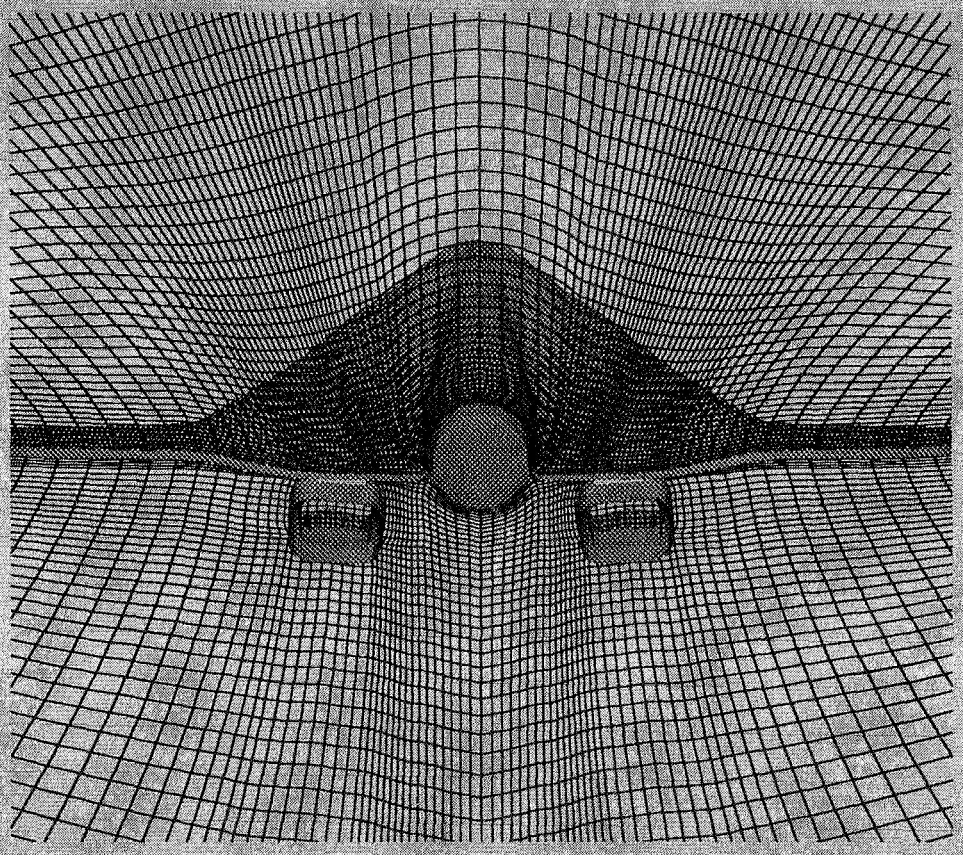
NASA Conference Publication 3143

Software Systems for Surface Modeling and Grid Generation



Proceedings of a workshop held at
Langley Research Center
Hampton, Virginia
April 28–30, 1992

NASA

NASA Conference Publication 3143

Software Systems for Surface Modeling and Grid Generation

Edited by
Robert E. Smith
Langley Research Center
Hampton, Virginia

Proceedings of a workshop sponsored by the
National Aeronautics and Space Administration,
Washington, D.C., and held at
Langley Research Center
Hampton, Virginia
April 28–30, 1992

NASA
National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Program
1992

PREFACE

The first NASA meeting devoted to the subject of "grid generation" was a workshop held at the Langley Research Center in 1980. The meeting was entitled "Numerical Grid-Generation Techniques" and the proceedings are found in NASA CP 2166. The pros and cons of differential, algebraic and conformal techniques particularly amenable to computational fluid dynamics (CFD) were presented and debated. At the time, numerical grids were obtained using mainframe or minicomputers, and interactive time-shared graphics terminals were the main communications link between users and their software. Most of the applications were two-dimensional or quasi three-dimensional.

In the following decade, considerable progress was made in the development and application of grid generation techniques and associated software. The ability to deal with complex three-dimensional domains and grid adaptivity was on the forefront of the progress. Unstructured grid generation, in addition to structured approaches, became viable for many CFD applications. Advancements in computer hardware and systems software, particularly high-performance graphics workstations and multi-windowed operating systems, pointed out the directions for constantly improving grid generation software.

In the late 1980's it became apparent that the biggest bottleneck in the creation of grids about complex geometries was modeling of the boundary surfaces and transfer of modeling information to grid generator software. Surface modeling and its associated computer-aided design (CAD) functions are critical to the successful application of CFD to realistic aerodynamic configurations. Consequently, surface modeling and grid generation are regarded as companion technologies for the application of CFD.

In December 1989 the NASA Workshop on Future Directions in Surface Modeling and Grid Generation was held at the NASA Ames Research Center. The purpose of the workshop was to assess U.S. capabilities in surface modeling and grid generation and to take steps to improve the focus and pace of these disciplines within NASA. The workshop report is NASA CP-10092. One of the recommendations from this report that has been implemented is the formation of a NASA Surface Modeling and Grid Generation Steering Committee. In addition to coordinating efforts within NASA, the committee is interested in improving the level of cooperation among the primary U.S. surface modeling/grid generation development activities. One of the first results from the steering committee is the plan for the current workshop.

Papers for the current workshop were solicited from government laboratories, universities and industries. The papers and the associated software demonstrations chosen for presentation represent the "state-of-the-art" in surface-modeling and grid-generation software systems. The software systems are embedded in the enabling technology of "cutting-edge" high-performance graphics workstations. In addition, several papers on innovative grid-generation techniques are included.

Robert E. Smith
Langley Research Center
Workshop Chairman

ORGANIZATION COMMITTEE

Larry D. Birckelbaw
Phone: 415 604-4468
e-mail: birckel@prandtl.nas.nasa.gov
NASA Ames Research Center

Matthew W. Blake
Phone: 415 604-4978
e-mail: blake@wk60.nas.nasa.gov
NASA Ames Research Center

Yung K. Choo
Phone: 216 433-5868
e-mail: sachoo@avelon.lerc.nasa.gov
NASA Lewis Research Center

Dennis L. Huff
Phone: 216 433-3913
e-mail: tohuff@bullwinkle.lerc.nasa.gov
NASA Lewis Research Center

Fred W. Martin
Phone: 713 483-4698
e-mail: martin @ecfc.jsc.nasa.gov
NASA Johnson Space Center

Pamela F. Richardson
Phone: 202 453-9857
e-mail: pfrichardson@oaetqm.hq.nasa.gov
NASA Headquarters

Robert E. Smith
Phone: 804 864-5774
e-mail: bobs@uxv.larc.nasa.gov
NASA Langley Research Center

William R. Van Dalsem
Phone: 415 604-4469
e-mail: vandal@prandtl.nas.nasa.gov
NASA Ames Research Center

Robert P. Weston
Phone: 804 864-2149
e-mail: weston@amb2.larc.nasa.gov
NASA Langley Research Center

Robert W. Williams
Phone: 205 544-3998
e-mail: bobw@tyrell.msfc.nasa.gov.
NASA Marshall Space Flight Center

Workshop Chairman
Robert E. Smith

Workshop Administrator
Marie S. Noland
Phone: 804 864-5779
e-mail: shetley@eagle.larc.nasa.gov

Coordinator of Demonstrations
Eric L. Everton
Phone: 804 864-5778
e-mail: everton@eagle.larc.nasa.gov

PROGRAM AND CONTENTS
Tuesday, April 28

Wednesday, April 29

*Presentation not included in text.

Thursday, April 30

4:30 **Grid Quality Estimators for Mesh Improvement and Adaptive Gridding Applications** . . . . . *
Kenneth Weems, Science Applications International Corp.

5:00 **ADJOURN MEETING**

THE NASA-IGES GEOMETRY DATA EXCHANGE STANDARD*

Matthew W. Blake
NASA Ames Research Center
Moffett Field, CA

Jin J. Chou
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA

Patricia A. Kerr
NASA Langley Research Center
Hampton, VA

Scott A. Thorp
NASA Lewis Research Center
Cleveland, OH

SUMMARY

This paper describes the the data exchange efforts and plans supported by the NASA Steering Committee for Surface Modeling and Grid Generation. Current methods for geometry data exchange between Computer Aided Design (CAD) systems and NASA Computational Fluid Dynamics (CFD) analysis systems are tedious and induce errors. A Geometry Data Exchange Standard is proposed, utilizing a subset of an existing national standard titled Initial Graphics Exchange Standard (IGES) (ref. 1). Future plans for Data Exchange Standardization include all aspects of CFD data. Software systems to utilize this NASA-IGES Geometry Data Exchange Specification are under development.

INTRODUCTION

The geometry data received by NASA scientists for analysis and modification is currently supplied in numerous formats which often require hundreds of hours of manipulation to achieve a format capable of being utilized by analysis software. This modified data set usually has lost a level of accuracy from the original data and often may not maintain the design intent of the original data as developed on the original designer's system.

---

*Dr. Chou's participation is provided under NASA Contract NAS 2-12961

In the spring of 1991 the NASA Steering Committee for Surface Modeling and Grid Generation (ref. 2) formed a Geometry Data Exchange Subcommittee of technical personnel from the NASA Ames, Langley, and Lewis Research Centers to develop a standard method for transferring complex vehicle geometries between various software systems. Following an analysis of existing and proposed standards, the Geometry Subcommittee selected the existing IGES format as the basis for a NASA standard. In the United States, the IGES format is the most widely used product data exchange method for complex geometries. The latest version of the IGES specification (version 5.1) provides an adequate set of geometric entities to cover the current data transfer needs for CFD research. A subset of the IGES document was selected and a draft NASA Technical Specification was released in September of 1991 titled "NASA Geometry Data Exchange Specification utilizing IGES" (ref. 3) commonly referred to as the NASA-IGES Geometry Data Exchange Specification or simply NASA-IGES. A second draft of the document should be available in early spring of 1992.

The NASA-IGES Geometry Data Exchange Specification is meant to provide a means for rapid and accurate geometry data transfer for engineering analysis utilizing Computational Fluid Dynamics and related methods. Specifically, it is to provide a method for the transfer of data between CAD systems and Grid Generation Software. This will greatly improve the ability of scientists and engineers to rapidly proceed with analysis of a wide variety of vehicle geometries.

This document explains the IGES/PDES Organization (IPO), describes the contents of the NASA-IGES Geometry Data Exchange Specification, discusses the NASA Steering Committee for Surface Modeling and Grid Generation future plans for Data Exchange Standardization, and describes some of the software under development to utilize the NASA-IGES Geometry Data Exchange Specification.

IGES / PDES ORGANIZATION

The IGES/PDES Organization (IPO) is a body of volunteers from industry, government, and academia who are dedicated to the development and implementation of world-wide standards for the digital representation and communication of product data. The IPO efforts focus in two main areas: IGES and PDES.

In the late 1970s and early 1980s The Initial Graphics Exchange Specification (IGES) was designed and developed as a neutral data format to allow the digital exchange of data among incompatible CAD systems. IGES Version 5.0 has been accepted by the American National Standards Institute (ANSI). Version 5.1 was released in September of 1991.

The Product Data Exchange using STEP (PDES) effort is the United States activity which supports the development of the International Standard referred to as the Standard for the Exchange of Product Model Data (STEP). The first version of STEP is yet to be released.

The IPO is headquartered at the National Institute of Standards and Technology (NIST) in Washington D. C. The IPO has regular quarterly meetings. IPO membership is in excess of 600 persons.

For further information on the IPO, see reference 4.

SPECIFICATION DESCRIPTION

The NASA-IGES Geometry Data Exchange Specification differs from standard IGES in one important aspect. Standard IGES files are typically very large and are an attempt to represent every aspect of product data. NASA-IGES is intended to provide a very small subset of this capability. NASA-IGES is designed solely to provide a data exchange format for transferring complex vehicle geometries between software systems. Very little other product data is included in the NASA-IGES Specification. This minimal approach should provide small organizations and individual research scientists with a data transfer method they can afford to utilize in their particular software. The NASA-IGES Specification also will provide software developers with a data transfer method that NASA has committed to utilizing for the foreseeable future. This continuity should allow greater communication between future software packages.

The NASA-IGES Geometry Data Exchange Specification includes several geometry entities and a few other entities. The geometry entities include simple shapes such as circles and conics and a complex representation method utilizing Rational B-Splines. Additional geometry related entities are included to provide information such as coordinate system transformation. The non-geometric entities included were the minimum deemed necessary to convey grouping and visualization information about the object as well as a brief description capability.

The following is a list, ordered by IGES Entity type number, of the specific entities included in the NASA-IGES Specification. For a description of each entity see either the NASA-IGES Geometry Data Exchange Specification or the IGES Version 5.1 document.

IGES Entity Type Number	Entity Name
Entity 0:	Null Entity
Entity 100:	Circular Arc
Entity 102:	Composite Curve
Entity 104:	Conic Arc
Entity 106:	Copious Data
Entity 108:	Plane
Entity 110:	Line
Entity 116:	Point
Entity 124:	Transformation Matrix
Entity 126:	Rational B-Spline Curve
Entity 128:	Rational B-Spline Surface
Entity 141:	Boundary
Entity 142:	Curve on a Parametric Surface
Entity 143:	Bounded Surface
Entity 212:	General Note
Entity 314:	Color Definition
Entity 402:	Associativity Instance
Entity 406, Form 1:	Definition Levels
Entity 406, Form 15:	Name

It is expected that most grid generation and analysis software that utilizes NASA-IGES data will represent the geometry internally as Non-Uniform Rational B-Splines (NURBS). Because of this, it is desired that CAD systems be able to provide geometry in a NURBS based form (entities 126 and 128) rather than as a Circular Arc, Line, Copious Data, Plane, Point, or Conic Arc. Also, Entity 104 (Conic Arc) was included so simple CAD systems utilizing only this geometric modeling capability could be utilized and to allow development of simple shapes when desired for a specific purpose. Due to accuracy problems in the method IGES uses to represent conics, it is highly desirable to always convert conics to B-Splines before generating a NASA-IGES file.

Existing CFD grid generation programs often input a mesh of points to represent the geometry. Because of this, the geometry data is often represented solely as a mesh of points. This mesh is often redistributed several times before achieving the desired grid for CFD. This meshing and re-meshing of the vehicle surface geometry induces and propagates errors. For this reason, the NASA-IGES Specification emphasizes the use of the original design curves and surfaces with continuous derivatives as the correct way to represent geometry rather than utilizing a mesh of discrete points. The intent is to transfer the highest order surface representation available.

To summarize, it is desirable to represent all geometric objects utilizing the following entities:

Entity 126:	Rational B-Spline Curve
Entity 128:	Rational B-Spline Surface
Entity 141:	Boundary
Entity 142:	Curve on a Parametric Surface
Entity 143:	Bounded Surface

The following geometry related entities are also needed:

Entity 102:	Composite Curve
Entity 124:	Transformation Matrix

The following geometric entities are allowed but discouraged for most purposes:

Entity 100:	Circular Arc
Entity 110:	Line
Entity 106:	Copious Data
Entity 108:	Plane
Entity 116:	Point

The following geometric entity is allowed but discouraged at all times:

Entity 104:	Conic Arc

The following non-geometric entities are available:

Entity 0:	Null Entity
Entity 212:	General Note
Entity 314:	Color Definition
Entity 402:	Associativity Instance
Entity 406, Form 1:	Definition Levels
Entity 406, Form 15:	Name

FUTURE PLANS

The NASA Steering Committee for Surface Modeling and Grid Generation has started a coordinated effort to standardize the exchange of geometry, grid, and solution data used in the analysis of computational aerophysics problems. The initial purpose is to provide rapid and accurate geometry data exchange between CAD systems and grid generation software. The long term goal is to provide a rapid, accurate, and stable data exchange method for use in comprehensive engineering design and analysis in support of such fields as Computational Fluid Dynamics (CFD), Computational Electro-Magnetics (CEM), and Finite Element Method (FEM) Structural Analysis.

The following timetable has been agreed to by the NASA Steering Committee for Surface Modeling and Grid Generation. Several items relate to national standards organization efforts, and therefore the time frame for completion is dependent upon performance by those organizations.

NASA-wide consensus on the NASA-IGES Geometry Data Exchange Specification:	- 4th quarter 1992
Approval of an IGES Application Protocol following the NASA_IGES Geometry Data Exchange Specification format:	- 4th quarter 1993
Approval of a STEP Application Protocol providing the geometry data exchange capability desired for CFD:	- estimate 1993 - 1994
NASA-wide consensus on a NASA specification describing grid and solution data for structured and unstructured methods:	- 2nd quarter 1993
Approval of an IGES Application Protocol for grid and solution data for structured and unstructured methods:	- 2nd quarter 1994
Approval of a STEP Application Protocol for grid and solution data for structured and unstructured methods:	- estimate 1994 - 1995

SOFTWARE UTILIZING NASA-IGES

The three NASA Research Centers are currently developing grid generation software capable of utilizing geometry data in the form specified in the NASA-IGES Geometry Data Exchange Specification. In direct support of this effort, personnel at the Numerical Aerodynamic Simulation Facility at NASA Ames are developing a reader-viewer-translator for IGES data files (ref. 5). NASA personnel at all three Research Centers are planning to modify software to utilize NURBS based NASA-IGES geometry data. A few of the programs include S3D - an interactive surface grid generation tool (ref. 6), VGRID - an unstructured grid generator and flow solver developed by ViGYAN under contract to

NASA Langley (ref. 7), GRIDGEN - a structured grid generation system being enhanced by MDA Engineering under contract to Computer Sciences Corporation for NASA Langley (ref. 8), and a turbomachinery analysis code under development NASA Lewis.

SPECIFICATION AUTHORS

The authors of the NASA-IGES Geometry Data Exchange Specification include more than 15 engineers and scientists from NASA Ames, Langley, and Lewis Research Centers. The coordinators of the project are listed as authors on this paper.

REFERENCES

1. Initial Graphics Exchange Specification (IGES) Version 5.1, distributed by the National Computer Graphics Association (NCGA) Technical Services and Standards, IPO Administrator, 2722 Merrilee Drive, Suite 200, Fairfax, VA 22031, telephone 703-698-9600 extension 325

2. NASA Steering Committee for Surface Modeling and Grid Generation, organized through the NASA Office of Aeronautics and Space Technology (OAST), contact Pam Richardson, NASA Headquarters

3. Geometry Subcommittee, NASA Steering Committee for Surface Modeling and Grid Generation, Matthew W. Blake Project Lead, NASA Geometry Data Exchange Specification Utilizing IGES (draft), September 30, 1991. To obtain a copy, contact Matthew Blake, MS258-5, NASA Ames Research Center, Moffett Field, CA 94035 or e-mail blake@nas.nasa.gov

4. IPO, National Institute of Standards and Technology, Gaithersburg, MD 20899

5. Blake, Matthew W.; Jasinskyj, Lonhyn; Chou, Jin: NASA-IGES Geometry Data Visualizer. *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA CP-3143, April 1992.

6. Luh, Raymond C.-C.; Pierce, Lawrence E.; Yip, David: S3D - An Interactive Surface Grid Generation Tool. *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA CP-3143, April 1992.

7. Parikh, Paresh; Pirzadeh, Shahyar: Recent Advances in Unstructured Grid Generation Program VGRID. *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA CP-3143, April 1992.

8. Steinbrenner, John; Chauner, John: Recent Enhancements to the GRIDGEN Structured Grid Generation System. *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA CP-3143, April 1992.

REQUIREMENTS FOR A GEOMETRY PROGRAMMING LANGUAGE FOR CFD APPLICATIONS

Arvel E. Gentry
Aerodynamics Research, Engineering Division
Boeing Commercial Airplane Group
Seattle, WA

SUMMARY

A number of typical problems faced by the aerodynamicist in using computational fluid dynamics are presented to illustrate the need for a geometry programming language. The overall requirements for such a language are illustrated by examples from the Boeing Aero Grid and Paneling System (AGPS). Some of the problems in building such a system are also reviewed along with suggestions as to what to look for when evaluating new software products.

INTRODUCTION

In this workshop we have an opportunity to hear about a number of new and exciting software efforts as developers are responding to our needs. However, we can sometimes also learn from history. The material presented in this paper is part history in that it explains what was done and why. It also contains some ideas as to what could be done if starting all over. But first, some history.

Existing CAD/CAM systems are devoted to the creation of computer definitions of hardware parts for manufacturing purposes. These systems vary from the small PC-based programs to giant corporate systems that are approaching a paperless design/build process. Although these types of systems might do what they were designed to do very well, they do not generally contain the capabilities needed for the efficient production use of computational fluid dynamics (CFD) applications. They also usually cannot respond to the immediate and changing needs of the aerodynamicist. CFD work requires much more than just having the surfaces of a vehicle described in the computer. A great variety of problems are faced in the aerodynamic design process. Some of these problems can be handled by a CAD system. Many others cannot. The aerodynamicist has to solve similar problems over and over again as aerodynamic design is refined. We have also found that it is very difficult to communicate our CFD needs to CAD/CAM people. This is understandable since a corporation may live or die by its CAD/CAM system, and the needs of the CFD community are very small when compared with the bigger airplane design and manufacturing picture.

In the late '70s at Boeing we could see that the newer CFD codes would soon outstrip the capabilities of our older geometry tools. Even the surface paneling of complex configurations was still a very time-consuming task. Examination of emerging CAD/CAM systems did not show much promise. After several years of mathematical research, a careful study of user needs, and several experimental pilot codes, we decided to build a new system that we initially called the Aero Surface Geometry System (ASGS); the name was later changed to the Aero Grid and Paneling System (AGPS).

The first version of AGPS was running in late 1980 and used a DEC VAX-780 with Vector General graphics devices. After a period of low level development and testing, the program went into production use near the end of 1983. The system has been greatly expanded in the years since. Additions to the

system were almost entirely driven by user needs. The system was first ported to workstations in 1984-86 and to various Unix platforms in 1987-91. A version also runs on the Cray Y-MP. Today there are several hundred active AGPS users on a variety of computer platforms.

AGPS is not available for general use outside of Boeing. Two previous papers have been published on AGPS (1,2). Several other presentations at this workshop discuss different aspects of the AGPS system.

The author of this paper was responsible for the creation of the AGPS program. He was also responsible for the geometry portion of a system called the AXXYZ Integrated CAD/CAM Software. This paper reviews the experiences gained over the past eleven years with these and other related projects and suggests functionality and system design aspects that might be useful to those developing new systems.

This paper first presents a number of examples to show just why a geometry programming language is needed. This is followed by a presentation of the high-level requirements for each aspect of the task. These include the user interface, basic mathematics, surface lofting tools, data structure, support functionality, and input/output systems. Some comments are provided relative to code portability, system maintenance, training and user support. During the workshop many of the ideas discussed here will be illustrated by live demonstrations of the current version of the AGPS program.

WHY A GEOMETRY PROGRAMMING LANGUAGE?

Early in the development of AGPS it was recognized that it was impossible to anticipate all of the types of problems that the system would have to eventually solve. The solution was to build a system that the user could extend and adapt to solve new problems. The only way that we could see to accomplish this was to devise a geometry programming language accessible by the end user. This approach allowed us to build a tool that would have useful functionality in a short period of time. By collecting the user written applications, refining them, and making them available to all users as an application library, we believed that we could greatly expand the functionality of the system at minimum cost.

The validity of this decision can best be demonstrated by reviewing a few examples that have been solved over the more than ten years that the program has been in use. Some of these problems are obviously geometry problems. Many are examples of how a geometry programming language can be used in place of FORTRAN to efficiently solve other problems. The following examples might also serve as test problems for new geometry software systems.

Geometry Problems

Wing Lofting Problem (Figure 1) - A wing loft is to be generated from basic shape defining information. The loft process is to be repeated a great number of times with different input data as the design progresses. Defining information includes:

- Wing planform information (plan view of leading edge and trailing edge lines).
- A stack of airfoils to be positioned at given span locations. (the airfoil data is normalized by chord length and thickness.)
- Wing twist distribution plus the definition of the line about which the airfoils are to be twisted.
- Wing thickness distribution as a function of span location.
- Shear distribution (how each airfoil is to be displaced vertically). After the basic wing is lofted, investigate the changes necessary in the shear distribution to make the trailing edges of the slats straight lines in the jig loft position.

Working With Digitized Data (Figure 2) - A wind tunnel model has been measured using a surface digitizing process. Convert the data to mathematical surfaces so that the configuration can be analyzed in our CFD codes. The slight discrepancies in measurement and in model alignment must be corrected in the process. Some areas of the model were not covered in the digitizing process and will have to be created with the limited available information.

Working With Airplane Measurement Data (Figure 3) - Photogrammetry and theodolite methods have been used to measure a research aircraft. In addition, large sheets of clear acetate were laid over the 3-D wing and the location of the photogrammetry points marked on the acetate along with the location of all screws and access ports. Digitize the data on the acetate sheets and convert to the real 3-D coordinates on the aircraft and compare with the original airplane drawings.

Develop Design Goal Cp Distribution for HLFC Glove (Figure 4) - An inverse CFD code is to be used to design the shape for a hybrid laminar flow glove (HLFC) on a research aircraft. Develop a method for examining the initial analysis pressure distribution and then generate the new design goal Cp input data.

Surface Comparisons (Figure 5) - An HLFC glove for a research aircraft has been designed. Determine the clearance between the glove and the basic hard wing.

Paneling Problems

A great variety of paneling problems must be handled by a geometry system for CFD applications. Rather than discussing individual examples, the point can be made by looking at the paneling examples in Figure 6. Each of these paneling problems was solved by procedures written with the AGPS geometry programming language. The important thing to note here is that, once the required surfaces were available, all of these paneling operations were accomplished automatically in a few minutes. The results were paneled data sets, or in many cases actual final data files ready for submittal to a panel code. Of course, the basic problem is that each paneling procedure written in the geometry programming language handles a given set and arrangement of surface components. If the arrangement of components is different from that provided for in the paneling procedure, then the procedure will have to be modified or rewritten. With a geometry programming language, the end user can do the modifications or rewrite the procedure in a short period of time. Without a geometry programming language system, you would have to do this by writing separate FORTRAN codes for each application, or in the worst case go back to the original code developer and request the required new functionality.

Grid Generation Problems

The use of a geometry programming language to attack field grid generation problems is described by another paper presented at this workshop so I will not dwell on this subject here * (see "An Interactive Multi-Block Grid Generation System, by T.J. Kao, T.Y. Su, and R. Appleby). Only a few comments on grid generation will be made at this time.

The generation of field grids is a major problem in the CFD community. Grid generation is still a very time-consuming process. As can be seen by the many grid generation papers presented in this workshop, we have about as many approaches as we have developers. Although the word GRID is prominent in the title of the AGPS program, we do not claim that AGPS is the last word in this area. We at Boeing use a variety of grid generation tools in addition to AGPS. However, we feel that a geometry programming language can play an important role in grid generation. Accurate grids must start with an accurate representation of the surfaces themselves. This is lacking in some of the early grid systems that we have seen. Many of the new grid generation programs are highly interactive. This often means that to

*See page 333.

solve a similar problem but with a new set of surfaces, you have to go through the whole laborious interactive process all over again. This may be fine for the CFD researcher or even for the user who only has one problem to solve. However, in a real aircraft design situation this is not acceptable. Many design iterations have to be made. Some grid generation systems have recognized this problem but their journaling and language capabilities are still rather crude.

The grid generation process must be reduced to an almost transparent part of the design process before the new advanced CFD tools really become productive. Many CFD researchers and code developers have included geometry operations and grid generation methods within the CFD codes themselves. This seems to achieve the goal of having the gridding task be transparent to the user. However, experience has shown that this approach severely limits the applicability of the program. And at the risk of offending a few people, frequently the CFD code developers have clearly been lacking in knowledge of the latest curve and surface mathematics technology.

Post-Processing Problems

The plotting and visualization of CFD results is an important part of the aerodynamic design process. We started to build a separate system to do this soon after AGPS became operational. We called it the Aero Flow Imaging System (AFIS). At first AFIS was developed along the same command and program structure concepts as had been used for AGPS. This would minimize the user training and give the same flexibility as was available in AGPS. As the development of AFIS proceeded we found that we were copying more and more from the AGPS program. When we realized this, we stopped the development of AFIS and simply included the necessary display commands and functions in the AGPS system itself. Since then other programs such as PLOT3D and FAST have been built specifically for the post-processing task. However, we still find that many post-processing tasks are best handled with the capabilities provided within a geometry programming language such as AGPS. Several of these are illustrated below.

Combined Data Display (Figure 7) - On a single plot display the geometry, paneling, panel method pressures, and surface streamlines for a complex configuration.

Surface Porosity Visualization (Figure 8) - A porosity measuring device has been used to measure the skin of an HLFC glove on a flight test aircraft. Rows of flutes lie under the porous skin and provide channels for the air that is removed from the boundary layer. The porosity measuring device was placed at intervals along these flutes and the porosity of the skin measured. Prepare a display showing the rectangular area for each measurement point and fill the rectangles with different colors to represent the surface porosity.

Vehicle Trajectory Visualization (Figure 9) - Output from a trajectory optimization program must be visualized. The user would like to see the trajectory flight path in 3-D space, the orientation of the vehicle along the flight path, and the variation of selected flight parameters. Many plots are to be routinely produced automatically for a variety of flight simulations.

Flight Path Clearance Problem (Figure 10) - Visualize the takeoff flight path necessary to clear nearby mountainous terrain.

Geometry Programming Example

We have found that a geometry programming language can be used for a very wide range of what might be thought of as non-geometry problems. The example below serves as a good illustration of these types of applications.

Momentum Thickness Reynolds Number Problem (Figure 11) - An Euler code has been used to calculate the flow about a configuration. Hybrid Laminar Flow Control (HLFC) studies need a quick

estimate of the momentum thickness Reynolds number along the wing leading edge attachment line. The CFD code output includes the inviscid flow properties. The momentum thickness Reynolds Number equation needs the flow velocity along the attachment line and a velocity derivative term calculated in a plane perpendicular to the attachment line. Figure 12 is a partial listing of the solution of this problem and can be studied to better understand and appreciate the great power of a geometry programming language such as AGPS.

GEOMETRY PROGRAMMING LANGUAGE REQUIREMENTS

The requirements for a geometry programming language would obviously fill a large volume. This paper is only an outline of some of the most significant issues. Again, many of these ideas have been gained not only in the over ten years of experience with AGPS but also in the development of a Boeing Computer Services CAD/CAM product called AXXYZ Integrated Software.

Early in the development of AGPS we recognized that a geometry programming language would face many of the same problems of other computer languages including:

- Training
- Language help systems
- Debugging procedures
- Language stability over time
- Code portability
- User support
- Application maintenance

The key feature to our approach to these problems was the decision to allow the user to either type in a command line or enter a command using a menu system, which then executed the command immediately. All of the system's graphics and language features could then be used to verify that the command or group of commands did what you wanted. The ability to write the commands entered during an interactive session to a file provides a means of saving part or all of an interactive session. This procedure we called journaling. The file could be edited to remove errors made in the interactive session. We then provided several ways to play back the file for repeated execution. The program could be operated purely by interactive operations, by running previously recorded procedures, or by a mixture of the two. These approaches were not new since this is the way that most computer operating systems work. In the initial design of the program, it was decided that not only should the code be highly interactive, but it should also run in the batch mode.

A few general requirements were a result of the computing environment that existed at the beginning of the program design. Initial pilots considered the use of a big mainframe computer. This was not possible at the time because of slow communications and lack of mainframe virtual memory capability. Workstations were not available when AGPS was initially developed. AGPS was originally developed by borrowing time on someone else's VAX-780 since our company division did not own one. We knew that computer hardware and graphics devices would probably change several times over the life of the program. The code had to be made as portable as possible, but the limited performance on the initial computer platform forced us to give up some portability for performance. We had to pay for this later on.

User Interface

Many aspects of the user interface determine the usability of a program. These include the workstation window system, graphics display and control functions, the use of the keyboard and mouse, etc. This discussion will only touch on those areas that are peculiar to the programming language aspects of the system.

Modern interactive programs tend to be menu driven where the developer has figured out or has been told about all of the options that the menus must hold. Some of these systems have been expanded to include a limited macro capability. The AGPS system was developed from the beginning as a command-line language system. Menu capabilities were added to improve user access to the system capabilities.

One criticism sometimes heard from first-time viewers of AGPS is that "it does not have a modern user interface". They apparently expect to see pop-up/pull-down menus with lots of icons, and with little if any text input. We have looked at this kind of user interface but have found it not suitable for a system that is essentially a programming language.

The heart of AGPS is the more than 160 commands. User access to these commands is an important part of the user interface. Most commands have a number of input parameters called keywords. Each keyword may have one or more options. Providing a convenient menu access to such a large number of commands and command options is no easy problem. In the current version of AGPS this is accomplished by providing a three level menu system. The user can switch between text input and menu mode whenever desired. The options in the highest level menu indicate the groupings of the different types of AGPS command functionality. Typical groupings are DATA STRUCTURE, DRAWING OBJECTS, and CREATE SURFACE.

Selecting the CREATE SURFACE option brings up the second level menu containing the names of all commands that create surfaces. Selecting a given method then brings up the third level menu that actually contains the command itself with its keywords and current defaults. The third level is not really a menu but an editing window where you type in or modify the keyword option inputs. Hitting the on-line help key at this stage brings up a separate help display that describes the command options along with several pages of more detailed description. Figure 13 is a screen dump of what the user sees. If users know the 3-character abbreviation of a command they can bring up the third-level command editing window without having to go through the first- and second-level menu selections. A potential enhancement to the AGPS third-level command window would be to allow the users to type in the object names required by the command and allow them to pick the other options from a menu. The text-input mode is frequently used for commands that the user is most familiar with, and it is faster. Menu mode is used for commands that the user might not be too familiar with.

A typical use of the Fit-Surface command in the text-input mode might appear as follows:

```
FIT-SURFACE SURF=SURF1 ARRAY=ARR1 DEGREE=CUBIC-B-SPLINE
```

Or, using the standard 3-character abbreviation (FSU in this case), and omitting the optional keywords by following the standard keyword order, this command could be typed in as:

```
FSU SURF1 ARR1 CUBIC-B-SPLINE
```

One of the main requirements for a geometry programming language is to make it as flexible as possible, including the menu system mechanics. We solved this in AGPS by having the contents of the menus driven from a text file. The users can copy the menu control file, called the geosyn.dat file, to their own account and customize it to meet their own needs or preferences. This provides a means of tailoring the program so that it looks like it was specifically written for a different application from the standard program defaults. Figure 14 contains an example of a menu system for use in designing an America's Cup yacht. The user can switch back and forth between the standard menu and alternate command system as often as required. These tailored and specialized menu systems may also include command files (customized macros) that give the user easy access to very complex operations. We call these specialized menu/command file systems "application packages" or "Task Menus". The default AGPS menu system includes several of these packages.

There are several user interface areas for a geometry programming language that do not fit

conveniently into a menu type of operation. These are the use of program flow control constructs (do-loops, if-then-else, etc.) and calls to specialized subroutines. These are best handled by text-input mode. However, it would not make sense to try to type in all of the lines of a large do-loop. A typing mistake means you have to abort the loop and type it all in again. It is for this reason that many complex procedures are frequently developed by a combination of typed-in, menu selected commands and command file fragments executed from an external text file created with an editor. A text editor that "knows" the syntax and commands of a geometry programming language can be a very helpful tool for the less experienced users. This idea was tried out using the Language Sensitive Editor provided on DEC VAX VMS computers. However, it was not pursued further because a similar editor was not available on any of the other host computer platforms that the program ran on.

Debugging a program is a problem common to all programming languages. This problem is handled in AGPS by a number of different techniques. Several AGPS commands can be used within a command file to help in the debugging process (MENU, PAUSE, EVALUATE, DISPLAY-FAMILY-TREE, $UNWIND, $WRITE, $ABORT). For example, the AGPS MENU command switches you from the text-input mode to the menu-input mode. The MENU command can be placed anywhere in a command file where you think you might need to interrogate the data structure, to plot data, or draw objects. When MENU is executed in a command file you are switched to the menu mode and can use any of the commands to help find out what the command file has accomplished to that point. Also because AGPS uses an interpretive language, portions of a new command file can be cut from an editing screen, pasted into the AGPS text input window, and other AGPS commands can be used to interrogate the data structure or draw objects to be sure that it did what you wanted (instead of what you thought you told it to do).

The ability to save all or portions of an interactive session is a key feature of a geometry programming language. This is called "journaling" and can be done in two different ways. One way is to save each command just as it is read by the command person from either the type-in or the menu mode. Reading the command file back into the program allows you to repeat the sequence of operations just as you did in the initial live session. If a command requires that an object be picked from the screen, then on the replay you will again be required to pick an object. This approach poses a problem when you have a complicated sequence of screen picking operations as is typical in a grid generation process. This problem is solved by a second type of journaling that not only saves the command executed but also saves the name of the object picked. During an interactive session the user should be allowed to switch between these two modes of journaling.

Many CAD systems draw everything that defines the current working entity to the screen. Items that you do not want to see are selected and made invisible. The opposite approach was used in the AGPS program. We knew that the data structure would frequently be very large, involving many strings of points, surfaces, surface grids/paneling, and space grids. We elected to place the burden on the user and have that person draw only those objects to the screen that are needed for the operations being performed. It might have been useful to attach line style and color to the different entity types as the case with many CAD systems but this was not done.

Most CAD systems allow several different views of an object on the screen at the same time. Multi-views are a desirable feature for a geometry programming language system, but were not provided in AGPS due to hardware/software limitations in the early years of its development. This capability would be a desirable future enhancement.

Mathematics

The current most popular approach for working geometry problems is non-uniform rational B-splines (NURBS). NURBS-based systems have the advantage over polynominal systems in that conic curves can be represented exactly without approximation. In a large corporate environment, the selection of the curve and surface mathematics for a geometry programming language is not necessarily a technical decision. You may have to provide capabilities to handle data from other and sometimes old outdated

systems. This was the situation with AGPS. We had to provide a variety of mathematics capabilities for most of the curve and surface generation functions. These varied all the way from procedurally defined surfaces (such as tubes, bodies of revolution, conic lofted surfaces), to NURBS-defined surfaces. There is also a requirement to be able to handle the variety of objects that are read in through IGES files from CAD systems. Regardless of how a curve or surface is stored in the program data structure, it is important that the rest of the program, such as the intersection routines, be able to work with the entities as though they are fully parametric curves and surfaces.

The form and details of the mathematics for working with parametric curves and surfaces are major subjects in themselves. This single area controls what can and cannot be done with the system. For example, the use of solids seems to be a general trend. Some CAD programs create solids by joining together surfaces with all of the edges matching up exactly. For CFD applications, however, we must approach the problem with care. How are the surfaces for the solid created? For example, in aerodynamic applications we would usually create the fuselage and wing surfaces so that they intersect each other. We would then find the intersection and trim off the portion of the wing that is inside of the body. The method used to trim the wing is important since you may not want to lose the original wing loft in the process. If you trim the wing using a method that has the effect of cutting off and losing the portion that is inside of the body, then what do you do when you apply an inverse design method to the body and find that you had trimmed off too much of the wing?

We have found that an entity type that we call a "subrange object" is very powerful for both geometry building and paneling/grid-generation processes. A subrange point consists of a pointer to a curve or surface and the parameter value specifying the location of the point. Many surface paneling operations can be accomplished simply by creating arrays of surface parameter values and a pointer to the surface itself. A subrange curve on another curve, or a subrange region of another surface is accomplished through a mapping process (Reference 1). This subrange capability allows curves or surfaces to be trimmed but retains the exact mathematical definition of the original object.

Surface Lofting Tools

The surface lofting area shares the most commonality with CAD systems. However, in CFD applications we have many strings of points representing airfoils, body cross-sections, and surface and space grids. It is cumbersome to work with this kind of data in many CAD systems. Also, most CAD systems are primarily devoted to the generation of hardware parts and are frequently rather limited in their ability to efficiently create the sculptured surfaces required in CFD applications.

A variety of surface lofting tools are needed in a geometry programming system. For curves, these should include the ability to fit splines through strings of points, the construction of classical conic curves, blending curves, and composite curves. Surface creation methods should include surface fit to a rectangular array of points (mesh), surface through curves, surface through intersecting networks of curves, a surface created by sweeping a cross-section along a control curve, plus the usual ruled and tube surfaces.

The ability to modify curves and surfaces is also important in the lofting process. This should also include the ability to convert between the different math forms (i.e., linear, polynominal based systems, NURBS).

The use of these powerful tools to create final lofts is not an easy problem. In the aircraft business, good experienced lofters are a vanishing breed. The use of CAD systems to create high quality surfaces requires a great deal of training and experience. Unfortunately, most aerodynamicists working lofting problems do not have this kind of background. The surfaces they create may have wiggles that are undesirable in the final product. These surfaces also may not meet manufacturing requirements. On the other side of the coin, surfaces generated by a designer on a CAD system may not meet aerodynamic requirements. The surfaces may look good on the screen but the arrangement and parameterization of the

surfaces may make it very difficult to work with them when you try to generate surface paneling or grids with automated tools. It is certainly possible to solve these problems entirely within the CAD environment, but the cost and flow time would be very high. Required changes to the CAD system to meet CFD requirements may take months or even years to achieve. A geometry programming language can be used effectively in attacking these problems providing the interface to and from the CAD systems is made as transparent as possible. Special tools must also be provided in the geometry programming language system for checking the quality of surfaces created by the aerodynamicist. These tools are sometimes lacking in CAD systems, and we have often found AGPS useful in checking the quality of surfaces generated in the CAD environment.

Data Structure

The data structure used in a geometry programming language will have a major impact not only on the performance of the system, but also on the users ability to learn and use the system. The data structure in AGPS was influenced greatly by the assumption that a variety of object types would be necessary and that interactive operations would involve frequently modifying the data. It was also important that certain objects in the data structure be of arbitrary dimension.

We thought that the primary building blocks would be strings of points that could be combined to form arrays to which surfaces would be fit. An array object would be defined as pointers to the locations of string objects, and a string would contain pointers to the individual points. If a point were modified, then the string and arrays would automatically be updated. Parametric surfaces fit to the arrays would involve the generation of surface patches for each set of four points in the arrays. The patches were stored without any reference to the original array. If a point was modified, the surface object would have to be deleted and a new surface fit to the array. The advantage of this approach is that the points, strings, and arrays are very easy to work with. The disadvantage of storing points, strings, and arrays in this manner is that it involves a lot of storage overhead. This became a problem when we started generating large field grids with many thousands of points. The solution was to add a data object to the system which reduced the overhead but required that the application programmer know how to find a specific point when needed.

In AGPS the user specifies the name for every object to be created. This has its advantages and disadvantages. Using logical names helps in understanding the process and in examining the data structure. An object called WING would be hard to mistake for something else. However, a special package of procedures usually has specific names assigned to objects that they create internally. This will sometimes cause conflicts when attempting to use a mixture of user generated operations and standard procedure packages. The object RENAME command becomes very useful in this situation.

The data structure of AGPS presently consists of over 30 object types. Additional object types are developed when needed. These object types can be grouped into several basic entity forms.

Geometric Representations:	Non-geometric Representations:
Points and Point-Matrices	Lists
Curves	Ostentations (show displays)
Surfaces and Solids	Text

The user of applications written with the AGPS geometry programming language does not normally have to know much about the data structure. However, the application developer must know a great deal about the data structure, and this has been one of the most difficult areas for the application programmer to learn. Commands such as the DIRECTORY, DUMP, and DISPLAY-FAMILY-TREE, are useful, but some new ideas in this area are really needed.

Support Functionality

The primary capabilities of a system are contained in the commands. AGPS has over 160 commands, some of which have previously been discussed. Identification of all of these commands is beyond the scope of this paper. A few of the other types of functions are discussed below.

Logic Control

A geometry programming language needs the usual do-loop and if-then-else constructs for controlling the solution logic. These constructs require that several lines of code be read in and stored before they are actually executed. If a mistake is made, you need some way of aborting the process and starting over. You also may have limitations on the number of commands used in a do-loop or if-then-else structured due to built-in buffer storage limitations. These problems were present in early versions of AGPS but are being removed in more recent versions.

Mathematical Expressions

The geometry programming language needs the ability to do mathematical calculations, and it is helpful if the syntax used is as close to FORTRAN as possible. The minimum set of math functions should include SIN, COS, TAN, ACOS, ASIN, ATAN, EXP, ABS, SQRT, LOG, and LOG10.

In AGPS we needed a way of separating the mathematical operations from the regular commands. This was done by placing a $-sign in front of all operations that were not to be processed by the regular command processor.

```
$X2=X*2.5
$Y2=Y*3.4/1.5
$Z2=Z*(X+Y)
REPLACE STR1.5 [(X2,Y2,Z2)]  ! Replace the string point STR1.5 with new values.
```

The variables X, Y, Z, X2, Y2, Z2 in the above examples are called symbols and their numerical values are stored in a buffer region. They are temporary variables and are not part of the regular object data structure.

Data Structure Access

The numerical values of objects stored in the data structure must frequently be retrieved so that they may be used in mathematical calculations. In AGPS this is accomplished with special $CALL directives. For example the directive GET_COORD will interrogate a curve or surface at the specified parameter location and place the coordinates in symbols.

A number of similar calls retrieve other data from the data structure or return information about the status of the system, such as computer platform identification, mode of operation, and picture viewing angles.

Miscellaneous Control Functions

Several other functions are needed to control command file processing:

- stop or continue command file processing when an error is found
- stop command file processing at this point
- exit from do-loop or if-then-else processing
- write text and symbol values to the screen

INPUT/OUTPUT CAPABILITY

A geometry programming language must have extensive capabilities to read and write data. Input data may arrive from a variety of external sources. Efficiency may also dictate that in-house standard or neutral files be readable.

The AGPS program has slowly expanded to include a number of input/output operations. Of primary importance is a flexible means of reading and writing strings of points in various formats. This input flexibility allows convenient interfaces with a variety of other codes. A number of special formats are also provided to allow direct processing of important CFD code output files such as PLOT3D.

AGPS has a WRITE-FORMATTED-DATA command with a variety of capabilities for format and output of data. Standard command files are available for the output of data in several standard Boeing formats such as WRITE-A502, WRITE-RMS, and WRITE-GGP. AGPS is frequently used in converting data file formats for other programs instead of using FORTRAN.

Every geometry or grid generation system should have the ability to read in at least a subset of IGES data that may originate from other systems. The Boeing experience, however, has shown that what is promised as conforming to the IGES standards may not always be correct. Parametric curves and surfaces from different systems may also not be parameterized in the same way. Some systems normalize parametric entities to 1.0 and others do not. The geometry system should have the ability to account for these variations. It should be able to reverse the direction of curve/surface parameterization and to refit the surface and change the mathematical representation when necessary. When this is necessary, tools should be available to check the accuracy of any surface refitting operations.

A geometry system should be able to save and restore all or part of a working session at any time. Many CAD systems automatically save data to disk at periodic intervals. Some systems also provide checkpoint restart capability. In AGPS we put the burden on the user to save data when desired. When restoring a file, object names should be checked by the system and conflicts with already existing objects automatically fixed and the user informed of all changes.

CODE PORTABILITY

The computer industry is still far from achieving standardization, particularly in the graphics programming area. Because the hardware manufacturers are always playing a game of one-upmanship, it is also hard to maintain a balance between achieving maximum performance on a given platform and maintaining portability between machines and systems. The long life of AGPS has presented a number of difficult portability problems. Work was started in 1986 to improve the portability of the system and this has paid significant dividends. Versions are now available on a variety of workstations including DEC VAXstations (VMS and Ultrix), Apollo, Hewlett Packard, Silicon Graphics, and IBM RS-6000. A version is also available on the Cray Y-MP. Currently we are heading toward the industry standard UNIX and X-windows environment.

SYSTEM MAINTENANCE

Program maintenance requirements are often not very well recognized by upper management. This is especially so in the case of a geometry programming language such as AGPS. Since the system is open-ended we are constantly seeing new applications and requirements. Company systems that we have to interface with such as CAD/CAM systems are evolving as are the CFD codes that we have to support.

Without a geometry programming language we would have to be spending a great deal more time and effort writing a myriad of computer programs and systems to keep up with the demands.

The first version of AGPS was written in nine months by a team of four people (a lead engineer, a mathematician, and two programmers). A great deal of effort has been expended over the past eleven years in system maintenance. The program has grown considerably in size although the original program structure and concepts have remained the same. Most of the program is still in FORTRAN. Some have suggested that a second generation version of AGPS be written using the C-language with object oriented programming concepts. This would be desirable but is not cost effective at the present time.

A major maintenance problem for any program is people. It is difficult for any programmer to learn and understand all aspects of a program as big and complex as AGPS. We have been fortunate in being able to keep an excellent experienced programming staff with minimum turnover.

The great success of a program also has its down side. A large number of users and applications over a long period of time imposes constraints on maintaining program stability. Because the aerodynamics community also controls the aerodynamic geometry programming language, changes are much easier to achieve than is the case with a big corporate CAD/CAM system. However, quick changes to the system are no longer as easy as they were earlier on. Capabilities may be required that cannot be achieved immediately within the AGPS system. As a result smaller special purpose programs spring up to meet specific needs. This frequently cannot be avoided since jobs have to get done. However, it does mean that overall maintenance costs for an organization will increase and the greater number of separate programs causes portability and maintenance problems.

