated refinement of cells (possibly using a recursive algorithm, as done in this work). This implies that the local refinement operation must already be available, at least as part of the grid-generation procedure. The remaining major operation required for adaptation, the local coarsening operation, can be directly and simply constructed from the same fundamental data-structural operators as the refinement operation, as outlined in Section 6.2.

A characterizing feature of the local refinement and the local coarsening operations in a Quadtree-based Cartesian grid is their simplicity, from both the data-structural and geometric aspects, as described in detail in Section 6.2. For example, the geometric equivalent of local refinement and local coarsening amounts respectively to the splitting of square cells into four square cells, and to the combining of four square cells that belong to a single superior cell together into the superior cell, as explained in Section 6.2. This simplicity makes these operations highly efficient.

The simplicity and the computational efficiency of h-adaptation in Quadtree-based Cartesian grids, as discussed above, are two of the main reasons for choosing to confine the adaptation type used in this work exclusively to the h-type. The other main reasons for doing so are more related to the deficiencies of the other alternative types of adaptation for the objectives of this work (which are described in Section 1.5), and are as follows:

- r-adaptation is, as explained in Sections 1.3 and 4.5, highly limiting, and places strict constraints on the allowed magnitudes of the motion of boundaries, and on topologic transformations. It also does not allow the total number of cells in the grid to change, and this alone proves to be too restrictive a constraint for transient problems with evolving, interacting features that may multiply in number and in total length (or area);

- p-adaptation is not appropriate for standard Finite-Volume solution-schemes, such as the one adopted in this work, because of the extreme difficulty of raising the order of the spatial accuracy in such schemes beyond 3; and,

- Re-meshing adaptation is neither necessary nor appropriate because of the use of cell-merging, the new technique developed in this work and generalized to handle moving-boundary problems.

The main advantages of h-adaptation over other types of adaptation were outlined in Section 4.5, and can be re-iterated as follows:

- Simplicity of implementation, especially for Cartesian grids in general, and for Quadtree-based Cartesian grids in particular;
- Absence of inherent restrictions on how the total number of cells in the grid may change during a computation, and allowance for arbitrary resolution at any point in the Computational Region; and,
- Inherent facilitation of the conservation of the Conserved Variables during refinement and coarsening, without the need for any special interpolation or projection calculations (as described in detail in the next sub-section).

As implied above and in preceding sections, an important feature of the Quadtree-based Cartesian grid is that any intensity of resolution may be obtained in any part of the Computational Region, without decreasing the resolution in another part, simply by locally adjusting the refinement level in the Quadtree to the desired level, and that this can be accomplished while ensuring the smoothness of the grid.

6.3.3 The Transfer of Attribute Data Between Nodes During Refinement and Coarsening

As described in the preceding sub-section, some of the attribute data which must be initialized or defined for a node whenever a node is created by refinement can be computed de novo from only fundamental global data, without regard to the connectivity of the node or the values of the attribute data of other nodes. This is the case for almost all of the geometry-related attribute data. Some of the attribute data, however, can only be derived from the corresponding attribute data of the node which is being refined to obtain the new node. Similarly, if a node is being coarsened, some of its attribute data can also only be derived (or updated) from the corresponding attribute data of its immediate sub-nodes.

Because of the need described in the preceding paragraph for data transfer between superior and sub-ordinate nodes, it is convenient for the refinement and coarsening operators to perform as much as possible of the attribute data calculation and assignment as an intrinsic sub-function of the operator, just as checking whether a node is leaf node or a penultimate node before performing a refinement or coarsening is also best included as an intrinsic sub-function of the operator.

An important example of the dependence of the attribute data of sub-nodes on the attribute data of their super-node is provided by the coordinate values that specify the geometry of the Cartesian squares corresponding to the Quadtree nodes. Whenever a node in the Quadtree is refined, the coordinate data of its sub-nodes is computed (partly by direct assignment, and partly by arithmetic averaging) from the coordinate data of the super-node. These calculations are done as part of the refinement operation, and making the calculation procedure part of the refinement operation ensures that the refinement operation intrinsically preserves the association between the nodes in the Quadtree and the Cartesian squares that they represent.

An important example of the dependence of the attribute data of sub-nodes on the attribute data of their super-node, and of the dependence of the attribute data of a super-node on the attribute data of its sub-nodes is provided by the gasdynamic state vector. Whenever a node is refined, the gasdynamic state vectors of its sub-nodes are calculated from that of the node, and whenever a node is coarsened, its gasdynamic state vector is calculated from those of its sub-nodes. These calculations are done as part of the refinement and coarsening operations, and making them part of these operations intrinsically ensures that mass, momentum, energy, and any other properties of the solution are transferred correctly between nodes.

Whatever the type of attribute data involved, its transfer between nodes must be performed correctly, consistently, and accurately. In the case of geometric data, there is usually little ambiguity about the manner in which the data must be transferred, calculated, or apportioned. As explained in Section 3.4, however, an additional, special requirement for any moving boundary composed of multiple straight-line segment is that its exact, un-truncated geometry must be preserved. Otherwise, discrepancies in the truncations of such a boundary as it moves across the cells of the Computational Region will introduce errors in the volume and swept-area calculations, causing violations of The Geometric Conservation Laws, as described in more detail in Section 3.4. In the case of gasdynamic data, an essential requirement is that the data should be transferred in a manner that ensures "conservation" of the conserved variables. Conservation during refinement and coarsening is ensured if and only if the following formula is satisfied

$$\vec{U}\sum_{i=1}^{n_s} V_i = \sum_{i=1}^{n_s} \vec{U}_i V_i \tag{6.1}$$

where $\vec{U}$ is the state vector of the super-node, $\vec{U}_i$ is the state vector of the i^{th} sub-node, V_i is the n-volume (here, the area) of the i^{th} sub-node, and n_s is the number of sub-nodes (which here is always equal to 4). The volumes V_i are specified individually and explicitly in the formula because they are not necessarily always equal (due to cutting of cells by boundaries). The specific manner in which gasdynamic data is transferred or transcribed during refinement and coarsening operations is next discussed.

During coarsening, the following conservation-satisfying formula is used to obtain the state vector attribute data of the super-node from the state vector attribute data of its sub-nodes

$$\vec{U} = \frac{\sum_{i=1}^{4} \vec{U}_i V_i}{\sum_{i=1}^{4} V_i} \tag{6.2}$$

where all terms are as defined for Equation 6.1 above. Since the dependent variables in this work are chosen to be the conserved variables, this is the only calculation required for the gasdynamic attribute data during a coarsening operation. The gradients of quantities need not be computed during the coarsening operation, since they are independently computed just before they are needed, from the state vectors, as part of the solution procedure, as explained in more detail in Section 3.2. However, for consistence, and in order to provide appropriate initial values, the gradients during coarsening are initialized using a weighted average of the form

$$\nabla \phi = \frac{\sum_{i=1}^{4} \nabla \phi_i w_i V_i}{\sum_{i=1}^{4} w_i V_i} \tag{6.3}$$

where $\nabla \phi$ is the gradient of the variable ϕ in the super-node, $\nabla \phi_i$ is the gradient of the variable ϕ in the i^{th} sub-node, V_i is the n-volume (here, the area) of the i^{th} sub-node, and w_i is the weighting for the i^{th} sub-node. Different weightings w_i are used in this work, depending on the variable whose gradient is being calculated: when ϕ corresponds to either density or pressure, the weighting w_i is set to 1; when ϕ corresponds to any of the components of the velocity vector, the weighting w_i is set to the density of the corresponding sub-node, that is, using $w_i = \rho_i$. As explained in Section 3.2, the gradients needed in the solution algorithm are only those for the four primitive variables listed above. No strong reason exists for using density weighting in the calculation of the gradients of the velocity components, and other weightings, including $w_i = 1$ are comparably suitable.

During refinement, the following conservation-satisfying first-order-accurate formula is used to compute the value of the state vector for each of the sub-nodes from the value of the state vector of the super-node

$$\vec{U_i} = \vec{U} \tag{6.4}$$

where $\vec{U}_i$ is the state vector of the i^{th} sub-node, and $\vec{U}$ is the state vector of the super-node. The corresponding second-order-accurate accurate formula used for this purpose is equivalent to the formula

$$\vec{U}_i = \vec{U} + \nabla\vec{U} \cdot (\vec{x}_{c_i} - \vec{x}_c) \tag{6.5}$$

where $\nabla\vec{U}$ is the gradient tensor of the vector of conserved variables in the super-node, $\vec{x}_{c_i}$ is the centroid location of the i^{th} sub-node, and $\vec{x}_c$ is the centroid location of the super-node. Using the fact that the slope in the super-node is treated as constant, it can easily be proved that computing the sub-node state vectors in the manner of Equation 6.5 ensures satisfaction of the conservation requirement, that is, that it ensures that Equation 6.1 is satisfied. During refinement, the gradients of all the gasdynamic properties in the sub-nodes are also initialized to their corresponding values in the super-node, that is, using the formula

$$\nabla\phi_i = \nabla\phi \tag{6.6}$$

where all terms are as defined for Equation 6.3. These gradients are then re-calculated just before they are used in the solution algorithm, as described in detail in Section 3.2. A calculation of the gradients that is higher-order-accurate than described above would require second and higher-order derivatives for the super-node, and is therefore not attempted here.

6.3.4 Specific Formulation of The Refinement Operator

The refinement-termination condition (for the halting of the refinement of Quadtree nodes) that was described in Step 2 of the basic grid-generation algorithm in the preceding sub-section is the only significant "free parameter" in that algorithm. Since this condition also largely determines the final grid that results from the recursive refinement procedure, its formulation must ensure that it incorporates all the required desiderata that are to be satisfied by the final grid. The default condition adopted in this work is, not surprisingly, a composite of several sub-conditions: it is evaluated by successively evaluating an ordered set of tests until the termination condition, which is *undetermined* at the start of testing, is either found to be *satisfied* or *unsatisfied*. If the condition is satisfied, then the node at which it was evaluated will not be refined; if it is unsatisfied, then the node at which it was evaluated will be refined. The ordered set of tests is as follows:

- Find all the spline segments (of all boundaries) that intersect the Cartesian square cell corresponding to the current node. A spline is determined to intersect a cell if there is at least one point in the Computational Region which belongs to both the spline and the cell. The search for these splines is made significantly faster than an exhaustive search by the use of the bounding-boxes of the splines, and by use of a tree-based data-structure in which these splines are arranged according to their location in the root square of the Quadree data-structure of the grid. If the number of intersecting splines is zero, the termination condition is satisfied, and the subsequent tests below are not evaluated. If the number of splines is greater than zero, the termination condition remains undetermined at the end of this first test.

- If the the number of intersecting splines evaluated in the previous test is one, determine if the resolution of the current cell (given by the refinement level in the Quadtree of the corresponding node) matches a pre-specified resolution level given by the user for that spline of that boundary. This condition allows specification of a minimum level of resolution along each spline of each boundary. If the specified level of resolution is matched or exceeded, then the termination condition is satisfied, and the subsequent tests given below are not evaluated. If the specified level of resolution is neither matched nor exceeded, then the termination condition is unsatisfied, and the subsequent tests given below are also not evaluated. As explained above, this failure will cause the current node to be refined. If the number of splines is two or more, the termination condition remains undetermined at the end of this second test.

- Determine if the (two or more) splines that intersect the current Cartesian square belong to different boundaries. If they do, then the Cartesian cell is determined to not sufficiently resolve the geometry, and the termination condition is unsatisfied, and the subsequent tests given below are not evaluated. This failure will cause the current node to be refined. The purpose of this sub-test is to ensure that different boundaries do not cut the same cell. For non-coalescing moving boundaries, the geometry update procedure independently ensures that the nearest distance between any two spline segments of any two bodies does not fall below a given value. Without this constraint, infinite grid resolution may be pursued in the attempt to resolve a thin gap between the boundaries. If the spline segments that intersect the current cell do not belong to more than one boundary, the termination condition remains undetermined at the end of this third test.

- As implied in the preceding test above, this final test is applied only if all of the (two or more) spline segments intersecting the Cartesian square belong to the same boundary. Determine if the number of these spline segments exceeds two. If it does, and if these segments are consecutively ordered, and collinear, and if they all individually satisfy the refinement level requirement for each of them, then the Cartesian square is determined to sufficiently resolve the geometry, and the termination condition is satisfied. Otherwise, the termination condition is unsatisfied. The purpose of this test is to ensure that opposite spline segments of a boundary do not cut the same cell, and that, for example, a boundary does not split a cell into disjoint parts (or subregions). However, in order to ensure that this test does not result in infinite refinement of a given cell, it is necessary to ensure that none of the control points of the splines coincides with a potential vertex of a Cartesian cell. This coincidence is always prevented, as described in Chapter V above. The condition on the collinearity of the segments may be relaxed to a condition on a threshold value for the sum of the absolute values of the angular displacements along the segments, given by $\sum_{n=1}^{N-1} |\theta_n| \leq \theta_L$, where N is the number of segments intersecting the cell, θ_i is the angle between segment i and segment $i+1$, and θ_L is a pre-specified threshold on the sum of the absolute angular displacements. This modified form is effectively a condition for refinement based on a measure of the curvature of the boundary, and it may also be replaced by more precise evaluations for the curvature.

Close examination of the composite termination condition described above shows that it ensures that all boundary geometries within a Cartesian square cell are resolved, including the individual geometric features of a particular boundary, and including the gaps between different boundaries. This is true provided all geometric length scales can be resolved within the arithmetic precision of the machine on which the calculations are being performed. Indeed, because of this precision limitation, it is useless (and also dangerous) to attempt to refine nodes beyond a tree depth of

40-50 for double-precision calculations (since $\left(\frac{1}{2}\right)^{50} < \phi$, where ϕ is the precision of arithmetic operations for double-word floating numbers on most modern 32-bit-word processors used in Workstations and Personal Computers). For this reason, an upper limit on the refinement level, here chosen to be 40 is imposed as an additional constraint in the refinement operator.

By construction of the refinement traversal, the termination condition described above would have to be satisfied at each node in the final Quadtree. For problems with moving boundaries, the termination condition must also be satisfied at each node for every one of the n_p boundary positions that are retained in the solution procedure during a time-step. This matter is discussed further in the next subsection.

The ordering of the tests in the composite termination criterion given above reflects heuristic choices that minimize the number of unnecessary tests that are evaluated for any given node.

The overall computational effort required for the grid-generation algorithm developed in this work evaluates in the best-case scenario to $\vartheta(n \log m)$, where n is the number of nodes in the tree (which is proportional to l, the number of leafs in the tree, which in turn is the same as the number of computational cells), and where m is the total number of splines (for all the boundaries or bodies) in the Computational Region. This is the case for the basic grid-generation algorithm described above, using the general-purpose refinement operator described above (which is mostly based on geometric resolution criteria).

6.3.5 Extension and Generalization of the Quadtree Algorithm, Especially for Unsteady Flows, Moving Boundaries, and Adaptive Calculations

The skeleton algorithm for generation of the initial grid that was outlined in Sub-Section 6.3.1 and elaborated in Sub-Section 6.3.4 is extended, enhanced, and generalized in the actual implementation. The main purpose of these modifications is to enable the algorithm to be used not only for generation of the initial grid, but also to be used for continuously changing the grid during the progress of the solution, to enable the handling of moving boundaries, and the handling of adaptation to the solution and to changes in the geometry of the boundaries.

The need for the extension and generalization described in the preceding paragraph arises because the basic algorithm described in Sub-Section 6.3.1 is invoked only once at the beginning of the solution procedure, and because the refinement-termination condition it uses (which is described in detail in Sub-Section 6.3.4) accounts only for the influence of the initial boundary geometry on the grid-generation process. These two factors imply that the algorithm outlined in Sub-Section 6.3.1 and elaborated in Sub-Section 6.3.4 without any enhancements would be sufficient only for generating a non-adaptive initial grid that remains unchanged throughout the course of the computation.

The extension and generalization described above to enable handling of unsteady-flow problems, moving-boundary problems, and adaptive computations is accomplished in this work mainly by invoking the basic grid-generation algorithm described in Sub-Section 6.3.1 not once only at the beginning of the solution procedure, but instead at regular intervals throughout the solution procedure. Furthermore, in each invocation, the basic grid-generation algorithm is actually invoked multiple times (instead of only once), each time with a different refinement-termination condition, including refinement-termination conditions that reflect the adaptation to the geometry and to the solution values.

The re-constitution and multiple invocation described above of the basic grid-generation algorithm not only generalizes the basic algorithm to enable handling of moving geometries, unsteady flows, and adaptive computations, but also unifies the treatment for the various types of problems that may be solved with the method developed in this work, and also makes the grid-generation process more flexible and more computationally-efficient. In the most general and unifying case, each invocation of the skeleton algorithm described in Sub-Section 6.3.1 results in the application of the algorithm four successive times (corresponding to four successive stages of grid generation or adaptation), each with a different refinement-termination condition. This is done every set number of solution-update cycles. These four different stages and the associated refinement-termination conditions are as follows:

- Stage 1: Generation of a Uniformly-Refined Grid: The refinement-termination condition is set to the following evaluation procedure: the condition is unsatisfied if the refinement level of the target node is less than a pre-specified value (which corresponds to the uniform refinement level desired in the grid); and, the refinement-termination condition is satisfied if the refinement level of the target node is greater than or equal to the pre-specified value. No regard whatsoever is given to the geometries of boundaries in this condition. The grid-generation algorithm described in Sub-Section 6.3.1 is launched with this refinement-termination condition (instead, for example, of the refinement-termination condition described in Sub-Section 6.3.4), resulting in the generation of a Cartesian grid of uniform refinement throughout the Computational

Region without any influence from the intersecting boundaries.

- Stage 2: Adaptation of the Grid to Boundary Geometries: The refinement-termination condition is set to the general, geometry-related refinement condition described in Sub-Section 6.3.4. The grid-generation procedure is launched with this refinement-termination condition, resulting in the adaptation of the grid generated from Stage 1 above to a grid that fully resolves all the geometric features in the Computational Region (in addition to satisfying the conditions imposed in Stage 1 above). The grid that results after this stage is characterized by a high resolution around all boundaries, which decays with distance from the boundaries, but only down to the uniform refinement level generated through the application of Stage 1 above. As explained in Sub-Section 6.3.4 above, the resolution around each boundary could either be uniform, or could increase in regions of higher curvature. In practice, for problems with complex geometry, this stage increases the number of cells in the grid significantly and is the most time- and memory-consuming step in the grid-generation process. As explained in Section 6.2, even if the refinement process is launched with a pre-existing Quadtree, any refinement operations will only be applied to leaf nodes.

- Stage 3: Adaptation of the Grid to the Solution Features: The refinement-termination condition is set to a general, solution-based condition which is described in detail in Section 6.5. The grid-generation procedure is launched with this condition, resulting in a grid that is fully adapted to any solution features being targeted in the current grid, in addition to meeting the resolution criteria satisfied in Stages 1 and 2 above. When invoked with the initial conditions of the solution (that is, with the conditions corresponding to time $t = 0$), this procedure results in a grid that is fully adapted to the initial conditions of the solution. This adaptation to the initial conditions is necessary for any time-accurate computation, especially if the initial conditions contain discontinuities.

- Stage 4: Application of Special-Purpose Refinement or Adaptation: The refinement-termination condition is set to a specific, user-defined condition that typically reflects a minimum desired level of resolution in certain sub-regions of the Computational Region. The grid-generation procedure is launched again with this fourth refinement-termination criterion, resulting in a grid that in addition to meeting all the resolution criteria satisfied in Stages 1, 2, and 3 above, also meets the special resolution criteria specified by the user. A typical application for such a condition arises if the user desires a certain level of accuracy in a certain location and wishes to obtain a specific, high level of resolution in that region that is independent of any solution adaptation.

Although the order of the four grid-generation stages described above may be interchanged, the specific ordering described above is the most logical and efficient.

The four-stage grid-generation procedure described above is used to generate the initial grid “from scratch”. However, examination of the individual stages involved and the actions they perform shows that repeated application of the four-stage procedure can be used to update the grid continuously during the overall solution procedure to account for changes in the geometry and in the solution, as indicated above. Such an examination also shows that some of the stages may be omitted during repeated invocations, depending on the type of problem being computed. As also indicated above, this repeated invocation of at least some of the stages in the above four-stage procedure (together with additional coarsening procedures, as described below) is indeed the means followed in this work to extend the basic grid-generation process to special requirements, such as to the handling of the motion of boundaries in the Computational Region, or to the handling of adaptation to the evolution of the solution. This possibility is partly a consequence of the fundamental generality of the node refinement procedure. The specific sequence of actions that are followed for different categories of problems in which grid adaptation is used are described below.

The only additional requirement for dynamic grid adaptation that is not explicitly described in the four-stage procedure described above is grid coarsening. Grid coarsening is necessary whether the problem is a steady-state or a transient one (with or without boundary motion). With no coarsening, regions of high refinement that are created during convergence to a steady-state solution, or regions of high refinement that are vacated by traveling solution features during transient computations will remain at high resolution, even if they are ultimately not required for the final solution. Grid coarsening can be considered as a fifth, additional stage, or as an extension to Stages 2, 3, and 4 described above. Just as it is for refinement, coarsening can be geometry-related or solution-related. The manner in which the geometry-related and the solution-related coarsening is applied is respectively described in detail in Sections 6.4 and 6.5.

