NASA Technical Memorandum 101072

A Knowledge-Based Approach to Automated Flow-Field Zoning for Computational Fluid Dynamics

Alison Andrews Vogel

April 1989

(NASA-TM-101072) A KNOWLEDGE-BASED APPROACH TO AUTOMATED FLOW-FIELD ZONING FOR COMPUTATIONAL FLUID DYNAMICS (NASA) 150 p CSCL 09B

N90-13048

Unclas
G3/61 0220030

National Aeronautics and
Space Administration

NASA Technical Memorandum 101072

A Knowledge-Based Approach to Automated Flow-Field Zoning for Computational Fluid Dynamics

Alison Andrews Vogel, Ames Research Center, Moffett Field, California

April 1989

NASA

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Summary

Flow-field zoning is becoming popular in computational fluid dynamics (CFD) as a means of alleviating the three-dimensional (3-D) grid-generation bottleneck caused by complex geometries, the need for selective grid refinement, and computer memory and speed limitations. As experience is gained with zonal grid methods, it has become apparent that flow-field zoning must be done *quickly* and *well* in order to reap the potential benefits of zoning and significantly ease the grid-generation bottleneck. An essential element in the satisfaction of both of these requirements is zoning expertise. Because zoning expertise is not widespread and is difficult to teach, and because even zoning experts find it difficult to visualize and specify general 3-D zonal boundary interface surfaces, the growing consensus is that the process of zoning a flow field must be automated.

The goals of the present research are to (1) lay the foundation for an automated 3-D zonal grid generation capability for CFD through the development of a demonstration computer program capable of automatically zoning the flow field of representative two-dimensional (2-D) aerodynamic configurations, and (2) determine the applicability of a knowledge-based programming approach to the domain of flow-field zoning. Several aspects of flow-field zoning make the application of knowledge-based techniques challenging: the need for perceptual information, the role of individual bias in the design and evaluation of zonings, and the fact that the zoning process has been modeled as a constructive, design-type task (for which there are relatively few examples of successful knowledge-based systems in any domain). Engineering solutions to the problems arising from these aspects have been developed, and a demonstration system has been implemented which can design, generate, and output flow-field zonings for representative 2-D aerodynamic configurations.

CHAPTER 1

INTRODUCTION

1.1 Computational Fluid Dynamics and the Grid-Generation Bottleneck

Computational fluid dynamics (CFD) is becoming an essential tool in the understanding of fluid physics, in the design of aerospace vehicles, and in a number of other areas. Kutler, Steger, and Bailey (1987) state that the long-range goal of CFD is to develop computer programs which can accurately simulate the viscous flow about realistic configurations quickly enough to drive vehicle design. Most CFD computer programs presently solve a discrete form of the fluid equations of motion. The discretized equations must be solved on a discretized computational domain, that is, at the nodes of a grid. In order to approximate the motion of a fluid accurately, the spacing of the grid nodes must be fine enough to resolve any gradients in the flow, but not so fine that the number of grid nodes is too large to admit a practical solution (due to computer limitations). Clearly the quality and tractability of a CFD flow solution is fundamentally influenced by the computational grid on which it is obtained.

Several elements of CFD were identified early in the decade by Chapman (1980) as pacing items, or areas which require significant technical advances before the long-range goal of CFD can be realized. Three-dimensional (3-D) grid generation figured prominently among those elements. In a later review, Kutler (1985) updated the list of pacing items and made a further distinction between those which are of primary importance and those which are secondary. At that time, 3-D grid generation was classified as a primary pacing item, and the most recent report by Kutler (1986) accords it the same prominence. In a comprehensive survey of grid generation capabilities for complex 3-D configurations, Thompson and Steger (1988) claim that "the problem of grid generation can still be as much an art form as it is a scientific discipline." That survey concludes that the theoretical groundwork of grid generation has been laid, but a great deal of effort is still needed in the efficient implementation of those ideas because the process of generating a grid about a complex configuration is still too time-consuming. They contend that grid generation continues to impede progress toward the goal of quick, realistic fluid flow simulation.

Despite the power and sophistication of present grid generation capabilities, it remains difficult to generate a reasonable, single grid about a general 3-D configuration. According to Rai (1986), the factors which are primarily responsible for this difficulty are geometric complexity (involving multiple-body configurations and bodies with components); the need for selective grid refinement (a fine grid is necessary in regions of large fluid variable gradients and higher derivatives, but potentially wasteful in other parts of the flow field); and size limitations of computer physical memory (trying to generate a grid with adequate resolution, minimum skewness, and no discontinuities is even more difficult if the allowed maximum number of grid nodes is insufficient). It is the combination of these factors which is propelling CFD researchers and practitioners towards domain decomposition approaches for challenging problems.

1.2 Flow-Field Zoning: A Partial Solution

Decomposition of the computational domain into simpler subdomains is an effective solution to the difficulties of 2-D and 3-D grid generation. The advantages of domain decomposition are several: (1) it can reduce the topological complexity of a complicated geometry and permit each component to be more easily fitted with an appropriate grid; (2) grid refinement can be selective if there is no constraint that grid lines be continuous across subdomain interfaces; and (3) if each subdomain contains a grid small enough to fit into the physical memory of the computer, the flow solution can proceed more quickly, and virtually any size problem can be computed either sequentially or in parallel.

There are many approaches and philosophies which can be included under the domain decomposition umbrella. Hessenius (1986) cites numerous examples of research and applications in this area and, within the family of structured grid methods, identifies two general classes of domain decomposition approaches in CFD: the patched grid approach (where the interface between subdomains is a shared curve or surface); and the overlapping, or overset, grid approach. The same distinction is made by Thompson and Steger (1988), who use the term "composite grid approach" to include grids composed of contiguous subgrids (which they call "blocked" grids instead of "patched" grids), as well as those composed of overlapping grids. The sketches in figure 1.1 (from Rai, 1985) illustrate patched and overlapping grids. Within the category of blocked or patched grids, grid lines from different subdomains may intersect the common interface at the same point and with the same slope; or they may intersect the interface at the same point, but with different slopes; or they may intersect the interface at entirely different points, resulting in complete grid discontinuity across the interface. Selective grid refinement is possible only when the grid lines are allowed to be discontinuous at subdomain interfaces.

Patched, blocked, and overlapping grids are all instances of composite grids and are the final product of domain decomposition. There exists an alternative terminology, based on the word "zone," which (1) is general enough to encompass both nonoverlapping and overlapping grids; (2) is more specific to CFD than "domain decomposition;" (3) is unlikely to produce confusion ("domain" is also a mathematical term for the range of an independent variable over which a function is defined); and (4) has distinct terms for the subdomain, the composition of subdomains, the grid within each subdomain, and the overall grid. Flow-field zoning is the term commonly used to denote the process of domain decomposition in CFD. The individual subdomains are zones, and the resulting composition of zones is called a flow-field zoning (or simply a zoning). When a domain has been decomposed, it is said to have been zoned. The interfaces between zones are zonal interfaces, and flow solvers designed to handle zonal logic and interface conditions are called zonal flow solvers. In this report, the subgrids will be referred to as zonal grids, and the resulting grid system will be called a composite zonal grid. Blocks are zones which are topologically four-sided in two dimensions, and six-sided in three dimensions. For a comprehensive discussion of

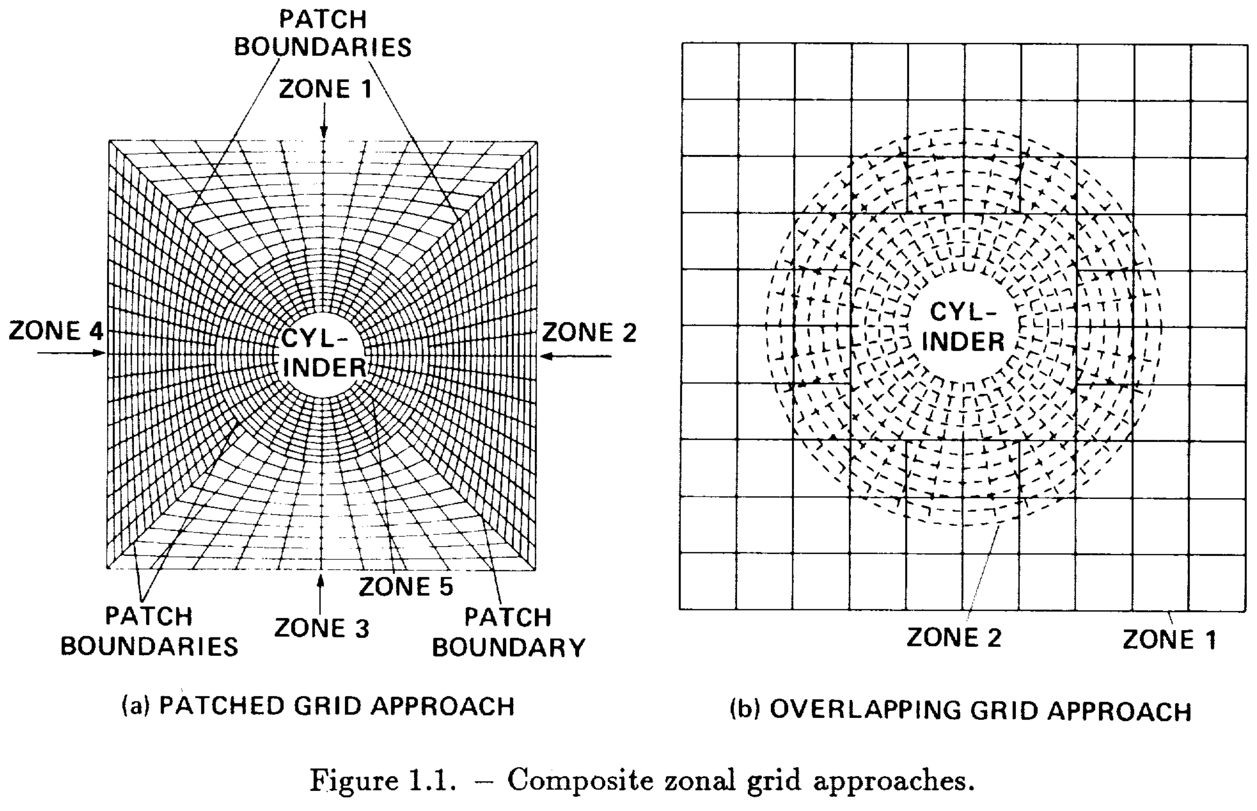


Figure 1.1. – Composite zonal grid approaches.

basic grid generation methods and terminology, the reader is referred to the well-known review article by Thompson, Warsi, and Mastin (1982).

Numerous CFD researchers and practitioners have adopted composite zonal grid approaches to alleviate grid generation difficulties, and this has enabled solutions to be computed for otherwise intractable problems. An interesting finding of Thompson and Steger's survey (1988) is that the composite block grid approach is the most prevalent version. As experience is gained with this approach, the following three facts have become apparent: (1) flow-field zoning must be performed *well* in order to reap its potential benefits; (2) flow-field zoning must be performed *quickly*; and (3) the first and second facts present a problem. Because the composite block grid approach is the most prevalent one, all subsequent references to composite zonal grids and flow-field zoning in this report will assume the choice of that approach.

To perform flow-field zoning well, a user must have experience with composite zonal grid methods, familiarity with the available grid generation capabilities, knowledge of the behavior of the zonal flow solver, fluid dynamics knowledge and some expectation of the physical features of the flow to be computed, and criteria for evaluating zonings. To perform flow-field zoning quickly, the user must have both the expertise just described, and fast, graphical, easy-to-use tools. The problem with the composite zonal grid approach lies primarily with the expertise requirement. The skill needed to do zoning quickly and well is not widespread and is not easily taught. Furthermore, even users with the requisite expertise find it difficult to visualize and specify general zonal interface surfaces in three dimensions. Finally, there are no established criteria for evaluating flow-field zonings. The growing consensus (Kutler, 1986; Steinbrenner, 1988; Thompson and Steger, 1988; Weatherill and Shaw, 1988) is that the process of zoning a flow field must be automated.

1.3 Using a Knowledge-Based Approach

All phases of CFD research and practice can be regarded as being composed of two types of activities: those for which a computer is preferable or necessary, such as high-speed numerical processing, and those for which a human is necessary, such as making judgements based on knowledge, reasoning, perception, and common sense. Efforts to enhance CFD methodology have been applied to both types of activities. Improvements to grid generators, solution methods, and algorithms have streamlined the computational part of obtaining a solution. The introduction of high-speed color graphics has aided humans in the decision-making associated with problem set-up and the interpretation of solutions. In all of these efforts, the traditional separation of labor between humans and computers has remained: humans make the decisions and computers do the number-crunching. This historically practical and satisfactory arrangement is beginning to show signs of strain, especially in the area of flow-field zoning and grid generation.

Several years ago, CFD researchers began to look to the field of artificial intelligence (AI) for approaches to automating some of the tasks still being performed by humans. Among the expected benefits of automating such tasks were relief from tedium; codification, preservation, and distribution of expertise; consistency in application; and reduced time in code development and use. One of the earliest areas to be investigated was computer symbolic mathematics. Symbolic manipulation codes have been used in CFD to perform stability and accuracy analyses and to generate FORTRAN code. Roache and Steinberg (1984) report on the use of Macsyma (a well-known symbolic manipulation program) in the generation of 3-D boundary-fitted coordinate transformations. They also give a brief history of symbolic manipulation in CFD, outline potential uses, and provide useful references to other work in the area. Steinberg and Roache (1986) claim that progress is being made in addressing the speed and memory limitations encountered in early work, but the use of symbolic manipulation tools in CFD has not yet become widespread. This is due mainly to the difficulty of effective use of these tools; one must learn a new language (repeatedly, if use is infrequent), and obtaining an answer in its simplest, most usable form can be challenging.

Knowledge-based systems (also called expert systems) are of more recent, and even greater, interest to the CFD community. Kutler and Mehta (1984) and McCroskey, Kutler, and Bridgeman (1984) recognize and speculate upon the potential applications of AI and knowledge-based techniques to CFD. Knowledge-based systems have been implemented in the areas of aerodynamic design (Tong, 1985, 1986; Nicklaus, Tong, and Russo, 1987); consultation in the use of large CFD computer codes (Conner and Purdon, 1985, 1986); and adaptive grid refinement (Dannenhoffer and Baron, 1986, 1987). Andrews (1988) analyzes each of these systems with respect to the nature of the problem addressed, the problem-solving method required, the development effort involved, system performance, and the return-on-investment, in order to gauge the progress made in the application of AI to CFD and to discover what obstacles remain. The flow-field-zoning problem was found to be a challenging application for knowledge-based techniques. Before elaborating on this point, some background information on knowledge-based programming is in order.

Knowledge-based programming is an AI problem-solving approach that has met with some success in the solution of real-world problems in a variety of fields. Pearl (1984) states that every problem-solving activity, including computer-based problem solving, requires three basic elements: a symbol structure (a way to represent all possible objects in the problem), tools for transforming one object to another, and a strategy for applying the tools so that a solution is obtained. A knowledge-based system, like any computer program, is based on the same three basic elements. In the case of a knowledge-based system, these elements are often referred to as the database, the knowledge base, and the control strategy or inference engine. However, a knowledge-based system has two primary, distinctive characteristics: (1) its architecture is that of an AI program, in which the basic elements are kept distinct—the knowledge and data are kept separate from the means by which they are processed; and (2) it contains extensive domain-specific knowledge. Davis (1982) explains that this last characteristic arises from the fundamental observation that "in the knowledge lies the power," an observation attributed to Feigenbaum, and based on the experience of many system developers indicating that the amount and quality of the knowledge is of paramount importance to problem-solving performance. The structure of a knowledge-based system is responsible for its transparency and relative ease of construction and modification. Its high level (sometimes expert level) of performance derives from the knowledge it contains. Knowledge-based systems have been successfully constructed in a variety of fields, from chemical analysis and geological survey to computer system configuration and circuit malfunction diagnosis. The text by Hayes-Roth, Waterman, and Lenat (1983) is an excellent resource about expert systems in general and about specific applications.

Useful guidelines for identifying appropriate, tractable problem domains for the application of knowledge-based programming techniques have evolved over the last several years. Based on the "recipe for success" developed by the Stanford Knowledge Systems Laboratory, as described by Hayes-Roth (1984), and on the rules presented by Lenat and McDermott (1984) in an expert systems tutorial, the following list of guidelines has been compiled:

1. The problem should have no closed-form or algorithmic solution and should involve primarily symbolic reasoning (otherwise, a knowledge-based approach would be nonsensical).
2. The problem should require some amount of expertise for its solution.
3. The time required for an expert to solve the problem should be reasonable for that domain (the Stanford recipe specifies three minutes to three hours).
4. The problem selected should be important—one which has a high payoff if solved.
5. The skill required to solve the problem should be routinely taught to nonexperts (this implies that it is well-understood and expressible).
6. The problem should be solvable over the telephone (a well-known heuristic to ensure that the solution method is primarily cognitive, and does not involve direct perception).
7. Experts should agree about the correctness of a solution to the problem, and there should be standard solution procedures.
8. It is easier to apply knowledge-based techniques to a problem which is solved by an analytic method (in which a solution is found by decomposing the problem into its primitive parts,

and then selecting existing solutions for those parts) than to a problem which is solved by a synthesis method (in which a solution is designed, or constructed, from primitives).

Flow-field zoning obeys many of these rules. Zoning is an ill-structured problem, an epithet usually applied to problems which have solution procedures that are difficult to describe in terms of an algorithm or formula, and are typically described by examples. Expertise is required to perform the task quickly and well. An expert can design and generate a flow-field zoning in several days or several weeks, depending on the complexity of the configuration. Finally, flow-field zoning is an integral part of the effort to make 3-D grid generation faster and easier, and thus has the potential for a very high payoff.

Unfortunately, flow-field zoning *breaks* others of these rules. The art of flow-field zoning is not easily taught, perhaps because there is no good language to describe the process. The task has an unmistakable perceptual element, involving qualitative shape and position information. While there are recognized zoning experts, their ideas about what constitutes a good zoning differ (and are even still evolving), so one expert's preferred solution may be unacceptable to another. This implicit "bias" affects the design and evaluation of flow-field zonings. Lastly, the process of flow-field zoning has been modeled as one in which a solution is designed rather than selected. A more complete characterization of the flow-field-zoning problem and a description of a model of the zoning process are included in chapter 2.

These latter aspects of flow-field zoning need not deter a knowledge-based system developer as long as a product is not expected for several months to a year; they simply make the job more challenging. The guidelines above are not intended to be hard-and-fast rules, but rather indicators of the potential difficulty of applying knowledge-based techniques to any given problem, and are based on corporate experience with the limitations of current knowledge-based programming techniques.

1.4 Research Objectives

The goal of the present research is two-fold: (1) to lay the foundation for an automated 3-D zonal grid generation capability for CFD through the development of a demonstration computer program capable of automatically zoning the flow field of representative 2-D aerodynamic configurations, and (2) to determine the applicability of a knowledge-based programming approach to the automation of flow-field zoning.

As stated in the previous section, certain aspects of flow-field zoning make the application of knowledge-based techniques to zoning somewhat challenging. The concepts and terminology needed to communicate zoning expertise have not been adequately formulated. There are few, if any, examples of working knowledge-based systems which address all three difficult aspects of zoning: visual perception, individual expert bias, and solution by design. Computer vision, spatial reasoning, user modeling, and design automation are all important areas of AI research, but the fruits of these research efforts are not yet mature enough for engineering application. Such difficulties preclude any rapid and straightforward application of existing knowledge-based techniques. It has been necessary to devise pragmatic solutions to these problems in order to develop a successful demonstration program. The development approach which has evolved can be summarized as follows:

1. Develop a model and language to describe the fundamental concepts and operations of 2-D flow-field zoning.
2. Debug the basic components of the model and language through the implementation of an *interactive* knowledge-based system, where the mechanics and bookkeeping of designing a zoning and generating interface curves are automated, but the user supplies the essential elements of perception, individual bias, and zoning design knowledge (i.e., the aspects of the problem which are more difficult to automate).

3. Increase the level of system automation *incrementally* by replacing the elements previously supplied by the user, one at a time, with automated versions based on the remaining components of the zoning model and language.
4. Use existing interactive grid generation capabilities.

The specific objectives of this research are to

1. Propose a model for the flow-field-zoning process that both represents it accurately and facilitates its automation.
2. Devise a language to describe zoning objects, relationships, processes, qualitative shapes and positions, expert design methodology, and individual bias.
3. Distill and encode zoning design expertise.
4. Develop a 2-D flow-field-zoning knowledge-based system which can be used both to *interactively* zone flow fields about arbitrary configurations and to *automatically* zone flow fields about representative aerodynamic configurations.
5. Propose a quantitative approach to the evaluation of flow-field zonings that incorporates individual bias.

The remainder of this report describes the accomplishment of each of these research objectives. Chapter 2 begins with an examination of flow-field zoning characteristics, followed by a model of the zoning process. The rest of the chapter is devoted to a description of the flow-field-zoning language developed to enable automation of the zoning process, and includes sections which describe how the perception problem is eliminated, individual bias is incorporated, and the design process is transformed to a simpler selection and assembly procedure. Chapter 3 contains the implementation details of EZGrid (Expert Zonal Grid generator), the knowledge-based system which was developed to perform 2-D flow-field zoning. It also includes a summary of the contents of the EZGrid knowledge base. Chapter 4 explains the scheme used to classify test-case geometries, and shows zonings and composite zonal grids for representative test cases. Chapter 5 presents a quantitative approach to the evaluation of flow-field zonings. Finally, chapter 6 contains a summary of the research, an assessment of how well the goals of this research have been met, and a discussion of opportunities for further research.

CHAPTER 2

A FLOW-FIELD ZONING MODEL

The key to automating any process lies in the ability to describe the process unambiguously. An unambiguous description is possible only if the process is understood, and if there exists a means of expressing the concepts involved, i.e., a language. This chapter begins with a more complete characterization of flow-field zoning than was given previously, focusing on the features which correspond to the knowledge-based-system development guidelines outlined in section 1.3. Section 2.2 proposes a model for the zoning process which is based, for the most part, on observation of how experts perform the zoning task. The remainder of this chapter is devoted to a language for describing flow-field zoning.

2.1 Flow-Field Zoning Characteristics

The knowledge-based-system development guidelines described in section 1.3 are summarized here:

1. The problem has no closed-form or algorithmic solution.
2. Expertise is required to solve the problem.
3. An expert can solve the problem in a reasonable amount of time.
4. The problem is an important one.
5. The skill is routinely taught to nonexperts.
6. Solution of the problem does not involve perception.
7. Experts agree on how to solve the problem.
8. Solution by analysis is easier than by synthesis.

A characterization of flow-field zoning which addresses most of these issues follows. Discussion of guidelines 4 and 8 is omitted here: the fact that the problem is important was established in chapter 1, and further treatment of the analysis versus synthesis issue is given in section 2.2.

Existence of other solutions – There is no theory which governs the way in which a flow field should be partitioned into zones. Eriksson and Rai (1988) performed an analysis of the stability of various zonal interface conditions, and Allmaras and Baron (1986) studied the effect of zonal interface formulations on solution accuracy and stability. These studies have yielded guidelines for the generation of zonal grids, but little information to influence the placement of the zonal interface curves or surfaces. Practical experience with computing flow solutions on zonal grids, as described by Hessenius and Rai (1986); Rai (1985); Hessenius (1986); Schuster and Birckelbaw (1985); Walters, Thomas, and Switzer (1986); Wardlaw, Priolo, and Solomon (1987); and Chaderjian (1988), has contributed heuristics to the pool of zoning knowledge. No comprehensive, systematic studies have been done to determine the effect on the computed solution of using different zonings for the same problem.

No satisfactory algorithmic solution to the general flow-field-zoning problem has been found. Weatherill and Shaw (1988) have developed an automatic "block decomposition" algorithm in two and three dimensions which is conceptually straightforward, successfully displaces singularities from body surfaces, and permits each body its own natural grid topology. Their method, however, results in a proliferation of blocks: in one example with a wing-canard cross-section, over 90 blocks were generated. Such a large number of blocks poses a problem when the grids are not continuous across block interfaces, because of the computational requirements of special block interface conditions. Also, grid continuity across block interfaces precludes grid refinement local to a block. Zoning literature contains no other mention of a closed-form or algorithmic solution to this problem.

Requirement for expertise – The existence, indispensability, and scarcity of flow-field zoning expertise is recognized by those members of the CFD community who use zonal grids. Steinbrenner, Karmen, and Chawner (1988) consider this to be the principal disadvantage of the zonal approach, and a "serious roadblock" to the development of fast, general grid generation methods. The same consideration prompted Thompson and Steger (1988) to name flow-field zoning as a pacing area of research in 3-D grid generation.

Solution time requirement – At best, fairly complex configurations require several weeks of an expert's time to zone and grid, which is unacceptable if the goal is to incorporate CFD within vehicle design. Design of the zoning topology and decisions as to the location of the interface curves or surfaces are the portions of the process which require expertise. The most time-consuming portion of the process is the actual generation of the interface curves or surfaces on a computer in a form usable by a numerical grid generator. One technique for zonal interface specification is to use simple, analytical geometric shapes, such as straight lines and circular arcs. Another technique (Cordova and Barth, 1988) is to use a hyperbolic grid generator (with shooting methods to permit some far boundary control) to generate body-oriented grids, and then select a grid coordinate line as the zonal interface curve. Yet another approach used by Flores, Chaderjian, and Sorenson (1987) and Chaderjian (1988) is to generate a single coarse global grid, and then identify zones which require further grid refinement, designating existing coordinate lines as zonal interfaces.

Task teachability – Terminology to describe an existing zoning has not been standardized and is a matter of some dispute. No terminology exists to describe the *process* of zoning a flow field. This fact, coupled with the expertise requirement, conspire to make the art of flow-field zoning difficult to teach. Oskam and Huizing (1986) give a pictorial account of the process used to zone the flow field of a three-element airfoil, but no verbal description is provided. The terms used in chapter 1 to introduce the subject of zoning represent an attempt at standardization. Terms for the zoning process are proposed in this chapter, in the sections on zoning objects and actions.

Role of perception – Perception of and reasoning about shapes and spatial relationships play a important role in zoning design decisions, affecting the overall topological structure of the zoning as well as the physical location of the zonal interfaces. This is evident in the way an expert, when asked to describe how he or she designed a zoning, often resorts to such phrases as, "well, I just look at the geometry," or "I can see that this body is going to have an effect on the body slightly below and downstream of it." Acknowledging the importance of perception, Thompson and Steger (1988) cite the "critical need" for graphical, interactive tools to perform surface grid generation and flow-field zoning, and to achieve better grid control.

Expert agreement on solution – As is evident from the introductory discussion of flow-field zoning, there are many approaches to zoning. Even within the category of blocked composite grids, which is the most common approach for complex problems, there are many examples of different zonings for similar configurations. If five different zoning experts with different experience, flow solvers, and computational objectives are given identical problems to zone, it is probable that five different zonings will result. Thus, zoning design exhibits an individual bias. It is not surprising, then, that the criteria used to evaluate flow-field zonings also vary from expert to expert. Short of computing a flow solution to determine grid deficiencies, evaluation of zonal grids is typically done qualitatively and graphically. Klopfer's analysis (1982) is an example of an attempt to apply quantitative criteria to the evaluation of grids, with the result that the criteria were dependent upon the flow solution. G. D. Kerlick (NASA Ames, private communication) states that the "goodness" of a grid depends not only on the geometry of the grid, but also on the partial differential equation (PDE) to be solved on the grid, the numerical method used to solve the PDE, and the PDE solution itself. Addressing the dependency on geometry only, he has developed several quantitative functions which can assess *local* grid quality. No criteria have been proposed for the evaluation of zonings before grids have been generated.

2.2 A Model of the Zoning Process

In the absence of theory, it is necessary to formulate a model of the flow-field zoning process based on observation of several experts' performance of the task. Consider first the perspective upon which the solution of a zoning problem is based. Does the expert focus on the bodies about which the flow is to be computed, or on the computational domain? The two different views can be described as object-based and region-based, respectively. In an object-based approach, the objects are operated upon by zoning actions, and a zoning is built up block by block from the objects out to the computational domain boundaries. In a region-based approach, the region is operated upon and subdivided to achieve a zoning. Some experts adopt an object-based approach, some a region-based one, and others a combination of the two. The model proposed here is object-based, as will be illustrated later.

The next issue to examine is the type of solution methodology involved. Problem-solving methods can be categorized as being either the analytic type, in which a solution is found by breaking the problem down into primitive parts for which solutions already exist and can be selected, or the synthesis type, in which a solution is synthesized, or constructed from primitive components. Here, flow-field zoning is modeled as a design process, and is therefore in the synthesis category. A zoning is designed through the application of a sequence of zoning actions which modify the initial situation (consisting of the input bodies embedded in a computational domain) by stages until an acceptable zoning is achieved.

In the design of a flow-field zoning, how does an expert know which zoning action to apply at any given stage? Generate-and-test is a common problem-solving paradigm in AI (Rich, 1983). The "generator" produces all possible solutions to a problem, and the "tester" weeds out the undesirable solutions. The process can be stopped when one acceptable solution has been found, or can be continued until all acceptable solutions have been found. In this flow-field zoning model, all possible solutions could be generated by applying all possible sequences of zoning actions. Depending on the complexity of the problem, this could result in hundreds or thousands of possible solutions, many of which would be redundant. Experts claim, believably, that they do not mentally (or otherwise) generate and discard hundreds of possible zoning solutions before settling on one they like. A hypothesis at the opposite extreme is that an expert has sufficient knowledge of the problem to be able to write down the solution directly, knowing just which sequence of zoning actions will result in the best zoning for that problem, without backtracking or evaluating alternative candidates. For the simplest configurations and conditions, this is a reasonable model of expert behavior, but for more challenging problems, it is common for an expert to try various combinations of actions, to backtrack, and perhaps even to generate several candidate solutions for comparison. For difficult problems, a modified generate-and-test paradigm may be the the most useful and plausible model of actual expert behavior. Clearly, an expert's generator does not simply apply zoning actions indiscriminately, but rather uses domain knowledge, past experience, and other factors to select only the most promising actions or sequences of actions.

The factors which may affect an expert's choice of a zoning action at any stage of the design process include

1. Basic zoning criteria and guidelines:
   a. Zones must be empty and topologically four-sided (2-D).
   b. There should be no unnecessary zones.
   c. The entire domain must be zoned (no gaps between zones).
   d. Zones must abut rather than overlap (recall that the composite block approach is assumed).
2. Geometry (body shapes, orientations, and positions relative to each other and the domain boundary).
3. Fluid physics:
   a. Inflow conditions (free-stream Mach number, angle of attack, flow steadiness)
   b. Type of solution sought (steady or unsteady, viscous or inviscid)

4. Individual bias:
   a. Capabilities of flow solver to be used (how zonal interface conditions are handled, treatment of singular points in the grid, turbulence model type, tolerance of grid skewness)
   b. Personal CFD experience (when and where in the flow can grid skewness, discontinuities, and singularities be tolerated without significant solution degradation, and when and where is the lowered accuracy at a zonal interface acceptable)
   c. Computational objectives (for example, if an accurate drag count is desired, high resolution and accuracy are necessary at the body surface and in the wake)
   d. Aesthetics ("what an ugly grid!")

In this model, more than one candidate zoning may emerge from the design process. An expert must then evaluate each candidate and select the best. Usually, evaluation is done "by eye," guided by the factors enumerated above. Often, grids are generated within each zone, and the grids are evaluated qualitatively and quantitatively by examining the Jacobians of the coordinate transformation and the grid line angles at boundaries. Final validation is possible only through the evaluation of the flow solution computed on the grids generated. These three evaluation steps have been listed in ascending order with regard to the time, effort, and cost involved.

The final consideration in the proposed model of the flow-field zoning process is that of how the expert views intermediate stages of the zoning design process. Because the primary purpose of this zoning model is to form the basis for automating the process on a computer, consistency is desirable. To enhance the consistency of this object-based model, consequences of zoning actions include the creation of new zoning objects which contain and are contained by the zoning objects existing at the previous stage of the zoning. Thus, at an intermediate stage of the zoning, a new object (resulting from a previous action) can itself be acted upon, and the computer can reason about the attributes and relations of the new object to determine the situation at that stage. The creation of and operation upon intermediate zoning objects causes this model of the flow-field-zoning process to deviate from observed expert performance in two ways: (1) an expert, even one with an object-based approach, does not think in terms of newly created zoning objects after each application of a zoning action, but rather in terms of newly created curves or surfaces; and (2) the application of a zoning action often simplifies the remaining problem for the expert, but for the computer, the creation of new objects as the design progresses leads to a proliferation of zoning objects to keep track of and reason about, actually complicating the remaining problem somewhat, rather than simplifying it.

2.3 A Zoning Language

A common language is the foundation of effective communication. Objects and actions, the first two elements of the zoning language proposed below, form the basis for communication about zoning, whether among human experts, between expert and student, or between human expert and computer in an interactive mode. They also form the basis for automated reasoning about zoning by a computer. The last three elements of the language deal with aspects of zoning which are difficult to automate according to the knowledge-based system development guidelines referred to above. These three elements are perceptual information, the incorporation of user bias, and the application of expert knowledge in the design of zonings and in the choice of interface location strategies.

2.3.1 Objects in the zoning world

Since this model of the flow-field-zoning process is object-based, it is important to be able to describe any object which may exist at any stage of the zoning. There are three principal types of objects in the zoning world: bodies, zoning environments, and zones. They are defined as follows:

1. **Body** – a solid region (an "original body," which is input by the user as an object about which flow is to be computed); or a nonsolid region which contains a body (a "virtual body").
2. **Zoning environment (z.e.)** – a nonsolid region which may or may not contain a body (in the former case, it is also a virtual body).
3. **Zone** – a zoning environment which is empty and has had a four-sided topology (in two dimensions) imposed upon it.

The initial situation, as presented to an expert or to an automated zoning program, consists of one or more original bodies specified by the poser of the problem (the "user") embedded within a computational domain which may or may not be specified by the user. An example is shown in figure 2.1, where b1 is an original body, ze1 is a zoning environment, and the computational domain is referred to as OB (for the outer boundary and the region within). There are two ways to view this initial zoning situation (and the situation at subsequent stages as well); much of the remaining fundamental object and action terminology depends on which of these views is adopted. Either OB contains body b1 *plus* a zoning environment that consists of all the nonbody region of OB, and those two objects fill (tile) OB as shown in figure 2.1.a, or OB contains b1 *only*, the nonbody region being an as-yet-unidentified part of OB, which is therefore not completely filled by objects, as depicted in figure 2.1.b. The implications of these two views are easy to discern.

Implications of the first view are

1. Zoning actions must produce either objects contained by ze1, or objects contained by OB whose enclosed area belongs to ze1. The latter option would result in the redefinition of ze1.
2. A zoning action intended to produce an object which contains b1 is impossible, since the region from which its area would be taken (ze1) does not contain b1.
3. Representation of ze1 is made difficult by the fact that it has two unconnected boundaries (an inner and an outer one) which may be changed by later zoning actions.

Implications of the second view are

1. Zoning actions produce objects contained by OB whose enclosed areas are taken from the unidentified region of OB.
2. A zoning action applied to b1 may produce an object which contains b1 and is contained by OB.
3. All objects are simply connected regions definable by a single, constant boundary, making representation easier.

The second way of viewing zoning situations is adopted here since it permits a simpler, more natural description of object-based zoning actions.

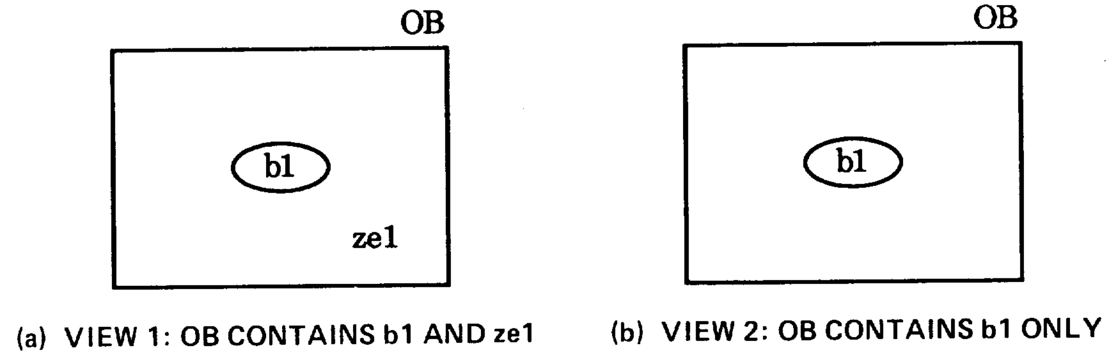

OB
b1
ze1
(a) VIEW 1: OB CONTAINS b1 AND ze1
OB
b1
(b) VIEW 2: OB CONTAINS b1 ONLY

Figure 2.1. – Two views of the same zoning situation.

An object can have a variety of properties which remain constant, such as its curvelist, vertexlist, sidelist, extrema, maximum length and width, and centroid. Explanations of the "list" quantities are given below:

Curvelist – an ordered list of curves which defines the perimeter of an object. The direction of each curve is clockwise, a convention adopted so that the interior and exterior of an object can be distinguished.

Vertexlist – an ordered list of vertices, each of which identifies a curve junction on the perimeter of an object. Each vertex is uniquely defined by the two curves whose junction it marks – its "in-curve" (the curve directed toward the vertex) and its "out-curve" (the curve directed away from the vertex), and thus its included angle and outward-direction angle are unique. Two vertices may occupy the same position in space and each have a different included angle and outward direction.

Sidelist – an ordered list of lists of curves, each list of one or more curves defining a topological side of an object. A sidelist composed of four lists of curves defines a topologically four-sided object. The curves belonging to an object's sidelist are also members of the object's curvelist, but are grouped to reflect its topology.

Curves play a major role in defining many of an object's properties, and are the basic building blocks of zonal interfaces. Curves have both constant and variable properties, such as curve type, parent, constituents, begin-u, end-u, and reverse (these are explained later). There are two types of curve: a **fundamental** curve is a 2-D curve which has been specified and generated, while a **subcurve** is either a portion of a fundamental curve, the reverse (defined later) of a fundamental curve, or the reverse of a portion. Each fundamental curve is nondimensionalized by its own length so that the nondimensional parameter u is set to 0.0 to mark the beginning of the curve (its "begin-u" property), and $u = 1.0$ denotes the end of the curve (its "end-u" property). The pointers which define a subcurve are its "parent" property, where the parent of a subcurve is the fundamental curve on which it is based, its begin-u, and its end-u. For example, a subcurve with parent curve = c7 (some fundamental curve), begin-u = 0.2, and end-u = 0.7 is a portion of curve c7, has half the length of c7 beginning one-fifth of the way along c7, and has the same directionality as c7. A subcurve which has an end-u with smaller value than its begin-u has a directionality opposite to that of its parent curve. A subcurve with begin-u = 1.0 and end-u = 0.0 is the "reverse" of its parent fundamental curve.

Subcurve creation can be regarded as fundamental curve division. The division of a fundamental curve at, for example, u = 0.37, results in the formation of two subcurves, one with begin-u = 0.0 and end-u = 0.37, and one with begin-u = 0.37 and end-u = 1.0. Subcurves are always created in pairs (with the exception of reverses, which can be created singly). If a fundamental curve has been divided, its "constituents" are the two subcurves into which it has been divided. Suppose further division of an already divided fundamental curve is required. The subcurve on which the division occurs is split into two subcurves, each of which has the original fundamental curve as its parent. The old subcurve has the two new subcurves as its constituents. It is often useful to have a list of the subcurves which are the smallest subdivisions of a fundamental curve. These are called the "least-common-denominator" constituents, or lcd-constituents, and are found by following the chain of constituent relations until each constituent of the fundamental curve has been replaced by a list of its smallest constituent subcurves. Figure 2.2 may help to clarify the curve concepts introduced here. In figure 2.2, c1 is a fundamental curve, and curves c2 through c6 are subcurves whose parent is c1. The constituents of c1 are c2 and c3, and the constituents of c3 are c4 and c5. The lcd-constituents of c1 are therefore c2, c4, and c5. Curve c6 is the reverse of c4.

Besides invariant properties, zoning objects also have relations and properties which vary depending on the current stage of the zoning design. The primary relations in this category are containment and direct containment. Parent, child, and sibling are relations derived from the containment relations. Object descriptors which can have only one of two possible values, either true or false, are connected, filled, empty, degenerate, and zoned. Other variable properties are functionals, such as corebody, contained environments, and filling objects. An explanation of these terms is given below. Figures 2.3 and 2.4 illustrate these ideas.

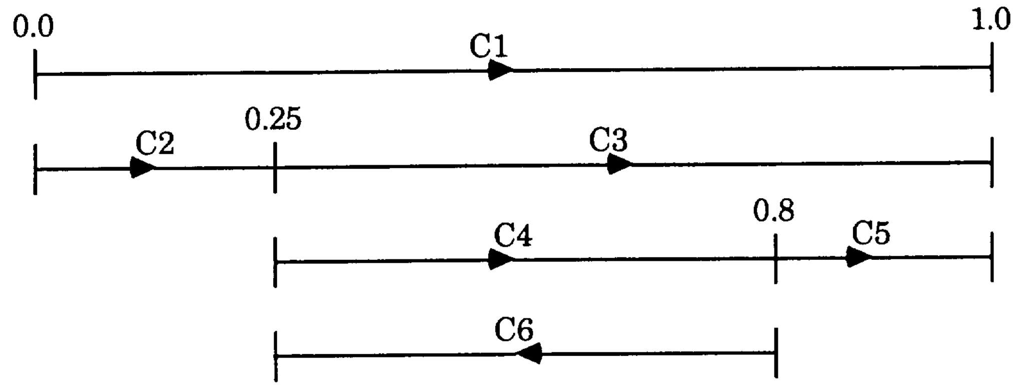


Figure 2.2. – Fundamental curves and subcurves.

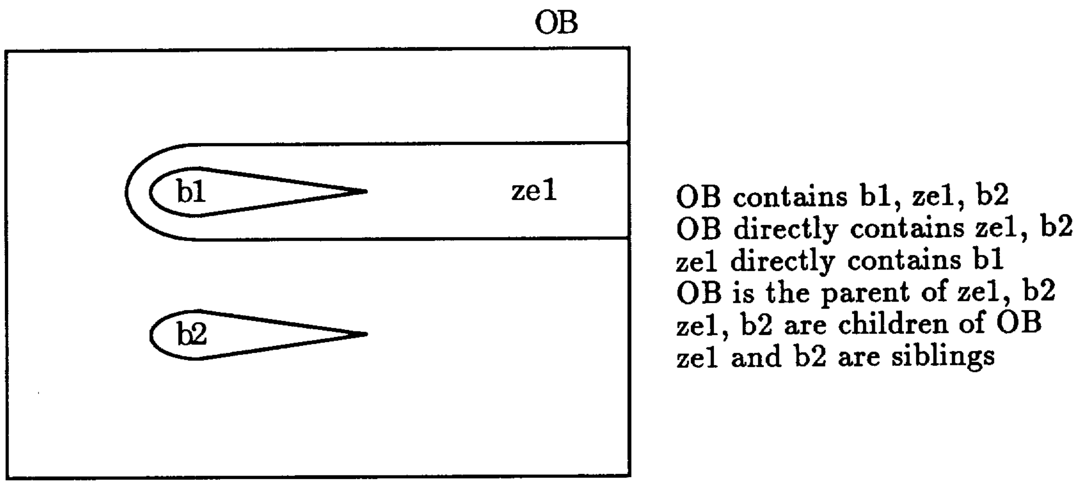
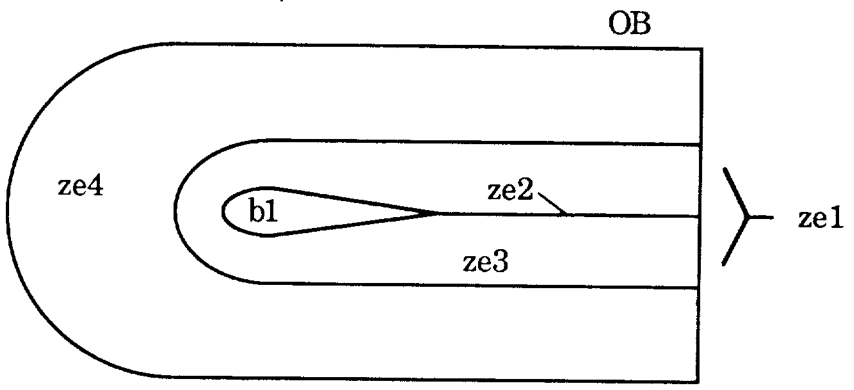

OB
b1
ze1
b2
OB contains b1, ze1, b2
OB directly contains ze1, b2
ze1 directly contains b1
OB is the parent of ze1, b2
ze1, b2 are children of OB
ze1 and b2 are siblings

Figure 2.3. – Object relations.

1. A zoning environment (z.e.) **contains** an object if each curve on the curvelist of the object is either completely inside the perimeter of the z.e., or coincident with one of the z.e. curves. If all of the object curves are coincident with z.e. curves, then at least one of the object curves must have the same directionality as its coincident z.e. curve in order to ensure containment.
2. A z.e. **directly contains** an object if it contains it, and there is no intermediate object which both is contained by the z.e. and contains the object.
3. A z.e. is a **parent** if it directly contains an object.
4. An object which is directly contained by a z.e. is a **child.**
5. If a z.e. directly contains more than one object, those objects are **siblings.**
6. The **corebody** of a z.e. is the group of bodies directly contained by the z.e. Its **coretype** is single if there is only one child body, and multiple if there are more than one.
7. The **contained environments** of a z.e. are the group of zoning environments directly contained by the z.e. (analogous to corebody for bodies).



b1 is an original body
ze1 is a virtual body
OB directly contains ze1, ze4
ze1 and ze4 are the contained environments of OB
ze1 is the corebody of OB
OB is filled by ze1 and ze4
ze1 directly contains b1, ze2, ze3
ze1 is filled by b1, ze2, ze3
OB is connected
ze3, ze4 are empty
ze2 is degenerate
ze3 and ze4 are zones
ze2, ze3, ze4 are zoned
ze1 is zoned (recursively)
OB is zoned (recursively)

Figure 2.4. – Variable object properties.

8. A z.e. is **connected** if at least one curve belonging to each child object of the z.e. is either shared by the z.e. or intersects a curve on the curvelist of the z.e.
9. A z.e. is **filled** if each curve belonging to its child objects is shared by another child or by the z.e. itself.
10. The **filling objects** of a z.e. are the group of objects which are directly contained by the z.e. if it is filled.
11. A z.e. is **empty** if it contains no objects.
12. A z.e. is **degenerate** if it encloses no area (i.e., for each curve on its curvelist, the reverse is also on the curvelist).
13. A z.e. is **zoned** if one of the following conditions is true (note the recursion in the third condition):
    a. It is a zone.
    b. It is degenerate.
    c. It is filled by 0 or more original bodies and 1 or more **zoned** zoning environments.

When OB, the zoning environment which consists of the entire computational domain, is **zoned**, then that candidate zoning is complete.

The vocabulary proposed here to describe zoning objects and situations is one of the two fundamental building blocks of the zoning language necessary for performing flow-field zoning interactively. The other building block is the vocabulary for describing zoning actions, presented in the next section.

2.3.2 Elements of a zoning action

The process of zoning a flow field has been modeled as one in which zoning actions are applied to zoning objects in order to construct a flow-field zoning. The process can also be viewed as one in which zoning environments are created and, in some cases (i.e., when they are empty), transformed into zones. It is difficult to communicate about zoning, other than graphically, because no language exists to describe zoning actions and their consequences. In this section, a conceptualization of zoning actions is proposed, leading to a vocabulary to describe them.

A zoning action can be considered to have an underlying *objective*, a *type* of application, and a means of *accomplishing* the objective. The underlying objective is the purpose of the action—for example, to connect two objects, to alter the shape of an object, or to separate objects. The type of application of the action depends on the relationships among the target objects or, if there is a single target object, on the way in which the action is to be applied to that object. Application types include parent-child, sibling-sibling, and self-exterior. Various means can be used for accomplishing an action. For example, forming a connection between two sibling objects can be accomplished by generating two curves which form the sides of a "bridge" between them, or creating an "enveloping" object which encloses one of the siblings, and which shares a curve with the other sibling.

One approach to describing a zoning action, then, would be to classify it according to objective, application type, and means of accomplishment—a three-part description. Instead, a scheme more consistent with common mathematical terminology has been adopted, in which a zoning action is composed of two elements: an operator and an operand. The objective and the means of accomplishing an action comprise its operator. The type of application and the actual objects to which it is to be applied make up the operand of the action. Below is a list of the ten possible zoning operators:

1. Connection/Bridge
2. Connection/Cut
3. Connection/Envelop
4. Connection/Wrap
5. Separation/Cut
6. Separation/Partition
7. Shape-Alteration/Envelop
8. Shape-Alteration/Wrap
9. Shape-Alteration/Cut
10. Identification/Identify

Note that most of the action objectives appear in more than one operator, as do many of the ways in which they are accomplished. Rather than describing each operator as a whole, each of the operator components will be described separately.

Action operators: objectives

1. Connection – action which connects two or more separate objects, or connects two different locations on or within a single object.
2. Separation – action which separates two or more sibling objects so that they are afterwards contained by different parent objects.
3. Shape Alteration – action which modifies, actually or virtually, the shape of an object.
4. Identification – action which elevates an empty, connected, unidentified region to the status of an object so that it can be acted upon (refer back to discussion of ways to view a zoning situation).

Action operators: how they are accomplished

1. Bridge – action accomplished by creating two new curves (forming the sides of the "bridge") which, with selected existing curves belonging to the objects on either side of the bridge, form a closed loop. A degenerate bridge involves the creation of only one new curve; the other side of the bridge is formed by the reverse of that curve, and no area is enclosed.

2. Cut – action accomplished by creating a single new curve which, with all of the existing curves belonging to the object or objects intersected by the new curve, form one or two closed loops.
3. Envelop – action accomplished by creating one or more new curves which, with zero or more selected existing curves, form a closed loop containing an object.
4. Wrap – action accomplished by creating one or more new curves which, with one or more selected existing curves, form a closed loop adjacent to and sharing one or more existing curves with an object.
5. Partition – action accomplished by creating one or more new curves which, with all of the existing curves of the parent object, form two or more closed loops, each containing one child object.
6. Identify – action accomplished by forming one or more closed loops from selected existing curves belonging to the parent object and its child objects.

To conclude the description of zoning action elements, zoning operands (the part which consists of the application type only) are described below:

1. Parent-Child – action applied between parent object and child object.
2. Sibling-Sibling – action applied among sibling objects.
3. Self-Interior – action applied within the interior of a single object.
4. Self-Exterior – action applied to the exterior of a single object.
5. Self-Actual – action applied to an object to affect the perimeter of the object directly.
6. Self-Virtual – action applied to an object to affect how the perimeter of the object is perceived by the parent of the object.
7. All-Objects – action applied to all objects contained by the same parent.
8. Region – action applied to an unidentified region (the only action that can be performed on an unidentified region is that of "identifying" it—all other actions can be applied to objects only).

Figure 2.5 contains example illustrations of each zoning action. A list of all possible zoning actions (operator-operand combinations), including a description of conditions under which each action is possible and the consequences of each action, is found in appendix A.

There is one more zoning event which is necessary to complete the transformation of a zoning environment into a zone, and that is "topology imposition." It is not called a zoning *action* because it affects only the type of the object in question, creates no new objects, and changes no object relationships. A zoning environment that is empty is a candidate for topology imposition, which is defined as the selection of the topological corners of the zone (there must be four corners for the composite block approach). There are five types of zone topologies provided in this zoning language:

1. C-topology (one family of coordinate lines are C-shaped).
2. O-topology (one family of coordinate lines are O-shaped).
3. H-topology (the two families of coordinate lines form an H).
4. Identifiable Corners (the topological corners are chosen to be the four vertices whose internal angles are closest to 90°).
5. Arbitrary (any four points can be chosen as zone corners).

The language for objects and actions in the zoning world presented thus far is complete enough to support the description of most zoning situations and processes arising in aerodynamic applications. An aerodynamic flow-field zoning which cannot be described as resulting from the composition of a sequence of these zoning actions has not been encountered.

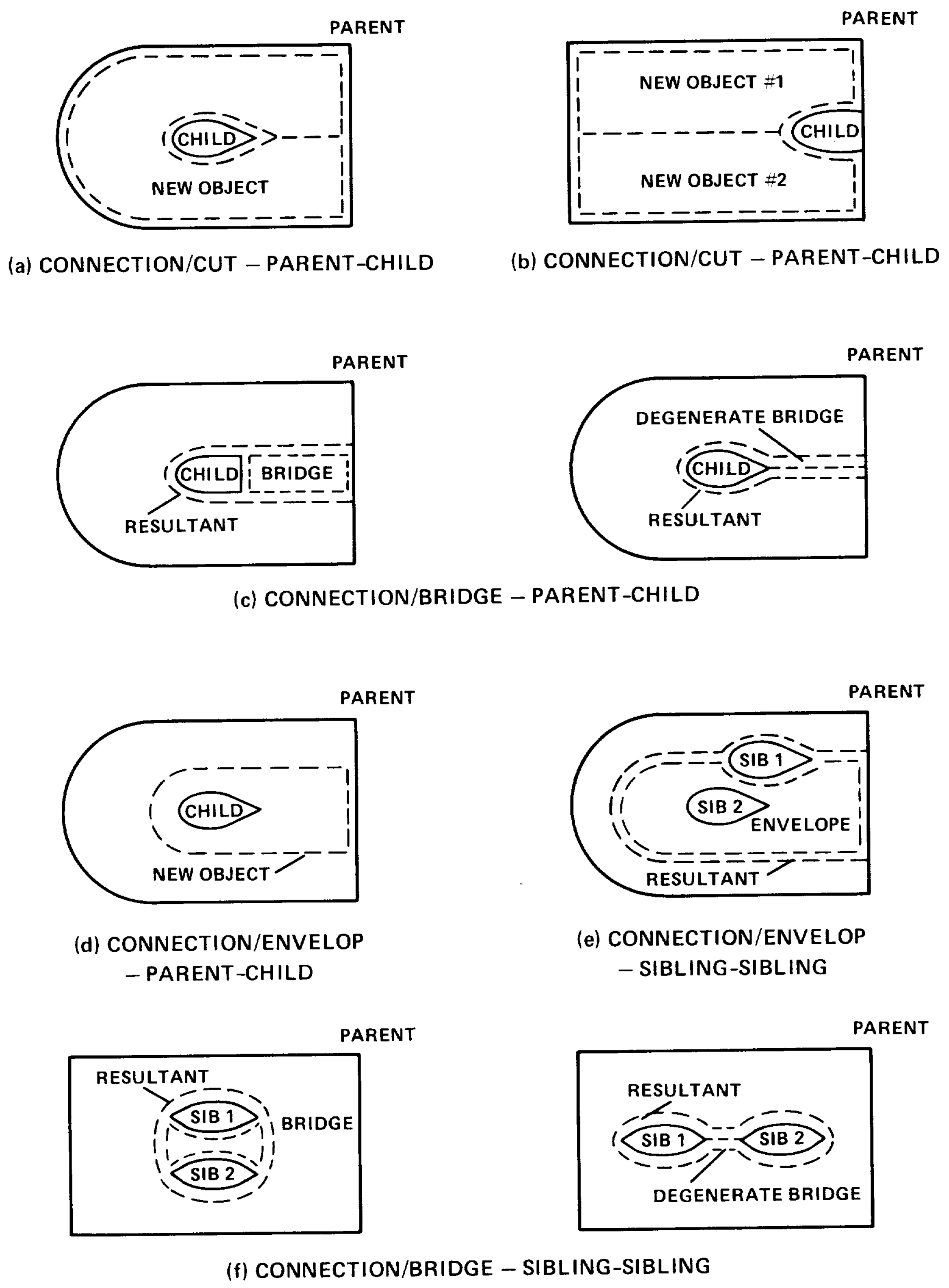
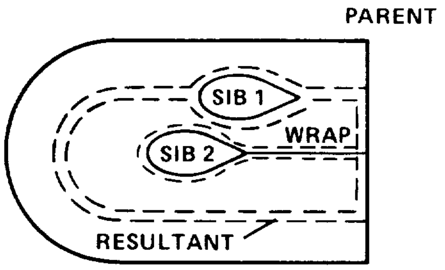
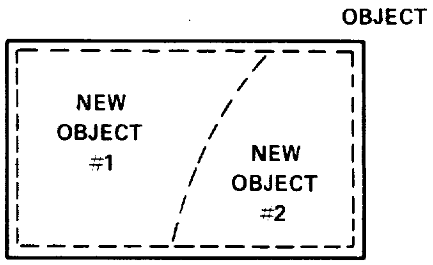


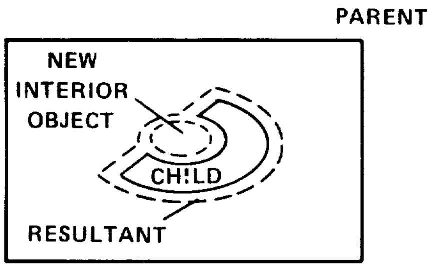
Figure 2.5. – Examples of zoning actions.