TRAINING AND USER SUPPORT

A flexible and extendible system such as AGPS imposes some tough training and user support problems. AGPS is not as easy to learn as is the usual CFD program. Geometry is not a favorite subject for the aerodynamicist. It is a necessary evil in solving design and analysis problems. As a result, geometry experts in the aerodynamic community are few and far between. Designers working on CAD/CAM systems are frequently given weeks of training followed by a closely supervised apprenticeship period. AGPS training has been usually limited to a week or less. Subsequent training is accomplished on the job. The major drawback of this approach is that more support is required from experienced full-time experts. Another aspect of the training problem is that the aerodynamics users will frequently have to do complex lofting tasks for which they may not be adequately trained. The result will be bad aerodynamic lofts that get sent to the big corporate CAD/CAM system and then need to be fixed. One solution is to increase the lofting training of those aerodynamicists who will be designing complex surfaces. Another important solution is to provide the aerodynamicist with the proper tools to evaluate the quality of the surfaces that are created. Another possibility is to train the aerodynamicist in the use of the big CAD systems. For some type of surfaces the CAD system may be a good approach. However, for many other types of surfaces the geometry programming language may do the job quicker.

A geometry programming language does a lot more things than just lofting surfaces. These include paneling, grid generation, and special post-processing problems. In AGPS several approaches were used to attack the user training and support issues for this great variety of problems. We assumed that users with limited training and experience would primarily be using applications written by other more experienced people. The more experienced users could modify existing procedures and write new ones of their own using the help of full-time experts when needed. A small group of full-time experts would prepare larger heavily used applications for a broader user community. They would also serve as consultants, direct the computer programmers developing and extending the executable code itself, and test and validate new versions of the code. The full-time experts would also maintain a standard library of applications written with the language.

CONCLUSIONS

More than ten years of development and experience with the AGPS geometry programming language has proven that it can be a powerful tool for CFD applications. The key to this has been the ability to solve problems that not only were not thought of in the beginning, but that would require a much greater amount of resources to solve using any other approach. Problems that frequently required weeks or days to complete are now run on a routine basis in hours or minutes. The major down side is that AGPS has at times been viewed as competing with corporate CAD systems. The existence of two systems that seem to be doing the same job has bothered some managers. It has sometimes been hard to convince them that both systems have their purpose and that the best approach is to use the right tools for the right job. A seamless interface between these systems should be the desired end goal.

ACKNOWLEDGMENT

Special thanks to Robyn Wittenberg for her valuable suggestions and skills with AGPS in preparing material for this paper.

REFERENCES

1. Snepp, David K., and Pomeroy, Roger C.: A Geometry System for Aerodynamic Design. AIAA-87-2902, 1987.

2. Capron, W. K., and Smit, K. L.,: Advanced Aerodynamic Applications of an Interactive Geometry and Visualization System. AIAA-91-0800, 1991.

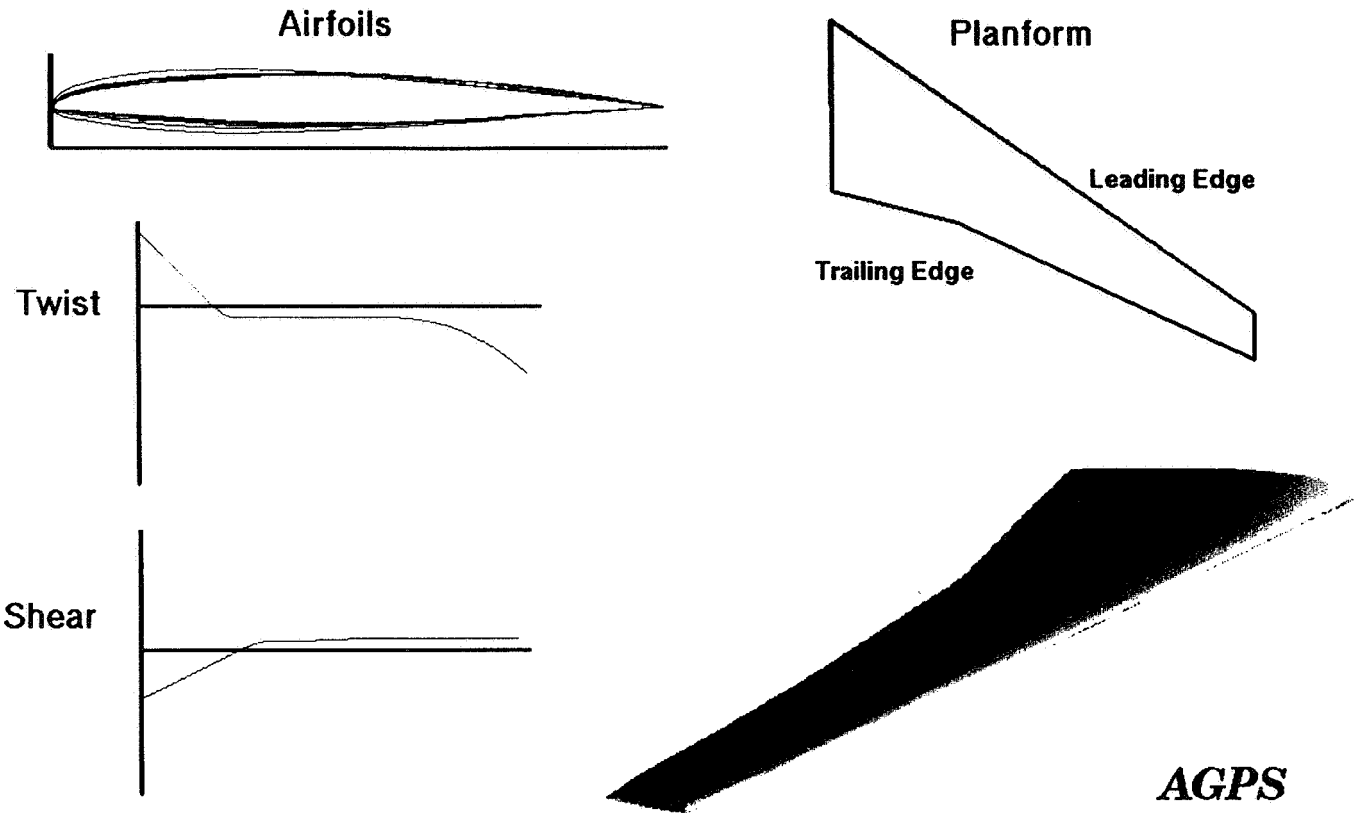


Figure 1. Typical wing loft problem.

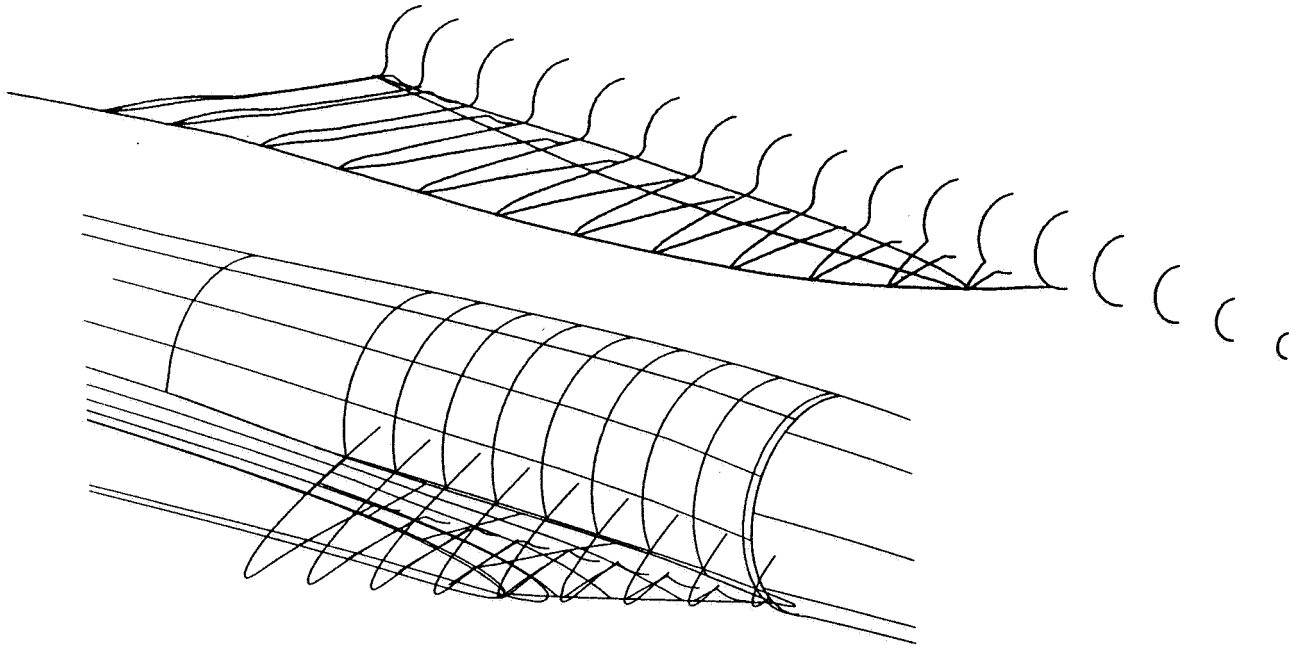


Figure 2. Working with digitized data.

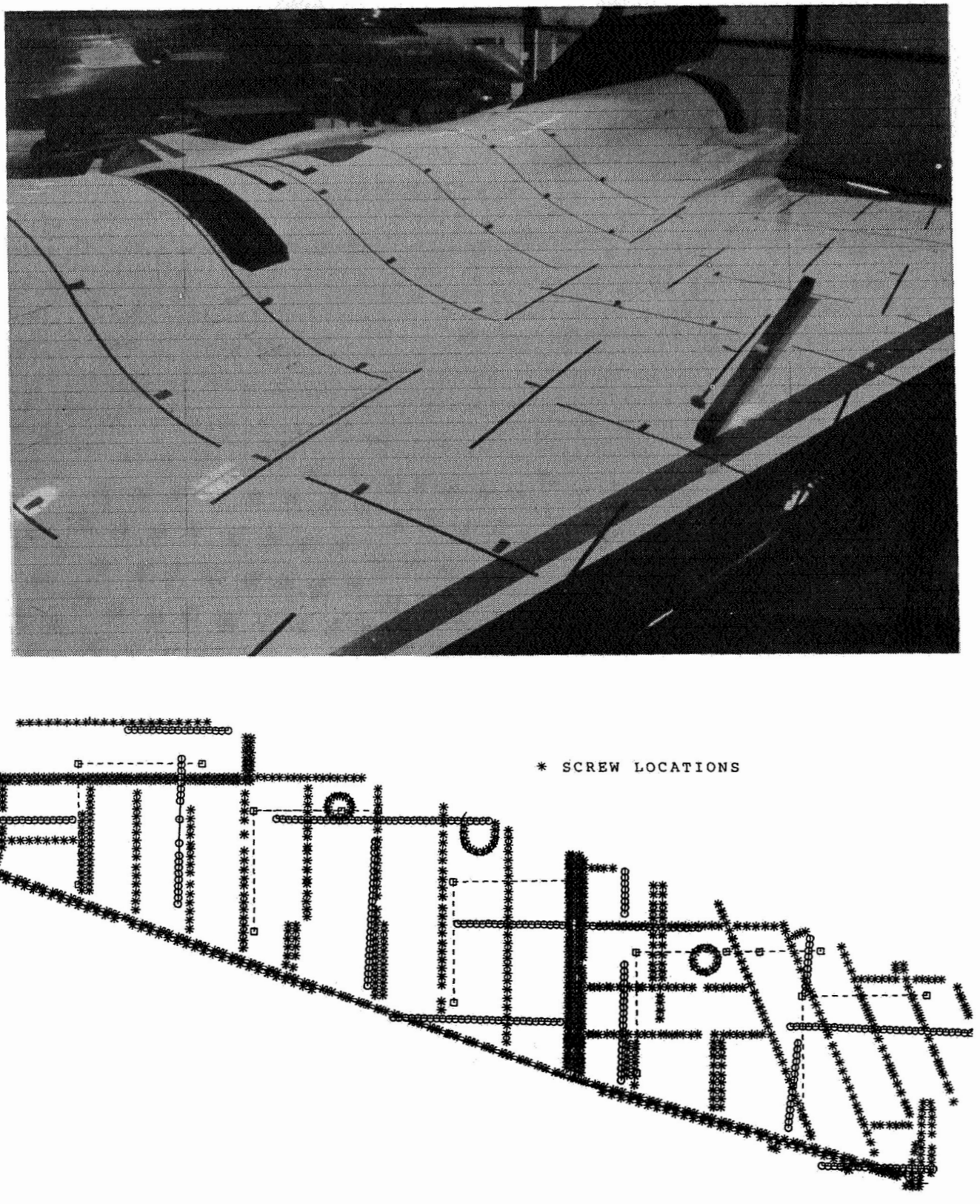


Figure 3. Working with airplane measurement data.

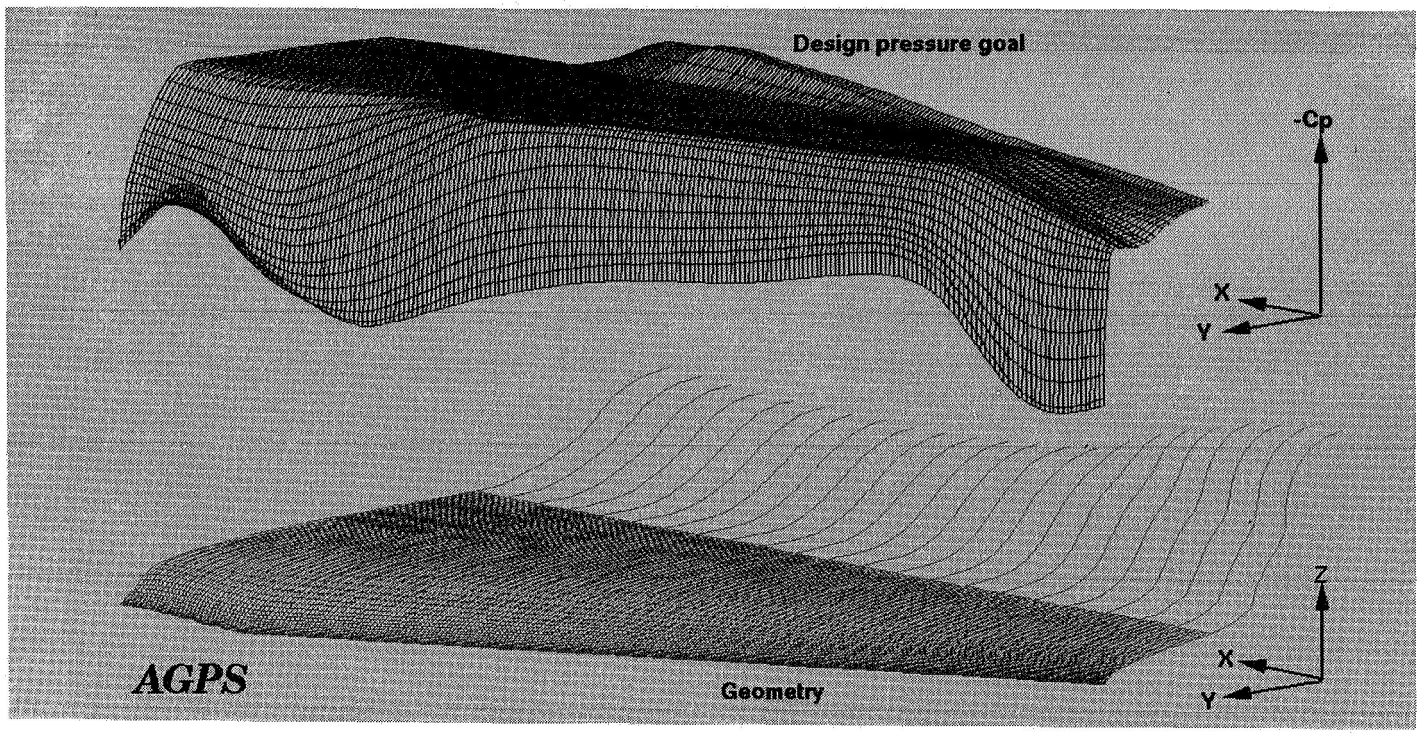


Figure 4. Developing design goal Cp distribution for HLFC glove.

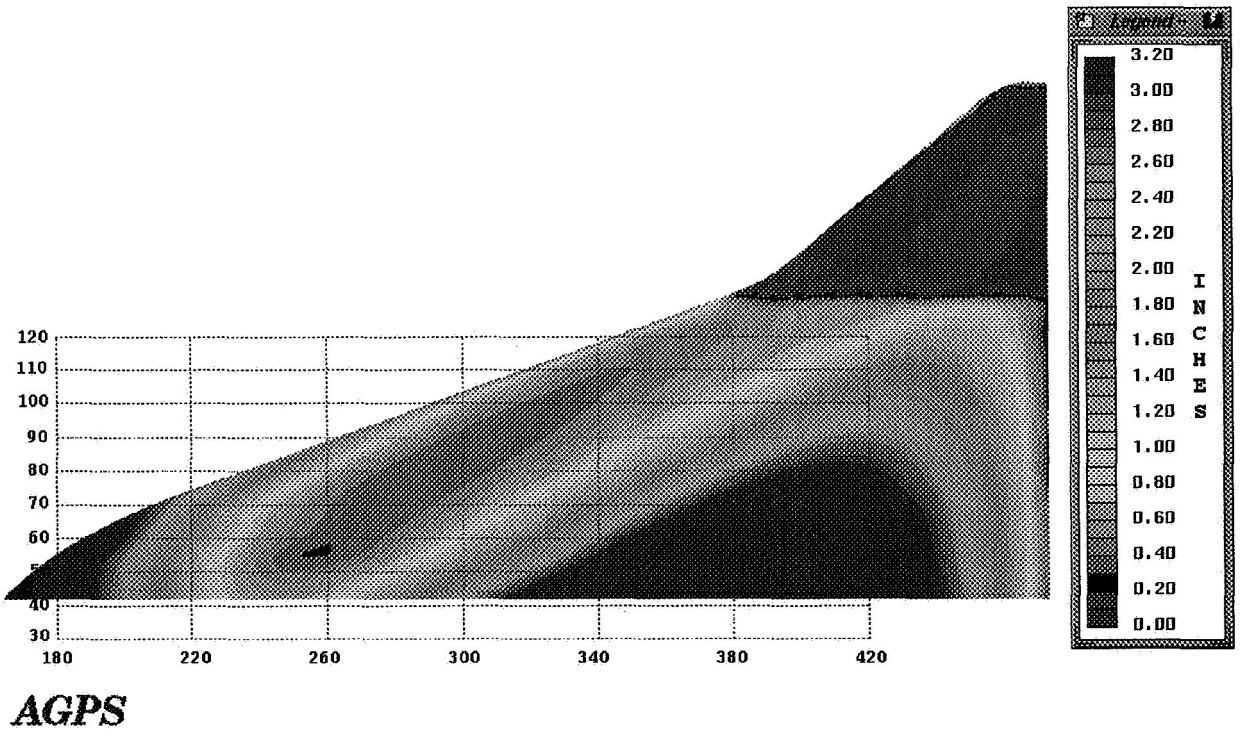


Figure 5. Comparison between two surfaces.

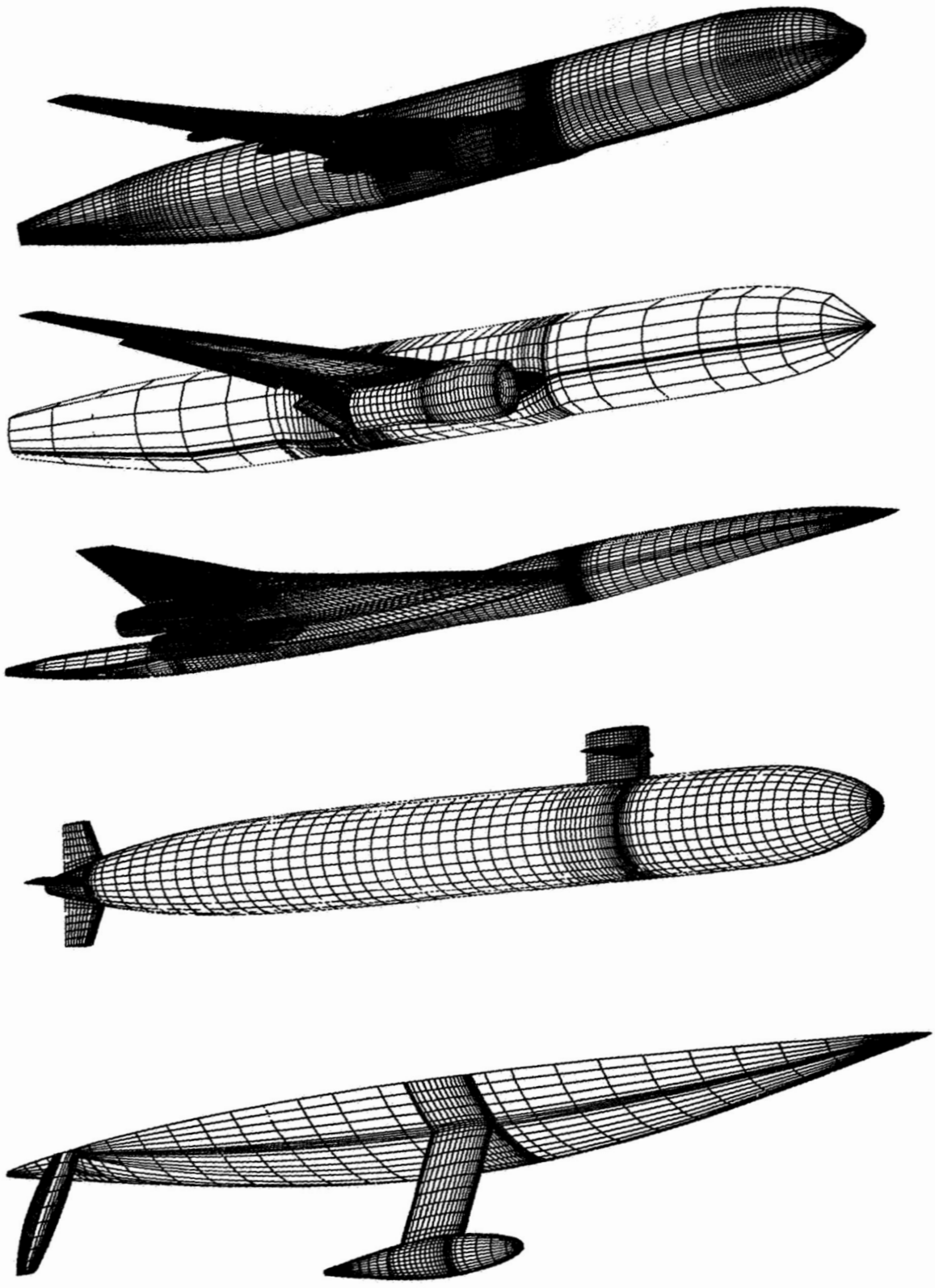


Figure 6. Examples of automatic paneling procedure results.

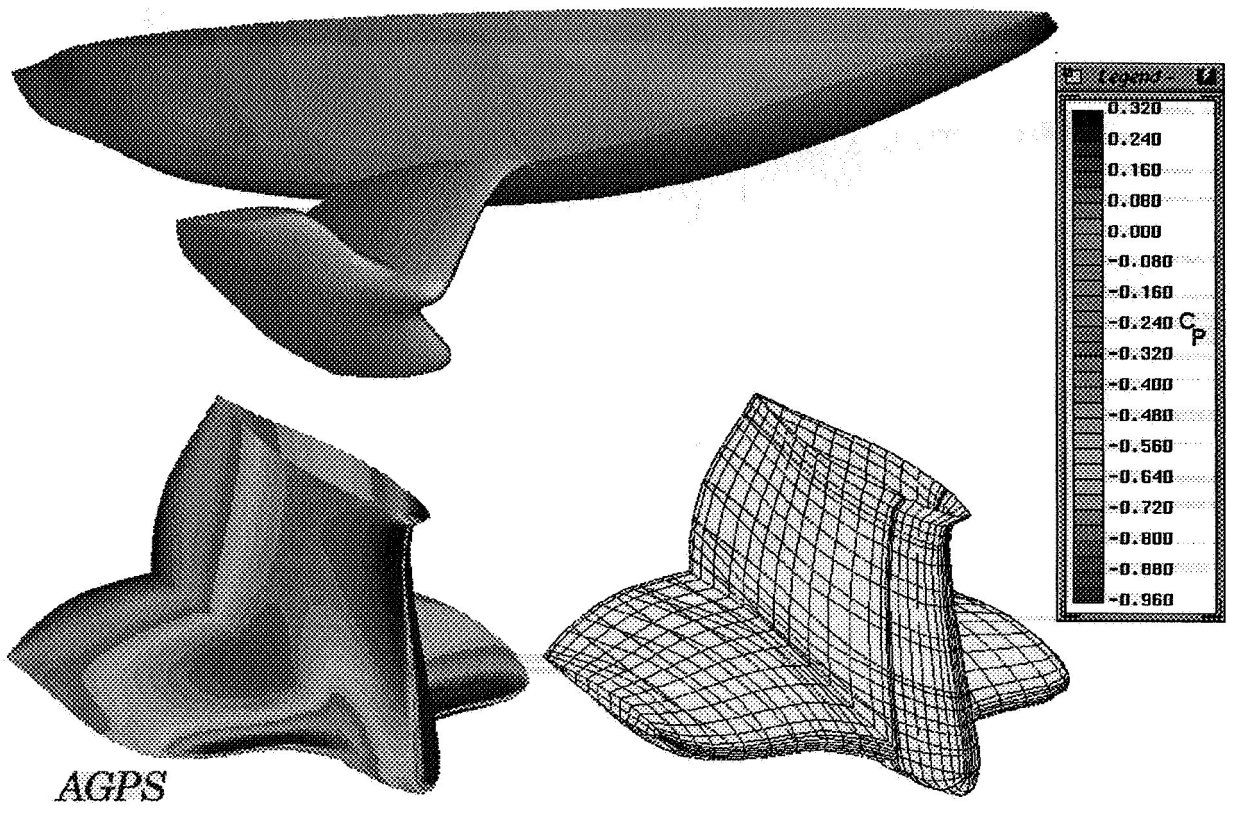


Figure 7. Display of both geometry and CFD results.

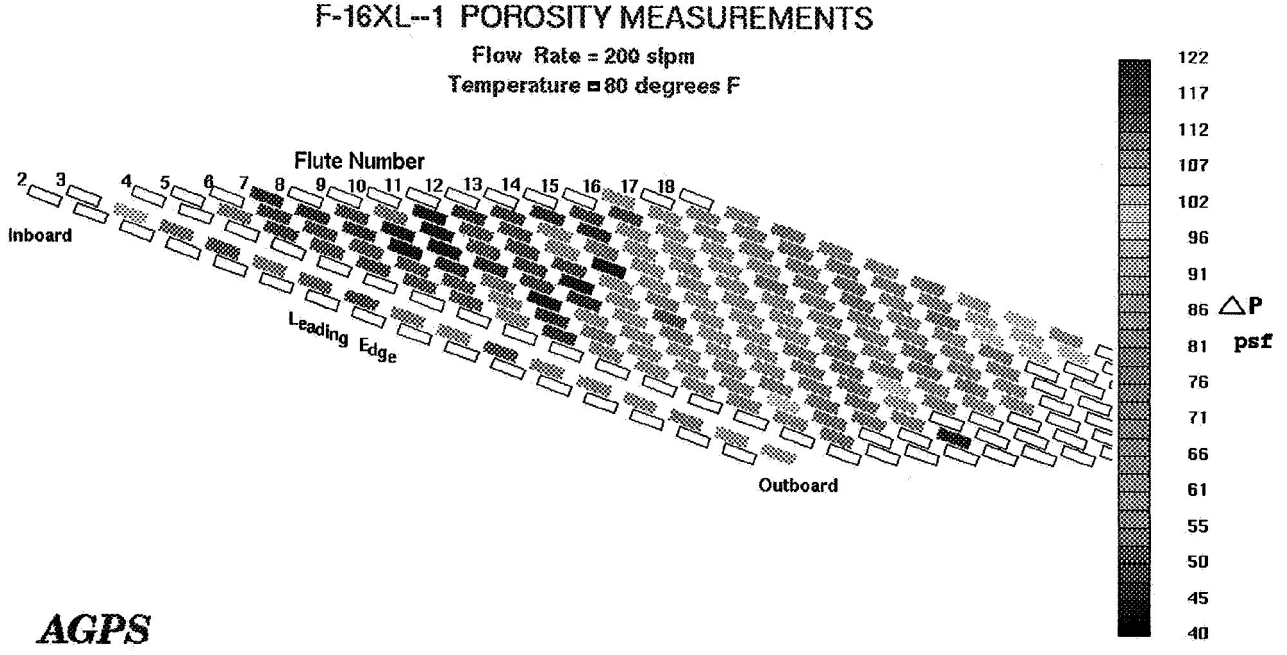


Figure 8. Display of surface porosity data measured on a flight test airplane.

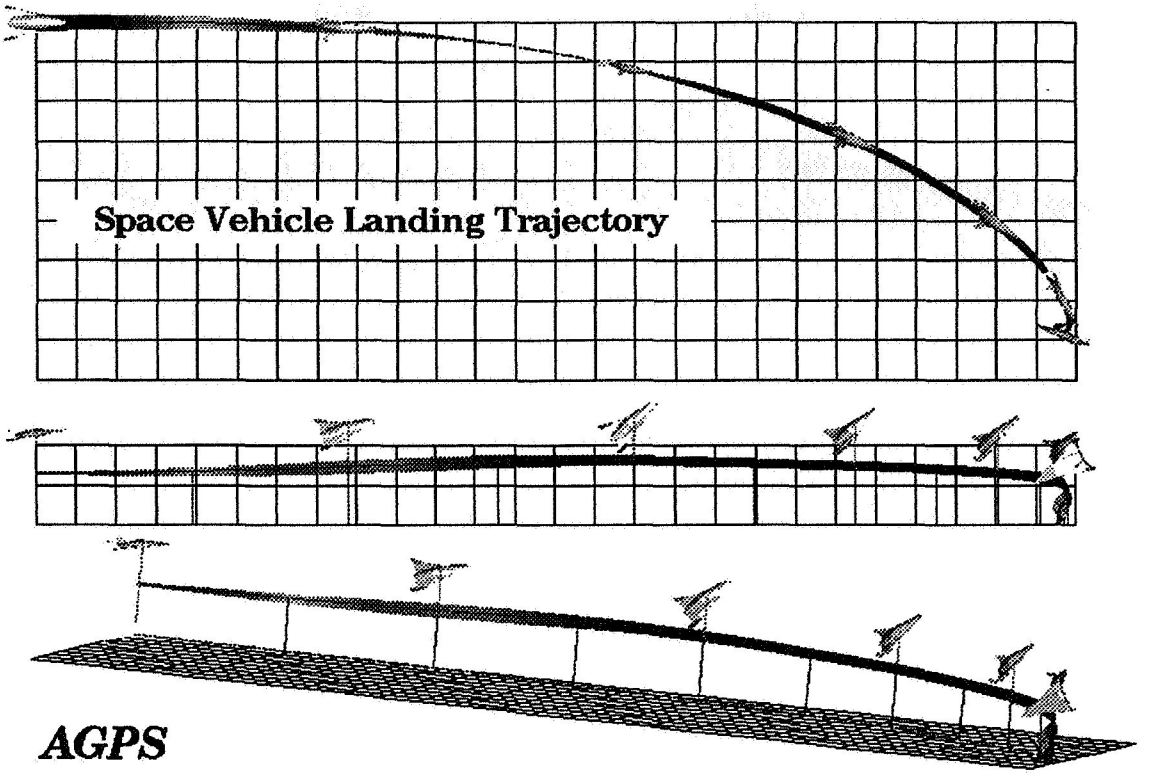


Figure 9. Using AGPS to display results from a trajectory program output.

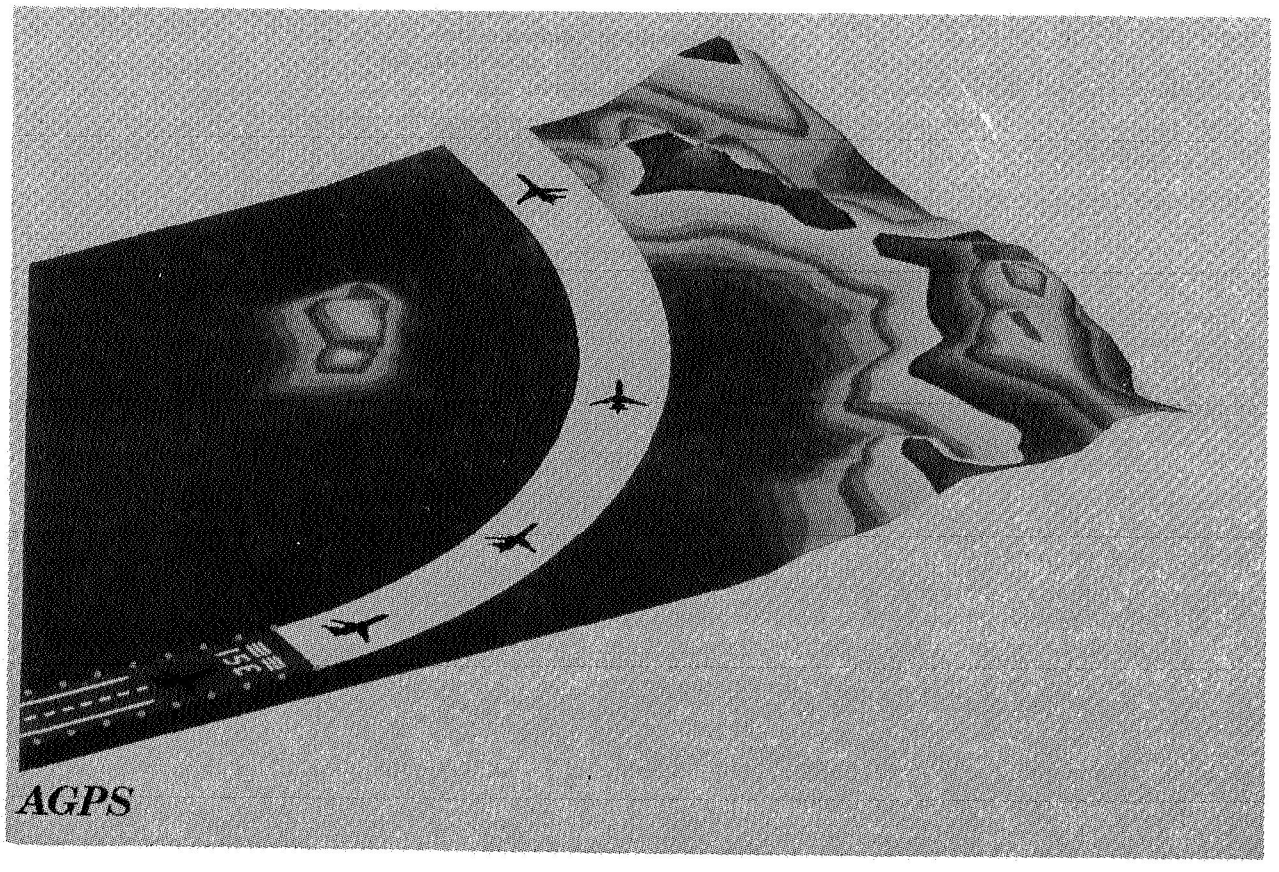
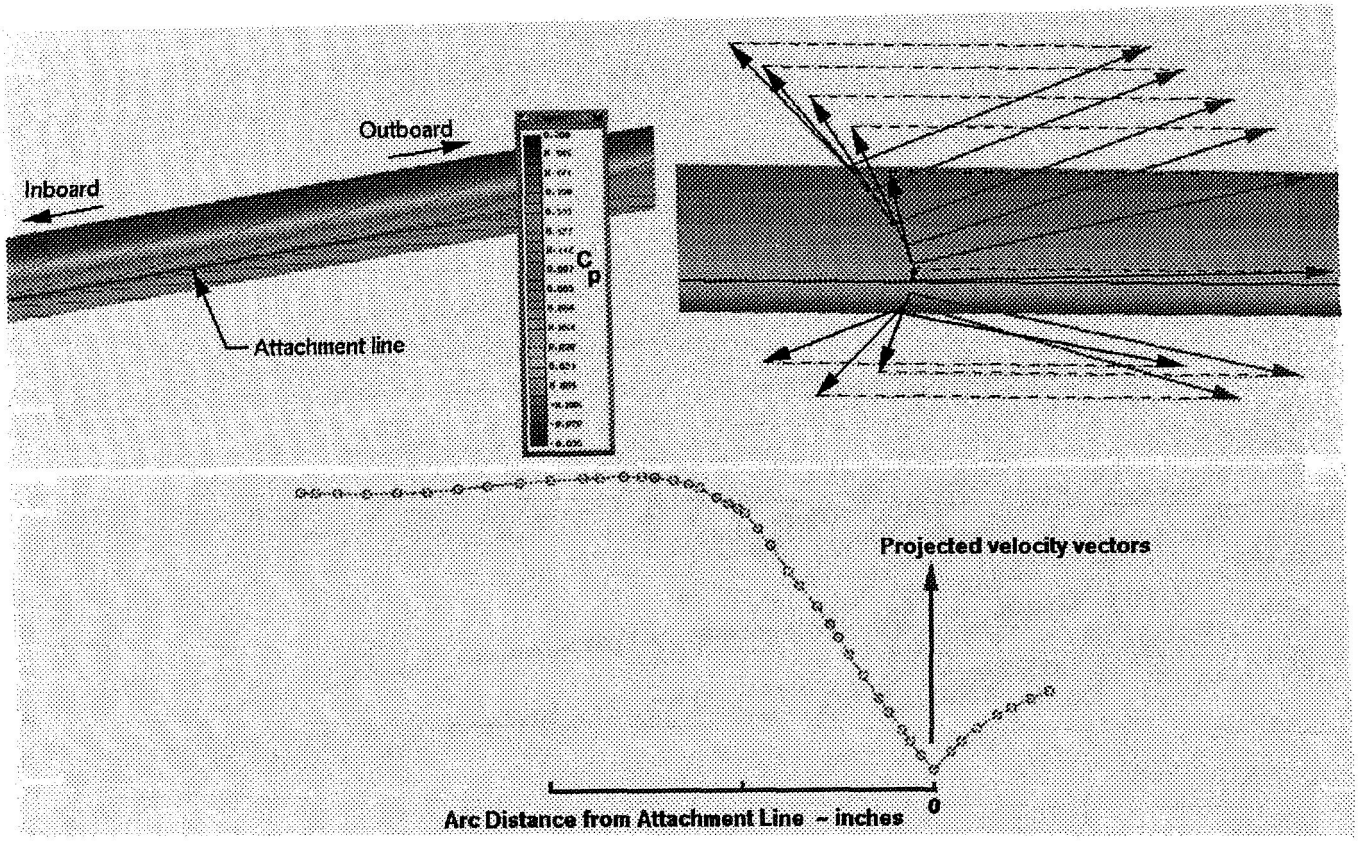


Figure 10. Using AGPS to examine takeoff clearance problem in mountainous terrain.



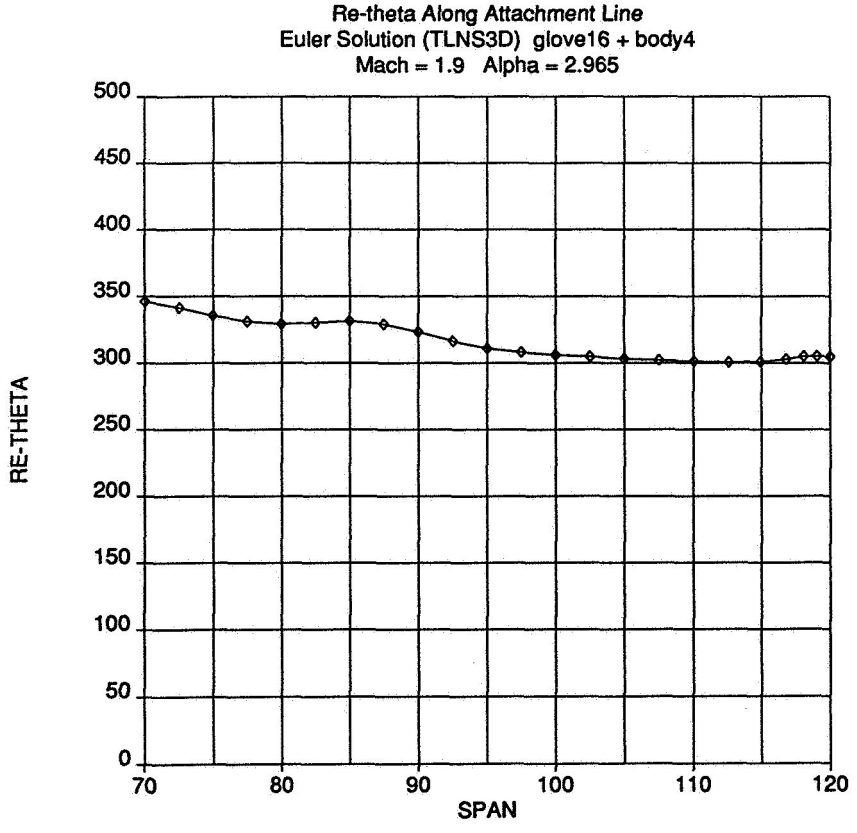


Figure 11. Using AGPS to calculate momentum thickness Reynolds number.

Figure 12. Partial listing of command file for calculating momentum thickness Reynolds Number.

```
!. RE-THETA_TLNS(QFILE=wb8i)
! PURPOSE: Generate the attachment line velocity gradient data and the resulting Retheta data. Cp along the attachment
!          line is also generated. This version of the program is set up for processing the data from the TLNS3D code.
! PREPARED BY: A. Gentry      DATE: 11/14/91 Modified from the STUFF version.
!-----------------------------------------
! INPUT FILES:
!   wing.ggp Output ggp file obtained from the p3dtoggp program selecting only the wing leading edge data region.
!          The TLNS3D data extracted with the p3dtoggp program must be as follows:
!            i 80 to 120 at 1 interval
!            j 1 to 1 at 1 interval
!            k 1 to 30 at 1 interval
!            functions: 114,150,151,152/

! INPUT SYMBOLS:
   $I1=8          ! Station starting value for the surface grid fit.
   $J1=1          ! Upper surface starting grid point for leading edge.
   $J2=70         ! Lower surface end grid point.
   $NA=200    ! Last cross-section on aftbody.
   $SF=0.750  ! Streamline starting s-value, forebody.
   $SA=0.550  ! Streamline starting s-value, aftbody.
   $ISTP=1      ! Stepping interval on the station data.

! INPUT FOR R-THETA CALCULATIONS:
   $EMAC=1.9              ! Flight Mach number.
   $CS=968.07             ! Speed of sound at 44,000ft.
   $ENU=0.000612283     ! Kinematic viscosity (ft2/sec) at 44,000ft.
   $VELO=EMAC*CS     ! Freestream velocity.

!  Z-location of the cuts (at the leading edge point). Create string of 1-D data.
   CST ZC [(62.5),(65),(67.5),(70),(72.5),(75),(77.5),(80),(82.5),(85)]
   CST ZC [(87.5),(90),(92.5),(95),(97.5),(100),(102.5),(105),(107.5)]
   CST ZC [(110),(112.5),(115),(117),(118),(119),(120)]
!-----------------------------------------------
! OUTPUT:
!    RTHE     String of points of Rthets vs. Z (span location).  Output to file rthe_qfile.ggp.
!    ATTCP    String of points of Cp vs Z along the attachment line. Output to file att_cp_qfile.ggp)
!    SVEL     Lists of strings containing the dw/ds normal velocity/Vinfinity derivative data.
!    STRL     List of attachment line point strings.
!                 STRL.1  Forebody attachment line.
!                STRL.2  Aftbody attachment line.
!    CUTS.*   Cuts normal to the attachment line at each of the normal plane stations.
!-----------------------------------------------
$ON ERROR EXIT
!===============================================
! Read the wing leading edge region geometry and surface velocity data from Boeing GGP file:
   GGP QFILE.GGP FG X,Y,Z
   GGP QFILE.GGP FUVW U,V,W
   GGP QFILE.GGP FCP CP

! Thin out the data some to reduce compute time.
      $CALL GET_LENGTH(FG,N)         ! How many strings in FG.
      $CALL GET_LENGTH(FG.1.1,M)   ! How many points per string.
 $FOR I=I1 STEP ISTP TO N-3 DO
    $FOR J=J1 TO M DO
       $CALL GET_COORD(FG.<I>.1.<J>,0,0,X,Y,Z)
       CST QS [(X,Y,Z)]        ! Put  X-Y-Z into a string.
```

Figure 12 (continued).

```
      $CALL GET_COORD(FUVW.<I>.1.<J>,0,0,U,V,W)
      CST QUVW [(U,V,W)]      ! Put U-V-W  into a string.
      $CALL GET_COORD(FCP.<I>.1.<J>,0,0,CP)
      CST QQCP [(CP)]              ! Put Cp into a string.
   $ENDDO
      ATL QSTR QS          ! Put the QS strings into a list.
      ATL QFUVW QUVW     ! Put  U-V-W strings into a list.
      ATL QCP QQCP            ! Put the Cp strings into a list.
        REN [QQCP,QS,QUVW] N=   ! Free up the names so that they can be reused.
$ENDDO
!     DEL [FG.*.*,FG,FUVW.*.*,FUVW]    ! Delete objects no longer needed.
        PURGE   ! Removed ancestors no longer needed.
      RVS QSTR.*          ! Reverse all of the strings.
      RVS QFUVW.*
      RVS QCP.*
      CAY FGA QSTR.*            ! Geometry array.
     CAY FUVWA QFUVW.*  ! Matching u-v-w array.
      CAY FCPA QCP.*            ! Matching Cp array (contains Cp only).
        REN [QCP,QSTR,QFUVW] N=

! Fit a surface to the x-y-z geometry data array.
   FSU FSURF FGA CUBIC       ! Fit surface to the geometry data.
!-------------------------------------------------------
! Build the S-T-Cp array and fit a surface to it.
   ESK FSURF NAA SUBRANGE=YES     ! Extract the array data as subrange points.
      $CALL GET_LENGTH(NAA,N)
      $CALL GET_LENGTH(NAA.1,M)
 $FOR I=1 TO N DO
    $FOR J=1 TO M DO
       $CALL GET_COORD(NAA.<I>.<J>.1,0,0,S,T)
       $CALL GET_COORD(FCPA.<I>.<J>,0,0,CP)
       CST QSCP [(S,T,CP)]
    $ENDDO
       ATL QSSCP QSCP
         REN QSCP N=
 $ENDDO
   CAY AFCP QSSCP.*
   FSU FSCP  AFCP   CUBIC S-PAR=FSURF T-PAR=FSURF ! Surface to Cp.
     DRA FSURF 2 51 5 51 E=YES COL=31 VIEW=0,-75,-10,1.8

! Calculate a streamline along the attachment line.
   FSU FSUVW FUVWA CUBIC S-PAR=FSURF T-PAR=FSURF ! Surface to u-v-w.
   CST STR (SF,1) SPACE=FSURF  ! Starting point for streamline.
   SCS FSURF FSUVW STR.1 STRL -0.1  ! Get the streamline.
     DEL STR
   RVS STRL.1          ! Reverse the string to make it inboard to outboard.
   DRA STRL.* E=NO COL=RED
```

Note: This listing shows about one third of the complete command file.

