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

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MURAT BULGÖK

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

OCTOBER 2005

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Kemal İder

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Yüksel Ortakaya	Prof. Dr. M. Haluk Aksel
Co-Supervisor	Supervisor

Examining Committee Members

Prof. Dr. Kahraman Albayrak	(METU, ME)	_____
Prof. Dr. M. Haluk Aksel	(METU, ME)	_____
Prof. Dr. Zafer Dursunkaya	(METU, ME)	_____
Prof. Dr. İsmail H. Tuncer	(METU, AEE)	_____
Asst. Prof. Dr. Cüneyt Sert	(METU, ME)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name :

Signature :

ABSTRACT

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

Bulgök, Murat
M. Sc., Department of Mechanical Engineering
Supervisor: Prof. Dr. M. Haluk Aksel
October 2005, 85 pages

A Cartesian method for solution of the steady two-dimensional Euler equations is produced. Dynamic data structures are used and both geometric and solution-based adaptations are applied. Solution adaptation is achieved through solution-based gradient information. The finite volume method is used with cell-centered approach. The solution is converged to a steady state by means of an approximate Riemann solver. Local time step is used for convergence acceleration. A multistage time stepping scheme is used to advance the solution in time. A number of internal and external flow problems are solved in order to demonstrate the efficiency and accuracy of the method.

Keywords: Euler equations, Cartesian method, Dynamic data structures, Finite volume method

ÖZ

EULER DENKLEMLERİNİN ÇÖZÜMÜ İÇİN ADAPTASYONA YÖNELİK KARTEZYEN AĞ ALGORİTMASI GELİŞTİRİLMESİ

Bulgök, Murat
Yüksek Lisans, Makine Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. M. Haluk Aksel
Ekim 2005, 85 sayfa

Bu tezde iki boyutlu Euler denklemlerinin kartezyen metodu ile çözümü incelenmiştir. Dinamik data yapıları kullanılmış ve geometrik ve çözüme yönelik adaptasyon uygulanmıştır. Çözüme yönelik adaptasyon için çeşitli parametrelerin akış içindeki değişimleri kullanılmıştır. Sonlu hacimler yöntemi hücre merkezli yaklaşımla birlikte uygulanmıştır. Çözüme ulaşmak için Riemann çözücüsü kullanılmıştır. Çözümü hızlandırmak için lokal zaman adımı yaklaşımından faydalanılmıştır. Metodun hassasiyetini inceleyebilmek için çeşitli iç ve dış akış problemleri çözülmüştür.

Anahtar Sözcükler: Euler denklemleri, Kartezyen metodu, Dinamik data yapıları, Sonlu hacimler yöntemi

ACKNOWLEDGEMENTS

I would like to thank my supervisor and co-supervisor, Prof. Dr. Haluk Aksel and Yüksel Ortakaya for their valuable support about Computational Fluid Dynamics. Without them I could not achieve this thesis.

I am grateful to TUSAŞ Aerospace Industry (TAI) for their financial support. I would also like to thank Hakan Tiftikçi for his invaluable support about C++ programming and computational geometry.

I give my biggest thanks to my parents, Ayşen and Oktay Bulgök, for their patience and support during my education. I owe them an enormous dept of gratitude.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS

A	area
a	speed of sound
E	internal energy per unit mass
H	enthalpy per unit mass
F	horizontal flux component vector
G	vertical flux component vector
U	conserved variable vector
u	horizontal velocity component
v	vertical velocity component
γ	specific heat ratio
ρ	density
Φ	flux vector

CHAPTER 1

INTRODUCTION

There are a huge variety of problems that are to be solved by using computational fluid dynamics. Despite the continuous increase in computational capabilities, more sophisticated techniques are needed due to the diversity and complexity of flow problems. These techniques reduce computational time and memory needed for many problems. Hence, very complex two and three-dimensional, steady and unsteady problems can be solved using the currently available computational capabilities.

It is almost impossible to handle this complexity by means of a single approach. Hence, a number of different approaches are to be used. Each one has its own advantages and disadvantages and selection of a specific one depends on the problem under consideration. The most commonly used techniques in computational fluid dynamics can be classified into two main groups: mesh generation and solution techniques.

1.1 Mesh Generation Techniques

In order to achieve spatial discretization of the governing equations, a suitable mesh is to be produced on the solution domain of the problem. In computational fluid dynamics there are three main mesh generation techniques: structured, unstructured and Cartesian methods.

1.1.1 Structured Methods

The main feature of structured methods is transformation. It means mapping or transforming from a physical domain to a computational one, which appears as a rectangle in two dimensions. All the solution process is performed in this

computational domain and the results are then remapped to the physical one for post-processing.

One of the main disadvantages of structured methods is that their use is usually restricted to simple shapes like a single airfoil. In case of a multi-element airfoil, for instance, complicated approaches like overlapping grids have to be used. Structured methods, on the other hand, require relatively simple data structures. Two or three-dimensional arrays are usually used for storing data. Consequently, structured methods are relatively faster than unstructured and Cartesian ones.

1.1.2 Unstructured Methods

Unlike structured ones, unstructured methods do not involve transformation between physical and computational domains. Instead, all the computation is performed in a physical domain only. This makes unstructured methods more suitable for complex shapes like a multi-element airfoil.

Unstructured methods, on the other hand, usually require more complicated data structures compared to structured ones. This results in more sophisticated computer programs and usually increases computational time and memory usage. Hence, unstructured methods should be preferred over structured ones in case of complex shapes only.

1.1.3 Cartesian Methods

Cartesian methods are simply a special class of unstructured methods. However, they require more complicated programs and data structures and are used for solving flows over even more complex geometries.

An important advantage of Cartesian methods is that any kind of adaptation is very easy to implement. By means of solution adaptation, for instance, a finer grid can be

obtained around a shock wave. Hence, very high accuracy levels can be obtained without increasing cell number and computational time significantly.

Structured and ordinary unstructured methods require user interfere to some extend. Cartesian methods, on the other hand, enable automatic mesh generation. The user has to define the problem only.

1.2 Solution Techniques

The two-dimensional Euler equations are governed by a system of four differential equations. Since an analytical solution technique does not exist, these equations are to be solved numerically. There are three main solution techniques in computational fluid dynamics: finite volume, finite difference and finite element. Moreover, each of these techniques can be implemented in two different ways: implicitly and explicitly.

1.2.1 Finite Difference Method

In finite difference method the differential form of the governing equations are used. The derivative terms in the equations are approximated with backward, forward or central differences. For this purpose, the solution domain is divided into a finite number of discrete points.

Finite difference method is widely used with structured mesh generation techniques. It enables faster computation compared to its counterparts. The drawback, however, is that in case of a complex geometry representation of derivative terms with finite differences may become difficult and reduce accuracy.

1.2.2 Finite Volume Method

Unlike finite difference, in finite volume method the integral forms of the governing equations are used. The solution domain is divided into a finite number of discrete areas. These are called cells. Finite volume method physically means to conserve mass, momentum and energy in each of these cells.

Finite volume method is more suitable for complex geometries and is commonly used with unstructured and Cartesian methods. In this thesis it is used for solution of the two-dimensional Euler equations in integral form.

1.2.3 Finite Element Method

As mentioned before, in finite volume method the solution domain is divided into a finite number of discrete areas. In finite element method, on the other hand, the solution domain is divided into a finite number of elements and the integral form of the governing equations is evaluated on each element by means of local and global interpolation functions.

This results in a set of algebraic equations to be solved by using various linearization, integration and acceleration methods. Some examples to these methods are Newton linearization, modified Euler method, Runge-Kutta schemes, Gauss-Seidel methods and artificial smoothing.

1.2.4 Implicit Methods

In implicit methods the system of governing differential equations are reduced to a large system of algebraic equations using finite difference, finite volume or finite element. These algebraic equations are then solved using a matrix solution technique in a coupled manner.

One of the main advantages of implicit methods is that convergence is guaranteed due to the fact that conservation of mass, momentum and energy are guaranteed in the next solution step. However, they are more complex compared to explicit methods and require more memory for storing a large number of algebraic equations.

1.2.5 Explicit Methods

The main feature of explicit methods is that mass, momentum and energy are conserved using the current values of flow variables in the current solution step. Hence, unlike implicit methods, there is no need for a large number of algebraic equations. This reduces memory requirement but does not guarantee convergence.

Explicit methods are simpler compared to implicit ones and there are special techniques for reducing the risk of divergence. One of these techniques is called multi-stage time stepping and it is used in this thesis.

1.3 Review of Literature

Cartesian methods were first proposed in 1975 as an alternative to structured and ordinary unstructured methods. The aim was to enhance automatic grid generation and facilitate solution adaptation. However, they did not take much attention until 1980s because of the relative inefficiency of commercial computers. The reason for this was that Cartesian methods require complicated data structures, which require more efficient computers.

In 1986 Clarke used Cartesian methods to solve two-dimensional steady inviscid flows over multi-element airfoils. Using structured methods to solve such problems is much more difficult and requires complicated techniques like overlapping grid generation [1]. This is followed by Mitchel, who developed alternative Cartesian grid generation techniques to solve the two-dimensional Euler equations [2].

In 1992 Tidd applied a Cartesian approach to solve flows over a complete aircraft [3]. He solved three-dimensional steady inviscid flows and achieved to reduce errors up to 1%. He also used multigrid as a convergence acceleration tool, which was almost a must for three-dimensional applications at that time due to relative computational inefficiency. Epstein later applied similar approaches to arbitrary aircraft configurations [4].

Being unstructured approaches, Cartesian methods are usually used with the finite volume method. However, Morinishi applied the finite difference method to the two-dimensional compressible Euler equations on Cartesian grids [5]. He used Runge-Kutta schemes for time stepping.

In 1993 De Zeeuw wrote a computer code to solve the two-dimensional Euler equations using a Cartesian method approach [6]. He proposed and successfully applied quadtree data structure. Linked list data structure was used in his study as well for a number of purposes. His code was applicable to both internal and external steady inviscid problems. Moreover, he applied a special multigrid technique called saw-tooth cycle successfully and he proved that any Cartesian method is very suitable for multigrid applications due to the data structures used. Hence, he achieved an approximately two-times increase in computational efficiency without increasing memory usage significantly. Moreover, the difficulties related with small cut cells were first eliminated by using a special local time step technique together with second-order spatial accuracy.

In 1994 Coirier developed a computer code to solve the two-dimensional Euler and Navier-Stokes equations [7]. He used a special hybrid grid technique for resolving boundary layers efficiently. Hence, his work was not truly about Cartesian methods due to the fact that Cartesian methods are inherently non-body-fitted. However, he applied Cartesian method techniques to hybrid grid generation for the first time.

Cartesian methods are modern approaches. Hence, solution techniques used are usually developed for more conventional approaches like structured methods. Quirk studied applicability of several solution techniques to Cartesian methods and offered a number of improvements [8].

In 1995 Aftosmis developed alternative techniques for three-dimensional Cartesian grid generation [9]. The main features were inside-outside test and polygon clipping. Afterwards, he applied Cartesian methods to three-dimensional geometries involving component-based geometries successfully [10]. Hence, the difficulties related with dirty surfaces were eliminated for the first time. This enables defining input geometries in graphical environment and transferring them for analysis using Cartesian methods, enabling flow analysis over much more complicated geometries. Before that, dirty surfaces were eliminated manually before solution process, which reduces the efficiency of any method significantly.

Coirier achieved an accuracy study of Cartesian mesh approaches for steady transonic inviscid flow problems [11]. He compared the results obtained by both uniformly and adaptively refined Cartesian mesh approaches with those by structured methods. He used some exact solutions of the steady Euler equations as well.

In 1995 Pember applied a Cartesian approach to solve unsteady compressible flows in irregular regions [12]. Since Cartesian methods are especially suitable for solving flows over complex geometries, his work was of great importance. Moreover, he proved that Cartesian methods are very efficient in solving unsteady inviscid flows.

One of the main drawbacks of any Cartesian method is that connectivity is very time-consuming to determine due to the complicated data structures used. In 1998 Khokhlov developed a special algorithm in order to eliminate this difficulty [13]. He grouped special cells together forming certain boxes and connect them together by means of extra pointers. Hence, time intervals needed for traversing the tree for

connectivity information were reduced significantly without increasing memory usage excessively.

Cartesian approaches have usually first or second-order spatial accuracy. Forrer, however, applied a special technique in 1998 for higher-order spatial accuracies in two-dimensional problems [14]. He also treated cut cells in a special manner. Using a special boundary treatment, cut cells were handled as whole cells. By this way stability problems inherent in Cartesian methods were avoided.

Wu applied an anisotrpic refinement technique to inviscid flow problems [15]. The method was originally developed in order to capture the inherent anisotropic nature of viscous boundary layers by the same person. It was slightly modified for capturing oblique and normal shock waves in two-dimensional inviscid flow problems.

Qian used a Cartesian method approach for two-fluid hydraulic flow problems [16]. He solved incompressible Euler equations for simulating compressible phenomena using the artificial compressibility factor. The computational domain encompassed both water and air regions and the interface in between was treated as a contact discontinuity.

In 2004 Hunt developed a much more sophisticated computer code to solve the three-dimensional Euler equations [17]. The data structures and main techniques were the same as those proposed by Aftosmis [9]. His code, however, was applicable to both steady and unsteady problems. He used a special technique called cell merging for efficient mesh generation in case of moving boundaries. He used it for eliminating the difficulties related with small cut cells as well due to the fact that local time step approach cannot be used in unsteady problems [6]. Moreover, he applied parallel programming to Cartesian methods successfully. Hence, he was able to solve complicated three-dimensional problems within reasonable time intervals.

Li used both isotropic and anisotropic refinement techniques for solving two-dimensional inviscid flow problems especially around strong shock waves [18]. He also applied finite difference to Cartesian grids. Hence, he proved the efficiency of Cartesian methods in capturing critical regions in a flow field by means of solution adaptation.

Unlike the remaining of the literature, French used a conservative cell-vertex Euler solver on Cartesian grids [19]. He also applied Lax-Wendroff time stepping unlike the previous studies that use multi-stage time stepping. He proved that, as modern approaches, Cartesian methods are suitable for many conventional solution techniques.

In 2004 Dadone introduced a new technique called the curvature-corrected symmetry technique (CCST) in order to handle cut cells more efficiently [20]. The method was originally used for body-fitted approaches and applied to Cartesian meshes successfully. His work was of great importance in that cut cells are the most critical part of any Cartesian method and more efficient techniques are needed to handle them.

In 2004 Keats applied anisotropic refinement to the steady and unsteady two-dimensional Euler equations [21]. The technique adapted by him was originally developed for incompressible laminar flows. By means of anisotropic refinement an excessive number of computational cells were prevented, resulting in significant computational savings. He proved that the importance of anisotropic refinement is much higher in unsteady flow problems compared to steady ones.

1.4 Present Study

In Chapter 2 a number of dynamic data structures are introduced and those that are used in this study are emphasized. In Chapter 3 the mesh generation technique is described in detail. A number of topics like line clipping and inside-outside test are

explained as well. Chapter 4 deals with adaptation, which is the most important part of Cartesian methods. Solution of the two-dimensional Euler equations is explained in Chapter 5. In Chapter 6 the basics of post-processing are given together with a number of crucial points. A number of test cases are given in Chapter 7 in order to demonstrate the efficiency and accuracy of the method. The results are then discussed in Chapter 8.

CHAPTER 2

DATA STRUCTURE

As mentioned previously, unstructured and especially Cartesian methods require sophisticated data structures. Since the number of cells cannot be predetermined, simple static data structures like two or three-dimensional arrays cannot be used efficiently. Instead, dynamic data structures like quadtree, octree, binary tree and linked list are preferred.

2.1 Quadtree Data Structure

As the name implies, in quadtree data structure cells are stored in a tree-like structure [6]. Each cell has a pointer to its parent and four pointers to its children. The cell that does not have a parent is called the root cell and its related pointer is set to zero. Similarly, some cells do not have children and their related pointers are set to zero as well. They are called leaf cells and are very important in solution and post-processing. Other cells have a parent and four children. An illustration of the quadtree data structure is given in Figure 2.1.

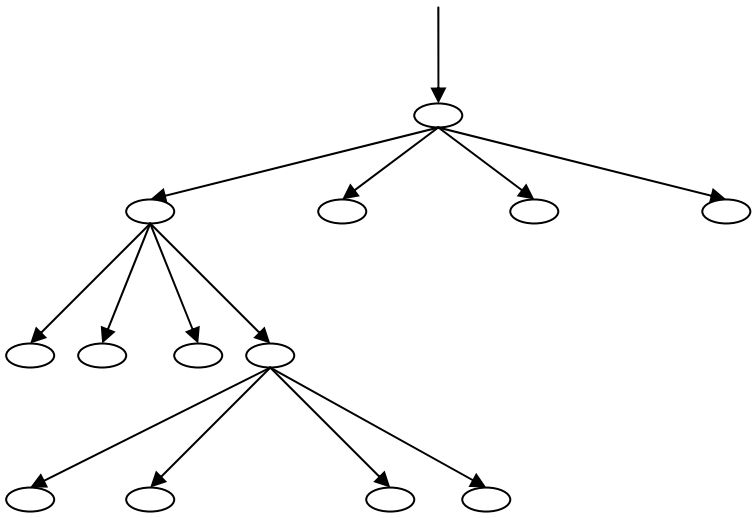


Figure 2.1 Quadtree Data Structure

Quadtree data structure is totally flexible. Any number of cells can be stored in any configuration. In an array, however, the number has to be fixed and known in advance. Hence, quadtree data structure is very suitable for Cartesian methods, which enable the analysis of flows over complex geometries.

In two and three-dimensional arrays an element in the array is referred to using its index. Hence, they are significantly faster compared to dynamic data structures. In quadtree, on the other hand, the elements are not stored in such an order and do not have an index. For this reason each element in the tree are referred to using recursive functions.

In structured methods connectivity information is apparent. In other words, the neighbors of a cell are predetermined and a special means for connectivity is not necessary. In Cartesian methods, on the other hand, connectivity information is extracted from the tree using the parent-children relationships between the cells through recursive functions.