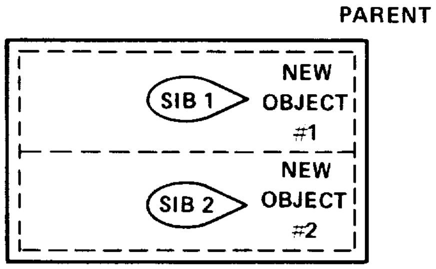
(g) CONNECTION/WRAP – SIBLING-SIBLING



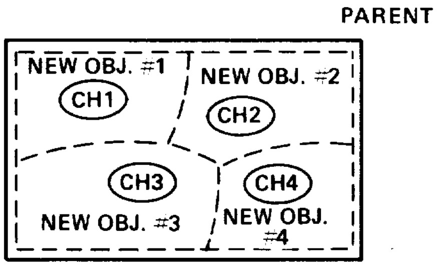
(h) CONNECTION/CUT – SELF-INTERIOR



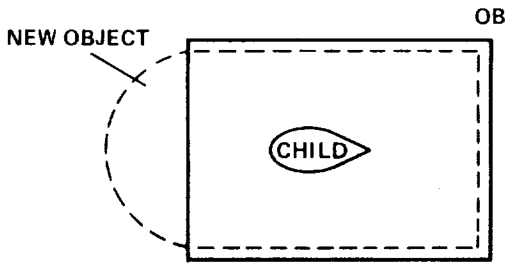
(i) CONNECTION/CUT – SELF-EXTERIOR



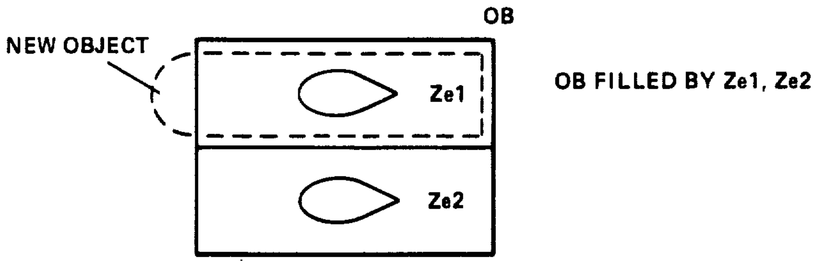
(j) SEPARATION/CUT – SIBLING-SIBLING



(k) SEPARATION/PARTITION – ALL-OBJECTS

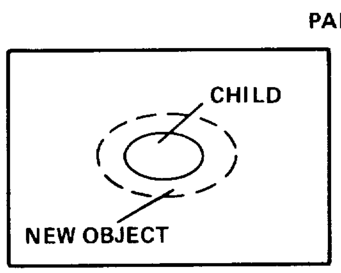


(l) SHAPE-ALTERATION/ENVELOP – SELF-ACTUAL

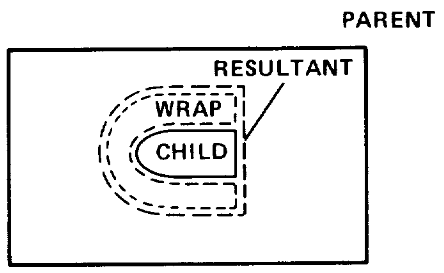


(m) SHAPE-ALTERATION/ENVELOP – SELF-ACTUAL

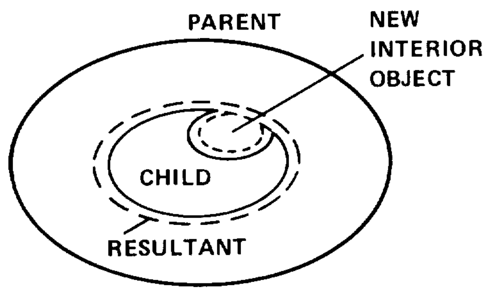
Figure 2.5. – Continued.



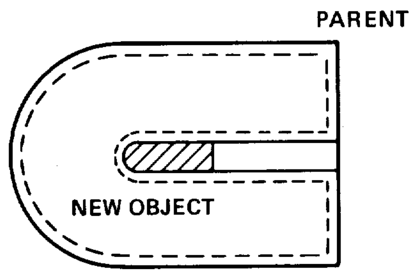
(n) SHAPE-ALTERATION/ENVELOP
– SELF-VIRTUAL



(o) SHAPE-ALTERATION/WRAP
– SELF-VIRTUAL



(p) SHAPE-ALTERATION/CUT
– SELF-VIRTUAL



(q) IDENTIFICATION/IDENTIFY
– REGION

Figure 2.5. – Concluded.

2.3.3 Describing shape and configuration qualitatively

The two elements of the proposed zoning language described earlier are fundamental building blocks needed for communication about zoning, and hence for interactive zoning. The remaining three elements, beginning with a language to describe shapes and configurations qualitatively, involve concepts and terminology necessary to *automate* flow-field zoning on a computer.

To automatically design a flow-field zoning, the computer program requires qualitative shape and configuration information. This information may be obtained in one of two ways: interactively (directly from the user) or automatically through extensive processing of the raw geometric data input by the user. Interactive input is the method chosen for automated flow-field zoning for the following reasons:

1. Data processing to educe qualitative information can be time-consuming, even in cases which are simple and obvious to a user.
2. The shape distinctions typically resulting from such processing are finer than necessary for this application.
3. The qualitative shape of an object, as viewed by a user, is a matter of interpretation, and can radically affect the resulting zoning design.
4. Having the user describe the configuration of input zoning objects permits the user to impose his or her own focus on object groupings, serving as a way to decouple portions of the problem where possible (objects which are far apart or separated by one or more other objects may have little influence on each other in terms of how the zoning is designed).

Interactive flow-field zoning also requires shape and configuration information, but it is provided by the user implicitly and throughout the zoning design process; the user does the visual processing and uses the results to directly make the zoning action and execution decisions. In automated flow-field zoning, that information is provided by the user *explicitly* and only at the outset of the zoning process, thereby allowing the system to make those decisions. To facilitate the interactive input of shape and configuration descriptions, a simple shape and configuration language is proposed which is based on the idea (Brady and Haruo, 1984) that all shapes have identifiable subshapes. In flow-field zoning, object shapes are composed of one or more primitive parts.

Each object part is described in terms of various attributes, such as type, orientation, ends, sides, shape, length, and width. The user must supply the part type, orientation, end, and side description. The other attributes are derived from these. The **type** of a part is either positive or negative, where a negative part is one which contributes negative area, i.e., area that is subtracted from the overall composition of parts (cavities are often described as negative parts). The **orientation** of a part is either horizontal or vertical. Each part has two **ends** (front and back), and two **sides** (top and bottom). While any shape could be represented as being composed of parts described in terms of ends and sides, such a scheme favors elongated shapes, such as are found in aerospace vehicles. For a horizontally oriented part, the front is the upstream-most end of the part. A vertically oriented part is assumed to face upward, so its front is located at or near the point of maximum y-value of the part (where y is the usual cartesian coordinate).

Both qualitative and quantitative values are associated with ends and sides. Ends (front and back) are described qualitatively as either blunt, sharp, or base, as shown in figure 2.6. Sides (top and bottom) can be straight, convex, or concave. The quantitative values determine the approximate dimensions of the part. They are supplied by the user, and are equal to the coordinate locations of representative points for each end and side. A blunt end is usually represented by its extremum. The representative point for a sharp end is usually the location of the sharp vertex. A base end has two representative points, one to indicate the top corner of the base, and one to indicate the bottom corner. Convex and concave sides are represented by one point, usually an extremum. A straight side needs no representative point to contribute dimension information. Selection of representative points has been made easy for the user in the implementation of this scheme, as is discussed in chapter 3.

Attributes with derived values, as opposed to input values, complete the list of attributes required to describe an object part in this language. The **shape** of each part is derived from combining the qualitative descriptions of the part ends and sides. Shapes based on common end/side combinations are given names, such as ellipse (both ends blunt, both sides convex) and bullet (one end blunt, one end base, both sides convex). Table 2.1 shows a list of commonly occurring shapes and the associated end/side combinations. Finally, each part has a **length** and a **width** which are calculated from the information provided by the representative point locations.

It is not sufficient to provide the above attributes for each of the parts which constitute the shape description of an object. A description of the manner in which the parts fit together must also be provided. The intersection of one part with another is called a **join**, a concept proposed by Brady and Haruo (1984) which provided the inspiration for the simpler version presented here. A join is described from the perspective of both of the parts involved. For each part, two join attributes are specified: its location on that part (which end or side is joined to the other part of the pair), and its position on that end or side. The part location attribute can have the values front-end, back-end, top-side, and bottom-side. The end/side position attribute can have the

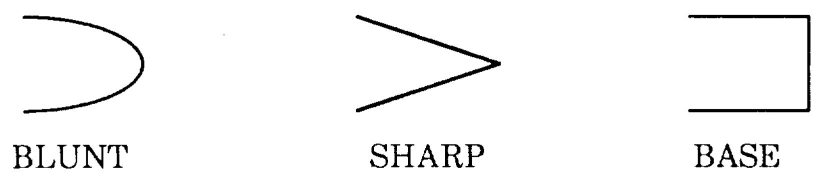
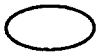
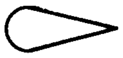
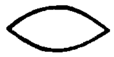
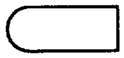
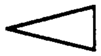
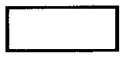


Figure 2.6. – Qualitative end descriptions.

Table 2.1 Common Part Shapes.

shape		end 1	end 2	sides
ellipse		blunt	blunt	convex
teardrop		blunt	sharp	any
eye		sharp	sharp	any
bullet		blunt	base	convex
wedge		sharp	base	any
rectangle		base	base	any

values at-corner, near-corner, centered, and entire-boundary. If the end/side position of the join has one of the first two values, the other end or side adjacent to the corner must also be specified. Figure 2.7 shows an example of an airfoil which has been represented as the composition of two primitive parts, one with a bullet shape, one with a wedge shape. The join between part1 and part2 is specified as ((part1 back-end at-corner top-side) (part2 front-end entire-boundary)).

Further characterization of the shape of an object is possible when the entire group of parts and joins needed to describe an object shape have been specified. By examining the joins, object protrusions can be identified. A **protrusion** is defined as a part (or group of parts connected by entire-boundary joins) whose entire boundary is joined to another part along only a portion of the boundary of that part. Object **cavities** are identified by examining all part types. Any part that is negative is considered a cavity. The **leading edge** and **trailing edge** of an object are identified, respectively, by finding the front-end of the upstream-most part and the back-end of the downstream-most part. Depending on the location of the representative point chosen for each end, the leading edge and trailing edge may not coincide with the upstream and downstream extrema of the object. Finally, the **type** of an object is derived from the number of parts needed to describe its shape. Possible object types are simple airfoil (describable with only one primitive part), compound airfoil (more than one primitive part necessary), simple blade (same as simple airfoil except for internal flows), and compound blade (same as compound airfoil except for internal flows).

The description of a configuration of objects makes use of the idea of primitive parts presented above, but is based primarily on the idea that objects sometimes occur in natural groupings. It is

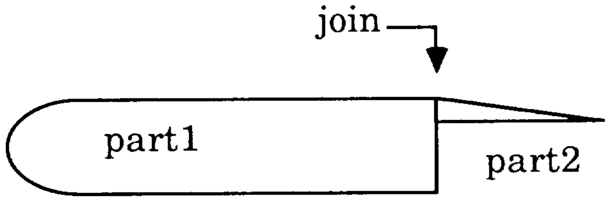


Figure 2.7. – Parts and joins.

not known just how to define what is meant by "natural" here—perception, geometric reasoning, and fluid dynamics knowledge are all involved, but the relationships among them have not been studied. Configuration description, like shape description, is therefore done interactively. The user must specify object groupings such that each grouping contains at least one object, no object is contained by more than one grouping, and the objects in the same grouping are those which are most likely to have the greatest influence on each other. Once groupings are established, relationships and approximate separation distances among objects within the same grouping and among groupings must be described. This is accomplished by identifying the **proximal portions** of the objects or groupings in question, and then specifying the **relation** that exists between them.

The proximal portion of an object consists of the name of the object part closest to the other object of the pair, and the end or side of that part which faces the other object. The proximal portion of a grouping is simply described as if the grouping were a horizontally oriented primitive part, that is, with a front-end, a back-end, a top-side, and a bottom-side. Possible relation values are before, behind, above, and below. Approximate separation distances are provided by the user through the selection of visually nearest points (again, as with representative point identification, the implementation of this scheme makes it easy for the user to provide distance information). Figure 2.8 illustrates one way in which the objects representing a four-element airfoil can be grouped. Objects B1 and B2 are in groupings of their own, and B3 and B4 are grouped together. The primitive parts of each object are also shown, where part1 is denoted p1, etc. The points chosen for computing separation distances are shown as heavy black dots on the perimeters of the objects. For such a configuration and grouping, the following description might be provided:

Object relations:
(B3 part5 bottom-side) above (B4 part6 front-end)

Object separations:
(B3 point5) (B4 point6)

Grouping relations:
(grouping1 front-end) behind (grouping2 back-end)
(grouping3 back-end) before (grouping2 front-end)

Grouping separations:
(grouping3 B1 point1) (grouping2 B2 point2)
(grouping2 B2 point3) (grouping1 B3 point4)

Providing a mechanism for grouping objects based on the user's perception and knowledge of fluid dynamics permits a decomposition of the problem into simpler subproblems. This is an important foundation for the discussion of how expert zoning methodology is encoded, which is found in section 2.3.5.

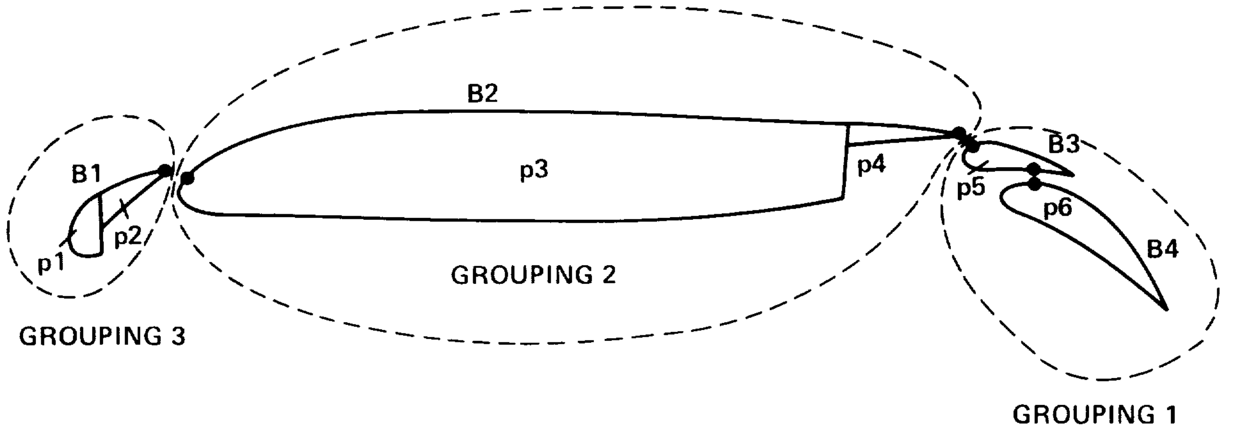


Figure 2.8. – Object groupings for a four-element airfoil.

2.3.4 Incorporating user bias

One of the problems associated with flow-field zoning is that the experts do not agree on what makes a good zoning. One way to deal with this problem in the development of a knowledge-based system might be to establish a standard set of guidelines. The drawback of this approach for zoning is the possibility that the standard could be totally unacceptable to some experts, and totally acceptable to none. Individual bias has been identified as playing a role in the way an expert designs and evaluates a zoning. A solution to this dilemma is to establish a *tunable user bias profile*. The user can tune the profile to reflect her or his own bias.

The criteria for designing and evaluating flow-field zonings can be categorized as objective or subjective. Objective criteria include basic zoning guidelines, such as that zones (the *block* variety) are empty and topologically four-sided, zones abut without gaps rather than overlap, zonal boundaries do not cross each other or the boundaries of the input geometry, and the outer boundary location depends on the physical conditions of the problem. Subjective criteria define what is commonly referred to in expert systems parlance as "standard practice." As noted above, zoning practice is not standard. These subjective criteria, which depend on an individual user's bias, form the basis for the tunable user bias profile.

The bias a user brings to a zoning design problem involves a variety of factors: (1) the particular capabilities of the user's flow-solver code–for example, how boundary conditions and singularities are handled, what sort of turbulence model is used, and what effect grid skewness has on the robustness of the code; (2) the user's experience (often with a specific flow solver), which determines the user's threshold of tolerance of inaccuracies caused by grid skewness, discontinuities, singularities at body surfaces, and zonal boundary intersections with body surfaces; (3) the user's objectives for the problem at hand–for example, is good quantitative accuracy required (and for which parameters) or do trends suffice? and (4) aesthetics–"I don't like the look of that discontinuity in the boundary curve."

To incorporate a user's bias into the design and evaluation processes, it must first be parameterized. The profile is defined as the collection of parameters chosen to represent zoning user bias, and is tuned by the assignment of qualitative weights to each parameter, reflecting the importance and/or acceptability of that parameter. The parameters identified as representing a user's bias are

1. Zoning **simplicity** – It is usually desirable to have few zones and zonal interfaces since zonal interface conditions are not vectorizable, and their encoding in the flow solver requires some extra work. In some cases, however, an emphasis on simplicity would compromise the quality of the zoning in terms of grid and solution features.

2. Zone corner **skewness** – Skewness in the sides of a zone often translates directly into zonal grid skewness, which can result in decreased flow-solver accuracy and robustness. Some amount of skewness is often tolerated in order to gain a simpler zonal topology.

3. Zone side **smoothness** – Discontinuities in the zonal interfaces often translate directly into discontinuities in the zonal grid coordinate lines, which can decrease the accuracy of the flow solution. As with skewness, some amount of discontinuity can sometimes be tolerated in order to achieve other zoning advantages.

4. Zone side **mapping** disparity – A large difference between the maximum and minimum distances separating opposite sides of a zone usually leads to difficulty in grid generation and the appearance of zonal grid discontinuities. Sometimes other factors (such as simplicity) can outweigh the perceived disadvantages of mapping disparity.

5. Grid point **efficiency** – The potential grid point efficiency of a zonal composite grid is related to the extent to which zonal interfaces share curves with original bodies. While efficiency is one of the motivations for using a composite zonal approach, emphasis on other aspects of the zoning might take precedence in some cases.

6. **Orthogonality** at body surfaces – The importance of grid line orthogonality at body surfaces depends on the type of numerical scheme and turbulence model used in the flow solver,

and the level of accuracy desired in the solution. The orthogonality of zone corners and the smoothness along zone sides sharing curves with original bodies affects the orthogonality of the grid.

7. **Surface vs. field** quantities – The influence of a flow-field zoning on the composite zonal grid in terms of either emphasizing (i.e., concentrating grid nodes near) geometry surfaces or the regions away from the surfaces is related to the extent to which entire zone sides share curves with original bodies (similar to the efficiency parameter).
8. **Wake** resolution – The placement of zonal interfaces can help or hinder the potential for zonal grid resolution of wakes. The importance of wake resolution is influenced by the computational objectives of the user.
9. Zone **tuple** points – Allowing three or more zones to intersect at the same point (a tuple point) can sometimes facilitate the design of a zoning, but can result in increased coding effort for the zonal interface conditions.
10. **Singularities** at body surfaces – Avoiding topological singularities at body surfaces is one of the motivations for adopting a composite zonal approach. Judiciously permitting some singularities at bodies can simplify the zoning design, however, and should be left to the user's discretion.
11. Zone/body **intersections** – The spatial accuracy of the flow solution is decreased by one order at zonal interfaces. Allowing zonal interfaces to intersect bodies can simplify zoning design, but can degrade the flow solution depending on the locations of the intersections.
12. **Viscosity** in more than one direction – This is a question of flow-solver capability rather than user discretion. If the form of the equations and the turbulence model are able to handle viscosity in two coordinate directions, both normal and tangential to body surfaces, then the zoning need not be limited to zones which share at most one side with a body.

A list of these 12 zoning parameters and their possible weight values is found in table 2.2. The user bias profile is intended both to guide the design of zonings and to evaluate completed zonings. The qualitative values are used to guide design decisions, and the quantitative values which correspond to them are used in the evaluation of zonings. A detailed description of the profile parameterization, calibration, and validation are presented in chapter 5.

2.3.5 Encoding expert zoning methodology

According to the model of flow-field zoning proposed in section 2.2, a zoning is *designed* through the stepwise application of zoning actions to zoning objects until an acceptable solution is constructed. Thus, any zoning can be defined by a sequence of zoning actions. It is in the determination of this sequence of actions that zoning expertise lies. To automate flow-field zoning, it is necessary to automate either the choice of an action at each stage of the design, or the construction or selection of the entire sequence of actions at the outset. The latter approach was adopted for two reasons. First, it is difficult to choose the best zoning action at an intermediate stage without knowledge of the actions which preceded it and which will follow it. Second, action decisions at intermediate stages require complete qualitative shape and configuration information, even for newly generated zoning objects. That information is guaranteed to be complete only for the initial situation since that is the point at which the user provided it, and the means for extracting the same information from new objects at later stages depends on the way in which the new objects were formed. The first reason is explained in more detail below.

Consider the generate-and-test problem-solving paradigm described in section 2.2, which can generate all possible sequences of zoning actions. This procedure produces a *tree* of all possible intermediate and final (located at the *leaves* of the tree) zoning situations for a given problem. At an intermediate stage of the zoning, especially early on, there is no way to predict which branch is most likely to lead to the best solution. Figure 2.9 shows part of such a tree, where some of the zoning situations possible at the first stage of the design lie directly under the initial situation, and

Table 2.2 User Bias Profile Parameters.

Profile Parameter	Qualitative Weight Values
Simplicity	
Zone corner skewness	
Zone side smoothness	*NO*
Zone side mapping disparity	*LOW*
Grid point efficiency	*MEDIUM*
Orthogonality at body surfaces	*HIGH*
Surface vs. field quantities	*(IMPORTANCE)*
Wake resolution	
Zone tuple points	*ALLOWED BUT NOT IMPORTANT*
Singularities at body surfaces	*ALLOWED*
Zone / body intersections	*SOMEWHAT DISCOURAGED*
Viscosity in more than one direction	*DISCOURAGED*
	STRONGLY DISCOURAGED
	NOT ALLOWED

four final candidate solutions based on situations (a) and (b) are found at the bottom. Situation (b) has the potential of leading to one of the simplest or one of the most complicated zonings of the four shown, so if zoning simplicity is considered important by the user, the path including situation (b) should be followed only if future action choices result in the candidate solution on the left. Suppose the user has tuned the zoning profile so that zone side mapping disparity is of high importance (and should therefore be discouraged). Either situation (a) or (b) could lead to a solution which has little mapping disparity or to one which has great mapping disparity. Again, any decision must take into account future action choices.

Constructing or selecting a sequence of actions at the outset of the zoning design instead of choosing a single action at each intermediate stage avoids the problem of needing information about unknown future actions, and eases the requirements for updated qualitative shape and configuration information. The next issue is whether to construct or select the action sequences. Selection of whole action sequences is the most straightforward approach, but requires that the knowledge base of the system contain predefined sequences for all possible zoning problems. A method of constructing action sequences is the more practical solution.

To describe zoning action sequences and their construction, the following language is proposed. A sequence of zoning actions which leads to a zoning design for a given configuration is called a **plan**. A plan is constructed, or composed, of one or more **subplans**. The subplans are primitive plan elements which are predefined, and reside in the system knowledge base. A subplan consists of the sequence of zoning actions necessary to zone a single grouping of objects. Also, associated with each action are one or more **strategies** which provide guidance in the execution

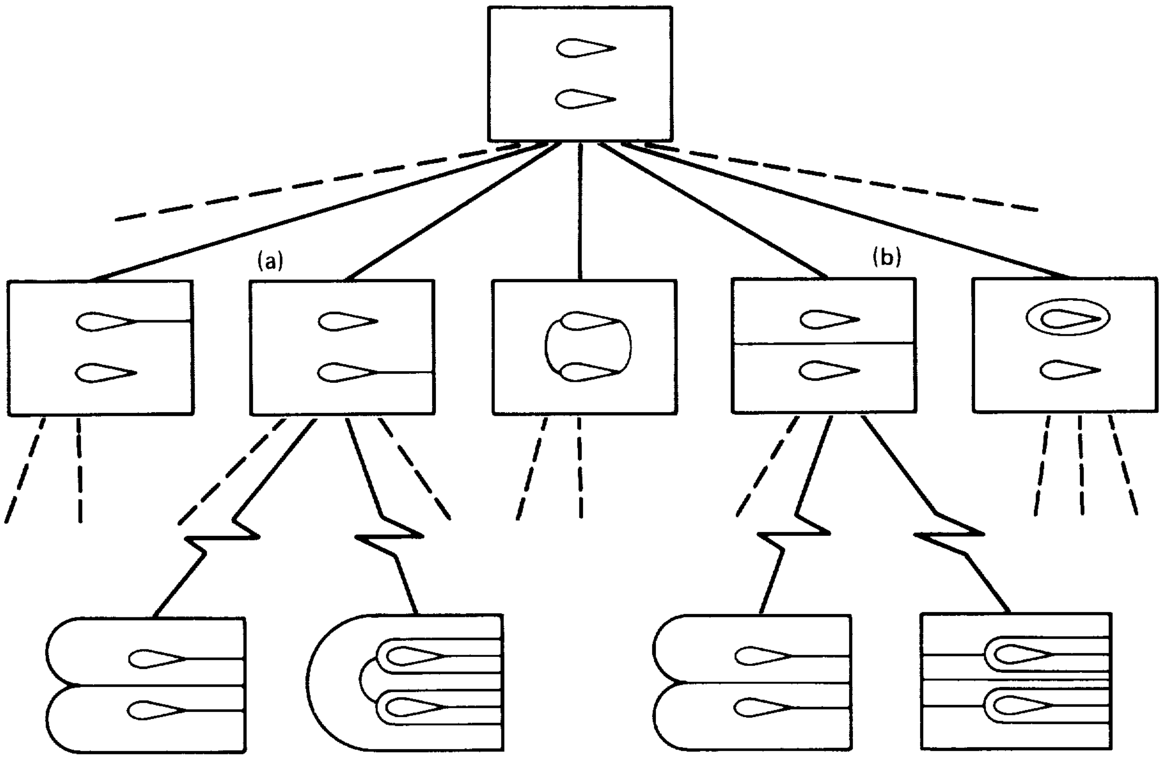


Figure 2.9. – A tree of possible zoning candidates.

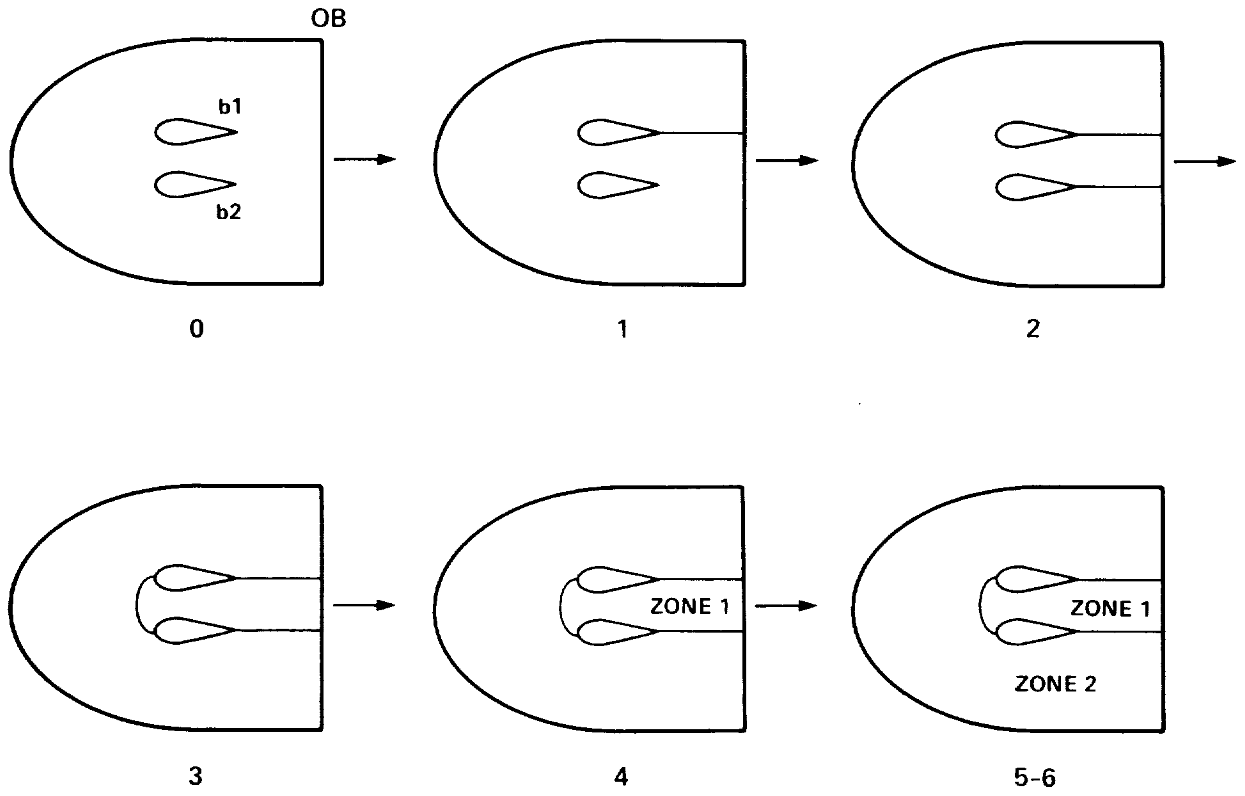
of that action. If a configuration is described by only one grouping, the plan is equivalent to the subplan selected for that grouping under the conditions of the particular problem. If the qualitative configuration description is based on more than one grouping, the selected subplans must be assembled to obtain the final plan. Subplan assembly is made possible by dividing the actions comprising a subplan into three different categories:

1. **Primary Actions** – These actions are applied to objects within the same grouping, and comprise the bulk of most subplans. They are usually connection- or shape-alteration-based actions which do not result in filling the OB (outer boundary) zoning environment.
2. **Secondary Actions** – These actions are applied to objects within different groupings, and usually either connect or separate the two groupings. They do not result in filling the OB zoning environment.
3. **Completion Actions** – These are the actions which fill the OB zoning environment and complete the zoning design. They typically include identification actions and topology imposition.

The method of combining subplans for the case of multiple groupings depends on the qualitative relationships between those groupings. There are three basic types of relationship: (1) no relationship, i.e., the groupings are contained by different parent objects; (2) vertical alignment, where one grouping is above the other; and (3) horizontal alignment, where one grouping is upstream of the other. In the case of unrelated groupings, subplans are simply joined together with no modification or rearrangement. Subplans for groupings which are vertically aligned may require the selection of an additional subplan containing secondary actions. If secondary actions are not needed, as is the case when the objects in the groupings do not need to be either connected or separated (this usually occurs when H-topologies surround the objects), the primary actions of the subplans selected for each grouping are combined, and the completion actions from each subplan are combined and appended to the primary actions. If secondary actions *are* needed, they are appended to the primary actions, and the completion actions belonging to the subplan containing the secondary actions are appended to the previous combination. In the final case of horizontal grouping alignment, the primary actions of the upstream groupings are modified so that the application type of the operand is changed from parent-child to sibling-sibling wherever applicable. The plan is assembled starting with the primary actions of the downstream-most grouping, followed by the modified primary actions of each successively upstream grouping, followed by a combination of the completion actions of all the subplans.

Figure 2.10 shows an example assembly of a plan for a vertically aligned pair of simple airfoils. The configuration description has the two airfoils in two different groupings. The conditions required for the selection of these three subplans included airfoil shape descriptions involving blunt leading edges and sharp trailing edges; a bullet-shaped OB zoning environment; inviscid flow solution sought; and user bias in which simplicity is important, zone/body intersections are allowed, singularities at bodies are allowed, and wake resolution is important.

Most of the zoning design expertise which has been captured is encoded directly in the predefined subplans. The methods by which the subplans are combined when more than one grouping is present contain the remainder of zoning design knowledge. This approach is efficient in the sense that it obviates the need to add new subplans for each new configuration. It is general because a variety of complex zoning problems can be zoned using plans composed of subplans for simpler problems. The design of a zoning has been transformed into an analytic problem solvable by breaking down the configuration into simpler groupings when possible, selecting subplans for each grouping, and assembling the subplans into a coherent zoning plan.



subplan sp1: (for first body)
primary actions:
1. connection/bridge – parent-child
   (strategies: degenerate-zone, follow-wake)

completion actions:
1. identification/identify – region
2. topology-imposition/impose-C

subplan sp2: (for second body) – same as sp1

subplan sp3: (to connect two bodies)
secondary actions:
1. connection/bridge – sibling-sibling
   (strategy: fill-in-bridge)
2. topology-imposition/identify-corners

completion actions:
1. identification/identify – region
2. topology-imposition/impose-C

final plan:
1. connection/bridge – parent-child/(OB b1)
2. connection/bridge – parent-child/(OB b2)
3. connection/bridge – sibling-sibling/($ze1 $ze2)
4. topology-imposition/identify-corners
5. identification/identify – region/($ze3)
6. topology-imposition/impose-C

Figure 2.10. – Subplan assembly example.

CHAPTER 3

EZGRID: A KNOWLEDGE-BASED SYSTEM FOR TWO-DIMENSIONAL FLOW-FIELD ZONING

A knowledge-based system called EZGrid (Expert Zonal Grid generator) has been developed for both interactive and automatic flow-field zoning of 2-D problems in CFD. This chapter provides the details of the implementation, beginning with the strategy that was employed. The programming languages and data representation schemes used in EZGrid are discussed in section 3.2, followed by a section outlining the structure and flow of control of the program. Section 3.4 completes this chapter with a description of the EZGrid knowledge base.

3.1 Implementation Strategy

The difficulties which arise in any attempt to automate flow-field zoning have already been described. The process of zoning is poorly understood and difficult to describe except via pictures. It has a distinctly perceptual element, not only in any description of the process, but also in the design decisions that are made throughout the process. Experts do not agree on what qualities determine a good zoning, indicating an element of individual bias. Finally, zoning has been modeled as a design process, which is a problem-solving approach for which there are few examples of successful knowledge-based systems, and hence less guidance available. The implementation strategy that was developed to meet these challenges involved two phases: the first phase was the implementation of the zoning model in terms of the two most basic elements of the zoning language–zoning objects and actions–resulting in an interactive system for flow-field zoning. The second phase was the incremental addition of the remaining zoning language elements–shape and configuration, user bias, and zoning design expertise–to enable automation of the process.

The interactive flow-field zoning system resulting from the first phase of development is somewhat automated. Use of the zoning language to describe objects and actions made possible automation of the mechanics and bookkeeping associated with flow-field zoning. The interactive system is able to

1. Compute the limits of the outer boundary of the computational domain using heuristics involving speed regime, flow steadiness, angle of attack, far-field boundary condition type, and desired level of accuracy
2. Deduce facts about the situation at each stage (for example, which objects directly contain other objects; whether an object is filled, empty, connected, or zoned; the contents of each object; and which curves are shared by more than one object)
3. Determine which zoning actions are appropriate in any given situation (for example, an action with a separation-based operator is meaningless in a situation where there is only one body)
4. Determine what objects are needed to execute an action and which parameters are relevant for the specification of the object curves
5. Execute the action by generating the curves (using the curve parameter information provided by the user), constructing the objects, computing new object attributes, and establishing new object relations

When using the interactive system, the *user* must supply the parts of the zoning language which are more difficult to automate:

1. Translation of perception of object shapes and relations into an internal qualitative description, which helps the user to make zoning design decisions and to formulate and execute curve generation strategies

2. Incorporation of user bias into the design, construction, and evaluation of zonings
3. Application of expert zoning methodology to the choice of a zoning action at each stage of the design

To complete the automation of the phase I system, phase II required the implementation of the remaining elements of the zoning language, replacing those parts previously supplied by the user:

1. The language to describe qualitative shape and configuration information was implemented as an interactive, menu-based input sequence occurring only during the setup phase of an automated system run (as explained in section 2.3.3, this element is not automated because of the scope of individual interpretation possible), making possible a qualitative representation internal to the computer.
2. User bias was parameterized and tested in the form of a tunable user bias profile, measurement functions for each parameter were developed and calibrated, and a simple input scheme was devised for parameter value assignment.
3. Subplans and action execution strategies incorporating good zoning practice, shape and configuration awareness, and user bias were developed incrementally.

This two-phase implementation strategy was found to be essential to the success of the system-development effort. By beginning with the two basic elements of the zoning language, zoning objects and actions, it was possible to separate their validation from that of the rest of the language, permitting them to be debugged independently and more efficiently. The resulting interactive system provided several further advantages: it formed a solid, reliable foundation on which to build the rest of the automated system, and it provided a platform on which to gain experience with manipulating objects through zoning actions, helping to identify which qualitative shape and configuration parameters are needed, what user bias factors affect zoning decisions, and what sequences of actions are best for a given situation.

3.2 Representation and Language

Equipped with a zoning model and language, a human zoning expert is better able to communicate his or her methodology to another human. However, to automate zoning on a computer, it is necessary to communicate this methodology in terms a computer can use. The representation challenge lies in describing zoning objects, situations, and actions at two different levels of abstraction. The lower level is numerical, and is where computations are performed to generate curves, find curve intersections, measure angles, calculate derivatives, and compute tangents and normals to curves. The higher level is symbolic, and is where decisions are made about what action to take in a given situation and what action execution strategy to use. A knowledge-based programming approach readily accommodates both levels of abstraction needed for this problem.

3.2.1 The role of C in EZGrid

The EZGrid knowledge-based system was implemented using three programming languages – C, Franz Lisp, and MRS. Each has different strengths; C is the foundation language of the entire system, because Franz Lisp is written in C, and MRS is written in Franz Lisp (other versions are based on Zetalisp, Common Lisp, and Interlisp), and because it is the language of choice on the Silicon Graphics Iris 2500 Turbo Workstation (a 68020-based machine with a Unix System V operating system), which is the hardware used for this development. Also, it is easier to program numerical operations in C than in either Lisp or MRS. A good source of information about C programming is the text by Kernighan and Ritchie (1978).

In EZGrid, body perimeter curves and zonal interface curves are generated using parametric cubic splines, so part of the representation of a curve involves a pointer to a numeric array containing the second derivatives of the cubic spline at each spline knot. The knots (x, y coordinate pairs) for each spline are stored in two large arrays, X and Y. The routines which generate the cubic splines (involving matrix inversion) are written in C, as are the routines which find the extrema of a cubic spline, the centroid of a cubic spline, and the first derivative at any point on a cubic spline.

3.2.2 Lisp: a basic AI language

Franz Lisp is a dialect of lisp, which is a list-processing language that provides convenient mechanisms for manipulating lists of arbitrary symbolic elements. Lisp is a functional language, and thus promotes program modularity. It is an interactive language which can also be compiled. One of the features of lisp which makes it particularly appropriate for the development of intelligent systems is its uniformity (Winston and Horn, 1981)—functions and data have the same form. This means that a lisp function can construct and analyze another lisp function. Lisp has a built-in data structure called a property list, which associates attributes with values, and can be attached to any atom (a word-like, nonlist object in lisp). Lisp makes it easy to generate a virtually unlimited number of new variable names during execution, and memory can be allocated as needed (list lengths are not specified ahead of time, as the length of an array must be). Recursion is permitted in lisp, which also facilitates list processing. Good references for lisp programming are the Winston and Horn text (1981) and Wilensky (1984).

Lisp property lists are used to represent EZGrid object properties which are invariant. Each object has on its property list

1. A curvelist (literally, a lisp list of curve names)
2. A vertexlist
3. A sidelist (a lisp list of lists of curve names)
4. Object extrema – downstream-point, upstream-point, top-point, and bottom-point (where a *point* is represented in terms of how it was found, if possible, as well as its actual location; e.g., ((curve C1 0.37) (1.34 2.8)) is a point which was selected from curve C1 at $u = 0.37$ (nondimensional position), and has the coordinates x = 1.34 and y = 2.8)
5. Numvertices (an integer representing the number of vertices of the object)
6. A maxlength and a maxwidth (real numbers denoting the distance between the extrema in the x-direction and those in the y-direction, respectively)
7. A leading edge and trailing edge (represented as points, these may not be coincident with upstream and downstream extrema, depending on the qualitative shape description of the object and the flow specification)
8. A chord length (real number denoting the straight-line distance between the leading edge and the trailing edge)
9. A centroid (list containing a pair of x, y coordinate values)
10. A partlist (list of primitive part names which comprise the shape description of the object)
11. A joinlist (list of joins, where each join is a list composed of the part location and the end/side location for each part involved in the join)
12. A type ("simple" or "complex," depending on how many parts are required to describe the shape of the object)

Curves, vertices, and parts also have lisp property lists on which most of their attributes are stored. On a curve's property list is a property called "formula," which is a pointer to the array of cubic spline second derivatives, as well as other properties: type (fundamental or subcurve), indices (a list of two integers denoting the first and last indices in the X, Y arrays belonging to the knots of that cubic spline), begin-u and end-u, reverse, parent (only subcurves have a parent curve), and constituent curves. Each vertex may have stored on its property list indices, an in-curve and an out-curve, an included angle, and a direction (an angle in radians). On the property list of each primitive shape part are the properties: type (positive or negative), orientation (horizontal or vertical), front, back, top, and bottom (each a lisp list beginning with a qualitative description, such as blunt or straight, and ending with one or two points denoting the location of the end or side).

The C function which generates cubic splines is invoked from a lisp function which stores the resulting second-derivative array and other curve properties on the property list of the new curve. Functions which find closed loops formed by curves, subtract curves, reverse curves, divide curves, determine curve intersections, update curve connection lists, compute curve lengths, compute vertex angles, construct new objects, and do many more such tasks, are written in lisp. In addition, user interface routines and functions which provide overall system control in EZGrid are written in lisp.

3.2.3 Logic programming using MRS

MRS is a logic programming language that was developed at Stanford (Genesereth et al., 1984). The name MRS stands for Meta-level Representation System; although in a strict sense it is a programming language, the name emphasizes the underlying philosophy that a user "programs" in MRS by representing facts rather than by specifying a procedure (Russell, 1985). This is known as the *declarative* programming paradigm, in which a computer is told what is true (and must deduce an answer from that information) rather than how to do a task. MRS uses logical rules of inference (such as modus ponens), pattern matching, and simple search strategies (primarily depth-first) to perform logical deduction on facts and rules represented as prefix predicate calculus (PPC) propositions. Predicate calculus is a centuries-old mathematical language which has as its basic components object constants and variables, function symbols, relation symbols, logical connectives (operators), and quantifiers. Its precision and well-established syntax have made it attractive to the AI community as a method of knowledge representation for automated inference. PPC is simply a version of predicate calculus in which the quantifiers, operators, functions, relations, and constants are placed at the beginning of a proposition in a certain order. An example of a simple PPC proposition is (downstream airfoil1 airfoil2), which uses a relation and two constants to express the fact that airfoil1 is downstream of airfoil2. A more complex example is (if (and (parent $x) (female $x)) (mother $x)). Variables in MRS are all preceded by the $ symbol. This PPC proposition is equivalent to the English statement, "If x is a parent and x is female, then x is a mother." Most of the zoning knowledge contained by EZGrid is in the form of PPC propositions—knowledge about determining the outer boundary shape and limits, what actions to perform in certain situations, how to execute those actions, what objects result and how they are related to existing objects, and definitions concerning zoning objects and situations. Nilsson (1980) contains a clear and complete discussion of predicate calculus and various inference and search procedures. Russell (1985) provides an entertaining and helpful explanation of many of those concepts in the context of programming in MRS.

PPC propositions are used in EZGrid to represent object properties which can vary from one stage of the zoning design to the next. Object relations (containment and direct containment); status (whether it is a body, zoning environment, or zone, and whether it is filled, empty, degenerate, connected, or zoned); and functions (the corebody, coretype, contained environments, filling objects, numbodies, and numenvironments of an object) are all in this category. Examples include

```
(contains OB B1 (stage 1))
(body B1 (stage 1))
(zoning-environment ZE2 (stage 2))
(empty ZE2 (stage 2))
(zone ZE2 (stage 4))
(corebody OB (B1 B2 B3) (stage 1))
(corebody OB (B1 ZE1) (stage 2))
(contained-environments OB (ZE1) (stage 2))
```

```
(filled OB (stage 5))
(zoned OB (stage 5))
```

Each of the propositions above ends with a term that denotes the stage of the zoning design at which the statement is true. This is necessary since, as can be seen from the two propositions involving the corebody of OB, there would otherwise be conflicting statements in the database, resulting in nonsense being generated by subsequent inference processes. Curves and vertices also appear in MRS propositions which are stored in the system database. Curve connectivity relations and information about curve generation are represented propositionally, for example:

```
(new-connection (C1 (0.0 (C17 1.0))) (stage 3))
(connection-list ((C1 (0.0 (C16 0.0) (C17 1.0)) (0.3 (C8 0.0))) (C5 (0.492 (C16 1.0)))
    (stage 4))
(curve-needed C6 (type new) (object ZE3) (stage 2))
(curve-parameter C6 (begin-point ((curve C1 0.3) (1.23 -0.41))) (stage 2))
(curve-divided C2 (stage 3))
(vertices ZE2 done (stage 2))
```

The connection list proposition can be interpreted as follows: at stage 4, curve C1 has two curves which are connected to it at $u = 0.0$ (i.e., at the beginning of the curve), C16 at its beginning, and C17 at its end. Curve C1 also has the beginning of C8 connected to it at $u = 0.3$. Curve C5 has the end of C16 connected to it at $u = 0.492$. The connection list is produced by a lisp function from a collection of new-connection propositions and the connection list from the previous stage. Another lisp function uses the connection list in conjunction with zone curvelists to determine the existence and number of tuple points (points where more than two zones meet), which is used in zone design evaluation.

In order to understand how the three different languages complement each other, and how EZGrid works, it is necessary to look in somewhat more detail at the way MRS performs automated inference. In MRS, separate databases, called *theories*, can be established. In the "global" theory are predefined PPC propositions which define the basic MRS utilities (e.g., how to assert a proposition, how to determine membership in a set, how to determine whether a symbol is a variable and whether it is bound to a value). An MRS programmer usually sets up additional distinct theories: one or more for propositions which contain the domain knowledge of the program being created (this collection of theories is usually referred to as the rule base or knowledge base), and one or more for the propositions which represent facts that have been asserted (input) by the user or deduced by the system for the particular problem being solved (these theories are called the database).

As mentioned above, MRS uses logical rules of inference, primarily modus ponens, to automatically deduce facts from other facts and rules. Modus ponens can be described as follows: if there are two propositions (if A B) and A, then B can be deduced. Another way to state this is that if there is a rule that says "if A is true, then B is also true," i.e., B follows logically from A, and it is known that A is true, then B must be true. For example, given the rule (if (man $x) (mortal $x)) and the fact (man Socrates), MRS could deduce that Socrates is mortal.

Modus ponens can be used in both the forward and backward directions in MRS. In forward reasoning (called forward chaining), inference is data-driven, meaning that when a propositional fact is asserted into the database, the rules in the knowledge base are examined to see whether the antecedent portion (in the rule (if A B), A is the antecedent and B is the consequent) matches that fact. If variables are involved, they are bound to values which aid the match. When a match is found, the consequent proposition is asserted into the database, which triggers another search through the rules for matches of rule antecedents to that proposition, and the "chain reaction" continues. Figure 3.1 depicts the forward-chaining process graphically. What has just been described illustrates forward chaining done in a depth-first fashion, where as soon as one match is found and its consequent asserted, the chain continues through the consequent. Breadth-first forward chaining would proceed to find all matches to the initial fact before trying to match any consequent facts that are asserted. Depth-first search is the default strategy in MRS.

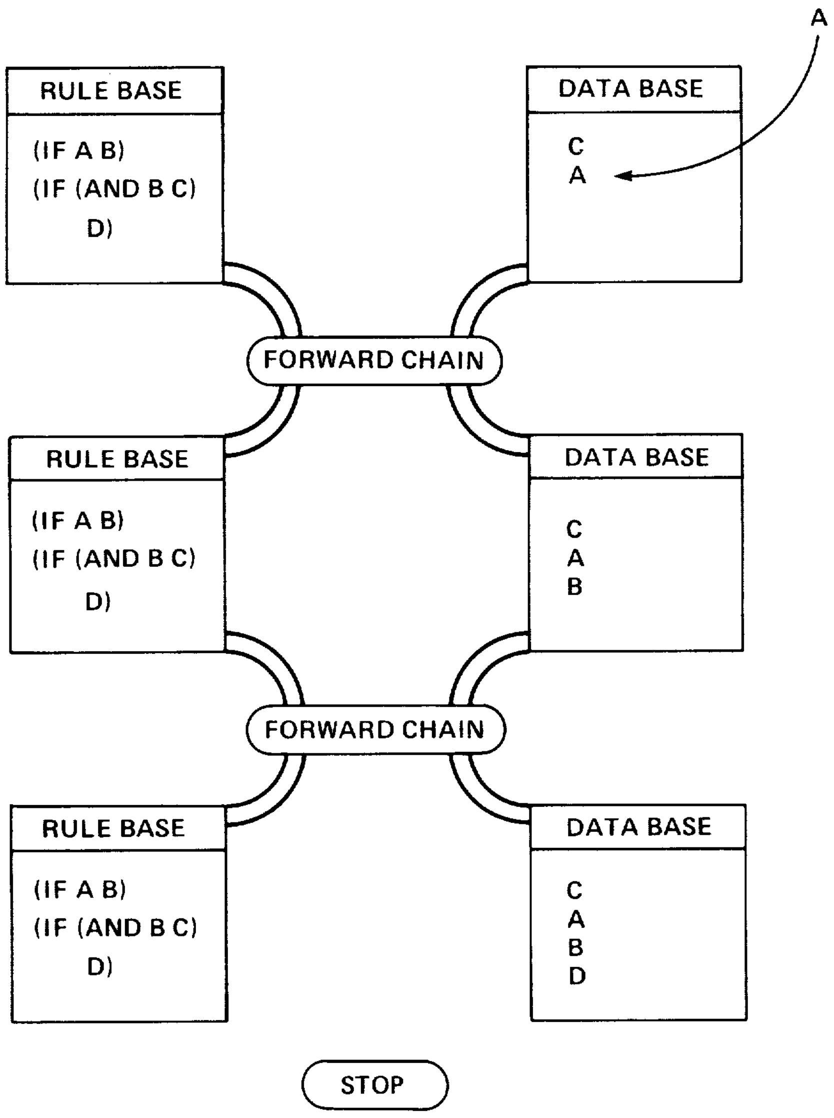


Figure 3.1. – Forward chaining diagram.

Backward reasoning (backward chaining) is goal-driven as opposed to data-driven; it is triggered not by the assertion of a fact into the database, but by a query to the system to establish the truth of a proposition (the goal). First, a procedure called a "lookup" is performed on the database to determine whether the goal proposition is already there. If it is not found, the rule base is searched for matches between the goal proposition and the *consequent* portion of the rules. When a match is found, the antecedent becomes the new goal proposition, the database is searched for its presence, then the rule base is resorted to, and the chain goes on until the current goal proposition is either found in the database, or there are no more matches to generate new goal propositions. Backward chaining is shown graphically in figure 3.2.

Establishing separate theories in MRS can have many advantages. Within the knowledge base, different theories can contain different types of knowledge (e.g., in EZGrid, one theory contains rules about curve generation, one contains rules about situation assessment, and so on). Theories can be activated and deactivated with a simple MRS command, so that searching for a match to a proposition can be limited to only those theories which are most pertinent at the time. Theories can be related to each other through directional links in a sort of "parent-child" relationship, where the child theory has access to all of the propositions in the parent theory, but

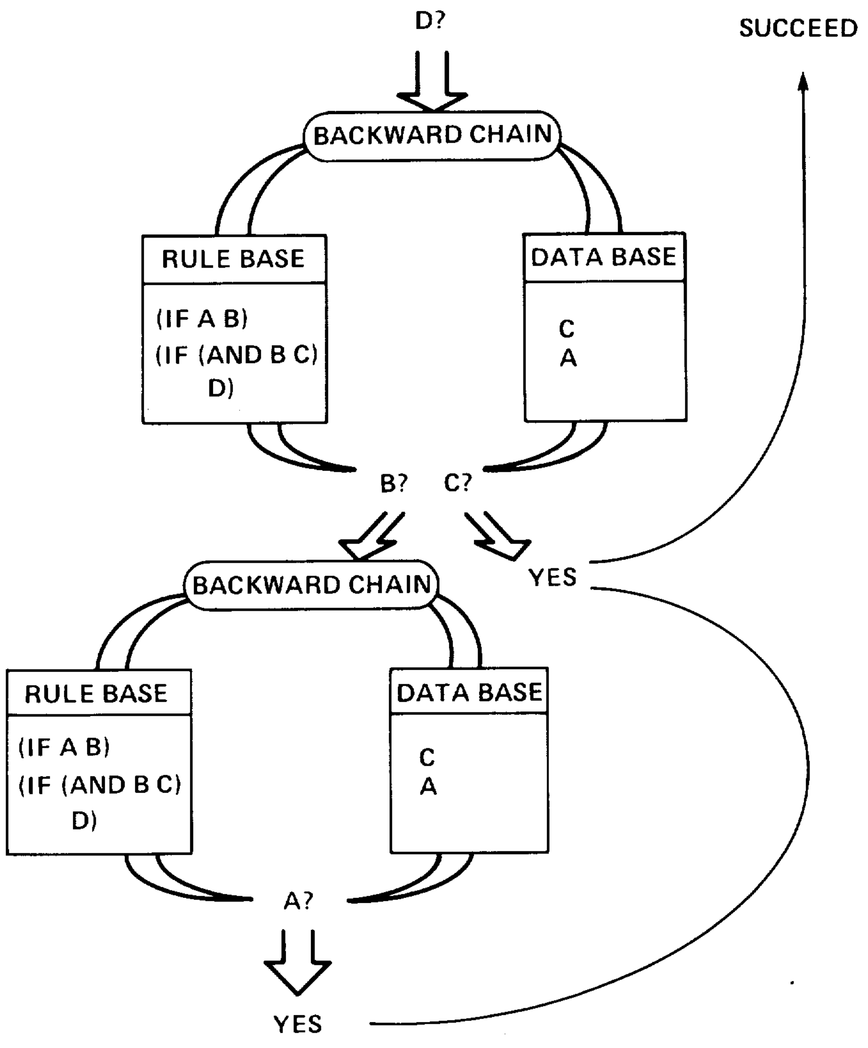


Figure 3.2. – Backward chaining diagram.

the reverse is not true. Sibling theories do not have access to the contents of each other. Using separate theories can also facilitate "truth maintenance" for the system by storing any suspect facts in a theory by themselves. The truth maintenance mechanism in MRS is called *residue*. Residue can be thought of as backward chaining in which the truth of the goal proposition need not be established as long as there is a rule in the rule base which states that that proposition can be *assumed* to be true (such a rule might look like (if (and A B) (assumable C)), where "assumable" is an MRS term). Any proposition deduced using residue is stored in a newly created theory that is a child of the current theory. All further deductions based on that proposition are also asserted into the new theory since their truth is dependent on the truth of the originally assumed proposition. In EZGrid, residue is used as a convenient mechanism for creating a tree structure of theories, where each theory contains the propositions which describe the situation at a single stage of a zoning design (see section 3.3).

Communication among the three programming languages is possible and fairly simple, allowing the strengths of each to be tapped. A lisp function may call a C function through a foreign function interface procedure. An MRS proposition may contain a term which calls a lisp function (a procedural attachment), resulting in the binding of an MRS variable or the determination of the truth of the proposition. A lisp function may assert propositions into the EZGrid database, and may initiate an MRS "lookup" or inference procedure. The result is a versatile system which can handle both numerical calculations and symbolic inference within a single framework.

3.3 Program Structure and Control

This section contains a description of the structure of EZGrid. The chart in figure 3.3 represents the conceptual structure of EZGrid as a knowledge-based system. An invocation of EZGrid places the user in the MRS environment, which is embedded within Lisp and C. Residing in the MRS environment are Lisp and C functions, a database, a knowledge base, and an inference engine (the MRS mechanisms for forward and backward chaining and residue, as described above). The arrows indicate the direction of interaction among the different components. The user deals directly with Lisp functions only, through various user interface functions. These Lisp functions can assert propositional facts into the database (which can trigger forward chaining), and they can invoke MRS lookup, backward chaining, and residue directly (hence the arrow to the inference engine). Lisp functions call C functions for cubic spline computations. The Lisp functions do not affect the knowledge base–there is no method of either interactive input of new rules or input via a learning algorithm. The knowledge base affects the database only through calls to Lisp functions and through the results of inference processes. Inference processes cause facts to be asserted into the database, and the assertion of facts can trigger further inference; hence the two-way arrow between those two components.

