4

CLaSSiC Project
Report CLaSSiC-90-07

January 1990

Composite-Grid Techniques and Adaptive Mesh Refinement in Computational Fluid Dynamics

Robertus Franciscus van der Wijngaart

Center for Large Scale Scientific Computation
Building 460, Room 313
Stanford University
Stanford, California 94305

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS NONE
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT UNLIMITED
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) CLaSSiC Report 90-07	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Stanford University; 6b OFFICE SYMBOL (If applicable) 2E254	7a NAME OF MONITORING ORGANIZATION Department of the Navy Office of Naval Research
6c ADDRESS (City, State, and ZIP Code) c/o Sponsored Projects Office Encina Hall, 660 Arguello Way Stanford, CA 94305	7b ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research; 8b OFFICE SYMBOL (If applicable) N00014	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0565
8c ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
COMPOSITE-GRID TECHNIQUES AND ADAPTIVE MESH REFINEMENT IN COMPUTATIONAL FLUID DYNAMICS unclassified

12 PERSONAL AUTHOR(S)
Robertus Franciscus van der Wijngaart

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
interim	FROM 86Jul1 TO 89Jun30	1990 January	227

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computational Fluid Dynamics, Navier-Stokes, Adaptive Grids, Composite Grids, Schwarz Alternating Procedure

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Viscous fluid flow is often smooth in most of the domain, with regions of rapid variation confined to some rather narrow zones in the field. These zones (boundary layers, shocks, etc.) cause problems during numerical solution of the equations governing the flow. The patched adaptive mesh refinement technique, devised at Stanford by Oliger, et al., copes with these sources of error efficiently by refining the computational grid locally. This is done by creating separate fine grids for every region of large error.

Because of the success of this approach, a project was started to extend its applicability to geometrically complex domains. As patched adaptive mesh refinement alread entails multiple grids, it was decided that geometrical complexity would also be tackled using several grids. An arbitrarily shaped domain typically cannot be covered by a singl

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Joseph Oliger	22b TELEPHONE (Include Area Code) (415) 723-0571; 22c OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE
S/N 0102-LF-014-6603 UNCLASSIFIED

grid without severe distortion, but a covering can be established with only mildly curved grids if more than one grid is allowed. Communication between these grids then becomes an issue, as well as their creation.

In this project various types of communications between grids, based on the Schwarz Alternating Procedure (SWAP), are examined for solving steady, two-dimensional incompressible-flow problems. Of these, the traditional SWAP on sets of overlapping grids works best and gives accurate results, despite the use of nonconservative interpolation procedures between grids. When reentrant problems occur, the pressures between grids may not match. A pressure-communication scheme is devised which solves this difficulty.

In addition, the design of a system to create the multiple grids (composite grid) interactively is presented, together with the data structures needed to define the grids and their interactions. A scenario for combining composite and adaptive grids is also described.

As a by-product, a new view on some classical solution procedures for incompressible flows is developed, and a nonstandard type of staggered grid is proposed. It is found that the latter has several advantages over the standard staggered grid, whereas its only drawback — an oscillatory pressure field — can easily be removed.

COMPOSITE-GRID TECHNIQUES AND ADAPTIVE MESH REFINEMENT IN COMPUTATIONAL FLUID DYNAMICS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Robertus Franciscus van der Wijngaart
December 1989

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Joel H. Ferziger
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Joseph E. Oliger

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Robert L. Street

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Gene H. Golub

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

Viscous fluid flow is often smooth in most of the domain, with regions of rapid variation confined to some rather narrow zones in the field. These zones (boundary layers, shocks, *etc.*) cause problems during numerical solution of the equations governing the flow. The *patched adaptive mesh refinement technique*, devised at Stanford by Oliger *et al.* [OLI84], copes with these sources of error efficiently by refining the computational grid locally. This is done by creating separate fine grids for every region of large error.

Because of the success of this approach, a project was started to extend its applicability to geometrically complex domains. As patched adaptive mesh refinement already entails multiple grids, it was decided that geometrical complexity would also be tackled using several grids. An arbitrarily shaped domain typically cannot be covered by a single grid without severe distortion, but a covering can be established with only mildly curved grids if more than one grid is allowed. Communication between these grids then becomes an issue, as well as their creation.

In this project various types of communications between grids, based on the Schwarz Alternating Procedure (SWAP), are examined for solving steady, two-dimensional incompressible-flow problems. Of these, the traditional SWAP on sets of overlapping grids works best and gives accurate results, despite the use of nonconservative interpolation procedures between grids. When reentrant problems occur, the pressures between grids may not match. A pressure-communication scheme is devised which solves this difficulty.

In addition, the design of a system to create the multiple grids (*composite grid*) interactively is presented, together with the data structures needed to define the grids and their interactions. A scenario for combining composite and adaptive grids is also described.

As a by-product, a new view on some classical solution procedures for incompressible flows is developed, and a nonstandard type of staggered grid is proposed. It is found that the latter has several advantages over the standard staggered grid, whereas its only drawback —an oscillatory pressure field— can easily be removed.

Acknowledgments

The work described in this thesis is the culmination of some four years of hard work. During that period an estimated 410 gallons of strong coffee has flowed under the bridge. It has been with the support of this precious liquid and of the friends that I made at Stanford that I was able to carry out my research and finish this dissertation.

So I should like to thank Peter James Coffee Company for the continuous supply of Vienna Roast. I should also thank my advisor, Joel Ferziger, for getting me started on my research. Joe Oliger, my co-advisor, deserves a lot of credit for the many hours he spent patiently with Steve Suhr and me, discussing computer science issues that proved so useful in this research. Steve Suhr in turn spent many additional hours with me, helping to fill the gaps in my computer science knowledge, and I greatly appreciate his contributions to this project. Gene Golub got me interested in matrices, and it has been through the seminars and workshops he organized that I got some of the best ideas contained in this work.

Then there are the persons whose main contributions have not been academic. First there are Steve and Monica Caruso, who stood by me during the hardest times I endured in this country, and who also gave me the greatest joys. They taught me about the blessings of unconditional friendship and garlic-laden Italian food. Then there are Andy and Eleanor Doty, in whose house I lived for most of my time at Stanford. Their affection and support and genuine interest in whatever I did have given me a sense of home and belonging, so many miles away from my native Holland. Ludolf Meester, my good friend and fellow countryman, gave me the opportunity to practice Dutch once in a while and to talk about all the things that are so much better in the Old Country. Next year, no doubt, we will be reminiscing together in the rain in Amsterdam about the wonders of the New World. Sergio Bordalo, Ramu Avva, Steve Tzuoo, Rebecca Moore, and Amala Krishna.

all former and current office mates, made my life at Stanford great by tolerating my sometimes somewhat irreverent behavior, by engaging in so many stimulating discussions about life, death, sex, and graduate school, and by bearing with the pungent smells of coffee pervading our shared space. I am particularly grateful to Amala, who went over several versions of this manuscript meticulously and caught many typos. Ramana Venkata, whose keen wit, intelligence and perseverance I admire, and whose friendship I value highly, I thank you for all the help you have given me and are giving me now. Ramani Pichumani, Ray Tuminaro, Dulce Ponceleon and Mark Kent, countless are the times you helped out when I was struggling with obstinate computers and mysterious manuals. I can give back nothing but my friendship and gratitude.

During the winter quarter of '88 I was able to do research in Tokyo at the Ship Research Institute through a grant from the Japanese government. I would like very much to thank my supervisor, Yoshiaki Kodama, and my host, Munehiko Hinatsu, for inviting me and for making my stay such a wonderful experience.

Then there are all the friends and relatives I left behind in Europe and who have helped maintain ties faithfully by writing to me and visiting me through all these years. It is with both joy and sadness that I now return to Holland. Joy because I will be among the people again whose company I had to miss for so long. Sadness because I am leaving new friends behind. It should be this way.

Finally, I gratefully acknowledge the financial support provided by the Office of Naval Research under contract number N00014-86-K-0-565, which has made this whole adventure possible.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Motivation and Objectives

Fluids are everywhere: the air that we breathe, the water we drink, the gas in our stoves, the blood in our veins. Fluids are capricious: sudden wind shear may cause planes to crash, dimples in golf balls make them move faster through air at high speed but slow them down at low speed, visibility in Los Angeles drops alarmingly due to smog on some days and is perfect on others. Ubiquity and whim have attracted man's attention since ancient times, but the inherently difficult nature of fluids has defied analysis for ages. It was not until the advent of modern mathematics and the analytical intellect of scientists like Newton, Euler, Bernoulli, and Stokes that qualitative as well as quantitative insight in the behavior of fluids was obtained. The formulation of the Navier-Stokes equations was the result of the combined efforts of these researchers and their contemporaries. Up to this day, a century and a half since their conception, these are the most important equations used to predict and explain fluid flows.

Unfortunately, only very few exact analytical solutions to the Navier-Stokes equations are known, and those are typically of little engineering interest. Virtually every realistic problem involving fluid flow requires approximate solution of the Navier-Stokes equations. An interesting semi-analytical approach is offered by asymptotic expansions, which yield solutions in terms of (infinite) series involving some small parameter, such as viscosity, or geometric deformation. The bulk of approximating techniques, however, is purely numerical. Of those numerical techniques, the most popular ones are finite-difference and finite-element methods. Which is chosen is largely a matter of taste and tradition. In this report we will discuss only finite-difference techniques.

The way these work is as follows: a set of discrete points is chosen on which a *sample* of the whole continuous flow field is computed. These points are not randomly picked, but chosen so that they can be indexed according to the dimensionality of the problem, *i.e.*, two indices for a two-dimensional problem, three for a three-dimensional problem, thus forming a lattice or *structured grid*. Indices of neighboring grid points differ by some small number. Derivatives are then approximated by difference quotients involving neighboring points. This leads to a set of algebraic equations that is subsequently solved for the flow quantities of interest.

The three steps —grid generation, discretization, solution— in the above described finite-difference method all pose particular difficulties that are, to some extent, intertwined. As derivatives are approximated using values at discrete points, these points must be sufficiently close together to yield accurate difference quotients (small mesh sizes). Grid lines are formed by connecting points through incrementing one index while keeping the other indices fixed. Grid lines of different families (different incremented indices) should not intersect at too small an angle, as this leads to inaccuracies in the approximations of derivatives, and hence of the flow equations. Moreover, grid lines themselves should not change direction too abruptly (no kinks).

These demands are not always easily met, especially if the domain of interest has a complex shape, or if the solution is difficult to describe locally. It is the purpose of this work to arrive at robust, general techniques for dealing with these difficulties.

1.2 Resolving Solution Complexity: *Adaptive Grids*

Sources of error in the numerical computation of flow fields are often confined to rather small regions. One may think of thin boundary layers, shock waves, flame fronts, *etc.* Computing accurate solutions in these zones requires very small mesh sizes. Applying these fine meshes throughout the domain is inefficient and may be prohibitively expensive. One would like to apply fine meshes only where locally required, and to use relatively coarse meshes elsewhere. Typically, we do not know in advance where the solution will be hard to approximate, so some kind of automated error estimation procedure should be used to determine dynamically where fine grids are needed. This is called solution-adaptive mesh refinement (adaptive grids). Many solution-adaptive mesh refinement methods have been devised. For an extensive description the reader is referred to [CAR85] and [BER82]. We will merely mention the major classes of grid adaptivity:

- **Global mesh refinement**: One starts with a fixed number of grid points. As the solution is computed iteratively, the error is estimated, and grid points are moved towards regions of large error. This technique, the oldest adaptive-refinement method, has been improved substantially over the years, and has reached a degree of sophistication not equalled by others. Initial difficulties, such as extreme deformation of grid lines, have largely been overcome. Nevertheless, the technique has some important drawbacks:
  - as points move from one region to another, some areas tend to become devoid of points; no minimum accuracy may be prescribed for a fixed number of points.
  - error-estimation procedures are notoriously noisy, which leads to incorrect inputs to grid-point-moving schemes.
- **Local mesh refinement**: Now the set of initial grid points is kept in place. Rather than moving points, new points are added where the error is large. This means that no regions become void, and a minimum accuracy can be guaranteed. However, depending on the particular type of adding strategy, new difficulties arise:
  - o *Embedded mesh refinement*: Added points are incorporated into the data structure of the already existing coarse grid. Typical ways of doing this are:
    - *Tensor-product refinement* (refinement by lines): Whole new lines of grid points are added to refine error zones. This works reasonably well for localized refinement of regions with small aspect ratios, but

becomes very inefficient when the error zone is elongated, in which case a substantial part of the grid must be refined. This problem is aggravated if the zone is diagonal to the grid lines.

- *Refinement by points*: New points are added only there where they are needed (see, *e.g.*, [DAN86]). This technique is efficient with respect to the number of points used to compute a solution, but requires a substantial overhead in the storage of the separate points, generation of boundary conditions for grid lines that end in the interior of the domain, and tracking of isolated points within refined regions ('holes').

o *Patched mesh refinement*: The approach taken by Berger [BER82] and Caruso [CAR85], called patched adaptive mesh refinement, involves separate refined grids for whole areas of large error. The method, which will be explained in greater detail below, has the advantage that it combines the robustness of embedded refinement with the efficiency of global refinement. Moreover, standard solution procedures for regular grids can be applied, as solutions are computed on sets of separate, regular grids

In the patched adaptive mesh refinement technique a solution is first computed on a relatively coarse grid, which is expected to be fine enough for the smooth regions of the flow, but too coarse for some other parts. Using the Richardson extrapolation technique, the error in the solution is estimated at every mesh point. If the error is above a certain threshold value, the point is flagged as a bad point. The bad points are clustered according to an algorithm which tries to recognize coherence of collections of points. Next, the clusters are fitted with rectangular grids of smaller mesh size (see Fig. 1.1). These will generally not be aligned with the grid lines of the coarse grids. Depending on the cover fraction (ratio of number of bad points over number of good points contained in the refined grids), a particular clustering is rejected or accepted. Sometimes several passes are necessary. Finally, boundary values for the refined grids are interpolated from the coarse grid, and independent solutions are obtained on the refinements. The procedure may be defined recursively by computing the error on the refined grids and invoking new levels of refinements.

Two bottlenecks exist in this approach. First, the creation of the refinements according to some clustering technique is complex and involves heuristics. Second, the interpolation of boundary values from the coarse grid to the refinements, and the communication between refinements themselves in case they overlap, is not conservative, in general. This has been known to lead to instabilities, and, in computing weak solutions to the Euler equations, may result in incorrect shock speeds and/or locations. Shocks need not concern us here, as we are primarily interested in solving incompressible, viscous flow problems, which allow no such solutions. Conservation is of importance, although it should be stressed that it is nothing to be dogmatic about. What we ultimately desire is an accurate solution, no more, no less. Some remedies have been suggested in the past to fix the lack of conservation of interpolation schemes, which will be discussed in Chapter 8. For successful implementation of the patched adaptive mesh refinement technique for rectangular domains we refer to Caruso [CAR85].

1.3 Resolving Geometric Complexity: *Composite grids*

Finite-difference methods are easiest to implement for rectangular regions. Grid lines are defined by Cartesian coordinate lines, leading to simple, rectangular grid cells and to families of grid lines that intersect each other at right angles. In addition, the boundaries of the domain coincide with grid lines, which makes for an easy implementation of boundary conditions. When the domain of interest is not a rectangle, but, for instance, a circle, several options exist. We may still want to apply a Cartesian grid, resulting in a stair-step approximation of the physical boundary (see Fig. 1.2). This has two main disadvantages. First, we have to do bookkeeping on all the cells on the boundary in order to determine which are interior points and which receive boundary conditions. Second, and more important, application of boundary conditions becomes inaccurate, as grid points no longer lie exactly on the boundary of the physical domain. It has been shown that this can give rise to spurious waves (see, *e.g.*, [PED86]) which corrupt the entire solution and decay very slowly as the mesh size decreases.

Hence, it has become common practice to use grids with curved grid lines which follow the boundary: so-called boundary-fitted grids. Here the concept of computational space is introduced explicitly. As the dimensionality of the physical problem equals the number of indices of the grid points, the indices may be viewed as coordinates in a transformed space: computational space. Using this concept, a grid is then defined by a rectangular array of indices —the domain in computational space— plus a mapping from the collection of pairs (or triplets) of indices to physical space. The mapping itself is called a coordinate transformation. It may be an explicit formula, as in algebraic grid generation procedures, or a solution of a differential equation, as in the numerical grid generation methods proposed by Thompson *et al.* [THO85], or simply a list of pairs of points in computational and physical space. If an explicit formula is available, it makes sense to consider it a continuous mapping from a compact domain in computational space to a compact range in physical space (see Fig. 1.3). In the other two cases a continuous mapping can be defined by interpolation of discrete data. This is of particular importance for describing the boundary of the physical domain in computational coordinates. If a continuous representation of the boundary is available, then any point on a bounding coordinate line in computational space is mapped into a point on the boundary of the physical space. Thus, if one changes the number of points in a coordinate direction in computational space, one is assured that boundary points in computational space still map into boundary points in physical space.

This seems to resolve the problems; all one has to do to create a boundary-fitted grid for a physical domain is to specify a continuous description of the boundary and define some means of mapping a region in computational space into the interior of the boundary in physical space. For the circular region mentioned above, polar coordinates are suitable to describe the boundary. Unfortunately, they lead to a singularity in the center (see Fig. 1.4), where one side of every grid cell shrinks to a point. Another mapping, shown in Fig. 1.5, removes that singularity, but introduces degeneracies on the boundary where coordinate lines of different families become parallel. The seemingly trivial shape of a circle turns out to be difficult to cover with an appropriate grid!

Should we choose to solve problems on domains such as the triangular or pentangular shapes shown in Fig. 1.6, similar difficulties occur. The root of the problem is the fact that we are trying to map a rectangular region in computational space into a region with a different number of corners in physical space. A continuous, nonsingular mapping cannot introduce new singularities (corners) on the boundaries, nor smooth out original ones. Even if we allow stair-step grids in computational space, which would remove the computational efficiency of using simple Cartesian-product subspaces in computational space, we would find that the number of corners of the grid is at least four, and can only increase by an even number by adding or removing grid cells. Hence, we conclude that only regions resembling warped rectangles ('warptangles'), or unions of those, can be furnished with proper boundary-fitted grids.

Even then, problems can be expected when trying to find grids for regions with narrow waists, such as the bow-tie-shaped region shown in Fig. 1.7. In order to have a sufficiently fine mesh near the outer edges, the middle section experiences congestion of grid lines, which is wasteful in case the solution there is smooth. At the same time, grid cells may become excessively skewed, which deteriorates accuracy.

These problems become even more acute in three dimensions. The solution proposed here is to use multiple —possibly overlapping— grids of mild curvature to cover the physical domain. This fits in nicely with the patched adaptive mesh refinement strategy, which also entails several grids. The result of tessellating the physical domain is a **composite grid** (see Fig. 1.8), and the tessellation itself is often called domain decomposition. We will now discuss the work of other researchers in this area.

Three different main approaches to domain decomposition exist. The first is purely numerical. As parallel computers are gaining popularity, a lot of attention is being paid to dividing large computing tasks efficiently into many small jobs that can be distributed among different processors. Although overhead is involved in exchanging data between processors (access and transport time) and in solving pieces of problems independently and combining results afterwards, it is still possible to obtain an increase in speed if enough processors are available to do the work. Most research in this area focuses on solving simple equations (*e.g.*, Poisson or Helmholtz) on simple domains very rapidly [ROD83], [EHR86], [QUA87], [TAN87], although interest in more difficult equations is increasing [CHA88B], [MAR87]. However, almost all the domains considered are of simple shape and could easily be covered with one grid.

Another approach to domain decomposition is to recognize different regions in a domain with different physical characteristics. In each such region a different equation is being solved. One of the oldest examples is Prandtl's scheme, in which inviscid flow is computed in an outer region, and boundary layers are resolved near solid bodies. More recent examples are found in [CHI87], [SCR88]. Again, the emphasis is on solving problems on relatively simple domains.

The third view of domain decomposition —the one we focus upon— is that it is a good tool to tackle geometrical complexities. Our aim is not so much to get *maximum speed* of computation, but to obtain *accurate solutions* for regions of difficult shape. In the engineering community this seems to be the overwhelming approach to domain decomposition. It should be noted that this approach is fully compatible with the first one, as geometrically simple subdomains can be divided even further until a number suitable for the number of (parallel) processors available is obtained.

One of the major applications of computational fluid dynamics is aerodynamical simulation. Traditionally, it has been in this field that the most sophisticated grid generation techniques were introduced, since many of the occurring shapes are complex (wing-fuselage combinations, rows of turbine blades, *etc.*).

One of the earlier attempts at computing aerodynamical flows on composite grids is reported by Atta and Vadyak [ATT83]. They solve the full-potential equation on sets of overlapping grids. Exchange of information between component grids takes place through interpolation of the velocity potential. Hence, conservation of mass, momentum, or energy is not possible.

Major work was done by Rai *et al.* In [HES86] a composite-grid method for the unsteady Euler equations is described. The components of the composite grid do not overlap but just touch (*patched grids*), and the exchange of information between patches takes place through the definition of flux-balance boundaries on which exact conservation is imposed. This is done relatively easily because the method is explicit, and the interior solutions on component grids can be updated using conventional integration schemes before the boundary values are computed. Moreover, the Euler equations have the form of hyperbolic conservation laws, which provide a natural structure for computing fluxes of all dependent variables. An extension of this method to implicit computations is presented in [RAI85] and [RAI86]. Here an iterative scheme is developed to update the boundary values of patches implicitly, again making heavy use of the flux-formulation of the Euler equations. At convergence full conservation is again obtained. In [RAI87] the step is made to the compressible Navier-Stokes equations. Now component grids are also allowed to overlap to provide greater flexibility in defining component boundaries. Conditions for patch boundaries are still conservative, but straightforward, nonconservative interpolation of the dependent variables is applied on overlap boundaries. In this case, some problems were experienced with high-frequency oscillations near overlap boundaries, possibly due to the the lack of conservation.

Holst *et al.* [HOL85], [KAY86] describe a composite-grid method for compressible flows in which component grids are allowed to overlap by a fixed number of mesh cells. Grid points on interfaces coincide exactly, so that information may be transferred fully conservatively (see also [VEN87A], [TAK], [HOL87]). Although this is a desirable property, it leads to relatively rigid grid systems. Composite grids around complex geometries are built by constructing a base grid of logically rectangular structure (in computational space). Finer grids are embedded in this structure by taking out sub-blocks and replacing them by grid blocks of smaller mesh size. Grid blocks at the finest level, although still logically rectangular, have at least one body-fitted side. An additional feature of the method is that different equations are solved on different grid blocks; Euler in inviscid regions, and thin-layer Navier-Stokes near solid boundaries. It should be noted that, although coarse-grid points on interfaces coincide, the grid point distribution in the direction normal to an interface may change abruptly. In other applications ([VEN87B], [BER86]) the grid lines are continous across interfaces in the normal as well as in the tangential direction. These composite grids can be interpreted logically as block-stair-step grids. An interesting variation on the techniques described above is offered by Nakahashi *et al.* [NAK87]. His composite grid consists of regular finite-difference grids around solid bodies plus a finite-element grid in between to fill the irregularly-shaped gap. At the interface between a finite-difference grid and a finite-element grid the grid points coincide to ensure conservation.

Another important contribution to aerodynamical composite-grid computations is the Chimera approach, developed at NASA Ames [DOU85], [BEN86] to solve the Euler equations. Here the concept of *overset* grids is introduced. Composite-grid construction again starts with the definition of a base grid, but now no blocks are taken out. Instead, smaller structures are defined on top of the base grid; they are 'overset'. In general, such a smaller grid overlaps the base grid, and also part of the main solid body around which the base grid is created. A scanning procedure detects the latter invasions and eliminates points from grids that fall inside solid bodies. Because grid points are not coincident, interpolation has to be used to exchange information between grids. The nonconservative interpolation scheme used causes some problems and the authors suggest refining the mesh to overcome these.

Composite-grid techniques for incompressible flows are less numerous than those for compressible flows and have started receiving attention only recently. One of the principal difficulties of composite-grid solutions to incompressible-flow problems is the conservative transfer of information between component grids. The continuity equation no longer has the hyperbolic-conservation-law form because of the constant density. Hence, the uniform flux transfer across flux boundaries, as employed by Rai *et al.*, is not applicable. In practice, researchers using composite grids for incompressible flows always let the components overlap and interpolate data in the region of overlap [FUC85], [MEA86], [MEA88B], thereby accepting the inherently nonconservative properties of interpolation. Correction factors to compensate for excess or defect mass flow are often used, but, as we will see in Chapter 8, these do not restore conservation at convergence. Fuchs [FUC87] gets around the nonconservation problem by using streamfunction and vorticity instead of the primitive variables velocity and pressure, and by interpolating values of these quantities between component grids. Häuser *et al.* [HAU86] solve the free-surface shallow-water equations for a very complex domain modeling the harbor of Hamburg, but their assumptions on the flow reduce the equations to the simple Laplace equation. Neither of the latter two approaches has difficulties with mass conservation, but they are not general enough for practical engineering applications.

1.4 Combining the Two: *Adaptive Composite Grids*

As was mentioned in Section 1.3, the patched adaptive mesh refinement method and the composite-grid technique employ several grids simultaneously to tackle solution complexity and geometry complexity, respectively. An obvious generalization is the combination of the two in a hybrid method called *adaptive composite grids*. We will expand on the problems associated with this generalization in Chapter 5.

1.5 Scope of this Investigation

In this report we will investigate in depth the strategies for solving fluid-flow problems on complex domains using composite grids. Emphasis will be laid on two-dimensional, incompressible, steady, viscous flows.

Chapter 2, which is somewhat philosophical, lays the general foundation for composite-grid solution methods. Here the definitions of concepts used throughout this report are presented. Special attention is devoted to the distinctions between the various types of iteration and convergence pertaining to composite grids.

Chapter 3 is concerned with the more practical issues of composite-grid constructs. In it we describe the ingredients necessary to formulate a physical problem on a general set of grids. The data structures needed for implementation of the technique follow quite naturally from the definitions of composite-grid descriptors. An example of a data file describing an actual composite grid is given in Appendix A.

Chapter 4 explains the design of an interactive system called M*E*S*H (Mesh Engineering System for Hydrodynamics), which is being developed to aid in the construction of a composite grid. A comprehensive description of the system functions is given in Appendix B.

Chapter 5 discusses the choices to be made when combining adaptive grids with composite grids. The most promising option is analyzed in greater detail, and a possible scenario is outlined. The main idea is that refinements should be defined independent of components of the composite grid for reasons of efficiency.

Chapter 6 contrasts two fundamental ways of solving differential equations on compound domains, *i.e.*, the Schur-Complement Method and the Schwarz Alternating Procedure (SWAP). Although these are theoretically equivalent, they differ substantially when implemented. The Schwarz alternating procedure is chosen as the basic scheme because it is easier to implement, has lower storage requirements, and can be used in conjunction with standard solution procedures on single grids. A variation (SWAPR) is investigated for use as an iterative method on sets of grids that do not overlap.

Chapter 7 presents the numerical method for solving the Navier-Stokes equations on a single grid. The discrete equations are derived by first writing the Navier-Stokes equations in computational coordinates, and subsequently integrating them over cells in computational space. A special feature is the use of a modified staggered grid which does not require specification of the pressure on the boundary of the grid. It is shown that the introduction of the checkerboard pressure pattern to which this grid is prone does not pose any problems, as it can be eliminated by simple averaging or projection methods. For the solution of the discrete equations, the well-known SIMPLE and SIMPLER methods [PAT80] are examined in terms of general matrix equations. The insight gained from this analysis enables us to formulate a simplified, more robust version of SIMPLER (essentially the same as PRIME [MAL83]).

Chapter 8 starts with an extension of SWAPR for two-dimensional problems, employing local variation of parameters to make the method suitable for problems involving inflow and outflow through a single grid boundary. The efficiency of SWAPR is compared with that of SWAP. The resulting method is applied to some practical problems for which experimental, numerical, or asymptotic data exist. These include everybody's favorites: the lid-driven cavity flow and the circular cylinder in crossflow. In addition, results are presented for the flo[illegible] [illegible]symmetrically constricted channel, and for the flow around a spinning cylinder in a straight channel.

Chapter 9 summarizes our findings and presents recommendations for future work, in particular in the direction of three-dimensional problems and a realistic implementation of the adaptive-composite-grid strategy.

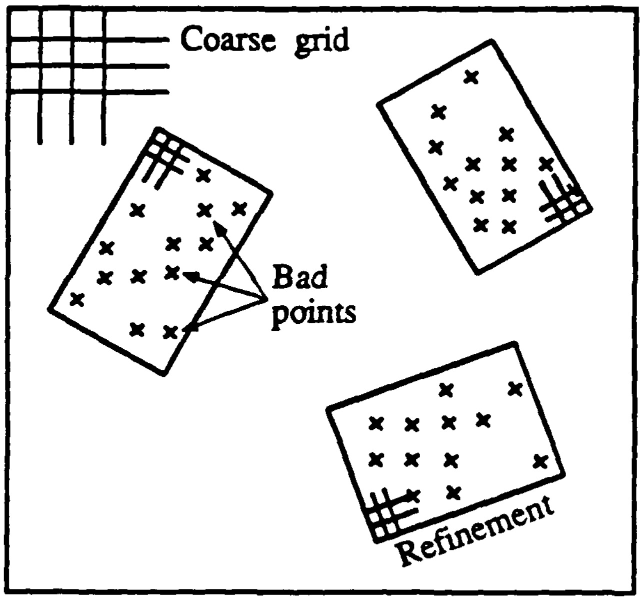


Figure 1.1: Bad points fitted with rectangular refinement patches

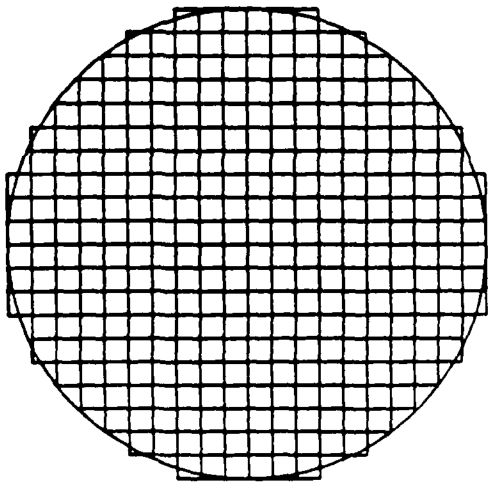


Figure 1.2: Stair-step grid for circular domain

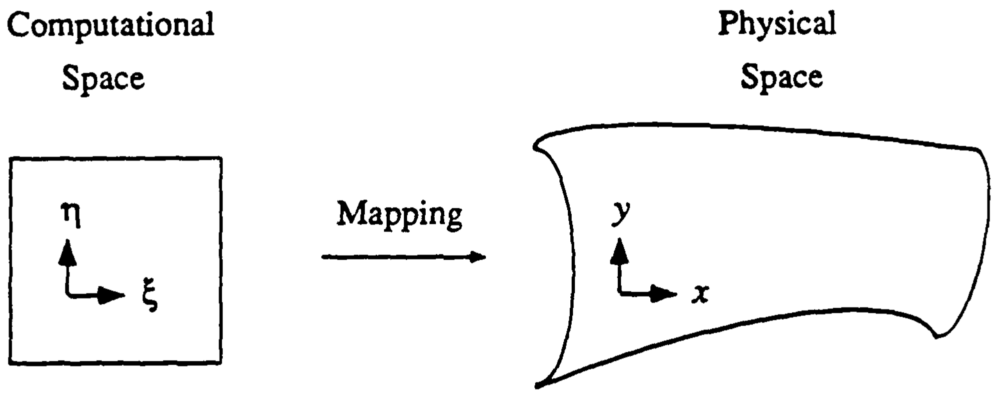


Figure 1.3: Mapping from computational space to physical space

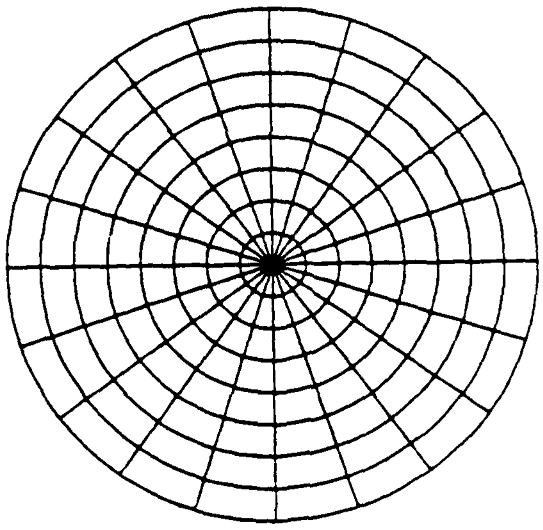


Figure 1.4: Polar coordinates for circular domain

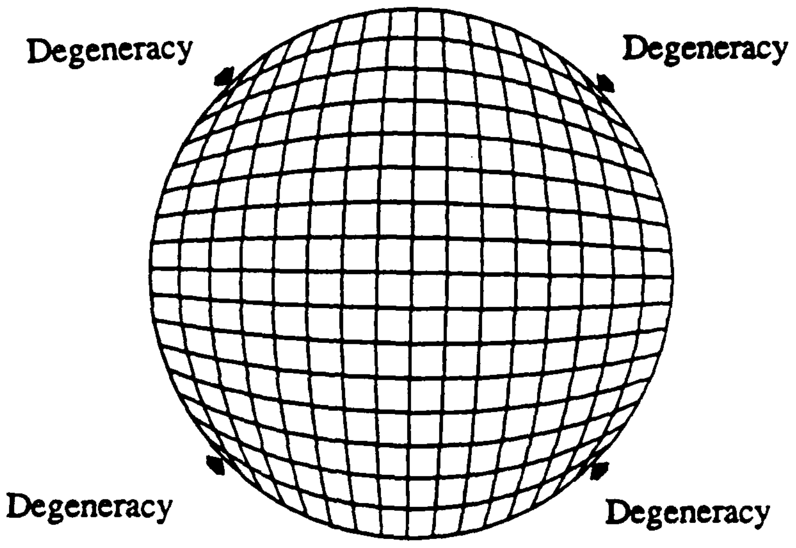


Figure 1.5: Escher-distortion coordinates for circular domain

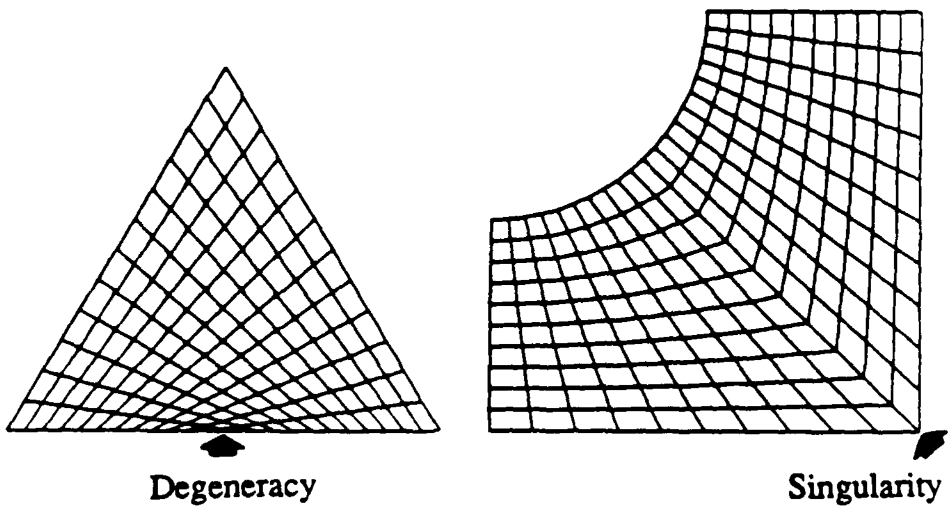


Figure 1.6: Regions with number of sides unequal to four

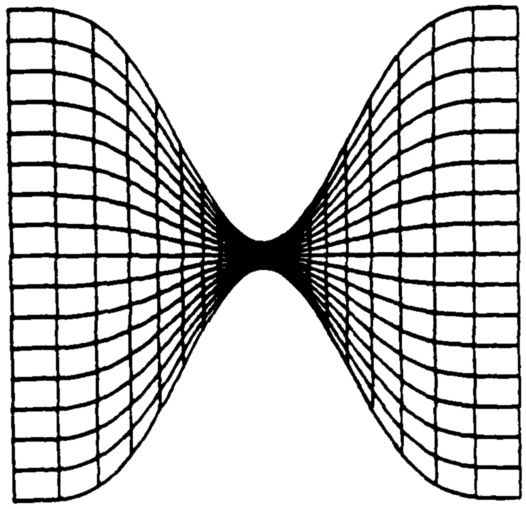


Figure 1.7: Congestion of grid lines in waist

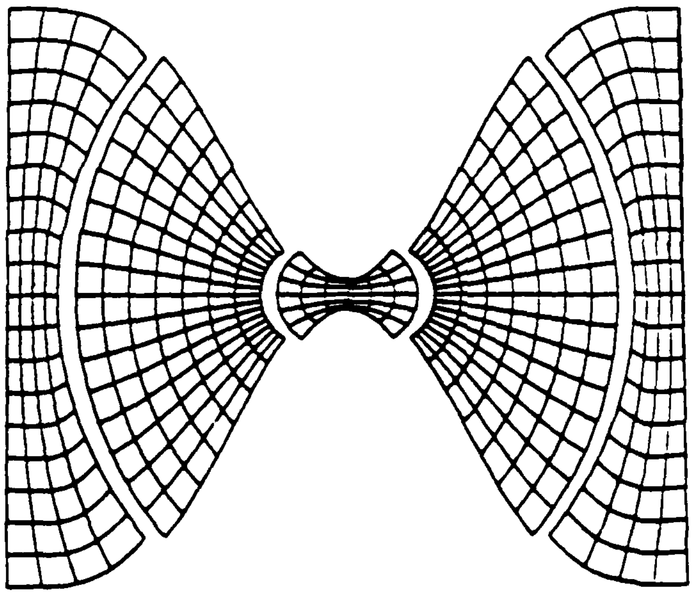


Figure 1.8: Composite grid for bow-tie-shaped domain

Chapter 2

SOLUTION OF FLUID-FLOW PROBLEMS: FOUNDATIONS

2.1 Introduction

In this chapter we lay the foundations of methods for solving fluid-flow problems numerically on multiple domains. Section 2 is devoted to a global description of the kind of physical problem we are interested in solving. Translating the physical problem into a numerical problem on a single grid can be a formidable job, which is even more complicated if several grids are used to represent the solution. Many solution algorithms exist; they are generally iterative.

Section 3 provides tools to discuss and construct methods for computing approximate solutions on multiple domains in a somewhat formal framework; definitions are given of numerical solutions, of different types of convergence, of numerical geometries, of numerical boundary conditions, and of global iteration processes on multiple domains. These definitions enable us to restrict the classes of problem definitions and solution strategies. Fundamental composite-grid philosophies are laid down here.

Section 4 discusses the basic implications of adding solution-adaptive grids to the numerical machinery of composite grids.

Section 5 summarizes the tasks facing the numerical analyst who wants to solve a problem on a composite grid. Here we will take the approach that as much of the numerical problem formulation as possible should be included in the capabilities of a grid-generation system. In that sense, the name *problem*-generation system might be more appropriate.

2.2 Physical Problem

We are interested in solving fluid-flow problems on complex, (possibly) multiply-connected, finite domains. That means that we want to satisfy some partial differential equation $L(w) = f$ in the interior of a region Ω. The boundary of Ω can be divided into an interior part (i) and an exterior part (e): $\partial\Omega = \partial\Omega^i \cup \partial\Omega^e$. Interior boundaries occur, for example, in interface problems (see Fig. 2.1). Boundary conditions can be manifold, although they can evidently all be captured in the single (cryptic) formula $B(w) = h$ on $\partial\Omega$, where B is an arbitrary operator. For simplicity, but without great loss of generality, we will restrict the classes of boundary conditions to:

a) 'Real', independent boundary conditions, for which B and h are prescribed

b) Periodic boundary conditions, for which h is a function of w on some other part of $\partial\Omega$

c) 'Hyperbolic' boundary conditions, for which B and h depend on the solution w in the interior of Ω.

When computing incompressible flows, we encounter the first two types of boundary conditions, although the third is sometimes applied on outflow boundaries of almost hyperbolic character (convection dominated). In unsteady problems, the partial differential equation, the boundary conditions, and the boundary itself may all change in time. In the sequel, however, we will only consider quasi-steady problems, which means we are interested in obtaining a solution on a given domain at a given time (boundary-value problem).

For classification purposes we will divide the **problem** boundaries into :

α) Demarcation or **physical** boundaries (types a and c)

β) **Periodic** boundaries (type b)

For a complete definition of the physical problem we need:

- A (piecewise) geometric description of $\partial\Omega$ according to the above classification (boundary types α or β)
  - for every point on the physical boundary an algorithm to compute B and h from a known interior solution, or a fixed formula for B and h
  - for every pair of periodic boundaries a mapping from the one into the other, plus a functional relationship between every pair of solution values (usually, this will simply involve equality of the solution values)
- A differential equation that holds in the interior of Ω

A function w which satisfies $L(w) = f$ in Ω, and $B(w) = h$ on $\partial\Omega$ is called a **physical solution.** The (normed) space of all physical solutions is denoted by W.

2.3 Numerical Problem

Now that physical problems and their solutions have been defined (in principle), it is time to introduce numerical approximations to them.

2.3.1 Single Domains

In essence, the numerical problem consists of the determination of a finite number of parameters that define a function w^n on Ω. For quasi-steady problems, that function is evaluated at a particular time.

DEFINITION: A function w^n is called a **numerical solution** if, in some sense, $\|w - w^n\|_\Omega$ is small (superscript n indicates 'numerical').

Because $W^n = \{w^n|$ parameters are permissible$\}$ is a finite-dimensional subspace of the physical solution space W, which, for arbitrary boundary conditions, is infinitely dimensional, we can not expect $\|w - w^n\|_\Omega = 0$. The best we can do is construct a sequence of functions $\{w_j^n\}_{j=1,\ldots,\infty}$ with increasingly more parameters (j is the number of *degrees of freedom* or *dimensions*), which converges to the solution if W is a separable space. Or, we can construct a sequence of problems whose solutions, if obtained, converge to the physical solution. This is the basis of theories of **classical convergence**.

Important as they may be, these theories are not the subject of the present investigation. In fact, we shall be satisfied with obtaining one function $w^n_{j_0}$, which we will call **the numerical solution**. Numerical solutions themselves can usually only be computed as limits of sequences of solutions to simplified (linearized) problems. These solutions are called **iterates**. The only requirement on iterates is that they converge to the numerical solution. We call this type of convergence: **iterative convergence**. The only requirement on the sequence of intermediate, simplified problems, including their boundary conditions, is that the final problem have the correct numerical solution.

2.3.2 Multiple Domains

Suppose now that we want to compute a numerical solution on a union of subdomains $\{\Omega_k\}$, with $\Omega \subseteq \bigcup_{k=1}^{K} \Omega_k$. Each numerical subdomain solution procedure has the properties that hold for the global domain, if only because we may set $K = 1$. A numerical solution on the union of subdomains is defined by a set of solutions on the subdomains plus an interpolation scheme in regions of overlap. The interpolation scheme may be simply a selection switch.

DEFINITION: A *union of subdomains* is called **geometrically consistent** if both $\partial\Omega^e = \partial(\bigcup_{k=1}^{K}\Omega_k)$, and $\partial\Omega^i \subset \bigcup_{k=1}^{K}(\partial\Omega_k)$

This means that every point on the boundary of Ω, be it an interior or an exterior boundary point, also lies on the boundary of at least one subdomain, and that the union of subdomains exactly covers the whole physical domain. See Fig. 2.2 for examples of consistent and inconsistent unions of subdomains. To avoid overly cumbersome notation, we will consider only exterior boundaries from now on, unless otherwise indicated. The discussion, however, holds for interior and exterior boundaries alike.

Two basic remarks apply:

- Body-fitted coordinates lead naturally to geometrical consistency for single domains, whereas stair-step grids do not.

- The strategy of blocking out parts of a grid that extend beyond the boundaries of the physical domain —as Chesshire [CHE86] does, for example— does not preclude geometrical consistency, as the excess cells are not part of the subdomain. Boundary tracking is indeed established using a body-fitted grid.

A geometrically consistent covering can always be found, since $\partial\Omega$ is known (*e.g.*, in terms of splines, conic sections, *etc.*). Its definition is contained in the physical model. In the present study we will focus on body-fitted coordinates and geometrically consistent unions of subdomains. Both have to do with easy and accurate representation of boundaries and boundary conditions. It should be stressed, however, that neither is required.

DEFINITION: Boundary conditions on a geometrically consistent union of subdomains are called **functionally consistent** if they lead to a single unambiguous solution when taken to convergence in the classical sense.

> *Intermezzo: Often, a numerical solution consists of a set of discrete node-point values plus a fuzzy concept of interpolation in between. (This is true in particular for finite-difference methods.) In fact, usually the interpolation scheme is intentionally left undefined. This makes the concept of unambiguous numerical solutions, if defined on noncoinciding discrete points in space, rather vague. A way out is to define nonambiguity as a limiting property of discrete functions as the mesh spacing goes to zero.*

We <u>must</u> require functional consistency of boundary conditions to ensure classical as well as general iterative convergence. This does not mean that boundary conditions on overlapping parts of boundaries need to be the same, not even at points that coincide. The only thing that counts is the *solution* on overlapping pieces of the boundary *at iterative convergence*.

We now give the basic definitions that describe iterative stepping procedures for problems on multiple subdomains in the spirit of the Schwarz Alternating Procedure (See Chapters 6 and 8).

DEFINITION: A **global iteration** I_j consists of a walk through an ordered sequence of subdomains: $\{\Omega_{\alpha_{l_j}} \mid l_j = 1, 2, \ldots, L_j, \text{ with } \alpha_{l_j} \in (1, K),\ L_j \geq 1\}$. Here j signifies the iteration number, L is the number of visits to subdomains (this is usually larger than the number of subdomains K), l is the sequence number of the visit, and α_l is the subdomain index pertaining to the l^{th} visit. A visit to a subdomain implies computations on that subdomain. A sequence is an explicit list, or a recipe to generate the list automatically.

This definition is general enough to describe any iterative process. In fact, one might describe any convergent solution process as a single global iteration. This would obviously lead to degeneration of the concept of iteration. Nevertheless, the above definition is useful to describe iteration processes in which a dynamic determination of visits to subdomains takes place.

DEFINITION: An **iteration process** consists of a series of global iterations $\{I_j\}$, with $j = 1, 2, \ldots$.

DEFINITION: A **stationary iteration process** consists of a repetition of global iterations $\{I\}$, which are constant. That means that the list (or the recipe to generate the list) does not depend explicitly on the iteration index j.

For a stationary iteration process to converge, we must have $(1, K) \subset \{\alpha_l\}$, or in words: every subdomain must be visited at least once during every global iteration. Even though the sequence of subdomains is fixed in a stationary iteration process, the problem, solved on a specific subdomain in the sequence, may still change from one iteration to the next (also, the successive problems to be solved on one physical subdomain which appears more than once in the sequence will be different, in general).

DEFINITION: A **stationary iteration process in the restricted sense** is a stationary iteration process in which the numerical algorithm for updating the solutions on subdomains is constant, *i.e.*, does not depend explicitly on the iteration index j.

Even in a stationary iteration process in the restricted sense, the numerical algorithm applied to the same physical subdomain that occurs more than once within one iteration sequence may be different for every occurrence.

A *stationary* iteration process in the restricted sense may converge to the correct numerical solution on a geometrically consistent union of subdomains, **only** if the boundary conditions on $\partial(\bigcup_{k=1}^{K}\Omega_k)$ are functionally consistent. Because the boundary conditions in this case are determined by the same algorithm during every global iteration, we can only ensure iterative (and classical) convergence if we apply the correct physical boundary conditions at every point of $\partial(\bigcup_{k=1}^{K}\Omega_k)$ at least once during every global iteration. By correct physical boundary conditions we mean those numerical boundary conditions which, in the limit of vanishing mesh size, would yield the physical or periodic boundary conditions as described in the section on the physical problem.

DEFINITION: Boundary conditions are called **strongly functionally consistent** if the correct physical boundary conditions are applied at every point of $\partial(\bigcup_{k=1}^{K}\Omega_k)$ at every occurrence in the global iteration.

2.4 Composite Adaptive Grids

The covering of the domain by a set of subdomains that is specified in advance will be called the **basic union of subdomains**. An interesting situation arises if new subdomains Ω_i are added dynamically to (and perhaps later deleted from) the basic union of subdomains, as is the case with the patched adaptive-grid-refinement technique. In principle, we should then abandon the concept of iteration processes as defined before, be they stationary or not. Because of the hierarchical structure of levels of refinement in patched adaptive-grid systems, however, it is possible to extend our definition of, say, stationary iteration processes to the present situation without great difficulty.

DEFINITION: A **quasi-stationary iteration process consists of a** series of global iterations, $\{I_j\}$, **which are constant with respect to the basic union** of subdomains $\bigcup_{k=1}^{K}\Omega_k$, *i.e.*, independent of the iteration index j (again, we may supply a —constant— recipe to determine a global iteration, rather than giving an explicit list).

This definition lends special significance to the basic union of subdomains covering Ω. That is no coincidence, since that set of subdomains is constructed explicitly by user interaction. Analogous definitions can be put forth to extend the notions of stationary iteration processes in the restricted sense, and of strong functional consistency. This effectively decouples the iteration process on the basic union of subdomains from the ones on unions of subdomains on successive levels of refinement. One might consider using the concept of strong functional consistency for patched adaptive-grid systems to indicate that the correct physical boundary conditions are applied at all points of the whole system that lie on $\partial\Omega$. There is no real incentive, though, to insist on strong functional consistency in this sense.

2.5 Grid Generation

It is important to realize that the definition of the numerical procedure to solve a problem using finite differences is composed of several conceptually different stages:

1. Creation of the basic union of subdomains as a strictly geometrical entity.
2. Construction of curvilinear (body-fitted) coordinate systems on the subdomains.
3. Specification of an iteration process on the union of subdomains.
4. Definition of numerical algorithms for updating the solution on each subdomain:
   (a) Construction of a finite-difference mesh on the subdomain.
   (b) Formulation of the set of (nonlinear) equations for the unknowns on the interior lattice points.

(c) Formulation of boundary conditions for all points on the boundary of the subdomain,

(d) Linearization and solution of the set of algebraic equatic ·; pertaining to the subdomain.

Strictly speaking, only (1), (2), and (4a) should be considered in a grid-generation system. Items (3), (4b), (4c), and (4d) are parts of the solution algorithm. However, it makes sense to incorporate at least (3) and (4c) in the grid generation system if we are willing to view that system as the collection of procedures specifying the information necessary to make the computational process during one global iteration unambiguous. One might say that a grid generation system in that sense is a template for computations, with initial values and an interior solution scheme as parameters. We can take this even one step further, including item (4b) in our system. Suppose we have a good physical intuition about the flow field. Then we might want to specify, in advance, different solution procedures in the interiors of subdomains in different parts of the field. The classical viscous/inviscid coupling and the more contemporary zonal-modeling approach to turbulence computations are examples, provided the latter is made fully automatic.

At any rate, these are just physical considerations and hence only contingent upon the location of subdomains within the field, and **not** on the position of the subdomain within the sequence to be visited during a global iteration or on the interaction with other subdomains that might overlap it. As such, incorporation of item (4b) into the grid generation system does not require much information to be specified.

This is not the case with the specification of boundary conditions, which is essentially a numerical affair. New conditions need to be formulated every time a certain subdomain is visited during the iteration process. Moreover, formulation of boundary conditions consists of two distinct parts: <u>what</u> type of boundary conditions to apply (*e.g.*, Neumann, Dirichlet, mixed), and <u>where</u> to get the information from to quantify the boundary conditions (*e.g.*, other subdomain(s), external physical boundary conditions).

To alleviate the burden of expressly specifying the entire iterative process by hand, we will make the following simplifying assumptions:

- consider only stationary iteration processes,
- consider only strongly functionally consistent boundary conditions for the subdomains.

These simplifications leave us with specifying different boundary conditions for points on $\bigcup_{k=1}^{K}(\partial\Omega_k)\backslash\partial(\bigcup_{k=1}^{K}\Omega_k)$ every time a specific subdomain is visited within the (constant) global iteration, which involves a still considerable amount of work. A compromise may be reached by providing some default, robust —but therefore nonoptimal— algorithm to determine boundary conditions, combined with an option to overrule that algorithm with a preferential setting.

Some general remarks concerning the components (1) through (4) of the grid generation system should be made before embarking on a more detailed description of the system in a subsequent chapter.

(1) and (2): It is possible, in principle, to define a subdomain as a simple, structureless patch on a surface by supplying a periodic, connected, one-dimensional set of points bounding the patch (and similarly for a block in space). In practice, however, a metric is almost automatically introduced by parameterization of the bounding curve. This is particularly true for the algebraic grid generation procedures on which we will focus. Control through piecewise specification of the boundaries with independent scaling and a still arbitrary interpolation in the interior of the domain is possible, but has a limited range. Hence, items (1) and (2) are not completely separable, although they are conceptually unrelated.

(4c): The phrasing of this item is intentionally left undefined with respect to the expression *'for all points on the boundary'*. We may choose to specify boundary conditions for lattice points on the boundary only (*i.e.*, after discretization), or we may define a (piecewise) continuous boundary-condition function using the parameterization of the boundary. In the latter case we would first have to determine the value of the parameter at a discrete point before the boundary condition can be evaluated. The discrete approach has the advantage that no information needs to be defined for points that are not on the lattice anyway. The continuous approach has the advantage that grid refinement can be treated very naturally, and it is the one that is being used in this study.

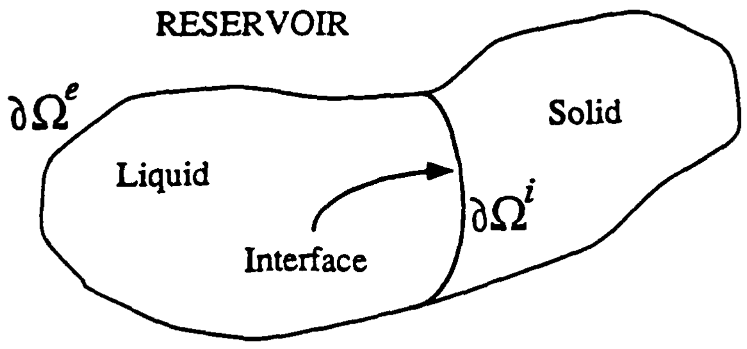


Figure 2.1: Physical problem with interior boundary

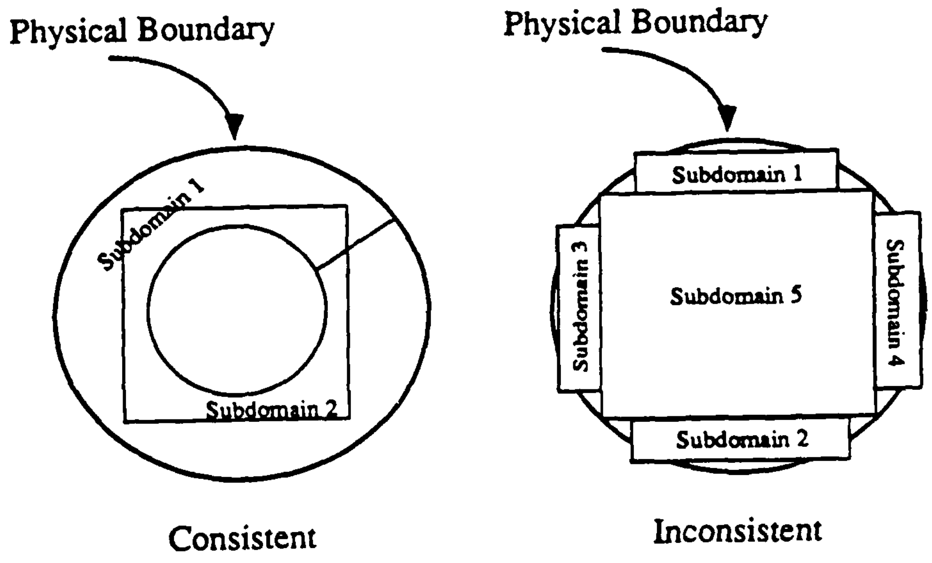


Figure 2.2: Geometrically consistent and inconsistent unions of subdomains

Chapter 3

DATA STRUCTURES FOR COMPOSITE GRIDS

3.1 Introduction

In the previous chapter we discussed the principles of numerical problem definition and solution on a single and a composite grid. When it comes to implementation, we have to worry about how to *represent* the problem in a computer. This chapter deals exclusively with problem representation, *i.e.,* with the data structures needed to formulate a composite-grid problem.