2.2 Octree Data Structure

Octree data structure is very similar to quadtree. There is one root cell and there are leaf cells in the tree. Each element in the tree has a pointer to its parent. The only difference is that each element has eight pointers to its children [9, 17].

Octree data structure is very suitable for three-dimensional Cartesian methods applications. Each element in the tree represents a cubic region in three dimensions. Similar to quadtree data structure, connectivity information is to be obtained using recursive functions.

2.3 Binary Tree Data Structure

Binary tree data structure is very similar to quadtree and octree. The only difference is that, as the name implies, each cell has two pointers to its children. The recursive structures are the same as quadtree and octree and the connectivity information is obtained via parent-children relationships.

Binary tree data structure is very suitable for anisotropic refinement, which is necessary for efficient solution of viscous problems, which are governed by the Navier-Stokes equations. As mentioned before, quadtree and octree data structures are usually used for solution of the two and three-dimensional Euler equations, respectively.

2.4 Linked List

Another important dynamic data structure is linked list. Each element has a pointer that points to the next element in the list. In an ordinary linked list the pointer of the final element is set to zero. A special pointer is used in order to distinguish the first element from the other ones in the list. An example is given in Figure 2.2.

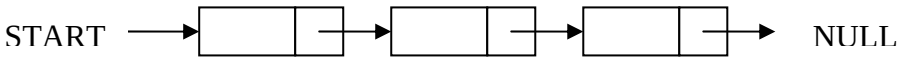


Figure 2.2 Ordinary Linked List

A special type of linked list is circular linked list. Unlike its ordinary counterpart, the pointer of the final element in the list is not set to zero. Instead, it points to the first element of the list. It can be used, for instance, for storing the points of a simple closed curve in two dimensions. An example to circular linked list is given in Figure 2.3.

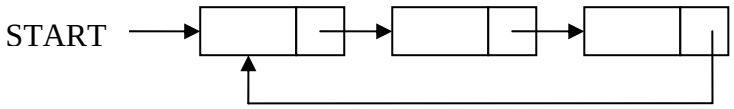


Figure 2.3 Circular Linked List

The most important drawback of an ordinary linked list is that it can be traversed in one direction only. In order to overcome this difficulty, doubly linked lists are used. In a doubly linked list each element in the list has two pointers: one for the next element and the other for the previous one. Hence, a doubly linked list can be traversed in both directions. An example to doubly linked list is given in Figure 2.4.

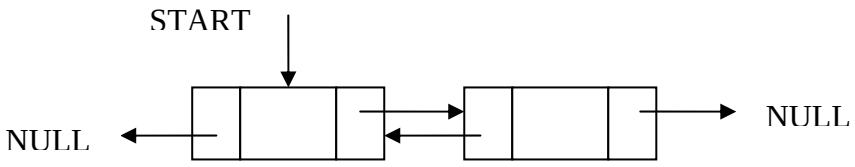


Figure 2.4 Doubly Linked List

2.5 Memory Usage

Every computer program needs a compromise between computational time and memory usage. Using the memory as much as possible reduces computational time by reducing recalculations. The limited capabilities of a computer, however, limit the number of variables that can be stored in the memory. Hence, any decision depends on the available computational capabilities.

In this thesis the integer variables stored per cell are

- 5 words – Pointers to one parent and four children cells
- 1 word – Cell type information
- 1 word – Cell level

while the real variables stored per cell are

- 4 words – Primitive variables, which are density, velocity components and pressure
- 4 words – Temporary conserved variables for multistage time stepping
- 2 words – Cell centroid
- 1 word – Cell area
- 8 words – Gradients of the primitive variables

- 4 words – Residuals for updating primitive variables
- 1 word – Local time step
- 4 words – Limiter values

CHAPTER 3

MESH GENERATION

In order to achieve spatial discretization of the Euler equations in two dimensions, a suitable mesh is required prior to the solution step. Cartesian methods are inherently unstructured and require sophisticated mesh generation techniques. Such techniques have to be suitable for generating efficient, robust and high-quality meshes around arbitrarily complex input geometries.

The technique described here is based on line clipping. No restricting assumption is made for the input geometry and it is valid for both internal and external flow problems. The mesh generation procedure is composed of three steps: determination of boundary segments, flow segments and area, centroid and type of each cell. Since line clipping and inside-outside test are used in determining boundary and flow segments, respectively, they are explained in detail as well.

3.1 Line Clipping

Given an oriented line segment and a simple closed region is space, line clipping means finding the part of the line segment residing in the given region [22, 23]. The important point is that the original orientation of the line segment is to be conserved. A general case is illustrated in Figure 3.1.

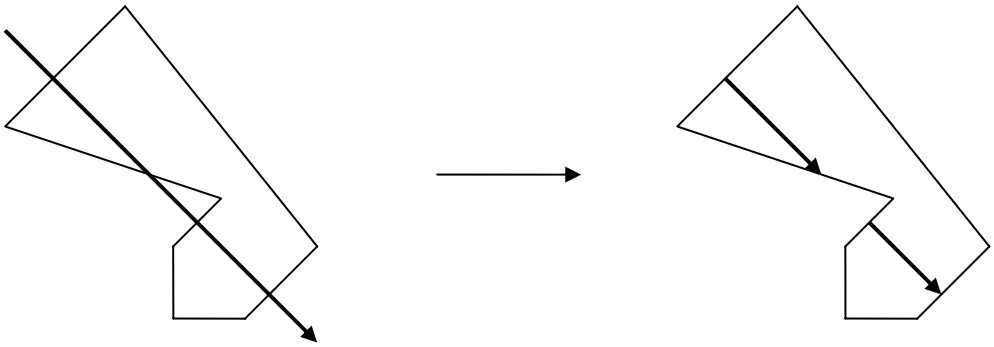


Figure 3.1 A General Line Clipping Example

In general, the simple closed region against which a line segment is clipped may be totally arbitrary in shape as illustrated in Figure 3.1. Hence, very complex configurations may result due to very complex interactions between the two objects. In Cartesian methods, however, only rectangular regions are considered as shown in Figure 3.2. This restriction enables the use of simpler algorithms and improves computational efficiency.

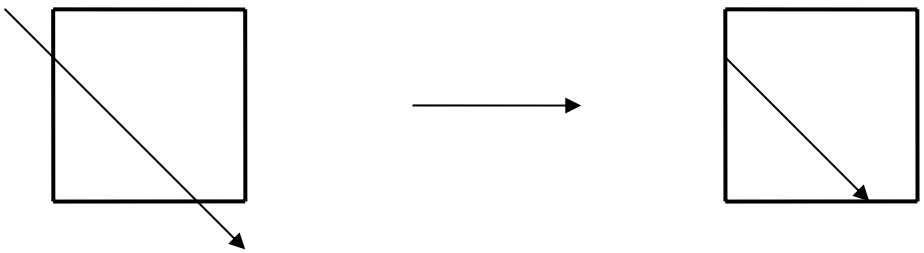


Figure 3.2 Clipping of an Oriented Line Segment against a Rectangular Region

There are a number of different methods, algorithms, in the literature for clipping arbitrary oriented line segments against rectangular regions. Some examples are Liang Barsky, Sutherland Cohen and Blinn's algorithm. Each one has its own advantages and disadvantages considering computational efficiency, simplicity and robustness. In this thesis Liang Barsky is used due to its simplicity compared to the other methods.

In the original Liang Barsky algorithm the line segments which are coincident with one of the faces of the rectangular region are considered to be inside the region. The orientation of the line segment is not taken into account in such a case. In Cartesian methods, on the other hand, each cell has a counterclockwise orientation. In other words, the faces of a cell are oriented in a counterclockwise direction with respect to the cell itself as shown in Figure 3.3.

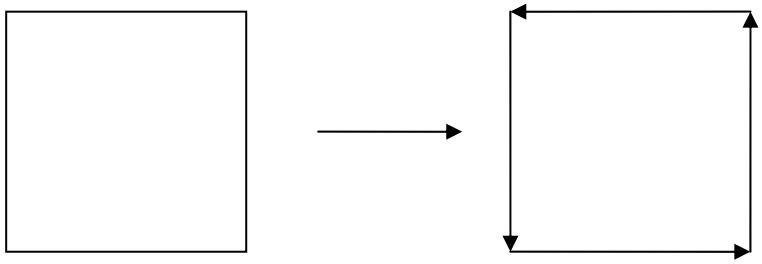


Figure 3.3 Orientation of a Cell

Hence, in this thesis a line segment which is coincident with one of the faces of a cell is considered to be inside if its orientation is the same as the orientation of the face under consideration. Otherwise it is considered to be outside and discarded. This is the only modification to the original Liang Barsky algorithm and illustrated in Figure 3.4.

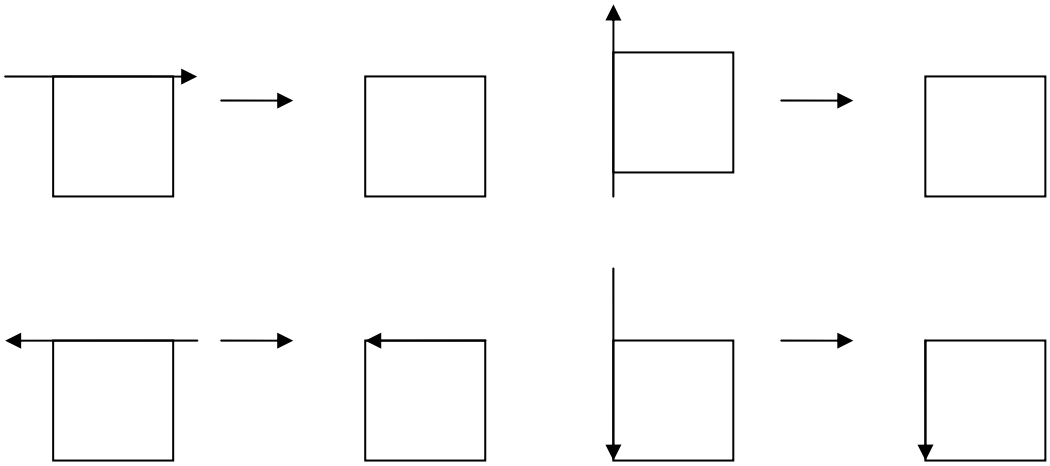


Figure 3.4 The Modification to the Original Liang-Barsky Algorithm

In some problems special interactions between rectangular regions and line segments may occur. Although such occurrences are very rare, for the sake of robustness, they are to be handled correctly. A number of them are given in Figure 3.5 together with the resulting configurations obtained by Liang Barsky. The empty boxes mean that there is not a line segment inside the cell.

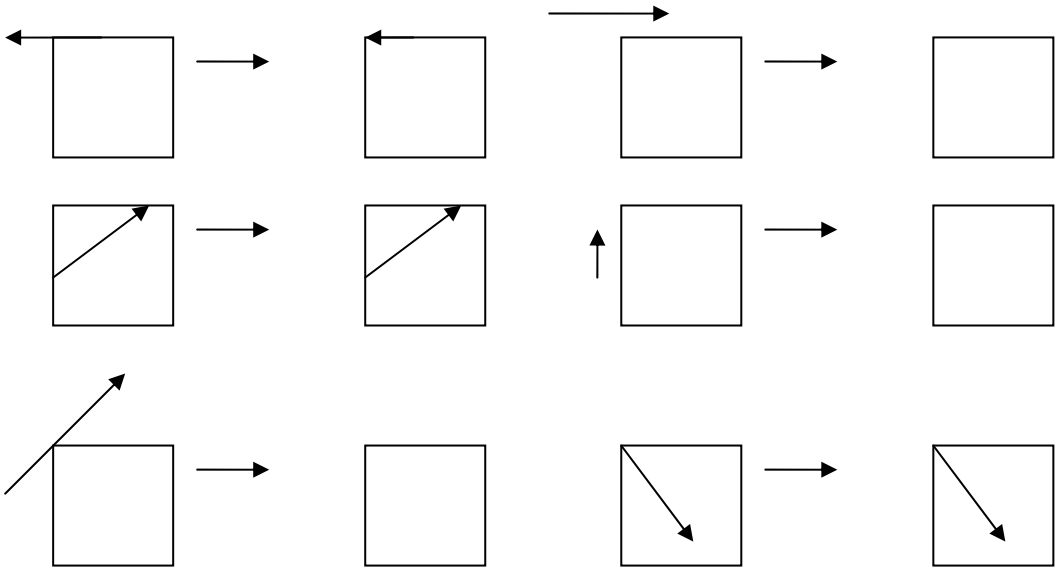


Figure 3.5 Special Cases of Line Segment-Rectangular Region Interactions

3.2 Boundary Segments

In almost all applications input geometries are defined as a combination of oriented line segments. (In three-dimensional space line segments are replaced by oriented triangles.) In Cartesian methods these line segments are allowed to intersect the cells arbitrarily. As a result, some cells contain line segments that are part of the input geometry. These oriented line segments are called boundary segments.

There are four boundary segment types: wall, far-field, inlet and outlet. Boundary segments of each cell are kept in a four-element array of pointers. Each pointer points to boundary segments of one type and it is set to zero if the cell does not have any boundary segments of that type.

The cell given in Figure 3.6, for instance, has three wall boundary segments and no far-field, inlet and outlet boundary segments. Hence, the second, third and fourth elements of the array are set to zero. The first element, on the other hand, points to a

linked list containing the three wall boundary segments of the cell. If the cell were an uncut cell, then all of the elements of the array would be set to zero.

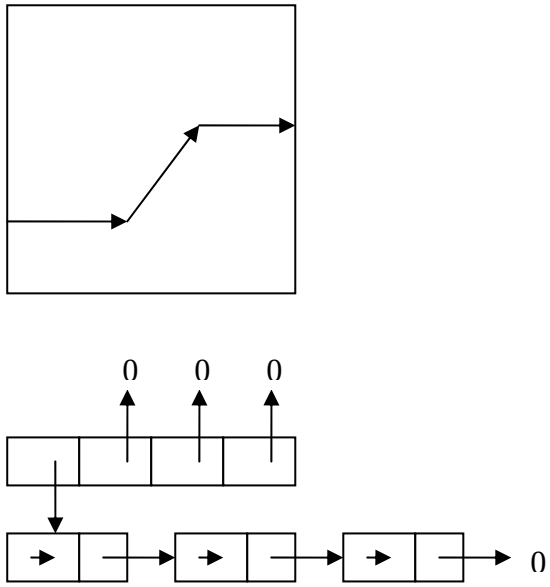


Figure 3.6 An Example for Boundary Segments

Wall boundary segments are boundary segments through which there is no fluid flow. This means that there is no mass flow rate. There is no energy flow rate as well. Considering momentum flow rate, only the pressure term is present. Far-field, inlet and outlet boundary segments, on the other hand, are boundary segments through which fluid flow is possible.

Boundary segments of a cell are determined by a four-step procedure. The procedure results in zero pointers in case of an uncut cell. Each step is explained below.

3.2.1 First Step

The first step in determining boundary segments of a cell is to copy the boundary segments of the parent cell into a separate linked list. This is a typical linked list creation process. A new linked list is created containing the same boundary segments with the original one. This is illustrated in Figure 3.7.

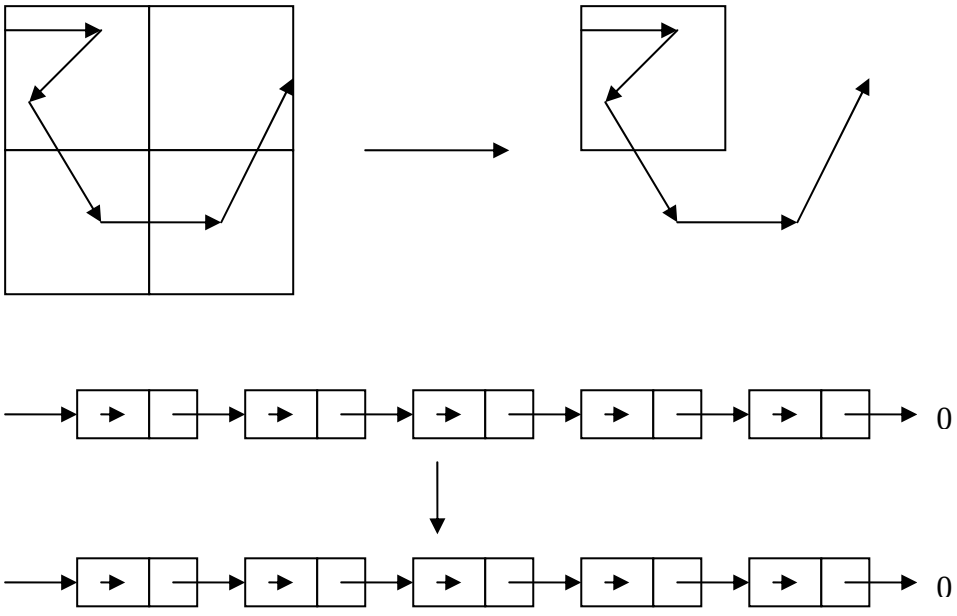


Figure 3.7 The First Step in Boundary Segment Determination

Considering only the boundary segments of the parent cell saves computational time. The input geometry is composed of much more oriented line segments. Since the boundary segments of a cell are to be the boundary segments of the parent cell at the same time, there is no need to consider all the line segments comprising the input geometry.

3.2.2 Second Step

In the second step all the line segments in the copy created in the first step are clipped against the cell itself using Liang Barsky algorithm explained previously. In other words, the parts of these line segments residing in the cell under consideration are determined. The line segments that are outside are flagged as well as shown in Figure 3.8.