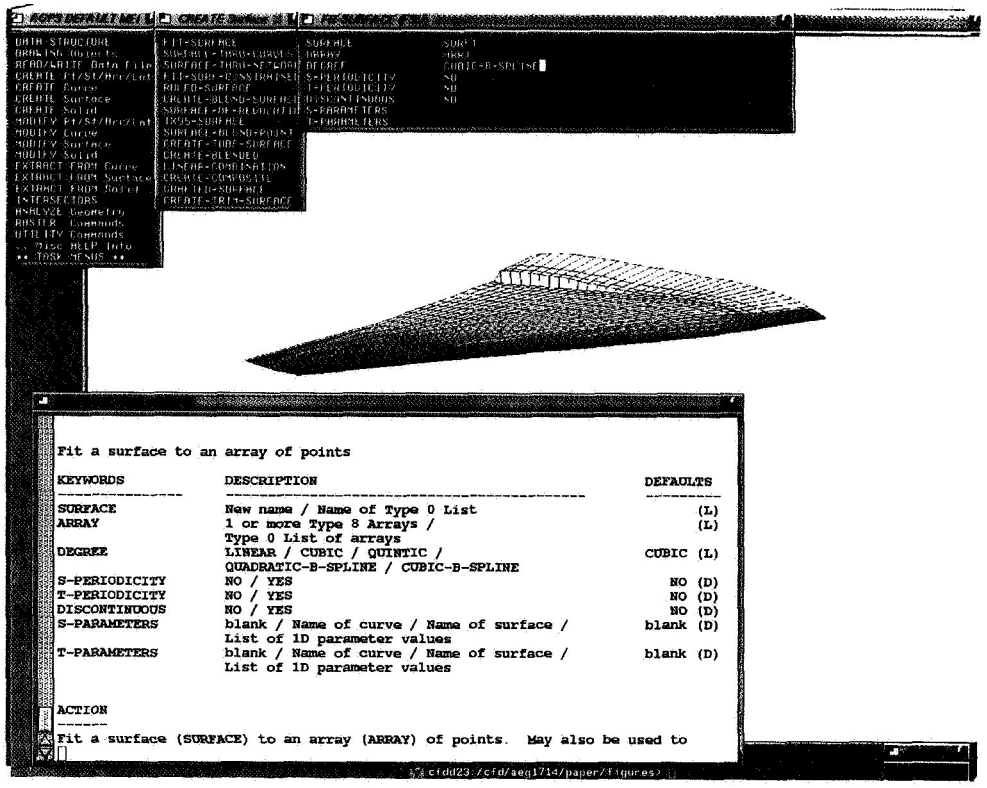


Figure 13. Screen dump of AGPS display showing menu operation.

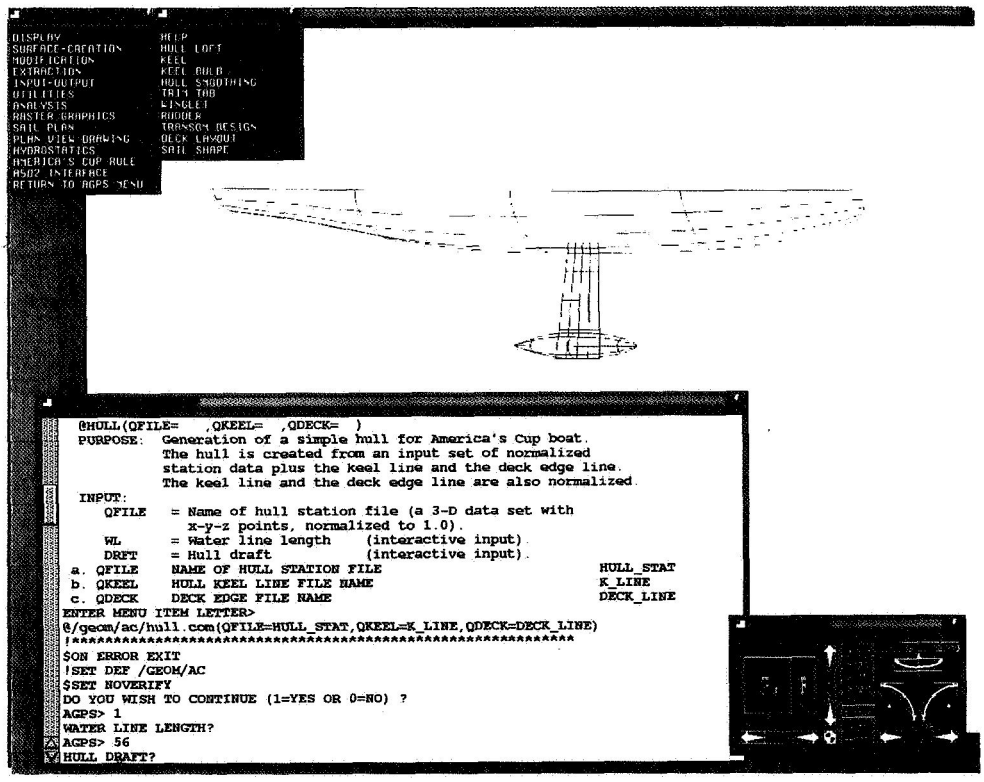


Figure 14. AGPS menu package for designing an America's Cup yacht.

GEOMETRY ACQUISITION AND GRID GENERATION - RECENT EXPERIENCES WITH COMPLEX AIRCRAFT CONFIGURATIONS

Timothy D. Gatzke, Walter F. LaBozzetta,
John W. Cooley, and Gregory P. Finfrock
McDonnell Aircraft Company
St. Louis, MO

ABSTRACT

As Computational Fluid Dynamics (CFD) analysis methods continue to mature, the ability to generate a suitable grid for complex configurations has become the pacing item in the application of CFD to engineering problems. The variety of forms and detail present in the geometry definition of real vehicles residing in a computer aided design system compounds the difficulties and contributes to the time required to generate a grid. This paper will discuss important issues involved in working with complex geometry and evaluate approaches that have been taken to address these issues in the McDonnell Aircraft Computational Grid System and related geometry processing tools. The issues that will be addressed include the efficiency of acquiring a suitable geometry definition, the need to manipulate the geometry, and the time and skill level required to generate the grid while preserving geometric fidelity.

INTRODUCTION

As Computational Fluid Dynamics (CFD) analysis methods continue to mature, they are being applied to problems that are more complex, both geometrically and in the physics of the flowfield. For example, the numerical solution of the Navier-Stokes equations is becoming increasingly important for the analysis of advanced inlets and nozzles and their integration with an airframe. The ability to generate a suitable grid has become the pacing item in the application of CFD to these problems. For a complex configuration, constructing the grid, including geometry acquisition and grid generation, can today require from several weeks to several months of effort.

Before beginning grid generation on a complex configuration, accurate configuration geometry must be available in a suitable form. Until recently, most geometry being analyzed was composed of simple analytical shapes, such as circular, ogive, cylindrical, bodies of revolution, that could be easily generated within a computer program. Another source of geometry data was cross section cuts through the vehicle surfaces. These cuts were defined by strings of points. Today, the geometry definition may be stored in any one of several computer aided design (CAD) systems currently in use throughout industry. Acquiring geometry from a CAD system presents an added complication to the CFD analysis process because most CAD geometry databases use surface definitions which are not readily compatible with most grid generation tools. Therefore, it is necessary to manipulate CAD geometry before grid generation can begin. Although this paper will focus on data that is defined in a CAD system, several of the issues discussed apply to any geometry, independent of origin.

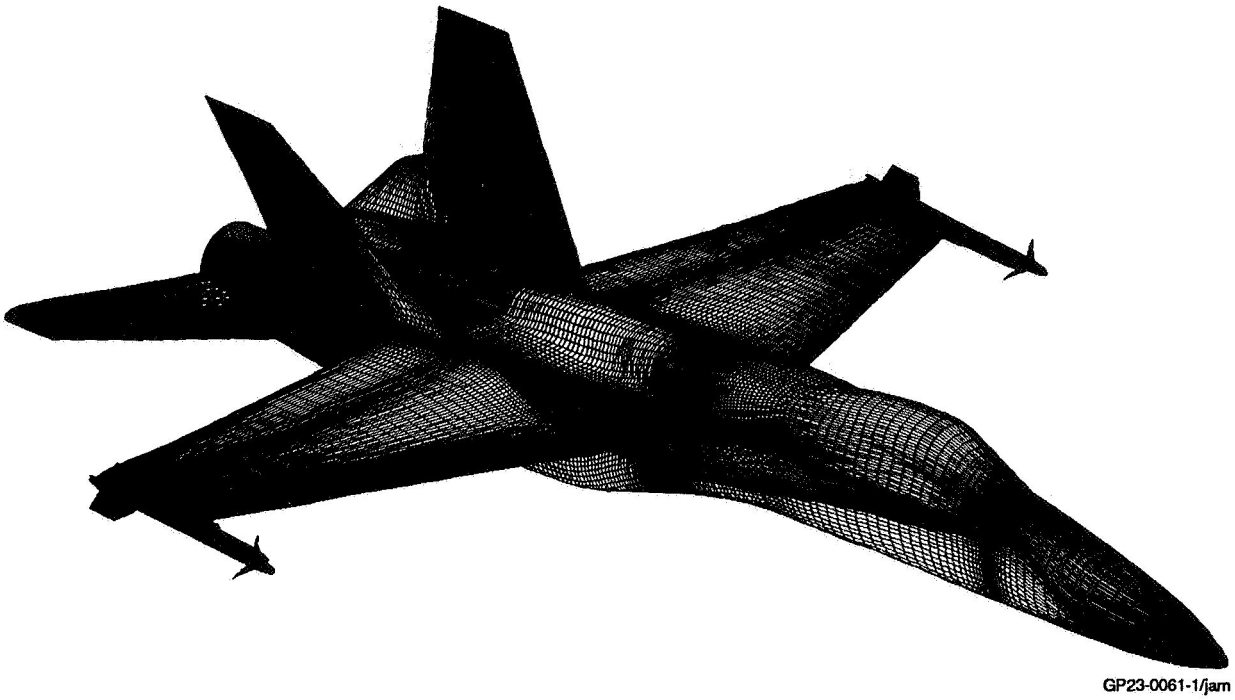
Within the CAD environment, the geometry data can take on varying levels of detail and complexity. The quality of the CAD model can vary as well. Flaws in the model are a practical consideration that must be acknowledged. Also, the CAD system typically provides a large number of different analytic surface types allowing the designer much flexibility in defining vehicle mold lines. This variety of forms and detail, which is present in the geometry definition of real vehicles, compounds the difficulties and contributes to the time required to generate a grid.

There are many issues involved in generating the grid for a complex geometry. These include the efficiency of getting the geometry into a grid generation system, the need to manipulate the geometric data, and the time and skill level required to generate the grid while preserving the fidelity of the geometry. Several approaches have been taken to address these issues in the McDonnell Aircraft Computational Grid System (MACGS) and related geometry processing tools. This paper will discuss the various approaches that have been implemented and identify what has been learned from their implementation.

BACKGROUND

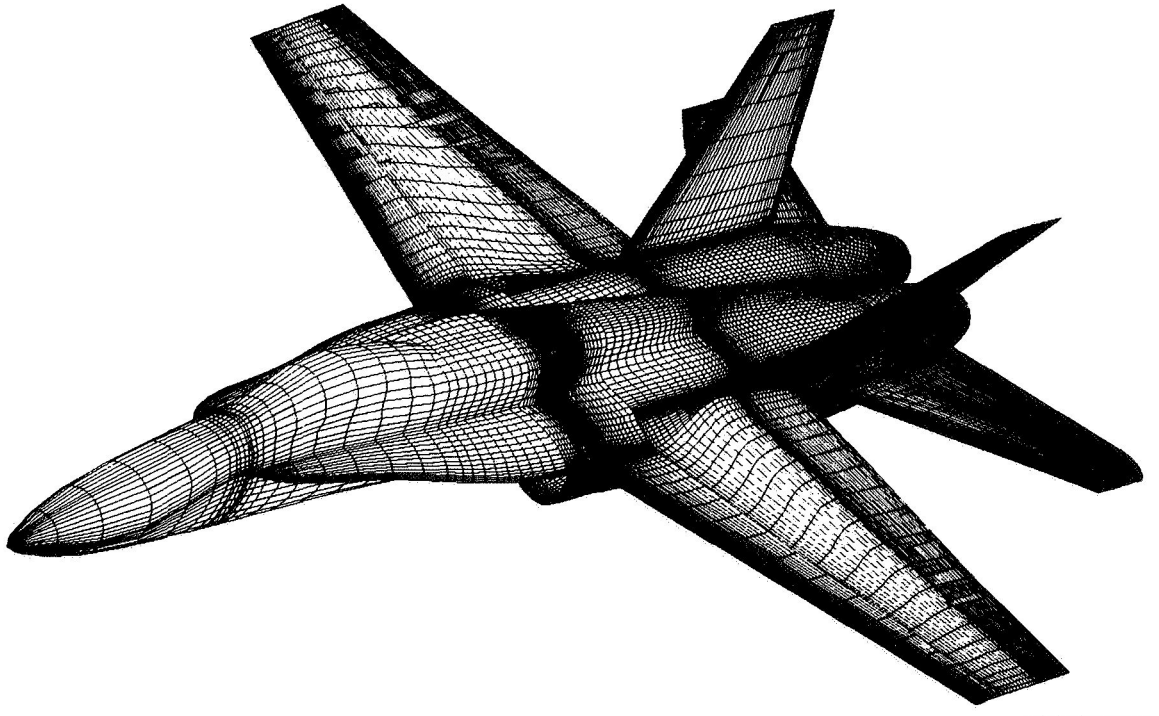
The complexity of geometry has progressed from problems such as wings, isolated forebodies, isolated inlets, and blended body configurations, to full vehicles integrating internal and external flow. Grid generation has progressed from batch methods aimed at a particular configuration, to batch methods able to handle arbitrary configurations, and most recently to interactive graphical tools for complex grid generation tasks. Batch methods for arbitrary topologies, such as EAGLE (Reference 1), gave the user more flexibility than previous methods, but still did not give the user rapid feedback during the decision making process of setting up the batch code inputs. Interactive graphical tools for grid generation, such as the Wright Laboratory (WL) GRIDGEN system (Reference 2), provide improved grids by giving the user more direct control of and feedback from the grid generation process. This is especially important when gridding a new complex configuration. The WL Interactive Graphics for Geometry Generation (I3G) program (Reference 3) was developed to manipulate geometry primarily for input to panel codes and has also been used as the basis for the WL VIRGO grid generation code (Reference 4). These methods have improved grid generation capability but need to address how complex geometry will be obtained.

One of the main modules of MACGS (Reference 5) is based in part on the I3G program. Extensive modifications to I3G at the McDonnell Aircraft Company (MCAIR) have enhanced its use as an arbitrary topology three-dimensional grid generation program, while retaining its ability to manipulate geometry data. MACGS uses an interactive zonal approach which does not require point continuity at zone-to-zone interfaces. This approach is compatible with MCAIR preferred flow solvers. The zonal approach maximizes flexibility and simplifies grid generation by subdividing the physical domain into simpler regions or zones. Figure 1 shows surface grids for a fighter aircraft configuration with a wing tip missile. This Euler grid contained 17 zones with point-continuity at zone interfaces, and required about 3.5 million grid points. In contrast, Figure 2 shows surface grids for a fighter aircraft configuration which included inlet and nozzle geometry. This Navier-Stokes grid contained 17 zones without point-continuity at zone interfaces, and required about 2.6 million grid points. This illustrates that point-mismatch zone interfaces can significantly reduce the number of grid points.



GP23-0061-1/jam

Figure 1. Surface grids for a fighter aircraft configuration with wing tip missile – point continuity at zone interfaces required by the flow solver.



GP23-0061-2/jam

Figure 2. Surface grids for a fighter aircraft configuration with inlet and nozzle – point mismatch at zone interfaces.

In order to facilitate geometry inputs to MACGS, several tools have been developed that allow the user easier access to the CAD geometry database and provide the ability to acquire the geometry data. These include interfaces to specific CAD systems which aid the user in preparing data for the grid generation process, and a tool to process the output of the CAD systems into a form that can be used by the grid generation system.

THE MACGS GRID GENERATION SYSTEM

MACGS is divided into three main modules: a boundary grid generation module, ZONI3G; a field grid generation module, GMAN; and a grid processing module, GPRO. This paper will address approaches that have been implemented in the ZONI3G and GMAN modules. ZONI3G provides the following capabilities:

1) Access to geometry data in various formats for interfacing flexibility.
2) General surface construction and manipulation tools.
3) Surface grid generation tools (algebraic and elliptic) for creating grids on the zone faces.
4) A wide range of grid point distribution functions.
5) Zone consistency checking for reducing user workload by forcing faces to match at edges (and edges to match at corners) of the zone.

GMAN provides the following capabilities:

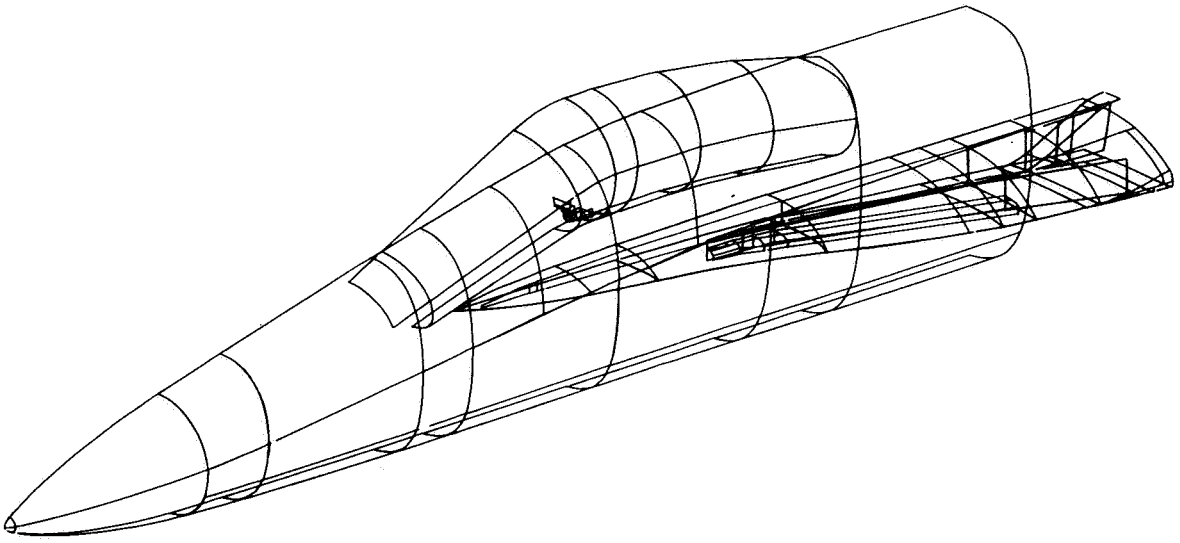
1) Algebraic and elliptic three-dimensional field grid generation methods.
2) Three-dimensional grid quality assessment tools.
3) Interactive specification of boundary conditions on any boundary or subset of a boundary.
4) Efficient computation of the interpolation factors for each point on a zone interface of one zone, relative to points located on the zone interface of the adjacent zone (for point-match or point-mismatch interfaces).
5) Output formats directly compatible with zonal Euler or Navier-Stokes flowfield prediction codes.

ZONI3G and GMAN are interactive, graphical, menu-driven modules that allow the user to work with geometry and grids in a user-friendly environment. Features of MACGS that affect the processing of geometry and the overall efficiency of the grid generation process will be described in subsequent sections.

GEOMETRY ACQUISITION ISSUES

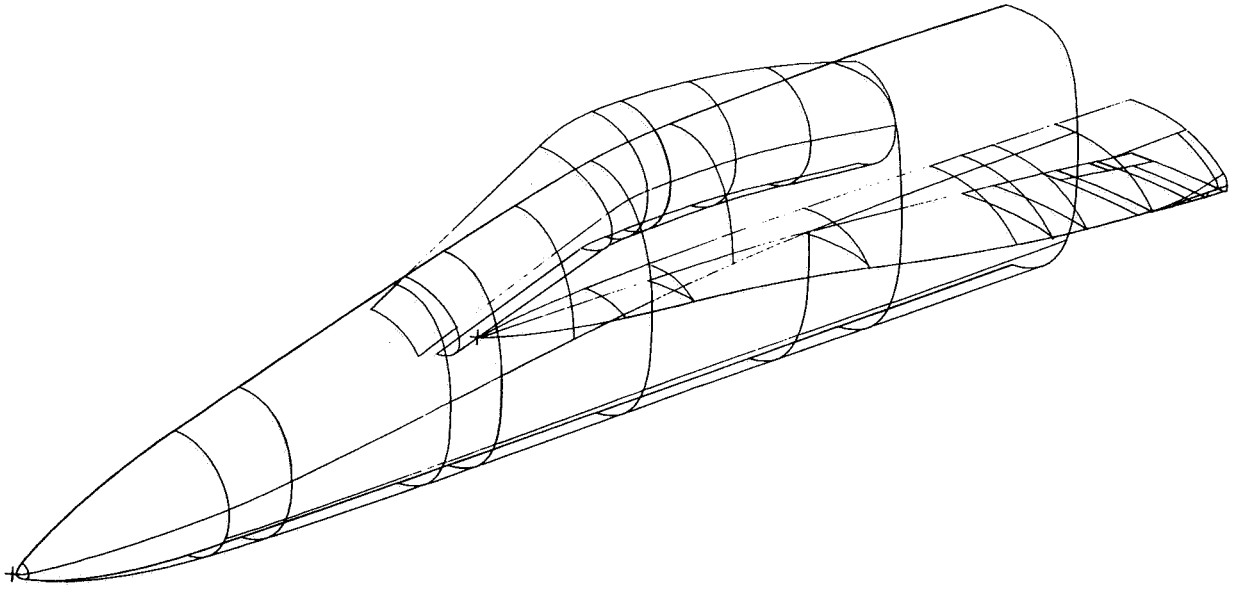
The initial step in the geometry acquisition process is the retrieval and inspection of the CAD model. In many instances, such as the surfaced forebody/LEX model shown in Figure 3, the CAD model contains surface definitions that are unneeded for the CFD analysis. For such a configuration the CAD operator should simplify the model to avoid unwanted section cut definitions in the ensuing step. Simplifying a surface definition may include eliminating surfaces in various areas. This often occurs with protuberances such as antennae, lights, exposed hinges and actuators, or armament fixtures. The CAD operator should have a rough idea of the emphasis of the CFD analysis and should eliminate the protuberances which are nuisances to the engineer generating the grid. If the elimination of protuberances

results in surface holes or mismatch, the CAD operator should correct the definition by creating simple surfaces in these areas. The simplified surfaced model is shown in Figure 4. Modifications to the surfaced model may also be necessary due to incomplete or poor surface definitions. Again the CAD operator should correct the holes and mismatched areas with simple surfaces.



GP23-0061-3/jam

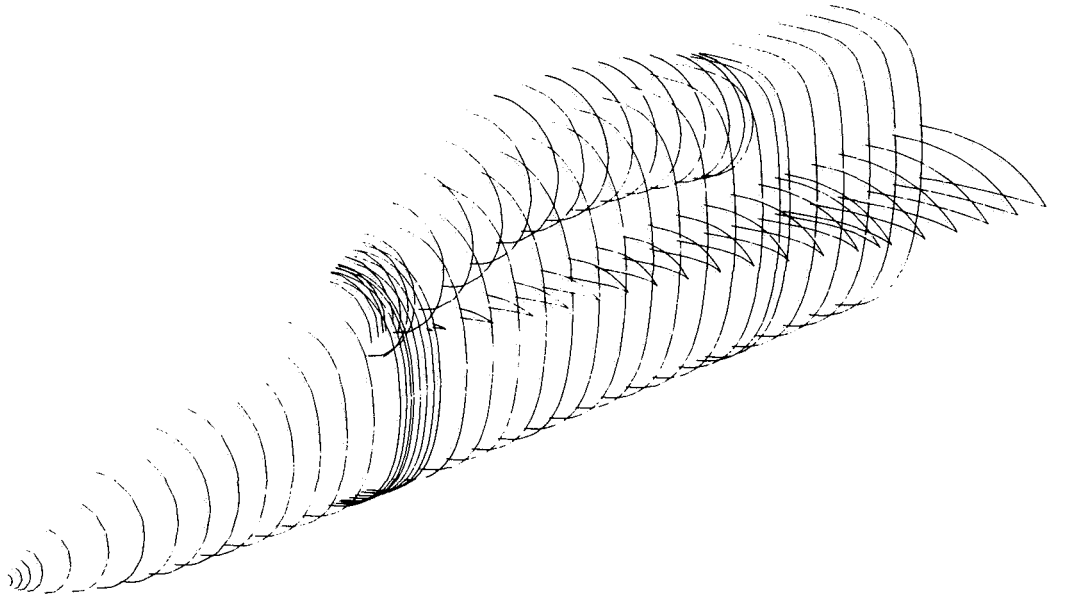
Figure 3. CAD surface model of forebody/LEX configuration.



GP23-0061-4/jam

Figure 4. Simplified CAD definition of forebody/LEX configuration.

For MCAIR CFD applications work, transitioning the CAD model surfaces to point surface definitions is done through the use of an intermediate model definition, composed of sets of analytical curves on the model surfaces. Most of the CAD curve generation functions can be used to generate the sets of curves. Most commonly, these are generated by passing a sequence of cutting planes through a set of surfaces defining the configuration. The user can locate the planes in the orientation that best defines the geometry, and can choose an appropriate spacing interval between planes for the local geometry detail. The number and type of curves created are dependent upon the CAD system, the surface type being cut, and the surface tolerance specified. Often it is necessary to trim curves because they lie on surfaces which intersect (this is usually easier than trimming a CAD surface to an intersection). Figure 5 shows the forebody/LEX configuration with surface curves at several locations. The untrimmed curves illustrate how CAD surfaces intersect one another. Figure 6 shows a set of surface curves, after all curves have been trimmed to the surface intersections.



GP23-0061-5/jam

Figure 5. Selected surface curve cuts on CAD definition of forebody/LEX configuration.

Most analytical curves have Initial Graphics Exchange Specification (IGES) (Reference 6) equivalents. Thus, the curve set surface definitions can be transferred from the CAD environment into the grid generation environment through the use of a CAD-to-IGES translator. PCPROC (originally developed at Douglas Aircraft Company and enhanced at MCAIR) is a tool that reads, reorganizes, and outputs curves using the IGES format. IGES entity curve types are converted (at user-controlled tolerance) to parametric cubic spline entities upon being read into PCPROC. Individual curves or entire curve sets can then be merged, sorted, transformed, or otherwise manipulated by using the versatile parametric cubic splines. PCPROC allows the user to generate sets of points on the curve sets using a variety of distribution functions such as equal arc, hyperbolic sine, hyperbolic tangent, and curvature dependent spacing. The end points of curve segments can be preserved, as well as the location of any slope discontinuities. The point surfaces or curves generated can then be written out as a rectangular surface grid for input to MACGS. Once a detailed set of curves has been generated to define a

configuration, it can be saved for future analyses. These sets of curves can then be accessed more quickly than the CAD database if different distributions of points, or some merging of the geometry for a redesigned component such as a canopy, inlet, or wing is required.

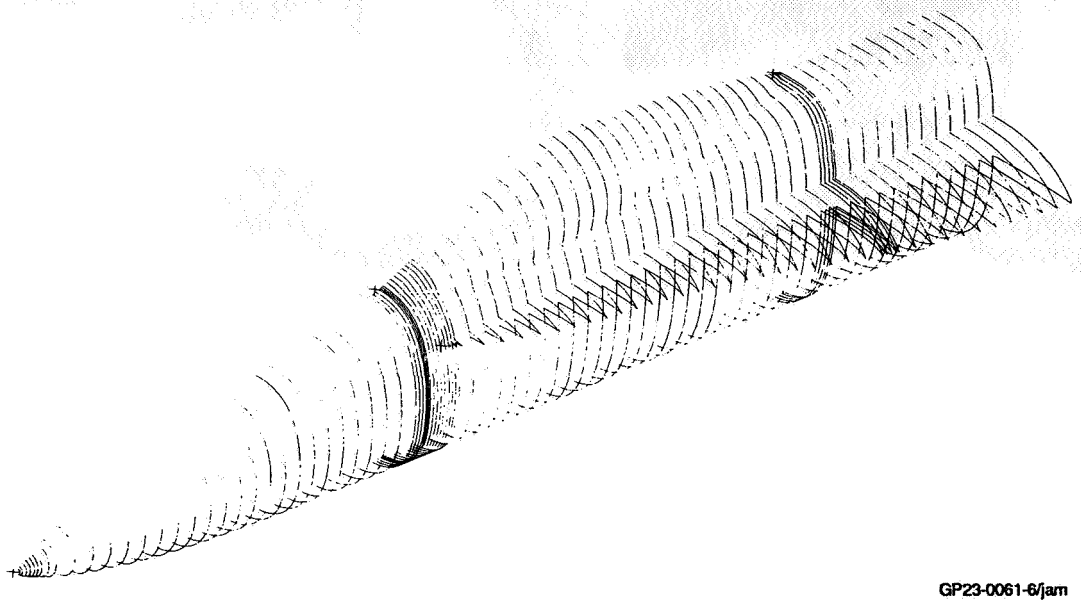
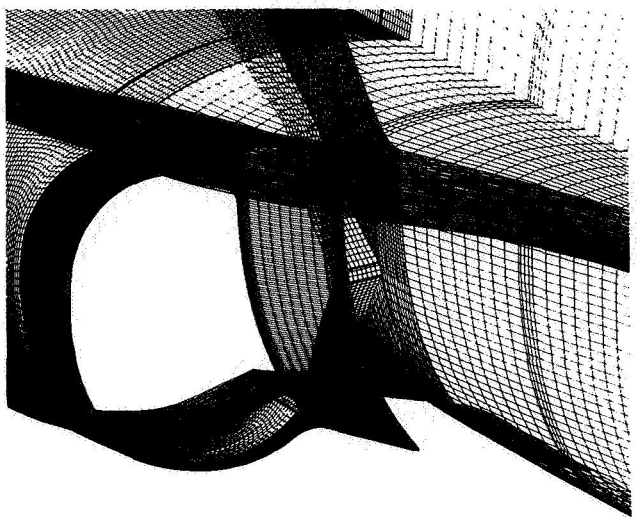


Figure 6. Trimmed analytic curve definition of forebody/LEX configuration.

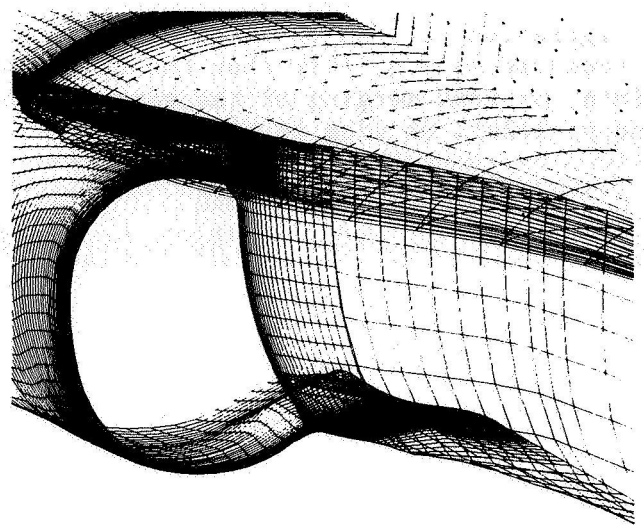
GEOMETRY MANIPULATION ISSUES

Identifying the optimum environment for manipulating geometry into the form desired for the final grid is an important geometry/grid system issue. Depending on the CFD analysis to be performed, the grid for a complex geometry may include faired inlets, nozzles, or boundary layer diverters. It may even involve the removal of a major component such as a wing, with a faired "cap" in its place. Minor modifications such as collapsing finite thickness trailing edges or extending horizontal tail trailing edges to the fuselage intersection must be dealt with on an everyday basis. Experience has shown that one-third to one-half of the time between obtaining point surfaces from CAD and completing the grid is spent manipulating the geometry into the form desired for the application at hand. Figure 7 shows the complex geometry of the nacelle/boundary layer diverter/LEX area of a fighter aircraft. Since the diverter may not be required for a particular analysis, a fairing, as shown in Figure 8, must be defined, maintaining as much of the original geometry definition as possible. Such geometry generation tasks are well suited to being defined in the CAD environment, with its wide range of surface generation capabilities. However, this requires an upstream communication between the CAD operator and the engineer generating the grid to define what faired or filled surfaces must be generated in the CAD system. Often, the CAD geometry is already defined when a new CFD analysis is to begin. The options in this situation are to have a CAD operator fabricate the desired faired surfaces, train the engineer generating the grid to operate the CAD system and let him create the geometry himself, or allow the creation of these surfaces in the "grid generation" phase, using grid generation system tools to modify the geometry acquired from the CAD system.



GP23-0061-7/jam

Figure 7. Nacelle/boundary layer diverter/LEX geometry for a fighter aircraft.



GP23-0061-8/jam

Figure 8. Nacelle/faired boundary layer diverter/LEX grid for a fighter aircraft.

The latter approach is taken in MACGS. CAD geometry of the configuration is modeled during the geometry acquisition phase of the CFD analysis. The resulting analytical curve definition is stored in a central location, to be used as a geometry source for future grids. Depending on the objectives of the current CFD analysis, the engineer may refine the configuration to suit his needs using the point geometry manipulation tools in ZONI3G. If a subsequent CFD analysis on the same vehicle geometry but with different objectives is needed, the original geometry is available without accessing the CAD system.

ZONI3G contains a wide range of geometry manipulation tools (presently for point definition surfaces), allowing surface-to-surface intersections, breaking surfaces at intersection curves, and creating surfaces in space within boundary curves. This approach puts the creation of fairings in the hands of the engineer generating the grid, who best understands what modifications should be made for the analysis at hand. Since the shape and definition of these surfaces may not be critical to the analysis, this approach may be more time and cost effective than creating these surfaces inside the CAD system.

MCAIR experience has shown that CFD engineers prefer having manipulation capability in the grid generation system because the need to modify the geometry may not be apparent until the grid generation process is well underway. However, this approach leaves no clear-cut distinction between the tool which should be used for geometry definition and the one to be used for grid generation. A complex aspect of this approach involves maintaining geometry integrity within an environment in which the user is free to change the shape of components by fairing or filling gaps, but wants to maintain the CAD definition of other components.

Geometry integrity is not as simple as ensuring that surface grid points lie on a CAD database geometry definition. Many of the surfaces to be gridded do not reside in the CAD system in their modified form. Figure 9 shows the area near the junction of a fighter nozzle and horizontal tail. The points highlighted by dots represent grid points lying on an artificial extension of the horizontal tail into the nozzle. These points cannot be referenced to any CAD database surface since in the actual geometry there is a gap between the tail and nozzle in this region.

Giving the CFD engineer the ability to fair and modify geometry within the grid generation tool inherently gives him the ability to lose the same geometry integrity the grid generation system must strive to maintain. As such, this approach assumes that the CFD engineer must take some responsibility for maintaining geometry integrity. The grid generation system alone cannot absolutely ensure that CAD geometry integrity has been maintained. If this is unacceptable, the CFD engineer must have a complete definition of the faired, filled, or otherwise modified geometry to be gridded available in CAD, and must also accept the additional upstream time or training which this requires.

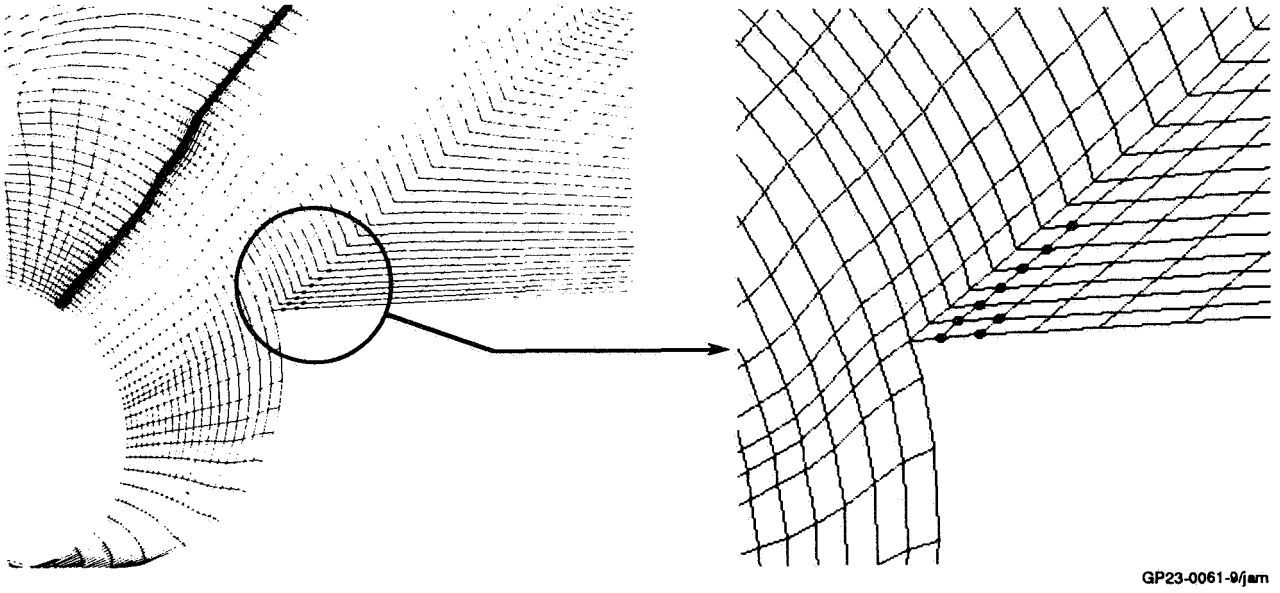


Figure 9. Points on horizontal tail extension not residing in CAD database.

GRID GENERATION ISSUES

Once the geometry has been obtained and is ready for grid generation, the issues to be addressed fall primarily into one of two categories: the preservation of the geometry during the grid generation process, and the overall efficiency of that process. One of the barriers to wider acceptance of CFD in the engineering community is the need to produce usable results in a timely manner (as defined by project goals and schedules). As mentioned previously, interactive methods provide the flexibility and immediate feedback to the user that is necessary for complex geometries. But to meet aggressive schedules, the grid generation system needs to assist the user throughout the process. This includes acquiring the geometry, generating face grids, putting faces together to form zones, generating the field grid, checking the quality of the grid, setting boundary conditions, and calculating interpolation factors at zone interfaces. Automation of much of the process is a key to future gains in grid generation efficiency.

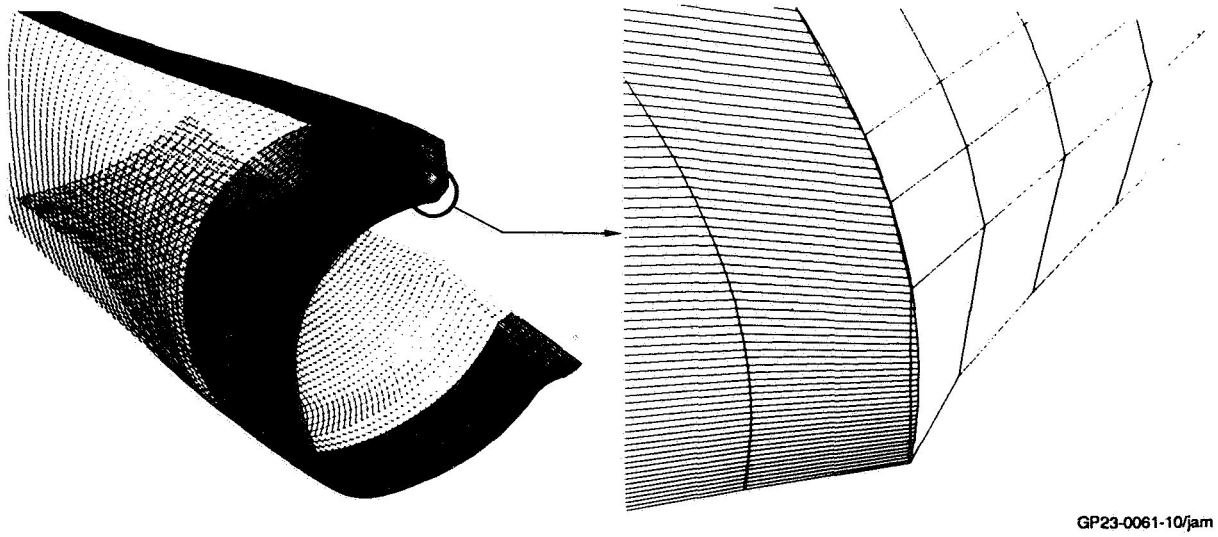
One of the main goals of ZONI3G is to help the user preserve the surface geometry integrity during the grid generation process. One approach is to distinguish between the surface definition and the grid distribution. Once surface geometry has been read into ZONI3G or created within ZONI3G, any subsequent grid

generation operation on that surface will refer back to the original surface definition. This prevents degradation of the surface geometry through successive gridding operations and thus preserves the integrity of the surface geometry. This was implemented by storing two types of surfaces within ZONI3G. The first type is a physical surface which is defined by physical coordinates. This type is used for the initial geometry that is read into the program as well as some surfaces created in the program from scratch. The second type is a parametric surface which is defined by parametric coordinates referencing one or more physical surfaces. All ZONI3G operations can be performed on both types of surfaces.

Operating on a parametric surface has distinct advantages for preservation of the geometry. An operation on the parametric coordinates will return parametric coordinates, ensuring that the new surface is on the original surface definition. This bookkeeping is transparent to the user. No subsequent operations, such as projecting a surface onto a geometry database, are necessary to preserve the geometry.

This approach works well for parametric surfaces that reference a single physical surface. However, after several operations, a parametric surface may reference several physical surfaces. Operations on parametric surfaces referencing multiple physical surfaces are difficult because the parametric coordinates defining the surface reference different coordinate systems, i.e., the local parametric coordinate system of each underlying geometry.

Another difficult area is the location where surfaces with different underlying geometries meet. It must be decided how points that lie between the underlying geometries should be specified and which geometry, if any, they will reference. This problem is apparent in Figure 10 which shows the geometry for a typical nacelle cowl-fuselage intersection. The high curvature of the nacelle cowl leading edge is well represented on the cowl surface, but not on the abutting fuselage surface. Distributing points on the splines of the two surfaces results in a mismatch due to the differing initial curve definition. Resolving these mismatches while maintaining underlying geometry can be very difficult. This issue becomes especially troublesome with viscous grids where a small difference in the geometry can be large compared to the grid point spacing.



GP23-0061-10/jam

Figure 10. Nacelle cowl – fuselage intersection geometry.

Another disadvantage to utilizing parametric surfaces in addition to physical surfaces is that it increases the size of the program, both in lines of source code and memory requirements, and increases development time. This approach can also decrease the efficiency of the system since conversions between physical and parametric surfaces may be required as additional steps in many operations. Further development is needed in order to maintain the geometry throughout the process without sacrificing efficiency.

ZONI3G operates on curves and surfaces. For interpolating between points, a curve is represented by an equation fit through the defining points. For a surface, curves in each direction can be fit. Originally ZONI3G used a nonlinear Akima fit (Reference 7). Since surfaces within ZONI3G are defined as a set of curves, it is not necessary for each curve to have the same number of points. The ability to operate on surfaces made up of curves with different numbers of points on each curve is very useful, since curves defining the geometry that are generated in CAD systems can be represented this way. Grid generation can proceed directly on a surface of this type without requiring an intermediate surface of a different form to be generated. As a result, one level of manipulation that could change the original geometry is eliminated.

For surfaces with the same number of points on each curve, an optional bicubic spline representation was also investigated. This method is the same as that used in EAGLE. One advantage of this representation is that the computations to convert from physical coordinates to parametric coordinates and back are faster. This method is acceptable for surfaces defined by points that are fairly evenly distributed, however the parametric space is highly dependent on the initial point distribution. This dependency can be overcome by scaling the parametric space coordinates using the arc lengths of the curves defining the surface. However, this approach has an associated sacrifice in speed. Even after scaling based on arc lengths, the Akima parametric space is still slightly less dependent on the curves that define the surface. The Akima surface representation also has the advantage of being a local fit. As a result, the surface representation in one area is not affected by the movement of a distant point on the surface. In general, the Akima surface representation produces fewer wiggles than the bicubic spline method and results in a better representation for realistic geometries. For either type of representation, where new distributions of points are specified, it is the local parametric values and the type of parametric space used which are stored, and not the new physical coordinates themselves.

Many complex configurations contain slope discontinuities. A nonlinear fit of these surfaces will model these discontinuities poorly, and introduces wiggles in the surface definition. To address this problem, a linear fit option was added to the surface representations in ZONI3G. A curve can be represented by either a linear or nonlinear fit through the defining points. For a surface, each direction can be represented by either a linear or nonlinear fit through the defining points, as specified by the user. A linear/nonlinear flag is stored in the definition of parametric surface types so that future operations on these surfaces use the appropriate fit. This option improves the integrity of surfaces near a discontinuity and has worked well for geometries containing both smooth surfaces and surfaces with discontinuities.

The efficiency of the process is another key issue in grid generation. The interactive graphical approach is a significant step toward improving efficiency. It is extremely beneficial to see the surface grid point distribution immediately rather than awaiting completion of a batch job. However, the interactive

environment by itself cannot completely solve the efficiency issue. With some previous grid generation codes, the user was responsible for orienting the six computational faces that define a three-dimensional structured grid zone. If the user is not meticulous, several attempts may be needed to get all of the surfaces correct. The ZONI3G zone entity improves three-dimensional zonal grid generation capability by grouping the boundary geometry and grids associated with each of the six computational faces of a zone into a single entity. The process of generating a zone in MACGS includes automated reordering of faces once the zone orientation has been established, and checking for point match along the common edge of adjacent faces.

In some cases, once the user has defined certain faces of a zone, defining the remaining faces is straightforward and the interactive specification of these remaining faces may be tedious. The ZONI3G user can accelerate this process by generating grids for all of the remaining faces of a zone automatically. Missing edges are generated as straight lines between corner points, and then each missing face is interpolated from its edge grids using transfinite interpolation.

For generating the 3-D field grid in the interactive environment, algebraic methods are very attractive because of their speed. On low-end workstations, elliptic methods are often too time-consuming to apply to the entire grid. To reduce time and computer requirements, the GMAN user can apply elliptic methods to a limited local region of the grid where the algebraic methods are inadequate. GMAN uses several algebraic and elliptic methods to locally refine a grid. These operations can be performed over a user-selected range of the grid, with modifications displayed immediately. Algebraic methods are adequate for most of the grid as long as the boundary grids are reasonable. In those cases, local elliptic refinement can be used to efficiently correct most flaws.

Grid quality assessment and correction improves the performance of the entire solution process by eliminating wasted flow solver computation time due to flawed grids and increasing solution convergence rate by providing a higher quality grid. Negative volume, crossed side, collapsed face, and zero volume checks of the QBERT grid evaluation code (Reference 8), developed at WL, have been implemented in GMAN. However, MCAIR experience shows that satisfying these checks is not always sufficient to ensure that the flow solver will run. Additional orthogonality and smoothness measures are being investigated. MCAIR experience has shown that if care is taken to ensure good boundary grids, generating the interior grids of the zones is fairly straightforward and requires minimal grid refinement. Therefore, grid quality checking for surfaces is planned for insertion into ZONI3G.

CONCLUSIONS

Acquiring a suitable geometry definition from a CAD system is a significant part of the overall process of generating a grid. The details that exist in the CAD model and the numerous forms that the surface can take complicate the process. The grid system must work with the CAD surfaces directly, or it must accurately convert analytical surface data in the CAD environment to discrete surface definitions in the grid generation environment.

Geometry manipulation is an integral part of the CFD grid generation process. Geometry manipulation currently occupies up to half of the time required to generate a grid for a complex configuration. Geometry manipulation can be done

either within the CAD system or within the grid generation system. The MACGS approach provides manipulation capability in the grid generation system. Therefore, the CFD engineer need not be an expert in the use of the CAD systems, and can often create the modified surfaces in a more timely manner than waiting for the availability of the CAD operator. Precisely maintaining the geometry integrity of these modifications is not critical. Thus, an associated CAD geometry is not required.

Maintaining geometry integrity of critical surfaces during the grid generation process, especially for the novice at generating grids, must be a concern, for any grid generation system. If provisions for automatically maintaining geometry integrity are included in the grid system, as is the case with MACGS, degradation of system efficiency needs to be minimized. After the geometry acquisition, manipulation, and preservation issues have been addressed, automation of the process to generate the grid is the critical factor. Automation of some grid generation steps has been shown to reduce grid generation time by 20-25%. Interactive operation will continue to be important to give the user direct control throughout the grid generation process. Improvement and/or automation in areas such as grid quality assessment techniques is needed to provide further reductions in grid generation time.

ACKNOWLEDGEMENTS

This research was partially supported by the McDonnell Douglas Independent Research and Development Program.

REFERENCES

1. Thompson, J. F.: Program EAGLE User's Manual. AFATL-TR-88-117, Vols. I-III, Sept. 1988.

2. Steinbrenner, J. P.; Chawner, J. R.; and Fouts, C. L.: The GRIDGEN 3D Multiple Block Grid Generation System. WRDC-TR-90-3022, Vol. I, II, July 1990.

3. LaBozzetta, W. F.; Cole, P. E.; Kreis, R. I.; and Finfrock, G. P.: Configuration Data Management System - System Documentation. AFWAL-TR-87-3064, Vol. I, II, December 1987.

4. Amdahl, Lt. D. A.: Interactive Multi-Block Grid Generation. The Second International Conference on Numerical Grid Generation in Computational Fluid Dynamics. Miami Beach, Fl., November 1988.

5. Gatzke, T. G.; LaBozzetta, W. F.; Finfrock, G. P.; Johnson, J. A.; and Romer, W. W.: MACGS: A Zonal Grid Generation System for Complex Aero-Propulsion Configurations. AIAA-91-2156, June 1991.

6. Smith, B.; and Wellington, Jr.: Initial Graphics Exchange Specification (IGES), Version 3.0. NBSIR-86-3359, April 1986.

7. Akima, H.: A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures. Journal of the Association for Computing Machinery, vol. 17, no. 4, October 1970, pp. 589-602.