A separate program —called M*E*S*H, described in Chapter 5— is used to create a composite grid. This program writes out a composite-grid data file that serves as input to a computational program that calculates a solution on the composite grid. The data file contains a high-level description of a composite grid, or rather, a set of instructions which is used by the computational program to construct the composite grid. Because the description is of such a high level, it is easy to change the shape, size, or definition of a composite-grid with few changes in the data file.

This is important at a point where M*E*S*H has not yet been implemented and where the data file is created manually, which is the case at the time this report is being written.

The remainder of this chapter is as follows: In Section 2 the elementary descriptors of a composite-grid are introduced. We limit ourselves to the kinds of problems mentioned in the last section of the preceding chapter, and to two space dimensions. Most descriptors are very fundamental, and are applicable, in principle, to any numerical solution procedure for boundary-value problems. In Section 3 generic data structures that are used to define the composite-grid problem are discussed. In Section 4 the definitions of the data structures used in this project are given. Most of these data structures will already have been introduced in concept in Section 2. Others, in particular the representation of curves in terms of fundamental curves *plus* an affine coordinate transformation, are discussed in detail in Appendix B.

3.2 Composite-Grid Concepts

Composite-grid calculations are complex operations. As we will see in this section, an unexpectedly large number of new constructs needs to be introduced to provide an environment general enough for our applications. The reader should keep in mind that the distinctions between these constructs exist for single-grid computations as well. However, in the single-grid case it is possible to combine certain functionally different components without ambiguity or confusion; this cannot be done for composite grids. The merging of functions typically takes place implicitly, without the numerical analyst realizing it (nor needing to realize it).

When creating a composite grid in an *interactive* way (our approach), it is natural to start with a screen on which the contours of the physical domain Ω are drawn. Because we only allow geometrically consistent unions of subdomains, that contour will be filled precisely by component grids. Thus, in a set-theoretic sense we may say: $\partial(\bigcup_{k=1}^{K}\Omega_k) = \partial\Omega$, if there are K component grids. Functionally, however, this is not accurate. Component grids are separate entities, with independently specified geometries and boundary conditions. One might describe the geometrical division of the domain Ω into subdomains as a mapping D from the Cartesian product (not the union!) of the K subdomains into the physical domain, so D : $\prod_{k=1}^{K}\Omega_k \mapsto \Omega$. Ambivalence will only occur if we try to project $\prod_{k=1}^{K}\Omega_k$ onto Ω, that is, by looking at the screen with the component grids (or parts of them) drawn on top of each other. In that case the distinct concepts of *physical* and *computational* space are mixed up in one picture.

To avoid this dilemma, we will define the initial stage of the grid generation process as a series of operations in physical space. The segments of $\partial\Omega$ that are parameterized by a single contiguous interval of a parameter s will be called **curves**. A curve is the graph of a vector function $\mathbf{c}$ of one variable s, so $\mathbf{c}$: $s \rightarrow \mathbf{c}(s)$. We will assume that c is continuous, although this is not strictly necessary. It follows that curves are continuous lines on the screen. Using the contour of the physical domain as a starting point we can add more curves in physical space, that is, draw more lines on the screen (how new curves are generated will be discussed in Chapter 4 and Appendix B). These lines have the same status as the ones making up the contour and are, in that sense, indistinguishable from them. In short, every line drawn in physical space is called a curve, independent of its specific function; curves are strictly geometrical entities. Several curves may be designated as a unit for the purpose of manipulating them as a single entity (for example to make a copy of them elsewhere on the screen). Upon copying, all components of a unit receive new names.

The relation between curves and the physical problem is established by the introduction of subcurves. Every curve consists of one or more bordering **subcurves**, which are contiguous intervals in the parameter describing the curve, tied to exactly one of the following **physical roles:** *physical boundary, periodic boundary,* or *auxiliary*. The first two roles are obviously derived from the original problem, whereas the third role is reserved for those curves, or parts of curves, that are not on the boundary, $\partial\Omega$, of the problem domain. Curves and subcurves together completely define the physical problem and set the stage for the generation of a composite grid.

So far, all we have done is draw some lines in physical space (*i.e.*, on the screen) and hand out roles to segments of these lines. Now we will take the step to the subdomain spaces by defining a **(grid) side** as a set of entire subcurves, adjacent in physical space, with one global, continuous parameterization.
Five remarks readily apply:

- subcurves comprising one side need not be part of the same curve;
- one subcurve may appear in the definition of several sides (this may happen when grids overlap);
- sides do not have roles; they are the equivalent in subdomain space of curves in physical space, which means they are strictly geometrical entities;

- the parameterization of a side is derived directly from the parameterizations of the composing subcurves; we will only use linear **rescalings** (shifts and multiplications of scalar parameters) of the latter to arrive at the former. A logical default procedure to parameterize subcurves within sides is to make the curve speed continuous across boundaries between subcurves (curve speed is defined as the increment in the arc length along the curve per unit of parameter);

- we might have to divide a semantically consistent single subcurve (*i.e.*, a subcurve with one physical role) into several new subcurves if we want to control the size and the shape of a grid, because only complete subcurves are allowed as constituents of grid sides.

The dual role of subcurves is apparent from the above, although there is no danger of ambiguity. In fact, the duality is intentional, as it enables us to connect the physical problem to the numerical problem.

Analogous to the definition of subcurves in physical space, we define **subsides** as contiguous, nonoverlapping intervals in the parameter describing the side, tied to exactly one of the following **subdomain roles:** *physical boundary, periodic boundary, reentrant boundary, interpolation boundary,* or *none*. The consequences of this definition deserve careful consideration.

The first two roles essentially establish the physical validity of the set of problems on the subdomains. As mentioned in Chapter 2, we only allow strongly functionally consistent boundary conditions, which means that physical boundary conditions are applied at all grid points that lie on the boundary of the physical domain. By designating a certain section of a grid side as a physical boundary, we can automatically retrieve, through the parameterizations of the pertinent side and curve, the applicable boundary conditions. For periodic boundary conditions a slight problem arises, as a target point gets its information from a (different) point in the physical domain, which might be contained in more than one component grid. In that case, which is similar to the case of interpolation boundaries, additional information is needed. In fact, interpolation boundary conditions can be viewed as special cases of periodic boundary conditions, if the latter are imposed on different grids.

Reentrant and interpolation boundaries are strictly computational entities. A reentrant boundary arises when a branch cut is introduced in a grid that connects to itself in physical space, in which case no physical boundary is present. Reentrant boundaries can always be avoided by splitting the self-connected grid into several grids. This reduces reentrant boundaries to simple interpolation boundaries.

Interpolation boundaries show up when two or more component grids overlap —or just touch— and need to exchange information.

Role *none* simply means that no role has been explicitly assigned (the default type). Eventually, every subside must have a nontrivial subdomain role.

Things are now pretty much settled; curves are lines drawn on the screen, which are dissected into subcurves that perform physical roles and figure as building blocks for grid sides. Grid sides, then, are divided into subsides that have subdomain roles. The only thing left is to forge the grid sides into grids and to define a global iteration process on the composite grid. There is a catch, however.

As we perform a complete walk through the set of subdomains, *i.e.*, as we carry out one global iteration, the boundary conditions on periodic and interpolation boundaries can be, and in general will be, different each time we visit the same grid. The subdomain roles of these boundaries obviously do not change, but the grids from which information is received (donors), will. A way to express this is to say that the **numerical roles** of parts of subsides will be different every time a grid is visited in a global iteration. A numerical role is composed of a subdomain role plus an identification of the donor grids (note: several grids may donate data to one point; it is then passed through some kind of filter). For convenience, a third piece of information will be added to the numerical role, namely the geometrical connection with the donor grid(s), which can assume the values *touching* or *overlapping*.

For a subside on $\partial\Omega$ there is no difference between its numerical, subdomain, and physical role, as there is no donor. By definition, a reentrant boundary receives information from the grid it is on, and touches itself, so that explicit specification of a numerical role is redundant. Consequently the only subsides with externally defined numerical roles are periodic and interpolation boundaries.

It follows that one subside may have several different numerical roles. Moreover, these roles can change with every visit to the particular grid in one global iteration. We might propose yet another subdivision, say of subsides into *numerical subsides*, which all have a single numerical role. A new subdivision would then be defined for every occurrence of a grid in a sequence of visits, giving the numerical subsides a dynamic appearance.

We will take a slightly different approach, though, which avoids the introduction of a new level of complexity, and which is more static. The idea is to chop up subsides into coherent parts which receive information from one or more other grids. These parts are still called subsides, and a list of possible donor grids, plus the appropriate geometrical connections, is supplied with them (this list need not be exhaustive). The mechanism to single out the actual donor grid(s) from this list will be provided as an algorithm contained in the computational program, rather than as an explicit enumeration. Evil-spirited (or pragmatic!) minds may still want to specify expressly which grid(s) on the list will be active, for example to be sure that a particular grid indeed acts as a donor. This deterministic feature may be incorporated in the algorithm, as long as the identification of the subside and the place of the grid in the grid sequence (global iteration) are passed to it as parameters. The basic idea behind all this remains that at any given time all the points that lie on a certain subside will receive information from one donor grid only, or from a fixed combination of them.

Boundary conditions themselves (B and g in Chapter 2) may also be determined by an automatic procedure for all subsides other than those on $\partial\Omega$.

One special kind of structure, which might be categorized as "an interpolation subside of the second kind", has not yet been discussed. It concerns additional interior (shadow) interpolation points that are used in certain applications [SKA87] in which extra smoothness of interpolated data is needed (see Fig. 3.1). These structures will not be incorporated into the grid-generation system, although they may very well be used in conjunction with it. A pragmatic reason for this is that it would be rather tedious for a user to create explicitly all the subsides and related constructs that define these shadow points, on top of defining the primary contours of the grids. Moreover, the location of the points (which generally lie on coordinate lines in the interior of a grid) is fixed by the mesh size and the interpolation procedure used to construct a curvilinear coordinate system on the grid, and is usually not known in advance.

A more fundamental reason for not defining the shadow points in the grid generation system is that their role is to smooth the interpolated data in the direction normal to the grid boundary. Therefore, shadow points in the interior of the grid should be related as closely as possible to the adjacent boundary points (same type and source of interpolated data). This is done most easily and consistently by assigning to an interior point the numerical role of *the corresponding point on the* boundary. This task can be performed by the computational program and need not be contained in the grid generation program. There should be tools available in the grid generation program that enable the user to define grids without too much effort for which the above strategy works (*e.g.*, diagnostics that signal if the amount of overlap between grids is not sufficient).

Summary

In this last part of the section we will summarize the main new constructs that have been introduced to describe a composite grid.

Curve: A parameterized line segment in physical space, demarcating the physical problem domain, or serving as a tool to dissect the domain.

Subcurve: A section of a curve, which has a specific physical role (boundary condition) if it is on the problem domain boundary.

Side: A complete side of a grid, composed of a number of adjacent subcurves.

Subside: A section of a side with a specific numerical role.

Curves and subcurves are entities that live in physical space. They have nothing to do, in principle, with the numerical problem, but define the physical problem in terms of the geometry (curves) and physical boundary conditions (subcurves). Sides and subsides are inhabitants of computational space. Their only connection with physical space is through the construction of sides using subcurves. They define the numerical problem in terms of geometry (sides) and the numerical boundary conditions (subsides).

The following table characterizes the functions that are fulfilled by the abovementioned constructs.

Table 3.1: *Functions of composite-grid components*

		SPACE	
		physical	*computational*
ROLE	*geometry*	curve	side
	boundary conditions	subcurve	subside

An illustration of the description of a problem using a two-component composite grid is shown in Fig. 3.2. Although the geometry is very simple, all four elements in Table 3.1 are needed, and no two sets of elements exactly coincide. Note that subcurve b occurs in the definitions of two different sides (tick marks indicate the subcurves comprising a grid side).

3.3 Data Structures

The preliminary work providing the foundation for the data structures to describe a composite grid is now completed. As mentioned before, the structures presented in this section will exist somewhere on a file produced by the grid-generation program M*E*S*H. Consequently, storage is sequential and the program reading the file must be informed explicitly when new structures are encountered. This can be accomplished by writing the reserved word **structure:** at the beginning of each line where a new entity is defined, followed by the specific **type of the entity**. When a particular structure can occur more than once within a larger structure, say curves within a composite grid, the need arises to distinguish the different instances of that structure by assigning labels. For this purpose we write the reserved word **name:**, followed by the particular *name* of the structure. A header for a curve in a file containing several curves may read as follows:

structure: curve **name:** *french* .

In Fortran77 the only 'higher-level' type of data structure is that of 'array'. Hence, the only reasonable way to store several instances of a certain type of structure is by letting names correspond to indices in an array. For efficiency reasons we omit the assignment of names in the current implementation altogether and read in structures sequentially in subsequent array positions. This does away with the necessity of keeping reference tables. Now we have to know in advance how many structures of the same type are going to be read, so this number becomes part of the data structures.

In addition to defining basic entities like grids and curves, there is a need to combine several of these atomic parts into bigger functional units. For this purpose the concept of **objects** is introduced. Objects consist of grids, curves, and other objects (and perhaps entities contained in the *graphic aids* category —see Appendix B). Any of these elements may belong to only one object. Thus, a strictly hierarchical organization is obtained. Grids and curves in an object that are not contained in deeper nestings of objects are called **radical** with respect to their containing object. For consistency reasons, all grids and curves at the base level of the composite grid that are not contained in user-defined objects will be considered radical with respect to a *root*-object. This root-object is created by the grid-generation system and contains the radical structures plus all objects, defined at the base level. Its syntax is that of a regular object.

Objects exist to make it easy to perform operations on sets of functionally coherent structures: objects may be moved or copied. In the latter case a new object name must be supplied. Difficulties would arise if an object to be copied contains a grid, but not all the curves that are referenced in the grid. This situation will be precluded: an object must be completely self-contained, which means that no references to external radicals or objects may occur.

Several levels of sophistication are conceivable in the definition of objects. They may be used as aliases for whole groups of elementary structures, which are copied as a unit when a copy of the object is made ('hard copy'). They may also serve as templates, in which case only *references* to previously defined structures are created, supplemented with certain changes in the parameters of these more basic structures. Then any copy of the object changes as the original changes ('soft copy'). 'Hard' and 'soft'-copying of objects may even be combined. In the currently envisioned implementation only hard-copying is allowed (all grids and curves in the composite grid really exist and are not hidden in references).

Objects are a convenient device for the grid generator; they define a composite grid almost completely, making the structures *grid-set* and *curve-set* (defined below) redundant with respect to the grid generation program. On the other hand, objects have no real impact on the data structures that are used by the computational program, which does not care about higher levels of structured information.

As a result of the differing needs of the computational program and the grid-generation program, it makes sense to create separate data files for each. Typically, a complete composite grid will be generated in several sessions, between which only intermediate data files, to be used by M*E*S*H, are written. Only when the grid is finished will we want to write the input file for the computational program. In the following subsection, definitions of data structures occurring in both files are given. The formats of corresponding structures in the two files may differ slightly. The ones described here refer to the data structures in the file that is read by the computational program.

Definitions

It makes sense to view a complete composite grid as one big record, comprised of several smaller records of variable length. They are: `global-iteration`, `object`, `grid-set`, `curve-set`, and `graphic-aids`. Some of these records in turn are divided into yet smaller records. Nested records like this were called **structures** above. The following is a list of the structures (and their respective components) that make up a composite grid. It is understood that every multiply-occurring structure has a unique identifier (a *name*, or index) attached to it, so no more explicit mention will be made of that.

A complete example of a data file describing a composite-grid problem for two overlapping grids is presented in Appendix A. Comments are added to make the file more readable; these are ignored by the computational program. The data structures defined below are most easily understood when studied together with the example.

structure: `composite-grid`

Computational program	*Grid generation program*
- Structure: `curve-set`	- Structure: `object`
- Structure: `grid-set`	
- Structure: `global-iteration`	- Structure: `global-iteration`
	- Structure: `graphic-aids`

structure: `curve-set`

- The number of curves used to describe the composite grid
- A list of all the structures: `curve`

structure: `grid-set`

- The type of inter-grid communication procedure (see Chapters 6 and 8)
- The number of component grids used to describe the composite grid
- A list of all the structures: `grid`

structure: `object`

- The number of curves contained in the object
- A list of the names (indices) of all the curves contained in the object
- The number of grids contained in the object

- A list of the names (indices) of all the grids contained in the object
- The number of (sub-)objects contained in the object
- A list of all these contained structures: `object`

  (Note: when an object is copied or moved, the user may change parameters of the elementary components (curves and grids), and may supply an affine transformation (see below) which relocates/reshapes the object in space. These parameter changes and/or transformations are incorporated explicitly in the definitions of the elementary components, and are not stored as global changes to the object. Thence, as the original object changes, the copy stays the same)

structure: `global-iteration`

- An indicator stating whether the global iteration is defined explicitly in terms of a list of grids to be visited sequentially (status: *manual*), or by the use of a traversal algorithm (status: *automatic*)
  - If status is *manual*: the length of the chain of grids to be traversed, plus the actual ordered list of grids,
  - If status is *automatic*: an identifier specifying which algorithm will be applied.

structure: `graphic-aids`

- TO BE DEFINED

  Note: **`graphic-aids`** consist of auxiliary information that is useful in building a composite grid, but that is not really a part of it.

  Examples are:

  - Command status, saved from a previous grid generation session,
  - Magnification factor and window coordinates in a diagnostic session.

- Marked points, defined by the user as reference points, that are not part of any other structure (*e.g.*, center of curvature, **midpoint**).

structure: curve

- An indicator showing whether the curve is *user-defined* or *standard*
    - If *user-defined*: the name (index) of the corresponding Fortran functions on a separate Fortran file (**usrfun.f**),
    - If *standard*: the name (index) of the corresponding predefined Fortran functions on a separate Fortran file (**stdfun.f**), plus a list of parameters fixing the functions, preceded by the length of the list.

  Note: Four parameterized functions are required for every curve, that is: $x(s)$, $y(s)$, and the derivatives $x_s(s)$, and $y_s(s)$, where s is the parameter on the curve. They are all uniquely identified by one common name (index). The number of *Fortran* functions required to describe a curve geometrically need not be four, as we may choose to combine several of the function evaluations for efficiency reasons. In the current implementation a subroutine is defined which returns four arguments. A switch is built in, so that arguments are not computed when they are not needed. Whenever new standard functions or user-defined functions are added to the existing libraries, the files **stdfun.f** and **usrfun.f** must be recompiled and linked

- The parameters of the affine transformation $\tilde{\mathbf{r}} = P\mathbf{r}+\mathbf{q}$, which maps points $\mathbf{r} = \binom{x}{y}$ into $\tilde{\mathbf{r}} = \widetilde{\binom{x}{y}}$ and thus allows us to shape new curves based on previously defined Fortran functions. P is a (2x2)-matrix and $\mathbf{q}$ is a (2x1)-column vector. Together they are defined by six constants. The original curve (default) is regained by setting P equal to the identity operator I, and by setting the offset vector $\mathbf{q}$ equal to $\mathbf{0}$, so we need not introduce a special indicator to mark this curve

- The number of subcurves that make up the curve

- A list of all the pertinent structures: **subcurve**

structure: `subcurve`

- The physical role of the subcurve
  - If *physical* boundary: the name (index) of the user-defined Fortran function on a separate boundary-condition file (`bcfun.f`). Note: must be compatible with the parameterization of the curve
  - If *periodic* boundary: the parameters describing the affine transformation that relates the boundary to the corresponding target curve in physical space
- The parameter interval, δs, it occupies within the curve

structure: `grid`

- An indicator showing whether or not the grid is *complete* and has been defined in a *consistent* way
  - If *incomplete* or *inconsistent*: a checklist of the status of all parts of the grid-definition process
- The geometrical type of the grid, *i.e.*, warped triangular, quadrilateral, hexagonal, *etc.*
- The number of sides defined so far (at most the maximum number of sides pertaining to the geometrical type of the grid)
- A list of all the structures: **side** of the grid, plus their respective relative locations (for warped quadrilaterals we may use the locations *bottom*, *top*, *left*, or *right*; in the current implementation, only four-sided grids are allowed, which are read in the order defined above)
- The number of mesh cells in the (two) coordinate directions
- Sets of stretching and shrinking factors in the two coordinate directions (by default no stretching is applied)

- The type of mapping used to interpolate coordinates in the interior of the grid from the parameterizations of the sides
  - If *transfinite interpolation*: the order and type of the interpolation (initially, only Coons Patch [FAU81] will be provided)

structure: `side`

- The number of subcurves comprising the side
- A list of the names (indices) of all the subcurves, plus for every subcurve the parameter interval δz it occupies within the side, the name (index) of the curve on which it lies, and the sequence number of the subcurve within the curve
- The number of subsides comprising the side
- A list of all the pertinent structures: `subside`

structure: `subside`

- The numerical role of the subside. If *periodic* or *interpolation*:
  - The number of candidate donor grids
  - A list of the names of all the candidate donor grids, plus for every donor the geometrical connection with the receiving grid side (*overlapping* or *touching*)
- The parameter interval, δz, it occupies within the side

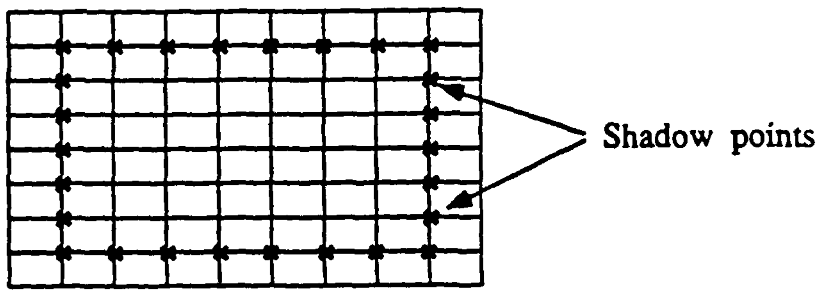


Figure 3.1: Interior (shadow) interpolation points for extra smoothness

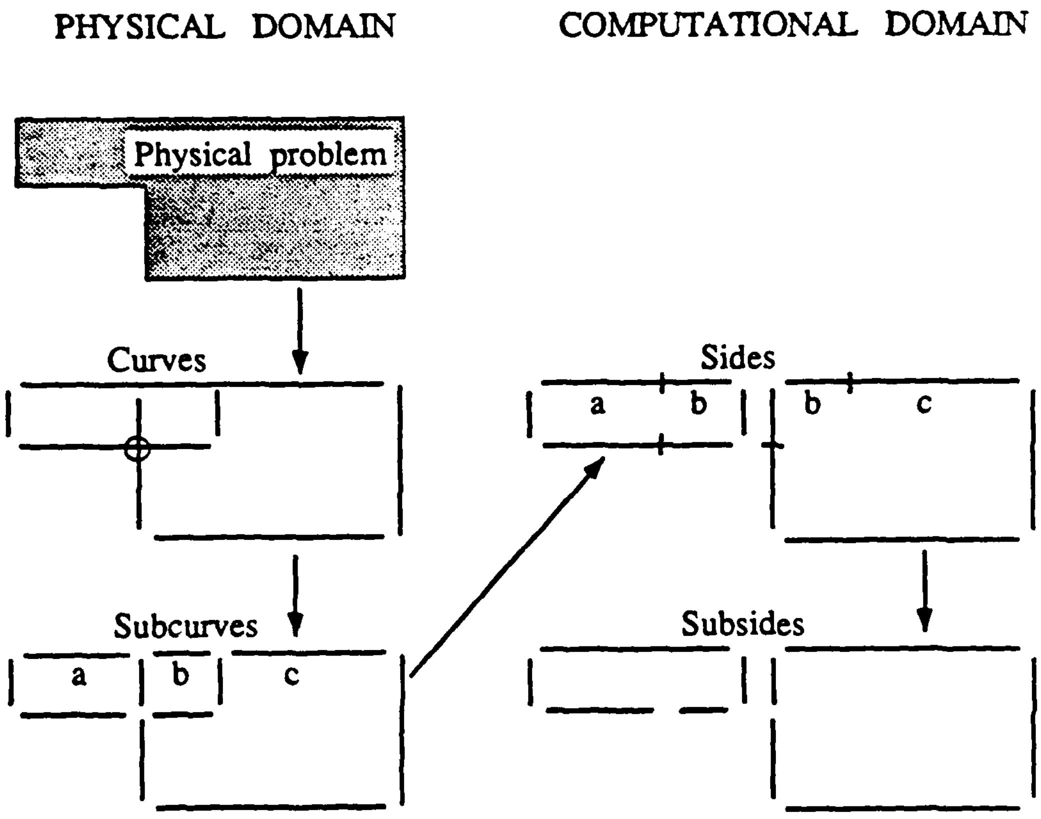


Figure 3.2: Composite-grid concepts for simple two-grid geometry

Chapter 4

M*E*S*H; CONSTRUCTION OF A COMPOSITE GRID

4.1 Introduction

The previous chapter on data structures may have left the reader puzzled about how to input the items needed to specify a composite grid. It takes considerable effort to define a composite-grid problem consistently, even in the simple case given in Appendix A. The problem is that we have to perform many tasks by hand, such as constructing intersection points between curves, specifying geometric transformations numerically, remembering which subcurve is on which curve, *etc*. These things, which are hard for the user, are simple for a computer. What a human is good at, is giving high-level commands and creating and manipulating spatial objects visually and conceptually, whereas computers handle the computing and data management efficiently.

To combine these two qualities effectively, a visual, interactive system called M*E*S*H (Mesh Engineering System for Hydrodynamics), has been conceived to aid the user in specifying a composite grid. Whereas a lot was learned from work by researchers like Eiseman [EIS79], Chesshire *et al.* [CHE86], [BRO89A], [BRO89B], and Thompson [THO87], none of their systems was deemed sufficiently flexible in terms of user interaction, graphical tools, or data structures.

Existing geometrical-design packages also cannot do the job, as their data structures are of a different type and are meant for design and not analysis, which makes them hard to use as graphical assistance systems *given* a geometrical description of the physical problem boundary. Moreover, their way of representing geometrical objects is geared towards shape, and does not care about the way they are 'filled in', *i.e.*, how lattices are generated in the interiors of grid patches. The latter is important for algebraic grid-generation procedures, which have our prime attention.

The reasons for choosing algebraic grid-generation algorithms are several:

- Simple, explicit formulas define mappings from computational to physical space. This makes it easy to calculate the coordinates of a point in computational space on one grid, given its computational coordinates on another, an operation that is of vital importance on multiple grids. In the current composite-grid solution program, the public-domain MINPACK routine `hybrj1` (a root-finder for systems of nonlinear equations) is used to invert coordinate mappings,
- Algebraic methods are trivial to implement, and, for domains that are not too contorted, robust. As one of the important goals of the composite-grid strategy is to create systems of grids, each of which has a simple shape, good grids can be guaranteed. An important class of algebraic grid-generation methods is defined through transfinite interpolation [THO85], the simplest example of which is the Coons Patch [FAU81]. Coons Patch is the currently implemented method.

A disadvantage of algebraic grid generators is that they generally do not yield orthogonal grids. However, loss of orthogonality is not a real problem, as we can construct grids with such flexibility that near-orthogonal grid line intersection can almost always be achieved.

Finally, it should be emphasized that, although algebraic grid generation suits our purpose and is convenient, it is not essential that we use it, and the design of M*E*S*H does not depend on it.

4.2 M*E*S*H Design Philosophy

The practical problem facing the engineer who wants to employ a composite grid for solving fluid-flow problems is the following. Given a (parametric) description of the contour of the physical domain of interest plus the pertinent boundary conditions, how do I divide the domain into reasonable subdomains and store information regarding curves, subcurves, grids, grid sides, subsides in the easiest fashion possible?

In order to unburden the user as much as possible without sacrificing the expert eye, M*E*S*H should be equipped with many tools that enable reusage of information in flexible ways. In addition, M*E*S*H should provide the user with a sufficiently large arsenal of standard tools for composing grid systems.

For example, a simple way of constructing a composite grid for the kidney-shaped domain shown in Fig. 4.1a would be to define a second curve that is a COPY of the contour, SHRUNK towards a geometrical center in the interior of the kidney (Fig. 4.1b). After that, the standard tool PERPENDICULAR could be invoked to cut up the 'collar' between contour and shrunk image (Fig. 4.1c), and, finally, a plain, rectangular grid could be created to cover the hole in the middle by drawing straight lines using the STRAIGHT facility (Fig. 4.1d). Should we want to create a second kidney for completeness, the REFLECT option would do the job adequately with only few key strokes –or mouse clicks– (Fig. 4.2). The latter is an example of a high-level command, as it creates copies of all the components contained in an object.

Some simple operational rules that were found *important* when examining other grid generation systems (most notably, they were found violated) are:

- One should be able to store and retrieve incomplete information. For example, one might first want to generate all grid patches without specifying the number of grid cells in both directions,

- It should be possible to retrieve information numerically that has been stored, such as the coordinates of the corner point just defined,

- It should be possible to end a composite-grid definition session at any intermediate stage, which means that the status of the session should be saved.

All these requirements can be met if a composite grid is viewed simply as a data structure that is created and filled in dynamically.

4.3 User Interface

In this section the way M*E*S*H looks to the outside world is presented, including some basic rules for manipulating information. Although the system does not exist physically at the time of this writing, its output is already being mimicked and used for composite-grid computations. Hence, M*E*S*H is not just wishful thinking.

M*E*S*H is a hybrid system which is in part command-driven, and in part question-driven. The following function types are available to manipulate grid-related data:

Modes: A mode is an environment which is entered or exited explicitly. The mode is entered by simply typing its name, or selecting it from a menu. Exiting is established by typing EXIT. Entering a mode leaves the control with the user (command-driven). There are no parameters. There may be a prologue and/or an epilogue in which certain questions should be answered (such as *what the target construct* will be). The names of MODES will be printed in roman capital letters.

Tools: A tool is invoked to enter a question/answer session with the system if a certain fixed format or order of actions is desired. The session is started by typing the name of the tool. Termination is done by the system, which then returns control to the calling mode. There are no parameters. Questions may be asked in terms of menus. The names of *TOOLS* will be printed in slanted capital letters.

Commands: A command is issued to bring about a certain effect immediately. A fixed number of parameters is supplied with the command. Upon execution of the command the control is returned to the calling mode; the system is ready to accept new commands or other interactions. The names of `COMMANDS` will be printed in typewriter-style capital letters

For clarity a certain hierarchy is imposed on the system, similar to the one for the storage and linking of directories under a UNIX operating system. The modes

-like directories- are arranged in a tree structure, with the tools and commands -like programs- embedded in the branches. All functions are local, that is, they pertain to the particular mode in which they are invoked, with the exception of the diagnostic mode SHOW and the screen operation tool *ZOOM*. These can be used at every point in the system. Note: every mode or tool has a HELP option.

In a powerful graphics-assisted system several ways should exist to input data.

Mouse: One can pick points on the screen or identify entities such as curves and grids by *clicking* them. Other functions are: find the point on a curve closest to a clicked point on the screen, define a window -for instance for zooming- by clicking the corners of a box, *drag* a curve across the screen (translation and rotation), *etc*.

Keyboard: Alphanumerical values are inputted through the key pad (unless a stored datum is reused, which can also be recalled by identifying its location in a (local) storage bin *-scratch pad-* by the mouse). Examples of alphanumerical data are: coordinates of a point, name of a grid, number of mesh cells, *etc*.

Combination: Certain data are inputted most conveniently by a combination of (alpha)numerical values and locations on the screen. One may construct a line perpendicular to a curve, emanating from a point on that curve (defined by the mouse), and extending by a certain length (inputted numerically).

It should be possible to specify mouse-generated information through the keyboard as well. Conversely, it should also be possible to input most of the keyboard data by using the mouse. However, complete symmetry should not be attempted, since the creation of names or the specification of a number of grid cells through mouse action would be rather awkward.

A note on menus: In many cases it will be appropriate to present the user with a menu of options. Selection of an option is possible either by mouse or through the keyboard. In case of a tool one must choose an option from the menu if it is present. In case of a mode one may ignore the menu.

As the description of all the system functions and the way they fit together is rather lengthy, the reader is referred to Appendix B for a detailed account.

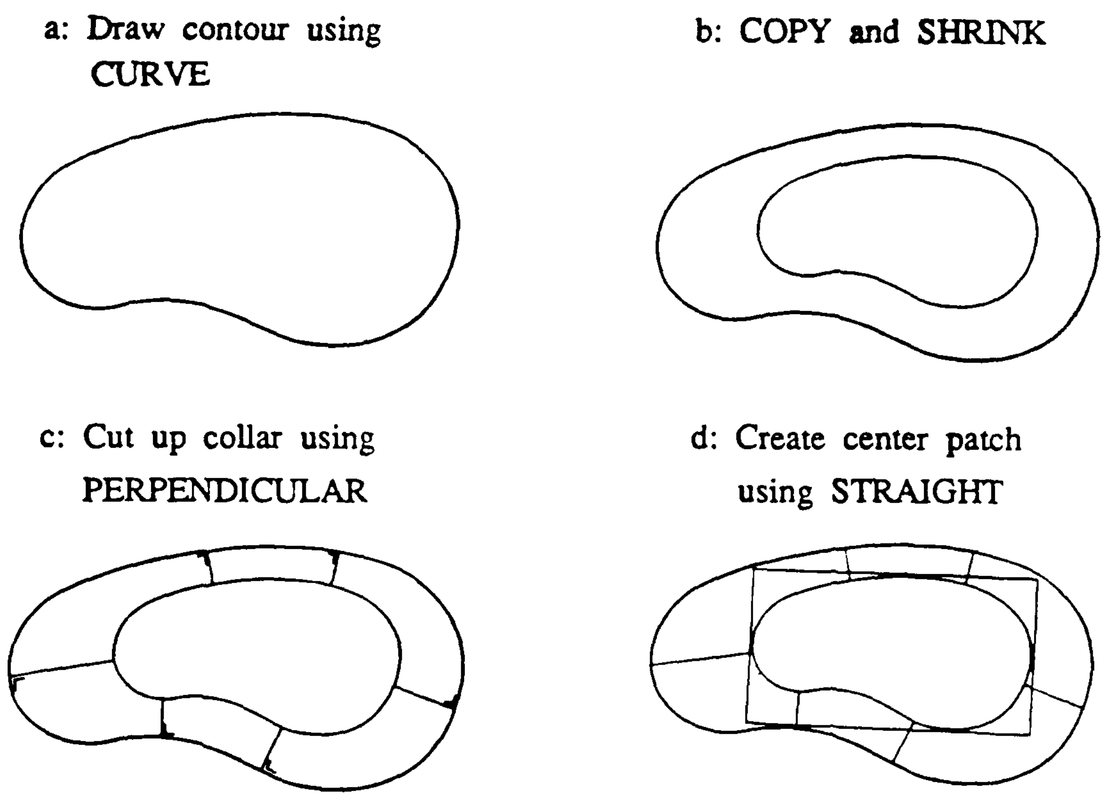

a: Draw contour using CURVE
b: COPY and SHRINK
c: Cut up collar using PERPENDICULAR
d: Create center patch using STRAIGHT

Figure 4.1: Generation of a sample composite grid

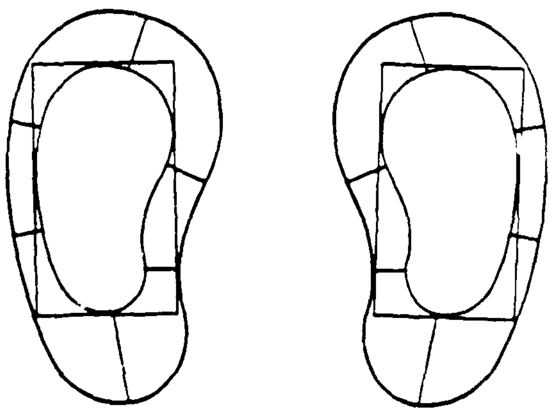


Figure 4.2: Duplicate whole structure using REFLECT

Chapter 5

AN ADAPTIVE-COMPOSITE-GRID SCENARIO

5.1 Introduction

The ultimate goal of this project is to effect the synthesis of composite-grid techniques and patched adaptive mesh refinement. That means using multiple simply-shaped grids to tackle the geometrical complexity of a problem, and adaptively patching refined grids on this composite grid to handle solution complexity.

Although implementation of the adaptive method is beyond the scope of this work, a study (largely two-dimensional) has been made of the feasibility of such a synthesis. The results are presented in this chapter.

In previous chapters the foundation was laid for the construction and implementation of composite grids for complex domains. It was judged that the generation of a composite grid is too formidable a task to be carried out automatically. Entirely manual grid generation. on the other hand, would put too much burden on the user. It was concluded that an interactive system which takes maximum advantage of the computing power of a machine and of the geometrical insight of the human should be developed.

The attractiveness of this set-up largely disappears when the adaptive-grid concept is added. A cornerstone of solution-adaptive mesh refinement is that new grids are created without user intervention. For steady-state problems one might allow some user monitoring during the computational process, but time-dependent problems necessitate a non-interactive environment (except for some simple options. such as: *stop computing*, or *let me look at the solution*). The aim of this investigation is to establish truly automatic solution-adaptive grid refinement on composite grids.

In the next two sections we will describe two ways of achieving this. The first one is flawed. Why discuss it? The answer is: Because it is an obvious way to go, and the reasons for its failure are somewhat subtle. The second one is more elegant and effective. Unfortunately, the second method requires computer science techniques that are somewhat involved and nonstandard in the field of numerical computations. However, the extra computational expense of the second method of *generating* refined grids is easily offset by the savings obtained when *computing solutions* on these refinements. As fewer refinements are generated, the cost of communicating between these grids goes down drastically.

5.2 Componentwise Grid Refinement

A straightforward way of constructing adaptive refinements on composite grids is based on the approach taken by Berger [BER82] and Caruso [CAR85]. The idea is to apply refinement to individual component grids without reference to the other grids. These refinements would be created *in computational space*, so that we only have to deal with rectangular regions. This problem has already been tackled by the abovementioned authors. It should be pointed out here that refinements are not generally aligned with the component grids. If we insist on alignment, the strategy may perform poorly for diagonally oriented flow phenomena. These require either a disproportionate number of aligned refinements, which slows down convergence due to the iterative character of the communication processes between them (see Chapter 6), or result in a very inefficient use of fine-grid area. In fact, the whole philosophy of rotated refinements for Cartesian grids, laid down by Berger, applies to curved component grids when viewed in the computational domain.

Precautions have to be taken to prevent refined grids from crossing the boundaries of the component grids. Thus, the cutting, shrinking, and folding techniques described by Caruso should be applied if refinements are to be rotated rectangles (*rotangles*) in computational space. The typical example that he gives is that of a shear layer, oriented diagonally across the domain. To cover the high-error shear-layer region with refinements takes at least three rotangles (see Fig. 5.1).

Great advantages of the componentwise scheme are modularity and geometrical simplicity. The only new element in the method is the furnishing of boundary conditions for refined grids touching component-grid boundaries. The following strategy may be followed: Masking arrays are created for every level of refinement, which flag points that are contained in deeper nestings of refinement. Whenever boundary conditions are needed from grid B for a point X in grid A, the B-coordinates of X are computed. Using the masking arrays, the highest level of refinement in B containing X is determined. Boundary conditions for X in A are obtained from this grid.

A drawback of this approach for composite grids is inefficiency in terms of number of refinements generated. Error zones, contiguous in physical space, but straddling component grids, would give rise to several —perhaps many— separate refinements within the individual components. This situation is sketched in Fig. 5.2. The thick solid line indicating a flow feature causing large errors cuts across all component grids. When the three component grids are mapped to their respective computational spaces, application of Caruso's technique in every such space would require at least three rotangles in each, for a total of nine. To cover one single error zone, nine refinements are created! This is not very efficient.

In addition, the patches near the grid boundaries that are grid-aligned —and hence not aligned with the flow phenomenon causing the large errors— may generate spurious waves, reflections, or numerical diffusion.

A more subtle, but equally harmful, drawback of this approach is the following. For safety and efficiency reasons one often introduces buffer zones around points of large error. The buffer zones serve to ensure that boundary conditions for the refined grids, obtained from the underlying coarse grid, are evaluated far away enough from the error zones to guarantee accuracy. Larger buffer zones lead to reduced update frequency of the refinements, but also to more work on the refinements. Obviously, an optimum buffer region of nonzero size exists. Should a region of large error border a component grid, but *not* project beyond that edge, then special precautions need to be taken. It is possible that the component grid on the other side of the interface does not contain any points with large error and therefore does not 'feel' the need to refine; consequently, no need for a buffer zone is perceived, although one is actually necessary. This dangerous situation, which is likely to occur in situations where unsteady problems are solved in which error zones travel through the field, stems from the fact that the component grids are treated independently. Apparently, a stronger intertwinement of grids is needed than has been outlined above.

5.3 Component-Independent Grid Refinement

A more appropriate approach to adaptive grid refinement on composite grids is characterized by globalness and independence. A scenario will be described in more detail after the following summary.

1. Construct a composite grid as described in Chapter 4 (Fig. 5.3).
2. Compute a solution on the composite grid, estimate the error, and mark the high-error points.
3. If the error is large *almost everywhere*, *refine* all grids and restart (Fig. 5.4).
4. Construct a Cartesian reference grid that covers the entire physical domain (Fig. 5.5).
5. Mark all *error points on this reference* grid (Fig. 5.5).
6. Cluster the error points on the reference grid into coherent groups.
7. Construct boundaries of refined grids covering the error clusters.
8. Compute the mesh sizes of the refined grids.
9. Construct a fence demarcating a *safety* zone around the physical boundaries of the original domain (Fig. 5.6).
10. Identify refined grids that penetrate the fence into the safety zone (Fig. 5.9).
11. Divide refined grids that violate the fence into interior points and safety-zone points. For the safety zone, define grid patches that are aligned (in computational space) with the composite-grid components.

12. Compute a solution on the whole set of refined grids at a certain level of refinement.

13. Repeat above recursively.

Now we give an in-depth description of the scenario.

Step 1 **Construct composite grid**

Create a composite grid covering the physical domain (see Fig. 5.3). This is done with the interactive system M*E*S*H, and by a postprocessing program that converts the continuous output of M*E*S*H into discrete data (this conversion program already exists for 2D problems). For nonadaptive composite-grid calculations, mesh sizes must be chosen such that the desired accuracy is reached without refinement; substantial coordinate stretching may be required.

Step 2 **Estimate error**

Compute the error (truncation or global error) on all component grids using the Richardson extrapolation technique. Flag all points that exceed the error threshold.

Step 3 **Decide on global refinement**

When adaptive-grid strategies are used, the initial composite grid (the *base* grid) will not generally provide the desired accuracy everywhere in the domain. As adaptive-grid refinement adds overhead, we must try to limit the number of grid cells contained in the refinements. If too much of the base grid needs refinement, it was too coarse in the first place and it is better to refine the whole grid (global mesh refinement) and start again. This statement needs to be quantified. The criterion is the following. Suppose a set of refined grids that covers all the bad points on the base grid is formed. The number of points in this set of local refinements is Nf_{local}. The number of points in the grid that would result from global refinement is Nf_{global}. If the ratio Nf_{local}/Nf_{global} is too large, then global refinement should be applied. Computing Nf_{local} is relatively expensive and should be avoided if possible. Thus, a two-stage approach is proposed. First, compute the ratio of the number of bad base grid points, N_{bad}, to the total number of base grid points, N. Obviously, $(Nf_{local}/Nf_{global}) \geq (N_{bad}/N)$, so if N_{bad}/N is large we can decide to apply global mesh refinement. If N_{bad}/N is below the threshold, the second stage is entered, consisting of the evaluation of Nf_{local}, which includes clustering, fitting grids, *etc.*

If global mesh refinement is chosen, the refinement may be accomplished straightforwardly (*e.g.*, by doubling the number of mesh cells in all directions on all component grids), or by using more sophisticated techniques that single out trouble-making component grids and refine only those (one might call this cellular adaptive mesh refinement).

Refinement should take place *within* the base-grid structure, that is, no new coordinate transformations and/or stretchings within component grids should be introduced (Fig. 5.4).

Global refinement can be repeated until less than the prescribed fraction of mesh cells is contained in local refinements.

It should be applied at every level of refinement, not just at the base-grid level.

Note: We do not monitor the fraction of the area that needs refinement (which would be cumbersome), but the fraction of the number of cells. This is because the work depends on the number of cells, not on the surface area (or volume).

Step 4 **Construct reference grid**

Construct a Cartesian reference grid with square (cubic in 3D) cells (see Fig. 5.5). It should be large enough to cover the entire physical domain. This grid will be used for clustering error points. The mesh size is based on the following considerations:

- As the base grid mesh size goes down (classical convergence), the total number of refined grids may not increase indefinitely, as the overhead would explode and the efficiency would likewise deteriorate. This can be prevented by setting a minimum size on the cells of the reference grid.

- The reference grid is used to capture the geometrical structure of the error, which is typically dictated by physical phenomena such as shocks, recirculation zones, *etc.* As the mesh size of the base grid decreases, this structure becomes less and less dependent on the mesh size, *i.e.*, its complexity does not increase beyond bound.

Consequently, a heuristic approach will be taken to define the reference-grid mesh size. A lower bound is dictated by efficiency considerations, whereas an upper bound can be based on the expected geometrical complexity of the error.

One can argue about the appropriateness of *square* mesh cells for the Cartesian reference grid. A flow field containing boundary layers may be fitted with component grids with high cell aspect ratios, but a uniform Cartesian grid overlaying this structure will typically have a cell aspect ratio of one. A clustering routine based on such a square-cell grid would perform poorly in some parts of the field. Part of this concern can be brushed aside, though, as refinements will not usually extend all the way to the boundary. To prevent refinements from protruding beyond the physical boundaries of the domain, a *safety zone* will be designated (see Step 9) within which only component-grid-aligned refinements are allowed. Thus, we need not worry about clustering bad points inefficiently near solid boundaries, as these areas get a special treatment. Strongly anisotropic error regions in the interior of the flow field (such as shocks, free shear layers) remain prone to inefficient clustering on the reference grid, but this is inevitable; self-adaptive solution strategies assume no *a priori* knowledge of the error, so no precautions can be taken.

Step 5 **Mark bad points in reference grid**

Walk through all the points in every component grid (in an index-lexicographical way) and mark all the cells in the reference grid that contain a 'bad' point. This is done as follows. Define the reference grid as a *logical* (Fortran) 2D or 3D reference array. No coordinates need to be stored; we only need a linear scale factor t which maps the integral indices i of the reference array to real coordinates x of the reference grid (*i.e.*, $x = i * t$). Convert coordinates on the component grid to those of the reference array. Truncate the (nonintegral) coordinates $(\hat{\imath}, \cdots)$ of the upper right (back) corner of the grid cell to integer values (*i.e.*, $i = \lfloor \hat{\imath} \rfloor$). The reference array entry with indices corresponding to these values is marked (see Fig. 5.5). No search process is needed.

Note: Buffer regions around bad points should be provided before clusters are formed on the reference grid. An approach which requires more work initially but might yield smaller refined grids is one in which every point in a component grid which is close enough to a bad point is flagged. Applying this method only on the reference grid is simpler, but cruder.

Step 6 **Cluster bad points**

Cluster bad points on the reference grid. The clustering algorithm, described by Berger [BER82], is one of the trickiest of the adaptive-grid strategy, and may require switching back and forth between this step and Step 7 (constructing refinements) until a satisfactory set of local refinements has been obtained, or global refinement has been applied. The trade-off is between having the smallest number of refined grids and having the smallest number of cells in refined grids that do not really need refinement. A deterministic, single-pass approach is used in which bad points are put in the same cluster when the summed absolute differences between their indices in the reference grid are below a certain value.

Step 7 **Construct refinements**

Construct grid patches covering clusters of bad points. We will focus on fitting the clusters with relatively simple shapes. The simplest shape is obviously the (rotated) rectangle. More complicated shapes are grids whose sides are made up of quadratic curves. These are useful when error zones exhibit significant curvature, like in bow shocks. Several ways are conceivable for defining these grids, one of which will be described here. Start by fitting one 'spine' curve to the whole cluster, using, for example, a least-squares fit. Subsequently, the sides along the fitted curve are found by applying restricted affine transformations (congruence transformations) to the spine such that all bad points are caught between two tranformed spinal curves. The 'lids' at the ends of the spine can be constructed by drawing straight lines perpendicular to it and by making sure all bad points are now completely contained in the warped quadrilateral patch. Obviously, a rectangular grid will be recovered in case of an amoeba-shaped set of bad points.

The refinements are constructed disregarding the physical boundaries, although they should be fully contained in the reference grid.

Step 8 **Determine refinement mesh size**

A simple way of arriving at the refined-mesh size is to find the smallest mesh size h_s of the cells contained in the refinement patch and to choose the cell size to be a specified fraction of h_s. This can become prohibitively expensive when the error exhibits strong anisotropy, which is often the case. In that case one might want to have refined grid cells with large aspect ratios. The most common sources of strong anisotropy are boundary layers, free shear layers, shocks, contact discontinuities, and flame fronts. The geometrical location of boundary layers is typically quite predictable, and any sensible composite grid will incorporate grid stretching on component grids near solid boundaries. Hence, it suffices to refine within the stretched-grid structure if we want to create stretched refinement cells as well. Step 9 describes how a so-called fence, placed outside the boundary layer, may be used to effect this type of refinement. The positions of shocks, *etc.*, are usually not known acccurately in advance, so in order to construct refinements with high cell aspect ratios for them we need to determine high error anisotropy automatically. More sophisticated error-estimation algorithms than are currently available [BER82], [CAR85] should be employed to detect this occurrence. Due to their very nature, error zones lead to refined grids that are aligned with them. We can use this alignment property to pack points tightly across the error zone, while spacing them more widely along it. Quantitative information on error anisotropy can be obtained by refining in all coordinate directions independently. Unfortunately, for a refinement ratio of r this is dr^{d-1} times as expensive as the isotropic error estimation in d-dimensional space (similar to semi-coarsening in multigrid methods).

Step 9 **Construct fence**

As was mentioned under Steps 4 and 8, a fence can be erected to signal the case in which a refined grid patch projects beyond the boundary of the physical domain, or invades a region in which refinement should be restricted to the underlying component grid for other reasons. The fence consists of a strongly connected set of squares (cubes) arranged in such a way that no curve connecting two points on the physical boundary cuts through the fence an odd number of times. The selected set of squares or cubes are taken from an array of cells that together form a Cartesian grid (the *fence grid*). See Fig. 5.6. This means that cells that are strongly connected have a whole side (face in 3D) in common. A fence cell then corresponds to 2 (or 3) indices in the Cartesian grid. We may choose the reference grid as the fence grid, but this is not necessary. The fence is stored as a set of scan lines (embedded in a set of scan planes in 3D). cutting through the physical domain, as is shown in Fig. 5.7. Every line is stored as a variable-length array of pairs of integer values representing locations of fence cells within the fence grid. For each pair we also store an indicator stating whether the area between the fence points:

- is properly contained in the interior of the physical domain, or
- is part of the *safety zone* and beyond, or
- consists of fence points itself (in this case the two integers of the pair are consecutive numbers; this way every fence point occurs exactly once as an entry in a pair, which makes the boundary-violation detection easier)

The construction of the fence can be established by overlaying the physical domain with the Cartesian fence grid with prescribed mesh size and using the mouse to 'click' fence cells, or by drawing a continuous line which is discretized by a program. In three dimensions this procedure can be carried out for a number of parallel planes, each representing a slice of the physical domain. As the fence needs to be strongly connected within each such plane, as well as in the direction perpendicular to the planes, information about the previous (parallel) plane must be available when processing the current plane. This may be done by plotting both planes on the terminal screen simultaneously, using different shadings that do not obscure each other. Corrective procedures may be applied by cutting slices out of the domain in the two other perpendicular directions as well.
Artificial Intelligence techniques are finding their way into geometrical problem solving in computational fluid dynamics these days ([VOG88]) and can undoubtedly assist in creating fences, although completely automatic construction is not envisioned in the near future. Checking connectivity of 3D-fences is a much easier task, and can be automated straightforwardly.

Step 10 **Detect fence violations**

Walk through the –ordered– list of grid refinements on a given level of refinement to determine whether any of these grids cuts through or into the fence. This procedure needs to be fully automatic, in contrast with the procedure for setting up the fence, which is partly interactive. We start by walking along the boundary of a refined grid using steps that are smaller than the mesh size of the fence grid. For this purpose we need to have an estimate of the arc length along the edge of the refined grid as a function of some parameter s. For rectangular grids this is easy to obtain, but for curved grids numerical quadrature is needed. Then we scale the coordinates of the grid boundary points to correspond to the (integral) reference grid and truncate the result to integer values. Subsequently, we match these integer values with the fence-cell indices using the scan lines. Whenever a match occurs, the boundary point is flagged (*i.e.*, stored in an array). After coming full circle around the grid, several possibilities exist:

a) one or more points have been flagged, which means that the grid violated the fence, or

b) no points have been flagged; this may mean that the grid is totally inside the fence or totally outside the fence, or that the grid completely contains a closed part of the fence (which corresponds to an 'island' or 'hole').

Case *b* can give rise to some ambiguity. Using the indicators on the fence array, it can easily be decided whether all points on the grid boundary are totally inside or totally outside the fence by just checking one point. The inclusion of a hole in the refined grid may still occur in either case, though, which might go undetected. Therefore, a list of 'danger points' (one for each hole) must be supplied at the time of the creation of the fence. Whenever a grid contains a danger point, it includes an entire hole, and appropriate action must be taken (see Fig. 5.8). To determine whether a grid contains a danger point a multi-stage technique may be applied.

1. Compute the distance between the center of the grid (which is defined as the point corresponding to the center of a rectangle in computational space) and every danger point; if that distance is greater than the radius of the grid (the maximum distance of any boundary point from the center of the grid), then the grid cannot include the danger point.

2. If the distance between the grid and the danger point is less than the radius of the grid, then the coordinates of the danger point are computed in terms of the refined grid (this requires inversion of a coordinate transformation), which unambiguously decides whether the danger point is contained or not.

Step 11 **Handle fence violation**

In case the fence has been violated, we need to refine the piece of the grid that invades the safety zone using the underlying component-grid metric structure, whereas the part of the grid which lies in the interior of the domain can be refined as is. In Step 10 we only identified the boundary points of the refined grid where fence violation has been detected, but now a more detailed analysis of the fence/grid intersection problem is needed. In fact, a complete map of the 2D or 3D intersection is needed. It is now easiest to walk through all the cells of the refined grid and mark matches between corners of refinement cells and fence cells. Finally, a set of grid cells will be found which effectively takes a 'bite' out of the grid (see Fig. 5.9). This bite needs refinement within the underlying component-grid structure(s), but perhaps more needs to be included in this refinement which cannot easily be refined otherwise. The idea is that the largest part possible of the refined grid remaining after taking out the bite should remain as it is. In general this remainder is a stair-step grid, which needs no modification if such grids are allowed (Fig. 5.10). If stair-step grids cannot be used, we would like to find the covering which uses the smallest number of rectangular grids (in computational space), but at the same time minimizes the number of cells not covered which need to be refined afterwards using the underlying component grid(s). Fig. 5.11 exemplifies this dilemma. Note that the unmarked cells —those cells not in the bite initially— are lumped with the bite in the end (indicated by the dot pattern). Care must be taken that these cells and the original bite form contiguous space, so that no new stray lumps are created.

After as much area as possible around the bite has been covered, the rest of the still uncovered points is lumped with the bite (note: all along we regard *all* points in the refinement as bad points, whether or not they were marked initially as error points. This means that their history is forgotten as soon as they are included in a refinement). These points will be refined within the underlying component grids. A slightly complicating factor is that one bite may straddle several component grids and, conversely, that one area in a component grid may carry bad points from several different bites. This means that for every point in every bite we must find out on which component grid it lies, and mark the appropriate cell in that component grid. After all the bite points are marked this way (which involves somewhat expensive coordinate-transformation inversions), we visit each component grid separately and fit the marked cells with as few aligned refined grids as possible without including too many unmarked cells. Reclustering may be necessary for very crooked clusters.

Step 12 **Compute solution on refinement**

Using SWAPR (see Chapter 6), a new approximate solution is computed on the set of refined grids. If a refinement lies on top of more than one coarse component grid, it does not matter which coarse grid donates the boundary conditions. The biggest problem in this step is obviously the large number of coordinate-transformation inversions that needs to be carried out in order to compute refinement/refinement and refinement/coarse-grid intersections.