The knowledge base is a collection of unrelated MRS theories, as shown in figure 3.4a. These theories contain flow-field zoning domain knowledge in the form of propositions that are if-then rules. The next section will cover the content of these theories. Figure 3.4b shows an example tree structure for the theories which compose the database. At the outset of an EZGrid run, only the theory INIT exists, and it is empty. After the initial data have been input by the user (geometry, inflow conditions, user bias) and the situation has been assessed, a new child theory is created with the name <old theory>-ACTION, which is where decisions are made regarding what zoning actions are executable (both possible and desirable) in the present situation. Using MRS

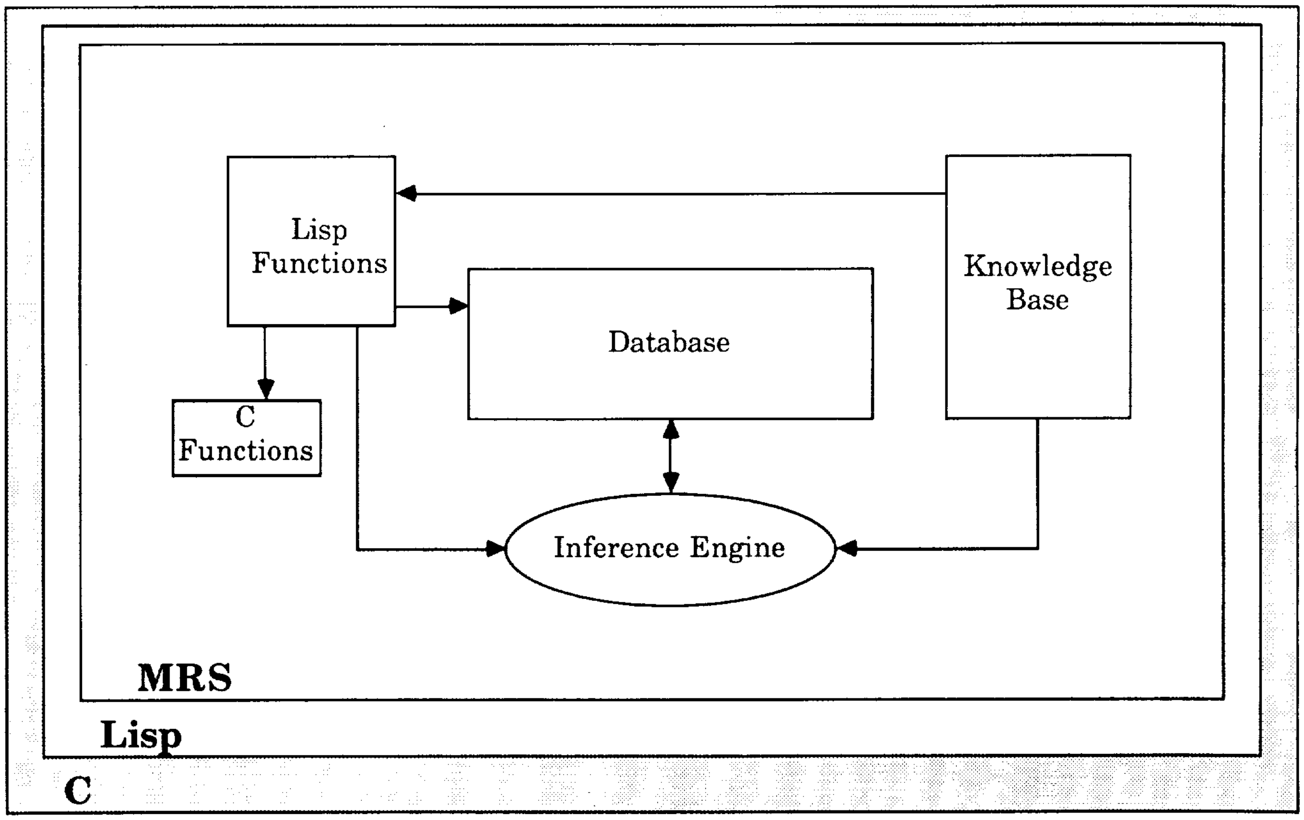
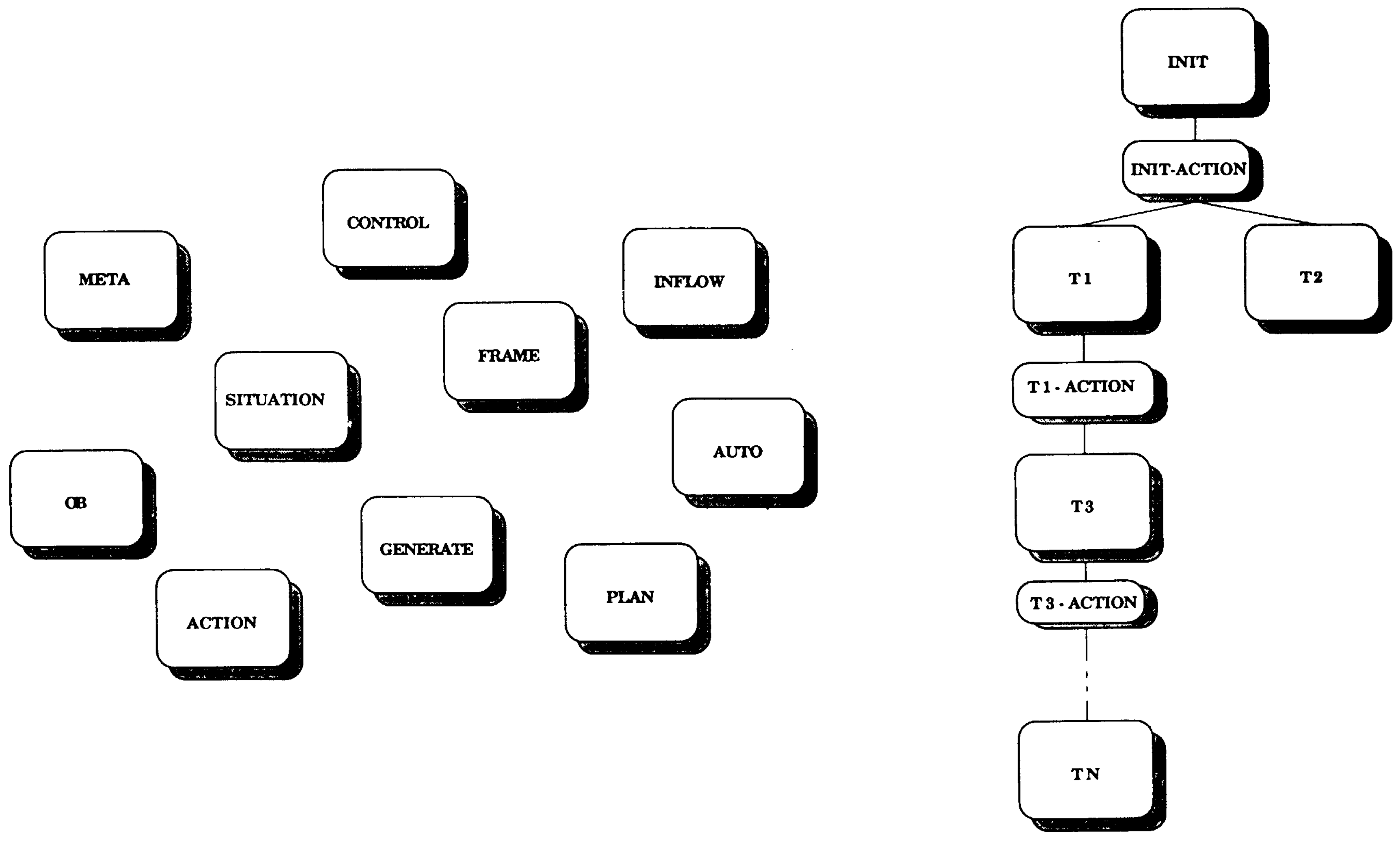


Figure 3.3. – EZGrid conceptual structure.



(a) EZGRID KNOWLEDGE BASE

(b) EZGRID DATABASE

Figure 3.4. – The use of MRS theories in EZGrid.

residue and this *assumable* rule from the theory CONTROL – (if (and (readytozone (stage $s1)) (executable $action (stage $s1))) (assumable (choose-action $action (stage $s2)))), new theories (children of INIT-ACTION) are created for each executable zoning action. Usually there are only one or two, and often the second action is simply a variant of the first. If there are two distinct actions possible, the first one is followed to the end (and a candidate zoning design is obtained), and then the trail of the other is picked up. If there is only one distinct executable action, as is often the case, the "tree" structure degenerates to a linear one. Each pair of theories – theory-name, <theory-name>-ACTION – contains the propositions that describe the situation at a single stage of the zoning design.

EZGrid has several modes of operation: interactive-multiple-step, interactive-single-step, and automatic. Within automatic mode, two submodes are possible: verification mode or batch mode. Both of the interactive modes have capabilities identical to the system which resulted from phase I of the EZGrid implementation, to which the user must add the perception, bias, and zoning design knowledge. There are two distinct phases in any system run, regardless of mode: the problem setup phase, which requires user input (even in automatic mode), and the zoning construction loop, which is traversed at each stage of a zoning design. This basic flow of control is illustrated in figure 3.5.

The multiple-step version uses the entire zoning action framework, so the user must specify the design in terms of a sequence of zoning actions, one at each stage. Figure 3.6 shows the events which occur during an interactive-multiple-step run of EZGrid after the mode of operation has been selected. The first step of the setup phase consists of tuning the user bias profile. The user must set the profile parameter values in one of three ways: by interactively using a menu of choices, by reading a file of parameter settings, or by keeping the default settings. The profile is used to guide zoning design in automatic mode only, but is used in the evaluation of completed zonings in all three modes.

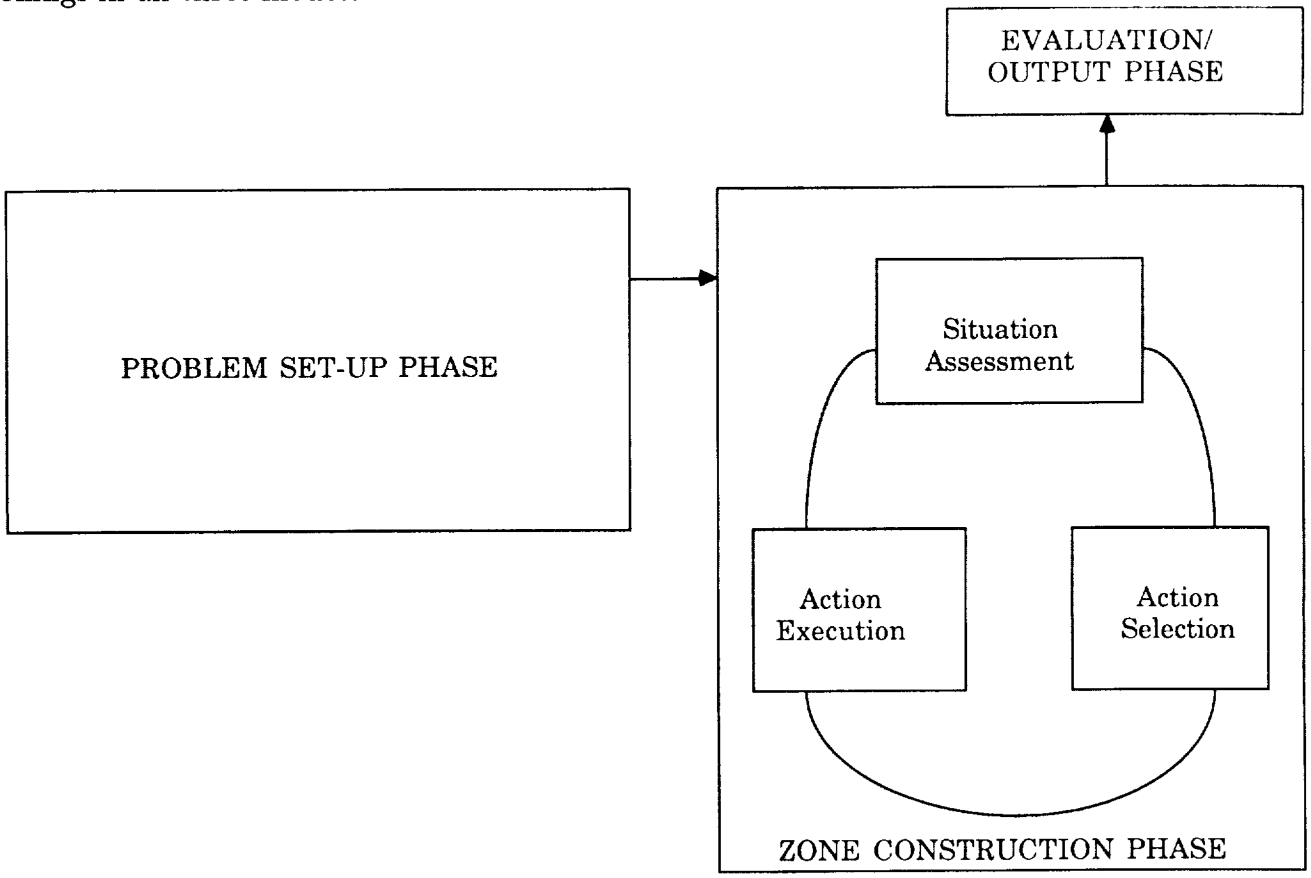


Figure 3.5. – Basic flow of control in EZGrid.

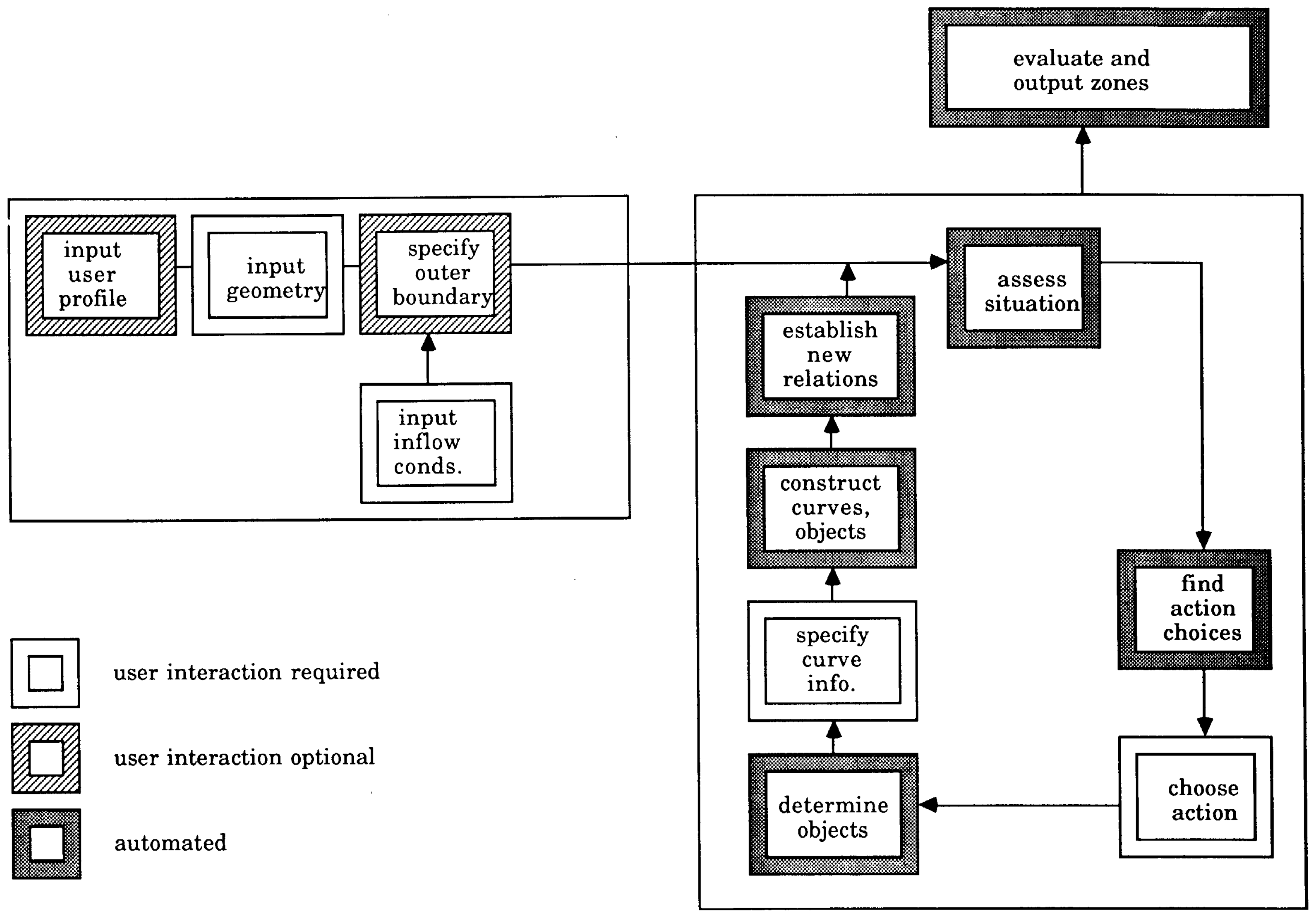


Figure 3.6. – Flowchart for interactive-multiple-step mode.

Geometry input can be interactive or can be accomplished by reading a data file of x,y coordinates. For both interactive input and reading a file, the user is asked whether the x and y values to be input are points which represent a continuous smooth curve, or are knots of a cubic spline, which can be sparse and discontinuous, but still represent a smooth curve. In the case of continuous-point input, any discontinuity is assumed by EZGrid to be a discontinuity in the surface of the body and a vertex is placed there, resulting in the end of one fundamental curve, or cubic spline, and the beginning of another. In the case of spline knot input, after each x-y pair is given, the user is asked whether that point is a vertex. Once EZGrid has this information, cubic splines are generated between each vertex, a vertexlist and a curvelist are constructed, and the appropriate properties are placed on the curve and object property lists. Propositions describing the new objects are also asserted into the database, e.g., (body B1 (stage 1)).

The outer boundary of the computational domain is then specified. The user is given the option of specifying the outer boundary him- or herself, or letting EZGrid do it automatically using heuristic rules relating inflow conditions to outer boundary limits. If the user chooses the former option, EZGrid asks if the input is to be via a data file of x,y coordinates or interactive. If it is to be interactive, the user is given four choices: (1) continuous point input, (2) cubic spline knot input, (3) curve parameter specification, and (4) object parameter specification. The first two choices are identical to the choices given in interactive geometry input. Curve parameter specification uses the menu-based curve specification capabilities developed for zonal interface curve specification during the loop phase. These capabilities are described in detail later. Object parameter specification involves choosing a shape (ellipse, bullet, or rectangle, as in the primitive part shapes described in section 2.3.3) and maximum dimensions in terms of distance in chordlengths upstream of the upstream-most point of any body, downstream of the downstream-most point of any body, above the topmost point of any body, and below the bottom-most point of any body. The reference chordlength used is that of the body with the largest chordlength. EZGrid has rules which can use this information to generate the appropriate outer boundary object (OB). If the automatic OB specification option is exercised, EZGrid asks the user to supply inflow conditions such as free-stream Mach number, angle of attack, and flow steadiness. These and other physical parameters may be needed later during the zoning design, so once their values have been provided, they are stored as propositions in the database. Whenever such a value is needed later during the same run, EZGrid first does a lookup on the database to see whether it is already known, and only if it is not found will the user be bothered for that information. There are rules in the knowledge base which relate the above inflow conditions to OB maximum dimensions. These rules provide the dimensions, the user is asked for the shape, and the object generation rules then proceed as before to construct OB. As with the flow-field geometry, OB has a curvelist and vertexlist placed on its property list, and relational and type information is stored as propositions in the database, e.g., (zoning-environment OB (stage 1)) and (contains OB B1 (stage 1)).

At this point the zoning construction loop, which is executed once for each stage of the zoning design, is entered. First, the situation is assessed. Situation assessment is triggered by the assertion of the proposition (end-of-initial-data (stage 1)) into the database when all object generation at that level has been completed and the associated information stored. The CONTROL rule which is triggered begins by calling a lisp function *(assert_data)* which invokes backward chaining to (1) check for direct containment relationships among objects; (2) count all zoning bodies and zoning environments in order to construct the corebody and contained environment lists; (3) check to see whether any zoning environments are candidates for topology imposition (if so, that process is executed, and the results are recorded as a sidelist on an object's property list and as a proposition (zone ZE3 (stage 3))), and (4) look for objects which have been filled, and find the filling object lists.

The CONTROL rule that initiated situation assessment then checks to see whether the zoning is complete; creates a new theory (INIT-ACTION, the first time around); and finally asserts propositions into that new theory which trigger zoning action rules. So the next two components of the loop as pictured in figure 3.6 take place within the next theory in the database tree. All possible zoning actions are found using rules from the ACTIONS theory of the knowledge base. These are presented to the user, who must choose one for execution. The proposition (executable

<chosen action> (stage 1)) is asserted into the database, enabling the residue mechanism to operate (recall the rule in which any executable action is *assumed* to be the chosen one) and create a new child theory. In this new theory, all propositions end with (stage n+1), where n is the number of the previous stage, because the execution of an action changes the zoning situation.

There are rules (called object identification rules) which enable EZGrid to determine the type and method of specification of each new object which must be constructed to realize a zoning action, as well as the number of objects required for the action. For the purpose of generation, the type of an object is either *created* or *resultant*, where a created object requires new curves (and possibly existing ones) for its construction, and a resultant object is composed of existing curves only, and usually contains a created object. Most zoning actions involve the construction of both created and resultant objects. There are five methods of specification: curve, object, traverse, merge, and subtract. The curve method requires the user to supply curve specification information, as described in the next paragraph. The object method involves the specification by the user of object parameters (as in the specification of OB described above), and the automatic determination of the curve specification information by EZGrid. Traverse, merge, and subtract do not require any input from the user since they involve existing curves that either have just been generated for a created object, or belong to the curvelists of the operand objects of the action being executed. Merge and subtract are used for resultant objects only. For example, two propositions that are asserted into the database by the firing of an object identification rule for a connection/bridge operation are (object-needed ZE3 (type created) (specification curve) (stage 2)) and (object-needed ZE4 (type resultant) (specification merge) (stage 2)).

If the object is to be specified via curve specification, the user must specify the number and type of curves needed. For the purpose of generation, there are two types of curves: *new* and *existing*. To specify a new curve, the user must supply the first and last points of the cubic spline, and may supply interior points and the angles at the beginning and end of the cubic spline (which are converted into first derivatives). Points and angles are input using menus. Table 3.1 shows the menu choices available to the user. Existing curves are specified by selecting two points on an object and specifying which portion between those two points (the clockwise or the counterclockwise portion) belongs to the curve being specified. The two points are selected in the same manner as for new curves.

If new curves have been specified, they are now generated by a call to a lisp function, which calls a C function and asserts all of the new curve properties. If existing curves have been specified, a lisp function performs whatever curve division might be necessary to identify the new segment of existing curve(s) in terms of one or more new subcurves. New zoning objects are then constructed according to the way in which they were specified. List traversal is the fundamental building block of object construction. If an object has been specified via curves or object parameters (which reduces to curve specification), traversal of the resulting list of new and existing curves to order them in a clockwise manner forms the curvelist of the new object. If traverse is the method of specification, then a list of existing curves is traversed and ordered to form a new curvelist. A resultant object specified by the merge method has its curvelist formed from the merging of the curvelists of the action operand objects and, usually, the curvelist of a newly created object (such as a bridge). The subtraction method forms a new curvelist by the subtraction of the curvelists of one or more existing objects (usually siblings) from the curvelist of another existing object (usually the parent object). After the construction of each new object, object properties are determined and stored, and the user is asked whether a plot of either the new object, all objects, a list of curves, or a list of objects is desired.

Finally, new object relations are established. Containment relations are determined by EZGrid rules using information about what zoning action has been executed and what sorts of objects have been constructed. Direct containment and other situational object properties are updated in the next step–situation assessment. During situation assessment, it is discovered whether any of the new objects constructed are empty. As soon as a proposition to that effect (e.g., (empty ZE5 (stage 4))) has been asserted into the database, a rule is triggered which calls a lisp function. This function queries the user as to the desirability of imposing a topology on the empty object. This is done because the criteria for an object becoming a zone are that it be

Table 3.1 Point and Angle Input Menus.

POINT MENU

1. PREVIOUS POINT
2. VERTEX LOCATION
3. POSITION ON AN OBJECT
4. POSITION ON A CURVE
5. x, y VALUES
6. AN OBJECT EXTREMUM
7. CONNECTION BETWEEN TWO OR MORE CURVES
8. ARCLENGTH DISTANCE ALONG A CURVE FROM A POINT
9. VECTOR FROM A POINT
10. POSITION ON AN OBJECT BETWEEN TWO POSITIONS
11. POINT BETWEEN ANY TWO OTHER POINTS
12. INTERSECTION WITH AN OBJECT OR CURVE
13. END OR SIDE OF AN OBJECT PART
14. ONE OF A GROUP OF PREVIOUSLY STORED POINTS

ANGLE MENU

1. SPECIFY FRACTION OF AN EXISTING ANGLE
2. SPECIFY ANGLE MEASURE FROM RIGHT-HORIZONTAL (DEGREES)
3. LEAVE IT UNSPECIFIED

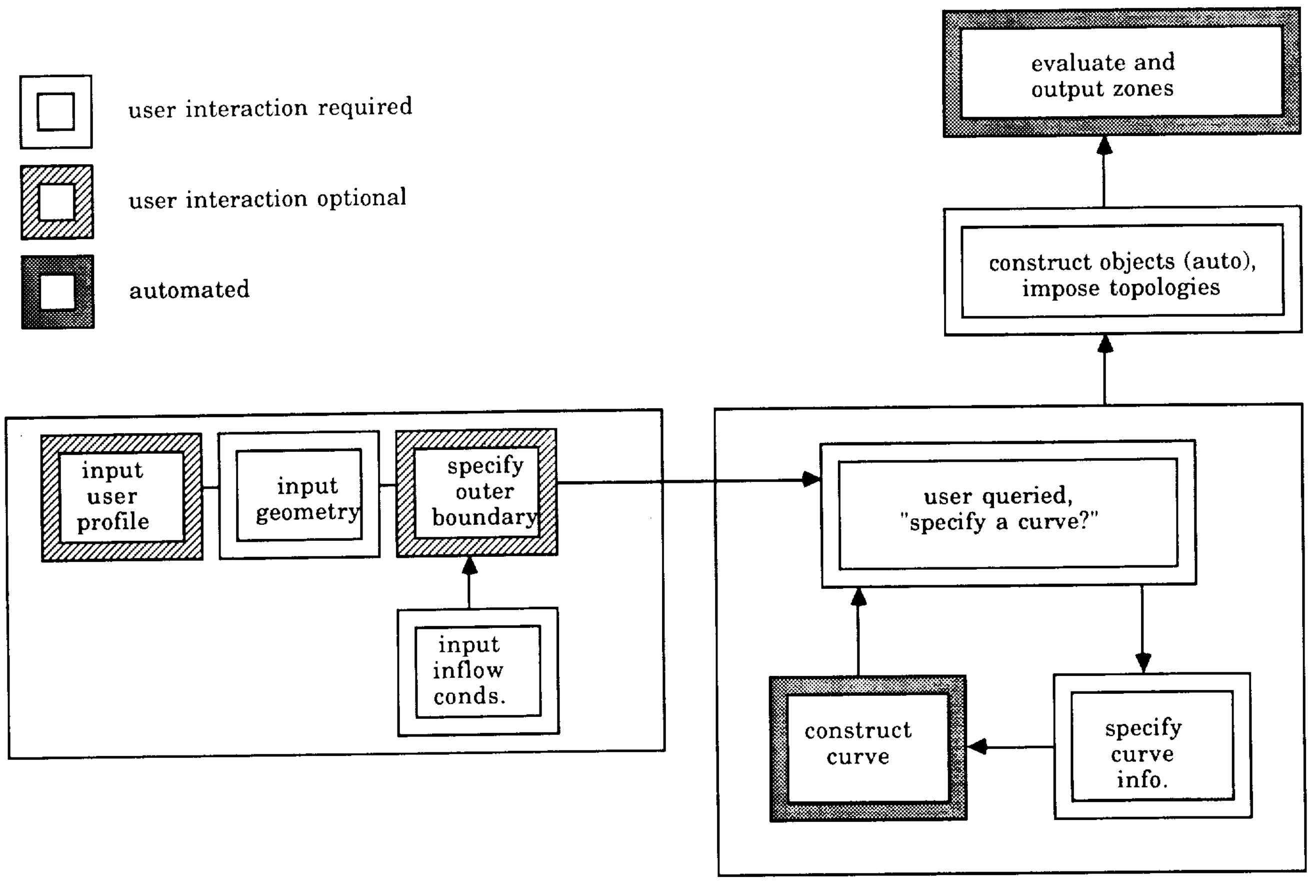
empty and topologically four-sided. If the user does want that object to become a zone at that point, he or she is given the following choices for imposing a topology on the object: impose-C, impose-H, impose-O, identify-corners, or choose-points. The first three choices result in the imposition of the familiar C, H, and O topologies. If none of these seems appropriate, the two additional options either choose the four vertices of the object with included angles closest to 90°, or allow the user to choose any four arbitrary points (using the same point selection menu as is used in curve specification) to be the topological corners of the object.

If, during situation assessment, it is discovered that OB has been zoned, the design is complete. The user may invoke the EZGrid zoning evaluator at this point (a discussion of zoning evaluation is provided in chapter 4). For output of zonal interface information to a grid generation program, a lisp function is called which reorders the zone sides, samples the curves of each zone's sidelist with a number of points specified by the user, and writes out the resulting x,y coordinates in accordance with the format of the grid generator of choice (in this case, GRIDGEN2D (Steinbrenner, 1986)). These points define the zone topology and interface locations, but are not usually suitable as actual grid-point locations. For simple cases, there are rules in the knowledge base which determine the number and distribution of grid points for each zone. In general, however, no attempt is made to automate the grid generation task.

The second mode of EZGrid operation is interactive-single-step, which does not use the zoning action framework, but rather just uses the convenient point and angle menu input to facilitate curve specification. The idea behind this mode is that the user does not want to engage the whole EZGrid machinery to do action selection, action execution, and situation assessment.

This mode is for the user who simply wants to directly specify the curves that become the zonal interfaces, and all at one stage so that the process is much quicker. The flowchart which illustrates this mode is shown in figure 3.7. The same problem setup is done, but the loop which is then entered is simply a curve-generation loop, where curves are specified and generated as described above. When no more curves are to be generated, the list of newly created curves plus the curves belonging to the curvelists of the original geometry and the outer boundary object is traversed to find all closed loops. These are designated the curvelists of the new zones. The user is asked to impose a topology on each zone, and the zoning is complete. The final steps again involve zoning evaluation and output of interface information, as in the interactive-multiple-step mode.

The flow of control in EZGrid's automatic mode is depicted by figure 3.8. The setup phase in automatic mode is identical to that in the two interactive modes, with the exception of two additional user input steps: qualitative shape description and qualitative configuration description. It is ironic, perhaps, that the mode which is touted as *automatic* requires more user input during the setup phase than do the interactive modes, but it is important to remember that the user provided that information in the other modes implicitly, throughout the zoning process, and repeatedly (it was never stored to avoid redundancy in input). By contrast, the input of that information in automatic mode is explicit, is done only once, and is stored in both propositional and property list forms so that the system can use it throughout the zoning process without having to ask the user for further information. The user is asked to provide a qualitative shape description for each original body after it is constructed during geometry input. This is accomplished using menus which prompt the user for each attribute required in the shape description (as discussed in section 2.3.3). The quantitative portion of primitive part description is provided by the user by means of the same point selection menu used in many other steps of the zoning



user interaction required
user interaction optional
automated
evaluate and output zones
construct objects (auto), impose topologies
input user profile
input geometry
specify outer boundary
input inflow conds.
user queried, "specify a curve?"
construct curve
specify curve info.

Figure 3.7. – Flowchart for interactive-single-step mode.

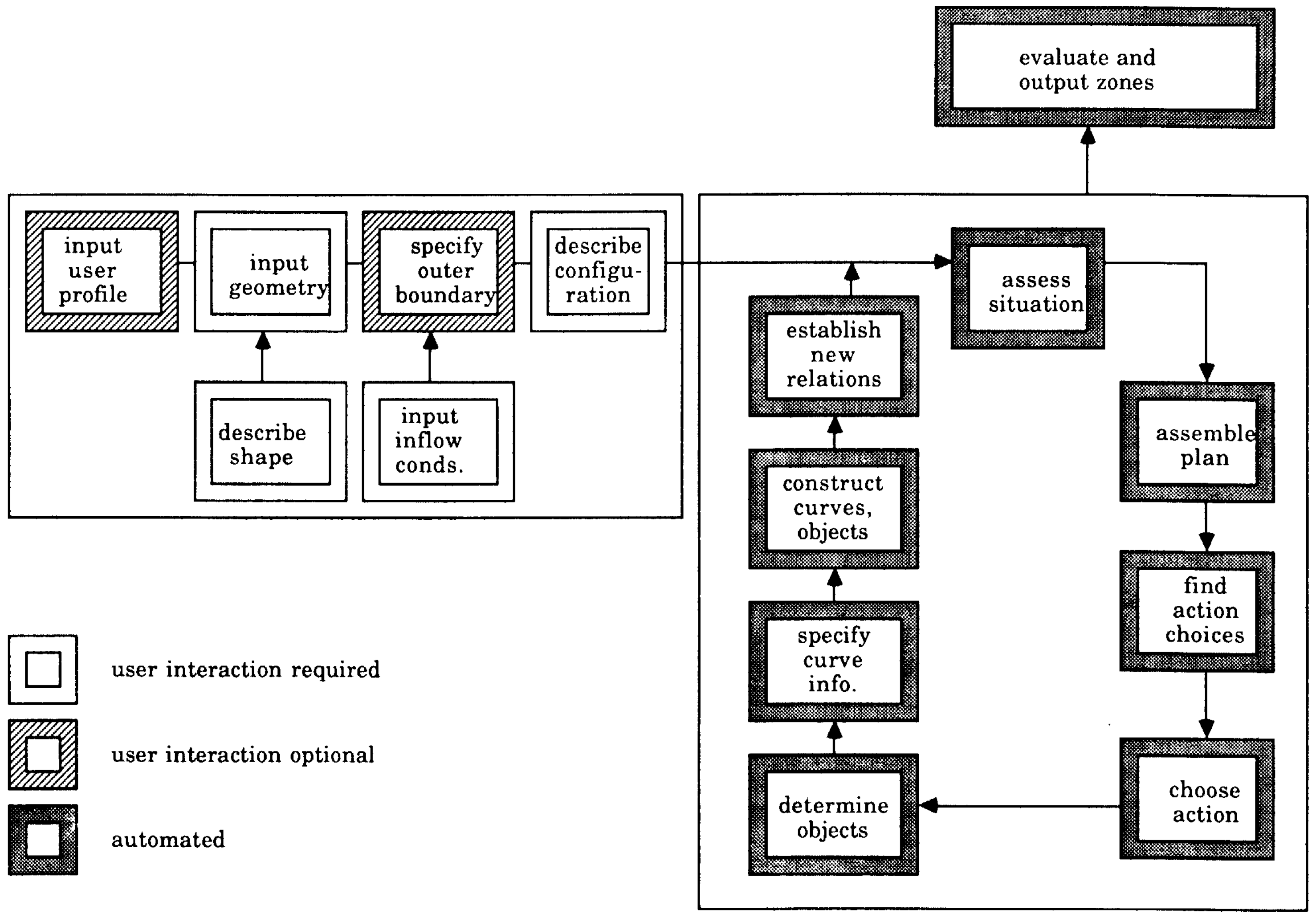


Figure 3.8. – Flowchart for automatic mode.

process. The user is asked to provide a qualitative configuration description after OB is specified. Again, menus, prompts, and the point-selection menu (for the specification of approximate minimum separation distances between objects and groupings) aid the user in this input. Once body shapes and the configuration have been described, the descriptions can be stored in files and read in lieu of interactive input when the information is requested in future runs.

The loop phase, in which a zoning is constructed stage by stage, differs in automatic mode from that in the interactive-multiple-step mode in three ways: (1) a new step is added between situation assessment and the finding of zoning action choices, which is the construction of a zoning plan, or sequence of zoning actions for each stage; (2) the step in which an action is chosen is automated–the user is no longer asked to choose an action (unless either no plan has been constructed or the plan has failed, in which case the system drops into interactive mode locally); and (3) the curve specification step is automated, and no longer requires user input. As can be seen from figure 3.8, the entire construction loop is automated in this mode.

A plan is represented as a proposition in the database, and contains two lists: a pending list and an executed list. An example of a plan is (plan P1 (pending (action4 action5 ...)) (executed (action3 action2 action1)) (stage 4)). The first time through the loop, rules which govern the applicability of subplans for single-object groupings are fired after the situation assessment step, and are combined into a plan, which is asserted into the database. At each subsequent stage in the zoning, a rule checks to see whether the plan has failed for any reason, and if it has not, reasserts the plan in its last asserted form (i.e., the same plan is kept). Plan failure is described below.

The choice of a zoning action is automated by comparing the action at the top of the list of pending actions of the plan to the list of possible zoning actions at that stage. If a match is found, that action becomes executable. If no match is found, the plan fails, and the proposition (failed-plan P1 (stage 2)) is asserted into the database, and interactive action choice is triggered.

In automatic mode, curve generation proceeds without input from the user. Appended to each action term in the plan proposition are one or more action execution strategies, such as follow-wake, wrap-body-wake-constant-offset, and bisect-vertical-pair. There are rules in the knowledge base (in the theory AUTO) which use these strategies to determine the number and type of curves needed to execute the action, and call lisp functions to compute the locations of spline interior points for many of the strategies.

Situation assessment involves several new events in automatic mode. At the beginning of situation assessment, the plan is updated to reflect the fact that an action has just been executed. That action is moved from the pending list to the top of the executed list, the old plan is removed from the database, and the updated plan is asserted. When an object has been determined to be empty, the plan is examined to see whether the next action on the pending list is topology-imposition. If it is, the strategy associated with the action (one of the topology-imposition options described earlier–impose-C, etc.) is used to define the object's topology. If it is not, nothing is done to the empty object at that stage (probably because it is going to be acted upon at a later stage).

Appendix B contains a listing of the contents of the EZGrid database at the completion of an automatic run in which the flow field about a single, highly cambered airfoil was zoned. The database consists of five MRS theories: INIT, INIT-ACTION, T1, T1-ACTION, and T2. Three stages were required to solve this simple problem.

3.4 The EZGrid Knowledge Base

As is true for most programs, much of the knowledge contained by EZGrid is implicit, and is therefore difficult to extract. One advantage to a knowledge-based approach is that there is also much knowledge that is explicit, and is therefore easily extracted and transported, or used in ways other than direct problem-solving, such as user training. This section represents an attempt to describe the explicit knowledge contained by EZGrid in the form of MRS propositions stored in the knowledge base. Appendix C contains example rules representing each type of knowledge in the knowledge base. The organization of this discussion follows that of the knowledge base itself—by separate theories, as shown in figure 3.4a.

META [*MRS Knowledge*] – This theory simply contains the meta-level propositions which link lisp files to MRS propositions and instruct MRS to forward chain any fact which is asserted into the database.

SITUATION [*Zoning Model Knowledge*] – This theory contains knowledge about how to determine an object's varying properties, such as whether it is zoned (if it is a zone, is degenerate, or is filled by zoned objects); empty (contains no objects); filled (all of the curves on the object's curvelist are shared by its corebody and contained environments); connected (at least one curve or point of each contained object is shared by the curvelist of the containing object); degenerate (for each curve on the object's curvelist, the reverse of that curve is also on the curvelist, so no area is enclosed); a body (original or virtual due to containing a body) or a zone (empty and topologically four-sided); what it contains and directly contains; and what are its corebody, coretype, contained environments, and filling objects. There are rules for determining the possibility and desirability (in interactive mode) of topology imposition. There is also a rule which determines whether the zoning is complete (by looking for the proposition (zoned OB (stage n)) in the database.

CONTROL [*Implementation Control Knowledge*] – This theory contains knowledge about how to control the formation of the database structure (via MRS residue), when to open a new theory for action selection, and when to begin constructing a plan.

FRAME [*Implementation Control Knowledge*] – The "frame" problem in artificial intelligence programming involves representing facts that change as well as those that do not (Rich, 1983). It is often handled by adding a ***state variable*** to the representation to indicate at what time the fact is true; this is done in EZGrid by adding (stage $n) to most propositions. When moving from one stage to the next in EZGrid, it is important to know which facts can be brought up to the next stage unchanged, and which must be deduced anew because they may be changed by the execution of an action. Object properties which are invariant and unrelated to the properties which are represented propositionally are stored on the lisp property list of the object, and do not need to be updated in any way (they are not tagged with state variables). Other object properties can be assumed to be invariant under certain conditions. This theory contains rules that determine when a propositional fact can be brought up to the next stage unchanged. Containment is invariant, but direct containment is not, unless intermediate containment is impossible because of an object being filled and/or zoned. The fact that an object is a zone, body, or zoning environment at one stage does not change at the next stage, although a zoning environment may become a body or a zone in addition to being a zoning environment. An object which is zoned, filled, or degenerate will remain so at the next stage. If an object is filled, it will have the same filling-objects, corebody, coretype, and contained environments at the next stage.

INFLOW [*Fluid Dynamics Knowledge*] – This theory contains rules about the flow speed regime and high-lift conditions, and calculates the Mach angle. The information deduced is stored in the database and used by the OB rules (the OB theory is described later) to determine the limits of the outer boundary of the computational domain. The knowledge is very basic:

1. If the angle of attack is greater than $5°$, high-lift conditions prevail (which tend to increase the upper and lower boundary limits).
2. If the Mach number is ≥ 1.0, the flow is supersonic; if the Mach number is between 0.7 and 1.3, the flow is transonic; and if the Mach number is < 1.0, the flow is subsonic.
3. The Mach angle is calculated as the arcsin of 1/(Mach number) for supersonic flow (it is used to determine the minimum width of the downstream boundary so that any shock exits at the downstream boundary instead of out the sides).

There are also simple rules that do nothing but call lisp functions which elicit data from the user (free-stream Mach number, angle of attack, flow steadiness, whether the fluid is viscous, what type of turbulence model is used, the desired level of accuracy of the computation, and the boundary condition type at the outer boundary), and assert the results as propositions into the database.

OB [*CFD Knowledge*] – OB stands for the outer boundary object, and this theory contains knowledge about what shape OB should be, and how far away from the input geometry its boundaries should lie. The rules which govern the shape of OB involve the geometry qualitative shape description and some elements of user bias. Among the rules for OB shape are

1. If OB contains only one body at the first stage, and that body has a blunt leading edge and a sharp or base trailing edge, then OB should be bullet shaped.
2. If OB contains a single body at stage 1, the body has blunt leading and trailing edges, and the user bias profile setting for wake resolution importance is low, then OB should be ellipse shaped.
3. If OB contains two bodies at stage 1, and both bodies have blunt leading edges, then OB should be bullet shaped.

The rules for determining the limits of OB begin by using the speed regime, desired accuracy, and boundary-condition-type information to set the boundary limits for both steady and unsteady flow. Subsonic upstream and downstream limits are farther away than those for a supersonic case, and upper and lower limits are much farther away for a transonic case. For a supersonic case, the upper and lower limits are found by multiplying the tangent of the Mach angle by the total distance from the upstream boundary to the downstream boundary. This ensures that any shock will pass through the downstream boundary instead of the sides, thereby simplifying boundary conditions. There are then rules which modify the OB limits if high-lift conditions prevail (tending to push the boundaries out farther).

ACTION [*Zoning Model Knowledge*] – Knowledge about the circumstances which make zoning actions possible in any zoning situation is found in the ACTION rules. Using information from the database involving object relationships and properties, these rules enable EZGrid to determine when the different operators and operands that make up zoning actions are possible. This knowledge is basically just the application of common sense to the zoning model for actions. For example, a separation-based operator does not make sense unless there are two or more sibling objects in a zoning environment. Connection-based operators are possible inside an object under two conditions only: if the object is a zoning environment that is neither filled nor empty, or if it is a zoning environment that is empty, but not zoned. Shape-alteration-based operators are possible for an object contained by a zoning environment which directly contains the object and is not filled. The identification/identify operator is possible only for connected zoning environments that are neither filled nor empty.

GENERATE [*Zoning Model Knowledge*] – This theory contains knowledge about zoning action execution. Once an action has been chosen, *object identification* rules determine how many new objects are needed to accomplish the action, and the type and specification of each object (e.g., type = created and specification = curve). *Curve identification* rules (used only in interactive mode) find out from the user how many curves are needed and what type (new or existing) are needed, and if an object is to be specified via object parameters, they obtain that information and convert it into curve parameter information. *Curve specification* rules (again, used only in interactive mode) get begin-point, end-point, begin-angle, end-angle, and interior-point spline information from the user. *Existing curve* rules keep track of curve division, subcurve creation, and new curve connections. *Curve parameter* rules convert the curve-specification information into the type of information needed by the spline-generation functions—number of knots; boundary condition types (free, specified, or periodic) and values; and a list of knot locations. *Completion* rules check for curve completion (successful return from lisp curve generation function), object completion (when all the curves needed to construct the object are completed), and action completion (when all the objects needed to accomplish the action are completed). *New property* rules are triggered by object completion propositions, and call lisp functions to compute properties (curvelist and vertexlist, primarily) for each new object, and assert object type (zoning environment) into the database. Finally, *new relation* rules, triggered by an action completion proposition, assert propositions describing the containment relationships of the new objects depending on the action just executed.

PLAN [*Zoning Design Knowledge*] – This theory must be activated (i.e., its contents made available to the inference engine) when EZGrid is run in automatic mode because it contains knowledge about what sequences of zoning actions (including execution strategies), or plans, are desirable for certain zoning situations. PLAN contains subplan rules which govern the zoning for single groupings based on the number of objects in the grouping, their qualitative shape and configuration description, user bias, OB shape, and general flow direction. There are also rules about how to assemble subplans into plans, which take into account number and relative positions of groupings, and rules about how to update already assembled plans. Finally, there are rules which tie in plans with the propositions which trigger action selection. They check for plan failure or absence, and return to interactive mode for action selection in either event.

AUTO [*Zoning Design Knowledge*] – Like PLAN, this theory contains zoning design knowledge and must be activated when EZGrid is run in automatic mode. AUTO contains knowledge about how to execute the chosen zoning action using the strategies suggested by the plan. There are two main types of rules: *curve identification* rules and *curve specification* rules. Recall that these two types of rules were also found in GENERATE, but there they involved eliciting the necessary information about curve numbers, types, and specification from the user. Here, these rules use the strategies from the plan, knowledge about zoning actions, qualitative shape information, and quantitative geometrical information (angles, distances, lengths) to specify the number and type of curves needed for each new object, and the begin and end points, begin and end angles, and interior points for each curve. There are an infinite number of ways to generate zone object curves, and only a finite number of AUTO rules, so whenever a rule cannot be found to cover a particular situation, interactive mode is entered to supply the missing information.

CHAPTER 4

COMPUTED RESULTS FOR REPRESENTATIVE TEST CASES

The interactive mode of EZGrid is general in scope. There are no restrictions placed on what sort of aerodynamic configuration can be zoned interactively; the machinery for curve and object generation and situation assessment is in place, while the user supplies the elements of perception, bias, and zoning design expertise, as discussed in the previous chapters. With those elements and the ability to generate arbitrary curves and objects interactively, the user may generate a zoning for an arbitrary flow field.

The automatic mode of EZGrid is more restricted in scope. It differs from the interactive mode by the incorporation of three elements previously supplied by the user—perception, user bias, and zoning design knowledge—which provide the information needed by the system to perform zoning automatically. The perception element is still supplied by the user during the setup phase of an automated run in the form of a qualitative shape and configuration description, in a manner which is simple and general for typical aerodynamic bodies. The element of user bias is incorporated through the assignment of qualitative values to parameters which characterize a user's own criteria for a good zoning (e.g., simplicity, orthogonality, existence of singularities, and wake resolution). Neither of these two elements restricts the scope of EZGrid in automatic mode. It is the third element, zoning design knowledge, which is the limiting factor.

Only a fraction of all zoning design knowledge has been captured and encoded in the form of subplan rules. The ability to assemble subplans for single groupings of objects into plans for configurations consisting of multiple groupings increases EZGrid's generality by extending the range of application of the encoded knowledge. In some cases, subplan assembly is straightforward and simple; in many other cases, additional zoning design knowledge is needed to assemble subplans intelligently. It becomes a question of under which "carpet" to sweep the problem; if the subplan rules cannot handle all possible configurations, effort must be put into the rules which combine them, while if the subplan assembly rules are inadequate, emphasis must be placed on the completeness of the subplan rules themselves. In either case, a finite number of rules results in a finite capability. It is for this reason that attention has been confined to encoding zoning design knowledge for *representative* aerodynamic configurations, resulting in an automated zoning system which is a proof-of-concept rather than an fully operational system guaranteed to zone any 2-D configuration automatically.

The first section of this chapter describes the classification scheme which has been applied to aerodynamic configurations in order to identify representative cases. The two sections whicl. follow present flow-field zoning results which have been generated automatically by EZGrid for selected test cases.

4.1 Class'fication Scheme

The simplest and most obvious feature of a flow-field geometry is the number of bodies: a configuration is either single-body or multiple-body. Section 4.2 presents automatically generated flow-field zonings for single-body configurations, and section 4.3 presents results for multiple-body configurations. Further distinctions can be made within each major class based on some of the zoning language concepts presented in earlier chapters, as shown in figure 4.1. A single-body configuration can have a body which is *simple* (describable qualitatively in terms of a single primitive part) or *compound* (requiring a qualitative description consisting of two or more primitive parts). A multiple-body configuration can consist of a single grouping of objects, or multiple groupings. Within the multiple-grouping configurations class are groupings which are directly contained by different parent objects, and those which are directly contained by the same parent object. The finest distinction within the subclass of multiple-groupings contained by the same parent object is 'he way in which the subplans selected for the groupings are assembled.

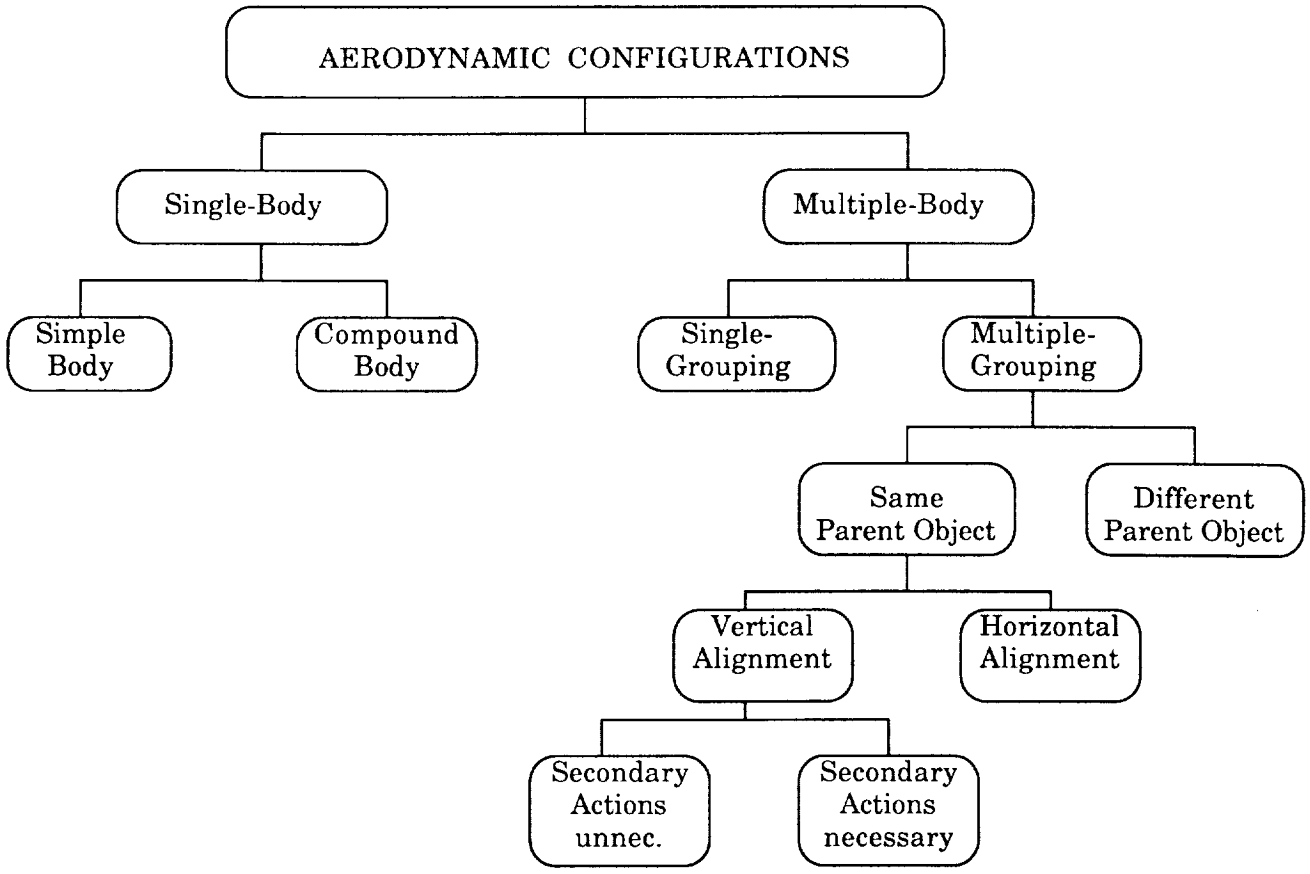


Figure 4.1. – Classification of aerodynamic configurations.

Section 2.5 characterizes subplan assembly methods in terms of positional relationships between groupings. Groupings within the same parent object can be either vertically or horizontally positioned with respect to each other. Subplans for vertically aligned groupings can be assembled one of two ways: with or without the inclusion of secondary actions. Subplans for groupings which are horizontally aligned are assembled without secondary actions, but with some modification of the primary actions required.

Twenty-three configurations have been selected to illustrate the way EZGrid handles each class described above. Flow-field zonings for seven simple single-body configurations have been computed for various inflow conditions, shape descriptions, and user bias profiles to demonstrate the influence of those factors on zoning design. Flow-field zonings for two compound single-body configurations have been computed using different shape descriptions to show the effect of shape description on the zoning of such bodies. Nine multiple-body configurations which consist of single groupings provide the platform for demonstrating the effect of distances and relationships among bodies (configuration), whether a computation is to be viscous or inviscid (physics), and the way a user tunes the user bias profile (user bias). The five remaining configurations involve multiple groupings, and the zoning for each one illustrates a different approach to subplan assembly. In all cases, default values for the user bias profile are used unless otherwise indicated. The default values are

1. Simplicity – low importance
2. Skewness – high importance
3. Smoothness – medium importance
4. Mapping Disparity – high importance
5. Efficiency – low importance

6. Orthogonality – high importance
7. Surface vs. Field Quantities – low importance
8. Wake Resolution – medium importance
9. Tuple Points – allowed
10. Singularities at Body Surfaces – allowed
11. Zone/Body Intersections – allowed
12. Resolution of Viscous Terms in More Than One Direction – allowed

4.2 Single-Body Configurations

The first three cases are intended to demonstrate the effect of free-stream Mach number and angle of attack on the zoning of a flow field. These three cases differ from all of the rest in that EZGrid was allowed to completely specify the outer boundary shape and location. In all of the remaining cases, EZGrid determined only the shape of the outer boundary; its suggested outer boundary dimensions were overridden in favor of closer boundaries simply for ease of presentation.

The first configuration consists of a single NACA 0012 airfoil. Figure 4.2 shows the zoning for this airfoil at 0° angle of attack, $M_\infty = 0.5$. The grid for this case (and for all of the cases shown here) was generated interactively using GRIDGEN2D (Steinbrenner, 1986). For this subsonic free-stream Mach number and *low* desired solution accuracy, the outer boundary was set four chordlengths away from the body in all directions. A solution accuracy setting of *high* would have produced upstream and downstream boundaries located six chords away, and the upper and

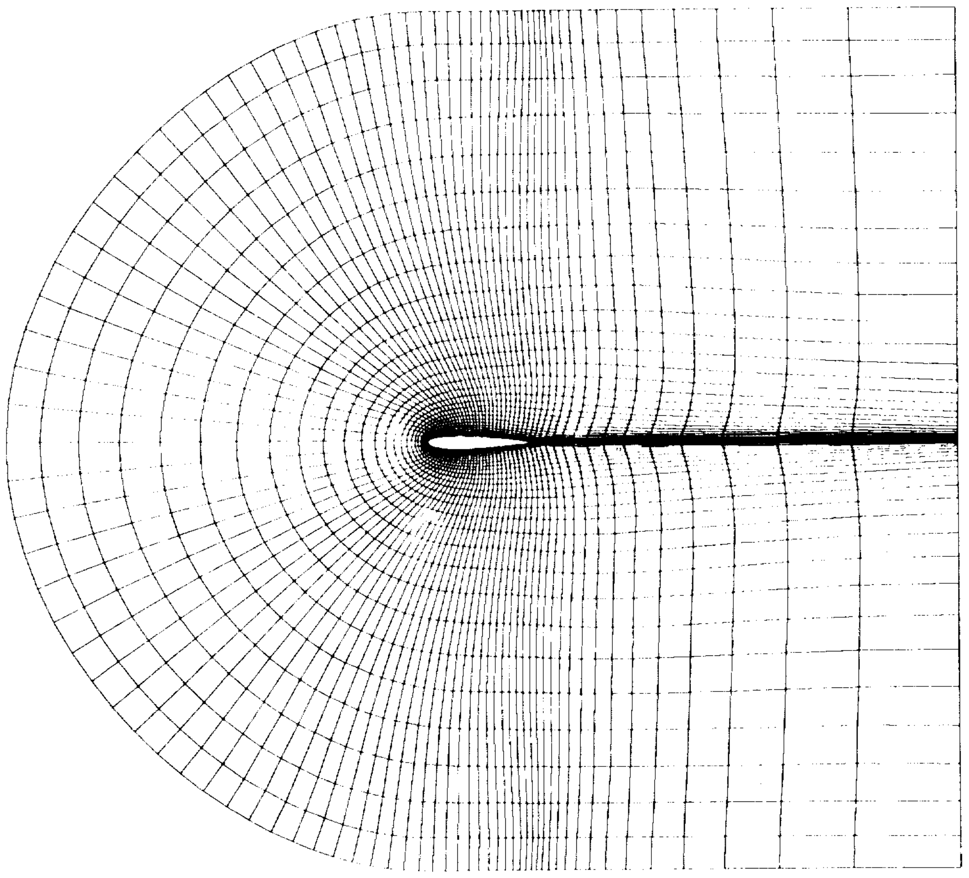


Figure 4.2. – Single-zone grid for an NACA 0012 airfoil at $\alpha = 0°, M_\infty = 0.5$.

lower boundaries ten chords away from the body. Because the angle of attack and the angle of the trailing edge of the airfoil are both 0°, the wake curve is a straight line from the trailing edge to the downstream boundary.