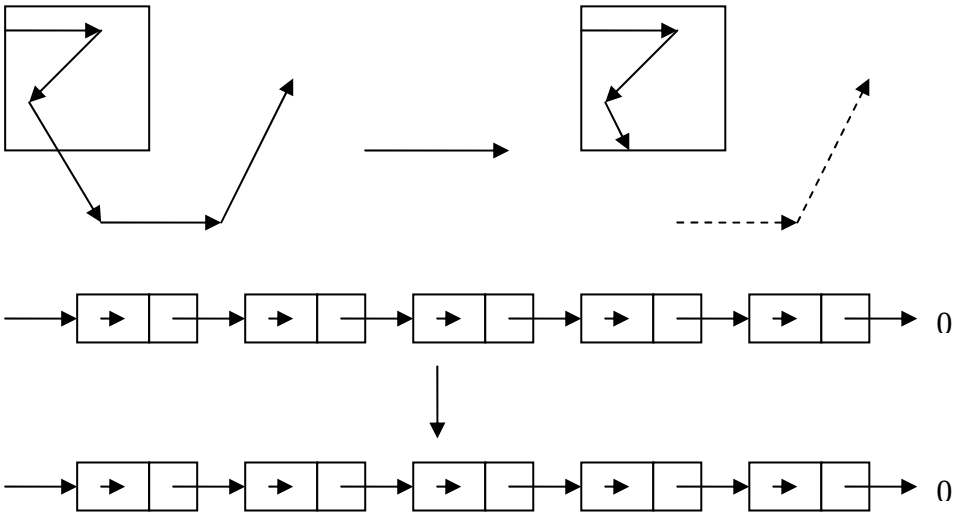


Figure 3.8 The Second Step in Boundary Segment Determination

3.2.3 Third Step

In this step the line segments that are outside and flagged in the second step are deleted from the linked list. This is a typical linked list deletion process. An illustration is given in Figure 3.9.

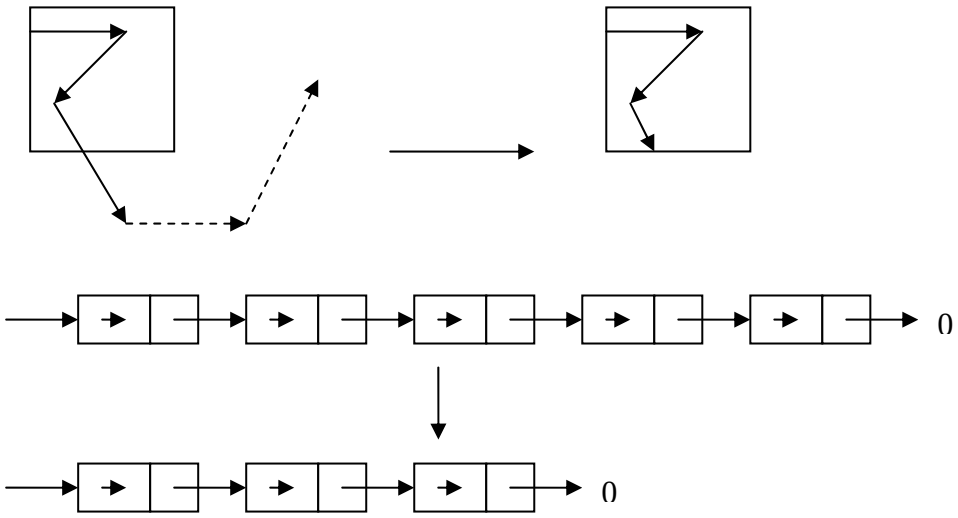


Figure 3.9 The Third Step in Boundary Segment Determination

The important point is that the line segments in the final linked list do not have to be in a specific order like clockwise or counterclockwise. Instead of this, they are stored in a totally arbitrary manner. The orientation of each line segment, on the other hand, is important and each one is to be oriented in a counterclockwise direction with respect to the cell itself.

3.3 Inside-Outside Test

Inside-outside test means determining whether a point is inside or outside a simple closed region. In general, a combination of such regions may be under consideration as well. Inside-outside test is almost a must in Cartesian methods and an efficient method is to be used for an efficient and robust mesh generator.

There are several methods in the literature used for inside-outside test. The most well-known and widely used ones are ray casting and winding number methods. The latter is used for all cases in this thesis. Since all methods may cause problems while

dealing with points that are exactly on or very close to the boundary of the region, a check should by applied for such cases in advance for the sake of robustness.

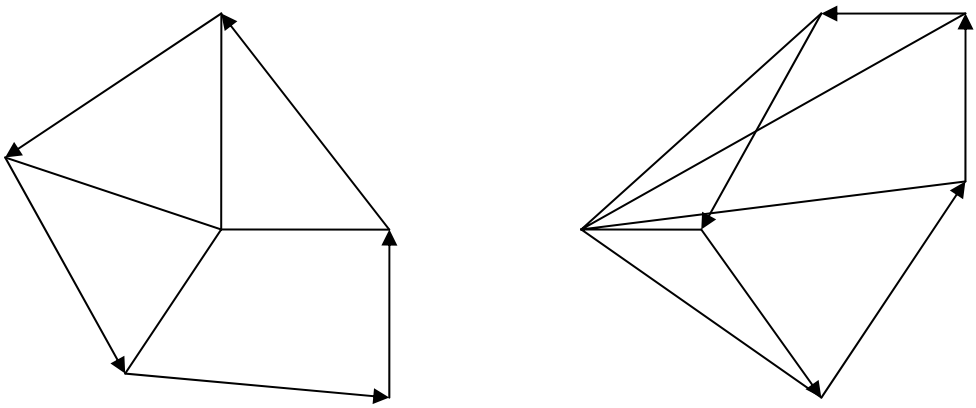
3.3.1 Ray Casting Method

One of the most commonly used methods for inside-outside test is ray casting. The main idea is to send a ray from the point under consideration and count the number of intersection points with the boundary of the region. If the number of intersection points is odd then the point is inside the region. Otherwise it is outside.

Ray casting method is considered to be more efficient compared to winding number due to the fact that it does not include complex mathematical operations like square root and inverse trigonometric functions. Moreover, it is readily extendable to three-dimensional applications. In ray casting method, however, there are some special cases which are to be taken into account for the sake of robustness. The ray, for instance, may be tangent to or coincident with the boundary of the region.

3.3.2 Winding Number Method

A less efficient but more robust way of inside-outside test is winding number or solid angle method. It involves a number of angle calculations and then summing them up in an arbitrary manner. If the summation is zero then the point is outside the region. If it is 2π then the point is inside. Round-off effects, however, are to be taken into account for the sake of robustness. An illustration is given in Figure 3.10.



The point is inside. The point is outside.

Figure 3.10 Winding Number Method

Like ray casting, winding number can be used for combinations of simple closed regions and in three-dimensional applications as well. Unlike ray casting, however, winding number requires the line segments comprising the boundary or boundaries of the region to be in a counterclockwise orientation. If there are holes in the region then the boundaries of these holes have to be in a clockwise orientation. An example is given in Figure 3.11.

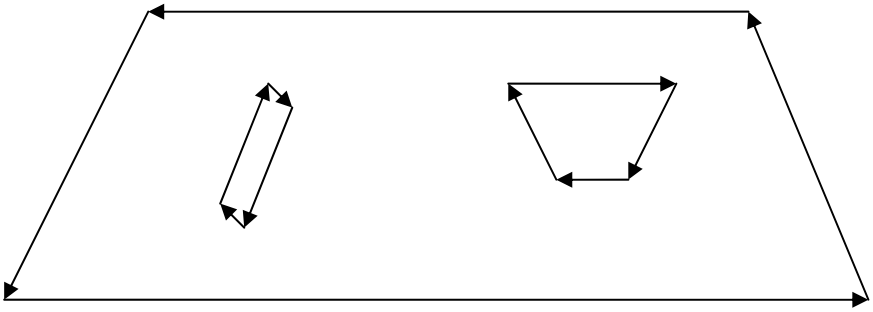


Figure 3.11 Oriented Boundaries Representing a Simple Closed Region

3.4 Flow Segments

Flow segments are line segments through which both mass and energy flows are possible together with momentum flows in each main direction. All computational cells have at least two flow segments and cells that do not have any flow segments are classified as unused cells.

Although boundary segments of a cell may be totally arbitrary in orientation, flow segments are to be either horizontal or vertical. No cell may have a flow segment that is inclined with respect to main directions. This is one of the basic properties of Cartesian methods that make them easier and more efficient compared to their counterparts.

Flow segments of each cell are kept in a four-element array of pointers. Each pointer points to flow segments associated with one of the faces of the cell. A pointer is set to zero if there are not any flow segments in the linked list.

In order to determine flow segments of a cell its boundary segments are to be determined in advance using the procedure explained previously. Flow segments are then determined using a four-step procedure.

3.4.1 First Step

The first step in determining the flow segments of a cell associated with one of its faces is to produce a linked list containing the two endpoints of the face and all the endpoints of the boundary segments of the cell that are on this face. This is a typical linked list creation process. An illustration is given in Figure 3.12.

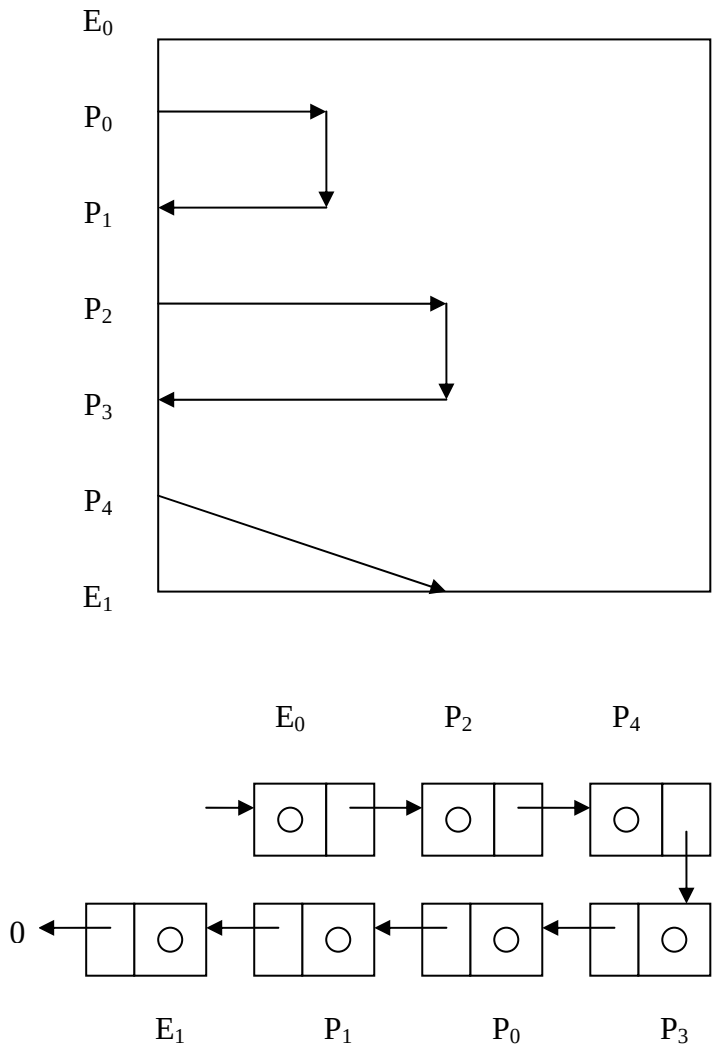


Figure 3.12 The First Step in Flow Segment Determination

Note that, since the boundary segments of a cell do not have to be in a specific order, the points in the linked list do not have to be in a specific order as well. This is obvious in Figure 3.12.

3.4.2 Second Step

In the second step the linked list created in the first step is sorted in order to obtain a linked list that is suitable for the third step. The sorting criterion depends on which face of the cell is under consideration. For the first face, for instance, the points are sorted with respect to increasing apses value. An illustration is given in Figure 3.13.

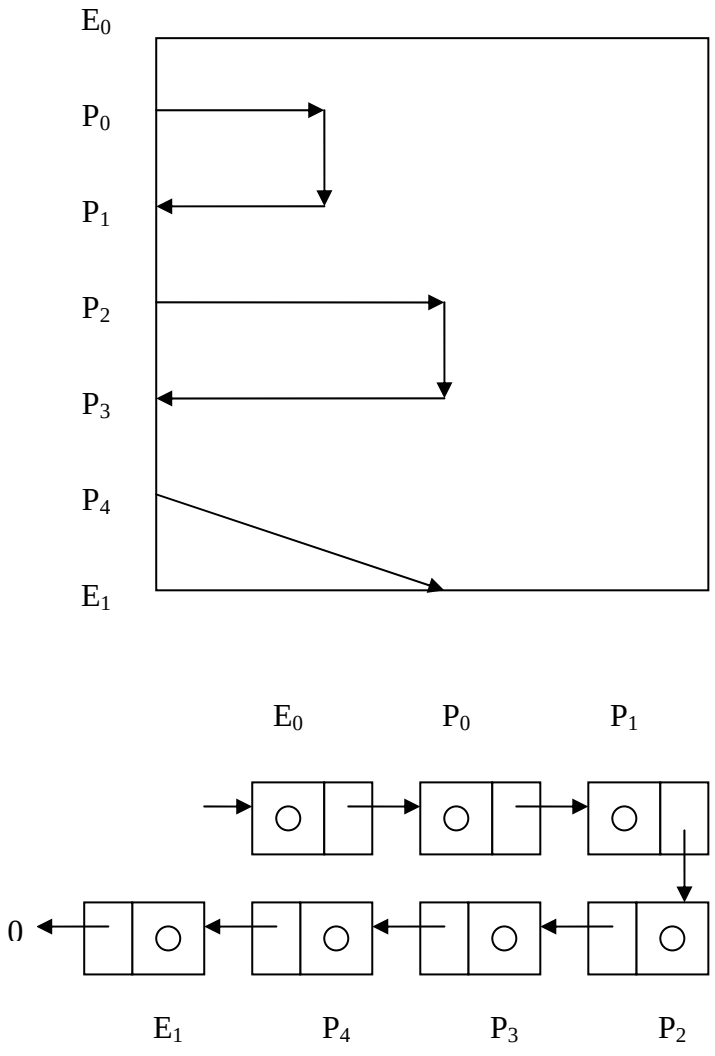


Figure 3.13 The Second Step in Flow Segment Determination

3.4.3 Third Step

Sorting produces a linked list containing point elements that are in a specific order. The next step is to create a linked list of line segments whose end points are the elements of the previous linked list. An ordinary linked list creation algorithm is used for this purpose together with the fact that there must be at least two points in the previous list. This means that there is to be at least one line segment in the new linked list. This step is illustrated in Figure 3.14.

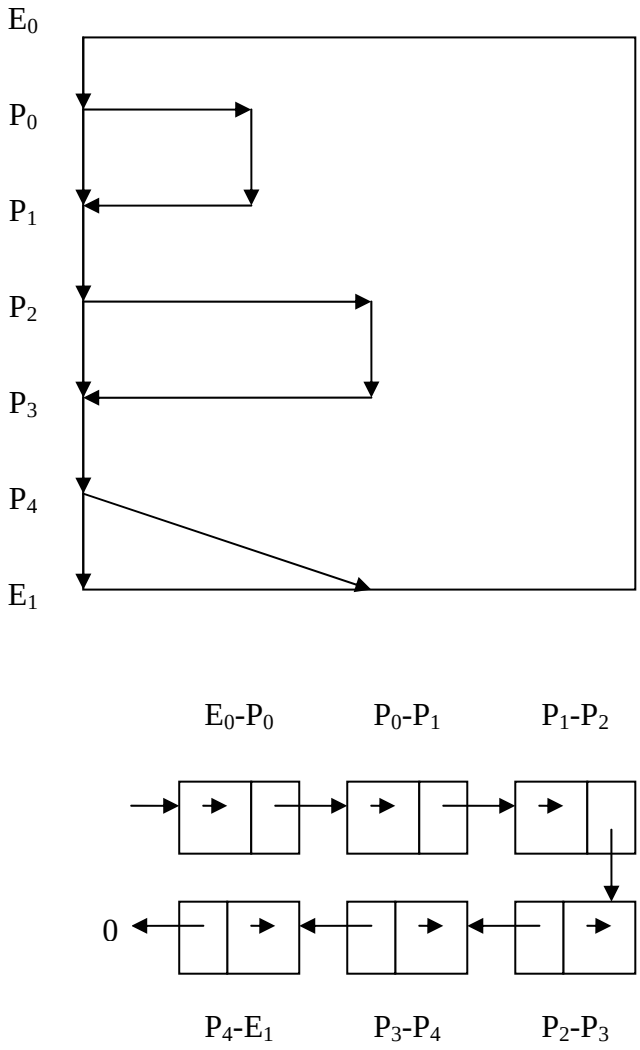


Figure 3.14 The Third Step in Flow Segment Determination

3.4.4 Fourth Step

The linked list created in the third step involves line segments. In the fourth and final step each of these line segments are checked for whether they are inside or outside the flow field. For this task, first, the midpoint of each line segment is determined. If the point is outside the flow field then the line segment is deleted

from the list. This test is performed by winding number method and is illustrated in Figure 3.15.

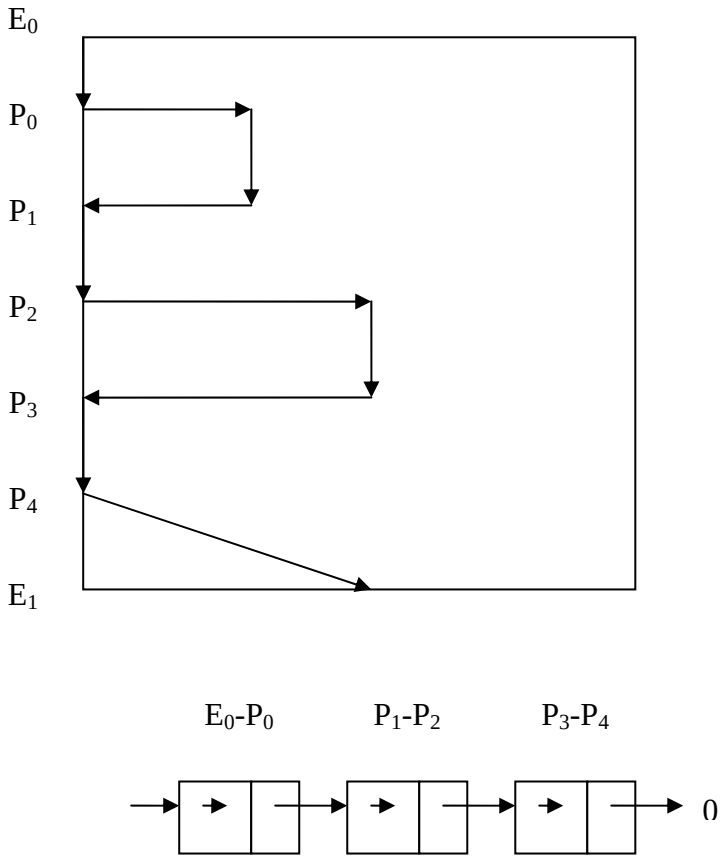


Figure 3.15 The Fourth Step in Flow Segment Determination

3.5 Cell Centroid, Area and Type

As mentioned before, boundary and flow segments of a cell do not have to be in a specific order. However, they are bound to form a simple closed curve or a combination of such curves in two dimensions. Hence, once the boundary and flow segments of a cell are determined, its centroid and area can be calculated using

simple geometric identities. In addition, cell type can be given considering the existence and type of flow and boundary segments.