Although the need to perform coarsening may be driven by or first triggered by geometry-related factors or by solution-related factors, grid coarsening is actually applied (only to the penultimate nodes in the Quadtree, as explained in the Section 6.2) only if all the leaf nodes of the target node simultaneously over-resolve the geometry *and* over-resolve the solution. Thus, each leaf node must be examined to determine whether all geometric features in it (that is, all the splines of all the boundaries in it) will remain adequately resolved if inserted in the super-node, and whether the solution will remain adequately resolved if projected to the super-node. If this test passes for all four leaf nodes of the current target node, then the target node may be coarsened. The specific implementation of the coarsening operation for a node is described in Section 6.2.

The specific sequence of global operations followed during an invocation of the grid-generation and grid-adaptation algorithm for the different categories of problems that can be computed with the overall solution technique developed in this work is as follows:

- Steady-State Problems: The initial grid is generated using the standard four-stage procedure described above. During the course of the computation, the grid is adapted by re-invoking the grid-generation process with only the refinement-termination conditions described in Stages 3 and 4 above, that is, with stages 1 and 2 omitted. This adaptation can be done every solution-update cycle (that is, every iteration, or every time-step if using a time-marching solution procedure as in this work), or much more efficiently, every pre-set number, which is typically chosen to be between 10 and 100, of the solution-update cycles. A global coarsening operation, driven by the solution values, just like the global refinement of Stage 3, is also applied every pre-set number of solution-update cycles after execution of Stages 3 and 4, and typical frequencies for the coarsening sweeps are once every 10 to 50 solution-update cycles. Since coarsening cannot decrease the resolution below the geometric- and solution-refinement criteria, because it is confined to coarsening only redundantly-refined cells, the only harm in attempting to coarsen too frequently is waste of computational effort.

- Transient Problems with Stationary Boundaries: The initial grid is generated using the four-stage procedure described above. During the course of the computation, the grid is adapted with only the refinement-termination conditions described in Stages 3 and 4 above, but (in contrast to the case with steady-state problems), the grid must be adapted every solution-update cycle, or every time-step. Otherwise, the solution features may move outside their refinement zones and become subjected to heavy smearing. This need for frequent update is not present in other Cartesian methods, which typically [291] use thicker swathes of refined cells around solution features. Nevertheless, both strategies will eventually adapt a similar number of cells in the course of the entire computation. A coarsening operation can be performed after execution of Stages 3 and 4, every pre-set number of time-steps, which is typically 2 to 10 for this category of problems. Again, just like it is for the preceding category of problems, the coarsenings must be driven by the solution values.

- Transient Problems with Moving Boundaries: The initial grid is generated using the four-stage procedure described above. Thereafter, for every motion step in the computation (and in this work, the motion steps and the time steps always coincide), the grid is adapted with the specific refinement-termination conditions described in Stages 2, 3, and 4 above. Compared to the treatment for transient problems without moving boundaries, this reflects the additional application of the refinement-termination condition of Stage 2. The purpose of this additional application is to repeatedly adapt the grid to the updated boundary positions and shapes after every motion step. Another major distinction in the treatment of problems with moving boundaries compared to the treatment of problems with stationary boundaries is that the refinement adaptation is applied to the multiple positions of a boundary figuring in the discrete solution algorithm instead of just one position. As explained in the preceding sub-section, the intersection evaluations performed in the refinement operator are automatically applied to the n_p discrete positions of every boundary during a motion step. However, $n_p = 1$ for every stationary boundary, while $n_p > 1$ for every moving boundary. As a result of this, the refinement adaptation of Stage 2 is applied to every discrete position that a boundary is required to assume during a motion step in the discrete solution procedure: the initial, the final, and all intermediate positions (if any exist). This ensures that for every motion step, the discrete motion of every boundary (represented by the n_p discrete positions of the boundary) is fully retained in an envelope of appropriately-refined cells.

The grid adaptation is performed ahead of a motion step, before any gasdynamic calculations are performed, and the grid that results from the adaptation procedures described above remains unchanged throughout the motion step. Instead, the computational cells of that fixed grid are recombined in the vicinity of boundaries by cell-merging, as described in Chapter VII, to form topologically invariant composite cells whose n-volume may change during the motion step as a boundary moves across their edges. Thus, the most important role of the geometric adaptation process for moving boundaries is to ensure that around every boundary, an envelope of suitably-refined cells is available in which individual cells can be merged together to form an appropriate set of topologically-invariant composite cells. Another additional consideration for moving boundaries is that the coarsening operation must be performed after execution of Stages 2, 3, and 4, every pre-set number of solution updates, which is typically 1 to 10 for this category of problems. For this category of problems, correct tuning of the frequency of coarsening operations for high-velocity boundaries is important in eliminating unnecessary cells without unnecessarily increasing the computational effort.

Unlike the situation for the preceding two categories of problems, the coarsenings for this category of problems must be driven by both geometry-related criteria, and solution-related criteria.

As mentioned above, deviations from the regular order of invocation of the different adaptation or generation stages may also be beneficial in some cases. For example, for time-accurate computations, the ratio of maximum to minimum n-volume across all the cells in the grid must be bounded in order to bound the phase error in the solution. If the ultimate depth of the tree is not known in advance, it may be necessary to increase the background (or uniform) refinement level of the tree during the calculation, by re-invoking the Stage 1 operation with a higher background refinement level than was specified with the initial invocation, to force the maximum difference in depth between all the leaf nodes in the tree to remain within a certain limit, for example, 5 or so.

No modifications to the algorithms described above are required if coalescence and disintegration of boundaries is activated. For example, if coalescence is allowed, then the boundary definition would be modified exclusively during the boundary update process. For example, if the closest distance between two boundaries has fallen below a pre-specified threshold, then the two boundaries would have been re-splined and re-configured as a single boundary during the geometry update procedure, which occurs before and separately from the grid generation or grid adaptation procedure. The grid-generation and adaptation procedures then operate on the new boundary configuration, without any regard to or influence from the previous geometric configuration or the topological change that has taken place. On the other hand, if the boundaries are defined as non-coalescing (as is the case for rigid bodies), then the smallest distance between these boundaries must be maintained above a certain threshold. This is also done in the geometry update procedure, without any interactions with the grid-generation procedure. If this distance were not maintained, the refinement in the gap between the boundaries will cause the local cell dimension to asymptotically shrink to zero, and this will in turn cause the global time-step for the gasdynamic update to also asymptotically diminish to zero.

An important characteristic of the adaptation procedures adopted and developed in this work is that the geometric-adaptation algorithms and the solution-adaptation algorithms are implemented and invoked separately and distinctly, as implied above. The advantages of this separation are not only reflected in greater modularity of the algorithms and the software implementation, but also in higher overall computational efficiency. As described above, this separation also allows the geometric-adaptation and the solution-adaptation algorithms (whether these algorithms are performing refinement or coarsening) to be invoked with different frequencies, to meet the requirements of the type of problem being simulated more efficiently and effectively. The next two sections describe the geometric-adaptation and the solution-adaptation procedures developed and adopted in this work in more detail.

Another important enhancement to the basic algorithm outlined in Sub-Section 6.3.1 and elaborated in Sub-Section 6.3.4 is specific to the fundamental node-refinement algorithm. This enhancement therefore affects all operations that involve refinement, whether they are being used to generate the initial grid, or to adapt the grid at any stage during the solution process. This modification to the fundamental refinement algorithm is motivated by the requirement that the refinement levels for any two cells that share an edge must differ by no more than one. This requirement ensures that a minimum area-smoothness is maintained throughout the grid. It also simplifies some of the operations that must be routinely performed in the Quadtree data-structure, such as the operation of finding the neighbors of a cell. This constraint is most easily and efficiently imposed within the recursive refinement operator as a permanent additional step, in the following manner: before refining any node, all its edge neighbors are determined, and each edge neighbor is examined to determine its refinement level. If the refinement level of the neighbor is already one less than that of the current target cell, then the neighbor is refined before the current cell is refined. Since the check for refinement level differences will again also be carried out automatically before refinement of the neighbor, the attempt to refine the neighbor could also result in the refinement of a neighbor of a neighbor, and in principle, the attempt to refine a cell in the center of the Computational Region could propagate a refinement chain that reaches all the way to the boundaries of the Computational Region. Such far-reaching effects are rarely observed in practice, however. Since the limiting of the refinement-level-difference is built into the refinement operator itself, the constraint on the maximum refinement-level-difference is satisfied not only in the final grid, but also at each stage during the grid-generation and grid-adaptation processes.

6.4 Adaptation to Geometric Features and Boundary Motion

Section 6.3 describes in detail the algorithms and techniques adopted and developed in this work for grid-generation. It also outlines the main techniques and procedures adopted and developed in this work for grid-adaptation, including treatment of moving boundaries, and adaptation for geometric and solution features, emphasizing how these procedures are integrated into the overall solution procedure. This sub-section focuses exclusively on adaptation to geometric features, collecting, extending, and describing in greater detail all the relevant material presented in Section 6.3, and recasting that material together with additional material into the most appropriate viewpoint for geometric adaptation as a distinct procedure or capability.

An unusual consequence of the design of the overall solution algorithm in this work is that a distinct algorithm that exclusively performs all the geometric-adaptation functions is an artificial, unnecessary construct. This arises because, as explained in detail in Sub-Section 6.3.5, the refinement and coarsening global traversals of the Quadtree that are executed in response to the geometric features of boundaries, and in response to the motion and deformation of boundaries are distributed across a multi-stage, combined grid-generation and grid-adaptation procedure, and are executed separately from each other, with possibly differing frequencies. As explained in the latter sub-section, this distribution and separation are adopted for greater flexibility, more generality, and greater computational efficiency. Despite the separation of the actions of the geometry-related refinement and the geometry-related coarsening, a distinct geometric-adaptation algorithm can still be readily defined and isolated. In this work, it is formed simply by composing the geometry-related refinement and the geometry-related coarsening operations into a single algorithm.

However, this distinct algorithm is actually rarely used in this work. Instead, the distributed form of it described in Sub-Section 6.3.5 is the one used on a routine basis.

Another unusual consequence of the design of the overall solution algorithm in this work is that the geometry-related adaptation that occurs purely in response to the geometric features (and not in response to the boundary motion) can be considered a part of the grid-generation procedure, and not a part of the geometric-adaptation procedure. As explained in Sub-Sections 6.3.1 and 6.3.4, the geometric-adaptation procedure for a fixed boundary can effectively be executed as an intrinsic part of the grid-generation algorithm that creates the initial grid. It is also explained in those sub-sections that the geometric adaptation in that case occurs by examining every Cartesian cell which is intersected by a boundary segment to determine whether the cell meets the minimum resolution requirement for that boundary segment (that is, whether $l_c \leq l_{min}$, where l_c is the cell dimension, and l_{min} is the specified minimum level of resolution on boundaries), and whether it also meets the applicable minimum curvature-refinement requirements. If the examination shows that the cell is coarser than necessary to meet these requirements, the cell is immediately refined during the traversal operation which is performing the grid generation. The distinction between adaptation to a fixed geometry and adaptation to a change in geometry is also mentioned in Section 4.5, and it is suggested there that contrary to popular usage, geometric-adaptation should strictly only refer to adaptation in response to changes in geometry.

As implied in the preceding paragraph, the initial, geometry-adapted grid in this work is generated purely by successive cell refinements, with no coarsenings being needed or applied. For boundaries that move or deform, however, cell refinement alone is not sufficient to generate an optimal, geometry-adapted grid: it is also necessary to coarsen cells, as, and for the reasons described in Section 6.3.5. Global geometry-related coarsening is done in a similar manner to global geometry-related refinement: by examining every Cartesian cell which is intersected by a boundary segment to determine whether it is finer than necessary to meet the resolution requirements. However, coarsening requires some additional checks that are not needed in refinement: (i) the immediate superior of the target cell is examined to determine whether it would be too coarse to meet the resolution requirements if coarsening is applied; and, (ii) the solution-adaptation criteria for the target cell are evaluated to determine whether coarsening would violate them. Coarsening is then performed only if both of these tests return a negative result.

As explained above, the single, distinct, composite version of the geometric-adaptation algorithm is formed in this work by combining the individual operations (described above and in Sub-Section 6.3.5) of global and local refinement and coarsening based on geometric criteria (together with the underlying geometric tests). This algorithm can be invoked independently to perform geometric adaptation on any Quadtree-based grid, regardless of the initial state of the grid. As would be anticipated from the manner in which it is formed, this algorithm operates as follows: it traverses the Quadtree and checks whether any cell that corresponds to a leaf node requires refinement, and whether any cell that corresponds to a penultimate node requires coarsening, using only geometric criteria (but also checking for violation of the relevant solution-related criteria in the case of coarsening), as described above. As also described above, the refinement or coarsening are immediately applied if they are deemed necessary and permissible for any node during the traversal. Another variation of this algorithm can be created by arranging for the geometry-related refinement and the geometry-related coarsening to be performed in two separate sweeps, and in fact this is often the better option, because it is the neater and the more modular one. Whether the refinement and the coarsening are performed in the same sweep or in separate sweeps, the single, composite geometric-adaptation algorithm just described can effectively meet all the required geometric-resolution criteria for all the problem types that can be simulated using the methodology developed in this work, including problems with moving boundaries. As stated in Sub-Section 6.3.5, however, this algorithm is not the one used on a routine basis to perform the geometric-adaptation: the routinely-used algorithm is one in which the operations of refinement and coarsening are separated, and performed in a distributed manner across the four stages of the grid-generation and grid-adaptation procedure.

If all boundaries are stationary and the initial grid has already been generated, invoking the geometric-adaptation algorithm will not cause any changes in the grid. This is the case whether the distributed, separated algorithm is used, or the single, composite algorithm is used. This property can be deduced from the specific algorithm followed to generate the initial grid.

As explained in Sub-Section 6.3.5, if moving boundaries are present, the geometric-adaptation algorithm should be invoked after every motion step, in order to enable the refinement of cells into which geometric features are moving, and in order to enable the coarsening of cells which are being vacated by geometric features. The refinement is necessary to ensure that the boundary representation within the grid remains resolved to within the required truncation error throughout the integration period; the coarsening is necessary to ensure that no cells remain unnecessarily refined and hence that no computational resources are consequently wasted. This requirement on the frequency of invocation of the geometric-adaptation algorithm holds whether the distributed, separated version of the geometric-adaptation algorithm is used, or whether the single, composite version is used.

In addition to performing its refinement and coarsening actions after every motion step, the geometric-adaptation algorithm (in either of its two versions) must meet one other additional requirement for any moving boundary: the adaptation must be applied for every boundary position used in the time-integration scheme, not just the initial or the final positions, as explained in detail in Sub-Section 6.3.5. In order to satisfy this requirement reliably and modularly in this work, the geometric-adaptation algorithm is automatically invoked for each boundary position included in the formulation of the discrete solution algorithm. As explained in Chapter III, the time-integration scheme adopted in this work uses two boundary positions (one at the start, and the other at the end of the timestep), and therefore geometric adaptation is applied only once for every new timestep.

6.5 Adaptation to the Solution

This section describes the manner in which the Quadtree-based Cartesian grid is adapted in this work in response to the discrete values of the computational solution, or to changes in those values. Adaptation that occurs in response to the geometric features of boundaries, or in response to the motion or deformation of these boundaries is discussed in Sections 6.3 and 6.4. The uniformity of the treatments for the steady and the unsteady cases is emphasized, and the absence of special additional requirements for treatment of moving boundaries are explained in detail towards the end of this section. This section also describes the limitations and shortcomings of the adaptation strategies adopted and explored in this work.

The most important concepts, principles, benefits, shortcoming, and definitions related to adaptation of grids to computational solutions are reviewed and discussed in Chapter IV, and especially in Section 4.5. The latter section also describes the main algorithmic techniques used in geometric-adaptation and in solution-adaptation, describes the four main types of adaptation, and compares the advantages and limitations of these four types. This section focuses mostly on the specific solution-adaptation methodology adopted and developed in this work, but presumes many of the ideas and concepts reviewed in Section 4.5. However, in order to also make this section as self-contained as possible in regard to adaptive methods for The System of Euler Equations, the most popular other adaptation techniques, adaptation indicators, and grid-optimization criteria that have been used for that system of equations are also briefly discussed in it as necessary.

The organization of this section aims to present the adaptation methodology adopted, developed, and explored in this work by describing its three main ingredients; namely: (i) the Adaptation Indicators; (ii) the Grid-Optimization Criteria; and, (iii) the Adaptation Operators. As explained in Section 4.5, these three constitutive ingredients largely define a solution-adaptive methodology, and largely determine its capabilities and characteristics.

For convenience, the practical demonstrations of the effectiveness, and of some of the major special features and characteristics of the solution-adaptation approaches developed and adopted in this work, as well as additional discussions of these issues, are given in Chapter VIII instead of in this section. In particular, Chapter VIII emphasizes the effectiveness of the approach adopted in this work in capturing traveling discontinuities as well as smooth features, and demonstrates this effectiveness for specific problems that combine features with widely differing length scales, and with computations which demand simultaneous resolution of sharp as well as smooth features.

6.5.1 Overview of the Solution-Adaptation Methodology

The solution-adaptation algorithm developed and adopted in this work operates by performing two successive traversals of the Quadtree-based grid. In the first traversal, certain solution variables are computed and stored for every cell in the grid. These solution variables are here chosen to be the magnitudes of the following four quantities: (i) the divergence of the velocity vector; (ii) the curl of the velocity vector; (iii) the gradient of the density; and, (iv) the gradient of the entropy production. As explained in more detail below, these four variables may be called the adaptation indicators, and were chosen because together they capture all of the discontinuities encountered in The System of Euler Equations, as well as the most relevant features in smooth regions of that system. In the second traversal, the values of each of the adaptation indicators in every cell in the grid are examined to determine whether the cell should be locally refined, locally coarsened, or left unchanged. If refinement or coarsening are found necessary, they are performed immediately after the determination is made, before the traversal continues to the next cell, just as done with geometric-adaptation. Refinement is applied if any of the adaptation indicators indicates that a cell is too coarse; coarsening is applied only if all the adaptation indicators indicate that a cell is too fine. As explained in more detail below, the second traversal effectively computes the grid-optimization criteria, and invokes the necessary refinement or coarsening operators.

The reason why the adaptation indicators are computed in a single traversal before the second single traversal which evaluates the grid-optimization criteria and executes the refinement and coarsening operations is that, as explained in Sub-Section

6.5.3, evaluation of the grid-optimization criteria chosen in this work requires knowledge of the statistical distributions of the values of the adaptation indicators, or at least knowledge of the minimum and maximum values of these indicators, for all the cells in the grid. Just as is the case for the geometric-adaptation algorithm, the refinement operations and the coarsening operations performed during the second traversal of the solution-adaptation algorithm could be separated so that they are performed during two individual traversals instead of during the same traversal.

The actions and properties of the solution-adaptation algorithm are very similar for steady-state computations and for transient computations. The major difference is that the solution-adaptation algorithm must be applied after every time-step in the latter case, but need only be applied once every several update cycles in the former case. Indeed, for steady-state computations, the adaptation need only be invoked once the solution has sufficiently converged on the unadapted grid (or on the grid that was formed from a previous adaptation cycle).

The local refinement and local coarsening operations for solution adaptation are carried out in the generic, unified, standard manner, as explained in Sections 6.2 and 6.3, just as they are carried out for geometric adaptation, and just as they are carried out during the generation of the initial grid.

6.5.2 Adaptation Indicators

This sub-section describes the adaptation indicators used in this work, and explains the reasons for choosing them in preference to the most relevant other alternatives. The relevant background for adaptation indicators, and the most important issues related to them are reviewed in Section 4.5. In particular, Section 4.5 defines adaptation indicators, explains the role of adaptation indicators in the solution-adaptation process, discusses the ideal properties of adaptation indicators, and reviews the various types of adaptation indicators that are in use for computational solution of hydrodynamic problems. The section also explains how the close relationship arises between adaptation indicators and the solution error, discusses the difficulty of deriving or calculating the solution error for problems which involve a complex, coupled system of governing equations, such as The System of Euler Equations, and outlines why the gradients of the solution variables are frequently used as measures of the solution error in computational solution of hydrodynamic problems.

As explained in Sub-Section 6.5.3 below, both of the two grid-optimization strategies adopted in this work attempt to track and resolve all the discontinuities that are present in a solution, and to use variable refinement to resolve the smooth regions in the solution. Therefore, the adaptation indicators chosen in this work are required to reliably detect not only all the possible types of solution discontinuities, but also to detect the variations associated with solution errors in smooth regions. The importance of not ignoring the resolution of the smooth regions in flows which contain discontinuities has been repeatedly pointed out [229, 402] in studies of solution adaptation. In one demonstration of the danger of under-resolving the smooth region [402], it was shown that an inaccurate solution in the smooth region significantly affected the overall solution, including the location of a strong shock, and it was concluded that in solution-adaptive refinement, it is necessary to ensure that increasing the number of cells will result in increasing the resolution throughout the Computational Region, not just around the discontinuities.

In concert with the grid-optimization strategy adopted and developed in this work, which is outlined above and described in detail in Sub-Section 6.5.3, all the adaptation indicators adopted in this work are based on measures of the magnitude of gradients of the computational-solution variables. Specifically, the adaptation indicators used in this work, where, as before, ρ is the density, $\vec{v}$ is the velocity vector, and c is the sound-speed, are as follows:

1. **Indicator 1**: $|\nabla \cdot \vec{v}|$;
2. **Indicator 2**: $|\nabla \times \vec{v}|$;
3. **Indicator 3**: $|\nabla \rho|$; and,
4. **Indicator 4**: $|\nabla p - c^2 \nabla \rho|$.