Figure 4.3 shows the zoning for the same airfoil at an angle of attack of 10° and $M_\infty = 3.0$. The outer boundary has been brought closer to the body (again, desired accuracy was *low*) since with a supersonic free stream, the downstream portion of the flow cannot influence the upstream portion. There is a rule in EZGrid which calculates the Mach angle of the free-stream Mach number and multiplies the tangent of that angle by the distance from the upstream boundary to the downstream boundary to determine the minimum half-width of the outer boundary. This ensures that any shock emanating from the upstream region of the body passes out through the downstream boundary instead of the side boundaries, a requirement that simplifies the boundary conditions. The wake curve in Case 2 is noticeably curved. This is due to an action execution strategy in EZGrid called *follow-wake* which specifies the wake curve such that it leaves the trailing edge of a body at an angle which bisects the trailing edge angle (if it is a blunt end rather than a sharp vertex, the curve angle is normal to the surface), and then relaxes to the free-stream angle of attack as $1/ln$(distance from the trailing edge).

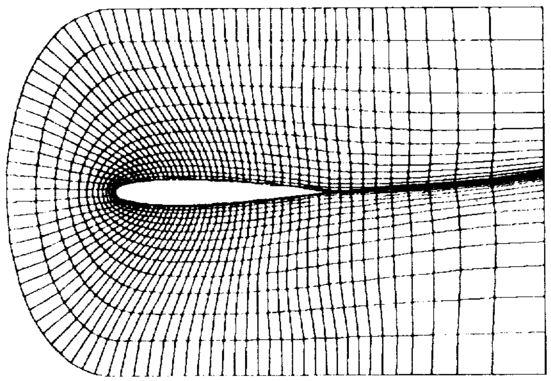
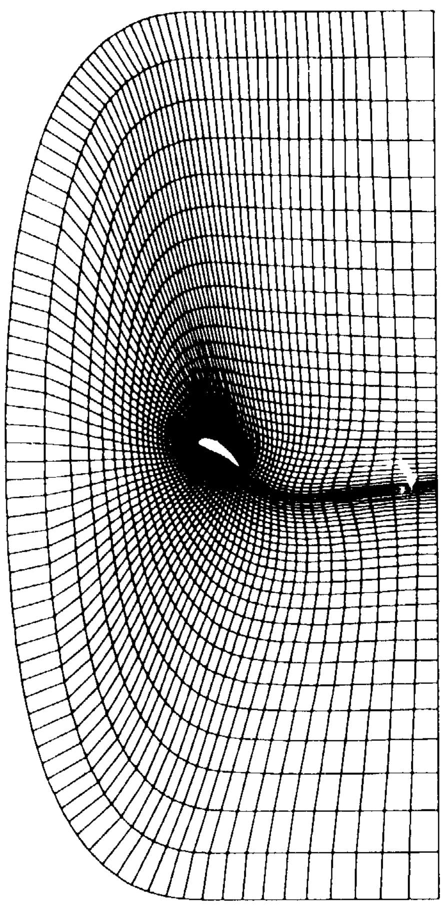


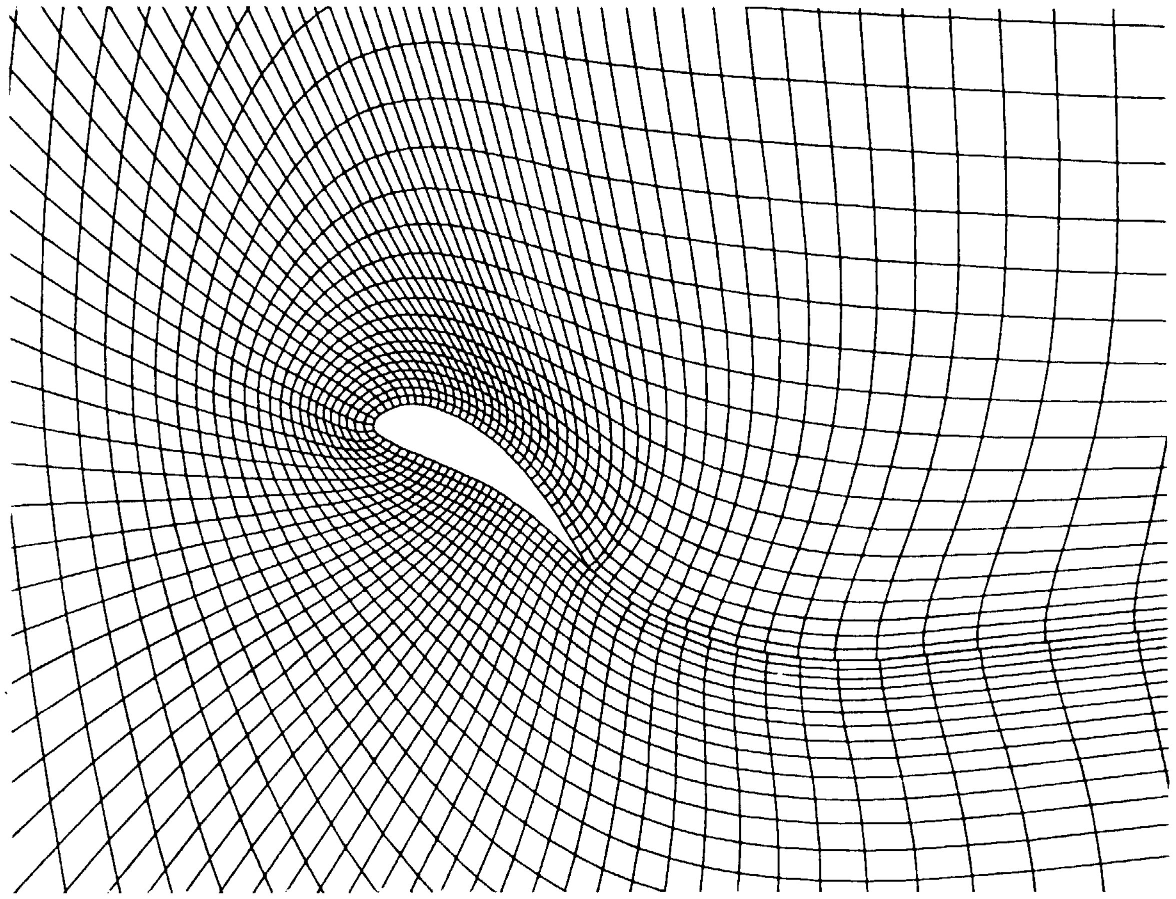
Figure 4.3. – Single-zone grid for an NACA 0012 airfoil at $\alpha = 10°, M_\infty = 3.0$.

The flow-field zoning for a highly cambered airfoil is shown in figure 4.4a. Here, the angle of attack is 5° and $M_\infty = 0.8$. For this transonic case, the upper and lower boundaries of the computational domain are farther away than in the subsonic case of figure 4.2. The large curvature of the wake curve is due to the difference between the angle at the trailing edge of the airfoil and the free-stream angle of attack. The grid that was generated for this configuration is of special interest because the grid generation parameters (i.e., the number of grid points in each direction, and the distribution of those points along the boundaries) were all determined automatically by EZGrid rules. The EZGrid output files were read into GRIDGEN2D and the grid was generated without modification of the number or distribution of the points. Figure 4.4b shows a closeup of the grid near the body. All of the *zonings* presented were generated automatically by EZGrid; this is the only example of a *grid* which was also generated automatically (some human intervention was required to run GRIDGEN2D using the EZGrid output files since there is no link between those two programs). As stated in an earlier chapter, no attempt was made to solve the problem of automating grid generation in any general fashion.

The zoning results for the next five cases demonstrate the effects of geometry and viscosity (specifically, whether the computation is viscous or inviscid). In these cases, as well as in all of the remaining cases, the outer boundary locations have been specified interactively in order to bring them in close to the bodies. Figure 4.5a shows the zoning and grid prepared for an inviscid solution of the flow about a biconvex airfoil at 5° angle of attack. Because the body has both sharp leading and trailing edges, and singularities are *allowed* under the default user bias profile, EZGrid has chosen two H-topology zones for this case. A *medium importance* setting of



(a) ENTIRE GRID

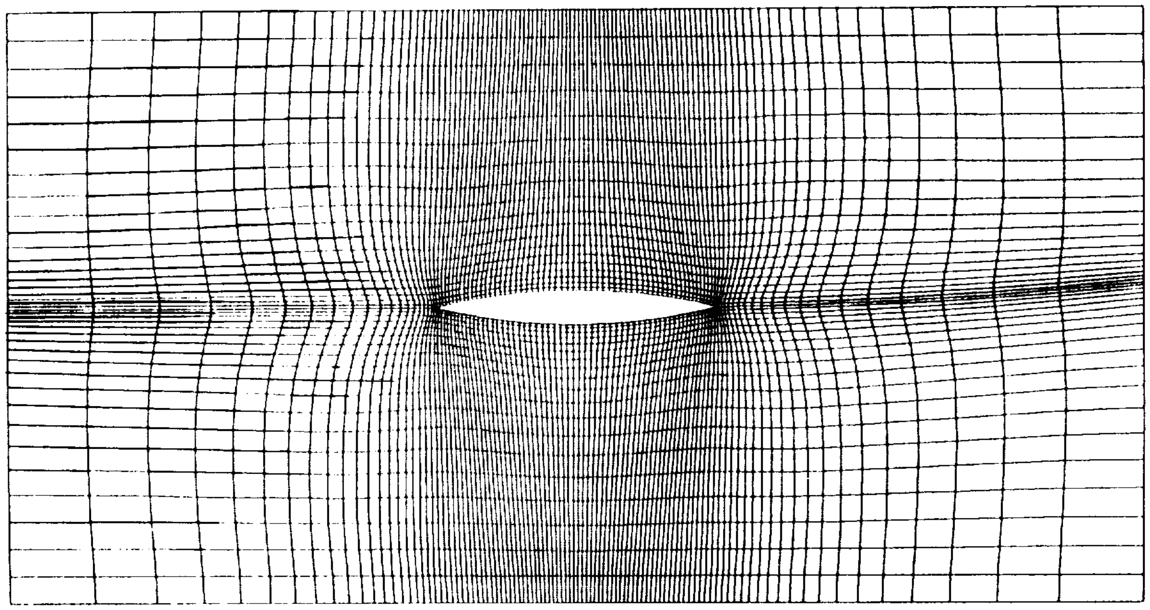


(b) GRID DETAIL NEAR THE BODY

Figure 4.4. – Single-zone grid for cambered airfoil at $\alpha = 5°, M_\infty = 0.8$.

wake resolution, one of the default profile values, resulted in a wake curve position depending on trailing-edge angle and angle of attack, as in all of the previous cases. Figure 4.5b shows the same body at the same angle of attack, but for a viscous computation of the flow field. The difference lies mainly in the extra zone placed upstream of the airfoil. By limiting the upstream extent of the two parallel H-type zones, the viscous clustering of grid lines near the body need not be carried far upstream where it is not needed. The clustering near the body is permitted downstream because of the relatively high importance placed on wake resolution.

The zonings in figures 4.6a-b differ from those in figures 4.5a-b only in different airfoil geometry. EZGrid produces a three-zone zoning for the wedge geometry, with one zone devoted to the


(a) 2-ZONE INVISCID GRID

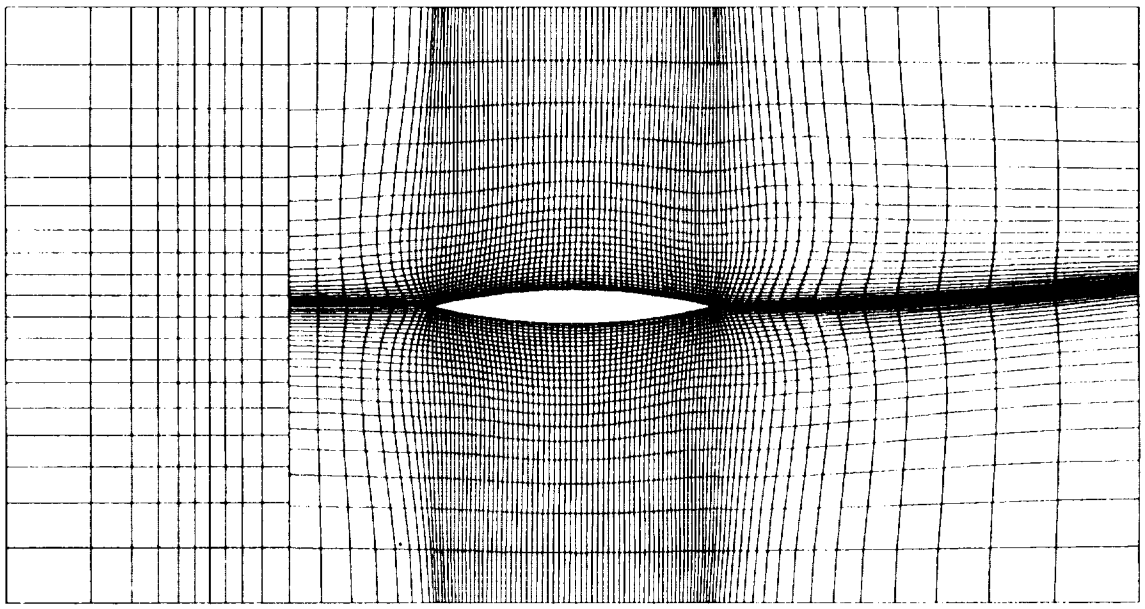
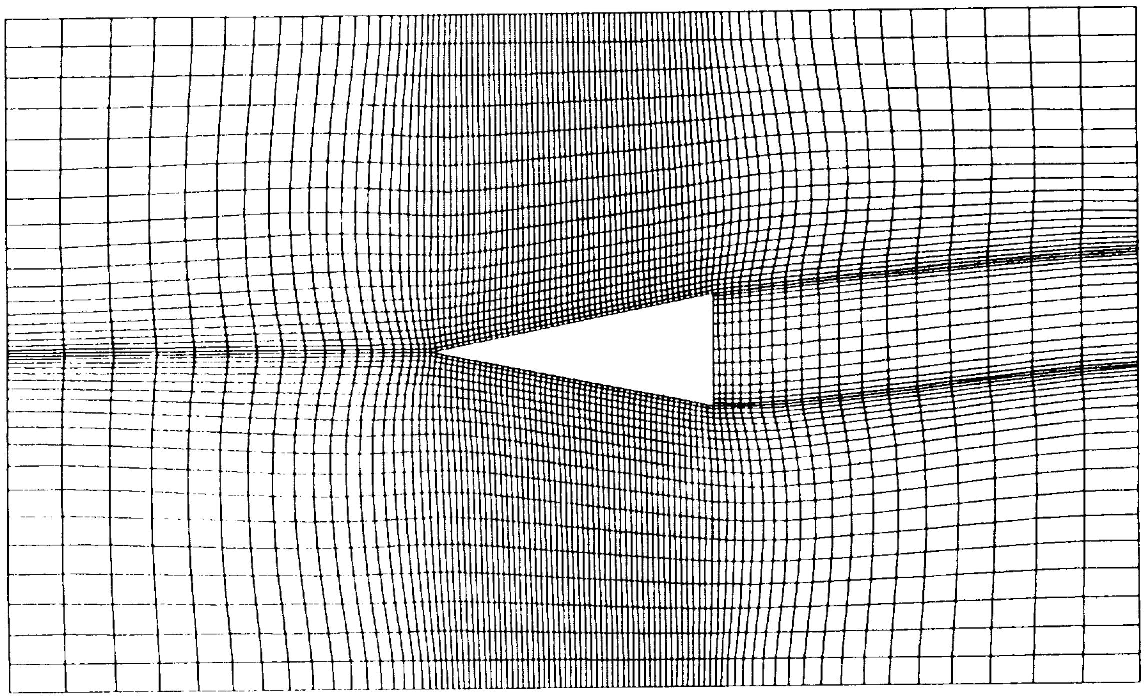

(b) 3-ZONE VISCOUS GRID

Figure 4.5. – Zonal grids for a biconvex airfoil at $\alpha = 5°$.


(a) 3-ZONE INVISCID GRID

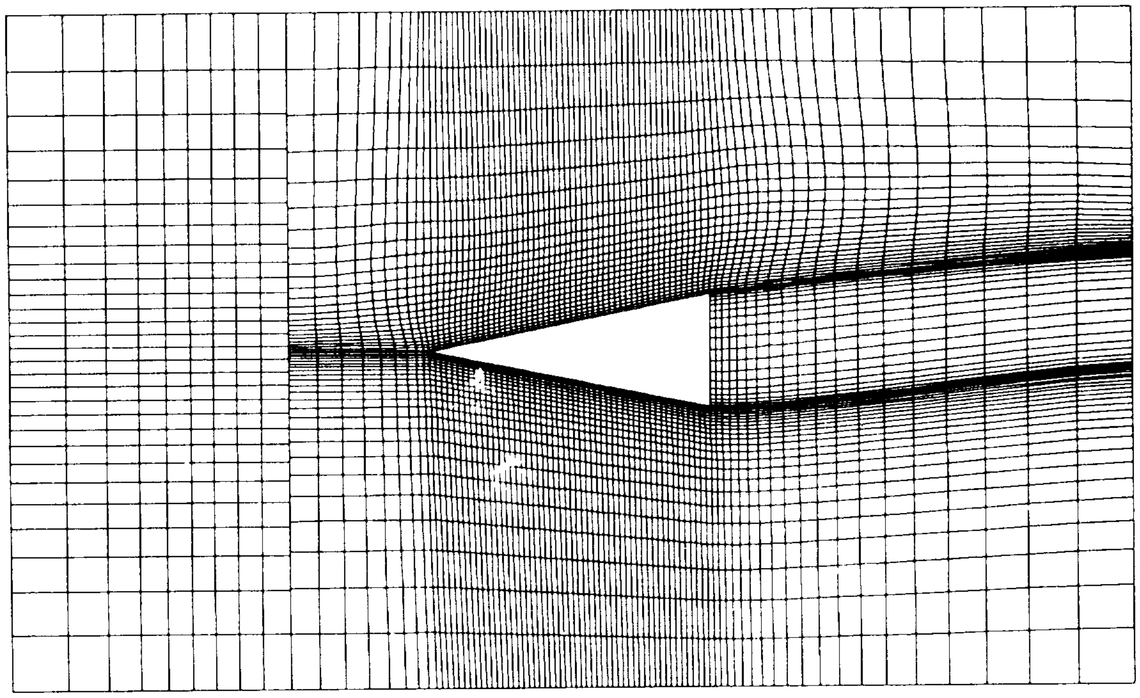

(b) 4-ZONE VISCOUS GRID

Figure 4.6. – Zonal grids for a wedge at $\alpha = 5°$.

base region of the airfoil. For the viscous case, an extra zone is placed upstream for the reasons given above. Figure 4.7 shows the O-type zoning for an airfoil of elliptical cross section. The zoning was generated for an inviscid flow, but EZGrid could have produced the same design for a viscous flow in this case, depending on the user bias profile setting. A viscous grid generated within an O-type zone would be more clustered near the body surface.

The next four cases show the effect of qualitative shape description on the zoning of a single body which looks something like the planform of a Shuttle-Orbiter-like vehicle. The way in which the shape of this body is described qualitatively by the user is a matter of individual interpretation. The body can be regarded as being composed of a single primitive part which has one of the following four shapes: ellipse, bullet, wedge, or teardrop. Figures 4.8a-d show the zonings and grids for each of these four shape descriptions, respectively, where all other factors are identical—each case is for inviscid flow at a 2° angle of attack. The user bias profile for each case uses the default settings except for the wake resolution parameter, which has been set to a *high-importance* value. This is the reason the zoning in figure 4.8a consists of a C-type zone instead of the O-type zone often used for an elliptical body. As can be seen, the zonings are quite different in terms of simplicity, skewness, smoothness, and regions of grid-point concentration.

Figures 4.9-4.11 provide three examples of zonings generated for cases in which only the user bias profile differs. The results in figures 4.9a-c were generated for inviscid flow at a 2° angle of attack about a body which is described as an ellipse. Figure 4.9a is identical to figure 4.8a, and is the zoning resulting from a wake-resolution setting of *high importance* and a surface-vs-field setting of *low importance*. The zoning in figure 4.9b differs from the previous zoning as a result of the reversal of the two user bias settings—wake resolution is deemphasized and the surface-vs-field quantities parameter is given a *high importance* value. The third zoning in this set is shown in figure 4.9c. It is a compromise, of sorts, in which both wake-resolution and surface-vs-field quantities are considered important. The zoning designed by EZGrid for this case uses two zones

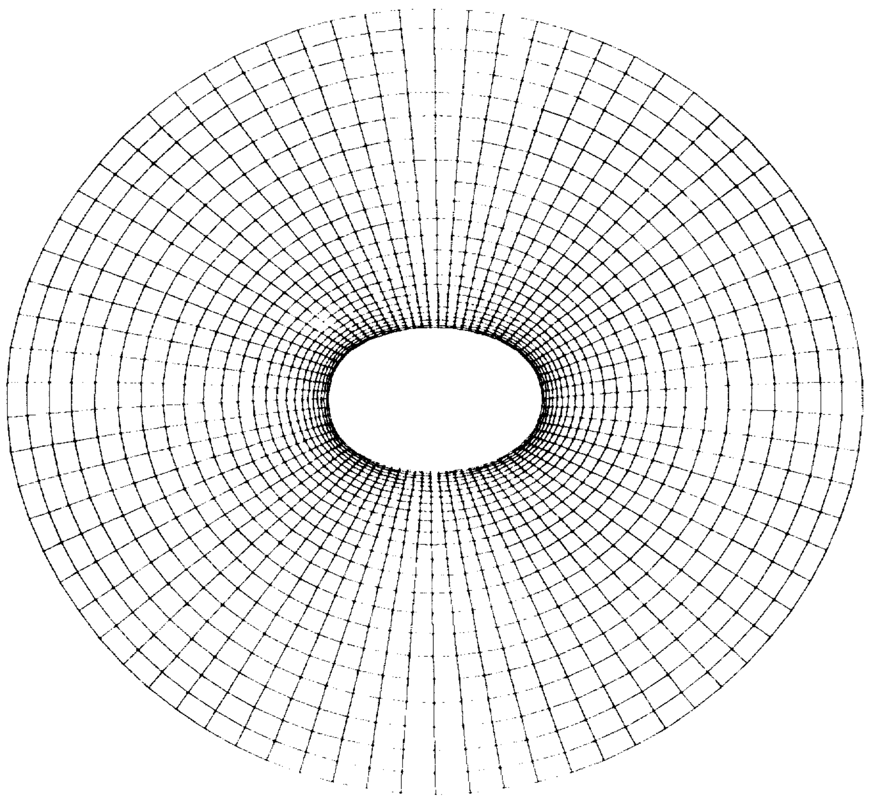


Figure 4.7. – Single-zone grid for an elliptical airfoil at $\alpha = 5°$.

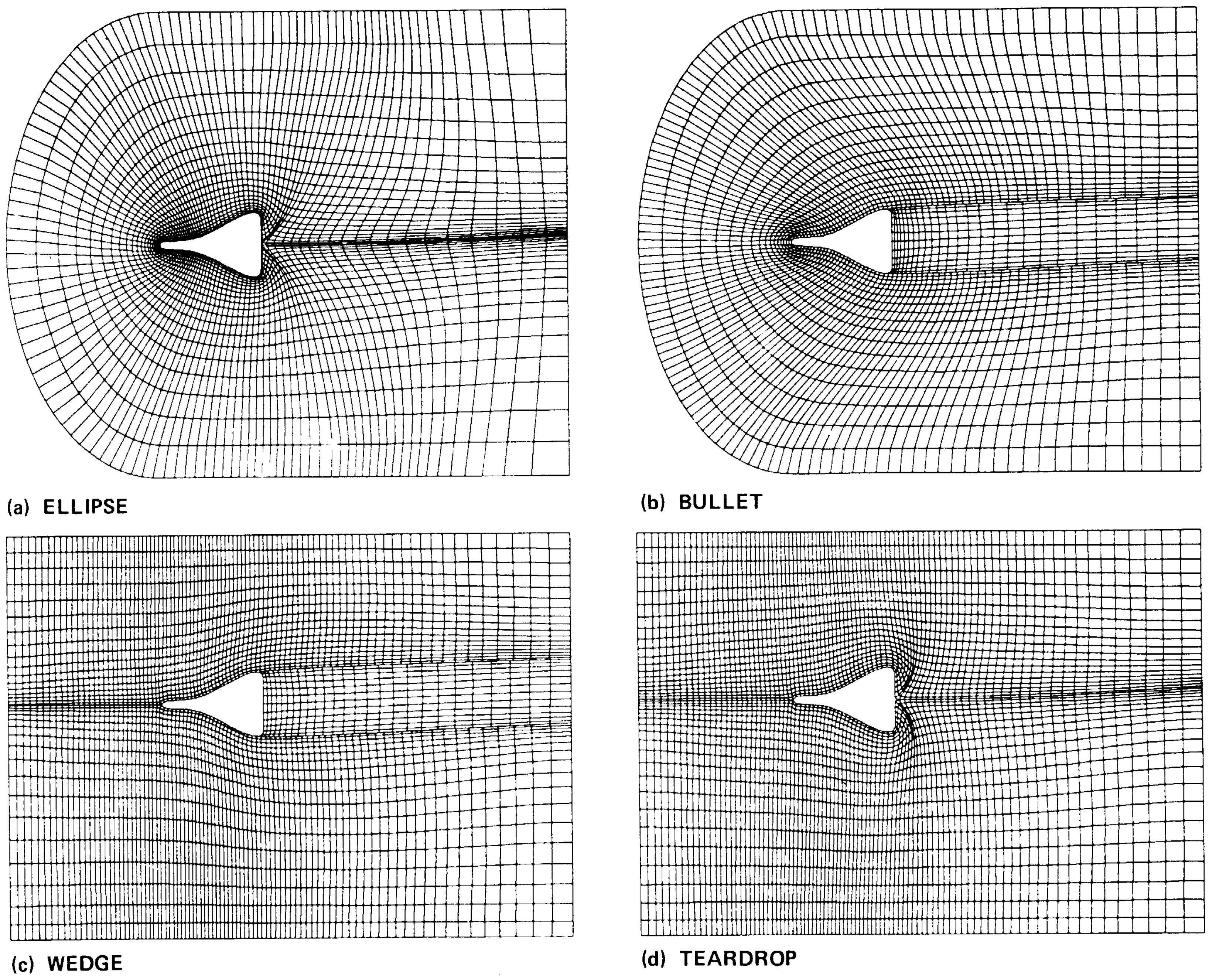
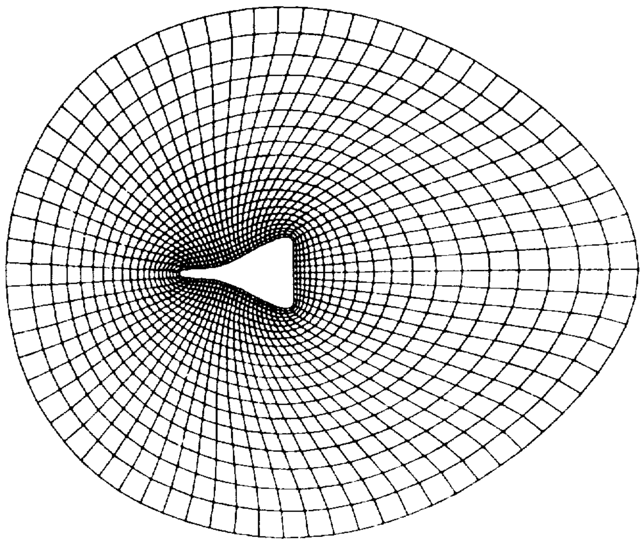
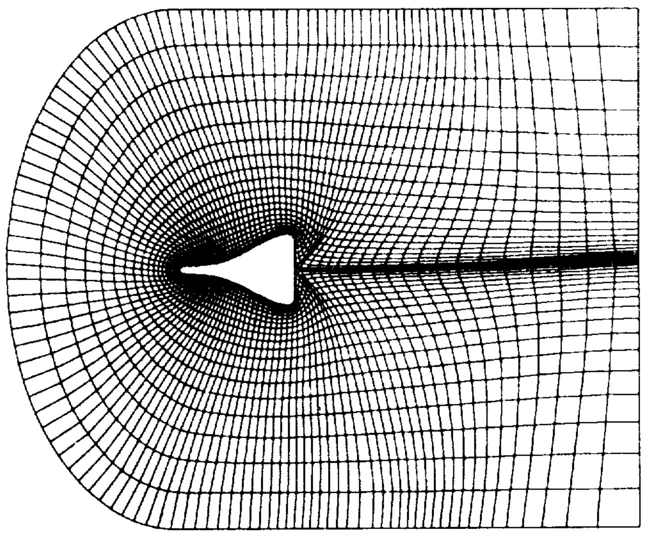


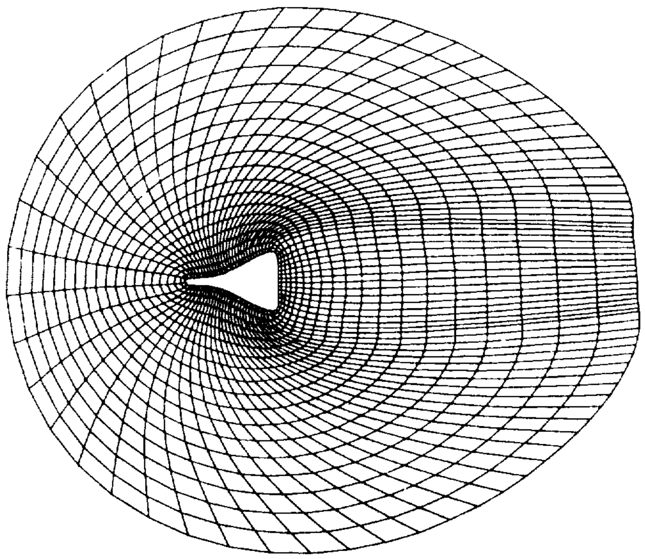
Figure 4.8. – The effect of qualitative shape description.



(a) WAKE – LOW IMPORTANCE
SURFACE QUANTITIES – HIGH IMPORTANCE

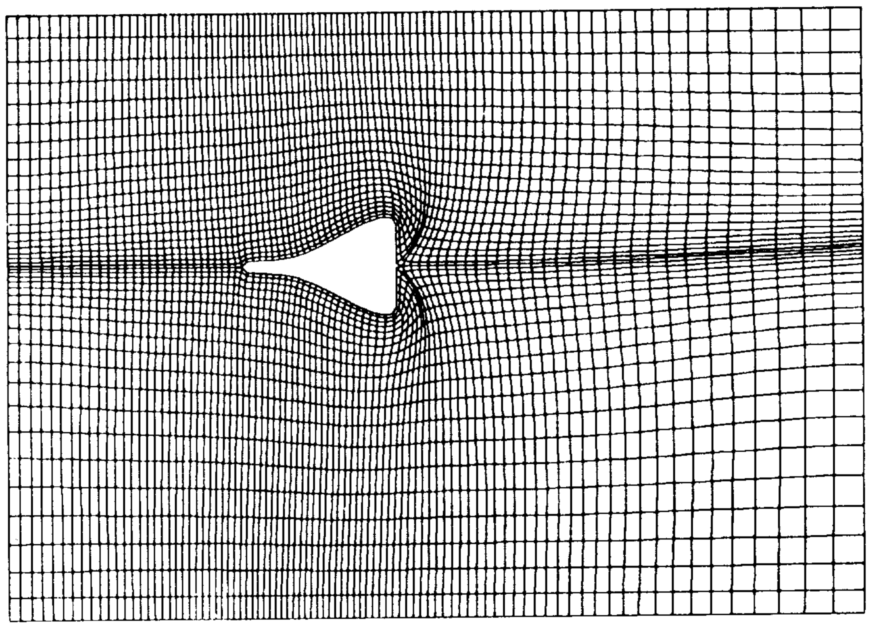


(b) WAKE – HIGH
SURFACE QUANTITIES – LOW



(c) WAKE – HIGH
SURFACE QUANTITIES – HIGH

Figure 4.9. – The effect of user bias on the zoning of a body described as an ellipse.


(a) WAKE = HIGH IMPORTANCE

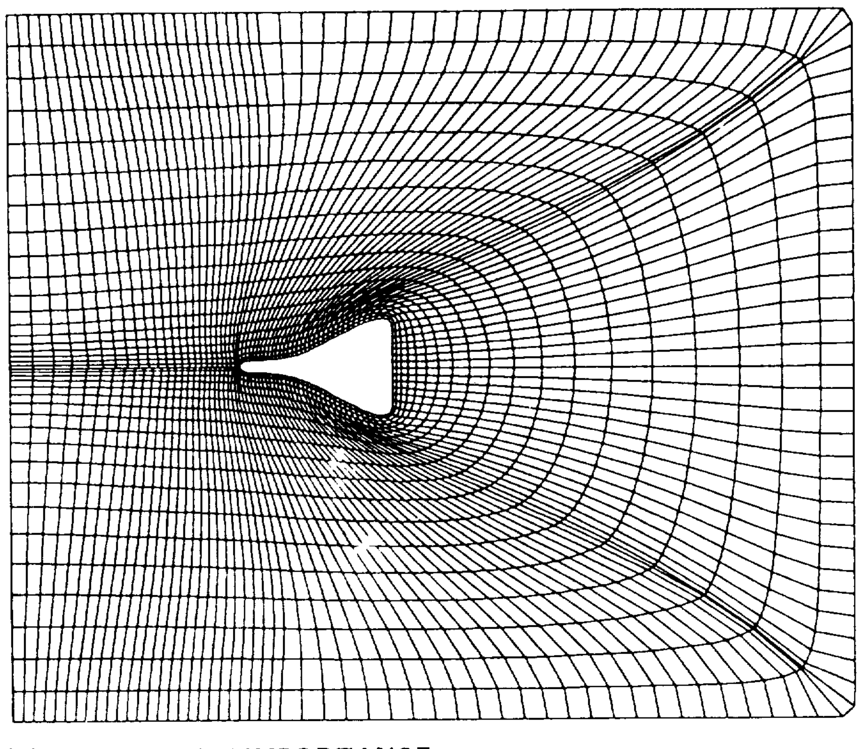
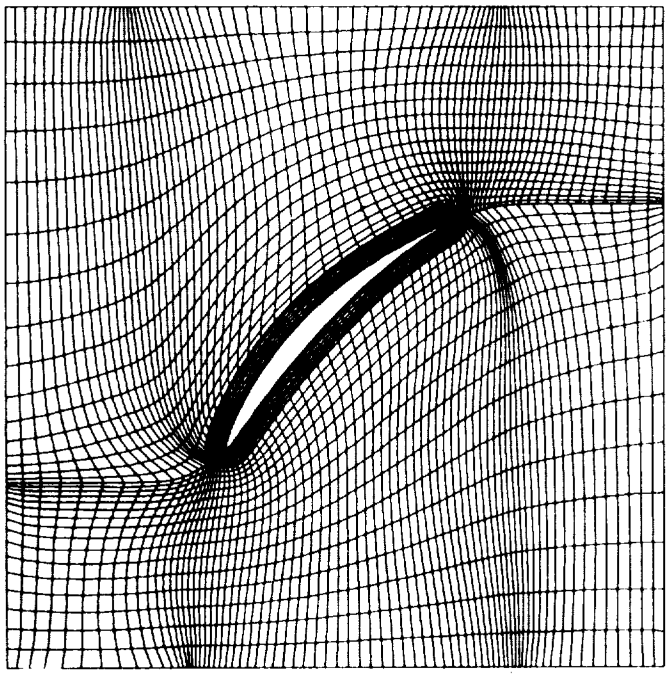

(b) WAKE = LOW IMPORTANCE

Figure 4.10. – The effect of user bias on the zoning of a body described as a teardrop.


(a) SMOOTHNESS = LOW IMPORTANCE

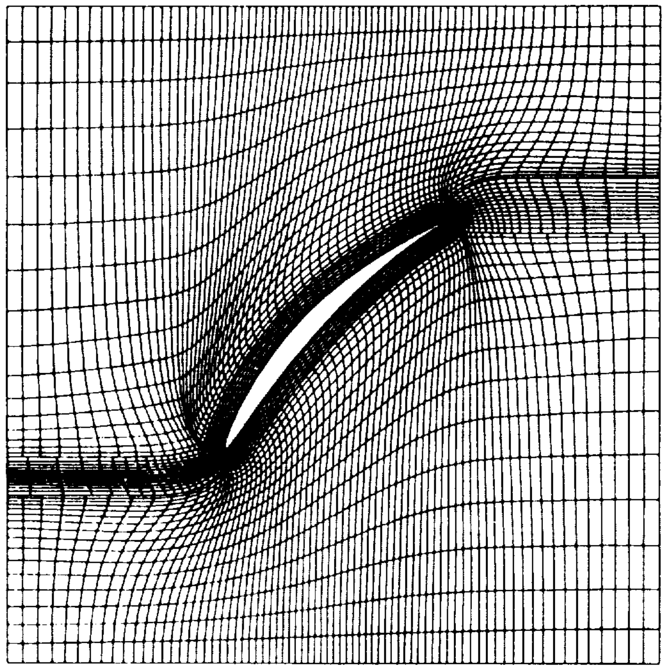

(b) SMOOTHNESS = HIGH IMPORTANCE

Figure 4.11. – Zonal grids for a rotor blade.

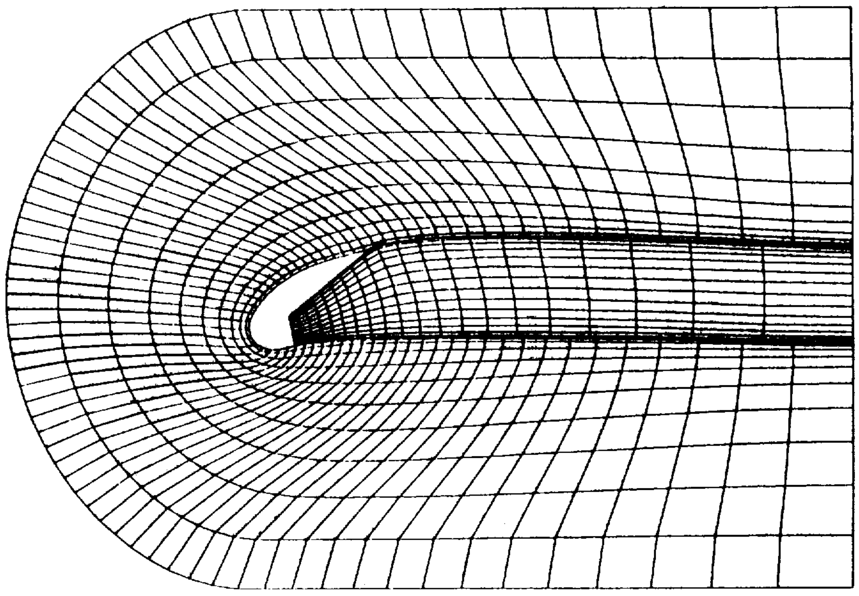
with the same topology to accomplish both concentration of grid points near the surface of the body and concentration of grid points in the wake region.

Figures 4.10a-b show zonings for the same geometry and flow conditions when the body shape is described as a teardrop. The zoning in figure 4.10a is identical to that shown in figure 4.8d, in which wake resolution is set to *high importance*. Figure 4.10b shows the zoning for the same shape description and conditions, but with wake resolution set to *low importance*. The resulting grids do reflect that emphasis.

The third example used to demonstrate the effect of user bias involves an internal flow field with a single rotor blade. The zonings are for a viscous flow, hence the body-oriented zone surrounding the blade. In figure 4.11a, zone smoothness is set to *low importance*, so EZGrid simply puts two H-type zones around the viscous O-type zone to complete the zoning. Figure 4.11b shows the zoning which results if the smoothness parameter is set to *high importance*. Additional zones are placed as buffers between the two H-type zones in an effort to smooth out the internal angles along those zonal interfaces. There is an increase in zone side smoothness from figure 4.11a to figure 4.11b, and the grid for the second case was easier to generate as a result.

The next four cases complete the single-body configuration study. These bodies are best described as compound bodies. There are more degrees of freedom (in individual interpretation) in the qualitative shape description of a compound body; one possibility is to describe it as a simple body, ignoring the fact that none of the primitive part shapes really describe it well. Another possibility is to vary the description of the various parts and how they are joined together. The zonings in figures 4.12a-b demonstrate the effect of describing what should be a compound body as a simple one. The zoning in figure 4.12a consists of two zones, one of which contains the region behind the step discontinuity in the geometry. In this case, the body shape has been described as compound, with two parts: the upstream part is bullet-shaped, with the blunt end forward, and the downstream part is wedge-shaped, with the base end joined to the base end of the bullet. The grids which were generated in these zones are smooth and can be clustered as needed (this case was computed for inviscid flow, but the same zoning would be appropriate for a viscous computation, with grid points more concentrated near the body surfaces). Figure 4.12b shows the zoning that results when the body shape is described as a simple body, with a single part of teardrop shape. Wake resolution is still possible, but the grid lines must turn the two sharp corners of the step discontinuity. The concave corner creates difficulties in grid generation because of the tendency (at least in any sort of elliptic solver) of the grid lines along the body to pull away from the corner, and of the grid lines normal to the body to coalesce.

The zonings in figure 4.13 demonstrate the effect of describing the parts of a compound body differently. The geometry is an axisymmetric cross section of an aeroassisted orbital transfer vehicle (AOTV). In both cases, the body has been divided into the same parts, as shown in figure 4.13a. In the first case, the upstream part has been described as a half-bullet (the *half-* denotes an axisymmetric part), where the blunt end of the bullet is upstream, and the base end is downstream, as shown in figure 4.13b. The base of the half-bullet is joined to the upstream end of a part described as a half-rectangle. In the second case, the downstream part is described in the same way, but the upstream part is seen as a half-ellipse, where the major axis of the ellipse is oriented vertically, as shown in figure 4.13c. The three-zone zoning in figure 4.13d is generated by EZGrid when given the first description and instructions that the computation is to be inviscid. Because the upstream part has a base end, EZGrid creates a bridging zone in the base region, connecting to the body at the point representing the top of the base. The alternative shape description results in the two-zone zoning shown in figure 4.13e. The elliptical shape description causes EZGrid to wrap an O-type zone around the upstream part. In both cases, the base end of the downstream part is treated in the same manner. Figures 4.13f-g show the grids and outer boundary shapes more clearly.


(a) COMPOUND SHAPE DESCRIPTION (2 PARTS)

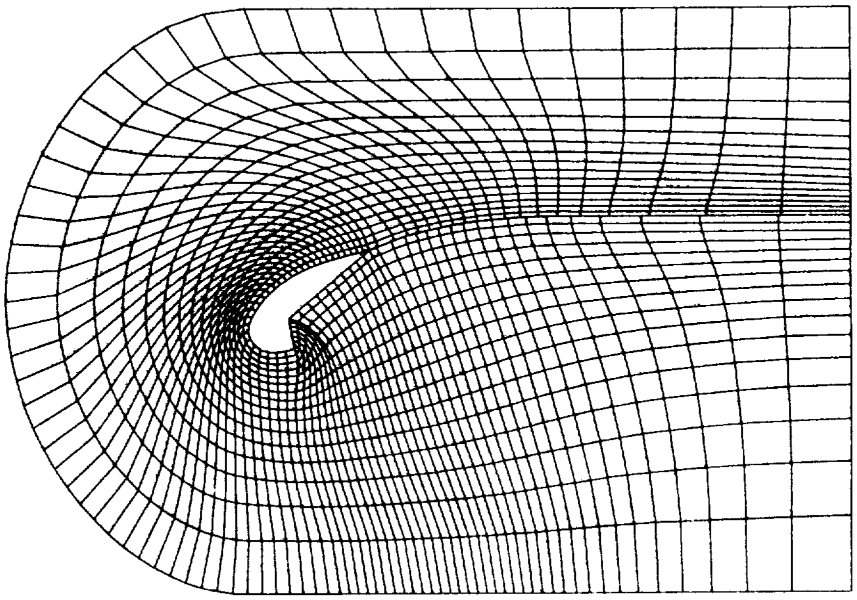
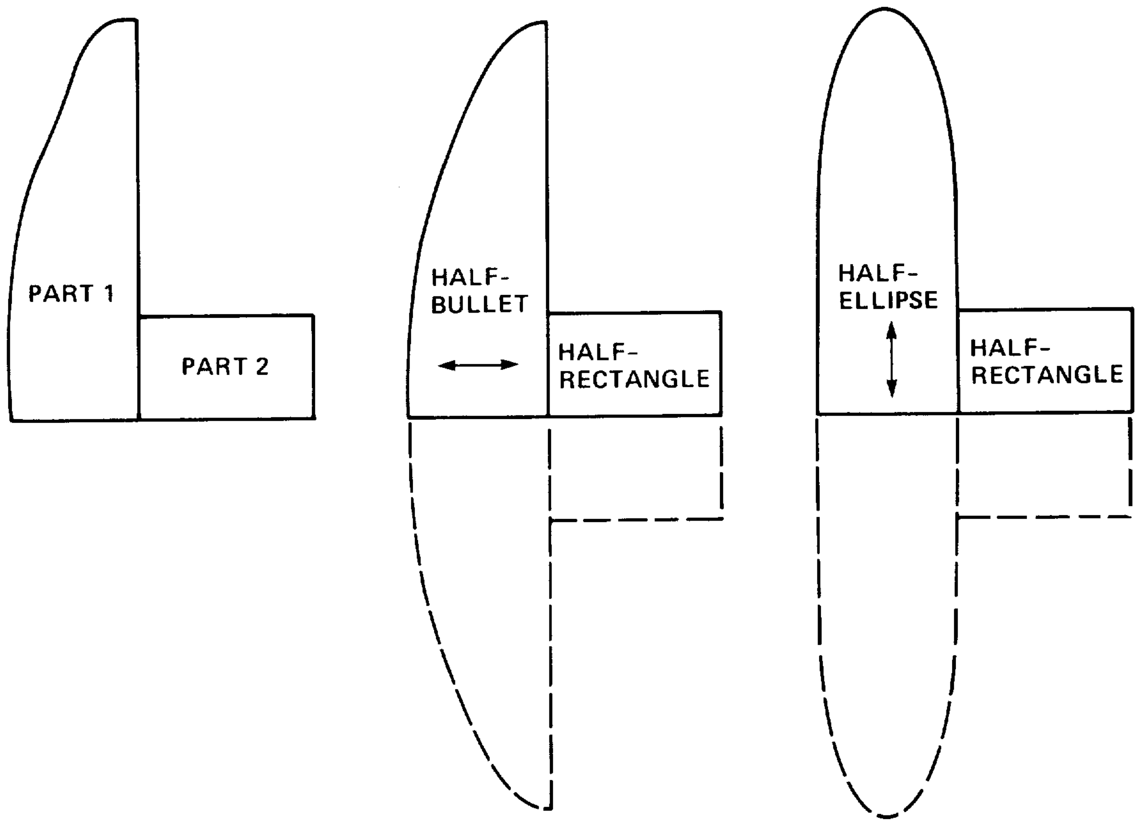

(b) SIMPLE SHAPE DESCRIPTION (1 PART)

Figure 4.12. – Zonal grids for an airfoil with discontinuity.



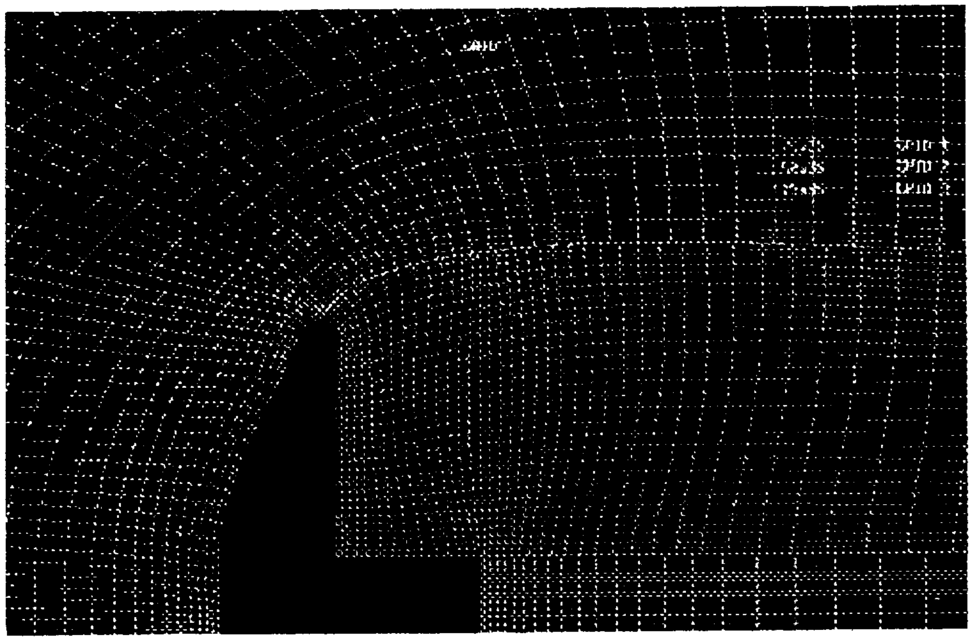
(a) 2-PART COMPOUND DESCRIPTION

(b) PART 1 DESCRIBED AS A HALF-BULLET

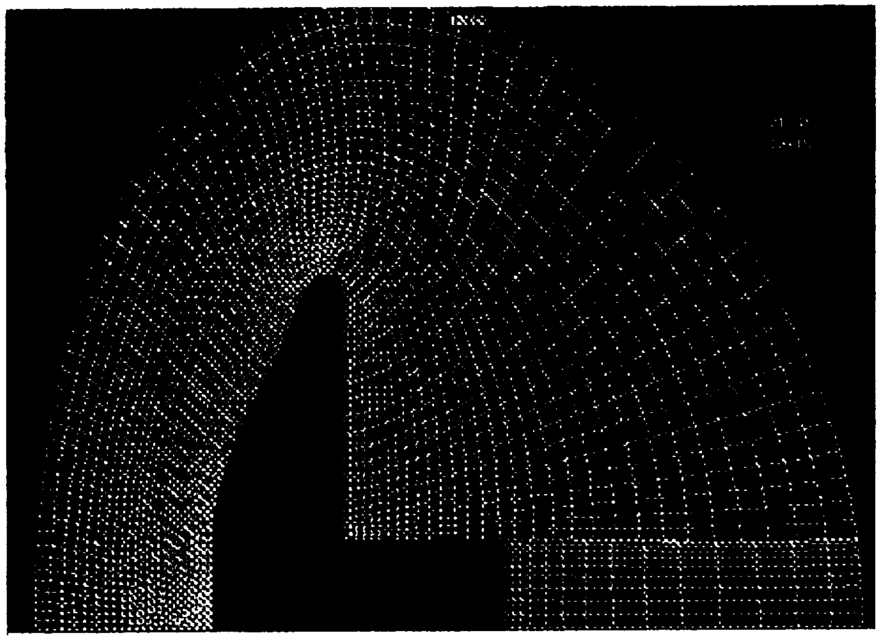
(c) PART 1 DESCRIBED AS A HALF-ELLIPSE

Figure 4.13. – AOTV cross-section.

ORIGINAL PAGE
COLOR PHOTOGRAPH



(d) Three-zone zoning for the half-bullet version.

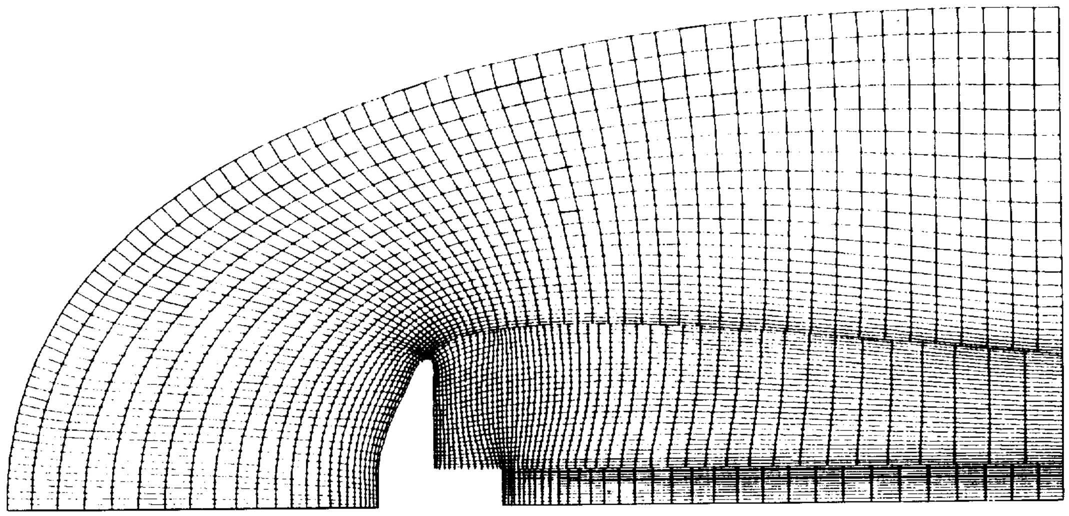


(e) Two-zone zoning for the half-ellipse version.

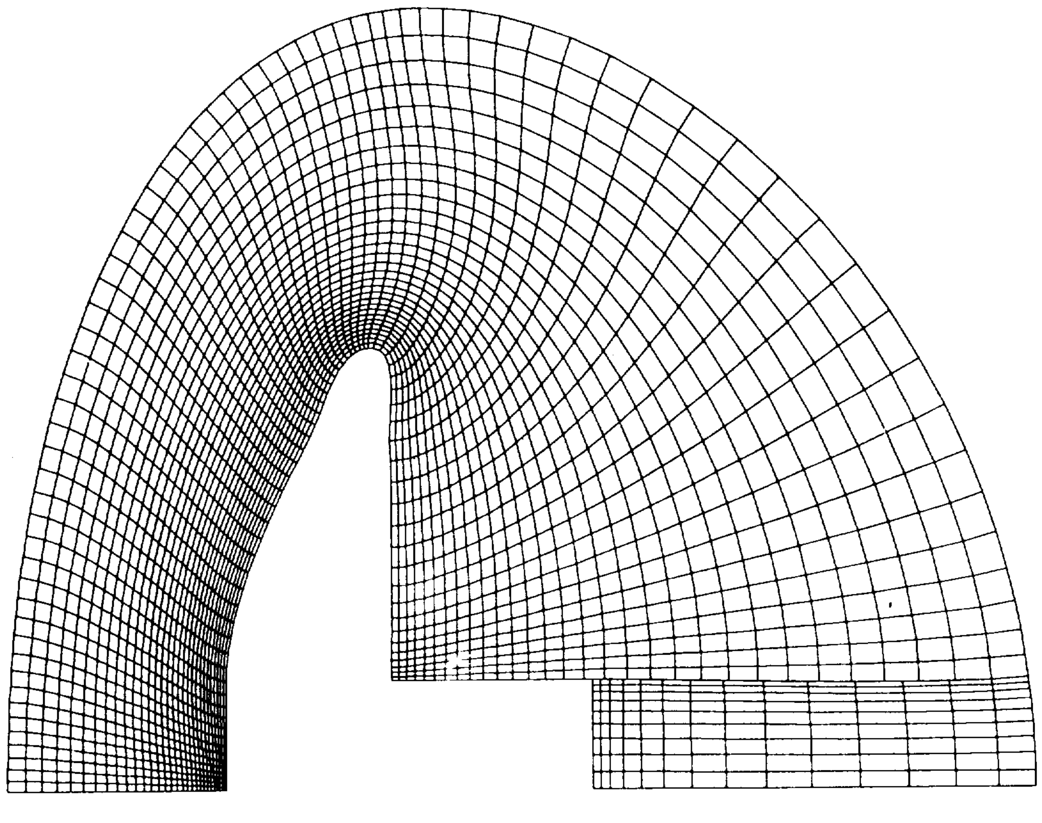
Figure 4.13. – Continued.