Cell centroid is used in case of second-order spatial accuracy. Cell area is necessary for residual calculation and cell type is mainly used for increasing the efficiency of recursive functions.

3.6 Split Cells

One of the most important problems associated with Cartesian methods is split cells. Unlike the usual case, flow and boundary segments of a split cell form at least two simple closed curves as shown in Figure 3.16.

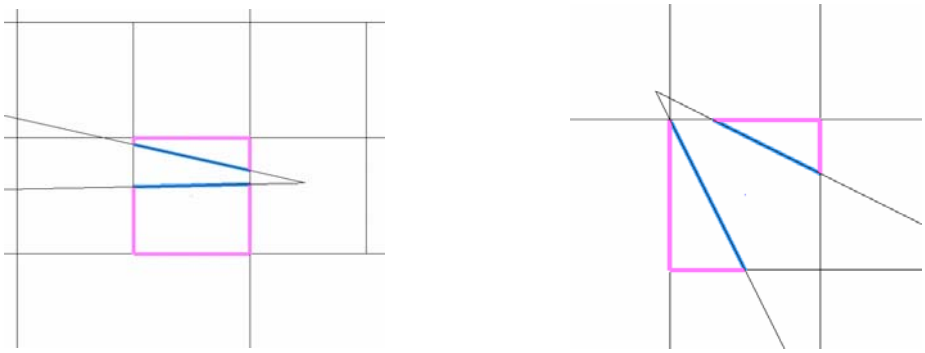


Figure 3.16 Split Cells

The most logical, robust and accurate way of handling split cells is to consider these curves being totally independent from each other. This approach is, however, inefficient in terms of computational time and requires much more complicated data structures.

For this reason, as an alternative approach, these curves are considered as a whole with only one set of primitive variables, gradients, frozen limiters and so on. The

inaccuracy introduced with this approach is assumed to be negligible due to the fact that split cells are cut cells and very small in size compared to the remaining of the grid.

In order to reduce the number and size of split cells, a special recursive function is used. Following the generation of an initial mesh, the whole tree is traversed and the split cells are divided until the remaining ones are sufficiently small. By this way the effect of split cells to the overall accuracy of the solution is minimized.

3.7 Machine Zero Effect

Computers work with a finite number of digits. Some use a very high number but no one uses infinitely many digits. For the sake of robustness this should be taken into account as well.

The usual way of taking machine zero effect into account is to define a global variable representing the minimum real value that the program can handle. All the values whose absolute values are less than this value are taken as zero. Hence, this global variable represents the maximum resolution of the program. In this thesis a value between 10^{-10} and 10^{-14} is used.

CHAPTER 4

ADAPTATION

In computational fluid dynamics high accuracy levels are usually required. One way of acquiring this is to use a fine mesh. By this way, the solution domain is divided into smaller regions, namely cells, and more accurate results can be obtained.

Using a fine mesh, however, increases computational time and memory usage significantly. Hence, the time required for the solution of a problem may increase to unacceptable values. An alternative is to use a finer mesh wherever necessary. For this purpose critical regions in the domain are identified where a finer mesh is required. This technique is known as adaptation.

One of the most important features of Cartesian methods is that any kind of adaptation can easily be applied to any input geometry. Hence, relatively high accuracy levels can be obtained with a relatively low number of cells. This saves computational time and memory significantly.

There are two kinds of adaptation: geometric and solution. After defining the input geometry, they are applied for obtaining a suitable mesh, or successive meshes, for the solution of a problem. Each of these steps is explained below.

4.1 Defining the Input Geometry

In Cartesian methods the whole procedure begins with defining the input geometry or flow domain. A suitable mesh is then produced on this domain and the problem is solved on this mesh. Finally, the results are post-processed using some means.

The flow domain is defined by specifying the boundaries. Four different boundary types are used in this thesis: wall, far-field, inlet and outlet. Wall and far-field

boundaries are used for external flows. Wall, inlet and outlet boundaries are used for internal flows.

Boundaries are specified as a combination of line segments. All boundaries are oriented in counterclockwise direction. The holes, on the other hand, are given in clockwise orientation. An illustration is given in Figure 4.1.

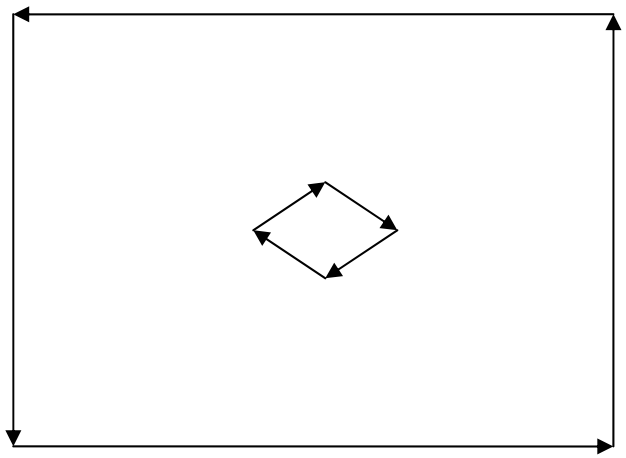


Figure 4.1 Boundary Orientation

The line segments comprising the boundaries of the flow domain are read from the input files and are given as the boundary segments to the root cell. In other words, the boundary segments of the root cell are given explicitly. For other cells, on the other hand, the boundary segments are determined using the procedure explained in Chapter 3. An illustration is given in Figure 4.2.

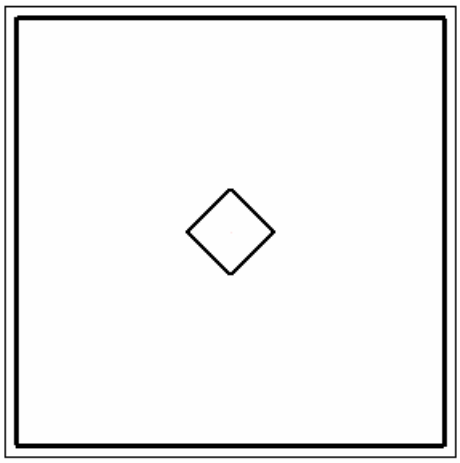


Figure 4.2 The Input Geometry

The flow segments of the root cell are set to zero by default. The reason for this is that the root cell has no neighbor and it can not have any flow segment. This completes defining the input geometry. The next step is producing a suitable initial mesh on this flow domain by means of geometric adaptation.

4.2 Geometric Adaptation

After defining the input geometry, geometric adaptation is applied in order to obtain a suitable initial mesh for the solution of a given problem. There are three types of geometric adaptation: all-cell, cut-cell and curvature. The amount of each type of adaptation is determined by the user.

4.2.1 All-Cell Adaptation

In all-cell adaptation all the leaf cells are divided without using any specific criterion. The aim is to obtain sufficiently small cells before applying other types of adaptation. If all-cell adaptation is not applied, the root cell itself is used as the initial mesh. In Figure 4.3 two levels of all-cell adaptation applied to the simple input geometry given in Figure 4.2 is shown.

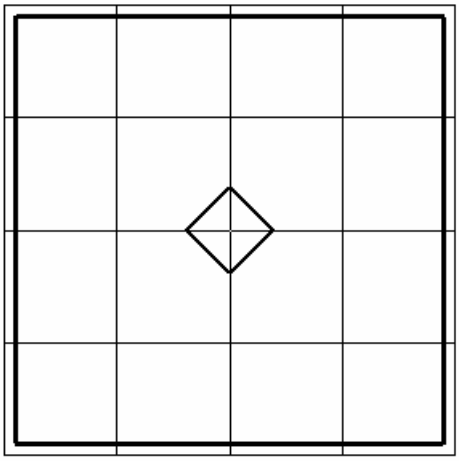


Figure 4.3 All-Cell Adaptation

During all cell adaptation the overall cell number increases exponentially. Hence, it is considered to be very costly compared to other types of geometric adaptation and excessive use of it should be avoided. For most cases two or three steps are found to give superior results.

4.2.2 Cut-Cell Adaptation

After all-cell adaptation cut-cell adaptation is applied in order to obtain smaller cells at solid boundaries. For this purpose, cells that are cut by the input geometry are determined and divided until a sufficiently fine mesh is obtained. Cut-cell adaptation is illustrated in Figure 4.4.

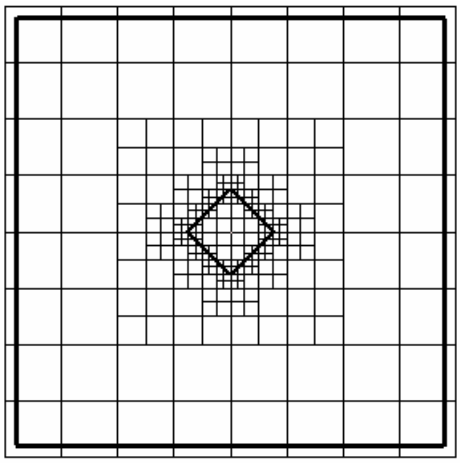


Figure 4.4 Cut-Cell Adaptation

4.2.3 Curvature Adaptation

There may be some regions of the input geometry with a high curvature such as the tip or tail of an airfoil. Such regions are expected to be more critical and require careful attention compared to those with a low curvature. Hence, high curvature regions are determined and resolved better by using smaller cells. This is achieved by curvature adaptation. In Figure 4.5, 3 levels of curvature adaptation are applied to the mesh given in Figure 4.4.

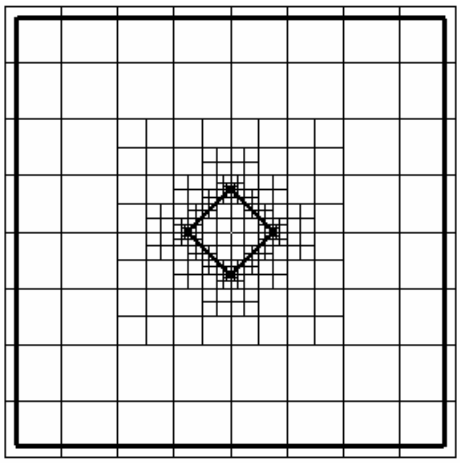


Figure 4.5 Curvature Adaptation

Curvature adaptation is implemented in two steps. In the first step the angles between the successive wall boundary segments of each cut cell are calculated. The cell has to contain at least two wall boundary segments. Uncut cells and cut cells which have just one wall boundary segment are discarded.

The maximum of these angles is then compared with a user-specified threshold value. If it is higher than this value then the cell is divided. In this thesis the threshold value of 20^{o} is used for all cases.

In some cases one of the endpoints of the line segments comprising the input geometry may coincide with one of the faces of a cut cell. For the sake of robustness these special cases are to be taken into account as well. For this purpose, the same threshold value is used and if the angle is higher than this value then both cells are divided.

4.3 Solution adaptation

By defining the input geometry and applying geometric adaptation, namely all-cell, cut-cell and curvature adaptations, a suitable initial mesh is produced for the

solution of the problem as given in Figure 4.5. The first solution is then obtained using this initial mesh.

In each problem, however, there may be critical regions where the flow variables undergo abrupt changes. Shock waves, contact layers and shear layers are examples of such features frequently encountered in inviscid and viscous problems. In order to obtain higher accuracy levels these features have to be resolved sufficiently.

One way of obtaining enough resolution is to use a much finer mesh. Such an approach, however, resolves unnecessary regions as well and increases the total number of computational cells significantly and unnecessarily. Hence, computational time and memory usage may increase to intolerable levels.

Hence, techniques for resolving the critical regions without increasing the total number of cells significantly are needed in order to find a compromise between accuracy and computational capabilities. Such techniques are generally called solution adaptation.

In conventional approaches solution adaptation is performed manually by the user at the beginning. The critical regions are estimated and a finer mesh is used in these regions. This method is, however, not efficient due to the fact that the critical regions in a flow field cannot be predetermined.

Hence, solution adaptation is most efficiently applied in an automated manner. The problem is solved on an initial mesh in order to determine the critical regions in the domain. The mesh is then refined in order to obtain a finer mesh in these critical regions.

There are several ways to distinguish such critical regions in a flow field. In this thesis the curl and divergence of velocity are used together. Curl of velocity is used for resolving shear layers while divergence of velocity is used for resolving shock

waves. Using two different criteria at the same time gives superior results compared to using a single one.

For each cell the curl τ_c and divergence τ_d of velocity are computed in weighted form as

$$\tau_c = \left|\nabla \times \vec{U}\right| l^{\frac{r+1}{r}} \qquad \tau_d = \left|\nabla \cdot \vec{U}\right| l^{\frac{r+1}{r}} \tag{4.1}$$

where l is the length scale for the cell. In this thesis r is always taken as 2. Once the curl and divergence of velocity are computed for all cells, the standard deviations about zero are computed as

$$\sigma_c = \sqrt{\frac{\sum_{i=1}^{n} \tau_c^{\,2}}{n}} \qquad \sigma_d = \sqrt{\frac{\sum_{i=1}^{n} \tau_d^{\,2}}{n}} \tag{4.2}$$

where n is the total number of cells. Hence, the conditions for refinement and coarsening may be written as

i. If $\sigma_c < \tau_c$ or $\sigma_d < \tau_d$ the cell is flagged for refinement.

ii. If $\tau_c < \sigma_c/10$ and $\tau_d < \sigma_d/10$ the cell is flagged for coarsening.

In order to demonstrate the effect of solution adaptation on the accuracy of the results, a supersonic flow over a wedge is solved. The free-stream Mach number is 2. The initial mesh is produced by means of 2 levels of all-cell, 7 levels of cut-cell and 3 levels of curvature adaptation. 4 levels of solution adaptation are then applied successively on this initial mesh. The results are given in Figure 4.6, 4.7, 4.8, 4.9 and 4.10. It is apparent from the figures that critical features like the double-reflecting shock and the expansion fan are resolved better increasing the level of solution adaptation.