Step 13 **Create new set of adaptive refinements**

After a new solution is computed, all the above steps from 2 to 12 are repeated on the refined level, with the possible omission of Step 4, as a reference grid already exists. Step 9 changes somewhat, as the erection of a new fence should now be established in a fully automatic way. This can be done as follows: discretize the boundaries of all the refinements on the reference grid, making sure that the resulting set of discrete curves is strongly connected. Next, use a contour-finding algorithm to strip away any

interior discrete curves. Finally, trim the contour —if necessary— using the original fence as a boundary in case the contour violates it.
Note: The construction of a new, restricted fence for the refinements is only needed if we insist on proper nesting of all levels of refinement. If we relax this requirement, which is no real concession for the steady, elliptic problems we are interested in, we can just monitor violation of the original coarse-level fence.

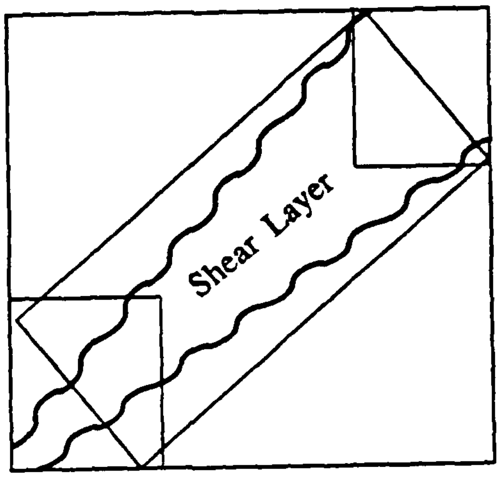


Figure 5.1: Three rotangles used to cover shear layer

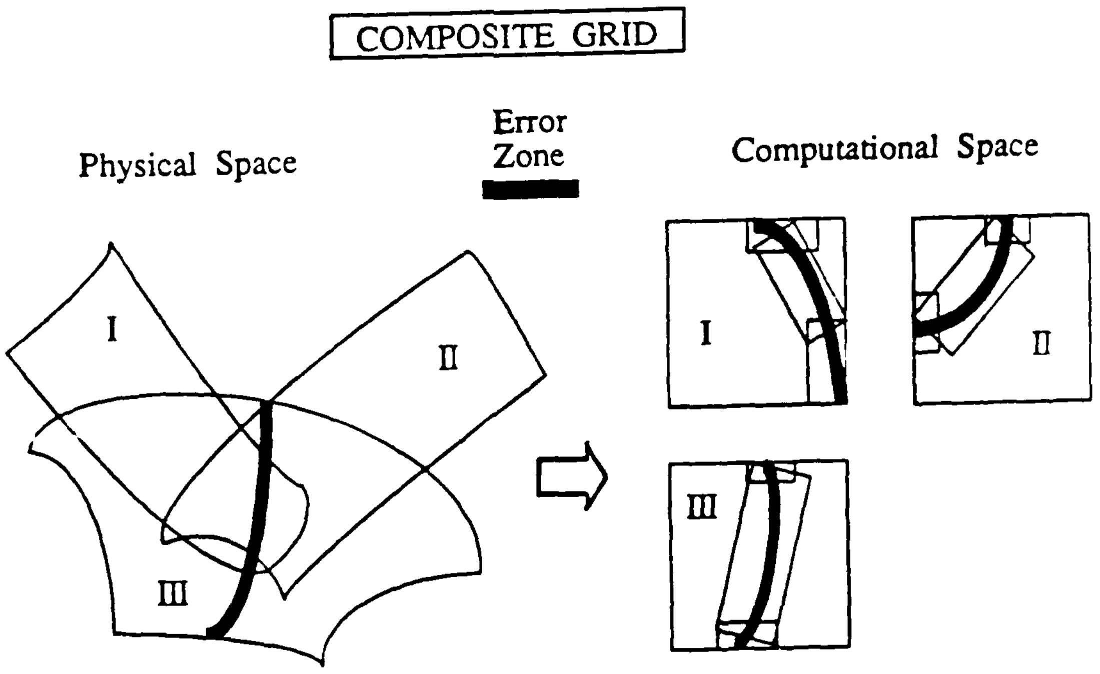


Figure 5.2: Nine refinements in computational space needed for single error zone

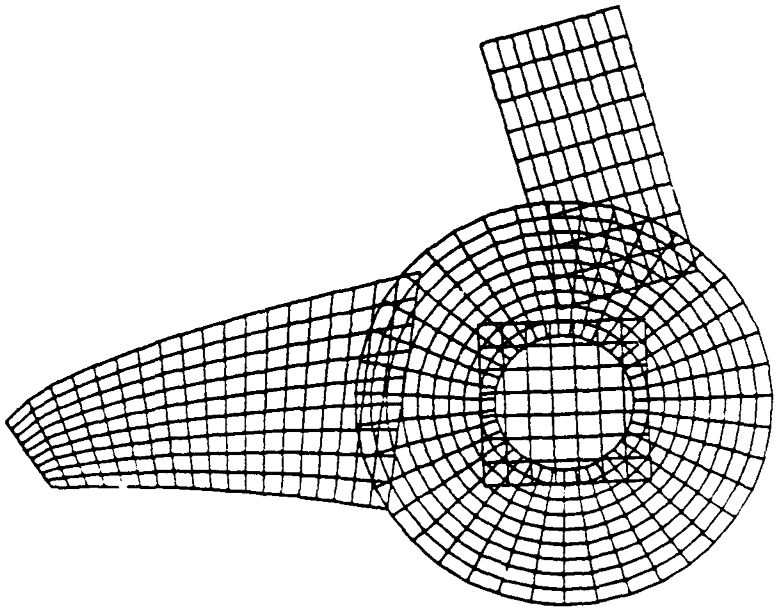


Figure 5.3: Sample composite grid

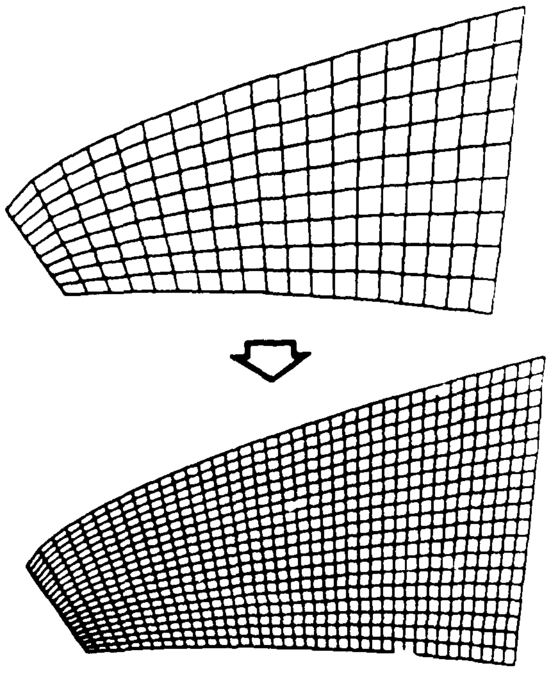


Figure 5.4: Global refinement within component-grid structure

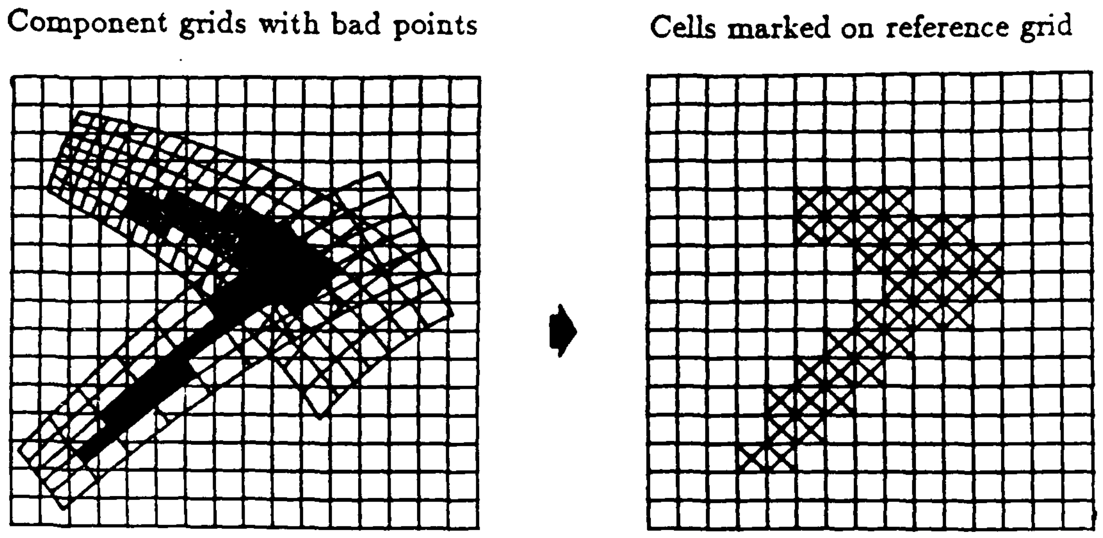

Component grids with bad points
Cells marked on reference grid

Figure 5.5: Cartesian reference frame to mark error points

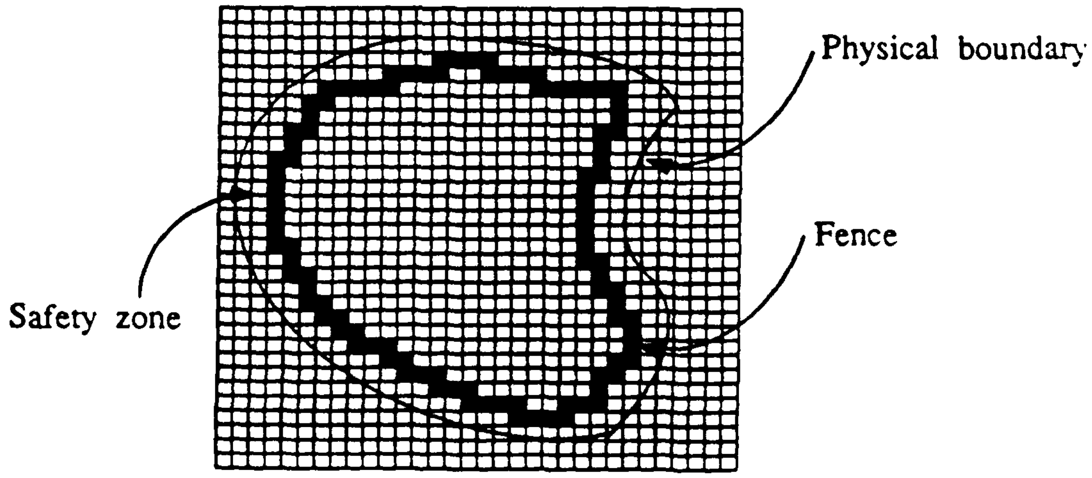


Figure 5.6: Sample fence on a fence grid

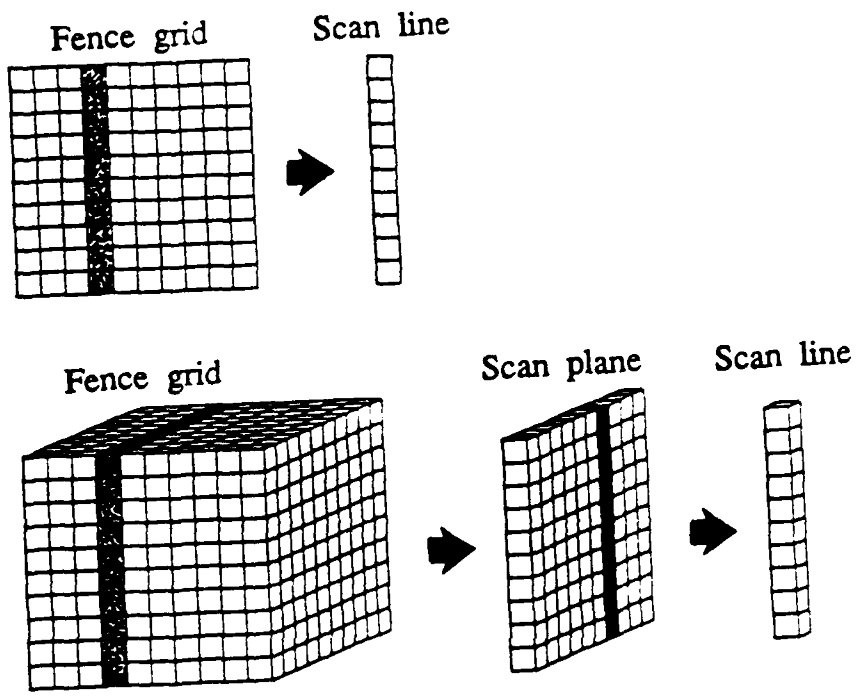


Figure 5.7: Scan lines and planes

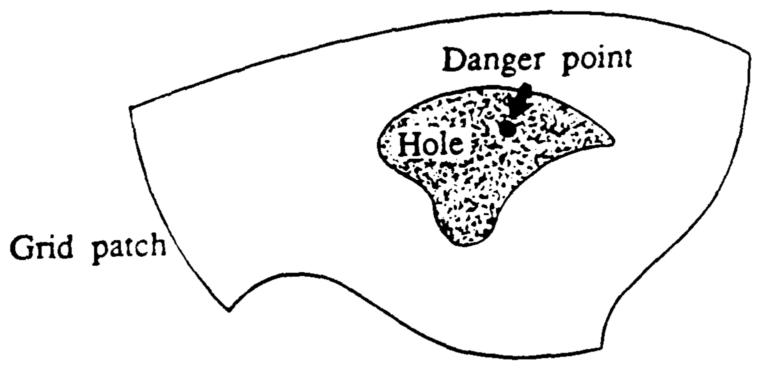


Figure 5.8: Inclusion of a danger point in a grid

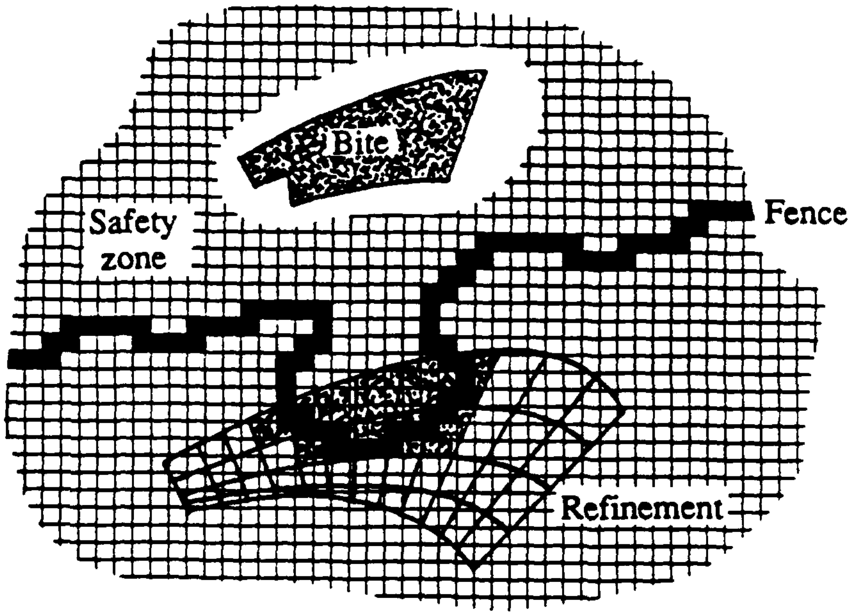


Figure 5.9: A fence-violating bite from a refinement

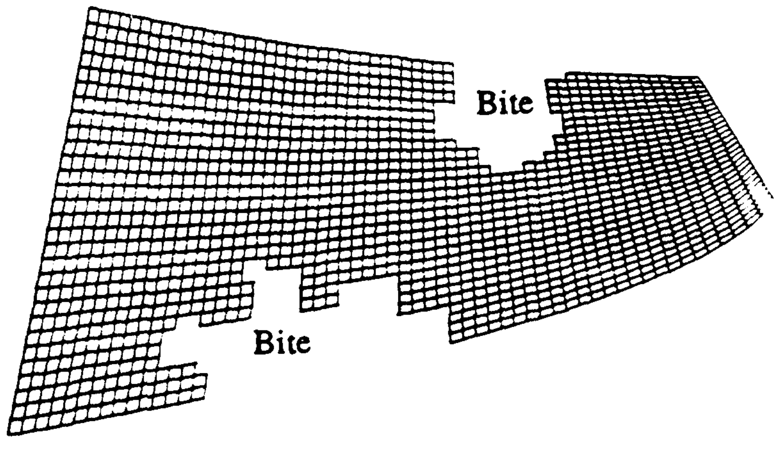


Figure 5.10: Stair-step grid for interior-refinement region

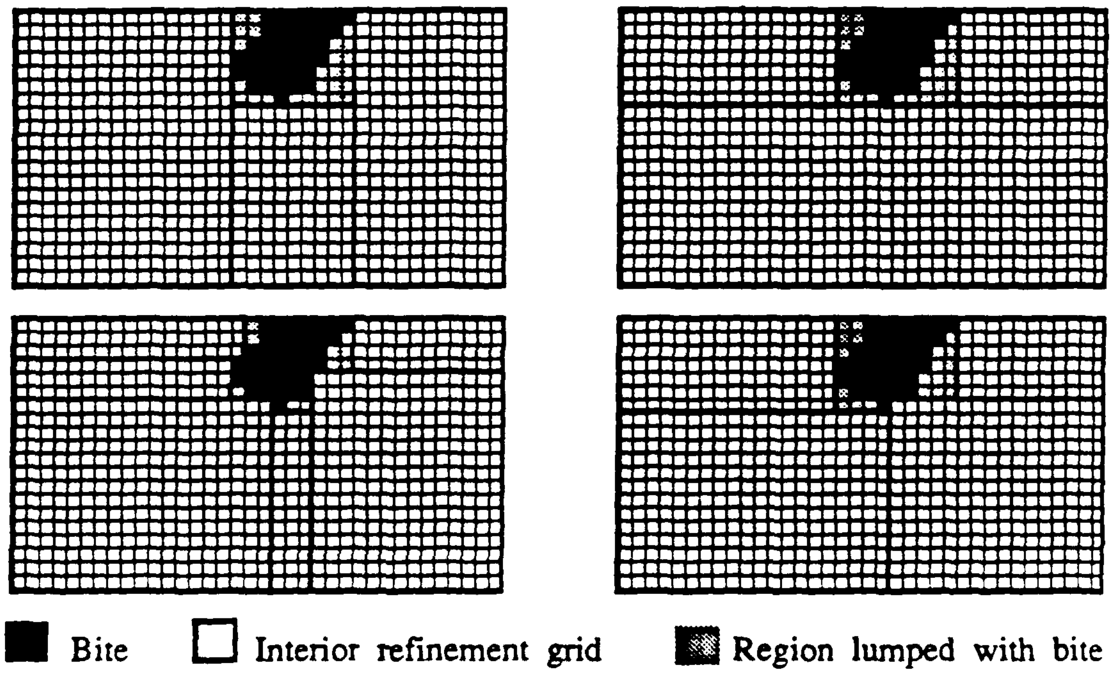


Figure 5.11: Possible coverings of interior refinement region (computational space)

Chapter 6

COMPUTING SOLUTIONS ON COMPOSITE GRIDS

6.1 Introduction

In this chapter we examine strategies for computing solutions on composite grids. The most appropriate method —the Schwarz Alternating Procedure (SWAP)— is selected for further study. A modified version of SWAP (called SWAPR) is formulated, which does not require component grids to overlap. Solution of one-dimensional model problems shows SWAP and SWAPR to be equally efficient and accurate.

6.2 Direct Solution

Several ways exist to compute solutions on composite grids. The most obvious way is to consider the discrete equations on the subdomains and all interpolation formulas that link the grids as one big system of equations, linearize the equations, and solve the system using a sparse-matrix solver. For two (overlapping) domains with unknowns named as indicated in Fig. 6.1 (u_1 corresponds to region Ω_1, u_2 corresponds to region Ω_2, and u_3 refers to the variables in the interface region Ω_3, which sometimes degenerates to a line), the whole linearized algebraic system can be written as:

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} . \qquad (6.1)$$

The first two (block-)rows represent the discretizations in the interiors of grids Ω_1 and Ω_2. The blocks A_{13} and A_{23} represent the coupling with the unknowns in the interface region Ω_3. The third row in the system indicates that the points in the interface region talk directly to points in both other regions.

Solving (6.1) directly is the approach taken by Henshaw [HEN85]. Although this is an attractive route, as it requires no special iteration schemes between grids, there are some severe drawbacks:

- Although the matrix is sparse, it does not have a regular structure that one can easily take advantage of. If it did, the problem could be better formulated as a single-grid problem.
- An inordinate amount of storage is required as the spatial dimension goes up and the grid resolution increases.
- Specialized solvers for rectangular domains (in computational space) cannot be used.
- When the original system is nonlinear, equation (6.1) has to be solved many times as part of an iterative procedure, which gobbles up much of the computational efficiency of the direct method.

For these reasons we will not explore this possibility any further.

6.3 Schur-Complement Method

Another option is first to apply block-Gaussian elimination to system (6.1). Eliminating all the unknowns in the interiors of regions Ω_1 and Ω_2 results in the following equation for the unknowns in the interface region:

$$(A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23})\, u_3 = \tilde{b}_3 \ . \qquad (6.2)$$

The complicated matrix in the left-hand side is called the Schur-complement of matrix block A_{33} in the whole matrix.

After solving for the interface variables, the other unknowns can be computed independently on their respective subdomains using dedicated solvers; only local information is needed. For a sequential computer this means that larger problems can be solved, as only one block of memory the size of the biggest subdomain needs to be available at any one time. For a parallel machine the independence of the subdomain problems means less slowdown through communication between different processors. The problem is evidently solving the interface equations first. The Schur-complement is dense and cannot generally be generated or factored efficiently. Moreover, formulating the interface equations requires information from the whole composite domain, which we sought to avoid.

Iterative procedures as described by Chan [CHA88B] get around the problem of formulating the dense Schur-complement matrix, but again no dedicated single-grid solver can be used to solve the interface equations, and information is still needed from the whole domain. When one wants to compute solutions to the Navier-Stokes equations in general curvilinear coordinates on a two-dimensional domain, many auxiliary arrays have to be stored besides the three solution vectors, u, v, and p. For example, in the currently implemented efficient solution procedure, at least 21 extra arrays (see Appendix C) are needed per grid in addition to the 15 metric arrays. If these have to be kept for every subdomain instead of being generated on the fly, severe memory problems will occur; we do not wish to store the whole matrix (6.1)!

6.4 Schwarz Alternating Procedure

To deal with the above problems, we use the traditional, decoupled approach first applied by Schwarz [SCH69] to prove existence of solutions to the Poisson equation on complex domains. Many authors have analyzed the method formally with the idea of utilizing it numerically. We refer the interested reader to [MIL65], [STO72], [ROD83], [TAN87]. If we regard the dissection of the domain in Fig. 6.1 as a union of two overlapping domains, Ω_l and Ω_r, where the variables in the overlap region are counted twice, the method for two one-dimensional subdomains (see Fig. 6.2) can be described as follows:

1. Guess boundary values on the edge of region Ω_l; this decouples Ω_l from Ω_r.

2. Solve the problem in the interior of region Ω_l.

3. Use that interior solution to determine boundary values on the interface boundary with region Ω_r.

4. Solve in the interior of region Ω_r.

5. Use that interior solution to determine boundary values on the interface boundary with region Ω_l.

6. Return to step 2.

It can be shown [MIL65] that this technique, called the SchWarz Alternating Method (SWAP), converges under quite generous conditions. Formally, an equivalence can also be established between SWAP and the Schur-Complement Method [CHA88A]. But, of course, the advantage of SWAP is that no special solution procedures are needed to solve interface equations, as boundary values are simply injected (interpolated) from one grid onto the other. This is the only way grids communicate. After boundary conditions for individual grids are obtained, the interior solutions can be generated independently, using dedicated fast solvers. Also, only auxiliary arrays pertaining to the grid on which a solution is being computed need to be stored. In a purely sequential iteration procedure (one grid at a time), the amount of storage needed goes down as the reciprocal of the number of subdomains if the subdomains are of comparable (discrete) size.

We will examine the properties of SWAP for some model problems to get insight in the way the method may be applied (and modified) to solve fluid-flow problems on a composite grid.

6.4.1 One-Dimensional Convection/Diffusion; Continuous Case

The model problem is:

$$f'' + c\,f' = g \;, \tag{6.3}$$

$$\text{with } f(0) = p \;, \quad f(1) = q \;. \tag{6.4}$$

The constant c is the convection speed, f is the transported variable (*positive* c corresponds to flow in the *negative* x-direction), and the prime denotes differentiation.

The general solution to equation (6.3) ***without boundary conditions*** is:

$$f(x) = h(x) + \sigma\, e^{-cx} + \kappa\,, \tag{6.5}$$

where $h(x)$ is the particular solution to problem (6.3) with boundary conditions (6.4),and σ and κ are arbitrary constants. Now we cut the domain $\Omega = [0,1]$ into two overlapping subdomains, $\Omega_l = [0,\beta]$ and $\Omega_r = [\gamma, 1]$ (see Fig. 6.2). Solutions f_l on Ω_l satisfying the left boundary condition are:

$$f_l(x) = h(x) + a(e^{-cx} - 1)\,, \tag{6.6}$$

whereas solutions f_r on Ω_r satisfying the right boundary condition are:

$$f_r(x) = h(x) + b(e^{-c(x-1)} - 1)\,. \tag{6.7}$$

The iterative Schwarz Alternating Procedure is started by assigning initial guesses to f_l and f_r at the interior boundary points of Ω_l and Ω_r:

$$f_l^0(\beta) = h(\beta) + \alpha^0\,, \tag{6.8}$$

$$f_r^0(\gamma) = h(\gamma) + \delta^0\,. \tag{6.9}$$

Here the superscripts indicate the iteration number. The procedure converges to the true solution h if

$$\lim_{n\to\infty} \alpha^n = \lim_{n\to\infty} \delta^n = 0\,. \tag{6.10}$$

With the assignments (6.8), (6.9), the following solution $f_l^0(x)$ is found:

$$f_l^0(x) = h(x) + \alpha^0 \frac{e^{-cx} - 1}{e^{-c\beta} - 1}\,, \tag{6.11}$$

which, evaluated at $x = \gamma$, is:

$$f_l^0(\gamma) = h(\gamma) + \alpha^0 \frac{e^{-c\gamma} - 1}{e^{-c\beta} - 1}\,. \tag{6.12}$$

A new value $f_r^1(\gamma)$ is computed using a relaxation factor θ_r:

$$\begin{aligned} f_r^1(\gamma) &= \theta_r f_l^0(\gamma) + (1 - \theta_r) f_r^0(\gamma) = \\ & h(\gamma) + \theta_r \alpha^0 \frac{e^{-c\gamma} - 1}{e^{-c\beta} - 1} + (1 - \theta_r)\delta^0\,. \end{aligned} \tag{6.13}$$

The new error δ^1 in the value of f_r at γ is:

$$\delta^1 = \theta_r \alpha^0 \frac{e^{-c\gamma} - 1}{e^{-c\beta} - 1} + (1 - \theta_r)\delta^0 . \tag{6.14}$$

So far, we have guessed boundary values at β and γ, computed the solution on Ω_l using the boundary value at β, and updated the boundary value at γ on Ω_r using the old boundary value *and* the new boundary value obtained from Ω_l. Now we can go the other way and compute the solution on Ω_r and update the boundary value at point β on Ω_l. Skipping some of the algebra, we summarize:

$$f_l^1(\beta) = \theta_l f_r^1(\beta) + (1 - \theta_l) f_l^0(\beta) = h(\beta) + \alpha^1 . \tag{6.15}$$

Here, θ_l is the relaxation factor pertaining to the boundary value in β on Ω_l, and α^1 is the new error in f_l at point β, which has the value:

$$\alpha^1 = \theta_l \delta^1 \frac{e^{-c(\beta-1)} - 1}{e^{-c(\gamma-1)} - 1} + (1 - \theta_l)\alpha^0 . \tag{6.16}$$

One complete SWAP iteration has been performed. Using these results, the successive error vectors $(\alpha^0, \delta^0)^T$ and $(\alpha^1, \delta^1)^T$ can be related through:

$$A \begin{pmatrix} \alpha^1 \\ \delta^1 \end{pmatrix} = B \begin{pmatrix} \alpha^0 \\ \delta^0 \end{pmatrix} , \tag{6.17}$$

with

$$A = \begin{pmatrix} 1 & -\theta_l \frac{e^{-c(\beta-1)}-1}{e^{-c(\gamma-1)}-1} \\ 0 & 1 \end{pmatrix} , \quad B = \begin{pmatrix} 1 - \theta_l & 0 \\ \theta_r \frac{e^{-c\gamma}-1}{e^{-c\beta}-1} & 1 - \theta_r \end{pmatrix} . \tag{6.18}$$

The iterative procedure converges if the spectral radius of the iteration matrix Z_{su} $(Z_{su} = A^{-1}B)$ is less than 1. It is easily found that:

$$\rho(Z_{su}) = \max[\tfrac{1}{2}|T \pm \sqrt{T^2 - 4\Delta}|] , \tag{6.19}$$

with

$$T = 2 - (\theta_l + \theta_r) + \theta_l\theta_r \frac{(e^{-c(\beta-1)} - 1)(e^{-c\gamma} - 1)}{(e^{-c\beta} - 1)(e^{-c(\gamma-1)} - 1)} , \quad \Delta = (1 - \theta_l)(1 - \theta_r) . \tag{6.20}$$

From these calculations we conclude that the convergence factor depends strongly on β and γ , and, in fact, $\lim_{\beta\to\gamma} \rho(Z_{su}) = 1$. This means that the SWAP iteration will never converge if there is no overlap, which is to be expected, because the initial guess at the common interior boundary point would never change.

Optimal convergence is obtained if both eigenvalues of the iteration matrix are zero, which means: $T = 0$ and $\Delta = 0$. Satisfying the latter condition implies that either $\theta_r = 1$, or $\theta_l = 1$, but **not** $\theta_l = \theta_r$. In other words, only one boundary-condition update should be relaxed, whereas the other should be left alone. Because the convergence factor depends symmetrically on both relaxation factors, we will set one equal to 1, and call the other simply θ to determine the optimal relaxation. The expression for the spectral radius reduces to:

$$\rho(Z_{sw}) = 1 - \theta\left\{1 - \frac{(e^{-c(\beta-1)} - 1)(e^{-c\gamma} - 1)}{(e^{-c\beta} - 1)(e^{-c(\gamma-1)} - 1)}\right\}, \tag{6.21}$$

which can indeed be made zero by choosing:

$$\theta_{opt} = \left\{1 - \frac{(e^{-c(\beta-1)} - 1)(e^{-c\gamma} - 1)}{(e^{-c\beta} - 1)(e^{-c(\gamma-1)} - 1)}\right\}^{-1}. \tag{6.22}$$

The following results are derived from the above analysis of SWAP:

- The convergence factor does not depend on the sign of the convection speed c.

- If no relaxation is applied at all ($\theta = 1$), the method always converges if $\beta > \gamma$.

- The optimal relaxation factor is always greater than 1 (overrelaxation).

An annoying fact about SWAP is that the convergence factor is a strong function of the amount of overlap of the subdomains. We would like to do away with overlap altogether for several reasons. First, points in the overlap region are visited twice per iteration, causing waste of computational effort.

Second, when grids overlap it is not easy to let points on the intersecting grids coincide, especially in higher-dimensional cases. This necessitates interpolation, which has the disadvantage of being nonconservative, *i.e.*, it does not preserve conservation properties integrally. More will be said about this in Chapter 8.

For the above reasons we investigate a modified Schwarz Alternating Procedure, which will be called SWAPR (Schwarz Alternating Procedure-REVISED).

The same model problem (6.3), (6.4) is solved on the same two subdomains, Ω_l and Ω_r. Interior boundary values for Ω_l at point β are again taken from the solution on Ω_r. What sets SWAPR apart from SWAP is that the the boundary condition at point γ for subdomain Ω_r is a *Neumann* condition, *i.e.*, the derivative of the solution is computed on Ω_l and used as a boundary condition for Ω_r (see Fig. 6.3). This procedure is asymmetric, as a Dirichlet problem is solved on one grid, and a Neumann problem is solved on the other. It is this asymmetry that allows us to let the overlap shrink to zero without deterioration of the method.

An analysis similar to the one employed for SWAP can be done for SWAPR, and the following expression for the convergence factor of SWAPR, $\rho(Z_{swr})$, is found:

$$\rho(Z_{swr}) = \max[\tfrac{1}{2}|T \pm \sqrt{T^2 - 4\Delta}|] , \qquad (6.23)$$

with

$$T = 2 - (\theta_l + \theta_r) + \theta_l\theta_r \frac{e^{-c(\beta-1)} - 1}{e^{-c\beta} - 1} , \quad \Delta = (1 - \theta_l)(1 - \theta_r) . \qquad (6.24)$$

The most remarkable feature of SWAPR is that the convergence factor does not depend on γ at all, which means that the size of the overlap is immaterial, and may be zero.

Again, $\rho(Z_{swr})$ depends symmetrically on the relaxation factors θ_l and θ_r, and one of them has to be equal to 1 for optimal convergence. Setting one equal to 1 again and calling the other θ, we get:

$$\rho(Z_{swr}) = 1 - \theta \frac{e^{-c(\beta-1)} - 1}{e^{-c\beta} - 1} , \qquad (6.25)$$

which means that the optimal relaxation factor is:

$$\theta_{opt} = \frac{1 - e^{-c\beta}}{1 - e^{-c}} . \qquad (6.26)$$

Besides the lack of the need for overlap, SWAPR has some more remarkable properties.

When no relaxation is applied (*i.e.*, $\theta = 1$), we find:

$$\rho(Z_{swr}) = \frac{e^{-c\beta} - e^{-c}}{e^{-c\beta} - 1} . \qquad (6.27)$$

From this expression we conclude that SWAPR without relaxation converges if

$$\beta > \frac{1}{c} \ln\left(\frac{2}{1+e^{-c}}\right) . \tag{6.28}$$

In Fig. 6.4, the graph of the critical values of β (those for which $\rho(Z_{swr}) = 1$) is shown. Apparently, β needs to be extended nearly to the right boundary of the whole domain in order to get convergence if c is negative. That means that almost all of the domain must be covered by the left subdomain if convection is in the positive x-direction. Recall that a Dirichlet problem is solved on Ω_l, which has β as an interior boundary point, and that a Neumann problem is being solved on Ω_r, which has γ as an interior boundary point. Apparently, if β is an outflow boundary point, solving a Dirichlet problem on its subdomain hurts. This fits in with engineering experience, which suggests that Neumann boundary conditions be prescribed on outflow boundaries. Conversely, if c has a significant positive (flow from right to left) magnitude, β may be virtually anywhere in the domain, as the critical point moves all the way to the left of the domain. This again conforms with engineering experience, because now β is an inflow boundary point for which a Dirichlet condition is suitable, whereas γ is an outflow boundary point, which comes with the appropriate Neumann condition.

From eq. (6.26) we learn that $0 < \theta_{opt} < 1$, which means that speed-up should be obtained through underrelaxation, regardless of the direction of the convection.

Finally, we find:

$$0 < \theta_{opt} \leq \beta \quad \text{if} \quad c \leq 0 \tag{6.29}$$

$$\beta < \theta_{opt} < 1 \quad \text{if} \quad c > 0 . \tag{6.30}$$

This implies that heavier underrelaxation should be applied if SWAPR is applied in the 'wrong' direction (Neumann condition on inflow boundary, Dirichlet on outflow boundary).

From the analysis of SWAPR we conclude that the seemingly arbitrary asymmetry needed for convergence is quite natural for flow problems. A natural bias is created by the direction of the flow.

6.4.2 Numerical Experiments

In this section we describe the results of numerical experiments with SWAP and SWAPR for the model equation (6.3). The two-domain case, which was analyzed in the previous section, is investigated most extensively.

All the convergence results derived for the continuous problem have been verified to hold for the discrete case as well, which implies, among others, that the *physical* overlap matters, not the number of points contained in it.

When the optimal relaxation factor is chosen, convergence takes place in 4 iterations for SWAP and SWAPR in all cases tested with the asymmetric relaxation factors.

No difference in convergence rate was observed between first- and second-order-accurate implementation of the Neumann boundary conditions for SWAPR for the (smooth) source functions used.

When more than two subgrids are used, the situation becomes much more complex, and has not been analyzed analytically. However, computations have been done, and it was found for the nonconvective case ($c = 0$) that roughly the same optimal relaxation factors hold for the multiple-subdomain case as for the two-subdomain case. In Table 6.1 the results of computations with several equisized nonoverlapping grids for the optimal relaxation factor ($\theta_{opt} = 0.5$) are shown. The computations were stopped if the relative changes in the boundary values dropped below 10^{-5}. Two different global iterations were tested: the one-way sweep solves sequentially from the first grid to the last, and then returns to the first grid, whereas the full sweep solves sequentially from first to last, then from last to first. If the number of subgrids is n, then per global iteration the one-way-sweep procedure solves on n subgrids, and the full-sweep procedure solves on $2(n-1)$ subgrids. The work spent by the two different procedures is measured in units corresponding to one complete solve on the global domain. Clearly, the work in the full-sweep case increases much more slowly than in the one-way-sweep case. Experiments with red-black orderings (first solve on all odd-numbered grids, then on even-numbered grids) show that some improvement can be obtained over the one-way-sweep method, although the savings are less than with the full-sweep strategy.

Table 6.1: *SWAPR on multiple nonoverlapping grids*

	one-way sweep		full sweep	
no. subgrids	no. iterations	work	no. iterations	work
2	4	4	4	4
3	15	15	20	27
4	27	27	17	26
5	40	40	16	26
6	54	54	15	25
7	70	70	20	34
8	88	88	18	32
9	98	98	20	36
10	120	120	25	45

Computations of the above kind have also been done for the convective case, but the results are not nearly as clear. The optimal relaxation factors computed for the two-grid case are not close to those for the multiple-grid case, and, in fact, the best results were obtained when relaxation factors were the same for all interior boundaries in both directions.

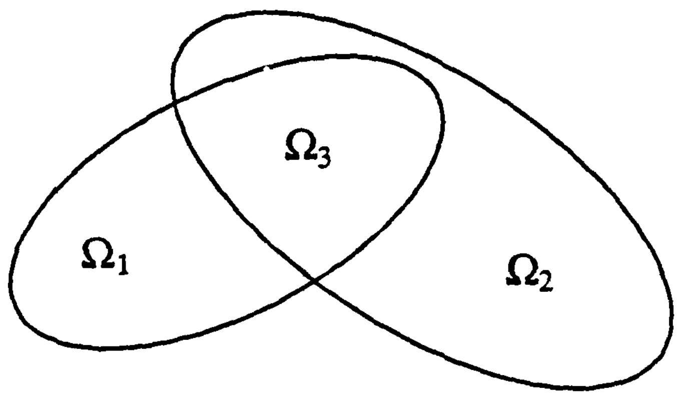


Figure 6.1: Two overlapping subdomains Ω_1 and Ω_2, with intersection Ω_3

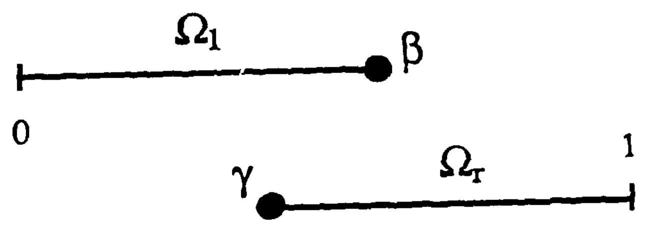


Figure 6.2: Two overlapping subdomains Ω_l and Ω_r, with interior boundary points β and γ

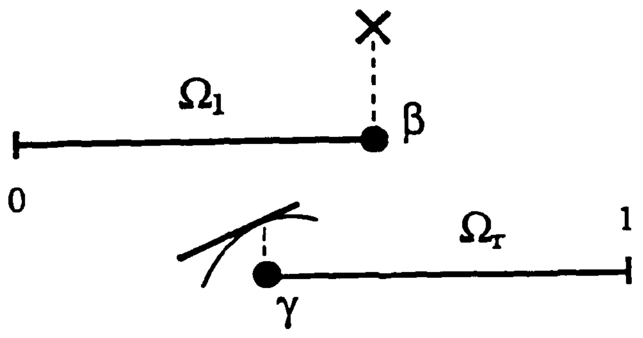


Figure 6.3: SWAPR: Neumann conditions at point γ, Dirichlet at β

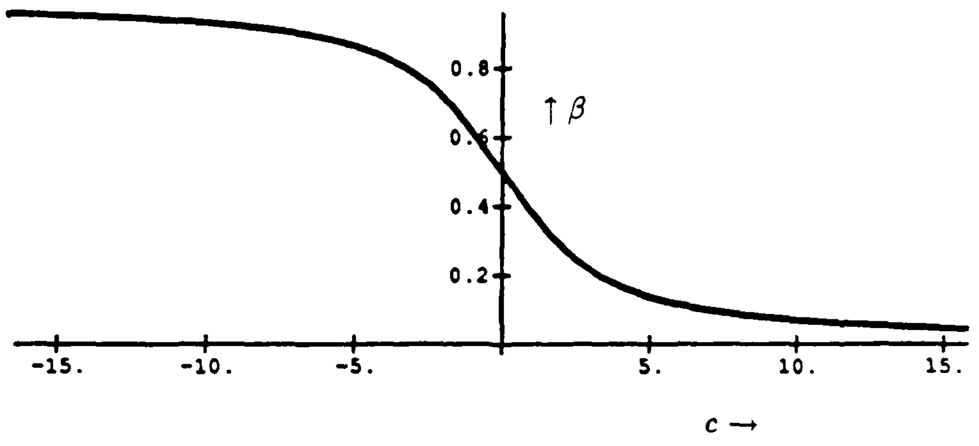


Figure 6.4: Critical values for boundary point β versus convection speed c

Chapter 7

NAVIER-STOKES SOLUTION PROCEDURE

7.1 Introduction

In this chapter the procedure for computing solutions to fluid-flow problems on single grids is described. The two main components of this procedure are the discretization of the differential equations on a nonstandard staggered grid, and the iterative solution of the resulting system of algebraic equations.

Additional issues are convergence criteria, implementation of Neumann boundary conditions, and stabilization of high-order-accurate difference schemes.

7.2 Navier-Stokes Equations in Curvilinear, Nonorthogonal Coordinates

In this section the form of the Navier-Stokes equations governing steady, incompressible, two-dimensional, viscous flow in a generalized, nonorthogonal coordinate system is derived, and some preliminary discretization is done. Although it is possible to reduce the number of equations describing the flow by one through the introduction of a streamfunction and the vorticity, we will not take that approach but solve the primitive equations (momentum + continuity). This is to enable straightforward extension to three space dimensions.

Using pseudo-tensor notation, the nondimensionalized momentum equations plus the continuity equation in Cartesian coordinates in the so-called strong conservation-law form can be written:

$$\frac{\partial u_i}{\partial x_i} = 0 \,, \tag{7.1}$$

and

$$\frac{\partial}{\partial x_i}(u_i u_j - \frac{1}{Re}\frac{\partial u_j}{\partial x_i}) = -\frac{\partial p}{\partial x_j} \,, \quad (j = 1, 2) \,. \tag{7.2}$$

Re is the Reynolds number, u_i is the Cartesian velocity vector, and p is the pressure. Introducing a general, nonsingular coordinate mapping $(\xi, \eta) \mapsto (x, y)$ and applying the chain rule of differentiation, $\frac{\partial}{\partial x_i} = \frac{\partial \xi_m}{\partial x_i} \frac{\partial}{\partial \xi_m}$, eqs. (7.1) and (7.2) transform into:

$$\frac{\partial \xi_m}{\partial x_i} \frac{\partial}{\partial \xi_m}(u_i) = 0 \, , \tag{7.3}$$

and

$$\frac{\partial \xi_m}{\partial x_i} \frac{\partial}{\partial \xi_m} \left(u_i u_j - \frac{1}{Re} \frac{\partial \xi_k}{\partial x_i} \frac{\partial}{\partial \xi_k} u_j \right) = - \frac{\partial \xi_m}{\partial x_j} \frac{\partial}{\partial \xi_m} p \, . \tag{7.4}$$

These equations are no longer in strong conservation-law form. This form can be restored, however, by multiplying through by the Jacobian determinant J, defined by

$$\mathrm{J} = det \left(\frac{\partial x_i}{\partial \xi_j} \right) \, , \tag{7.5}$$

and by using the fundamental metric identity

$$\frac{\partial}{\partial \xi_m} \left(\mathrm{J} \frac{\partial \xi_m}{\partial x_i} \right) \equiv 0. \tag{7.6}$$

Thus, we obtain

$$\frac{\partial}{\partial \xi_m} \left(\mathrm{J} \frac{\partial \xi_m}{\partial x_i} u_i \right) = 0 \, , \tag{7.7}$$

and

$$\frac{\partial}{\partial \xi_m} \left(\mathrm{J} \frac{\partial \xi_m}{\partial x_i} u_i u_j - \frac{1}{Re} \mathrm{J} \frac{\partial \xi_m}{\partial x_i} \frac{\partial \xi_k}{\partial x_i} \frac{\partial u_j}{\partial \xi_k} \right) = - \frac{\partial}{\partial \xi_m} \left(\mathrm{J} \frac{\partial \xi_m}{\partial x_j} p \right) . \tag{7.8}$$

These are the basic equations to be solved. The unknowns are the *Cartesian* components of velocity and the pressure.

Note: it is possible to formulate the Navier-Stokes equations using as unknowns the contravariant components of velocity U^m (the components of velocity in the direction of the curvilinear coordinates), defined by

$$\mathrm{U}^m = \frac{\partial \xi_m}{\partial x_j} u_j , \tag{7.9}$$

but this leads to comparatively complex equations, the solution of which is also susceptible to distortion due to sudden changes in metrics, according to Meakin [MEA86].

Hence, we only introduce the contravariant components of velocity into the continuity and Navier-Stokes equations as a notational convenience. They make it easier to linearize and to derive discretized equations, which are ultimately expressed in terms of Cartesian components of velocity. If we also introduce the metric tensor g, defined by

$$g^{mk} = \frac{\partial \xi_m}{\partial x_i}\frac{\partial \xi_k}{\partial x_i}, \tag{7.10}$$

we can write:

$$\frac{\partial}{\partial \xi_m}(\mathrm{J}\mathrm{U}^m) = 0 , \tag{7.11}$$

and

$$\frac{\partial}{\partial \xi_m}\left(\mathrm{J}\mathrm{U}^m u_j - \frac{1}{Re}\mathrm{J}g^{mk}\frac{\partial u_j}{\partial \xi_k}\right) = -\frac{\partial}{\partial \xi_m}\left(\mathrm{J}\frac{\partial \xi_m}{\partial x_j}p\right). \tag{7.12}$$

The term in parentheses in eq. (7.11) is the mass flux in the ξ_m-direction. The expression in parentheses on the left hand side of eq. (7.12) is the u_j-momentum flux in the ξ_m-direction. The latter consists of a convective and a diffusive part. The diffusive part, in turn, is composed of an 'orthogonal' part (the ξ_m-derivative) and a 'skewed' part (cross derivatives).

As Meakin [MEA86] explains, this distinction is made because we do not know *a priori* the sign of the off-diagonal terms of the metric tensor g (the diagonal terms are always positive). The wrong sign may destroy diagonal dominance of the matrix representing the difference scheme that follows from integrating the momentum equation. In order to preserve diagonal dominance, which is a sufficient condition for convergence for several important iterative solution procedures, all terms pertaining to the skewed fluxes will be lumped into the source term of the difference scheme to be derived; they are lagged one iteration.

To make the distinction between skewed and orthogonal fluxes explicit, we write the flux $\Psi_{\xi_i}(u_j)$ of u_j-momentum in the ξ_i-direction as the sum of those two fluxes:

$$\Psi_{\xi_i}(u_j) = \Psi^o_{\xi_i}(u_j) + \Psi^s_{\xi_i}(u_j) , \tag{7.13}$$

with

$$\Psi^o_{\xi_i}(u_j) = \mathrm{J}\mathrm{U}^i u_j - \frac{1}{Re}\mathrm{J}g^{ii}\frac{\partial u_j}{\partial \xi_i} \quad \text{(no summation over } i\text{)} \tag{7.14}$$

and

$$\Psi^{s}_{\xi_i}(u_j) = -\frac{1}{Re} J g^{ik}(1-\delta_{ik})\frac{\partial u_j}{\partial \xi_k} \text{ (no summation over } i\text{)}, \tag{7.15}$$

where δ is the usual Kronecker symbol. Using this notation, the momentum equations can be written as:

$$\frac{\partial}{\partial \xi_m}\Big(\Psi^{o}_{\xi_m}(u_j) + \Psi^{s}_{\xi_m}(u_j)\Big) = -\frac{\partial}{\partial \xi_m}\Big(J\frac{\partial \xi_m}{\partial x_j}p\Big) \ . \tag{7.16}$$

'Raw' finite-difference forms of the continuity and momentum equations are obtained by integrating eqs. (7.11) and (7.16) over a cell, or control volume, with dimensions $\Delta\xi$ by $\Delta\eta$ in computational space (a cell is the rectangular region —in computational space— staked out by four neighboring grid points whose indices differ by at most one; see Fig. 7.1). The line integrals on the left, right, top and bottom faces of the cell, indicated by w, e, n and s, respectively (for west, east, north and south), are computed using the midpoint rule. How quantities at the midpoints of cell faces are determined is the subject of modern wizardry, some of which will be discussed in the next section. The raw equations themselves are:

$$\Big\{(JU^1)_e - (JU^1)_w\Big\}\Delta\eta + \Big\{(JU^2)_n - (JU^2)_s\Big\}\Delta\xi = 0 \ , \tag{7.17}$$

and

$$\begin{aligned}
&\Big\{\big(\Psi^{o}_{\xi}(u_j)\big)_e - \big(\Psi^{o}_{\xi}(u_j)\big)_w\Big\}\Delta\eta + \Big\{\big(\Psi^{o}_{\eta}(u_j)\big)_n - \big(\Psi^{o}_{\eta}(u_j)\big)_s\Big\}\Delta\xi + \\
&\Big\{\big(\Psi^{s}_{\xi}(u_j)\big)_e - \big(\Psi^{s}_{\xi}(u_j)\big)_w\Big\}\Delta\eta + \Big\{\big(\Psi^{s}_{\eta}(u_j)\big)_n - \big(\Psi^{s}_{\eta}(u_j)\big)_s\Big\}\Delta\xi = \\
&-\Big\{(J\frac{\partial \xi}{\partial x_j}p)_e - (J\frac{\partial \xi}{\partial x_j}p)_w\Big\}\Delta\eta - \Big\{(J\frac{\partial \eta}{\partial x_j}p)_n - (J\frac{\partial \eta}{\partial x_j}p)_s\Big\}\Delta\xi \ .
\end{aligned} \tag{7.18}$$

7.3 Discretization on Modified Staggered Grid

The choice of grid for the Navier-Stokes equations depends on the type of pressure boundary conditions prescribed. For boundary-layer flows the pressure is supplied explicitly. For environmental flows the hydrostatic pressure might be given. In either case the pressure is not an unknown, so no boundary conditions are needed.

Many problems in mechanical engineering require computing internal flows for which the pressure is not known at the boundary. Therefore, we wish not to have discretization points on the boundary where the pressure needs to be specified. The standard trick is to apply a staggered variable arrangement, shown in Fig. 7.2. Here the physical quantities describing the flow are specified not at grid points, but at the centers of cells (pressure) and on the cell faces (velocity). The preeminent feature of this arrangement (loosely called staggered grid) is that every u-velocity node is straddled horizontally by two pressure nodes, and that every v-velocity node is straddled vertically by two pressure nodes. This enables us to compute conveniently x-derivatives of the pressure at those u-velocity nodes, and y-derivatives at the v-velocity nodes, which is exactly what we need to discretize the momentum equations in a Cartesian coordinate system; Moreover, no pressure nodes are needed on the boundary.

An *additional* advantage of the standard staggered grid is the simple, compact difference stencil for the continuity equation; for the cell indicated in Fig. 7.3, only two u-velocity nodes and two v-velocity nodes are needed. The staggered grid also provides momentum and kinetic energy conservation.

Finally, on the staggered grid every *discrete momentum equation couples* two adjacent pressure nodes. This strong coupling inhibits the spurious oscillations in the pressure that are allowed by the nonstaggered variable arrangement (see, *e.g.*, [PAT81]).

All these reasons have led to wide acceptance of the staggered grid for internal incompressible-flow calculations. Unfortunately, most of the advantages of the staggered grid are a coincidence associated with Cartesian coordinate systems.

A look at eq. (7.18) shows that derivatives of the pressure in *both* generalized-coordinate directions occur in both momentum equations. This not only increases the number of pressure nodes included in the difference stencils for the momentum equation, but necessitates the introduction of pressure nodes on the boundary as well, even for a simple, rotated Cartesian coordinate system! Similarly, for the continuity equation both components of velocity are needed on every cell face. This again enlarges the difference stencil significantly. The difference stencils in computational space for the linearized u-momentum equation and the continuity equation on a standard staggered grid are shown in Fig. 7.4.

One can use the above difference stencils, but then funny things happen to the pressure gradient terms and the continuity equation. The indices i and j are used in the ξ- and η-directions respectively (half indices indicate points in between regular grid points). The following discretization of the continuity equation is obtained if we use the obvious central differencing for all derivatives and approximate midpoint values by symmetric averages:

$$\begin{aligned}
&\{(\mathbf{J}U)_e - (\mathbf{J}U)_w\}\,\Delta\eta + \{(\mathbf{J}U)_n - (\mathbf{J}U)_s\}\,\Delta\xi \approx \\
&\Bigg\{\mathbf{J}_{i,j-\frac{1}{2}} \left[\left(\frac{\partial\xi}{\partial x}\right)_{i,j-\frac{1}{2}} u_{i,j} + \left(\frac{\partial\xi}{\partial y}\right)_{i,j-\frac{1}{2}} (v_{i,j} + v_{i+1,j} + v_{i,j-1} + v_{i+1,j-1})/4\right] - \\
&\quad \mathbf{J}_{i-1,j-\frac{1}{2}} \left[\left(\frac{\partial\xi}{\partial x}\right)_{i-1,j-\frac{1}{2}} u_{i-1,j} + \right. \\
&\qquad \left.\left(\frac{\partial\xi}{\partial y}\right)_{i-1,j-\frac{1}{2}} (v_{i-1,j} + v_{i,j} + v_{i-1,j-1} + v_{i,j-1})/4\right]\Bigg\}\,\Delta\eta + \\
&\Bigg\{\mathbf{J}_{i-\frac{1}{2},j} \left[\left(\frac{\partial\eta}{\partial x}\right)_{i-\frac{1}{2},j} (u_{i,j} + u_{i,j+1} + u_{i-1,j} + u_{i-1,j+1})/4 + \left(\frac{\partial\eta}{\partial y}\right)_{i-\frac{1}{2},j} v_{i,j}\right] - \\
&\quad \mathbf{J}_{i-\frac{1}{2},j-1} \left[\left(\frac{\partial\eta}{\partial x}\right)_{i-\frac{1}{2},j-1} (u_{i,j-1} + u_{i,j} + u_{i-1,j-1} + u_{i-1,j})/4 + \right. \\
&\qquad \left.\left(\frac{\partial\eta}{\partial y}\right)_{i-\frac{1}{2},j-1} v_{i,j-1}\right]\Bigg\}\,\Delta\xi = 0\,. \qquad (7.19)
\end{aligned}$$

Under the assumption that the spacing in computational space is unity in both coordinate directions, the above expression can be written in case of a rotated Cartesian grid (no indexing of metric terms needed, $\mathbf{J} \equiv \mathit{constant}$):

$$\begin{aligned}
&\frac{\partial\xi}{\partial x}(u_{i,j} - u_{i-1,j}) + \frac{\partial\xi}{\partial y}(v_{i+1,j} + v_{i+1,j-1} - v_{i-1,j} - v_{i-1,j-1}) + \\
&\frac{\partial\eta}{\partial x}(u_{i,j+1} + u_{i-1,j+1} - u_{i,j-1} - u_{i-1,j-1}) + \frac{\partial\eta}{\partial y}(v_{i,j} - v_{i,j-1}) = 0\,. \qquad (7.20)
\end{aligned}$$

If we now apply the seemingly innocuous transformation $x = -\eta$, $y = \xi$, which corresponds to a rotation through an angle of 90 degrees, two terms drop out of eq. (7.20), resulting in:

$$v_{i+1,j} + v_{i+1,j-1} - v_{i-1,j} - v_{i-1,j-1} - u_{i,j+1} - u_{i-1,j+1} + u_{i,j-1} + u_{i-1,j-1} = 0\,. \quad (7.21)$$

Consequently, all the velocity nodes on the cell faces drop out of the difference equation, leaving only nodes further removed from the cell, as indicated in Fig. 7.5 by the arrows. The standard staggered variable arrangement, which is so convenient for the aligned Cartesian grid, becomes inappropriate for the rotated Cartesian grid. A similar situation arises with respect to the pressure nodes in the momentum equations. The stencils again become a lot larger, skipping nodes closer to the discretization point. This problem cannot be overcome by using other differencing schemes; for some angle of rotation the stencil will again deteriorate. Clearly, this situation is undesirable and unnatural, as rotation should not introduce any difficulties.