These four specific adaptation indicators were chosen in this work because the model system of equations studied in this work, The System of Euler Equations, which is described in detail in Chapter II, admits three types of discontinuities: (i) shock waves; (ii) shear waves; and, (iii) contact waves. The first adaptation indicator, $|\nabla \cdot \vec{v}|$, is an excellent detector of shock waves. The second adaptation indicator, $|\nabla \times \vec{v}|$, is an excellent detector of shear waves (and oblique shock waves). The third adaptation indicator, $|\nabla \rho|$, is an excellent detector of contact waves. The manner in which each of the first three adaptation indicators is ideally-suited for its target discontinuity is explained in detail in Chapter II, and especially in Section 2.2 which reviews the relevant definitions and the jump relations and properties across the three types of discontinuity admitted in The System of Euler Equations.

In addition to the three types of fundamental discontinuity that are detected by the first three adaptation indicators, all of the four adaptation indicators listed above detect the occurrence of important phenomena in smooth regions. In particular, the first and third adaptation indicators detect expansions and smooth compressions, while the second adaptation indicator detects vortical structures and interactions. The first, second, and third adaptation indicators therefore have an important role to play in adapting smooth regions since adequate resolution of expansions, smooth compressions, and vortical structures is often necessary to achieve the desired global accuracy in computational solutions for problems with compressible flow, especially if accurate predictions of lift and drag forces are required. The fourth adaptation indicator used in this work, $|\nabla p - c^2 \nabla \rho|$, measures the magnitude of the gradient of the entropy. However, as explained in Chapter II, the entropy in smooth regions is preserved in The System of Euler Equations. Therefore, this indicator effectively measures the variation (and specifically, the generation) of entropy due to numerical dissipation, and therefore provides a measure in smooth regions of the solution "quality" and of the level of refinement required to adequately resolve the flow features there [271].

As implied in the preceding two paragraphs, the first three of the four adaptation indicators used in this work can individually be chosen from physical reasoning, especially one founded on an understanding of the general solution properties of The System of Euler Equations, and of the jump relations that characterize the three types of discontinuities in that system of equations. In addition, the first three of these adaptation indicators can be derived from analysis of the strengths and the speeds of the three different possible types of propagating waves in solutions of The System of Euler Equations [271]. The analysis not only leads to the first three individual adaptation indicators listed above, but also to the specific combination of these three indicators. The importance of this is that it shows that the first three adaptation indicators taken in combination are a basis for a comprehensive set of indicators for measuring or characterizing all the possible smooth and discontinuous flow features of The System of Euler Equations which are related to the hyperbolic character of this system of equations.

As implied in the preceding paragraph, in addition to choosing the individual adaptation indicators for a given solution-adaptive methodology, it is also necessary to choose the procedure by which the values of these individual adaptation indicators are weighted and combined into a single value, or into a single criterion, which is then used to drive the solution-adaptation process. The specific formulation and implementation of the weighting and combining procedures used in this work is explained in detail in Sub-Section 6.5.3 below.

The four adaptation indicators adopted in this work, and especially the first three of them, have shown their effectiveness in practice over a wide range of applications of The System of Euler Equations [102, 103, 74, 297]. In order to use them to full effect, however, it is still necessary to assign appropriate problem-dependent weightings to the individual adaptation indicators in order to combine them appropriately within a given grid-optimization criterion, as mentioned in the preceding paragraph, and as explained in detail in Sub-Section 6.5.3 below. It may also be necessary to assign location-dependent weightings if there are features of different strength in different parts of the Computational Region.

The use of adaptation indicators that are derived from an understanding of the physical properties of The System of Euler Equations, especially ones that are aimed at detection of discontinuities in the solution, is common. In addition to the four adaptation indicators chosen in this work, the following four have also often been used for The System of Euler Equations: (i) $|\nabla p|$; (ii) $|\vec{v} \cdot \nabla \vec{v}|$; (iii) $|\nabla s|$; and, (iv) $l_c^2 \left(\left| \frac{\partial^2 \rho}{\partial x^2} \right| + \left| \frac{\partial^2 \rho}{\partial x \partial y} \right| + \left| \frac{\partial^2 \rho}{\partial y^2} \right| \right)$ [277], where l_c is the local length scale of the computational cell, and all other terms in these four expressions have their usual meanings. The first of these indicators detects shocks, the second detects shears, the third detects entropy variations, which may either be generated numerically in smooth regions or physically and numerically across discontinuities, and the fourth detects shocks and contacts, as well as smooth features such as expansions. These adaptation indicators may either be used individually or in combination.

An important limitation of the individual adaptation indicators that can be derived from physical reasoning, including all the adaptation indicators adopted in this work, and the four adaptation indicators described in the preceding paragraph, is that none of them on its own is capable of simultaneously identifying all the discontinuities or solution features that are present in a general solution for The System of Euler Equations. As explained in Section 2.2, for example, there are no jumps in pressure or density across pure shear layers, and there are no jumps in pressure or velocity across pure contact discontinuities. The velocity does jumps across shock waves and shear waves, but the undivided differences in the velocity scale very differently across these two types of discontinuity [101]. Thus, the common practice of using only one or only a few of the single adaptation indicators described above in an adaptive scheme is highly limiting and undesirable, and leads to a very poor adaptive scheme for the generic problem, as demonstrated, for example in [101, 271]. Because of the difficulty of devising a single adaptation indicator that measures all the relevant solution and discontinuity features for The System of Euler Equations [101, 271], the idea of using several indicators in combination is frequently used in fully-automated adaptive schemes. Each of the individual adaptation indicators used in such a combination is specialized in detecting a certain discontinuity or flow feature, so that the combination is capable of capturing all the possible features [101, 271] in the system of equations being solved. Combining adaptation indicators in this manner is the most appealing and general approach for a fully-automated adaptive scheme, and is the one adopted in this work.

Another category of adaptation indicators commonly used for The System of Euler Equations [101] is based on the evaluation of undivided differences in the discrete solution. These adaptation indicators include the following (where the operator δ refers to the undivided difference in a quantity across a cell face or between neighboring cells): (i) $|\delta\rho|$; (ii) $|\delta p|$; and, (iii) $|\delta q|$ [98], where $q = \sqrt{\vec{v} \cdot \vec{v}}$. The types of discontinuities that are detected by each of these indicators can be deduced from the characteristic jumps across each of the three types of discontinuities supported by The System of Euler Equations, as described above and in Chapter II, especially in Section 2.2.

An important limitation of the adaptation indicators that are based on undivided differences is that they have a directional dependence, which may be easily overcome to a large extent by taking the cell-assigned value to be the average or the maximum of the undivided differences across all the cell faces.

Another, relatively new, category of adaptation indicators is based on the idea of using the values of the second derivatives of the wave strengths in the discrete solution [73]. These wave strengths are computed directly from the Approximate Riemann Solver (defined and described in Chapter III) used in the solution algorithm. The new approach [73] also uses the wave speeds and the propagation directions to predict the regions that would require refinement ahead of the propagating waves.

The major shortcoming of the new wave-strength-based approach is that the wave strengths must be projected onto cell-face-aligned directions, leading to incorrect identification of the true wave strengths and propagation directions, and to unnecessary invocation of refinement and coarsening operations, as explained in [73]. This shortcoming, however, should be significantly alleviated by use of a grid-independent wave decomposition, as explained further in [73, 271]. This effect can also be largely eliminated [73] by examining the value of the local residual in addition to the value of the wave strengths: if the residual is small, then no refinement or coarsening is applied, even if the waves are strong. This pure wave-strength-based approach was not adopted in this work because of its high sensitivity to noise, especially behind shock waves and around shear waves, because of its requirement for extra storage in order to implement it in a modular manner, and because it does not appear to offer any practical advantages over the adaptation indicators based on the divergence and the curl of the velocity vectors used in this work, which have already been demonstrated [271] to be related to the wave strengths.

A comparison of the strengths and weaknesses of several of the adaptation indicators described in this sub-section is presented in References [101] and [271].

An important consideration in choosing adaptation indicators for The System of Euler Equations is that the discontinuities and the flow features supported by this system are the same for steady-state and transient flows. Therefore, the same set of adaptation indicators can be expected to work equally well for computations of both types of flows, even if there are also moving boundaries. One of the important findings of this work is the practical confirmation of this expected behavior for the class of compressible, inviscid, ideal-gas flows. For example, the adaptation indicators can have the same settings and weightings for a problem in which a shock collides with a stationary boundary, as for a problem obtained from the preceding problem by a Galilean Transformation that renders the shock stationary and drives the boundary into the shock. More generally, the optimal settings are almost identical for transient flows and stationary flows having similar flow-features and similar extreme values of the solution.

6.5.3 Grid-Optimization Criteria

This sub-section describes the grid-optimization criteria used in this work. The relevant background for grid-optimization criteria, and the most important issues related to them are reviewed in Section 4.5. In particular, Section 4.5 defines grid-optimization criteria, explains the role of grid-optimization criteria in the solution-adaptation process, and outlines the different approaches to devising grid-optimization criteria, including the approach of the "error equidistribution principle" (which is also defined and explained in Section 4.5). Section 4.5 also describes how the difficulty of formulating accurate estimates of the solution error (and hence of formulating adaptation indicators based accurately on the solution error), causes a reversion to ad hoc and approximate grid-optimization criteria, including reversion to simplified forms of the "equidistribution principle". In particular, the section describes how many grid-optimization criteria in common practice employ an assumed functional relationship between the desired length-scale of a computational cell, and the magnitude of the local first- or higher-gradients of the solution in that computational cell. Section 4.5 also presents several of the ad hoc grid-optimization criteria that are commonly used in h-adaptation.

Two grid-optimization criteria were implemented and explored in this work, and both of them fall within the category of ad hoc, elementary, approximate approaches described in the preceding paragraph and in Section 4.5. In particular, in both of the grid-optimization criteria explored in this work, the relation between the value of the adaptation indicator and the optimal local length-scale required in the Computational Region is assumed to be of the form expressed in Equation 4.2. As explained in Section 4.5, this is equivalent to assuming that the error is proportional to the solution gradient(s), and that equidistributing the (non-directionally-aligned) undivided differences in the solution values between all the cells (by adjusting the length-scales of those cells by h-refinement or h-coarsening) equidistributes the solution error across the grid. Since the numerical scheme used in this work is formally second-order-accurate in space (degenerating to a first-order-accurate scheme in regions of non-uniform refinement, which are typically the regions in which the solution adaptation is most intense), this treatment is justified in the smooth regions, but not across discontinuities. In an attempt to overcome this deficiency, the second grid-optimization criterion used in this work attempts to treat discontinuities separately, as described in detail below.

The first grid-optimization criterion implemented and explored in this work is a variant of the first of the "ad hoc" optimization criteria for h-adaptation described in Section 4.5. As described in Section 4.5, this method determines whether individual cells will be refined, coarsened, or left unmodified depending on where the values of their adaptation indicators fall within the statistical distributions of the values of the adaptation indicators for all the cells in the grid [189]. Specifically, Equation 4.3 expresses the test that must be satisfied for refinement, and Equation 4.4 expresses the test that must be satisfied for coarsening, as described in more detail in Section 4.5. The latter two equations are respectively repeated here for convenience, as follows:

$$|\epsilon_{\phi c}| \geq \mu_\phi + k^r_\phi \sigma_\phi \tag{6.7}$$

and

$$|\epsilon_{\phi c}| < \mu_\phi - k^c_\phi \sigma_\phi \tag{6.8}$$

where all terms are as defined in Section 4.5.

The values of k_ϕ^r and k_ϕ^c are assigned in this work in accordance with the relevant governing principles discussed in Section 4.5 (and in the original references given there), using respectively relations of the form $k_\phi^r = \frac{\alpha}{l_c^\beta}$ and $k_\phi^c = \frac{\tilde{\alpha}}{l_c^{\tilde{\beta}}}$, where l_c is the cell dimension, and where typical values for the numerical parameters α, $\tilde{\alpha}$, β, and $\tilde{\beta}$ are respectively chosen in this work to be 5.0, 0.2, 1.5, and 1.5. This arrangement provides two parameters to control each of k_ϕ^r and k_ϕ^c, for a total of four parameters for each adaptation indicator ϕ. The total number of parameters controlling the overall grid-optimization scheme for a specific computation may be reduced from the maximum possible, which is here 16, by choosing the four parameters α, $\tilde{\alpha}$, β, and $\tilde{\beta}$ to have uniform settings for all the adaptation indicators used (that is, by choosing the parameters independently of ϕ). With this unification, the entire grid-optimization methodology adopted in this work can be "tuned" with a total of four parameters.

Although it is possible to effectively capture all the important features of highly complex flows with the grid-optimization technique described above, the use of only four parameters for each of the adaptation indicators was found to present difficulties in controlling the form of the functional dependence of the refinement distribution on the values of the adaptation indicators. As a result, it was often necessary (especially for computations of transient problems) to restart calculations from the beginning several times, after repeatedly adjusting the four parameters to alter the manner in which the cell sizes cluster around the various features of the solution.

Another detraction of the grid-optimization method described above was the difficulty it presented in controlling the maximum refinement level, leading to difficulties in controlling the total number of cells, and the total calculation time. These effects and difficulties were more pronounced in computations in which a large number (of discontinuous and smooth) flow features of differing strengths were created during the course of the computation, such as in most of the shock diffraction examples presented in Section 8.5. In order to ensure that none of the relevant flow features were lost through insufficient spatial resolution, it was found necessary to decrease the value of α (from the default setting of 5 given above to, say, 2), or to decrease the value of β (from the default setting of 1.5 given above to, say, 1.2), thereby increasing the number of cells by as much as a factor of 5 or more compared to the second grid-optimization criterion explored in this work, with little improvement in the solution quality.

Except for problems of the type just described, the technique expressed in Equations 6.7 and 6.8 was found to be highly effective, especially for steady state flows. This conclusion corroborates the findings reported in various works that have used this technique [189, 101, 271] in similar solution-adaptive settings.

The second grid-optimization criterion implemented and explored in this work can be considered a variant of the second of the grid-optimization methods described in Section 4.5, as expressed in Equation 4.5 for the case of refinement. The specific implementation adopted in this work differs from the standard one described in Section 4.5 in two major ways.

The first of the two differences described in the preceding paragraph is that the criterion implemented in this work reserves the highest level of refinement available in the grid exclusively to the discontinuities present in the solution, regardless of the type or strength of these discontinuities. The remaining available refinement levels, starting from the uniform, background refinement level, all the way to the highest refinement level but the last, are used to resolve the “smooth” regions of the solution, in accordance with the “error equidistribution principle” described and discussed in

Section 4.5. Accomplishing this resolution distribution requires the overall solution-adaptation technique to track all discontinuities in the solution, and to capture them within swathes of cells that are at the highest allowed refinement level. Examples of this tracking and "exclusive" resolution of the discontinuities in the solution are shown extensively throughout Chapter VIII. In particular, all the transient and moving-boundary computations presented in Chapter VIII are obtained using the second grid-optimization criterion explored in this work.

The rationale for adopting the approach of exclusively resolving the discontinuities with the highest refinement level is motivated by the shortcomings and limitations of the "error equidistribution principle", as discussed, for example, in Section 4.5. In particular, since the principle breaks down in non-smooth regions, the aim of this grid-optimization criterion is to confine the application of the principle to its regions of validity (that is, to the regions of smooth solution), and to assume that solution discontinuities (which are always of infinitesimal thickness in The System of Euler Equations) should be separately resolved, with the finest length-scale available. Additional discussions of this issue are presented towards the end of this sub-section.

The second of the two differences between the grid-optimization criterion adopted in this work and the standard variant of it described in Section 4.5 is that the criterion κ_ϕ in Equation 4.5 is formulated to no longer be a constant, but instead to be a function of the adaptation indicator it is related to. The benefits of this generalization are discussed in more detail below.

The specific implementation of the second grid-optimization criterion adopted and developed in this work can be expressed in the form

$$\frac{|\epsilon_{\phi c}| - \epsilon_{\phi min}}{\epsilon_{\phi max} - \epsilon_{\phi min}} > \kappa_r(\phi) \tag{6.9}$$

for the refinement criterion, and in the form

$$\frac{|\epsilon_{\phi c}| - \epsilon_{\phi min}}{\epsilon_{\phi max} - \epsilon_{\phi min}} < \kappa_c(\phi) \tag{6.10}$$

for the coarsening criterion, where $\epsilon_{\phi c}$ is the value of the adaptation indicator for variable ϕ for cell c, $\epsilon_{\phi min}$ and $\epsilon_{\phi max}$ are respectively the minimum and maximum values of the adaptation indicator for variable ϕ over all the cells in the grid, and $\kappa_r(\phi)$ and $\kappa_c(\phi)$ are respectively the refinement- and the coarsening-optimization "criteria values" for variable ϕ. As explained below, it is often advantageous to replace the term $\epsilon_{\phi min}$ in Equations 6.9 and 6.10 by a (possibly user-specified) threshold value, which is slightly greater than $\epsilon_{\phi min}$.

As noted above, the manner in which the formulation described in Equations 6.9 and 6.10 above differs from the standard formulation of this grid-optimization criterion is that the criteria $\kappa_r(\phi)$ and $\kappa_c(\phi)$ are functions of the value of the adaptation indicator variable ϕ instead of being constants. The manner in which the grid-optimization criteria of Equations 6.9 and 6.10 are used to effect the grid optimization in this work is explained in more detail in Figure 6.2 and the next few paragraphs.

Figure 6.2 depicts a typical composite grid-optimization criterion, using the combination of a single refinement-criterion function, $\kappa_r(\phi)$, and a single coarsening-criterion function, $\kappa_c(\phi)$, as described in the preceding paragraph. The figure shows the two criteria (which are hereafter respectively also called the refinement and coarsening functions) separately. The specific adaptation indicator I in the figure may be any one of the four indicators used in this work, for example, the indicator $|\nabla \rho|$, and the coarsening and refinement functions shown must obviously be for the same indicator. As implied in the notation chosen for the functions $\kappa_r(\phi)$ and $\kappa_c(\phi)$, each adaptation indicator in this work has its individual refinement and coarsening functions.

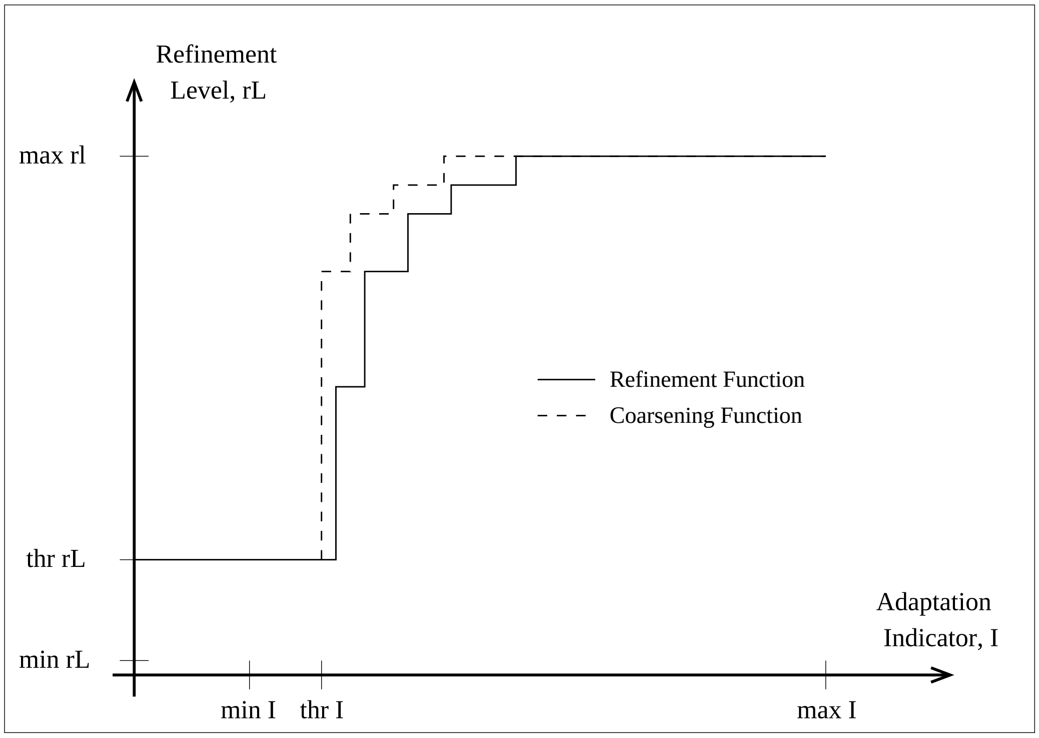

Figure 6.2: A graph of the refinement-level vs. the value of the adaptation indicator, showing the refinement and coarsening functions, and indicating the action of the grid-optimization criterion. The terms "min", "max", and "thr" represents respectively the minimum, maximum, and threshold values for the adaptation indicator over all the cells in the grid.

As described above, the refinement criterion will seek to refine any cell whose adaptation indicator indicates that it is under-refined, that is, any cell for which the criterion of Equation 6.9 fails. Thus, the refinement function will seek to ensure that no cell falls below its graph (that is, that no cell falls below the graph of Refinement-Level vs. Adaptation Indicator Value). On the other hand, the coarsening function will seek to coarsen any cell whose adaptation indicator indicates that it is over-refined, that is, any cell for which the criterion of Equation 6.10 fails. Thus, the coarsening function will seek to ensure that no cell falls above its graph (that is, that no cell falls above the graph of Refinement-Level vs. Adaptation Indicator Value). This holds true even though, as explained in Sub-Section 6.5.4 and elsewhere, the manner in which the refinement and coarsening criteria cause adaptation of the grid is different: refinement is effected if any refinement criterion indicates that a cell requires refinement; coarsening is effected only if all the coarsening criteria indicate that a cell requires coarsening.