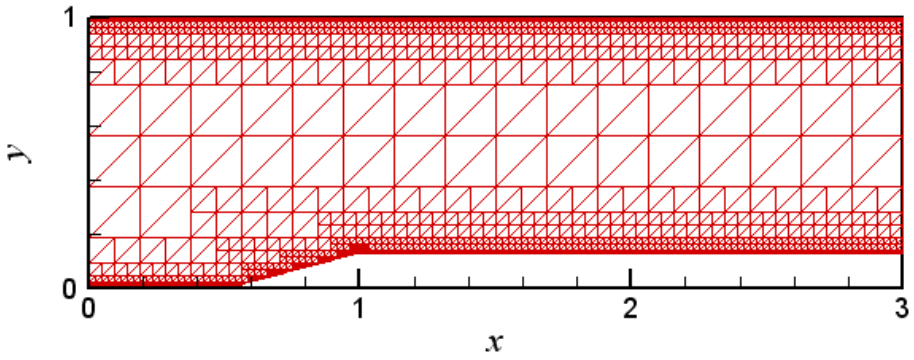


Figure 4.6a Mesh without Any Solution Adaptation

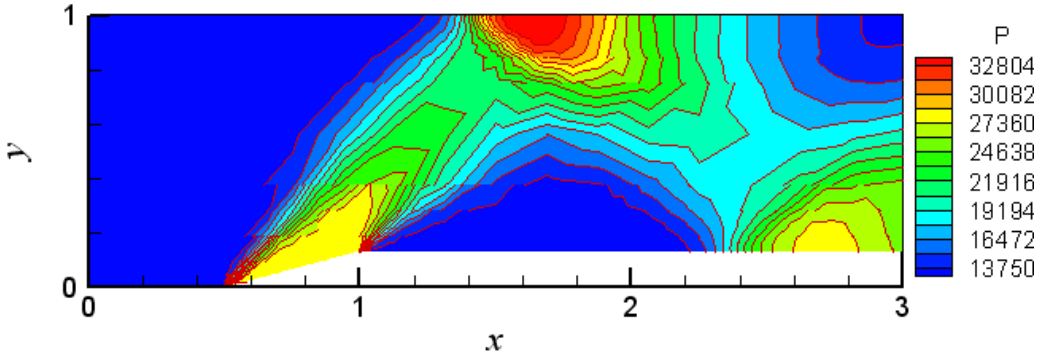


Figure 4.6b Pressure Contours without Any Solution Adaptation

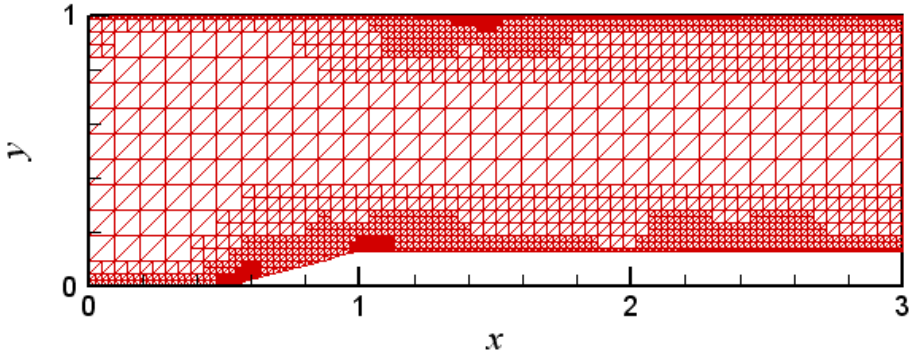


Figure 4.7a Mesh after 1 Level of Solution Adaptation

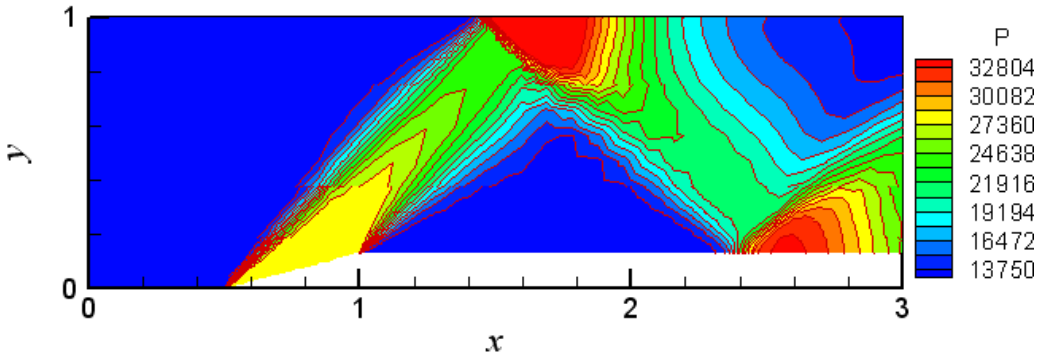


Figure 4.7b Pressure Contours after 1 Level of Solution Adaptation

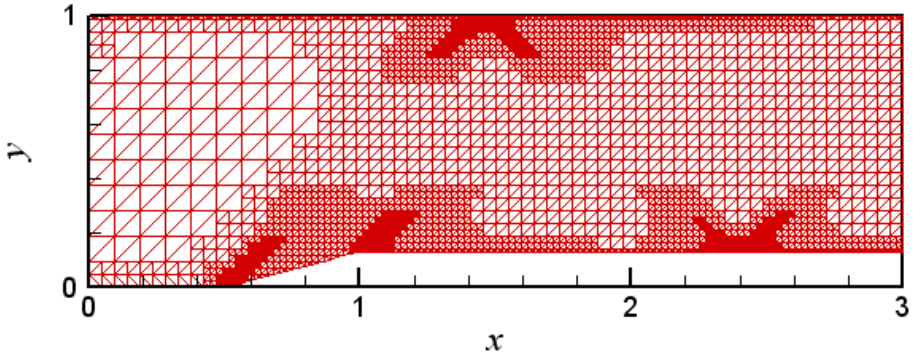


Figure 4.8a Mesh after 2 Levels of Solution Adaptation

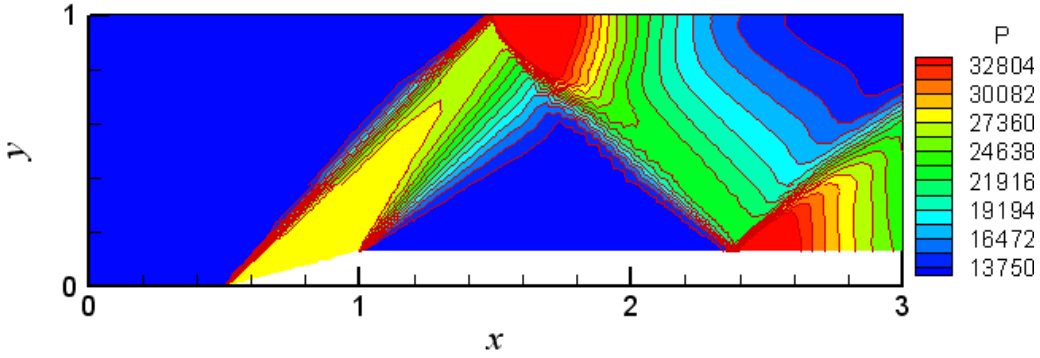


Figure 4.8b Pressure Contours after 2 Levels of Solution Adaptation

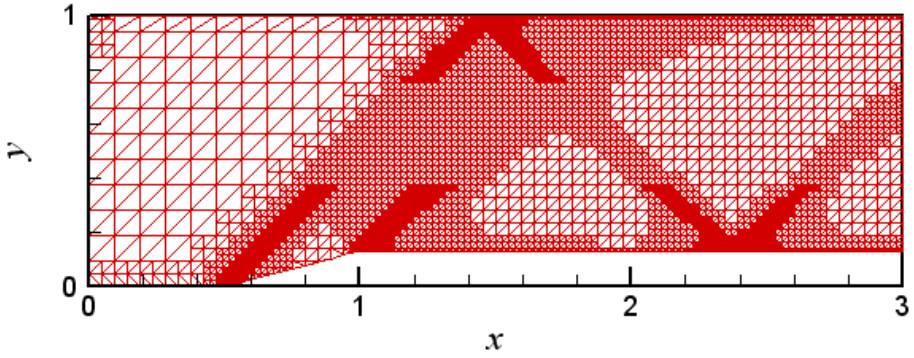


Figure 4.9a Mesh after 3 Levels of Solution Adaptation

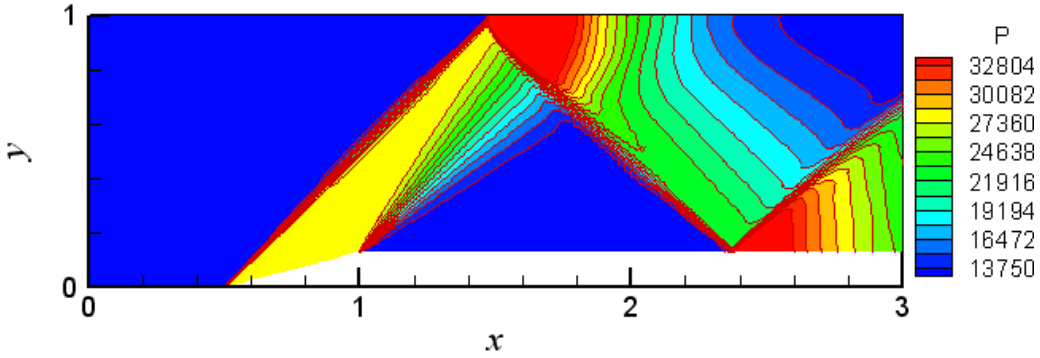


Figure 4.9b Pressure Contours after 3 Levels of Solution Adaptation

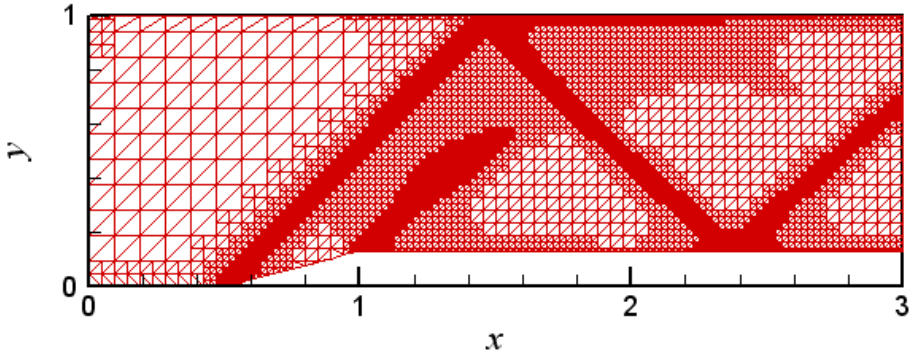


Figure 4.10a Mesh after 4 Levels of Solution Adaptation

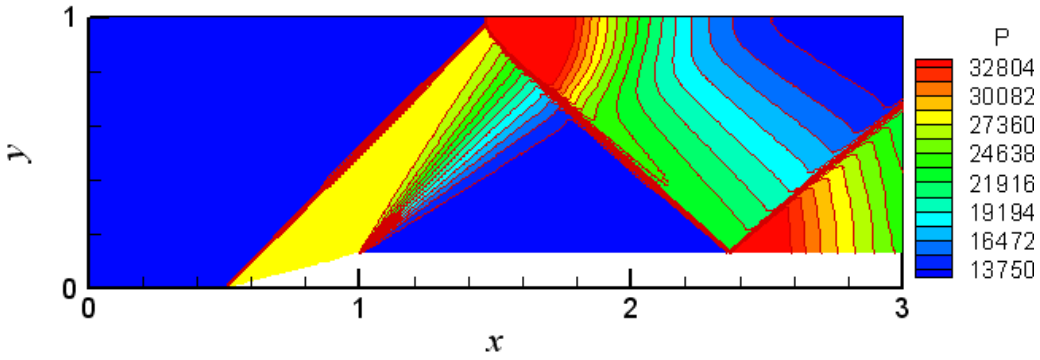


Figure 4.10b Pressure Contours after 4 Levels of Solution Adaptation

CHAPTER 5

SOLVER

As mentioned before, the first step in Cartesian methods is to define the input geometry. A suitable initial mesh is then produced by means of all-cell, cut-cell and curvature adaptations. The next step is to solve the governing equations on this mesh.

In this thesis, cell-centered approach is used in order to discretize the finite volume form of the two-dimensional Euler equations. Flux difference splitting and flux vector splitting methods are used for calculating mass, momentum and energy flow rates through the cell faces.

For first-order spatial accuracy, the cell-centered values of the primitive variables are used directly in flux calculation. For second-order spatial accuracy, on the other hand, these values are reconstructed at the midpoints of the cell faces. For this purpose, the gradients of the primitive variables and limiting are used.

In this thesis, steady solutions of the two-dimensional Euler equations are sought. Hence, high convergence rates are usually required. In order to achieve this, local time step technique is used.

5.1 Gradient Estimation

Gradients of the primitive variables are used for two main purposes: second-order spatial accuracy and solution adaptation. For second-order spatial accuracy, primitive variables at cell centroids are reconstructed at the midpoints of the cell faces using gradients and limiting. For solution adaptation, cells are flagged for refinement or coarsening with respect to a criterion or some criteria involving gradients of the primitive variables.

There are two methods that are widely used for gradient estimation: least squares and path integral. Although the least squares method has some advantages, the path integral method is used for all cases in this thesis due to its computational efficiency.

5.1.1 Path Integral Method

One of the methods that are used for estimating gradients of the primitive variables in a cell is the path integral method. As the name implies, the main idea is to construct a suitable path around the cell under consideration. Divergence theorem is then applied on this path.

In order to construct such a path, the neighboring cells are to be collected in a counterclockwise order. For an uncut cell, which is the most common case in Cartesian methods, the procedure is relatively simple. In Figure 5.1a a normal path is illustrated, which is the simplest case. In Figure 5.1b, on the other hand, a path that is altered due to the differences between the length scales of the neighboring cells is given.

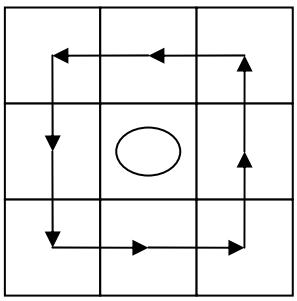


Figure 5.1a Normal Path for an Uncut Cell

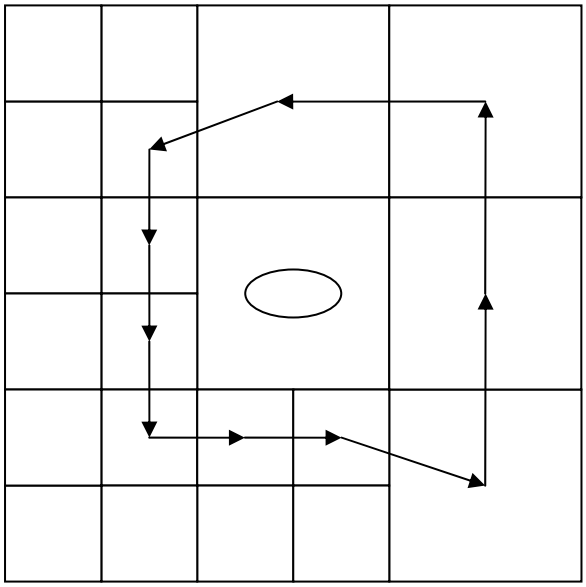


Figure 5.1b Altered Path for an Uncut Cell

In case of a cut cell the procedure is more complicated. The main difficulty is that, for a cut cell, at least one of neighboring cells is bound to not exist. In such a case, the cell itself is to be used instead of the missing cells. This is illustrated in Figure 5.2. In the figure the dotted cells are the missing neighbors.

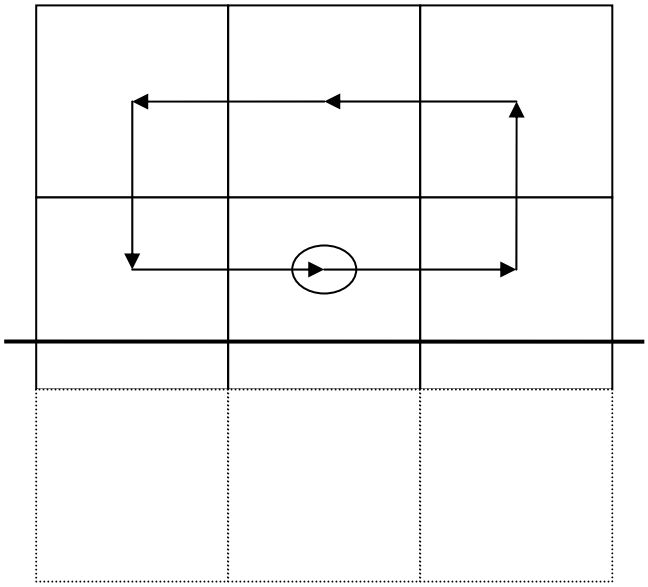


Figure 5.2 Path for a Cut Cell

As mentioned before, once a suitable path is constructed, divergence theorem is applied on this path [24]. This gives the gradient of a quantity ∇W_k in a cell as

$$\nabla W_k = \frac{1}{A_\Omega} \int_{\partial\Omega} W_k \vec{n}\, d\ell \tag{5.1}$$

where A_Ω represents the area enclosed by the path of integration $\partial\Omega$. It can be calculated by triangulating the area and summing the areas of these triangles.

5.1.2 Least Squares Method

Another way of estimating gradients of the primitive variables is the least squares method [6]. It relies on the solution of a weighted least squares system. Such a system being solved for the gradient of *u* is

$$L\nabla u = f \qquad L = \begin{pmatrix} \omega_1 \Delta x_1 & \omega_1 \Delta y_1 \\ . & . \\ . & . \\ . & . \\ \omega_N \Delta x_N & \omega_N \Delta y_N \end{pmatrix} \qquad f = \begin{pmatrix} \omega_1 \Delta u_1 \\ . \\ . \\ . \\ \omega_N \Delta u_N \end{pmatrix} \tag{5.2}$$

where

$$\begin{aligned} \Delta x_i &= x_i - x_0 \\ \Delta y_i &= y_i - y_0 \\ \Delta u_i &= u_i - u_0 \end{aligned} \tag{5.3}$$

where the subscript *i* represents a neighboring cell and the subscript *0* represents the cell in which the gradient of *u* is to be calculated. *x* and *y* represents the coordinates of the cell centroids. *N* is the total number of cells that are used for gradient estimation. The result may be written as

$$\begin{aligned} \nabla_x u &= \frac{1}{c_1 c_3 - c_2^{\,2}} \sum_{i=1}^{N} (c_3 \Delta x_i - c_2 \Delta y_i) \Delta u_i \\ \nabla_y u &= \frac{1}{c_1 c_3 - c_2^{\,2}} \sum_{i=1}^{N} (c_1 \Delta y_i - c_2 \Delta x_i) \Delta u_i \end{aligned} \tag{5.4}$$

where c_1, c_2 and c_3 are defined as

$$c_1 = \sum_{i=1}^{N} \Delta x_i^{\,2} \qquad c_2 = \sum_{i=1}^{N} \Delta x_i \Delta y_i \qquad c_3 = \sum_{i=1}^{N} \Delta y_i^{\,2} \tag{5.5}$$

Similar to the path integral method, the least squares method require collecting the neighboring cells. The same procedure explained for the path integral method may be used for the least squares method as well. However, the latter does not require the neighboring cells to be collected in a counterclockwise order.

5.2 Limiting

Gradient estimation is based on the fact that flow variables change continuously in the region of estimation. Near solid boundaries or shock waves, however, flow variables may change discontinuously and gradient estimation may give results that are totally inaccurate.

In order to eliminate this drawback and obtain accurate, bounded values of the primitive variables, the estimated gradient values are limited before being used in calculations [6]. The limiter used is a diffusive limiter and is given as

$$\phi = \min\left\{ \begin{array}{l} 1 \\ \min_k \left(\dfrac{W_k{}^c - \max_{\text{path}}(W_k)}{W_k{}^c - \max_{\text{cell}}(W_k)} \right) \\ \min_k \left(\dfrac{W_k{}^c - \min_{\text{path}}(W_k)}{W_k{}^c - \min_{\text{cell}}(W_k)} \right) \end{array} \right. \tag{5.6}$$

In this thesis four limiters are used: one for each primitive variable. This gives more accurate results due to the fact that each primitive variable has its own limiter and excessive limiting of the gradients is eliminated. The drawback, however, is that it requires four real variables and expensive in terms of memory.

Alternatively, only the minimum of these limiters can be stored and used for all primitive variables. This method is less expensive in terms of memory and more conservative compared to the previous one. The drawback, however, is that the effect of second-order spatial accuracy decreases. It may be more suitable for three-dimensional applications.

5.3 Flux Formulation

As mentioned before, flow segments are line segments through which mass, momentum and energy flow are possible. Estimation of these flow rates requires a suitable flux formulation. Flux formulation described here corresponds to solution of the Euler equations. In this thesis two well-known flux formulation methods are examined: Roe's flux difference splitting and Van Leer's flux vector splitting.