The root of the problem is that the staggered grid makes it easy to evaluate certain derivatives, but makes it hard to compute others. It works for a regular Cartesian grid, because only special derivatives occur. In general, however, all derivatives of the pressure and the velocities are needed. We therefore propose a modified staggered grid, shown in Fig. 7.6. This particular variable arrangement, also known as the ICED-ALE arrangement, has been investigated by other researchers [PER85], [SHI89]. Shih *et al.* compared the modified staggered grid with eight other variable arrangements, including the nonstaggered grid and the standard staggered grid. The modified staggered grid came out best on a list of eleven quality criteria, even when applied to a regular Cartesian coordinate system. The only two drawbacks according to these authors are that the discrete continuity equations can become inconsistent, and that the pressure may exhibit a checkerboard oscillation pattern. The first objection is easily removed. Shih argues that an exact benchmark solution of polynomial shape is not preserved exactly by the modified staggered grid. However, the polynomial is of degree four, for which the central differencing used in the test cases is not exact, and no inconsistency exists. The second objection, pressure oscillations, can also be taken away, as we will show. Furthermore, for arbitrary coordinate systems the modified staggered grid has the nice properties that the standard staggered grid has exclusively for regular Cartesian coordinates, *i.e.*:

- easy evaluation of central differences of all flow variables,

- no necessity for pressure nodes on the boundary,
- compact difference stencils.

Every interior velocity node is straddled in both generalized-coordinate directions by two pairs of pressure nodes. Thus, derivatives in both directions can be evaluated by averaging among pairs of pressure nodes and taking the proper central differences between these pairs. For example, in Fig. 7.7 the pressures at the nodes indicated by open circles (○) are averaged at the location marked by an asterisk (∗). Similarly, the pressures indicated by the solid circles (●) are averaged at their asterisk location. The derivative at the central velocity node in the generalized vertical direction is now computed by taking the difference between the two asterisk pressures. In the same vein, pairs of pressures containing one open and one solid circle each can be averaged at locations indicated by plus signs. Differences between pressures at the plus locations then yield derivatives in the generalized horizontal direction. Pressure nodes on the boundary are never needed.

The averaging process carried out above might tempt one to pose the question: why not specify the pressure nodes at the asterisk and plus locations, as apparently the pressures at these locations have more physical significance than the pressures at the cell centers? Although the latter is true, severe difficulties are created by placing pressure nodes at the cell faces. First, more pressure nodes are required than in the cell-center alternative, whereas no new difference equations are introduced: the system becomes undetermined. Second, the cell-face pressures are staggered with respect to one another on adjacent (half)grid lines, which makes for difficult data structures. For these reasons we will stick to the cell-centered pressures as *primary* pressure unknowns. The *physical* pressures in between can be computed afterwards through averaging.

Let us now turn to the issue of checkerboard patterns occurring in the pressure. It is important to realize that oscillations are a feature of the primary pressure, and not of the physical pressure. To identify any spurious solution components in the primary pressure, we will determine the null space of the discrete (pressure) gradient operator on a uniform Cartesian grid with square mesh cells. In other words, we will find those solutions for the primary pressures that make no contribution to the pressure gradient. Hence, we require the discrete $\xi-$ and $\eta-$derivatives to be zero. To avoid using fractional indices in the analysis, we set $\nu_{i,j} = p_{i-\frac{1}{2},j-\frac{1}{2}}$, so

$$\nu_{i+1,j+1} + \nu_{i+1,j} - \nu_{i,j+1} - \nu_{i,j} = 0 \ , \tag{7.22}$$

$$\nu_{i+1,j+1} - \nu_{i+1,j} + \nu_{i,j+1} - \nu_{i,j} = 0 \ . \tag{7.23}$$

Difference equations of this kind have solutions of the form $\nu_{i,j} = z^i w^j$ (no boundary conditions need to be satisfied, since there are no pressure nodes on the boundary). Substituting this expression for ν into eqs. (7.22) and (7.23) yields:

$$zw + z - w - 1 = 0 \ , \tag{7.24}$$

$$zw - z + w - 1 = 0 \ . \tag{7.25}$$

Adding and subtracting these equations gives:

$$zw = 1, \qquad z = w, \qquad \text{so :} \quad z = \pm 1 \ . \tag{7.26}$$

Thus, the null space of the discrete pressure gradient operator has dimension 2, and its elements can be written in the following way:

$$\nu_{i,j} = \alpha + \beta(-1)^{i+j} \ . \tag{7.27}$$

The first constant, α, represents the fact that the zero level of pressure is undetermined. The second term represents the checkerboard pressure pattern, a spurious mode. If we now define the true pressures, $\tilde{p}$, on the ξ-grid lines by:

$$\tilde{p}_{i-\frac{1}{2},j} = \left(p_{i-\frac{1}{2},j-\frac{1}{2}} + p_{i-\frac{1}{2},j+\frac{1}{2}}\right)/2 \ , \tag{7.28}$$

and those on η-grid lines by:

$$\tilde{p}_{i,j-\frac{1}{2}} = \left(p_{i-\frac{1}{2},j-\frac{1}{2}} + p_{i+\frac{1}{2},j-\frac{1}{2}}\right)/2 \ , \tag{7.29}$$

then, from eq. (7.27), all spurious modes in these pressures collapse to:

$$\tilde{p} = \alpha \ , \tag{7.30}$$

i.e., a constant can be added to the entire pressure field. No nonphysical oscillations can occur in these pressures.

Several ways are available to store the physical pressures in a more convenient way than at the zig-zag locations on the cell faces. The easiest way is to introduce yet another averaging procedure: define the pressure, $\hat{p}$, at every grid point where the velocities are specified as the average of the four surrounding values of $\tilde{p}$, so:

$$\hat{p}_{i,j} = \left(\tilde{p}_{i-\frac{1}{2},j} + \tilde{p}_{i+\frac{1}{2},j} + \tilde{p}_{i,j-\frac{1}{2}} + \tilde{p}_{i,j+\frac{1}{2}}\right)/4\,. \tag{7.31}$$

Using eqs. (7.28) and (7.29), this definition simplifies to:

$$\hat{p}_{i,j} = \left(p_{i-\frac{1}{2},j-\frac{1}{2}} + p_{i+\frac{1}{2},j-\frac{1}{2}} + p_{i-\frac{1}{2},j+\frac{1}{2}} + p_{i+\frac{1}{2},j+\frac{1}{2}}\right)/4\,. \tag{7.32}$$

This means we can omit the computation of $\tilde{p}$.

A more elegant way of getting rid of oscillations is to subtract the checkerboard component from the primary pressures. This has the advantage that no additional storage is required. The idea is as follows: Assume the primary pressure can be written as a true pressure, p^t, plus a checkerboard component, βp^{ch}, where p^{ch} is the null vector corresponding to a spurious mode, so $p = p^t + \beta p^{ch}$. Then determine the checkerboard component by taking the inner product, and subtract it from the primary pressure to get the true pressure, *i.e.*, $p^t = p - p^{ch}(p.p^{ch})/\|p^{ch}\|_2^2$. This corresponds to a projection of the primary pressure onto a space orthogonal to the oscillatory component.

Although this filtering looks slightly involved, it is actually a very simple process. On an n by m grid, we have: $\|p^{ch}\|_2 = \sqrt{(nm)}$. The inner product is determined by adding and subtracting adjacent primary pressures alternatingly.

Another way of looking at the spurious oscillations is the following: what we are computing ultimately is a pressure *gradient* field. Depending on the discrete operator used to represent the gradient, spurious modes that occur in the pressure may not affect its gradient. But what counts is that the gradients be accurate. Should we want to invert the gradient operator, we will have to deal with its null space, or make sure it does not enter into the final pressures. The latter can be achieved through the filtering or averaging procedures.

It should be noted that these strategies are applicable for other staggered variable arrangements. As Anderson [AND88] points out, the nonstaggered-grid discrete gradient operator has a four-dimensional null space. In this case three independent 'spurious' modes are present. These can be eliminated by averaging or by applying the filtering method to each oscillatory component.

Finally, it should be pointed out that spurious modes are artifacts of variable staggering, having nothing to do with grid curvature. Hence, the filtering method will work in exactly the same way on a general, curvilinear, nonorthogonal grid as it does on a uniform Cartesian grid with square mesh cells. To see this, set the discrete pressure gradient on a general grid equal to zero, again making use of the variable ν to avoid fractional indices:

$$a_{i,j}(\nu_{i,j+1}+\nu_{i+1,j+1})+b_{i,j}(\nu_{i,j}+\nu_{i+1,j})+c_{i,j}(\nu_{i+1,j}+\nu_{i+1,j+1})+d_{i,j}(\nu_{i,j+1}+\nu_{i,j})=0\ , \tag{7.33}$$

$$e_{i,j}(\nu_{i,j+1}+\nu_{i+1,j+1})+f_{i,j}(\nu_{i,j}+\nu_{i+1,j})+g_{i,j}(\nu_{i+1,j}+\nu_{i+1,j+1})+h_{i,j}(\nu_{i,j+1}+\nu_{i,j})=0\ . \tag{7.34}$$

The coefficients $a_{i,j}$ through $h_{i,j}$ represent the changing metrics. The above equations are simply two discrete conditions on the four-point difference stencil involving the midpoint averages of the pressure nodes surrounding each velocity node. For these equations to make sense, every pair must be linearly independent. In general no simple solutions to these equations exist, but we will try exponential solutions of the form $\nu_{i,j}=z^iw^j$ anyway. Upon substitution we find:

$$(1+z)(w\ a_{i,j}+b_{i,j})+(w+1)(z\ c_{i,j}+d_{i,j})=0\ , \tag{7.35}$$

$$(1+z)(w\ e_{i,j}+f_{i,j})+(w+1)(z\ g_{i,j}+h_{i,j})=0\ . \tag{7.36}$$

Obviously, despite the fact that the coefficients vary from point to point, the solution $w=z=-1$ still holds. It is not clear that $w=z=1$ is also a solution. Rather, we will require explicitly that it be a solution, because we want a constant pressure field to be preserved under a general discrete coordinate transformation. This requirement poses restrictions on the way the metric terms are evaluated.

It follows that the discrete pressure gradient operator has the same null vectors on a general curved grid as it does on a uniform Cartesian grid with square mesh cells. There are no more null vectors if the discrete transformation is nondegenerate. Hence, the checkerboard pattern in the pressure occurs on curved grids and can be eliminated in the same way.

One subtle detail of the filtering procedure needs to be discussed. The removal of oscillatory components was based on the observation that seemingly pathological pressure fields could satisfy the condition: $\text{grad}\, p = 0$, due to the 'transparency' of checkerboard patterns to the discrete gradient operator. The checkerboard had to go. But, as we will see shortly, the filtering procedure described above does not leave every constant pressure field intact, although disturbances grow smaller and smaller with decreasing mesh size. How is this possible? It is tempting to say that there is energy in the length scales of the order of mesh size, so that there truly is a component of the solution matching the checkerboard pattern. This component then would become less and less significant, as its spatial frequency goes up with decreasing mesh size. This explanation, however, is flawed. Uniform fields have no energy in any nontrivial frequency. Moreover, no problem occurs when the total number of grid cells is even.

So what is going on here? The explanation is a plain linear algebra argument: We want to be able to preserve uniform pressure fields and, at the same time, eliminate spurious oscillations. Hence, we should add a *combination* of null vectors to the solution which does just that. This combination has to be orthogonal to the uniform pressure field solution if we want to leave constant solutions intact. Interestingly, the checkerboard vector $(-1)^{i+j}$ may not be orthogonal to the constant vector, depending on the parity of the number of grid cells. If the parity is even, as many grid cells get weighted with $+1$ as get weighted with -1, and orthogonality obtains. But if the parity is odd, one more grid cell will get weight $+1$, and orthogonality is lost. Clearly, the way out is to orthogonalize the null vector to be added to the solution with respect to the constant vector. This process is vaguely reminiscent of Gram-Schmidt orthogonalization of the null space of the gradient operator, because one of the null vectors is indeed the constant vector itself. The analogy fails, however, if a null space of dimension higher than two is encountered. In that case, orthogonalization does not take place recursively, but only with respect to the constant vector. To establish the orthogonalization. write the oscillatory null vector, p^o, as the sum of the checkerboard vector, p^{ch}, and a constant component, p^c, so $p^o = a\,p^c + p^{ch}$. Demand that $(p^o.p^c) = 0$, so $a = -(p^o.p^c)/\|p^c\|_2^2$. For an n by m grid, we obtain:

$$p^o_{i,j} = -\frac{(mn) \bmod 2}{mn} + (-1)^{i+j} \,. \tag{7.37}$$

Ultimately, a wiggle-free solution, p^f, is obtained by subtracting the oscillatory component from the 'raw' pressure:

$$p^f = p - \mu p^o \,, \quad \text{with } \mu = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{p^o_{i,j}\, p_{i,j}}{mn - (mn \bmod 2)/mn} \,. \tag{7.38}$$

Figs. 7.8-7.10 show the effects of filtering the pressure field of the lid-driven cavity flow (see later in this chapter) for a 21x21 grid. The initial pressure vector has components in both directions of the null space of the discrete gradient operator. In Fig. 7.8 these raw pressures are shown. The checkerboard pattern is clearly visible, and solution features are obscured. In Fig. 7.9 the pressure field is filtered by subtracting the checkerboard pattern *without* preserving a uniform pressure field. Although the quality of the solution is much better, some wiggles are still present. Finally, Fig. 7.10 shows the solution filtered by eq. (7.38). No wiggles are present, except those due to the coarseness of the grid and the resolution of the contour-plotting routine.

Now that an appropriate variable staggering arrangement has been selected, we return to the discretization of the continuity and momentum equations. Without loss of generality we can use unit spacing in the computational domain, so $\Delta\xi = \Delta\eta = 1$. Employing symmetric averages and central differences in eq. (7.17), the following discrete approximation to the continuity equation is obtained:

$$\Bigg\{ \mathbf{J}_{i+1,j+\frac{1}{2}} \left[\left(\frac{\partial\xi}{\partial x}\right)_{i+1,j+\frac{1}{2}} (u_{i+1,j+1} + u_{i+1,j})/2 + \left(\frac{\partial\xi}{\partial y}\right)_{i+1,j+\frac{1}{2}} (v_{i+1,j+1} + v_{i+1,j})/2 \right] -$$

$$\mathbf{J}_{i,j+\frac{1}{2}} \left[\left(\frac{\partial\xi}{\partial x}\right)_{i,j+\frac{1}{2}} (u_{i,j+1} + u_{i,j})/2 + \left(\frac{\partial\xi}{\partial y}\right)_{i,j+\frac{1}{2}} (v_{i,j+1} + v_{i,j})/2 \right] \Bigg\} +$$

$$\Bigg\{ \mathbf{J}_{i+1,j+\frac{1}{2}} \left[\left(\frac{\partial\eta}{\partial x}\right)_{i+\frac{1}{2},j+1} (u_{i+1,j+1} + u_{i,j+1})/2 + \left(\frac{\partial\eta}{\partial y}\right)_{i+\frac{1}{2},j+1} (v_{i+1,j+1} + v_{i,j+1})/2 \right] -$$

$$\mathrm{J}_{i+\frac{1}{2},j}\left[\left(\frac{\partial\eta}{\partial x}\right)_{i+\frac{1}{2},j}(u_{i+1,j}+u_{i,j})/2+\left(\frac{\partial\eta}{\partial y}\right)_{i+\frac{1}{2},j}(v_{i+1,j}+v_{i,j})/2\right]\Bigg\}=0\,. \tag{7.39}$$

Symbolically, this is most easily represented by using the location subscripts sw for south-west (indices (i,j)), nw for north-west (indices $(i,j+1)$), se for south-east (indices $(i+1,j)$), and ne for north-east (indices $(i+1,j+1)$), and the function superscripts cu for u-coefficient of the continuity equation and cv for v-coefficient of the continuity equation. The location indicators are relative to the center point of the continuity-equation control volume with indices $(i+\frac{1}{2},j+\frac{1}{2})$, which is the point at which the pressure is specified. Using the capital letter A as a generic coefficient, we write eq. (7.39) as:

$$A^{cu}_{sw}u_{sw}+A^{cu}_{nw}u_{nw}+A^{cu}_{se}u_{se}+A^{cu}_{ne}u_{ne}+A^{cv}_{sw}v_{sw}+A^{cv}_{nw}v_{nw}+A^{cv}_{se}v_{se}+A^{cv}_{ne}v_{ne}=0\,. \tag{7.40}$$

The values of the coefficients are listed in Appendix C.

The discretization of the momentum equations is a little more involved, due to the convection/diffusion operator. We will use the procedure outlined by Meakin [MEA86] to represent the orthogonal and skewed momentum fluxes, which ensures diagonal dominance of the discrete convection/diffusion operator. Meakin's approach rests upon the discretization technique described in Patankar [PAT80] for the scalar convection/diffusion equation on a Cartesian grid. This discretization is adopted because it leads to stable iterative procedures.

Because this technique is well-documented, we will simply summarize it here for the orthogonal flux, $\Psi^{o}_{\xi}(u)=\mathrm{J}\mathrm{U}^{i}u-\frac{1}{Re}\mathrm{J}g^{11}\frac{\partial u}{\partial\xi}$, and the skewed flux, $\Psi^{s}_{\xi}(u)=-\frac{1}{Re}\mathrm{J}g^{1k}(1-\delta_{1k})\frac{\partial u}{\partial\xi_k}$, of u-momentum through the *east* face of the momentum control volume in computational space. Indexing is now relative to the center of the momentum control volume at the point P with indices (i,j), which coincides with a velocity node. Subscripts for the velocities are capitalized to indicate that they are integers. Subscripts in lower case letters refer to fractional indices. So we have: W and w for west (indices $(i-1,j)$ and $(i-\frac{1}{2},j)$, respectively), E and e for east (indices $(i+1,j)$ and $(i+\frac{1}{2},j)$, respectively), N and n for north (indices $(i,j+1)$ and $(i,j+\frac{1}{2})$, respectively), and S and s for south (indices $(i,j-1)$ and $(i,j-\frac{1}{2})$, respectively). See Fig. 7.11 for a diagram of locations for the u-velocities.

With these definitions, the orthogonal flux is computed as follows:

$$\begin{aligned}
\left(\Psi_\xi^o(u)\right)_e &= F_e\, u_p + \{D_e\, \Lambda(|P_e|) + \max[-F_e, 0]\}\ (u_P - u_E)\,, && (7.41)\\
F_e &= \mathrm{J}_e \mathrm{U}_e^1 = \mathrm{J}_{i+\frac{1}{2},j} \mathrm{U}^1_{i+\frac{1}{2},j}\,, && (7.42)\\
P_e &= F_e / D_e\,, && (7.43)\\
D_e &= 2\left[\frac{1}{\Gamma_P} + \frac{1}{\Gamma_E}\right]^{-1}\,, && (7.44)\\
\Gamma_P &= \Gamma_{i,j} = \frac{1}{Re}\mathrm{J}_{i,j} g^{11}_{i,j}\,, && (7.45)\\
\Gamma_E &= \Gamma_{i+1,j} = \frac{1}{Re}\mathrm{J}_{i+1,j} g^{11}_{i+1,j}\,. && (7.46)
\end{aligned}$$

F stands for mass flux, P is the cell Peclet number, D is the weighted conductance, and Γ is the local effective diffusivity. The definition of the function Λ depends on the kind of differencing used for the convective term. Upwind differencing corresponds to $\Lambda(P) \equiv 0$, central differencing to $\Lambda(P) = 1 - P/2$, and the so-called power law —which gradually changes from central to upwind differencing as the cell Peclet number increases— corresponds to $\Lambda(P) = \max[0, (1 - 0.1\,|P|)^5]$. The contravariant component of velocity $\mathrm{U}^1_{i+\frac{1}{2},j}$ is approximated by:

$$\mathrm{U}^1_{i+\frac{1}{2},j} = \left(\frac{\partial \xi}{\partial x}\right)_{i+\frac{1}{2},j} (u_{i+1,j} + u_{i,j})/2 + \left(\frac{\partial \xi}{\partial y}\right)_{i+\frac{1}{2},j} (v_{i+1,j} + v_{i,j})/2\,. \qquad (7.47)$$

Now we define the skewed flux through the east face:

$$\left(\Psi_\xi^s(u)\right)_e = -\frac{1}{Re}\mathrm{J}_{i+\frac{1}{2},j} g^{12}_{i+\frac{1}{2},j} \left\{(u_{i,j+1} - u_{i,j-1})/2 + (u_{i+1,j+1} - u_{i+1,j-1})/2\right\}/2\,. \qquad (7.48)$$

Similar definitions can be given for orthogonal and skewed u- and v-momentum fluxes through the other faces of the momentum control volume.

Symbolically, the difference approximations to the momentum equations can be written as follows, using the function superscripts u and v for velocity coefficients and up and vp for the pressure coefficients in the u- and v-momentum equations, respectively:

$$A^u_P\, u_P = A^u_E\, u_E + A^u_W\, u_W + A^u_N\, u_N + A^u_S\, u_S + sc^u +$$

$$A^{up}_{nw}\, p_{nw} + A^{up}_{sw}\, p_{sw} + A^{up}_{ne}\, p_{ne} + A^{up}_{se}\, p_{se} \,, \tag{7.49}$$

$$\begin{aligned} A^{v}_{P}\, v_P = {} & A^{v}_{E}\, v_E + A^{v}_{W}\, v_W + A^{v}_{N}\, v_N + A^{v}_{S}\, v_S + sc^{v} + \\ & A^{vp}_{nw}\, p_{nw} + A^{vp}_{sw}\, p_{sw} + A^{vp}_{ne}\, p_{ne} + A^{vp}_{se}\, p_{se} \,. \end{aligned} \tag{7.50}$$

The coefficients in these discrete equations are given in Appendix C. The quantity sc is a source term, containing the skewed flux terms, body forces, boundary conditions, and so-called defect corrections defined below.

7.4 Central-Difference Correction

We are interested in high-accuracy difference schemes, but these tend to become unstable for high cell Peclet numbers. The idea then is to solve a system with an inaccurate but stable difference operator on the left-hand side, and to add a correction term to the right-hand side, which is the difference between the effect of the left-hand-side operator and the operator of the desired accuracy. Let the low-accuracy operator be L_1, and the high-accuracy operator L_2. Discretizing using only L_1 leads to systems of the form:

$$L_1 u^{k+1} = f \,, \tag{7.51}$$

where the superscript indicates that we are computing the solution iteratively, $k+1$ being the new iteration level. If we try to solve this system using L_2 on the left hand side, the iterative procedure will not converge. Instead, we solve:

$$L_1 u^{k+1} = L_1 u^k - L_2 u^k + f \,. \tag{7.52}$$

At convergence, $u^{k+1} \equiv u^k$, and the terms involving $L_1 u$ cancel, resulting in a higher accurate solution. Roughly speaking, we get the robustness and stability of the low-accuracy operator, but the precision of the high-accuracy operator. It has been observed, though, that convergence slows down if the difference between the two operators is too big. Therefore, we use as the low-accuracy operator the power-law scheme, which comes as close as possible to the central difference operator without losing diagonal dominance of the convection/diffusion operator. The high-accuracy scheme is central differencing. For that reason, the term added to the source term is called the central-difference correction (cdc).

Some words of caution regarding cdc are in order:

- The addition of the cdc to the source term *does* influence the stability of the iterative procedure, which can be seen most easily by writing the correction scheme as a matrix splitting. In the limit we want to solve $L_2 u = f$, but inverting L_2 iteratively cannot be done in a stable fashion. Hence, write L_2 as $L_1 - (L_1 - L_2)$. Usually, we cannot invert L_1 directly either, so it is split in turn into two parts: $L_1 = M - N$. Ultimately, we solve the iterative sequence: $Mu^{k+1} = (N + [L_1 - L_2])u^k + f$. The error-amplification matrix, E, of this iteration is $E = M^{-1}(N + [L_1 - L_2])$. Convergence is governed by the spectral radius of this matrix, which is not the matrix of the system without cdc ($E = M^{-1}N$).
- It is important that the computation of $L_1 u^k$ in the cdc be consistent with the computation of $L_1 u^{k+1}$. This means that the *same coefficients* should be used on both sides.

7.5 Application of Modified Staggered Grid: U-Shaped Channel

We now employ the modified staggered grid to compute the flow in a strongly curved geometry in order to test its ability to handle large degrees of rotation. A good candidate is the two-dimensional U-shaped channel (Fig. 7.12), which exhibits a 180-degree turning angle. The grid used is shown in Fig. 7.13. Plain Poiseuille flow obtains sufficiently far upstream and downstream of the bend, whereas asymptotic results can also be obtained for the flow inside the curved section sufficiently far away from the straight sections. Assuming rotation-symmetric, tangential flow in the semi-circular bend ($u_r = 0$, $u_\theta = u_\theta(r)$), the momentum equations simplify to:

$$-\frac{\partial u_\theta^2}{\partial r} = -\frac{\partial p}{\partial r}, \tag{7.53}$$

$$0 = -\frac{1}{r}\frac{\partial p}{\partial \theta} + \frac{1}{Re}\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial r u_\theta}{\partial r}\right). \tag{7.54}$$

The continuity equation is satisfied trivially.

From the above, it follows that the pressure can be written as:

$$p = \alpha\theta + g(r)\,, \tag{7.55}$$

where g is a function of the radius r.

Using the coordinate system and the dimensions shown in Fig. 7.12, we obtain for the tangential velocity:

$$u_\theta = Q\frac{\frac{1}{2}r\ln\left(\frac{r^2}{R^2-d^2}\right) + [(R^2+d^2)r - (R^2-d^2)^2/r]\frac{1}{4Rd}\ln\left(\frac{R-d}{R+d}\right)}{\frac{(R^2-d^2)^2}{4Rd}\ln^2\left(\frac{R-d}{R+d}\right) - Rd}\,, \tag{7.56}$$

in which Q is the nondimensionalized mass flux. The multiplication factor α is:

$$\alpha = \frac{2Q}{Re}\left(\frac{(R^2-d^2)^2}{4Rd}\ln^2\left(\frac{R-d}{R+d}\right) - Rd\right)^{-1}. \tag{7.57}$$

The pressure difference across the channel is determined by integrating eq. (7.53) numerically between two stations directly adjacent to opposite walls, $r_1 = 2.01478$ and $r_2 = 2.98522$, given $R = 2.5$, $d = 0.5$, and $Q = \frac{3}{4}d$ (mass flux for Poiseuille flow with maximum velocity of 1). The result is: $\Delta p = p_2 - p_1 = 0.216191$. For the same parameters we find that $\alpha = -20.1088/Re$.

Results of numerical computations with the modified staggered grid for Reynolds numbers 100, 300 and 600 are shown in Fig. 7.14. The solid lines and dashed lines represent the pressures near the outer radius and the inner radius of the channel, respectively. The pressure difference yields the centripetal force. The thin parallel lines in each frame represent the asymptotic values of the two pressures for the rotation-symmetric annular flow. In all cases the asymptotic values are reached within 0.5% accuracy. No separation was observed, despite the occurrence of adverse pressure gradients at the outer wall at the beginning of the bend, and at the inner wall at the end of the bend. Some of the computed θ-velocity profiles are shown in Fig. 7.15.

7.6 Evaluation of Metrics

The metric terms in the discretizations of the Navier-Stokes equations in Section 7.3 have not yet been defined. As long as the metrics are computed consistently, the overall discretization of the Navier-Stokes equations will be consistent, and the numerical solution will converge to the physical solution as the mesh size goes to zero.

In many engineering applications, however, it is considered desirable that the numerical solution for a finite mesh size share some special features with the physical solution; among these are the conservation properties which guarantee that, even on a coarse mesh, mass and momentum are conserved exactly (up to machine accuracy). The finite-volume-type approach taken in Section 7.3 ensures conservation, regardless of how the metric terms are computed.

Another property deemed desirable is free-stream preservation. It means that uniform flow is an *exact* solution to the discrete problem. This is not a necessary requirement, because every nontrivial solution suffers from truncation error. There is no profound reason to demand that uniform flow not be distorted at all, as long as the error goes to zero as the mesh size goes to zero. So is the requirement of free-stream preservation only an esthetic one? No! There are two good reasons for creating schemes that exactly reproduce uniform flow. The first is that the computation of uniform flow is an excellent debugging device whose importance should not be overlooked, especially on curvilinear, nonorthogonal grids. The second is that in many cases it has been observed that free-stream-preserving schemes have superior convergence and accuracy properties [FLO83]. For these reasons we will construct our metrics such that uniform flow is an exact solution on *any* grid.

Uniform flow is defined by: $u(\xi,\eta) \equiv u$, $v(\xi,\eta) \equiv v$, $p(\xi,\eta) \equiv p$. Substituting these expressions into the u-momentum equation (7.18) yields:

$$\begin{aligned}
&\left\{\left(\Psi^o_\xi(u)\right)_e - \left(\Psi^o_\xi(u)\right)_w\right\}\Delta\eta + \left\{\left(\Psi^o_\eta(u)\right)_n - \left(\Psi^o_\eta(u)\right)_s\right\}\Delta\xi + \\
&\left\{\left(\Psi^s_\xi(u)\right)_e - \left(\Psi^s_\xi(u)\right)_w\right\}\Delta\eta + \left\{\left(\Psi^s_\eta(u)\right)_n - \left(\Psi^s_\eta(u)\right)_s\right\}\Delta\xi = \\
-p&\left[\left\{\left(\mathbf{J}\frac{\partial\xi}{\partial x_j}\right)_e - \left(\mathbf{J}\frac{\partial\xi}{\partial x_j}\right)_w\right\}\Delta\eta + \left\{\left(\mathbf{J}\frac{\partial\eta}{\partial x_j}\right)_n - \left(\mathbf{J}\frac{\partial\eta}{\partial x_j}\right)_s\right\}\Delta\xi\right] . \qquad (7.58)
\end{aligned}$$

Computing the metric terms $\dfrac{\partial \xi_m}{\partial x_i}$ directly is difficult and inefficient. Instead, we make use of the metric identity:

$$\left(\frac{\partial \xi_m}{\partial x_i}\right) \equiv \left(\frac{\partial x_m}{\partial \xi_i}\right)^{-1} , \tag{7.59}$$

and the definition of the Jacobian determinant $\mathbf{J}$ to write:

$$\frac{\partial \eta}{\partial x} = -\frac{1}{\mathbf{J}}\frac{\partial y}{\partial \xi} , \quad \frac{\partial \xi}{\partial x} = \frac{1}{\mathbf{J}}\frac{\partial y}{\partial \eta} , \quad \frac{\partial \eta}{\partial y} = \frac{1}{\mathbf{J}}\frac{\partial x}{\partial \xi} , \quad \frac{\partial \xi}{\partial y} = -\frac{1}{\mathbf{J}}\frac{\partial x}{\partial \eta} . \tag{7.60}$$

For uniform flow the pressure-gradient terms in the momentum equations should vanish independently. If the Jacobian determinant is evaluated at the same locations as $\dfrac{\partial x_m}{\partial \xi_i}$, then the term in square brackets multiplying the pressure in eq. (7.58) becomes:

$$\left\{-\left(\frac{\partial y}{\partial \xi}\right)_n + \left(\frac{\partial y}{\partial \xi}\right)_s\right\}\Delta\xi + \left\{\left(\frac{\partial y}{\partial \eta}\right)_e - \left(\frac{\partial y}{\partial \eta}\right)_w\right\}\Delta\eta . \tag{7.61}$$

Approximating all the derivatives by central differences gives:

$$\begin{aligned}&\left\{-\left(\frac{y_{i+\frac{1}{2},j+\frac{1}{2}} - y_{i-\frac{1}{2},j+\frac{1}{2}}}{\Delta\xi}\right) + \left(\frac{y_{i+\frac{1}{2},j-\frac{1}{2}} - y_{i-\frac{1}{2},j-\frac{1}{2}}}{\Delta\xi}\right)\right\}\Delta\xi \; + \\ &\left\{+\left(\frac{y_{i+\frac{1}{2},j+\frac{1}{2}} - y_{i+\frac{1}{2},j-\frac{1}{2}}}{\Delta\eta}\right) - \left(\frac{y_{i-\frac{1}{2},j+\frac{1}{2}} - y_{i-\frac{1}{2},j-\frac{1}{2}}}{\Delta\eta}\right)\right\}\Delta\eta \equiv 0 .\end{aligned} \tag{7.62}$$

Consequently, the discrete pressure gradient vanishes identically if central differencing is applied to compute the metrics.

Now consider the momentum-flux terms; it is easily verified that the diffusive fluxes drop out, irrespective of how the metrics are evaluated. This takes care of the entire skewed flux terms, Ψ^s, and of most of the orthogonal flux terms, Ψ^o. The remaining parts are:

$$\Psi^o_e(u) = \mathbf{J}_e U^1_e u , \quad \Psi^o_w(u) = \mathbf{J}_w U^1_w u , \quad \Psi^o_n(u) = \mathbf{J}_n U^2_n u , \quad \Psi^o_s(u) = \mathbf{J}_s U^2_s u . \tag{7.63}$$

Substituting these expressions into the u-momentum equation and using the definition of the contravariant components of velocity and the metric identity:

$$U^1 = \frac{\partial \xi}{\partial x}u + \frac{\partial \xi}{\partial y}v = +\frac{1}{\mathbf{J}}\frac{\partial y}{\partial \eta}u - \frac{1}{\mathbf{J}}\frac{\partial x}{\partial \eta}v , \tag{7.64}$$

$$U^2 = \frac{\partial \eta}{\partial x}u + \frac{\partial \eta}{\partial y}v = -\frac{1}{J}\frac{\partial y}{\partial \xi}u + \frac{1}{J}\frac{\partial x}{\partial \xi}v\,, \tag{7.65}$$

we obtain the combined flux terms:

$$\left\{+\left(\frac{\partial y}{\partial \eta}\right)_e u - \left(\frac{\partial x}{\partial \eta}\right)_e v - \left(\frac{\partial y}{\partial \eta}\right)_w u + \left(\frac{\partial x}{\partial \eta}\right)_w v\right\}\Delta\eta\; +$$
$$\left\{-\left(\frac{\partial y}{\partial \xi}\right)_n u + \left(\frac{\partial x}{\partial \xi}\right)_n v + \left(\frac{\partial y}{\partial \xi}\right)_s u - \left(\frac{\partial x}{\partial \xi}\right)_s v\right\}\Delta\xi\,. \tag{7.66}$$

If we now introduce the operator D through:

$$D = \left[\left(\frac{\partial}{\partial \eta}\right)_e - \left(\frac{\partial}{\partial \eta}\right)_w\right]\Delta\xi + \left[-\left(\frac{\partial}{\partial \xi}\right)_n + \left(\frac{\partial}{\partial \xi}\right)_s\right]\Delta\eta\,, \tag{7.67}$$

expression (7.66) can be written as:

$$D(y)u - D(x)v\,. \tag{7.68}$$

According to eqs. (7.61) and (7.62), $D(x_i) \equiv 0$ if central differences are used for the metric terms. In that case the orthogonal fluxes also cancel, and the discrete u-momentum equation exactly captures the free stream. A similar argument can be applied to the v-momentum equation.

Because of the constancy of u, the left hand side of the discrete continuity equation (7.39) is a multiple of the orthogonal u-flux terms. Hence, it also vanishes. and the continuity equation is satisfied identically for uniform flow.

Thus, we find:

- Exact free stream capture is obtained by:
  - evaluating $\frac{\partial x_i}{\partial \xi_j}$ and the Jacobian determinant $\mathbf{J}$ at the locations where they are needed (no averaging to obtain mid-point values),
  - using central differences for $\frac{\partial x_i}{\partial \xi_j}$.
- No restriction is placed on the fashion in which $\mathbf{J}$ is calculated.
- No restriction is placed on the method for calculating the metric tensor components, g^{nm}, nor on where they are computed, as long as they are determined consistently.

7.7 Boundary-Condition Implementation

In fluid-flow problems non-Dirichlet boundary conditions often are applied on some parts of the boundaries. Dirichlet conditions are no problem; whenever a certain boundary value occurs in a difference stencil at an interior node, the known value is used.

Neumann conditions (or mixed conditions) do pose some problems, since the boundary values are part of the unknown solution. The honest way of incorporating Neumann boundary conditions is to augment the system of discrete equations for the interior nodes with the discretized boundary conditions. This, however, disrupts the banded structure of the matrix problem, because difference stencils for Neumann boundary conditions can be of arbitrary size (depending on the accuracy required) and must be one-sided.

To circumvent this difficulty we suggest the following procedure. Do not incorporate Neumann boundary conditions directly. As solution methods will be iterative, a reasonable guess for the boundary value is available from the result of the previous iteration. Hence, solve every problem as a Dirichlet problem and update the boundary values after every iteration, *i.e.*, the Neumann boundary conditions are lagged.

To analyze this lagging procedure we study the convergence behavior of the classical Gauss-Seidel method for the scalar, one-dimensional convection/diffusion equation:

$$-u_{xx} + au_x = f \ , \quad \text{with} : u(0) = \alpha \ , u'(1) = \beta \ . \tag{7.69}$$

Note that a Neumann condition is applied at the downstream boundary. Using central differences for all interior points of the uniform grid with mesh size h, we find:

$$(-1 - \tfrac{ah}{2})u_{i-1} + 2u_i + (-1 + \tfrac{ah}{2})u_{i+1} = h^2 f_i \ , \quad i = 1, n-1 \ , \tag{7.70}$$

$$\text{with} \quad u_k = u(kh) \ , f_k = f(kh) \ .$$

The Dirichlet boundary condition at the left boundary is included by setting $u_0 = \alpha$. A first-order-accurate approximation to the Neumann condition at the right boundary is: $u_n - u_{n-1} = h\beta$. This expression can be substituted into the difference equation at node $n-1$ to eliminate u_n from it, and we obtain:

$$(-1 - \tfrac{ah}{2})u_{n-2} + (1 + \tfrac{ah}{2})u_{n-1} = h^2 f_{n-1} - (-1 + \tfrac{ah}{2})h\beta\,. \tag{7.71}$$

The system of linear equations now reads:

$$Ax = b\,, \tag{7.72}$$

with

$$A = \begin{pmatrix} 2 & -1+\frac{ah}{2} & & & \\ -1-\frac{ah}{2} & 2 & -1+\frac{ah}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -1-\frac{ah}{2} & 2 & -1+\frac{ah}{2} \\ & & & -1-\frac{ah}{2} & 1+\frac{ah}{2} \end{pmatrix}, \tag{7.73}$$

and

$$b = \begin{pmatrix} h^2 f_1 + (1+\frac{ah}{2})\alpha \\ h^2 f_2 \\ \vdots \\ h^2 f_{n-2} \\ h^2 f_{n-1} - (-1+\frac{ah}{2})h\beta \end{pmatrix}, \quad x = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix}. \tag{7.74}$$

The straightforward Gauss-Seidel method can be written:

$$M_1 x^{k+1} = N_1 x^k + b\,, \tag{7.75}$$

with

$$M_1 = \begin{pmatrix} 2 & -1+\frac{ah}{2} & & \\ & \ddots & \ddots & \\ & & 2 & -1+\frac{ah}{2} \\ & & & 1+\frac{ah}{2} \end{pmatrix}, \quad N_1 = \begin{pmatrix} 0 & & & \\ 1+\frac{ah}{2} & 0 & & \\ & \ddots & \ddots & \\ & & 1+\frac{ah}{2} & 0 \end{pmatrix}. \tag{7.76}$$

This is the iterative procedure for *direct* implementation of the Neumann boundary condition. The *lagged* version works as follows.

Rewrite the Neumann boundary condition as:

$$u_n^{k+1} = u_{n-1}^k + h\beta \,. \tag{7.77}$$

If the condition at the right boundary were of the Dirichlet type, the discrete equation for node $n-1$ in the split system would be:

$$(-1 - \tfrac{ah}{2})u_{n-2}^k + 2u_{n-1}^{k+1} + (1 - \tfrac{ah}{2})u_n^{k+1} = h^2 f_{n-1} \,. \tag{7.78}$$

The lagged Neumann boundary condition can now be implemented by substituting eq. (7.77) into (7.78):

$$(-1 - \tfrac{ah}{2})u_{n-2}^k + 2u_{n-1}^{k+1} + (1 - \tfrac{ah}{2})u_{n-1}^k = h^2 f_{n-1} - (1 - \tfrac{ah}{2})h\beta \,. \tag{7.79}$$

The Gauss-Seidel method with the lagged Neumann boundary conditions is:

$$M_2 x^{k+1} = N_2 x^k + b \,, \tag{7.80}$$

with

$$M_2 = \begin{pmatrix} 2 & -1+\frac{ah}{2} & & \\ & \ddots & \ddots & \\ & & 2 & -1+\frac{ah}{2} \\ & & & 2 \end{pmatrix}, \quad N_2 = \begin{pmatrix} 0 & & & & \\ 1+\frac{ah}{2} & 0 & & & \\ & \ddots & \ddots & & \\ & & 1+\frac{ah}{2} & 0 & \\ & & & 1+\frac{ah}{2} & 1-\frac{ah}{2} \end{pmatrix}. \tag{7.81}$$

The convergence rates of the two methods can be measured by comparing the spectral radii ρ_1 and ρ_2 of their respective iteration matrices, defined by:

$$\rho_1 = \rho(M_1^{-1}N_1) \,, \quad \text{and } \rho_2 = \rho(M_2^{-1}N_2) \tag{7.82}$$

Table 7.1 lists values of ρ_1 and ρ_2 for different magnitudes of the cell Reynolds number Re_c ($Re_c = ah$) on a grid of size 50. Apparently, the lagged method converges faster when the cell Reynolds number is below 2, the diffusion-dominated case, although differences are small. When the cell Reynolds number is larger than 2, both schemes are unstable, which is expected with central differences. Underrelaxation can again produce stable schemes. Some results with a crude attempt at optimization are presented in Table 7.2. The relaxation parameter is indicated by ω.

The following conclusions can be drawn from this investigation:

Table 7.1: *Spectral radii for Gauss-Seidel iteration*

Re_c	ρ_1	ρ_2
0.02	0.9985	0.9960
0.20	0.9872	0.9861
1.00	0.7473	0.7471
2.00	0	0
4.00	2.988	2.989

Table 7.2: *Spectral radii for underrelaxed Gauss-Seidel iteration*

Re_c	ω	ρ_1	ρ_2
4.00	0.8	2.191	2.191
4.00	0.5	0.9942	0.9943
4.00	0.2	0.8679	0.8680
10.0	0.1	1.491	1.491
10.0	0.08	0.9925	0.9927
10.0	0.075	0.9450	0.9625
10.0	0.07	0.9491	0.9650
10.0	0.06	0.9400	0.9700
10.0	0.05	0.9622	0.9750

- both implementations are stable or unstable at the same time,
- the lagged implementation is faster in the diffusion-dominated range, whereas the direct method is superior when convection is important,
- when optimal underrelaxation is applied, the convergence rate of the direct method is markedly better than that of the lagged version.

Although there are ranges where one method is better, the differences are usually small for nonoptimal relaxation. Numerical experiments with the Navier-Stokes equations show that not much can be gained from implementing the more involved direct method. In fact, in all our test computations the lagged version converged faster. For this reason this implementation is used in the more complex calculations.

7.8 Convergence Criteria

When solving problems iteratively, it is important to have an estimate of how well the solution approximates the true solution, *i.e.*, how well-converged the solution is. In general, we do not know the true solution, so the estimate needs to be based on less exact information. Changes in the solution are not sufficient to determine convergence, as convergence factors close to 1 can yield very small changes between iterates, even when a solution is far from converged.

As we cannot compare the iterate to the true solution, we have to monitor how well it satisfies the (discrete) equation, *i.e.*, the residual should be computed. For the problem $Au^n = b$, discretized at n points, we can define the residual r^n of the approximate solution $\tilde{u}^n$ as: $r^n = b - A\tilde{u}^n$, and the residual norm as $R^n = \|r^n\|$. This definition is appropriate for a fixed-size problem —that is, for fixed n— but fails when we want to make consistent approximations to a continuous problem. Hence, we use the *normalized* p-norm of the discrete residual:

$$\overline{R}_p = \sqrt[p]{\frac{1}{n}\sum_{i=1}^{n} |b_i - (A\tilde{u}^n)_i|^p} \,. \tag{7.83}$$

Next, we want to eliminate the effect of the mesh size on the residual. In finite-volume computations the discrete equations are obtained by integrating the partial differential equation, $Lu = f$, over a mesh cell:

$$\int_{\Omega_{cell}} Lu \, d\Omega_{cell} = \int_{\Omega_{cell}} f \, d\Omega_{cell} \,. \tag{7.84}$$

Typically, Au^n approximates $\int_{\Omega_{cell}} Lu \, d\Omega_{cell}$, and b approximates $\int_{\Omega_{cell}} f \, d\Omega_{cell}$.

Assuming a residual of magnitude r (*i.e.*, $L\tilde{u}^n - f = r$), we find:

$$\overline{R}_p = \sqrt[p]{\frac{1}{n}\sum_{i=1}^{n} \left|\int_{\Omega_{cell}} Lu \, d\Omega_{cell} - \int_{\Omega_{cell}} f \, d\Omega_{cell}\right|^p} = \sqrt[p]{\frac{1}{n}\sum_{i=1}^{n} \left|\int_{\Omega_{cell}} r \, d\Omega_{cell}\right|^p} \,. \tag{7.85}$$

If we furthermore assume uniformity of r and the cell area, we get:

$$\overline{R}_p = \sqrt[p]{\frac{1}{n}|r|^p \sum_{i=1}^{n} \Omega_{cell}^p} = |r|\Omega_{cell} \,. \tag{7.86}$$

Clearly, the discrete residual depends on the mesh size. The dependence can be eliminated by dividing by the cell area, so that the new definition, R_p^*, of the discrete-residual norm reads:

$$R_p^* = \sqrt[p]{\frac{1}{n}\sum_{i=1}^{n}\left|\frac{b_i - (A\bar{u}^n)_i}{\Omega_{cell}}\right|^p} . \qquad (7.87)$$

Note that the Navier-Stokes equations in curvilinear coordinates were obtained by multiplying by the Jacobian of the coordinate transformation. As the (discrete) Jacobian is inversely proportional to the cell area, we do not need to correct the residual as in eq. (7.87); the definition of $\overline{R}_p$, eq. (7.83), suffices.

A remaining difficulty in assessing residuals is the relative scaling of the momentum and continuity equations. The equations are nondimensionalized, so that the velocities and the pressures are of order 1. Hence, we also have: $\mathcal{O}(uv) = \mathcal{O}(u) = \mathcal{O}(v) = \mathcal{O}(u^2) = \mathcal{O}(v^2) = \mathcal{O}(1)$. This means that momentum and mass fluxes are of the same order of magnitude, so the terms in the momentum equation have the same size as those in the continuity equation and no scaling is necessary. This is true in flows in which the *convective* momentum flux dominates. If viscosity is important, so are the *diffusive* momentum fluxes, and we should compare these with the mass flux in the continuity equation. However, the diffusive fluxes have the reciprocal of the Reynolds number in front of them. This means that in a diffusion-dominated flow, the residual of the momentum equation will go down as the Reynolds number goes up for equally bad iterative solutions. This has been observed in the the lid-driven cavity flow (see later in this chapter), where diffusion dominates. As the Reynolds number increased, the number of iterations to reach a certain residual threshold decreased, which is counterintuitive. The reason for the apparent speed-up is the lower initial and overall residual governed by the diffusive terms; the actual convergence *rates* get worse with increasing Reynolds number.

Unfortunately, there is no universal remedy for this problem. Asking for a reduction of the relative residual (with respect to the first iterate) is not totally fair either, as this would imply very tight convergence of the continuity equation. If residuals are to be the only source of error information, one still has to use physical insight to interpret what a small residual means.

7.9 Solution Strategies

Now that we have discretized the equations and have defined convergence criteria, we need to compute solutions. Among the most popular iterative methods for the computation of incompressible, viscous flow are SIMPLE (Semi-IMplicit Pressure-Linked Equations) and SIMPLER (SIMPLE-Revised), described by Patankar in [PAT80], [PAT81]. These methods rely on conversion of the strongly coupled momentum and continuity equations into weakly coupled momentum and pressure (correction) equations, which are used to advance the velocity and the pressure more or less independently. Each subproblem requires the inversion of a diagonal or a diagonally dominant matrix.

In this section we show that the decoupling can be interpreted as an approximation to a simple matrix-splitting procedure. The latter has been formulated separately by Maliska and Raithby [MAL83], and later independently by Perng [PER89A]. Convergence of this simplified scheme for the Stokes equations is proved using a theorem from optimization theory. When applied to the Navier-Stokes equations, the simplified scheme is more robust than SIMPLER, especially as the Reynolds number increases and the coordinate system becomes less regular.

7.9.1 Historic Formulation of SIMPLE and SIMPLER

We will first give the historical description of SIMPLE(R) so as to appreciate better the matrix formulation that follows. SIMPLE(R) was formulated originally for a standard staggered grid, and seems inseparable from it. But, as will be demonstrated, SIMPLE(R) is a *philosophy* for solving the Navier-Stokes equations which can be applied on *any* grid. We shall describe the method for our modified staggered grid.

The indexing scheme Patankar uses [PAT81], based on the relative indices 'east', 'west', 'north', and 'south', works *only for* Cartesian coordinates on a standard staggered grid. In all other cases it becomes too complex. On the other hand, with numerical indices the expressions become so involved that all insight is lost. For that reason we introduce a means of writing difference equations in a generic way which does not involve any indices at all. It is a merger of difference stencils and algebra, which one might call *stencil calculus*.

The idea is that all the difference formulas used are symmetric about some geometric center. Let this center correspond to the center (on paper) of the generic stencil. For example, the equation

$$(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) = c(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) \tag{7.88}$$

could be represented as:

$$\begin{array}{c}\bullet\\ \bullet\\ \bullet\end{array} u = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} u \;, \quad \text{or as:} \quad \begin{array}{ccc} & \bullet & \\ \bullet & \bullet & \bullet \\ & \bullet & \end{array} u = 0 \,. \tag{7.89}$$

Here a bullet ($\bullet$) denotes a nonzero coefficient, whereas an open circle ($\circ$) means that the coefficient is essentially zero. When a difference stencil has odd numbers of nodes in all directions, the center node corresponds to the center variable on which the stencil operates. When a difference stencil has an even number of nodes in some direction, indices are relative with respect to a line in between two grid nodes, so the symbolic equation is again unambiguous. A single bullet represents a scalar multiple of the center variable.

Using this formalism, we write the discrete u- and v-momentum equations (7.49), (7.50), derived earlier, as:

$$\bullet\, u = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} u + \begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} p + sc_u \,, \tag{7.90}$$

$$\bullet\, v = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} v + \begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} p + sc_v \,, \tag{7.91}$$

whereas the discrete continuity equation (7.40) becomes:

$$\begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} u + \begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} v = 0 \,. \tag{7.92}$$

The reason for splitting the five-point star representing the convection/diffusion operator into a center node on the left-hand side and neighboring nodes on the right-hand side will become clear soon.

SIMPLE starts as follows. Using a guessed pressure field p^*, solve the momentum equations (7.90), (7.91) for the velocities; call these u^* and v^*. Subsequently, define corrections u', v' and p' by

$$u = u^* + u' \,, \tag{7.93}$$

$$v = v^* + v' \,, \tag{7.94}$$

$$p = p^* + p' \,. \tag{7.95}$$

Substitute eqs. (7.93), (7.94) and (7.95) into the momentum equations and neglect the terms $\begin{smallmatrix} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{smallmatrix} u'$ and $\begin{smallmatrix} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{smallmatrix} v'$ (*i.e.*, the neighbors of u' and v'). The result of these manipulations is:

$$u = u^* + \begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix} p' \,, \tag{7.96}$$

$$v = v^* + \begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix} p' \,. \tag{7.97}$$

The velocities can be computed from eqs. (7.96) and (7.97) once the pressure corrections p' are known. An equation for the pressure corrections can be derived by demanding that the new velocity field satisfy the continuity equation. Hence, substitute eqs. (7.96), (7.97) into this discrete continuity equation (7.92). This requires the definition of a product in our stencil calculus; to compute the product $S_1 S_2$, simply replace every node in S_1 by the complete stencil S_2, centered at the S_1-node, allowing overlap between stencils. For example:

$$\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix} \begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix} = \begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix} \,, \qquad \text{and :} \quad \begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix} \begin{pmatrix} \bullet & \bullet & \bullet \end{pmatrix} = \begin{smallmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{smallmatrix} \,. \tag{7.98}$$

Now we can write the discrete continuity equation as:

$$\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix} \left(u^* + \begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix} p' \right) + \begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix} \left(v^* + \begin{smallmatrix} \bullet & \bullet \\ \bullet & \bullet \end{smallmatrix} p' \right) = 0 \,, \tag{7.99}$$

which can be simplified to the final pressure-correction equation:

$$\begin{smallmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{smallmatrix} p' + ms' = 0 \,. \tag{7.100}$$

The term ms' is a mass source arising from the starred velocity field which does not generally satisfy continuity, *i.e.*,

$$ms' = \begin{smallmatrix}\bullet & \bullet \\ \bullet & \bullet\end{smallmatrix} u^* + \begin{smallmatrix}\bullet & \bullet \\ \bullet & \bullet\end{smallmatrix} v^* \,. \tag{7.101}$$

Summarizing the SIMPLE procedure, we get:

1) Guess a pressure field p^*,
2) Solve the momentum equations (7.90), (7.91) to get u^* and v^*,
3) Compute the mass source ms' from eq. (7.101) and solve the pressure-correction equation (7.100),
4) Correct the pressure and velocity fields using eqs. (7.95), (7.96), (7.97),
5) Regarding the corrected pressure field p as a new guess p^*, return to step 2 and repeat until convergence.

Before describing SIMPLER (an improved version of SIMPLE), some remarks are in order:

In the SIMPLE procedure, three matrix problems are solved per iteration in a two-dimensional calculation, and four in a three-dimensional calculation. This includes one Poisson-type problem for the pressure correction, and two or three inversions of the convection/diffusion operator. These inversions are performed iteratively as well, so that a nesting of inner and outer iterations occurs.

The pressure-correction equation was arrived at by manipulating the momentum equations and invoking the continuity equation; only linear algebraic manipulations were involved, so no spurious solutions were introduced.

An important drawback of SIMPLE is that an initial pressure field needs to be supplied. Typically, it is much easier to guess a reasonable velocity field than a reasonable pressure field.

SIMPLE does not always converge; especially if no underrelaxation is applied, divergence will usually result. Patankar recommends underrelaxation factors between 0.5 and 0.8.

In order to improve convergence and to overcome the difficulty with the initial pressure field, Patankar formulated a variation of the SIMPLE technique, which came to be known as SIMPLER (SIMPLE-Revised). He noticed that the omission of the term $\begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} u'$ in the velocity-correction equations (7.96) and (7.97) was acceptable with respect to the velocity corrections, but led to poor estimates for the pressure correction p'. Hence, he proposed a new equation for the true pressure, and used the old one (eq. (7.100)) only to obtain the corrected velocities. This new equation is derived by defining the pseudo-velocities $\hat{u}$ and $\hat{v}$:

$$\hat{u} = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} u + \overline{sc_u}\,, \qquad \text{and} \quad \hat{v} = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} v + \overline{sc_v}\,, \tag{7.102}$$

through rearranging the discrete momentum equations (7.90), (7.91) without the pressure gradient. Employing the pseudo-velocities, the momentum equations can be written:

$$u = \hat{u} + \begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} p\,, \tag{7.103}$$

$$v = \hat{v} + \begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} p\,. \tag{7.104}$$

The pseudo-velocities can be computed straightforwardly from the last approximation to the velocity field. If the rewritten momentum equations are substituted into the continuity equation (7.92), the new pressure equation follows:

$$\begin{array}{ccc} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{array} p + ms = 0\,, \tag{7.105}$$

with the new mass source given by

$$ms = \begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} \hat{u} + \begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array} \hat{v}\,. \tag{7.106}$$

Combining this with the previously described steps for SIMPLE, we arrive at the SIMPLER algorithm:

1) Guess a velocity field u, v.