Several properties of the refinement and coarsening functions can be inferred from the discussion in the preceding paragraph. It can be concluded, for example, the refinement and coarsening functions, $\kappa_r(\phi)$, and $\kappa_c(\phi)$, respectively, must satisfy the condition $\kappa_r(\phi) < \kappa_c(\phi)$ everywhere. It can also be concluded from the above that the actions of the refinement and coarsening criteria together seek to place all the cells that are subject to solution adaptation in the region enclosed between the refinement and coarsening functions. This region may therefore be called the "optimal" region (as established by the chosen grid-optimization criterion). Figure 6.2 also illustrates the main constraints on the forms of, and the relationships between the refinement and coarsening functions, and also shows how these functions behave near the smallest and greatest values of the adaptation indicator values.

The approach of using refinement and coarsening functions separately provides a very high degree of control over the grid-optimization process. It not only enables the maximum and the minimum refinement levels in the grid to be fixed a-priori, but also enables direct control over the distribution of refinement levels in between the highest and lowest levels, unlike the situation with the first grid-optimization criterion explored in this work.

The thresholds $thrrL$ and $thrI$ described in Figure 6.2 are used to provide additional control over the solution-adaptation process. The refinement-level threshold,

$thrrL$, is the refinement level below which no solution-adaptation is applied (or evaluated). The main role of this threshold is to ensure that the solution-adaptation algorithm does not attempt to coarsen any cell so that its refinement-level drops below $thrrL$ (as the algorithm may attempt to do for cells in regions of uniform flow, for example), and this is done by checking the refinement level of a cell before attempting to coarsen it. The value of $thrrL$ is simply set to that of the uniform-refinement-level (which is the background level below which the refinement-level of no cell may fall, as explained in Section 6.3, and this value, which is chosen to suit the problem and the specific requirements of the computation, is typically 3-6).

The adaptation-indicator threshold, $thrI$, is the value of the adaptation indicator below which the coarsening function will not attempt to coarsen a cell (and hence below which the refinement function will also not attempt to refine a cell, since the refinement function must always fall below the coarsening function). The main role of this threshold is to counter the activation of the solution-adaptation algorithm by the numerical "noise" in a calculation, and this role is found to be especially useful for very smooth flows or flows with regions of uniform-state. The specific value of $thrI$ is chosen in a manner that is highly dependent on the flow features anticipated in a computation. For flows with discontinuities, the value is typically set to the minimum value of the adaptation indicator, say, zero. Figure 6.2 clearly shows the settings for the thresholds $thrrL$ and $thrI$, and how these settings partly determine the form of the refinement and coarsening functions.

The refinement-criterion and coarsening-criterion functions for each adaptation indicator are specified in this work in discrete form using one "real-valued" array for each, giving a total of eight arrays for the entire solution-adaptation algorithm. The index in each array corresponds to the refinement-level (of a cell), and the value stored for that index corresponds to the limiting value of the normalized adaptation indicator for that refinement level. The limiting value for the refinement-function arrays is the maximum value of the normalized adaptation indicator for that refinement level. That is, the i^{th} entry in the refinement-function array is the maximum value of the quantity $\frac{\epsilon_{\phi c}-\epsilon_{\phi thr}}{\epsilon_{\phi max}-\epsilon_{\phi thr}}$ for refinement level i. The limiting value for the coarsening-function arrays is the minimum value of the normalized adaptation indicator for that refinement level. That is, the i^{th} entry in the coarsening-function array is the minimum value of the quantity $\frac{\epsilon_{\phi c}-\epsilon_{\phi thr}}{\epsilon_{\phi max}-\epsilon_{\phi thr}}$ for refinement level i. The organization of data in these arrays implies that the refinement- and coarsening-functions are encoded in inverse form, as mappings from the integer-valued refinement level to the real-valued limits of the normalized adaptation-indicator value. The only reason for adopting this inverse form is its greater convenience and the reduced operation count it allows.

The arrays described above are established and initialized at the start of the computation, entirely from input supplied by the user, who must therefore have some knowledge of the problem being computed, of the flow-features that are likely to develop during the evolution of the solution, and of the relative strengths of these features.

The refinement and coarsening solution-adaptation traversals of the second grid-optimization method adopted in this work again follow the standard procedure. During a refinement traversal of the solution-adaptation algorithm, the normalized adaptation indicator values are determined from the four indicator values for each cell visited. If for *any* of the adaptation indicators, the normalized value is found to exceed the limiting value stored for that refinement level, the cell is refined. Similarly, during a coarsening traversal of the solution-adaptation algorithm, the normalized adaptation indicator values are determined from the four indicator values for each cell. If for *all* of the adaptation indicators, the normalized value is found to fall below the limiting value stored for that refinement level, the super-node of the cell is coarsened (and only if the geometric-adaptation criteria allow).

Although a single refinement or coarsening may not bring about the optimal refinement level at a certain location, the refinement and coarsening functions will always change the refinement level in the correct way, so that repeated applications of refinement or coarsening will achieve the optimal refinement level. As explained in several other places in this chapter, compared to refinement operations, coarsening operations must always satisfy an additional set of constraints or tests before they are executed. In particular, before any coarsening is applied, a check is made on the super-node of the target node to ensure that the result of the coarsening will not create a cell that will require refinement on the next refinement cycle, and a check is also made on the state of the other three sub-nodes of the latter super-node to ensure that all four of the sub-nodes are over-refined. This checking is identical to that carried out before coarsening in the first grid-optimization criterion explored in this work. If the conditions given above are not satisfied, then the coarsening is not executed. On the other hand, any cell identified by the adaptation criterion to be under-refined is individually and immediately refined, without regard to the status of its resulting sub-nodes.

As implied in the preceding paragraph, a slight bias is purposely built into the refinement and coarsening procedures in the solution-adaptation algorithm to inhibit coarsening, by resolving conflicts between refinement and coarsening in favor of refinement. The purpose of this is simply to prevent unnecessary cycles of successive coarsening and refinement. These cycles not only waste computational resources, but also unnecessarily add dissipation to the solution.

In order for the refinement and coarsening functions to give a suitable overall grid-optimization criterion, the coarsening function must clearly lie at least one refinement level above the refinement function, and the vertical gap between the two functions must be appropriate for the number of refinement levels in the calculation. That is, the vertical thickness of the optimal refinement-level distribution zone must be appropriately chosen. Otherwise, the two functions will be countering and reversing the actions of each other, causing the two types of loss described in the preceding paragraph (namely, a reduction in the computational efficiency, and also the introduction of additional dissipation in the solution). Typically, the coarsening threshold (or limiting value) is set to about 70-90% of the corresponding refinement threshold, as indicated in Figure 6.2, but this ratio depends strongly on the number of refinement levels in the grid, and the type of features in the flowfield being computed.

The horizontal top asymptote of the refinement function is what "captures" the sharpest or strongest features in the flowfield; namely, the discontinuities. Typically, for a flow with discontinuities, this asymptote spans at least 50% of the range of the normalized adaptation indicator values (that is, of the quantity $\epsilon_{\phi max} - \epsilon_{\phi thr}$). This relatively high ratio is chosen to ensure that the grid-optimization criterion will be able to isolate all discontinuities and resolve them with the highest level of resolution, as described above. Unfortunately, unless the user can adequately estimate the strengths of the various features in a computation, it is possible to make this ratio too high (causing some smooth regions to be resolved with the highest available level of resolution), or too low (causing some of the discontinuities to be resolved with a refinement level that falls below the highest available one).

Despite the high degree of arbitrariness and initial involvement of the user in the second grid-optimization procedure adopted in this work, and despite the shortcoming described in the preceding paragraph, the technique was found to be extremely effective for transient flows partly because it enables the user to selectively focus the adaptation and tracking on specific desired solution features, such as shear waves or vortical structures, for example, and to emphasize or de-emphasize adaptation to the discontinuous or the smooth features in a solution. The most important advantage of this technique, however, is that it allows complete control over the highest level of refinement, which is chosen a-priori. This not only allows the user to choose the overall resolution level of a computation, but also provides an extremely effective means of controlling the total memory requirements, and the total computational effort devoted to a computation. This is because increasing the maximum refinement level by 1 typically increases the total memory requirement by a factor of 4, and typically increases the computational effort requirement by a factor of 8. An additional benefit of having explicit control over the maximum cell size (in addition to the explicit control over the minimum cell size already provided in the grid-generation algorithm) is that it enables the user to limit the phase error, which for computations of transient problems depends strongly on the ratio of the smallest to the largest cells in the grid.

For a given maximum refinement level, it was found that the final refinement pattern, the final results, and the total use of computational resources is generally insensitive to the exact settings of the relevant thresholds and parameters in the second grid-optimization method adopted in this work, provided the chosen values fall within appropriate ranges.

It may be noted that the second grid-optimization criterion adopted in this work does not explicitly incorporate a factor of the form l_c^α, with $\alpha > 1$, in the adaptation indicator or the grid-optimization criterion to counter the effect of increasing tendency of refinement of cells as their dimensions decrease [402], as described in more detail in the preceding sub-section. However, it can be seen that the technique of using refinement and coarsening functions enforces the same behavior required to prevent the adaptation from focusing completely on refinement in the neighborhood of the discontinuities in the flow by explicitly specifying the maximum and minimum refinement levels, by fixing the distribution of all the allowed cell sizes, and by enforcing refinement even in regions where the adaptation indicator is low, as shown in Figure 6.2. Thus, the technique of using refinement and coarsening functions together causes the refinement of all the cells in the grid to increase if the maximum refinement level is increased, not just the refinement level of cells around discontinuities.

In devising the form of the refinement and coarsening functions in the second grid-optimization criterion adopted in this work, the principle of equidistribution of the truncation error was used to provide guidance. In particular, given that the local cell length-scale is specified in terms of the refinement level by the relation

$$d = \frac{D}{2^{r-1}}$$

where D is the dimension of the Cartesian Root Square, r is the refinement level of the given cell, and d is the dimension of that local cell, the equidistribution principle, given in Equation 4.2, implies that the desired relation between r and the refinement indicator is given by

$$r = \log_2 \left(\frac{2D}{k} \epsilon_\phi \right) \tag{6.11}$$

where, as explained above, ϵ_ϕ is the value of the adaptation indicator for variable

ϕ, and all other terms are as defined above and as defined for Equation 4.2. The functional form of Equation 6.11 justifies the logarithmic shape of the refinement and coarsening functions appearing in the grid-optimization functions, as shown, for example, in Figure 6.2. This guideline is clearly not relevant across discontinuities, which are identified separately, and which typically have far higher values for the associated adaptation indicator than the smooth regions of the flow.

This sub-section described how the decision is made for each cell in the grid, whether to refine it or not, and for each group of four cells with a common supernode, whether to coarsen them and re-agglomerate them into their supernode or not. The actual operations of coarsening and refinement are executed by the refinement and coarsening operators, as described in the next sub-section.

6.5.4 Execution of the Refinement and Coarsening Adaptations

The preceding sub-section described how the grid-optimization criteria are applied to determine for any computational cell in the grid whether refinement, coarsening, or neither action is desired. The grid-optimization criteria are actually invoked and determined through a tree traversal, in the same way as done for all other generic global functions, as explained in Section 6.2. The refinement and coarsening decisions that result from evaluation of the grid-optimization criteria are executed either during the same traversal as that for determination of the grid-optimization criteria, or in separate traversals, as indicated in Section 6.3, and for the reasons given there.

During a refinement traversal, only those nodes whose cells require refinement are identified, and once such a node is identified, all the necessary refinement actions are immediately performed on that node, before the traversal proceeds to the next node in the traversal order. Similarly, during a coarsening traversal, only those penultimate nodes whose cells require coarsening are identified, and once such a node is identified, all the necessary coarsening actions are immediately performed on that node before the traversal proceeds to the next node in the traversal order. A node is identified as requiring refinement if any of the four adaptation indicators for the corresponding cell indicate that the cell is under-resolved. On the other hand, a node is identified as requiring coarsening only if all four adaptation indicators for the corresponding cell indicate that the cell is over-resolved.

Whether in refinement or in coarsening, the actions of the grid-optimization criteria are limited to the identification of the cells that require refinement or coarsening; all the refinement and coarsening actions are separately performed respectively by the refinement and coarsening operators (which may therefore be collectively called the adaptation operators), as described in detail in Sections 6.2 and 6.3. This separation between the actions of the grid-optimization criteria and the adaptation operators was also described in Section 4.5, and is primarily motivated by the desire for modularity of function, especially because the same adaptation operators are used to apply geometric adaptation, and even to generate the initial grid.

The specific actions performed by the refinement and coarsening operators on the Quadtree data-structure that represents the grid (including those performed on the attribute data) to respectively refine and coarsen cells in the grid are described in detail in Sections 6.2 and 6.3. As explained in these two sections, the refinement and coarsening operations are performed on a node-by-node basis (and hence on a cell-by-cell basis). These two sections, especially the former, also explain how the actions of the refinement and coarsening operators include special constraints and subsidiary operations to automatically and intrinsically ensure the smoothness of the grid that results from the cell-by-cell action of these operators, and to ensure that the compatibility of the grid with the boundaries and with the boundary conditions is maintained.

Section 6.3 also explains how the refinement and coarsening traversals for solution-adaptation are controlled, and how these traversals are invoked with appropriate frequencies during the calculation procedure, depending on the category of problem for which the computation is being carried out. For example, Section 6.3 explains how the refinement and coarsening traversals are carried out with different frequencies and at different stages in the calculation procedure, depending on whether the problem is a steady-state problem, or, for example, a time-dependent problem with moving boundaries. As that section describes, the major differences between solution-adaptation for problems with moving boundaries compared to solution-adaptation for problems with stationary boundaries are manifested in the sequencing and frequency of calls to the solution-adaptation procedure (which evaluates the adaptation indicators, computes the grid-optimization criteria, and calls the refinement and coarsening operators), and not in the specific actions performed during the refinement and coarsening operations. That section and others also emphasize that the procedures performed during a refinement or a coarsening are the same whether the refinement or coarsening is being carried out for grid generation, for geometric-adaptation, or for solution-adaptation.

6.5.5 Requirements of the Flow Solver

All solution algorithms that employ a Finite-Volume, Finite-Element, or Finite-Difference discretization require the connectivity between the computational cells, elements, or nodes in the grid to remain unchanged during an individual solution-update cycle [1]. The distances between nodes, and the geometries of cells or elements may change (and appropriate averages of any varying geometric properties may then be used in the solution algorithm), but the solution-update procedures, which often involve matrix operations, always require a single, fixed inter-dependence or influence-pattern between the solution values across an update cycle in all the individual cells or elements in the grid.

The requirement for fixed connectivity discussed in the preceding paragraph holds even for solution-adaptive methods, including the method developed in this work. Because of this requirement, all changes in the grid other than deformations (which here occur in merged cells in the immediate vicinity of moving boundaries, as explained in detail in Chapter VII, and in Section 3.4) must be applied exclusively "between" individual solution-update cycles, leaving the grid "frozen" during solution-update cycles. Such changes in the grid include all local coarsenings and refinements, including all adaptations, whether solution-related or geometry-related, and all cell mergings or cell un-mergings.

Because the grid connectivity is not allowed to change during an individual solution-update cycle, even in a solution-adaptive method, there are no fundamental intrinsic differences between the solution algorithms for methods that use adaptive grids and those that use invariant grids: in both cases, the solution algorithm performs the update cycle mostly by computing and summing fluxes on a fixed grid. The only additional requirement placed by solution algorithms for methods that use adaptive grids is that provisions must be made for changes in the grid and its connectivity between individual solution-update cycles. Assuming that the grid-generation

[1]As also explained elsewhere in this document, a solution-update cycle refers to either the advancing of the solution by a single timestep (in a computation of a transient problem), or to the advancing of the solution by a single "pseudo timestep" or by a single iteration (in a computation of a steady-state problem).

and grid-adaptation algorithms are distinct from the solution algorithm, as is the case in the method developed in this work, by far the two most important such provisions are the following: (i) a procedure to transfer data from the original grid to the modified grid which is created by the adaptation so that the solution can be carried to and then advanced on the newly-adapted grid over the next update cycle and the following ones; and, (ii) an organization of the data structures so that the necessary changes in connectivity can be carried out with minimal computational effort, relative to the computational effort required by the solution algorithm. As evident from their description, even though these two provisions are required to enable useful functioning of the solution algorithm as part of an adaptive-grid method, and even though they are critical elements for such a method, neither of them is directly related to the solution algorithm itself. The remainder of this sub-section is mostly devoted to further discussion of these two provisions.

The initialization of data for a grid created between solution-update cycles by adaptation of an original grid must be performed correctly and accurately. All data, including solution-related and geometry-related data, for the adapted grid must either be determined anew or must be extracted from the original grid during the grid adaptation process that occurs between solution-update cycles. In the case of the geometry-related data, the data for the adapted grid can be calculated directly and exactly from the geometries of the cells and of the boundaries they intersect, with little need for the corresponding data in the original grid. In the case of the solution-related data, the data for the adapted grid must typically either be obtained by transfers from sets of four subnodes to their corresponding supernodes (during coarsening), or from supernodes to their sets of four subnodes (during refinement). The data transfer procedures used for geometry-related and solution-related data are described in detail in Sub-Section 6.3.3. That sub-section also explains that most of these transfers are highly localized and occur exclusively during the refinement or coarsening of nodes, as an intrinsic part of the refinement or coarsening operations. The sub-section also explains the specific special requirements for the transfer of geometry-related data, especially the requirement of ensuring the satisfaction of The Geometric Conservation Laws for moving boundaries, and for transfer of solution-related data, especially the requirement of ensuring the conservation of the Conserved Variables. In the case of solution-related data, all formulae used to transfer the solution data between nodes during refinement and coarsening are given and explained in detail.

The formulae used to transfer the geometry-related and the solution-related data during cell merging and cell unmerging operations are described in detail in Chapter VII. As mentioned above, all merging and unmerging operations and their related data transfers are carried out only between solution-update cycles, just as done with adaptation operations. In the case of solution-related data, the formulae used to transfer data during merging and unmerging are respectively analogous to those used for transferring solution-related data during coarsening and refinement, as they are given and explained in the Sub-Section 6.3.3. This similarity is to be expected, since merging and unmerging can respectively be viewed as an agglomeration of one or more individual cells (into a composite cell), and as a de-agglomeration of a composite cell (into one or more individual cells). In other words, merging and unmerging can be respectively viewed as a less restricted, non-hierarchical variant of the coarsening and refinement procedures carried out in the Quadtree.

As explained above, the second major requirement for a solution algorithm of a method using adaptive grids is that the grid-related data-structures must be designed and implemented in a way that allows the grid connectivities to be modified between solution-update cycles with minimal computational effort compared to the computational effort required by the solution algorithm. This is especially true for methods that involve dynamic adaptation (that is, automatic, run-time-executed adaptation), as does the method developed in this work, and it is true whether the dynamic adaptation involves merging, unmerging, local refinement, or local coarsening of cells. The overall computational performance of different adaptive methods will largely depend on how easily and economically such connectivity modifications can be carried out.

One of the characterizing features of the Quadtree-based grid data-structure as it is used in this work is that, as emphasized in several sections of Chapter VI, the grid connectivity data is largely confined to and incorporated in the hierarchical connectivity data of the Quadtree. This arrangement implies not only that the grid connectivity data storage is highly localized, but also that the volume of this data is kept almost to the minimum possible. No pre-processed, global data-structures, such as index tables, arrays, or link-lists are used to store connectivities in this work, except for a secondary link-list which is temporarily re-created and used only in the data-output operation. As explained in Chapter VII, the connectivity data for cell merging is also similarly minimal and highly localized. The consequence of these arrangements is that all connectivity modifications and all grid adaptations, including cell refinements and coarsenings, and cell mergings and unmergings are performed by purely local modifications to the data-structure, and therefore consume minimal computational effort. Moreover, because the solution algorithm used in this work is executed as part of a tree traversal (in which the fluxes across each cell face in the grid are computed and accumulated, cell by cell), it requires no input of global connectivities or global variables (such as the total number of cells in the grid, for example), and therefore requires none of the usual input parameters that change whenever a grid changes between two solution-update cycles.

The data-structures developed and used in this work contrast strongly with global, fixed, array-based or table-based data-structures storing global connectivities in pre-set orders. In the latter type of data-structures, modifications to the grid, even by the addition or removal of a single cell, for example, may require the connectivity for the entire grid to be re-constructed anew. Such data-structures are therefore not suitable for methods using dynamic adaptation, especially for adaptive computations of transient problems.

One consequence of having a fixed grid connectivity during each solution-update cycle is that if flow features move by a sufficiently large distance during the cycle, then the grid adaptation at the start of the cycle may become inappropriate, and even deleterious, by the end of the cycle. This may occur to the extent, for example, that sharp features may move far out of their refinement zones and become heavily smeared. This effect is more relevant and critical for computations of transient problems than for computations of steady-state problems, but in both cases, it can be countered by increasing the thickness of the refinement zones, as described, for example, in [73]. The thickness can either be increased uniformly, or along preferred, solution-dependent directions, as demonstrated in [73]. In this work, however, no special treatment is necessary to overcome or circumvent this effect. This is because, as explained in Chapter III, the solution algorithm in this work performs the time-integration using an explicit predictor-corrector formulation, and since the timesteps taken in such an explicit solution algorithm are so small that the fastest waves in the solution barely cross a single cell during a single update cycle, there is never enough time for the grid adaptation to become non-optimal for the solution (given that the solution-adaptation procedure is invoked after every time-step in calculations of transient problems, as explained in Sub-Sections 6.3.5 and 6.5.1). The only way in which this effect can be introduced (temporarily) in the technique developed in this work is if too low a setting is chosen for the frequency of the solution-adaptation cycle (which is explained in Sub-Section 6.3.5) for a computation of a steady-state problem.