PRECEDING PAGE BLANK NOT FILMED

PAGE 64 INTENTIONALLY BLANK



(f) 3-ZONE GRID (HALF-BULLET)



(g) 2-ZONE GRID (HALF-ELLIPSE)

Figure 4.13. – Concluded.

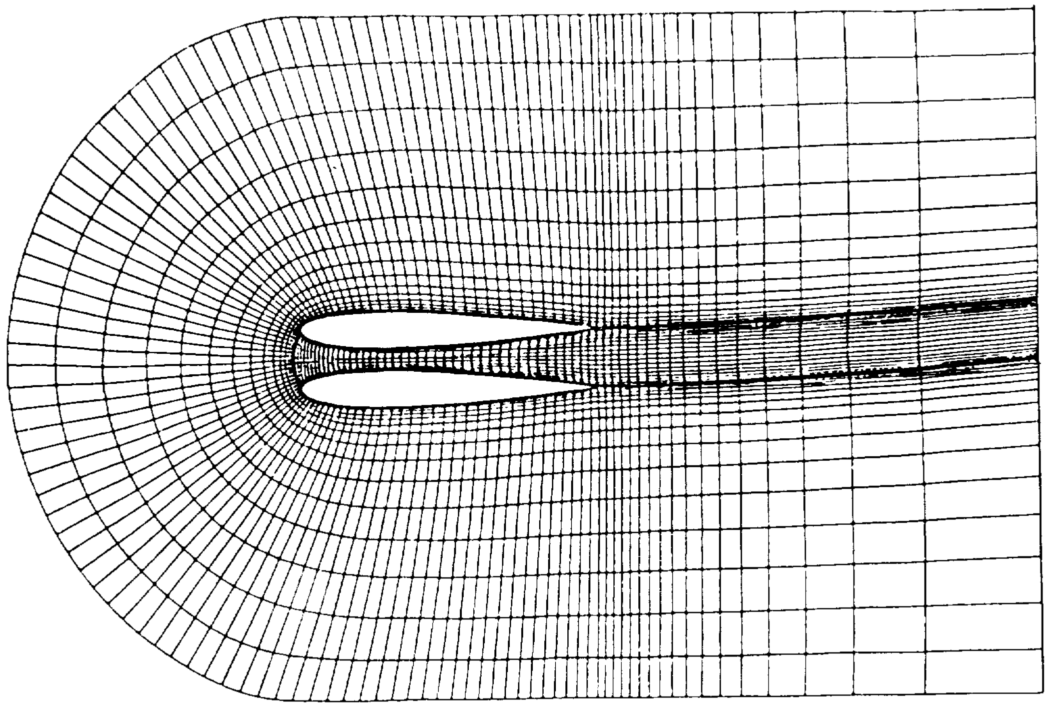
4.3 Multiple-Body Configurations

The bodies which appear in multiple-body configurations can be grouped by the user to reflect the expected influence of each body on the others. All of the bodies can be assigned to the same grouping (the single-grouping category), or they may be separated into several groupings (the multiple-grouping category). Nine configurations are used here to illustrate the effect of relative position, physics, and user bias on the zoning of single groupings. All of the zonings shown for the single-grouping configurations are generated for either inviscid or viscous flow at an angle of attack of 5°.

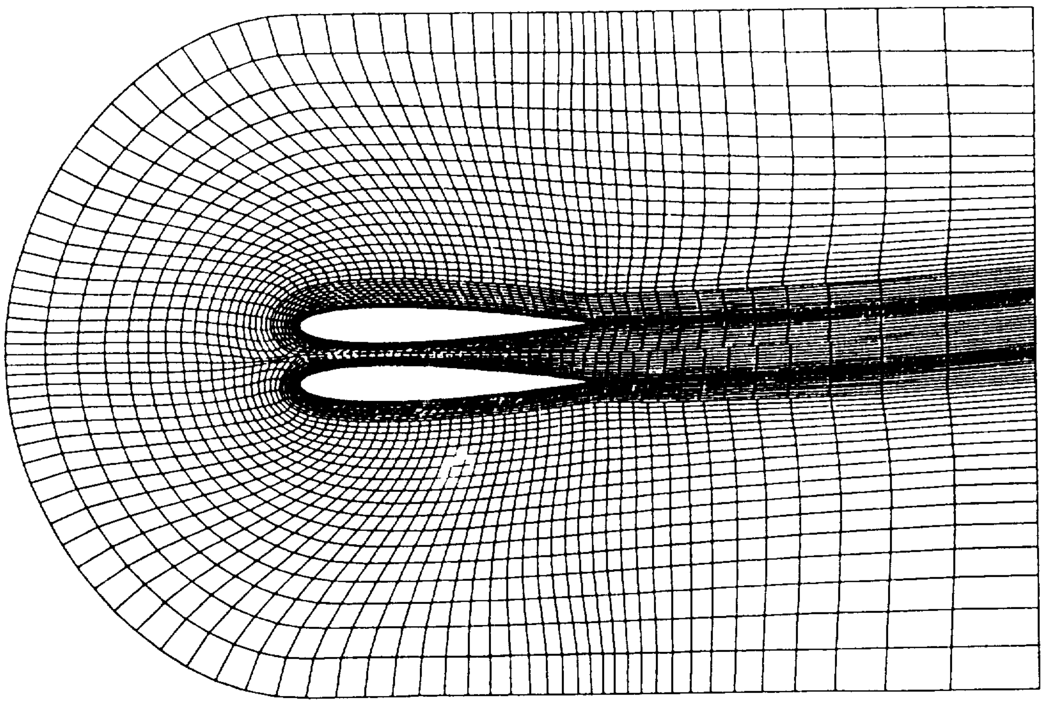
The first configuration in this category is a pair of NACA 0012 airfoils that are vertically aligned and have an offset distance of one-fifth chord. Figure 4.14a shows the zoning generated for an inviscid computation in which singularities and tuple points (where three or more zones meet) were allowed (the default user bias profile). This zoning consists of two zones, one which forms a bridge between the two airfoils, and one which is a C-type zone around the resultant body composed of the two airfoils and the bridge. The same configuration was zoned for a viscous computation using the same user bias profile; the result is shown in figure 4.14b. Body-oriented C-type zones were placed by EZGrid around each airfoil to facilitate the computation of the viscous terms of the fluid dynamic equations. Because the airfoils are relatively close to each other, the C-type zones are joined together without any intermediate zone between them. The remaining flow field is contained by an outer C-type zone.

Figures 4.14c-d show the zonings for a configuration which differs from the previous one only in a larger offset distance between the airfoils. Both zonings are for a viscous computation. The user bias profile for the zoning in figure 4.14c is identical to that of figure 4.14b. The difference in the zonings is due solely to the different offset distance. The user bias profile for the zoning in figure 4.14d was changed to disallow tuple points, to discourage singularities at bodies, and to reduce the importance of zone side mapping disparity (the ratio of the maximum and minimum distances separating opposite sides of a zone). As can be seen, the resulting zoning has no tuple points or singularities (the zoning in fig. 4.14c has two tuple points), and has a fairly large zone side mapping disparity. Also of note is the fact that the zoning in figure 4.14d is simpler than either of the zonings of figures 4.14b-c (it has only two zones, which meet along a single, straight interface, as opposed to three or four zones with one or more tuple points and several curved zonal interfaces). The zoning in figure 4.14d is not much simpler than the zoning in figure 4.14a, but is better suited to viscous flow computations. The major drawback to this zoning design is that the large mapping disparity (it would be much larger if the outer boundary were to assume its normal dimensions for an actual computation) causes grid generation difficulties. It was extremely difficult to distribute the grid points along the boundaries of those two zones to achieve even the relatively poor-quality grid shown. When setting the user bias profile, the user must evaluate the tradeoffs involved in the various parameters.

The next set of configurations are all of the *staggered pair* variety, where one airfoil is offset from the other in both the vertical and horizontal directions. The first three configurations demonstrate the effect of trailing-edge angle and position on the wake curves of both airfoils. The zonings were generated for inviscid computations with the default user bias profile. Figure 4.15a shows the zoning for a pair of airfoils in which the lower airfoil is slightly downstream of the upper one, and is cambered so that the trailing edge points away from the neighboring airfoil. In such a situation, the planned wake curve of the downstream airfoil does not interfere with or influence the wake curve which would be generated for the upper airfoil, and so both curves can be generated independently. The zoning which results is similar to that in figure 4.14a. Figure 4.15b shows the zoning for a configuration in which the upper airfoil is cambered and slightly downstream of the lower airfoil. In this situation, there is clearly the potential for interference between the wake curves. EZGrid first uses the "follow-wake" strategy for the downstream airfoil, so that its wake curve is as it would be if the airfoil were alone. Then a strategy called "follow-neighbor-wake" is employed for the upstream airfoil wake curve. As can be seen from figure 4.15b, the wake curve of the lower airfoil does indeed follow the path of the wake curve above. The zoning pictured in figure 4.15c is for a configuration in which the upper airfoil is cambered and is slightly upstream

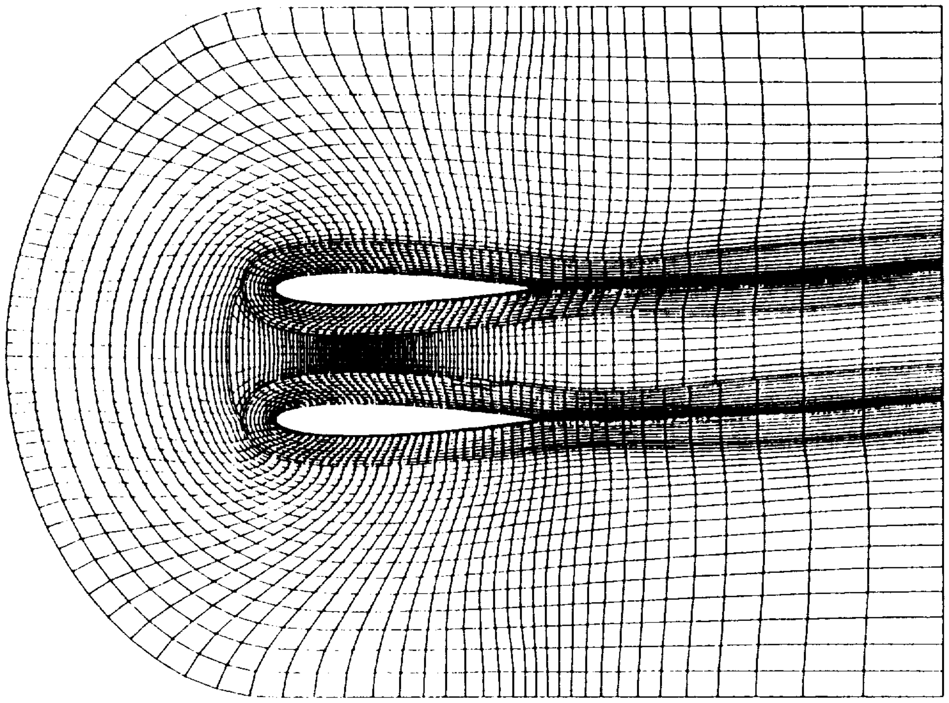


(a) INVISCID 2-ZONE GRID

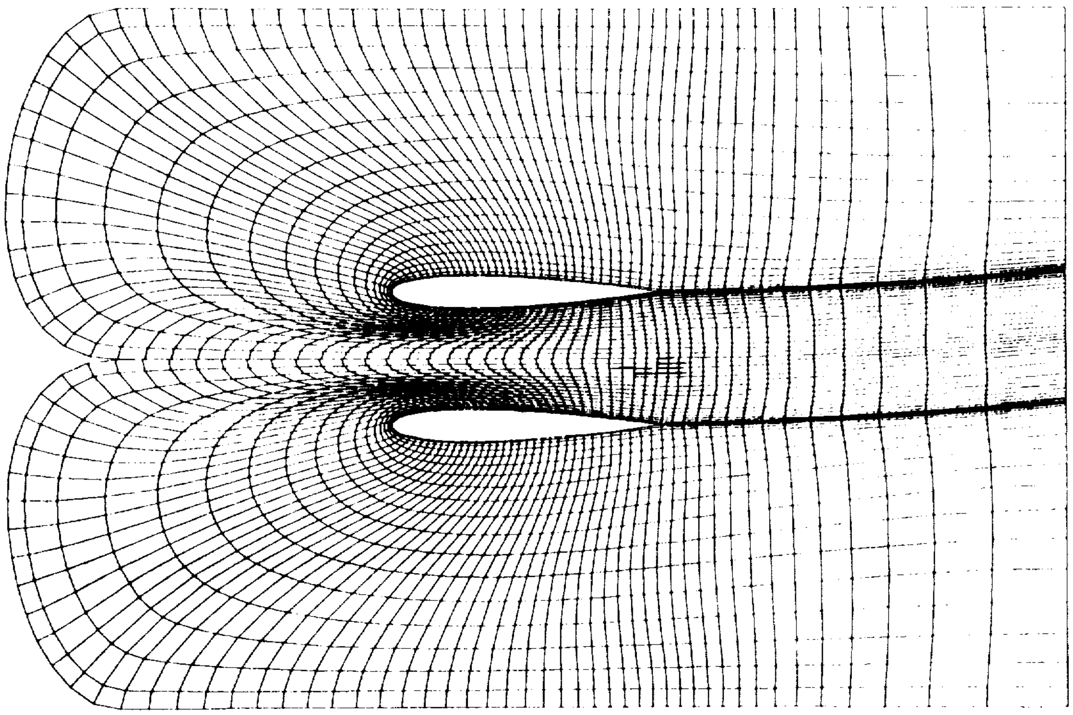


(b) VISCOUS 3-ZONE GRID – BODIES CLOSE TOGETHER

Figure 4.14. – Zonal grids for a vertically aligned NACA 0012 pair.

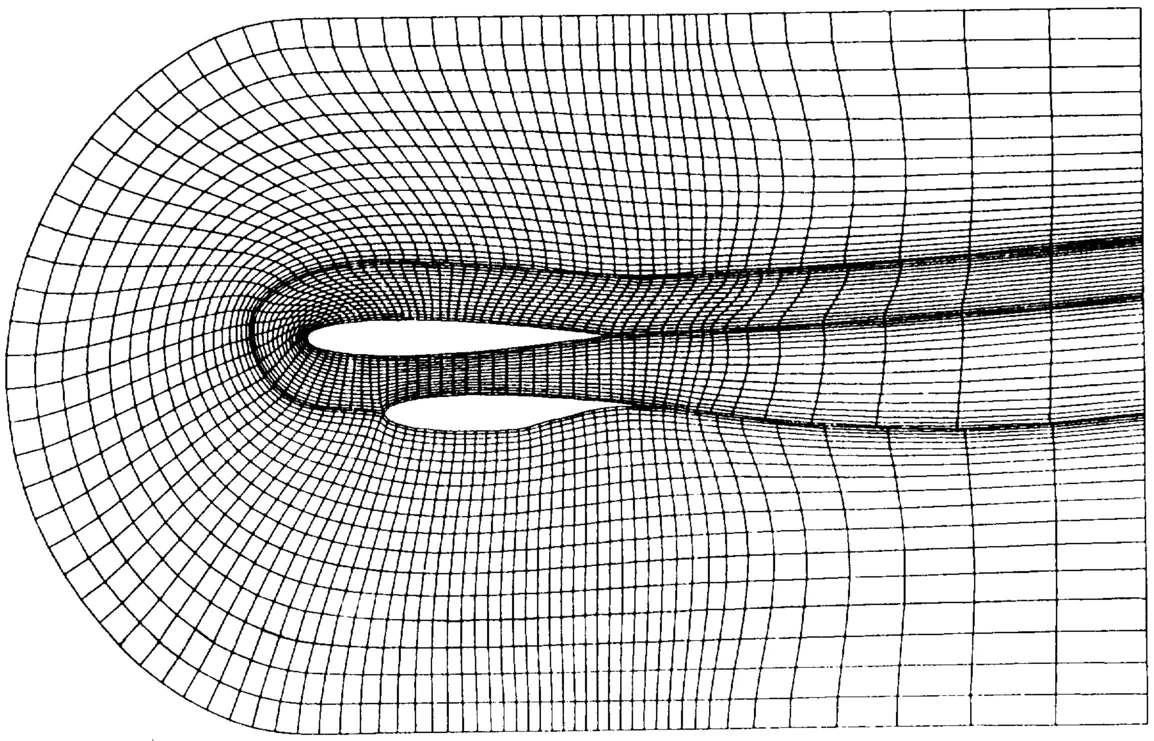


(c) VISCOUS 4-ZONE GRID – BODIES FAR APART



(d) VISCOUS 2-ZONE GRID – NONDEFAULT USER BIAS PROFILE

Figure 4.14. – Concluded.


(a) CAMBERED AIRFOIL BELOW AND DOWNSTREAM

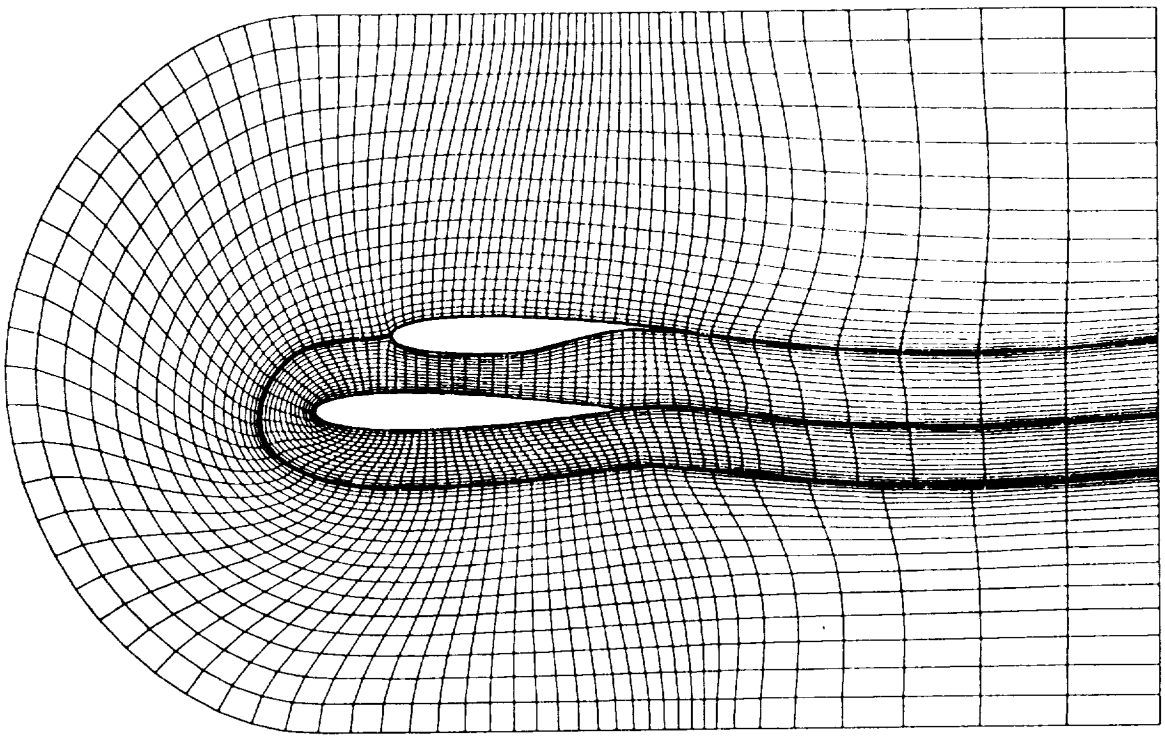
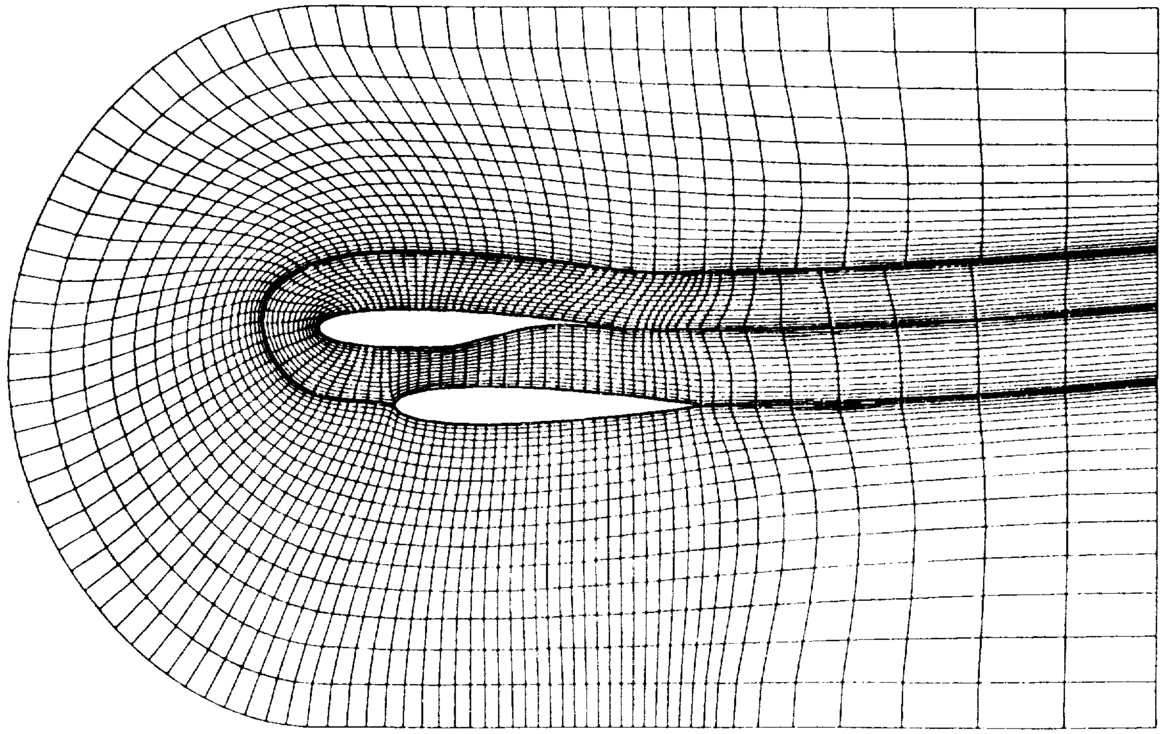

(b) CAMBERED AIRFOIL ABOVE AND DOWNSTREAM

Figure 4.15. – Staggered airfoil pair wake curve study.



(c) CAMBERED AIRFOIL ABOVE AND UPSTREAM

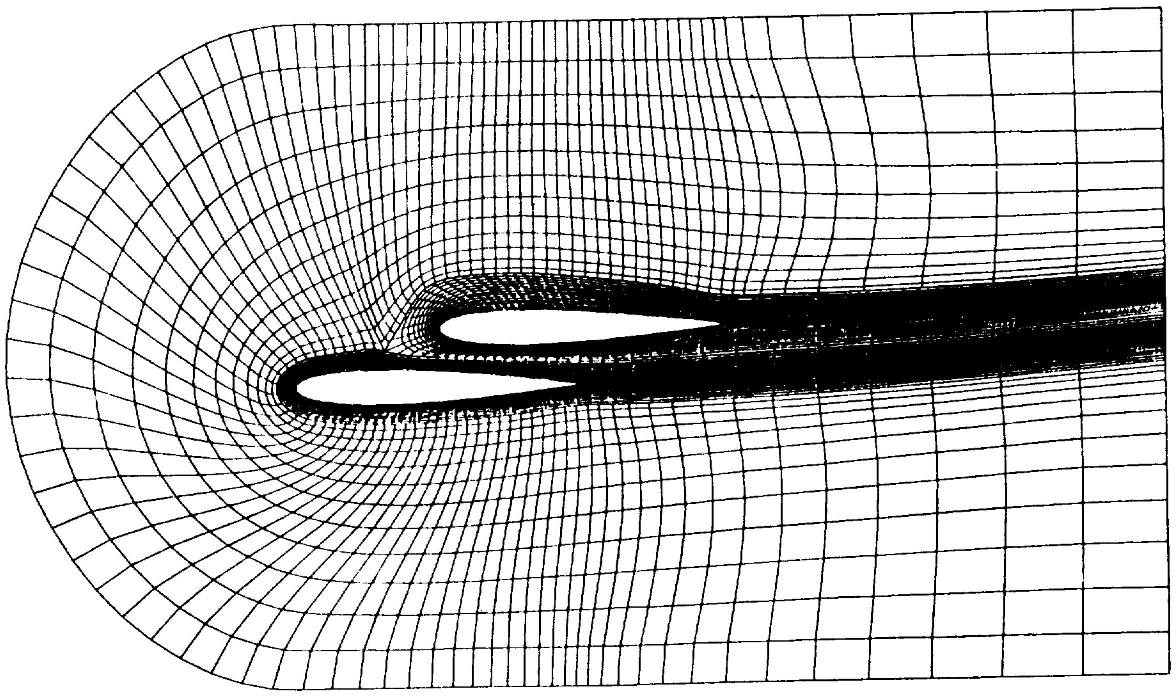
Figure 4.15. – Concluded.

of the lower airfoil. EZGrid concentrates attention first on the wake curve of the downstream airfoil, so in this situation, the interference is resolved by adapting the converging upper wake curve to the path of the lower wake curve, as seen in figure 4.15c.

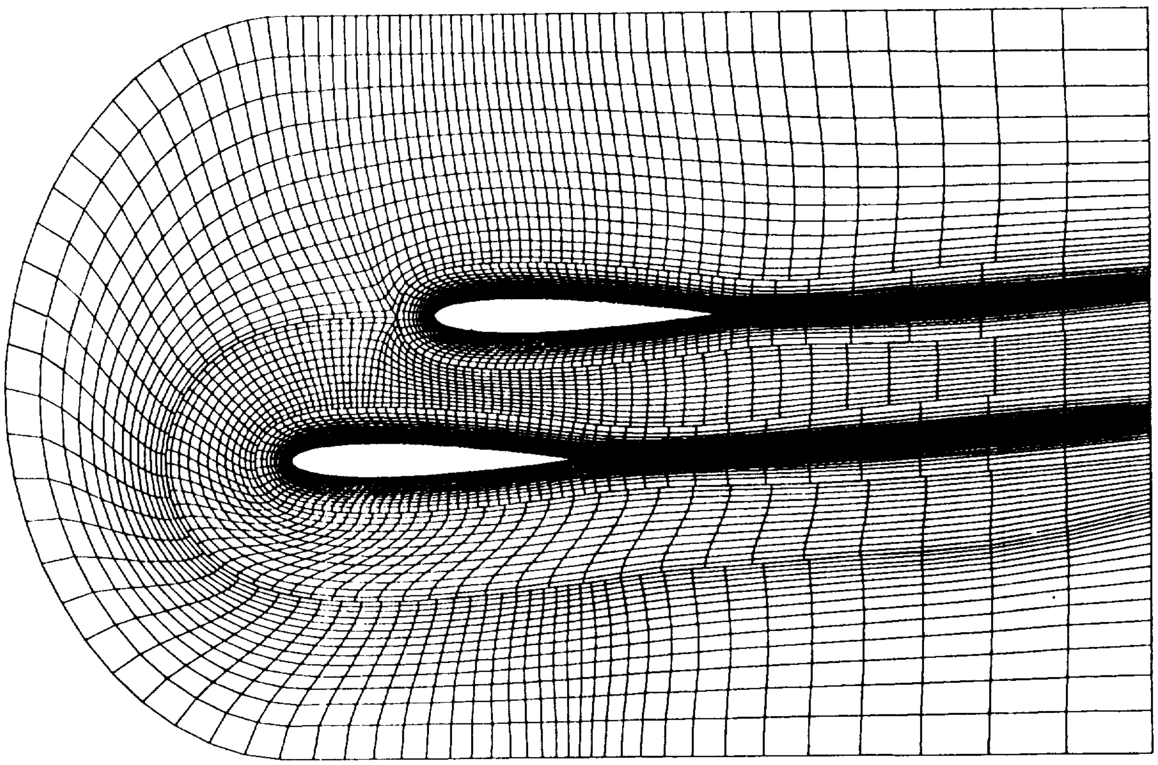
The last two configurations in the staggered-pair family are used to demonstrate the effect of offset distance and user bias profile on the zoning of such configurations. All of these zonings are for viscous computations. The zoning of figure 4.16a shows two C-type zones around the airfoils, with an outer C-type zone containing the rest of the flow field. As in figure 4.14b, the bodies are close enough together that the C-type zones are constructed adjacent to each other. Figure 4.16b shows the result of moving the two bodies farther apart. Instead of placing a simple bridging zone between the two resultant bodies, as was done in figure 4.14c, EZGrid chose to connect the bodies by wrapping another C-type zone around the upstream one which incorporates part of the boundary of the downstream body. The first two zonings were generated using the default user bias profile. For the third zoning, shown in figure 4.16c, the profile was modified to discourage both singularities and tuple points, and to set mapping disparity to *medium importance*. The resulting zoning is similar to that in figure 4.14d.

Three different zonings were generated by EZGrid for the next configuration, which consists of two NACA 0012 airfoils horizontally aligned, one directly behind the other. Each of these zonings would be appropriate for either an inviscid or viscous computation. The difference in the zonings is due to a difference in the user bias profile for each case. The zoning in figure 4.17a resulted from the use of the default user bias profile, in which singularities at bodies are allowed. Figure 4.17b shows the result of disallowing singularities. When both singularities and tuple points were discouraged or disallowed, and mapping disparity and smoothness were rated as less important, the zoning in figure 4.17c was generated.

The final multiple-body configuration in which the bodies all belong to a single grouping is a tri-element augmentor wing airfoil. The two different zonings shown in figures 4.18a-b are for an inviscid and a viscous computation, respectively. The viscous zoning includes C-type zones for each airfoil element. All three bodies, the upstream element and the two downstream resultant

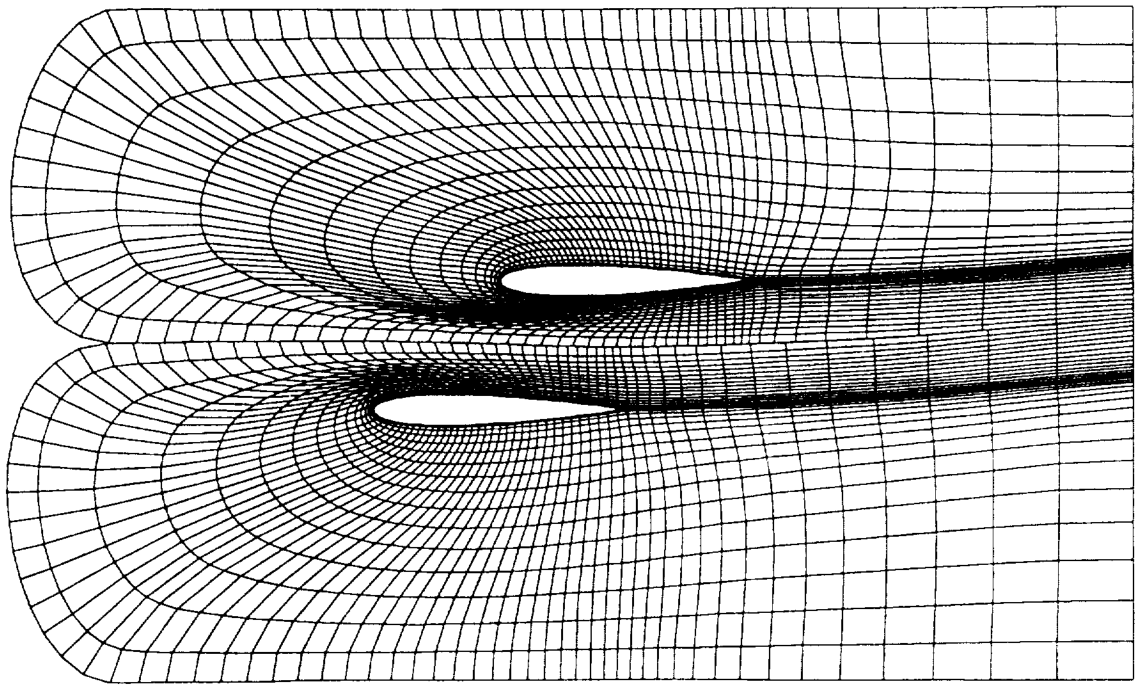


(a) VISCOUS 3-ZONE GRID – BODIES CLOSE TOGETHER



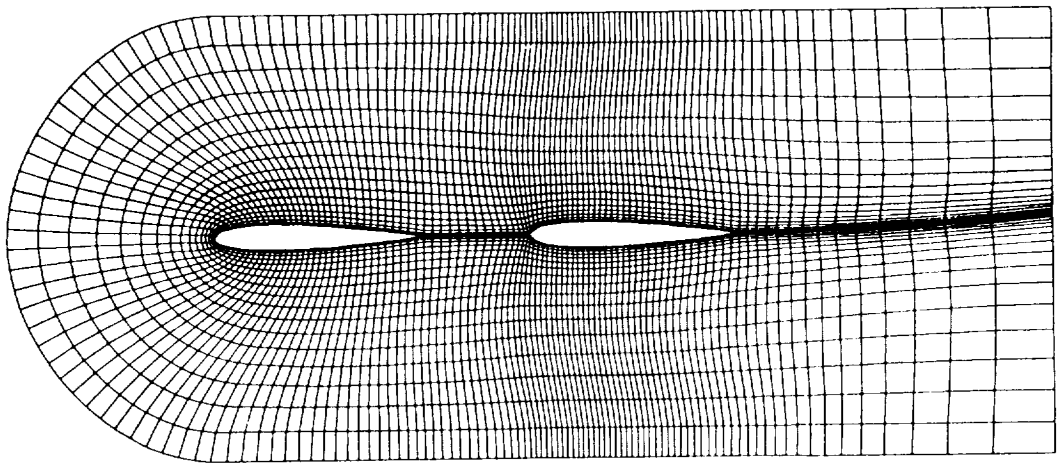
(b) VISCOUS 4-ZONE GRID – BODIES FAR APART

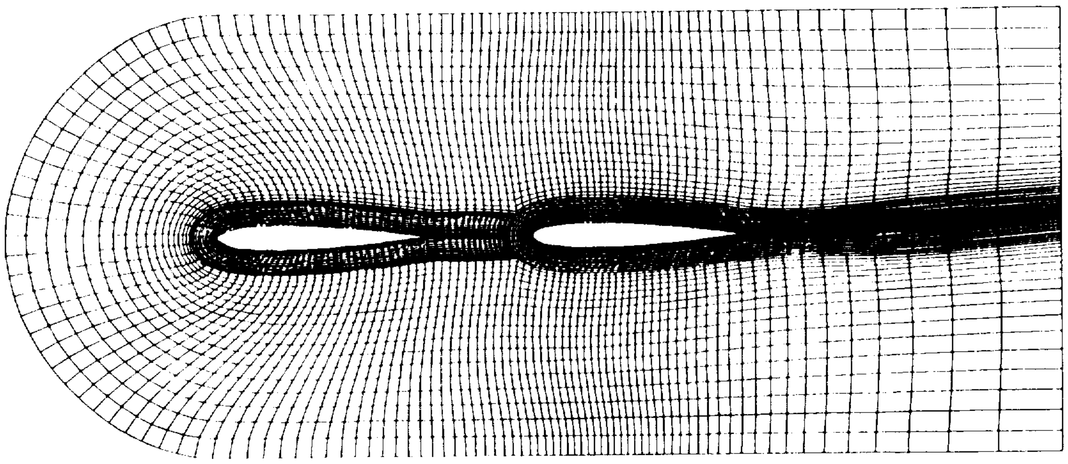
Figure 4.16. – Zonal grids for a staggered NACA 0012 pair.



(c) VISCOUS 2-ZONE GRID – NONDEFAULT USER BIAS PROFILE

Figure 4.16. – Concluded.


(a) SINGLE-ZONE GRID – DEFAULT USER BIAS PROFILE


(b) 3-ZONE GRID – SINGULARITIES = NOT ALLOWED

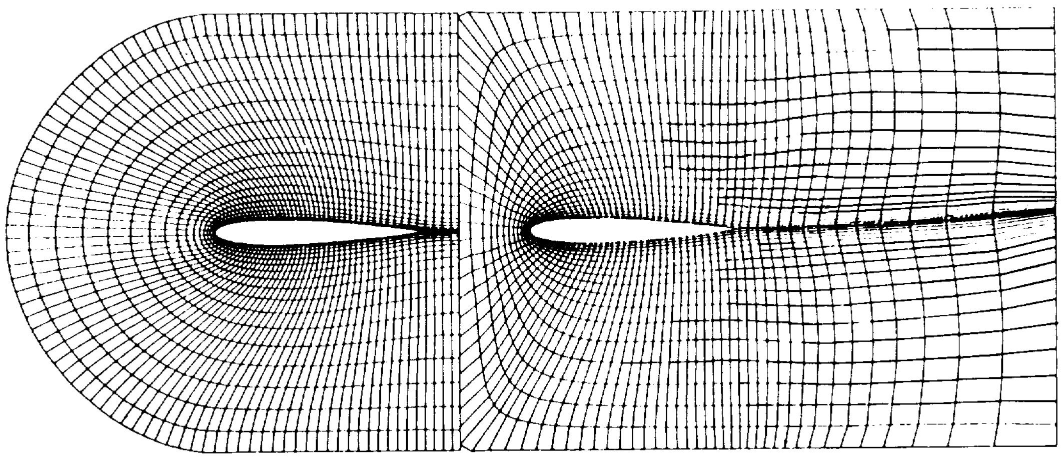
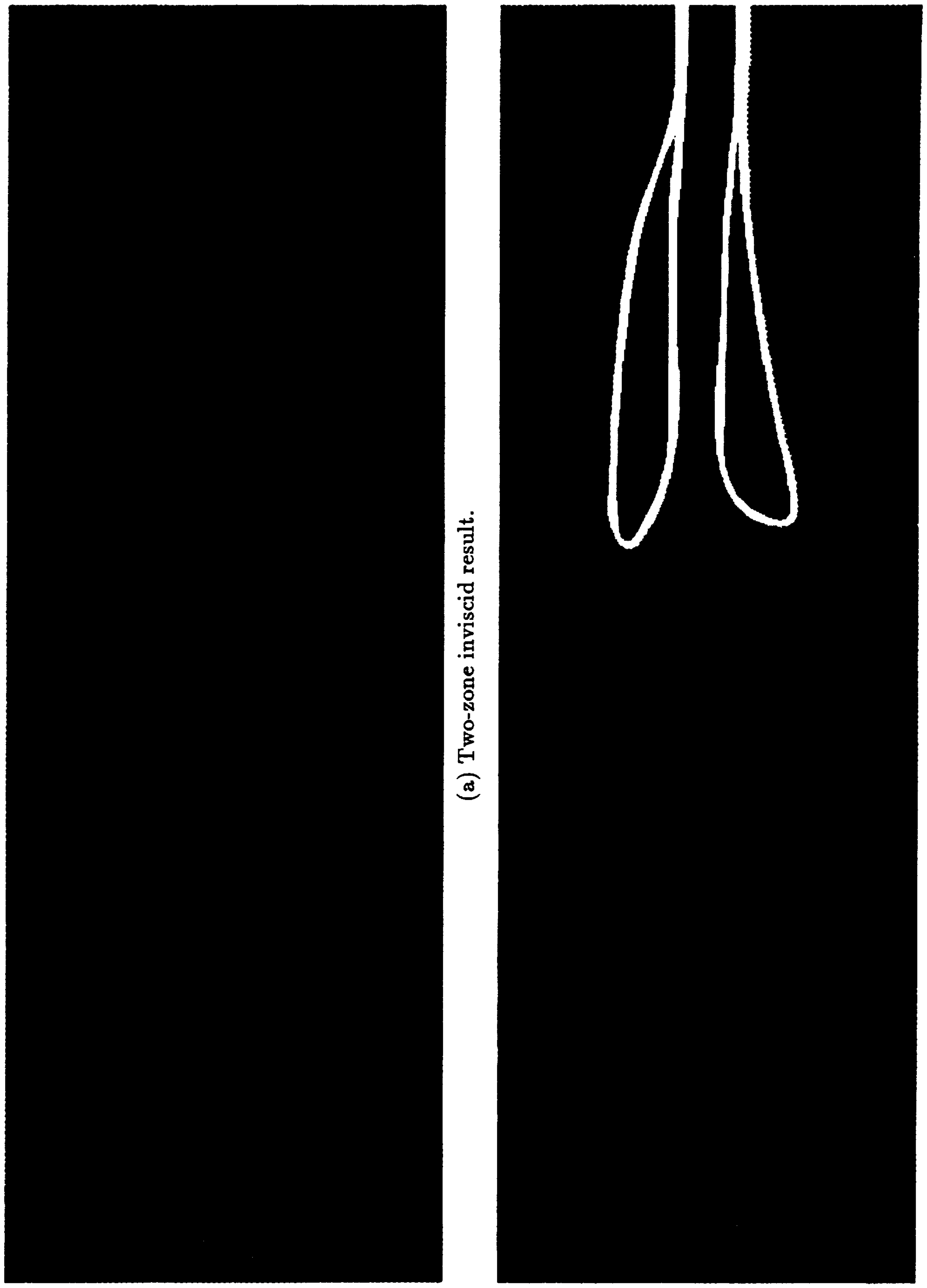

(c) 2-ZONE GRID – SINGULARITIES AND TUPLES = DISCOURAGED, MAPPING DISPARITY = LOW

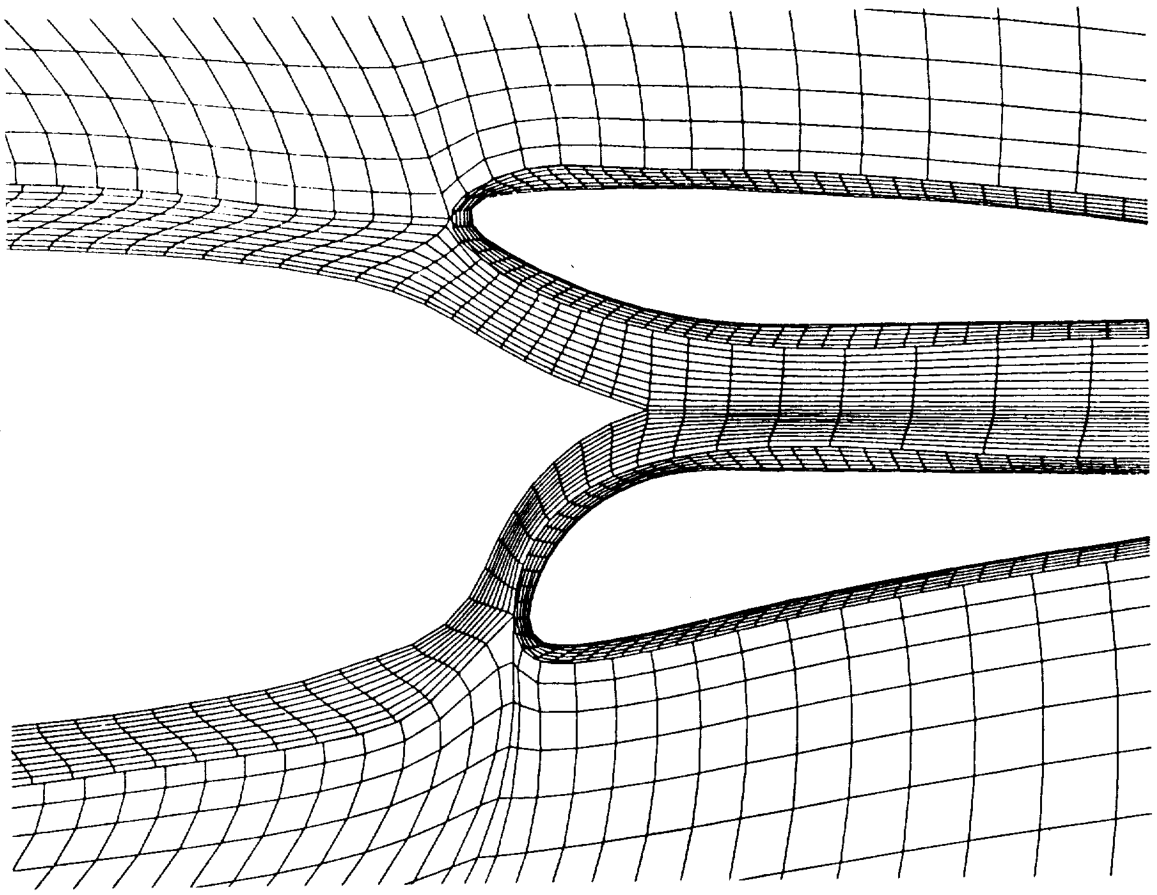
Figure 4.17. – Zonal grids for a horizontally aligned NACA 0012 pair.



(a) Two-zone inviscid result.

(b) Four-zone viscous result.

Figure 4.18. – Augmentor-wing airfoil.



(c) DETAIL OF VISCOUS GRID

Figure 4.18. – Concluded.

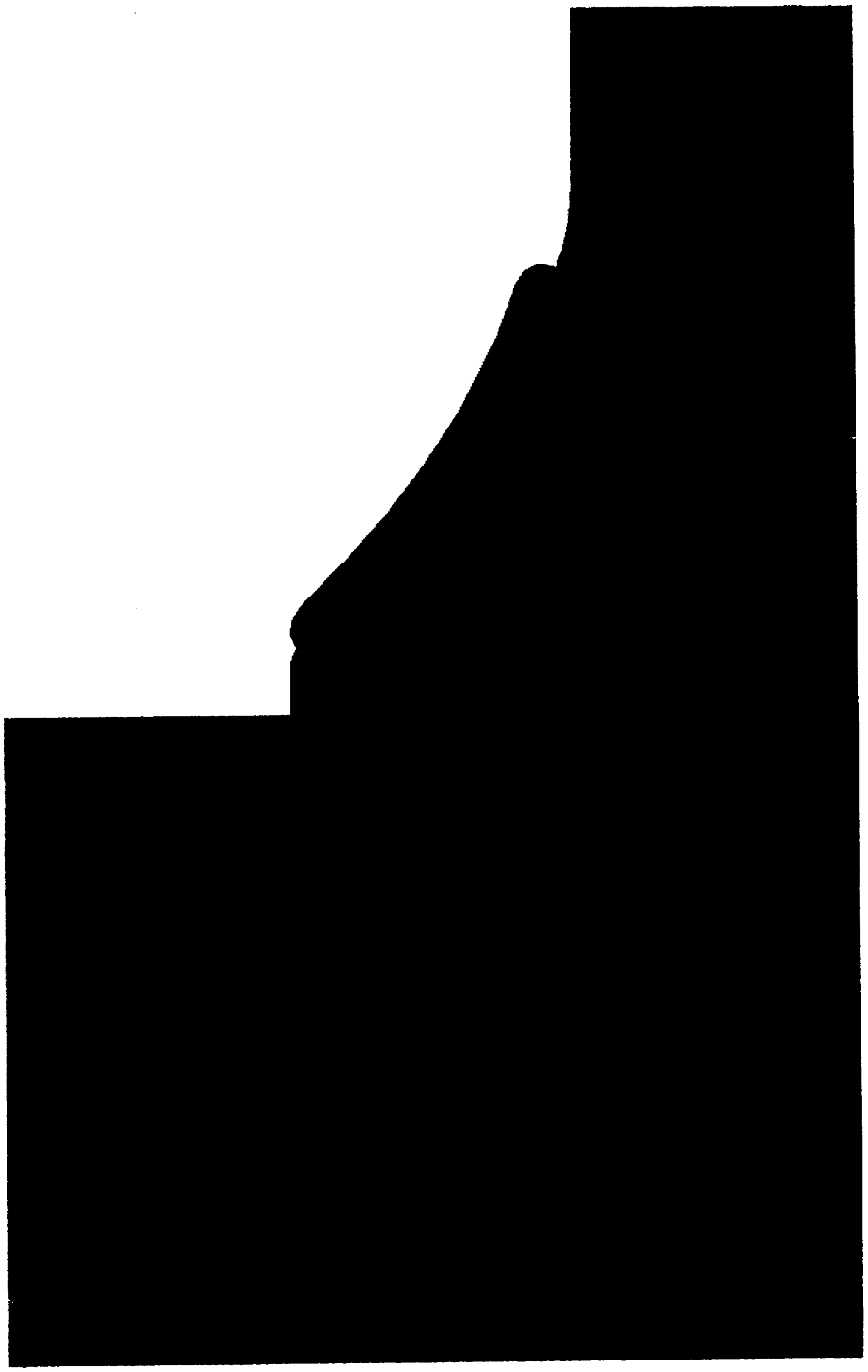
bodies, are then connected by a C-type zone surrounding the upstream body and incorporating part of the boundary of each resultant body. The rest of the flow field is contained by a C-type zone surrounding the final resultant body. A closeup of the grid for the viscous zoning is shown in figure 4.18c. Close inspection of the grid lines at the zonal interfaces near the tuple points reveals some discrepancy in interface location. The representation of these zonal interfaces in EZGrid ensures that the shared curves of each interface are, in fact, the same curve. It is only when that curve is sampled (discretized) separately for each zone (for the purpose of zonal grid generation) that this discrepancy in resolution manifests itself.

The last five configurations represent the class of multiple-grouping configurations. The first is a rotor-stator, where the rotor is separated from the stator by a moving boundary. The location and shape of the moving boundary must be specified by the user during the input phase of an EZGrid run, just after OB has been constructed and before the qualitative configuration description has been provided. The regions on either side of the moving internal boundary (here, simply a straight line halfway between the rotor and the stator) belong to separate objects, each directly contained by OB. In such a situation, the only sensible way to describe the configuration is as one consisting of two separate groupings, where each grouping is composed of one object (the rotor or stator) and is contained by a different parent object. A subplan identical to the one used to generate the single rotor zoning of figure 4.11a (for a viscous computation and a user bias profile that includes a *low importance* smoothness setting) was selected twice, once for each grouping. Because the groupings are contained by different objects, subplan assembly consisted simply of appending one subplan to the end of the other, resulting in a plan which produced the six-zone zoning of figure 4.19a. The composite zonal grid for this case is shown in figure 4.19b.

The next multiple-grouping case is a pair of rotor blades. These two blades could have been grouped within the same grouping since they are directly contained by the same parent object, thereby putting this case in the single-grouping category. An additional subplan rule specifically tailored for two vertically aligned blades would then be required. The need for one additional rule does not pose a problem. Suppose, however, that there were 15 or 20 rotor blades. Subplan rules must be specific in the number and positions of the bodies involved, so whenever the number of blades changes, another subplan rule is required. The alternative is to group each blade in its own grouping, thus requiring only the subplan rule for a single blade. In this case, since the blades are reasonably far apart, this approach worked well, resulting in the zoning shown in figure 4.20a. The composite grid is shown in figure 4.20b. The subplan for a single rotor blade calls for H-type zones surrounding the resultant body (composed of the original blade and an O-type zone around it). H-type zones can be easily merged, obviating the need for secondary actions in the subplan assembly for this double-blade configuration. Assembly of the two identical subplans consisted of appending the primary actions of one to those of the other, followed by the merged completion actions of both.

Figure 4.21 shows a negative result, included here because of the valuable lesson it contains. The two airfoils of this configuration are the aft airfoil elements of the augmentor wing of figure 4.18. Here they were grouped into two separate object groupings, and a plan was assembled from two identical subplans, each for a single teardrop-shaped airfoil in an inviscid flow computation and using the default user bias profile. Serendipity alone is responsible for the fact that the wake curves did not cross. Since the airfoils were not handled by the same subplan rule, no check was performed on potential wake curve interference. The subplan for a single teardrop-shaped airfoil calls for a C-type zone around the body (rather than an easy-to-merge H-type zone), so a connection-type secondary action was required in the assembly of the plan for this case. The secondary action specified a bridge object to be placed between the two airfoil objects after they had been connected to OB by wake curves (the primary actions of the subplans). The merged completion actions then finished the zoning with a C-type zone around the resultant object. The bridge object becomes far too narrow downstream because of the wake curves' ignorance of each other. The problems with this zoning could have been avoided in either of two ways: (1) the airfoils should never have been grouped separately because of their proximity and potential influence on each other, or (2) the rules for assembling the subplans should have been sophisticated enough to modify the primary actions of the subplans so that potential object interference is taken into account. The final result presented in this chapter is an example of successfully making the subplan assembly rules sophisticated enough to handle groupings that are close together.

The result for the next case is also not entirely successful, thus strengthening the case for careful object grouping by the user and/or a less naive subplan assembly. This configuration consists of a group of three NACA 0012 airfoils in which two are vertically aligned and grouped together and the third is grouped by itself and is located upstream of the vertical pair, and is thus an example of the horizontally aligned groupings category. Two subplan rules were selected by EZGrid for this case: one which governs the zoning for a vertical pair of teardrop-shaped airfoils and one which governs the zoning for a single teardrop-shaped airfoil. Both subplans were intended for an inviscid computation and assumed the default user bias profile. Subplan assembly involved the straightforward modification of the primary actions of the subplan for the upstream grouping such that all connection-based operations involving a parent-child operand were changed to connection operations for a sibling-sibling operand. For example, the connection/bridge – parent-child/(OB B1) action for the upstream airfoil was modified to connection/bridge – sibling-sibling/(B1 $ZE1) so that no attempt would be made to attach the upstream airfoil to OB by passing through the downstream objects. The resulting zoning is shown in figure 4.22. It is a reasonable zoning under the circumstances, but suppose the user had specified smoothness to be of *high importance*? There is no mechanism to incorporate this bias into the zoning, because the subplan for the single airfoil in the upstream grouping is not affected by the smoothness parameter. It would be necessary to incorporate the effect of smoothness in the subplan assembly



(a) Six-zone zoning.

Figure 4.19. — Rotor-stator.