2) Compute the pseudo-velocities $\hat{u}$ and $\hat{v}$ using eq. (7.102),

3) Compute the mass source ms from eq. (7.106) and solve the pressure equation (7.105),

4) Regarding this pressure field as p^*, solve the momentum equations (7.90), (7.91) to obtain u^* and v^*,

5) Compute the mass source ms' from eq. (7.101) and solve the pressure-correction equation (7.100),

6) Using the p' field, correct the starred velocities according to eqs. (7.96) and (7.97), but do not apply the pressure corrections to the pressure field,

7) Regarding the latest velocity field as a new guess, return to step 2 and repeat until convergence.

Note that now during every outer iteration four matrix problems must be solved (five for three-dimensional problems), *i.e.*, two Poisson-like equations for pressure and pressure correction, and two or three momentum equations. Storage must be allocated for u, $\hat{u}$, u^*, v, $\hat{v}$, v^*, p, and p' respectively.

SIMPLER generally converges faster than SIMPLE, and less underrelaxation is required. Typically, underrelaxing the velocities by 75% and leaving the pressure alone gives good results, according to Patankar. Moreover, the iterative procedure no longer depends on an initial guess for the pressure field. Instead, an initial velocity field must be supplied, which is more convenient.

We now make a slight digression to give another, more graphical demonstration of the inappropriateness of the standard staggered variable arrangement for computations on curved and/or rotated grids. Here we can conveniently exploit the stencil calculus. Besides the already mentioned bullet (•) and open circle (∘), we will also introduce the symbol ©, which is used when a certain coefficient in a difference equation is generally *nonzero* on a non-Cartesian grid. Using the standard variable staggering, we find:

U-momentum equation:

$$\bullet\, u = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} u + \begin{array}{cc} \odot & \odot \\ \odot & \odot \\ \odot & \odot \end{array} p + sc_u\,, \tag{7.107}$$

V-momentum equation:

$$\bullet\, v = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} v + \begin{array}{ccc} \odot & \odot & \odot \\ \odot & \odot & \odot \end{array} p + sc_v\,, \tag{7.108}$$

Continuity equation:

$$\begin{array}{cc} \odot & \odot \\ \odot & \odot \\ \odot & \odot \end{array} u + \begin{array}{ccc} \odot & \odot & \odot \\ \odot & \odot & \odot \end{array} v = 0\,. \tag{7.109}$$

Consequently, the pressure-correction equation becomes:

$$\begin{array}{ccccc} & \odot & \odot & \odot & \\ \odot & \odot & \odot & \odot & \odot \\ \odot & \odot & \odot & \odot & \odot \\ \odot & \odot & \odot & \odot & \odot \\ & \odot & \odot & \odot & \end{array} p' + ms' = 0\,. \tag{7.110}$$

For a Cartesian grid, these reduce to the canonical forms:

$$\bullet\, u = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} u + \bullet\ \bullet\, p + sc_u\,, \tag{7.111}$$

$$\bullet\, v = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} v + \begin{array}{c} \bullet \\ \bullet \end{array} p + sc_v\,, \tag{7.112}$$

$$\bullet\ \bullet\, u + \begin{array}{c} \bullet \\ \bullet \end{array} v = 0\,, \tag{7.113}$$

and

$$\begin{array}{ccc} & \bullet & \\ \bullet & \bullet & \bullet \\ & \bullet & \end{array} p' + ms' = 0\,. \tag{7.114}$$

However, if this Cartesian grid is rotated through an angle of 90^0, then the expressions become:

$$\bullet\, u = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} u + \begin{array}{cc} \bullet & \bullet \\ \circ & \circ \\ \bullet & \bullet \end{array} p + sc_u\,, \tag{7.115}$$

$$\bullet\, v = \begin{array}{ccc} & \bullet & \\ \bullet & \circ & \bullet \\ & \bullet & \end{array} v + \begin{array}{ccc} \bullet & \circ & \bullet \\ \bullet & \circ & \bullet \end{array} p + sc_v \,, \tag{7.116}$$

$$\begin{array}{cc} \bullet & \bullet \\ \circ & \circ \\ \bullet & \bullet \end{array} u + \begin{array}{ccc} \bullet & \circ & \bullet \\ \bullet & \circ & \bullet \end{array} v = 0 \,, \tag{7.117}$$

and

$$\begin{array}{ccccc} & \bullet & \bullet & \bullet & \\ \bullet & \circ & \bullet & \circ & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \circ & \bullet & \circ & \bullet \\ & \bullet & \bullet & \bullet & \end{array} p' + ms' = 0 \,. \tag{7.118}$$

Clearly, these stencils are much more complicated than those for the modified staggered grid.

7.9.2 Matrix Formulation of SIMPLE and SIMPLER

The SIMPLE and SIMPLER methods can be illuminated by formulating them in terms of matrices. First, write the discretized, linearized Navier-Stokes equations (2 momentum equations + the continuity equation) as:

$$\begin{pmatrix} D^u + R^u & & -\triangledown_\xi \\ & D^v + R^v & -\triangledown_\eta \\ \nabla_\xi & \nabla_\eta & \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix} = \begin{pmatrix} r_u \\ r_v \\ r_c \end{pmatrix}, \tag{7.119}$$

where D and R denote the diagonal and off-diagonal parts of the discretized convection/diffusion operator, respectively. The discrete gradient operator is $\left(\triangledown_\xi^{\mathrm{T}}, \triangledown_\eta^{\mathrm{T}}\right)^{\mathrm{T}}$, and the discrete divergence operator is: $(\nabla_\xi, \nabla_\eta)$. Notice the slight typographical difference between the gradient (triangle $\triangledown$) and divergence (nabla ∇) operators, indicating they are not exactly the same for curved (or staggered) grids. The symbols u, v, and p represent the discrete velocity and pressure vectors. The right-hand-side vector of eq. (7.119) contains source terms, and, most importantly, the boundary conditions.

SIMPLE proceeds as follows: Assume an initial guess p^* for the pressure field, and from that, compute a starred velocity field by solving:

$$(D^u + R^u)u^* = \triangledown_\xi p^* + r_u \,. \tag{7.120}$$

$$(D^v + R^v)v^* = \nabla_\eta p^* + r_v \,. \tag{7.121}$$

This looks like part of the block-Gauss-Seidel splitting:

$$\begin{pmatrix} D^u + R^u & & 0 \\ & D^v + R^v & 0 \\ \nabla_\xi & \nabla_\eta & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k+1} = \begin{pmatrix} r_u \\ r_v \\ r_c \end{pmatrix}$$

$$+ \begin{pmatrix} 0 & & \nabla_\xi \\ & 0 & \nabla_\eta \\ & & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k} \tag{7.122}$$

which is undefined because of the zero third column of the left-hand-side matrix.

After computation of the starred velocity field, a new iterate u^{k+1}, v^{k+1}, p^{k+1} (level $k+1$) is formed as the sum of the starred field plus a correction, indicated by a prime:

$$\begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k+1} = \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{*} + \left\{ \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k+1} - \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{*} \right\} = \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{*} + \begin{pmatrix} u \\ v \\ p \end{pmatrix}' \tag{7.123}$$

At convergence the corrections are zero, so they can be inserted anywhere in the iterative process without affecting the final solution.

The corrected quantities must satisfy the original equation (7.119) at convergence. Because this is too hard to solve directly, a disturbance that vanishes at convergence is added to the right-hand side to make solution easier.

The particular choice of SIMPLE is:

$$\begin{pmatrix} D^u + R^u & & -\nabla_\xi \\ & D^v + R^v & -\nabla_\eta \\ \nabla_\xi & \nabla_\eta & \end{pmatrix} \begin{pmatrix} u^* + u' \\ v^* + v' \\ p^* + p' \end{pmatrix} = \begin{pmatrix} r_u \\ r_v \\ r_c \end{pmatrix}$$

$$+ \begin{pmatrix} R^u & & \\ & R^v & \\ & & 0 \end{pmatrix} \begin{pmatrix} u' \\ v' \\ p' \end{pmatrix} \tag{7.124}$$

Although this equation looks rather contrived, it is easy to show, using equations (7.120) and (7.121), that it is equivalent to:

$$\begin{pmatrix} D^u & & -\nabla_\xi \\ & D^v & -\nabla_\eta \\ \nabla_\xi & \nabla_\eta & \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}' = \begin{pmatrix} 0 \\ 0 \\ -\nabla_\xi u^* - \nabla_\eta v^* + r_c \end{pmatrix} . \qquad (7.125)$$

Applying block-Gaussian elimination to this system (which is simple due to the ease with which the diagonal matrices D can be inverted) yields:

$$\begin{pmatrix} D^u & & -\nabla_\xi \\ & D^v & -\nabla_\eta \\ & & \tilde{\Delta}_r \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}' = \begin{pmatrix} 0 \\ 0 \\ -\nabla_\xi u^* - \nabla_\eta v^* + r_c \end{pmatrix} , \qquad (7.126)$$

with

$$\tilde{\Delta}_r = \nabla_\xi (D^u)^{-1} \nabla_\xi + \nabla_\eta (D^v)^{-1} \nabla_\eta \, . \qquad (7.127)$$

The last row of eq. (7.126) represents the pressure-correction equation (7.100), the right-hand-side term being the mass source ms' (7.101); $\tilde{\Delta}_r$ is a discrete Laplace-like operator. Back-substitution of the pressure correction into the first two rows of eq. (7.126) yields equation (7.96) for the corrected velocities, which completes the SIMPLE iteration cycle.

At this stage it is useful to recast eq. (7.125) into yet another shape, using the defining equation (7.123):

$$\begin{pmatrix} D^u & & -\nabla_\xi \\ & D^v & -\nabla_\eta \\ \nabla_\xi & \nabla_\eta & \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k+1} = \begin{pmatrix} r_u \\ r_v \\ r_c \end{pmatrix} + \begin{pmatrix} -R^u & & \\ & -R^v & \\ & & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}^* . \qquad (7.128)$$

This would be a Jacobi-type splitting, applied to the upper right block-(2×2) submatrix, if the starred quantities were an initial guess. However, they are not independent quantities, as they are related through eqs. (7.120) and (7.121).

One may interpret SIMPLE as an attempt to solve a Schur-complement problem. be it that the block-matrix inversions are done in the reverse order, *i.e.*, first the momentum equations are solved and then the pressure equation.

The above analysis of SIMPLE can help in understanding SIMPLER; if on the right-hand side of eq. (7.128) the starred quantities are replaced by a guess –or previous iterate– *i.e.*,

$$\begin{pmatrix} u \\ v \\ p \end{pmatrix}^{*} \longleftarrow \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k} , \tag{7.129}$$

then the first two steps of SIMPLER consist of the evaluation of this right-hand side ($\hat{u}$ and $\hat{v}$). Notice that this does not involve a starting guess for the pressure.

The third step comprises the application of block-Gaussian elimination to the resulting system and the computation of the new pressure field from the last row of the triangularized system:

$$\tilde{\Delta}_r p^{k+1} = \nabla_\xi (D^u)^{-1}\{R^u u^k - r_u\} + \nabla_\eta (D^v)^{-1}\{R^v v^k - r_v\} + r_c \, . \tag{7.130}$$

Now it would be natural to complete the Schur-complement computation by obtaining u^{k+1} and v^{k+1} through back-substitution of p^{k+1} into eq. (7.128), which would merely require inversions of the diagonal matrices D^u and D^v. However, SIMPLER regards p^{k+1} as a better guess for the pressure field and returns to eqs. (7.120) and (7.121) to invert the whole convection/diffusion difference operator in step four.

To improve on the results obtained from these inversions, that is, to obtain a better approximation to the solution of eq. (7.128), another cycle of velocity corrections is carried out in steps five and six, similar to the cycle decribed under SIMPLE. The pressure field computed in the first step of the generalized Jacobi splitting is not updated.

7.9.3 Simplified Solution Method

From the above analysis we conclude that both SIMPLE and SIMPLER are variations of the one-step splitting procedure:

$$\begin{pmatrix} D^u & & -\nabla_\xi \\ & D^v & -\nabla_\eta \\ -\nabla_\xi & -\nabla_\eta & \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k+1} = \begin{pmatrix} r_u \\ r_v \\ r_c \end{pmatrix} + \begin{pmatrix} -R^u & & \\ & -R^v & \\ & & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ p \end{pmatrix}^{k} , \tag{7.131}$$

which Maliska and Raithy [MAL83] called PRIME (PRessure Implicit, Momentum Explicit).

In algorithm form:

1) Compute the pseudo-velocities:

$$\hat{u} = (D^u)^{-1}\left(r_u - R^u u^k\right) \text{ and } \hat{v} = (D^v)^{-1}\left(r_v - R^v v^k\right) ,$$

2) Compute the mass source $ms = -\nabla_\xi (D^u)^{-1}\hat{u} - \nabla_\eta (D^v)^{-1}\hat{v}$,

3) Solve the pressure equation $\tilde{\Delta}_r p^{k+1} = ms + r_c$ (note: not usually iterated to convergence),

4) Compute the new velocities:

$$u^{k+1} = \hat{u} - (D^u)^{-1}\nabla_\xi p^{k+1} \text{ and } v^{k+1} = \hat{v} - (D^v)^{-1}\nabla_\eta p^{k+1} .$$

PRIME can be used to analyze convergence properties. Moreover, it can serve as a basis for acceleration methods like multigrid [PER89A], which are not so easily implemented for SIMPLER. In the current implementation, one iteration of the simplified scheme takes about 60% of the operations required by SIMPLER, and it needs significantly less storage. On Cartesian grids and for low Reynolds numbers. SIMPLER is substantially faster than PRIME. However, as the Reynolds number increases, the simplified method gains ground. An additional advantage is that usually no underrelaxation is required, whereas the stability and convergence rate of SIMPLER depend vitally on the choice of relaxation parameters. For non-Cartesian grids, SIMPLER quickly deteriorates, whereas the performance of PRIME remains unaffected.

7.9.4 Convergence Analysis

In this section we examine the convergence properties of SIMPLE, SIMPLER, and the simplified scheme for some model problems, starting with the one-dimensional

Navier-Stokes equations on a nonstaggered grid. Dropping redundant subscripts, the discretized, linearized Navier-Stokes equations in one dimension can be written:

$$\begin{pmatrix} D+R & -\nabla \\ \nabla & \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} r_u \\ r_c \end{pmatrix} . \qquad (7.132)$$

Assuming that the number of velocity nodes is even, all the matrices in the following analysis are nonsingular and square, so we can freely use their inverses. The simplified method derived in the previous section can be written as:

$$\begin{pmatrix} D & -\nabla \\ \nabla & \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{k+1} = \begin{pmatrix} -R & 0 \\ 0 & \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{k} + \begin{pmatrix} r_u \\ r_c \end{pmatrix} . \qquad (7.133)$$

The error amplification matrix Z is easily computed:

$$Z = \begin{pmatrix} 0 & 0 \\ \nabla^{-1}R & 0 \end{pmatrix} . \qquad (7.134)$$

Obviously, $\rho(Z) = 0$, so that an exact solution is obtained after only two iterations (provided the pressure equation is solved exactly). This should come as no surprise, since the continuity equation in one space dimension can be solved independently.

However, SIMPLE is not able to duplicate this result. Analyzing its convergence properties is slightly more complicated. The method can be broken into two steps involving an intermediate variable (indicated by the asterisk '$*$'):

Step 1:

$$\begin{pmatrix} D+R & \\ 0 & I \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{*} = \begin{pmatrix} 0 & \nabla \\ 0 & I \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{k} + \begin{pmatrix} r_u \\ 0 \end{pmatrix} , \qquad (7.135)$$

Step 2:

$$\begin{pmatrix} D & -\nabla \\ \nabla & \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{k+1} = \begin{pmatrix} -R & 0 \\ 0 & \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{*} + \begin{pmatrix} r_u \\ r_c \end{pmatrix} . \qquad (7.136)$$

It is again easy to compute the iteration matrix Z_S:

$$Z_S = \begin{pmatrix} 0 & 0 \\ 0 & \nabla^{-1}R(D+R)^{-1}\nabla \end{pmatrix} . \qquad (7.137)$$

The spectral radius is now not equal to zero. Instead, we find:

$$\rho(Z_S) = \rho\left(R(D+R)^{-1}\right) . \tag{7.138}$$

Even for the Stokes equation, we get $\rho(Z_S) > 1$. Apparently, SIMPLE is too simple, and underrelaxation is needed to stabilize the method.

The analysis of SIMPLER is even more complex. Now two intermediate variables need to be introduced (indicated by '$*$' or '$**$'):

Step 1:

$$\begin{pmatrix} I & \\ & \nabla D^{-1}\nabla \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{*} = \begin{pmatrix} I & 0 \\ \nabla D^{-1} R & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{k} + \begin{pmatrix} 0 \\ r_c - \nabla D^{-1} r_u \end{pmatrix} \tag{7.139}$$

Step 2:

$$\begin{pmatrix} D+R & \\ & I \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{**} = \begin{pmatrix} 0 & \nabla \\ 0 & I \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{*} + \begin{pmatrix} r_u \\ 0 \end{pmatrix} \tag{7.140}$$

Step 3:

$$\begin{pmatrix} u \\ p \end{pmatrix}^{k+1} = \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}^{**} + \begin{pmatrix} \nabla^{-1} r_c \\ 0 \end{pmatrix} \tag{7.141}$$

From this sequence the iteration matrix Z_{SR} can be determined:

$$Z_{SR} = \begin{pmatrix} 0 & 0 \\ \nabla^{-1} R & 0 \end{pmatrix} . \tag{7.142}$$

This matrix is the same as the one for the simplified method, and convergence is again obtained in two iterations.

Some of the above inverses may not exist, for example because an odd number of velocity nodes was chosen, or because some of the matrices are not square due to the application of a staggered grid. The results of the analysis are still valid in the sense that (generalized) eigenvalue equations can be formulated by multiplying by appropriate matrices to eliminate all inverses.

The situation is not nearly as transparent in two dimensions. The same steps can be followed, but the results are not very revealing. For example:

$$\rho(Z) =$$

$$\rho\begin{pmatrix}(D^u)^{-1}(I - \nabla_\xi \tilde{\Delta}_r^{-1} \nabla_\xi (D^u)^{-1})R^u & -(D^u)^{-1}\nabla_\xi \tilde{\Delta}_r^{-1}\nabla_\eta (D^v)^{-1}R^v \\ -(D^v)^{-1}\nabla_\eta \tilde{\Delta}_r^{-1}\nabla_\xi (D^u)^{-1}R^u & (D^v)^{-1}(I - \nabla_\eta \tilde{\Delta}_r^{-1}\nabla_\eta (D^v)^{-1})R^v\end{pmatrix}, \tag{7.143}$$

and

$$\rho(Z_S) = \rho(\tilde{\Delta}_r^{-1}\{\nabla_\xi (D^u)^{-1}R^u(D^u+R^u)^{-1}\nabla_\xi + \nabla_\eta (D^v)^{-1}R^v(D^v+R^v)^{-1}\nabla_\eta\}) . \tag{7.144}$$

These expressions cannot be simplified much in general. The convergence factor for SIMPLER is considerably uglier than these two and will not be given here. Moreover, the convergence factors for SIMPLER and the simplified method can no longer easily be related. There is a convergence result, however, for the Stokes equations in multiple dimensions on a nonstaggered grid, given in [DYN83]. Dyn and Ferguson prove the following theorem:

Convergence of Iterative Schemes

The iterative sequence

$$\begin{pmatrix} B & E \\ E^T & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^{k+1} = \begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^{k} + \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \tag{7.145}$$

converges to the solution of the equation

$$\begin{pmatrix} A & E \\ E^T & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \tag{7.146}$$

if the (sub-)matrix splitting $A = B - C$ is used and the following conditions hold:

1. *A is a real, symmetric, nonnegative-definite matrix,*

2. *E is a real matrix with full column rank,*

3. *A and E^T have no nontrivial null vectors in common,*

4. *B is a real, nonsingular matrix,*

5. *$A + C + C^T$ is a positive-definite matrix.*

This can be applied to the simplified method for the Stokes equations on a nonstaggered grid as follows.

The matrix A is the discrete Laplacian, E is the discrete gradient operator (whose transpose is the discrete divergence operator), and the splitting is defined by peeling off the off-diagonal part of A, which means that B is a scalar multiple of the identity matrix. From this it trivially follows that conditions 1, 3, and 4 are fulfilled if the whole problem is multiplied by -1. Condition 2 can be shown to hold by using a symmetry argument, extending the one-dimensional result (the one-dimensional discrete gradient operator on a nonstaggered grid with an even number of unknowns is a square, nonsingular matrix and hence has full column rank). The only nontrivial condition is 5. However, $A + C + C^T$ is irreducibly diagonally dominant, symmetric, and has positive diagonal elements. This means that it is positive-definite, and the iterative sequence defined by the simplified splitting method converges.

This is encouraging, but not sufficient; if we have convection in addition to diffusion, and we use a staggered grid, only conditions 3, 4, and 5 are true. Calculations will have to prove the applicability of the simplified method.

7.9.5 Computational Results

In order to compare the efficacy of the simplified method with SIMPLER's, two test problems are considered:

- The square lid-driven cavity (see Fig. 7.16)
- The polar lid-driven cavity (see Fig. 7.17)

Both flow geometries consist of enclosures, three walls of which are stationary, whereas the fourth moves at constant linear (angular) speed. The discretizations are second order accurate. Stability is increased by using a central-difference correction. To avoid specifying the pressure on the boundary the modified staggered grid (40 by 40 cells) is used. Inner iterations are performed using a line-SOR iteration procedure. Increased accuracy is obtained by stretching the grids, so that points are clustered closer to the walls, where boundary layers need to be resolved. Both uniform-grid and stretched-grid calculations (see Fig. 7.18) were done. Some computed velocity fields are shown in Fig. 7.19. The computations are carried out on a Cydra 5 data-flow machine with a special-purpose numerical processor. A solution is called converged if the 2-norm of the total residual vector of the nondimensionalized equations falls below 10^{-5}. Results of computations for the lid-driven cavity are summarized in Table 7.3.

Table 7.3: *Iteration results for square cavity*

		Uniform Grid				Stretched Grid			
		simplified		SIMPLER		simplified		SIMPLER	
Re	ω	iters	cpu (s)	iters	cpu (s)	iters	cpu (s)	iters	cpu (s)
60	1.0	263	103.5	67	48.5	221	87.2	—	—
	0.9			83	59.8			—	—
	0.8			92	66.1			—	—
	0.7			102	73.4			—	—
	0.6			114	81.7			84	60.6
	0.5							95	68.5
350	1.0	166	66.1	—	—	169	66.9	—	—
	0.9			—	—			—	—
	0.8			60	43.6			—	—
	0.7			64	46.4			—	—
	0.6			72	52.0			69	50.0
	0.5			83	59.8			79	57.2
500	1.0	159	63.1	—	—	167	66.2	—	—
	0.9			—	—			—	—
	0.8			—	—			—	—
	0.7			56	40.7			—	—
	0.6			68	49.2			72	52.1
	0.5			79	57.0			82	59.1

In this table, Re stands for Reynolds number, ω stands for relaxation factor for the components of velocity, 'iter' means number of iterations, and 'cpu' means cpu-time spent (in seconds). Because the convergence of the simplified scheme always slows down as it is underrelaxed, only cases without relaxation are listed. When a bar '—' appears in a column, the method was unstable in that case.

The surprising phenomenon that the *number* of iterations goes down as the Reynolds number is increased is due to the fact that in the residual computations the diffusive component is multiplied by the *reciprocal* of the Reynolds number (see section on convergence criteria).

It should be noted, however, that convergence *rates* worsen as the Reynolds number goes up. This is due mainly to the fact that the convection/diffusion operator $D + R$ in the momentum equations becomes less and less diagonally dominant. In general, convergence of both SIMPLER and the simplified scheme is oscillatory.

It was observed that performing more than one inner iteration per outer iteration did not yield any gain. For the simplified scheme the number of outer iterations stayed the same, which means that cpu-time was wasted by tighter inner-iterative convergence. The number of outer iterations for SIMPLER decreased as more inner iterations on the pressure/pressure correction were performed, but the total cpu-time did not. Underrelaxation was applied only to the velocity; relaxing the pressure deteriorates performance.

For the rectangular cavity (orthogonal grids) SIMPLER always converges faster than the simplified method, *if* it is stable. Whereas the simplified scheme needs no underrelaxation for all the cases tested, SIMPLER's stability depends crucially on the underrelaxation parameter. As the Reynolds number goes up, more underrelaxation is needed and the differences in convergence rates between SIMPLER and the simplified method become smaller. Grid stretching also affects the stability of SIMPLER, and heavier underrelaxation is needed to control it.

In order to determine the convergence properties of SIMPLER and the simplified scheme for situations in which the discrete operators for the pressure gradient and the velocity divergence are not transposes of each other, the polar cavity flow was computed. Results for this case are summarized in Table 7.4. The situation is now quite different. At low Reynolds numbers for the unstretched grid, SIMPLER still beats the simplified method, but as the Reynolds number goes up or as the grid is stretched, the performance of SIMPLER deteriorates rapidly and heavy underrelaxation is needed, whereas the simplified method has no problem. Although very careful tuning of the relaxation parameter can speed up SIMPLER significantly,

Table 7.4: *Iteration results for polar cavity*

		Uniform Grid				Stretched Grid			
		simplified		SIMPLER		simplified		SIMPLER	
Re	ω	iters	cpu (s)	iters	cpu (s)	iters	cpu (s)	iters	cpu (s)
60	1.0	306	123.6	—	—	258	102.0	—	—
	0.9			—	—			—	—
	0.8			97	70.8			—	—
	0.7			108	78.7			—	—
	⋮							⋮	⋮
	0.2							214	152.7
	0.1							423	301.2
350	1.0	192	78.2	—	—	170	67.4	—	—
	⋮			⋮	⋮			⋮	⋮
	0.6			95	69.0			—	—
	0.5			109	79.4			—	—
	⋮							⋮	⋮
	0.2							69	50.0
	0.1							79	57.2
500	1.0	189	76.6	—	—	171	67.9	—	—
	⋮			⋮	⋮			⋮	⋮
	0.5			109	79.3			—	—
	⋮							⋮	⋮
	0.2							158	113.2
	0.1							289	206.2

small errors lead to drastic decay of convergence rates, which makes this strategy impractical.

A word on nomenclature; many popular methods contend to solve part of the discrete equations implicitly, and part explicitly. This is potentially confusing, since all the schemes examined (SIMPLE, SIMPLER, and PRIME) formulate the momentum and continuity equations implicitly, *i.e.*, spatial derivatives in the unsteady equations are evaluated at the new time level. When steady solutions are sought using relaxation procedures, there is no new time level and the word 'explicit' loses its meaning. All methods are equally implicit; they only differ in the way the matrix system is split.

We now summarize the results of the last sections. The popular methods SIMPLE and SIMPLER for incompressible flow calculations have been cast in matrix form, which makes them easier to understand and analyze. Both methods rely on a splitting of the discrete convection/diffusion operator into a diagonal and an off-diagonal part, and on substitution of the thus split system into the discrete continuity equation, resulting in an equation for the pressure. A simplified scheme (PRIME) that employs the same splitting and solves the whole system using the Schur-complement approach has been examined. A convergence proof for this scheme was given for the Stokes equations. Numerical experiments with the square and polar lid-driven cavity flow show this scheme to be more stable than SIMPLER for convection dominated problems on curved, nonuniform grids.

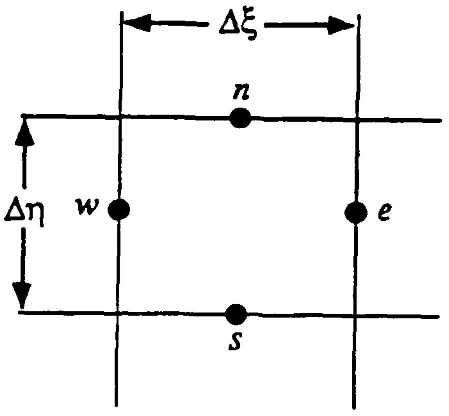


Figure 7.1: Generic control volume with relative cell-face indices

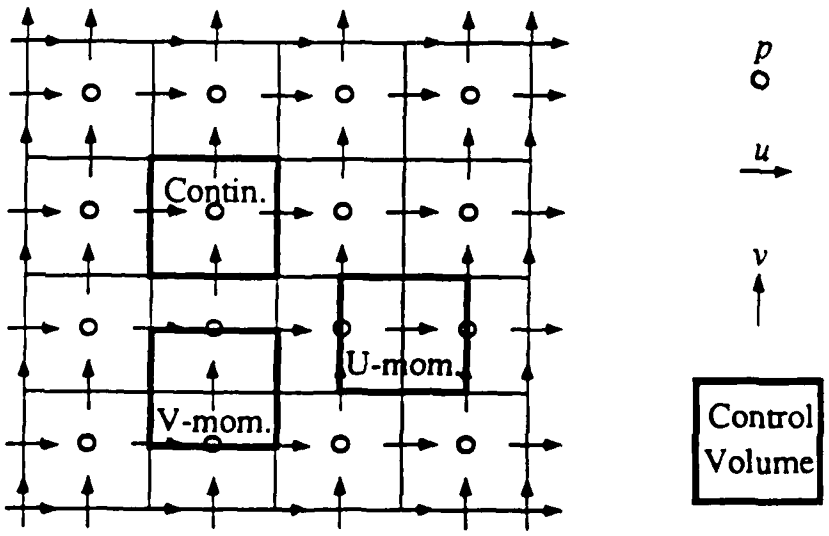


Figure 7.2: Standard staggered variable arrangement

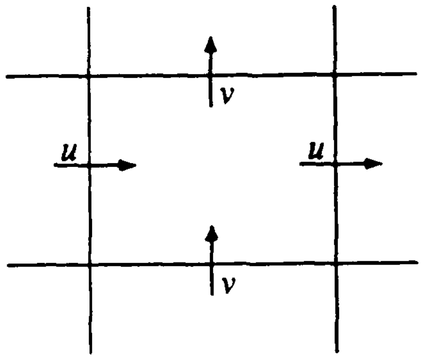


Figure 7.3: Standard staggered Cartesian grid; stencil for continuity equation

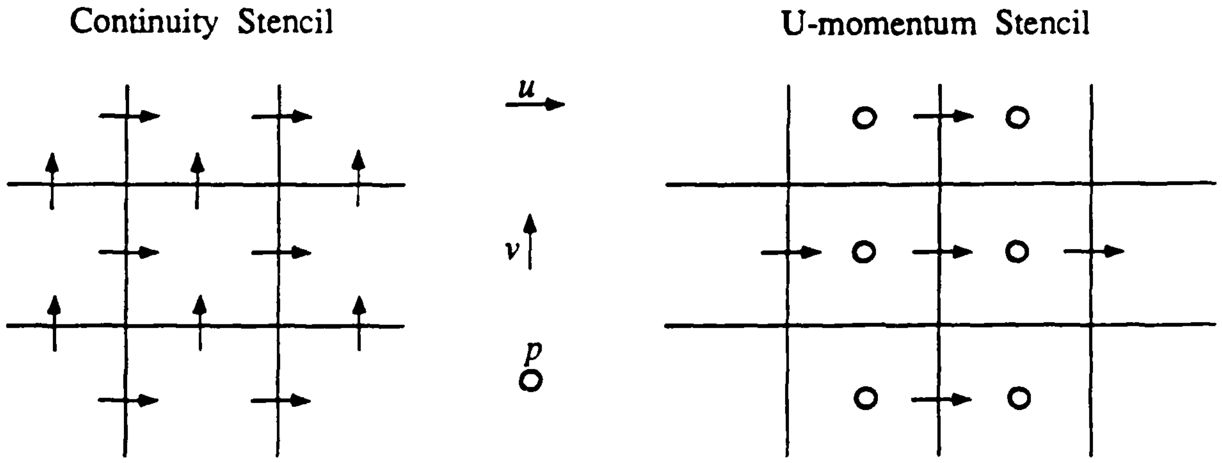


Figure 7.4: Standard staggered non-Cartesian grid; continuity and u-momentum stencils

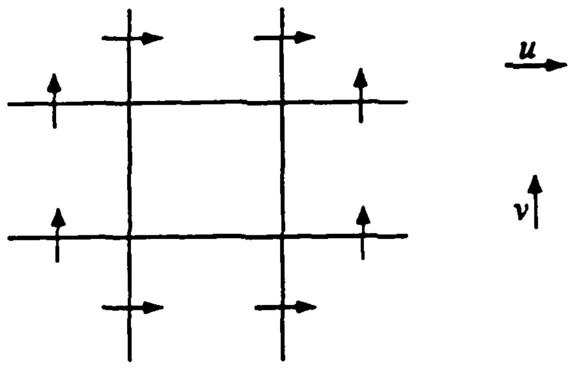


Figure 7.5: Standard staggered rotated Cartesian grid (90°); continuity stencil

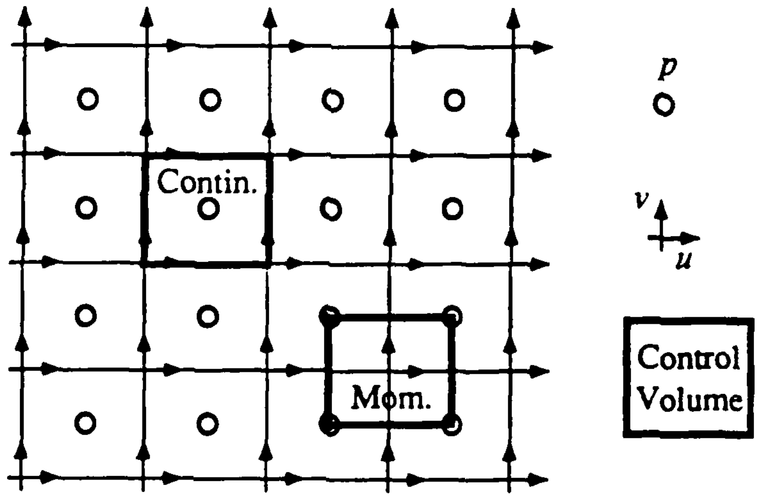


Figure 7.6: Modified staggered variable arrangement

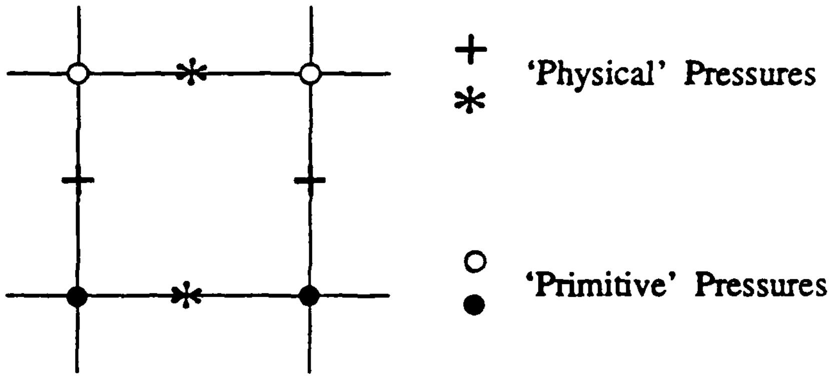


Figure 7.7: 'Primitive' and inter-node 'physical' pressures

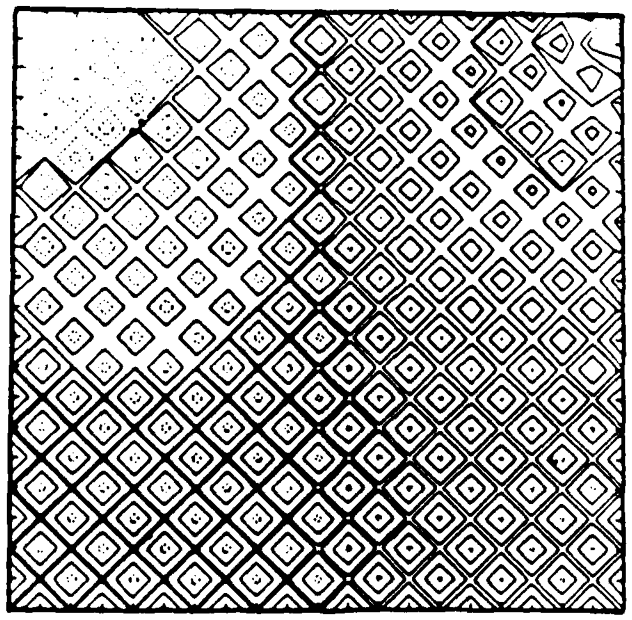


Figure 7.8: Unfiltered pressure; strong oscillations

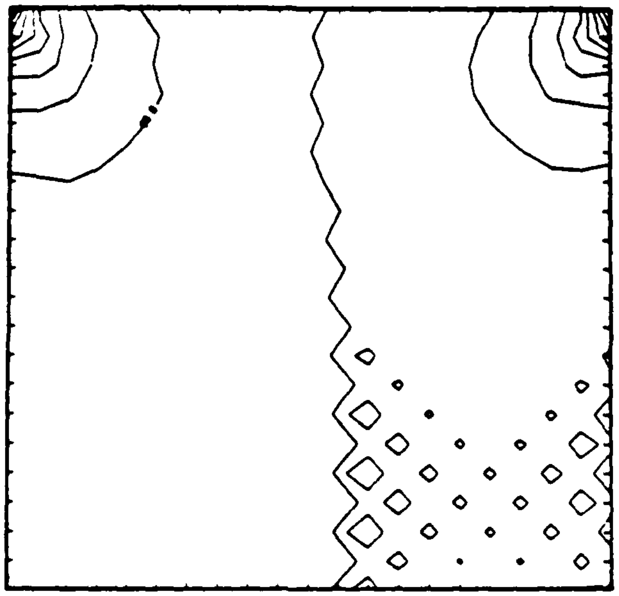


Figure 7.9: Filtered pressure without uniform-field preservation; weak oscillations

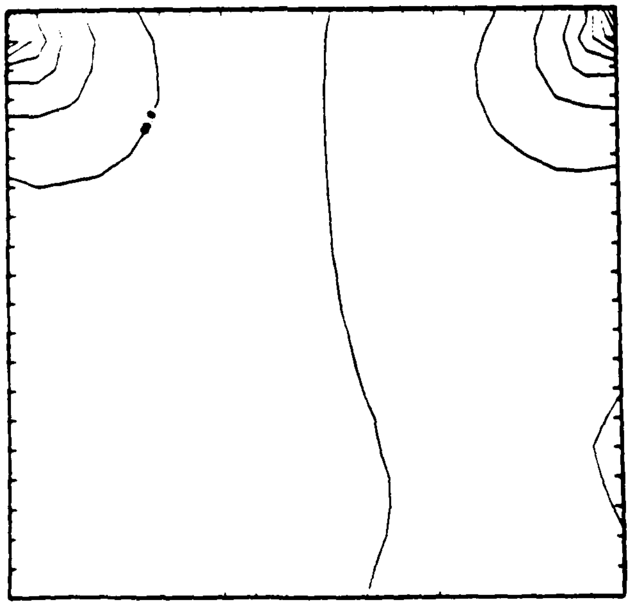


Figure 7.10: Filtered pressure with uniform-field preservation; no oscillations

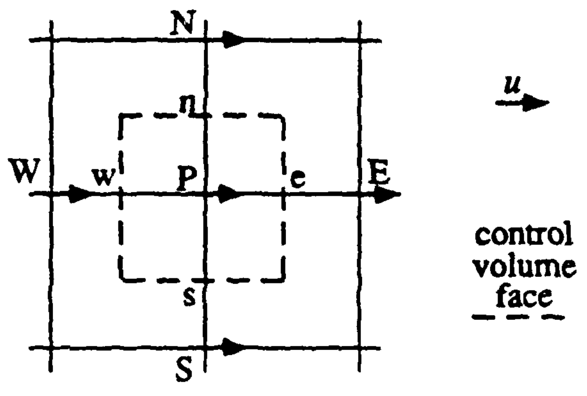


Figure 7.11: Diagram of variable locations for discrete u-momentum equation

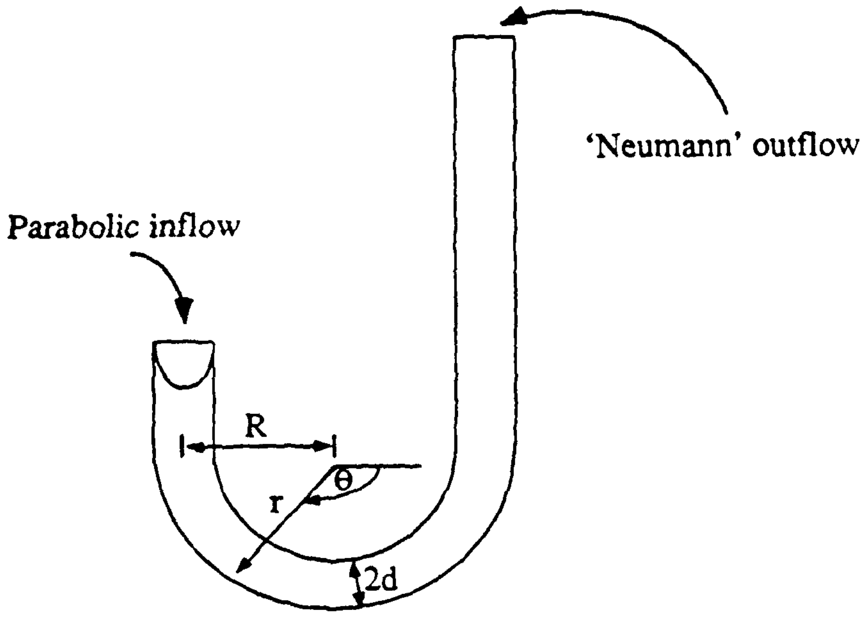


Figure 7.12: U-shaped channel; physical domain

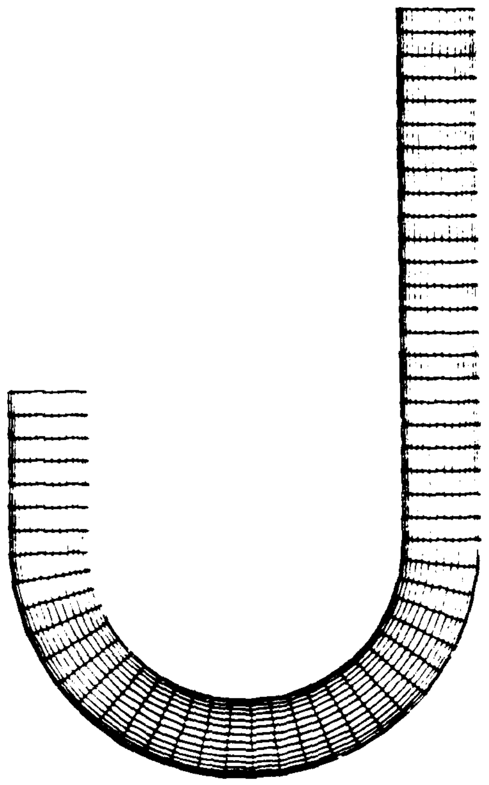


Figure 7.13: Grid for U-shaped channel (every 4th grid line shown)

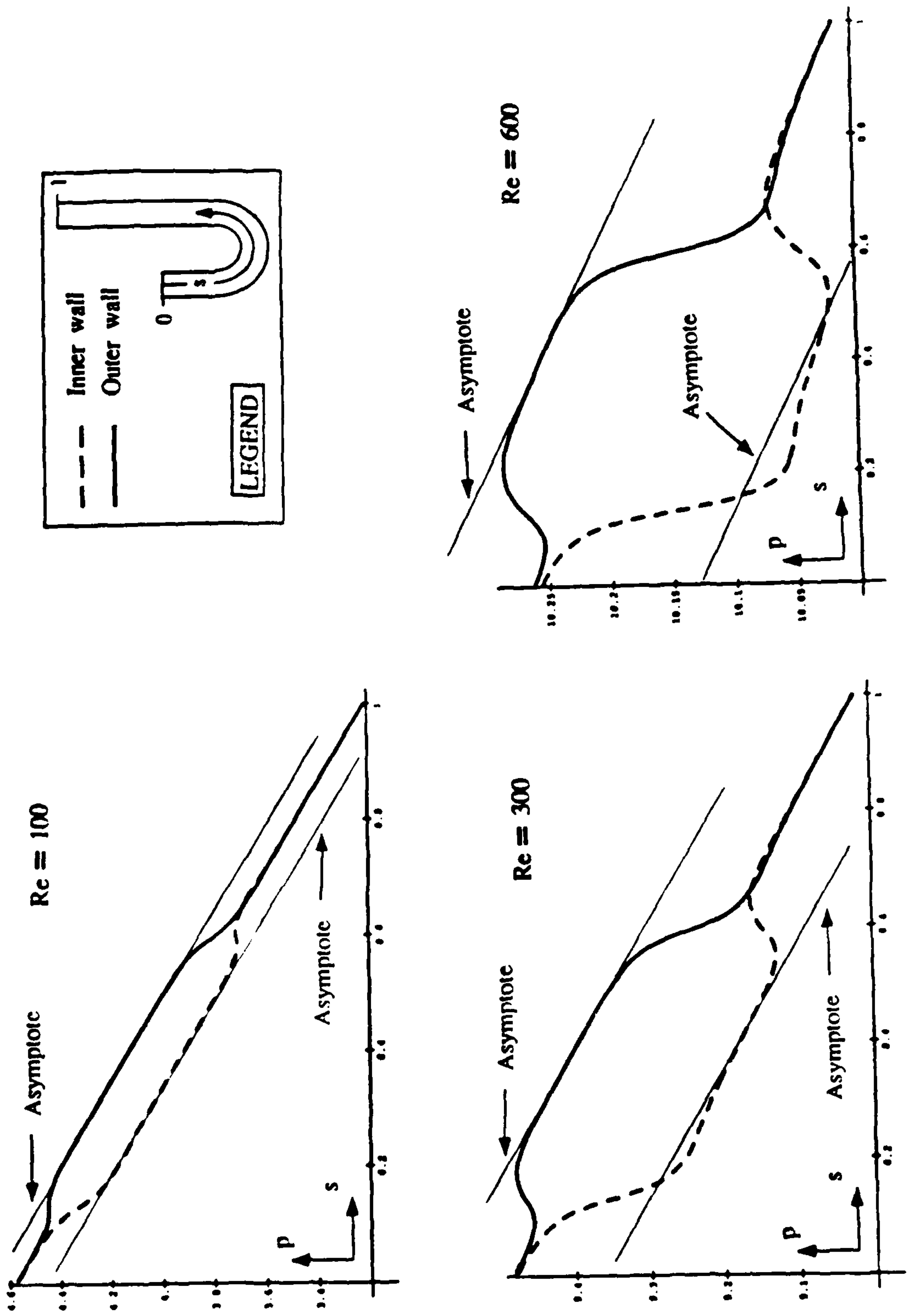


Figure 7.14: Inner- and outer-wall pressures in U-shaped channel

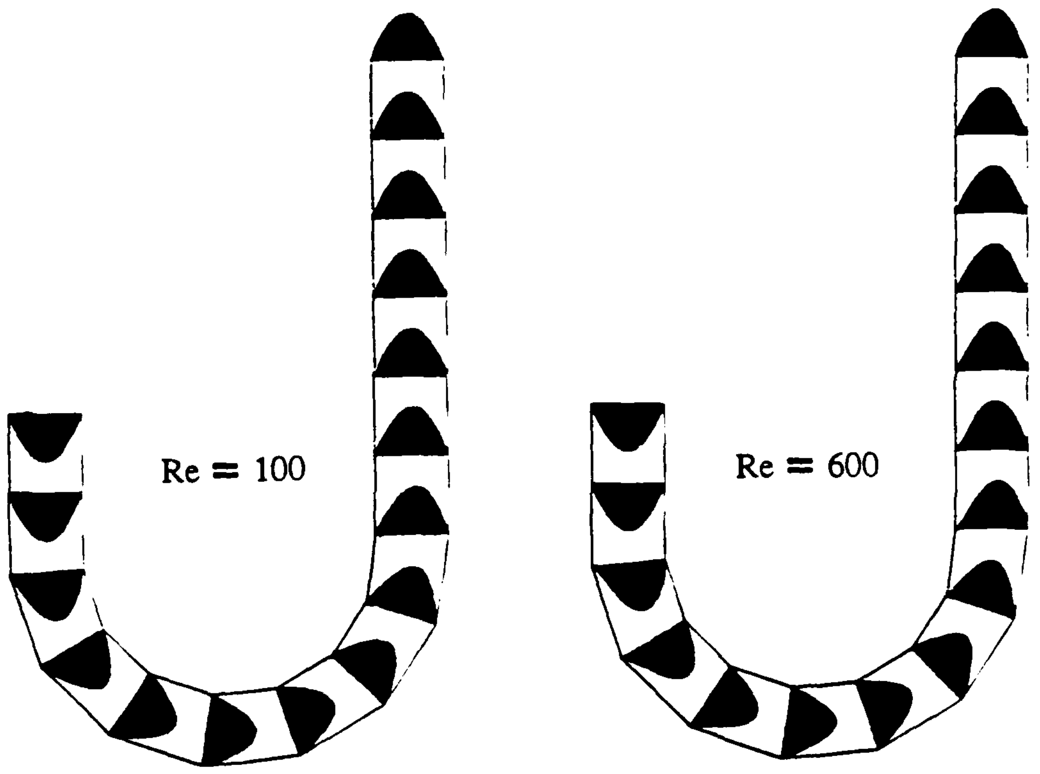


Figure 7.15: Velocity profiles in U-shaped channel

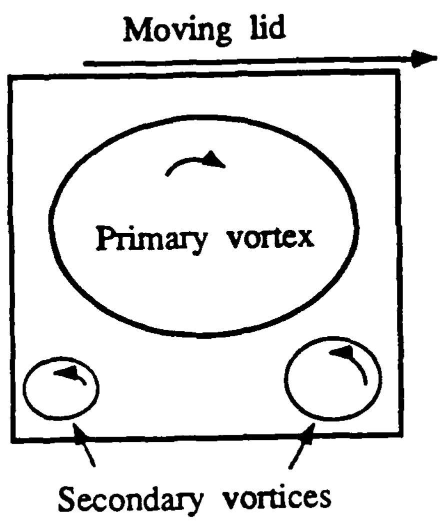


Figure 7.16: Square lid-driven cavity

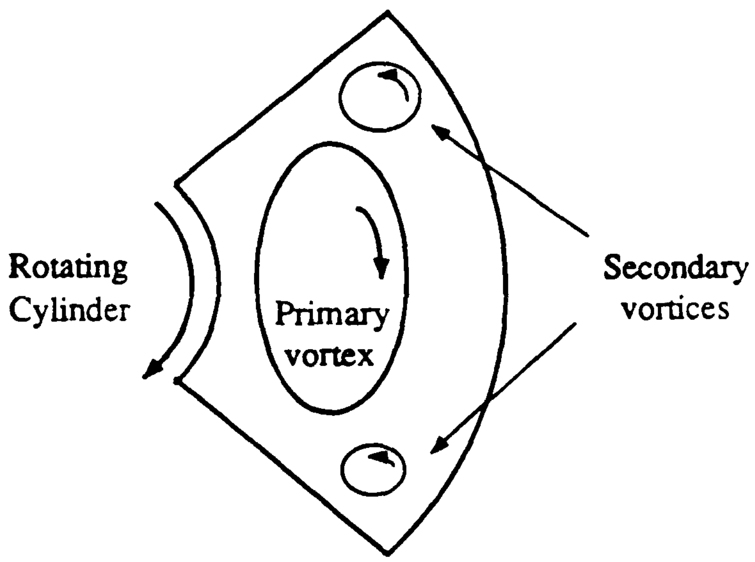


Figure 7.17: Polar lid-driven cavity

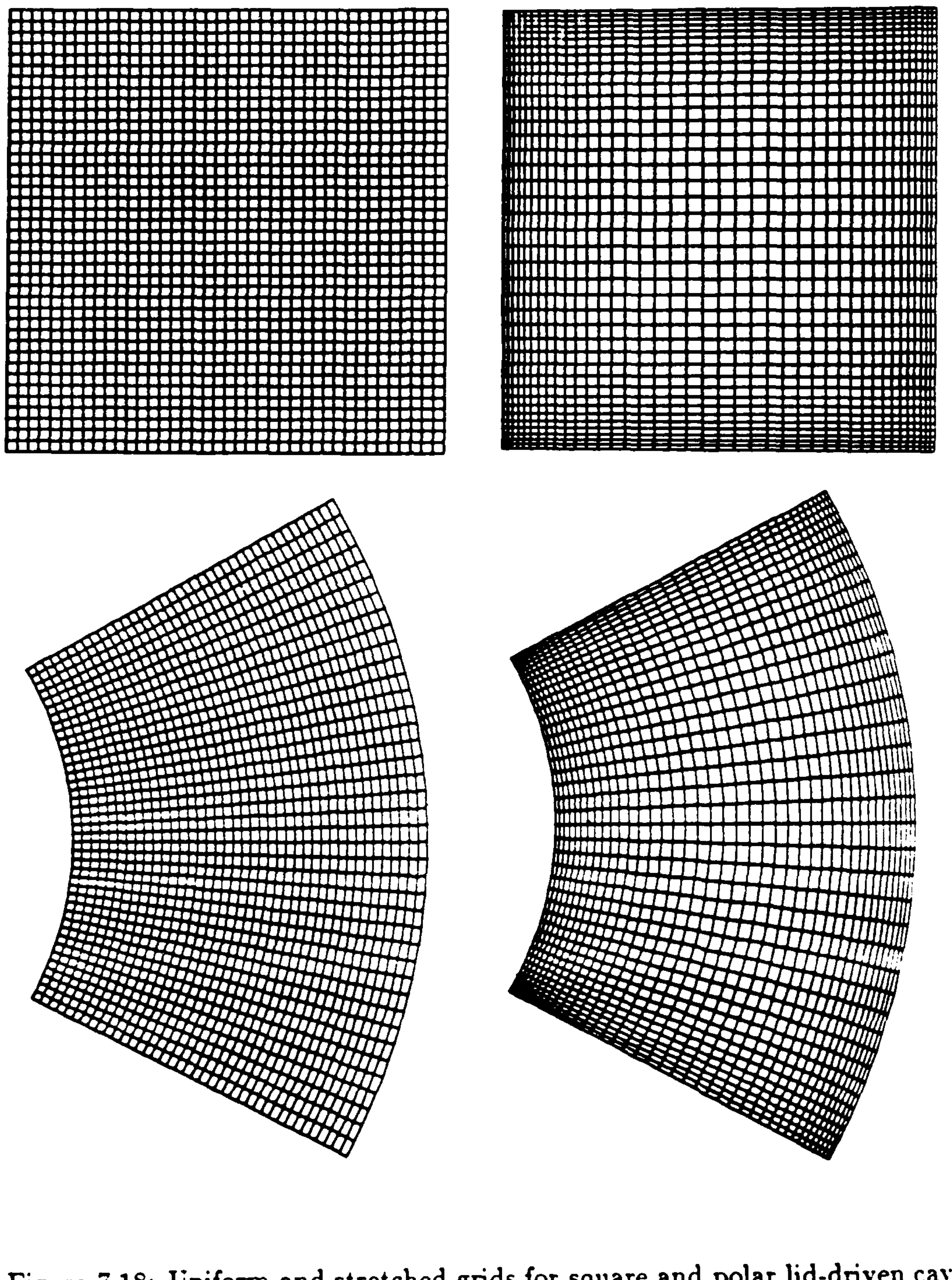


Figure 7.18: Uniform and stretched grids for square and polar lid-driven cavity

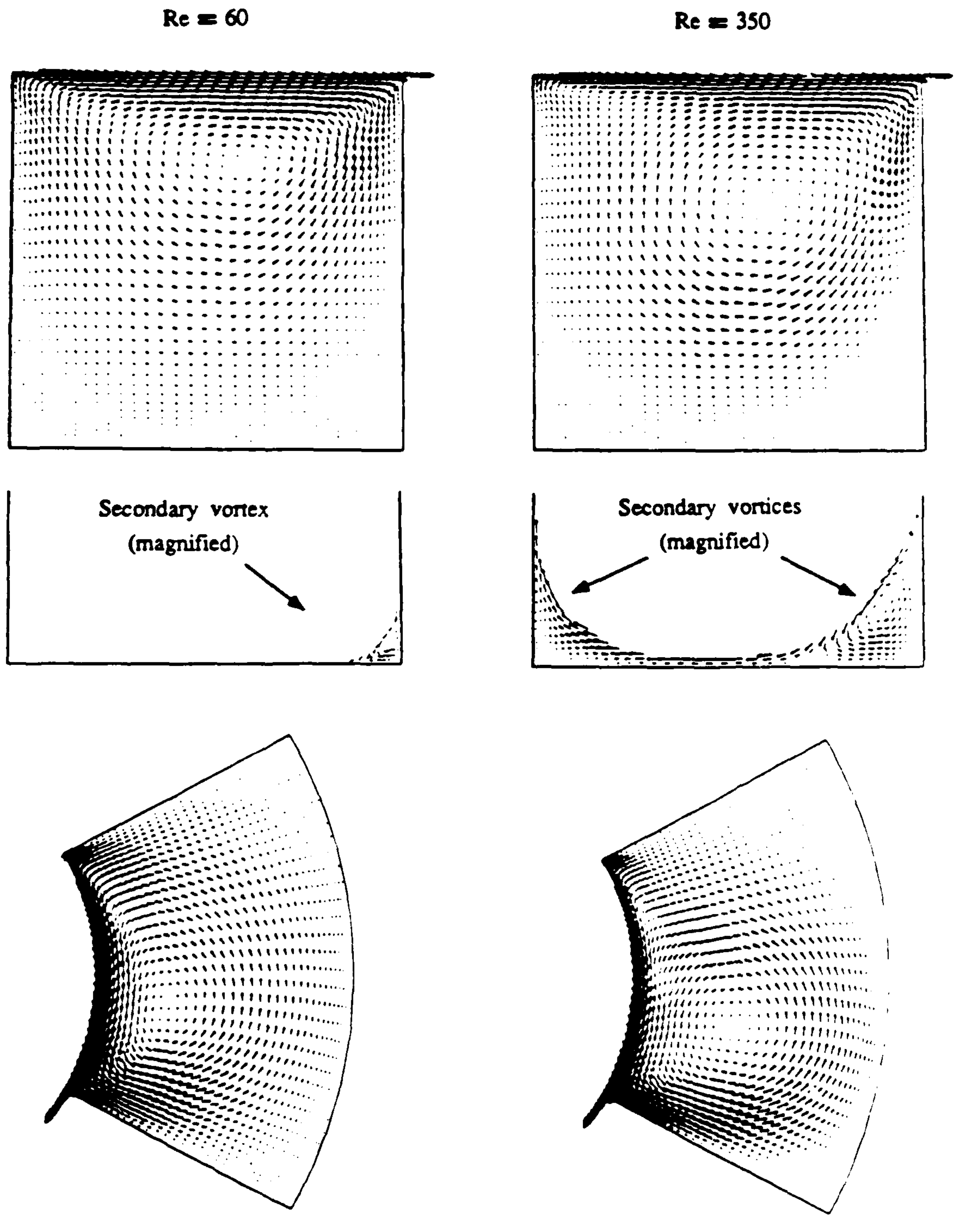


Figure 7.19: Velocity fields for square and polar lid-driven cavity

Chapter 8

NAVIER-STOKES SOLUTIONS ON COMPOSITE GRIDS

8.1 Introduction

In this chapter we discuss fundamental and practical issues concerning the solution of the incompressible, steady Navier-Stokes equations on composite grids. In addition, numerical computations of several flows are presented.