6.5.6 Treatment for Moving-Boundary Problems

No special requirements are imposed on the solution-adaptation procedures developed in this work by motion of boundaries, and no special extensions of or treatments in these procedures are applied for computations with moving boundaries. If moving boundaries are present, then the solution adaptation responds only to the physical phenomena and the flow structures created by this motion, without any account being explicitly taken of the boundary motion itself.

The built-in algorithmic indifference to the motion of boundaries described in the preceding paragraph reflects and is fully justified by the fact that the physical phenomena involved, and the structure of the flow patterns established in a problem do not depend on whether the boundaries that are present in the problem are stationary or moving per se, but only on the relative speeds involved. A solution-adaptation algorithm founded on correct physical principles should therefore be able to handle in a uniform manner problems with moving boundaries and problems with stationary boundaries. Moreover, this can be accomplished purely by detecting and tracking the flow features generated in the solution, without regard to the presence of any boundary motion, as done in this work. The invariance of the solution and the solution-adaptation patterns in this work to a Galilean Transformation is demonstrated in Section 8.5 in the calculation results shown for the interaction between a planar shock wave and a cylinder: nearly identical solutions and solution-adaptation patterns are obtained whether the shock travels and collides with the stationary cylinder, or whether the cylinder travels in the opposite direction and collides with the stationary shock.

The uniform treatment described in the preceding paragraph of problems with moving and stationary boundaries by the solution-adaptation algorithm developed in this work extends to every component of this algorithm, including the adaptation indicators, the grid-optimization criteria, and the adaptation operators. These components are therefore valid and uniformly used for all the types of calculations that may be performed with the methodology developed in this work, including calculations with moving boundaries and topologic transformations. The only manner in which the type of computation affects the adaptation process is through the frequency and the ordering of the invocations of the adaptive refinement and coarsening operations, as explained in Sub-Section 6.3.5 and others.

6.6 Fundamental Operations in the Quadtree, II

6.6.1 Neighbor-Determination in the Quadtree

This sub-section describes and explains the neighbor-determination methods that are adopted and developed in this work, and discusses their computational costs, and their main advantages and drawbacks. Although neighbor-determination is required even as part of the Quadtree-based grid-generation process, this sub-section was deferred till this stage because understanding it requires knowledge of the Quadtree data-structure, and of the way in which a Quadtree represents a grid.

Two computational cells A and B are **neighbors** if and only if there is at least one point in the Computational Region that belongs to the boundaries of both of cells A and B. Two cells are **vertex neighbors** if they share a common vertex (but not a common edge). Two cells are **edge neighbors** if they share a common edge (and, for non-degenerate Cartesian cells, this implies that they also share exactly two vertices). The extensions of these two definitions of neighborhood type to **face neighbors** for 3-D grids, and to neighborhood types of even higher dimensionality for grids in higher-dimensional spaces are obvious: in general, for a grid in n-dimensional space, neighboring computational cells may share different types of geometric entities, where each type can be described by its i-volume, where $0 \leq i \leq (n-1)$. Note, however, that for Quadtrees, the terms "edge-neighbor" and "face-neighbor" may (misleadingly) be used synonymously.

The **neighbor-determination operation** can be defined as the operation of identifying the (one or more) cell(s) that share(s) a given face, a given edge, or a given vertex with a given target cell. In this usage, the target cell is the known, given cell for which a set of (one or more) neighbors of given type is sought.

The neighbor-determination operation is used intensively during the flow-solver update, for example, in calculating the flux on a face or edge separating two cells, in determining the stencil to be used for reconstructing the gradient in a cell, and in performing various interpolation and extrapolation calculations. The neighbor-determination operation is also extensively used during the grid-generation process itself, as well as during the input, output, and plotting operations.

The concept of *neighborhood* used in the preceding two paragraphs was in the context of spatial or geographic connectedness or adjacency, and this is the context that is most relevant during actual use of a grid, as indicated in the above examples.

However, this context is only implicitly present in the Quadtree data-structure, and only if the Quadtree is taken to represent a spatial-subdivision process or some other geometric-modeling process. In such cases, the concept of spatial connectedness may also be reflected in the attribute data stored in the Quadtree nodes, but typically still not explicitly in the links connecting these nodes. Indeed, as described in Sections 6.1 and 6.3, the links that connect the nodes in the Quadtree data-structure that represents the grid are typically purely hierarchical: explicitly, they only associate nodes with their super-nodes or their sub-nodes, without regard to geographic proximity between the nodes. Because of this arrangement, the procedure for locating spatial neighbors in a Quadtree that represents a "spatial grid" is based on deduction of the spatial neighborhood between cells in the grid from analysis of the hierarchical connectivity between nodes in the Quadtree. This analysis and deduction are done through local traversal of an appropriate portion of the Quadtree.

The neighbor-determination operation is typically performed only for leaf nodes, since only these nodes are associated with computational cells (between which knowledge of spatial connectedness is required). In this case, the interior nodes in the tree are only used to establish the hierarchical connectivity between the leaf nodes, as will be described below. Nevertheless, the neighbor-determination methods developed in this work are applicable to nodes of any type, including interior nodes.

Two generic neighbor-determination procedures are used in this work: one for locating edge-neighbors, and one for locating vertex-neighbors. As described above, these are the only two types of neighbors that are relevant in a Quadtree-based grid. Despite sharing the same fundamental principles, these two procedures are developed and implemented separately in this work, for reasons that are made evident below. For the most part, these two procedures are here also described separately.

Before describing the neighbor-determination procedures developed and used in this work, however, it is convenient to define several commonly-used terms that relate to the geographic directions relative to a cell, and to the "directions" that can be assigned to the links of a Quadtree.

Let C be a given computational cell. The **D-edge-neighbor** of cell C is the cell that shares the D^{th}-edge of cell C, where D here may be one of either S, W, N, or E, where these symbols signify the South, West, North, and East directions, respectively. The **D-vertex-neighbor** of cell C is the cell that shares the D^{th}-vertex of cell C, where D here may be one of either SE, SW, NW, or NE, where these symbols signify the South-East, South-West, North-West, and the North-East directions, respectively. For vertex-neighbors, the direction may also be signified by the notation D_1D_2, where each of D_1 and D_2 must be either S, W, N, or E, where $D_1 \neq D_2$, and where the ordering of the letters in D_1D_2 is in accord with the standard convention for ordering geographical directions. The directions S, W, N, E, SE, SW, NW, and NE all have their usual geographic meanings (when viewing the grid in its regular alignment with the Cartesian axes, in which the x-axis is taken to point in the E direction, and the y-axis is taken to point in the N direction).

Although, as mentioned above, the links in a Quadtree do not inherently possess geographical directionality, this directionality can be implicitly and uniquely imposed by the relative locations of the computational cells to which the Quadtree sub-nodes correspond. In particular, the four descending links of every super-node can be assumed to be a set of one "SE", one "SW", one "NE", and one "NW" links, such that each of these descending links connects the super-node with one of its four sub-nodes, such that the direction of the link corresponds to the location (within the square corresponding to the super-node) of the square corresponding to the sub-node.

Similarly, every sub-node has a single non-NULL link that ascends to its super-node in either the SE, SW, NW, or NE direction, where this direction is most easily arrived at by identifying the vector connecting the centroid of the sub-node square with the centroid of the super-node square, or by reversing the direction of the link connecting the super-node to that sub-node.

Let D_1D_2 be the direction of a link in the Quadtree (that is, D_1D_2 is one of the directions SE, SW, NW, or NE). The **reflection** of direction D_1D_2 in an axis-aligned edge (or line), say, a $\tilde{D}$-edge, is called an **edge-reflection** or a **line-reflection**, and can be constructed as follows: exactly one of D_1 or D_2 must be aligned along the $\tilde{D}$-edge; leave this parallel component unchanged, and reverse the direction of the other component. The other component here will always be the component of D_1D_2 that is normal to $\tilde{D}$. This is analogous to the reflection of a finite line segment in a line that is parallel to either the x- or the y-axis in plane geometry. For example, the reflection of the direction SW in an S-edge gives the direction NW, while the reflection of the direction SW in an N-edge also gives the direction NW, while the reflection of the direction NE in an E-edge gives the direction NW. Similarly, a **vertex-reflection** may also be defined by construction, as follows: the reflection of direction D_1D_2 in any vertex results in the direction D_2D_1, and the point of origin of the reflected direction is determined in accordance with the corresponding geometric reflection in plane geometry (of the finite line segment corresponding to D_1D_2, in a point corresponding to the location of the chosen vertex). The **reversal** of a given direction has the usual meaning, and can also be obtained by rotating that direction though π radians.

6.6.1.1 Determination of Edge Neighbors

In a Quadtree-based grid (which is always a 2-D grid), an edge neighbor for a given computational cell, cell C, may either be a South neighbor, a West neighbor, a North neighbor, or an East neighbor of the given cell. Let the sought neighborhood direction be D, where D is either S, W, N, or E. The procedure used to determine the edge neighbor in direction D is as follows:

1. Starting in the Quadtree at the node corresponding to cell C, ascend the tree using the node-to-super-node links, one by one, storing (in sequence) the individual ascent directions, until an ascent direction is traversed that has direction D as one of its two direction components. As explained above, the ascent directions possible are SE, SW, NW, and NE, and each of these directions contains two components, either of which may be aligned with D. Store the super-node, node S, arrived at during this last ascent. The traversal of the D direction is called a **cross-over** event, and this event is guaranteed to occur for any ascent, except in one situation: cell C is a cell that lies on the D-boundary of the grid and therefore has no neighbors in the D direction. In this case, the cross-over condition cannot be achieved, and the ascent continues through the root node of the Quadtree to the (NULL) super-node of the root node. When this happens, the desired neighbor of cell C is identified and returned as the NULL (that is, the non-existent) neighbor, and the neighbor-determination procedure is terminated. Otherwise, the algorithm continues through Steps 2 and 3 below.

2. Reflect (as described above) the individual segments of the ascent path sequence traced in Step 1 above in a D-line passing through the node S, then

reverse the direction of each reflected segment, and then invert the sequential order of each reflected, reversed segment. The sequence of links that is created by these reflections and reversals and by the order-inversion defines a descent path that is used to locate the sought edge neighbor. All reflection and reversal operations are efficiently carried out in modulo-4 integer arithmetic.

3. Starting at node S, the node in which the ascent sequence terminated in Step 1 above, follow the descent path obtained by the reflections and reversals and the order inversion of Step 2 above. This descent path will terminate in the Quadtree node that corresponds to the sought neighbor of cell C. The cases where cell C and its sought neighbor are not at the same refinement level, however, require either truncation or extension of the descent path, and this is done as follows: If the descent path cannot be pursued to its end because the path has encountered a leaf node before the path has terminated, then that leaf node is the one corresponding to the sought neighbor of cell C. In this case, the sought neighbor is one refinement level coarser than cell C, and the descent path will be truncated by exactly one level, corresponding to the last segment (or leg) of the path. Both of these outcomes are a result of the one-refinement-level-difference restriction imposed in this work. If the descent path terminates before reaching a leaf node, then the descent path is extended by adding two unique branches, each one level deep. The directions of the two unique branches are obtained by selecting the two unique descent directions that contain components that are opposite to D. These two branches will lead to two unique cells that are the sought neighbors of cell C, and that are one level of refinement finer than cell C. Again, both of these outcomes are consequences of the one-level-refinement-difference restriction imposed in this

work. The two exceptions described above are the only ones that may arise under the one-level-refinement-difference restriction imposed in this work, and, as shown above, both of them are treated in a manner that discloses the unique and complete identity of the sought neighbor(s). The handling described in this step of the two cases of unequal refinement levels between neighbors is illustrated further in two of the examples given below.

Figures 6.3, 6.4, 6.5, and 6.6 illustrate the behavior of the edge-neighbor-determination algorithm described above in four different common scenarios. In all these figures, the arrows represent ascending or descending links, and the starting and ending points of these arrows are always located at the centroids of the uncut cells corresponding to the Quadtree nodes. Thus, the location of the starting and ending points of any arrow defines the Quadtree nodes in which the arrow starts and ends.

In Figure 6.3, the neighbor sought for the target cell is the E neighbor. As always, the first ascent segment is the link that connects the target node with its immediate super-node. This link has the direction NE, and since this contains the component E of the required neighbor direction, the cross-over condition is achieved in the first ascent, and no further ascents are carried out. The ascent sequence is thus given by $\{NE\}$. The elements of this ascent sequence are reflected in an E-edge passing through the centroid of the super-node in which the ascent path terminates, to give the **reflected sequence**, which here evaluates to $\{NW\}$. The components in the reflected sequence are then reversed to give the reversed, reflected sequence. In this case, the reversal of the single segment in the reflected sequence, NW, gives the segment SE. Thus, the reversed, reflected sequence is given by $\{SE\}$. Inverting the order of this sequence (which leaves it unchanged) gives the descent sequence, which therefore here evaluates to $\{SE\}$. Pursuit of the resulting descent sequence from

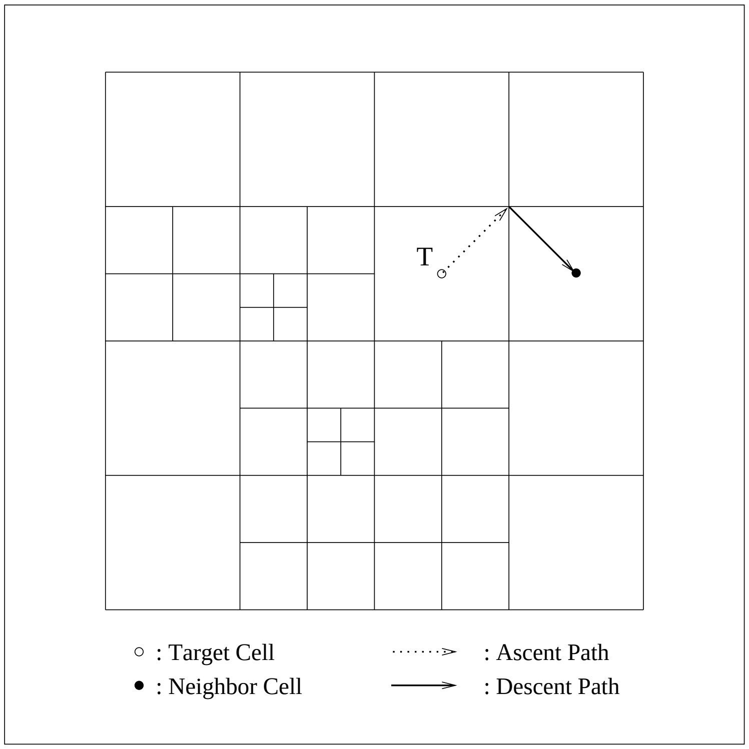

Figure 6.3: An example showing the ascent and descent paths traversed to locate the E edge-neighbor of a specific target cell.

the node in which the ascent path terminates leads to the required neighbor cell, as indicated in the figure.

The reason why the ascent and descent paths in the example of Figure 6.3 are each only one segment long is that both the target cell and the sought neighbor are sub-nodes of the same super-node. This is the most favorable possible situation in terms of path lengths and computational effort requirements. As this example demonstrates, the neighbor determination can be highly localized.

In Figure 6.4, the neighbor sought for the target cell is again the E neighbor. However, in this case the target cell and its sought neighbor are not sub-nodes of the same super-node, and the ascent and descent paths are longer than in the example of Figure 6.3. In this case, the ascent sequence proceeds through four

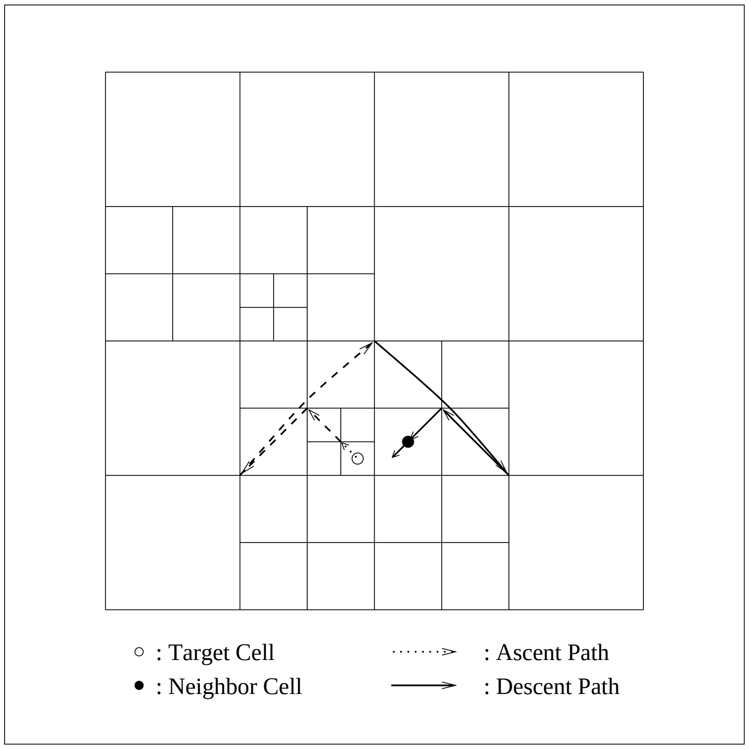

Figure 6.4: An example showing the ascent and descent paths traversed to locate the E edge-neighbor of a specific target cell.

segments, until the E-direction cross-over is attained by the NE ascent in the fourth segment. In this case, the ascent sequence is given by $\{NW, NW, SW, NE\}$, and reaches all the way to the root node of the Quadtree. This happens because the target cell and its sought neighbor are separated by a "1-major" bisector of the root square. Reflection of the ascent path in an E-line passing through the centroid of the root square and reversal of the segment directions results in the reflected, reversed sequence $\{SW, SW, NW, SE\}$. Inverting the order of this sequence gives the sequence $\{SE, NW, SW, SW\}$, which is the descent sequence. In this case, however, the last segment of the descent path cannot be pursued because the path encounters a leaf node, and the last segment is truncated (or ignored). As shown in the figure, the descent path leads to the required neighbor, which is one level of refinement coarser than the target cell.

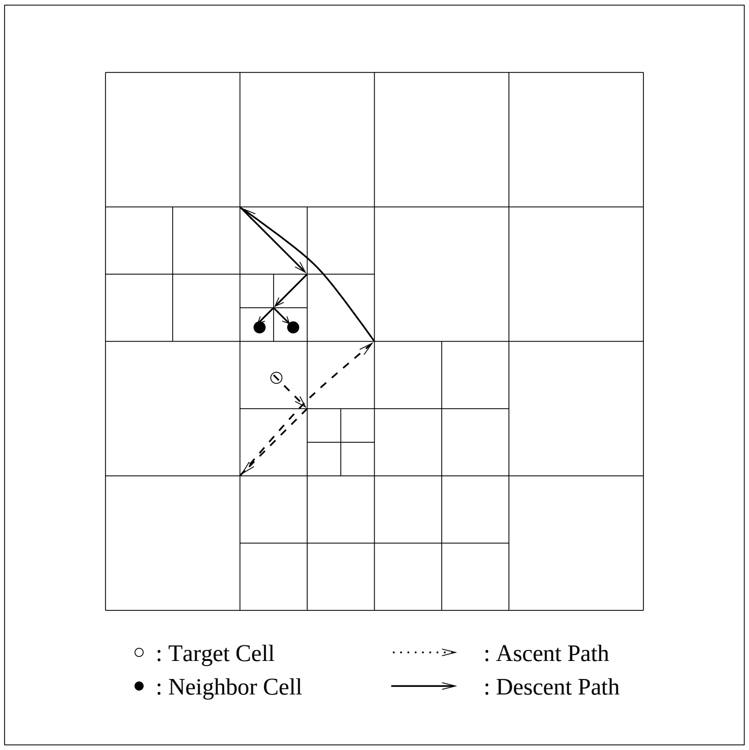

Figure 6.5: An example showing the ascent and descent paths traversed to locate the two N edge-neighbors of a specific target cell.

In Figure 6.5, the neighbor sought for the target cell is the N neighbor. This case illustrates an opposite to the previous example in that the sought neighbor(s) are one level of refinement finer than the target cell. In this case the ascent sequence proceeds through three segments, until the N direction cross-over is attained by the NE ascent in the third segment. In this case, the ascent sequence is given by $\{SE, SW, NE\}$, and reaches all the way to the root node of the Quadtree. Again, this happens because the target cell and its sought neighbors are separated by the a 1-major bisector of the root square. Reflection of the ascent sequence in an N-line passing through the centroid of the root square, followed by reversal of the segment directions results in the sequence $\{SW, SE, NW\}$. Inverting the order of this sequence gives the descent sequence, $\{NW, SE, SW\}$. However, in this case, the last segment of the descent path does not terminate a leaf node. Because of this, the descent path is extended by the two unique branches SE and SW (which are the only two directions that have a component opposite to the N direction). The resulting two branches lead to the two cells that are the N neighbors of the target cell, as shown in the figure, where these two leaf cells are one level of refinement finer than the target cell.