8. Strang, W. Z.: QBERT: A Grid Evaluation Code. AFWAL-TM-88-193, July 1988.

SDL: A SURFACE DESCRIPTION LANGUAGE

Raymond C. Maple, Capt USAF
Wright Laboratory, Weapon Flight Mechanics Division
Eglin AFB, FL

SUMMARY

A new interpreted language specifically designed for surface grid generation is introduced. Many unique aspects of the language are discussed, including the farray, vector, curve and surface data types and the operators used to manipulate them. Custom subroutine libraries written in the language are used to easily build surface grids for generic missile shapes.

INTRODUCTION

Most new software packages for surface modeling and grid generation attempt to ease the process by providing a graphical "point and click" environment for surface construction. While this type of environment provides many advantages over previous "runstream" type methods, it does have some shortcomings. Flexibility and power are often sacrificed for ease of use. Total reliance on the user interface often means repeating tedious sequences over and over again. Finally, complex user interfaces generally make modification and extension of the surface generation code very difficult. The Surface Description Language (SDL) was written for those situations where the need for power, flexibilty, and extensibility outweighs the need for an easy to use interface.

THE SDL LANGUAGE

SDL is a portable high level interpreted language optimized for the manipulation of one and two dimensional arrays of three–dimensional vectors. The language has its roots in text based grid generators such as EAGLE (Ref 1), but improves in them by providing a more flexible, powerful command syntax. With SDL, surface grid generation is approached with the same logic and problem solving process used when writing a computer program in FORTRAN or C, but SDL simplifies the task by providing language features specifically designed for surface generation.

Like most conventional computer languages, SDL provides named variables, looping and branching (if–then) constructs, and user defined subroutines. While these features give SDL much of its power and flexibility, it is the unique variable types, coupled with the easy manipulation of these types, that make SDL particularly well suited to surface grid generation.

Farrays, Vectors, Curves, and Surfaces

One of the difficulties encountered when using a conventional computer language to generate a surface grid is that such languages are designed to deal with only one floating point number at a time, whereas curves and surfaces are made up of many triplets of numbers. Curves and surfaces must be represented by n x 3 or m x n x 3 arrays of real numbers, which can be difficult for the novice or occasional programmer to manipulate. SDL remedies this problem by providing vectors, curves, and surfaces as basic data types which can be referenced as a single entity. An additional data type, the farray, allows one to treat arrays of floats in a similar manner.

Farrays. Because singly dimensioned arrays of real numbers are often required to specify distributions, etc., the farray (pronounced "f array") data type is provided. This is a one–dimensional array of floating point numbers. Most arithmetic operators are defined for farrays.

Vectors. The vector is a single entity representing a triplet of floating point numbers. It is the basic component from which curves and surfaces are constructed. Many basic operators, including "+", "-", "*" and "/" understand the structure of a vector and perform the correct vector or scalar operation when used in expressions. In addition, special vector operators, such as ".." (vector dot product), ".*" (vector cross product), and "|*vector*|" (vector magnitude) are defined. Individual components of a vector can be accessed by using ".x", ".y", and ".z" suffixes.

Curves. Curves are one–dimensional collections of vectors. As with farrays and vectors, the basic structure of a curve is known to SDL and thus to the basic operators. Curves can appear in expressions involving scalars, farrays, vectors, and other curves (or parts of curves) with the same dimension. The dimension of a curve can be determined at any time by means of the ".d1" suffix. The vector component suffixes can be used with curves, and result in farrays.

Surfaces. Surfaces are two-dimensional collections of vectors that share the same properties as curves. Dimensions are referenced through the ".d1" and ".d2" suffixes. Examples involving vectors, curves, and surfaces in expressions are given later in this paper.

Parametric Curves and Surfaces

In addition to the basic three-space curves and surfaces, data types for parametric curves and surfaces are defined. Objects declared with one of these data types (pcurve or psurface) are independant of any particular parameterization. They are an abstract representation of a curve or surface with parameter(s) varying between zero and one. The actual form of parameterization is remembered when a parametric variable is assigned, and taken into account when the variable is used to distribute points in three-space. Only one operator, the "=" assignment operator, recognizes the parametric data types. Currently implemented parameterizations include linear interpolations and cubic (bicubic) splines.

Other Data Types

Ints and Floats. SDL provides for variables with the traditional integer (int) and floating point (float) data types. Variables with these types are single values, and cannot be used to form arrays.

Strings and Files. The string and file data types are special purpose data types that hold quoted string and file IO "unit" numbers respectively.

Parts of Objects: the Range Operator

With conventional computer languages, components of an array are referenced by providing array indices. These indices are used to reference a single array element. Working with a portion of a "curve" or "surface" involves looping over the proper array indices. SDL extends the concept of the array index and provides the range operator, which allows one to specify a range of indices with one term.

While parts of curves and surfaces referenced with the range operator can be thought of as array elements, it is more useful to think of them as vectors, sub–curves, and sub–surfaces. When a single element of a curve or surface is referenced, i.e. simple indices are used, that element functions as a vector. When a range operator is used with a curve, the result is functionally a curve of smaller or equal dimension. The results with a surface depend on whether the range operator is used in one index (result is a curve) or both (result is a surface).

The range operator is simple to use. Instead of referencing a single element, as in `c[7]`, (SDL uses square brackets for indexing) one specifies the beginning and end of the desired range separated by a colon, as in `c[3:7]`. If no index is given, as in `s[1,]`, where `s` is a surface and no index or range is given for the second index, then the entire range for that dimension is assumed. (Therefore the example specifies the entire first i line in surface `s`, which is functionally a curve.)

The range operator can be used with any dimensioned object, including farrays, curves, and surfaces. Because the resulting sub–objects are functionally identical to their "complete" counterparts, sub–objects can be used anywhere a complete object is appropriate. This makes it easy to restrict operations to portions of an object.

Expressions Using Farrays, Vectors, Curves, and Surfaces

As was previously mentioned, many arithmetic operators are defined for farrays, vectors, curves, and surfaces. In this section, several examples of expressions using these objects are provided. In the following examples, `v` is a variable of type vector, `c` is a variable of type curve, and `s` is a variable of type surface.

Scaling an Object

One of the simplest geometric operations that is regularly performed is to scale an existing curve or surface. In SDL, this is achieved by simple scalar multiplication.

```
s *= 2.0;
```

The preceding example multiplies each component of each element in s by 2.0. (NOTE: `a *= b`, like in the C language, is equivalent to writing `a = a * b`. Also, as in C, statements in SDL are terminated with a semicolon.) Scaling a single component of an object is performed using a component suffix. For example:

```
s.x *= 2.0;
```

scales only the x components of s.

Translating an Object

Translating an object is just as simple as scaling one. To perform a translation, one adds a constant vector to each element of the object. For example:

```
c += {0,10,0};
```

will translate the curve c 10 units in the y direction. Another way to achieve the same results would be:

```
c.y += 10;
```

Point Distribution on a Line

A slightly more complex example is the distribution of points along a straight line between two given points. For this example, suppose we have an farray, fa, with the same dimension as curve c. This variable will hold the relative distribution of points, varying from 0 to 1. Furthermore, assume that the curve starting and ending points (vectors) have been placed in the first and last elements of c. Distributing the interior points reduces to:

```
c = c[1] + fa * (c[c.d1] - c[1]);
```

Note that a vector `(c[c.d1] - c[1])` multiplied by an farray results in a curve, each element of which is formed by multiplying the vector by the respective element of the farray. Since the distribution begins at 0, the resulting curve starts at (0,0,0). Adding the vector c[1] to the result translates the curve to the proper location.

As a final example of the conciseness and power of these expressions, a Bézier curve with four control points is constructed. The Bézier curve is a parametric curve defined by the following vector function. (Ref 2)

$$\mathbf{P}(u) = \sum_{k=0}^{n} p_k B_{k,n}(u) \quad (1)$$

$$B_{k,n}(u) = C(n,k)u^k(1-u)^{n-k} \quad (2)$$

$$C(n,k) = \frac{n!}{k!\,(n-k)!} \quad (3)$$

In the above equations, u is the parameter varying between 0 and 1, and p_k are the $n+1$ control points. The SDL code to implement these equations is:

```
c = p0*(1-u)^3 + p1*3*u*(1-u)^2 + p2*3*u^2*(1-u) + p3*u^3;
```

where u is an farray containing the values of the parameter u, p0, p1, etc. are vectors containing the control points, and $C(3,0) = C(3,3) = 1$ and $C(3,1) = C(3,2) = 3$.

User Subroutines

Clearly one does not want to have write and rewrite SDL code to do such common tasks as generating arcs and lines. The obvious thing to do is to create subroutines which perform those functions. By producing a set of subroutines to perform common functions such as curve generation, interpolation, blending, etc., one reduces most surface generation tasks to series of subroutine calls. Such a subroutine library is included with SDL, and is called stdsubs.sdl.

While the standard subroutine library supplies a large number of basic capabilities, there are always requirements specific to a particular task. SDL provides an easy way to automate portions of a task by allowing the end user to build specialized subroutines that either work in conjunction with or replace the subroutines in the standard library. This makes SDL infinitely expandable. Examples of the use of custom subroutine libraries are given below.

SDL subroutines have the following properties:

- A subroutine can have any number of declared arguments.
- Subroutine arguments can be passed back or not at the users option.
- Subroutine arguments automatically inherit the dimensions of the object passed in. Thus if a portion of a curve or surface is used as a subroutine calling argument, the subroutine recieves a curve or surface with the dimensions of the portion passed in.
- The final declared argument in a subroutine can be declared as a multiple argument, allowing a variable number of objects of the same type to passed into the subroutine. Special constructs and operators allow sequential access to the objects passed in.

While "native" SDL is very powerful, it does have its limitations. Some operations, such as rotations about an axis, must be done on a component by component basis. The coding, while certainly possible, is not simplified, and the overhead of executing the SDL code would result in some unnecessary performance loss. Other functions, such as the splining of objects and distribution of points on a spline, would be far too cumbersome to program in SDL. To perform these types of functions, SDL uses intrinsic, or built–in functions.

Intrinsic Functions in SDL

FORTRAN programmers are familiar with intrinsic functions which perform trigonometric, logarithmic, and other mathematical functions. SDL provides these same functions, but it also provides others that are specifically oriented toward curve and surface generation, such as curve and surface rotation, and parametric distribution functions. These functions, in combination with the subroutines provided in the standard library, complete the basic surface generation capability of SDL.

Adding Intrinsic Functions to SDL

SDL is designed to be easily extended by adding new intrinsic functions. This is done by writing a C function which performs the actions taken by the intrinsic function. This function must comply with a simple, well defined typing and argument specification. Once the function is written, it is added to SDL by adding an entry to a list which contains such information as the name of the intrinsic function and the number and types of its arguments. When SDL is recompiled, the new function will be a part of the language. This ease of adding new functions makes SDL an ideal testbed for new algorithms.

EXAMPLE: MISSILE BODY SURFACE

Two bodies commonly modeled to validate Computational Fluid Dynamics (CFD) codes are the tangent–ogive–cylinder and the tangent–ogive–cylinder–ogive. These generic missile shapes occur with and without fins. If the CFD data is being compared to wind tunnel test results, a sting is usually modeled. The following examples illustrate the use of SDL library routines to easily generate the missile body surface grids.

Naca.sdl

This file contains routines which generate NACA airfoil sections using the defining polynomials. Specifically, the subroutine naca4, which generates a NACA 4 series airfoil, is used. (Ref 3)

Mslbody.sdl

This file contains routines which generate a tangent–ogive–cylinder or tangent–ogive–cylinder–ogive body surface. All parameters controlling the dimensions of the body are configurable. In addition, the routines will optionally generate a sting and make allowances for any number of fins of an arbitrary cross section. If fins are specified, then a section of the surface lying between two fins is generated. Otherwise one half of the missile surface is produced.

Mslbody.sdl is a relatively complex SDL code, and is too long (about 350 lines of SDL code) to include in this paper. It is a good deal shorter than if it were written in a conventional language, however. Most of its complexity comes from the number of parameters that can be varied.

Stdsubs.sdl and Intrinsic Functions

The following examples also use several variables and subroutines from the standard subroutine library, along with some intrinsic functions. A complete reference manual describing all of the functions and subroutines in SDL is beyond the scope of this paper. However, a brief description of those used in the examples follows:

PI, PI_d2 – These are global variables defined in stdsubs.sdl. They contain the values of Pi and Pi/2 respectively.

arc – Subroutine to generate a circular arc in the xy plane. Requires four arguments: `curve` to be filled (returned); start angle; end angle; and radius.

line – Subroutine to generate straight line with linear distribution. Requires one argument: `curve` (returned) to be filled, with endpoints preloaded.

dline – Subroutine to generate straight line with specified point distribution. Requires two arguments: `curve` (returned) to be filled, with endpoints preloaded; `farray` with desired point distribution.

curdist – Subroutine to redistribure points on a curve by splining the curve and redistributing points on the spline. Requires two arguments: `curve` (returned) to be redistributed; `farray` with desired point distribution.

d_tanh – Intrinsic to generate one or two sided hyperbolic tangent point distribution. Requires five arguments: start of distribution; end of distribution; start spacing; end spacing; number of points in distribution. One zero spacing results in a one sided distribution. Returns `farray`.

makecc – Intrinsic function to generate cubic spline from curve. Requires three arguments: `curve` to be splined; `string` specifying end conditions (*"natural", "quadratic", or "specified"*); `string` specifying spline basis. (*"arc" or "index"*). Returns `pcurve`.

distc – Intrinsic function to distribute points on a pcurve. Requires two arguments: `pcurve` to be distributed on; `farray` containing desired point distribution. Returns `curve`.

Case 1: Tangent–Ogive–Cylinder, No Fins

To demonstrate the simplicity of using SDL libraries, a tangent–ogive–cylinder with no fins is generated. (see Figures 1 and 2) Library calls are made to set the physical dimensions of the body, then the body surface is generated with a call to mk_missile_body(). Note that the i and j dimensions of the generated surface are determined by the dimensions of the surface variable provided as an argument.

The SDL code in Figure 1 would typically be only part of a longer program which would generate the additional surfaces necessary for the generation of a volume grid. Figure 3 demonstrates how one might continue the program and generate upper and lower reflection planes. This code example makes use of standard subroutine library calls to generate lines and arcs, and to do an arclength based transfinite interpolation . Intrinsic functions are used to generate point distributions and parametric representations of curves. The results are shown in Figure 4.

Case 2: Tangent–Ogive–Cylinder, w/ sting, fins.

In this example, a much more complex surface is generated. Figure 5 shows, however, that the required SDL code is not much more involved than in the simple example. Note that the NACA library is used to generate an airfoil section, which is then splined to redistribute points. The resulting section, along with positional information, is recorded using mslbody routines. Additional information about the rear ogive and sting are also provided, and the surface is generated with a call to mk_missile_body(). The results are shown in Figure 6.

CONCLUSIONS AND RECOMMENDATIONS

SDL is a powerful language specifically designed for the generation of curves and surfaces. It

provides a maximum degree of flexibility by allowing the end user to indefinitely expand, customize, and improve its surface grid generation capabilities. Because the SDL interpreter is written in standard portable C and does not depend on hardware architecture, it should be easily ported to any hardware platform.

SDL is recommended for use anywhere its unique capabilities are needed. Potential uses range from production work in a grid generation shop to the classroom, where its implementation of basic vector algebra makes it ideal for learning and exploring geometric algorithms.

REFERENCES

1. Thompson, J. F.; and Gatlin, B.: *Program EAGLE User's Manual, Volume II: Surface Generation System*, AFATL-TR-88-117, September 1988.

2. Hearn, D.; and Baker, M.P.: *Computer Graphics*, Prentice-Hall Inc, Englewood Cliffs, NJ, 1986.

3. Abbot, I.H.; and Von Doenhoff, A.E: *Theory of Wing Sections*, Dover Publications Inc, New York, 1959.

```
# This file uses the mslbody.sdl library routines to generate a
# tangent-ogive-cylinder body surface.

load "mslbody.sdl"

surface body[60,20];

set_fwd_ogive(5);  #forward ogive radius is 5.
set_barrel(.75,7.5); #cylinder has radius of 1 and len of 5

mk_missile_body( body );

# body now holds 1/2 missile surface
```

Figure 1: SDL code for Tangent–Ogive–Cylinder.

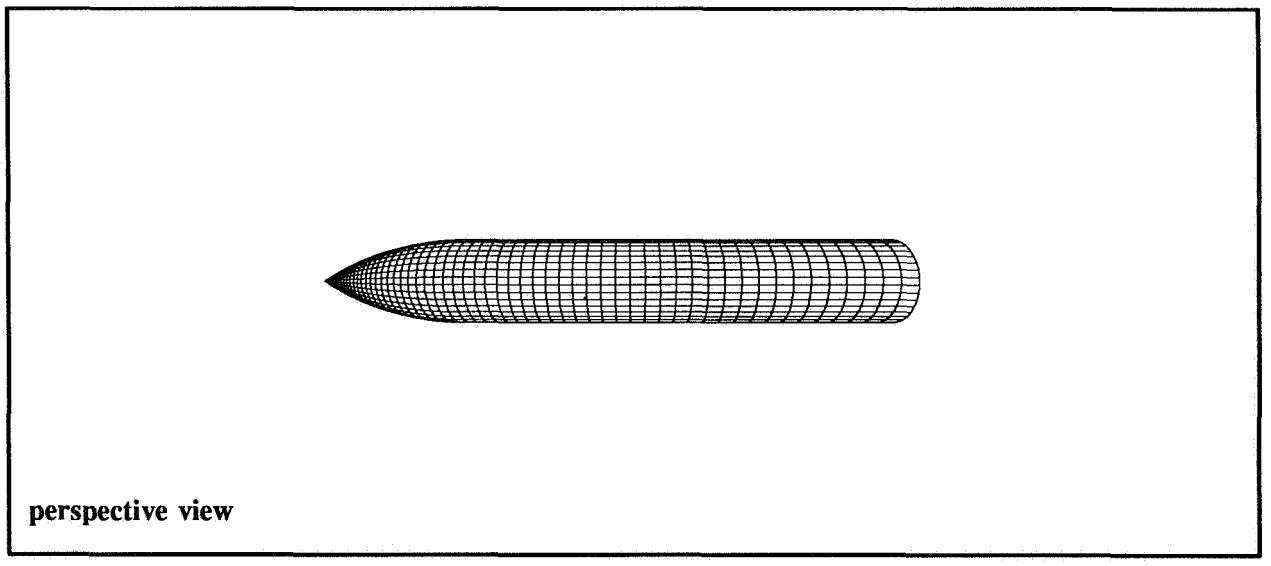


Figure 2: Tangent–Ogive–Cylinder body surface.

```
# ..... continued from above

curve outer_profile[60];              # get the req'd curve
surface uref[60,25], lref[60,25];     # and surface

# build the outer profile

arc( outer_profile[ :45], PI, PI_d2, 1.2 * body[60,1].x );
outer_profile[1].y = 0.0;  #eliminate roundoff error!

outer_profile[60] = body[60,1];
outer_profile[60].y = 1.2 * body[60,1].x;
line( outer_profile[45:] );
curdist( outer_profile, d_tanh(0,1, .07, .01, 60) );

# fill the upper reflection plane

uref[ ,1] = body[ ,1];
uref[ ,25] = outer_profile;
dline( uref[1, ], d_tanh(0,1,.01, 0, 25) );
dline( uref[60, ], d_tanh(0,1,.01, 0, 25) );

transfinite( uref );

# fill the lower reflection plane

lref = uref;
lref.y *= -1;
```

Figure 3: SDL code for Tangent–Ogive–Cylinder reflection planes.

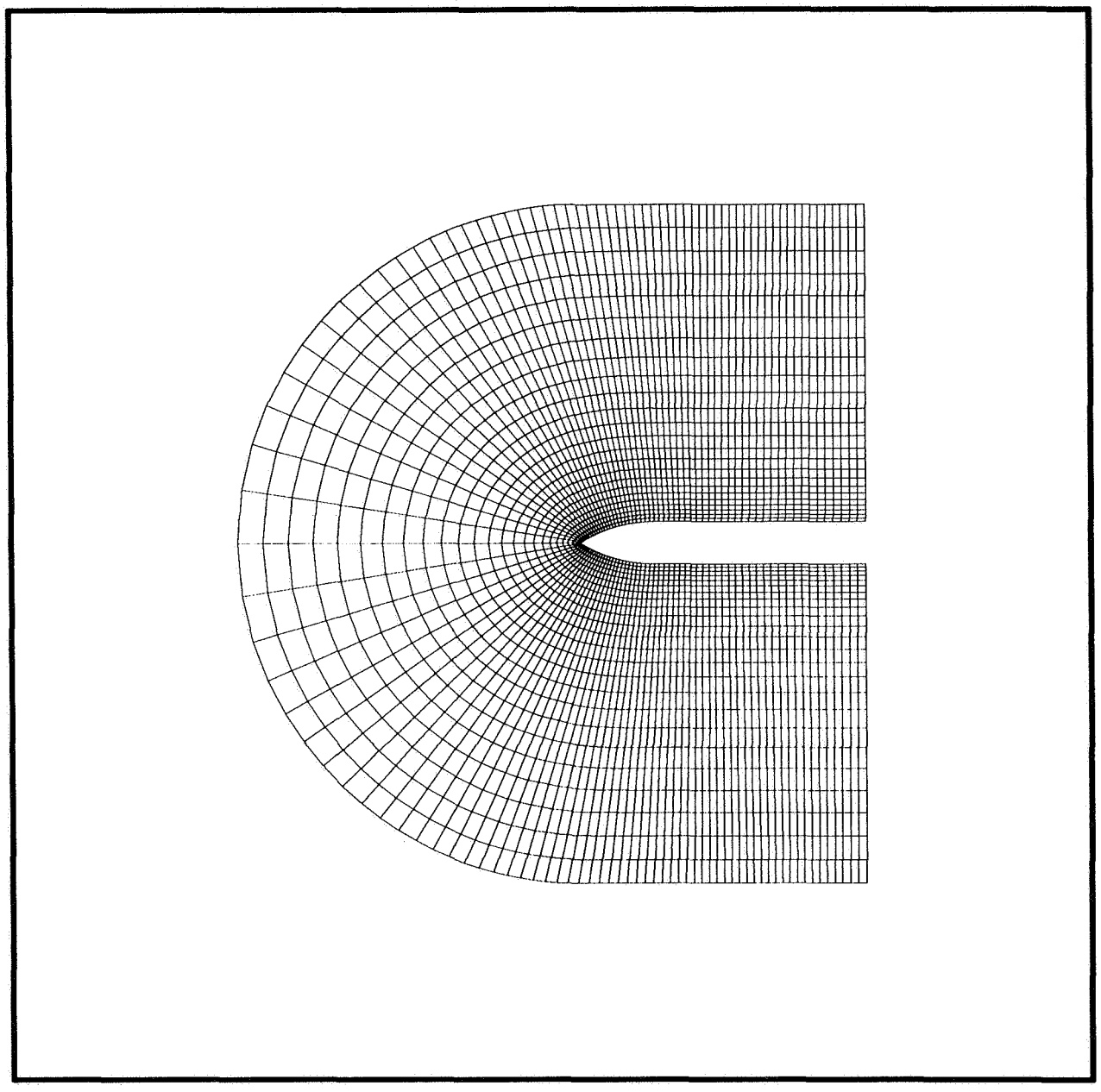


Figure 4: Tangent–Ogive–Cylinder reflection planes.

```
# This file produces a tangent-ogive-cylinder-ogive body, with sting
# and allowance for 4 NACA 0012 fins.

load "stdsubs.sdl"
load "mslbody.sdl"
load "naca.sdl"

# main body parameters

surface body[90,11];
float x_fwd_shoulder;

set_fwd_ogive( 5);
set_barrel(.75,7.5);
set_rear_ogive( 3 );

get_fwd_shoulder( x_fwd_shoulder );

# generate fin profile and set fin parameters

curve naca_profile[200], root[20];

naca4(naca_profile,0,0,12);
pcurve proot = makecc( naca_profile, "quadratic", "arc" );
root = distc( proot, d_tanh( 0,1,0.05,0, 20 ) );

set_fin_root( root );
set_fin_cnt(4);
set_fin_loc(x_fwd_shoulder + 6.3, 45);    # fin starts at i = 40
set_off_fin_sp( .03 );

# set sting parameters

set_sting( .4, 10, 15 );

mk_missile_body( body );

# body now holds 1/4 missile surface
```

Figure 5: SDL code for Tangent–Ogive–Cylinder–Ogive.

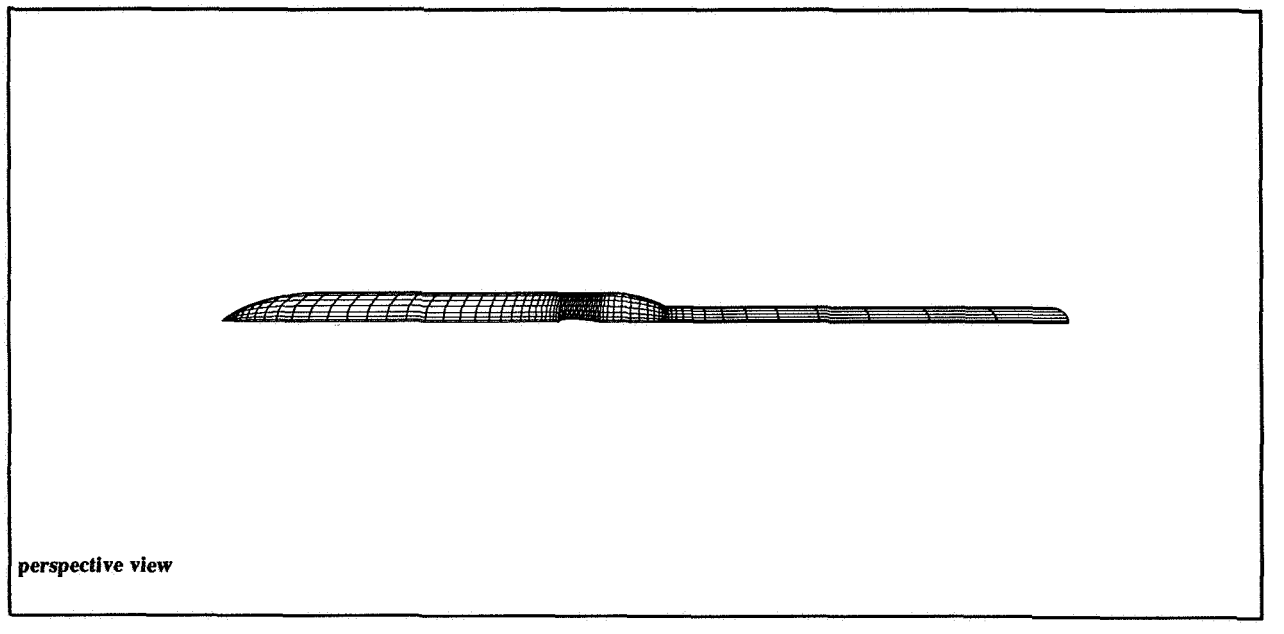


Figure 6: Tangent–Ogive–Cylinder–Ogive body surface.

S3D - AN INTERACTIVE SURFACE GRID GENERATION TOOL*

Raymond Ching-Chung Luh
MCAT Institute
Moffett Field, CA

Lawrence E. Pierce
Sterling Software
Moffett Field, CA

David Yip
NASA Ames Research Center
Moffett Field, CA

SUMMARY

This paper describes S3D, an interactive software tool for surface grid generation. S3D provides the means with which a geometry definition based either on a discretized curve set or a rectangular point set can be quickly processed towards the generation of a surface grid for CFD applications. This is made possible as a result of implementing commonly encountered surface gridding tasks in an environment with a highly efficient and user-friendly graphical interface. Some of the more advanced features of S3D include surface-surface intersections, optimized surface domain decomposition and recomposition, and automated propagation of edge distributions to surrounding grids.

INTRODUCTION

Advances in computer technology and Computational Fluid Dynamics (CFD) in the recent years have made possible the computation of flow fields around realistic three-dimensional geometries. One problem with realistic geometries is that they tend to be complex as well. No longer are these geometries defined by simple, analytic forms as was usually the rule in the past. As a result, the task of generating a compuational grid, which is the first step in any CFD analysis, has become so labor-intensive and time-consuming that it can no longer be taken for granted. The question is not whether a grid can be generated but rather how long the process would require.

A computational grid is the collection of a large number of points about or within an object. Of particular importance are those points which lie directly on the surface of the object to help define its shape. Surface grid generation, a prerequisite to volume grid generation, is seen to have a dominant effect on the quality of the volume grid and to be extremely time-consuming. S3D is a new and evolving surface grid generation tool based on the work described in Refs. 1 and 2 that promises to significantly reduce the turn-around time.

---

* Dr. Luh's participation in this project was under cooperative agreement to MCAT Institute, NCC2-513, and Mr. Pierce's participation under contract to Sterling Software, NAS2-13210.

The program is implemented in the dynamic color-graphics environment of a workstation to take advantage of the latest in interactive technology. Much attention has been given to the issues of ease of use and user-friendliness in the design of the graphical interface for S3D. Considering the highly visual nature of the surface grid generation process, every effort has been put into facilitating the use of the mouse instead of the keyboard for maximum efficiency in interactivity. Thus, the user is able to perform nearly all of the tasks, including object manipulation, by moving the mouse around and clicking on the appropriate buttons. Windows for custom panels are opened and closed automatically as the need for them comes and goes so that the screen always displays just the right amount of information for the task at hand. A typical S3D layout is shown below.

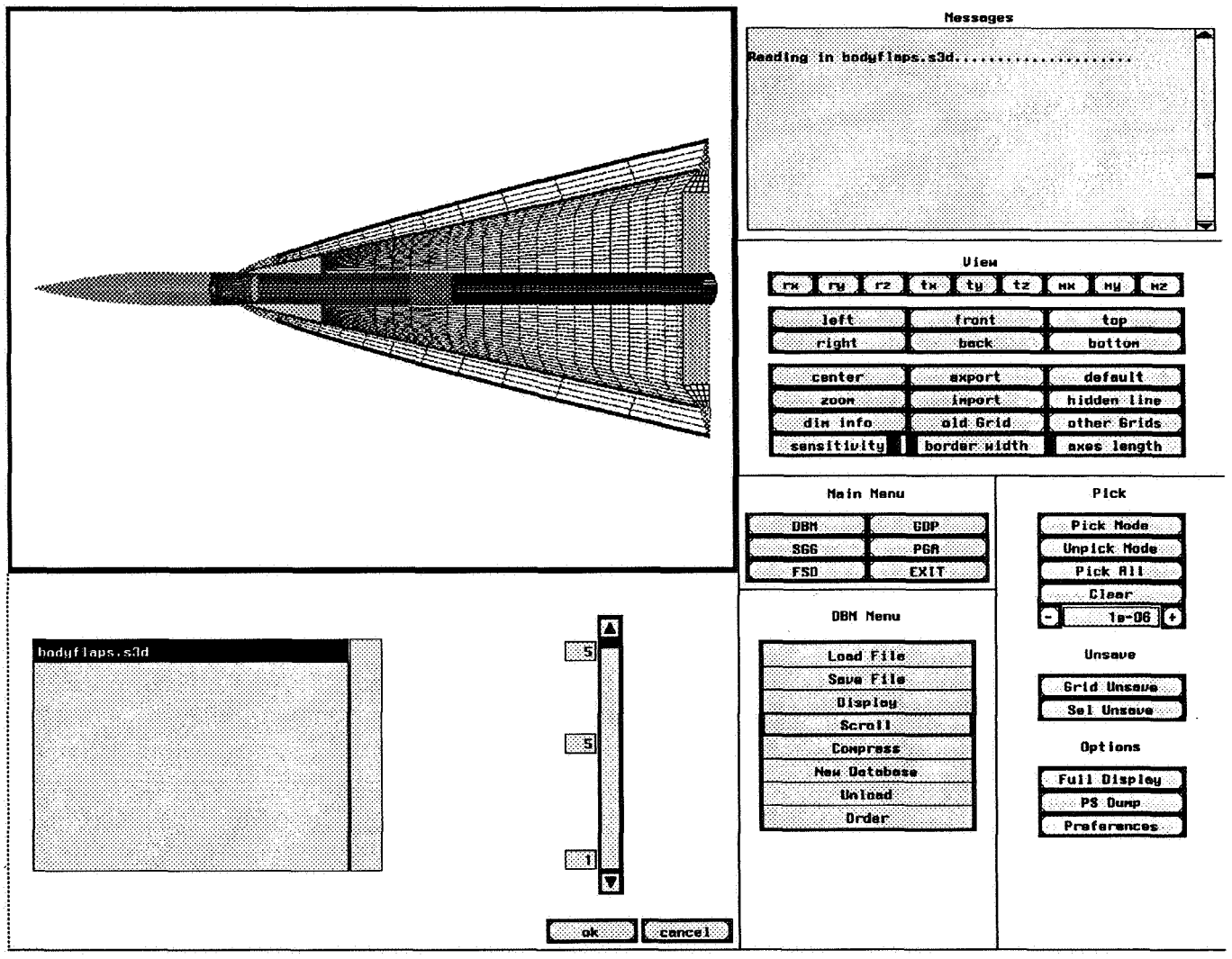


Figure 1. Typical screen layout of S3D

There are five major elements to S3D - Data Base Manager (DBM), Patch Grid Assembler (PGA), Geometry Data Processor (GDP), Surface Grid Generator (SGG), and Free-form Surface Designer (FSD). DBM controls the functions that allow the user to manipulate the S3D database, such as loading and saving

files, choosing which loaded files to use, and keeping track of the history of the database development. Functions in GDP permit processing of geometries made up of either a discretized curve set or a rectangular point set. A surface definition in the form of a reference grid can be obtained. In the process, data points which highlight geometrical discontinuities and high-curvature regions can be frozen to preserve the prominent features they represent. SGG functions work on a reference grid, or a rectangular point set. This is where surface grids can be customized. Work is under way to enhance this module so that geometry data from CAD systems can be directly input and analyzed without any approximation. PGA is a utility module that serves a variety of needs during the course of a surface grid generation operation. Deficiencies in the original geometry, if present, can be corrected or eliminated. Decomposition and recomposition of surface patches are made simple. The FSD module, which is still in the conceptual stage, will contain functions for designing surfaces which are necessary in the construction of faces, other than those given, for volume grid generation. Details regarding some of the more important functions in these modules are described below. A complete description of all existing functions can be found in Ref. 3.

DATA BASE MANAGER (DBM)

It is empirically evident that ease of input and output operations, and also the ability to retain and manage the database in hierarchical as well as cross-relational senses are vital to a patch-oriented system like S3D. DBM is the module in S3D that handles these functions in a menu-driven environment.

The input and output operations presently support three basic file types. Each file type can be manifested as a formatted (ASCII) file, an unformatted file, or a binary (C-binary) file. The three basic file types are the multi-grid PLOT3D[4] format, a section data format, and a S3D historical format. The PLOT3D format, which has grown in popularity as a means for transferring data between various codes in the CFD processes, allows reading and writing of (x,y,z) data for one or more rectangular patches. This format is best suited for an existing surface definition or grid.

The sectional format is meant for surface data that is represented by a set of randomly distributed sectional points. At present, this format also serves as a buffer between the surface definition that might be stored in a CAD system and one that would be constructed using GDP. A plan to incorporate some of the more popular CAD standard formats such as IGES is being considered for future implementation.

The S3D historical format allows the surface data to be input or output with a large quantity of other information that is associated either with the geometry or with the hierarchy and the relationship of one patch to another. This format would be the best choice during the surface grid generation process because it allows a maximum level of information to flow in and out of S3D.

The broad historical information that is part of the S3D internal data structure permits DBM to feature a wide range of data management functions. It allows the user to make some data sets, which are typically collections of patches from a configuration, active (i.e. displayed) and make other data sets inactive (in memory, but not currently displayed). In addition, the user can select those data-objects such as patches, sections, and points on which to apply some function of S3D (e.g. redistributing patches, inserting points, etc.).

DBM also allows the user to manipulate the data in a historical fashion. A one-step "undo" feature makes it possible for the user to undo the previous operation. The user can also do a one step undo on any patch (even one that was not involved in the previous operation) or collection of patches at any time. The

user can also recall patches from any historical level, e.g. ten steps back, or construct new data sets using patches from various historical levels.

PATCH GRID ASSEMBLER (PGA)

This module is necessitated by the variety of cutting and pasting needs that arise as a result of the grid generation requirement or the way in which a geometry is given. The graphical interactive approach of PGA provides an easy and efficient way to accomplish this simple but bothersome task. The patch boundaries function is one of the most important tools in this module. It conveniently simplifies decomposition and recomposition of surface patches.

The add and change point functions allow the user to type in data points that are known to be missing or to replace bad existing ones. The delete points function allows the user to remove spurious or bad data points. There are also functions for manipulating how the geometry is stored with respect to the I and J axes. The user can swap the axes as well as reverse their direction.

The collapse edge feature enables the user to set all the points along an edge to the same coordinates. One use for this would be to correct small inconsistencies in the geometry being processed. For example, the coordinates of the points making up a collapsed point may be different by a small amount. This function would allow all the points to be set to the same coordinates.

Finally, a mirror function allows the user to create a new patch that is simply a reflection of the existing geometry. The reflection is made along either the x, y or z axis. The entire geometry can be reflected or just a portion of it. This may be useful if, for example, only half of a symmetric geometry was defined originally. The other half can be easily generated.

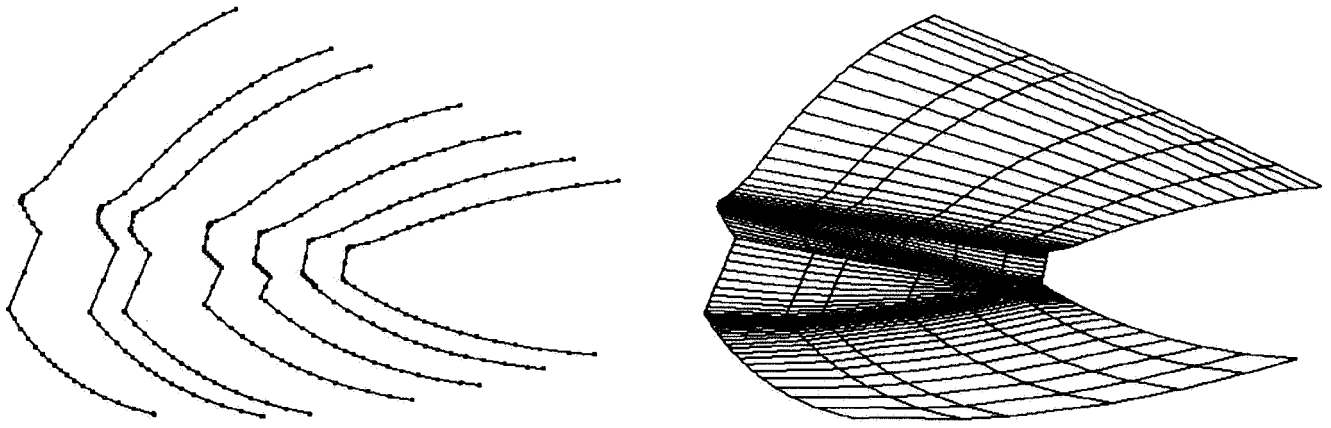
GEOMETRY DATA PROCESSOR (GDP)

From practical experience, the geometry data one obtains for a configuration may often be at a primitive level, i.e. with only a preliminary surface definition. The fuselage in such a geometry definition, for instance, may be given in cross sections with each section represented by a different number of points. There may be geometrical discontinuities or body-embedded singularity points which would require special attention. Usually, the work required to process such data outside of a surface grid generation package is partly to blame for the long delay in getting the surface grid.

GDP works like a section data editor allowing primitive point data to be processed into a surface definition based on a rectangular grid. Along the way, it permits domain decomposition, associated with the composite block-structured approach necessary for complex geometry, to be carried out on the surface. The key function under this menu is the redistribution of sections. The starting point for this operation is a set of cross sections, each with an arbitrary number of points distributed randomly. Each curve data is fitted with piecewise parametric cubic polynomials. This involves the determination of derivatives at each data point which is most critical in ensuring the accuracy of interpolation. A method adapted from Akima's scheme[5] has shown to be highly accurate because of its second-order nature and independence of interval widths in the data. It should be pointed out that one may opt to redistribute points in the uni-variate sense of GDP even if the geometry data in question is rectangular. Uni-variate redistribution is preferred when

one is satisfied with the point distribution in one direction and wishes to preserve the positioning and shapes of those curves.

Not unexpectedly, geometry defined by cross sections could be missing data points that may be critical in outlining prominent geometric features in the opposing direction, i.e. in the direction of the cross sections. These are usually missing because they are not needed for adequately defining the cross-sectional curves. For such deficiencies, the global/local insert functions under GDP become rather convenient in allowing point insertions through interpolated means. The Break-line feature of GDP is another function that is essential in preserving prominent geometric features in the cross-sectional curves. With this function, one can essentially freeze any point(s) on the curves to keep them invariant under redistribution. Figure 2 below illustrates the results of carrying out the uni-variate redistribution on all curves to obtain the same number of points while preserving sharp corners and clustering around high-curvature regions.



(a) Cross-section data (b) Reference grid

Figure 2. Cross sections with different numbers of points are first turned into a reference grid by applying GDP's uni-variate redistribution function

SURFACE GRID GENERATOR (SGG)

The output of GDP is a set of surface patches which clearly defines the surface geometry and is ready for surface grid generation. As redistribution in GDP is confined to the individual sectional curves, the bi-directional redistribution of these patches becomes the central task in SGG. One can think of GDP as making the transition from one form of surface definition (cross-sectional data) to another (reference grid). As explained above, one can redistribute a rectangular grid in the uni-variate sense using the functionality provided in GDP or, as is more often the case, in the bi-variate sense of SGG described below.

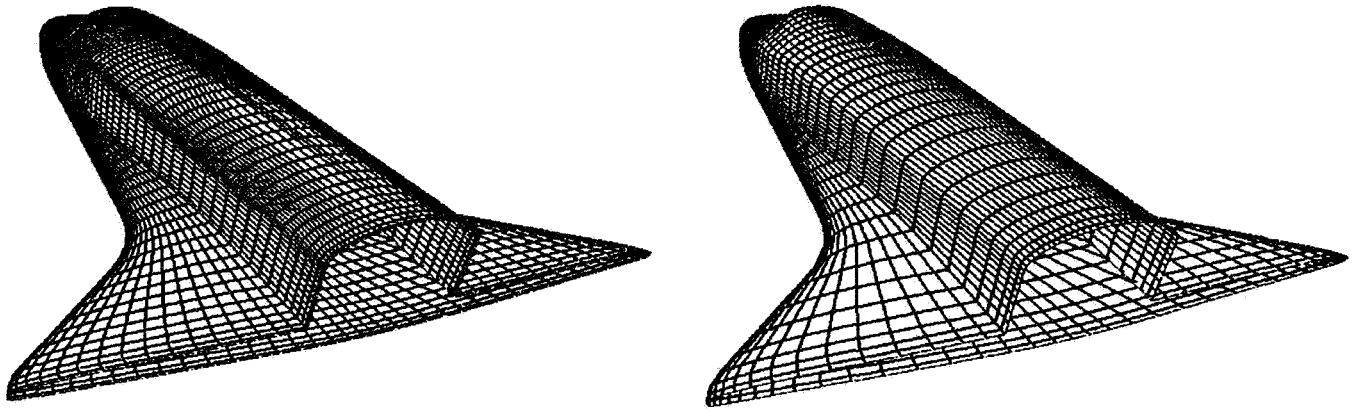
The first step in the redistribute function of SGG is to fit the grid points with piecewise bi-cubic patches based on a Pseudo Normalized Arc-Length Parameter Space (PNALPS) parametrization, which maps the tri-variate surface to a bi-variate unit square. Again, the critical derivative information is calculated at each

grid point using a method adapted from Akima's scheme.[6] The desired point distribution in terms of PNALPS parameters is then set up interactively by the use of hyperbolic-function-based, two-sided stretching functions developed by Vinokur.[7] Finally, the coordinate values corresponding to the new PNALPS are obtained by interpolating on the bi-cubic patches. Detailed descriptions are found in Ref. 1.

One highly significant feature of S3D is its ability to set up patch neighborhood connectivity information automatically. It cannot recognize staggered neighboring patches but it can contend with the multiple neighbor scenario if there is point continuity among all the patches at their corners. The user is able to pick one or more (or even all) of the patches for redistribution as long as they are valid neighbors in the sense of being topologically rectangular. Users never need be concerned with the connectivity or orientation of the patches. The result of such a redistribution is one single patch. Another equally powerful feature of S3D, made possible by the automatic patch connectivity capability, is the ability to export point distributions to neighboring patches at the touch of a button. This permits any local modification on the surface grid to be immediately reflected on the surrounding regions without additional effort.

Another frequently encountered problem tackled in SGG is the surface-surface intersection. Given two patches which intersect each other, each patch alternates between being a target and a source. An intersection curve is calculated for each source patch from the points where its grid lines intersect the target surface. The patches can then be redistributed further for point and slope continuities at the intersection or patch boundaries.

One frequently encountered task in surface grid generation is the changing of grid density. This seemingly simple task can nevertheless be annoyingly time-consuming, especially when the geometry in question contains many features of physical significance. SGG is perfectly suited for such a task because of the ease with which existing grids can be redistributed. Figure 3(a) shows the space shuttle forebody represented by a single grid of dimension 66 by 77. In very little time, a new grid of dimension 48 by 60, which is a reduction of nearly 50% from the original, is obtained as shown in Figure 3(b). The less dense grid is generated such that grid point concentration near and around all major feature lines in the shuttle geometry is main-

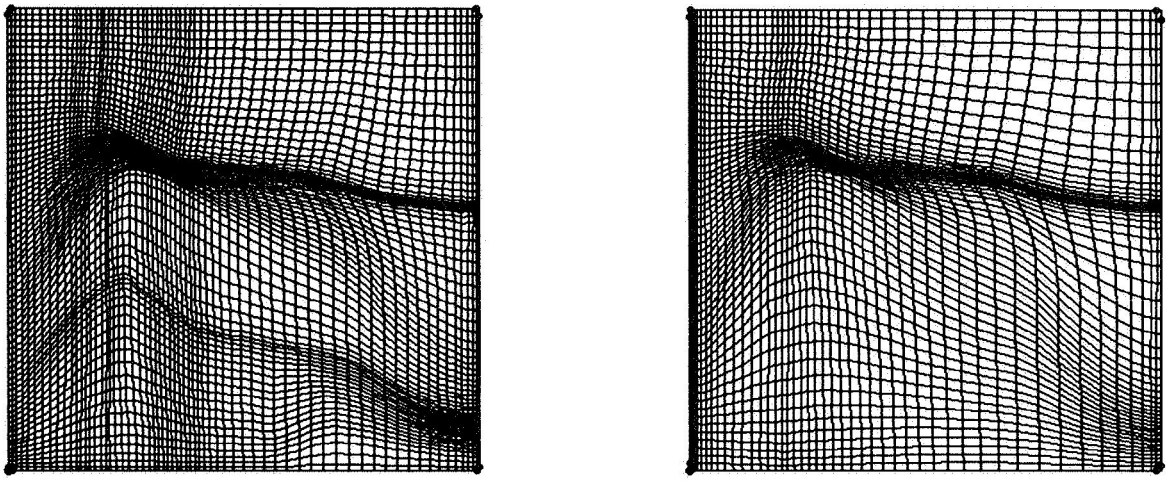


(a) 66 by 77 (b) 48 by 60

Figure 3. The grid in (b) is obtained by reducing the density and redistributing the grid in (a)

tained. This is made possible by first decomposing the original single grid into multiple grids or patches, then redistributing each patch and finally recomposing the multiple grids back into one piece.

The similarity in grid point concentration for the two grids above can be appreciated by looking at the PNALPS before and after the redistribution as shown in Figure 4(a) and 4(b), respectively. The blunt nose is on the left edge of the parameter space and the coarser grid is designed with more grid point concentration locally. The streak in the middle running down to the right side is where the wing leading edge is. Note that the coarser grid does not have the spacing discontinuities which are quite visible in the original.



(a) Before (b) After

Figure 4 Comparison of PNALPS before and after grid density reduction on the shuttle body (fig. 3)

CURRENT AND FUTURE EFFORT

At present, S3D can start with a geometry definition based either on a discretized curve set or a rectangular point set as described above. There is however an increasing awareness in the CFD community of the need to preserve and utilize Computer-Aided-Design (CAD) representations throughout the geometry modeling and surface gridding process, so that the resulting grid points stay true to the exact definition. In order for a surface modeling tool such as S3D to fulfill this requirement, it must be modified to accept geometry data directly from the CAD systems, and to analyze such data without any approximation. As a first step, work is under way to add CAD data input to the existing I/O capabilities of S3D. Most existing CAD systems output data using Initial Geometry Exchange Standards (IGES), and hence the proposed input function would be one that reads CAD data in IGES format. In particular, the CAD input function will adhere to NASA-IGES[8] which is an emerging geometry data exchange standard based on a subset of IGES specifically designed for CFD applications. NASA-IGES includes formats for representation of curves and surfaces by Non-Uniform Rational B-Splines (NURBS) which are fast becoming the industry standard for surface designs and analyses.