5.3.1 Euler Equations

The governing equations for two-dimensional inviscid flows in the absence of body forces may be written in differential form as

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0 \tag{5.7}$$

where

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{pmatrix} \mathbf{G} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vH \end{pmatrix} \tag{5.8}$$

where ρ is the density, p is the thermodynamic pressure, u and v are the velocity components in the x and y directions, E is the total energy per unit mass and H is the total enthalpy per unit mass [24, 25, 26, 27, 28]. Taking the integral of Equation (5.7) over a cell with an area of A leads to

$$\int_A \frac{\partial \mathbf{U}}{\partial t} dA + \int_A \frac{\partial \mathbf{F}}{\partial x} dA + \int_A \frac{\partial \mathbf{G}}{\partial y} dA = 0 \tag{5.9}$$

which may be written as

$$\frac{\partial \mathbf{U}}{\partial t}\int_A dA + \int_A \frac{\partial \mathbf{F}}{\partial x}dxdy + \int_A \frac{\partial \mathbf{G}}{\partial y}dydx = 0 \tag{5.10}$$

due to the fact that the area of the cell is constant. Using the divergence theorem for the second and third terms of the left hand side of Equation (5.10) leads to

$$\frac{\partial \mathbf{U}}{\partial t}A + \int_S \mathbf{F}dy - \int_S \mathbf{G}dx = 0\,. \tag{5.11}$$

where S represents the circumference of the cell. With a second-order accuracy, the second and third terms of Equation (5.11) may be combined and written as

$$\frac{\partial \mathbf{U}}{\partial t}A + \sum_{\text{faces}}\left(\mathbf{F}\cdot\Delta y - \mathbf{G}\cdot\Delta x\right) = 0 \tag{5.12}$$

which may be arranged as

$$\frac{\partial \mathbf{U}}{\partial t} = -\frac{1}{A}\sum_{\text{faces}}\left(\mathbf{F}\cdot\Delta y - \mathbf{G}\cdot\Delta x\right) \tag{5.13}$$

where Δx and Δy are the changes in the x and y coordinates along a face of the cell. Defining the face length and the normal and tangential velocity components as

$$\Delta s = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{5.14}$$

and

$$u_n = \frac{u\Delta y - v\Delta x}{\Delta s}$$

$$u_t = \frac{u\Delta x + v\Delta y}{\Delta s} \tag{5.15}$$

the flow rates through a face may be written as

$$\vec{F}\,\Delta y - \vec{G}\,\Delta x = \begin{pmatrix} \rho u_n \\ \rho u_n u + p\dfrac{\Delta y}{\Delta s} \\ \rho u_n v - p\dfrac{\Delta x}{\Delta s} \\ \rho u_n H \end{pmatrix} \Delta s = \mathbf{\Phi}\Delta s \tag{5.16}$$

where $\mathbf{\Phi}$ represents the fluxes through a face. Hence, Equation (5.13) may be written as

$$\frac{\partial \mathbf{U}}{\partial t} = -\frac{1}{A}\sum_{faces} \mathbf{\Phi}\Delta s\,. \tag{5.17}$$

Defining the residuals of a cell as

$$\mathbf{Res}(\mathbf{U}) = -\sum_{faces} \mathbf{\Phi}\Delta s \tag{5.18}$$

Equation (5.17) may be written as

$$\frac{\partial \mathbf{U}}{\partial t} = -\frac{1}{A}\mathbf{Res}(\mathbf{U})\,. \tag{5.19}$$

Calculating the flux vector $\mathbf{\Phi}$ is the most critical part of the solution process. In this thesis it is performed by means of flux splitting. The residuals are then calculated for each cell and integrated in time. These are explained below.

5.3.2 Roe's Flux Difference Splitting

Roe's flux difference splitting is one of the most accurate methods for flux calculation. The flux vector **Φ** is defined as

$$\mathbf{\Phi}(\mathbf{U}_L, \mathbf{U}_R) = \frac{1}{2}\left[\mathbf{\Phi}(\mathbf{U}_L) + \mathbf{\Phi}(\mathbf{U}_R)\right] - \frac{1}{2}\sum_{k=1}^{4}\left|\overline{a_k}\right|^{*} \Delta V_k \, \overline{\mathbf{R}_k} \tag{5.20}$$

with

$$\bar{\mathbf{a}} = \begin{pmatrix} \overline{u_n} - \bar{c} \\ \overline{u_n} \\ \overline{u_n} \\ \overline{u_n} + \bar{c} \end{pmatrix} \quad \Delta\mathbf{V} = \begin{pmatrix} \dfrac{\Delta p - \bar{\rho}\bar{c}\Delta u_n}{2\bar{c}^2} \\ \dfrac{\rho \Delta u_t}{\bar{c}} \\ \Delta\rho - \dfrac{\Delta p}{\bar{c}^2} \\ \dfrac{\Delta p + \bar{\rho}\bar{c}\Delta u_n}{2\bar{c}^2} \end{pmatrix} \quad \overline{\mathbf{R}} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ \bar{u} - \bar{c}\dfrac{\Delta y}{\Delta s} & \bar{c}\dfrac{\Delta x}{\Delta s} & \bar{u} & \bar{u} + \bar{c}\dfrac{\Delta y}{\Delta s} \\ \bar{v} + \bar{c}\dfrac{\Delta x}{\Delta s} & \bar{c}\dfrac{\Delta y}{\Delta s} & \bar{v} & \bar{v} - \bar{c}\dfrac{\Delta x}{\Delta s} \\ \overline{H} - \overline{u_n}\bar{c} & \overline{u_t}\bar{c} & \dfrac{\bar{u}^2 + \bar{v}^2}{2} & \overline{H} + \overline{u_n}\bar{c} \end{pmatrix} \tag{5.21}$$

where

$$\bar{\rho} = \sqrt{\rho_L \rho_R} \qquad \bar{u} = \frac{\sqrt{\rho_L}u_L + \sqrt{\rho_R}u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \qquad \bar{v} = \frac{\sqrt{\rho_L}v_L + \sqrt{\rho_R}v_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}$$

$$\overline{H} = \frac{\sqrt{\rho_L}H_L + \sqrt{\rho_R}H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{5.22}$$

and

$$\overline{u_n} = \frac{\bar{u}\Delta y - \bar{v}\Delta x}{\Delta s} \qquad \overline{u_t} = \frac{\bar{u}\Delta x + \bar{v}\Delta y}{\Delta s} \qquad \bar{c} = \sqrt{(\gamma - 1)\left(\overline{H} - \frac{\bar{u}^2 + \bar{v}^2}{2}\right)} \tag{5.23}$$

where γ is the specific heat ratio. The summation term in Equation (5.20) provides the upwind character and stabilizes the scheme [25].

In order to prevent expansion shocks, which may occur computationally but are totally unphysical, an entropy fix is imposed [11]. Hence, the entropy controlling term $\left|\overline{a_k}\right|$ is replaced by a smoothed value $\left|\overline{a_k}\right|^*$ when k takes the values of 1 and 4. This leads to

$$\left|\overline{a_k}\right|^* = \begin{cases} \left|\overline{a_k}\right| & \left|\overline{a_k}\right| \geq \frac{1}{2}\delta a_k \\ \frac{\overline{a_k}^2}{\delta a_k} + \frac{1}{4}\delta a_k & \left|\overline{a_k}\right| \leq \frac{1}{2}\delta a_k \end{cases} \qquad \delta a_k = \max\left(4\Delta a_k, 0\right) \qquad \Delta a_k = a_{kR} - a_{kL}\,. \tag{5.24}$$

5.3.3 Van Leer's Flux Vector Splitting

Van Leer's flux vector splitting is one of the alternatives to Roe's flux difference splitting. Although proved to be more dissipative, it is faster and more robust [25]. The flux vector $\mathbf{\Phi}$ is defined as

$$\mathbf{\Phi} = \mathbf{T}^{-1}\left[\mathbf{H}^{+}\left(\mathbf{U}_L\right) + \mathbf{H}^{-}\left(\mathbf{U}_R\right)\right] \tag{5.25}$$

where

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \Delta y/\Delta s & -\Delta x/\Delta s & 0 \\ 0 & \Delta x/\Delta s & \Delta y/\Delta s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{H} = \begin{pmatrix} \rho \cdot u_n \\ \rho \cdot u^2{}_n + P \\ \rho \cdot u_n \cdot u_t \\ \rho \cdot u_n \cdot H \end{pmatrix}. \tag{5.26}$$

The split fluxes $\mathbf{H}^{+}$ and $\mathbf{H}^{-}$ are defined for various Mach numbers as

$$M_n \geq +1 \quad \mathbf{H}^+ = \mathbf{H} \quad \mathbf{H}^- = 0$$
$$M_n \leq -1 \quad \mathbf{H}^- = \mathbf{H} \quad \mathbf{H}^+ = 0$$
$$|M_n| \leq 1 \quad \mathbf{H}^\pm = \begin{pmatrix} h_1^\pm \\ h_2^\pm \\ h_3^\pm \\ h_3^\pm \end{pmatrix} = \begin{pmatrix} \pm \rho c (M_n \pm 1)^2 / 4 \\ h_1^\pm c \left[(\gamma - 1) M_n \pm 2 \right] / \gamma \\ h_1^\pm c M_t \\ h_1^\pm c^2 \left(\left[(\gamma - 1) M_n \pm 2 \right]^2 / \left[2(\gamma^2 - 1) \right] + M_t^2 / 2 \right) \end{pmatrix} \tag{5.27}$$

where M_n and M_t are the Mach numbers calculated using the normal and tangential velocity components, respectively. Unlike Roe's flux difference splitting, Van Leer's flux vector splitting does not have an entropy controlling term. Hence, in some cases expansion shocks may not be eliminated. This is one of the reasons why it is less accurate compared to Roe's flux difference splitting.

5.4 Time Stepping

The residuals for each computational cell are calculated as described above. These residual values are then used to update the primitive variables by means of multi-stage time stepping. Local time step is used in order to increase convergence rate. Since the Euler equations include a time derivative and a number of spatial derivatives, a suitable initial guess and a number of boundary conditions are necessary as well.

5.4.1 Local Time Step

For steady problems, local time step concept can be used in order to increase convergence rate. The main idea is that each cell has its own time step depending on the size and flow properties. Hence, using higher time step values increases convergence rate.

There are different local time step definitions. In this thesis it is defined as

$$\Delta t = \frac{A/S}{c + \sqrt{u^2 + v^2}} \tag{5.28}$$

where A is the area of the cell, S is the circumference of the cell, u and v are the velocity components and c is the speed of sound. In order to prevent divergence and for the sake of robustness, in some problems local time step values should be multiplied with a suitable safety factor as well.

Note that local time step concept cannot be used for unsteady problems. In unsteady problems all the cells are to have the same time step value. It can be found by calculating the local time step value for each cell and then using the minimum of them.

5.4.2 Multistage Time Stepping

The general m-stage time stepping scheme is defined as

$$\begin{aligned} \mathbf{U}^{(0)} &= \mathbf{U}^{n} \\ \mathbf{U}^{(k)} &= \mathbf{U}^{(0)} + \upsilon \frac{\alpha_k \Delta t}{A} \mathbf{Res}\left(\mathbf{U}^{(k-1)}\right) \\ \mathbf{U}^{n+1} &= \mathbf{U}^{(m)} \end{aligned} \tag{5.29}$$

where ν and α_k are chosen for optimal smoothing [6]. A number of different choices are given for first and second-order spatial accuracies in Table 5.1 and 5.2.

Table 5.1 Multi-Stage Coefficients for a First-Order Spatial Discretization

	Stages				
	2	3	4	5	6
ν	1.0	1.5	2.0	2.5	3.0
α_1	0.3333	0.1481	0.0833	0.0533	0.0370
α_2	1.0000	0.4000	0.2069	0.1263	0.0851
α_3		1.0000	0.4265	0.2375	0.1521
α_4			1.0000	0.4414	0.2562
α_5				1.0000	0.4512
α_6					1.0000

Table 5.2 Multi-Stage Coefficients for a Second-Order Spatial Discretization

	Stages				
	2	3	4	5	6
ν	0.4693	0.6936	0.9214	1.1508	1.3805
α_1	0.4242	0.1918	0.1084	0.0695	0.0482
α_2	1.0000	0.4929	0.2602	0.1602	0.1085
α_3		1.0000	0.5052	0.2898	0.1885
α_4			1.0000	0.5060	0.3050
α_5				1.0000	0.5063
α_6					1.0000

5.4.3 Convergence Criteria

In order to stop the solution process a suitable convergence criterion is needed. Although there are several choices for this purpose, in this thesis the residual of the

first primitive variable, which is density, is used. The solution process is stopped when the residual of density is reduced to a user-specified value.

In most cases the average value of the residuals is used. Since it is bound to be smaller compared to the maximum one, the number of iterations and computational time needed is reduced. Another approach is to use the maximum residual value in the domain. Although this approach takes considerably longer computational time, it is more conservative and gives more accurate results.

5.4.4 Initial Guess and Boundary Conditions

Due to the time derivative term in the two-dimensional Euler equations, a proper initial guess is needed. For external flow problems the far-field boundary conditions are given to the computational cells as the initial guess. The solution is then converged to steady state. For internal flow problems the inlet boundary conditions are used as the initial guess.

The user, however, does not have to give these initial guesses. The program enables other choices as well. For an internal flow problem, for instance, the average of the inlet and outlet boundary conditions may be used as the initial guess. This may increase the convergence rate due to the fact that the Euler equations are a system of non-linear partial differential equations and highly sensitive to the initial guess.

Due to the spatial derivatives, in addition to a proper initial guess, proper boundary conditions are needed as well. For flow-through boundaries, which are inlet, outlet and far-field boundaries, the boundary conditions are specified in the ghost cells just outside the computational domain and the Riemann problem is then solved yielding the boundary fluxes and flow rates. As mentioned before, Roe’s flux difference splitting is used for this purpose in this thesis.

Since the Euler equations represent inviscid flows, the flow velocity is not necessarily zero on the wall. Instead, only the normal component of the velocity is set to zero. The tangential component does not need to be zero. The physical meaning of this is that mass and energy flow rates through the wall are set to zero. Considering the momentum flow rate, only the pressure term is kept. One alternative to this approach is to use ghost cells just outside the walls. Density, pressure and the tangential component of the velocity are taken to be the same. The normal component of the velocity is taken just to be the opposite.

CHAPTER 6

POST-PROCESSING

Mesh generator and solver are two important parts of Cartesian methods. However, an efficient mesh generator and an accurate solver are not enough for an efficient and high-quality solution of a given problem. The process of presenting the final results, which is known as post-processing, is very important as well.

For presenting the final results as accurately as possible, gradients of the primitive variables are used. This means that the primitive variables at the desired positions are obtained by reconstructing the cell-centered values using the gradients. In order to obtain accurate, bounded values limiting is applied as well.

6.1 Contour Drawing

As mentioned before, the primitive variables are stored at cell centers. However, contour drawing requires nodal values of the primitive variables. Hence, cell-centered values of the primitive variables are reconstructed in order to find nodal values. This is illustrated in Figure 6.1. Other variables like Mach number and total pressure are then calculated from nodal values of the primitive variables.

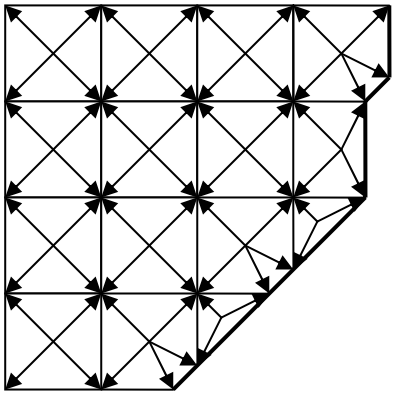


Figure 6.1 Reconstruction to Find Nodal Values from Cell-Centered Ones

6.2 Triangulation

As mentioned before, in Cartesian methods, cells do not have a unique shape. This may cause difficulties in post-processing. Hence, after the mesh generation and solution steps, cells are triangulated. This means that each cell is represented as a combination of triangles. An illustration is given in Figure 6.2.

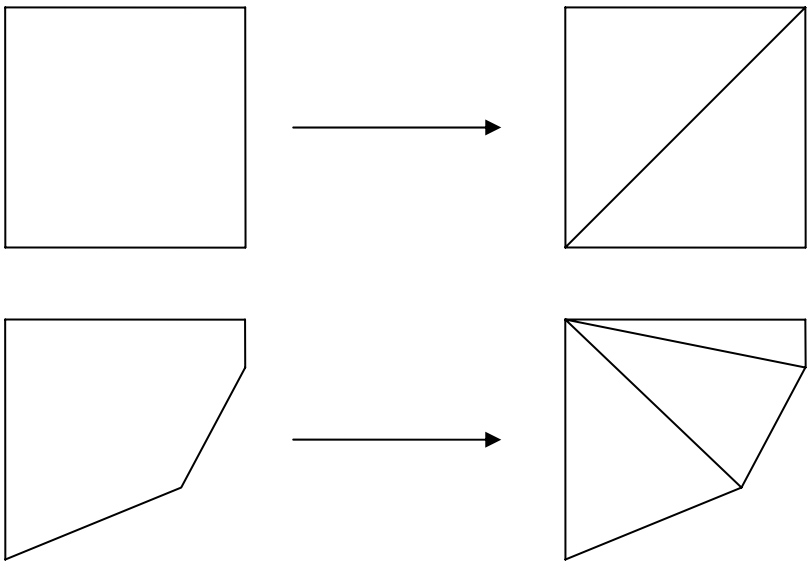


Figure 6.2 Cell Triangulation

6.3 Pressure Distribution

Another step of post-processing is presenting pressure distribution on the input geometry. For this purpose, cell-centered values are reconstructed in order to find the values on the input geometry. This is illustrated in Figure 6.3.

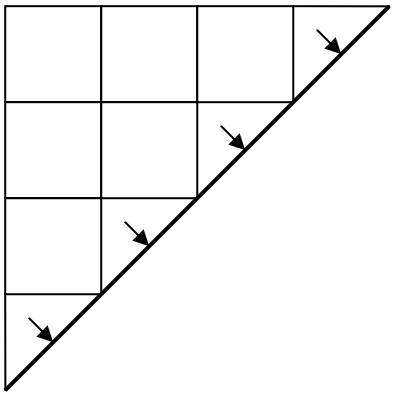


Figure 6.3 Reconstruction of Pressure for Pressure Distribution

6.4 Residual History

Another step of post-processing is to represent the convergence history of the solution. For this purpose, a residual graph is also supplied. It gives residual values at each iteration. The residual values are usually given in logarithmic scale.