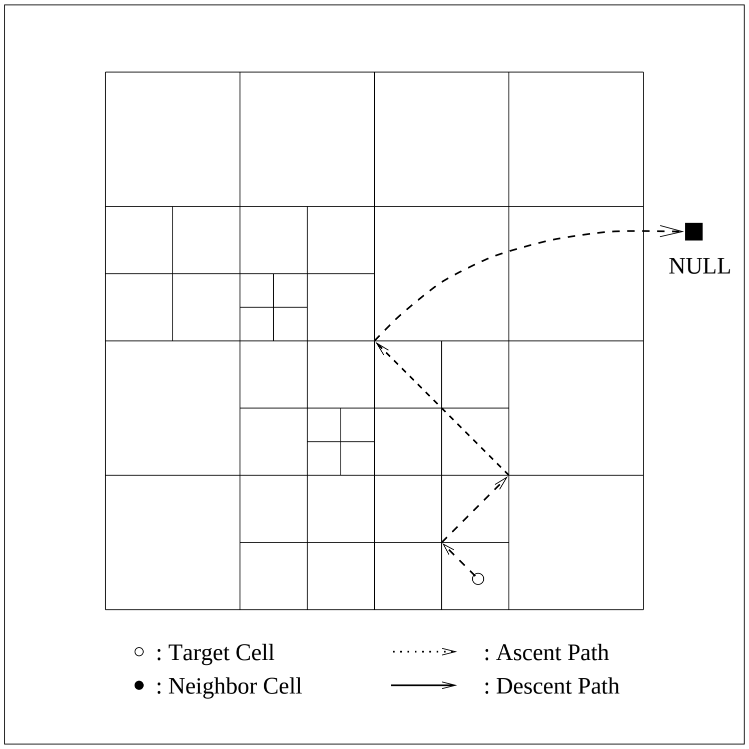

Figure 6.6: An example showing the ascent path traversed in the attempt to locate the S neighbor(s) of a specific target cell, and the determination that the target cell has no neighbor(s) in the selected direction.

In Figure 6.6, the neighbor sought for the target cell is the S neighbor. The ascent sequence in this case is given by $\{NW, NE, NW, NULL\}$, where the last element in the sequence indicates that the ascent path has surpassed the root node of the tree without achieving the cross-over condition (for the S direction). In this case, no attempt is made to construct a descent path. Instead, the sought neighbor of the target node is identified as the NULL (that is, the non-existent) node. As shown in the figure, the target node has no neighbor in the sought direction.

The correctness of the above algorithm can be proved from the following corollaries, which can be individually proved as indicated:

1. Let N be any node in a Quadtree, and let S be the Cartesian square in the Computational Region that corresponds to this node. Let $\tilde{n} = \{n_1, \cdots, n_m\}$ be the set of all leaf nodes that are (directly or indirectly) subordinate to node N (allowing for the degenerate case $\tilde{n} = \{N\}$), and let $\tilde{s} = \{s_1, \cdots, s_m\}$ be the set of squares corresponding to the leaf nodes that are subordinate to node N, such that square s_i corresponds to leaf node n_i. Let e be any edge separating any two squares s_i and s_j in $\tilde{s}$ (implying that, possibly except for sets of points of measure zero in $\mathbf{R^2}$, e lies wholly in the interior of S). Then, there exists a path of links within the sub-tree rooted in N that leads from node n_i to node n_j, and vice-versa. This corollary can be proved as follows: since n_i and n_j are subordinate to N, there exist ascending paths of links from each of them to N, and there also exist descending paths from N to each of them.

2. Let e be an edge of a square that corresponds to a leaf node in a Quadtree such that e does not lie wholly in the boundary of the root square of the Quadtree. Then there exists at least one interior node whose corresponding square wholly contains e in its interior, and also wholly contains the two unique squares that are joined by the edge e. This corollary can be proved from the construction procedure of the edges and squares during the spatial sub-division process.

3. Let N be any node in a Quadtree, let P_d be any path along descending links

from N, and let P_a be any path along ascending links terminating in N. Let D be any of the two directions traversed by any link l in P_d or P_a. Then the "extent of travel" along D in link l is greater than the sum of the "extents of travel" in the direction D of all the links that are subordinate to link l. Here, extent of travel may be measured, for example, in terms of physical distance, or in terms of the number of cells at the finest refinement level in the tree. This corollary can be proved from the fact that any term in a finite geometric series of ratio $\frac{1}{2}$ is greater than the sum of all the succeeding terms in the series.

4. Let n be a leaf node in a Quadtree, let s be the square that corresponds to n, let D be any direction from the set S, W, N, E, and let e_D be the edge of s in the D direction. Let P be the (unique) ascending path from n that passes through the root node. The first ascending link l in P that has a component in the direction D leads to a node N whose corresponding square contains the edge e_D within its interior. This can be proved from Corollaries 2 and 3 above. This corollary corresponds to the cross-over condition described in the neighbor-determination algorithm given above. This corollary also accounts for the case of the non-existent neighbor.

5. The node N identified in Corollary 4 above is the "least superior" super-node in the Quadtree that contains within its interior the edge e_D that was described in Corollary 4 above. This can be proved from Corollaries 3 and 4 above.

Corollaries 5 and 2 above can be combined to show that the node N identified in Corollary 5 satisfies the requirements of the node N identified in Corollary 1, proving the existence of a path leading from the node n identified in Corollary 4 to its "neighbor node" in the D direction. The ascending portion of the path has already been identified in Corollary 4. For a uniformly-refined Quadtree, the correctness of the construction of the descending portion of the path can be proved from the symmetry about the x-axis and the y-axis of the directed links in the Quadtree data-structure.

For a non-uniformly-refined Quadtree, the truncation or extension of the descent path to arrive at the required neighbor in the manner described in the above algorithm is intuitively obvious, and its correctness is not proved in this work. For a uniformly-refined Quadtree, the proofs given above are applicable not only to the "layer" of leaf nodes, but also to any other layer of interior nodes (where a layer in a tree data-structure is the set of all nodes that are at some refinement level, or depth, r, such that $1 \leq r \leq d$).

An important consideration with any neighbor-determination procedure for unstructured grids is its computational performance, in terms of both operation count and storage space, especially relative to a standard structured-grid reference, as described further in Chapter IV. The operation count for the above algorithm can be computed exactly for a uniformly-refined grid, and can be closely estimated for grids with non-uniform refinement, as described in more detail below.

For a uniformly-refined Quadtree with depth $d \geq 1$ (where here depth is defined such that $d = 1$ corresponds to the root node), the average number of ascents executed during an edge-neighbor-determination operation for a leaf node, $\overline{n}_a$, can be expressed as the Expected Value for the number of ascents; to wit:

$$E_a = \sum_{i=1}^{d} \quad iP(i) \tag{6.12}$$

where E_a is the Expected Value for the number of ascents, i is the dummy integer variable representing the number of ascents, and $P(i)$ is the probability that for an arbitrarily-selected leaf node, i ascents will be required in the Quadtree.

Equation 6.12 can be evaluated in closed form by deriving a general, closed-form expression for $P(i)$ which, by definition, is the fractional number of leaf nodes that require i ascents, that is, $P(i) = \frac{N(i)}{\sum_{i=1}^{d} N(i)}$, where $N(i)$ is the number of leaf nodes that require i ascents. The values of $N(i)$ for $1 \leq i \leq d$ can be derived by examination of the link structure in a Quadtree, and the correspondence of this link structure to the spatial location of the Cartesian squares corresponding to the leaf nodes.

Figure 6.7 shows the number of ascents required (to achieve the cross-over condition described in Step 2 of the edge-neighbor-determination algorithm) for locating the E edge-neighbor for each leaf node in a uniformly-refined Quadtree of depth 5. As shown in the figure, the set of nodes corresponding to the right-most column of cells will require d ascents, since the ascending path must proceed beyond the root node in order to determine that the cross-over condition cannot be achieved. There is exactly one 1-major vertical bisector of the root square, and all the nodes corresponding to the column of cells to the immediate left of this bisector require $d - 1$ ascents to achieve the cross-over condition, since the cross-over condition is only achieved through the ascending link that passes through the root node. There are exactly two 2-major vertical bisectors of the root square, and all the nodes corresponding to the two columns of cells to the immediate left of these two bisectors require $d - 2$ ascents to achieve the cross-over condition. Similarly, there are exactly four 3-major vertical bisectors of the root square, and all the nodes corresponding to the four columns of cells to the immediate left of these four bisectors require $d - 3$ ascents to achieve the cross-over condition, and so on.

In its most general form, the pattern that emerges from the arrangement shown in Figure 6.7 can now be expressed as follows: for the 2^{m-1} m-major vertical bisectors

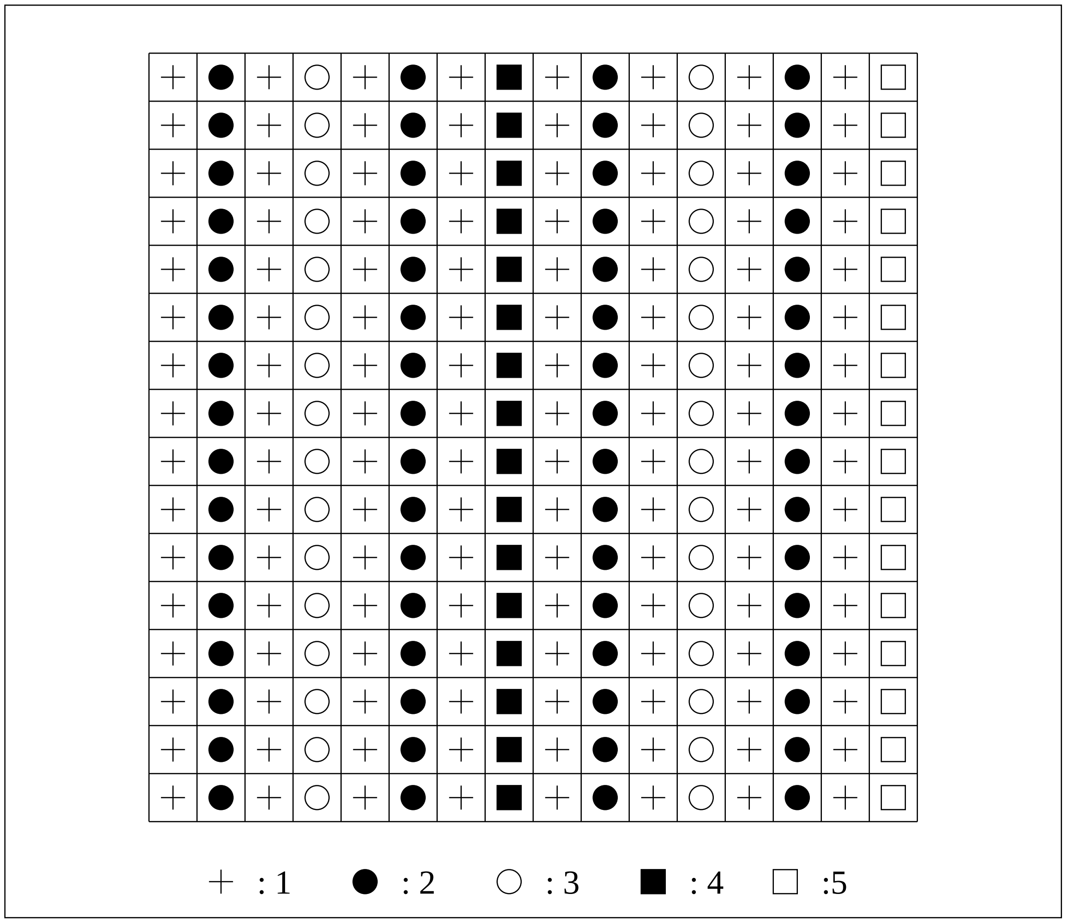

Figure 6.7: A Quadtree-based grid of depth 5, showing the patterns for the number of ascents required in each cell to locate the corresponding E edge-neighbor.

of the root square, all the nodes corresponding to the 2^{m-1} columns of cells to the immediate left of these 2^{m-1} bisectors require $d - m$ ascents to achieve the cross-over condition.

Although the cross-over condition specifically examined in Figure 6.7 corresponds to that required to locate the E edge-neighbor, by the symmetry of Quadtree grids, the results obtained and the conclusions drawn above generalize to the location of edge-neighbors in any direction.

In terms of the relation for $N(i)$ as a function of $1 \leq i \leq d$, the results obtained above from examination of the structure of Quadtree can be expressed in the following table:

$$N(d) = 2^{d-1}$$

$$N(d-1) = 2^{d-1} \times 2^0$$

$$N(d-2) = 2^{d-1} \times 2^1$$

$$\vdots$$

$$N(d-i) = 2^{d-1} \times 2^{i-1}$$

$$\vdots$$

$$N(1) = 2^{d-1} \times 2^{d-2}$$

For each $1 \leq i \leq d$, the fractional number of nodes, $P(i)$, is obtained by dividing $N(i)$ by the total number of cells in the grid; namely, $\sum_{i=1}^{d} N(i) = 2^{d-1}2^{d-1}$. The average number of ascents may be computed by substituting these results in Equation 6.12, to give

$$E_a = \left(\sum_{i=1}^{d-1} \frac{2^{d-1}}{2^{d-1}2^{d-1}} \quad 2^{i-1} \quad (d-i) \right) + \frac{d2^{d-1}}{2^{d-1}2^{d-1}} \tag{6.13}$$

where the term for $P(d)$ has been treated in isolation to avoid artificially casting it into the general pattern of the other terms.

Equation 6.13 can be re-written in the form

$$E_a = \frac{1}{2^{d-1}} \left(\frac{d}{2} \sum_{i=1}^{d-1} 2^i - \frac{1}{2} \sum_{i=1}^{d-1} i2^i \quad + d \right) \tag{6.14}$$

which can be reduced to

$$E_a = 2 - \frac{2}{2^d} \tag{6.15}$$

by using the expansions $\sum_{i=1}^{d-1} 2^i = 2(2^{d-1} - 1)$ and $\sum_{i=1}^{d-1} i2^i = 4(d-1)2^{d-1} - 2d2^{d-1} + 2)$, where the first expansion can be derived from the standard formula for a geometric series, and the second expansion can be proved by induction, or by substitution in the general formula

$$\sum_{i=1}^{n} im^i = \frac{nm^{n+2} - (n+1)m^{n+1} + m}{(m-1)^2}$$

In words, Equation 6.15, which is valid for any $d \geq 1$, shows that the average number of ascents for the leaf nodes of a uniformly-refined Quadtree asymptotically approaches 2 from below. A similar result is obtained in [318] for an average that includes all the nodes in the tree, and under different smoothness constraints than applied in this work. The result obtained above is clearly also equally valid for any layer of nodes in a uniformly-refined Quadtree. Moreover, since the dependence on d in Equation 6.15 is weak, and since the number of nodes increases exponentially with increasing d, the result can be taken as an accurate average for determination of in-layer edge-neighbors across the entire tree, not just the layer of the leaf nodes.

Since in all cases in this work, the average number of descents for existing edge neighbors differs by no more than 1 from the number of ascents (being either equal to, greater than by 1, or less than by 1), the average total number of link-tracing operations can be approximated by 4. The effect of the presence of boundary cells (that is, cells with no neighbors) is to reduce the average number of link-tracing operations since for the nodes corresponding to such cells, no descents are executed.

In addition to the average operation count required to locate an edge neighbor, the above discussion also implicitly presented the maximum and minimum possible number of operations for the arbitrary node: the minimum number of ascents is 1; the maximum number is d. Operation counts for the overall grid can also be deduced: no matter what the form of the grid is, exactly one half of the cells will require exactly one ascent and one descent to locate an edge neighbor. This is also evident in the expression for $N(1)$ given above. In the worst case, the maximum number of ascents is obviously equal to the depth of the tree, while the maximum number of combined ascents and descents for $d \geq 2$ will be $2(d-1)$. For all $d \geq 2$, the effects of refinement-level differences are typically negligible, since they tend to have opposing effects on the lengths of the ascent and descent paths. Extensive testing of the edge-neighbor-determination algorithm with grids having depths varying between 5 and 20 showed that the average predictions given above are respected typically to within a few per-cent in the actual implementation.

For grids with non-uniform refinement, the maximum and minimum number of ascents and descents remains unchanged, and exactly half the leaf nodes will still require exactly one ascent and one descent to locate an edge neighbor. However, the average number of ascents and descents depends on the location of the finest cells. For example, if the finest cells are clustered near the centroid of the root square, then the crossing of coarser bisectors in the edge-neighbor-determination operation will become more frequent, and the operation count will increase beyond what it would be if the fine cells where confined in a minor quadrant or sub-quadrant of the root square.

Compared to the neighbor-determination operation for a structured grid, the operation count for the algorithm described above still requires far more computational effort. In the least, the neighbor-determination operation for a structured grid does not even require a function call. However, for the entire range of large-scale cases that were profiled in this work, the neighbor-determination operations required no more than a few per-cent of the computational effort required for the flux-calculation operations.

In regard to storage space, one of the chief advantages of the algorithm described above is that it does not require any extra permanent storage (beyond the storage for the Quadtree data-structure itself). Although it is possible to store the neighborhood connectivity by links from each node to its neighbors, this is not done in this work for several reasons: (i) it requires more memory; (ii) it could result in little saving in the computational effort for an adaptive simulation in which the cell-to-cell neighborhood patterns change frequently; and (iii) it is messy and inelegant, requiring extensive checking and updating, especially for vertex-neighbors.

Another major attraction of the above algorithm is that is it entirely based on the "topology" or connectivity of the tree, with no recourse to geometric calculations or searches. This contributes strongly to its robustness and speed.

Additional implementation considerations, especially those regarding the trade-off between storage space and computational effort are discussed in Section 6.8.

The above discussion can be generalized to edge-neighbor-determination in Octrees, and more generally, to higher-dimensional 2^n-ary trees. This is the case both for the neighbor-determination algorithm and for the operation-count derivations.

6.6.1.2 Determination of Vertex Neighbors

In a Quadtree-based grid (which is always a 2-D grid), a vertex neighbor for a given computational cell, cell C, may either be a South-East neighbor, a South-West neighbor, a North-West neighbor, or a North-East neighbor of the given cell. Let the sought neighborhood direction be D_1D_2, where D_1 and D_2 are either S, W, N, or E (and where, as explained above, $D_1 \neq D_2$, and the ordering of the symbols D_1 and D_2 in the expression D_1D_2 follows the standard convention for geographical directions). The procedure used to determine the vertex neighbor in direction D_1D_2

is as follows:

1. Starting in the Quadtree at the node corresponding to cell C, ascend the tree using the node-to-super-node links, one by one, storing (in sequence) the individual ascent directions, until each of the two individual components D_1 and D_2 of the sought neighborhood direction D_1D_2 have been traversed by the individual components of the ascent directions. As explained above, the ascent directions possible are SE, SW, NW, and NE, and each of these directions contains two components, either of which may be aligned with D_1 or D_2. Note that the traversal of the D_1 and D_2 directions may occur simultaneously, or sequentially. For example, suppose D_1D_2 is SE. The ascent-direction sequence SW, SW, NW, NE satisfies the requirement since the first element in the sequence satisfies the S component, and the last element in the sequence satisfies the E component. Similarly, the single ascent direction SE satisfies the requirement since the two components of the ascent direction simultaneously satisfy the two components S and E. Store the super-node, node S, arrived at during the last ascent in the sequence. The traversal of each of the directions D_1 and D_2 is, as mentioned above, called a cross-over event, and this event is guaranteed to occur twice for any ascent, except in one situation: cell C is a cell that lies on either the D_1-boundary or the D_2-boundary of the grid, and therefore has no neighbors in the D_1D_2 direction. In both of these cases, at most only one of the two required cross-over conditions can be achieved, and the ascent continues through the root node of the Quadtree to the (NULL) super-node of the root node. When this happens, the desired neighbor of cell C is identified and returned as the NULL (that is, the non-existent) neighbor, and the neighbor-determination procedure is terminated. Otherwise, the

algorithm continues through Steps 2 and 3 below.

2. Reflect (as described above) the individual segments of the ascent path sequence traced in Step 1 above in either a D_1-edge passing through the node S, or a D_2-edge passing through the node S, or the D_1D_2 vertex of cell C, as described further below. This reflection process is clearly more complicated than the corresponding reflection process for edge-neighbor-determination, partly because the reflections for different segments may differ and because they have to be determined by analysis of the ascent path. The specific procedure used to determine and perform the reflections for the individual segments in the ascent path sequence is as follows:

    (a) If any segment in the ordered ascent path, say, segment s_a, satisfies exactly one of the required direction components D_1 and D_2, then all segments occurring after this segment in the ascent sequence are reflected in an edge passing through node S such that the orientation of that edge is given by whichever of D_1 and D_2 was not satisfied by segment s_a. Segment s_a and all segments occurring before it in the ascent sequence are vertex-reflected in the vertex D_1D_2 of cell C. This procedure is applied whether or not the last segment in the ascent sequence satisfies only the remaining unsatisfied direction, or whether it satisfies both of the directions D_1 and D_2 simultaneously.

    (b) If the two required direction components, D_1 and D_2 are traversed simultaneously by a single ascent, and neither is satisfied separately in any ascent, then all the segments in the ascent path are vertex-reflected in the vertex D_1D_2 of cell C.

The two possibilities described above are exhaustive. After all the required reflections are carried out, the directions of the links are reversed, converting in the process every ascending link into a descending link. All reflection and reversal operations are efficiently carried out in modulo-4 integer arithmetic, as done for the edge-neighbor-determination algorithm. After reflecting and reversing the links, the sequential order of each reflected, reversed segment is inverted to generate a new sequence. The sequence of links that is created by these reflections and reversals and by the order-inversion defines a descent path that is used to locate the sought vertex neighbor.