The implementation of the proposed CAD input function would enable direct down-loading of surface geometries designed by CAD systems into S3D where the data would be stored in its exact form. One approach being considered for generating the surface grid for this high-order geometry definition is as follows. It is proposed that a reference grid that would lie exactly on the true surface be extracted from each CAD surface. Slope and curvature information of the CAD surfaces would be used to determine the distribution of points on the reference grids. At this point, the present capability of S3D in bi-variate redistribution of grid points would be applicable. The resulting grids would then be projected onto the CAD surfaces to ensure accuracy.

CONCLUDING REMARKS

An overview of S3D, a new and evolving surface grid generation tool, has been presented. Some of the more useful features and capabilities of the program have been described. When completed, S3D will be able to process geometry definitions based on discretized curve sets, rectangular point sets, or CAD surface representations. Creative use of state-of-the-art interface technology together with innovate algorithms designed to automate repetitious and labor-intensive tasks makes S3D an indispensable as well as a convenient tool. The efficiency and ease with which S3D can be used to handle commonly encountered tasks in surface grid generation are already helping CFD analysts to drastically reduce the time required in gridding complex surface geometries.

REFERENCES

1. Luh, Raymond C.-C.: Surface Grid Generation for Complex Three-Dimensional Geometries. Appears in Sengupta, S.; Häuser, J.; Eiseman, P. R.; and Taylor, C., eds.: *Numerical Grid Generation in Computational Fluid Mechanics.* Pineridge Press Ltd., 1988.

2. Luh, Raymond C.-C.; Yip, David; and Pierce, Lawrence E.: Interactive Surface Grid Generation. Appears in Arcilla, A.S.; Häuser, J.; Eiseman, P.R.; and Thompson, J.F., eds.: *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields.* North-Holland, 1991.

3. Luh, Raymond; Pierce, Larry; and Yip, David: *S3D User Manual*, Version 1.0, 1991.

4. Walatka, P. P.; Buning, P. G.; Pierce, L.; and Elson, P.A.: *PLOT3D User's Manual*, Version 3.6. NASA TM-101067, 1990.

5. Akima, H.: A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures. *J. of the ACM*, Vol. 17, No.4, 1970, pp. 589-602.

6. Akima, H.: A Method of Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures. *Comm. of the ACM*, Vol. 17, No. 1, 1974, pp. 18-20.

7. Vinokur, M.: On One-Dimensional Stretching Functions for Finite Difference Calculations. *JCP*, Vol. 50, No. 2, 1983, pp. 215-234.

8. Blake, Matthew: The NASA-IGES Geometry Data Exchange Standard. *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA CP-3143, April 1992.

Elliptic Surface Grid Generation in Three-Dimensional Space

Lee Kania
Sverdrup Technology, Inc.
Structures and Dynamics Dept.
Huntsville, Alabama 35806

SUMMARY

A methodology for elliptic surface grid generation in three-dimensional space is described. The method solves a Poisson equation for each coordinate on arbitrary surfaces using successive line over-relaxation. The complete surface curvature terms have been discretized and retained within the nonhomogeneous term in order to preserve surface definition; there is no need for conventional surface splines. Control functions have been formulated to permit control of grid orthogonality and spacing. A method for the interpolation of control functions into the domain has been devised which permits their specification not only at the surface boundaries but within the interior as well. An interactive surface generation code which makes use of this methodology is currently under development.

INTRODUCTION

Developments in grid generation techniques have been a pacing item for application of fluid dynamics codes to complex configurations. In these instances the task of grid generation is as significant a problem as the flow field analysis itself. Clearly, increased emphasis in research and development of grid generation methods is necessary to improve the efficiency with which CFD analyses are conducted. Indeed, recent years have seen the arrival of some powerful and flexible techniques which address this issue.

Significant contributions to the grid generation effort over the years have included, to name only a few, the GRAPE2D code of Sorenson[1], the EAGLE surface/grid code of Thompson[2] and the GRIDGEN codes by Steinbrenner[3] et al. The GRAPE2D code solves Poisson's equation in two-dimensions and utilizes what was a novel approach for determination of the boundary control functions. Use of this code remains widespread throughout the CFD community. Though the code is extremely versatile, grid optimization is somewhat hampered due to the batch mode implementation. The EAGLE code combines techniques in surface grid generation as well as two- and three-dimensional grid generation. As originally released, the code operated in the batch mode on the Cray. Since its inception, attempts to alleviate the inefficiencies of this platform have been made; an interactive version of the code has recently been released. The GRIDGEN series of codes is a more recent appearance. These codes permit interactive surface design and batch grid construction. A host of options are avaliable which afford a high degree of flexibility, however, intelligent use of the majority

of these options requires the user to be well versed in current grid generation techniques. In addition, invocation of the elliptic surface generation mode may become rather cumbersome as the user may be required to intervene in order to ensure construction of well defined surface splines.

The current work builds upon past research efforts and also exploits the capability of present day engineering workstations to produce an easy to use interactive elliptic surface generation code which affords the user a high degree of flexibility and robustness. All data necessary to control grid evolution is specified interactively. Grid optimization is greatly enhanced as solution constraints may be adjusted as the grid evolves. The following sections describe the method and its implementation in greater detail.

DESCRIPTION OF METHOD

Governing Equations

The current method solves the vector Poisson equation

$$\alpha\vec{r}_{\xi\xi} - 2\beta\vec{r}_{\xi\eta} + \gamma\vec{r}_{\eta\eta} = -J^2(P\vec{r}_{\xi} + Q\vec{r}_{\eta}) + [(\alpha\vec{r}_{\xi\xi} - 2\beta\vec{r}_{\xi\eta} + \gamma\vec{r}_{\eta\eta}) \cdot \hat{n}]\,\hat{n} \tag{1}$$

for the surface coordinates where

$$\vec{r} = (x, y, z) \tag{2}$$

and

$$\begin{aligned} \alpha &= \vec{r}_{\eta} \cdot \vec{r}_{\eta} \\ \beta &= \vec{r}_{\xi} \cdot \vec{r}_{\eta} \\ \gamma &= \vec{r}_{\xi} \cdot \vec{r}_{\xi} \end{aligned} \tag{3}$$

In this equation, the Jacobian is given as

$$J = |\vec{r}_{\xi} \times \vec{r}_{\eta}| \tag{4}$$

and the normal vector is given as

$$\hat{n} = (n_x)\hat{i} + (n_y)\hat{j} + (n_z)\hat{k} = \frac{1}{J}(\vec{r}_{\xi} \times \vec{r}_{\eta}) \tag{5}$$

Equation (1) was originally formulated by Warsi[4]. The last term on the right hand side of equation (1) represents the surface curvature terms. Previous researchers have employed a less rigorous approach involving a solution of equation (1) for x and y only, the z-coordinate being provided by splines of the form $z = z(x, y)$. Creation of splines having this functional form will present a problem for surfaces not single valued in z. In the current implementation the complete surface curvature terms are retained in each equation in order to maintain surface definition; construction of surface splines is not required. Equation (1) is solved using successive line over-relaxation. Central differences are used to approximate all derivatives

which results in a scalar tridiagonal system for each sweep through the grid. At each level the Thomas algorithm is used to solve the system of equations for the coordinate vector $\vec{r}^{\,*}$. The coordinates at the previous iterate, n, are then corrected to yield the new coordinate

$$\vec{r}^{\,n+1} = \vec{r}^{\,n} + \omega(\vec{r}^{\,*} - \vec{r}^{\,n}) \tag{6}$$

where ω is the relaxation parameter.

Control Functions

Evaluation of the control functions is patterned after the work of Sorenson and Steger[5]. In their work the control functions are based on a prescribed step size and orthogonality constraint. For surfaces in three-dimensional space we impose these same constraints which, for the $\xi = constant$ boundaries, are expressed as

$$s_\xi = |\vec{r}_\xi| \approx \frac{\Delta s}{\Delta \xi} \tag{7}$$

$$\vec{r}_\xi \cdot \vec{r}_\eta = 0 \tag{8}$$

and a second orthogonality constraint which is expressed as

$$\vec{r}_\xi \cdot \hat{n} = 0 \tag{9}$$

For the $\eta = constant$ boundaries the constraints are written as

$$s_\eta = |\vec{r}_\eta| \approx \frac{\Delta s}{\Delta \eta} \tag{10}$$

$$\vec{r}_\xi \cdot \vec{r}_\eta = 0 \tag{11}$$

and

$$\vec{r}_\eta \cdot \hat{n} = 0 \tag{12}$$

For the $\xi = constant$ boundaries, equations (7) through (9) are solved for $\vec{r}_\xi$ to yield

$$\vec{r}_\xi = \frac{s_\xi(\vec{r}_\eta \times \hat{n})}{|\vec{r}_\eta \times \hat{n}|} \tag{13}$$

A similar procedure for the $\eta = constant$ boundaries using equations (10) through (12) yields

$$\vec{r}_\eta = -\frac{s_\eta(\vec{r}_\xi \times \hat{n})}{|\vec{r}_\xi \times \hat{n}|} \tag{14}$$

These expressions provide the first derivative at the respective surface boundaries without the need to difference into the field. The partial derivatives $\vec{r}_\eta$ and $\vec{r}_{\eta\eta}$ for the $\xi = constant$ boundaries and $\vec{r}_\xi$ and $\vec{r}_{\xi\xi}$ for the $\eta = constant$ boundaries are evaluated using the surface edge distributions. The derivatives are invariant with iteration as long as the prescribed step size and boundary distributions themselves do not change. In the event that boundary points are permitted to "float" with the solution as an alternate means to achieve orthogonality, the

particular derivative must, of course, be re-evaluated after each cycle. The remaining second derivatives $\vec{r}_{\xi\xi}$ and $\vec{r}_{\eta\eta}$ on the $\xi = constant$ and $\eta = constant$ boundaries, respectively, are the only quantities which vary with iteration. They are evaluated using one-sided derivatives as the grid evolves.

With the boundary coordinate derivatives known, the control functions P and Q may now be evaluated from equation (1). The apparent uniqueness problem due to three equations in two unknowns is removed by reducing the set to two equations by formation of the scalar product of equation (1) with $\vec{r}_{\xi}$ and also with $\vec{r}_{\eta}$. Straightforward algebraic manipulation of these equations then yields

$$P = \frac{\beta(\vec{\varrho} \cdot \vec{r}_{\eta}) - \alpha(\vec{\varrho} \cdot \vec{r}_{\xi})}{J^2(\alpha\gamma - \beta^2)} \tag{15}$$

and

$$Q = \frac{\beta(\vec{\varrho} \cdot \vec{r}_{\xi}) - \gamma(\vec{\varrho} \cdot \vec{r}_{\eta})}{J^2(\alpha\gamma - \beta^2)} \tag{16}$$

where

$$\vec{\varrho} = \alpha\vec{r}_{\xi\xi} - 2\beta\vec{r}_{\xi\eta} + \gamma\vec{r}_{\eta\eta} \tag{17}$$

Though the β term vanishes for orthogonal conditions, it is retained for the case where the orthogonality constraint must be necessarily compromised. These expressions for the control functions are, of course, valid anywhere within the field. Typically, the control functions are evaluated at the surface boundaries and interpolated by some means into the interior. However, more flexibility can be obtained by instituting a more generalized procedure in which interior regions can come under direct control as well. This approach, however, introduces additional difficulties as the interpolation for the control functions across the surface is more complicated.

Control Function Interpolation

The solution of equation (1) includes the effects of a source or nonhomogeneous term which involves both P and Q, hence, these quantities must be specified across the grid. Rather than evaluate the interior control functions based on an initial distribution of grid points, we opt for a method of interpolation, similar to the approach taken by Sorenson and Steger[5], in which the value of the control functions within the interior is determined from the boundary values. For the case in which control functions are specified only on the surface boundaries, transfinite interpolation is used to determine the control functions within the interior. This approach yields a reasonably smooth variation and also affords a good measure of flexibility due to the arbitrary form of the blending functions. These functions permit the user to control the damping rate of the control functions. In the current implementation a cosine function is used as the functional form of the blending functions with the decay half-period, in terms of a point count, specified by the user. Outside the region of user control the grid becomes locally Laplacian which yields the smoothest grid possible.

While extremely useful, this approach does have its shortcomings as the blending functions are most easily constructed to be a function of grid index. Repeated iteration on the grid alters the physical distance over which the control functions are damped to zero; this is, to varying degrees, reflected in the resulting grid. A variation on this method which attempts to fix the physical decay distance has also been formulated and will be described in a later section.

For the case in which interior points are to come under direct control and, hence, control functions are to be specified within the interior of the surface, the interpolation for the control functions becomes somewhat more involved. The methodology currently incorporated into the computer code uses a sequential application of transfinite interpolation at the surface boundaries and at all interior regions of the surface. This particular approach is quite limited as it assumes that, for any point within the grid, the control function is determined by a single set of boundaries only, be it the surface boundaries or those of any interior region. Stated another way, this limitation permits repeated application of transfinite interpolation for all regions with the stipulation that the effects of any region vanish before the effects of any other region are introduced. For the region outside that of an interior patch, the control function variation is determined by using a combination of 1D and 2D interpolations

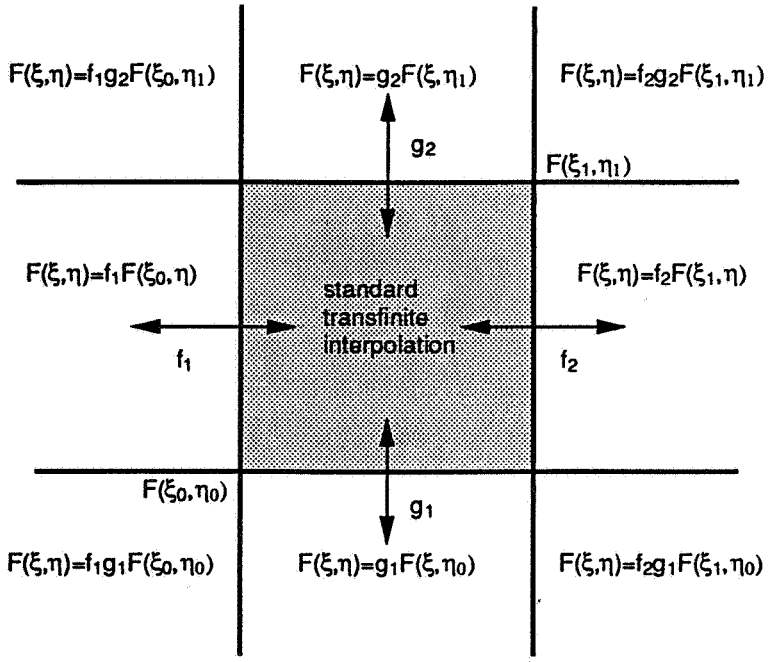


Figure 1. Determination of control functions in the vicinity of an internal patch.

Though this technique is not as robust as we might like, it remains a useful option. This approach is under continued development.

Surface Fitting of Control Functions

A shortcoming inherent in the basing of interpolation blending functions on grid index is evident as the influence of the control functions may be observed to become restricted to an increasingly smaller region of the surface as the grid is refined. A variation of this method which maintains the initial physical decay distance of the control functions has been formulated. Control functions are evaluated as described above though the interpolation is performed on the original surface grid which is, of course, unchanging. At each point on the original grid the control function first- and second-derivatives ($P_\xi, P_{\xi\xi}$, etc.) in transformed space are first evaluated. An interpolation onto the current grid using a Taylor series expansion about the "neighbor" points in the initial surface is then made. These "neighbor" points are determined through a minimization process which will be described in the next section. This method for interpolation tends to provide an additional measure of control as the effects of the control functions are typically felt over a larger area of the surface.

Surface Coordinate Correction

In application it has been found that, in regions of high surface curvature, the evolving surface tends to deviate from the initial or basis surface. To preclude the deviation, a surface coordinate correction scheme has been devised which makes use of "neighbor" point relationships and both physical and transformed quantities to adjust the coordinates to second-order after the grid has converged. A variation of this procedure was used to adjust surface grid points in a three-dimensional adaptive grid scheme with excellent results[6].

The surface correction method is non-iterative and makes use of the basis grid and its mapping to computational space

$$(x, y, z)_o \rightarrow (\xi, \eta)_o \tag{18}$$

where the subscript o denotes the basis surface.

The point on the basis surface closest to (ξ, η) is then determined through a minimization process as (ξ_o, η_o) and distances in the transformed plane are determined

$$\Delta\xi_o = (\xi_x)_o\Delta x + (\xi_y)_o\Delta y + (\xi_z)_o\Delta z \tag{19}$$

$$\Delta\eta_o = (\eta_x)_o\Delta x + (\eta_y)_o\Delta y + (\eta_z)_o\Delta z \tag{20}$$

where

$$\Delta\vec{r} = \vec{r}(\xi, \eta) - \vec{r}_o(\xi_o, \eta_o) \tag{21}$$

The "neighbor" point is obtained when equations (19) and (20) have been minimized. The corresponding surface point is then obtained from the Taylor series expansion about the basis grid point

$$\begin{aligned}\vec{r}(\xi, \eta) = {} & \vec{r}_o(\xi_o, \eta_o) + \left(\frac{\partial\vec{r}_o}{\partial\xi_o}\right)\Delta\xi_o + \left(\frac{\partial\vec{r}_o}{\partial\eta_o}\right)\Delta\eta_o + \left(\frac{\partial^2\vec{r}_o}{\partial\xi_o\partial\eta_o}\right)\Delta\xi_o\Delta\eta_o \\ & + \frac{1}{2}\left\{\left(\frac{\partial^2\vec{r}_o}{\partial\xi_o^2}\right)\Delta\xi_o^2 + \left(\frac{\partial^2\vec{r}_o}{\partial\eta_o^2}\right)\Delta\eta_o^2\right\}\end{aligned} \tag{22}$$

INTERACTIVE CODE DEVELOPMENT

Development of an interactive tool incorporating the surface grid generation methodology is in progress. The final code is to be included as an option in the interactive surface generation code currently under development by Warsi[7] which encompasses aspects of interactive curve generation, surface generation and a novel approach for surface point adjustment termed "surface constrained transfinite interpolation". There are further plans to include surface constrained Bezier curves as well which will provide an invaluable means to effect local grid control. In the implementation of the current approach, pop-up menus are used to promote ease of use and facilitate data entry. Specification of boundary/interior step sizes and control function decay rates are performed interactively with the mouse. Surface evolution is displayed in real time during which the user has the option of revising all constraints and inputs and then restarting or continuing the elliptic generation process. Furthermore, interior surface grid patches and associated constraints are also specified interactively.

APPLICATIONS

To demonstrate the capability of the method, various externally generated surfaces have been subjected to refinement. The results to be presented will highlight the strengths and weaknesses of the approach. The techniques developed to circumvent the shortcomings of the method are also evaluated. In addition, the mode of interior grid control is also demonstrated.

The first surface refined by the current method was constructed using an exponentially damped sine wave, the specific funtion having the form

$$y = 3e^{-\frac{3x}{2x_L}} e^{-2|\frac{z}{z_L}|} sin(\frac{4\pi z}{z_L}) \tag{23}$$

which is defined over $0 \leq x \leq x_L$ and $-z_L \leq z \leq z_L$ where $x_L = 10$ and $z_L = 12$. This particular functional form was selected as it would permit an assessment of the method in regions exhibiting significant curvature variation. The initial surface, which is shown in figure 2, has dimensions of 101×41 and is uniform in each direction. With the elliptic surface generator an attempt was made to increase the leading and trailing near-edge resolution and maintain orthogonality. Figure 3 shows the final surface generated after approximately 50 cycles with $\omega = 0.90$. Note also in this application that the left and right boundary points have been permitted to "float" in order to provide orthogonality and the desired spacing in the vicinity of the corners. The control functions over the interior of the surface were interpolated from the 4 edges using standard transfinite interpolation as described earlier. The functional form of the blending functions utilized the cosine function with the decay half-period specified as 10 points. No control functions were applied at the left or right boundaries.

Since the surface equation is available, the degradation introduced through the solver can be easily examined. Given the new "planform" coordinates x and z, a new y coordinate was calculated using equation (23). The difference between the solved y coordinate and that given by equation (23) is shown as a carpet plot in figure 4 in which the vertical dimension has been scaled by a factor of 50 to highlight the discrepancies. Note the "spikes" present

at regions of significant curvature change, curvature and spike amplitude being directly proportional. Also note that at the leading edge, where the finer resolution has been prescribed, the surface degradation is minimal. This leads to the conclusion that the extent of surface degradation is dependent upon local curvature variation and resolution; a problem inherent in discrete surface representation.

A means to restore the correct surface shape was presented in the section entitled "Surface Coordinate Correction". This scheme was applied after the elliptic generation process, the final surface is shown in figure 5. Visual inspection reveals no change in the surface grid point distribution or the surface shape. The corresponding carpet plot of the surface error on the expanded scale is shown in figure 6. Note that the surface shape has been essentially restored.

In the previous application the prescribed damping period for the spacing control function was set at the relatively high value of 10. Despite this relatively long period, the attraction of points to the leading and trailing edges does not appear to be very strong. In the section of the paper entitled "Surface Fitting of Control Functions" an alternate method of determining the variation of the control functions across the surface was described. Application of this method to the sine wave surface, using constraints identical to those of the previous application, produces the surface shown in figure 7. Note that the near-edge resolution is much finer as the effects of the control function are permitted to propagate further out from the respective edge. A carpet plot of the y surface coordinate error is presented in figure 8. Note that the extent of surface degradation at the leading edge is reduced due to the increased resolution. Application of the surface correction method restored the surface shape, though again, no change was visible.

To contrast the methods presented for the interpolation of the control functions, near-edge detail for the initial surface, the surface created using direct transfinite interpolation and that using the surface fitting method is shown in figure 9. The alternate method clearly provides a more powerful means to achieve increased grid control.

The next surface refined by the method involves a generic fuselage forebody with canopy. This configuration, which is shown in figure 10, features a lobed cross section at the canopy region with circular cross sections at both nose and aft regions. It was expected that the near-canopy region would provide the most difficulty for the method due to the relatively large curvature variation. Increased resolution at the symmetry plane was prescribed and, in this case, there was a visible degradation of the surface in the vicinity of the canopy. Exploded views of the canopy region for the initial, intermediate and corrected surfaces are shown for comparison in figure 11 in which the surfaces have been reflected in order to highlight the degradation. For comparison with the initial surface shown in figure 10, the final corrected surface grid is shown in figure 12. Note the extensive movement of points to the canopy region. This surface was obtained with $\omega = 1.0$ after approximately 75 iterations.

The final application involves an attempt to control the grid within the domain interior. The exponentially damped sine wave surface is used again, though in this case, control

functions are specified rather than interpolated on the boundaries of a rather large section of the interior in order to provide for increased local resolution. The control functions across the remainder of the surface interior are determined using the alternate means described earlier within this paper. The resulting grid is shown in figure 13. Note that the spacial variation of the step size is quite smooth although orthogonality has not been achieved near some portions of the interior boundary. This feature of the code is under continuing development.

CONCLUSIONS

The utility and robustness of the method has been shown. The approach permits elliptic generation on somewhat arbitrary surfaces with relative ease. Though the method does, in essence, require a form of surface splines, their use is transparent to the user. The scheme devised to correct for surface degradation has been found to perform quite well. In addition, the variation on the approach used in the interpolation of the control functions has been found to permit increased grid control. Optional control of interior grid points has been demonstrated although the approach is not yet as general as necessary.

FUTURE WORK

Work to formulate a more robust means to permit user control within the interior of arbitrary surfaces is in progress. This would bring arbitrary grid lines as well as rectangular regions under user control. The means to interpolate the control functions on the remainder of the grid is also under development. Control function damping rates will also be double valued at the desired sections in order to provide bi-directional grid control. Once the method has been fully developed it will be included as an option in the surface code under development by Warsi[7].

REFERENCES

[1] Sorenson, R., "A Computer Program to Generate Two-Dimensional Grids About Airfoils and Other Shapes by the Use of Poisson's Equation", NASA TM 81198, May 1980.

[2] Thompson, J., "A Composite Grid Generation Code for General 3D Regions-the EAGLE Code", *AIAA Journal*, Vol. 26, No. 3, March 1988.

[3] Steinbrenner, J., Chawner, J. and Fouts, C., "The GRIDGEN 3D Multiple Block Grid Generation System", WRDC TR 90-3022, July 1990.

[4] Warsi, Z.U.A., "Numerical Grid Generation in Arbitrary Surfaces through a Second-Order Differential-Geometric Model", *Journal of Computational Physics*, Vol. 64, No. 1, May 1986.

[5] Sorenson, R. and Steger, J, "Numerical Generation of Two-Dimensional Grids by the Use of Poisson Equations with Grid Control at Boundaries", NASA CP 2166, October 1980.

[6] Kania, L., "Three-Dimensional Adaptive Grid Generation with Applications in Nonlinear Fluid Dynamics", AIAA 92-0661, presented at the 30^{th} Aerospace Sciences Meeting, Reno, NV, January 1992.

[7] Warsi, S. A., "Algebraic Surface Grid Generation in Three-Dimensional Space", In *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA-CP-3143, April 1992.

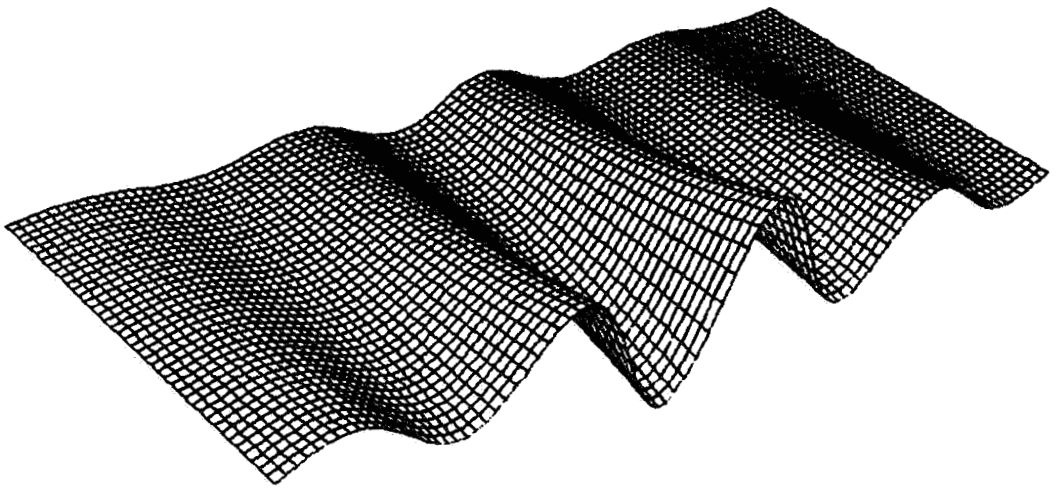


Figure 2. Damped sine wave surface.

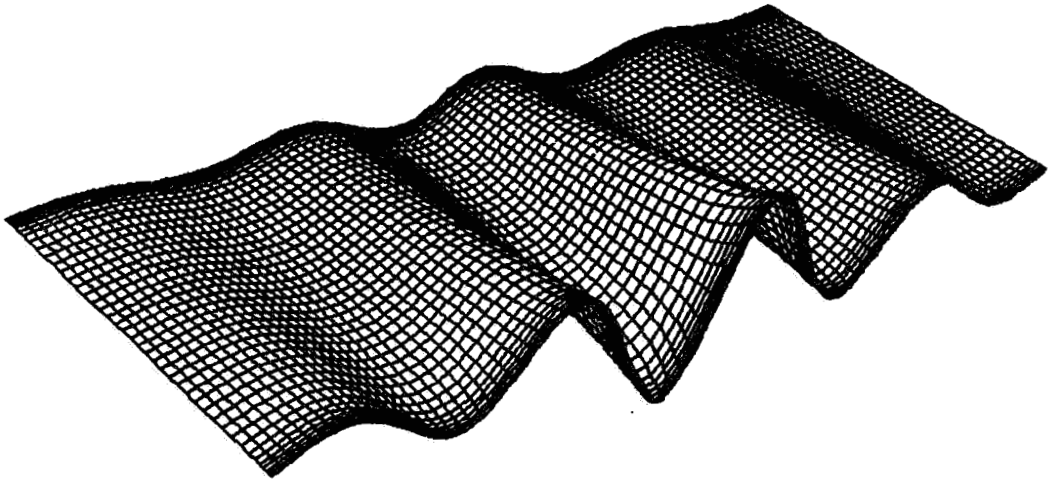


Figure 3. Elliptically generated surface with control functions by standard TFI.

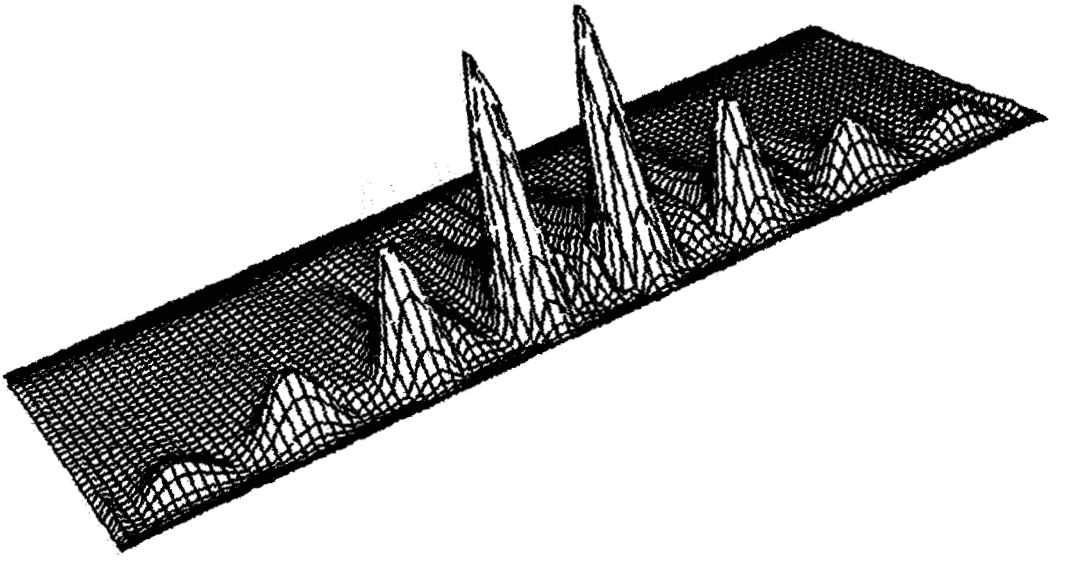


Figure 4. Surface error distribution on elliptic surface.

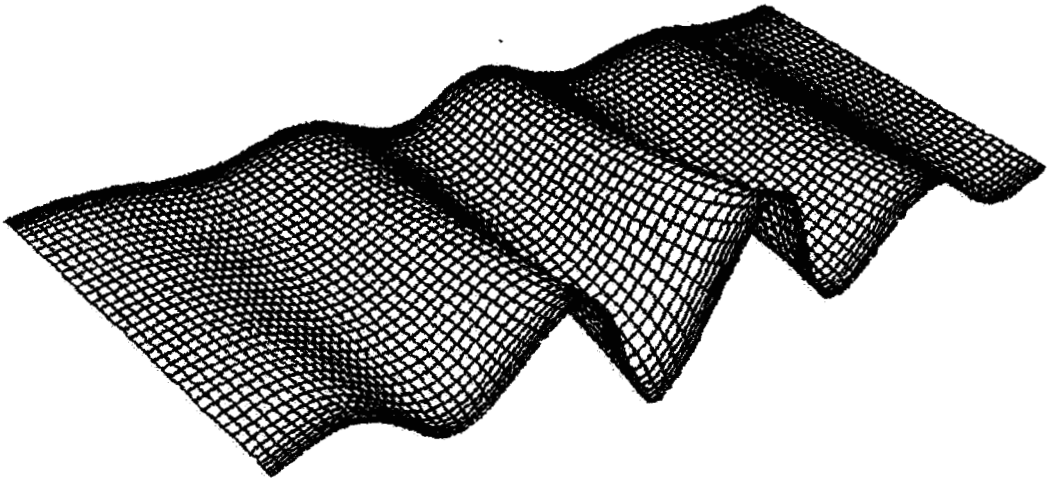


Figure 5. Elliptically generated surface after surface correction.

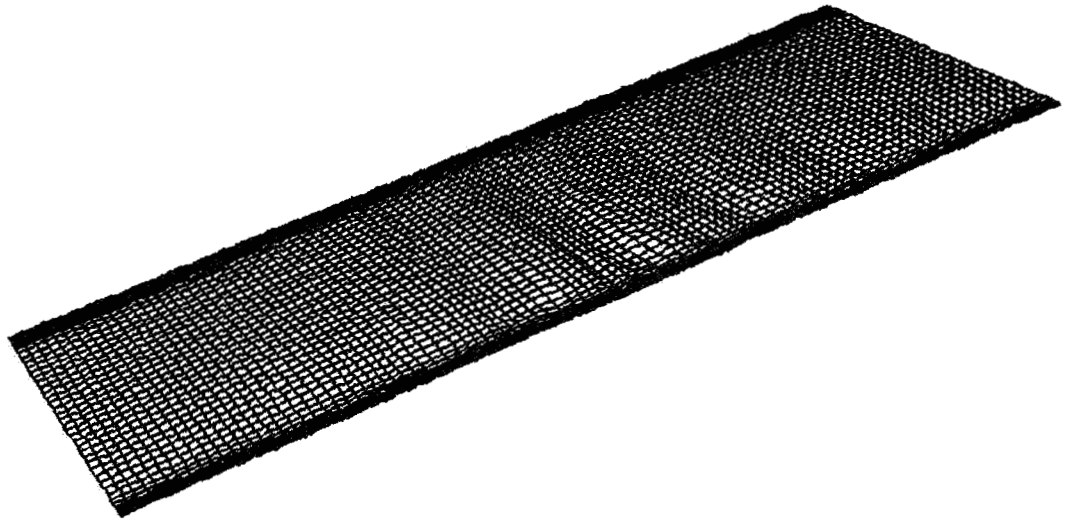


Figure 6. Surface error distribution after surface correction.

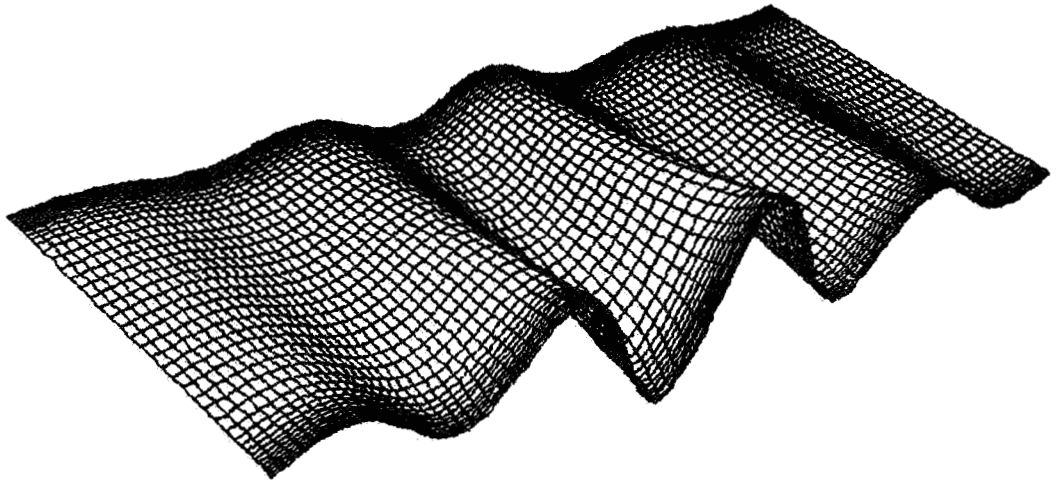


Figure 7. Elliptically generated surface using surface fitting of control functions.

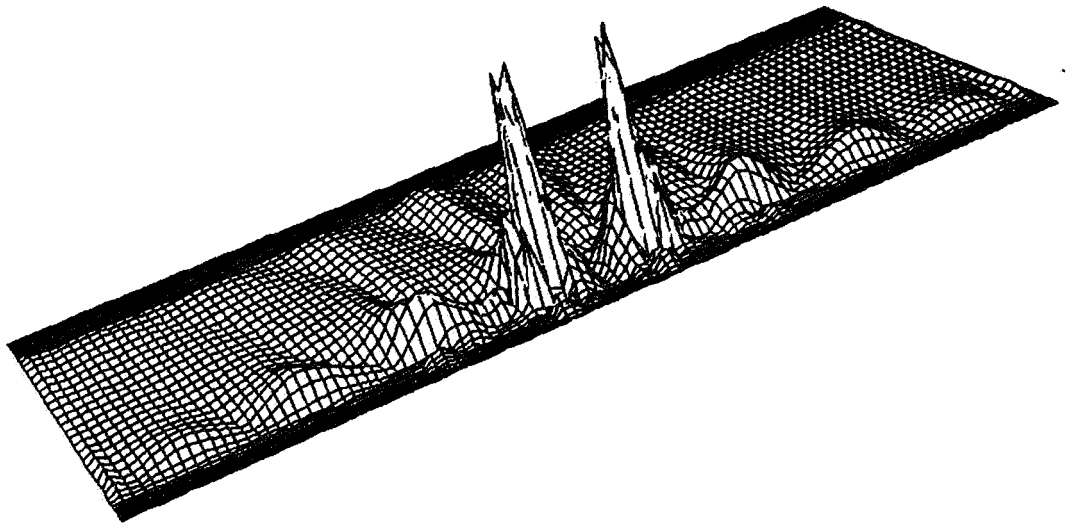


Figure 8. Surface error distribution on elliptic surface.

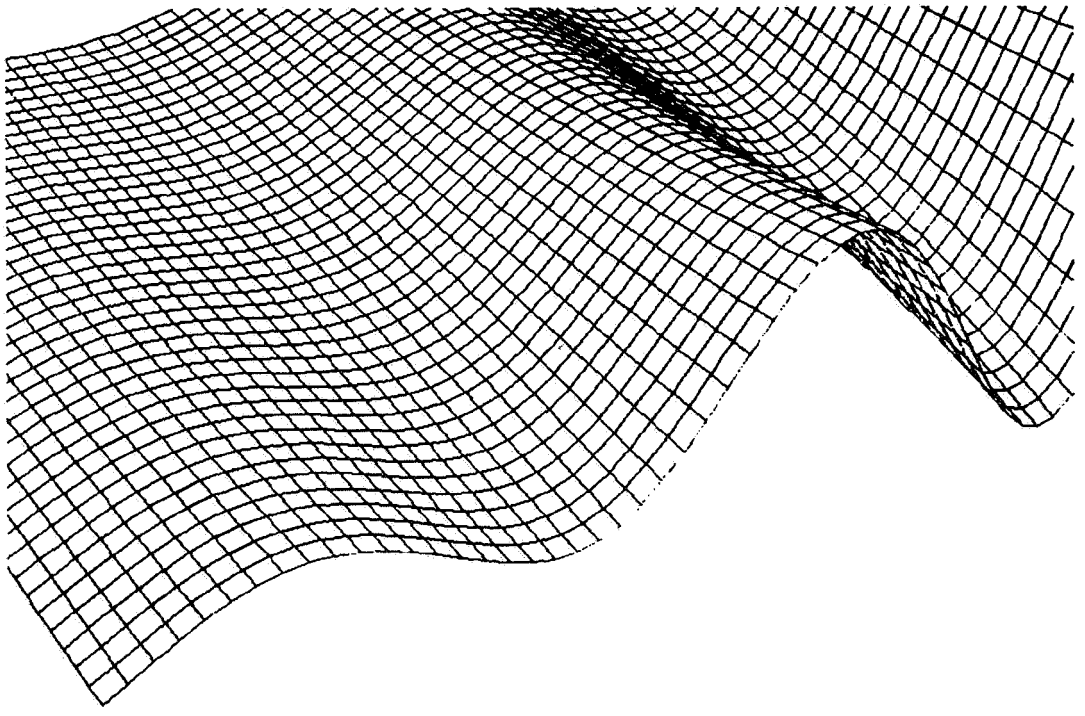


Figure 9a. Damped sine wave surface, near-edge view.

C-2

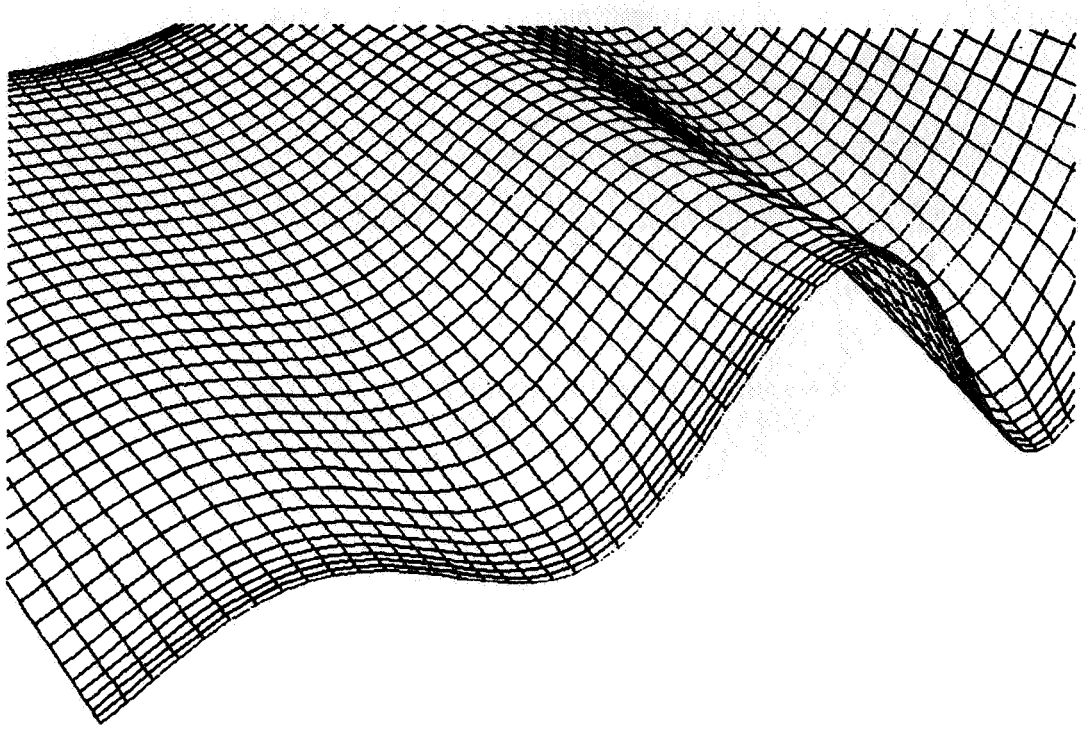


Figure 9b. Elliptic surface, near-edge view (control functions by standard TFI).

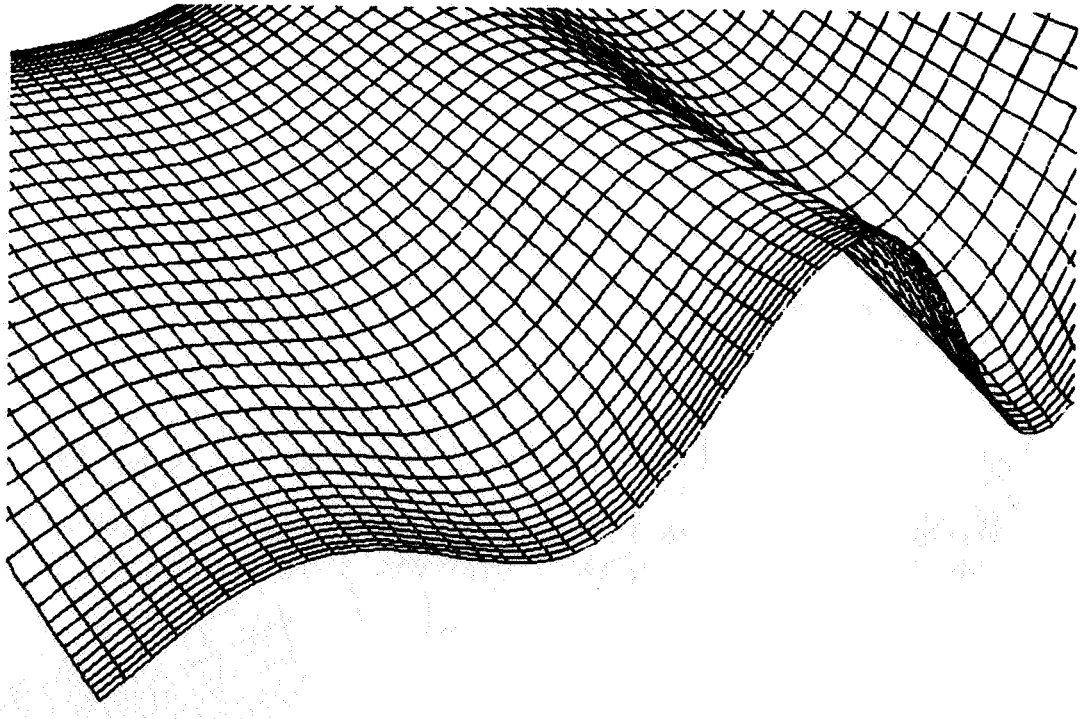


Figure 9c. Elliptic surface, near-edge view (surface fitting of control functions).

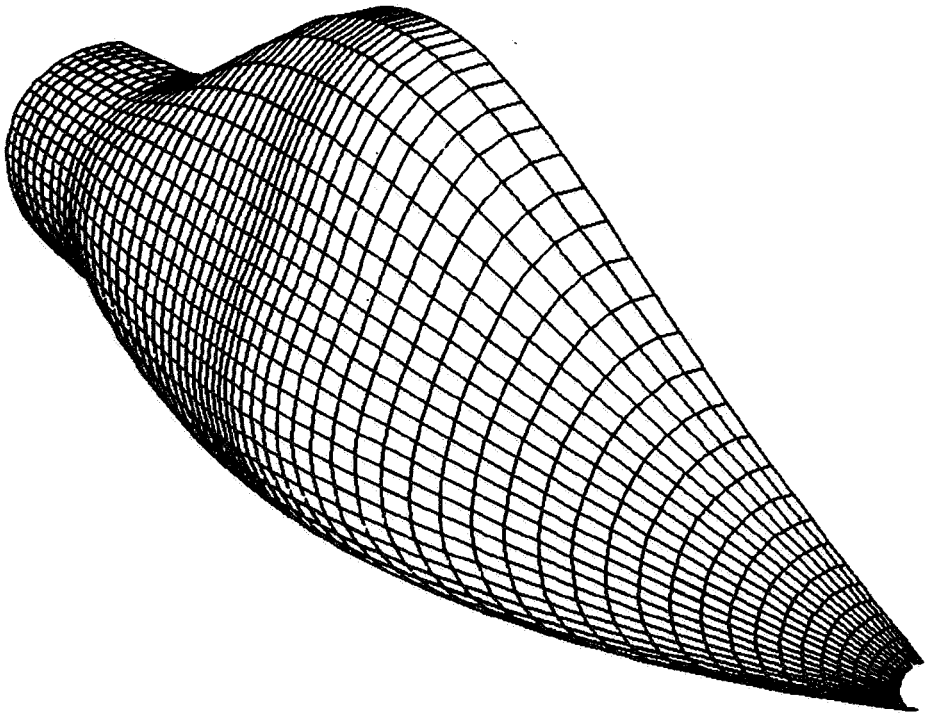


Figure 10. Generic fuselage forebody surface.

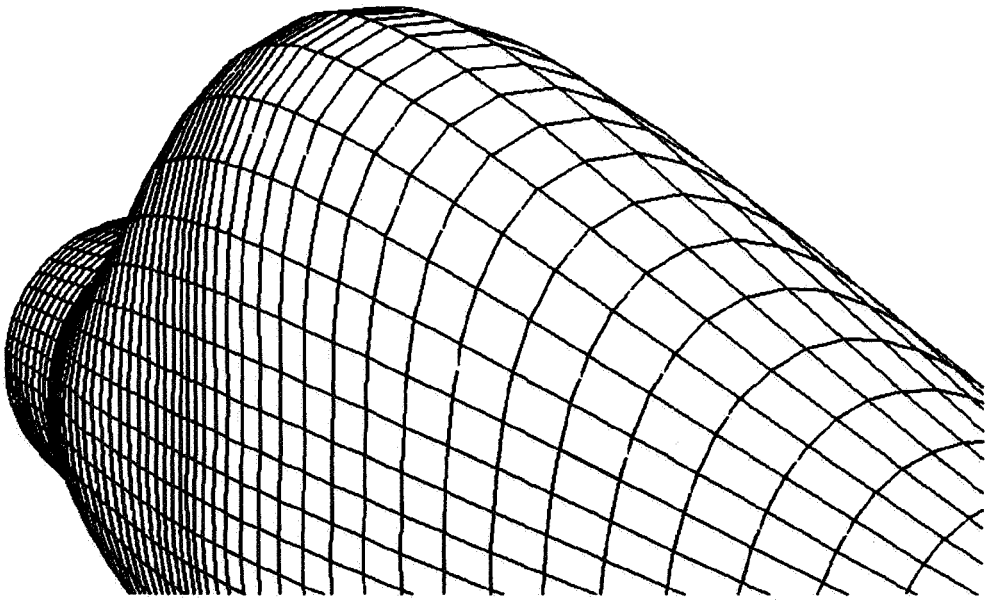


Figure 11a. Generic fuselage forebody surface, near-canopy view.