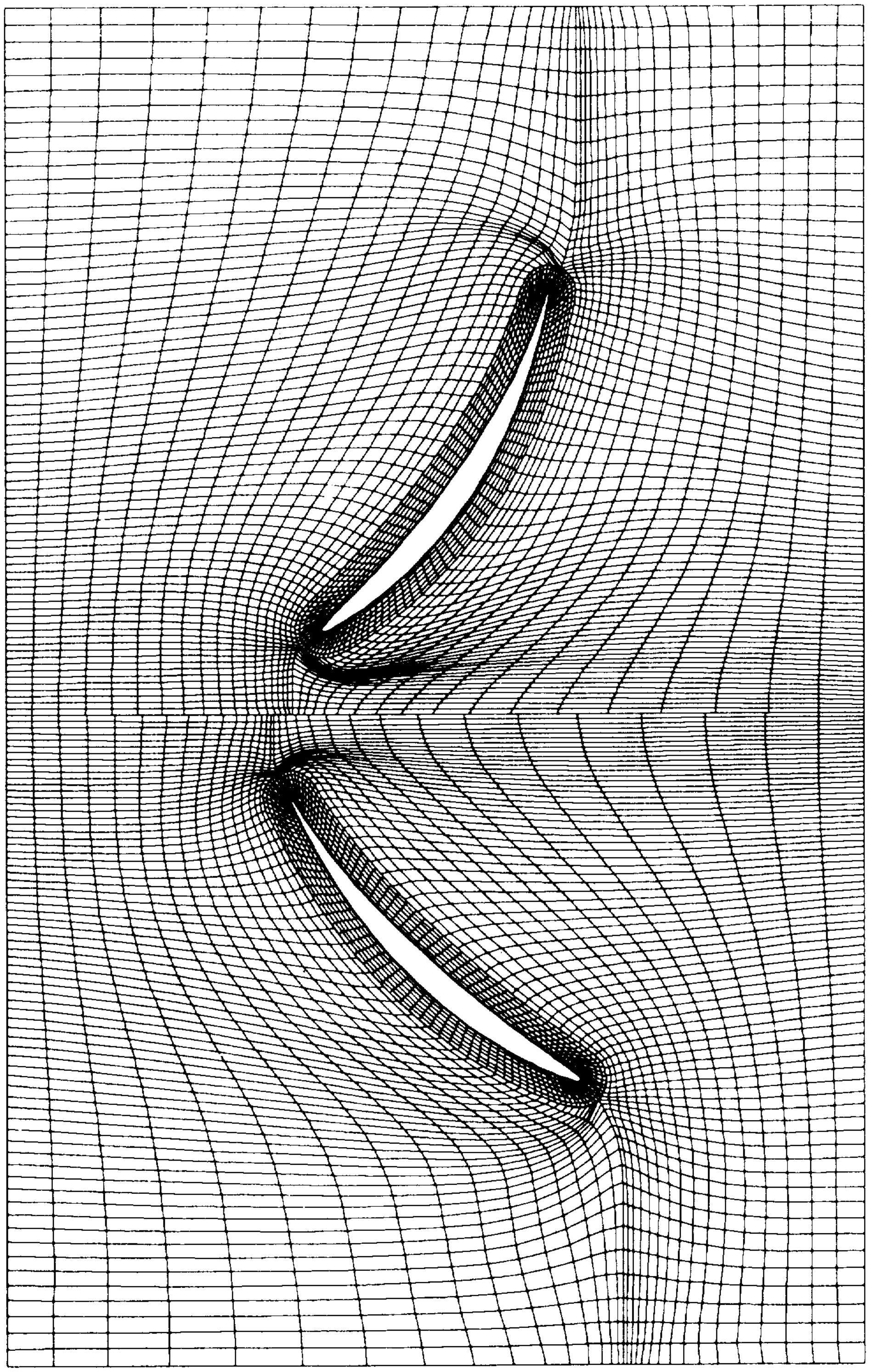
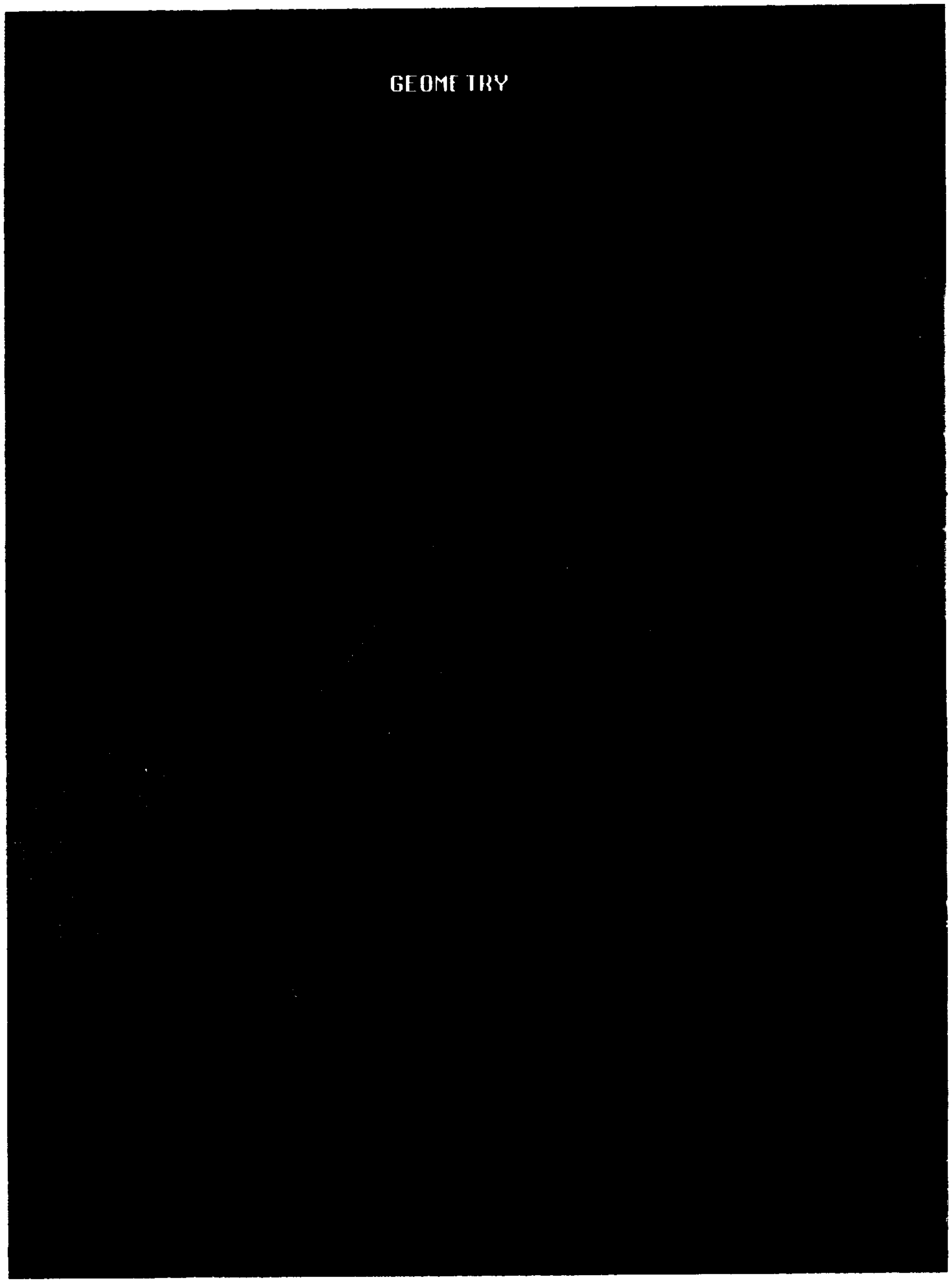

(b) ZONAL GRID

Figure 4.19. — Concluded.



(a) Five-zone zoning.

Figure 4.20. – Rotor pair.

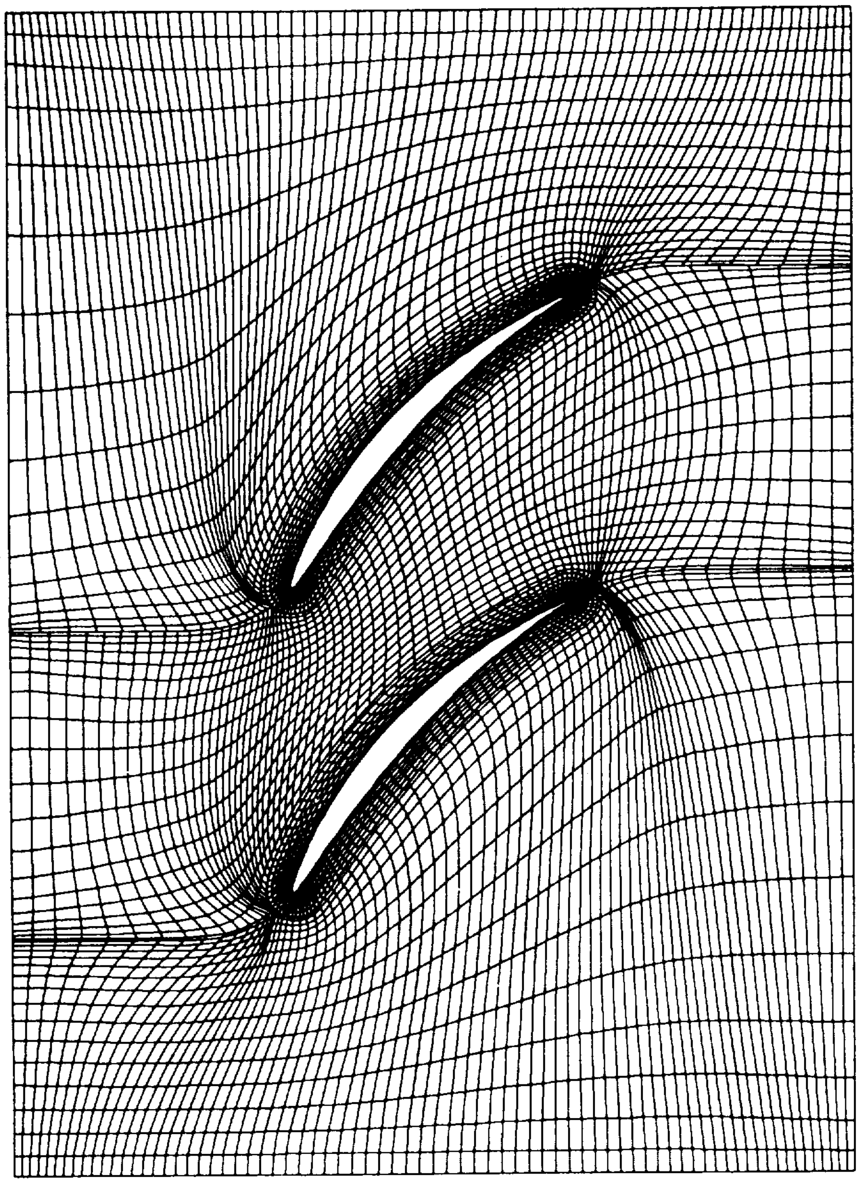

(b) ZONAL GRID

Figure 4.20. – Concluded.

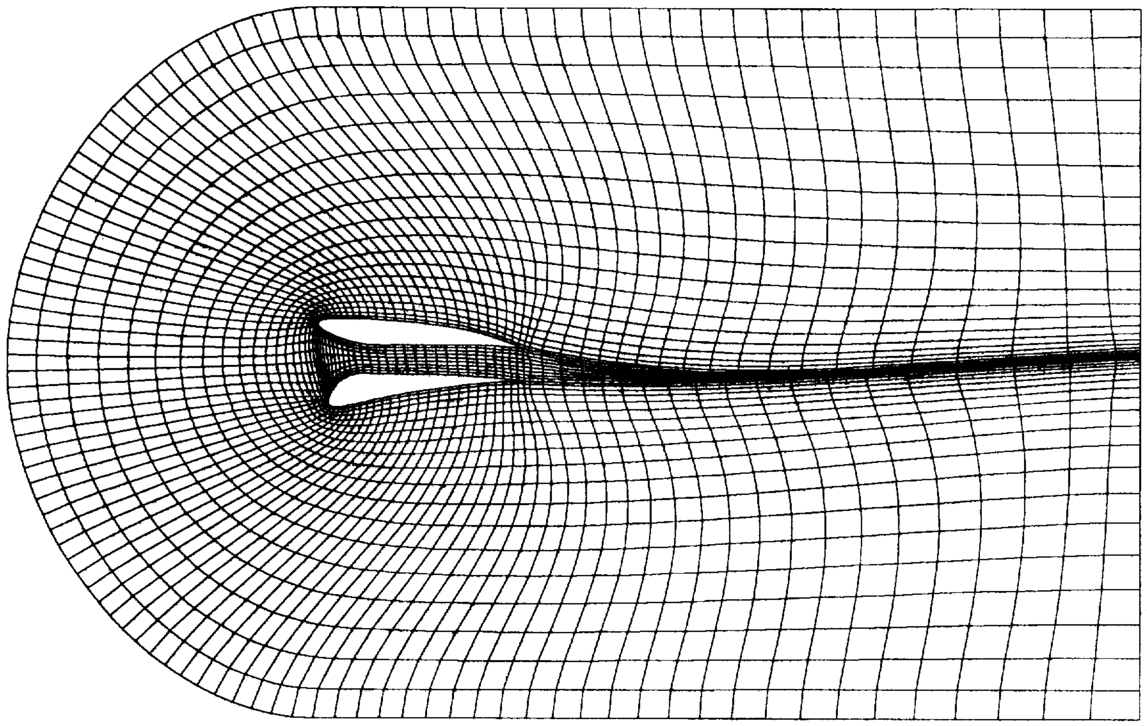


Figure 4.21. – Two-zone grid for Augmentor-wing aft elements grouped separately.

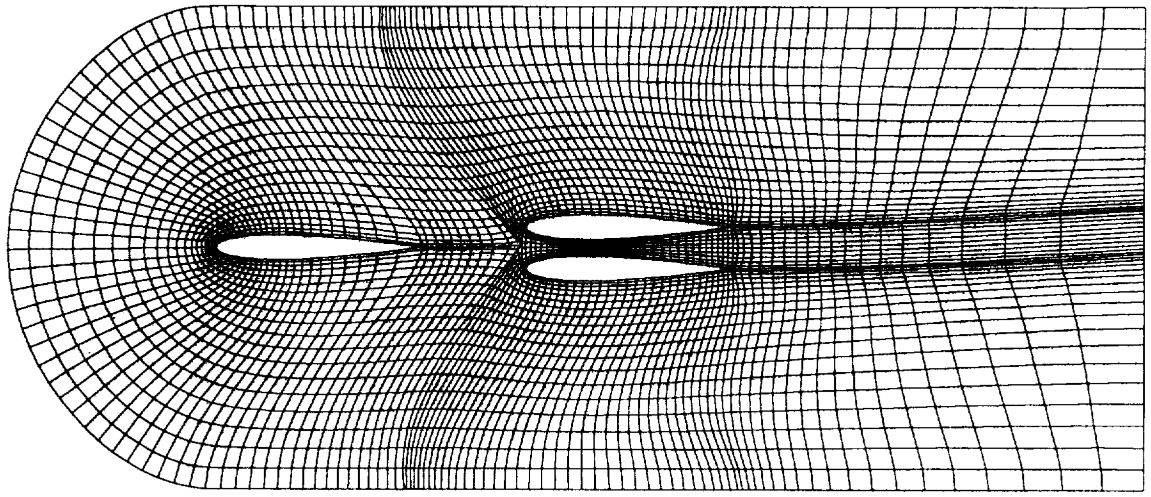
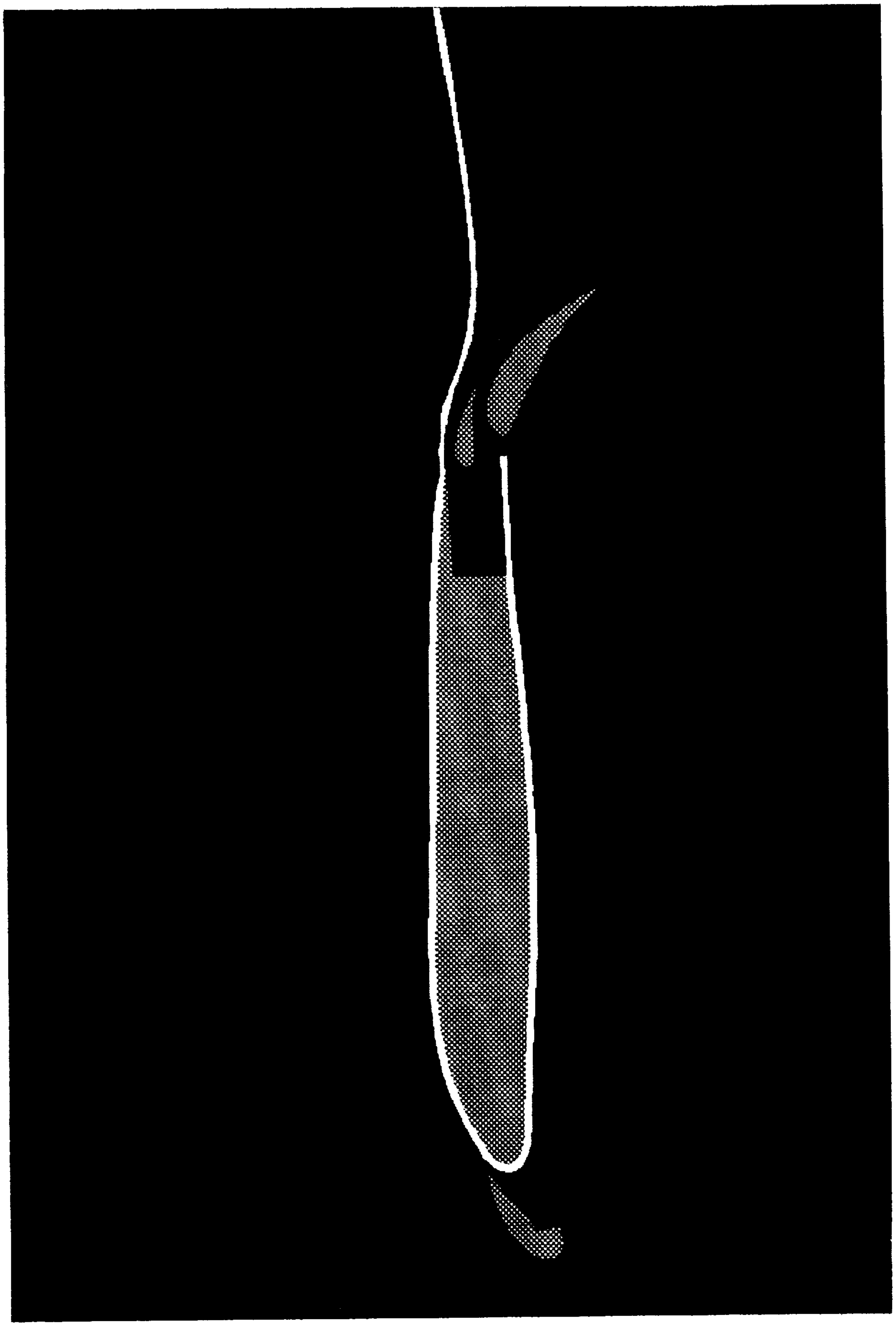


Figure 4.22. – Two-zone grid for three NACA 0012 airfoils in two groupings.

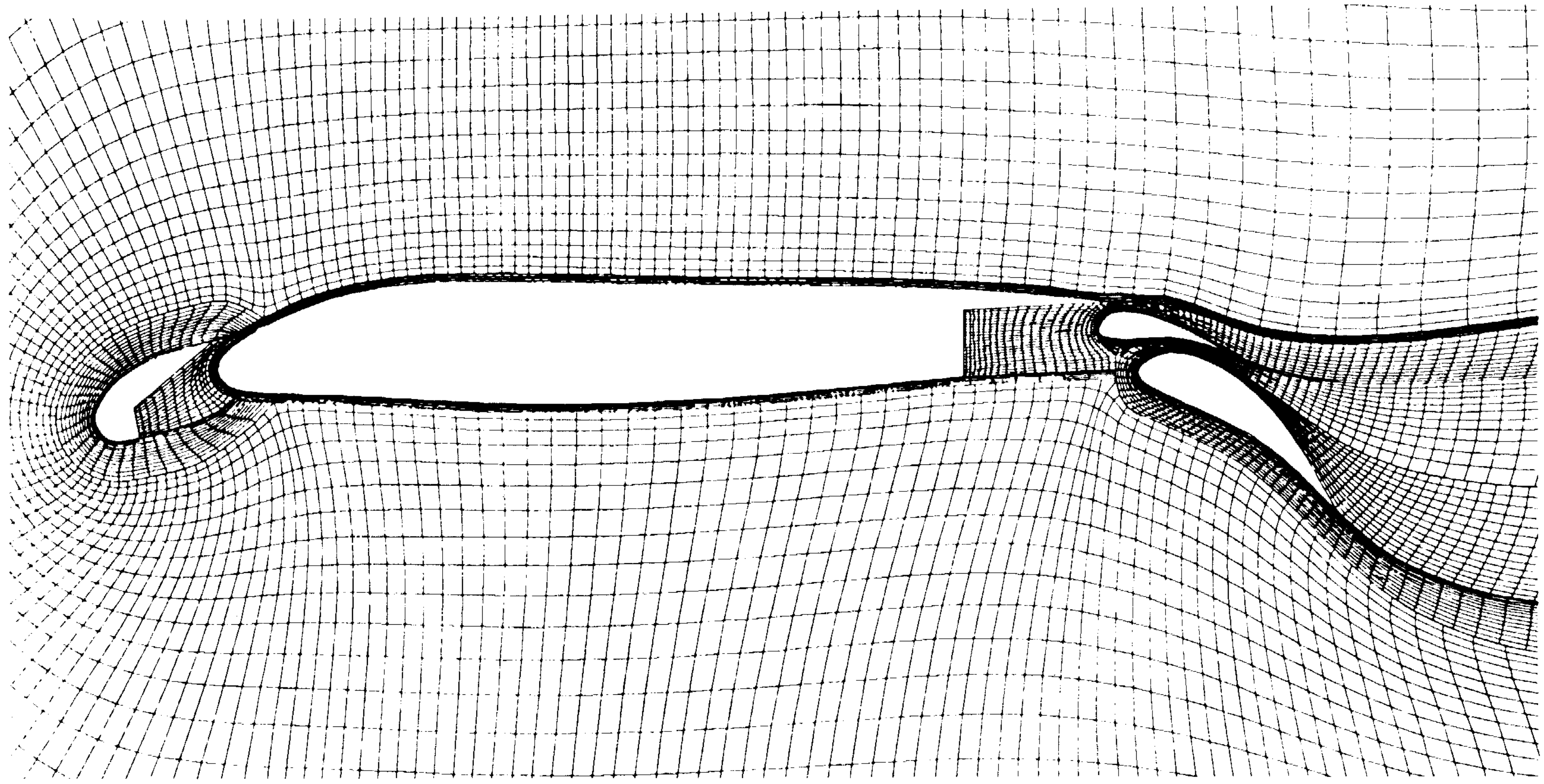
rule, resulting in the placement of a C-type zone around the upstream airfoil with width equal to the width of the downstream resultant body.

The final result presented here is that of a four-element, high-lift airfoil whose elements have been grouped into three separate groupings: one for the upstream-most body, one for the main body, and one for the two smaller aft bodies. The seven-zone zoning which was generated automatically by EZGrid for this rather difficult case is shown in figure 4.23a. Grid details are found in figures 4.23b-d. The groupings are horizontally aligned, so that a modification of the primary actions of the two upstream groupings similar to that described above was required during subplan assembly. As may be seen from the figure, the trailing edge of the main body is extremely close to the two downstream bodies, and should influence the primary actions performed within that grouping. This was accomplished by the inclusion in the original subplan rules of more configuration position and distance information than had previously been used. The result is a successful, complex zoning resulting from the assembly of three simpler zoning subplans, each of which already existed in the knowledge base from earlier test cases.



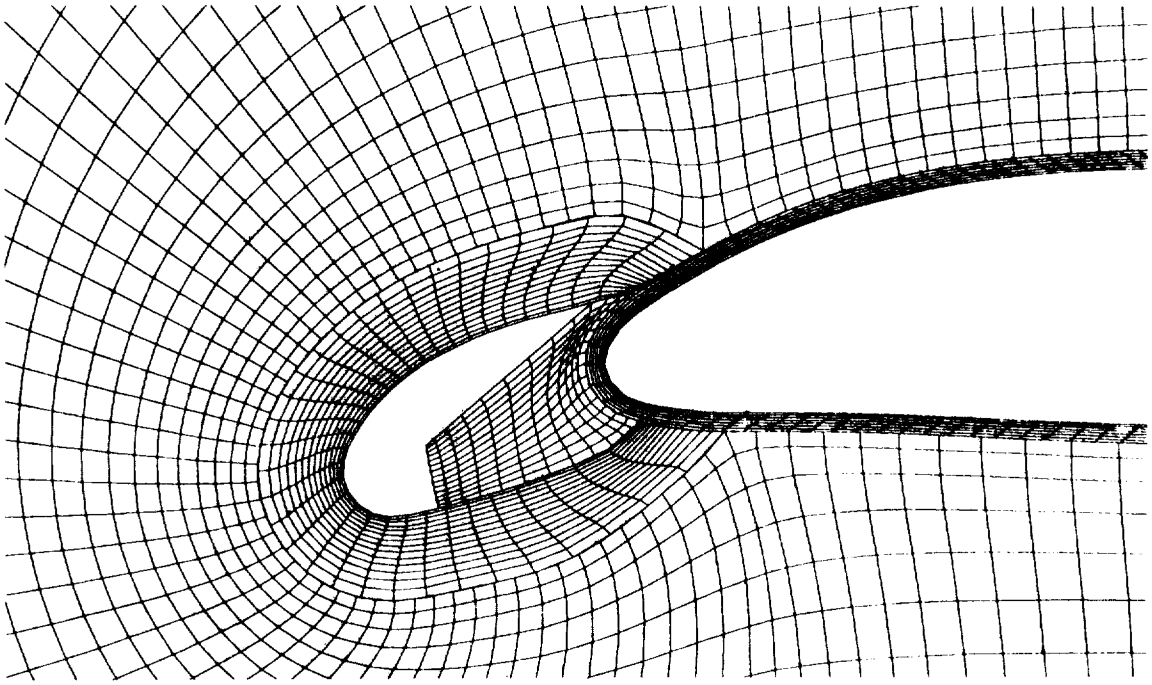
(a) Seven-zone zoning.

Figure 4.23. – Four-element airfoil.

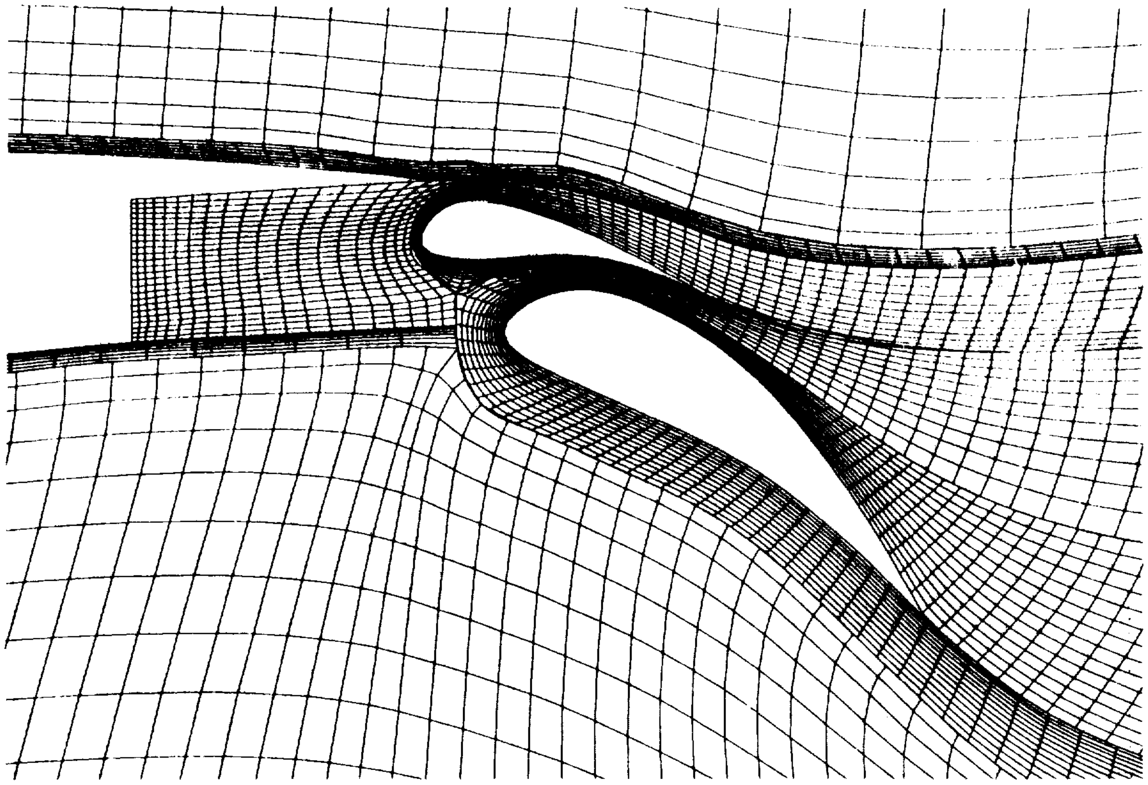


(b) ZONAL GRID

Figure 4.23. – Continued.



(c) UPSTREAM GRID DETAIL



(d) DOWNSTREAM GRID DETAIL

Figure 4.23. – Concluded.

CHAPTER 5

AN APPROACH TO THE EVALUATION OF FLOW-FIELD ZONINGS

5.1 The Challenge of Zoning Evaluation

That EZGrid is capable of designing and generating flow-field zonings for a variety of aerodynamic configurations is evident from the results presented in the previous chapter. What must be established, however, is whether those zonings are good. It was stated at the outset that zoning must be done *well* in order to gain the full benefits of a composite zonal grid approach, so it is important to be able to determine the quality of a zoning. Also, the method by which zoning quality is determined must be automated so that its application to any flow-field zoning, regardless of origin, is consistent. Consistent evaluation is also necessary in order to validate the results of automated flow-field zoning.

The earlier zoning quality can be assessed, the greater the savings in time, effort, and money. Recall that in section 2.3.5 (Encoding Expert Zoning Methodology) the process of flow-field zoning was described in terms of a generate-and-test procedure which produces a tree of all possible intermediate and final zoning situations for a given problem. Because it is impossible to predict early in the process which branch of the tree is most likely to lead to the best zoning, the first practical opportunity for zoning evaluation is at the completion of a candidate zoning. The next opportunity for evaluation is after zonal grids have been generated within each zone, to which existing grid quality measures can be applied. At present, most grid evaluation is done qualitatively and graphically ("by eye"). As stated in section 2.2, Klopfer (1982) and Kerlick (1987) investigated quantitative grid evaluation criteria based on local grid geometry, but found that geometry alone does not provide complete information about a grid; grid quality is also dependent upon the type of equations to be solved on the grid and upon the solution to those equations. Of course, the ultimate validation of a flow-field zoning is through the validation of the solution obtained upon the zonal grids, through comparison with exact solutions, well-established computed solutions, experimental data, and/or flight test data. Flow-field solutions are expensive and time-consuming to generate and interpret, however, and to obtain solutions for a variety of candidate zonings so that they can be evaluated is impractical. Clearly, direct evaluation of a flow-field zoning prior to grid generation or flow solution is the most desirable since it is the earliest stage in the process at which evaluation is possible, and evaluation at later stages would not be sufficiently superior to offset the much greater investment required.

To date, no evaluation criteria have been identified which can be applied directly to a flow-field zoning, so it has been necessary to develop an approach to evaluation as part of the present work. Three factors contribute to the difficulty of zoning evaluation: (1) ideas about the best zoning approaches and the traits of a good zoning are still evolving as experience is gained with composite zonal grid methods; (2) those ideas vary from expert to expert, so no consensus exists; and (3) the requirements for a good zoning are dependent on the particular problem to be solved and on the resources available. One method for dealing with the difficulties arising from these factors might be to establish a standard set of criteria to be applied to all zonings independent of the factors listed above. The drawback to this method is that the standard used could be unacceptable to some experts for some problems, and might never be completely acceptable. Instead, the approach adopted here is to establish a *tunable* standard called the **user bias profile**, which can reflect the ideas and requirements of any user for any problem.

In chapter 2, the user bias profile was defined as a collection of parameters which characterize a user's bias in designing and evaluating flow-field zonings. The 12 parameters chosen to represent user bias were described, and it was explained that tuning the profile consists simply of assigning a qualitative weight to each parameter. In chapter 4 the effect on zoning design of assigning different qualitative weights to the user bias profile was demonstrated. This chapter contains a discussion of how the qualitative weight values are translated into quantitative ones, and how the profile parameters are measured so that a numerical score which takes user bias into account can be computed for any zoning, thus providing the means for consistent comparison and evaluation of flow-field zonings.

5.2 A Tunable Profile of User Bias

The development of the user bias profile—the determination of its defining parameters and their measurement functions, the discretization of possible qualitative weight values, and the translation of those qualitative values into quantitative ones—was accomplished by means of a calibration test involving the participation of five flow-field zoning experts. The basic steps of the test were to

1. Select test case configurations
2. Generate candidate zonings for each test case
3. Ask each expert to tune the user bias profile
4. Ask each expert to rate the candidate zonings for each case
5. Calculate EZGrid ratings for the candidate zonings
6. Compare expert ratings to EZGrid ratings
7. Modify the user bias profile and return to step 3 if necessary

Three simple multiple-body configurations were selected as zoning test cases: a pair of vertically oriented NACA 0012 airfoils, a pair of horizontally oriented NACA 0012 airfoils, and an NACA 0012 airfoil and a cambered airfoil in staggered formation. For each test case, several candidate zoning designs were generated using EZGrid in interactive mode; these are shown in figures 5.1, 5.2, and 5.3. The number of test cases chosen represents a compromise between a smaller number, requiring less time of the zoning experts, but probably resulting in poorer calibration results, and a larger number, requiring more time of the experts (at which some may have balked), but giving better results.

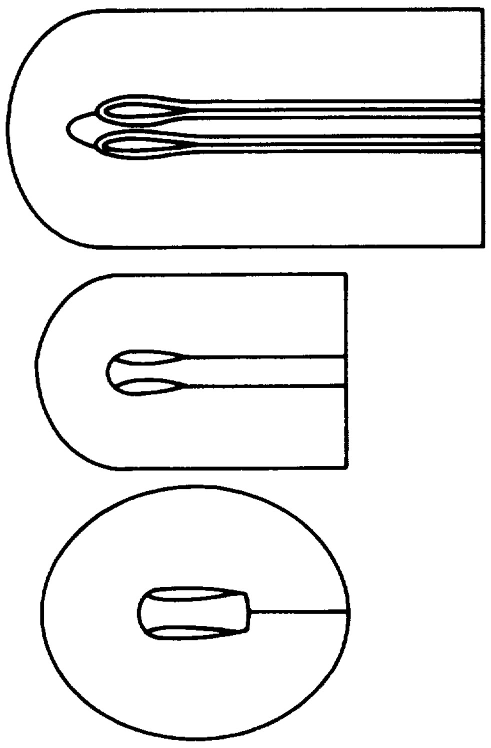


Figure 5.1. – Candidate zonings for test case 1: horizontal configuration.

C-2

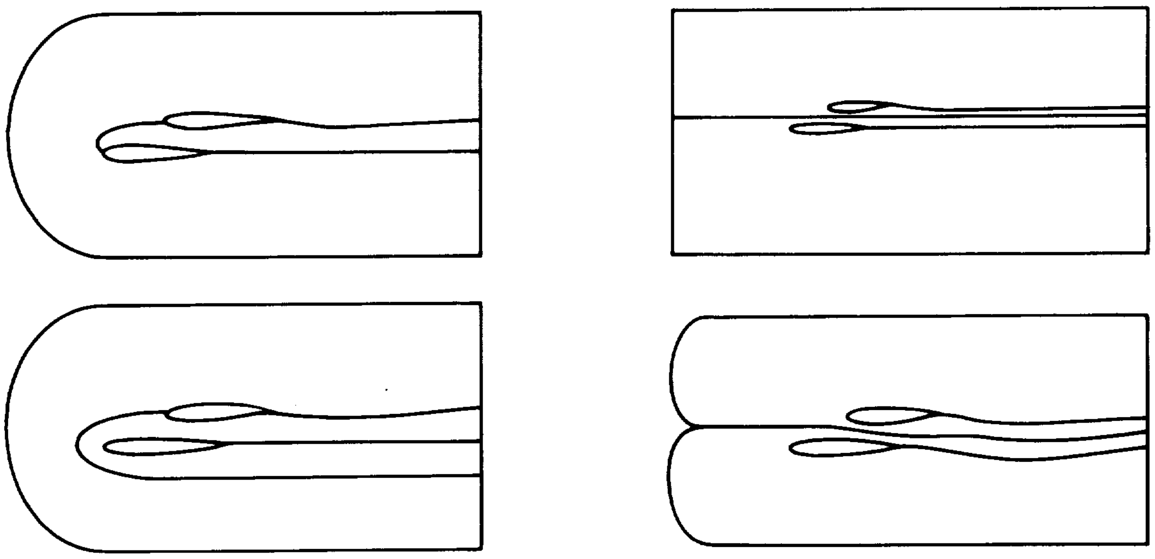


Figure 5.2. – Candidate zonings for test case 2: staggered configuration.

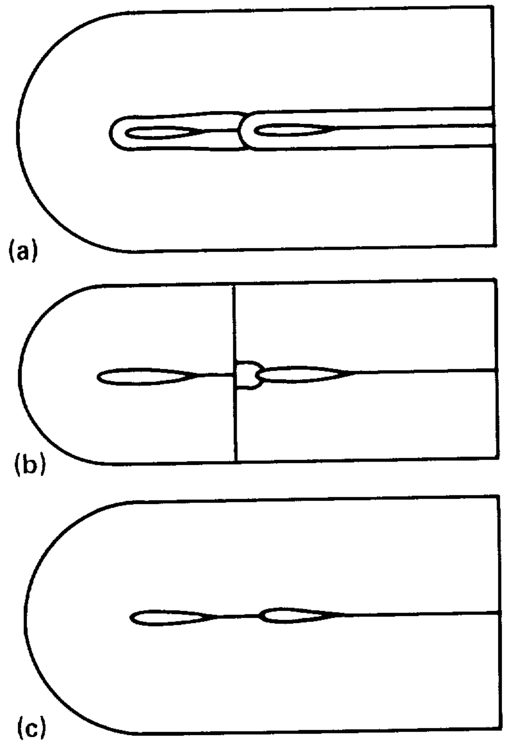


Figure 5.3. – candidate zonings for test case 3: vertical configuration.

The user bias profile described in section 2.3.4 is the product of two iterations of the sequence of steps listed above. For the first iteration, the profile consisted of only three parameters:

1. Zonal simplicity
2. Promotion of good, easy-to-generate grids
3. Accommodation of fluid physics

Each parameter had associated with it three measurement functions to assess different aspects of the zoning. The functions which measure zonal simplicity count the number of zones, the number of zonal interfaces, and the number of zone tuple points (where three or more zones meet). The way in which a zoning promotes good grids was estimated by measuring the skewness of the zone corners, the angles of zone vertices that do not lie on topological corners (a measure of side smoothness), and the disparity between the maximum and minimum distances separating the opposite sides of a zone. Accommodation of fluid physics was roughly predicted by measuring the orthogonality of zone corners belonging to sides shared by body surfaces, looking for singularities at body surfaces (defined as any intersection of a zonal interface curve and a body curve occurring on a single zone side), and determining the number of sides per zone shared by a body surface (to see how many directions of viscous resolution might be required). The experts were asked to assign any real number between 0 and 10 as the parameter weights, where 0 denotes no importance and 10 denotes high importance. The measurement functions were applied to each candidate zoning, and the resulting values were multiplied by the expert-supplied weights and summed to produce a numerical score for each zoning. The measurement functions are cast in terms of penalties for the aspect being measured, so the higher the score, the less desirable the zoning. Comparison of the scores yielded a rating of the candidates based on EZGrid's "preference." The experts' ratings were then compared to EZGrid's ratings.

The results of this first calibration attempt were poor. Very few of the expert ratings were matched by EZGrid. Although the experts were given a theoretically infinite range of possible parameter weights, they all tended to think first in qualitative terms and then translate to some representative number (e.g., "this is fairly important, so I'll give it an 8.5"). By coincidence, the tuned profiles of several of the experts were identical. None of the expert ratings of the candidate zonings were identical, however, and there should have been a stronger correlation between profile and rating. A follow-up interview with the experts whose profiles were identical revealed that they each did have somewhat different zoning criteria, but that the three parameters were not able to capture those differences. Based on the results of this first iteration, the following hypotheses were formed:

1. The user bias profile was parameterized too coarsely.
2. The possible parameter weights should be qualitative.
3. New measurement functions should be added.
4. Some existing measurement functions should be modified.
5. There may be some human inconsistency in the selection of parameter weights to reflect the user's bias.

The second iteration of the calibration test represents an attempt to test the first four of the five hypotheses listed above. The number of profile parameters was expanded from 3 to 12 by breaking up the previous groups of measurement functions so that only one measurement function is associated with each parameter (with one exception), and by adding several new measurement functions. The range of possible weight values was restricted and only qualitative values were allowed. A list of the 12 parameters and their possible qualitative weights is found in table 5.1. Two existing measurement functions were modified to better measure the associated parameter.

Zone simplicity is the only parameter which has two measurement functions associated with it; those functions measure the number of zones and zonal interfaces, just as before. The existence and number of tuple points was identified as a separate parameter, with possible weights of *allowed, discouraged*, etc. Skewness, smoothness, mapping disparity, and orthogonality are new parameters that are based on existing measurement functions. The measurement function for orthogonality was modified to include the influence of noncorner vertices as well as corner ones. Efficiency and surface-vs-field quantities are new parameters which are both based on a variant of the same new measurement function. This new function computes the ratio of the combined length of all body curves to the total combined length of both body and nonbody curves for those zone sides that are shared by body surfaces, and subtracts it from 1.0. Wake resolution, a new parameter, is measured by identifying the number of trailing-edge points of each body in the configuration (usually one, as in a teardrop-shaped airfoil, or two, as in an airfoil with a base trailing edge), and subtracting from that the number of wake curves in the zoning. This function penalizes trailing-edge points that do not have wake curves emanating from them. The parameter for singularities is new, and is based on a modified version of the old measurement function which does not classify the intersection of a wake curve with a body as a singularity, as was erroneously done in the first iteration. The final additional profile parameter involves intersections between nonbody zone sides and body surfaces. It is measured simply by searching for such intersections and summing their occurrences. The equations on which each parameter measurement function is based are listed in appendix D.

In the second iteration, the five experts were asked to tune the user bias profile using the new parameters and qualitative weights. Further refinement of the possible qualitative weights was necessary to capture each expert's criteria satisfactorily (at the beginning of the second iteration, the weight choices for the last four parameters consisted only of *allowed* and *not allowed*, which was insufficient for the rather fine distinctions made by some of the experts). They did not change their original ratings of the candidate zonings. This time, no two profiles were the same, as can be seen in table 5.2. To calibrate the profile, two types of values were adjusted in order to maximize the number of matches between human expert and EZGrid ratings: relative function weights (factors which multiply several of the raw measurement function values so that each function has the potential to contribute the same order-of-magnitude value to the overall score for a zoning), and the numerical translations of the user-supplied qualitative weights. The final quantitative values associated with the qualitative weights are shown in the rightmost column of table 5.1.

The results of this iteration are displayed in table 5.3. The three test cases appear on the far left, with their associated candidate zonings represented by the letters in the adjacent column. The five columns in the box at the right are the ratings of the candidate zonings, where the number on the left of each "/" represents the human expert's preference, and the number on the right represents EZGrid's preference for that case. For example, both Expert 1 and EZGrid rated candidate zoning C as the best zoning among the choices for the first case. These results are

Table 5.1 A Tunable User Bias Profile

Profile Parameter	Possible Weight Values	
Simplicity		
Zone corner skewness	*NO*	*0*
Zone side smoothness	*LOW*	*1*
Zone side mapping disparity	*MEDIUM*	*5*
Grid point efficiency	*HIGH*	*10*
Orthogonality at body surfaces	*(IMPORTANCE)*	
Surface vs. field quantities		
Wake resolution		
Zone tuple points	*ALLOWED BUT NOT IMPORTANT*	*1*
Singularities at body surfaces	*ALLOWED*	*5*
Zone / body intersections	*SOMEWHAT DISCOURAGED*	*20*
Viscosity in more than one direction	*DISCOURAGED*	*40*
	STRONGLY DISCOURAGED	*60*
	NOT ALLOWED	*100*

encouraging since EZGrid was able to match most of the ratings given by the human experts. With two exceptions, it chose the same "best" candidate zoning for each case as the experts, and in several instances, EZGrid matched an expert's rating of all the candidates of a case. It would be misleading to state that these results are significant in any statistical sense; it is reasonable to claim, however, that user bias does have a measurable effect on flow-field zoning design, and that the proposed user bias profile, with its associated measurement functions, is a promising method of evaluating zoning results in the absence of universally accepted criteria.

5.3 EZGrid Performance Assessment

Validation of a knowledge-based system is often based on a comparison of system performance and human expert performance. Calibration of the user bias profile has provided the means of comparing EZGrid performance to that of flow-field zoning experts. In order to apply the user bias profile evaluation functions, it is necessary that the expert's zoning design be represented in the same manner as an EZGrid design. This requirement is easily satisfied by using EZGrid in interactive mode to input the expert's design. As an initial check on the adequacy of EZGrid's zoning design knowledge base, the five different expert user bias profiles used in the profile calibration described above were read into EZGrid, and zonings were generated automatically for all three calibration test cases. EZGrid successfully reproduced the same zoning design as was chosen by each expert to be the best candidate.

Table 5.2 User Bias Profiles for Five Zoning Experts.

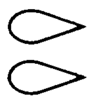
Parameter	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5
Simplicity	M	M	M	M	M
Skewness	M	L	L	M	M
Smoothness	M	M	M	M	M
Mapping disparity	H	M	H	H	M
Efficiency	H	L	H	H	L
Orthogonality	H	M	H	H	H
Surface vs. Field	M	M	H	H	M
Wake	H	M	M	M	M
Tuple Points	5	3	4	2	3
Singularities	2	4	2	2	4
Intersections	2	2	5	1	3
Viscosity 2	2	2	2	2	2

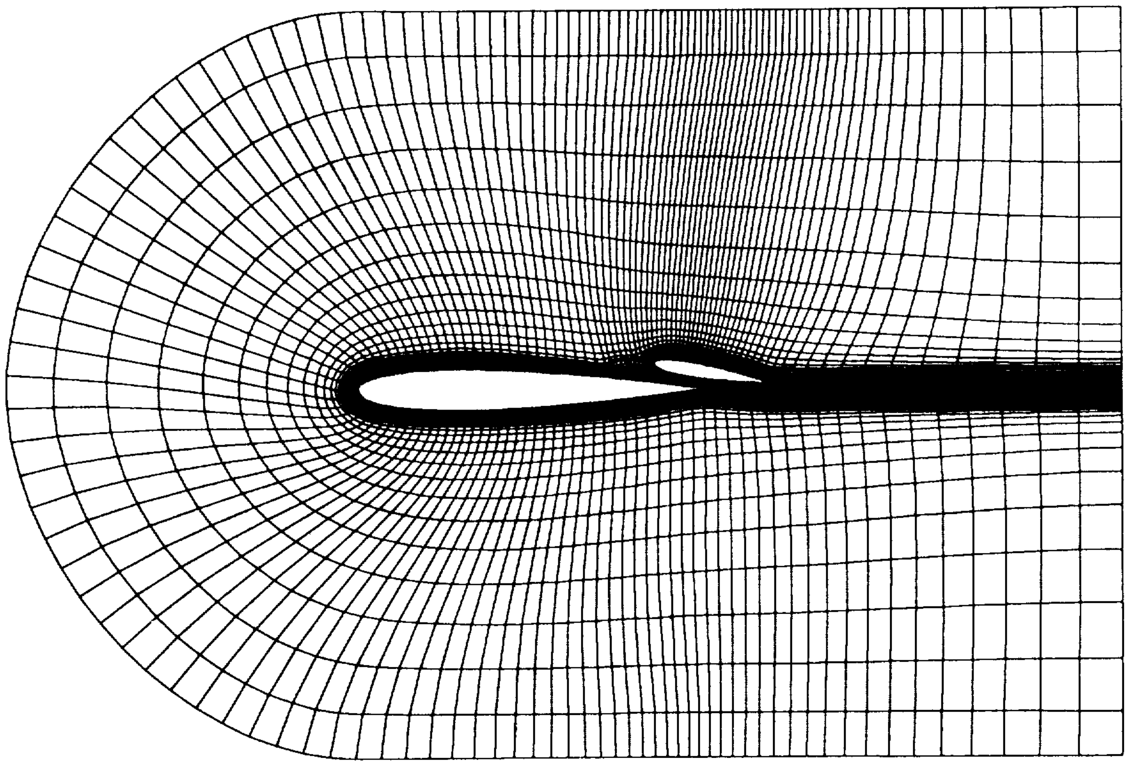
LEGEND			
	N = no importance	1 = allowed but not important	5 = strongly discouraged
	L = low importance	2 = allowed	6 = not allowed
	M = medium import.	3 = somewhat discouraged	
	H = high importance	4 = discouraged	

The next study focused on comparing the performance of EZGrid to just one zoning expert, Expert 2 of the calibration test. Instead of presenting the expert with a selection of already-generated zoning candidates to rate in order of preference, the expert was given two problem statements and asked to tune the profile and design a zoning for each problem. One configuration consisted of an airfoil with a flap above and downstream, in which the flap tilted downward. The second configuration was identical except that the flap was angled upward, away from the airfoil. The problem statements and the expert's responses are reproduced in Appendix E. Zonings and grids based on the design sketches returned by the expert were generated using EZGrid in interactive mode, and are shown in figures 5.4a-b and 5.5a-b. The outer boundaries for both cases were specified as bullet-shaped and 10 chordlengths away in all directions. The zonings generated using EZGrid kept the suggested outer boundary shape, but brought the boundaries in to only one chordlength away in all directions, simply for ease of presentation. EZGrid was given the same two problems and was furnished with the user bias profile as tuned by the expert for each problem. The zonings which were generated automatically (as before, the grids were generated interactively) are shown in figures 5.4c-d and 5.5c-d. EZGrid suggested bullet shapes for the two outer boundaries, and suggested an upstream and downstream boundary distance of 6 chords and an upper and lower boundary distance of 18 chords. The distance suggestions were ignored and an outer boundary one chord away from the body in all directions was generated for both cases, as in the interactive zonings. Each of the zonings was evaluated by EZGrid using the profiles given and the profile parameter measurement functions. In both cases, EZGrid rated its own zonings as superior to those of the expert.

The zonings in figures 5.4a-b and 5.4c-d appear comparable. The primary differences are that the EZGrid zoning is simpler (i.e., has fewer zones and zonal interfaces), has one fewer tuple points, and has C-type zones around each body. These aspects of the zoning are intrinsically neither advantageous nor disadvantageous—what matters is how they are viewed by the user (here an expert) as reflected in the user bias profile. For this case, the expert had set the simplicity parameter to *high importance* and the tuples parameter to *discouraged*. The larger penalties on

Table 5.3 Profile Calibration Test Results.

GEOMETRY	CANDIDATE ZONING	CANDIDATE ORDERING BY PREFERENCE (HUMAN EXPERT/EZGRID)				
		EXPERT 1	EXPERT 2	EXPERT 3	EXPERT 4	EXPERT 5
HORIZONTAL NACA0012 PAIR	A	2/2	1/1	2/2	1/1	1/1
	B	3/3	2/3	3/3	2/3	3/3
	C	1/1	3/2	1/1	3/2	2/2
STAGGERED NACA0012 PAIR	A	3/3	4/4	4/4	2/2	4/4
	B	1/1	3/3	1/1	1/1	1/3
	C	4/4	2/2	3/3	4/4	3/2
	D	2/2	1/1	2/2	3/3	2/1
VERTICAL NACA0012 PAIR 	A	2/2	1/1	1/1	3/2	1/1
	B	1/1	2/2	2/2	2/1	2/2
	C	3/3	3/3	3/3	1/3	3/3



(a) 4-ZONE – EXPERT

Figure 5.4. – Case 1 results.

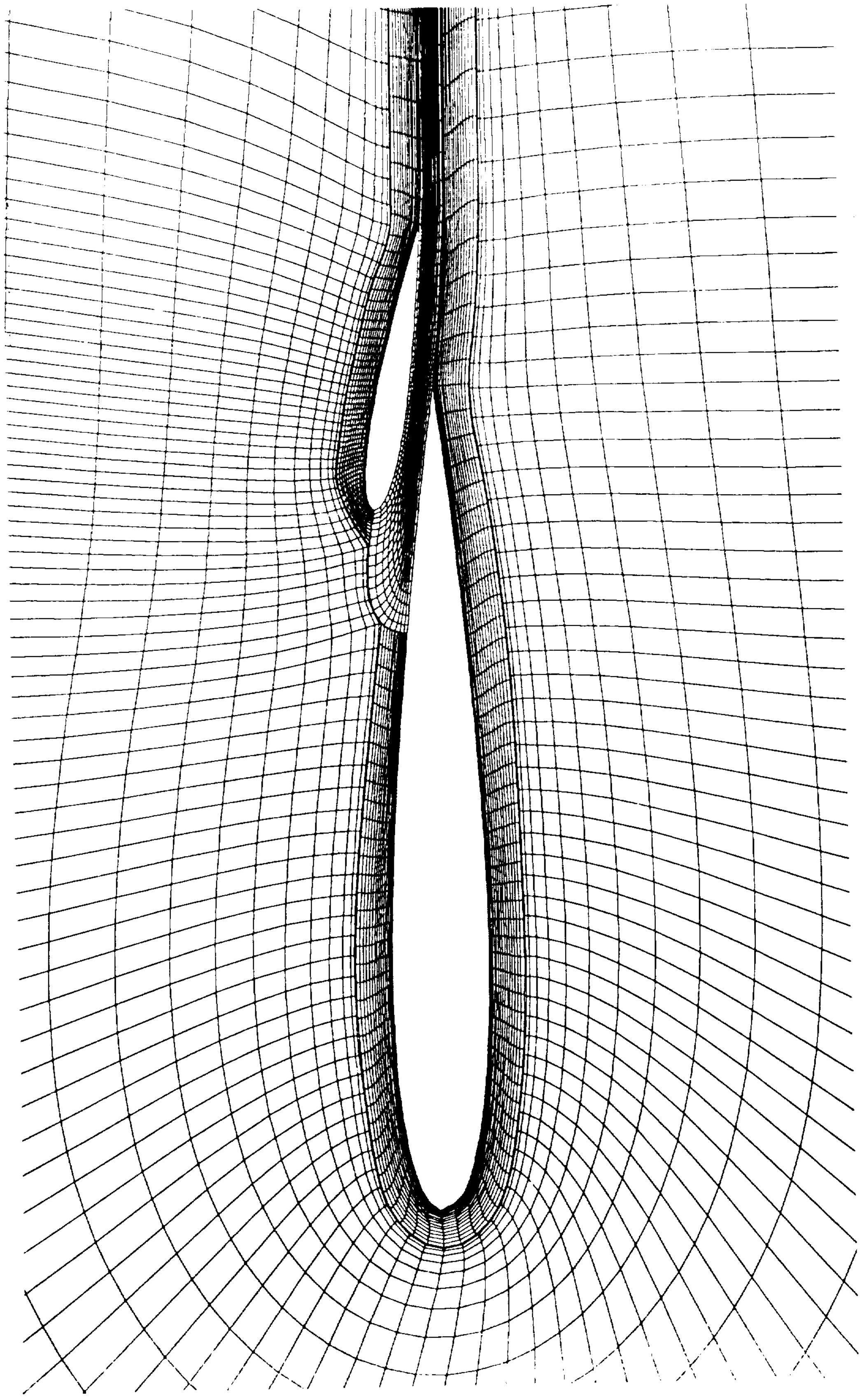

(b) GRID DETAIL – EXPERT

Figure 5.4. – Continued.

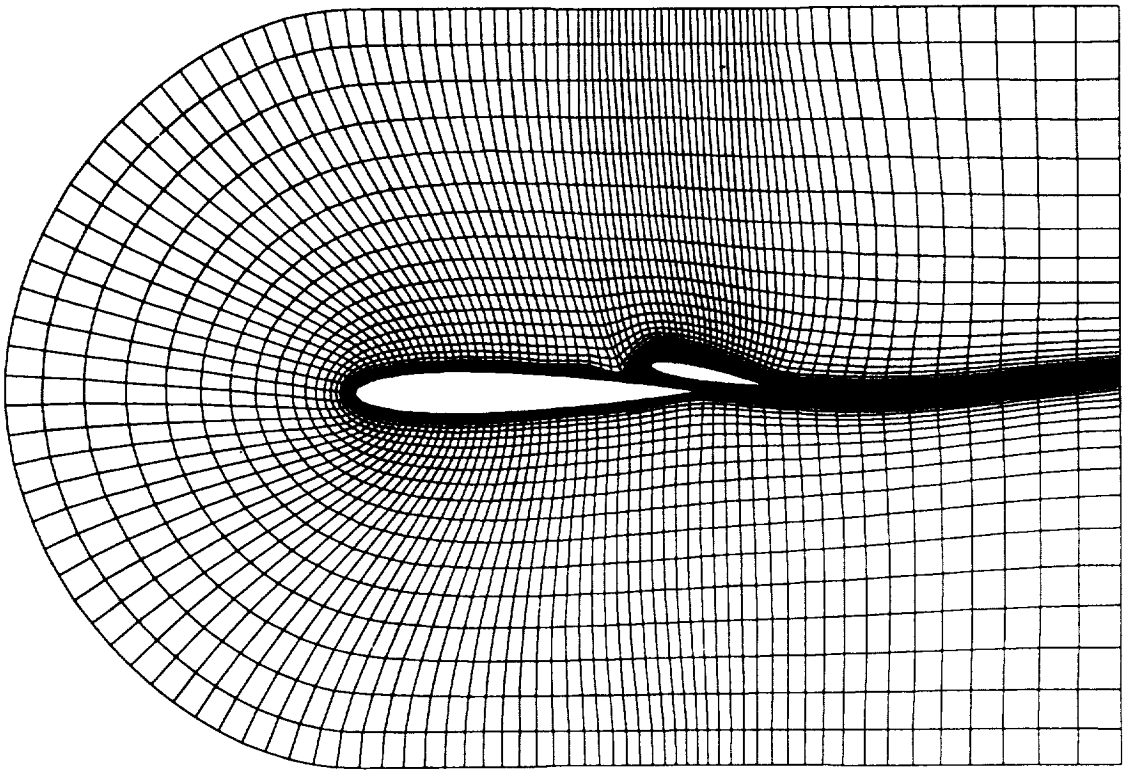
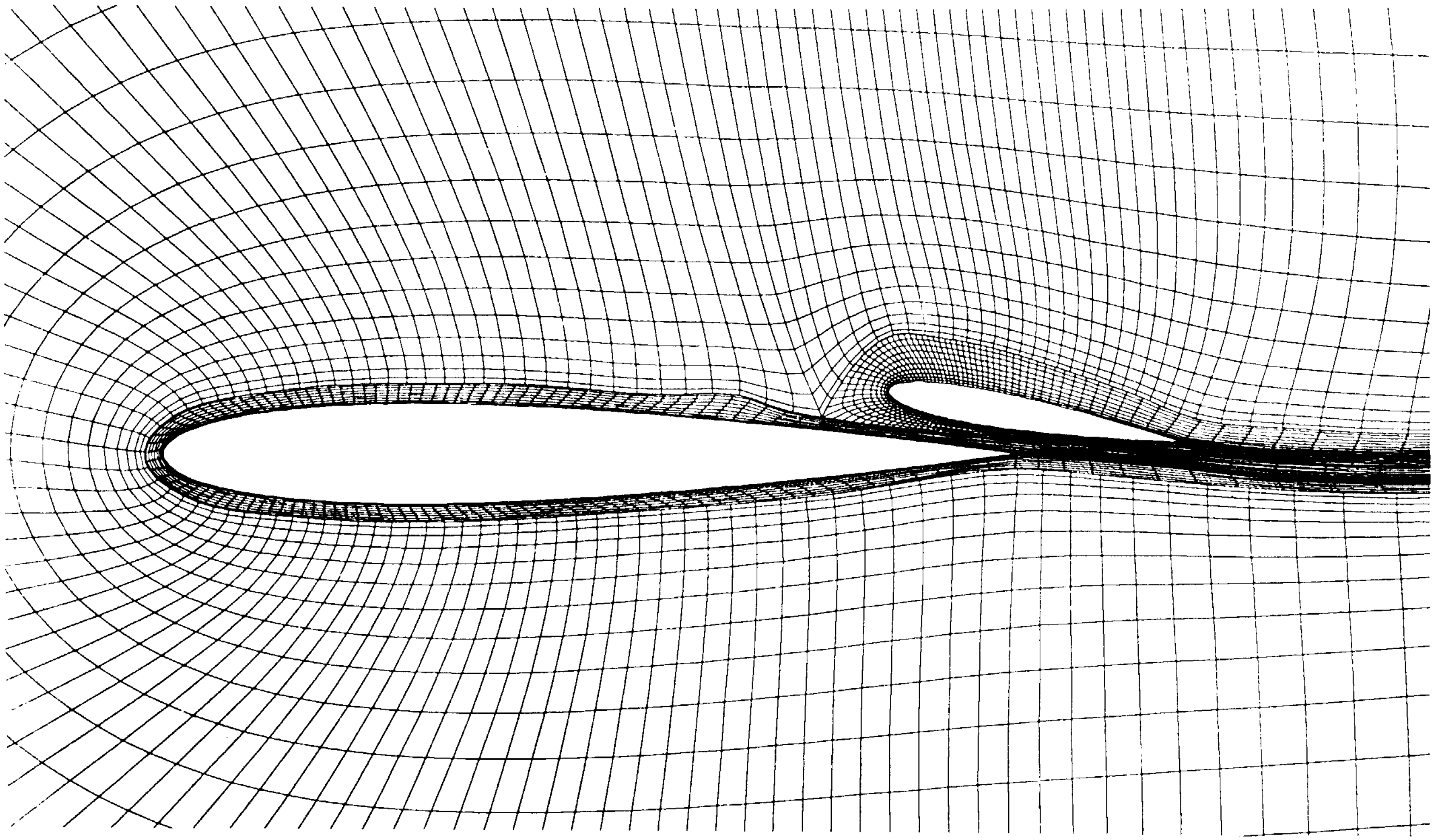

(c) 3-ZONE GRID – EZGRID

Figure 5.4. – Continued.