3. Starting at node S, the node in which the ascent sequence terminated in Step 1 above, follow the descent path obtained by the reflections and reversals and the order inversion in Step 2 above. This descent path will terminate in the Quadtree node that corresponds to the sought neighbor of cell C. The cases where cell C and its sought neighbor are not at the same refinement level, however, require either truncation or extension or the descent path, and this is done as follows: If the descent path cannot be pursued to its end because the path has encountered a leaf node before the path has terminated, then that leaf node is the one corresponding to the sought neighbor of cell C. In this case, the sought neighbor is either one or two refinement levels coarser than cell C, and the descent path will be truncated by, respectively, either exactly one or exactly two levels, corresponding, respectively, to either the last one or the last two segment(s) of the path. These outcomes are the result of the one-refinement-level-difference restriction imposed in this work. If the descent path terminates before reaching a leaf node, then the descent path is extended by adding either one or two segments, extending the depth of the path, respectively, by either

one or two levels. The directions of the extension segment(s) are given by D_2D_1, that is, by the reverse of the direction of the sought-neighbor. The extended descent path will lead to the unique cell that is the sought neighbor of cell C, and that is either one or two levels of refinement finer than cell C. Again, these outcomes are consequences of the one-level-refinement-difference restriction imposed in this work. The two exceptions described above are the only ones that may arise under the one-level-refinement-difference restriction imposed in this work, and, as shown above, both of them are treated in a manner that discloses the unique and complete identity of the sought vertex neighbor. Unlike the case for the edge-neighbor-determination algorithm, no branching is required in extending the descent path for the vertex-neighbor-determination algorithm, since (no matter what differences are allowed in the refinement level of adjacent cells) no cell in a 2^n-ary tree can have more than one vertex neighbor. The handling described in this step of the two cases of unequal refinement levels between neighbors is illustrated further in all three of the examples given below.

Figures 6.8, 6.9, and 6.10 illustrate the behavior of the vertex-neighbor-determination algorithm described above in three different common situations. In all these figures, ascending and descending segments are represented by arrows in the same manner as described above for the edge-neighbor-determination algorithm. Because of the similarity of the main ideas and procedures in the vertex-neighbor-determination algorithm with those of the edge-neighbor-determination algorithm, these examples are described more briefly than the examples presented for the edge-neighbor-determination algorithm.

In Figure 6.8, the neighbor sought for the target cell is the SE neighbor. The ascent sequence in this case is $\{NE, NW, SW\}$, where the first cross-over condition (here, in the direction E) is separately achieved by the first element, and the second cross-over condition (here, in the direction S) is separately achieved by the last element. In accordance with the prescription in Step 2(a) in the vertex-neighbor-determination algorithm, the first segment must be reflected in the SE vertex of the target cell, and the second and third segments must be reflected in an S-edge passing through the centroid of the square associated with the terminal node in the ascent path. The edge-reflected and vertex-reflected path is therefore given by $\{SW, SW, NW\}$. Reversing the segment directions gives the sequence $\{NE, NE, SE\}$, and re-ordering the sequence gives the required descent path sequence, $\{SE, NE, NE\}$. In this case, the last segment of the descent path cannot be pursued, and is truncated, and this is because the sought vertex-neighbor is one refinement level coarser than the target cell.

In Figure 6.9, the neighbor sought for the target cell is again the SE neighbor. The ascent sequence in this case is $\{NW, NW, NW, SE\}$, where the two required cross-over conditions are (simultaneously) achieved only by the last element. The reflected sequence for this rare case is obtained as described in Step 2(b) in the vertex-neighbor-determination algorithm, by reflecting all segments in the vertex SE of the target cell, to give the sequence $\{SE, SE, SE, NW\}$. Reversal of the segment directions gives the sequence $\{NW, NW, NW, SE\}$, and inversion of the order of the elements gives the descent path sequence, $\{SE, NW, NW, NW\}$. In this case, however, the last two segments in the descent path must both be truncated, and this is because the sought vertex-neighbor is two refinement levels coarser than the target cell.

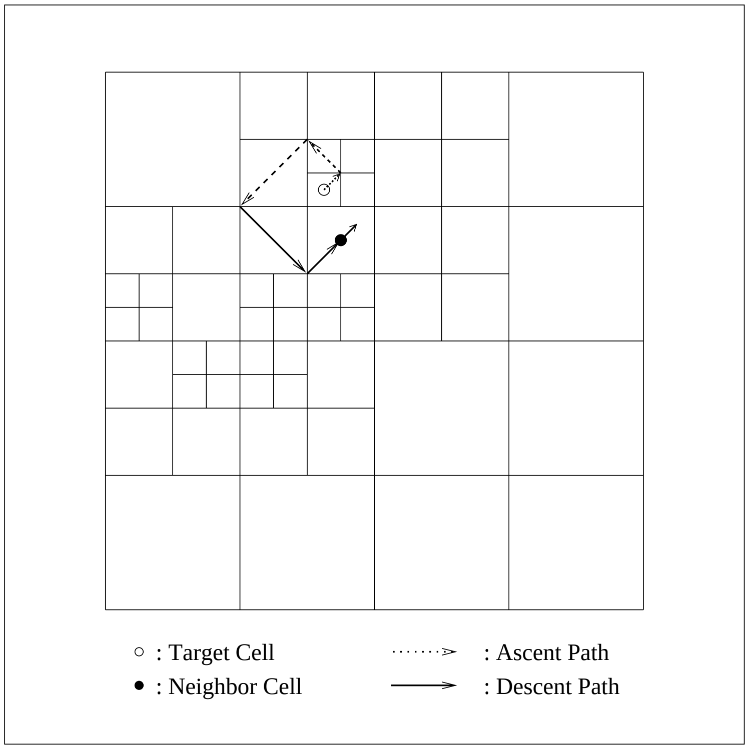

Figure 6.8: An example showing the ascent and descent paths traversed to locate the SE vertex-neighbor of a specific target cell.

In Figure 6.10, the neighbor sought for the target cell is the NW neighbor. The ascent sequence in this case is $\{SE, SE, SW, NE\}$, where the first cross-over condition is separately achieved in the third element, and the second cross-over condition is separately achieved in the fourth element. As described in Step 2(a) of the vertex-neighbor-determination algorithm, the fourth element of the ascent sequence must be reflected in an N-edge passing though the terminal node of the ascent path, and the first three elements must be reflected in the NW vertex of the target cell. The reflected sequence therefore evaluates to $\{NW, NW, NE, SE\}$. Reversing the direction of the elements of the sequence gives the sequence $\{SE, SE, SW, NW\}$. Inverting the order of this last sequence gives the descent path sequence, $\{NW, SW, SE, SE\}$. Again the last segment in the descent path must be truncated because of the differ-

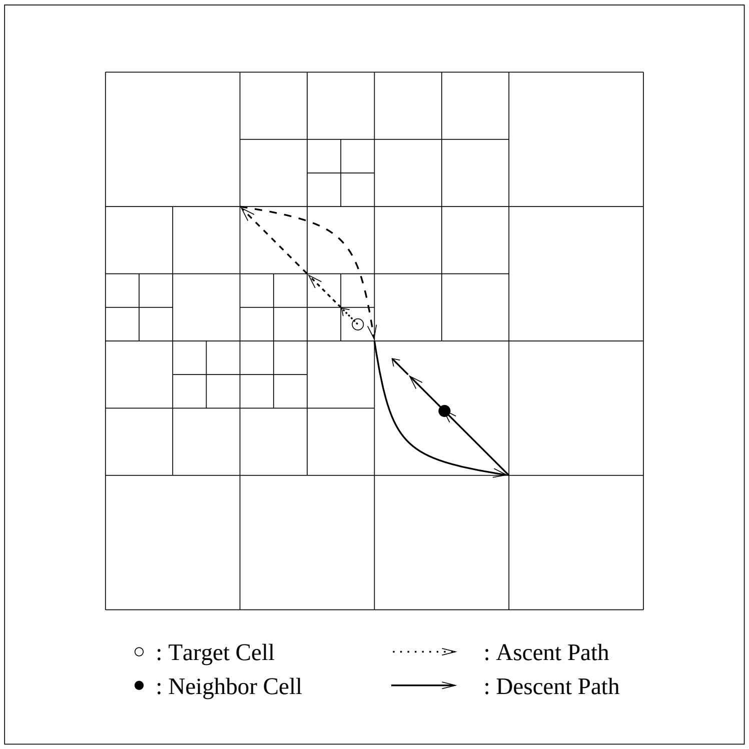

Figure 6.9: An example showing the ascent and descent paths traversed to locate the SE vertex-neighbor of a specific target cell.

ence in refinement level between the target node and its sought neighbor, as shown in the figure. Had the sought neighbor been three levels finer than it is in the figure, the descent path would have been extended by two SE segments, as described in Step 3 of the vertex-neighbor-determination algorithm.

The correctness of the vertex-neighbor-determination algorithm described above can be established in a similar manner to that followed for the edge-neighbor-determination algorithm. The same corollaries can be used. Alternatively, those corollaries can be replaced or augmented by the analogous corollaries for vertices (instead of edges). Indeed, in many respects, locating a vertex neighbor corresponds to locating two "offset" edge neighbors.

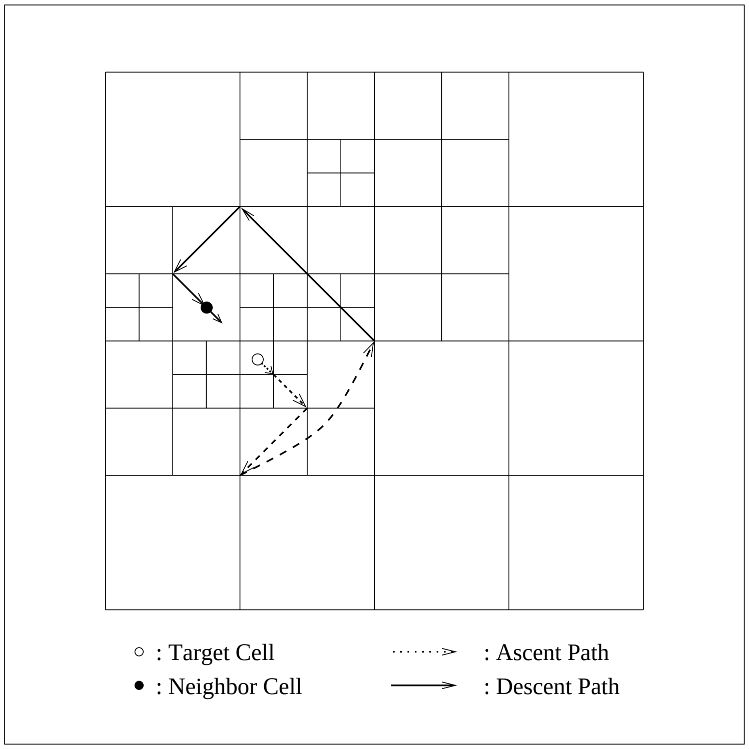

Figure 6.10: An example showing the ascent and descent paths traversed to locate the *NW* vertex-neighbor of a specific target cell.

The remarks made for the edge-neighbor-determination algorithm in regard to the extension of the proof of correctness of that algorithm to the handling of non-uniformities in refinement level also apply to the proof of correctness of the vertex-neighbor-determination algorithm. In addition, the remarks made for the edge-neighbor-determination algorithm in regard to the applicability of the proof of correctness of that algorithm to neighbor determination in any homogeneously-refined layer of nodes in the Quadtree also apply to the corresponding applicability of the proof of correctness of the vertex-neighbor-determination algorithm.

The computational performance of the vertex-neighbor-determination algorithm described above can also be evaluated in a similar manner to that used for the edge-neighbor-determination algorithm. As for the edge-neighbor-determination algorithm, the operation count for the vertex-neighbor-determination algorithm can be computed exactly for a uniformly-refined grid, and can be closely estimated for grids with non-uniform refinement, as described in more detail below.

For a uniformly-refined Quadtree with depth $d \geq 1$, (where here depth is defined such that $d = 1$ corresponds to the root node), the average number of ascents executed during an edge-neighbor-determination operation for a leaf node, $\overline{n}_a$, can be expressed as the Expected Value for the number of ascents, as given by Equation 6.12.

As done for the edge-neighbor-determination algorithm, Equation 6.12 can be evaluated in closed form by deriving a general, closed-form expression for $P(i)$ which, by definition, is the fractional number of nodes that require i ascents, that is, $P(i) = \frac{N(i)}{\sum_{i=1}^{d} N(i)}$, where $N(i)$ is the number of nodes that require i ascents. The values of $N(i)$ for $1 \leq i \leq d$ can be derived by examination of the link structure in a Quadtree, and the correspondence of this link structure to the spatial location of the Cartesian squares corresponding to the leaf nodes. This process, however, is more involved for vertex-neighbor-determination than for edge-neighbor-determination, largely because of the need to avoid double-counting of nodes.

Figure 6.11 shows the number of ascents required (to achieve the cross-over conditions described in Step 2 of the vertex-neighbor-determination algorithm) for locating the NE vertex-neighbor for each leaf node in a uniformly-refined Quadtree of depth 5. As shown in the figure, the set of nodes corresponding to the right-most and top-most columns of cells will require d ascents, since the ascending path must proceed beyond the root node in order to determine that the required cross-over conditions cannot be achieved.

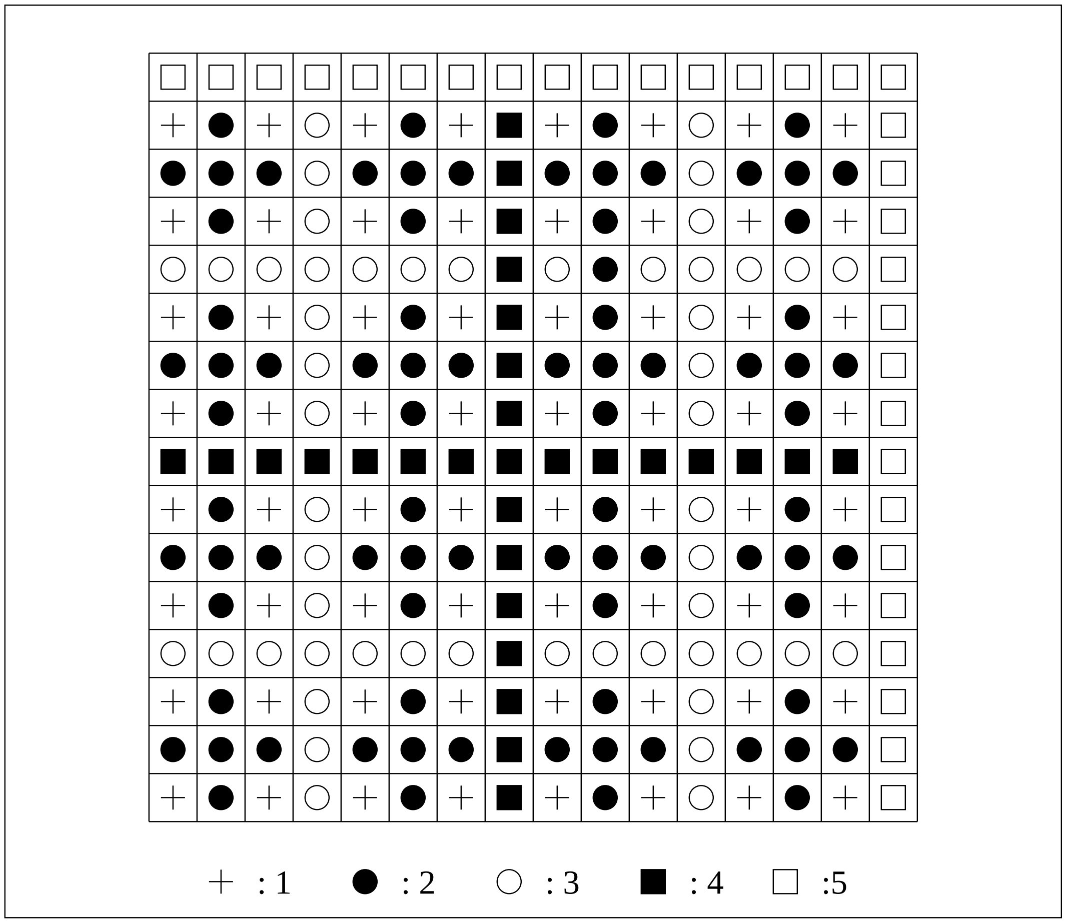

Figure 6.11: A Quadtree-based grid of depth 5, showing the patterns for the number of ascents required in each cell to locate the corresponding NE vertex-neighbor.

As shown in Figure 6.11, there is exactly one 1-major vertical bisector of the root square, and exactly one 1-major horizontal bisector of the root square. All the nodes corresponding to the column of cells to the immediate left of the 1-major vertical bisector, except for the node that falls in the top-most row of nodes, require $d-1$ ascents to achieve the cross-over conditions (since the E cross-over condition can only be achieved through the ascending link that passes through the root node). Similarly, all the nodes corresponding to the row of cells immediately below the 1-major horizontal bisector, except for the node that falls in the right-most row of nodes, require $d-1$ ascents to achieve the cross-over conditions (since the N cross-over condition can only be achieved through the ascending link that passes through the root node). In addition, the node that is shared by the vertical column of nodes and the horizontal row of nodes that were just described must not be double-counted. Adding the number of nodes in this case shows that there are exactly $2\left(2^{d-1}-1\right)-1$ nodes that require $d-1$ ascents to achieve the cross-over conditions.

Figure 6.11 also shows that there are exactly two 2-major vertical bisectors of the root square. All the nodes corresponding to the two columns of cells to the immediate left of these two bisectors, except for the two nodes that fall in the top-most row of nodes and the two nodes that fall in the 1-major row of nodes require $d-2$ ascents to achieve the cross-over condition. Similarly, there are exactly two 2-major horizontal bisectors of the root square. All the nodes corresponding to the two rows of cells immediately below these two bisectors, except for the two nodes that fall in the right-most column of nodes and the two nodes that fall in the 1-major column of nodes require $d-2$ ascents to achieve the cross-over condition. The nodes which are subject to double-counting, of which there are 2×2 here, must also be taken into account.

Figure 6.11 also shows that there are exactly four 3-major vertical bisectors of the root square. All the nodes corresponding to the four columns of cells to the immediate left of these four bisectors, except for the four nodes that fall in the top-most row of nodes and the four nodes that fall in the 1-major row of nodes and the eight nodes that fall in the two 2-major rows of nodes require $d-3$ ascents to achieve the cross-over condition. Similarly, there are exactly four 3-major horizontal bisectors of the root square. All the nodes corresponding to the four rows of cells immediately below these four bisectors, except for the four nodes that fall in the right-most column of nodes and the four nodes that fall in the 1-major column of nodes and the eight nodes that fall in the two 2-major columns of nodes require $d - 3$ ascents to achieve the cross-over condition. The nodes that are subject to double-counting here number 4×4.

Figure 6.11 also shows that there are exactly eight 4-major vertical bisectors of the root square. All the nodes corresponding to the eight columns of cells to the immediate left of these eight bisectors, except for the eight nodes that fall in the top-most row of nodes and the eight nodes that fall in the 1-major row of nodes and the sixteen nodes that fall in the two 2-major rows of nodes and the thirty-two nodes that fall in the four 3-major rows of nodes require $d - 4$ ascents to achieve the cross-over condition. Similarly, there are exactly eight 4-major horizontal bisectors of the root square. All the nodes corresponding to the eight rows of cells immediately below these eight bisectors, except for the eight nodes that fall in the right-most column of nodes and the eight nodes that fall in the 1-major column of nodes and the sixteen nodes that fall in the two 2-major columns of nodes and the thirty-two nodes that fall in the four 3-major columns of nodes require $d - 4$ ascents to achieve the cross-over condition. The nodes that are subject to double-counting here number 8×8.

The procedure used in the four preceding paragraphs clarifies the pattern for the number of ascents required at leaf nodes, and can be applied to a tree of any depth. Furthermore, although the cross-over conditions specifically examined in Figure 6.11 corresponds to those required to locate the NE vertex-neighbor, by the symmetry of Quadtree grids, the results obtained and the conclusions drawn above generalize to the location of vertex-neighbors in any direction.

In terms of the relation for $N(i)$ as a function of $1 \leq i \leq d$, the results obtained above from examination of the structure of Quadtree can be expressed in the following table:

$$N(d) = 2 \times 2^{d-1} - 1$$

$$N(d-1) = 2\left(2^{d-1} - 1\right) - 1$$

$$N(d-2) = 4\left(2^{d-1} - 1 - 1\right) - 2 \times 2$$

$$N(d-3) = 8\left(2^{d-1} - 1 - 1 - 2\right) - 4 \times 4$$

$$N(d-4) = 16\left(2^{d-1} - 1 - 1 - 2 - 4\right) - 8 \times 8$$

$$\vdots$$

$$N(d-i) = 2^{i}\left(2^{d-1} - \sum_{j=2}^{i} 2^{j-2} - 1\right) - 2^{i-1}2^{i-1}$$

$$\vdots$$

$$N(1) = 2^{d-1}\left(2^{d-1} - \sum_{j=2}^{d-1} 2^{j-2} - 1\right) - 2^{d-2}2^{d-2}$$

The general expression for $d-i$ ascents can be simplified by reducing the term corresponding to the finite geometric series (using the relation $\sum_{j=2}^{i} 2^{j-2} = 2^{i-1} - 1$), to give

$$N(d-i) = 2^{i}\left(2^{d-1} - 2^{i-1}\right) - 2^{i-1}2^{i-1}$$

For each $1 \leq i \leq d$, the fractional number of nodes, $P(i)$ is obtained by dividing $N(i)$ by the total number of cells in the grid; namely, $\sum_{i=1}^{d} N(i) = 2^{d-1}2^{d-1}$. The average number of ascents may be computed by substituting these results in Equation 6.12, to give

$$E_a = \left(\sum_{i=1}^{d-1}\left(\frac{2^{i}(2^{d-1} - 2^{i-1}) - 2^{i-1}2^{i-1}}{2^{d-1}2^{d-1}}\right)(d-i)\right) + \frac{(2 \times 2^{d-1} - 1)d}{2^{d-1}2^{d-1}} \qquad (6.16)$$

where the term for $P(d)$ has been treated in isolation to avoid artificially casting it in the general pattern of the other terms.