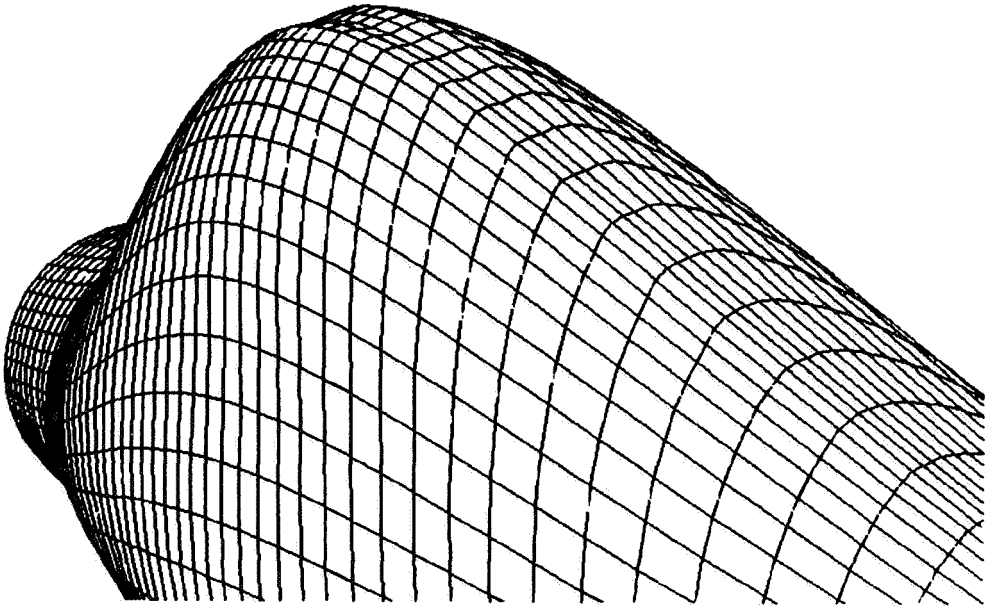


Figure 11b. Elliptic fuselage forebody surface, near-canopy view.

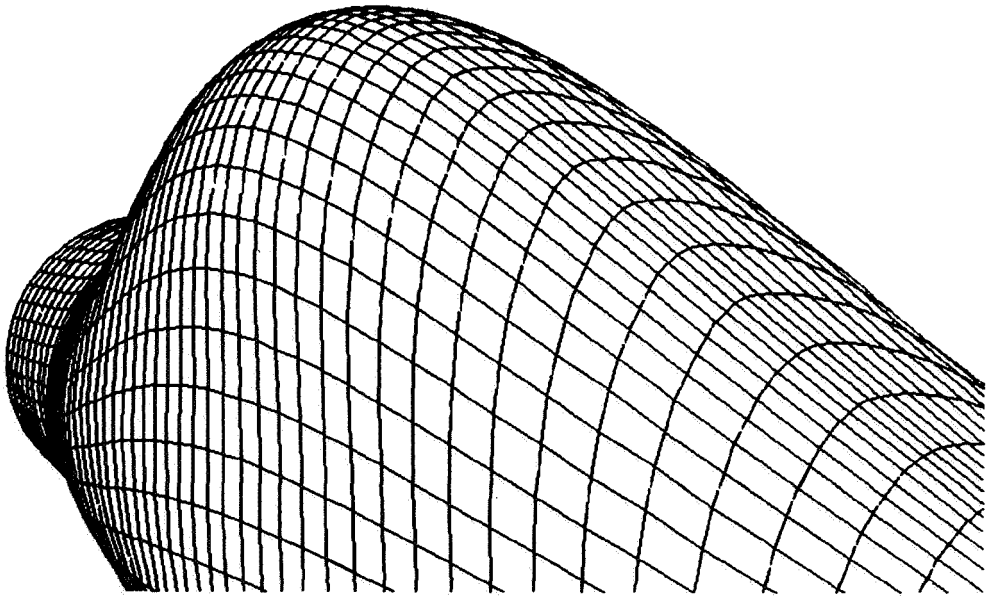


Figure 11c. Surface corrected elliptic fuselage forebody surface, near-canopy view.

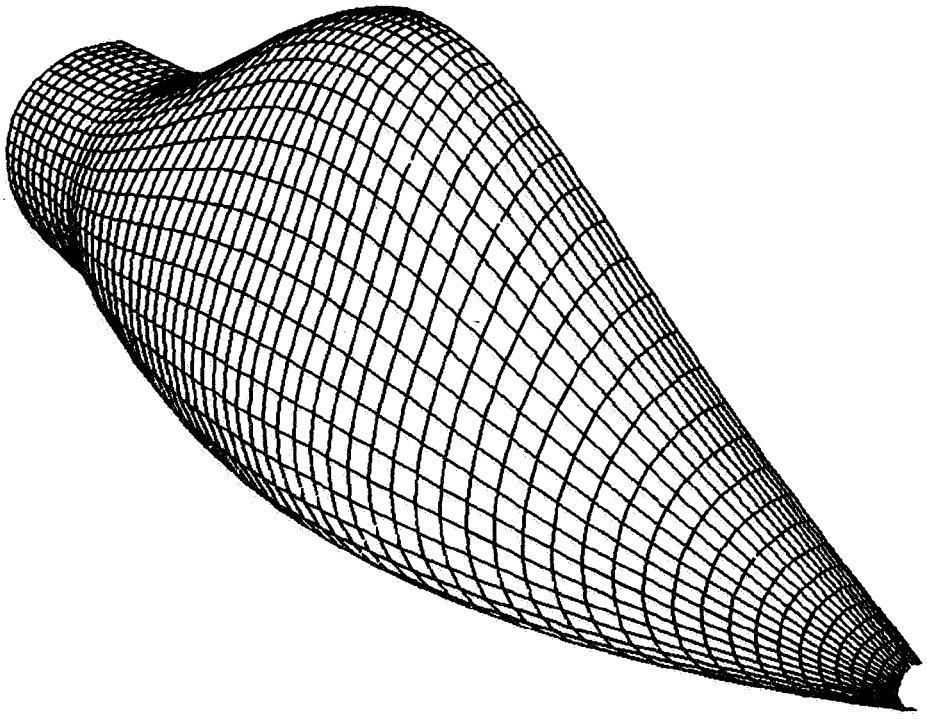


Figure 12. Final elliptic fuselage forebody surface.

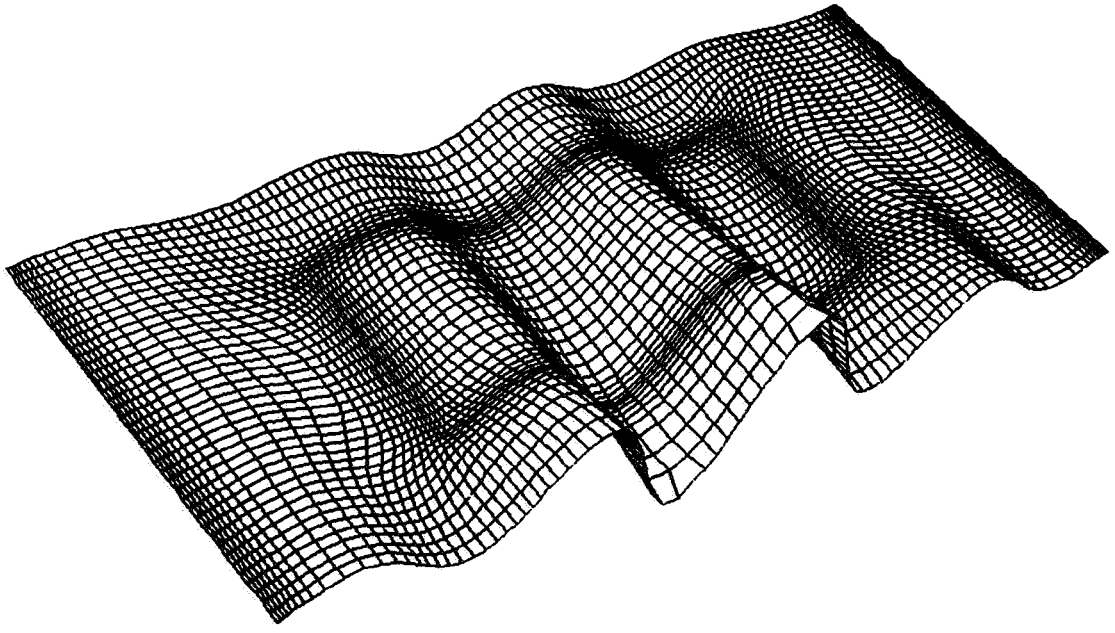


Figure 13. Elliptic surface with interior control option.

IGGy - AN INTERACTIVE ENVIRONMENT FOR SURFACE GRID GENERATION*

Nathan C. Prewitt
Sverdrup Technology Inc.
Eglin AFB, FL

SUMMARY

A graphically interactive derivative of the EAGLE boundary code is presented. This code allows the user to interactively build and execute commands and immediately see the results. Strong ties with a batch oriented script language are maintained. A generalized treatment of grid definition parameters allows a more generic definition of the grid generation process and allows the generation of command scripts which can be applied to topologically similar configurations. The use of the graphical user interface is outlined and example applications are presented.

INTRODUCTION

The EAGLE code [1-3] was the first large-scale, widely accepted, multi-block, structured grid generation code for general CFD applications. It is broken into two pieces: the EAGLE boundary code [2] and the EAGLE grid code [3]. With the grid generation problem approached as the solution to a boundary-value problem, the boundary code is used to generate surface grids which define the boundaries and the grid code is used to generate the volume grid from these boundaries. Both codes operate off of a series of batch oriented commands constructed from reading a FORTRAN *namelist*. Together, these commands define a primitive language for grid generation. Although it has been surpassed by more user-friendly interactive grid generation systems, EAGLE still displays some advantages over these systems via its batch oriented command structure.

One of the advantages of the EAGLE code is its ability to build scripts of the grid generation process using its language-like commands. The scripts not only serve as a convenient way of storing or exchanging grids but also force the user to obtain a good mental picture of the grid generation process. Command scripts are most powerful through the use of indirectly addressed parameters. These parameters are in the form of three component vectors representing points in space, integer values representing numbers of points on boundary segments, and real values representing grid point spacings along segments. The use of these parameters allows the user to redimension the grid, change the grid point distribution, or even alter the geometry of the physical boundaries by editing only the commands which define these parameters. Once altered, the script is reprocessed, in a

*Work done under contract with the Air Force Development Test Center, Eglin AFB, Florida

batch mode, with the alterations propagating through both the surface and grid codes to produce the final grid.

When given a new configuration about which a grid is to be generated, the first step is to plan the grid topology. From the standpoint of block structured grids, this includes the number, location, and connectivity of the grid blocks [4]. This topology definition is the basic input to the grid code. In the construction of the boundaries of each block, it is necessary to specify certain points in space, the number of points along the boundaries, and the distribution of these points [1]. The specification of these parameters is the initial step in setting up input for the boundary code. References [1] and [5] give several example applications and stress the advantages of using indirectly addressed parameters in the efficient application of the EAGLE boundary code. In general, however, the specification of grid parameters is not flexible enough to allow the generation of generic scripts which can be applied to topologically similar configurations.

IGGy (Interactive Grid Generation sYstem) is a graphically interactive derivative of the EAGLE boundary code which provides a more user-friendly, intuitive interface. It is an interactive front end to the basic geometry engine that is found in the EAGLE boundary code and allows you to interactively build and execute commands and immediately see the results. The definition of the grid parameters has been generalized from point coordinates, numbers of points, and spacings to include any vector, integer, or real values needed. This extension of the indirectly addressed parameters allows a more generic definition of the grid generation process; thus, scripts can more easily be applied to topologically similar configurations.

DESIGN PHILOSOPHY

IGGy is designed for the novice user who is familiar with the applicability of different grid generation methods, but is not an expert at using the code. He/she may be new to grid generation, or may be someone whose expertise lies in another area of computational simulation and who uses grid generation only as a prerequisite to accomplishing another task. Thus the user interface is designed to be easy to use and very intuitive.

IGGy uses a namelist-like command syntax which identifies each command with a descriptive name. The commands may be entered at the console or selected from a menu system. When building a command from the menu system, the user is prompted for all necessary input. To reduce the amount of input required to accomplish a single task, IGGy commands have been patterned after the operation of the UNIX operating system: commands do a single, well defined task and do it well. It is up to the user to chain these basic commands together to create the desired effect. If an additional capability is needed, a new command is created rather than adding additional input variables to an existing command. This simplifies the command input, gives the user a stronger sense of the command capabilities, and hopefully builds a more robust, more easily maintained software system.

IGGy operates like a grid microprocessor with only one register. Many segments can be held in

memory, but only one can be operated on at a time. This segment is said to be in *current* position. Most commands either generate a new *current* segment or operate on the *current* segment. These commands fall into the two categories of geometry definition and geometry manipulation. The geometry definition commands consist of the most basic Computer Aided Design (CAD) capabilities. More complex curve and surface shapes need to be imported from an outside source. The geometry manipulation commands provide the capability to distribute points on curves and surfaces in a manner appropriate for CFD analysis. This formal separation of tasks simplifies the command operation and reduces the input required for a given command.

COMMAND STRUCTURE

The IGGy input commands are of the following format:

E$INPUT ITEM = "operation", variable = value, ... $

where the string assigned to ITEM identifies the command being executed, and the list of comma separated assignments specify relevant quantities for the command. Arrays are assigned using a comma separated list of values; while particular elements of an array are assigned using the array name followed by the array element within parentheses. Since only one namelist (INPUT) is used and each command is identified by a string assigned to the variable ITEM, the following shorthand notation is available:

$"operation", variable = value, ... $

where the string immediately following the opening $ is assigned to ITEM.

To implement the more general definition of the parameters, a new parser has been written for IGGy using the standard UNIX utilities *yacc* [6] and *lex* [7]. Yacc is a general purpose parser generator; while lex performs low level lexical analysis. Lex identifies low level constructs, called tokens, by matching user specified regular expressions. Yacc groups these tokens, according to user specified context-free grammar rules, to build higher constructs. The output from these utilities is C code which can be linked with the main routines and called whenever an input command is to be parsed.

In IGGy's parser, lex recognizes variable names, integer values, floating points values, quoted strings, parameter references, intrinsic functions, and punctuation symbols, and returns appropriate tokens. Yacc uses these tokens to build integer expressions, real expressions, vector expressions, and assignment constructs which conform to the syntax specified by the grammar rules. To complete the parsing of a command, yacc evaluates all expressions and performs the assignments.

The syntax used to represent references to stored parameters are shown in Table 1, where *index* is any integer value. Any variable can access a parameter stored at the specified index if the variable

and stored parameter are of the same type. After parsing, the variable will contain the value stored at the specified index. This eliminates the need for special coding to recognize parameter references.

Table 1: Syntactic Constructs for Indirectly Addressing Parameters

~*index*	vector storage reference
!*index*	integer storage reference
#*index*	real storage reference

Table 2 lists the valid syntactic constructs for generating integer expressions. Here, *ival* represents any integer value, *iexp* represents any valid integer expression, and *rexp* represents any valid real expression. The capital letters and punctuation marks are part of the syntax. The use of *iexp* in the definition of integer expressions denotes that this definition is recursive. The order of precedence of the arithmetic operators is fashioned after FORTRAN or C; however, parentheses can be used to alter the order of evaluation of integer subexpressions. Any integer variable or array can be assigned a value using any of these constructs.

Table 2: Valid Integer Expressions

!*ival*	integer parameter reference
ival	integer value
ival:*ival*	range of integers
iexp + *iexp*	addition
iexp − *iexp*	subtraction
iexp * *iexp*	multiplication
iexp/*iexp*	division
MOD(*iexp*,*iexp*)	modulus value
iexp ** *iexp*	exponentiation
ABS(*iexp*)	absolute value
MIN(*iexp*,*iexp*)	minimum value
MAX(*iexp*,*iexp*)	maximum value
−*iexp*	unary minus
(*iexp*)	subexpression grouping
INT(*rexp*)	type conversion

Table 3 lists the valid syntactic constructs for generating real expressions. Here, *ival* represents any integer value, *rval* represents any real value, *iexp* represents any valid integer expression, *rexp* represents any valid real expression, and *vexp* represents any valid vector expression. The capital letters and punctuation marks are part of the syntax. This definition is also recursive and parentheses are available for subexpression grouping. Any real variable or array can be assigned a value using any of these constructs.

Table 4 lists the valid syntactic constructs for generating vector expressions. Again, *ival* represents any integer value, *rexp* represents any real expression, and *vexp* represents any vector

Table 3: Valid Real Expressions

#ival	real parameter reference
rval	real value
PI	3.14159265...
rexp + *rexp*	addition
rexp − *rexp*	subtraction
rexp * *rexp*	multiplication
rexp/*rexp*	division
rexp ** *rexp*	exponentiation
SIN(*rexp*)	sine
COS(*rexp*)	cosine
TAN(*rexp*)	tangent
ASIN(*rexp*)	inverse sine
ACOS(*rexp*)	inverse cosine
ATAN(*rexp*)	inverse tangent
LOG(*rexp*)	natural logarithm
LOG10(*rexp*)	common logarithm
SQRT(*rexp*)	square root
ABS(*rexp*)	absolute value
MIN(*rexp*,*rexp*)	minimum value
MAX(*rexp*,*rexp*)	maximum value
vexp.X	x component
vexp.Y	y component
vexp.Z	z component
|*vexp*|	vector magnitude
vexp.*vexp*	dot product
−*rexp*	unary minus
(*rexp*)	subexpression grouping
REAL(*iexp*)	type conversion

expression. All other punctuation is part of the syntax. The definition of vector expressions is also recursive; however, braces and brackets are used to alter the evaluation of vector subexpressions. A vector variable is any real array with three elements which correspond to the three Cartesian coordinate directions. The most basic form of specifying a vector value is three, comma separated real values enclosed in curly braces. If the curly braces are replaced with square brackets, the specified vector will be normalized. The use of braces or brackets is required to obviate a context sensitive ambiguity that would exist otherwise.

Table 4: Valid Vector Expressions

$\tilde{}ival$	vector parameter reference
$\{rexp,rexp\}$	2d vector
$\{rexp,rexp,rexp\}$	3d vector
$[rexp,rexp]$	2d normalized vector
$[rexp,rexp,rexp]$	3d normalized vector
$rexp * vexp$	scalar/vector multiplication
$vexp/rexp$	vector/scalar division
$vexp + vexp$	vector addition
$vexp - vexp$	vector subtraction
$vexp\hat{}vexp$	vector cross product
$vexp * vexp$	component by component multiplication
$vexp/vexp$	component by component division
$-vexp$	unary minus
$\{vexp\}$	vector grouping
$[vexp]$	normalized vector grouping

USER INTERFACE

IGGy is written for use on Silicon Graphics IRIS workstations using IRIS Graphics Language (GL) calls. The software development was started on an IRIS 3000 series workstation, and its use has been extended to the line of IRIS 4D workstations. Compatibility with the older hardware platform has been maintained through this version of the software.

Upon executing IGGy, a single graphics window is opened. This window is 1024 by 768 pixels in dimensions, and therefore takes up the full screen on the lower resolution 3000 workstation and most of the screen on the higher resolution 4D workstations. On the 4D machines, this window may be expanded to fill the entire screen, or may be left at the default size to provide easier access to other application windows.

Figure 1 shows the layout of the main window. To make the software easier to use, most of its

capabilities are displayed on the screen. The buttons, which are displayed, may be *picked* using the cursor and the left mouse button. This is done by positioning the cursor over the button using the mouse, and pressing and releasing the left mouse button. Some regions of the screen, which do not appear as buttons, may also be picked. These regions are discussed where applicable.

As can be seen in Figure 1, the main window is broken into seven pseudo-windows. These are not true windows with respect to the window manager but are merely reserved sections (viewports) of the main window. They can not be resized independently, nor can they be moved relative to each other. Each of these windows is described in greater detail in the following sections.

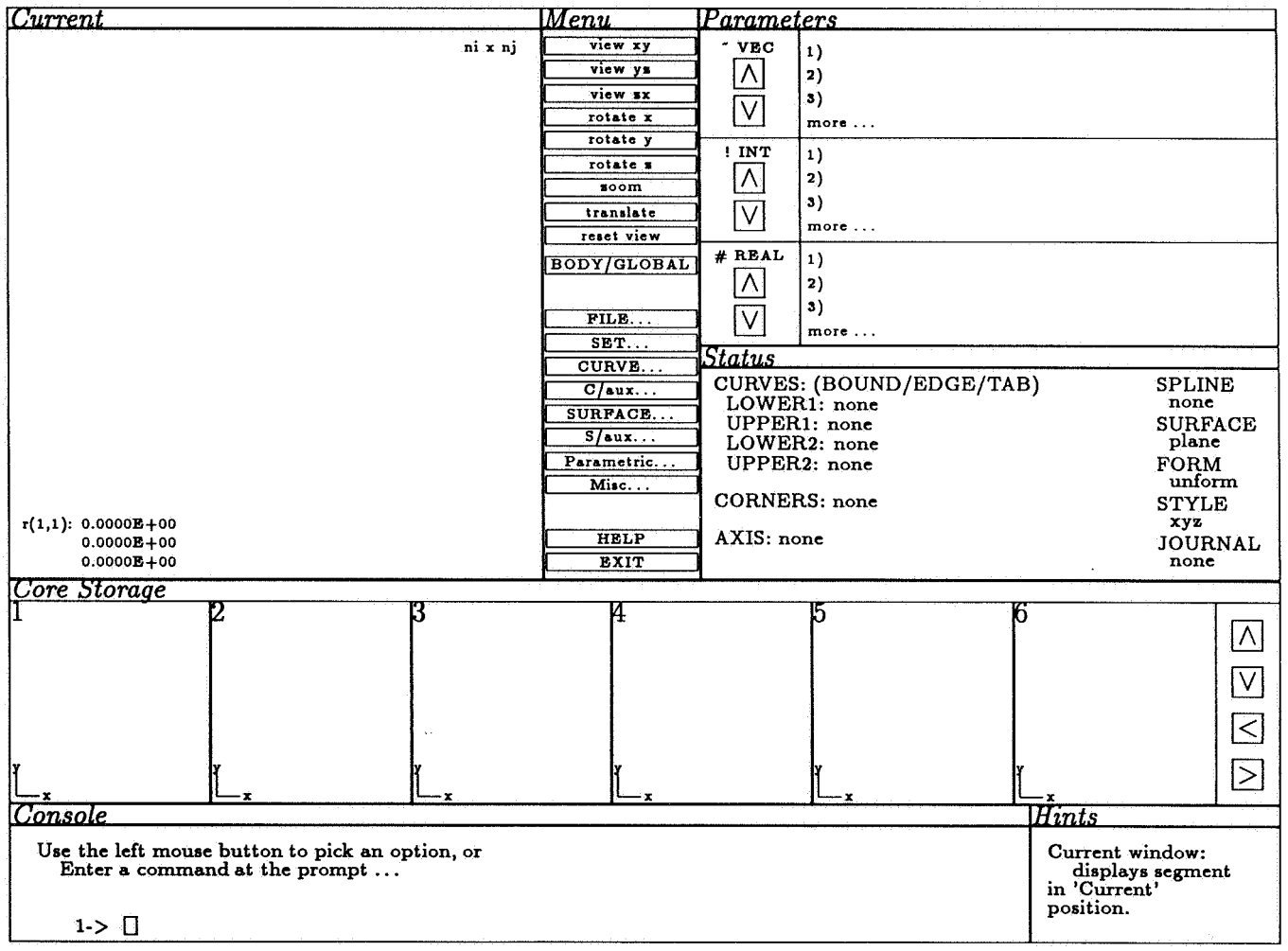


Figure 1: Layout of IGGy's main screen.

Current Window

The Current window displays a wireframe drawing of the segment in *current* position. The curvilinear coordinate lines which construct this segment are color coded, with green lines being in

the first coordinate direction (lines of constant η, or ξ lines) and yellow lines being in the second coordinate direction (lines of constant ξ, or η lines). A cross hatch of red line segments is used to highlight a particular node on the segment. This highlight can be moved with the arrow keys. The left and right arrow keys decrease and increase, respectively, the first index of the highlighted point; while the up and down arrow keys increase and decrease, respectively, the second index. The coordinates of the node that is highlighted are displayed in the lower left corner of the Current window; and in the upper right corner are displayed the dimensions of the *current* segment.

Menu Window

The menu window contains an array of buttons broken into three sections. The top section is used to control the view of the *current* segment. These viewing manipulation buttons allow the user to rotate the segment, translate the segment, scale the segment, and change viewing direction. The last button in this section, labeled "BODY/GLOBAL", is used to toggle between different axes of rotation. When BODY is highlighted in red, rotations are defined about the body axes. When GLOBAL is highlighted, rotations take place about a global set of axes defined relative to the screen. To use the rotation, translation, or zoom buttons, pick the appropriate button. The button chosen will be highlighted in cyan and the cursor will disappear. Move the mouse horizontally for rotating, vertically for zooming, and in both directions for translating. Once a new view is set, click (press and release) any mouse button or keyboard key and the cursor will reappear.

The bottom section contains two buttons: HELP and EXIT. To access the help facility, pick the HELP button. A prompt will request that a second button be picked. The HELP facility will then display, in the Current window, all available information relating to the function represented by the second button. Clicking a mouse button or pressing a keyboard key will close the HELP facility. To exit execution of IGGy, pick the EXIT button. A prompt will request a Continue or Abort signal. Pressing the Enter key represents Continue and will cause execution to cease; while the Esc key represents Abort and will cause normal operations to resume.

The middle section displays the menu system. If a menu item represents a submenu, the submenu name with following ellipsis is displayed. Any menu item may be selected by picking the appropriate button. Picking a submenu displays the items of the submenu. To traverse back up one level in the menu system, you may either press Esc or pick the background area above or below the presently displayed menu options. If a submenu contains more options than can be displayed at once, the last menu button will display "more..." and the remaining options are displayed as a submenu of this button.

The menu system is organized into eight primary menus: "File..." contains all of the i/o commands; "Set..." contains all commands for setting parameters; "Curve..." contains all commands which generate curves; "C/aux..." contains all commands which operate on or alter curves; "Surface..." contains all commands which generate surfaces; "S/aux..." contains all commands which operate on or alter surfaces; "Parametric..." contains all commands which deal with parametric space; and "Misc..." contains all other commands or IGGy specific functions which do not fit into one of the other menus. Some commands operate on both curves and surfaces, and thus have been duplicated in the menu system. If a curve is displayed in *current* position,

picking C/aux... will display all of the commands appropriate for altering it; likewise, if a surface is in *current* position, picking S/aux... will display all of the commands appropriate for altering it.

When using the menu system to build commands and a character value is being prompted for, the legal responses are displayed in a menu. Some non-character variables may also be assigned character values. In this case, a menu of the legal responses is displayed. Picking any of these menu choices causes the appropriate response to be entered into the command being built.

Parameters Window

The Parameters window displays lists of all the indirectly addressable parameters that have been set by the user. The window is separated into three sections corresponding to the three types of parameters: vector, integer, and real. Along the left side of each section is a pair of up and down arrow buttons. These buttons are used to scroll through the parameter lists. When building commands using the menu system, and a vector, integer, or real value is being prompted for, picking a value of the appropriate type from the parameter lists causes the corresponding storage index to be referenced using the proper syntax.

Status Window

The Status window displays the status of various variables and parameters which affect the operation of IGGy. Displayed are the current value of SURFACE, which controls the operation of parametric mode, the current values of FORM and STYLE, which specify the format for file i/o, and the name of the journal file currently being processed. Also displayed are flags to denote that edge curves, bounding curves, or tab curves have been set; corner points have been set; and an axis curve has been set.

When building commands using the menu system, and an integer value is being prompted for, the Status window is cleared and a slider for specifying integer values is displayed as shown in Figure 2. Likewise, whenever a real or vector value is being prompted for, the Status window is cleared and a calculator is displayed as shown in Figure 3. These two facilities allow commands to be entered without switching between using the mouse and using the keyboard.

To operate the slider, pick the slider knob and move the mouse horizontally, while keeping the left mouse button depressed. The present value of the slider is displayed in the small outline, that is centered near the bottom of the window. The knob will wrap around the ends of the slider to allow specification of integers beyond the range displayed. Pressing the Enter key, after moving the slider knob, will cause the value of the slider to be entered into the command being built.

To operate the calculator, numbers are entered by picking the buttons of the numeric keypad, located on the right side of the window, and picking the Enter button. Since the calculator is based on Reverse Polish Notation (RPN), two values must be entered before a binary operator is specified. Values are entered from the bottom of the stack, which is displayed on the left side of the window and which contains three vector registers (only two of which are displayed) and thus nine scalar

registers (three components for each vector). The center section of the calculator window displays the function keys. The tilde button in the upper left corner of the function keys toggles between the scalar functions and the vector functions and also designates a vector Enter from a scalar Enter. Values and vectors can be entered into the calculator by picking an index from the parameter lists or by picking the coordinate display section of the Current window. To designate that these values should be entered into the calculator rather than into the command which is currently being built, this picking should be done using the right mouse button instead of the left mouse button. Once the desired value is calculated, pressing the Enter key will cause the value to be entered into the command being built.

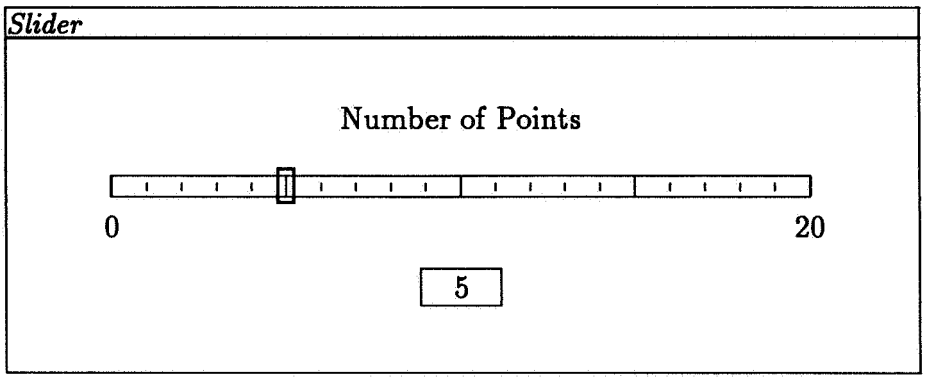


Figure 2: Integer slider.

Core Storage Window

The Core storage window displays wire frame drawings of the segments in core storage. This window is divided into six smaller viewports representing a small range of the core storage indices. The core storage index corresponding to each segment is displayed in the upper left corner of the appropriate viewport. When using the menu system, picking the appropriate viewport causes the corresponding core storage index to be inserted into the command being built. Since only six segments are displayed at once, four buttons are displayed along the right side of the Core storage window: up arrow, down arrow, left arrow, and right arrow. The left and right arrows step through core storage, one index at a time. The up and down arrows shift through core storage, six indices at a time.

The segments displayed in the Core storage window are scaled globally. This ensures that all segments are visible, but causes some segments to become indistinguishable by sight if there exists a great disparity among the size of the segments. In the upper right corner of each small viewport are displayed the dimensions of the segment stored at that index. This can be helpful in identifying smaller segments.

The only viewing manipulations available for the core storage are rotations of 90 degrees. Picking a core segment using the middle mouse button causes a horizontal rotation. Picking a core segment using the right mouse button causes a vertical rotation. The symbol in the lower left corner

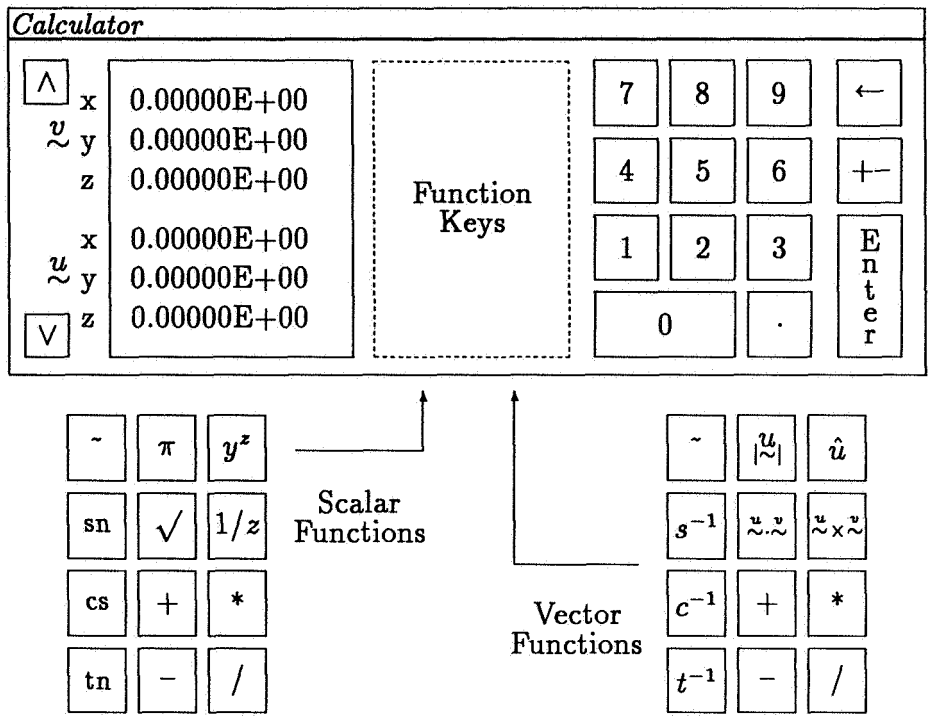


Figure 3: RPN vector/scalar calculator.

of each viewport displays the orientation of the axis of each viewport.

Console Window

The Console window is a text port which allows the user to enter any legal command from the keyboard. When executing commands from the menu system, the Console window is also used to prompt the user for input. For any input variable, the user may use the mouse to access the slider, calculator, or menu choices, or may type the appropriate response from the keyboard.

A command history facility is accessible using Ctrl-up arrow and Ctrl-down arrow. Holding down the Ctrl key and pressing the up arrow key displays the command line previously entered. Successive presses of the up arrow displays commands from earlier in the execution process. Holding down the Ctrl key and pressing the down arrow key displays commands from later in the execution process.

Table 5 shows the line editor commands available from the Console window. All of the commands are implemented as single key control sequences.

Table 5: Command Line Editor Keystrokes

Ctrl-a	move to beginning of line
Ctrl-b	move back one character
Ctrl-d	delete character below the cursor
Ctrl-e	move to end of line
Ctrl-f	move forward one character
Ctrl-h	delete character to left of cursor
Ctrl-k	kill from character to end of line
Ctrl-u	delete the entire line

Hints Window

The Hints window is used to display hints about the operation of IGGy. The text displayed is controlled by the placement of the cursor on the screen. As the cursor is moved around the screen, the message displayed describes the button or window beneath the cursor.

Whenever the slider or calculator is displayed in the Status window, the Hints window is also cleared and two buttons are displayed as shown in Figure 4. These buttons are the graphical equivalents to the Enter and Esc keys. The Abort button can be picked at any time that it is displayed to abort the command currently being built and to force a return to normal operations.

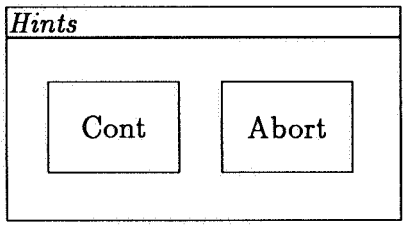


Figure 4: Continue or abort buttons.

EXAMPLE APPLICATIONS

NACA 0012 Airfoil

The following example script generates a simple two-dimensional C-type grid about a symmetric airfoil. This example corresponds to the first example given in reference [1] and is presented to highlight the advantages of using the generalized parameters. The script closely imitates the original example for comparison purposes.

Figure 5 shows the boundaries of the configuration used. The vector parameters that are used as point locations are designated by the appropriate indices inscribed in circles. Each pair of points delineate a boundary segment that is created in the script. Throughout the script, integer parameters are used to define the number of points along the boundary segments and the point distributions are defined using spacings stored in real parameters. In addition to these parameters, two real parameters are used to define the distances from the airfoil to the outer boundary.

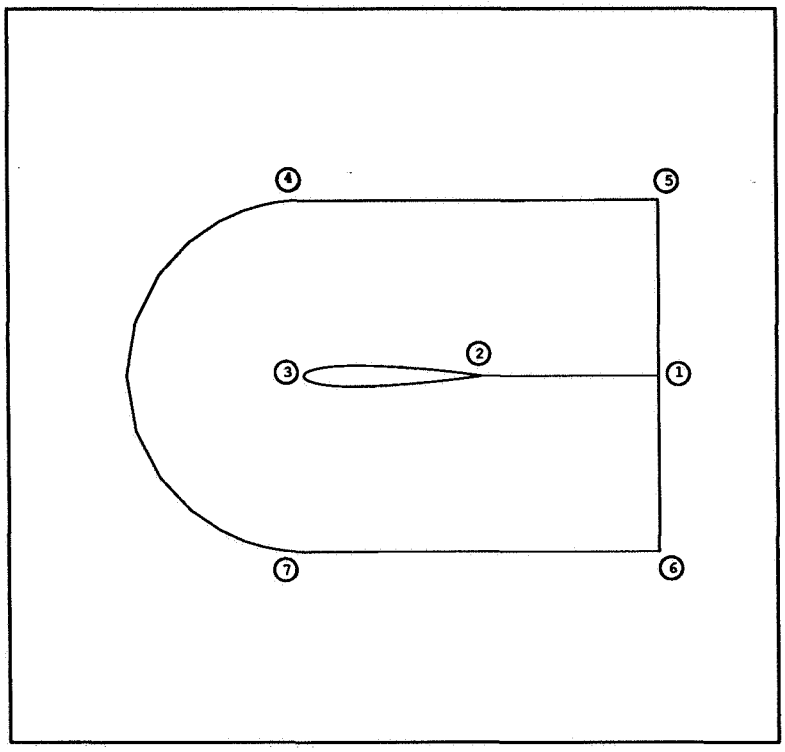


Figure 5: Boundaries of C grid about symmetric airfoil.

Looking at Figure 6, the comments at the beginning of the script show that the real value stored at index 5 defines the radius of the circular arc segment used in defining the outer boundary, and the real value stored at index 6 defines the distance from the trailing edge of the airfoil to the exit plane. The use of these parameters allows the user to alter the distance to the outer boundary. The remaining parameter setting commands allow the grid to be redimensioned and allow grid point clustering to be altered as with EAGLE.

After the initial parameters setup, the shape of the symmetric airfoil is read in. The first and last points on this airfoil shape are then extracted to define points number 2 and 3. The locations of points 1, 5, and 6 are then calculated relative to the trailing edge of the airfoil. Note, the distance stored in real index 6 is used to define the offset of point 1 in the x direction, and the radius stored in real index 5 is used to define the offset of points 5 and 6 in the +y and −y directions, respectively. Without the use of vector expressions, points 1, 5, and 6 would have been defined explicitly.

The circular arc segment used in defining the outer boundary is generated at the beginning of Figure 7. Without the use of the generalized parameters, an explicited value would have been given for 'RADIUS'. With this key parameter buried in the interior of the script, the possibilities for producing errors, when altering the grid, are increased.

```
*
*  set radius of circular arc
*
E$INPUT ITEM='SETREAL', INDEX=    5, RVAL=30. $
*
*  set distance to downstream boundary
*
E$INPUT ITEM='SETREAL', INDEX=    6, RVAL=29. $
*
*  set appropriate integer values for use as numbers of points
*
E$INPUT ITEM='SETINT', INDEX=    9, IVAL=81 $
E$INPUT ITEM='SETINT', INDEX=    1, IVAL=!9+!9-1 $
E$INPUT ITEM='SETINT', INDEX=    2, IVAL=31 $
E$INPUT ITEM='SETINT', INDEX=    3, IVAL=20 $
E$INPUT ITEM='SETINT', INDEX=    7, IVAL=141 $
E$INPUT ITEM='SETINT', INDEX=    6, IVAL=41 $
E$INPUT ITEM='SETINT', INDEX=    5, IVAL=!2+!1+!2-2 $
E$INPUT ITEM='SETINT', INDEX=    8, IVAL=!5-!6-!7+2 $
*
*  set appropriate real values for use as spacings
*
E$INPUT ITEM='SETREAL', INDEX=    1, RVAL=.001 $
E$INPUT ITEM='SETREAL', INDEX=    2, RVAL=.01 $
E$INPUT ITEM='SETREAL', INDEX=    3, RVAL=.0001 $
E$INPUT ITEM='SETREAL', INDEX=    4, RVAL=.0015 $
*
*  read airfoil shape
*
E$INPUT ITEM='CURRENT', FILEIN=12, FORM='LIST', STYLE='XYZ', POINTS=100 $
*
*  extract points 2 and 3 from the airfoil definition
*
E$INPUT ITEM='GETVEC', POINT='FIRST', INDEX=    3,
   VECTYP='POINT' $
E$INPUT ITEM='GETVEC', POINT='LAST', INDEX=    2,
   VECTYP='POINT' $
*
*  calculate location of points 1, 5, and 6
*
E$INPUT ITEM='SETVEC', INDEX=    1, VVAL=~2+{#6,0.,0.} $
E$INPUT ITEM='SETVEC', INDEX=    5, VVAL=~1+{0.,#5,0.} $
E$INPUT ITEM='SETVEC', INDEX=    6, VVAL=~1-{0.,#5,0.} $
*
*  distribute points on the airfoil
*
E$INPUT ITEM='CURDIST', POINTS=!   9, DISTYP='BOTH', SPLTYP='QUAD',
   SPACE=#   4,#   2, RELATIV='NO','NO' $
E$INPUT ITEM='OUTPUT', COREOUT=    9 $
*
E$INPUT ITEM='SWITCH', REORDER='REVERSE1' $
E$INPUT ITEM='SCALE', SCALE={1.,-1.,1.} $
E$INPUT ITEM='INSERT', COREIN=    9 $
E$INPUT ITEM='OUTPUT', COREOUT=    1 $
```

Figure 6: IGGy script for C grid about symmetic airfoil.

```
*
*  generate circular arc segment
*
E$INPUT ITEM='CONICUR', TYPE='CIRCLE', POINTS=100, RADIUS=#   5,
   ANGLE=270.,90. $
E$INPUT ITEM='CURDIST', POINTS=!   7, DISTYP='BOTH', SPLTYP='QUAD',
   SPACE=#   3,#   3, RELATIV='YES','YES' $
E$INPUT ITEM='OUTPUT', COREOUT=    7 $
*
*  extract points 4 and 7 from ends of circular arc segment
*
E$INPUT ITEM='GETVEC', POINT='FIRST', INDEX=    7,
   VECTYP='POINT' $
E$INPUT ITEM='GETVEC', POINT='LAST', INDEX=    4,
   VECTYP='POINT' $
*
*  extract the spacing generated at the ends of the circular are segment
*
E$INPUT ITEM='GETSPA', END='FIRST', INDEX=    7 $
*
E$INPUT ITEM='LINE', POINTS=!   6, R1=~   7, R2=~   6 $
E$INPUT ITEM='CURDIST', POINTS=!   6, DISTYP='TANH', SPLTYP='LINEAR',
   SPACE=#   7, RELATIV='NO' $
E$INPUT ITEM='SWITCH', REORDER='REVERSE1' $
E$INPUT ITEM='OUTPUT', COREOUT=    6 $
*
E$INPUT ITEM='LINE', POINTS=10, R1=~   4, R2=~   5 $
E$INPUT ITEM='CURDIST', POINTS=!   8, DISTYP='TANH', SPLTYP='LINEAR',
   SPACE=#   7, RELATIV='NO' $
E$INPUT ITEM='OUTPUT', COREOUT=    8 $
*
E$INPUT ITEM='CURRENT', COREIN=    6 $
E$INPUT ITEM='INSERT', COREIN=    7 $
E$INPUT ITEM='INSERT', COREIN=    8 $
E$INPUT ITEM='OUTPUT', COREOUT=    5 $
*
E$INPUT ITEM='LINE', POINTS=10, R1=~   2, R2=~   1 $
E$INPUT ITEM='CURDIST', POINTS=!   2, DISTYP='TANH', SPLTYP='LINEAR',
   SPACE=#   2, RELATIV='NO' $
E$INPUT ITEM='SWITCH', REORDER='REVERSE1' $
E$INPUT ITEM='OUTPUT', COREOUT=    2 $
*
E$INPUT ITEM='LINE', POINTS=10, R1=~   1, R2=~   6 $
E$INPUT ITEM='CURDIST', POINTS=!   3, DISTYP='TANH', SPLTYP='LINEAR',
   SPACE=#   1, RELATIV='NO' $
E$INPUT ITEM='OUTPUT', COREOUT=    3 $
*
E$INPUT ITEM='LINE', POINTS=10, R1=~   1, R2=~   5 $
E$INPUT ITEM='CURDIST', POINTS=!   3, DISTYP='TANH', SPLTYP='LINEAR',
   SPACE=#   1, RELATIV='NO' $
E$INPUT ITEM='OUTPUT', COREOUT=    4 $
*
E$INPUT ITEM='COMBINE', FILEOUT=1, COREIN=1,2,3,4,5, FORM='E',
   STYLE='CONTENT' $
E$INPUT ITEM='END' $
```

Figure 7: Continuation of IGGy script above.

As seen in the 'CURDIST' statement following the 'CONICUR' statement, relative spacings are used to define the distribution of points along this circular arc segment. To match this spacing when generating neighboring segments, a 'GETSPA' command is used to extract the resulting absolute spacing at the ends of the segment and to store the spacing at real index 7. The remainder of the boundary segments are then generated using the defined parameters and are written to a file for importing to the grid code.

Multi-Store Interference Configuration

The following example addresses a multi-block grid about generic missile shapes in the mutual interference configurations whose frontal views are shown in Figure 8. Assuming a zero angle of attack and no interference from outside sources, the flow analysis can be performed on a small segment of the entire geometry. The dotted lines of Figure 8 can be drawn due to geometric symmetry; the dashed lines can be drawn due to the assumption of zero angle of attack. This reduces the flow field to a 90 degree wedge for the two store case and a 60 degree wedge for the three store case. With such a configuration, effects of separation distance and toe in or toe out angle can be investigated.

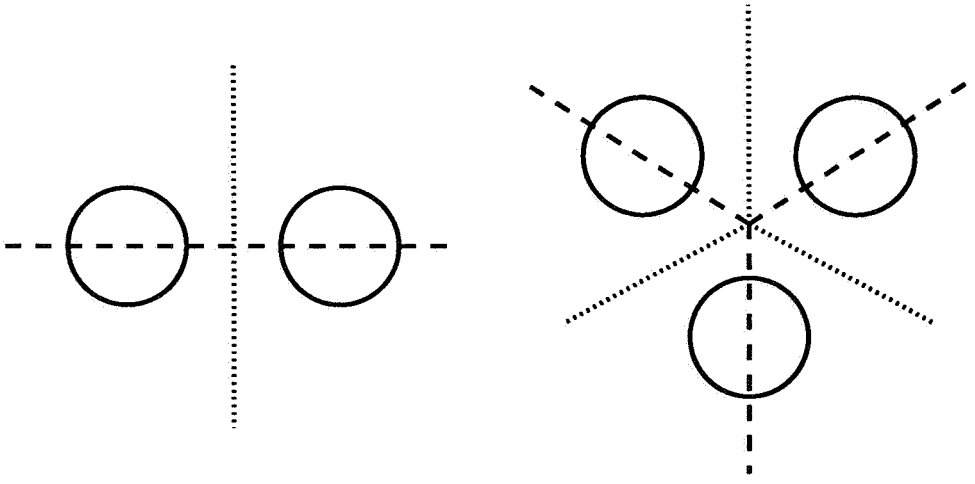


Figure 8: Two and three store configuration.

The script used to produce the boundary segments for this configuration is too long to be included in its entirety; however, Figure 9 shows the parameter definitions that are placed at the beginning of this script. The real value parameter stored at index 16 defines the angle between the planes of symmetry and thus allows the two and three store cases to be generated from the same script. As the comments imply, other real value parameters are used to control the location of the outer boundary, the separation distance between stores, and the toe in or toe out angle. No point locations are specified explicitly; rather, all necessary points are defined relative to the geometric definition of the store.