In this thesis the average value of the residuals of the first primitive variable, namely density, is presented. Hence, residual graphs give these values at each iteration. Other values like the maximum of the residuals of the fourth primitive variable, namely pressure, may also be presented.

6.5 Solution Data

For each problem a data file is produced that contains information about the problem itself and the solution process. Such a file is important in that it specifies which problem is solved under which conditions for later reference. Moreover, it helps specifying the performance of the solution process. A typical data file may contain the information given below.

- The number of all-cell, cut-cell, curvature and solution adaptations
- The order of the solution and whether limiting is used or not
- Computational time and the total number of iterations
- The number of unused, computational and boundary cells
- The maximum and minimum cell levels
- The maximum and minimum residual values

CHAPTER 7

RESULTS

In order to verify the accuracy and efficiency of the method, a number of test cases are solved. The results are compared with analytical and experimental ones as well as the results obtained by other solvers.

7.1 Comparison with an Analytical Solution

The geometry is composed of a forward-facing and a backward-facing ramp. The angles are 15 degrees and the height is 0.134 units. The distance between the two ramps is 2 units. The free-stream Mach number and the free-stream total pressure values are 2 and 101325 Pa, respectively. The total temperature value is 300 K.

The initial mesh is produced by 2 levels of all-cell, 7 levels of cut-cell and 3 levels of curvature adaptation. 5 levels of solution adaptation are then applied successively. There are 15648 computational cells in the final mesh. The minimum cell level is 2 and the maximum one is 15. The final mesh around the forward-facing and backward-facing ramps is given in Figure 7.1a and 7.1b, respectively. The problem is solved in 13 minutes and the total number of iterations is 1547. The residual history is given in Figure 7.1c. The peaks are due to the 5 levels of solution adaptation applied.

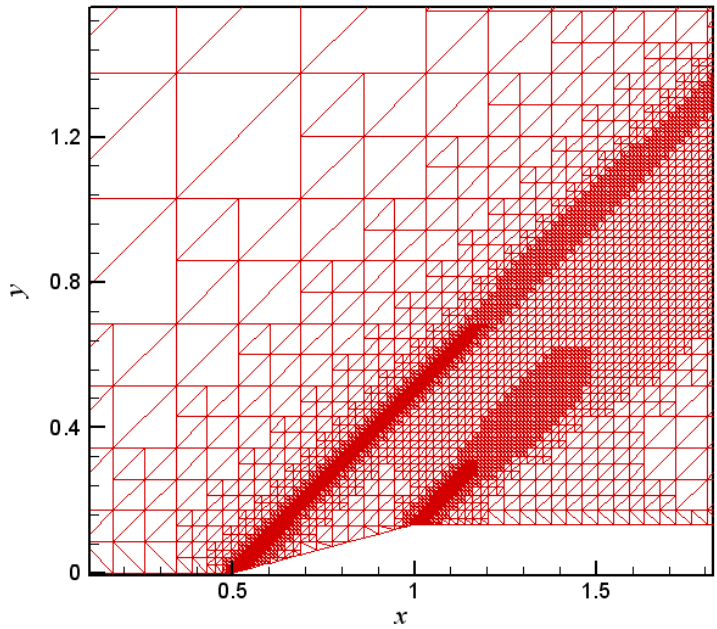


Figure 7.1a Final Mesh around the Forward-Facing Ramp at $M_\infty = 2$

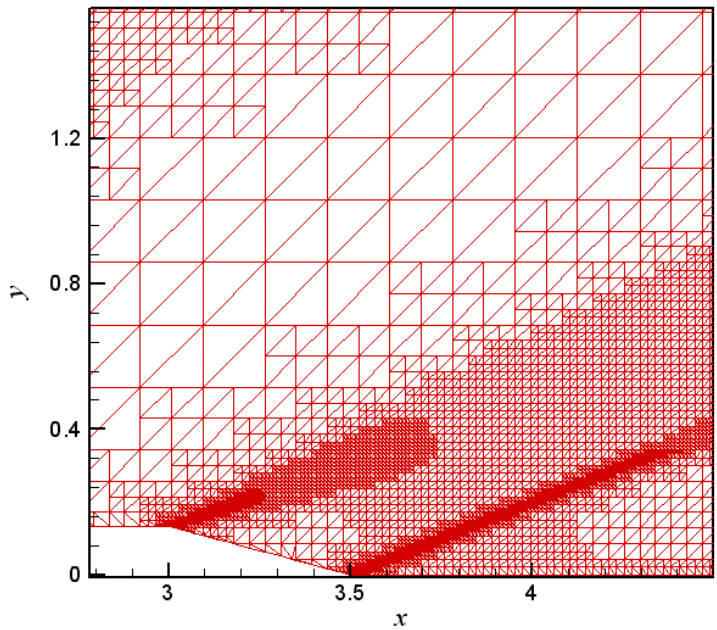


Figure 7.1b Final Mesh around the Backward-Facing Ramp at $M_\infty = 2$

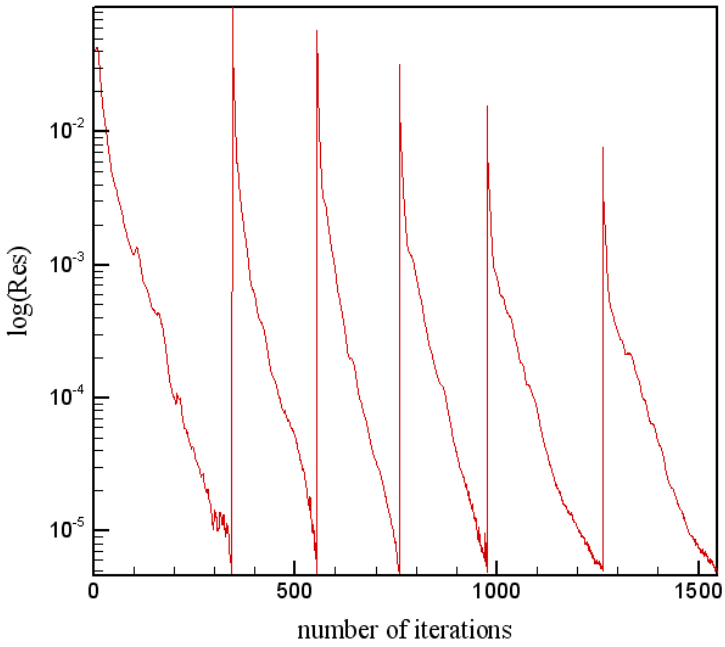


Figure 7.1c Residual History at $M_\infty = 2$

The pressure and Mach number contours are given in Figure 7.1d and 7.1e, respectively. There are two oblique shock waves attached to the two compression corners. In addition to this, there are two expansion fans attached to the two expansion corners.

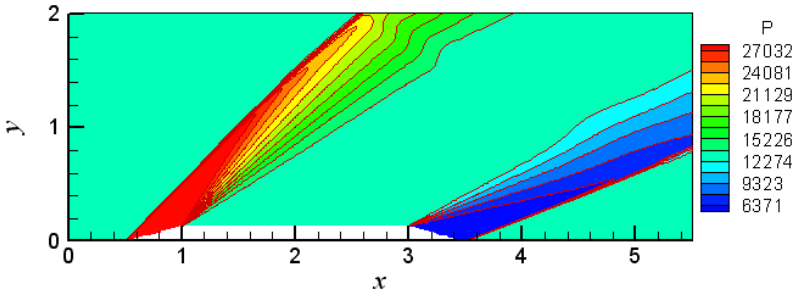


Figure 7.1d Pressure Contours at $M_\infty = 2$

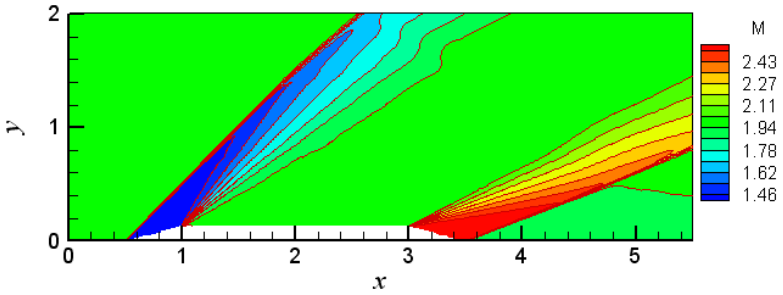


Figure 7.1e Mach Number Contours at $M_\infty = 2$

The computed and analytically calculated pressure distributions are compared in Figure 7.1f. The computed values are very close to the analytically calculated ones. The Mach number distributions are compared in Table 7.1.

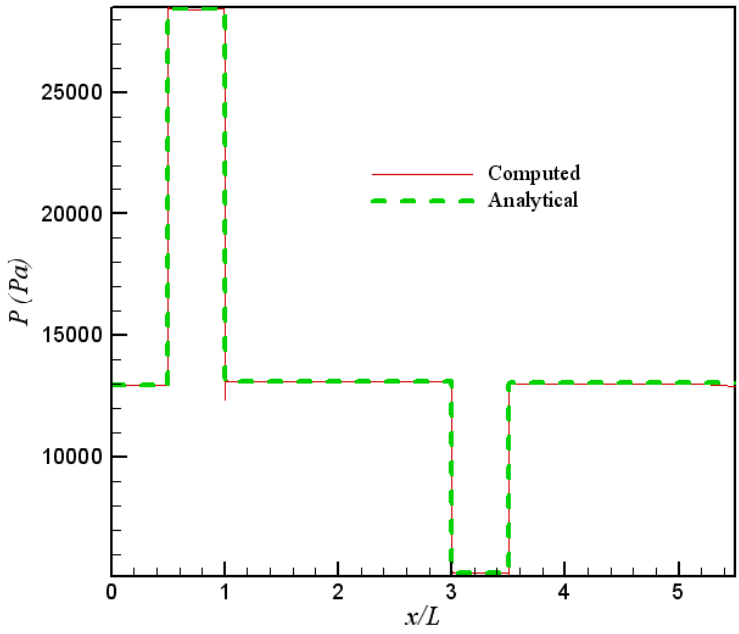


Figure 7.1f Computed and Analytical Pressure Distributions at $M_\infty = 2$

Table 7.1 Exact and Computed Mach Number Values at $M_\infty = 2$

	Exact	Computed	Error (%)
Between the First Shock Wave and the First Expansion Fan	1.445	1.443	0.114
Between the Two Expansion Fans	1.961	1.953	0.390
Between the Second Expansion Fan and the Second Shock Wave	2.551	2.536	0.613
After the Second Shock Wave	1.914	1.911	0.146

7.2 Backward-Facing Step

The geometry is composed of a backward-facing step. The step height is 0.443 units. The domain extends 4 units upstream of the step, 12 units downstream of the step and 6.25 units above the step. The free-stream Mach number and the free-stream total pressure values are 2.5 and 101325 Pa, respectively. The total temperature value is 300 K.

The initial mesh is produced by 2 levels of all-cell, 7 levels of cut-cell and 3 levels of curvature adaptation. 4 levels of solution adaptation are then applied successively. There are 46145 computational cells in the final mesh. The minimum cell level is 2 and the maximum one is 16. The problem is solved in 42 minutes and the total number of iterations is 6286. The residual history is given in Figure 7.2a.

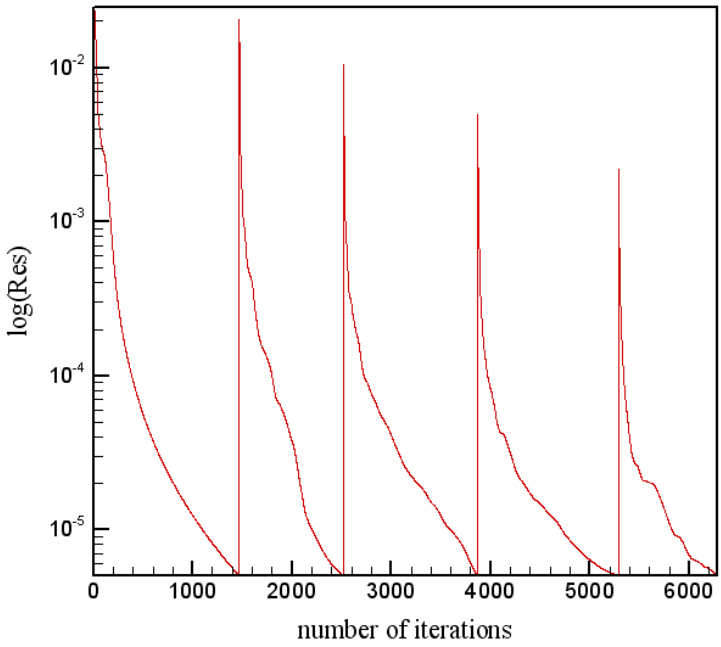


Figure 7.2a Residual History for Backward-Facing Step at $M_\infty = 2.5$

The pressure and Mach number contours are given in Figure 7.2b and 7.2c, respectively. The details are given in Figure 7.2d and 7.2e. The flow expands over the corner of the step. There is an oblique shock wave above the lower surface of the duct downstream the step. The computed pressure distribution on the lower surface of the duct downstream the step is compared with experimental data [29] in Figure 7.2f.

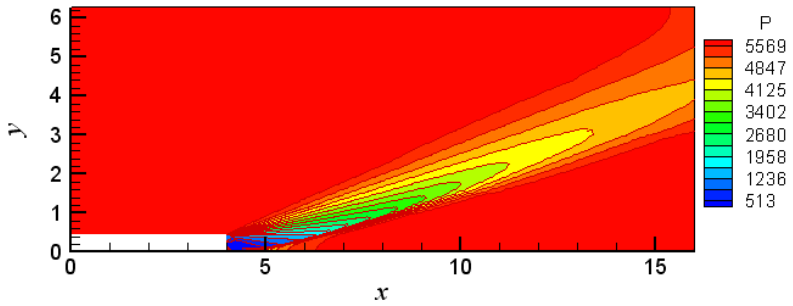


Figure 7.2b Pressure Contours for Backward-Facing Step at $M_\infty = 2.5$

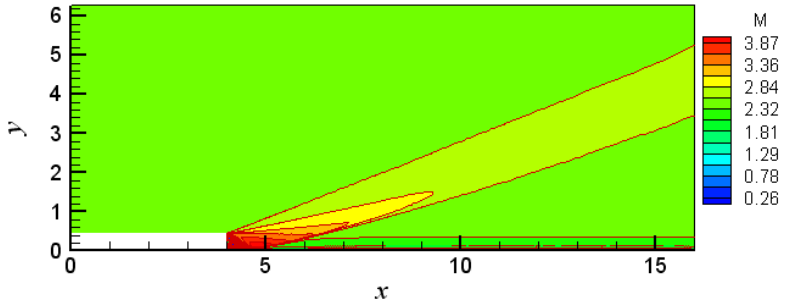


Figure 7.2c Mach Number Contours for Backward-Facing Step at $M_\infty = 2.5$

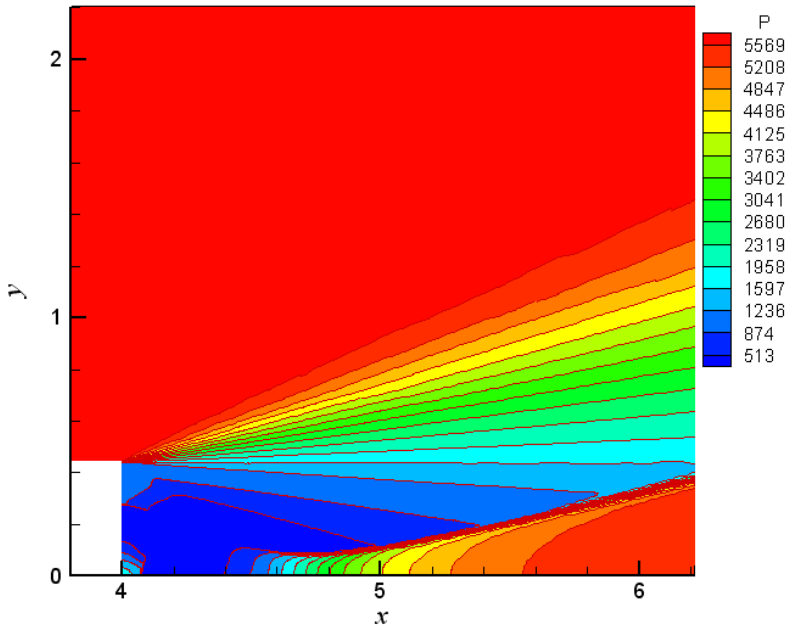


Figure 7.2d Pressure Contours around the Step at $M_\infty = 2.5$

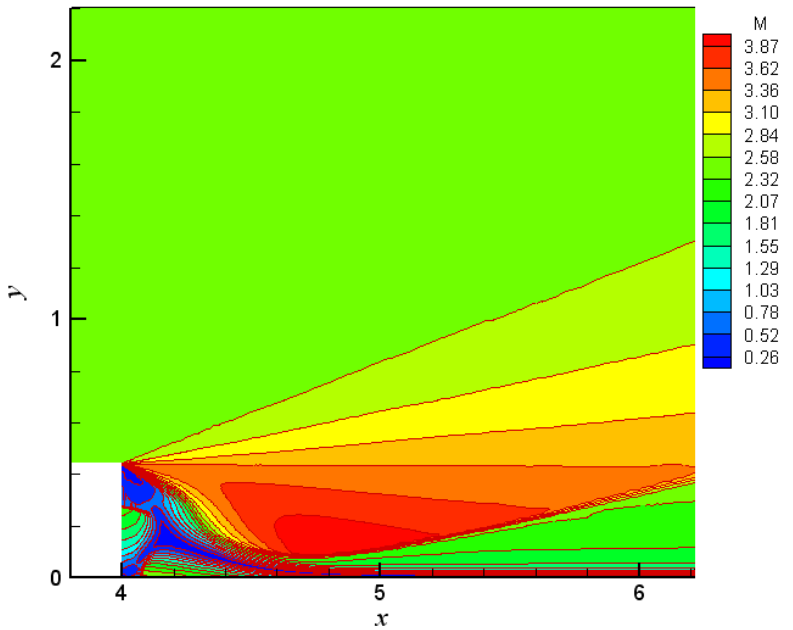


Figure 7.2e Mach Number Contours around the Step at $M_\infty = 2.5$

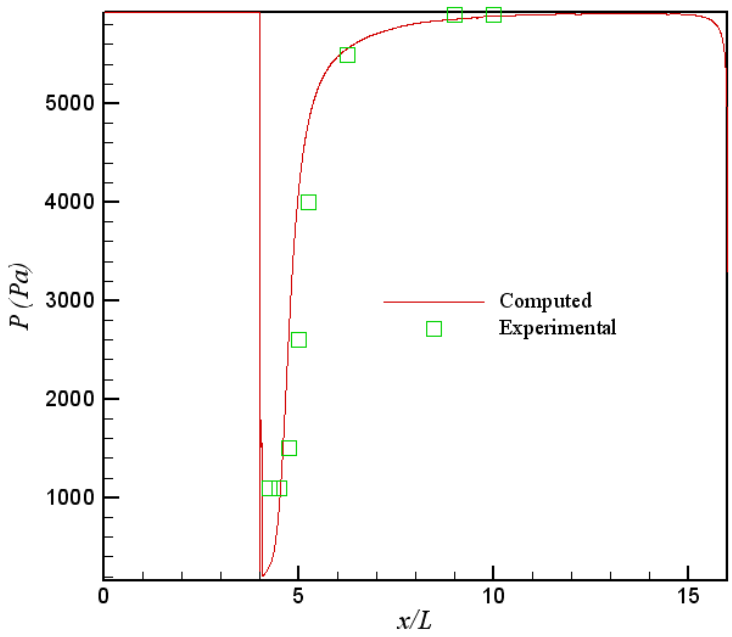


Figure 7.2f Computed and Experimental Pressure Distributions

7.3 Wedge

The geometry is composed of a forward-facing ramp. The angle is 15 degrees and the height of the ramp is 0.134 units. The domain extends 0.5 units upstream of the ramp, 2.5 units downstream of the ramp and 0.866 units above the ramp. The free-stream Mach number and the free-stream total pressure values are 2 and 101325 Pa, respectively. The total temperature value is 300 K.