8.2 Interpolation and Conservation

The SWAP communication procedure takes data from the interior of one domain and uses it as boundary conditions for another. Because data are available only on discrete meshes, interpolation is necessary as grid points on overlapping grids often do not coincide. This interpolation causes problems, because it is not conservative. When solving the incompressible Navier-Stokes equations for interior flows, we usually do not prescribe the pressure on the boundaries of the grid(s). That means that momentum fluxes through grid boundaries are not known and cannot (and need not) be preserved. Mass fluxes, however, are known on the boundaries, and a discrete version of the Gauss Divergence Theorem holds for them. Consequently, the residual of the discrete continuity equation cannot be driven to zero on a grid if the discrete mass flux through the boundaries is not exactly zero, *i.e.*, if the boundary values are not conservative; convergence can only be obtained within the error of the interpolating scheme.

Several fixes for this problem have been suggested, the most obvious one being the application of a mass-flux correction factor ([FUC85], [CAR85], [MEA86]).

Outgoing mass fluxes are computed and multiplied by a correction factor so as exactly to balance incoming mass fluxes. Although this correction often speeds up convergence somewhat (Meakin reports an acceleration of approximately 10%), it does not ameliorate the nonconservative properties of interpolation. At convergence, the equation '*correction factor* = 1' must be satisfied, which implies adding another equation to the already closed set of discretized continuity and momentum equations. When grid points in regions of overlap exactly coincide, the correction equation is consistent with interior discretizations (*i.e.*, a linear combination of them) and full convergence is possible. When grid points do not coincide, the correction equation is inconsistent with the interior discretizations and the problem becomes overspecified. Problems become especially acute on relatively coarse meshes and with poor initial guesses, as correction factors may be large and can even diverge, and interpolation is inaccurate.

Apparently, corrective approaches are imperfect. An *a priori* strategy, outlined by Berger [BER84], makes explicit use of the conservation laws that are being discretized. Where two grids intersect, the conservation laws are solved on cell fragments to yield mass fluxes (normal velocities) from one grid into the other (see Fig. 8.1). Although this technique in principle establishes fully conservative communication between grids, it becomes unworkably complex when several grids partially overlap, especially in three dimensions.

Pragmatic engineers will have to settle for interpolation, accepting its inherent nonconservative properties while trying to reduce inconsistencies by increasing its accuracy and decreasing the mesh size.

8.3 Asymmetrically Constricted Channel

The test case described in this section is based on work by Osswald [OSS83], who computed the separated flow in an infinitely long channel containing an asymmetric constriction. The purpose of his investigation was ~compare the upstream and downstream influence of a smooth geometric disturba of finite length on fully developed channel flow. Osswald employed conformal mappings in combination with a

Navier-Stokes solution procedure for curvilinear, orthogonal, boundary-fitted grids. In order conveniently to construct a conformal mapping for a distorted channel, he used the streamlines of the potential flow around a circular cylinder as one family of coordinate lines, and the lines of constant potential as the other family. The walls of the channel are two streamlines passing the cylinder on the same side. An overview of the geometry is given in Fig. 8.2.

Although this geometry is readily amenable to conformal mapping, its drawback is that the constriction is not truly of finite extent, and asymptotic results obtained by Smith [SMI76], [SMI77] and Kumar and Yajnik [KUM80] are not matched well. Instead, we investigate the geometry of a true straight channel of unit width with a confined bump on the bottom wall. The straight section extends 5 channel widths upstream of the bump and 20 downstream. The shape of the bump is given by:

$$y = a\left(1 - \left[\frac{x-c}{w}\right]^2\right)^p , \tag{8.1}$$

where y is the distance from the bottom wall, a is the bump height, x is the horizontal coordinate, c is the x-location of the top of the bump, w is the half-width of the bump, and p is a parameter controlling smoothness (the bottom wall is C^{p-1} continuous). The parameters are chosen such that the constriction and the radius of curvature at the bump are the same as for Osswald's Case IV and the bottom wall is twice continuously differentiable, *i.e.*, $p = 3$, $a = 0.228$, $w = 0.792$. This geometry (truncated) is shown to scale in Fig. 8.3. An overview of the single grid, including a magnified section located at the bump, is shown in Fig. 8.4. Stretching is applied in both coordinate directions to cluster points near the solid boundaries (especially the bottom wall) and at the bump. The *tanh*-stretching is used, mapping $s \in [0, 1]$ to $\hat{s} \in [0, 1]$ through

$$\hat{s} = \frac{\tanh \alpha(s - s_c) + \tanh \alpha s_c}{\tanh \alpha(1 - s_c) + \tanh \alpha s_c} . \tag{8.2}$$

This stretching is centered around the point s_c. The factor α determines the strength of the stretching, with $\lim_{\alpha \to 0}$ corresponding to no stretching.

8.3.1 Single-Grid Computations

Single-grid computations on a 240 by 60 cell mesh were done for the Reynolds numbers 100, 200, 300, 400, and 1000. Close-ups of the separation bubble which appears behind the bump for Reynolds numbers of 200 and up are shown in Fig. 8.5. Upstream boundary conditions consisted of a parabolic velocity profile, whereas second-order-accurate Neumann outflow conditions (zero normal derivative) were used at the downstream boundary. Five important parameters were computed for these flows, *viz.*, the upstream point x_1 and the downstream point x_2 at which the flow differs by less than 5% from fully developed channel flow, the detachment point x_d, the reattachment point x_r, and the reattachment length $\Delta x = x_r - x_d$. According to the nonlinear asymptotic analysis by Smith, the upstream influence of the bump is proportional to $Re^{1/7}$, whereas the downstream influence is proportional to Re for large Reynolds number. Kumar and Yajnik agree with Smith on the downstream length scale, but contend that the upstream length scale does not continue to increase with increasing Reynolds number.

The results of our computations are summarized in Table 8.1. In order to compare with asymptotic results, we also compute the scaled quantities $\hat{x}_1 = x_1/(Re^{1/7})$, $\hat{x}_2 = x_2/Re$, and $\Delta\hat{x} = \Delta x/Re$.

Table 8.1: *Characteristic lengths for channel with bump*

Re	x_1	x_2	x_d	x_r	Δx	$\hat{x}_1$	$\hat{x}_2$	$\Delta\hat{x}$
100	-0.832	1.476				-0.431	0.0148	
200	-0.887	1.860	0.351	0.875	0.524	-0.416	0.0093	0.00262
300	-0.925	2.138	0.293	1.103	0.811	-0.410	0.0071	0.00270
400	-0.955	2.371	0.262	1.312	1.050	-0.406	0.0059	0.00262
1000	-1.075	3.606	0.188	2.422	2.234	-0.401	0.0036	0.00223

The results for the upstream influence of the bump agree well with Smith's analysis, thus disproving Kumar and Yajnik. However, the downstream influence, measured by x_2, is clearly off. A much better match of our data can be obtained by introducing the downstream length scale $\hat{x}_2 = x_2/(Re^{0.35})$, as is evidenced by the log-log plot of the characteristic lengths in Fig. 8.6. In this figure the best power-law fits to all the data are indicated. It should be noted that Osswald's graphs (see below) are straight lines (dashed), because he only supplies two data points. The slopes of these lines, however, exhibit a marked discrepancy with the asymptotic predictions.

The length of the reattachment zone, another measure of the downstream influence of the bump, does scale with the Reynolds number. The iterative procedure had to be carried out to a very high degree of convergence in order to obtain steady values (within 1%) of the characteristic lengths. The l_2-norm of the residual in all cases had to be driven to 10^{-7} to obtain a converged solution. For a Reynolds number of 200 this took 4055 iterations. The severe limit on the residual stems in part from the fact that detachment and reattachement lengths are very sensitive to changes in the computed solution, since they are determined by computing zero-crossings of components of velocity near the wall. Moreover, in large parts of the domain the flow is very regular and the residuals are small, thus making the l_2-norm of the residual deceptively small.

Osswald did computations for Reynolds numbers 100 and 1000. His results are summarized in Table 8.2. Again, the results for $\hat{x}_2$ do not agree well with

Table 8.2: *Characteristic lengths for channel with bump; Osswald's results*

Re	x_1	x_2	x_u	x_d	Δx	$\hat{x}_1$	$\hat{x}_2$	$\Delta\hat{x}$
100	-2.49	2.62	1.20	0.41	0.79	-1.29	0.0262	0.0079
1000	-4.57	16.59	3.78	0.19	3.59	-1.70	0.0166	0.0036

the asymptotic analysis. The other quantities do not scale well with the Reynolds number either. This discrepancy might be attributed to the infinite length of the bump in Osswald's channel.

8.3.2 Two-Grid Computations

The asymmetrically-constricted-channel geometry was selected because it can easily be divided into subgrids on whose interfaces the flow is unidirectional. The dissection into two grids is shown schematically (truncated) in Fig. 8.7. A close-up of the overlap region is shown in Fig. 8.8. Note that grid points on the two grids do not coincide in the region of overlap, although the horizontal grid lines are continuous. The overlap region occupies 4% of the entire area of the domain and accounts for 2% of the points in the two grids. Calculations on this composite grid were done for $Re = 200$. Using second-order bi-Lagrangian interpolation to compute boundary values for the two components of velocity on the interfaces between the grids yielded very good agreement with the single-grid case. Profiles of the horizontal velocities near the centerline of the channel as well as near the bottom wall are shown in Fig. 8.9. The dashed line signifies the single grid solution, whereas the solid line pertains to the two-grid case. A close-up of the profiles in the overlap region is shown in Fig. 8.10. The detachment and reattachment points were computed to be $x_d = 0.351$ and $x_r = 0.876$, respectively. These are within 0.12% of the single-grid values. The number of iterations on each component grid is 12% more than in the single-grid case. On each component grid a maximum of 5 inner iterations of the Navier-Stokes solution procedure is carried out before iterations on the other grid start. If the interior solution procedure converges more tightly than the change in the boundary values through the SWAP interpolation, the inner iteration stops and the solution process is resumed on the other grid. The maximum number of 5 iterations per grid is somewhat arbitrary; iterating once or twice often led to instability, whereas more than 10 iterations deteriorated performance. Several different relaxation factors for updating interface boundary values were tried, but little effect on the convergence rate was noticeable. The computation reported here was carried out using symmetric SWAP relaxation factors of 0.5.

In addition to the regular SWAP procedure, SWAPR was also applied to the two-grid problem. Using exactly the same dissection of the domain as before, the Dirichlet interface condition on the outflow boundary of the left grid was replaced by a second-order-accurate Neumann condition. Again it was found that $x_d = 0.351$ and $x_r = 0.876$, respectively, but now twice as many iterations were needed on the downstream grid. When the amount of overlap between the grids was reduced to zero, convergence was so slow that no final converged solution was obtained, no matter how the relaxation factors for boundary-information transfer were chosen.

This result is disappointing; the one-dimensional analysis of SWAPR, presented in Chapter 6, appears not to extend to two-dimensional situations, despite the fact that the flow is locally virtually one-dimensional. We will show in the next section that results can be obtained with SWAPR in the case of no overlap, although the current method is by no means competitive with SWAP.

8.4 Lid-driven Cavity Flow

The lid-driven cavity flow, although easier to compute than the asymmetrically-constricted channel flow on a single grid because Dirichlet conditions are given on all boundaries, poses new problems for composite-grid computations. It was found in Chapter 6 that for one-dimensional problems, Neumann conditions should be used as interface conditions on outflow boundaries, and Dirichlet conditions on inflow boundaries. However, in two-dimensional problems, inflow and outflow may occur on the same boundary. For example, if the lid-driven cavity, shown in Fig. 8.11, is cut in two, roughly half of the interface boundary experiences inflow, and the other half outflow. If we are to apply SWAPR uniformly to the whole interface boundary, then it is not clear whether the interface should be regarded as an inflow or an outflow boundary, and SWAPR would perform suboptimally on at least part of it.

A solution to the dilemma is to apply SWAPR locally; at every point of the boundary we determine whether inflow or outflow takes place and apply Dirichlet or Neumann conditions accordingly. Let τ denote the 'Dirichlet-ness' of a boundary condition, *i.e.*, $\tau = 1$ means full Dirichlet, and $\tau = 0$ means full Neumann. Then the distribution of τ along the interface boundary at convergence would look like Fig. 8.12. Unfortunately, when τ is determined after every SWAPR iteration, the square-wave profile of Fig. 8.12 is obtained only after very many iterations; the interface conditions keep flipping from one type to the other, and the solution converges very slowly. Apparently, the switch from full Dirichlet to full Neumann, depending on the local velocity normal to the interface, is too crude. Instead of this discrete switch, we propose a smoother transition from Dirichlet to Neumann conditions by prescribing mixed conditions of the type:

$$\tau u + (1-\tau)\frac{\partial u}{\partial n} = f \;, \tag{8.3}$$

in which τ varies continuously from 1 to 0 as the flow changes from inflow to outflow (n is the coordinate normal to the boundary). The variation of τ along the cavity interface boundary then looks like Fig. 8.13.

A somewhat subtle difficulty arises at the point where τ passes through $\frac{1}{2}$. If we denote the solution on the left side of the interface as u_l, and on the right side as u_r, then the boundary condition imposed on the interface for the left domain is:

$$L_l u_l = L_l u_r \;, \tag{8.4}$$

where

$$L_l \equiv \tau_l + (1-\tau_l)\frac{\partial}{\partial n} \tag{8.5}$$

and τ_l is determined from the local inflow into the left grid. A similar condition holds for the interface of the right domain. In case the two grids touch, the two inflow parameters τ_l and τ_r will be complementary at convergence, *i.e.*, $\tau_l = 1 - \tau_r$. At the point where $\tau_l = \frac{1}{2}$, we have $L_l \equiv L_r$, which means that the same condition is applied left and right of the interface. If that is so, no convergence can be obtained for the same reason that convergence of the (symmetric) SWAP is not possible without overlap; asymmetry is necessary. The saving grace is that deterioration of convergence of SWAPR only occurs at the time when the normal velocities on both sides of the interface already match. Indeed, as we will see shortly, convergence is possible using the smooth boundary-condition transition scheme, although it is barely superior to the simple discrete switch.

All computations in this section are performed using a Reynolds number of 10, based on the cavity width and lid speed. The central-difference correction is applied, and solutions are called converged if the combined norm of the residual of the momentum and continuity equations drops below 10^{-5}, and if at the same time the change in the boundary conditions is less than 10^{-5}. The size of the grid is 20 by 20 mesh cells.

A benchmark solution on a single grid was obtained in 220 iterations. This is called one work unit (1 *wu*). The resulting velocity field is shown in Fig. 8.14.

A first set of composite-grid solutions was obtained by applying fixed interface conditions on interior boundaries, *i.e.*, Dirichlet on the right boundary of the left grid, and Neumann on the left boundary of the right grid (Fig. 8.15). Convergence rate and solution accuracy were influenced strongly by the order of accuracy of the Neumann-boundary-condition implementation (O_{bc}) and the order of the Lagrangian interpolation (O_{int}). Although grid points on the interface boundary coincide, the velocity derivatives need to be evaluated numerically. This is done by differentiating the Lagrangian interpolation polynomials. Computations were done for: ($O_{bc} = O_{int} = 1$), ($O_{bc} = 2, O_{int} = 1$), ($O_{bc} = 1, O_{int} = 2$), costing 0.80, 1.47, and 1.80 *wu*'s, respectively. However, all there results are unacceptably inaccurate; the vortex is severely flattened (Fig. 8.16) and a plot of the vertical velocity on a line through the center of the vortex shows a distinct kink in the composite-grid solution (Fig. 8.17).

The reasons for this deterioration of accuracy are twofold. The first is that the degradation of accuracy of the boundary conditions leads to an overall first-order accuracy of the discrete solution. The second is that one-sided difference schemes have to be used for implementation of the Neumann conditions on the 'receiving' grid, as well as for the interpolation of the derivatives on the 'donor' grid; on touching grids the difference stencils for receiving and donating do not coincide, which makes the boundary conditions less accurate. When the grids are allowed to overlap and SWAPR is applied, it is found in one-dimensional test computations that the order of accuracy of the solution remains one for linear interpolation and first-order-accurate Neumann boundary conditions, but the error is reduced substantially, simply because the boundary difference stencils coincide.

When both $O_{bc} = 2$ and $O_{int} = 2$ for the cavity flow, accurate solutions are obtained. Fig. 8.18 shows the composite-grid vertical-velocity profile across the vortex to be in good agreement with the single grid solution. Higher-order interpolation slows down convergence and does not improve the accuracy, so it is not investigated further.

Solution accuracy does have its price; even when optimal relaxation factors are used to update the interface boundary values, we have to spend several work units to obtain a good solution. For the optimal combination, $\theta_l = 1.2$, $\theta_r = 0.5$, convergence takes 3.63 *wu*'s. For nonoptimal relaxation parameters or for different interface conditions (*e.g.*, full Dirichlet on the left grid and mixed Neumann/Dirichlet on the right grid), the cost is slightly higher. As very careful tuning of relaxation parameters is often impractical and yields limited savings, we will only use symmetrical relaxation factors from now on, that is, $\theta_l = \theta_r = \theta$. The optimal case for comparison is $\theta = 0.8$, which takes 4.68 *wu*'s.

When the local SWAPR procedure is adopted —τ depends on the local normal velocity— results are not very encouraging. Letting τ increase linearly from 0 to 1 as the local normal velocity ranges from maximum outflow to maximum inflow leads to a cost of 7.80 *wu*'s! This figure can be improved significantly by letting the $\tau = \frac{1}{2}$ passage occur 'faster' (less smooth). When the τ-response to the normal-velocity distribution becomes steeper (Fig. 8.19), finally approaching a square wave corresponding to the abovementioned discrete switch, the number of iterations decreases sharply, goes through a mild minimum, and ultimately reaches a value of 4.31 *wu*'s (the same as for the discrete switch), which is only slightly better than the fixed-relaxation-factor base case. These results are summarized in Table 8.3, where the cases in the first column refer to the curves of increasing steepness in Fig. 8.19. Curves D and E, although still continuous, are so steep that they cannot be distinguished from each other on the scale of the figure.

Table 8.3: *Work units for local change of SWAPR relaxation factors*

Case	*wu*'s
A	7.80
B	5.42
C	4.90
D	4.30
E	4.31

When overlap between the two component grids is allowed and regular SWAP is applied —Dirichlet conditions on both interface boundaries— the situation changes drastically. Even if the extra work for solving on the larger subdomains is accounted for, SWAP is significantly faster than SWAPR. The results for various amounts of overlap are presented in Table 8.4.

Table 8.4: *Work units for SWAP with varying overlap sizes*

θ	overlap				
	50%	40%	30%	20%	10%
0.4	1.72	1.75	—	—	—
0.5	1.70	1.66	1.77	2.54	8.57
0.6	1.74	1.61	1.64	2.23	7.23
0.7	1.83	1.63	1.57	1.99	6.19
0.8	1.85	1.64	1.51	1.81	5.32
0.9	1.85	1.63	1.46	1.67	4.60
1.0	1.80	1.64	1.42	1.55	2.97
1.1	1.73	1.63	1.38	1.46	2.53
1.2	1.66	1.58	1.35	1.15	2.27
1.3	1.63	1.65	1.33	1.13	2.41
1.4	1.65	1.84	1.34	1.26	1.92
1.5					1.36
1.6					1.56
1.7					3.36

A curious thing is that the work decreases as the overlap is reduced, almost reaching the value of 1 *wu* for 20% overlap. This is because the number of iterations for PRIME or other nonoptimal schemes on a single grid is a strong function of the number of grid points: the decreased intergrid convergence rates due to smaller overlap are offset by the improved intragrid convergence rates due to a smaller number of grid points. Clearly, the convergence becomes more sensitive to θ as the overlap is reduced.

8.4.1 Mesh Refinement

To study the effect of mesh refinement, computations were also done for a 40-by-40-cell grid. Now one work unit corresponds to 696 iterations on a single grid. The best result for SWAPR on touching grids requires 5.6 *wu*'s, whereas SWAP with 20% overlap needs only 0.98 *wu*'s! The growing discrepancy between SWAP and SWAPR as the number of cells increases explains why, for the very large grids needed to cover the asymmetrically constricted channel, no satisfactory solution was found within reasonble computing times using SWAPR without overlap.

8.4.2 SWAP versus SWAPR

So far, we have found that SWAP, which requires overlap between component grids, can be nearly as efficient as a single-grid method, if tuned properly. SWAPR without overlap and with low order of interpolation or low-order-of-accuracy implementation of Neumann boundary conditions converges in about the same number of iterations, but is inaccurate (one order lower than the interior difference scheme). Accuracy is regained with increased order of interpolation and more accurate implementation of boundary conditions, but this takes many more iterations. Accuracy with the low-order-of-accuracy boundary treatment is also improved by allowing overlap, in which case convergence is again as fast as with SWAP. However, the order of accuracy of the solution remains one. Moreover, this again introduces the problem of conservation, and nothing is gained by employing the more sophisticated SWAPR scheme. Hence, for all practical purposes, SWAP remains the method of choice when solving the incompressible Navier-Stokes equations.

A possible way out is offered by one-dimensional computations which show that use of a direct (noniterative) solver on component grids leads to the same number of SWAPR iterations for first- and second-order-accurate implementations of the Neumann boundary conditions; apparently, the slowing of SWAPR convergence for the cavity flow using second-order-accurate Neumann boundary conditions is caused by the reduced performance of the *interior* iterative scheme, and not by the SWAPR communication procedure *per se*. This conjecture is supported by single-grid computations with Neumann outflow boundary conditions. Solving a Poiseuille-flow problem in a straight channel on a 20-by-20-cell grid takes 80% more iterations if second- instead of first-order-accurate boundary conditions are used.

Analytical results can be obtained for the one-dimensional convection/diffusion equation studied in Section 7.4. We redo the convergence analysis for the lagged implementation of Neumann boundary conditions on a grid of size 40 using the point-Gauss-Seidel iteration method. In Table 8.5 the values are listed of the spectral radii ρ_{1st} and ρ_{2nd} of the iteration matrices pertaining to first- and second-order-accurate diffencing of the boundary conditions, respectively.

Table 8.5: *Spectral radii for first- and second-order-accurate Neumann boundary conditions*

Re_c	ρ_{1st}	ρ_{2nd}
0.02	0.9942	1.1271
0.20	0.9859	1.0991
1.00	0.7458	0.7669
2.00	0	0
4.00	2.9820	2.9852

Again, the first-order-accurate implementation has the best convergence rate on a single grid.

Hence, if faster iterative solution procedures on component grids are used (such as multigrid) whose convergence rates are less sensitive to the order of accuracy of the boundary-condition differencing, the efficiency of SWAPR may well improve, rendering it an attractive method because of its conservative properties.

An interesting and successful application of SWAPR is described by Funaro, Quarteroni, and Zanolli [FUN88]. They use a spectral method within subdomains to solve the scalar Helmholtz equation, $\Delta u + \mu u = f$, and transfer interface information through asymmetric Dirichlet/Neumann boundary conditions. It is proved in [FUN88] that a proper choice of relaxation parameters —which depends on the geometry— yields very fast convergence of the SWAPR iterative procedure, although numerical experiments presented are limited to the one-dimensional Helmholtz equation and the two-dimensional Poisson equation (*i.e.*, $\mu = 0$). An important difference from the SWAPR method presented here is that no iterations are carried out on the subdomains; single-grid solutions are obtained directly. This is impractical for nonlinear problems, especially in multiple space dimensions. If the method by Funaro *et al.* is to be extended to the steady, incompressible Navier-Stokes equations, another problem surfaces in that the continuity equation cannot be solved directly if the subdomain Dirichlet boundary conditions are not conservative. Iterative methods within subdomains must then be applied, and cannot be carried to convergence before switching to another subdomain.

8.5 Periodicity and Single-valuedness

It can be shown [TEM77] that the boundary-value problem for the Stokes equations in which only the velocity is specified on the boundaries of the domain is well-posed under very generous conditions on differentiability and geometric complexity.

The steady Navier-Stokes equations with velocity boundary conditions (NS-BVP) sometimes permit multiple solutions, a well-known example being the Taylor-vortex flow between two rotating, concentric cylinders. Because of this nonuniqueness, the NS-BVP is not well-posed in general.

However, we will focus on steady, laminar flows for which only one solution is known. Hence, we assume that the NS-BVP is well-posed. This implies that the solution is single-valued and differentiable everywhere in the interior of the domain. Although this requirement appears trivial, it is not always easily satisfied for reentrant problems. Reentrant problems arise whenever a domain is not simply connected (contains holes), or when (sequences of) subproblems form closed loops in space. The latter may occur even in a simply-connected region, for example when a chain of refined grids is constructed adaptively (see Fig. 8.20).

To illustrate these issues, we consider the two-dimensional annular Couette flow problem. The equations are almost the same as for the asymptotic solution in the 180-degree circular bend discussed in Chapter 7, *i.e.*,

$$-\frac{\partial u_\theta^2}{\partial r} = -\frac{\partial p}{\partial r}, \tag{8.6}$$

$$0 = \frac{1}{Re}\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial r u_\theta}{\partial r}\right). \tag{8.7}$$

Boundary conditions are:

$$u_\theta|_{(R-d)} = u_1 , \qquad u_\theta|_{(R+d)} = u_2 . \tag{8.8}$$

The unique solution is:

$$u_\theta = \frac{1}{4Rd}\left\{(R-d)\left(\frac{(R+d)^2}{r} - r\right)u_1 + (R+d)\left(r - \frac{(R-d)^2}{r}\right)u_2\right\} , \tag{8.9}$$

$$p = \int^r \frac{u_\theta^2}{r}\, dr . \tag{8.10}$$

However, should we relax the requirements of continuity and differentiability on the pressure, then we can add solutions of the type of the circular bend to the velocity profile and write:

$$u_\theta = \frac{1}{4Rd}\left\{(R-d)\left(\frac{(R+d)^2}{r} - r\right)u_1 + (R+d)\left(r - \frac{(R-d)^2}{r}\right)u_2\right\} + Q\frac{\frac{1}{2}r\ln\left(\frac{r^2}{R^2-d^2}\right) + \left[(R^2+d^2)r - (R^2-d^2)^2/r\right]\frac{1}{4Rd}\ln\left(\frac{R-d}{R+d}\right)}{\frac{(R^2-d^2)^2}{4Rd}\ln^2\left(\frac{R-d}{R+d}\right) - Rd} \tag{8.11}$$

$$p = \int^r \frac{u_\theta^2}{r}\, dr + \frac{2Q}{Re}\left(\frac{(R^2-d^2)^2}{4Rd}\ln^2\left(\frac{R-d}{R+d}\right) - Rd\right)^{-1}\theta . \tag{8.12}$$

The variable Q signifies the added mass flux due to the azimuthal pressure gradient. Clearly, this solution is unacceptable and we should have $Q \equiv 0$. If a branch cut were made at $\theta = 0$, we would find that $p(r,0^+) \neq p(r,0^-)$, the difference being proportional to Q. Hence, in the continuous case it is easy to single out the physical solution by demanding continuity of the pressure.

The discrete problem is more difficult. As was shown in Chapter 7, staggered grids eliminate the need to specify the pressure on the boundary of the domain as they do not have pressure nodes on boundaries. This is fine if a grid side is a physical boundary, but if it is a reentrant boundary (branch cut) it would be useful to have pressure points on that side so that the pressure is uniquely defined there (see Fig. 8.21). In the case of annular Couette flow, we may define one reentrant grid that exactly covers the domain. If only the velocities are matched at the branch cut, a jump in the pressure may occur across the cut. This has been observed numerically. The problem may go undetected since the size of the jump depends on the iterative scheme used. Moreover, we may accept jumps in the pressure field from one grid to another, as pressure is a relative quantity to which an arbitrary constant can be added. This is an obscuring factor; an inconsistent reentrant pressure field can only be unmasked by moving through the closed chain of grids and bringing the pressure fields in agreement at every grid interface through addition of a constant to the pressure on each grid. After coming full circle, no jump may occur between the last grid and the first grid.

If the annular domain is covered with one partially self-overlapping grid, we can interpolate boundary conditions for the velocity in the SWAP or SWAPR style from the interior of the grid itself and compute the whole flow, including the pressure field. A three-dimensional, unsteady version of this case is described in [MEA86]. Results were satisfactory and in good agreement with experimentally obtained data. However, when a two-dimensional, steady computation was done in this study, the solutions did not converge and showed ever-increasing azimuthal pressure gradients, with jumps across branch cuts. In this case the problem can be eliminated by prescribing the mass flux in the annulus, but generally this is not a known quantity.

The true source of the problem is that there is no communication between pressure fields at grid interfaces. They should be matched explicitly. For the annular grid (or any other single, self-connected grid) this can be achieved by formulating the problem as a true periodic problem, *i.e.*, by discretizing the differential equations at all points inside the annulus, including those on the branch cut. The momentum equation stencils for velocities on the branch cut will then include pressure nodes straddling the interface, and single-valuedness and smoothness at convergence are guaranteed. One might call this procedure the 'direct' solution of the periodic problem.

The attractiveness of the direct method diminishes when more than one grid is used to cover the annulus, because then solutions need to be computed on all grids simultaneously to maintain periodicity; we prefer to use SWAP. That means that velocities on boundaries are interpolated from adjacent overlapping grids. Consequently, the momentum equations are not enforced on grid boundaries and a natural way of connecting pressure nodes on different grids is lost. Therefore the following strategy is used.

Formulate discrete momentum equations on the target grid as before, but replace the pressures at the nodes adjacent to the interpolation boundary by the average of the pressures on the target grid and the donor grid (donor values must be interpolated). At convergence the two will be equal and the pressures match.

Test computations of nonreentrant problems show that the pressure-communication scheme slows convergence considerably with respect to 'unconstrained' iteration. Even on a single grid it has been observed that prescribing the correct pressure at nodes adjacent to the boundary causes considerable slowdown. Hence, the pressure is only matched on reentrant grids, or reentrant sequences of grids.

An alternative approach to solving the pressure-communication problem is offered by Perng [PER89B], [PER89C]. He formulates pressure equations for every component grid and solves these globally using a Schwarz-like communication procedure. This ensures continuity of the pressure field across grid interfaces at convergence. The time-accuracy of his explicit, unsteady method guarantees that full convergence of the pressure is possible at every time step.

8.6 Cylinder in Crossflow

An interesting problem involving reentrant grids and more challenging composite-grid generation is that of the steady, laminar flow past a circular cylinder. It is a classical problem that has been studied by many researchers.

8.6.1 Flow Description

One of the earliest quantitative experimental and computational investigations was carried out by Thom [THO33], but the results are rather crude, and no values for the length of the separation bubble behind the cylinder —the prime parameter of interest— are published. Later experiments conducted by Taneda [TAN56] show that when the Reynolds number, based on cylinder diameter and free-stream velocity ($Re = U_\infty D/\nu$), is less than 5, the flow remains attached (Fig. 8.22a). At the critical Reynolds number of about 5, the boundary layer separates and two symmetric, steady vortices form behind the cylinder (Fig. 8.22b). These become elongated as the Reynolds number increases, but the flow remains steady up to $Re = 35$, at which time disturbances occur far downsteam of the cylinder (Fig. 8.22c). The vortices themselves remain stable and symmetric until $Re = 45$. Then the disturbances propagate upstream and start to affect the wake structure, leading to the Kármán vortex street (Fig. 8.22d). This regular, unsteady phenomenon persists from $Re \approx 60$ to $Re \approx 5000$, after which fu[illegible]y turbulent mixing sets in.

Most researchers focus on the intriguing properties of the inherently unsteady flow at larger Reynolds numbers, but some experimental and computational results are available for the steady, laminar range. These are given in Table 8.6. The headings of the columns denote the references. The dimensionless quantity L/D signifies the ratio of bubble length to cylinder diameter, as indicated in Fig. 8.22b.

Additional experiments by Acrivos *et al.* [GRO64], [ACR68] yield the curve fit $L/D = 0.07Re - 0.5$. Clearly, a lot of scatter exists in the data. Results published in [TAN56], [KAW66], and [KAR87] were only presented graphically. The graph in [TAN56] is particularly poor (see footnote for data point $Re = 42$). Collins and Dennis [COL73] report that the bubble length was still increasing slightly at the time the computations were terminated. On the other hand, they remark that further grid refinement tends to decrease the bubble length.

8.6.2 Numerical Solution

The physical domain used here is the one employed by Karniadakis [KAR87], shown in Fig. 8.23. Boundary conditions are also taken from [KAR87]. On the left boundary we specify uniform inflow (Dirichlet). At the top and bottom boundaries of the domain, far away from the vorticity-generating cylinder, we prescribe velocities pertaining to the invis[illegible] potential flow around a circular cylinder (Dirichlet). A fully-developed-flow condition is imposed on the right boundary (Neumann: $\frac{\partial}{\partial x} \equiv 0$). At the cylinder wall a no-slip condition is used (Dirichlet).

Table 8.6: *Previous results for separation-bubble length L/D*

Re	TAN56[1]	KAW66[2]	DEN70[3]	COL73[2]	KAR87[2]
7			0.10		
9	0.2				
10		0.34	0.26	0.26	0.24
11	0.3				
13	0.4				
17	0.7				
20		1.5	0.94		1.11
21	1.1				
27	1.4				
30		2.0			1.90
35	1.8				
39	2.2				
40		3.0	2.35	2.15	3.29
42	2.2[4]				

[1] Experiment

[2] Unsteady computation, relaxing to steady state

[3] Steady computation

[4] This data point was cited in [HON69] as a result of the experiment performed in [TAN56]

We apply SWAP on overlapped grids for this geometry. An overview of the composite grid, consisting of five components, is presented in Fig. 8.24. Only every fourth grid line is shown for clarity. Notice the strong stretching applied on the annular grid (No. 2) in order to resolve the separation bubble. Stretching is also applied on grids 1 and 3 to concentrate more points near the cylinder, and near the wake. Pressure matching is only applied on the reentrant boundaries of the annular grid.

Calculations were done for Reynolds numbers 10, 20, 30, and 40. Several sequences for stepping through the chain of grids were tried, but only minor differences in efficiency were observed. The final sequence used in this study is: $\{1,2,3,4,5,2\}$. Sufficient convergence criteria are: residual drops below 10^{-6}, solution changes on interface boundaries are less than $5 * 10^{-6}$. Biquadratic Lagrangian interpolation was used for communication between grids, and the Neumann condition was second-order accurate. For Reynolds numbers 10, 20, and 30, the SWAP relaxation factor was 1, and the velocity relaxation factors were 0.8. Reynolds number 40 required stronger underrelaxation: 0.8 for SWAP and 0.6 for velocity. The results of the computations are given in Table 8.7.

Table 8.7: *Composite-grid results for separation-bubble length L/D*

Re	L/D
10	0.24
20	0.87
30	1.46
40	2.01

Graphical representation of these data, together with results published by other researchers, is given in Fig. 8.25. Good agreement with the experimental values is obtained. Accuracy of the present solutions was verified by changing the mesh size, the order of interpolation, and the severity of the stretching of the annular grid for the case $Re = 20$. Deviations of less than 1% were observed with respect to the value listed in Table 8.7.

Extra-fine-grid calculations were performed for Reynolds numbers 30 and 40, in which the number of cells on all grids was multiplied by 1.5 in both coordinate directions. The separation-bubble length increased from 1.43 to 1.46 for $Re = 30$, and from 1.95 to 2.01 for $Re = 40$. Applying Richardson extrapolation, we estimate that the errors in the finest-grid computations are 1.2% and 2.5%, respectively. In other calculations we consistently observed an increase in bubble length as the mesh size was decreased. Hence, the grid-independent bubble length is estimated to be 1.48 for $Re = 30$, and 2.06 for $Re = 40$.

A solution at $Re = 40$ was also computed for a 'composite' grid consisting of only the annular grid, with uniform-flow conditions (Dirichlet) prescribed on its perimeter. This yielded a nondimensional bubble length of 1.59, which demonstrates that the composite-grid communication procedure works well; the annular grid by itself is not sufficient to compute accurate solutions. In another experiment the potential-flow conditions on top and bottom boundaries of the domain were replaced by uniform flow conditions. In that case a value of 1.97 was obtained for the bubble length; apparently, the top and bottom 'wall' boundary conditions do not exert a strong influence on the bubble length.

Close-ups of the separation bubbles are presented in Fig. 8.26. It should be noted that a very high degree of symmetry is obtained in the numerical solutions, despite the fact that the problem is solved as a truly two-dimensional problem without a symmetry plane. Velocity profiles for $Re = 40$ at various locations along the domain are shown in Fig. 8.27. Reflections of the upper halves of these profiles, indicated by dashed lines, are superimposed on the lower halves. Differences are negligible.

Fig. 8.28 shows the pressure profiles along the centerline of the self-overlapped annular grid, and on a line perpendicular to the cylinder in the region of overlap ($Re = 10$). Pressures from different parts of the computational domain, evaluated at the same physical locations, are indicated by dashed lines and solid lines, respectively. Excellent agreement is found between the two pressures in the overlap region. Apparently,, the pressure communication scheme across grid interfaces works well.

In addition to the nondimensional bubble length we also compute the drag on the cylinder as a function of the Reynolds number. The total-drag coefficient C_D is the sum of the viscous-drag coefficient $C_{D,v}$ and the pressure-drag coefficient $C_{D,p}$. The results of our computations (NEW) are presented, together with those of others', in Tables 8.8, 8.9, and 8.10.

Table 8.8: *Viscous-drag coefficients $C_{D,v}$ for circular cylinder*

Re	THO33[1]	KAW66	COL73	NEW
10	1.5	1.3	1.29	1.36
20	0.9	0.8		0.87
30		0.7		0.67
40		0.6	0.54	0.55

[1] Computation

A curve-fit, presented in [GRO64], gives the simple relation $C_{D,r} = 0.62 + 12.6/Re$ for relatively large Reynolds numbers ($Re \geq 40$). Again the spread in the data is considerable. The drag coefficients for the composite-grid computations, although somewhat large, agree well with the other results.

Table 8.9: *Pressure-drag coefficients $C_{D,p}$ for circular cylinder*

Re	THO33[1]	THO33[2]	KAW66	COL73	NEW
10	1.9	1.8	1.6	1.65	1.93
11		1.4			
20	1.2		1.3		1.44
30			1.1		1.26
37		1.0			
40			1.0	1.02	1.15

[1] Computation
[2] Experiment

Table 8.10: *Total-drag coefficients C_D for circular cylinder*

Re	WIE[1]	THO33[2]	KAW66	DEN70	COL73	NEW
10	3.0	3.4	2.9	2.9	2.94	3.29
20	2.5	2.1	2.1	2.0		2.31
30	2.4		1.8			1.93
40	2.1		1.6	1.5	1.56	1.68

[1] Experiment by Wieselsberger, quoted from [SCH79]
[2] Computation

In order to demonstrate the full power of the composite-grid strategy for this geometry, an asymmetric problem is also solved. It concerns the same cylinder, spinning at constant angular speed of $5/\pi$ Hz in a channel with solid walls. The inflow condition is Poiseuille flow (Dirichlet), whereas the outflow condition is fully developed flow again (Neumann). No-slip conditions are applied on top and bottom walls. A close-up of the flow field around the cylinder for $Re = 20$ is shown in Fig. 8.29. The Reynolds number in this case is based on the cylinder diameter and the maximum inflow velocity. The channel width is 10 times the cylinder diameter. Interestingly, the boundary layer remains attached and the rotation merely deflects the flow. The drag coefficients in the flow direction are: $C_{D,v} = 0.79$, $C_{D,p} = 1.23$, $C_D = 2.02$, whereas the corresponding lift coefficients are: $C_{L,v} = 0.14$, $C_{L,p} = 1.21$, $C_L = 1.35$.

In another computation with a spinning frequency of $0.5/\pi$ Hz and a Reynolds number of 40, the flow structure is hardly altered by the rotation, and two nearly symmetrical vortices are formed. Now the drag coefficients in the flow direction are: $C_{D,v} = 0.49$, $C_{D,p} = 1.00$, $C_D = 1.49$, and the lift coefficients are: $C_{L,v} = 0.01$, $C_{L,p} = 0.12$, $C_L = 0.13$.

In both spinning cases the flow downstream of the cylinder returns to symmetry very rapidly.

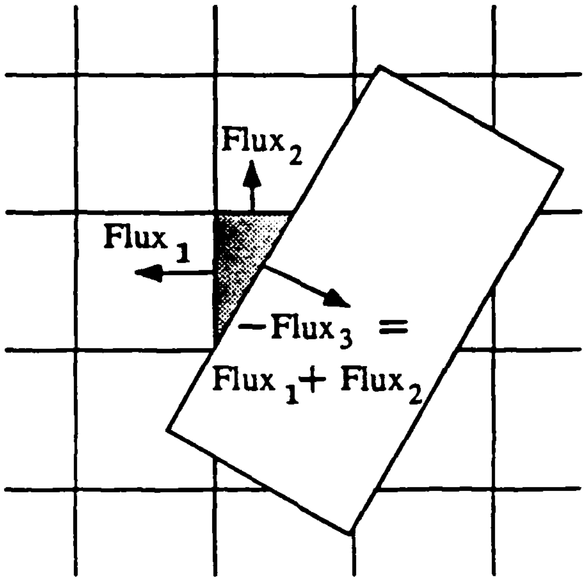


Figure 8.1: Conservation law used on cell fragment to compute velocity

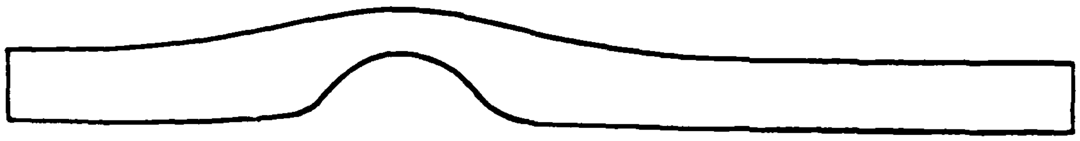


Figure 8.2: Overview of Osswald's asymmetrically constricted channel

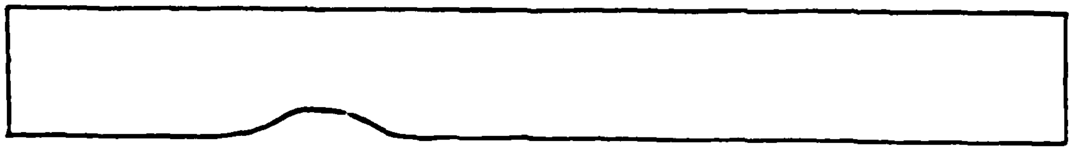


Figure 8.3: Overview of new asymmetrically constricted channel

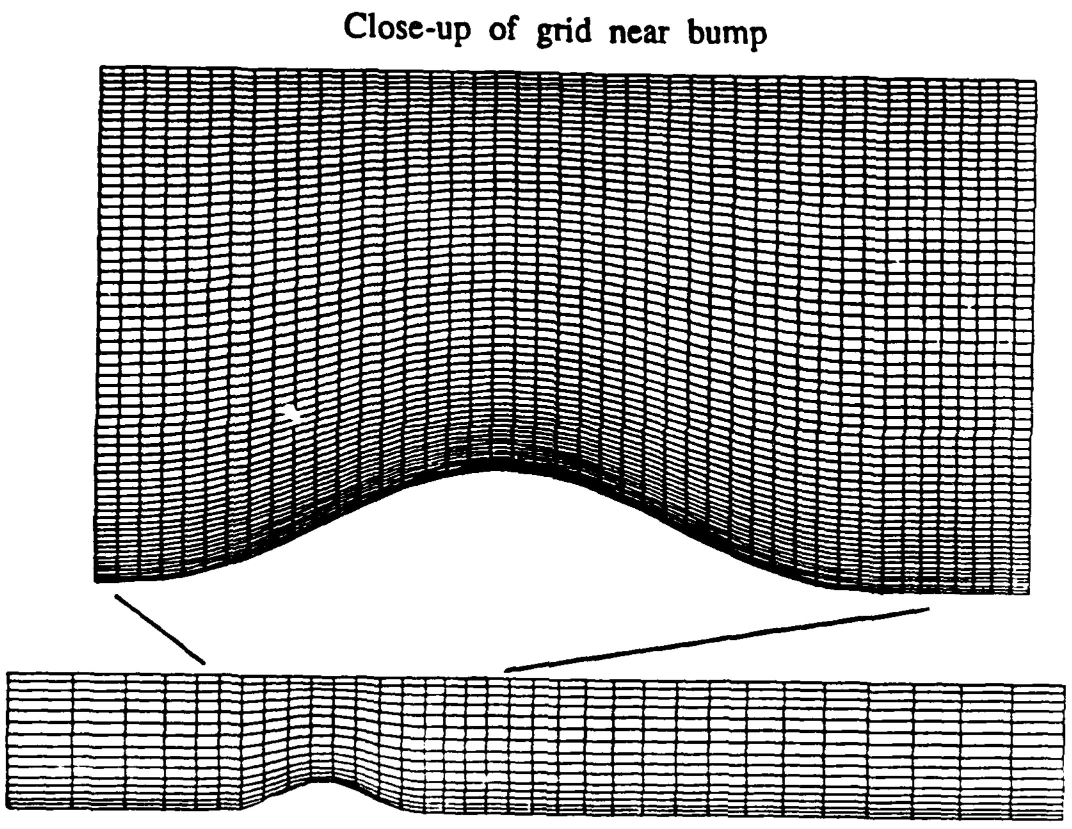


Figure 8.4: Single grid (truncated) for asymmetrically constricted channel

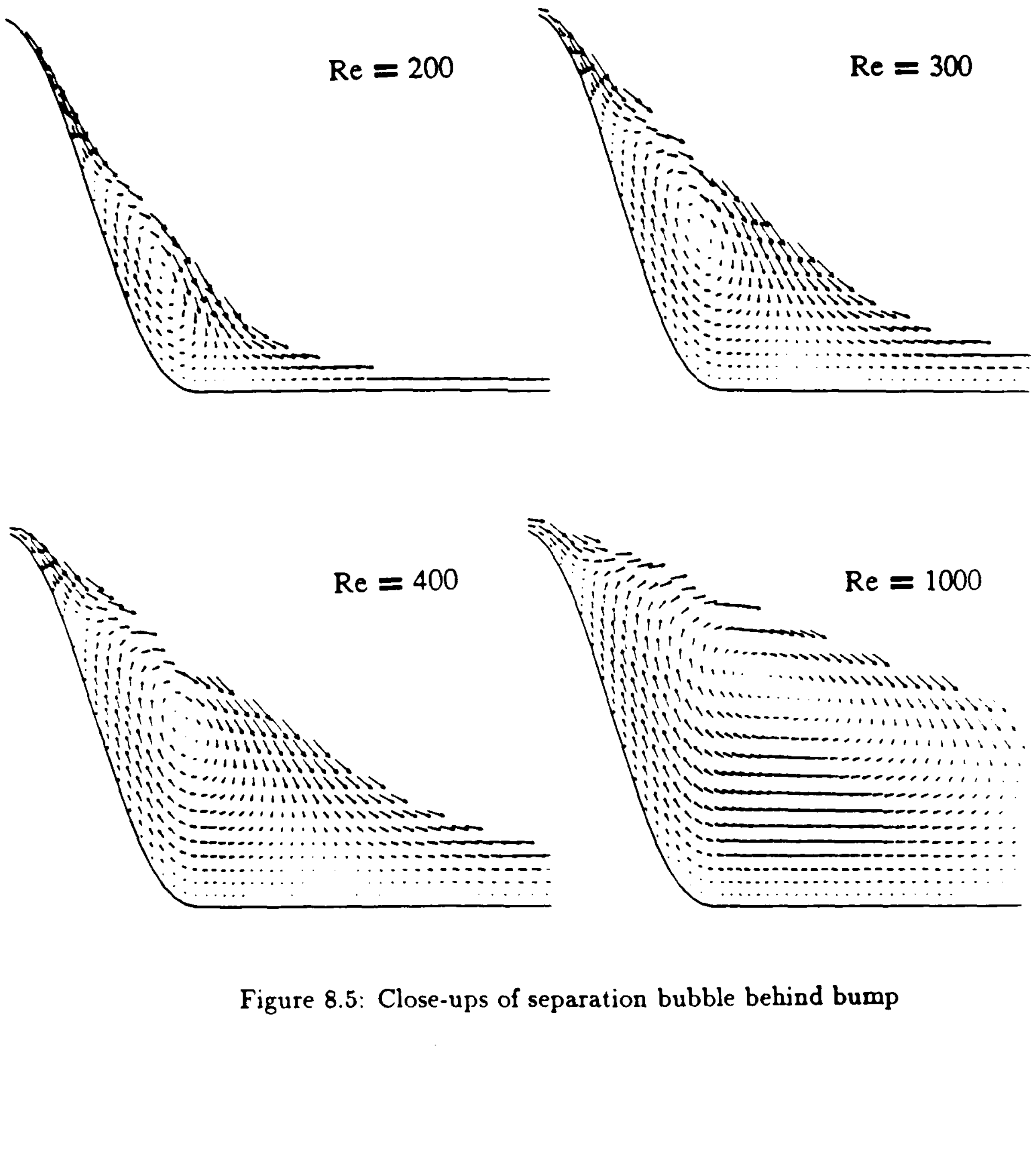


Figure 8.5: Close-ups of separation bubble behind bump

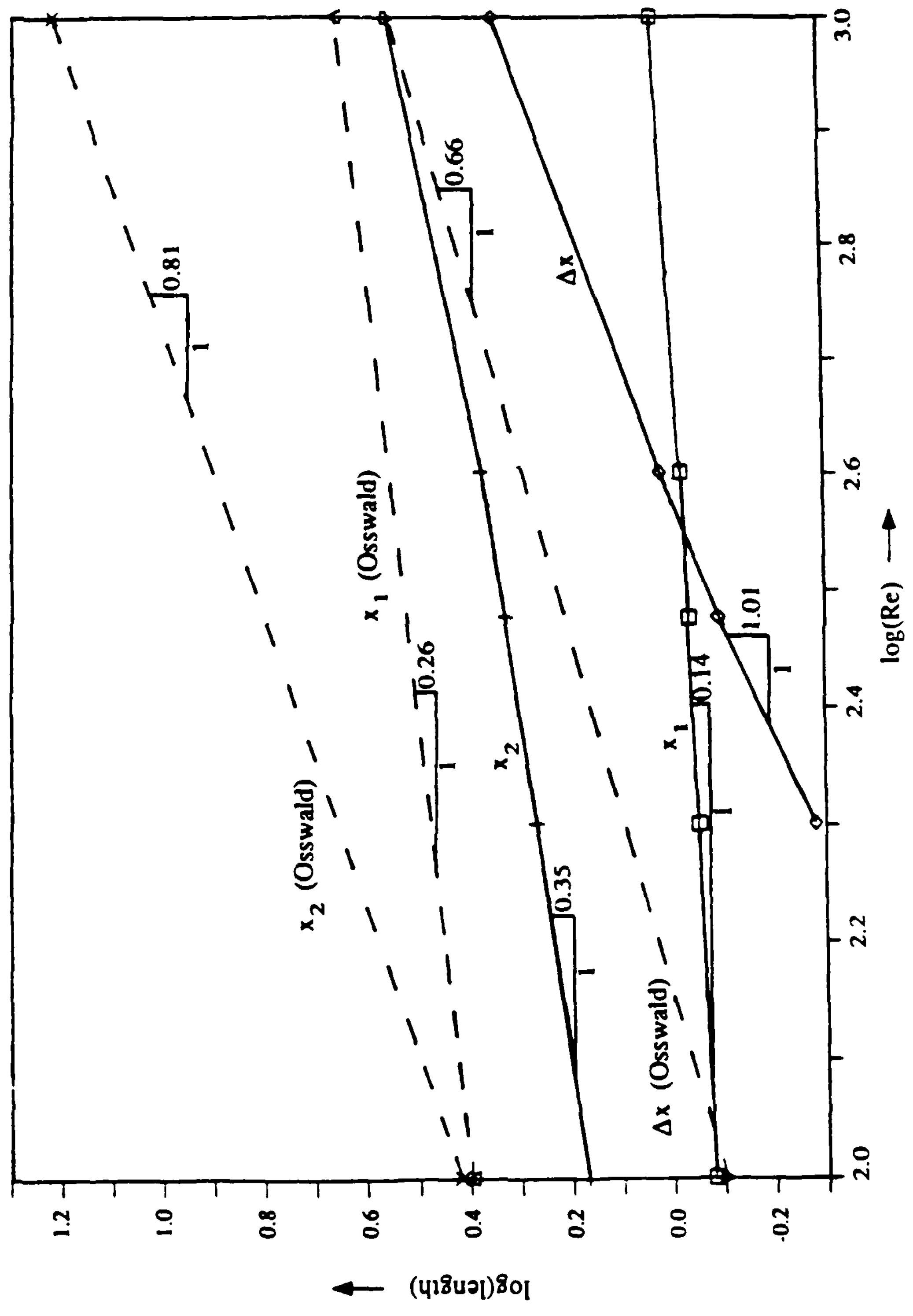


Figure 8.6: Log(characteristic lengths) versus log(Re)