Figures 10 and 11 display perspective views of example boundary grids generated using IGGy. Figure 10 is the two store configuration; while, Figure 11 is the three store configuration. These grids consist of two blocks; and the outer boundaries are moved in close to the store for plotting

```
*
*  this runstream generates the boundary surfaces for a two block
*   grid about a ogive/cylinder/ogive/sting missile configuration.
*   two store or three store configurations are possible
*
*  length of the sting
*
E$INPUT ITEM='SETREAL', INDEX=   11, RVAL=60. $
*
*  length of the stagnation line
*
E$INPUT ITEM='SETREAL', INDEX=   12, RVAL=60. $
*
*  distance from reflection plane to centerline of missile
*
E$INPUT ITEM='SETREAL', INDEX=   13, RVAL=1.8 $
*
*  toe in or toe out angle.  toe in is positive. abs(angle) <= 5 deg.
*
E$INPUT ITEM='SETREAL', INDEX=   14, RVAL=0.*pi/180. $
*
*  distance from inner block to outer boundary
*
E$INPUT ITEM='SETREAL', INDEX=   15, RVAL=60. $
*
*  angle of rotation of vertical plane of symmetry.  this angle is
*   zero for the two store configuration and 30 deg for three stores.
*
E$INPUT ITEM='SETREAL', INDEX=   16, RVAL=0.*pi/180. $
*
*
*  set numbers of points along segments
*
E$INPUT ITEM='SETINT', INDEX=    1, IVAL=24 $       along the nose
E$INPUT ITEM='SETINT', INDEX=    2, IVAL=30 $       along the shaft
E$INPUT ITEM='SETINT', INDEX=    3, IVAL=15 $       along the tail
E$INPUT ITEM='SETINT', INDEX=    4, IVAL=23 $       along the sting
E$INPUT ITEM='SETINT', INDEX=    5, IVAL=15 $       1/3 around missile
E$INPUT ITEM='SETINT', INDEX=    6, IVAL=15 $       2/3 around missile
E$INPUT ITEM='SETINT', INDEX=    7, IVAL=!5+!6-1 $  total around missile
E$INPUT ITEM='SETINT', INDEX=    8, IVAL=32 $       normal to missile
E$INPUT ITEM='SETINT', INDEX=    9, IVAL=25 $       along stagnation line
E$INPUT ITEM='SETINT', INDEX=   10, IVAL=!9+!1+!2+!3+!4-4 $  total along x-axis
E$INPUT ITEM='SETINT', INDEX=   11, IVAL=20 $       j direction of outer blk
*
*  set spacings
*
E$INPUT ITEM='SETREAL', INDEX=    1, RVAL=.04 $    tip of nose
E$INPUT ITEM='SETREAL', INDEX=    2, RVAL=.08 $    base of nose
E$INPUT ITEM='SETREAL', INDEX=    3, RVAL=.08 $    base of shaft
E$INPUT ITEM='SETREAL', INDEX=    4, RVAL=.1 $     base of tail
E$INPUT ITEM='SETREAL', INDEX=    5, RVAL=.005 $   normal to missile
E$INPUT ITEM='SETREAL', INDEX=    6, RVAL=5. $     begn & end i dir outr bndry
E$INPUT ITEM='SETREAL', INDEX=    7, RVAL=.1 $     normal to v reflec pln
```

Figure 9: Parameter definitions for multi-store configuration.

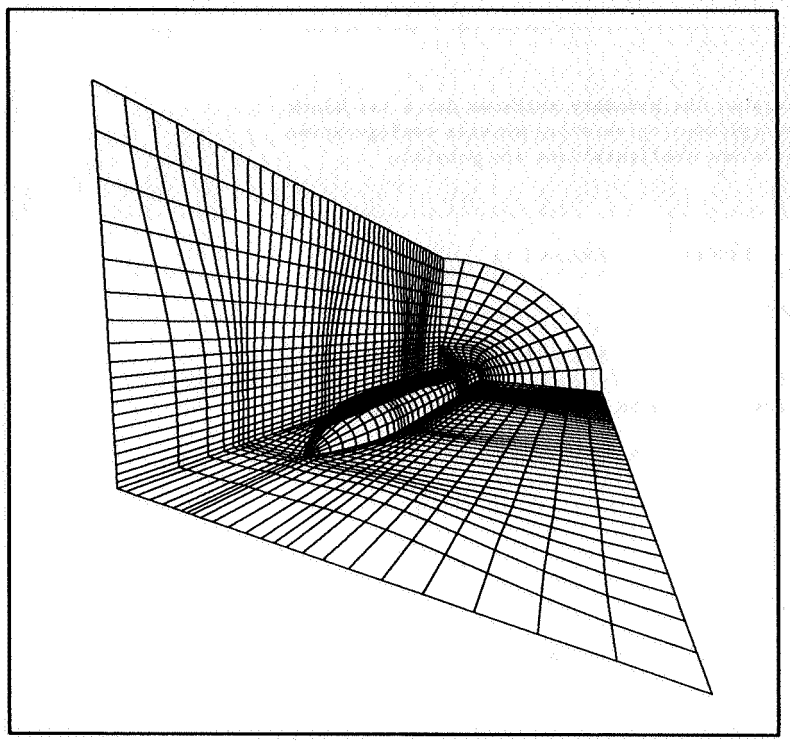


Figure 10: Example boundary grids for two store configuration.

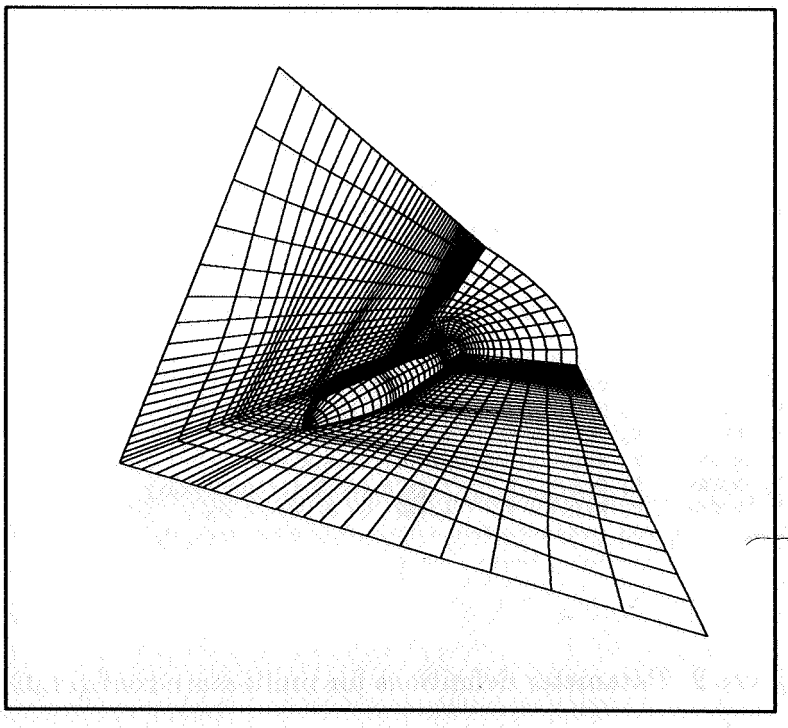


Figure 11: Example boundary grids for three store configuration.

purposes. Both of these grids were generated by altering only the parameters shown in Figure 9, with the only difference between the two being the value assigned to the real parameter stored at index 16.

CONCLUSIONS AND RECOMMENDATIONS

Many of the ideas relating to the generation of the needed parser and the syntax of the scalar and vector expressions were taken from SDL [8], a general purpose language for surface grid generation. Whereas EAGLE provides a user-friendly command structure with limited flexibility, and SDL provides the complete flexibility of a programming language (including looping constructs, conditional statements, and subroutines), IGGy has sought a compromise in flexibility and power while retaining the user-friendly command structure and providing a user-friendly graphical environment.

The recent trend in extensions to the EAGLE code has involved the generation of new commands which are built around the capabilities of the original commands. These new commands essentially automate some function which would have previously taken several commands to accomplish. This trend goes against the philosophy taken in IGGy. Rather, it would be preferable to produce a macro language which would allow the user to produce such capabilities from the commands which already exist. Such a macro language would not produce the flexibility of a programming language but would give the user the added flexibility to tailor the code to a particular application.

REFERENCES

1. Lijewski, L. E., Cipolla, J., et. al., "Program EAGLE User's Manual Volume I - Introduction and Grid Applications", AFATL-TR-88-117, September 1988.

2. Thompson, J. F. and Gatlin, B., "Program EAGLE User's Manual Volume II - Surface Generation Code", AFATL-TR-88-117, September 1988.

3. Thompson, J. F. and Gatlin, B., "Program EAGLE User's Manual Volume III - Grid Generation Code", AFATL-TR-88-117, September 1988.

4. Eiseman, Peter R., "Applications of Algebraic Grid Generation", AGARD Fluid Dynamics Panel Specialists' Meeting on Applications of Mesh Generation to Complex 3-D Configurations, Loen, Norway, May 1989.

5. Thompson, J. F., Lijewski, L. E., and Gatlin, B., "Efficient Applications Techniques of the EAGLE Grid Code to Complex Missle Configurations", AIAA-89-0361, 27th Aerospace Sciences Meeting, Reno, Nevada, January 1989.

6. Johnson, S. C., "Yacc: Yet Another Compiler Compiler", Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, New Jersey, 1975.

7. Lesk, M. E., "Lex - A Lexical Analysis Generator", Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, New Jersey, October 1975.

8. Maple, Raymond C., "SDL - A Surface Description Language", NASA Workshop on Software Systems for Surface Modeling and Grid Generation, NASA CP- 3143 , 1992.

Algebraic Surface Grid Generation in Three-Dimensional Space

Saif Warsi
Sverdrup Technology, Inc.
Structures and Dynamics Dept.
Huntsville, Alabama 35806

SUMMARY

An interactive program for algebraic generation of structured surface grids in three-dimensional space has been developed on the IRIS4D series workstations. Interactive tools are available to ease construction of edge curves and surfaces in three-dimensional space. Addition, removal, or redistribution of points at arbitrary locations on a general three-dimensional surface or curve is possible. Additionally, redistribution of surface grid points may be accomplished through use of conventional surface splines or a method called here as "surface-constrained transfinite interpolation". This method allows the user to redistribute the grid points on the edges of a surface patch; the effect of the redistribution is then propagated to the remainder of the surface through a transfinite interpolation procedure where the grid points will be constrained to lie on the surface. The program has been written to be highly functional and easy to use. A host of utilities are available to ease the grid generation process. Generality of the program allows the creation of single and multi-zonal surface grids according to the user requirements. The program communicates with the user through popup menus, windows, and the mouse. General picking mechanisms have been implemented so the user does not have to keep track of curves or surfaces through any sort of identification mechanism external to the program. Further, this program has been written in the C language to allow use of dynamic memory allocation so that the required memory during grid generation grows as needed.

INTRODUCTION

The growing capabilities of modern computers are driving engineers to apply their algorithms to increasingly more complex geometries. As a consequence, traditional grid generation procedures using batch grid generation and plotting programs are no longer suitable tools to provide acceptable meshes in a reasonable time. For any highly complex geometry, generation of acceptable surface grids may consume more time than obtaining the flow solution itself.

To meet the demands of obtaining reasonable surface grids in a timely manner, an interactive program using the graphical capabilities of todays high speed engineering workstations has been developed. The interactive approach is reliable and timely because it allows the user to correct errors without delay. Due to the interactive nature of the program, the results of

any operations performed on curves and surfaces are immediately visualized and immediate steps may be taken to correct the final results.

The variety of different configurations which may be considered may range from simple airfoils to complex three-dimensional geometries of a whole aircraft, nozzles, inlets, etc. Therefore the grid generation techniques have been designed to be flexible, general and easy to use. To meet the above requirements this program allows arbitrary geometries to be prepared, generated, and refined interactively through a user interface consisting of popup-menus and windows; generally the user has the use of the entire graphics window. General algorithms and data structures have been implemented to ease the grid generation process. The program has been written to be an "easy to use" tool for constructing curves and surfaces without having to be a grid generation "expert".

EDGE CURVE GENERATION

The program has extensive edge curve generation and editing capabilities. Edge curve generation in the current program may be accomplished through extraction of interactively specified grid lines from an existing surface, interactively designed straight lines, circular and elliptical arc segments or three-dimensional bézier curves. The circular and elliptical arc segments may lie in any arbitrary three-dimensional plane. End points for any curve may be either interactively specified, input or picked from any existing surface or curve. To aid endpoint selection, cursor movement to arbitrary points in three-dimensional space is easily done without the user having to do complex rotations or translations. Additional curves may be generated by scaling, translating, appending to, rotating any existing curve or splitting existing curves into multiple segments as specified by the user. The rotation of curves may be done about any user-specified axis.

The bézier curves [1] are an especially useful edge curve generator tool since a wide variety of curves may be easily designed.

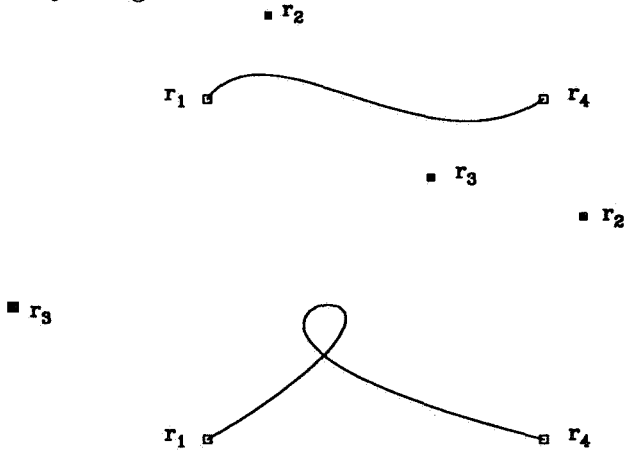


Figure 1. Control point location influence on bézier curves.

Slopes at the end points and overall curve shape are entirely under user control. The bézier curve may be written in matrix notation as

$$\mathbf{r}(u) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r_1} \\ \mathbf{r_2} \\ \mathbf{r_3} \\ \mathbf{r_4} \end{bmatrix} \tag{1}$$

where u is a parameter with $0 < u < 1$ and the $\mathbf{r}$'s are the coordinate vectors of control points defining the curve. Referring to Figure 1, $(\mathbf{r}_1, \mathbf{r}_4)$ are the endpoints of the curve and $(\mathbf{r}_2, \mathbf{r}_3)$ are the control points whose locations guide the shape of the curve. The ($\mathbf{r}_2$ and $\mathbf{r}_3$) control points are entirely under user control and may be moved to any location in three-dimensional space to obtain the desired curve. As figure 1 shows, the influence of the location of the control points allows a variety of curves to be easily generated.

To ease construction of edge curves for complex three-dimensional geometries, a bézier curve with either endpoint $\mathbf{r}_1$ or endpoint $\mathbf{r}_4$ being on a surface will have its control points located in three-dimensional space such that the initial curve will be orthogonal to the surface at that endpoint.

Curve editing facilities include redistribution and addition/deletion of points along a given user specified segment of a curve. These procedures use parametrized cubic splines to either redistribute or add/delete points along the specified segment. Redistribution of grid points uses the Vinokur [2] hyperbolic stretching function; the desired spacing at either end of the specified segment may be either input by the user or interactively adjusted to user requirements. As there are no restrictions in this method, any curve can be given the proper number of points with desired distribution characteristics. Curves which have a discontinuity can be handled easily by placement of an endpoint of the chosen segment at the discontinuity such that spline construction and evaluation do not occur across the discontinuity itself.

SURFACE GRID GENERATION

The surface generation and editing package in the program is also quite general. Initial surfaces may be read from a file or created within the program. Surfaces may be created using the following methods:

1. Arc-Length based transfinite interpolation given the edge curves (one collapsed edge is possible).
2. Surface of revolution by rotating a given curve about any user-specified axis.
3. Automatic splining of a given set of cross sections.
4. Rotation of a surface about any user-specified axis.
5. Scaling, translating and mirroring a surface.
6. Appending surfaces to create a new surface.

To create a surface given its edge curves, the transfinite interpolation (TFI) method of Soni [3] is used to calculate an initial interior surface grid. The algebraic grid generated via TFI may require smoothing to remove discontinuities; future plans include implementing the interactive elliptic smoothing package of Kania [4]. This grid generator solves the most general elliptic equations for surfaces (see ref. [5]) using a line successive over-relaxation (LSOR) method and maintains the surface curvature.

A surface of revolution may be easily generated by revolving a given curve about any arbitrary user-specified axis. The user simply specifies the number of points in the rotation direction and either inputs or interactively specifies the rotation angle. A surface may also be easily created by splining a set of specified cross sections. The user picks the desired cross sections and specifies the number of points through the sections. The code will automatically place a uniformally distributed surface through the sections, however, grid lines on the generated surface will not neccessarily lie exactly on the original cross sections except at the first and last specified cross sections.

The surface editing package has similar features as the curve editing package. Addition or deletion of grid points may be performed easily by interactively drawing a patch on a given surface (the patch may be as small as one cell wide) and specifying the number of points to add or remove. Figure 2 shows a generic forebody surface grid of an aircraft; figure 3 shows a case where the number of points was increased in the canopy region. Discontinuites in a surface, as for a curve, may be handled easily by selecting appropriate patches so that spline constructions and evaluations do not occur at the discontinuites.

Redistribution of grid points on a surface may be performed in two modes. The first mode allows the user to specify any portion of a constant ξ or η line on a given surface. The grid points may then be redistributed using parametric surface splines. For example, redistributing a portion of any constant ξ line will cause all grid points in the specified range of η on each ξ constant line to also be redistributed; similarly a constant η line may be selected along which to redistribute. Returning to the forebody surface grid, let the ξ coordinate be along the body and let η be the transverse coordinate. Figure 4 shows an application of this mode; notice that the grid points have been redistributed to the top of the canopy in the η direction and near the end of the body and from the nose to just ahead of the canopy in the ξ direction.

The second mode allows the user to choose a portion of a surface as a surface patch and a new system of coordinates (ξ_1,η_1) may then be generated which will be constrained to line on the original surface. This mode allows the user to make local grid modifications without disturbing the grid distributions elsewhere on the grid. Choosing a portion of a surface as a patch, the edges of the patch (both the ξ constant and η constant) may then be redistributed. To generate the grid in the interior of the patch based upon the new ξ_1 and η_1 coordinates on the edges, a standard transfinite interpolation may be applied if the surface is planar or constraint of the new coordinates to the previous ξ and η system is not required. However, if constraining the new coordinates to lie on the previous ξ and η system is required, a new method, called here "surface-constrained transfinite interpolation" (SCTI) may be applied.

Description of the SCTI method

The SCTI method incorporates a bicubic spline interpolating technique. Referring to figure 5, the method first constructs and stores parametric cubic splines $C_\xi(s_\xi)$ along each η constant line of the patch based on the original, unmodified surface (Basis Grid). The C indicates the cubic spline and s is the normalized arclength which is used as the parameter, in the subscripted (ξ) direction. A transfinite interpolation procedure is then applied

$$TFI \Rightarrow s_{\xi_1}(\xi_1, \eta_1) \text{ and } s_{\eta_1}(\xi_1, \eta_1),$$

where $s_{(\ )}$ indicates unit arclength distributions based on the new ξ_1 and η_1 edge coordinates. Using the $s_{\xi_1}(\xi_1, \eta_1)$ distribution the new coordinates are evaluated from the splines $C_\xi(s_\xi)$. A series of one-dimensional parametric splines C_1 are then constructed based on these new coordinates and the $s_{\eta_1}(\xi_1, \eta_1)$ distribution is used to evaluate the C_1 splines for the final surface constrained coordinates (SCTI Grid). The SCTI method implemented in the program can also handle singular patch edges.

The SCTI method discussed here is equivalent to the differential method discussed in [6]. In ref. [6] two partial differential equations (PDE's) have to be solved in which the dependent variables are the new coordinates (ξ_1,η_1) and the independent variables (ξ,η) are the old coordinates. It seems that the present algebraic approach is much faster and computationally simpler than the PDE approach.

Figure 6 shows a rather extreme example of SCTI applied to the forebody surface grid. A patch was selected and the grid point distributions on its edges were modified such that extreme coordinate concentration would occur near the corners; the SCTI method was then used to generate the new coordinates. Notice that the new coordinates are constrained to the original surface and that the influence of the edge redistributions has been propagated to the remainder of the surface by selecting appropriate surface patches and then applying SCTI.

DATA STRUCTURES

Curve or surface data, read in from a file or constructed within the program, is stored in structures consisting of one-dimensional arrays. All storage required is dynamically allocated, reallocated or freed as needed; this feature allows users to have various size curves and surfaces in memory simultaneously. The spline, transfinite interpolation, and various other routines require temporary storage to complete their operations; this temporary storage is also allocated and then freed upon completion of the required operation.

Dynamic memory management alleviates the user from having to stop and recompile the program simply because the amount of memory allocated at compile time may not have been sufficient to complete the required grid(s). The program will warn users and suggest steps to continue the grid generation process if the user should exceed the memory limitations of the machine.

APPLICATIONS

Figure 7 shows a set of curves generated for a generic re-entry vehicle as an example of curve generation and editing facilities. Since actual surface definition data was not available, each of the curves shown was generated with the curve segment generator in the program. The majority of the curves were generated using the bézier curve generator however, the complex curves at the trailing edge of the wing were generated by appending multiple bézier curves, elliptical, circular and straight line segments.

Figure 8 shows the initial surface grid generated for the generic re-entry vehicle using the previously designed curves shown in figure 7 and the surface generation facilities of splining cross-sectional data and transfinite interpolation with specified edge curves. Figure 9 shows the final surface grid for the generic re-entry vehicle after using the surface editing facilities. Notice that grid distributions are now much smoother and point resolution in areas of interest is better, while the original surface geometry is maintained.

Figure 10 shows a sample far field boundary and blocking arrangement for the re-entry vehicle after performing a domain decomposition. Figure 11 shows the mesh on selected block faces around the re-entry vehicle (coarsened grids are shown for clarity).

Figure 12 shows a global view of the surface grids generated for a wing/pylon/load configuration. Figure 13 shows some detail of the surface grids in the wing/pylon intersection region. Once again, all of the initial and final curves and surfaces were generated entirely within the program.

The interactive approach to surface and edge generation, as implemented in this program, is very efficient. For instance, generation of the initial curves for the re-entry vechicle was the most time consuming portion of the grid generation process since vehicle surface definition data was not available. The process of generating the initial curves was iterative so that a "proper" vehicle could be constructed; this process consumed approximately five working days on a part-time basis. The creation of the initial surfaces given the curves required approximately ten minutes. Editing the initial surfaces, domain decomposition and generation of the outer boundary blocking structure and surfaces took approximately two hours, since this procedure was also iterative.

CONCLUSIONS

An interactive algebraic grid generator has been developed which is capable of generating the surfaces for general three-dimensional geometries. The grid generator allows users to easily decompose complex domains and create any three-dimensional edge curve. Various utilities have been developed to ease coordinate generation on both curves and surfaces. The functionality, versatility, and efficiency of the interactive approach to surface generation has been demonstrated.

FUTURE WORK

Earlier, an interactive code for single block, planar grids, called GEN2D [*] was released. A very useful feature in GEN2D is the ability of the user to reshape any portion of any grid line via bézier curves. The effect of the reshaped grid line can then be propagated by TFI to the remainder of the grid. Since obtaining a desired grid using elliptic smoothing is quite difficult in highly complex or very coarse regions, this method is a quick and cheap way to obtain desired grids even in highly complex and/or coarse regions. A similar feature is now being planned for the current program. Therefore, a method to reshape grid lines on general three-dimensional surfaces using "surface constrained bézier curves" is being formulated. With the use of the SCTI method (or a variant thereof), the effect of the reshaped grid line(s) may be introduced into the remainder of the domain.

As mentioned previously, the interactive elliptic smoothing package of ref. [4] is also to be implemented into the program after re-coding the package from FORTRAN to C.

REFERENCES

[1] I.D. Faux and M.J. Pratt. *Computational Geometry for Design and Manufacture*, Ellis Horwood, 1979.

[2] Vinokur, M., "On One-Dimensional Stretching Functions for Finite-Difference Calculations", NASA-CR-3313, October 1980.

[3] Soni, B.K., "Two- and Three-Dimensional Grid Generation for Internal Flow Application of Computational Fluid Dynamics", AIAA Paper No. 85-1526, 1985.

[4] Kania, L., "Elliptic Surface Grid Generation in Three-Dimensional Space", In *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*,NASA CP-3143, April 1992.

[5] Warsi, Z.U.A., "Numerical Grid Generation in Arbitrary Surfaces through a Second-Order Differential-Geometric Model", *Journal of Computational Physics*, Vol. 64 No. 1, May 1986.

[6] Warsi, Z.U.A., "Theoretical Foundation of the Equations for the Generation of Surface Coordinates", *AIAA Journal*, Vol. 28 No. 6, June 1990.

[*] Warsi, S.A., "User's Guide to GEN2D - An Interactive Program for Generating Simply Connected Grids", Sverdrup Technology, Huntsville, AL. 1990, (unpublished).

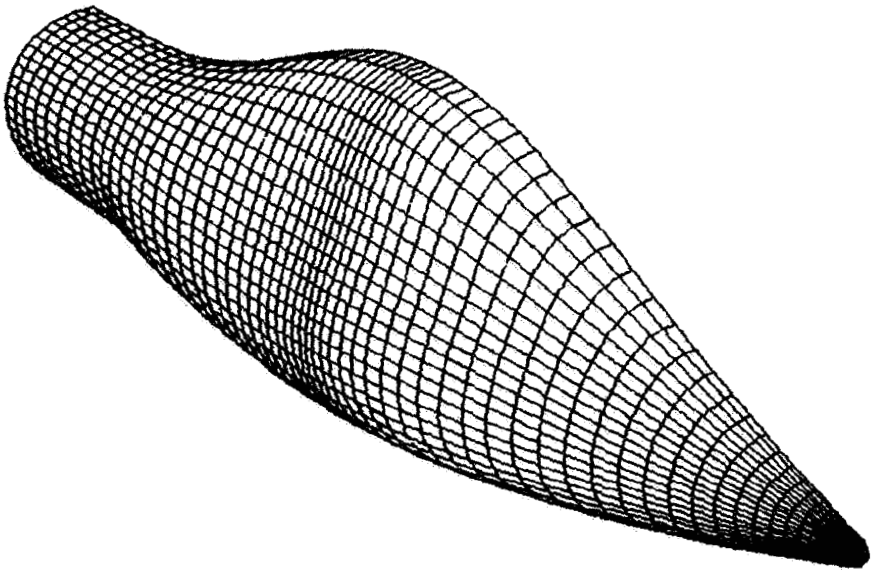


Figure 2. Generic forebody surface grid.

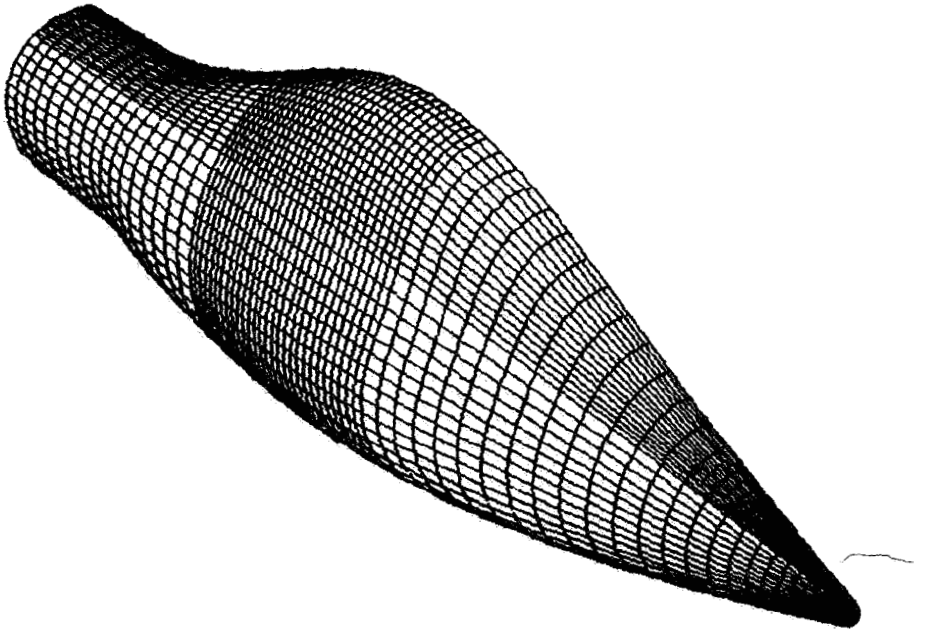


Figure 3. Enrichment of grid points in canopy region of forebody surface.

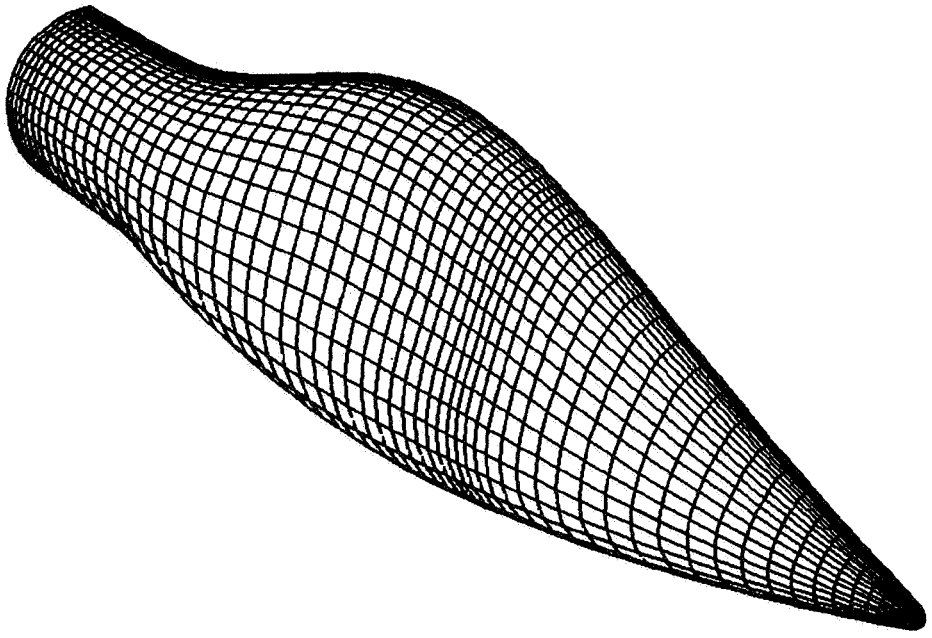


Figure 4. Redistribution of grid points on forebody surface using conventional splines.

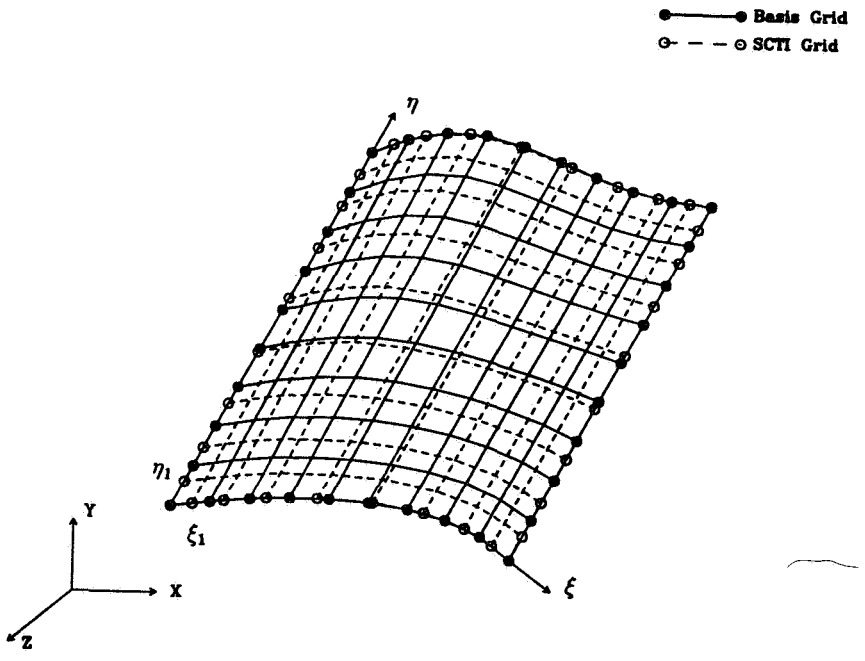


Figure 5. Concept of the SCTI method

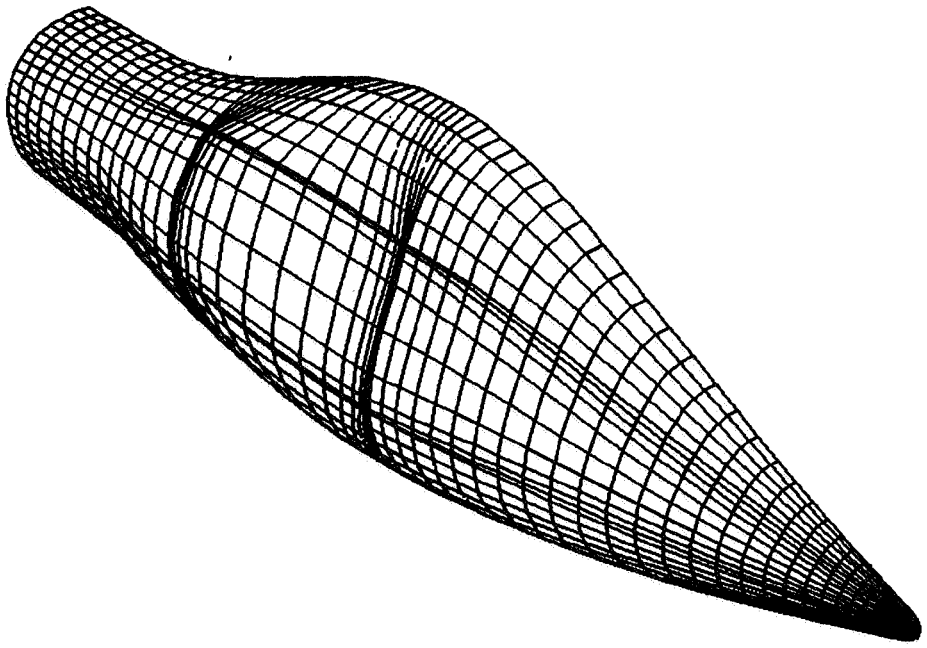


Figure 6. Application of SCTI method to forebody surface grid

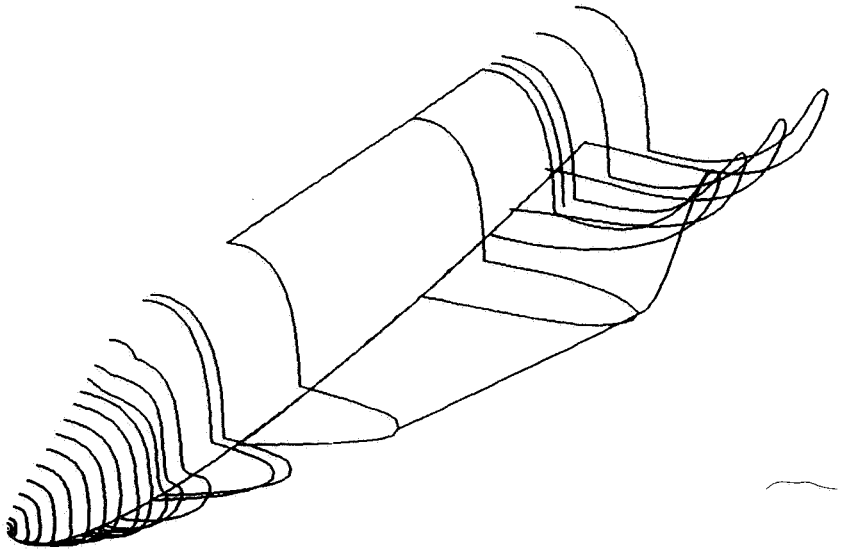


Figure 7. Surface definition curves for generic re-entry vehicle.

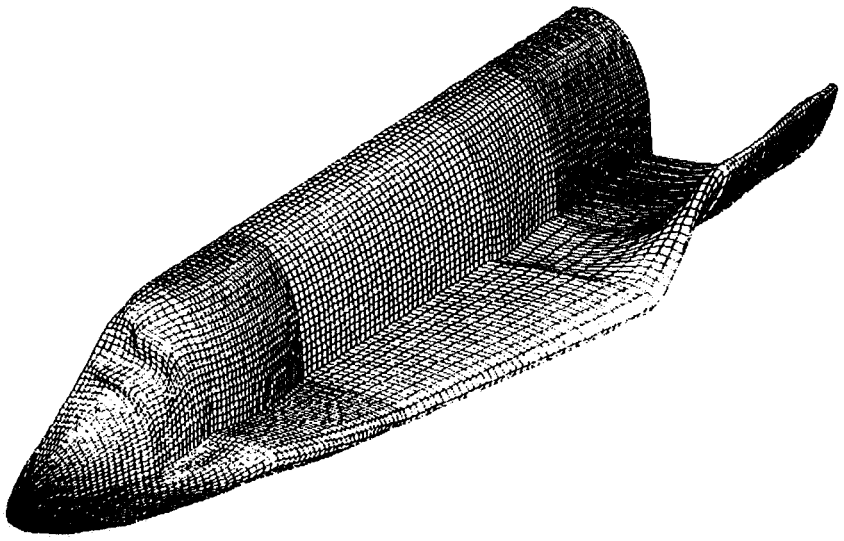


Figure 8. Initial surface grid for generic re-entry vehicle.

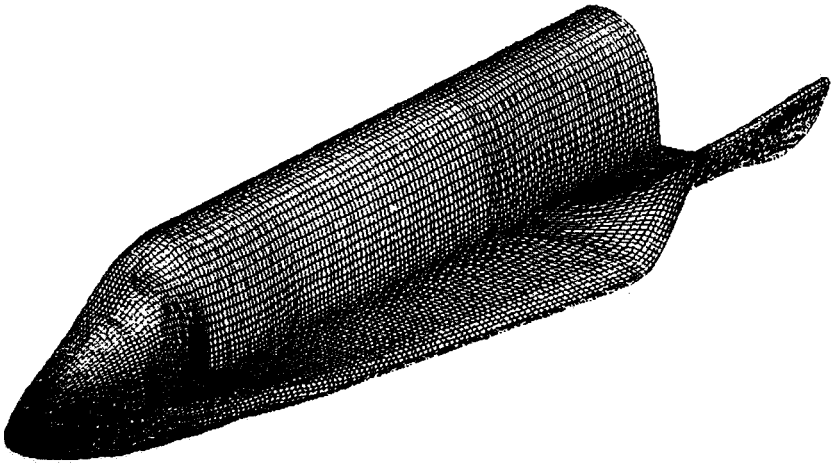


Figure 9. Final surface grid for generic re-entry vehicle.

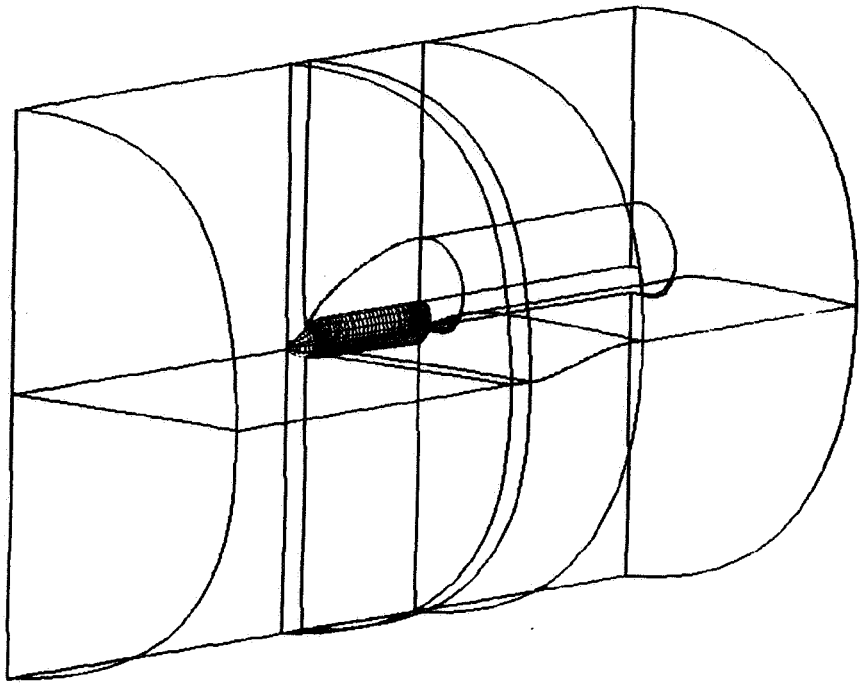


Figure 10. Example farfield and blocking arrangement for re-entry vehicle.

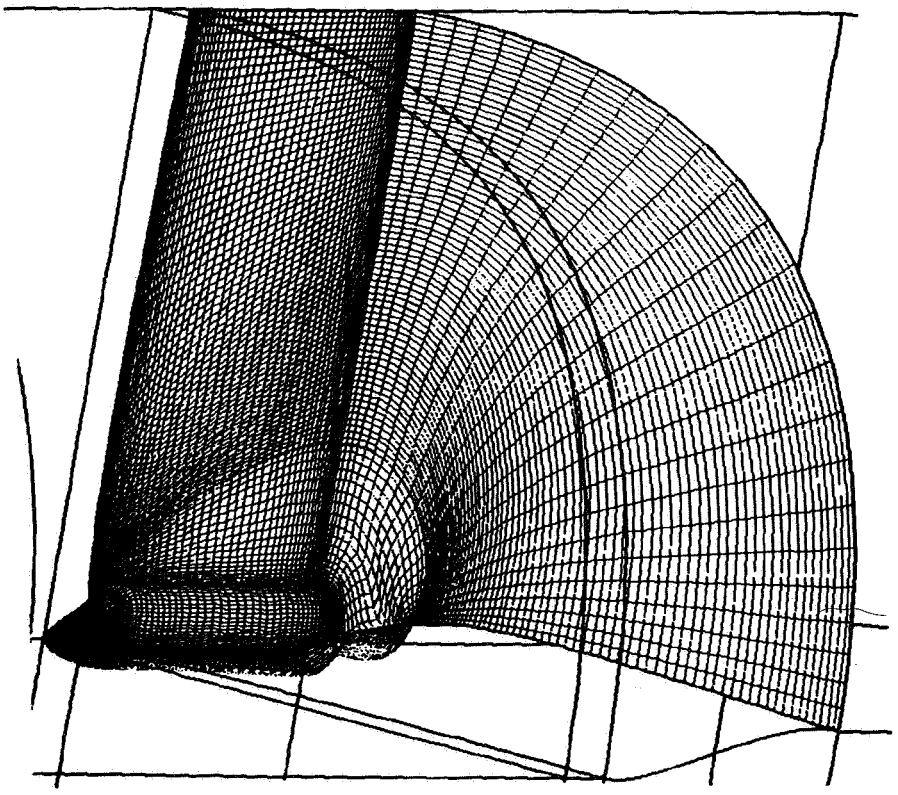


Figure 11. Grids on block faces around re-entry vehicle.

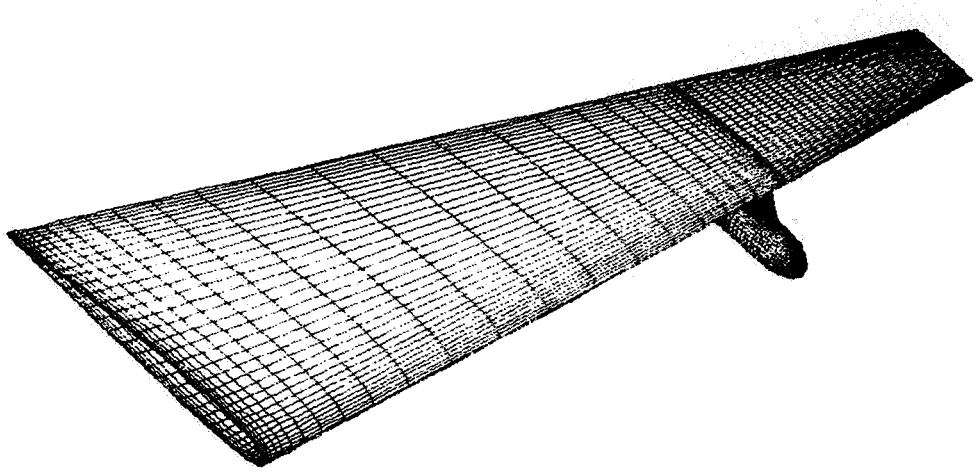


Figure 12. Surface grids for wing/pylon/load configuration.

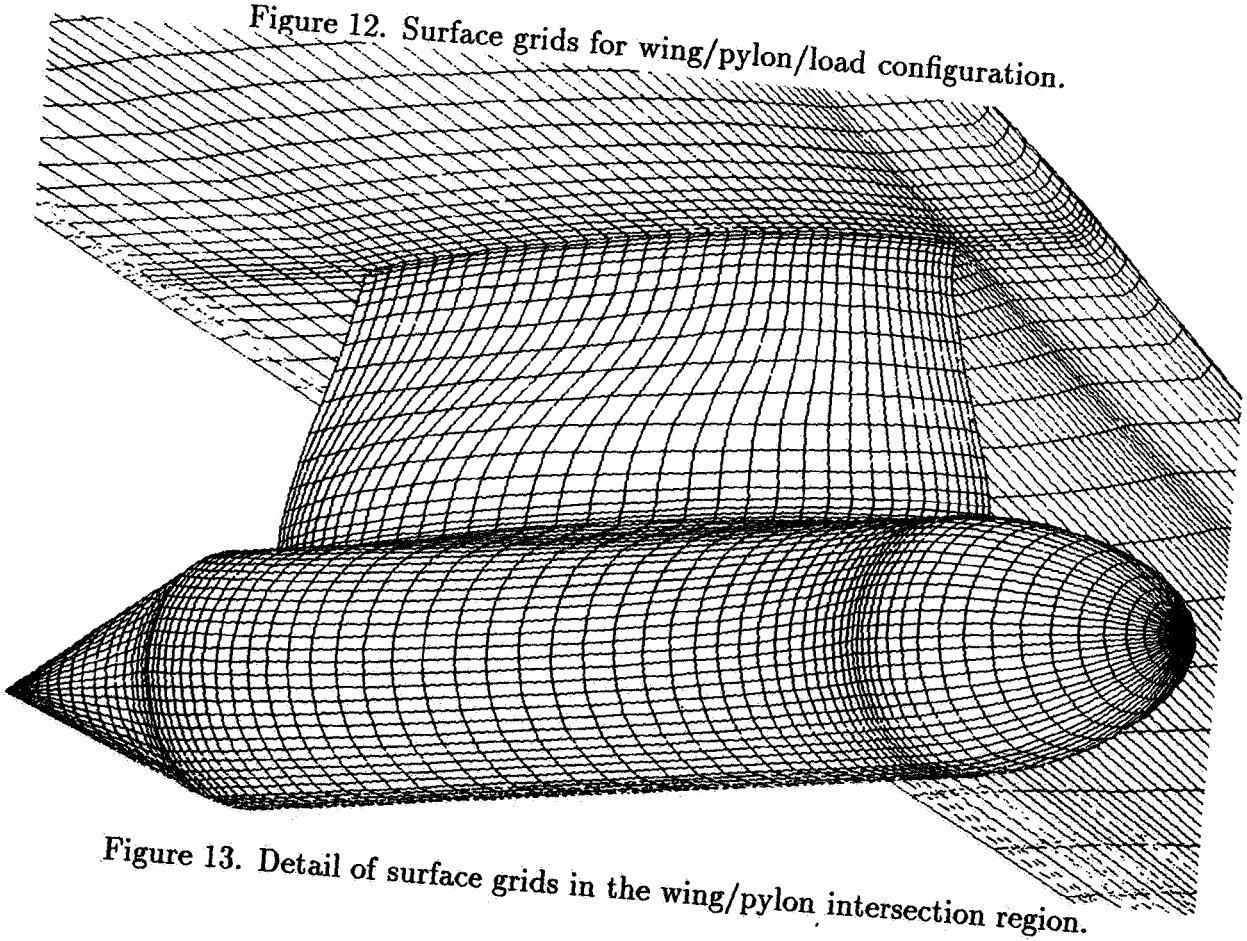


Figure 13. Detail of surface grids in the wing/pylon intersection region.

ALGEBRAIC SURFACE DESIGN AND FINITE ELEMENT MESHES

Chandrajit L. Bajaj *
Department of Computer Science,
Purdue University,
West Lafayette, IN 47907

INTRODUCTION

This paper summarizes some of the techniques used in constructing C^0 and C^1 continuous meshes of low degree, implicitly defined, algebraic surface patches in three dimensional real space $\mathbb{R}^3$. These meshes of low degree algebraic surface patches are used to construct accurate computer models of physical objects. These meshes are also utilized in the finite element simulation of physical phenomena (e.g. heat dissipation, stress/strain distributions, fluid flow characteristics, etc.) required in the computer prototyping of both the manufacturability and functionality of the geometric design.