The initial mesh is produced by 2 levels of all-cell, 7 levels of cut-cell and 3 levels of curvature adaptation. 4 levels of solution adaptation are then applied successively. There are 8910 computational cells in the final mesh. The final mesh is given in Figure 7.3a. The minimum cell level is 2 and the maximum one is 14. The problem is solved in 35 minutes and the total number of iterations is 4258. The residual history is given in Figure 7.3b.

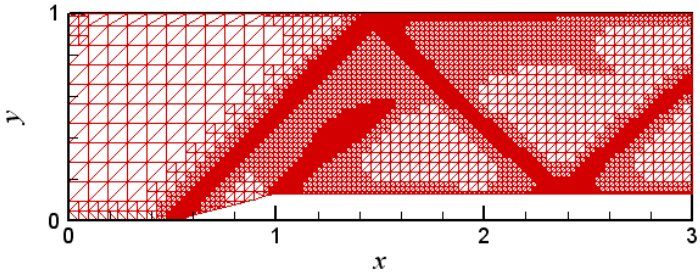


Figure 7.3a Final Mesh for Wedge at $M_\infty = 2$

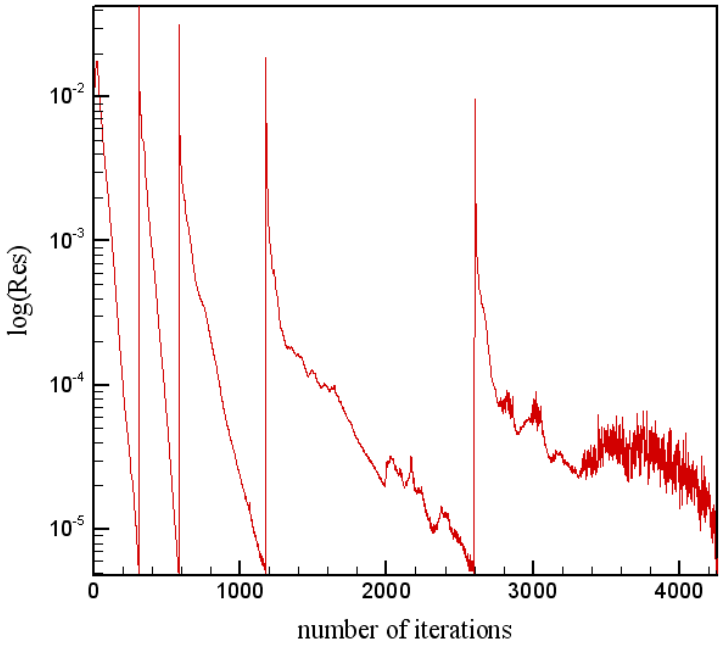


Figure 7.3b Residual History for Wedge at $M_\infty = 2$

The pressure and Mach number contours are given in Figure 7.3c and 7.3d, respectively. There is an oblique shock wave attached to the compression corner. It reflects from the upper and lower walls of the duct and leaves the computational domain. In addition to this, there is an expansion fan around the expansion corner. It

acts to weaken the reflected shock wave. Another expansion region exists between the reflected shock wave and the upper wall of the duct.

The computed pressure distributions on the upper and lower walls of the duct are compared with those obtained by another Cartesian code written by De Zeeuw [6] in Figure 7.3e. The results are almost the same. The Mach contours are also compared with those obtained by the same Cartesian code in Figure 7.3f. The solid lines are from De Zeeuw. Similarly, the two codes give almost the same results.

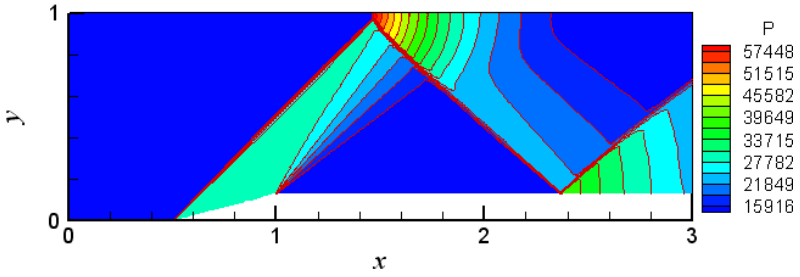


Figure 7.3c Pressure Contours for Wedge at $M_\infty = 2$

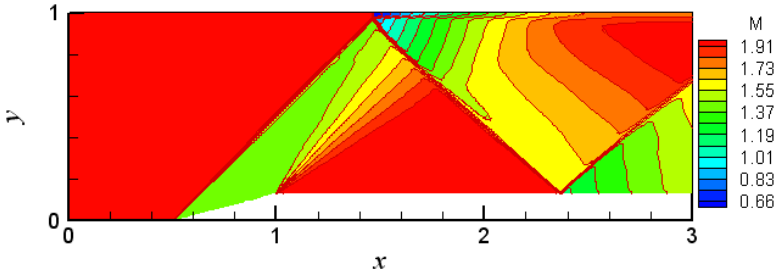


Figure 7.3d Mach Number Contours for Wedge at $M_\infty = 2$

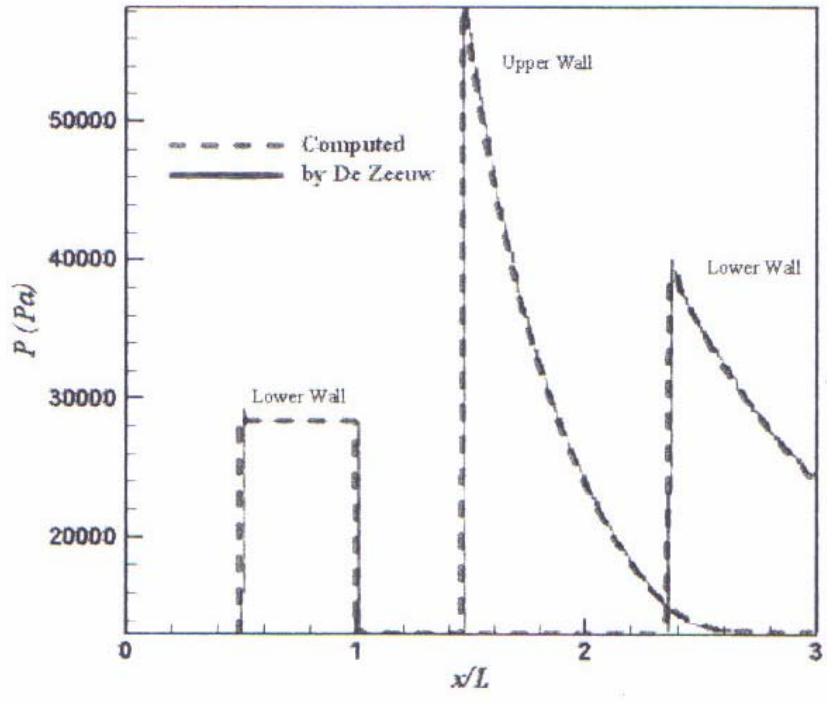


Figure 7.3e Comparison of Pressure Distributions for Wedge at $M_\infty = 2$

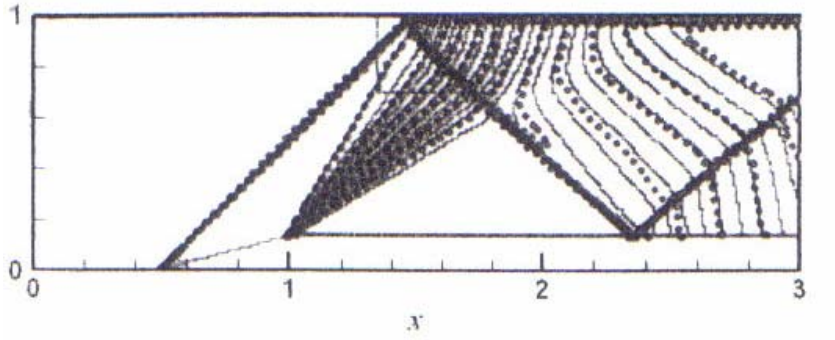


Figure 7.3f Comparison of Mach Number Contours for Wedge at $M_\infty = 2$

CHAPTER 8

CONCLUSIONS

8.1 Summary

A method is developed by which steady solution of the two-dimensional Euler equations can be obtained using geometry and solution-based adaptations for flows around arbitrarily complex input geometries.

A quadtree data structure is applied successfully. The cells are connected to each other by means of pointers in a parent-child relationship. Connectivity information is extracted from the tree by using these relationships. Ordinary linked list is used with static data structures like one or two-dimensional arrays as well.

The mesh is generated with a minimum number of inputs for arbitrarily complex bodies. The user specifies a set of points defining the input geometry, boundary conditions and the amount of geometry and solution-based adaptations. The mesh is then generated in a totally automatic manner.

The problem is then solved by using a multi-stage time stepping scheme with Roe's flux difference splitting method. Local time step approach is applied in order to increase convergence rate.

Finally, the results are post-processed. Residual graphs and pressure distributions are determined together with contours of several variables including the primitive variables. Comparisons with experimental data, analytical solutions and results of some other computer codes are performed in order to demonstrate the efficiency and accuracy of the code.

8.2 Conclusions

One of the most important needs for computational fluid dynamics is to generate automatic and robust meshes around arbitrarily complex input geometries and compute accurate solutions on these meshes. Cartesian methods are proved to overcome this problem successfully.

Accurate results usually require fine meshes. However, computational time and memory usage increase with increasing cell number. In order to obtain satisfactorily accurate results without increasing cell numbers unnecessarily, geometry and solution-based adaptations are usually needed. Cartesian methods are proved to be more suitable for both types of adaptation compared to its counterparts like structured and ordinary unstructured methods.

The technique of using local time step is applied successfully. Since in Cartesian methods there are significant differences in length scales due to geometry and solution-based adaptations, local time step is highly effective.

8.3 Future Work

In this thesis the two-dimensional Euler equations are solved. Extension to three dimensions is straightforward. In two dimensions input geometries are composed of oriented line segments. Hence, mesh generation relies on line clipping. In three dimensions input geometries are composed of oriented triangles. Consequently, line clipping is replaced by polygon clipping in three dimensions. Sutherland Hodgemann algorithm, which is used for clipping convex polygons against convex regions in three dimensions, seems to be very suitable for Cartesian methods.

Extension to the Navier-Stokes equations is less straightforward. The reason for this is that with the current data structure only isotropic refinement is possible. For resolving boundary layers with a minimum number of cells, however, anisotropic

refinement is necessary. Hence, extension to the Navier-Stokes equations requires a significant improvement in data structure.

In this thesis steady solutions of the Euler equations are sought. Hence, local time step approach is used in order to increase convergence rate. For unsteady solutions, however, the same time step is to be used for all cells. Moreover, unsteady problems require more sophisticated mesh generation techniques like cell merging.

As mentioned before, different regions of a split cell are considered as a whole and are given the same set of primitive variables. Although the effect of this approach to the overall accuracy of the solution is negligible, more accurate results can be obtained by considering each region being totally independent from each other. This approach, however, requires a more complicated mesh generation process and an advanced data structure.

In Cartesian methods a significant portion of computational time is spent for obtaining connectivity information. One way of reducing this is to determine connectivity once at the beginning and then store it for each cell. This approach, however, increases memory need unreasonably. A compromise in between may be realized by changing the data structure slightly. Grouping the cells into a number of boxes and combining these boxes to each other by pointers may reduce computational time without increasing memory usage significantly.

REFERENCES

[1] D. K. Clarke, D. Salas, H. A. Hassan, “Euler Calculations for Multielement Airfoils Using Cartesian Grids”, AIAA Journal, 24, 353-358, (1986).

[2] R. A. Mitchel Tree, M. D. Salas, H. A. Hassan, “Grid Embedding Technique Using Cartesian Grids for Euler Solutions”, AIAA Journal, 26, 754-756, (1988).

[3] D. M. Tidd, D. J. Strash, B. Epstein, A. Luntz, A. Nachshon, T. Rubin, “Multigrid Euler Calculations over Complete Aircraft”, Journal of Aircraft, 29, 1080-1085, (1992).

[4] B. Epstein, A. Luntz, A. Nachshon, “Cartesian Euler Method For Arbitrary Aircraft Configurations”, AIAA Journal, 30, 679-687, (1992).

[5] K. Morinishi, “A Finite-Difference Solution of the Euler Equations on Non-Body-Fitted Cartesian Grids”, Computers & Fluids, 21, 331-344, (1992).

[6] D. L. De Zeeuw, “A Quadtree-Based Adaptively-Refined Cartesian-Grid Algorithm for Solution of the Euler Equations”, PhD thesis, University of Michigan, (1993).

[7] W. J. Coirier, “An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations”, PhD Thesis, University of Michigan, (1994).

[8] J. J. Quirk, “An Alternative to Unstructured Grids for Computing Gas-Dynamic Flows Around Arbitrarily Complex 2-Dimensional Bodies”, Computers & Fluids, 23, 125-142, (1994).

[9] M. J. Aftosmis, “3D Applications of a Cartesian Grid Euler Method”, AIAA Paper 95-0853, (1995).

[10] M. J. Aftosmis, “Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry”, AIAA Paper 97-0196, (1997).

[11] W. J. Coirier, K. G. Powell, “An Accuracy Assessment of Cartesian-Mesh Approaches for the Euler Equations”, Journal of Computational Physics, 117, 121-131, (1995).

[12] R. B. Pember, J. B. Bell, P. Colella, W. Y. Crutchfield, M. L. Welcome, “An Adaptive Cartesian Grid Method for Unsteady Compressible Flow in Irregular Regions”, Journal of Computational Physics, 120, 278-304, (1995).

[13] A. M. Khokhlov, “Fully-Threaded Tree Algorithms for Adaptive Refinement Fluid Dynamics Simulations”, (1998).

[14] H. Forrer, R. Jeltsch, “A Higher-Order Boundary Treatment for Cartesian-Grid Methods”, Journal of Computational Physics, 140, 259-277, (1998).

[15] Z. N. Wu, K. Li, “Anisotropic Cartesian Grid Method for Steady Inviscid Shocked Flow Computation”, International Journal for Numerical Methods in Fluids, 41, 1053-1084, (2003).

[16] L. Qian, D. M. Causon, D. M. Ingram, C. G. Mingham, “Cartesian Cut Cell Two-Fluid Solver for Hydraulic Flow Problems”, Journal of Hydraulic Engineering, 129, 688-696, (2003).

[17] J. Hunt, “An Adaptive 3D Cartesian Approach for the Parallel Computation of Inviscid Flow about Static and Dynamic Configurations”, PhD thesis, University of Michigan, (2004).

[18] K. Li, Z. N. Wu, “Nonet-Cartesian Grid Method for Shock Flow Computations”, Journal of Scientific Computing, 20, 303-329, (2004).

[19] A. D. French, “Solution of the Euler Equations on Cartesian Grids”, Applied Numerical Mathematics, 49, 367-379, (2004).

[20] A. Dadone, B. Grossman, “Ghost-Cell Method for Inviscid Two-Dimensional Flows on Cartesian Grids”, AIAA Journal, 42, 2499-2507, (2004).

[21] W. A. Keats, F. S. Lien, “Two-Dimensional Anisotropic Cartesian Mesh Adaptation for the Compressible Euler Equations”, International Journal for Numerical Methods in Fluids, 46, 1099-1125, (2004).

[22] F. P. Preparata, M. I. Shames, “Computational Geometry”, (1985).

[23] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, “Computational Geometry”, (2000).

[24] C. Hirsch, “Numerical Computation of Internal and External Flows”, (1990).

[25] C. B. Laney, “Computational Gas Dynamics”, (1998).

[26] H. Lomax, T. H. Pulliam and D. W. Zingg, “Fundamentals of Computational Fluid Dynamics”, (2001).

[27] J. H. Ferziger and M. Peric, “Computational Methods for Fluid Dynamics”, (1996).

[28] D. A. McCaughey and M. M. Mafez, “Frontiers of Computational Fluid Dynamics”, (1994).

[29] H. E. Smith, “The Flow Field and Heat Transfer Downstream of a Rearward Facing Step in Supersonic Flow”, (1967).