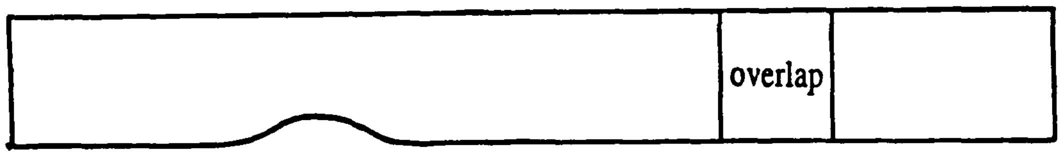


Figure 8.7: Composite domain (truncated) for asymmetrically constricted channel

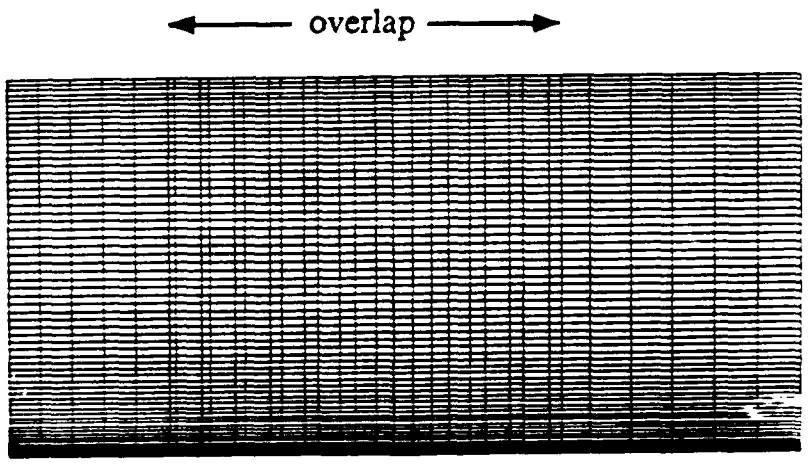
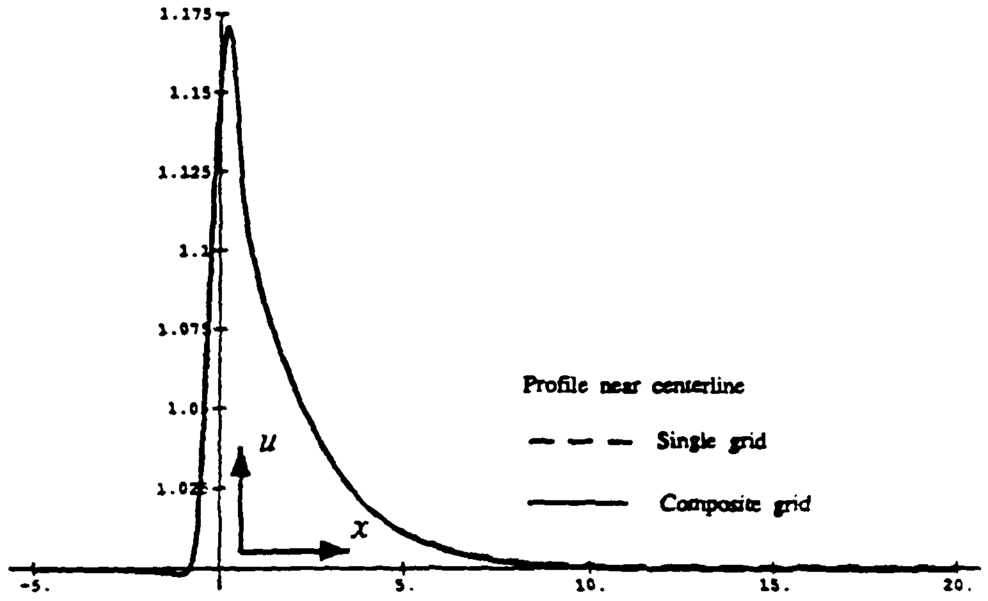


Figure 8.8: Close-up of overlap region on composite grid



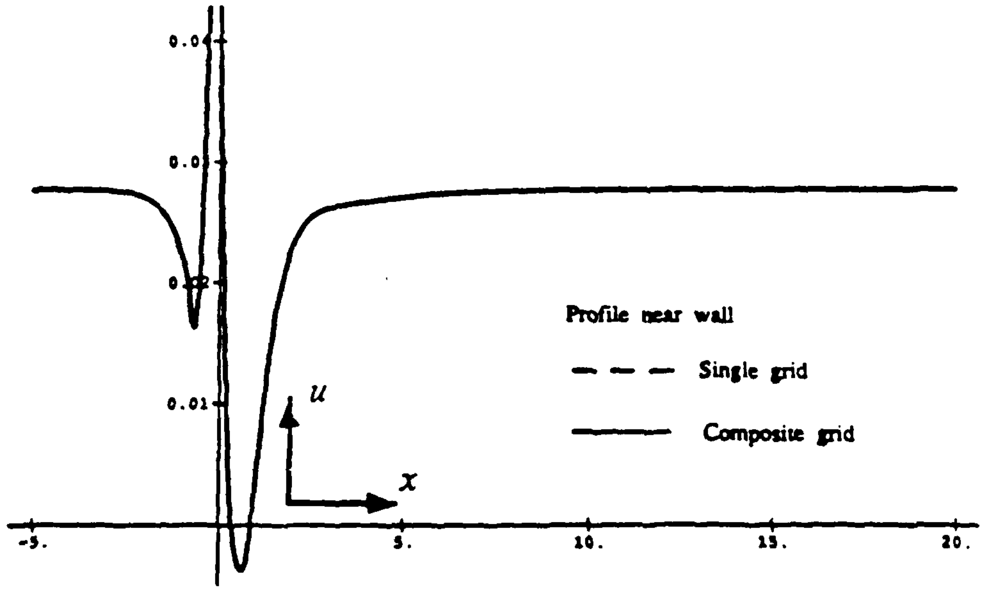
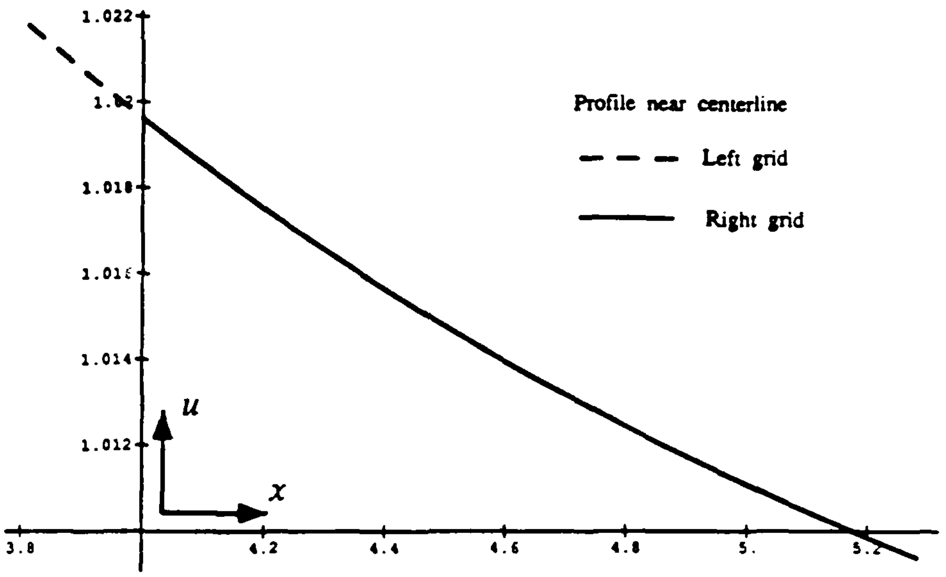


Figure 8.9: Horizontal velocities on single and composite grids



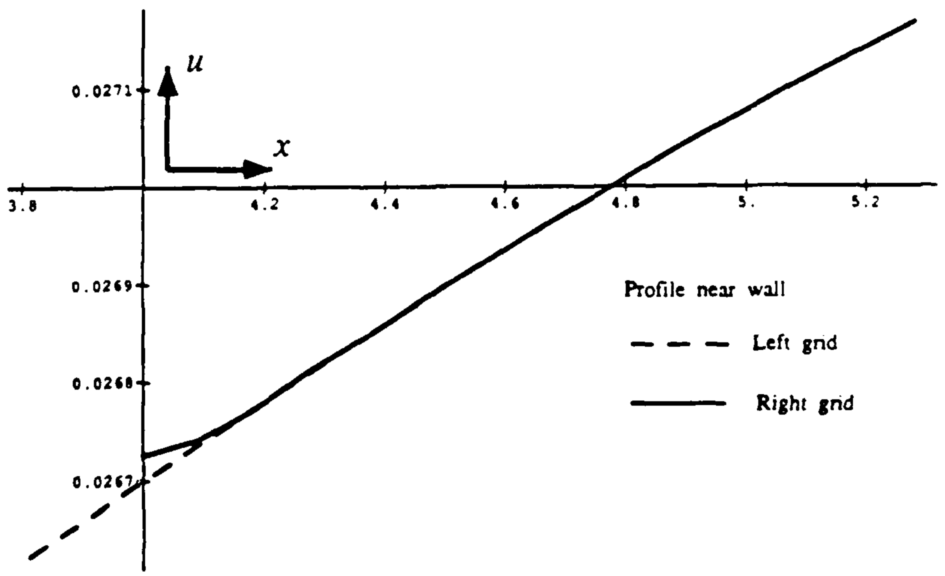


Figure 8.10: Close-ups of horizontal velocities in overlap region

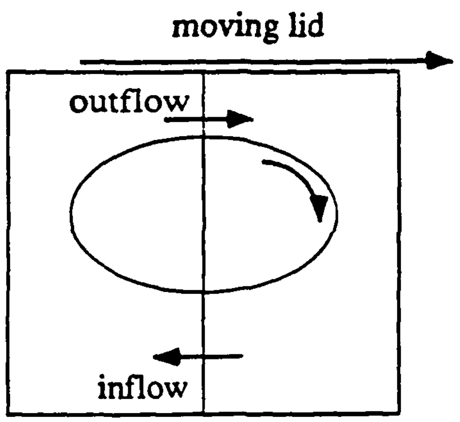


Figure 8.11: Inflow and outflow on interface boundary

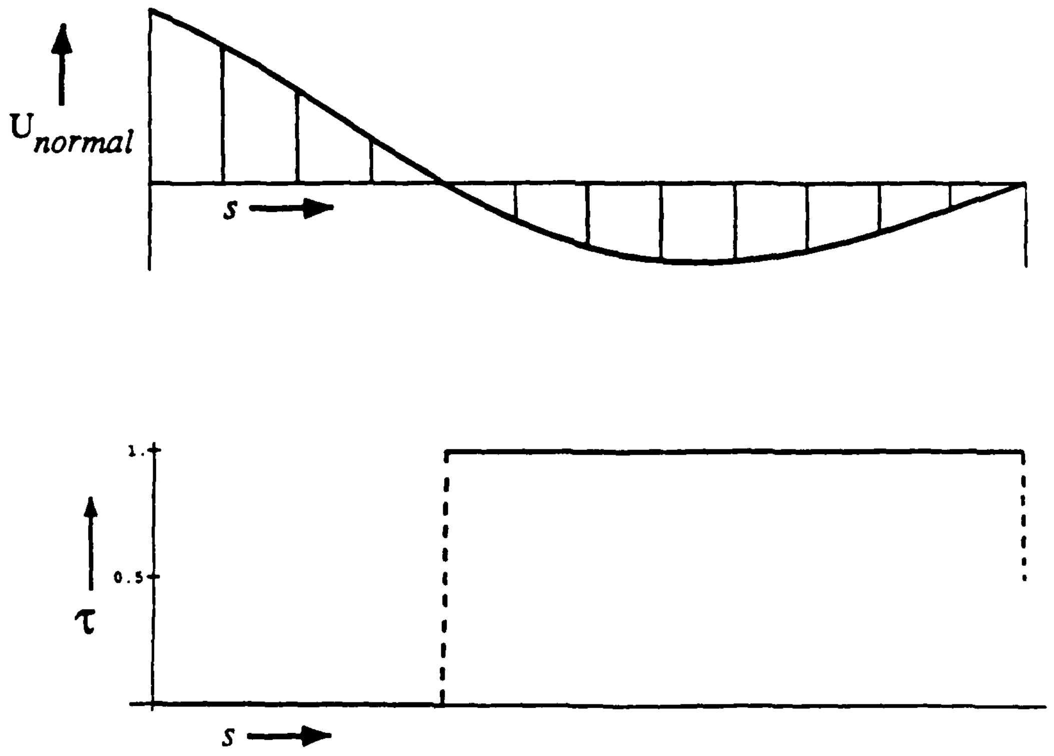


Figure 8.12: Distribution of τ along interface; discrete switch

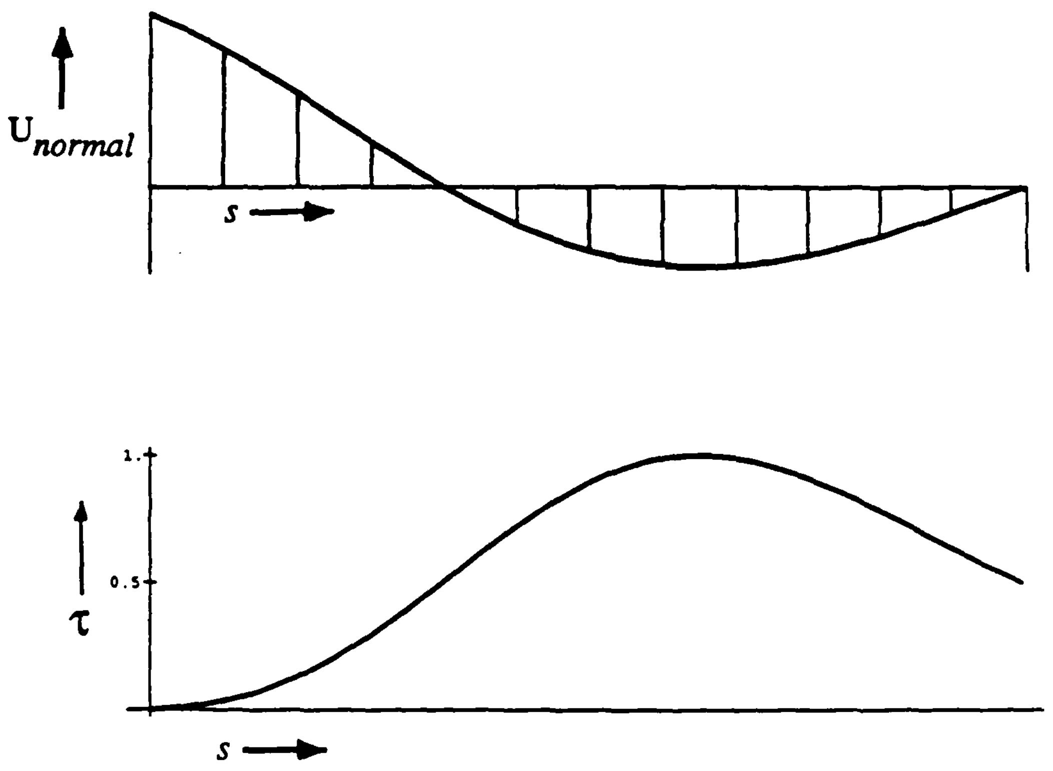


Figure 8.13: Distribution of τ along interface; smooth transition

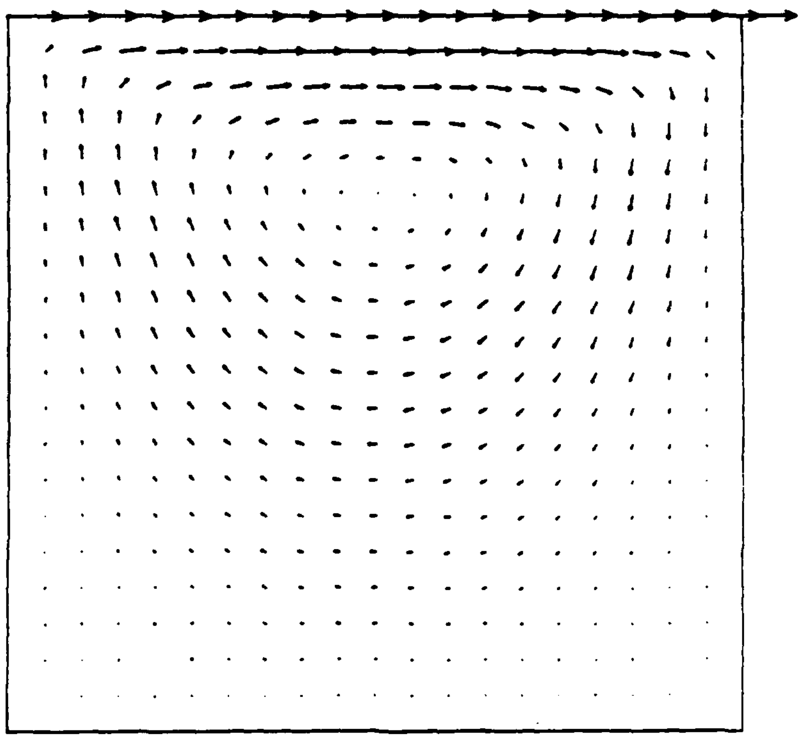


Figure 8.14: Single-grid solution for lid-driven cavity

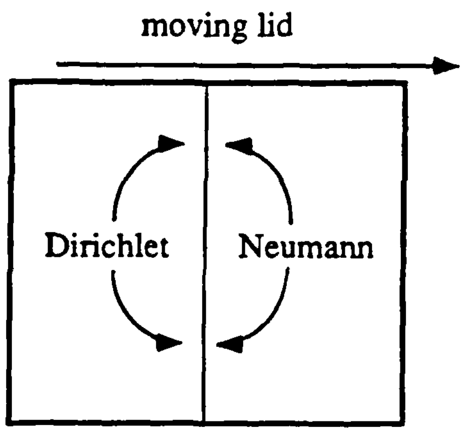


Figure 8.15: Fixed boundary conditions on both sides of interface

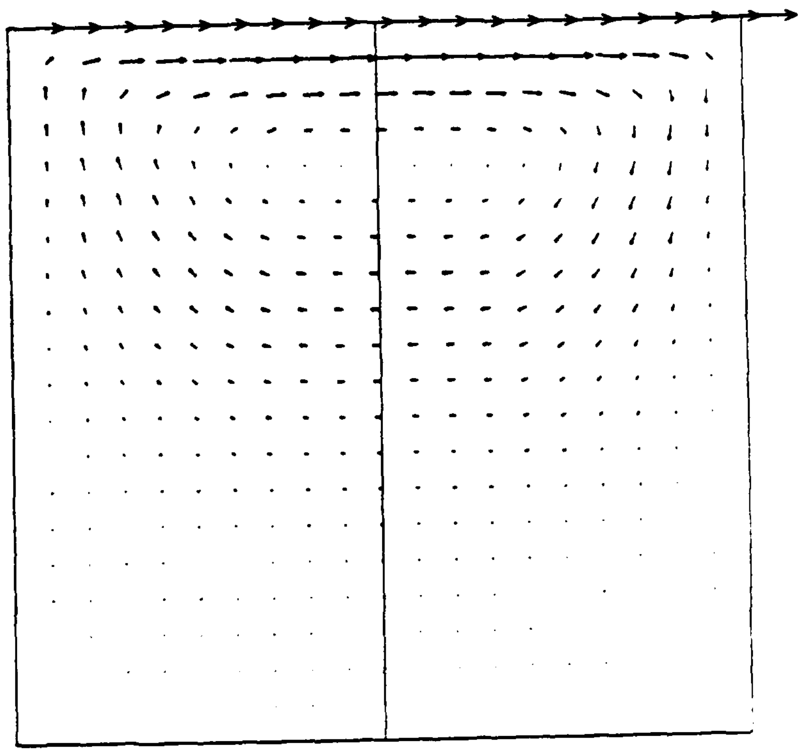


Figure 8.16: Flattened vortex for low-order approximations

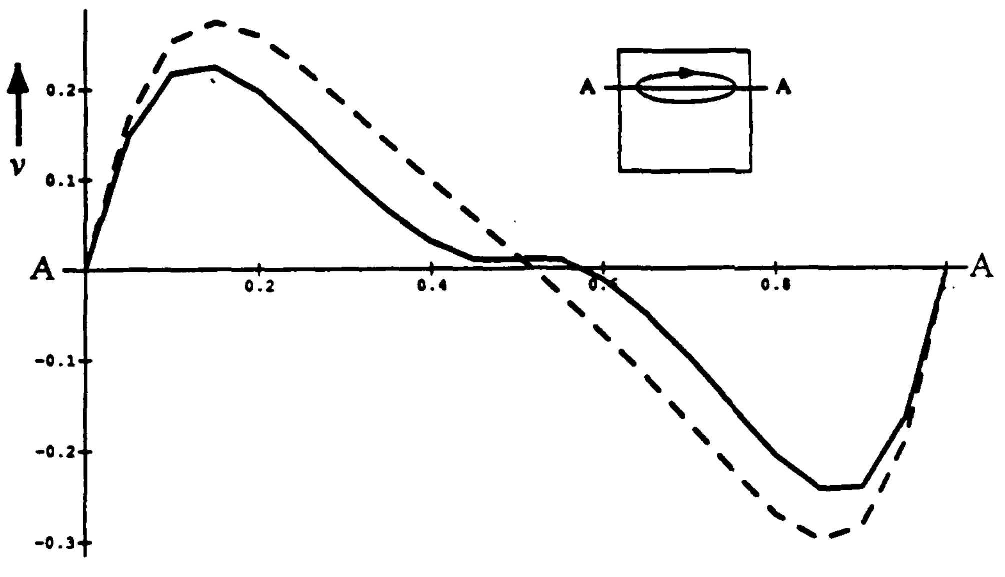


Figure 8.17: Vertical-velocity profile for low-order approximations

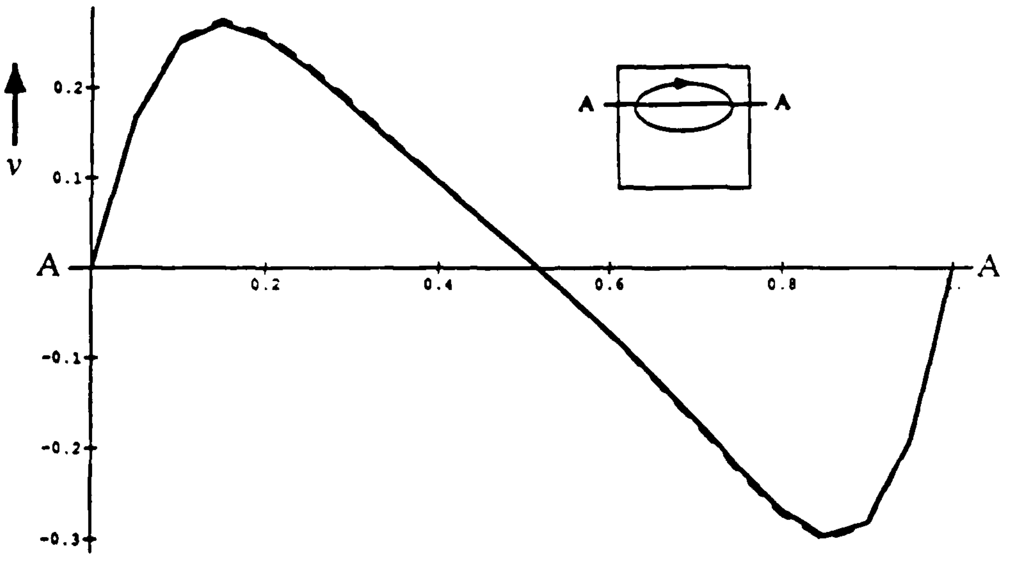


Figure 8.18: Vertical-velocity profile for high-order approximations

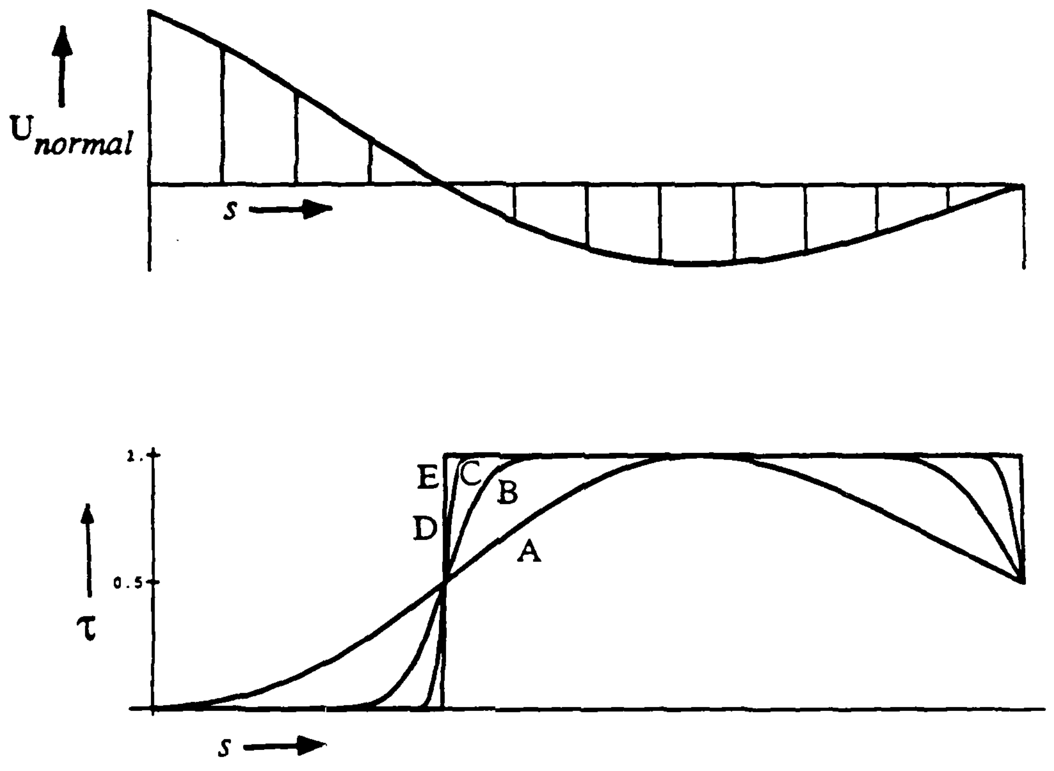


Figure 8.19: Steepened τ-response to normal velocities

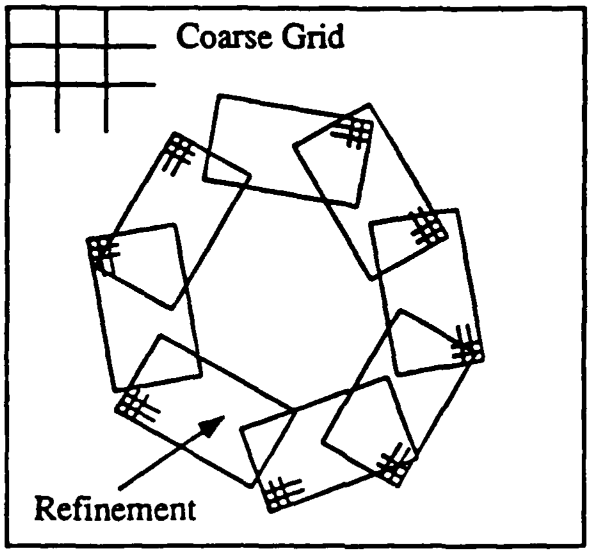


Figure 8.20: Reentrant chain of refined grids

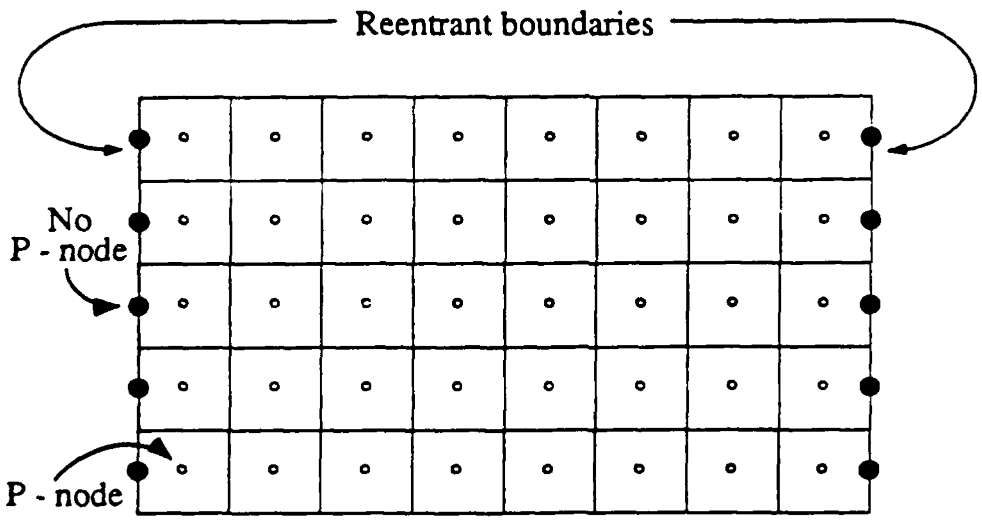


Figure 8.21: Reentrant pressure nodes lacking on staggered grid

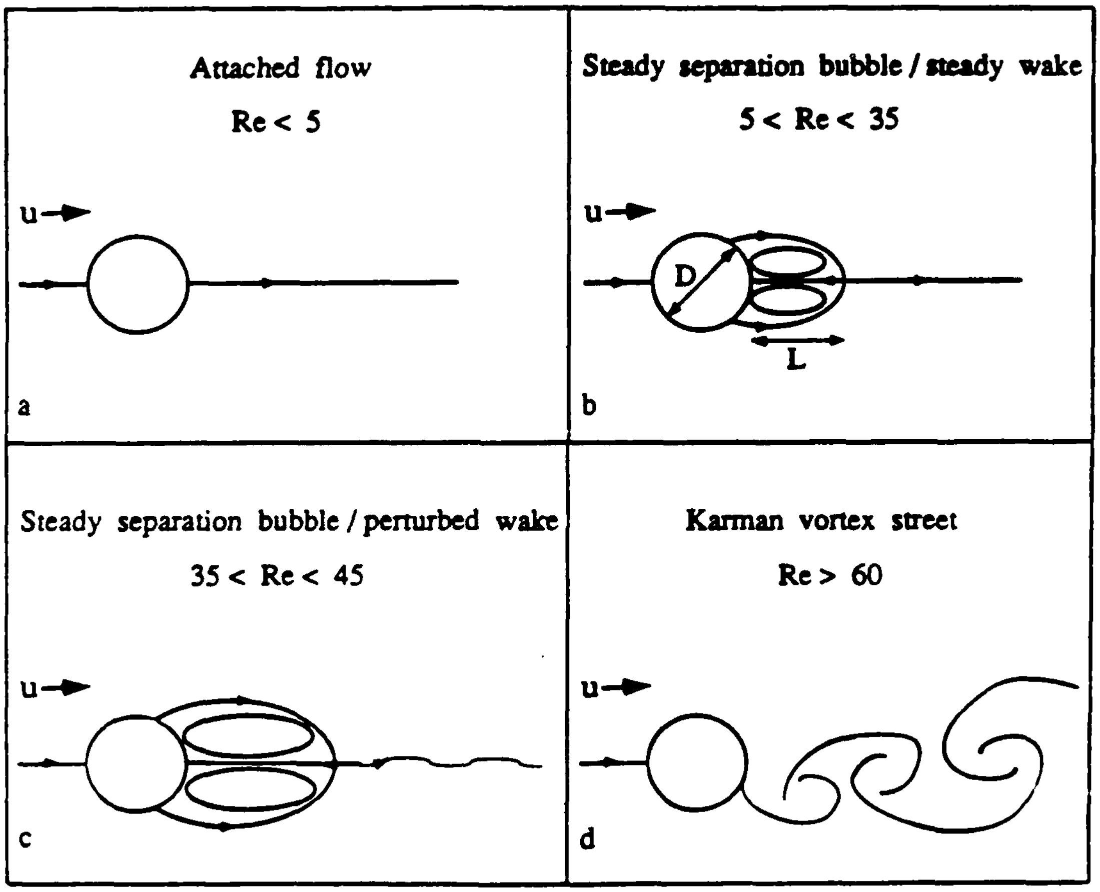

Attached flow
Re < 5
u
a
Steady separation bubble / steady wake
5 < Re < 35
u
D
L
b
Steady separation bubble / perturbed wake
35 < Re < 45
u
c
Karman vortex street
Re > 60
u
d

Figure 8.22: Laminar flow regimes of circular cylinder in crossflow

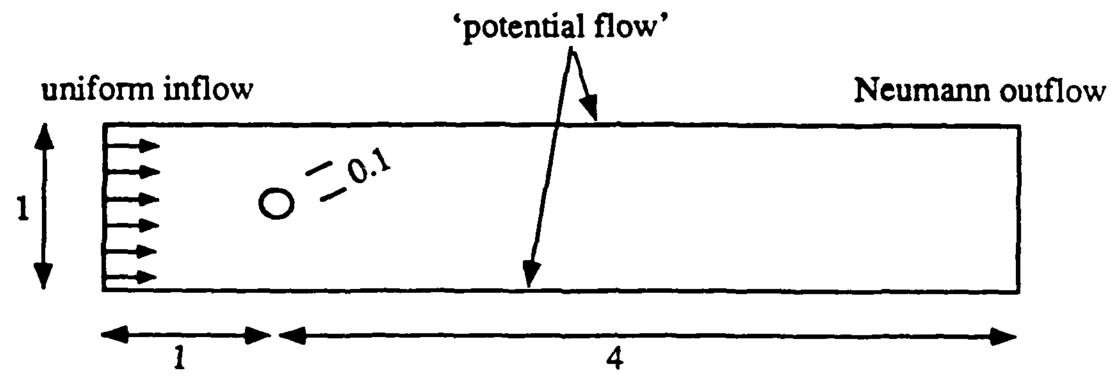


Figure 8.23: Physical domain and boundary conditions for cylinder in crossflow

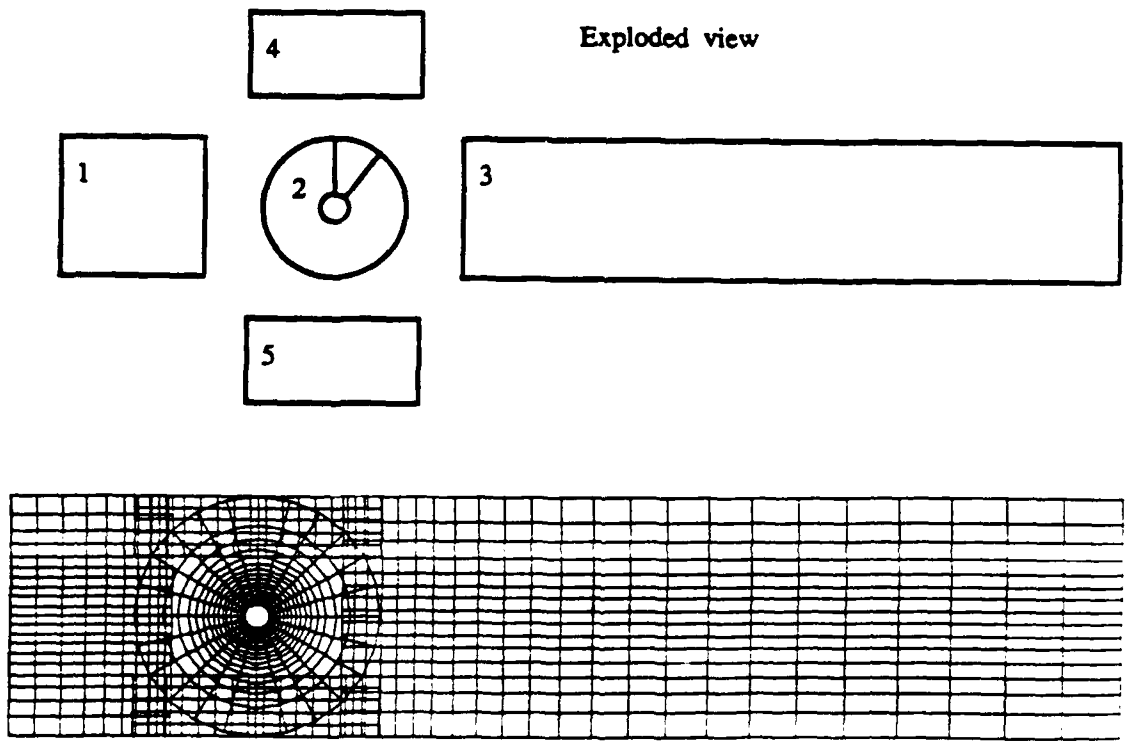


Figure 8.24: Composite grid for cylinder in crossflow (every 4th grid line shown)

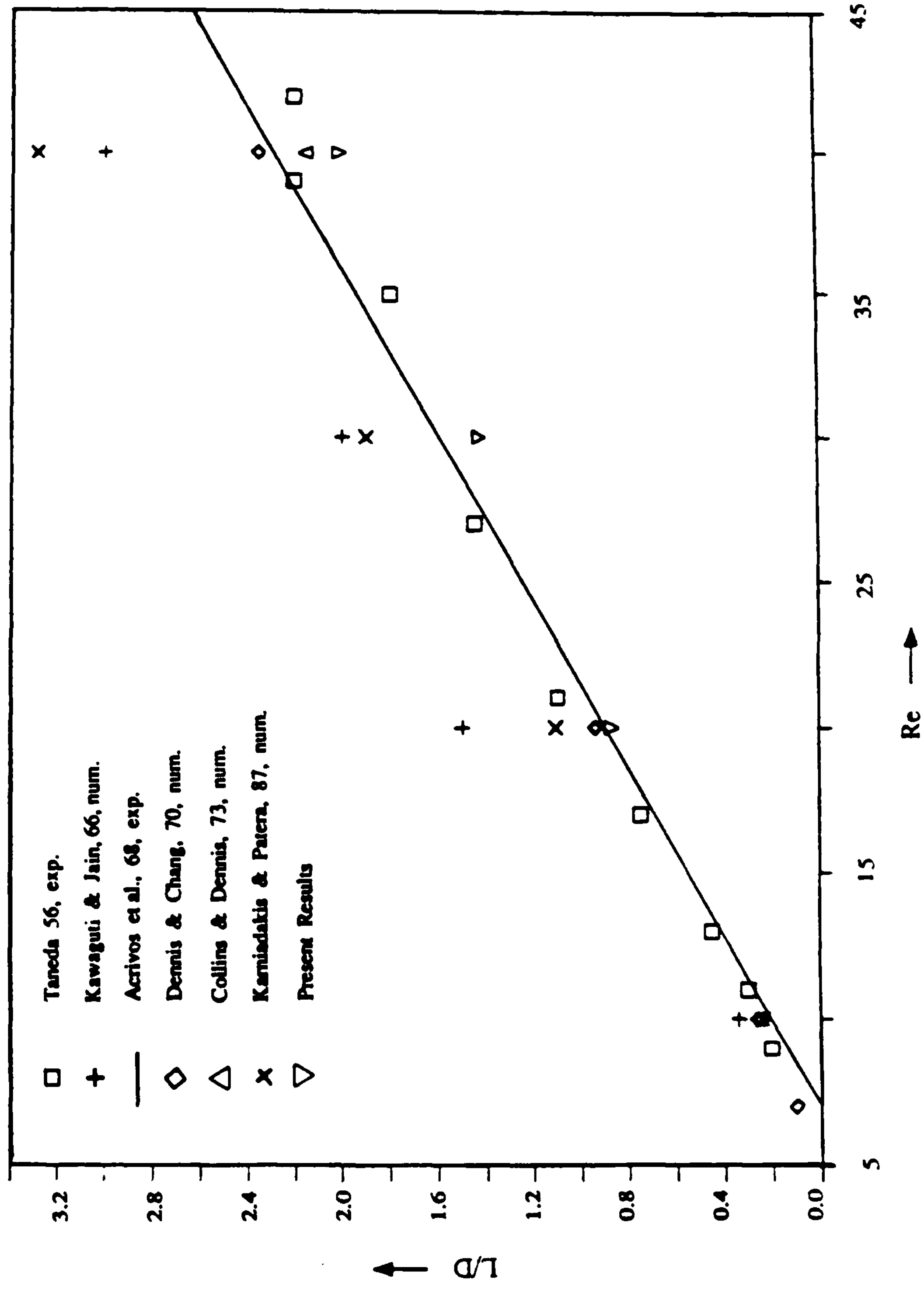


Figure 8.25: Separation-bubble length versus Reynolds number

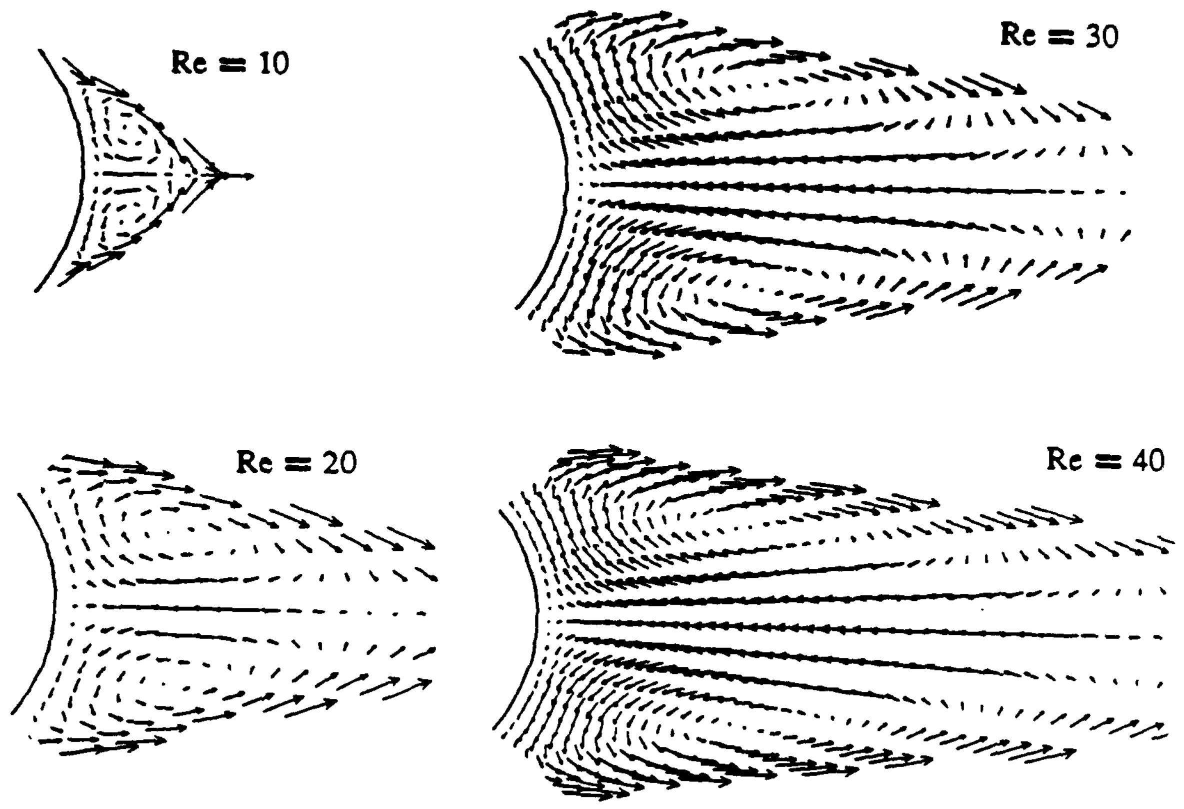


Figure 8.26: Close-ups of separation bubble behind cylinder

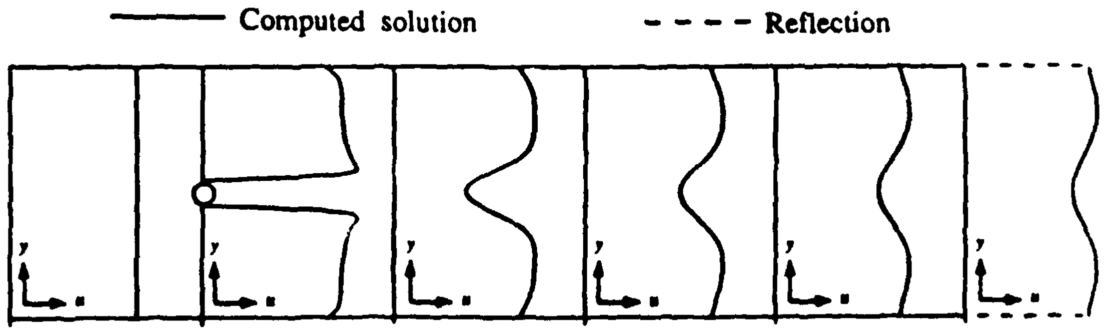


Figure 8.27: Horizontal-velocity profiles for cylinder in crossflow

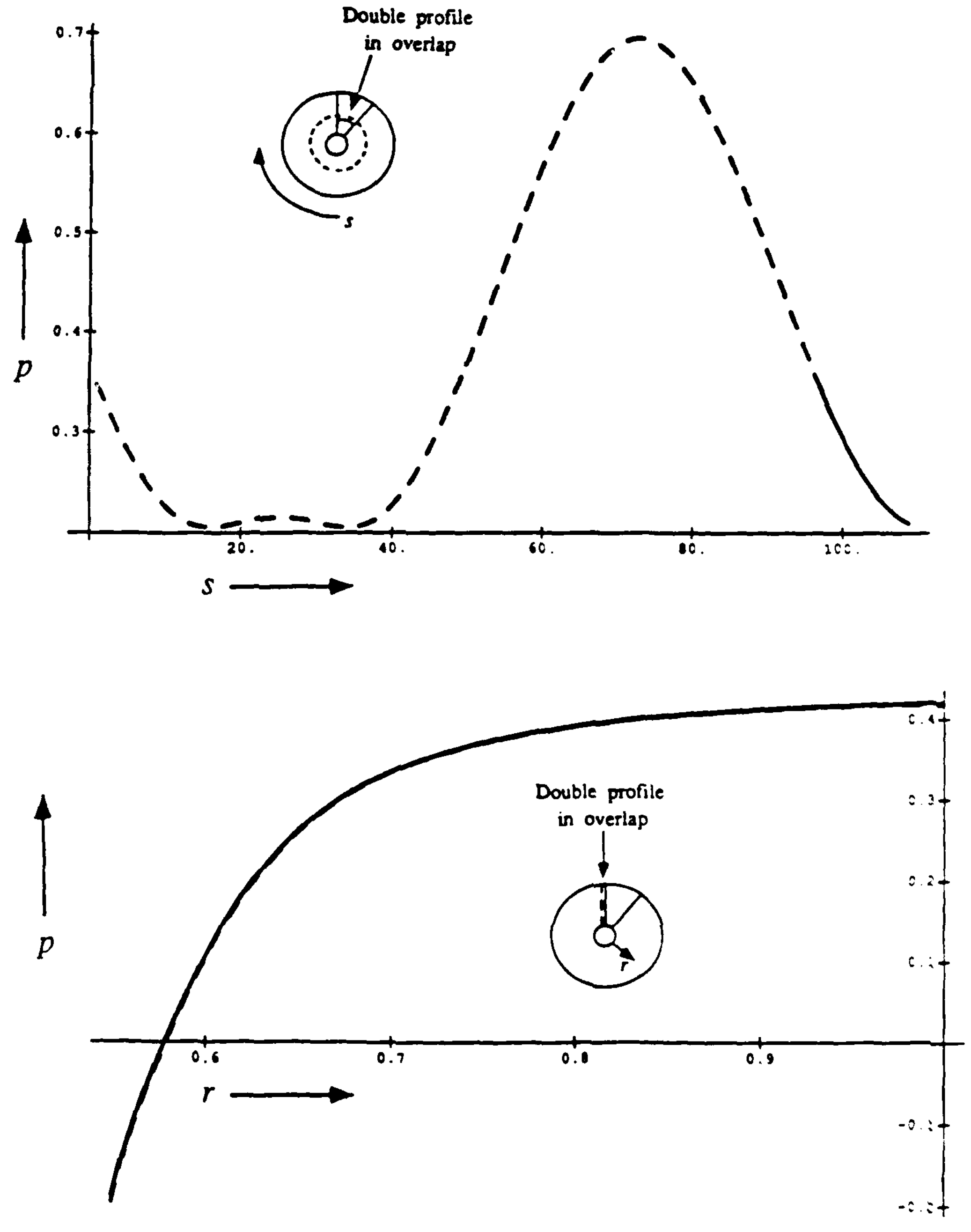


Figure 8.28: Pressure profiles on self-overlapped annular grid

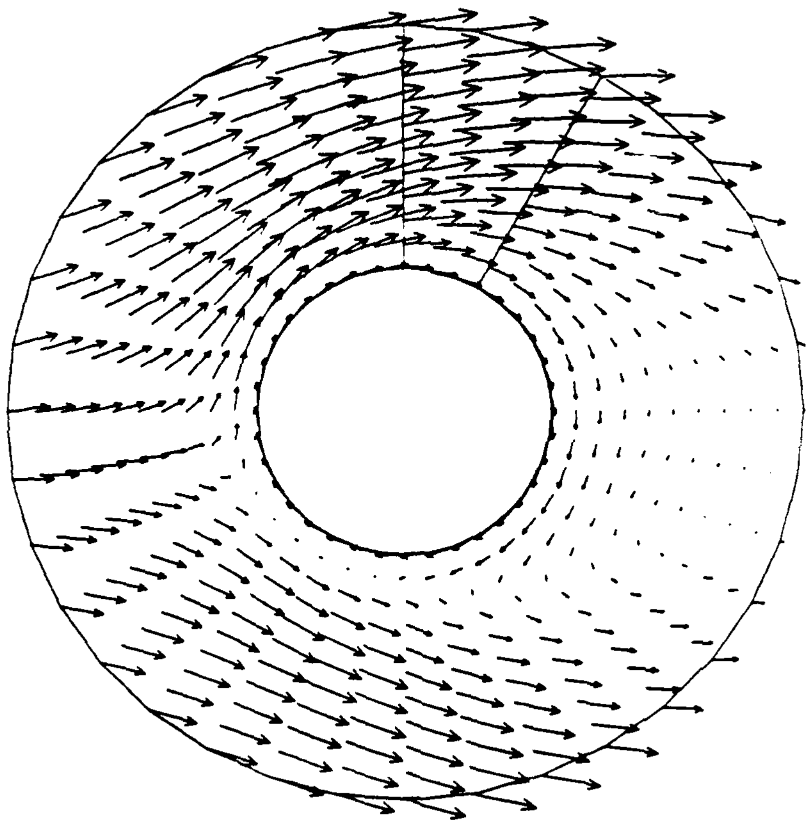


Figure 8.29: Close-up of flow field around spinning cylinder

Chapter 9

CONCLUSIONS AND RECOMMENDATIONS

9.1 Summary and Conclusions

In this study we have presented ways of computing accurate solutions to the incompressible, steady, two-dimensional Navier-Stokes equations on geometrically complex domains. The approach was to dissect the domain into pieces of simple shape that were fitted with grids independently. The result is called a composite grid. Communication between the component grids takes place through interpolation. If no special care is taken, this interpolation is not conservative, which limits the final degree of convergence that can be attained.

Different types of intergrid boundary conditions were investigated, namely Dirichlet everywhere (SWAP = SchWarz Alternating Procedure), and Dirichlet/Neumann on different sides of interfaces (SWAPR = SWAP-Revised). SWAPR offers the perspective of full conservation if no overlap between subdomains occurs, whereas SWAP always requires overlap and hence is nonconservative. However. two-dimensional Navier-Stokes computations revealed that low order of accuracy of Neumann-boundary-condition implementation and of interpolation degrades the accuracy of SWAPR results, whereas higher order of accuracy leads to poorly converging iterative procedures. Consequently, SWAP was chosen as the basic communication procedure for composite-grid calculations.

As the aim of this project was to extend the solution-adaptive mesh refinement techniques, described in [BER82], [CAR85], [SKA87], to geometrically complex domains, a scenario for combining composite grids and adaptive grids was presented. A careful analysis was made of the necessary ingredients of an adaptive-composite procedure, but no implementation was provided.

The construction of a composite grid was found to be a nontrivial task which should be automated as much as possible to unburden the user. For this purpose a grid-generation system was designed, called M*E*S*H, which has as output the complex data structures describing a composite grid. M*E*S*H itself was not implemented, but a preprocessor that 'digests' the essentially continuous M*E*S*H output was, demonstrating the effectiveness of the composite-grid data structures.

An essential component of the flow simulation program is the Navier-Stokes solution procedure for a single, curvilinear, nonorthogonal grid. Such a procedure was developed for a modified staggered grid, which has the important properties that pressure nodes are never needed on the boundary, and that the difference stencil is compact, leading to greater accuracy than the standard staggered grid. It was also shown that spurious pressure oscillations can easily be removed by an averaging or filtering method. Writing the discretized equations in matrix form led to increased insight in classical solution procedures, such as SIMPLE, SIMPLER, and PRIME. A convergence proof for the latter when $Re \to 0$ was given using a result from optimization theory. PRIME was identified as the simplest and most natural iterative method. It also was the most robust method on curved, nonuniform grids, and was therefore selected as the solution procedure for realistic composite-grid calculations. To validate the Navier-Stokes solver, the flow in a U-shaped channel was calculated, which exhibits a large turning angle. Computed results match asymptotic solutions available for the flow in the bend very well.

To test the composite-grid procedure, three different geometries were considered: an asymmetrically constricted channel, the lid-driven cavity, and the cylinder in crossflow. The asymmetrically constricted channel, inspired by work by Osswald [OSS83], was chosen because asymptotic analyses of upstream and downstream disturbances were available. It was found that the upstream length scale is proportional to $Re^{1/7}$, whereas the length of the separation region behind the constriction, which is a measure of downstream length scales, is proportional to Re. This conforms with the findings of Smith [SMI77], and partially disproves those of Kumar and Yajnik [KUM80]. Composite-grid solutions using SWAP were obtained at almost the same cost as single-grid solutions.

The lid-driven cavity was chosen because it features inflow and outflow on a single grid boundary. It was shown that a local version of SWAPR, which lets the choice of Neumann or Dirichlet boundary conditions depend on the local velocity normal to the boundary, can be made to work for nonoverlapping grids. However, its efficiency was poor. This was attributed to the reduced convergence rate of the Navier-Stokes solver used in case of higher-order-accurate Neumann boundary condition implementation. SWAP, on the other hand, yielded accurate solutions very efficiently.

Finally, the cylinder-in-crossflow problem was selected because it involves a non-simply-connected domain, thus making a composite-grid approach necessary. Moreover, the problem of inconsistent pressures on reentrant grids, arising from the lack of pressure boundary points, had to be tackled. A strategy was developed to incorporate the communication of pressure values across grids in the SWAP procedure, which eliminates discrepancies in the pressure at convergence. The results that were obtained for the length of the separation bubble in the wake of the cylinder were in good agreement with measurements and other computations. No problems were experienced with nonconservation due to interpolation.

Hence, it is concluded that accurate solutions to the steady, incompressible Navier-Stokes equations can be computed efficiently using overlapping composite grids.

9.2 Recommendations

Avenues for extension and completion of the present method abound. The designed M*E*S*H system should be built, using a novel language VORPAL (Value-ORiented ProgrAmming Language) now under development at Stanford. In addition, the strategy outlined for making the composite-grid procedure solution-adaptive should be implemented and refined. Extending the method to three space dimensions, although not difficult conceptually, leads to interesting geometrical problems which need to be addressed. In order to make SWAPR more competitive, efforts should be made to speed up convergence of the basic Navier-Stokes solver, probably by implementing a multigrid procedure. If the method is to be used for practical engineering calculations, a turbulence model should also be incorporated.