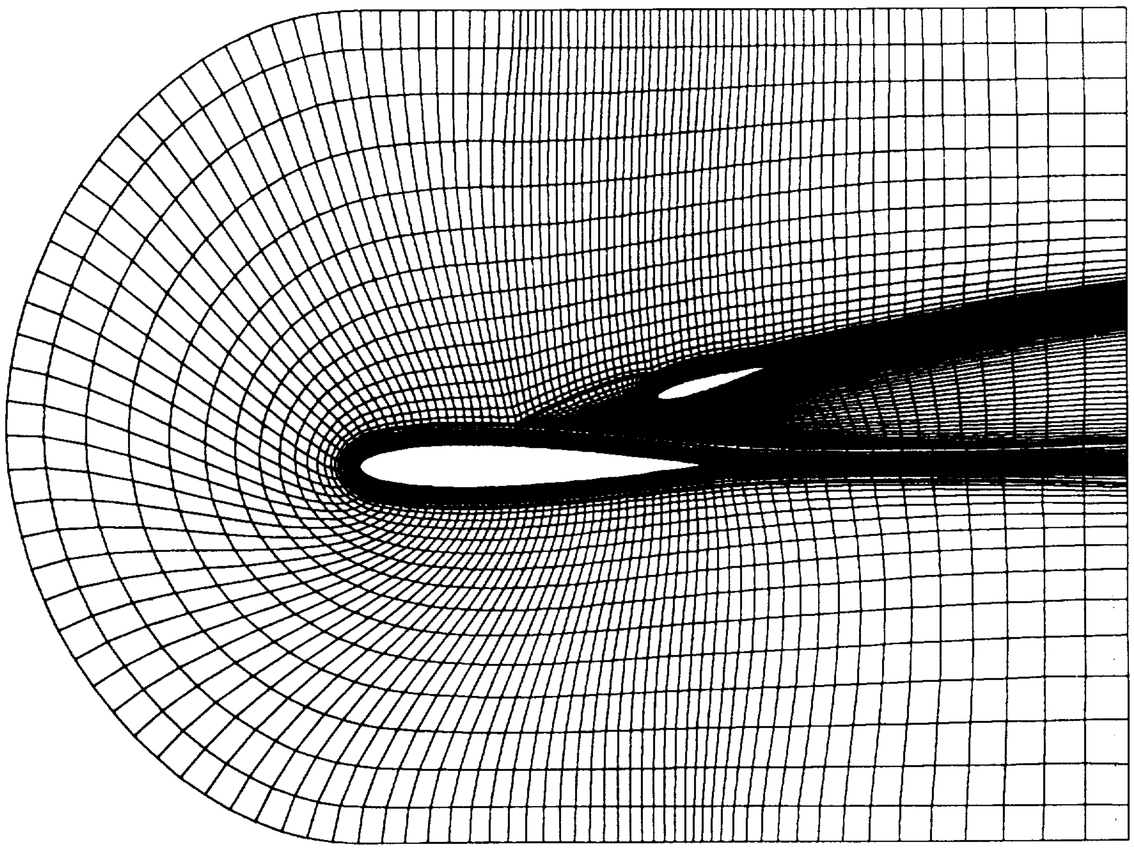


(d) GRID DETAIL – EZGRID

Figure 5.4. – Concluded.

the expert's zoning for simplicity and tuple points were multiplied by the large numerical weights corresponding to those two qualitative values, which was responsible for its poorer score. When asked why he did not put a C-type zone around the downstream body, the expert replied that he did not wish to divide the narrow region between the two bodies with a zonal interface because it might interfere with the determination of the length scale used in the turbulence model. The potential difficulty caused by the resulting zone/body intersection at the nose of the downstream body was disregarded by the expert (which is consistent with his profile parameter setting for that parameter) because he planned to generate the grids in the two downstream body zones so that grid lines would be continuous across their common interface.

The EZGrid zoning of figures 5.5c-d looks inferior to that of the expert in figures 5.5a-b, despite the fact that EZGrid's evaluation rated its own zoning higher. The explanation for this apparently incorrect evaluation lies again in the user bias profile. The profile used in this case also included a ***high importance*** setting for simplicity and a ***discouraged*** setting for tuple points. This emphasis on simplicity and discouragement of tuples again tipped the scales in favor of the EZGrid zoning since it has only two zones (to the expert's four) and no tuple points (to the expert's two). The grids generated in the EZGrid zoning are poor, particularly in the upper C-type zone around the flap. There is a great deal of skewness, and the number of grid cells in the lower grid along the upstream portion of the interface corresponding to a single grid cell in the upper grid is too large for flow-solution stability. Why did the profile parameters that are intended to predict grid goodness (e.g., smoothness, skewness, and mapping disparity) not penalize the EZGrid zoning? The grid line skewness evident in the interior of the upper grid is due not to skewness or discontinuities at zone vertices (either corner or noncorner vertices), as



(a) 4-ZONE GRID – EXPERT

Figure 5.5. – Case 2 results.

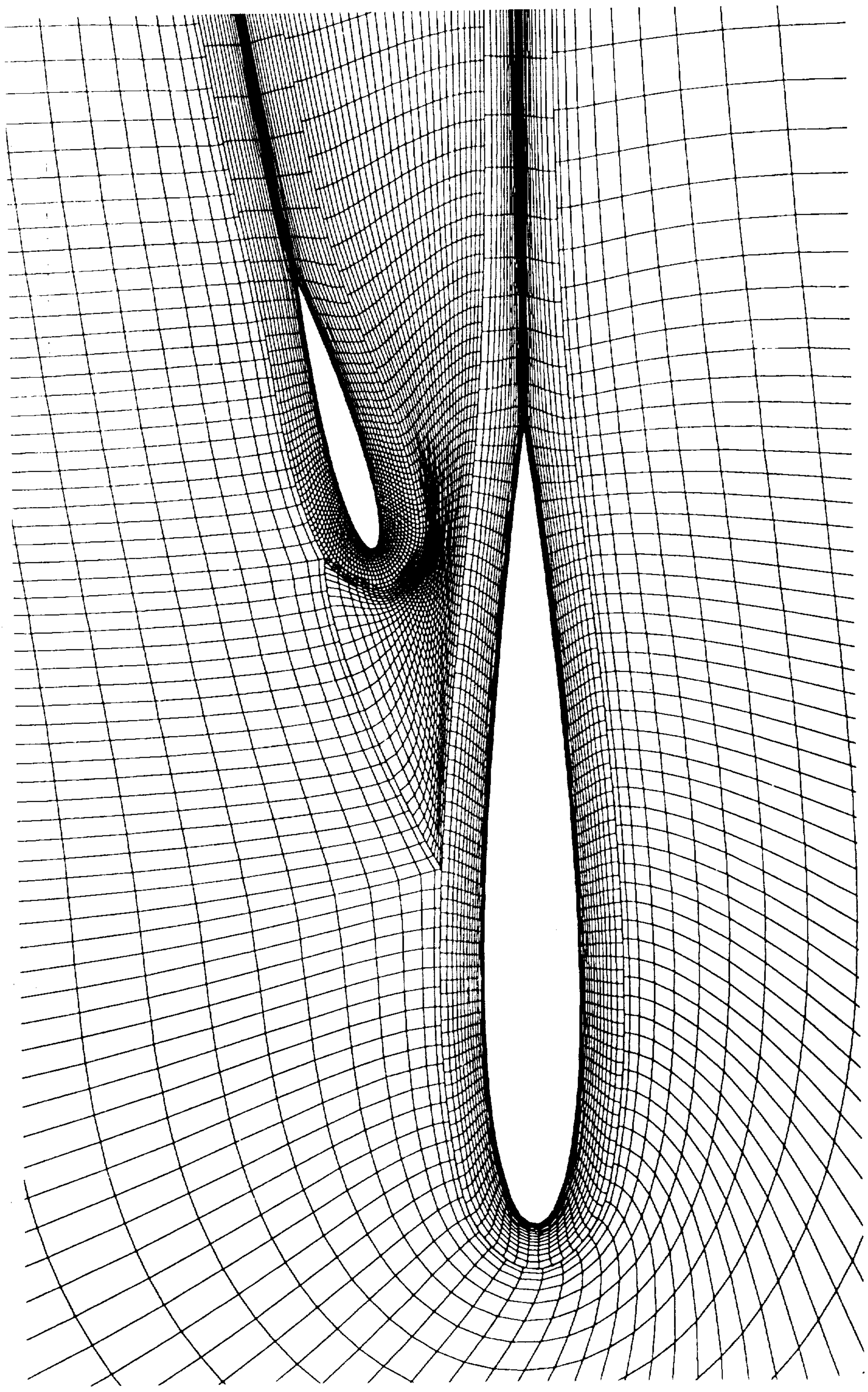
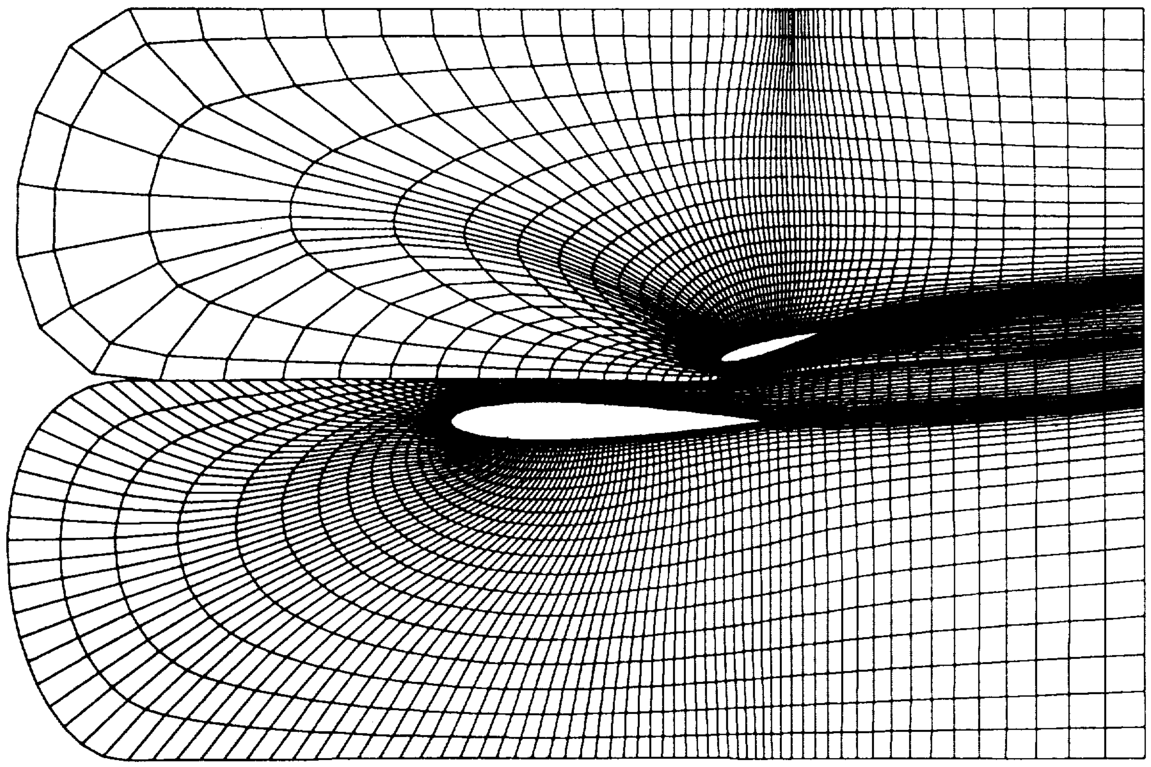
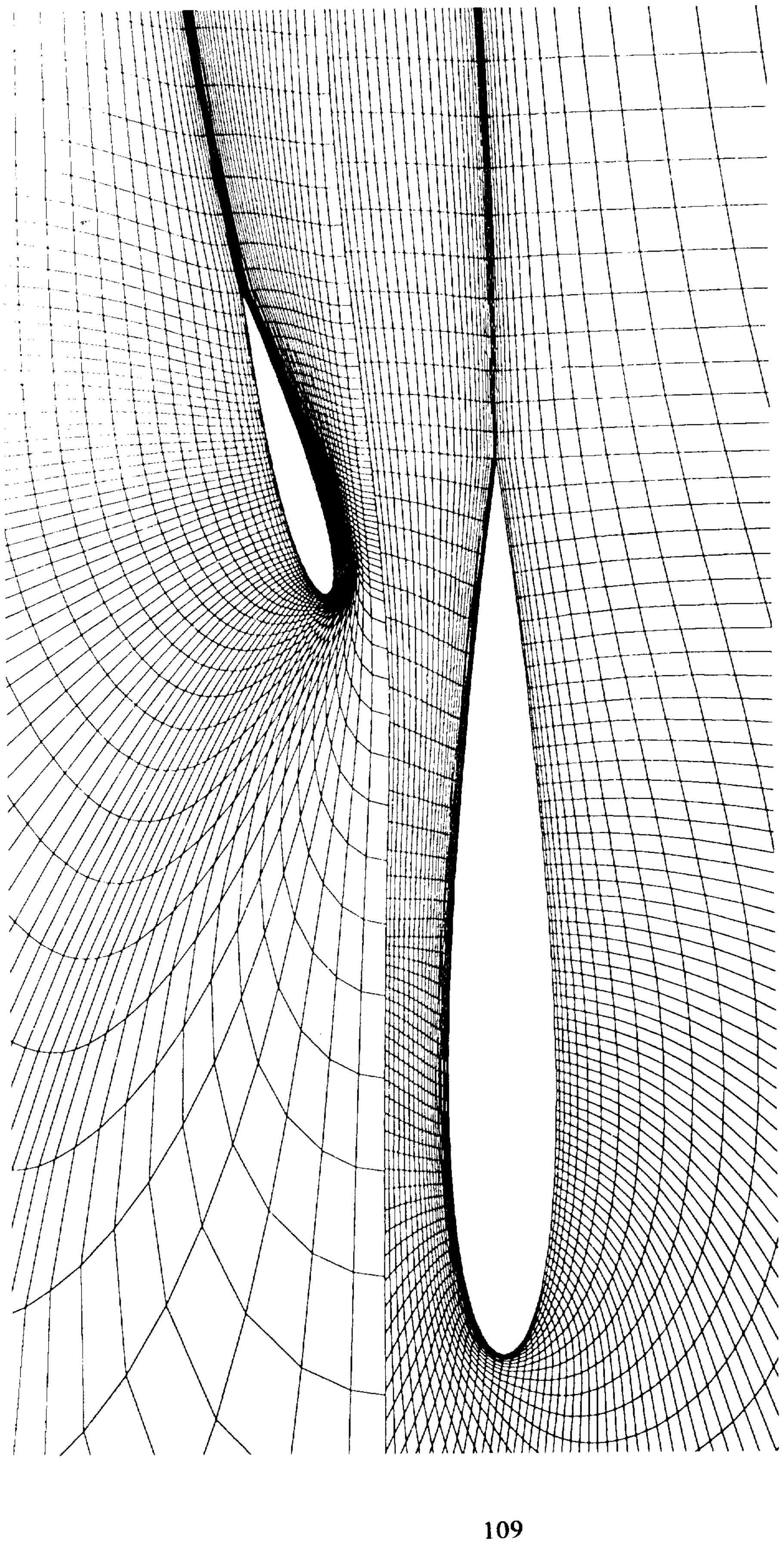

(b) GRID DETAIL – EXPERT

Figure 5.5. — Continued.



(c) 2-ZONE – EZGRID

Figure 5.5. – Continued.



(d) GRID DETAIL – EZGRID

Figure 5.5. — Concluded.

would be detected by the measurement functions for those parameters, but to zone side mapping disparity. In the user bias profile for both cases, mapping disparity is set to *medium importance.* In the calibration of the user bias profile, the expert who chose figure 5.2d (a two-zone zoning similar to the present zoning) as the best candidate for the staggered airfoil pair had assigned a weight of *medium importance* to the mapping disparity parameter, so that setting is consistent with EZGrid's zoning design for this case.

The expert exhibited some inconsistency in the tuning of the user bias profile when, for example, he discouraged tuple points but included two in his design. Based on a follow-up discussion of the results of this study, another EZGrid run was performed for the second problem in which the default user bias profile was used, which includes the following settings: tuples – *allowed*, simplicity – *low importance*, and mapping disparity – *high importance.* The zoning generated automatically by EZGrid for that input is shown in figure 5.6. This result is more similar to the design specified by the expert for the second problem.

The expert stated that it required about 20 min to design the zoning for case 1, and about 15 min for the design of the case 2 zoning. He estimated that approximately 5 *days* (3 days for case 2) would be needed to put the zoning design on the computer and to generate grids for the zoning. He had allocated 25% of that total time to grid generation, so for a more accurate comparison of the time required for zoning, his estimates would be over 3.5 days and just over 2 days for the two cases. Interactive EZGrid input of the two expert zoning designs required 2.5 hr and 1.5 hr, respectively. EZGrid generated the automated results in 1 hr (case 1) and 50 min (case 2). It is clear that the use of EZGrid, even in interactive mode, represents a large savings in time.

The final comparison between EZGrid and human performance was a simple study in which the author, who is not a zoning *expert*, but who is familiar with composite zonal grid methods, designed and generated (using EZGrid in interactive mode) a zoning for the four-element airfoil shown in figure 4.23a. The interactive zoning result predated the automated EZGrid result and represents the author's best attempt to design a zoning for that configuration. Both zonings are shown in figures 5.7a-b. The automated zoning was generated using the default user bias profile, so that profile was used to evaluate both the interactive and the automated results. EZGrid rated its own zoning higher, which is difficult to dispute in this case. The zonings appear to be comparable, but according to EZGrid, the automated result is better in terms of skewness (by an order of magnitude), orthogonality (by a factor of four), singularities (by an order of magnitude), and zone/body intersections (less than half as many). The result of this simple test is encouraging because it shows that EZGrid can produce a reasonable zoning for a complicated configuration (reasonable by virtue of its similarity to the zoning generated interactively) better than a nonexpert user.

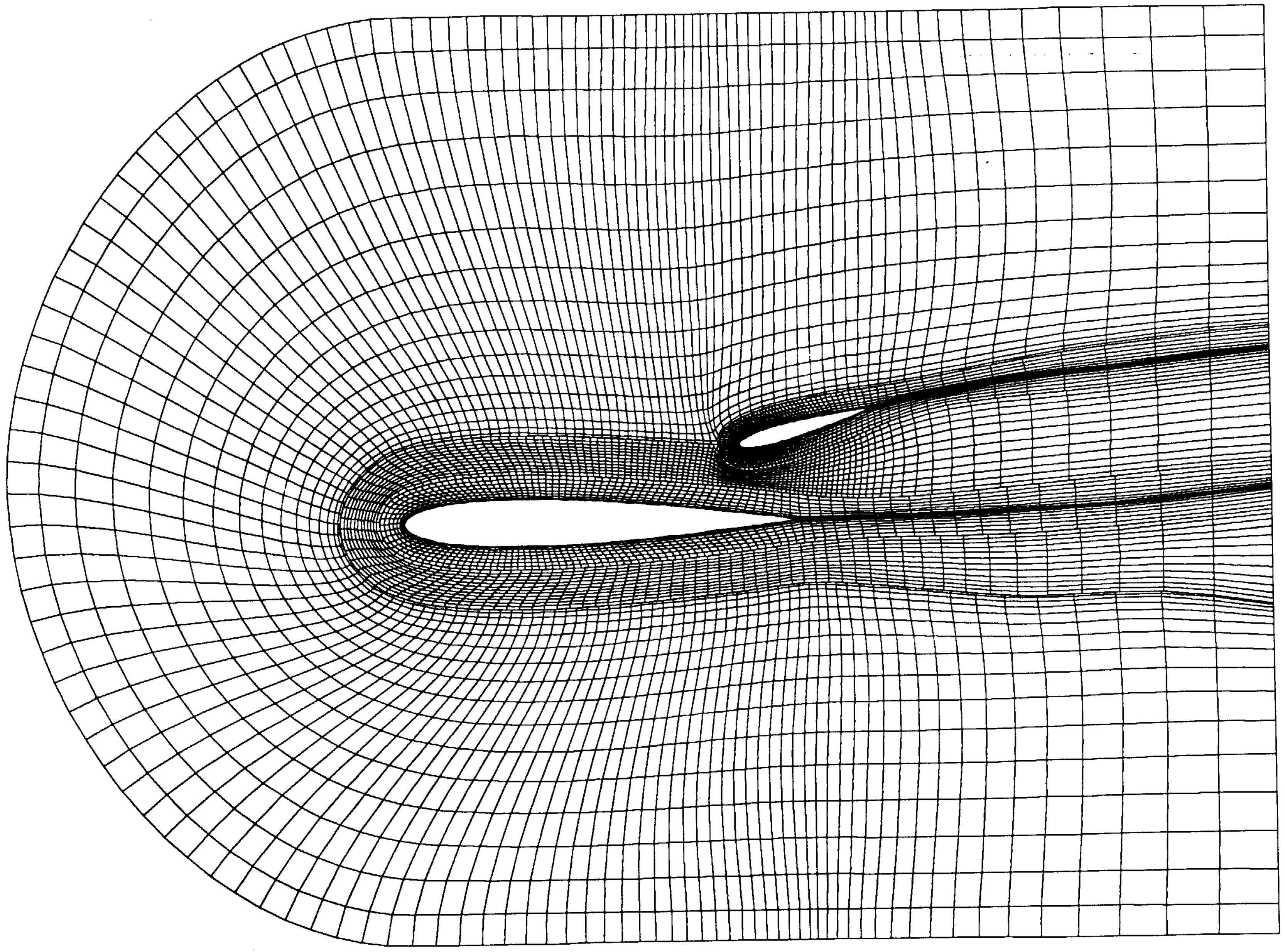
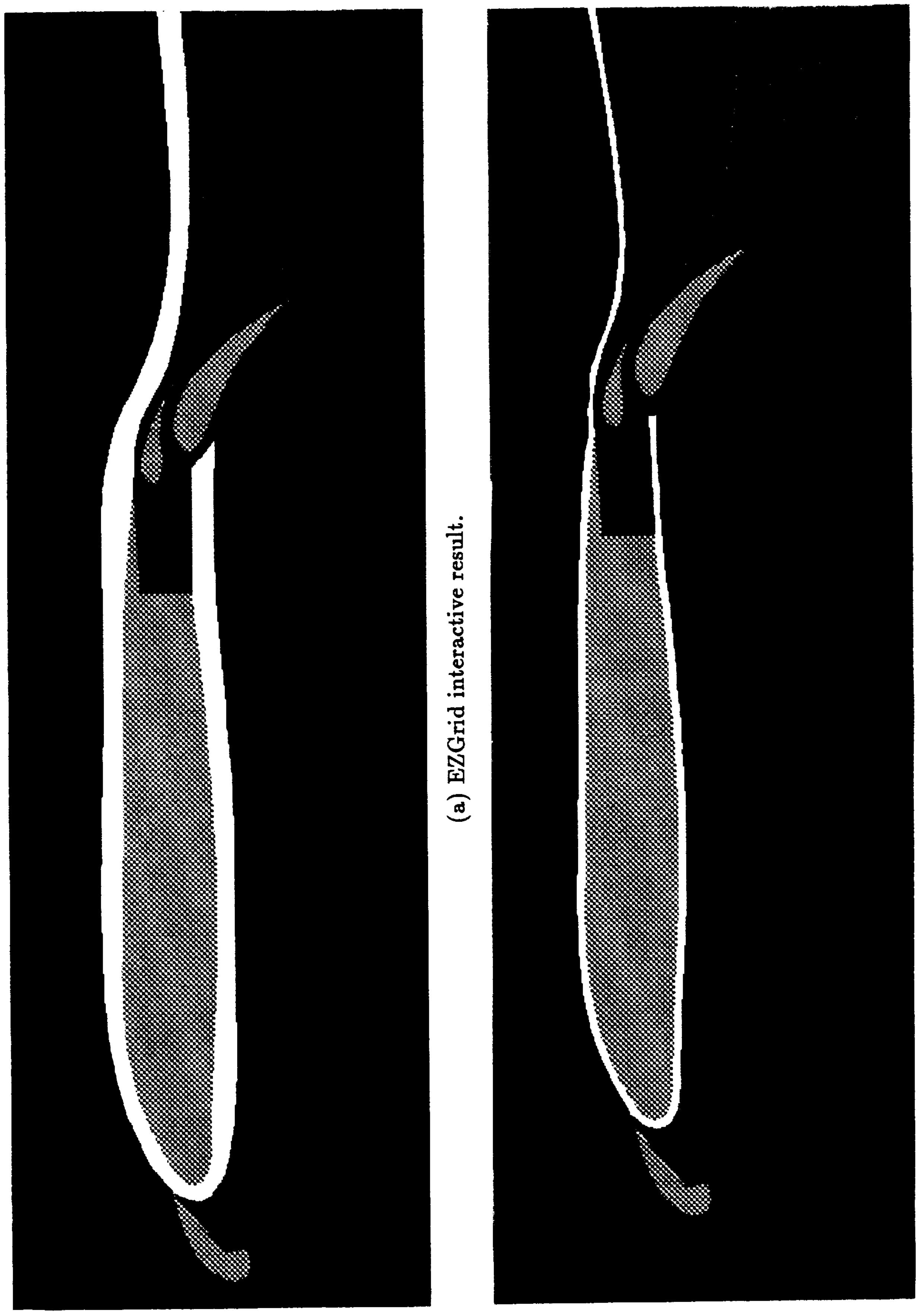


Figure 5.6. – Alternative EZGrid four-zone grid for case 2.

ORIGINAL PAGE
COLOR PHOTOGRAPH



(a) EZGrid interactive result.

(b) EZGrid automated result.

Figure 5.7. – Four-element airfoil zoning comparison.

PRECEDING PAGE BLANK NOT FILMED

PAGE 112 INTENTIONALLY BLANK

CHAPTER 6

SUMMARY AND DISCUSSION

6.1 Summary

Three-dimensional grid generation continues to pace the progress of CFD research and practice. The difficulties of grid generation, caused primarily by geometric complexity, the need for selective grid refinement, and current computer limitations, can be alleviated through the decomposition, or zoning, of a flow field into simpler regions, or zones. The resulting composite grid, made up of a collection of zonal grids, is usually superior in quality to any single grid that could be generated for the entire flow field of a complex geometry, is easier to obtain, and provides a convenient mechanism for zone-by-zone sequential processing (alleviating memory restrictions) or even parallel processing of the flow solution. Users of such zonal grid approaches have discovered the importance of zoning a flow field *well* and *quickly*. They have also discovered that those two requirements are difficult to satisfy because the element essential to both—zoning expertise—is scarce, is not easily taught, is not well understood or formulated, and criteria for judging zoning quality varies from expert to expert. The consensus is that flow-field zoning should be automated.

Automation of flow-field zoning is a difficult problem; there is no theory underlying it, and it is an ill-structured problem, with solution procedures more easily described in terms of examples rather than an algorithm or formula. The existence of these traits combined with the obvious role of zoning expertise led to the exploration of a knowledge-based approach for this problem. The goals of the present research were to (1) lay the foundation for an automated 3-D zonal grid generation capability through the development of a demonstration system that can automatically zone the flow field of representative 2-D aerodynamic configurations, and (2) determine the applicability of a knowledge-based programming approach to the automation of flow-field zoning. Both goals have been satisfied through the development of a knowledge-based system called EZGrid (Expert Zonal Grid generator), which can be used to *interactively* zone any 2-D problem, and is capable of *automatically* zoning a variety of representative 2-D problems. Chapter 4 presents results which demonstrate the range of EZGrid's automated capability.

A knowledge-based approach to the automation of flow-field zoning was successful despite the fact that some aspects of zoning disobey the guidelines for identifying tractable knowledge-based system application domains listed in section 1.3. Zoning is difficult to teach; it involves visual perception; experts do not agree on zoning methodologies or evaluation criteria; and it requires zoning design knowledge. These obstacles were overcome through the development of a zoning model, a qualitative shape and configuration description language, a tunable user bias profile, and a method of problem decomposition and plan construction. These four solutions made system development possible. They also provided valuable feedback to the CFD community in the form of better understanding of the zoning process and the role of user bias, and to the AI community in the form of experience in the application of knowledge-based techniques to a "real-world" problem of less-than-ideal suitability according to the accepted guidelines. Clearly, these guidelines do not necessarily determine the success of an application. They are good indicators of whether an application is likely to be quick and straightforward, however, and outside a research environment those are factors upon which success often depends.

6.2 EZGrid as a Foundation

The first part of the two-fold goal of this research was to lay the foundation for an automated 3-D zonal grid generation capability for CFD by developing a demonstration system for automated flow-field zoning in two dimensions. EZGrid is the system which was implemented for this purpose, and has been shown to be capable of automatically zoning the flow field about a variety of 2-D aerodynamic configurations. The question is, what sort of a foundation has been laid for future work in 3-D zonal grid generation?

The *concepts* used in the formulation of a model and language for flow-field zoning are transferable with little modification from 2-D to 3-D. Since the motivations for zoning are the same in 3-D as in 2-D, the same zoning actions would be valid. The extra dimension would necessitate modification of the language for qualitative shape and configuration description; not only the vocabulary would need modification, but probably the means of interactive input (moving to a more graphical one) would as well. The idea of plan construction through subplan assembly remains viable in 3-D. The basic flow of control of EZGrid would be appropriate for the zoning of a 3-D problem as well as for a 2-D one. Some of the knowledge about fluid dynamics and zoning is specific to 2-D problems, but because that knowledge is contained explicitly in the form of rules in the knowledge base, identification of those rules needing modification is feasible.

Work on the additional task of automating zonal grid generation has been initiated in 2-D in EZGrid. The parameters that are important in the specification of a zonal grid have begun to be identified, and rules governing the number and distribution of grid points for simple 2-D zones have been successfully developed. Although this is an extremely difficult problem, the framework provided by EZGrid facilitates the classification of zones and their interface boundaries needed for intelligent grid generation. For example, the way in which curves and objects are represented in EZGrid permits curves to be classified as *wake, body,* or *outer boundary* curves; makes it easy to determine which curves are shared over what portions; and allows for descriptions of zones in terms such as side 1 – wake-body-wake, side 2 – outer boundary curve, etc. More work is needed to capture the most effective parameterization of the 2-D grid generation problem, and in 3-D additional parameters will be necessary.

6.3 Applicability of a Knowledge-Based Approach to Flow-Field Zoning

The second part of the two-fold goal of this work was to determine the applicability of a knowledge-based programming approach to the domain of flow-field zoning. Some background on knowledge-based programming was provided in section 1.3, including guidelines, or rules, for identifying tractable problem domains for the application of knowledge-based techniques. It was recognized at the outset that a knowledge-based approach for the automation of 2-D flow-field zoning would present some difficulty because of aspects of zoning that break some of those rules. The process of zoning was not well formulated and no language existed to describe it; thus it was a difficult skill to teach. Furthermore, zoning a flow field involves perception, user bias (reflecting the lack of expert consensus in evaluating zoning quality), and zoning design knowledge, none of which are conducive to straightforward application of knowledge-based programming techniques. The manner in which these difficulties were overcome is described in detail in chapters 2, 3, and 5, and is summarized briefly here. A model and language to describe the process of zoning was developed as the first step in making the process automatable. Perceptual information is provided interactively by the user using a simple qualitative shape and configuration description language. The way in which a configuration is described (in terms of object groupings) determines whether a problem can be decomposed into simpler subproblems. User bias is incorporated by means of a tunable user bias profile, which consists of parameters measuring such quantities as the degree of zone side orthogonality and the existence of singularities. The profile is tuned by the assignment of qualitative weights to each parameter which reflect that parameter's importance or acceptability to the user. Zoning design knowledge is encoded in the form of subplans, or sequences of actions which govern the zoning of a single grouping of objects and which are assembled into plans for the zoning of entire configurations.

These four developments—a zoning model, a shape and configuration description language, a tunable user bias profile, and a method of problem decomposition and plan construction – represent more than just engineering solutions to the problems of zoning automation. They also represent contributions to both CFD and AI. The formulation of a model and language for the process of zoning is not only necessary for its automation, but has the potential to provide valuable insight to present users of flow-field zoning methods. The tunable user bias profile may be valuable to zoning experts because it represents an attempt to identify and parameterize aspects of zoning design that are not well agreed upon. Identification of differences is the first step toward their resolution. Even if expert consensus remains elusive, the ability to measure the consistency of a single user in his or her tuning of the profile and in the resulting zoning design could prove to be useful. J. S. Steger (NASA Ames, private communication) envisions a graphical consistency-checker based on the tunable user bias profile that not only evaluates a zoning (obtained with or without the help of EZGrid), but highlights the aspects that are most heavily penalized both before and after the weights of the tuned profile have been applied. With such a system at his disposal, the expert in the comparison study in section 5.3 could have performed a sensitivity analysis when confronted with apparent inconsistencies, leading perhaps to a modification of either his profile or his zoning for case 2 (figures 5.5a-b), and resulting in a more favorable comparison with the automatically generated candidate zoning of figures 5.5c-d.

The contributions of this research to the AI community fall primarily in the category of *feedback* to the sector of that community concerned with the application of AI/knowledge-based techniques to "real-world" problems. The techniques developed by AI researchers using small, well-groomed test problems must be applied to the larger, less well-behaved problems typically encountered outside the laboratory in order to discover any limitations due to scale, complexity, or other factors. The application of knowledge-based techniques to the automation of flow-field zoning has demonstrated several points:

1. A knowledge-based approach can be applied successfully to a problem which has perceptual aspects—the language developed to describe qualitative shape and configuration (position) information is simple, yet effective.
2. The absence of expert consensus on solution methodology and evaluation need not deter application of knowledge-based techniques to a problem domain as long as it is possible to characterize the differences among experts and incorporate those differences in such a way that acceptable solutions can be obtained.
3. Within the category of design-type problems are those whose solutions are designed through the composition of primitives (as opposed to being designed by the perturbation of an existing solution). Many of those problems might be transformable to simpler selection-type problems, as is the case with flow-field zoning, where subplans governing the zoning of single object groupings are *selected* and assembled into a plan for the zoning of an entire configuration.

Because EZGrid is intended as a demonstration rather than a production system, the issue of scale has not been fully addressed by this work. EZGrid contains over 400 rules in its knowledge base; many times that number would be required to reach a broader degree of competence. However, since test problems were selected that are representative of the full range of 2-D zoning problems, the run-time complexity and scale of automatically zoning any 2-D problem has been explored. Little effort was expended to make EZGrid efficient, with the exception of judiciously activating and deactivating different theories in the knowledge base (to reduce the search through the rules), and converting most backward-chaining rules to forward-chaining ones. Interestingly, one of the most time-consuming portions of an EZGrid run (in both interactive-multiple-step and automatic modes) is an activity that is easy and fast for a human—that of assessing the situation prior to making a decision about the next appropriate zoning action to execute. To do this, EZGrid must determine containment relations and object attributes (whether an object is empty, connected, filled, etc.) for each object at each stage. It was stated in section 2.2 in the description of the zoning model that as the design of a zoning progresses, more and more objects are created. This proliferation of objects slows down the situation assessment phase at each stage of an EZGrid run, whereas for a human, as the design progresses the situation becomes simpler and easier to assess at a glance.

Of course, there are aspects of flow-field zoning that encourage the use of a knowledge-based approach, based on the guidelines of section 1.3. The primary motivation for this approach was the *nature* of flow-field zoning; it is an ill-structured problem for which no solution algorithm or formula exists. Knowledge-based techniques (separating knowledge and data from the use and control of the knowledge and data, and emphasizing knowledge, including the heuristic variety) worked well for this type of problem. The facts that zoning was not well formulated and that some ideas about zoning were still evolving, while cited as deterrents to a knowledge-based approach, actually became positive factors in the choice of such an approach. The evolutionary nature of knowledge-based system development permitted ideas for solving this problem to evolve as well, rather than requiring a well-planned specification prior to development.

The choice of a knowledge-based approach for the automation of flow-field zoning in 2-D was therefore a reasonable one. The aspects of zoning which encouraged such an approach were indeed positive contributors to the success of the research, and the difficulties arising from the aspects which discouraged that approach were overcome, providing valuable feedback to the CFD and the AI communities alike. The guidelines for a successful application of knowledge-based programming techniques are clearly not hard-and-fast rules, but rather are good indicators for how straightforward and quick the application is likely to be.

6.4 Opportunities for Further Research

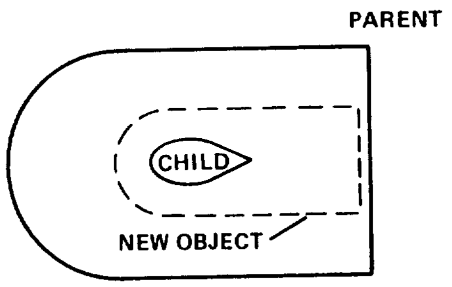
Since EZGrid is intended to be the foundation of a future automated zonal grid generation system in 3-D, future directions consist primarily of the actual extension to 3-D and the inclusion of an automated zonal grid generation capability. The second section of this chapter briefly discussed the extent to which EZGrid succeeds in forming a foundation for a 3-D system. A modification of the language for qualitative shape and configuration description is certain to be necessary. Above, below, upstream, and downstream provide insufficient relational information in 3-D, and ellipse, wedge, and the other primitive part shape descriptors are clearly limited to 2-D. The *join* concept might have to be extended as well. A full 3-D civil transport aircraft geometry would require a compound shape description involving many more parts than are encountered in the 2-D test case geometries. The use of model templates for the identification of and reasoning about typical aircraft shapes might ease the user's shape description burden. The implementation of the 3-D version of this language should provide a more graphical user interface. In fact, user interaction, especially of a graphical nature, is expected to be of more importance in a 3-D system. This expectation is based upon the greater complexity in geometric descriptions and computations, and the need for more flexibility in order to gain user acceptance (the user should be able to guide the system and input additional information whenever appropriate), since a 3-D system would be ultimately intended as a production code rather than a proof-of-concept code (like EZGrid). The hardware used for EZGrid development—the IRIS Graphics Workstation—would still be suitable for the greater demands of a 3-D system on graphics and user interaction, but access to computer-aided design (CAD) software would be necessary for 3-D geometric manipulations, including zonal interface surface generation.

Automated zonal grid generation involves more than simply the construction of a knowledge-based system to control the execution of an existing grid generator. The number and distribution of grid points on zonal interface curves (surfaces in 3-D) must be specified; these quantities are intimately related to the relationship of a zone to the other zones in the flow field, the types of curves (surfaces) that make up the perimeter of the zone, and many of the same factors which influenced the zoning, such as user bias and inflow conditions. The rules in EZGrid's knowledge base which determine the grid point placement for a few simple cases represent a good start on the solution to this problem, but all of the important parameters have not yet been identified, particularly for grid generation in 3-D.

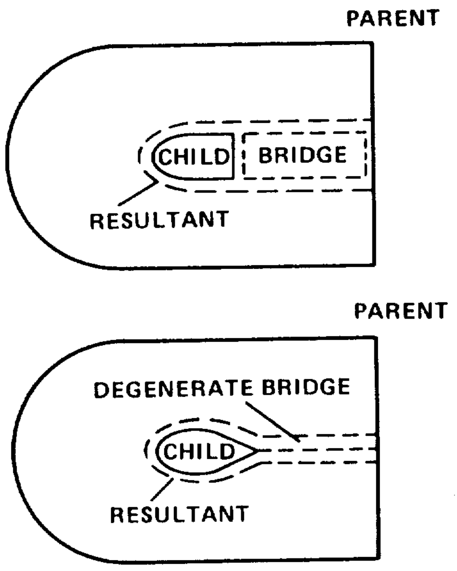
APPENDIX A
ZONING ACTIONS AND CONSEQUENCES

This appendix contains a complete list of all possible zoning actions, the conditions necessary for the application of those actions in a zoning situation, and the consequences (i.e., changes to the zoning situation) of each action. Also included are examples of each action, as were shown in figure 2.5.

1. ZONING ACTION:
   connection/envelop – parent-child
   CONDITIONS FOR APPLICABILITY:
   – parent object is a zoning environment
   – parent is not filled
   – parent is not connected
   – parent directly contains an object (child)
   CONSEQUENCES:
   – 1 new object created
   – parent contains new object
   – new object contains child

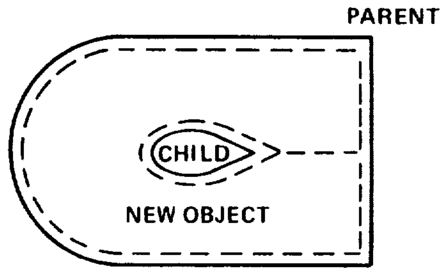


2. ZONING ACTION:
   connection/bridge – parent-child
   CONDITIONS FOR APPLICABILITY:
   – parent object is a zoning environment
   – parent is not filled
   – parent directly contains an object (child)
   CONSEQUENCES:
   – 2 new objects created
     – bridge object
     – resultant object (= bridge + child)
   – parent contains resultant object
   – resultant object contains bridge, child



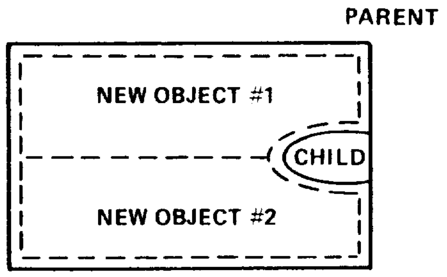
3. ZONING ACTION:
    connection/cut – parent-child
  CONDITIONS FOR APPLICABILITY:
  - parent object is a zoning environment
  - parent is not filled
  - parent is not connected
  - parent directly contains an object (child)

  CONSEQUENCES:
  - 1 new object created
  - parent contains new object
  - parent still directly contains child
  - parent filled by new object and child



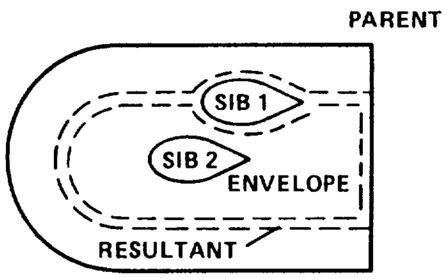
4. ZONING ACTION:
    connection/cut – parent-child
  CONDITIONS FOR APPLICABILITY:
  - parent object is a zoning environment
  - parent is not filled
  - parent is connected
  - parent directly contains an object (child)

  CONSEQUENCES:
  - 2 new objects created
  - parent contains both new objects
  - parent still directly contains child
  - parent filled by new objects and child



5. ZONING ACTION:
    connection/envelop – sibling-sibling
  CONDITIONS FOR APPLICABILITY:
  - parent object is a zoning environment
  - parent is not filled
  - parent directly contains 2 objects (siblings)

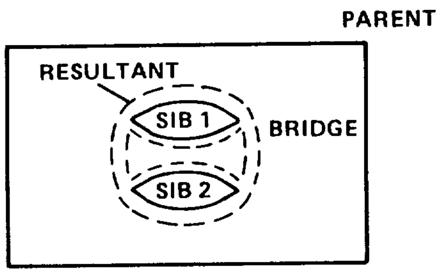
  CONSEQUENCES:
  - 2 new objects created
    - envelope object
    - resultant object (= envelope + sibling #1)
  - parent contains resultant object
  - envelope contains sibling #2
  - resultant object contains envelope, sibling #1

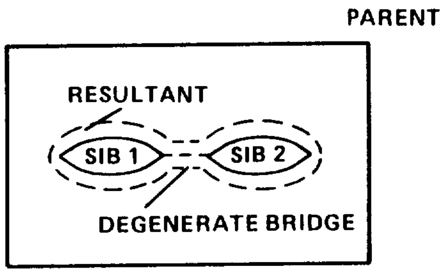


6. ZONING ACTION:
connection/bridge – sibling-sibling

CONDITIONS FOR APPLICABILITY:
- parent object is a zoning environment
- parent is not filled
- parent directly contains 2 objects (siblings)

CONSEQUENCES:
- 2 new objects created
  - bridge object
  - resultant object (= bridge + both siblings)
- parent contains resultant object
- resultant object contains bridge, siblings

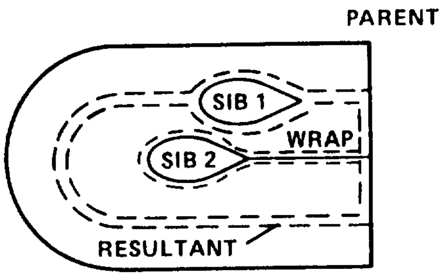




7. ZONING ACTION:
connection/wrap – sibling-sibling

CONDITIONS FOR APPLICABILITY:
- parent object is a zoning environment
- parent is not filled
- parent directly contains 2 objects (siblings)

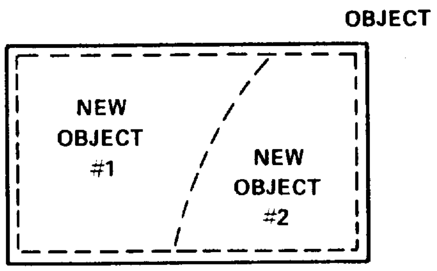
CONSEQUENCES:
- 2 new objects created
  - wrapping object
  - resultant object (= wrapping object + both siblings
- parent contains resultant object
- resultant object contains wrapping object, both siblings



8. ZONING ACTION:
connection/cut – self-interior

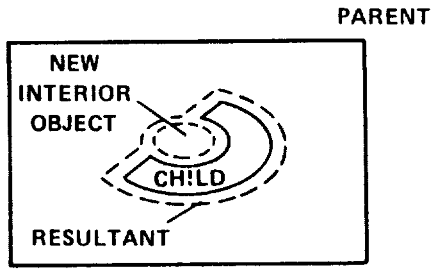
CONDITIONS FOR APPLICABILITY:
- object is a zoning environment
- object is not filled
- object is not zoned

CONSEQUENCES:
- 2 new objects created, one on each side of cut
- object contains both new objects



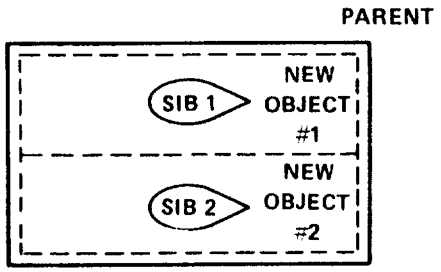
9. ZONING ACTION:
connection/cut – self-exterior
CONDITIONS FOR APPLICABILITY:
- parent object is a zoning environment
- parent is not filled
- parent directly contains an object (child) (action applied to child)

CONSEQUENCES:
- 2 new objects created
  - object enclosing region between child and cut
  - resultant object (= first new object + child)
- parent contains resultant object
- resultant object contains first new object, child



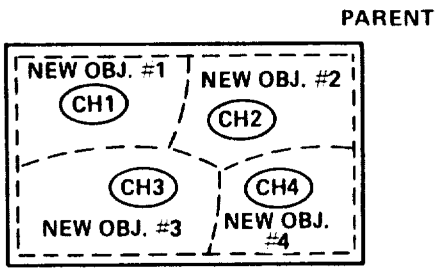
10. ZONING ACTION:
separation/cut – sibling-sibling
CONDITIONS FOR APPLICABILITY:
- parent object is a zoning environment
- parent is not filled
- parent directly contains at least two different objects (siblings)

CONSEQUENCES:
- 2 new objects created, one on each side of cut
- parent contains both objects
- each new object contains one or more siblings



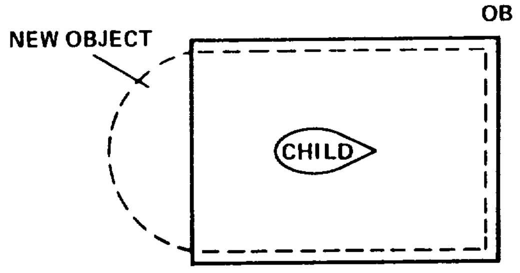
11. ZONING ACTION:
separation/partition – all-objects
CONDITIONS FOR APPLICABILITY:
- parent object is a zoning environment
- parent is not filled
- parent directly contains at least 2 objects (children)

CONSEQUENCES:
- n new objects created (where n = number of children)
- parent contains all new objects
- each new object contains a child object
- parent filled by new objects



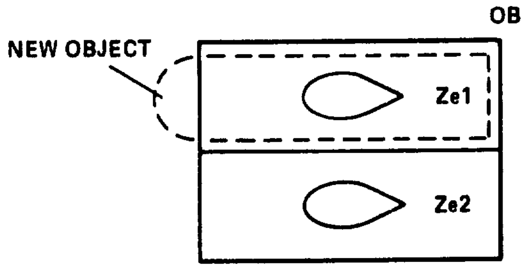
12. ZONING ACTION:
shape-alteration/envelop – self-actual
CONDITIONS FOR APPLICABILITY:
- object is the outer boundary zoning environment (OB)
- OB is not filled

CONSEQUENCES:
- 1 new object created
- OB contains new object
- new object contains all children of OB
- assertion that new object fills OB (tantamount to setting new object *equal* to OB – that not done directly so that OB's actual perimeter curves remain unchanged for alternative zoning candidates)



13. ZONING ACTION:
shape-alteration/envelop – self-actual
CONDITIONS FOR APPLICABILITY:
- object is a zoning environment
- object is not filled
- object is not zoned
- OB is filled
- object is one of OB's filling objects

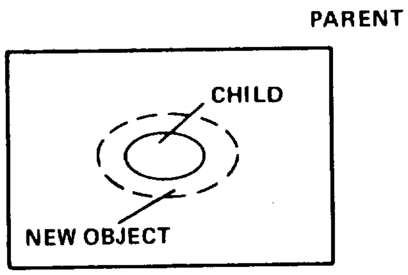
CONSEQUENCES:
- 1 new object created
- old object contains new object
- new object contains all children of old object
- assertion that new object fills old object (tantamount to setting new object *equal* to old object – that not done directly so that old object's actual perimeter curves remain unchanged for alternative zoning candidates)



OB FILLED BY Ze1, Ze2

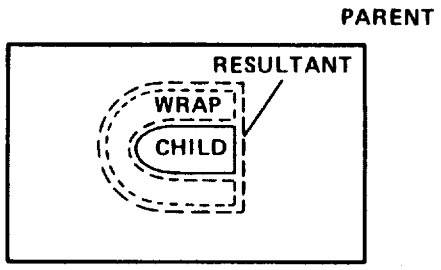
14. ZONING ACTION:
shape-alteration/envelop – self-virtual
CONDITIONS FOR APPLICABILITY:
- parent is a zoning environment
- parent is not filled
- parent directly contains an object (child)

(action applied to child)
CONSEQUENCES:
- 1 new object created
- parent contains new object
- new object contains child



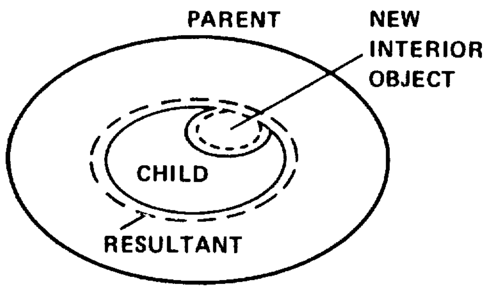
15. ZONING ACTION:
shape-alteration/wrap – self-virtual
CONDITIONS FOR APPLICABILITY:
- parent is a zoning environment
- parent is not filled
- parent directly contains an object (child)

(action applied to child)
CONSEQUENCES:
- 2 new objects created
  - wrapping object
  - resultant object (= wrapping object + child)
- parent contains resultant object
- resultant object contains wrapping object, child



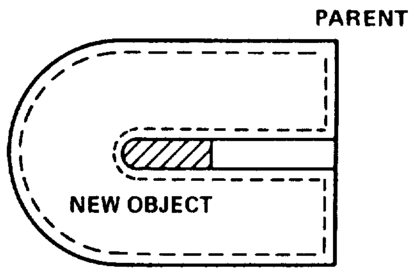
16. ZONING ACTION:
shape-alteration/cut – self-virtual
CONDITIONS FOR APPLICABILITY:
- parent is a zoning environment
- parent is not filled
- parent directly contains an object (child)

(action applied to child)
CONSEQUENCES:
- 2 new objects created
  - object enclosing region between child and cut
  - resultant object (= first new object + child)
- parent contains resultant object
- resultant object contains first new object, child



17. ZONING ACTION:
identification/identify – region
CONDITIONS FOR APPLICABILITY:
- parent is a zoning environment
- parent is not filled or empty
- parent is connected

(action applied to unidentified region of parent)
CONSEQUENCES:
- n new objects created
  (where n = number of separate, unidentified regions within parent object)
- parent contains all new objects
- all new objects empty
- parent filled by new objects and all other objects that it directly contains



APPENDIX B
AN EXAMPLE OF AN EZGRID DATABASE

The listing beginning on the next page details the contents of the EZGrid database at the conclusion of an automatic run in which the flow field about a tilted, cambered airfoil was zoned. Three stages were required to design and generate the single-zone result shown in the figure below. Each proposition is surrounded by parentheses, just as it appears in the database. Note that propositions involving zoning actions, e.g.,

(executable (plan p1) (operator bridge)
(application parent-child (OB b1)) (objective connection) (stage 1))

describe the action in a somewhat different form than the operator-operand pair of section 2.3.2. The form of an action used here has three elements: an objective, an application, and an operator. The objective and operator of this form are combined to be the operator of section 2.3.2. The application and the objects to which the action is to be applied are combined to be the operand of section 2.3.2. The reason for the difference lies in the fact that originally, zoning actions were modeled as having three elements, and that is the model that was implemented. The operator-operand model is conceptually identical, but is easier to explain and is more consistent with usual mathematical terminology.

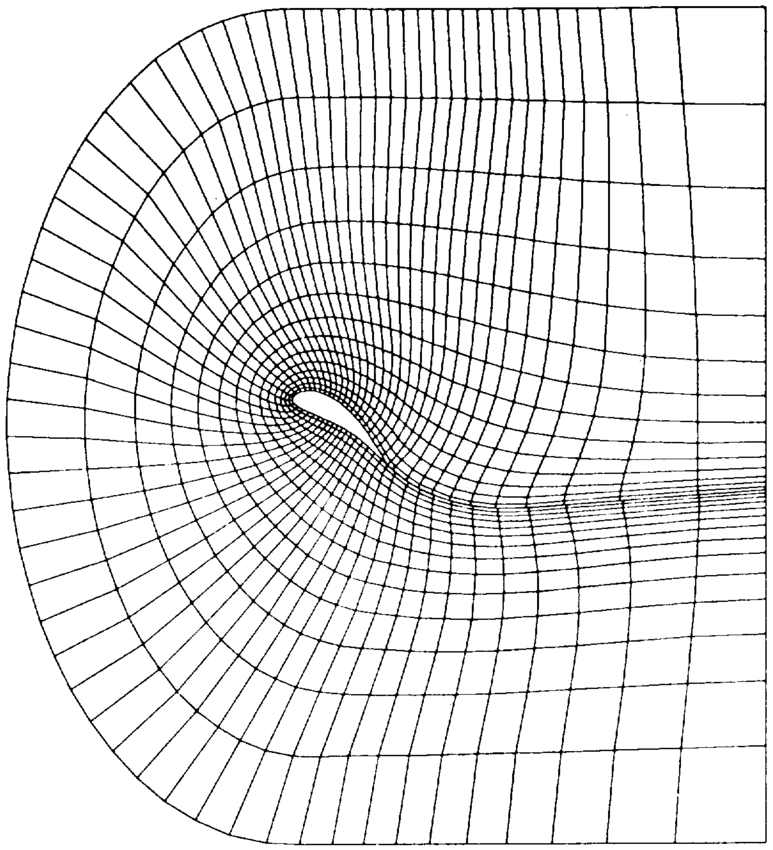


Figure B.1. – Single-zone grid for cambered airfoil at $\alpha = 5°, M_\infty = 0.8$.