Equation 6.16 can be re-written in the form

$$E_a = \frac{2d}{2^d}\sum_{i=1}^{d-1} 2^i - \frac{2}{2^d}\sum_{i=1}^{d-1} i2^i - \frac{3d}{2^{2d}}\sum_{i=1}^{d-1} 2^{2i} + \frac{3}{2^{2d}}\sum_{i=1}^{d-1} i2^{2i} + \frac{(2^d-1)d}{2^{d-1}2^{d-1}} \tag{6.17}$$

which can be reduced to

$$E_a = \frac{2d}{2^d}(2^d-2) - \frac{2}{2^d}((d-2)2^d+2) - \frac{3d}{2^{2d}}\frac{(2^{2d}-2^2)}{3} + \frac{3}{2^{2d}}\frac{(3d2^{2d}-4\times 2^{2d}+4)}{9} + \frac{4(2^d-1)d}{2^{2d}} \tag{6.18}$$

which, by using the relations for geometric series, and for sums of the form $\sum_{i=1}^{n} im^i$ as described in the preceding sub-section, can be reduced to

$$E_a = 4 - \frac{4}{3} - \frac{4}{2^d} + \frac{4}{3\times 2^{2d}} = \frac{8}{3} - \left(\frac{12\times 2^d - 4}{3\times 2^{2d}}\right) \tag{6.19}$$

In words, Equation 6.19, which is valid for $d \geq 1$, shows that the average number of ascents for the leaf nodes of a uniformly-refined Quadtree asymptotically approaches $\frac{8}{3}$ from below. A similar result is obtained in [318] for an average that includes all the nodes in the tree, and under different smoothness constraints than applied in this work. The result obtained above is clearly also equally valid for any layer of nodes in a uniformly-refined Quadtree. Moreover, since the dependence on d in Equation 6.19 is weak, and since the number of nodes increases exponentially with increasing d, the result can be taken as an accurate average for determination of in-layer vertex-neighbors across the entire tree, not just the layer of the leaf nodes.

Since in all cases in this work, the average number of descents for existing vertex neighbors differs by no more than 2 from the number of ascents (being either equal to, greater than by 1, greater than by 2, less than by 1, or less than by 2), the average total number of link-tracing operations can be approximated by $\frac{16}{3}$. The effect of the presence of boundary cells (that is, cells with no neighbors) is to reduce the average number of link-tracing operations since for the nodes corresponding to such cells, no descents are executed.

In addition to the average operation count required to locate a vertex neighbor, the above discussion also implicitly presented the maximum and minimum possible number of operations for the arbitrary node: the minimum number of ascents is 1; the maximum number is d. Operation counts for the overall grid can also be deduced: no matter what the form of the grid is, exactly one quarter of the cells will require exactly one ascent and one descent to locate a vertex neighbor. This is also evident in the expression for $N(1)$ given above. In the worst case, the maximum number of ascents is obviously equal to the depth of the tree, while the maximum number of combined ascents and descents for $d \geq 2$ will be $2(d-1)$. For all $d \geq 2$, the effects of refinement-level differences are typically negligible, since they tend to have opposing effects on the lengths of the ascent and descent paths. Extensive testing of the vertex-neighbor-determination algorithm with grids having depths varying between 5 and 20 showed that the average predictions given above are respected typically to within a few per-cent in the actual implementation.

For grids with non-uniform refinement, the maximum and minimum number of ascents and descents remains unchanged, and exactly one quarter of the leaf nodes will still require exactly one ascent and one descent to locate a vertex neighbor. However, the average number of ascents and descents depends on the location of the finest cells. For example, if the finest cells are clustered near the centroid of the root square, then the crossing of coarser bisectors in the vertex-neighbor-determination operation will become more frequent, and the operation count will increase beyond what it would be if the fine cells where confined in a minor quadrant or sub-quadrant of the root square.

The remarks made for the edge-neighbor-determination algorithm in regard to the comparison of its operation count with that for structured grids apply equally well the vertex-neighbor-determination algorithm. Similarly, the remarks made in regard to the storage space implementation options for the edge-neighbor-determination algorithm apply equally well to the the vertex-neighbor-determination algorithm. The remark regarding the basing of the edge-neighbor-determination algorithm entirely on the "topology" or connectivity of the tree, and the contribution of this to the speed and robustness of the algorithm also apply equally well to the vertex-neighbor-determination algorithm.

Additional implementation considerations, especially those regarding the trade-off between storage space and computational effort are discussed in Section 6.8.

The above discussion can be generalized to vertex-neighbor-determination in Octrees, and more generally, to higher-dimensional 2^n-ary trees. This is the case both for the neighbor-determination algorithm and for the operation-count derivations.

6.6.2 Transplanting and Re-rooting

Any Quadtree can be inserted in its entirety (as a single sub-tree) into any other Quadtree. This can be accomplished, for example, simply by discarding a leaf node in one of the trees and replacing it with the root node of the other tree. Clearly, the root node of the inserted tree has to be appropriately re-linked, and re-configured as a sub-node of the "host" tree. In addition, all attribute data in the nodes of the inserted tree that relate, for example, to depth in the tree or to spatial location, must be updated to reflect the new status of these nodes in the host tree in which they have been transplanted. For example, for Quadtrees that represent spatial sub-division processes, the geometry of the root square of the inserted tree must be re-defined if necessary so that it is identical to that of the Cartesian square of the replaced leaf node, and the geometric definition of all the sub-nodes of the transplanted Quadtree must also be re-defined accordingly.

A Quadtree can also be transplanted into another Quadtree by replacing selected leaf nodes in the host tree with individual sub-trees of the transplanted tree. An example of this, which is used in this work, is illustrated in Figure 6.12. The figure shows how the four immediate sub-nodes of the root node of the transplanted tree are used to replace the four “inner” leaf nodes of another Quadtree having a depth of 3. The larger root square is constructed to have exactly twice the dimension of the smaller root square in this case. Once the four sub-trees of root node of the original tree have been re-attached as described, the root node is eliminated. The geometric attributes in the transplanted sub-trees need no updating in this case in principle, since the Cartesian square geometries are perfectly matched. In practice, however, such an updated is carried out to ensure consistence of the geometric definitions throughout the host tree to the available arithmetic precision. The refinement levels in the in the transplanted sub-trees are also updated, and several sweeps of grid smoothing are also carried out to ensure that any refinement zones from the transplanted sub-trees extend smoothly into the outer ring of 12 coarse Cartesian squares.

As can be deduced from Figure 6.12, the effect of the transplant described above is to double the size of the Computational Region, without disturbing the original grid. This is used to enable a computation in which moving bodies or flow features are approaching or are about to cross the outer boundaries of the original Computational Region to be continued automatically without the need to halt the computation and

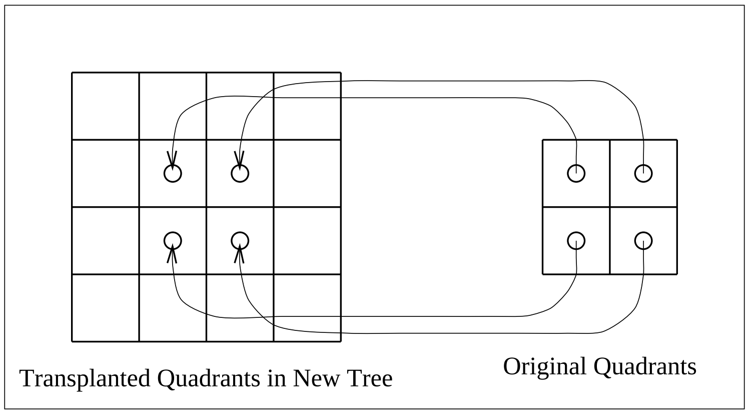

Figure 6.12: The transplanting of the four 1-major sub-trees of a Quadtree into the four "inner" 2-major sub-trees of another Quadtree.

then restart it with a new expanded grid.

The inverse of the transplanting process can be considered to be the re-rooting process, in which selected sub-trees are re-attached directly to a new root node, allowing one or more layers of interior nodes in the original Quadtree to be eliminated, and allowing the Computational Region to be contracted accordingly. A re-rooting procedure was not implemented in this work, mostly because contracting the Computational Region is neither essential nor does it enable significant savings in the total computational effort of a calculation.

6.7 Relevant Properties of Quadtree-Based Cartesian Grids

Several advantageous (and somewhat inter-related) properties and characteristics can be inferred for unstructured Cartesian grids in general, and for the Quadtree-based sub-category of these grids in particular. These properties can be inferred from the fundamental geometric characteristics of unstructured Cartesian grids (which are described in several sections in this chapter), from the fundamental properties of the Quadtree data-structure (which are described in Section 6.1), and from the properties of the grid-generation and grid-adaptation algorithms (which are described in Section 6.3 and elsewhere in this chapter). These properties are as follows:

1. Directional isotropy of the spatial resolution: The square shape of uncut computational cells, and the directional uniformity of the local spatial resolution make the resulting grids ideal for numerical solution of systems of equations that have isotropic length-scales. An example of such a system is the Euler System in smooth regions. Across discontinuities in the Euler System (which are generally curved lines in 2D, and curved planes in 3D), however, the isotropy of the spatial resolution imposes a penalty on the meshing efficiency.

2. Spatial localization of the grid generation and adaptation activity: This localization enables the attainment of arbitrary local resolution (to the allowable arithmetic precision) in any location in the Computational Region, to resolve arbitrary local geometric and solution features. The localization in this work is only constrained (slightly) by an externally-imposed requirement for grid smoothness.

3. Reliability and robustness of the grid generation and adaptation procedures: Compared to most other unstructured-grid generation techniques, the Quadtree-Based Spatial-Subdivision Algorithm is extremely robust, and can generate valid grids for highly complex geometries. If the geometry cannot be adequately resolved because, for example, of external constraints on the refinement level imposed in the grid-generation algorithm, then the geometry may be truncated or even eliminated in the grid-generation process, but that process will usually still generate a valid grid. This is an especially important advantage for 3D complex geometries. Similarly, because of the simplicity of the procedures used

to split or to consolidate squares or cubes (that is, the node refinement and coarsening operations in the Quadtree), the adaptation process is also very robust. The grid generation and adaptation processes are not only robust, but they are also computationally economical, since relatively few arithmetic and data-structural operations are required, except for cut cells (that is, for cells in the immediate vicinity of boundaries).

4. Automatability of the grid generation and adaptation operations: This enables close integration of the grid generation and adaptation operations with the flow-solution algorithm, making for highly-effective dynamic adaptation algorithms, as demonstrated in this work. A major contributor to this automatability is the requirement for relatively minimal user input for control of the grid generation and adaptation processes. Another major contributor to this automatability is the modularity of the adaptation operators, which can be invoked with any adaptation criteria, even ones that depend on the solution data, and the localization of the spatial and data-structural regions of action of these operators.

5. Efficiency in the use of computational resources: This efficiency is relative to that obtainable with other types of unstructured grids, and this efficiency is in terms of both memory (or storage space) requirements, and computational effort requirements. In regard to memory requirements, the major contributors to higher relative efficiency are the following two: (i) the high value of the ratio $\frac{n_l}{n_t}$, where n_l is the number of leaf nodes in the tree (which are the only nodes in the tree that represent active computational cells), and where n_t is the total number of nodes in the tree. As shown in Section 6.1, this ratio is approxi-

mately $\frac{3}{4}$, and since leaf nodes require at least 4 times as much storage space as interior nodes, the memory-requirement overhead from maintaining the tree-based form of the grid representation is typically less than 10% of the total memory requirement; and, (ii) the elimination of the need to store direct links to encode the neighborhood connectivity information (as explained in Section 6.6). This factor is relatively far more advantageous in 3-D, where the number of neighbors for each cell could be very large. In regard to computational effort, the major contributors to higher relative efficiency are the following two: (i) the low value of the operation count for identifying cell neighbors, amounting on average (as derived in Section 6.6) to about 2 tree ascents and 2 tree descents to locate an edge neighbor, and about 3 tree ascents and 3 tree descents to locate a vertex neighbor; and, (ii) the linearity of the grid-generation time in the number of tree nodes, and hence its linearity in the number of computational cells, as explained in Sections 6.1 and 6.3. This advantage is particularly important for grids with a large number of cells, where the form of the function of the grid-generation time versus the number of cells is often an important criterion of the overall performance of the grid-generation method. All the properties discussed in this item were explained or derived in detail, mainly in Sections 6.1, 6.3, and 6.6.

6. Separability of the representations of the boundary geometry and of the grid: This allows the grid generation process and the geometric definition process to be developed or modified largely independently of each other. This is especially valuable for moving-boundary problems, where, for example, in this work, boundary motion in the grid generation process is treated simply as an update to the geometry, and handled exclusively by refining in regions into

which the boundary moves, and coarsening in regions which are vacated by the boundary. In the geometric representation, the motion of boundaries is determined by the physics of the problem, and is completely independent of the grid and does not interact directly with any of the grid-generation operators. Cartesian methods also do not require surface-grids to be generated, since the boundary is automatically constructed by the cell-cutting procedure, and this is, again, especially advantageous for 3D problems.

7. Insensitivity of the overall solution (and the total number of cells in the grid) to the location of the far-field boundaries: This contributes to greater accuracy in applying far-field boundary conditions, and to greater automation in the grid-generation process, especially for moving-boundary problems where the trajectories of boundaries may be sought as part of the solution.

The detractions of Cartesian grids can also be inferred in the same way as the benefits listed above. The most important of these detractions are the following:

1. Non-boundary-conformality of the grid: This is probably the most significant disadvantage of Cartesian-grid methods. It results in the creation of arbitrarily-cut cells in the immediate vicinity of boundaries, and hence in arbitrary variations in cell geometry and cell n-volumes in the immediate vicinity of boundaries, even including features of severe non-orthogonality and non-smoothness. One of the specific problems caused by this non-conformality for the System of Euler Equations is the "small-cell problem", which is described in Appendix A.1. Fortunately, for systems of equations that do not contain diffusive terms, this problem and almost all other specific problems that are caused by the non-conformality can be overcome by handling the cells near boundaries in

one of a number of special ways, as described in Appendix A.2. However, for systems of equations that contain diffusive terms, such as the Navier-Stokes System, it appears impossible to simultaneously satisfy the requirements of positivity, consistence, and accuracy in the solution [85] near boundaries. Another negative consequence of the non-boundary-conformality, and a practical one, is the need for extensive computational-geometry capabilities (especially in 3D) within the grid-generation procedures, and the attendant additional computational expenses for intersection calculations and other geometric operations. However, it should be noted that regular boundaries are always of lower dimensionality than the Computational Region so the additional expense becomes asymptotically insignificant with increasing cell-count.

2. Directional isotropy of the spatial resolution: Although this property was described above as an advantage for problems with isotropic length-scales, this isotropy becomes a major detraction for systems of equations with strongly anisotropic length-scales, such as the Navier-Stokes System. In such cases, the meshes generated by Cartesian methods are typically unaffordably non-optimal.

3. Grid-dependence of the boundary geometry and the boundary discretization: This arises because the boundary faces of the grid are defined through the cell-cutting process, typically by connecting the points of intersection of the boundary with each cell by a single line segment, truncating curved boundaries and small features. This process results in the dependence of the discrete representation of the geometry of boundary surfaces on the resolution of the grid through which the boundaries pass. For moving boundaries, this problem

does not arise in this work, however, because the boundaries in this case are represented as piece-wise linear segments, and because vertices and angles in the boundary are not truncated when they fall inside cells. For boundaries that are not represented as piece-wise linear segments, the cell-cutting process could also result in large variations in resolution of the boundary features. Both of these disadvantages can be overcome to a large extent by geometric adaptation, as done in this work, for example. This grid-dependence can be viewed as a negative aspect of the advantage of not requiring a surface grid in Cartesian methods. As would be expected from the explanations in the preceding items, this grid-dependence has a limited effect on the solution quality for the Euler System, for example, but a significant effect on the solution quality for the Navier-Stokes System, for example, especially if turbulence models are used.

4. The added requirement for special data-structures and algorithms to create and maintain the Quadtree, and to perform operations in it: As described throughout this chapter, in order to take full advantage of the benefits of the Quadtree data-structure, most operations on the grid, including the gasdynamic solution and update operation, must be formulated and re-cast in terms of operators on the Quadtree data-structure.

Some of the detractions described above can be reduced, eliminated, or circumvented, while others are inherent in the Cartesian approach and cannot be eliminated without discarding the defining characteristics of this approach. Further discussion of these detractions and how they may be reduced or circumvented, as well as several relevant ideas are presented in Chapter IX.

Considering both the advantages and disadvantages discussed above, it can be stated that in most respects, unstructured, Cartesian-grid approaches, including the specific Quadtree-based approach chosen and developed in this work, are particularly well-suited for solution of the Euler System of Equations for arbitrary geometries and for moving-boundaries. It can also be observed that for the class of inviscid compressible flows, the chosen grid-generation methodology satisfies to a large extent all the relevant requirements and criteria that were identified in Chapter IV for a successful grid generation procedure.

6.8 Some Implementation Considerations

The Quadtree data-structure was described and defined in a recursive form, in Section 6.1. The Quadtree-based grid-generation and grid-adaptation algorithms were also described and formulated in recursive forms, in Section 6.3. The algorithms that perform the main operations in the Quadtree were based on the recursive form of the Quadtree data-structure, as described in Sections 6.2 and 6.6. The motives for and the advantages of adopting the recursive form were also described and implied in Sections 6.1, 6.2, and 6.3. For similar motives and advantages, a Quadtree-based, Cartesian-grid-generation algorithm is also best implemented in a strict, tree-based, recursive form, as opposed to, for example, a multi-linked-list form that retains the key elements of the tree-based form, but that only contains the leaf nodes of the corresponding tree. Such link-list forms may be simpler and easier to implement for non-adaptive, unstructured, Cartesian grids. However, for the adaptive variants of these grids, implementing such link-list forms is more difficult and more error-prone than the corresponding tree forms, and their computational performance is expected to be worse.

If, as done in this work, a recursive, tree-based form is adopted for the Quadtree-based grid-generation algorithm, then the algorithm is best implemented in a programming language that supports recursive functions and dynamic memory allocation. Good options include the Fortran 90, the C, and the C++ languages. Choosing a programming language that does not allow recursive functions will make the programming far more difficult than necessary, and choosing a programming language that does not allow dynamic memory allocation will make the implementation inefficient and cumbersome.

In this work, a multi-linked-list image of the Quadtree is used in addition to the Quadtree form, but only because the link-list form is the most convenient form for the method chosen in this work to output the solution and restart data. This link-list image is dynamically created from the Quadtree form immediately before execution of the data output operation, and then destroyed immediately after completion of the output operation.

Regardless of whether a tree-based, recursive implementation is chosen, as described in Section 6.3 and others, it is good practice for every main operation to automatically execute all its subordinate operations as built-in operations. For example, the operation of splitting a node in this work automatically executes, as subsidiary operations, all the computations associated with the node splitting itself. This includes computation of the geometric properties of the subordinate nodes, and computation of the attribute vectors of these nodes. The splitting operation also automatically executes all the operations associated with accounting for the side-effects of the splitting of a node. This includes executing all the necessary neighbor interrogations and all the necessary refinement operations to enforce the one-refinement-level-difference rule across the grid. The advantage of such an approach, which may be said to conform to the so-called object-oriented programming methodology, is that the higher-level operations, such as grid refinement and coarsening can be designed and implemented without concern for or knowledge of the details of the subsidiary operations that must be carried out during the splitting or consolidation of nodes. Thus, the node and its operators are made to contain all the information about the operations required during the splitting or consolidation, as well as all the functions that require invoking. If the node data-structure should change, or if the higher-level operations should require re-organization, the separation between the internal properties and the functionality associated with nodes will be confined (in principle, wholly) to the nodes and their built-in operations.

At several locations in this chapter, it was shown that are opportunities for altering the trade-off between the computational effort requirements and the memory space requirements. This flexibility is greatest when a tree-based implementation is chosen. For example, it was mentioned in Section 6.3 that all the relevant geometric attributes of a cut cell may be stored in an "intersection-configuration data-structure" if it is desired to avoid calculating these attributes more than once during a motion step. Similarly, in finding edge and vertex neighbors, it was chosen to retrieve the neighborhood-connectivity data by direct, immediate calculation, as described in Section 6.6, whenever a neighbor is required. An obvious alternative to this is to perform all the edge and vertex neighbor determinations once at the beginning of the calculation, and again only whenever the grid changes locally, and to store the neighborhood-connectivity data (through direct node-to-node links) for each leaf node. In this work, the re-calculation approach is chosen because it saves a significant amount of memory (about eight links per leaf node), at the expense of increasing the total computational effort, but by only a few per-cent.