Appendix A

SAMPLE COMPOSITE-GRID DATA FILE

Here we present a data file that has been used to compute the flow in the lid-driven cavity. The composite grid has two equal-sized components that overlap by 20%. A successful computation taking 1.4 work units has been carried out using this data file.

```
                         SAMPLE DATA FILE
This file contains the information which specifies a composite grid,
consisting of two rectangular grids which together cover the unit
square. The overlap is 20 percent of the area of the square. Boundary
conditions specify cavity flow. Every line in this data file is read
by a preprocessor to the computational program, starting from "SAMPLE
DATA ...". Curve and boundary condition identification numbers refer
to subroutines in the files "stdfun" (standard functions) and "bcfun"
(boundary condition functions). Shortened versions of these files are
included in this appendix. For more complicated computations, such as
the flow around a circular cylinder, stdfun and bcfun contain many
subroutines, which makes changing geometry and boundary conditions
very easy once a basic composite grid has been defined.

Note: Keywords in this file are "Structure:" and "Entry:". They signal
      the start of new pieces of information. Other indentifiers, such
      as "End", "number", "part", or blank space (indentation) are
      ignored; they are used to make the data file more readable.


Structure: ****** COMPOSITE_GRID ******       Name: CAVITY

Structure: CURVE_SET
   entry: total number of curves in the composite grid
   6

   Structure: CURVE  (number 1)
      This curve is used for the bottoms of the two grids
      entry: type of curve
      standard
      entry: curve identification (straight line segment)
      1
      entry: 4 parameters, specifying begin- and end-point-coordinates
      4      0.0  0.0           1.0  0.0
      entry: affine transformation (identity in this case)
      1.0   0.0   0.0  1.0            0.0   0.0
      entry: total number of subcurves in the curve
      3

      Structure: SUBCURVE (number 1)
         entry: numerical role of the subcurve
```

```
        physical
        entry: boundary condition identification
        4
        entry: parameter interval subcurve occupies within curve
        0.0   0.40
    End SUBCURVE
    Structure: SUBCURVE (number 2)
        entry: numerical role of the subcurve
        physical
        entry: boundary condition identification
        4
        entry: parameter interval subcurve occupies within curve
        0.40   0.60
    End SUBCURVE
    Structure: SUBCURVE (number 3)
        entry: numerical role of the subcurve
        physical
        entry: boundary condition identification
        4
        entry: parameter interval subcurve occupies within curve
        0.60   1.0
    End SUBCURVE
End CURVE
Structure: CURVE   (number 2)
    This curve is used for the tops of the two grids
    entry: type of curve
    standard
    entry: curve identification (straight line segment)
    1
    entry: 4 parameters, specifying begin- and end-point-coordinates
    4      0.0  1.0      1.0  1.0
    entry: affine transformation (identity in this case)
    1.0   0.0   0.0   1.0           0.0   0.0
    entry: total number of subcurves in the curve
    3
    Structure: SUBCURVE (number 1)
        entry: numerical role of the subcurve
        physical
        entry: boundary condition identification
        1
        entry: parameter interval subcurve occupies within curve
        0.0   0.40
    End SUBCURVE
    Structure: SUBCURVE (number 2)
        entry: numerical role of the subcurve
        physical
        entry: boundary condition identification
        1
        entry: parameter interval subcurve occupies within curve
        0.40   0.60
    End SUBCURVE
    Structure: SUBCURVE (number 3)
        entry: numerical role of the subcurve
        physical
        entry: boundary condition identification
        1
        entry: parameter interval subcurve occupies within curve
        0.60   1.0
    End SUBCURVE
End CURVE
Structure: CURVE   (number 3)
    This curve is used for the left boundary of the left grid
    entry: type of curve
    standard
    entry: curve identification (straight line segment)
    1
    entry: 4 parameters, specifying begin- and end-point-coordinates
    4      0.0  0.0       0.0  1.0
    entry: affine transformation (identity in this case)
    1.0   0.0   0.0   1.0              0.0   0.0
    entry: total number of subcurves in the curve
    1

    Structure: SUBCURVE
       entry: numerical role of the subcurve
       physical
       entry: boundary condition identification
       4
       entry: parameter interval subcurve occupies within curve (whole )
       0.0    1.0
    End SUBCURVE
End CURVE

Structure: CURVE    ( number 4 )
    This curve is used for the right boundary of the right grid
    entry: type of curve
    standard
    entry: curve identification (straight line segment)
    1
    entry: 4 parameters, specifying begin- and end-point-coordinates
    4      1.0  0.0       1.0  1.0
    entry: affine transformation (identity in this case)
    1.0   0.0   0.0   1.0              0.0   0.0
    entry: total number of subcurves in the curve
    1

    Structure: SUBCURVE
       entry: numerical role of the subcurve
       physical
       entry: boundary condition identification
       4
       entry: parameter interval subcurve occupies within curve (whole )
       0.0    1.0
    End SUBCURVE
End CURVE

Structure: CURVE    ( number 5 )
    This curve is used for the left side of the right grid
    entry: type of curve
    standard
    entry: curve identification (straight line segment)
    1
    entry: 4 parameters, specifying begin- and end-point-coordinates
    4      0.40  0.0       0.40  1.0
    entry: affine transformation (identity in this case)
    1.0   0.0   0.0   1.0              0.0   0.0
    entry: total number of subcurves in the curve
    1

    Structure: SUBCURVE
       entry: numerical role of the subcurve
       auxiliary
       entry: parameter interval subcurve occupies within curve (whole )
       0.0    1.0
    End SUBCURVE;
End CURVE

Structure: CURVE    ( number 6 )
    This curve is used for the right side of the left grid
    entry: type of curve
    standard
    entry: curve identification (straight line segment)
    1
    entry: 4 parameters, specifying begin- and end-point-coordinates
    4      0.60  0.0       0.60  1.0
    entry: affine transformation (identity in this case)
    1.0   0.0   0.0   1.0              0.0   0.0
```

```
        entry: total number of subcurves in the curve
        1

        Structure: SUBCURVE
           entry: numerical role of the subcurve
           auxiliary
           entry: parameter interval subcurve occupies within curve (whole )
           0.0  1.0
        End SUBCURVE;
     End CURVE
  End CURVE_SET

  Structure: GRID_SET
     entry: type of SWAPR procedure (discrete switch)
     1
     entry: total number of grids in the composite grid
     2

     Structure: GRID (number 1)
        This is the left one of the two grids in the system
        entry: status of the grid
        consistent
        entry: geometrical type of the grid
        quadrilateral
        entry: number of sides in the grid
        4

        Structure: SIDE (number 1)
           This is the bottom side of the left grid
           entry: number of subcurves composing the side
           2

           entry: parameter interval of FIRST subcurve within side
           0.0   0.6666666666
           entry: number of curve on which the subcurve resides
           1
           entry: sequence number of subcurve within this curve
           1

           entry: parameter interval of SECOND subcurve within side
           0.6666666666    1.0
           entry: number of curve on which the subcurve resides
           1
           entry: sequence number of subcurve within this curve
           2
           entry: number of subsides in the side
           1

           Structure: SUBSIDE
              entry: numerical role of the subside
              physical
              entry: parameter interval of subside within the side (whole side)
              0.0 1.0
           End SUBSIDE
        End SIDE

        Structure: SIDE (number 2)
           This is the top side of the left grid
           entry: number of subcurves composing the side
           2

           entry: parameter interval of FIRST subcurve within side
           0.0   0.6666666666
           entry: number of curve on which the subcurve resides
           2
           entry: sequence number of subcurve within this curve
           1

           entry: parameter interval of SECOND subcurve within side
           0.6666666666    1.0
           entry: number of curve on which the subcurve resides
```

```
      2
      entry: sequence number of subcurve within this curve
      2
      entry: number of subsides in the side
      1

      Structure: SUBSIDE
         entry: numerical role of the subside
         physical
         entry: parameter interval of subside within the side (whole side)
         0.0  1.0
      End SUBSIDE
   End SIDE

   Structure: SIDE (number 3)
      This is the left side of the left grid
      entry: number of subcurves composing the side
      1

      entry: parameter interval of subcurve within side (whole side)
      0.0   1.0
      entry: number of curve on which the subcurve resides
      3
      entry: sequence number of subcurve within this curve
      1
      entry: number of subsides in the side
      1

      Structure: SUBSIDE
         entry: numerical role of the subside
         physical
         entry: parameter interval of subside within the side (whole side)
         0.0   1.0
      End SUBSIDE
   End SIDE

   Structure: SIDE (number 4)
      This is the right side of the left grid
      entry: number of subcurves composing the side
      1

      entry: parameter interval of subcurve within side (whole side)
      0.0   1.0
      entry: number of curve on which the subcurve resides
      6
      entry: sequence number of subcurve within this curve
      1
      entry: number of subsides in the side
      1

      Structure: SUBSIDE
         entry: numerical role of the subside
         interpolation
         entry: number of candidate donor grids
         1
         entry: list of candidate donor grids + geometrical connections
         2   overlap
         entry: parameter interval of subside within the side (whole side)
         0.0  1.0
      End SUBSIDE
   End SIDE

   entry: number of grid cells in both directions
   12  20
   entry: stretching direction
   none
   The grid is shrunk towards the lid in order to capture steep gradients
   entry: shrinking direction
   eta
   entry: shrinking parameters
   2.0   1.0
   entry: type of grid mapping (transfinite)
```

```
    transfin
    entry: order of transfinite interpolation (Coons Patch)
    1
End GRID

Structure: GRID (number 2)
    This is the right one of the two grids in the system
    entry: status of the grid
    consistent
    entry: geometrical type of the grid
    quadrilateral
    entry: number of sides in the grid
    4

    Structure: SIDE (number 1)
        This is the bottom side of the right grid
        entry: number of subcurves composing the side
        2

        entry: parameter interval of FIRST subcurve within side
        0.0   0.3333333333
        entry: number of curve on which the subcurve resides
        1
        entry: sequence number of subcurve within this curve
        2

        entry: parameter interval of SECOND subcurve within side
        0.3333333333   1.0
        entry: number of curve on which the subcurve resides
        1
        entry: sequence number of subcurve within this curve
        3
        entry: number of subsides in the side
        1

        Structure: SUBSIDE
            entry: numerical role of the subside
            physical
            entry: parameter interval of subside within the side (whole side)
            0.0  1.0
        End SUBSIDE
    End SIDE

    Structure: SIDE (number 2)
        This is the top side of the right grid
        entry: number of subcurves composing the side
        2

        entry: parameter interval of FIRST subcurve within side
        0.0   0.3333333333
        entry: number of curve on which the subcurve resides
        2
        entry: sequence number of subcurve within this curve
        2

        entry: parameter interval of SECOND subcurve within side
        0.3333333333  1.0
        entry: number of curve on which the subcurve resides
        2
        entry: sequence number of subcurve within this curve
        3
        entry: number of subsides in the side
        1

        Structure: SUBSIDE
            entry: numerical role of the subside
            physical
            entry: parameter interval of subside within the side (whole side)
            0.0  1.0
        End SUBSIDE
    End SIDE
```

```
      Structure: SIDE (number 3)
         This is the left side of the right grid
         entry: number of subcurves composing the side
         1

         entry: parameter interval of subcurve within side (whole side)
         0.0   1.0
         entry: number of curve on which the subcurve resides
         5
         entry: sequence number of subcurve within this curve
         1
         entry: number of subsides in the side
         1

         Structure: SUBSIDE
            entry: numerical role of the subside
            interpolation
            entry: number of candidate donor grids
            1
            entry: list of candidate donor grids + geometrical connections
            1   overlap
            entry: parameter interval of subside within the side (whole side)
            0.0  1.0
         End SUBSIDE
      End SIDE
      Structure: SIDE (number 4)
         This is the right side of the right grid
         entry: number of subcurves composing the side
         1

         entry: parameter interval of subcurve within side (whole side)
         0.0   1.0
         entry: number of curve on which the subcurve resides
         4
         entry: sequence number of subcurve within this curve
         1
         entry: number of subsides in the side
         1

         Structure: SUBSIDE
            entry: numerical role of the subside
            physical
            entry: parameter interval of subside within the side (whole side)
            0.0  1.0
         End SUBSIDE
      End SIDE
      entry: number of grid cells in both directions
      12  20
      entry: stretching direction
      none
      The grid is shrunk towards the lid in order to capture steep gradients
      entry: shrinking direction
      eta
      entry: shrinking parameters
      2.0   1.0
      entry: type of grid mapping (transfinite)
      transfin
      entry: order of transfinite interpolation (Coons Patch)
      1
   End GRID
End GRID_SET
Structure: GLOBAL_ITERATION
   entry: type of iteration
   manual
   entry: length of iteration chain, plus actual chain
   2          1  2
End GLOBAL_ITERATION
End COMPOSITE_GRID
```

Next follow the files 'stdfun' and 'bcfun' that contain functions specifying standard curves and boundary conditions, respectively. Use has been made of the Fortran 'computed-goto' statement to obtain an easily extendable structure.

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C                                                                      C
C                         SUBROUTINE  STDFUN                           C
C----------------------------------------------------------------------C
C                                                                      C
C  This subroutine contains the descriptions of standard curves with which  C
C  a set of parameters must be supplied.                               C
C                                                                      C
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine STDFUN( s, id, cparm, flag, x, y, dxds, dyds, maxCPR )

      integer maxCPR, id, flag
      real*8 s, cparm(maxCPR), x, y, dxds, dyds
c     s      - variable  running along the curve
c     id     - curve identification number
c     cparm  - array containing curve parameters
c     flag   - if 1 --> return coordinates on the curve
c              if 2 --> return tangential derivatives
c     x, y   - Cartesian coordinates
c     dx/yds - tangential derivatives
c     maxCPR - maximum number of curve parameters

c     Jump to the label corresponding to the curve identification number.
c     One danger exists with the following computed goto statement: if the
c     argument "id" is not in the proper range of integers from 1 to 9
c     the first in the list of labels is used. Therefore we need to do an
c     additional check when a jump to label 1 occurs.

      go to ( 1, 2, 3, 4, 5, 6, 7, 8, 9 ) id

c     When a jump to label 1 occurs, we first check if the original
c     identification number was indeed 1. This is done most easily by
c     adding 1 to id and jumping to a label in another  computed goto
c     list. If id was 1, then the jump will take place to the second
c     label  in the list because of the addition of 1. If id was any-
c     thing else than one, then a jump will occur to the first label
c     in the list, upon which the program will be terminated
 1    goto (200,300) id+1
 200  print *, 'Wrong standard curve identification number: id =', id
      stop

 300  call ONE( s, cparm, flag, x, y, dxds, dyds, maxCPR )
      return

 2    call TWO( s, cparm, flag, x, y, dxds, dyds, maxCPR )
      return

 3    continue
 4    continue
 5    continue
 6    continue
 7    continue
 8    continue
 9    continue

      print *, 'Illegal standard curve identification number', id

      stop
      end

      subroutine ONE( s, cparm, flag, x, y, dxds, dyds, maxCPR )
      integer maxCPR, flag
      real*8 s, cparm(maxCPR), x, y, dxds, dyds, delx, dely
c     This routine creates a line segment from (x1,y1) to (x2,y2)
c     PARAMETER DESCRIPTION
c      1  -> x1
c      2  -> y1
c      3  -> x2
c      4  -> y2
      delx = cparm(3) - cparm(1)
      dely = cparm(4) - cparm(2)
      if ( flag .eq. 1 ) then
         x = cparm(1)  + s * delx
         y = cparm(2)  + s * dely
      else
         dxds = delx
         dyds = dely
      endif
      return
      end


      subroutine TWO( s, cparm, flag, x, y, dxds, dyds, maxCPR )
      integer maxCPR, flag
      real*8 s, cparm(maxCPR), x, y, dxds, dyds, delalp, theta,
     $       conv
c     This subroutine creates a circle segment with an arc ranging from
c     alpha1 to alpha2 and a radius of R. The center is (xc,yc). When
c     the parameter 'degree' is larger than zero, all angles are
c     interpreted in degrees. Otherwise, radians are used. For this
c     purpose we use a conversion factor 'conv' from degrees to
c     radians (i.e. (pi/4) / 45), or from radians to radians (i.e. 1)
c     PARAMETER DESCRIPTION
c      1  -> alpha1
c      2  -> alpha2
c      3  -> xc
c      4  -> yc
c      5  -> R
c      6  -> degree
      if ( cparm(6) .gt. 0.d0 ) then
         conv = datan(1.d0) / 45.d0
      else
         conv = 1.d0
      endif
      delalp =   cparm(2) - cparm(1)
      theta =  ( cparm(1)  + s * delalp  ) * conv
      if ( flag .eq. 1 ) then
         x = cparm(3) + cparm(5) * dcos(theta)
         y = cparm(4) + cparm(5) * dsin(theta)
      else
         dxds = -cparm(5) * dsin(theta) * delalp
         dyds =  cparm(5) * dcos(theta) * delalp
      endif
      return
      end
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C                                                                      C
C                          SUBROUTINE  BCFUN                           C
C----------------------------------------------------------------------C
C                                                                      C
C  This subroutine contains the specifications of the boundary conditions  C
C  for all the subcurves that are on a physical boundary of the domain and  C
C  that are not on periodical boundaries.                              C
C                                                                      C
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        subroutine  BCFUN( s, x, y, id, Abc, Bbc, Cbc, Fbc )
        real*8 s, Abc(2,2), Bbc(2,2), Cbc(2,2), Fbc(2), x, y
        integer id
c       s       - variable running along curve
c       x,y     - cartesian coordintes on curve
c       id      - boundary condition identification number
c       Abc, Bbc, Cbc, Fbc - coefficients of linear boundary condition:
c
c            Abc u  + Bbc du/d(xsi) + Cbc du/d(eta) = Fbc

c       Jump to the label corresponding to the boundary condition (bc)
c       identification number.
c       One danger exists with the following computed goto statement: if the
c       argument "id" is not in the proper range of integers from 1 to 9
c       the first in the list of labels is used. Therefore we need to do an
c       additional check when a jump to label 1 occurs.
        goto ( 1, 2, 3, 4, 5, 6, 7, 8, 9 ) id
c       When a jump to label 1 occurs, we first check if the original
c       identification number was indeed 1. This is done most easily by
c       adding 1 to id and jumping to a label in another  computed goto
c       list. If id was 1, then the jump will take place to the second
c       label  in the list because of the addition of 1. If id was any-
c       thing else than one, then a jump will occur to the first label
c       in the list, upon which the program will be terminated
 1      goto (200,300) id+1
 200    print *, 'Wrong bc identification number: id =', id
        stop
 300    call  bONE ( s, x, y, Abc, Bbc, Cbc, Fbc )
        return
 2      call  bTWO ( s, x, y, Abc, Bbc, Cbc, Fbc )
        return
 3      continue
 4      continue
 5      continue
 6      continue
 7      continue
 8      continue
 9      continue
        print *, 'Illegal bc Identification number', id
        stop
        end

        subroutine bONE( s, x, y, Abc, Bbc, Cbc, Fbc )
        real*8 s, Abc(2,2), Bbc(2,2), Cbc(2,2), Fbc(2), x, y
        integer i, j
c       This routine sets up Dirichlet no-slip conditions
        do 10    i = 1, 2
```

```
             do 20    j = 1, 2
                Abc(i,j) = 0.d0
                Bbc(i,j) = 0.d0
                Cbc(i,j) = 0.d0
 20          continue
 10       continue

          Abc(1,1) = 1.d0
          Abc(2,2) = 1.d0
          Fbc(1)   = 0.d0
          Fbc(2)   = 0.d0

          return
          end

          subroutine bTWO( s, x, y, Abc, Bbc, Cbc, Fbc )

          real*8 s, Abc(2,2), Bbc(2,2), Cbc(2,2), Fbc(2), x, y
          integer i, j

c         This routine sets up Dirichlet uniform, horizontal flow
c         conditions

          do 10    i = 1, 2
             do 20    j = 1, 2
                Abc(i,j) = 0.d0
                Bbc(i,j) = 0.d0
                Cbc(i,j) = 0.d0
 20          continue
 10       continue

          Abc(1,1) =  1.d0
          Abc(2,2) =  1.d0
          Fbc(1)   =  1.d0
          Fbc(2)   =  0.d0

          return
          end
```

Appendix B

M*E*S*H SYSTEM FUNCTIONS

In this appenix the M*E*S*H system functions are presented. Starting from a *root* mode, we describe all the successive modes, which are arranged in a tree structure. Tools and commands are discussed within their pertinent modes. The nesting of modes, reminiscent of the UNIX directory structure, is indicated typographically in UNIX style, *e.g.*, /MESH/CURVE/COPY signifies that COPY is a submode of CURVE, which in turn is contained in MESH.

As *ZOOM* and SHOW can be invoked at any point in the system, they will not be mentioned in any specific mode, but separately at the end of the list of system functions. An overview of all the system functions is given on the last page of this appendix.

MESH

The root mode in which we are placed upon entering the program is called MESH. No commands may be given and no tools can be invoked.

Submodes: CURVE, GRID, OBJECT, ORDER, FILE

MESH/CURVE

When operating on a specific curve, this curve first has to be identified. During creation (see DEFINE) every curve receives a sequence number, and (optionally) a user-supplied name. Thus, when M*E*S*H is asked to perform an operation on a particular curve and replies by requesting identification of that curve, the user may type its name or sequence number. Otherwise, the mouse shou[illegible]u to the pertinent curve (proximity test).

Submodes: DEFINE, MODIFY, MOVE, COPY, SUBCURVE

Commands: `CLEAR`, `DELETE`

`CLEAR`	Remove the subdivision of the curve into subcurves (leaving only one subcurve corresponding to the whole curve intact, which is assigned the role '*auxiliary*')
`DELETE`	Remove the curve from the database

MESH/CURVE/DEFINE

Submodes: STANDARD, USERDEF

Tools: *HERMITE, BEZIER, SPLINE*

HERMITE	Define an Hermitian-interpolation polynomial, using two points, two slopes, and two tuning parameters
BEZIER	Define a Bézier interpolation polynomial
SPLINE	Fit a spline function through a set of points

MESH/CURVE/DEFINE/USERDEF

This mode is used to select curves that have been specified by the user on a separate file (user-defined functions). The user may request an overview of the available functions.

MESH/CURVE/DEFINE/STANDARD

This mode is used to select curves using standard functions whose parameters are to be specified. Although the parameters of the functions are stored internally in a fixed format, such as center, radius, and arc of a circular segment, the user can specify them in a number of ways. In the above example, one may wish to supply the center of the circle plus the two points on the circle that bound the arc. This information is translated by the system into the standard representation. Only a small number of conceivable standard functions is listed here. The user may request an overview of the available functions.

Tools: *STRAIGHT, CONICAL, POLYNOM*

STRAIGHT	Construct a straight line segment
CONICAL	Construct a conical section (circle, ellips, hyperbola, parabola)
POLYNOM	Construct a polynomial curve

MESH/CURVE/MODIFY

This mode is used to effect certain changes to curves that are not within the MOVE category, and which do not necessitate the redefinition of curves.

Tools: *PARM, TRIM*

PARM	Change the numerical parameters of a certain curve
TRIM	Trim a curve by restricting its running parameter

MESH/CURVE/COPY (MOVE)

This mode is used to define new curves by copying old ones and subjecting them to an affine transformation (note: an affine transformation $(P, \mathbf{q})$ maps points

$\mathbf{r} = \binom{x}{y}$ into $\tilde{\mathbf{r}} = \widetilde{\binom{x}{y}}$ through $\tilde{\mathbf{r}} = P\mathbf{r} + \mathbf{q}$. P is a (2x2)-matrix and $\mathbf{q}$ is a (2x1)-column vector. Together they are defined by six constants).
MOVE: Same as COPY, but remove old curve after construction of the new curve.

Tools: *FIT, SHIFT, ROTATE, SHRINK, MIRROR, AFFINE*

FIT	Fit a curve between two given points by restricted affine transformation (uniform shrinkage, then rotation, and then translation, *i.e.*, a similarity transformation)
SHIFT	Simple parallel displacement
ROTATE	Rotate through a given angle about a specified center of rotation
SHRINK	Multiply with respect to a given geometrical center
MIRROR	Reflect in a given line
AFFINE	Specify the six parameters of a general affine transformation explicitly

---

MESH/CURVE/SUBCURVE

Within a curve we distinguish subcurves, which are sections of the curve. One subcurve with role NONE is associated with every curve by default. A meaningful subdivision into subcurves is brought about by manipulations of endpoints of subcurves on the curve.

Submodes: BCS

Commands: `SPLIT, MOVE, MERGE, ROLE, CLEAR`

`SPLIT`	Introduce new subcurves by splitting an old subcurve into two

MOVE Move an endpoint along a curve (not to cross other subcurve boundaries, in which case MERGE must be used)

MERGE Join two subcurves by removing one common endpoint

ROLE Give a nontrivial role (*i.e.*, *periodic*, *physical*, or *auxiliary*) to a subcurve

CLEAR Clear a subcurve of its role and erase all pertinent information, such as boundary-condition ID, donor grid, *etc*.

MESH/CURVE/SUBCURVE/BCS

On a subcurve boundary conditions of mixed, linear type (Robin problem) may be imposed. These can be obtained in various ways, depending on the source of the conditions (physical or periodic boundary condition, or conditions interpolated from another grid; the former two are defined in the BCS mode).

Tools: *PHYSICAL, PERIODIC*

PHYSICAL The user may specify constants defining the linear, mixed-conditions formulation interactively. More sophisticated boundary conditions must be specified by referring to a (predefined) Fortran function on a separate boundary-condition file (bcfun)

PERIODIC When periodic conditions are imposed, the user must indicate where in space the -linear- boundary condition operator must be evaluated. Effectively, an affine transformation linking the subcurve under consideration to an image must be defined. Boundary conditions for a specific point are then obtained by applying the boundary condition operator to the solution in the image point. The affine transformation can be specified explicitly numerically, or by constructing it using the facilities described under COPY

MESH/GRID

In this mode a component grid is constructed or changed.

Submodes: CHECK, DEFINE, MODIFY, SIDE

Commands: `DELETE`

`DELETE` Remove the definition of a grid from the database, keeping the constituting curves and/or subcurves intact

---

MESH/GRID/CHECK

In this mode checks are performed to determine the quality of a grid. The consistency check is mandatory, but need only be performed once the grid has been defined completely. The folding check is more expensive to perform. It should only be invoked when the chance exists that a grid mapping become singular.

Commands: `LOGIC, FOLDING`

`LOGIC` Check if a grid is defined in a consistent way, *i.e.*, if logically intersecting sides really intersect. If not, an error message is issued. Other parameterization issues are solved by the system, such as parameters running in the wrong direction, or mappings defining a left-handed curvilinear coordinate system. The grid is marked for consistency after the check

`FOLDING` Check if the Jacobian of the mapping of the grid becomes singular

---

MESH/GRID/DEFINE (MODIFY)

In this mode a new grid is defined. Mode MODIFY, which can also be invoked in GRID, does exactly the same as DEFINE, but operates on the definition of an already existing grid.

Submodes: SIDE

Tools: *STRETCH, CELLS, MAPPING, ORIGIN*

STRETCH Define stretching (or shrinking) functions on the grid

CELLS Specify the number of mesh cells in both directions

MAPPING Choose a mapping for the grid (Coons patch, isoparametric macro-element)

ORIGIN Supply the origin of the local curvilinear coordinate system in terms of absolute x, y-coordinates (this pins down the relative locations of the sides of the grid, once the sides are defined)

MESH/GRID/DEFINE/SIDE

In this mode functional and geometrical operations on grid sides (including creation and elimination) are performed.

Submodes: GEOMETRY, FUNCTION

Commands: `DELETE, CLEAR`

`DELETE` Remove the side definition if unsatisfactory. This does not mean that curves or subcurves are affected!

`CLEAR` Remove the subdivision into subsides and set the role of the only remaining subside to NONE, *i.e.*, the definition of the side is reset functionally, but not geometrically

---

MESH/GRID/SIDE/DEFINE/GEOMETRY

In this mode a list of subcurves specifying a grid side is manipulated (geometrical definition).

Submodes: CONNECT

Tools: SUBCURVE, CURVE

SUBCURVE Add a specific subcurve to or delete it from the list describing the side

CURVE Identify all the subcurves comprising a whole curve at once, and add them to or delete them from the list of subcurves describing a side

---

MESH/GRID/DEFINE/SIDE/GEOMETRY/CONNECT

In this mode important information regarding grid metrics is defined. If a grid side consists of exactly one curve or subcurve, it is obvious for an algebraic grid generator to let the s-interval in computational space parameterizing that side (*i.e.*, $s : 0 \rightarrow 1$) correspond linearly to the z-interval parameterizing the pertinent curve or subcurve. So if z runs from z_0 to z_1, we compute $z = z_0 + s * (z_1 - z_0)$. The situation becomes ambiguous when subcurves on different curves make up a side. Then there is no obvious unique mapping from s to z. If there are N adjacent subcurves making up the side, then the continuity requirement along that side with respect to the corresponding s-parameter intervals $\{(s_{i_0}, s_{i_1})\}_{i=1}^{N}$ is just: $s_{i_1} = s_{i+1_0}$ (naturally, there are also the consistency requirements $s_{1_0} = 0, s_{N_1} = 1, s_{i_0} \leq s_{i_1}$). However, this means that curve speed (rate of change of arc length along a side as s increases) may be discontinuous across subcurve interfaces. In general this is undesirable.

CONNECT provides the possibility of computing these s-parameter intervals automatically and obtaining a globally continuous curve speed. When defining the

intervals by hand, the C^0 continuity requirement is enforced by allowing the user only to specify *s*-parameter *values* at subcurve interface points (nodes), rather than *intervals*.

Tools: *MANUAL*

Commands: `AUTO`

MANUAL The user may point to interior nodes and supply *s*-values

`AUTO` The *s*-values are computed automatically to ensure continuous curve speed

---

MESH/GRID/DEFINE/SIDE/FUNCTION

In this mode the functional definition of grid sides is given. This comes down to dividing grid sides into functionally monolithic segments called subsides, which play exactly one numerical role. Initially, every side consists of one subside with the dummy role NONE.

Tools: *LOCATION, ROLE*

Commands: `SPLIT, MOVE, MERGE, CLEAR`

LOCATION Name the relative location of the side (*left, right, etc.*)

ROLE Assign a nontrivial role to a subside, *i.e.*, *interpolation, physical, periodic* or *reentrant*

`SPLIT` Introduce new subsides by splitting an old subside into two. The roles of the new subsides are the same as the role of the parent subside

`MOVE` Move an endpoint along a side (not to cross other subside boundaries, in which case MERGE should be used). Roles remain unchanged

MERGE Forge two subsides together by removing one common endpoint. If the respective roles of the old subsides were the same, then this will be the role of the newly synthesized subside. NONE otherwise

CLEAR Set the role of a certain subside equal to NONE

MESH/OBJECT An important feature of M*E*S*H is that larger structures can be built from simple elements, which can subsequently be embedded in yet larger structures. Thus, a hierarchical system of structures called objects can be created. The main benefit of objects is that they can be moved or copied as entities without referring to their atomic constituents. The computational program processing the output of M*E*S*H ignores objects; it only uses the definitions of curves and grids.

Submodes: DEFINE, MODIFY, MOVE, COPY

Commands: `DELETE, CLEAR`

DELETE Delete an entire object, including all its constituents. This potentially destructive feature should be adequately protected

CLEAR Remove the definition of an object from the database without changing its constituents (in other words: *unpack* the object)

MESH/OBJECT/DEFINE (MODIFY)

In this mode an object is assembled. Note: the constituents of the object itself remain untouched. MODIFY does the same as define, but works on the definition of an already existing object.

Tools: *CURVE, GRID, OBJECT*

CURVE Identify a curve and add it to or delete it from the definition of the object

GRID Identify a grid and add it to or delete it from the definition of the object

OBJECT Identify another object and add it to or delete it from the definition of the object

---

MESH/OBJECT/COPY (MOVE)

In this mode a whole object is copied using an affine transformation (see above). MOVE works similarly, but the original object is not retained.
Note: The MOVE utility does change the definitions of the constituents of the object.

MESH/ORDER

This mode enables the user to specify the order in which component grids will be traversed while performing SWAP(R) iterations. In principle, the list is unbounded and a particular grid may appear any number of times.

Submodes: DEFINE, MODIFY

Commands: `CHECK`

`CHECK` Check if a grid sequence is legal, and, if so, if it is complete, *i.e.*, if all the grids in the composite grid are visited at least once

---

MESH/ORDER/DEFINE (MODIFY)

DEFINE is used to specify a new list, whereas MODIFY changes an existing grid sequence.

Tools: *ALGORITHM, MANUAL*

ALGORITHM	Select a standard traversal algorithm
MANUAL	Specify a traversal list by hand

MESH/FILE

This mode enables the user to save information on a file during the session, and to load information from a file.

Tools: *LOAD, WRITE, OWRITE*

LOAD	Read grid-generation data from a file
WRITE	Write the current grid-generation information to a new file. A distinction is made between files that will be reused in another M∗E∗S∗H session, and those that are going to serve as input for the computational program
OWRITE	Same as WRITE, but now an already existing file is overwritten

---

SHOW, ZOOM

The ZOOM tool is purely graphical and allows the user to get a close-up view of parts of the composite grid, or to obtain a global picture. The SHOW mode is diagnostic; information about any of the defined entities can be called to the screen, either graphically or numerically. Both ZOOM and SHOW are nonhierarchical and can be invoked at any time.

Overview of System Functions

```
                                   |#USERDEF       |$STRAIGHT            LEGEND:
                    |#DEFINE →     |#STANDARD →    |$CONICAL               # → MODE
                    |              |$HERMITE       |$POLYNOM               $ → TOOL
                    |              |$SPLINE                                & → COMMAND
                    |              |$BEZIER
                    |
                    |#MODIFY →     |$PARM
                    |              |$TRIM
                    |
                    |              |$FIT
                    |              |$SHIFT
                    |#COPY →       |$ROTATE
         |#CURVE →  | (MOVE)       |$SHRINK
         |          |              |$MIRROR
         |          |              |$AFFINE
         |          |                              |$PHYSICAL
         |          |              |#BCS →         |
         |          |              |&SPLIT         |$PERIODIC
         |          |#SUBCURVE →   |&MOVE
         |          |              |&MERGE
         |          |&CLEAR        |&ROLE
         |          |&DELETE       |&CLEAR
         |                                                                      |$MANUAL
         |                         |&LOGIC                      |#CONNECT →     |
         |          |#CHECK →      |&FOLDING   |#GEOMETRY →     |$SUBCURVE      |&AUTO
         |          |                          |                |$CURVE
         |          |              |#SIDE →    |&DELETE
         |          |#DEFINE →     |$STRETCH   |&CLEAR          |$LOCATION
#MESH →  |#GRID →   | (MODIFY)     |$CELLS     |                |$ROLE
         |          |              |$MAPPING   |#FUNCTION →     |&SPLIT
         |          |&DELETE       |$ORIGIN                     |&MOVE
         |                                                      |&MERGE
         |          |#DEFINE →     |$CURVE                      |&CLEAR
         |          | (MODIFY)     |$GRID
         |#OBJECT → |#COPY         |$OBJECT
         |          | (MOVE)
         |          |&DELETE
         |          |&CLEAR
         |                         |$MANUAL
         |          |#DEFINE →     |
         |#ORDER →  | (MODIFY)     |$ALGORITHM
         |          |&CHECK
         |                                                      NONHIERARCHICAL
         |          |$LOAD                                        #SHOW
         |#FILE →   |$(O)WRITE                                    $ZOOM
```

Appendix C

NAVIER-STOKES DISCRETIZATION COEFFICIENTS

In this appendix the complete coefficients for the discretized Navier-Stokes equations, as derived in Chapter 7, are given. The discrete continuity equation for the grid cell with center-point indices $(i+\frac{1}{2}, j+\frac{1}{2})$ is:

$$A^{cu}_{sw}u_{sw} + A^{cu}_{nw}u_{nw} + A^{cu}_{se}u_{se} + A^{cu}_{ne}u_{ne} + A^{cv}_{sw}v_{sw} + A^{cv}_{nw}v_{nw} + A^{cv}_{se}v_{se} + A^{cv}_{ne}v_{ne} = 0 \ . \quad \text{(C.1)}$$

The coefficients are:

$$A^{cu}_{sw} = \left\{ -\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i,j+\frac{1}{2}} - \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i+\frac{1}{2},j} \right\}/2 \ , \quad \text{(C.2)}$$

$$A^{cu}_{nw} = \left\{ -\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i,j+\frac{1}{2}} + \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i+\frac{1}{2},j+1} \right\}/2 \ , \quad \text{(C.3)}$$

$$A^{cu}_{se} = \left\{ +\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i+1,j+\frac{1}{2}} - \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i+\frac{1}{2},j} \right\}/2 \ , \quad \text{(C.4)}$$

$$A^{cu}_{ne} = \left\{ +\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i+1,j+\frac{1}{2}} + \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i+\frac{1}{2},j+1} \right\}/2 \ , \quad \text{(C.5)}$$

$$A^{cv}_{sw} = \left\{ -\left(\mathbf{J}\frac{\partial\xi}{\partial y}\right)_{i,j+\frac{1}{2}} - \left(\mathbf{J}\frac{\partial\eta}{\partial y}\right)_{i+\frac{1}{2},j} \right\}/2 \ , \quad \text{(C.6)}$$

$$A^{cv}_{nw} = \left\{ -\left(\mathbf{J}\frac{\partial\xi}{\partial y}\right)_{i,j+\frac{1}{2}} + \left(\mathbf{J}\frac{\partial\eta}{\partial y}\right)_{i+\frac{1}{2},j+1} \right\}/2 \ , \quad \text{(C.7)}$$

$$A^{cv}_{se} = \left\{ +\left(\mathbf{J}\frac{\partial\xi}{\partial y}\right)_{i+1,j+\frac{1}{2}} - \left(\mathbf{J}\frac{\partial\eta}{\partial y}\right)_{i+\frac{1}{2},j} \right\}/2 \ , \quad \text{(C.8)}$$

$$A^{cv}_{ne} = \left\{ +\left(\mathbf{J}\frac{\partial\xi}{\partial y}\right)_{i+1,j+\frac{1}{2}} + \left(\mathbf{J}\frac{\partial\eta}{\partial y}\right)_{i+\frac{1}{2},j+1} \right\}/2 \ . \quad \text{(C.9)}$$

The discrete u-momentum equation for the node with indices (i,j) is:

$$A_P^u\, u_P = A_E^u\, u_E + A_W^u\, u_W + A_N^u\, u_N + A_S^u\, u_S + sc^u + \\ A_{nw}^{up}\, p_{nw} + A_{sw}^{up}\, p_{sw} + A_{ne}^{up}\, p_{ne} + A_{se}^{up}\, p_{se}\,. \tag{C.10}$$

The coefficients are:

$$A_E^u = D_e \Lambda(|P_e|) + \max[-F_e, 0]\,, \tag{C.11}$$

$$A_W^u = D_w \Lambda(|P_w|) + \max[+F_w, 0]\,, \tag{C.12}$$

$$A_N^u = D_e \Lambda(|P_n|) + \max[-F_n, 0]\,, \tag{C.13}$$

$$A_S^u = D_e \Lambda(|P_s|) + \max[+F_s, 0]\,, \tag{C.14}$$

$$A_P^u = A_E^u + A_W^u + A_N^u + A_S^u\,, \tag{C.15}$$

$$A_{nw}^{up} = \left\{+\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i-\frac{1}{2},j} - \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i,j+\frac{1}{2}}\right\}/2\,, \tag{C.16}$$

$$A_{sw}^{up} = \left\{+\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i-\frac{1}{2},j} + \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i,j-\frac{1}{2}}\right\}/2\,, \tag{C.17}$$

$$A_{ne}^{up} = \left\{-\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i+\frac{1}{2},j} - \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i,j+\frac{1}{2}}\right\}/2\,, \tag{C.18}$$

$$A_{se}^{up} = \left\{-\left(\mathbf{J}\frac{\partial\xi}{\partial x}\right)_{i+\frac{1}{2},j} + \left(\mathbf{J}\frac{\partial\eta}{\partial x}\right)_{i,j-\frac{1}{2}}\right\}/2\,, \tag{C.19}$$

where the quantities F, D, P, and Λ are defined using the relations (7.42) through (7.46).

The discrete v-momentum equation for the node with indices (i,j) is:

$$A_P^v\, v_P = A_E^v\, v_E + A_W^v\, v_W + A_N^v\, v_N + A_S^v\, v_S + sc^v + \\ A_{nw}^{vp}\, p_{nw} + A_{sw}^{vp}\, p_{sw} + A_{ne}^{vp}\, p_{ne} + A_{se}^{vp}\, p_{se}\,. \tag{C.20}$$

The coefficients are:

$$A_E^v \equiv A_E^u\,, \tag{C.21}$$

$$A_W^v \equiv A_W^u\,, \tag{C.22}$$

$$A_N^v \equiv A_N^u\,, \tag{C.23}$$

$$A_S^v \equiv A_S^u \,, \tag{C.24}$$

$$A_P^v \equiv A_P^u \,, \tag{C.25}$$

$$A_{nw}^{vp} = \left\{ + \left(\mathbf{J} \frac{\partial \xi}{\partial y} \right)_{i-\frac{1}{2},j} - \left(\mathbf{J} \frac{\partial \eta}{\partial y} \right)_{i,j+\frac{1}{2}} \right\} /2 \,, \tag{C.26}$$

$$A_{sw}^{vp} = \left\{ + \left(\mathbf{J} \frac{\partial \xi}{\partial y} \right)_{i-\frac{1}{2},j} + \left(\mathbf{J} \frac{\partial \eta}{\partial y} \right)_{i,j-\frac{1}{2}} \right\} /2 \,, \tag{C.27}$$

$$A_{ne}^{vp} = \left\{ - \left(\mathbf{J} \frac{\partial \xi}{\partial y} \right)_{i+\frac{1}{2},j} - \left(\mathbf{J} \frac{\partial \eta}{\partial y} \right)_{i,j+\frac{1}{2}} \right\} /2 \,, \tag{C.28}$$

$$A_{se}^{vp} = \left\{ - \left(\mathbf{J} \frac{\partial \xi}{\partial y} \right)_{i+\frac{1}{2},j} + \left(\mathbf{J} \frac{\partial \eta}{\partial y} \right)_{i,j-\frac{1}{2}} \right\} /2 \,. \tag{C.29}$$

Note that the coefficients for the convection/diffusion operator are the same for the u- and v-momentum equations, which means that fewer field arrays need to be stored. This is another beneficial property of the modified staggered grid.

The source terms for both momentum equations are composed of skewed fluxes and central-difference corrections.

References

[ACR68] A. Acrivos, L.G. Leal, D.D. Snowden, F. Pan, *Further Experiments on Steady Separated Flows past Bluff Objects,* Journal of Fluid Mechanics, Vol. **34**, Part 1, pp. 25-48, 1968

[AND88] C.R. Anderson, *Derivation and Solution of the Discrete Pressure Equations for the Incompressible Navier-Stokes Equations,* Report to Appear, University of California, Los Angeles, Department of Mathematics, 1989

[ATT83] E.H. Atta, J. Vadyak, *A Grid-Overlapping Scheme for Flowfield Computations about Multicomponent Configurations,* AIAA Journal, Vol. **21**, No. 9, pp. 1271-1277, 1983

[BEN86] J.A. Benek, J.L. Steger, P.G. Buning, *Chimera: A Grid-Embedding Technique,* Arnold Engineering Development Center, Report AEDC-TR-85-64, 1986

[BER82] M.J. Berger, *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations,* Ph.D. Thesis, Stanford University, 1982

[BER84] M.J. Berger, *On Conservation at Grid Interfaces,* ICASE Report No. 84-83, NASA Langley Research Center

[BER86] T. Berglind, *A Comparison of Single-Block and Multi-Block Grids around Wing-Fuselage Configurations,* The Aeronautical Research Institute of Sweden. Report No. FFA TN 1986-42, 1986

[BRO89A] D.L. Brown, G. Chesshire, B. Henshaw, *Getting Started with CMPGRD: Introductory User's Guide and Reference Manual,* Los Alamos Rept. No. LA-UR-89-1294M, 1989

[BRO89B] D.L. Brown, G. Chesshire, B. Henshaw, *Composite Grid Data; An Explanation of the CMPGRD Composite Grid Data Structure,* IBM Watson Research Center, Report No. RC 14354 (# 64299), 1989

[CAR85] S.C. Caruso, *Adaptive-Grid Techniques for Elliptic Fluid-Flow Problems,* Ph.D. Thesis, Stanford University, 1985

[CHA88A] T.F. Chan, D. Goovaerts, *Schwarz=Schur: Overlapping Versus Nonoverlapping Domain Decomposition,* CAM Report 88-21, University of California, Los Angeles, 1988

[CHA88B] T.F. Chan, *Domain Decomposition Algorithms and Computational Fluid Dynamics,* CAM Report 88-25, University of California, Los Angeles, 1988

[CHE86] G.S. Chesshire, *Composite Grid Construction and Applications,* Ph.D. Thesis, California Institute of Technology, 1986

[CHI87] R.C.Y. Chin, G.W. Hedstrom, J.S. Scroggs, D.C. Sorensen, *Parallel Computation of a Domain Decomposition Method,* CONF-870677-8, DE87 011444

[COL73] W.M. Collins, S.C.R. Dennis, *Flow past an Impulsively Started Circular Cylinder,* Journal of Fluid Mechanics, Vol. **60**, Part 1, pp. 105-127, 1973

[DAN86] J.F. Dannenhoffer III, J.R. Baron, *Robust Grid Adaptation for Complex Transonic Flows*, AIAA-86-0495, Reno, Nevada, 1986

[DEN70] S.C.R. Dennis, G-Z. Chang, *Numerical Solutions for Steady Flow past a Circular Cylinder at Reynolds Numbers up to 100,* Journal of Fluid Mechanics, Vol. **42**, Part 3, pp. 471-489, 1970

[DOU85] F.C. Dougherty, J.A. Benek, J.L. Steger, *On Applications of Chimera Grid Schemes to Store Separation,* NASA Technical Memorandum 88193, 1985

[DYN83] N. Dyn, W.E. Ferguson, Jr., *The Numerical Solution of Equality-Constrained Quadratic Programming Problems,* Mathematics of Computation, Vol. **41**, No. 163, pp. 165-170, 1983

[EHR86] L.W. Ehrlich, *The Numerical Schwarz Alternating Procedure and SOR,* SIAM Journal of Scientific and Statistical Computing, Vol. **7**, No. 3, pp. 989-993, 1986

[EIS79] P.R. Eiseman, *A Multi-Surface Method for Coordinate Generation,* Journal of Computational Physics, Vol. **33**, pp. 118-150, 1979

[EIS87] P.R. Eiseman, G. Erlebacher, *Grid Generation for the Solution of Partial Differential Equations*, ICASE Report 87-57, 1987

[FAU81] I.D. Faux, M.J. Pratt, *Computational Geometry for Design and Manufacture*, John Wiley & Sons, 1981

[FLO83] J. Flores, T.L. Holst, D. Kwak, D.M. Batiste, *A New Consistent Spatial Differencing Scheme for the Transonic Full Potential Equation*, AIAA-83-0373, Jan. 1983

[FUC85] L. Fuchs, *Multi-grid Solutions on Grids with Non-Aligning coordinates*, Progress Report, Royal Institute of Technology, Stockholm, Sweden, 1985

[FUC87] L. Fuchs, *Numerical Computation of Viscous Incompressible Flows in Systems of Channels*, AIAA-87-0367, Reno, Nevada, 1987

[FUN88] D. Funaro, A. Quarteroni, P. Zanolli, *An Iterative Procedure with Interface Relaxation for Domain Decomposition Methods*, SIAM Journal on Numerical Analysis, Vol. **25**, No. 6, pp. 1213-1236, 1988

[GRO64] A.S. Grove, F.H. Shair, E.E. Petersen, A. Acrivos, *An Experimental Investigation of the Steady Separated Flow past a Circular Cylinder*, Journal of Fluid Mechanics, Vol. **19**, pp. 60-80, 1964

[HAU86] J. Häuser, H.G. Paap, D. Eppel, S. Sengupta, *Boundary Conformed Co-Ordinate Systems for Selected Two-Dimensional Fluid Flow Problems. Part II: Application of the BFG Method*, International Journal for Numerical Methods in Fluids, Vol. **6**, pp. 529-539, 1986

[HEN85] W.D. Henshaw, *Part I: The Numerical Solution of Hyperbolic Systems of Conservation Laws; Part II: Composite Overlapping Grid Techniques*, Ph.D. Thesis. California Institute of Technology, 1985

[HES86] K.A. Hessenius, M.M. Rai, *Applications of a Conservative Zonal Scheme to Transient and Geometrically Complex Problems*, Computers and Fluid, Vol. **14**. No. 1, pp. 43-58, 1986

[HOL85] T.L. Holst, S.D. Thomas, U. Kaynak, K.L. Gundy, J. Flores, N.M. Chaderjian, *Computational Aspects of Zonal Algorithms for Solving the Compressible*

Navier-Stokes Equations in Three Dimensions, NASA Technical Memorandum 86774, 1985

[HOL87] J.E. Holcomb, *Development of a Grid Generator to Support 3-D Multizone Navier-Stokes Analysis*, AIAA-87-0203, Reno, Nevada, 1987

[HON69] H. Honji, S. Taneda, *Unsteady Flow past a Circular Cylinder*, Journal of the Physical Society of Japan, Vol. **27**, No. 6, pp. 1668-1677, 1969

[KAR87] G.E. Karniadakis, A.T. Patera, *Numerical Simulation of Forced Convection Heat Transfer from a Cylinder in Crossflow*, Rept. No. FML-87-001, Fluid Mechanics Laboratory, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1987

[KAW66] M. Kawaguti, P. Jain, *Numerical Study of a Viscous Fluid Flow past a Circular Cylinder*, Journal of the Physical Society of Japan, Vol. **21**, No. 10, pp. 2055-2062, 1966

[KAY86] Ü. Kaynak, T.L. Holst, B.J. Cantwell, *Computation of Transonic Separated Wing Flows Using an Euler/Navier-Stokes Zonal Approach*, NASA Technical Memorandum 88311, 1986

[KUM80] A. Kumar, K.S. Yajnik, *Internal Separated Flows at Large Reynolds Numbers*, Journal of Fluid Mechanics, Vol. **97**, pp. 27-51, 1980

[MAL83] C.R. Maliska, G.D. Raithby, *Calculating Three-Dimensional Fluid Flows using Nonorthogonal Grids*, Proceedings of the Third International Conference on Numerical Methods in Laminar and Turbulent Flows, Seattle, pp. 656-666, 1983

[MAR87] L.D. Marini, A. Quarteroni, *A Relaxation Procedure for Domain Decomposition Methods Using Finite Elements*, Instituto di Analysi Numerica del Consiglio Nazionale Delle Ricerche, Pavia, Italy, Publication No. 577, 1987

[MEA86] R.L. Meakin, *Application of Boundary Conforming Coordinate and Domain Decomposition Principles to Environmental Flows*, Ph.D. Thesis, Stanford University, 1986

[MEA88A] R.L. Meakin, R.L. Street, *Simulation of Environmental Flow Problems in Geometrically Complex Domains. Part I: A General Coordinate Transformation*, Computer Methods in Applied Mechanics and Engineering, Vol. 68, pp. 151-175, 1988

[MEA88B] R.L. Meakin, R.L. Street, *Simulation of Environmental Flow Problems in Geometrically Complex Domains. Part II: A Domain-Splitting Method,* Computer Methods in Applied Mechanics and Engineering, Vol. 68, pp. 311-331, 1988

[MIL65] K. Miller, *Numerical Analogs to the Schwarz Alternating Procedure,* Numerische Mathematik, Vol. 7, pp. 97-103, 1965

[NAK87] K. Nakahashi, S. Obayashi, *Viscous Flow Computations Using a Composite Grid,* AIAA-87-1128-CP, Honolulu, Hawaii, 1987

[OLI84] J. Oliger, *Adaptive Grid Methods for Hyperbolic Partial Differential equations,* Inverse Problems of Acoustic and Elastic Waves, F. Santosa, Y.-H. Pao, W.W. Symes, C. Holland, Eds., SIAM, Philadelphia, 1984

[OSS83] G.A. Osswald. *A Direct Numerical Method for the Solution of Unsteady Navier-Stokes Equations in Generalized Orthogonal Coordinates,* Ph.D. Thesis, University of Cincinnati, 1983

[PAT81] S.V. Patankar, *A Calculation Procedure for Two-Dimensional Elliptic Situations,* Journal of Numerical Heat Transfer, Vol. 4, pp. 409-425, 1981

[PAT80] S.V. Patankar. *Numerical Heat Transfer and Fluid Flow,* McGraw-Hill, 1980

[PED86] G. Pedersen, *On the Effects of Irregular Boundaries in Finite Difference Models,* International Journal for Numerical Methods in Fluids, Vol. 6, pp. 497-505, 1986

[PER85] M. Perić. *A Finite Volume Method for the Prediction of Three-Dimensional Fluid Flow in Complex Ducts,* Ph.D. Thesis, Imperial College, University of London, 1985

[PER89A] C-Y. Perng, R.L. Street, *Three-Dimensional Unsteady Flow Simulations: Alternative Strategies for a Volume-Averaged Calculation,* International Journal for Numerical Methods in Fluids, Vol. 9, pp. 341-362, 1989

[PER89B] C-Y. Perng, R.L. Street, *A Domain-Decomposition Technique for Solving Geometrically Complex Flow Problems,* Proceedings, 1989 ASME Winter Annual

Meeting on Numerical Simulation of Convection in Electronic Equipment Cooling, December 1989

[PER89C] C-Y. Perng, R.L. Street, *A Coupled Multigrid-Domain-Splitting Technique for Simulating Incompressible Flows in Geometrically Complex Domains,* Submitted to International Journal for Numerical Methods in Fluids, August 1989

[QUA87] A. Quarteroni, G. Sacchi Landriana, *Parallel Algorithms for the Capacitance Matrix Method in Domain Decomposition,* Instituto di Analysi Numerica del Consiglio Nazionale Delle Ricerche, Pavia, Italy, Publication No. 595, 1987

[RAI85] M.M. Rai, *An Implicit, Conservative, Zonal-Boundary Scheme for Euler Equation Calculations,* AIAA-85-0488, Reno, Nevada, 1985

[RAI86] M.M. Rai, *A Relaxation Approach to Patched-Grid Calculations with the Euler Equations,* Journal of Computational Physics, Vol. **66**, pp. 99-131, 1986

[RAI87] M.M. Rai, *Navier-Stokes Simulations of Rotor/Stator Interaction Using Patched and Overlaid Grids,* Journal of Propulsion, Vol. **3**, No. 5, pp. 387-396, 1987

[ROD83] G. Rodrigue, J. Simon, *A Generalization of the Numerical Schwarz Algorithm,* Proceedings of the Sixth International Conference on Computing Methods in Applied Sciences and Engineering, Paris, France, 1983

[SCH69] H.A. Schwarz, *Über einige Abbildungsaufgaben,* Journal für die Reine und Angewandte Mathematik, Vol. **70**, pp. 105-120, 1869

[SCH79] H. Schlichting, *Boundary-Layer Theory,* McGraw-Hill, 1979

[SCR88] J.S. Scroggs, *The Solution of a Parabolic Partial Differential Equation via Domain Decomposition: The Synthesis of Asymptotic and Numerical Analysis,* Argonne National Laboratory, Mathematics and Computer Science Division. Technical Memorandum 123, 1988

[SHI89] T.M. Shih, C.H. Tan, B.C. Hwang, *Effects of Grid Staggering on Numerical Schemes,* International Journal for Numerical Methods in Fluids, Vol. **9**, pp. 193-212, 1989

[SKA87] W.C. Skamarock, *Adaptive Grid Refinement for Numerical Weather Prediction*, Ph.D. Thesis, Stanford University, 1987

[SMI76] F.T. Smith, *Flow through Constricted or Dilated Pipes and Channels, Part 1 and 2*, Quarterly Journal of Mechanics and Applied Mathematics, Vol. **29**, Part 3, pp. 343-376, 1976

[SMI77] F.T. Smith, *Upstream Interactions in Channel Flows*, Journal of Fluid Mechanics, Vol. **79**, part 4, pp. 631-655, 1977

[STO72] D.R. Stoutemyer, *Numerical Implementation of the Schwarz Alternating Procedure for Elliptic Partial Differential Equations*, Ph.D. Thesis, Stanford University, 1972

[TAK] T. Takagi, K. Miki, *An Approach to the Thermal-Hydraulic Analysis of Complicated Piping by Domain Decomposition and Overlapping Techniques*, Proceedings of the Sixth GAMM-Conference on Numerical Methods in Fluid Mechanics, Eds. Rues & Kordulla, Friedrich Vieweg & Sohn, Braunschweig/Wiesbaden, W.-Germany

[TAN56] S. Taneda, *Experimental Investigation of the Wakes behind Cylinders and Plates at Low Reynolds Numbers*, Journal of the Physical Society of Japan, Vol. **11**, No. 3, pp. 302-307, 1956

[TAN87] W-P. Tang, *Schwarz Splitting and Template Operators*, Ph.D. Thesis, Stanford University, 1987

[TEM77] R. Temam, *Navier-Stokes Equations; Theory and Numerical Analysis*, Studies in Mathematics and its Applications Vol. 2, North-Holland, 1977

[THO33] A. Thom, *The Flow past Circular Cylinders at Low Speeds*, Proceedings of the Royal Society of London, Series A, Vol. **141**, pp. 651-669, 1933

[THO85] J.F. Thompson, Z.U.A. Warsi, C.W. Mastin, *Numerical Grid Generation; Foundations and Applications*, Elsevier Science, 1985

[THO87] J.F. Thompson, *A Composite Grid Generation Code for General 3-D Regions*, AIAA-87-0275, Reno, Nevada, 1987

[VEN87A] E. Venkatapathy, C.K. Lombard, N. Nagaraj, *Numerical Simulation of Compressible Flow around Complex Two-Dimensional Cavities*, AIAA-87-0116, Reno, Nevada, 1987

[VEN87B] E. Venkatapathy, C.K. Lombard, J. Bardina, R. C-C. Luh, *Accurate Numerical Simulation of Supersonic Jet Exhaust Flow with CSCM on Adaptive Overlapping Grids*, AIAA-87-0465, Reno, Nevada, 1987

[VOG88] A. Andrews-Vogel, *A Knowledge-Based Approach to Automated Flow Field Zoning for Computational Fluid Dynamics*, Ph.D. Thesis, Stanford University, 1988