Theory INIT – Stage 1

```
(mode automatic)
(userbias simplicity low)
(userbias skewness high)
(userbias smoothness medium)
(userbias mapping high)
(userbias efficiency low)
(userbias orthogonality high)
(userbias surface-vs-field low)
(userbias wake high)
(userbias tuples allowed)
(userbias singularities allowed)
(userbias intersections allowed)
(userbias viscosity allowed)
(body b1 (stage 1))
(original-body b1 (stage 1))
(contains OB b1 (stage 1))
(newcurve c1 (object b1) (stage 1))
(newobject b1 (stage 1))
(vertices b1 done (stage 1))
(flowcondition flow-direction left-to-right)
(zoning-environment OB (stage 1))
(contains-body OB (stage 1))
(body OB (stage 1))
(flowconditionsneeded (angle-of-attack mach-number steadiness
    accuracy boundary-conditions))
(flowcondition mach-number 0.8)
(flowcondition speed-regime transonic)
(flowcondition speed-regime subsonic)
(flowcondition steadiness steady)
(flowcondition angle-of-attack 0.0698132)
(flowsolverinfo accuracy low)
(flowsolverinfo boundary-conditions nil)
(accuracy determined)
(angle-of-attack determined)
(steadiness determined)
(mach-number determined)
(describe-preliminary-coreobjects (stage 1))
(preliminary-corebody OB (b1) (stage 1))
(dimensionsneeded OB)
(construction-shape OB bullet)
(suggested-distance-in-chords upstream OB 4.0)
(suggested-distance-in-chords downstream OB 4.0)
(suggested-distance-in-chords above OB 6.0)
(suggested-distance-in-chords below OB 6.0)
(compute OB (stage 1))
(operation-objects (OB) (stage 1))
(object-needed OB (type created) (specification object) (stage 1))
(object-parameter OB (shape bullet) (stage 1))
(object-parameter OB (upstream-length 2.03655) (stage 1))
(object-parameter OB (downstream-length 2.7154) (stage 1))
(object-parameter OB (above-width 2.7154) (stage 1))
```

```
(object-parameter OB (below-width 2.7154) (stage 1))
(object-parameters OB done (stage 1))
(object-curves OB (c2 c3) (stage 1))
(curve-needed c2 (type new) (object OB) (stage 1))
(curve-parameter c2 (begin-point ((point) (4.49353 -3.47055))) (stage 1))
(curve-parameter c2 (begin-angle unspecified) (stage 1))
(curve-parameter c2 (interior-points (((point) (1.09928 -3.47055))
    ((point) (1.07891 -3.47055)) ((point) (-0.93727 -0.486344))
    ((point) (1.07891 2.49786)) ((point) (1.09928 2.49786)))) (stage 1))
(curve-parameter c2 (end-point ((point) (4.49353 2.49786))) (stage 1))
(curve-parameter c2 (end-angle unspecified) (stage 1))
(curve-parameters c2 done (stage 1))
(numknots c2 7)
(bctypes c2 (free free))
(bcvalues c2 (0.0 0.0 0.0 0.0))
(knot-locations c2 ((4.49353 -3.47055) (1.09928 -3.47055)
    (1.07891 -3.47055) (-0.93727 -0.486344) (1.07891 2.49786)
    (1.09928 2.49786) (4.49353 2.49786)) (stage 1))
(curve-completed c2 (stage 1))
(newcurve c2 (object OB) (stage 1))
(curve-needed c3 (type new) (object OB) (stage 1))
(curve-parameter c3 (begin-point ((point) (4.49353 2.49786))) (stage 1))
(curve-parameter c3 (begin-angle unspecified) (stage 1))
(curve-parameter c3 (interior-points nil) (stage 1))
(curve-parameter c3 (end-point ((point) (4.49353 -3.47055))) (stage 1))
(curve-parameter c3 (end-angle unspecified) (stage 1))
(curve-parameters c3 done (stage 1))
(numknots c3 2)
(bctypes c3 (free free))
(bcvalues c3 (0.0 0.0 0.0 0.0))
(knot-locations c3 ((4.49353 2.49786) (4.49353 -3.47055)) (stage 1))
(curve-completed c3 (stage 1))
(newcurve c3 (object OB) (stage 1))
(object-completed OB (stage 1))
(operation-completed (stage 1))
(newobject OB (stage 1))
(vertices OB done (stage 1))
(flowcondition domain-type external)
(describe-containment (stage 1))
(directly-contains OB b1 (stage 1))
(describe-coreobjects (stage 1))
(corebody OB (b1) (stage 1))
(numbodies OB 1 (stage 1))
(coretype OB single (stage 1))
(contained-environments OB nil (stage 1))
(numenvironments OB 0 (stage 1))
(describe-topology (stage 1))
(describe-filling (stage 1))
(configuration-grouping OB g1 (b1))
(configuration-groups OB 1)
(flowconditionsneeded (viscosity turbulence turbulence-model
    flow-direction angle-of-attack domain-type))
(flow-direction determined)
```

```
(viscosity determined)
(batch-mode batch)
(endofinitialdata (stage 1))
(flowcondition viscosity inviscid)
```

Theory INIT-ACTION – Stage 1

```
(preparesubplans (stage 1))
(subplan sp1 g1 (primary (((objective connection)
     (application parent-child $pc) (operator bridge)
     (strategy degenerate-zone) (strategy follow-wake)
     (strategy identify (original-body b1)))))
     (completion (((objective identification) (application region ($z1))
     (operator identify)) (topology-imposition $z2 (strategy impose-c)))))
(readytoplan (stage 1))
(plan p1 (pending (((objective connection)
     (application parent-child $578) (operator bridge)
     (strategy degenerate-zone) (strategy follow-wake)
     (strategy identify (original-body b1)))
     ((objective identification) (application region ($576))
     (operator identify)) (topology-imposition $577 (strategy impose-c))))
     (executed nil) (stage 1))
(readytozone (stage 1))
(possible (objective connection) (stage 1))
(possible (objective shape-alteration) (stage 1))
(objective-executables (stage 1))
(executable (objective connection) (stage 1))
(possible (application parent-child (OB b1)) (objective connection) (stage 1))
(possible (application self-interior (OB)) (objective connection) (stage 1))
(possible (application self-exterior (b1)) (objective connection) (stage 1))
(application-executables (stage 1))
(executable (application parent-child (OB b1))
     (objective connection) (stage 1))
(possible (operator envelop) (application parent-child (OB b1))
     (objective connection) (stage 1))
(possible (operator bridge) (application parent-child (OB b1))
     (objective connection) (stage 1))
(possible (operator cut) (application parent-child (OB b1))
     (objective connection) (stage 1))
(operation-executables (stage 1))
(executable (plan p1) (operator bridge) (application parent-child (OB b1))
     (objective connection) (stage 1))
```

Theory T1 – Stage 2

```
(choose-action (plan p1) (operator bridge)
    (application parent-child (OB b1)) (objective connection) (stage 2))
(assert-strategies (stage 2))
(strategy degenerate-zone (stage 2))
(strategy follow-wake (stage 2))
(strategy identify (original-body b1) (stage 2))
(chosen (operator bridge) (application parent-child (OB b1))
    (objective connection) (stage 2))
(operation-objects (ze1 ze2) (stage 2))
(object-needed ze1 (type created) (specification curve) (stage 2))
(object-curves ze1 (c4) (stage 2))
(curve-needed c4 (type new) (object ze1) (stage 2))
(curve-parameter c4 (begin-point ((curve c1 0.0) (1.77813 -0.755149)))
    (stage 2))
(new-connection (c1 (0.0 (c4 0.0))) (stage 2) (theory t1))
(curve-parameter c4 (begin-angle 5.36057) (stage 2))
(curve-parameter c4 (interior-points (((point) (1.90781 -0.864631))
    ((point) (2.05507 -0.94899)) ((point) (2.21346 -1.00992))
    ((point) (2.37852 -1.04939)) ((point) (2.54706 -1.06934))
    ((point) (2.71675 -1.07159)) ((point) (2.88605 -1.05976))
    ((point) (3.05535 -1.04792)))) (stage 2))
(curve-parameter c4 (end-point ((curve c3 0.577209)
    (4.49353 -0.947162))) (stage 2))
(new-connection (c3 (0.577209 (c4 1.0))) (stage 2) (theory t1))
(curve-divided c3 (stage 2))
(curve-parameter c4 (end-angle 3.14159) (stage 2))
(curve-parameter c4 (begin-derivative-weight 0.0848561) (stage 2))
(curve-parameter c4 (end-derivative-weight 0.720851) (stage 2))
(curve-parameters c4 done (stage 2))
(numknots c4 10)
(bctypes c4 (specified specified))
(bcvalues c4 (0.0512308 -0.0676459 0.720851 -1.71864e-06))
(knot-locations c4 ((1.77813 -0.755149) (1.90781 -0.864631)
    (2.05507 -0.94899) (2.21346 -1.00992) (2.37852 -1.04939)
    (2.54706 -1.06934) (2.71675 -1.07159) (2.88605 -1.05976)
    (3.05535 -1.04792) (4.49353 -0.947162)) (stage 2))
(curve-completed c4 (stage 2))
(newcurve c4 (object ze1) (stage 2))
(object-completed ze1 (stage 2))
(zoning-environment ze1 (stage 2))
(degenerate ze1 (stage 2))
(zoned ze1 (stage 2))
(newobject ze1 (stage 2))
(vertices ze1 done (stage 2))
(object-needed ze2 (type resultant) (specification merge) (stage 2))
(object-completed ze2 (stage 2))
(operation-completed (stage 2))
(contains OB ze2 (stage 2))
(contains ze2 b1 (stage 2))
(contains OB b1 (stage 2))
(contains ze2 ze1 (stage 2))
(contains OB ze1 (stage 2))
(zoning-environment ze2 (stage 2))
(contains-environment ze2 (stage 2))
```

```
(newobject ze2 (stage 2))
(vertices ze2 done (stage 2))
(leading-edge ze2 done (stage 2))
(updateplan (stage 2))
(plan p1 (pending (((objective identification)
    (application region ($737)) (operator identify))
    (topology-imposition $738 (strategy impose-c))))
    (executed (((objective connection) (application parent-child (OB b1))
    (operator bridge) (strategy degenerate-zone) (strategy follow-wake)
    (strategy identify (original-body b1))))) (stage 2))
(updateframe (stage 2))
(connection-list ((c1 (0.0 (c4 0.0))) (c3 (0.577209 (c4 1.0))))
    (stage 2) (theory t1))
(updated connections (stage 2) (theory t1))
(body b1 (stage 2))
(original-body b1 (stage 2))
(contains-body ze2 (stage 2))
(body ze2 (stage 2))
(body OB (stage 2))
(zoning-environment OB (stage 2))
(contains-body OB (stage 2))
(contains-environment OB (stage 2))
(endofinitialdata (stage 2))
(describe-containment (stage 2))
(directly-contains ze2 b1 (stage 2))
(directly-contains ze2 ze1 (stage 2))
(directly-contains OB ze2 (stage 2))
(describe-coreobjects (stage 2))
(corebody ze1 nil (stage 2))
(numbodies ze1 0 (stage 2))
(corebody ze2 (b1) (stage 2))
(numbodies ze2 1 (stage 2))
(coretype ze2 single (stage 2))
(corebody OB (ze2) (stage 2))
(numbodies OB 1 (stage 2))
(coretype OB single (stage 2))
(contained-environments ze1 nil (stage 2))
(connected ze1 (stage 2))
(numenvironments ze1 0 (stage 2))
(empty ze1 (stage 2))
(contained-environments ze2 (ze1) (stage 2))
(connected ze2 (stage 2))
(numenvironments ze2 1 (stage 2))
(contained-environments OB (ze2) (stage 2))
(connected OB (stage 2))
(numenvironments OB 1 (stage 2))
(describe-topology (stage 2))
(describe-filling (stage 2))
(filled ze2 (stage 2))
(filling-objects ze2 (ze1 b1) (stage 2))
(zoned ze2 (stage 2))
```

Theory T1-ACTION – Stage 2

```
(readytoplan (stage 2))
(readytozone (stage 2))
(possible (objective connection) (stage 2))
(possible (objective shape-alteration) (stage 2))
(possible (objective identification) (stage 2))
(objective-executables (stage 2))
(executable (objective identification) (stage 2))
(possible (application region (OB)) (objective identification) (stage 2))
(application-executables (stage 2))
(executable (application region (OB)) (objective identification) (stage 2))
(possible (operator identify) (application region (OB))
     (objective identification) (stage 2))
(operation-executables (stage 2))
(executable (plan p1) (operator identify) (application region (OB))
     (objective identification) (stage 2))
```

Theory T2 – Stage 3

```
(choose-action (plan p1) (operator identify) (application region (OB))
     (objective identification) (stage 3))
(assert-strategies (stage 3))
(chosen (operator identify) (application region (OB))
     (objective identification) (stage 3))
(operation-objects (ze3) (stage 3))
(object-needed ze3 (type resultant) (specification subtract) (stage 3))
(object-completed ze3 (stage 3))
(operation-completed (stage 3))
(zoning-environment ze3 (stage 3))
(newobject ze3 (stage 3))
(vertices ze3 done (stage 3))
(contains OB ze3 (stage 3))
(updateplan (stage 3))
(updateframe (stage 3))
(contains OB ze2 (stage 3))
(contains ze2 b1 (stage 3))
(contains OB b1 (stage 3))
(contains ze2 ze1 (stage 3))
(contains OB ze1 (stage 3))
(body b1 (stage 3))
(original-body b1 (stage 3))
(body ze2 (stage 3))
(body OB (stage 3))
(zoning-environment ze1 (stage 3))
(degenerate ze1 (stage 3))
(zoned ze1 (stage 3))
(zoning-environment ze2 (stage 3))
(contains-body ze2 (stage 3))
(contains-environment ze2 (stage 3))
```

```
(zoning-environment OB (stage 3))
(contains-body OB (stage 3))
(contains-environment OB (stage 3))
(zoned ze2 (stage 3))
(filled ze2 (stage 3))
(connection-list ((c1 (0.0 (c4 0.0))) (c3 (0.577209 (c4 1.0))))
    (stage 3) (theory t2))
(filling-objects ze2 (ze1 b1) (stage 3))
(coretype ze2 single (stage 3))
(corebody ze2 (b1) (stage 3))
(numbodies ze2 1 (stage 3))
(contained-environments ze2 (ze1) (stage 3))
(connected ze2 (stage 3))
(numenvironments ze2 1 (stage 3))
(endofinitialdata (stage 3))
(describe-containment (stage 3))
(directly-contains ze2 b1 (stage 3))
(directly-contains ze2 ze1 (stage 3))
(directly-contains OB ze3 (stage 3))
(directly-contains OB ze2 (stage 3))
(describe-coreobjects (stage 3))
(corebody ze3 nil (stage 3))
(numbodies ze3 0 (stage 3))
(corebody ze1 nil (stage 3))
(numbodies ze1 0 (stage 3))
(corebody OB (ze2) (stage 3))
(numbodies OB 1 (stage 3))
(coretype OB single (stage 3))
(contained-environments ze3 nil (stage 3))
(connected ze3 (stage 3))
(numenvironments ze3 0 (stage 3))
(empty ze3 (stage 3))
(contained-environments ze1 nil (stage 3))
(connected ze1 (stage 3))
(numenvironments ze1 0 (stage 3))
(empty ze1 (stage 3))
(contained-environments OB (ze3 ze2) (stage 3))
(connected OB (stage 3))
(numenvironments OB 2 (stage 3))
(describe-topology (stage 3))
(possible topology-imposition ze3 (stage 3))
(strategy impose-c (stage 3))
(desirable topology-imposition ze3 (stage 3))
(sidelist ze3 done (stage 3))
(zone ze3 (stage 3) (theory t2))
(zoned ze3 (stage 3))
(plan p1 (pending nil) (executed ((topology-imposition ze3
    (strategy impose-c)) ((objective identification)
    (application region (OB)) (operator identify))
    ((objective connection) (application parent-child (OB b1))
    (operator bridge) (strategy degenerate-zone) (strategy follow-wake)
    (strategy identify (original-body b1))))) (stage 3))
(describe-filling (stage 3))
```

```
(filled OB (stage 3))
(filling-objects OB (ze3 ze2) (stage 3))
(zoned OB (stage 3))
(zoningcomplete (stage 3) (theory t2))

(describe-grid ze3 (c7 c1 c4) (c9) (c2) (c6) (stage 3))
(grid-type ze3 inviscid (body-sides eta0) (stage 3))
(grid-size ze3 66 15)
(grid-spacing ze3 xi0 (15 0.00910158 0.273047))
(grid-spacing ze3 xi1 (15 0.00666667 0.2))
(describe-xi-spacing ze3 (c7 c1 c4) 7.39676 (c2) 14.7501 (stage 3))
(grid-control-point ze3 eta1 ((curve c2 0.25546) (0.911677 -3.45999)))
(grid-control-point ze3 eta1 ((curve c2 0.645673) (-0.227126 1.77827)))
(grid-control-point ze3 eta1 ((curve c2 0.884181) (2.86871 2.49641)))
(grid-spacing ze3 eta0 (66 0.086024 0.08611))
(grid-spacing ze3 eta1 (12 0.176552 0.0833333) (24 0.0479767 0.0416667)
    (24 0.0771451 0.0416667) (6 0.0862374 0.394416))
(grid-spacing ze3 done (stage 3))
```

APPENDIX C
EXAMPLE RULES FROM THE EZGRID KNOWLEDGE BASE

META RULES

META rules link lisp files to MRS propositions, and contain forward chaining instructions, as shown by the following two examples:

```
(assert '(lisp WhichVertexFQ which_vertex))
```
- note: the first function name is the MRS name, the second is the actual lisp function name

```
(assert '(toassert &p fc))
```
- note: the & denotes an MRS meta-level variable, which can be matched with propositions rather than just with simple terms, as is the case with ground-level variables such as $x

SITUATION RULES

These rules update the zoning situation at each stage by determining the varying properties of each object, such as whether it is zoned, empty, or filled, and what it contains.

```
(if (and (sidelist $z done (stage $n))
         (zoning-environment $z (stage $n))
         (empty $z (stage $n))
         (PROVABLE (zoneshaped $z)))
    (zone $z (stage $n)))
```
- note: PROVABLE is an MRS term which invokes backward chaining on the proposition which follows it

```
(if (and (describe-containment (stage $n))
         (zoning-environment $z1 (stage $n))
         (contains $z1 $z2 (stage $n))
         (DirectlyContainsRQ $z1 $z2 $n))
    (directly-contains $z1 $z2 (stage $n)))
```
- note: DirectlyContainsRQ is a lisp function that returns true or false, and which looks for intermediate containment relations to determine direct containment

```
(if (and (describe-filling (stage $n))
         (zoning-environment $z (stage $n))
         (UNKNOWN (filled $z (stage $n)))
         (contained-environments $z $ce (stage $n))
         (corebody $z $cb (stage $n))
         (ApplyFQ append ($ce $cb) $clist)
         (AllCurvesSharedRQ $z $clist))
    (filled $z (stage $n)))
```

CONTROL RULES

CONTROL rules contain knowledge about how to build up EZGrid's database structure and when to begin plan construction.

```
(if (and (end-of-initial-data (stage $n))
         (mode automatic)
         (> $n 1)
         (AssertDataRQ $n)
         (UNKNOWN (zoning-complete (stage $n))
         (EvalFQ theory $th)
         (ApplyFQ concat ($th -action) $newname)
         (StartNewTheoryRQ $th $newname))
    (ready-to-plan (stage $n)))
```

FRAME RULES

FRAME rules contain knowledge about which variable object properties can be assumed invariant and reasserted at the next stage of the zoning.

```
(if (and (update-frame (stage $n))
         (- $n 1 $m)
         (filled $z (stage $m)))
    (filled $z (stage $n)))

(if (and (update-frame (stage $n))
         (- $n 1 $m)
         (filled $z (stage $m))
         (corebody $z $cb (stage $m)))
    (corebody $z $cb (stage $n)))
```

INFLOW RULES

These rules contain basic fluid dynamics knowledge about speed regime, Mach angle, and high lift conditions.

```
(if (and (flow-conditions-needed $condition-list)
         (MEMBER mach-number $condition-list)
         (GetFlowConditionRQ mach-number))
    (mach-number determined))
```

– note: the lisp function GetFlowConditionRQ gets the information from the user and asserts a proposition (flow-condition $condition $value) into the database

```
(if (and (flow-condition angle-of-attack $a)
         (≥ $a 0.087)) ; radians
    (flow-condition classification high-lift))
```

OB RULES

The knowledge contained by OB rules consists of basic CFD knowledge, such as what shape the outer boundary should be for a given configuration, and how far away the boundaries should be placed based on inflow conditions, boundary conditions, and desired solution accuracy.

```
(if (and (flow-condition speed-regime subsonic)
         (UNKNOWN (flow-condition speed-regime transonic))
         (dimensions-needed OB)
         (flow-condition steadiness steady)
         (flowsolver-info accuracy $a)
         (flowsolver-info boundary-conditions $b)
         (MaximumRangeNeededRQ $a $b))
    (and (suggested-distance-in-chords upstream OB 6.0)
         (suggested-distance-in-chords downstream OB 6.0)
         (suggested-distance-in-chords above 10.0)
         (suggested-distance-in-chords below 10.0)))

(if (and (flow-condition classification high-lift)
         (UNKNOWN (correction high-lift (stage $n)))
         (dimensions-needed OB)
         (suggested-distance-in-chords above OB $da)
         (suggested-distance-in-chords below OB $db)
         (* $da 1.5 $d1)
         (* $db 1.5 $d2)
    (and (correction high-lift (stage $n))
         (suggested-distance-in-chords above OB $d1)
         (suggested-distance-in-chords below OB $d2)))
```

ACTION RULES

These rules contain knowledge about what circumstances make certain zoning actions possible.

```
(if (and (ready-to-zone (stage $n))
         (zoning-environment $z (stage $n))
         (UNKNOWN (possible (operator connection/$op) (stage $n)))
         (UNKNOWN (Filled $z (stage $n))
         (UNKNOWN (Empty $z (stage $n)))
    (possible (operator connection/$op) (stage $n)))

(if (and (executable (operator connection/$op) (stage $n))
         (zoning-environment $z (stage $n))
         (UNKNOWN (filled $z (stage $n)))
         (coretype $z multiple (stage $n))
         (directly-contains $z $b1 (stage $n))
         (directly-contains $z $b2 (stage $n))
         (UNPROVABLE (IS $b1 $b2))
         (UNKNOWN (possible (operator connection/$op)
         (operand sibling-sibling/($b2 $b1)) (stage $n))))
(possible (operator connection/$op) (operand sibling-sibling/ ($b2 $b1))))
```

GENERATE RULES

GENERATE rules contain knowledge about how to execute zoning actions. The examples selected here show a chain of inference leading up to the construction of one or more new objects, beginning with the number and type of objects required and the curves needed to construct that object, proceeding to what curve parameters are needed for each curve and the translation of those parameters into the inputs needed by the curve generation function, and ending with completion requirements and new object type and containment information.

```
(if (and (chosen (operator shape-alteration/envelop) (operand self-virtual ($o1 $o2)))
         (GetNewNameFQ ZE $z))
    (and (operation-objects ($z) (stage $n))
         (object-needed $z (type created) (specification curve) (stage $n))))

(if (and (object-needed $z (type created) (specification curve) (stage $n))
         (UNKNOWN (mode automatic))
         (chosen (operator $or) (operand $od) (stage $n))
         (SpecifyCurveFQ $z $or $od $n $type)
         (MEMBER $type (new existing))
         (GetNewNameFQ C $c))
    (and (curve-needed $c (type $t) (object $z) (stage $n))
         (another-curve $c (stage $n))))

(if (and (curve-needed $c (type $t) (object $z) (stage $n))
         (UNKNOWN (mode automatic))
         (object-needed $z (type created) (specification curve) (step $n))
         (SpecifyBeginPointfQ $c $t $bp))
    (curve-parameter $c (begin-point $bp) (stage $n)))

(if (and (curve-parameter $c1 (begin-point ((curve $c2 $u) $p) (stage $n))
         (curve-needed $c1 (type new) (object $z) (stage $n))
    (new-connection ($c2 ($u ($c1 0.0))) (stage $n)))

(if (and (curve-parameters $c done (stage $n))
         (curve-parameter $c (begin-angle unspecified) (stage $n))
         (curve-parameter $c (end-angle $ea) (stage $n))
         (NUMBER $ea)
         (curve-parameter $c (end-derivative-weight $ew) (stage $n))
         (GetDirectionDerivativesFQ unspecified nil $ea $ew (($dx1 $dy1) ($dx2 $dy2))))
    (and (bc-types $c (free specified))
         (bc-values $c ($dx1 $dy1 $dx2 $dy2))))

(if (and (curve-completed $c (stage $n))
         (object-curves $z $oclist (stage $n))
         (MEMBER $c $oclist)
         (AllCurvesDoneRQ $oclist $n))
    (object-completed $z (stage $n)))
```

```
(if (and (object-needed $z2 (type resultant) (specification merge) (stage $n))
         (operation-objects ($z1 $z2) (stage $n))
         (object-completed $z1 (stage $n))
         (object-needed $z1 (type created) (specification curve) (stage $n))
         (chosen (operator $or) (operand parent-child/($p $c)) (stage $n))
         (GetFQ $c curvelist $clist1)
         (GetFQ $z1 curvelist $clist2)
         (ApplyFQ append ($clist1 $clist2) $clist3)
         (MergeCurvelistsFQ $clist3 ($clist4))
         (PutpropRQ $z2 $clist4 curvelist))
    (and (object-completed $z2 (stage $n))
         (zoning-environment $z2 (stage $n))
         (newobject $z2 (stage $n))))

(if (and (operation-completed (stage $n))
         (chosen (operator shape-alteration/envelop) (operand
         parent-child/($p $c))) (operation-objects ($z) (stage $n))
    (and (contains $p $z (stage $n))
         (contains $z $c (stage $n))))
```

PLAN RULES

These rules contain knowledge about the sequences of zoning actions and execution strategies that best zone a flow-field grouping in certain situations, about how to assemble subplans into plans, and how to tie in plans with the propositions that trigger zoning action selection.

```
(if (and prepare-subplans (stage 1))
         (configuration-grouping $ze $g ($b))
         (user-bias wake high)
         (construction-shape OB bullet)
         (original-body $b (stage 1))
         (flow-condition flow-direction left-to-right)
         (GetFQ $b type simple-airfoil)
         (GetFQ $b leading-edge (blunt $le))
         (GetFQ $b trailing-edge (sharp $te))
         (GetNewNameFQ SP $sp)
    (subplan $sp $g
         (primary (((operator connection/bridge) (operand parent-child/ ($p $c)) (strategy de-
         generate-zone) (strategy follow-wake) (strategy identify (original-body $b)))))
         (completion (((operator identification/identify) (operand region/($z1))) (topology-im-
         position $z2 (strategy impose-C)))))

(if (and (ready-to-plan (stage 1))
         (configuration-groups OB 1)
         (configuration-grouping OB $gname $g)
         (subplan $sp $gname (primary $pl) (completion $cl))
         (GetNewNameFQ P $p)
         (ApplyFQ append ($pl $cl) $pend))
    (plan $p (pending $pend) (executed nil) (stage 1)))

(if (and (operator-executables (stage $n))
        (mode automatic)
        (plan $pl (pending (((operator $or) (operand $od)) . $r)) (executed $el) (stage $n))
        (possible (operator $or) (stage $n))
        (UNKNOWN (undesirable (operator $or) (stage $n)))
    (executable (operator $or) (stage $n)))
```

AUTO RULES

These rules contain knowledge about how to execute the chosen zoning action using the execution strategies suggested by the chosen plan.

```
 (if (and (object-needed $z (type created) (specification curve) (stage $n))
          (mode automatic)
          (strategy degenerate-zone (stage $n))
          (chosen (operator connection/bridge) (operand $od) (stage $n))
          (GetNewNameFQ C $c))
     (and (object-curves $z ($c) (stage $n))
          (curve-needed $c (type new) (object $z) (stage $n))))

 (if (and (curve-needed $c (type new) (object $z) (stage $n))
          (mode automatic)
          (strategy (follow-base-protrusion ($p1)) (stage $n))
          (object-needed $z (type created) (specification curve) (stage $n))
          (object-curves $z ($c $c2) (stage $n))
          (chosen (operator connection/bridge) (operand parent-child/ ($p $ch)) (stage $n))
          (ApplyFQ get_joins ($ch $p1) ((($p1 front-end $l1) ($p2 back-end (at-corner top-side))
          )))
          (GetFQ $p1 back-end (base ($topbase $bottombase))))
     (curve-parameter $c (begin-point $bottombase) (stage $n)))
```

APPENDIX D
USER BIAS PROFILE PARAMETER MEASUREMENT FUNCTIONS

1. Zonal Simplicity

$$simplicity = 0.5 \frac{\#\text{zones}}{\#\text{body sides}} + \frac{\#\text{zonal interfaces}}{\#\text{zones}}$$

(0.5 is a relative function weight)

2. Zone Corner Skewness

$$skewness = \frac{1}{\#\text{ zones}} \sum_{\text{all zones}} \quad \sum_{\text{corner vertices}} \frac{|90^\circ - \text{corner angle}|}{45^\circ}$$

3. Zone Side Smoothness

$$smoothness = \frac{1}{\#\text{ zones}} \sum_{\text{all zones}} \quad \sum_{\text{non-corner vertices}} \frac{|180^\circ - \text{vertex angle}|}{90^\circ}$$

4. Zone Side Mapping Disparity

$$mapping = 0.15 \frac{1}{\#\text{ zones}} \sum_{\text{all zones}} \left((ratio1 \times ratio2) - 1.0 \right)$$

(0.15 is a relative function weight)

ratio1 is the ratio (≥ 1) of the lengths of one pair of opposite sides of a zone, and ratio2 is the ratio (≥ 1) of the length of the remaining pair of opposite zone sides

5. Potential Grid Point Efficiency

$$efficiency = \frac{1}{\#\text{ body sides}} \sum_{\text{all body sides}} \left(1.0 - \frac{\text{length of body curves}}{\text{length of all curves}}\right)$$

6. Zone Side Orthogonality at Body Surfaces

$$orthogonality = \frac{1}{\#\text{ bodies}} \sum_{\text{all zones}}$$

$$\left(\sum_{\text{body side corner vertices}} \frac{|90^\circ - \text{corner angle}|}{45^\circ} \right.$$

$$\left. + \sum_{\text{body-nonbody noncorner vertices}} \frac{|180^\circ - \text{vertex angle}|}{90^\circ} \right)$$

7. Surface Versus Field Quantities

$$surface - vs - field = efficiency \times \frac{\#\text{ body sides}}{\#\text{ bodies}}$$

8. Wake Resolution

$$wake = \frac{1}{\#\text{ trailing-edge points}} \left(\sum_{\text{all bodies}} \#\text{ trailing-edge points per body} \right)$$

$$- \quad (\#\text{ wake curves})$$

9. Zone Tuple Points

$$tuples = \frac{1}{\#\ \text{zones}} \sum_{\text{all curve connections}} (\text{connections between 3 or more zones})$$

10. Singularities at Body Surfaces

$$singularities = \frac{1}{\#\ \text{bodies}} \sum_{\text{all zones}}$$

$$\sum_{\text{body-nonbody noncorner vertices}} \frac{|180^\circ - \text{vertex angle}|}{90^\circ}$$

11. Zone/Body Intersections

$$intersections = \frac{1}{\#\ \text{bodies}} \sum_{\text{all zones}} \sum_{\text{body-nonbody corner vertices}} 1$$

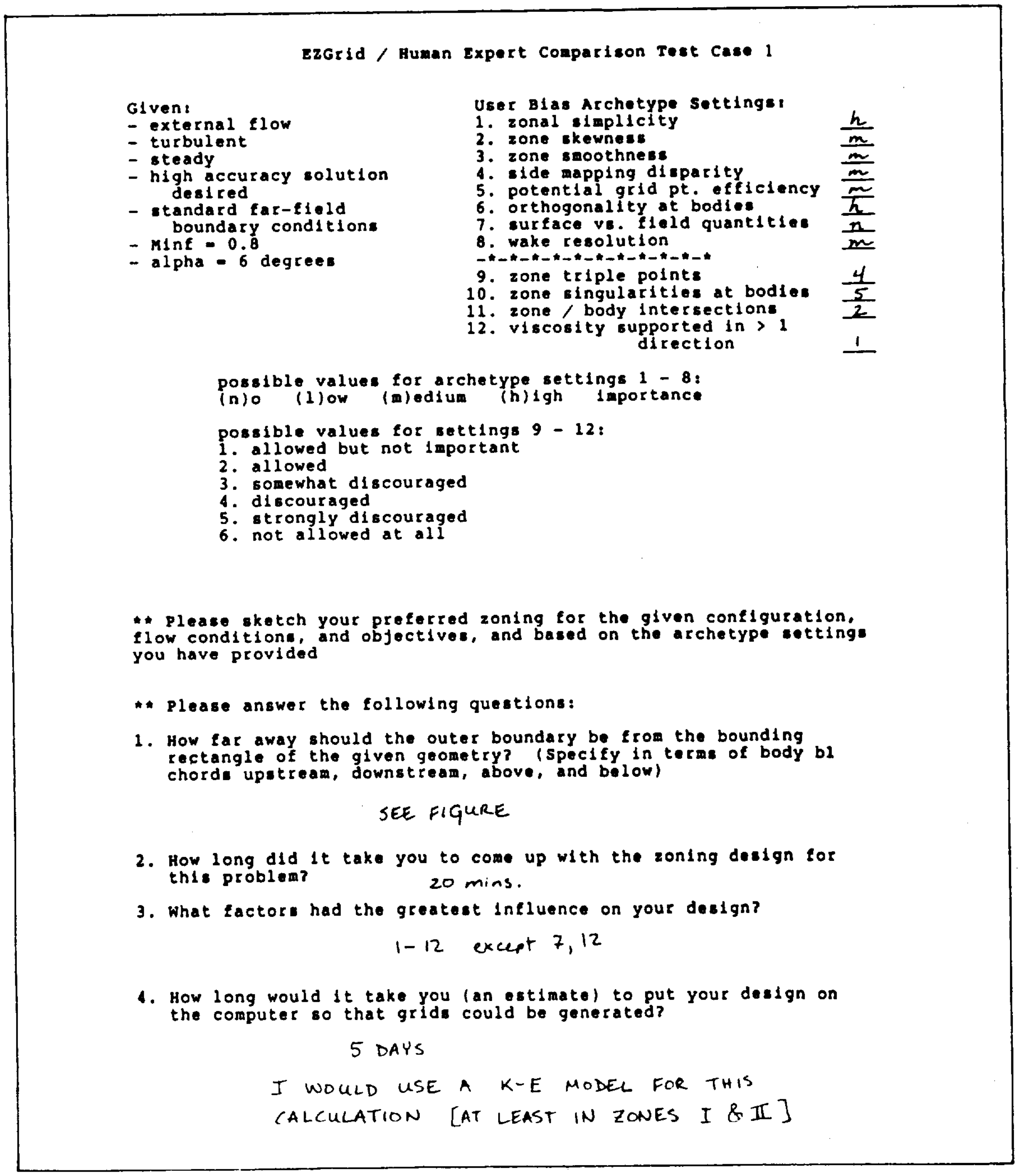
12. Viscosity in More Than One Direction

$$viscosity = \frac{1}{\#\ \text{bodies}} \sum_{\text{all zones}}$$

$$(\#\ \text{coordinate directions having at least one body curve} - 1)$$

APPENDIX E
COMPARISON STUDY TEST CASE PROBLEM STATEMENTS

This appendix contains copies of the problem statement worksheets given to the zoning expert and returned with the expert's responses for the two test cases described in section 5.3. The expert was given the flow conditions and geometry, and was asked to tune the user bias profile and sketch a zoning for each case.



EZGrid / Human Expert Comparison Test Case 1

Given:
- external flow
- turbulent
- steady
- high accuracy solution desired
- standard far-field boundary conditions
- Minf = 0.8
- alpha = 6 degrees

User Bias Archetype Settings:

1. zonal simplicity	h
2. zone skewness	m
3. zone smoothness	m
4. side mapping disparity	m
5. potential grid pt. efficiency	m
6. orthogonality at bodies	h
7. surface vs. field quantities	n
8. wake resolution	m
-*-*-*-*-*-*-*-*-*-*-*-*	
9. zone triple points	4
10. zone singularities at bodies	5
11. zone / body intersections	2
12. viscosity supported in > 1 direction	1

possible values for archetype settings 1 - 8:
(n)o (l)ow (m)edium (h)igh importance

possible values for settings 9 - 12:
1. allowed but not important
2. allowed
3. somewhat discouraged
4. discouraged
5. strongly discouraged
6. not allowed at all

** Please sketch your preferred zoning for the given configuration, flow conditions, and objectives, and based on the archetype settings you have provided

** Please answer the following questions:

1. How far away should the outer boundary be from the bounding rectangle of the given geometry? (Specify in terms of body bl chords upstream, downstream, above, and below)

   SEE FIGURE

2. How long did it take you to come up with the zoning design for this problem? 20 mins.

3. What factors had the greatest influence on your design?

   1 - 12 except 7, 12

4. How long would it take you (an estimate) to put your design on the computer so that grids could be generated?

   5 DAYS

I WOULD USE A K-E MODEL FOR THIS CALCULATION [AT LEAST IN ZONES I & II]

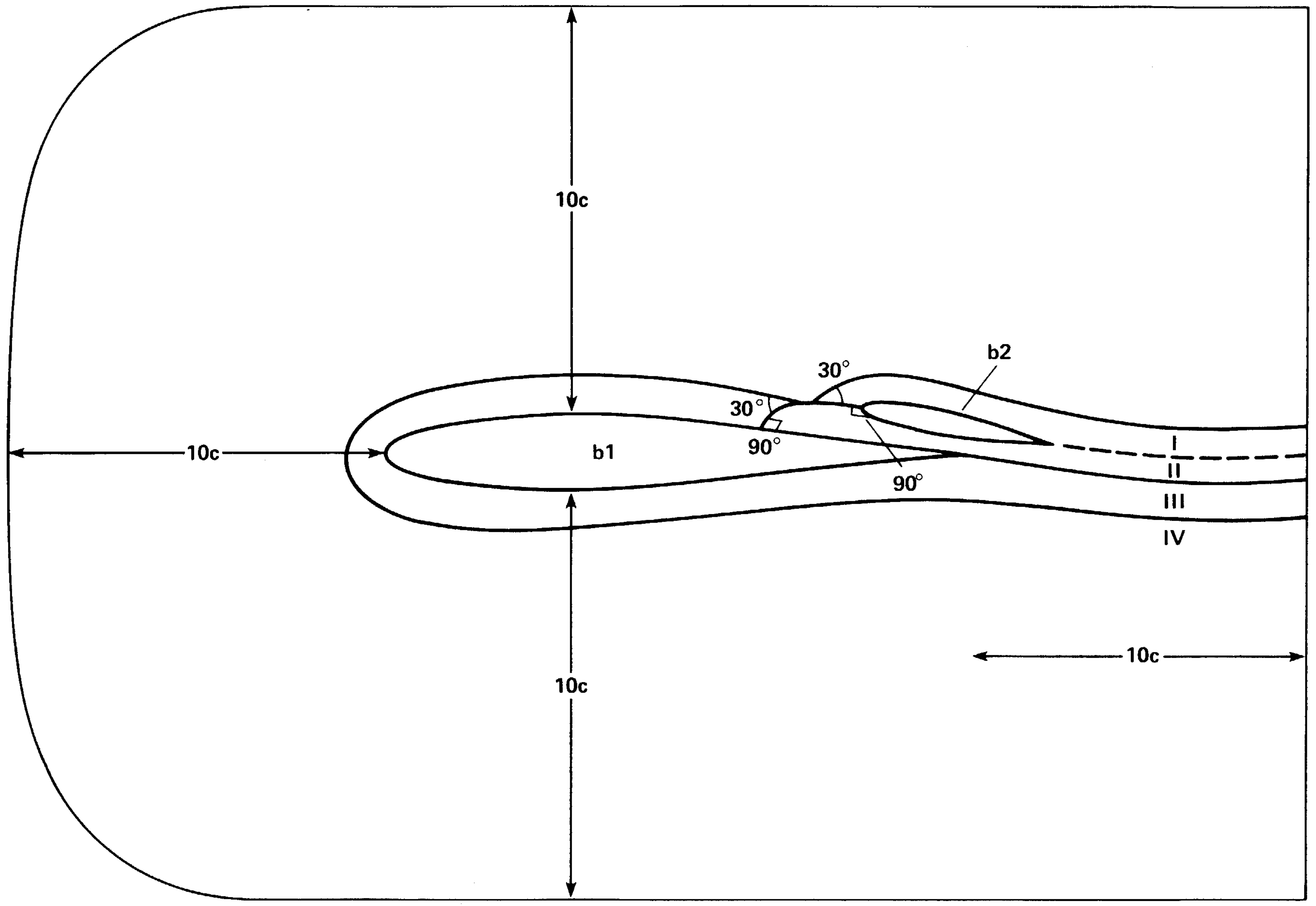


Figure E.1. – Sketch of expert zoning for test case 1.

EZGrid / Human Expert Comparison Test Case 2

Given:
- external flow
- turbulent
- steady
- high accuracy solution desired
- standard far-field boundary conditions
- Minf = 0.8
- alpha = 6 degrees

User Bias Archetype Settings:

1. zonal simplicity	h
2. zone skewness	m
3. zone smoothness	m
4. side mapping disparity	m
5. potential grid pt. efficiency	m
6. orthogonality at bodies	h
7. surface vs. field quantities	n
8. wake resolution	m
-*-*-*-*-*-*-*-*-*-*-*	
9. zone triple points	4
10. zone singularities at bodies	5
11. zone / body intersections	4
12. viscosity supported in > 1 direction	1

possible values for archetype settings 1 - 8:
(n)o (l)ow (m)edium (h)igh importance

possible values for settings 9 - 12:
1. allowed but not important
2. allowed
3. somewhat discouraged
4. discouraged
5. strongly discouraged
6. not allowed at all

** Please sketch your preferred zoning for the given configuration, flow conditions, and objectives, and based on the archetype settings you have provided

** Please answer the following questions:

1. How far away should the outer boundary be from the bounding rectangle of the given geometry? (Specify in terms of body bl chords upstream, downstream, above, and below)

   SEE FIGURE

2. How long did it take you to come up with the zoning design for this problem? 15 mins.

3. What factors had the greatest influence on your design?

   1-12 except 7, 12

4. How long would it take you (an estimate) to put your design on the computer so that grids could be generated?

   3 DAYS

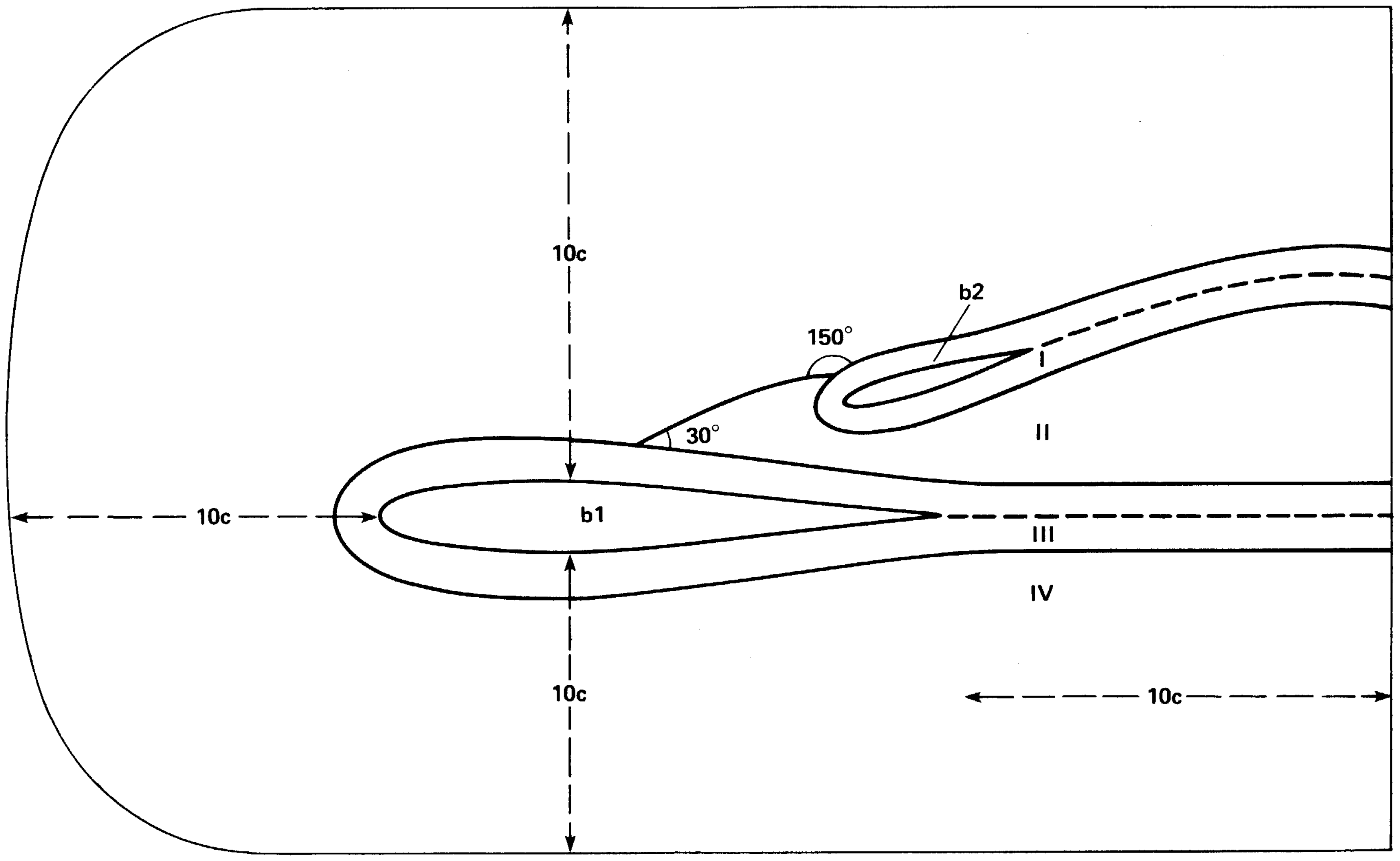


Figure E.2. – Sketch of expert zoning for test case 2.

REFERENCES

Allmaras, S. R.; and Baron, J. R.: Embedded Mesh Solutions of the 2-D Euler Equations: Evaluation of Interface Formulations. AIAA Paper 86-0509, Jan. 1986.

Andrews, A. E.: Progress and Challenges in the Application of Artificial Intelligence to Computational Fluid Dynamics. AIAA J., vol. 26, no. 1, Jan. 1988, pp. 40-46.

Brady, M.; and Haruo, A.: Smoothed Local Symmetries and Their Implementation. Intern. J. Robotics Res., vol. 3, no. 3, Fall 1984, pp. 36-61.

Chaderjian, N. M.: Navier-Stokes Simulation of Transonic Wing Flow Fields Using a Zonal Grid Approach. NASA-TM 100039, 1988.

Chapman, D. R.: Trends and Pacing Items in Computational Aerodynamics. Lecture Notes in Physics. vol. 141, W. C. Reynolds and R. W. MacCormack, eds., Springer-Verlag (New York), 1980, pp. 1-12.

Conner, R. S.; Purdon, D. J.; and Wamsley, F. K.: PAN AIR Consultation Using Expert System Techniques. AIAA Paper 85-4094, Oct. 1985.

Conner, R. S.; Purdon, D. J.; and Wamsley, F. K.: PAN AIR Knowledge System. AIAA Paper 86-0239, Jan. 1986.

Cordova, J. Q.; and Barth, T. J.: Grid Generation for General 2-D Regions Using Hyperbolic Equations. AIAA Paper 85-0520, Jan. 1988.

Dannenhoffer, J. F. III; and Baron, J. R.: Robust Grid Adaptation for Complex Transonic Flows. AIAA Paper 86-0495, Jan. 1986.

Dannenhoffer, J. F. III; and Baron, J. R.: A Hybrid Expert System for Complex CFD Problems. AIAA Paper 87-1111, June 1987.

Davis, R.: Expert Systems: Where Are We? And Where Do We Go From Here? The AI Magazine, Spring 1982, pp. 3-22.

Eriksson, L. E.; and Rai, M. M.: A Stability Analysis of Various Patched Grid Interface Conditions for Hyperbolic Equations. Flygtekniska Försöksanstalten, FFA Report 144, Stockholm, Sweden, 1988.

Flores, J.; Chaderjian, N. M.; and Sorenson, R. L.: Simulation of Transonic Viscous Flow Over a Fighter-Like Configuration Including Inlet. AIAA Paper 87-1199, June 1987.

Genesereth, M. R.; Greiner, R.; Grinberg, M. R.; and Smith, D. E.: The MRS Dictionary. Stanford Heuristic Programming Project Report No. HPP-80-24, Jan. 1984.

Hayes-Roth, F.; Waterman, D. A.; and Lenat, D. B., eds: Building Expert Systems. Teknowledge Series in Knowledge Engineering, vol. 1, Addison-Wesley Publishing Co. (Reading, Massachusetts), 1983.

Hayes-Roth, F.: Knowledge-Based Expert Systems. IEEE Computer, Oct. 1984, pp. 263-273.

Hessenius, K. A.; and Rai, M. M.: Applications of a Conservative Zonal Scheme to Transient and Geometrically Complex Problems. Computers and Fluids, vol. 14, no. 1, 1986, pp. 43-58.

Hessenius, K. A.: Conservative Zonal Schemes for Patched Grids in Two and Three Dimensions. Ph.D. Dissertation, Mechanical Engineering Department, Stanford University, 1986.

Kernighan, B. W.; and Ritchie, D. M.: The C Programming Language. Prentice-Hall, Inc., 1978.

Klopfer, G. H.: Analysis of a Finite Difference Grid. Nielsen Engineering & Research, Inc. Report TR 267, Feb. 1982.

Kutler, P.; and Mehta, U. B.: Computational Aerodynamics and Artificial Intelligence. NASA TM-85994, 1984.

Kutler, P.: A Perspective of Theoretical and Applied Computational Fluid Dynamics. AIAA J., vol. 23, no. 3, 1985, pp. 328-341.

Kutler, P.: A Perspective of Computational Fluid Dynamics. NASA TM-88246, May 1986.

Kutler, P.; Steger, J. L.; and Bailey, F. R.: Status of Computational Fluid Dynamics in the United States. AIAA Paper 87-1135-CP, June 1987.

Lenat, D. B.; and McDermott, J.: Building Expert Systems: Part 1 - Fundamentals. American Association for Artificial Intelligence National Conference on Artificial Intelligence, Tutorial No. 4, Aug. 1984.

McCroskey, W. J.; Kutler, P.; and Bridgeman, J. O.: Status and Prospects of Computational Fluid Dynamics for Unsteady Transonic Viscous Flows. NASA TM-86018, USAAVSCOM TM-84-A-8, 1984.

Nicklaus, D. J.; Tong, S. S.; and Russo, C. J.: Engineous: A Knowledge Directed Computer Aided Design Shell. Proceedings of the 3rd Conference on Artificial Intelligence Applications, Computer Society of the IEEE, Feb. 1987, pp. 308-314.

Nilsson, N. J.: Principles of Artificial Intelligence. Tioga Publishing Co. (Palo Alto, California), 1980.

Oskam, B.; and Huizing, G. H.: Flexible Grid Generation for Complex Geometries in Two Space Dimensions Based on Variational Principles. AGARD Conference Proceedings No. 412, NATO Advisory Group for Aerospace Research & Development, Apr. 1986.

Pearl, J.: Heuristics, Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Publishing Co. (Reading, Massachusetts), 1984.

Rai, M. M.: Navier-Stokes Simulations of Rotor-Stator Interaction Using Patched and Overlaid Grids. AIAA Paper 85-1519-CP, July 1985.

Rai, M. M.: A Conservative Treatment of Zonal Boundaries for Euler Equation Calculations. J. Computational Phys., vol. 62, no. 2, 1986, pp. 472-503.

Rich, E.: Artificial Intelligence. McGraw-Hill Book Co., Inc., 1983.

Roache, P. J.; and Steinberg, S.: Symbolic Manipulation and Computational Fluid Dynamics. AIAA J., vol. 22, no. 10, Oct. 1984, pp. 1390-1394.

Russell, S.: The Compleat Guide to MRS. Department of Computer Science, Stanford University, Report No. STAN-CS-85-1080, June 1985.

Schuster, D. M.; and Birckelbaw, L. D.: Numerical Computation of Viscous Flowfields about Multiple Component Airfoils. AIAA Paper 85-0167, Jan. 1985.

Steinberg, S.; and Roache, P. J.: A Toolkit of Symbol Manipulation Programs for Variational Grid Generation. AIAA Paper 86-0241, Jan. 1986.

Steinbrenner, J. P.: GRIDGEN2D (Interactive Elliptic Surface Grid Generation) User's Manual. General Dynamics, Fort Worth Division, CFD Report No. 063-4-8601, June 1986.

Steinbrenner, J. P.; Karmen, S. L.; and Chawner, J. R.: Generation of Multiple Block Grids for Arbitrary 3D Geometries. AGARDograph No. 309, Yoshihara, H., ed., North Atlantic Treaty Organization Advisory Group for Aerospace Research and Development, 1988, pp. 40-55.

Thompson, J. F.; Warsi, Z. U. A.; and Mastin, C. W.: Boundary-Fitted Coordinate Systems for Numerical Solution of Partial Differential Equations—A Review. J. Computational Phys., vol. 47, no. 1, 1982, pp. 1-108.

Thompson, J. F.; and Steger, J. L.: Three-Dimensional Grid Generation for Complex Configurations – Recent Progress. AGARDograph No. 309, Yoshihara, H., ed., North Atlantic Treaty Organization Advisory Group for Aerospace Research and Development, 1988, pp. 1-14.

Tong, S. S.: Design of Aerodynamic Bodies Using Artificial Intelligence/Expert System Technique. AIAA Paper 85-0112, Jan. 1985.

Tong, S. S.: Coupling Artificial Intelligence and Numerical Computation for Engineering Design. AIAA Paper 86-0242, Jan. 1986.

Walters, R. W.; Thomas, J. L.; and Switzer, G. F.: Aspects and Applications of Patched Grid Calculations. AIAA Paper 86-1063, May 1986.

Wardlaw, A. B. Jr.; Priolo, F. J.; and Solomon, J. M.: Multiple-Zone Strategy for Supersonic Missiles. J. Spacecraft & Rockets, vol. 24, no. 4, July-August 1987, pp. 377-384.

Weatherill, N. P.; and Shaw, J. A.: Component Adaptive Grid Generation for Aircraft Configurations. AGARDograph No. 309, Yoshihara, H., ed., North Atlantic Treaty Organization Advisory Group for Aerospace Research and Development, 1988, pp. 29-39.

Wilensky, R.: LISPcraft. W.W. Norton & Co. (New York), 1984.

Winston, P. H.; and Horn, B. K. P.: Lisp. Addison-Wesley Publishing Co. (Reading, Massachusetts), 1981.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-101072	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A Knowledge-Based Approach to Automated Flow-Field Zoning for Computational Fluid Dynamics		5. Report Date April 1989
		6. Performing Organization Code
7. Author(s) Alison Andrews Vogel		8. Performing Organization Report No. A-89044
		10. Work Unit No. 505-60-01
9. Performing Organization Name and Address Ames Research Center Moffett Field, CA 94035		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001		14. Sponsoring Agency Code

15. Supplementary Notes

Point of Contact: Alison Andrews Vogel, Ames Research Center, MS 258-1
Moffett Field, CA 94035 (415) 694-5396 or FTS 464-5396

16. Abstract

The goals of the present research are to (1) lay the foundation for an automated three-dimensional zonal grid generation capability for computational fluid dynamics through the development of a demonstration computer program capable of automatically zoning the flow field of representative two-dimensional (2-D) aerodynamic configurations, and (2) determine the applicability of a knowledge-based programming approach to the domain of flow-field zoning. Several aspects of flow-field zoning make the application of knowledge-based techniques challenging: the need for perceptual information, the role of individual bias in the design and evaluation of zonings, and the fact that the zoning process has been modeled as a constructive, design-type task (for which there are relatively few examples of successful knowledge-based systems in any domain). Engineering solutions to the problems arising from these aspects have been developed, and a demonstration system has been implemented which can design, generate, and output flow-field zonings for representative 2-D aerodynamic configurations.

17. Key Words (Suggested by Author(s)) Computational fluid dynamics; Artificial intelligence, Grid generation; Domain decomposition; Expert systems	18. Distribution Statement Unclassified-Unlimited. Subject Category – 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	162	A08