Why use algebraic surfaces ? A real algebraic surface S in $\mathbb{R}^3$ is implicitly defined by a single polynomial equation $\mathcal{F} : f(x, y, z) = 0$, where coefficients of f are over the real numbers $\mathbb{R}$. Manipulating polynomials, as opposed to arbitrary analytic functions, is computationally more efficient. Furthermore algebraic surfaces provide enough generality to accurately model most complicated rigid objects.

Why use implicit representations ? While all real algebraic surfaces have an implicit definition $\mathcal{F}$ only a small subset of these real surfaces can also be defined parametrically by the triple $\mathcal{G}(s,t) : (x = G_1(s,t), y = G_2(s,t), z = G_3(s,t))$ where each G_i, $i = 1,2,3$, is a rational function (ratio of polynomials) in s and t over $\mathbb{R}$. The primary advantage of the implicit definition $\mathcal{F}$ is the closure properties of the class of algebraic surfaces under modeling operations such as intersection, convolution, offset, blending, etc. The strictly smaller class of parametrically defined algebraic surfaces $\mathcal{G}(s,t)$ are not closed under any of the operations listed before. Closure under modeling operations allows cascading repetitions without any need of approximation. Furthermore, designing with a larger class of surfaces leads to better possibilities of being able to satisfy the same geometric design constraints with much lower degree algebraic surfaces. The implicit representation of algebraic surfaces also naturally yields sign-invariant regions $\mathcal{F}^+ : f(x, y, z) \geq 0$ and $\mathcal{F}^- : f(x, y, z) \leq 0$, a fact quite useful for intersection and offset modeling operations. One aim of this paper is to exhibit that implicitly defined algebraic surfaces are very appropriate for constructing curved finite element meshes.

*Supported in part by NSF grant CCR 90-02228, NSF grant DMS 91-01424 and AFOSR contract 91-0276

C^0 PLANAR MESHES ON RATIONAL PARAMETRIC SURFACES

It would seem that rational parametric surfaces are easy to mesh since it is trivial to generate points on a parametric surface. However in constructing a planar finite element mesh on the entire surface, two difficulties arise:

1. To mesh the entire surface one must allow the parameters to somehow range over the entire parametric domain, which is infinite. Any bounded portion of the parametric domain leaves a hole in the surface.

2. Even when restricting the surface to a bounded part of the parametric domain, the rational functions describing the surface may have poles over that domain, where the surface will become discontinous. If a surface is discontinuous, an arbitrary triangulation of the parametric domain will usually result in an incorrect triangulation of the the parametric surface.

In [13]* we give solutions to the above problems for the C^0 meshing of rational parametric curves, surfaces and hypersurfaces of any dimension [8]. The technique is based on homogeneous linear (projective) reparameterizations and yields a complete and accurate C^0 planar mesh of free-form, discontinuous rational parametric domains. For the CARTAN surface example, a single reparameterization $[x = s * t, y = s^2, z = t]$ removes the pole $t = 0$ of the original parameterization.

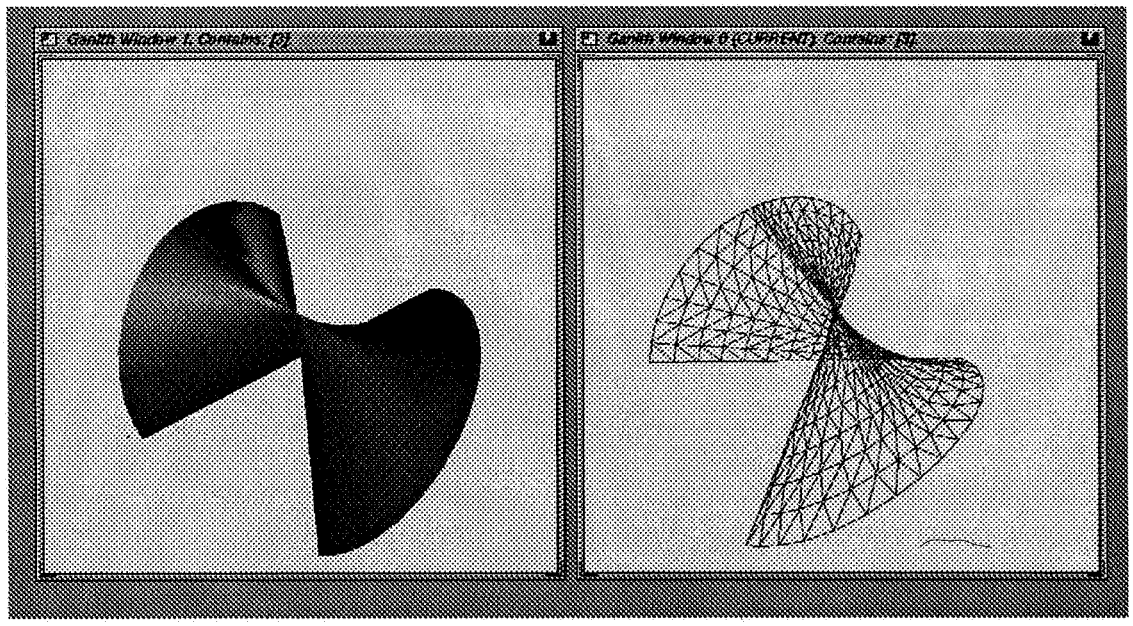


Figure 1: A finite element mesh on the CARTAN umbrella surface $[x = s, y = \frac{s^2}{t^2}, z = t]$

*This paper contains references 1-37.

C^0 PLANAR MESHES ON RATIONAL PARAMETRIC SURFACES, (Contd.)

For the STEINER surface example, four different projective reparameterizations yield a complete covering of the rational parametric surface. In [13] for surfaces, four reparameterizations always suffice. In general 2^d projective reparameterizations suffice for a d dimensional parametric hypersurface. The algorithms which compute these reparameterizations as well as generate the C^0 planar meshes have been implemented in C in our GANITH toolkit[12]. The pictures have been generated using this toolkit.

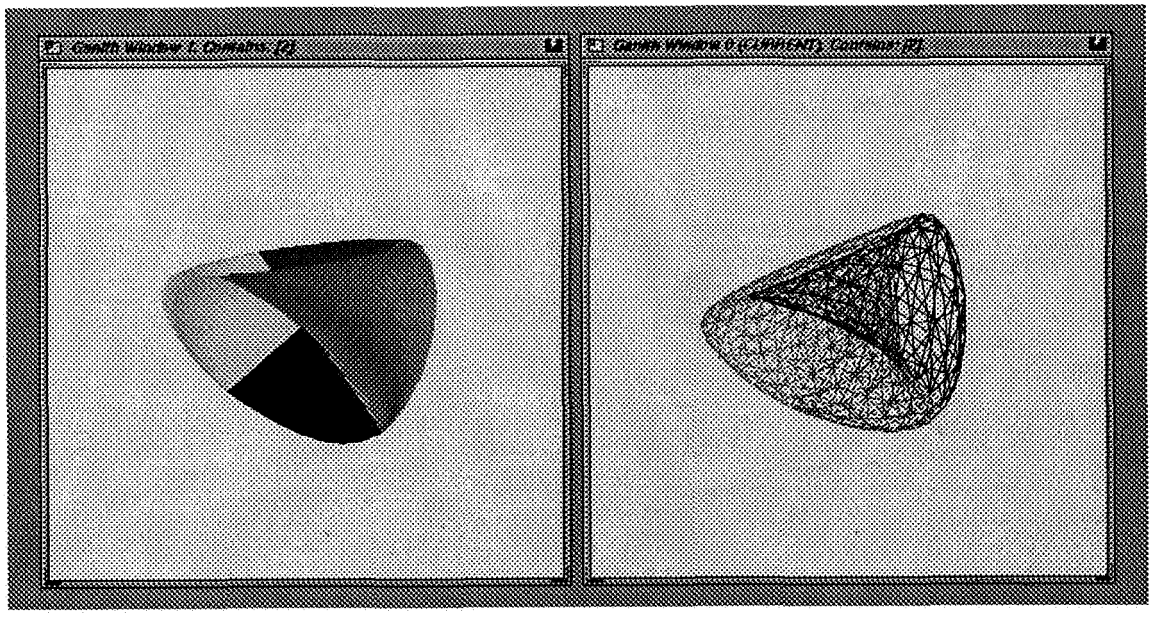


Figure 2: A finite element mesh on the STEINER surface $[x = \frac{2*s*t}{1+s^2+t^2}, y = \frac{2*s}{1+s^2+t^2}, z = \frac{2*t}{1+s^2+t^2}]$

C^0 PLANAR MESHES ON IMPLICIT ALGEBRAIC SURFACES

One way to mesh implicitly defined rational algebraic surfaces of low degree is to symbolically compute global rational parametric equations of the surface [1, 2, 8], and then use the meshing methods of the previous section. Direct schemes which work for arbitrary implicit algebraic surfaces are based on either the regular subdivision of the cube [16] or a finite subdivision of an enclosing tetrahedron [29]. The quartic surface example below was meshed in GANITH [12] by a implementation based on the methods of [16, 29].

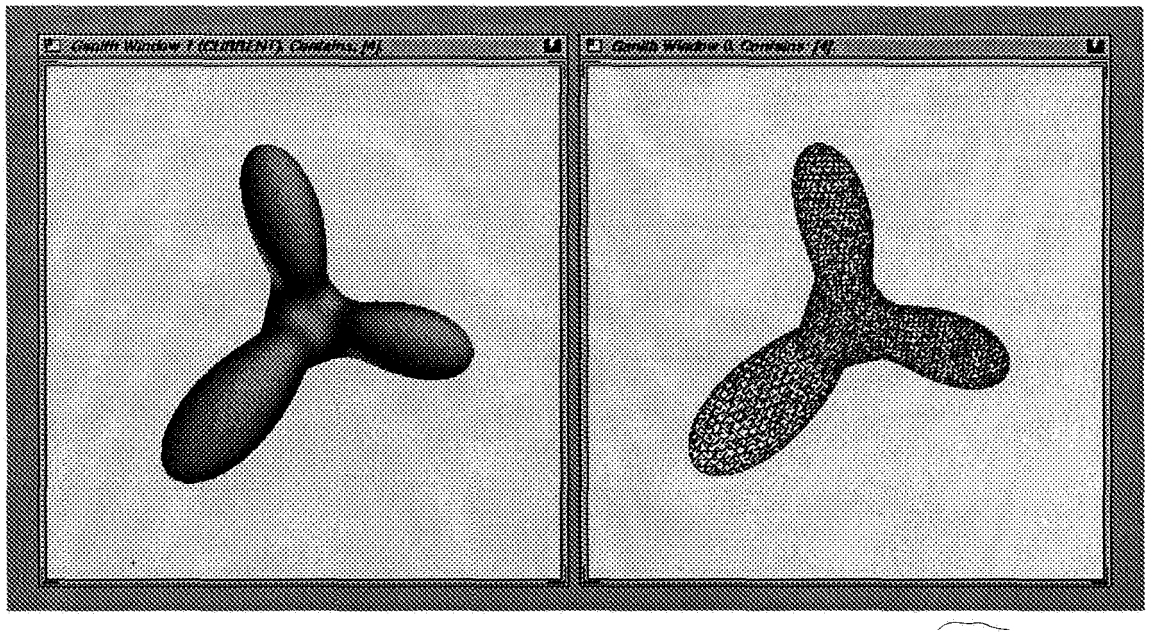


Figure 3: A finite element mesh of an implicitly defined quartic surface

C^0 PLANAR MESHES ON IMPLICIT ALGEBRAIC SURFACE PATCHES, (Contd.)

The implicitly defined triangular quintic surface patch example below was meshed in GANITH [12] by an implementation based on a space curve tracing method reported in [10]. In this method, various curves joining the boundary of the triangular patch are adaptively traced to yield a mesh of triangles which accurately C^0 approximate the triangular surface region.

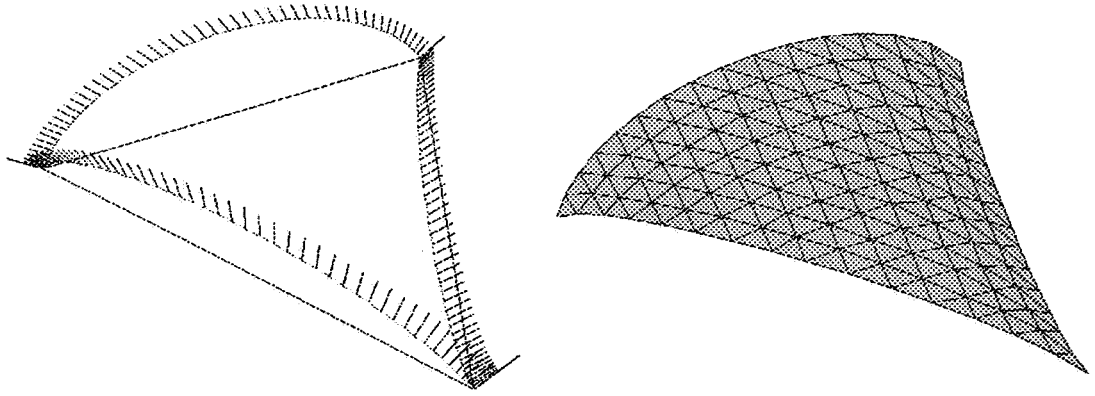


Figure 4: A finite element mesh of implicit triangular quintic patches used in C^1 smoothing polyhedra.

BOUNDED ASPECT RATIO TRIANGULATIONS IN THREE DIMENSIONS

Though a number of algorithms exist for triangulating a point set or a polyhedron in two and three dimensions [17, 23, 31], few address the problem of guaranteeing the shape of the triangular elements. To reduce ill-conditioning as well as discretization error, finite element methods require triangular meshes of bounded aspect ratio [5, 25]. By aspect ratio of triangles or tetrahedra, one may consider the ratio of the radii of the circumscribing circle to that of inscribing circle (spheres in case of tetrahedra).In 2D, two distinct approaches are known to produce guaranteed quality triangulations. The first approach, based on *Constrained Delaunay Triangulations*, was suggested by Chew [18]. He guarantees that all triangles produced in the final triangulation have angles between 30° and 120°. The other approach based on *Grid Overlaying* was used by Baker, Grosse, and Raferty in [14] to produce a non-obtuse triangulation of a planar polygon. Recently, [15], Bern, Eppstein, and Gilbert give algorithms for producing guaranteed quality triangulations. They use a regular subdivision of a square, i.e. a quadtree.

In [21] an algorithm is presented to generate good triangulations of the convex hull of a point set in three dimensions. New points are added to generate good tetrahedra with the restriction that all points are added only inside or on the boundary of the convex hull. Good triangulations of convex polyhedra are a special case of this problem. A robust implementation of this 3D triangulation algorithm, in the presence of numerical errors under finite precision arithmetic [22], has been made in our SHILP toolkit [4]. Below is one example of this implementation in SHILP.

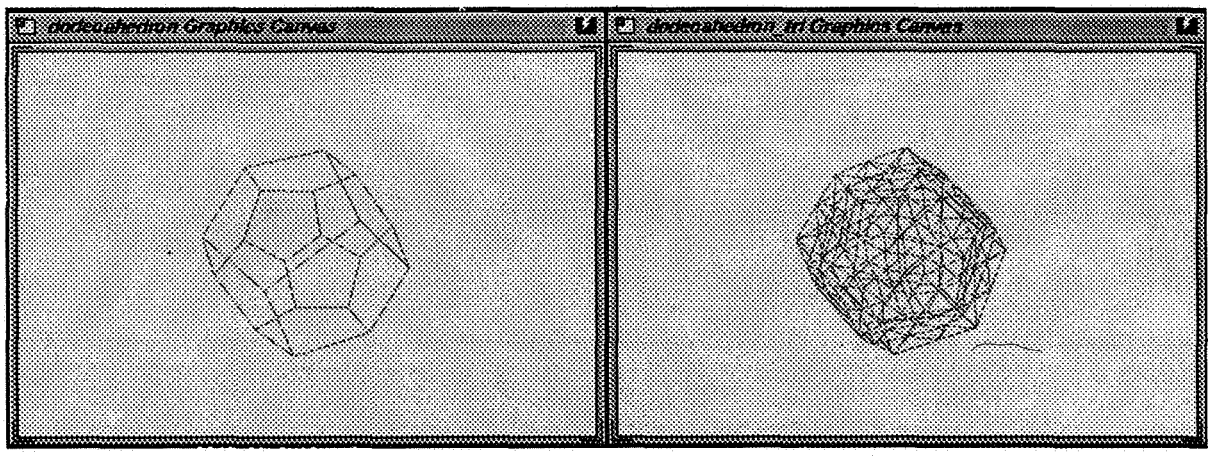


Figure 5: A finite element mesh on the surface of a dodecahedron

FINITE ELEMENT SOLID MODEL RECONSTRUCTION FROM CT/MRI MEDICAL IMAGING DATA

Skeletal model reconstruction from voxel data has been an active research area for many years. There are primarily two classes of model reconstruction techniques. One class of methods first constructs planar contours in each CT/NMR data slice and then connects these contours by a triangulation in three dimensional space. The triangulation process is complicated by the occurrence of multiple contours on a data slice (i.e. branching). Early contributions here are by Keppel [32], Fuchs, Kedem and Uselton [26]. The other class of methods uses a hierarchical subdivision of the voxel space to localize the triangular approximation to small cubes [33]. This method takes care of branching; however the local planar approximation based on the density values at the corner of the subcube may sometimes be ambiguous. An extension of this scheme which computes a C^1 piecewise quadratic approximation to the data within subcubes is given in [34].

In [9] we present an algorithm belonging to each of the above classes to construct C^1-smooth models of skeletal structures from CT/NMR voxel data. The boundary of the reconstructed models consists of a C^1-continuous mesh of triangular algebraic surface patches. The first algorithm starts by constructing C^1-continuous piecewise conic contours on each of the CT/NMR data slices and then uses piecewise triangular algebraic surface patches to C^1 interpolate the contours on adjacent slices. The other algorithm works directly in voxel space and replaces an initial C^0 triangular facet approximation of the model with a highly compressed C^1-continuous mesh of triangular algebraic surface patches. Both schemes are adaptive, yielding a higher density of patches in regions of higher curvature. These algorithms have been implemented in our VAIDAK toolkit [6]. An example of this reconstruction scheme from MRI data is shown below.

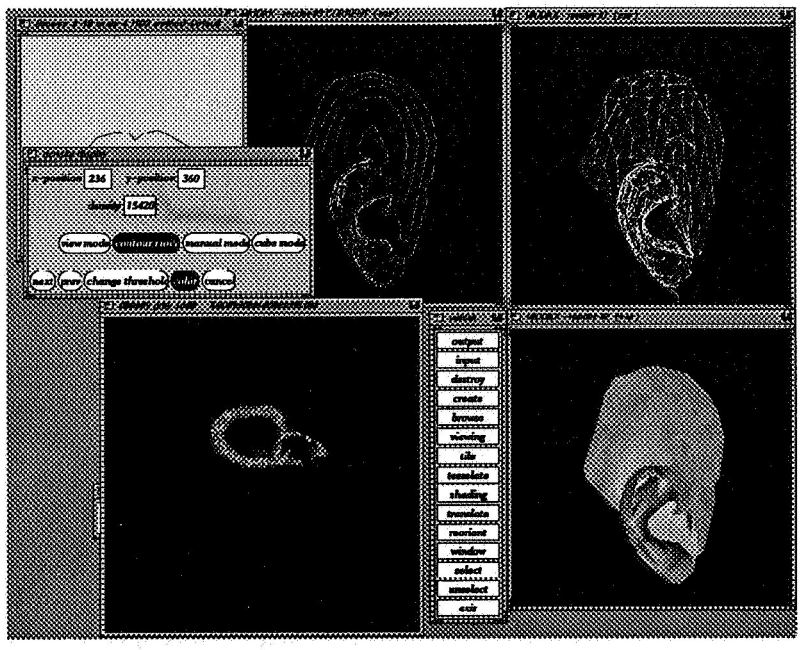


Figure 6: A finite element mesh around the ear of a human head

C^1 CURVED FINITE ELEMENT MESHES OVER TRIANGULATED DOMAINS

The generation of a C^1 mesh of smooth surface patches or *splines* that interpolate or approximate *triangulated space data* is one of the central topics of geometric design. Chui [19] summarizes much of the history of previous work. Prior work on splines has traditionally worked with a given planar triangulation using a polynomial function basis. More recently surface fitting has been considered over closed triangulation in three dimensions using a parametric surface for each triangular face [24, 27, 28, 30, 35].

Little work has been done on spline basis for implicitly defined algebraic surfaces. Sederberg [36] shows how various smooth implicit algebraic surfaces can be manipulated as functions in Bezier control tetrahedra with finite weights. Dahmen [20] presents the construction of tangent plane continuous, piecewise quadric surfaces The technique however works only if the original triangulation of the data set allows a transversal system of planes and hence is restricted. Warren [37] computes low degree blending and joining implicit surfaces by using the products of surfaces defining the blending and joining curves. Bajaj and Ihm [11] show how blending and joining algebraic surfaces can be computed via C^1 interpolation. In [10] we consider an arbitrary spatial triangulation $\mathcal{T}$ consisting of vertices (x_i, y_i, z_i) in $\mathbb{R}^3$ (or more generally a simplicial polyhedron $\mathcal{P}$ when the triangulation is closed), with possibly "normal" vectors at the vertex points. An algorithm is given to construct a C^1 continuous mesh of low degree real algebraic surface patches S_i , which respects the topology of the triangulation $\mathcal{T}$ or simplicial polyhedron $\mathcal{P}$, and C^1 interpolates all the vertices (x_j, y_j, z_j) in $\mathbb{R}^3$. The technique is completely general and uses a single implicit surface patch for each triangular face of $\mathcal{T}$ of $\mathcal{P}$, i.e. no local splitting of triangular faces. Each triangular surface patch has local degrees of freedom which are used to provide local shape control. This is achieved by use of weighted least squares approximation from points (x_k, y_k, z_k) generated locally for each triangular patch from the original patch data points and normal directions on them. These algorithms have been implemented in C in our SHILP toolkit [4]. An example is shown below.

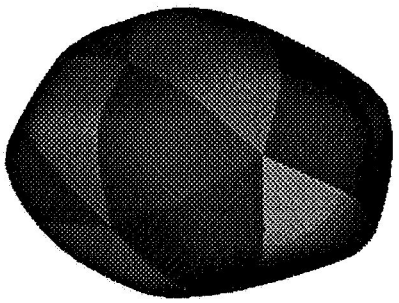


Figure 7: A C^1 curved finite element mesh over a closed spatial triangulation

References

[1] Abhyankar, S., and Bajaj, C., (1987) "Automatic Parameterization of Rational Curves and Surfaces I: Conics and Conicoids", *Computer Aided Design*, 19, 1, pp. 11 - 14.

[2] Abhyankar, S., and Bajaj, C., (1987) "Automatic Parameterization of Rational Curves and Surfaces II: Cubics and Cubicoids", *Computer Aided Design*, 19, 9, pp. 499 - 502.

[3] Alfeld, P., (1989) "Scattered Data Interpolation in Three or More Variables", *Mathematical Methods in Computer Aided Geometric Design*, ed. Lyche, T. and Schumaker, L., pp. 1-34.

[4] Anupam, V., Bajaj, C., Dey, T., Ihm, I., (1991), "The SHILP Solid Modeling and Display Toolkit", Computer Science Tech. Report, CSD-TR-91-064, CAPO-91-29, Purdue University.

[5] Babuska and Aziz, A., (1976), "On the Angle Condition in the Finite Element Method", *SIAM Numerical Analysis*, 13, 214-226.

[6] Bailey, B., Bajaj, C., Fields, M., (1991), "The VAIDAK Medical Imaging and Model Reconstruction Toolkit", Computer Science Tech. Report, CSD-TR-91-066, CAPO-91-31, Purdue University.

[7] Bajaj, C., (1989) "Geometric Modeling with Algebraic Surfaces" *The Mathematics of Surfaces III*, ed., D. Handscomb, Oxford University Press, pp. 3 - 48.

[8] Bajaj, C., (1990) "Rational Hypersurface Display", Proc. of 1990 Symposium on Interactive 3D Graphics, *Computer Graphics*, 24, 2, pp. 117 - 128.

[9] Bajaj, C., (1991) "Electronic Skeletons: Modeling Skeletal Structures with Piecewise Algebraic Surfaces", in Curves and Surfaces in Computer Graphics and Vision II, *Proc. of the Symposium on Electronic Imaging Science & Technology*, Boston, MA, vol. 1610, pp. 231 - 237.

[10] Bajaj, C., and Ihm, I., (1991), "The C^1 Smoothing of Polyhedra with Implicit Surface Patches", Computer Science Tech. Report, CSD-TR-91-060, CAPO-91-27, Purdue University.

[11] Bajaj, C. and Ihm, I., (1992), "Algebraic Surface Design with Hermite Interpolation" *ACM Transactions on Graphics*, 19, 1, (1992).

[12] Bajaj, C., and Royappa, A., (1991), "The GANITH Algebraic Geometry Toolkit, V1.0", Computer Science Tech. Report, CSD-TR-91-065, CAPO-91-30, Purdue University.

[13] Bajaj, C., and Royappa, A., (1992), "The Robust Display of Free-Form Rational Parametric Surfaces", Computer Science Tech. Report, CSD-TR-92-005, CAPO-92-04, Purdue University.

[14] Baker, B., Grosse, E., and Rafferty, C., (1988), "Nonobtuse Triangulation of Polygons", *Discrete and Computational Geometry*, 3, pp. 147-168.

[15] Bern, M., Eppstein, D., and Gilbert, J., (1990), "Probably Good Mesh Generation", *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, pp. 231-241.

[16] Bloomenthal, J., (1988), "Polygonization of implicit surfaces", *Computer Aided Geometric Design*, vol. 5, pp. 341–355.

[17] Chazelle, B., and Palios, L., (1990), "Triangulating a Non-convex Polytope", *Discrete and Computational Geometry*, 5, pp. 505-526.

[18] Chew, L. P., (1989), "Guaranteed-Quality Triangular Meshes", Technical Report TR-89-983, Cornell University.

[19] Chui, C., (1988), *Multivariate Splines*, Regional Conference Series in Applied Mathematics, SIAM.

[20] Dahmen, W., (1989) "Smooth Piecewise Quadric Surfaces", *Mathematical Methods in Computer Aided Geometric Design*, ed. Lyche, T. and Schumaker, L., pp. 181-194.

[21] Dey, T., Bajaj, C., and Sugihara, K., (1991) "On Good Triangulations in Three Dimensions", Proc. of the ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, (1991), pp. 131-141.

[22] Dey, T., Sugihara, K., and Bajaj, C., (1992) "Triangulations in Three Dimensions with Finite Precision Arithmetic", Computer Science Tech. Report, CSD-TR-92-001, CAPO-92-01, Purdue University.

[23] Edelsbrunner, H., Preparata, F., and West, D., (1986), "Tetrahedrizing Point Sets in Three Dimensions", Tech. Report UIUCDCS-R-86-1310.

[24] Farin, G., (1986), "Triangular Bernstein-Bezier Patches", *Computer Aided Geometric Design*, 3, pp. 83 - 127.

[25] Fried, I., (1972), "Condition of Finite Element Matrices Generated from Nonuniform Meshes", *AIAA J.*, 10, pp. 219-221.

[26] Fuchs, H., and Kedem, Z., and Uselton, S., (1977), "Optimal Surface Reconstruction from Planar Contours", *Communications of the ACM*, vol. 20, pp. 693–702.

[27] Gregory, J., and Charrot, P., (1980) "A C^1 Triangular Interpolation Patch for Computer Aided Geometric Design" *Computer Graphics and Image Processing*, 13, pp. 80-87.

[28] Hagen, H. and Pottmann, H., (1989) "Curvature Continuous Triangular Interpolants", *Mathematical Methods in Computer Aided Geometric Design*, ed. Lyche, T. and Schumaker, L., pp. 373-384.

[29] Hall, M., and Warren, J., (1990), "Adaptive Polygonalization of Implicitly Defined Surfaces", *IEEE Computer Graphics and Applications*, pp. 33–42.

[30] Herron, G., (1985) "Smooth Closed Surfaces with Discrete Triangular Interpolants", *Computer Aided Geometric Design*, 2, 4, pp. 297-306.

[31] Joe, B., (1989), "Three-dimensional Triangulations from Local Transformations", *SIAM J. Sci. Stat. Comput.*, 10, pp. 718-741.

[32] Keppel, E., (1975), "Approximating Complex Surfaces by Triangulation of Contour Lines", *IBM J. Research and Development*, vol. 19, pp. 2-11.

[33] Lorensen, W. and Cline, H., (1987), "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *Computer Graphics*, pp. 163-169.

[34] Moore, D., and Warren, J., (1991) "Approximation of Dense Scattered Data using Algebraic Surfaces", *Proc. of the 24th Hawaii Intl. Conference on System Sciences*, Kauai, Hawaii, pp. 681-690.

[35] Peters, J., (1990) "Smooth Mesh Interpolation with Cubic Patches" *Computer Aided Design*, 22, 2, pp. 109-120.

[36] Sederberg, T., (1985), "Piecewise Algebraic Surface Patches", *Computer Aided Geometric Design*, 2, pp. 53-59.

[37] Warren, J., (1989), "Blending Algebraic Surfaces", *ACM Transactions on Graphics*, vol. 8, no. 4, pp. 263-278.

Practical Quality Control Tools for Curves and Surfaces

SCOTT G. SMALL†

November, 1991

Abstract. Curves and surfaces created by Computer Aided Geometric Design systems in the engineering environment must satisfy two basic quality criteria: the geometric shape must have the desired engineering properties, and the objects must be parameterized in a way which does not cause computational difficulty for geometric processing and engineering analysis. This paper describes interactive techniques in use at the Boeing company to evaluate the quality of aircraft geometry prior to Computational Fluid Dynamic analysis, including newly developed methods for examining surface parameterization and its effects.

Key Words. Computer Aided Geometric Design, Computational Fluid Dynamics, Numerical Grid Generation, Geometric Processing

1. Introduction. Users of Computer Aided Geometric Design (CAGD) systems in Aerodynamic applications are keenly aware of geometric or "shape" properties of curves and surfaces, especially those properties which affect aerodynamic performance. Less attention is paid, however, to the parameterization of geometric objects, even though parametric properties can have a large impact on the cost and accuracy of subsequent Computational Fluid Dynamic (CFD) analysis. This paper focuses on the distinction between geometric and parametric properties, and suggests that considerable cost savings can be achieved by detecting problems in geometry before CFD analysis or other processing is performed.

Section 2 of this paper defines the basic difference between geometric and parametric properties, describes some graphical tools for visualizing specific geometric properties, and suggests an application-driven approach to defining and measuring geometric quality. Section 3 presents techniques similar to those in section 2, but applied to parametric properties, and proposes a qualitative definition of parametric quality which applies across many applications. Section 4 contains observations and opinions regarding implementation of quality control in CAGD, and suggests topics for further research.

All examples shown in this paper were produced using the Aero Grid and Paneling System (AGPS), a Boeing CAGD program used for design and analysis of aircraft geometry. References [3] and [4] describe the AGPS system in more detail.

2. Geometric Properties Affecting Quality. I will define a *geometric property* of a curve $\mathbf{c} : [0,1] \rightarrow \mathbf{R}^3$ or surface $\mathbf{s} : [0,1] \times [0,1] \rightarrow \mathbf{R}^3$ as any property which is invariant under reparameterization. These are intrinsic properties having to do only with the shape of the curve or surface. Geometric properties are extensively studied in

† Boeing Computer Services, P.O. Box 24346, Mail Stop 7H-93, Seattle, Washington 98124

the CAGD community; for example, the concept of *geometric continuity* is used to describe continuity of position, tangent, and curvature independently of parameterization. Many tools are available for inspection and manipulation of these properties, but there seem to be no general rules regarding exactly what characteristics are desirable or "good." I claim that this is not surprising, since CAGD systems are used in a wide variety of applications, and quality can only be defined with respect to a particular set of application-dependent requirements. To formalize this concept, I propose the following definition:

> A *geometric quality metric* is a measure of how well the geometric properties of a curve or surface satisfy a particular set of engineering requirements.

The basic goal of Geometric Design in the Engineering environment is the development of curves and surfaces whose geometric properties satisfy a particular set of requirements. It is not unusual for each piece of geometry to be subject to its own unique list of constraints, and the relationships between stated engineering goals and measurable geometric properties can be extremely complicated. In aircraft work, for example, a wing surface might be required to interpolate a large number of fixed data points, possibly with additional tangency and curvature constraints. Structural considerations may limit surface area and enclosed volume, while aerodynamic performance is affected by the distribution of curvature over the entire surface. In this example the interpolation requirements are trivially related to geometric properties, the area and enclosed volume requirements are related through a simple mathematical model, and the aerodynamic characteristics are related in ways which are very difficult to model. Also, many of these requirements will differ from one wing to another. My point is that it is unreasonable to expect a general purpose CAGD system to provide general-purpose geometric quality metrics. Although the same set of geometric properties will always exist, the relationship to quality is totally application dependent. It therefore makes sense to provide a set of flexible tools for interrogating and displaying geometric properties, assisting engineering judgement rather than trying to eliminate it.

2.1. Geometric Properties of Curves. Aerodynamicists are very interested in a class of planar curves known as airfoils, which are the cross-sectional sectional shapes of aircraft wings[1]. These curves are typically defined by a number of discrete points (typically 25 to 100), with the actual parametric curve being constructed by a piecewise polynomial interpolation method. Assuming that the interpolation method produced a reasonable curve with positional and tangential continuity, the designer will be interested in the distribution of curvature along the airfoil, since curvature is directly related to aerodynamic properties. The most common tool for visualizing this type of scalar property is a 2-D plot of the property as a function of the curve's parameter. Illustration 1 shows the type of plot used to examine the curvature of an airfoil. Scalar torsion, and certain types of curvature defined only for curves which lie on surfaces, can also be displayed in this fashion. For a more detailed mathematical description of those properties, see chapter 11 of [1].

The term "fairness" is commonly used to describe the quality of a curve's curvature distribution, and designers may perform a "curve fairing" process where the data and/or interpolation method is adjusted to produce a more desirable curvature plot. Although the details of curve fairing are beyond the scope of this paper (see [7], for example), curvature plots such as the one shown in Illustration 1 are an essential tool in this process.

---

[1]See Chapter 13 of [6] for a more detailed description.

2.2. Geometric Properties of Surfaces. Graphical techniques for evaluation of geometric properties related to surface quality have existed in the automotive and aerospace industries for over ten years[2]. In addition to analyzing planar cuts of a surface using methods described in section 2.1, engineers typically examine shaded renderings under various light sources. Color plots showing a surface "painted" according to a scalar curvature property are also very useful. Finally, "bristle" displays are used to illustrate the surface normal or other vector property at many locations simultaneously. The following sections contain some examples of graphical displays showing geometric surface quality.

2.2.1 Shaded Images. Many techniques for generating realistic, aesthetically pleasing shaded renderings of surfaces have been developed in the Computer Graphics industry.[3] In geometric quality control applications, however, the goal is almost the opposite: we want to make unwanted surface features more obvious. Gross defects such as positional gaps may show up clearly when a standard shading technique is used, but specialized "non-realistic" renderings are sometimes more useful.

One of the most useful methods for checking the behavior of a surface's normal vector is to take a "realistic" grey-shaded rendering and replace adjacent shades of grey with contrasting colors[4]. This trick produces apparent contour curves much like the isophote method described in [5], and can be used to verify planarity or geometric continuity. Illustration 2 shows this technique applied to an aircraft wing.

2.2.2 Color Displays of Scalar Properties. Several different scalar curvature quantities can be defined at a point on a parametric surface[5]. Boeing's AGPS system, for example, can display mean or Gaussian curvature on a surface, as well as the curvature in each of the parametric directions. The user may request that the curvature be sampled on a uniform grid of parametric values, or that an adaptive sampling technique be used. This type of display gives important information about the surface, especially in aerodynamic applications, but their interpretation requires some sophistication and experience. Illustration 3 shows curvature on the surface of an aircraft wing in the span-wise direction. The obvious curvature discontinuity exists because the wing consists of two different surfaces assembled into a single "composite" surface.

2.2.3 Displays of Vector Properties. One way to investigate the behavior of a surface's normal vector is to simply draw a representation of this vector at various points on the surface. For example, one may draw surface normals at specific points on a wing, as shown in Illustration 4. Rather than drawing surface normals at many points simultaneously, it is often more useful to allow the user to interactively select and change the location at which a single vector is displayed. In this way, one may study the behavior of the surface normal in regions of interest.

---

[2]For example, [8] describes color graphic curvature inspection tools in use at General Motors in the early 1980's.

[3]Standard Computer Graphics textbooks such as [9] discuss Illumination and Shading at great length.

[4]This can be done easily in many CAGD systems by manipulating the entries in a color look-up table.

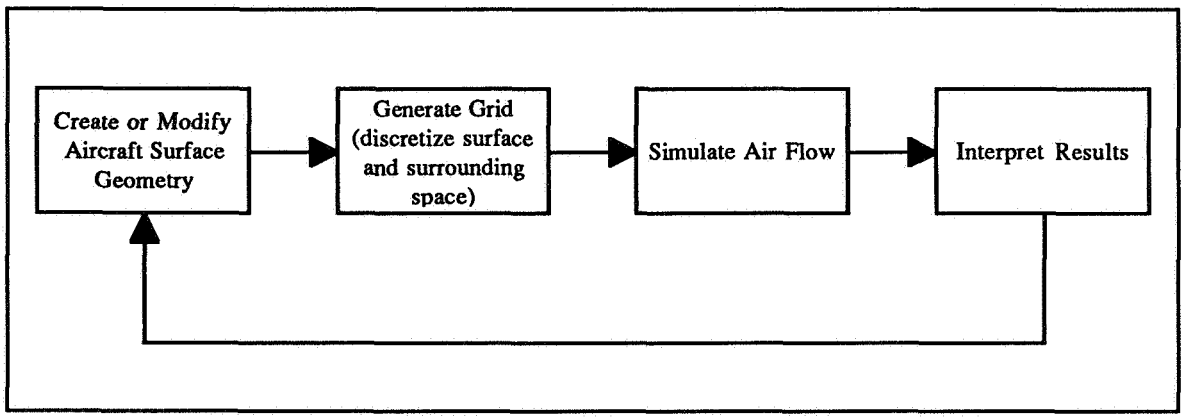
[5]For a more detailed discussion of surface curvature, see a Differential Geometry or Geometric Design text such as chapter 5 of [10].

3. Parametric Properties Affecting Quality. Given curves and surfaces whose geometric properties satisfy the relevant engineering requirements, why does it matter how the geometry is parameterized? It seems reasonable to focus on geometric properties when modeling a physical object in a CAGD system, since parameterization is only a mathematical artifact which ideally should have no physical manifestation. The main message of the following sections, however, is that parametric properties are critical when the CAGD model is subjected to geometric processing or computational analysis, and that in some cases, choices made regarding the paramaterization of a surface can have a drastic effect on the surface's geometric shape. Paralleling section 2, I propose the following definition:

> A *parametric quality metric* is a measure of how well a curve or surface's parameterization satisfies the requirements of the geometric processing, grid generation, and engineering analysis algorithms to be used.

Assuming that parameterization does matter, users would like their CAGD system to automatically choose a good parameterization as objects are created, or at least to repair any problems which developed. Although automatic quality assurance is an important research topic, the current state of the art depends heavily on the user to detect problems, and repair techniques are usually ad hoc and difficult. Section 3 of this paper focuses on graphic tools which make the job of parameterization checking easier.

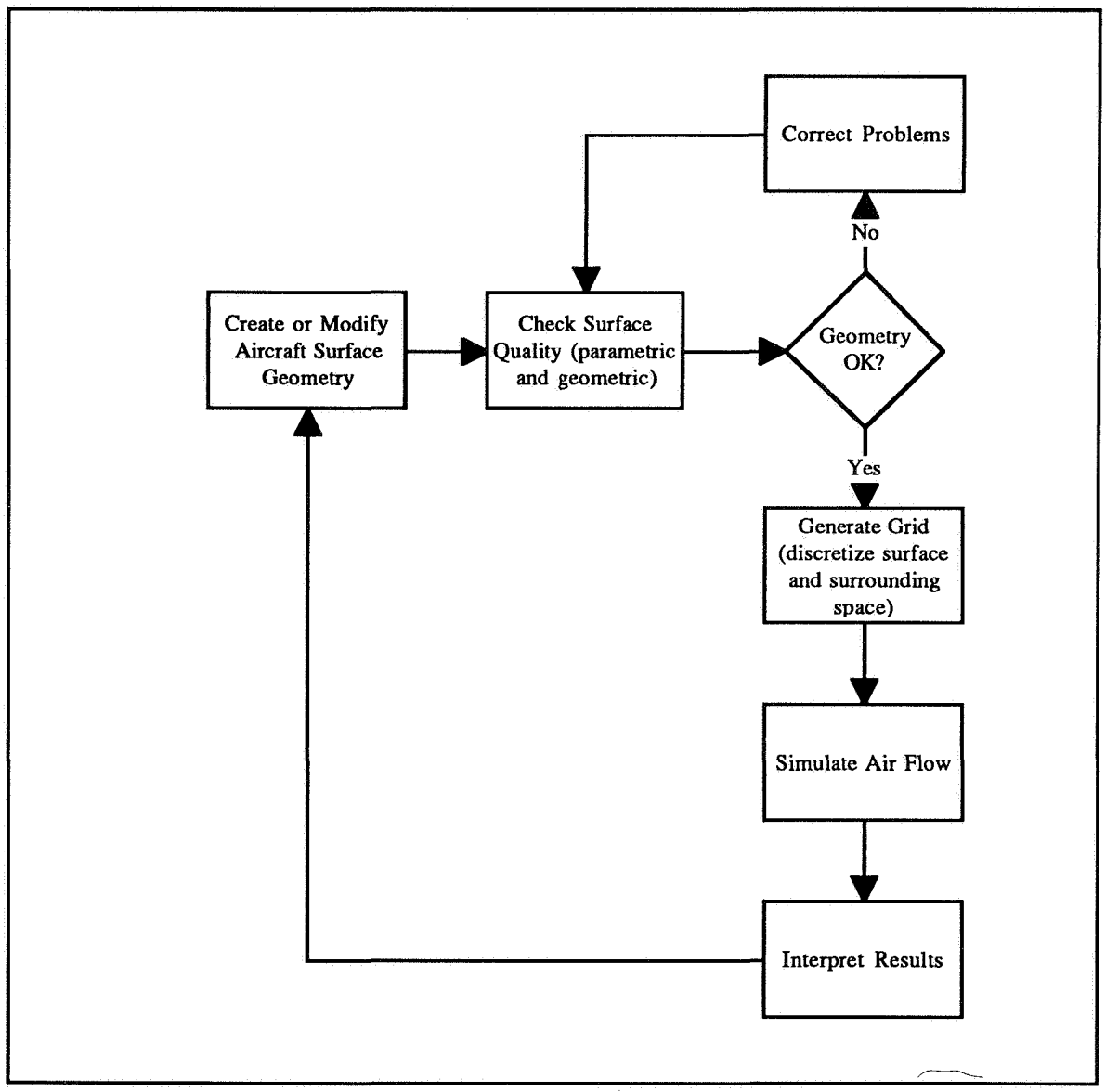
In the Computational Fluid Dynamics (CFD) environment, many proposed aircraft configurations are modeled in a CAGD system and analyzed by a separate computer program which simulates air flow around the surfaces of interest. The results of the simulation are then examined using a visualization tool (which in our case is also AGPS), and the configuration model is modified and analyzed again, as shown in the following diagram:



The preceding diagram may be viewed as a manufacturing process, where the product being produced is an aircraft surface definition, and the tools used are those provided by the CAGD system. In order to optimize a process, it is necessary to measure the quality of its output. Also, if a defect is introduced into a process, the resulting cost is generally lower if the defect can be detected earlier. Thus it makes sense to provide low-cost inspection tools in a CAGD system, and to encourage the engineers to use them throughout the geometry creation process.

Some issues related to Numerical Grid Generation provide a good illustration of the benefits of inserting quality control steps into a Geometric Design process. Most fluid flow

simulation programs require a discretized representation of the aircraft surface as input; that is, the CAGD system is required to produce a "grid" of discrete points which, when interpolated in some way, represent the surface to a specified accuracy. Also, the engineer will want to increase the grid density at specific locations to capture certain features of the flow. Grid generation is a complex issue by itself[6], but the important point here is that most grid generation techniques are sensitive to curve and surface parameterization. The mathematical details remain to be shown, but the point is that a significant amount of frustration can be avoided by inserting a "quality control" step into the process as shown in the following modified diagram:



[6]Textbooks such as [11], and several yearly conferences, are devoted just to grid generation.

Geometric processing methods are also highly sensitive to curve and surface parameterization. Intersecting two geometric objects is a fundamental operation in all CAGD systems, and the iterative methods commonly used depend upon parametric continuity[7]. Again, "bad" parameterization can cause slow performance, inaccuracy, and even total failure in this type of operation. In the following sections, I will give some explanation of why this happens, and show some tools which can help the user detect this type of problem early in the design process.

3.1. Parametric properties of curves. For a curve $\mathbf{c} : [0,1] \rightarrow \mathbf{R}^3$ which maps a parametric variable u to points ($x(u)$, $y(u)$, $z(u)$), the most important scalar parametric property is the *parametric velocity*[8] $PV(u)$ defined by

$$PV(u) = \sqrt{(x'(u))^2 + (y'(u))^2 + (z'(u))^2} \ ,$$

where the prime indicates differentiation with respect to u. To see why parametric velocity is of interest, note that $PV(u)$ is the magnitude of the curve's tangent vector at the parameter value u, and that the arc length of the curve between the parameter values u_0 and u_1 is given by

$$ARCLEN(u_0,u_1) = \int_{u_0}^{u_1} PV(u)du$$

Since this integral generally cannot be computed in closed form for non-linear curves, a numerical method[9] must be used, and the convergence of this method can be very sensitive to the smoothness of the integrand. Jump discontinuities in a curve's PV have the worst effect on arc length convergence, while a smooth, slowly varying (or constant) PV is best. In a typical grid generation application, where a number of points are to be distributed along a curve according to some specified arc length distribution, the majority of the CPU time is spent in the multiple evaluations of $PV(u)$ required by the numerical integration routine. Curves with well-behaved PV can clearly be gridded more accurately at lower cost.

The basic tool for evaluation of a curve's parametric velocity is a plot of $PV(u)$ such as the one shown in Illustration 5. Users would like an automated, quantitative method for evaluating the quality of a curve's parameterization, but it is difficult to find any applicable mathematical results. The main source of difficulty is the "black box" paradigm used in many CAGD systems, wherein geometry can only be evaluated at individual parameter values. Without an analytic description or any other global information, we can only work with discrete samples of properties such as $PV(u)$. An obvious approach to estimating the cost of arc length calculation is simply to approximate a curve's arc length with an adaptive quadrature method, and count the number of evaluations of $PV(u)$. This technique measures the computational cost associated with curve parameterization, but it cannot

---

[7]See the article [12] or the textbook [10] for descriptions of typical surface-surface intersection algorithms.

[8]The term "parametric velocity" has historically been used for this scalar quantity, although "parametric speed" would be more correct.

[9]The AGPS system uses an adaptive Romberg integration method, with recursive subdivision in difficult cases.

evaluate the accuracy of the computations. Several people have suggested the application of discrete Fourier transform techniques to a sample of PV(u), but it is not clear how to relate the results to computational cost or accuracy. The area of quantitative parametric quality metrics seems to be wide open for research.

3.2. Parametric Properties of Surfaces. For a surface $\mathbf{s} : [0,1] \times [0,1] \rightarrow \mathbf{R}^3$ which maps the parametric variables u and v to points ($x(u,v)$, $y(u,v)$, $z(u,v)$), one may evaluate the parametric velocity of various curves passing through a given point on the surface. That is, one may evaluate the surface's parametric velocity in any desired parametric direction. The most obvious choices are simply the u and v directions, and the AGPS system provides color graphic display of the properties PV_u and PV_v defined by

$$PV_u(u,v) = \sqrt{(x_u(u,v))^2 + (y_u(u,v))^2 + (z_u(u,v))^2} \quad \text{and}$$

$$PV_v(u,v) = \sqrt{(x_v(u,v))^2 + (y_v(u,v))^2 + (z_v(u,v))^2} \quad ,$$

where the subscript u and v inside the radical denote differentiation of the coordinate functions with respect to one parameter.

In addition to the parametric velocity in a particular direction on a surface, one may compute an "area expansion factor" AEF defined by

$$AEF(u,v) = \sqrt{E(u,v)\ G(u,v) - F(u,v)^2} \quad ,$$

where

$$\begin{aligned} E(u,v) &= (x_u(u,v))^2 + (y_u(u,v))^2 + (z_u(u,v))^2 \\ F(u,v) &= x_u(u,v)x_v(u,v) + y_u(u,v)y_v(u,v) + z_u(u,v)z_v(u,v) \\ G(u,v) &= (x_v(u,v))^2 + (y_v(u,v))^2 + (z_v(u,v))^2 \end{aligned}$$

To visualize this property, consider a small rectangle with area $du * dv$ in parameter space, located approximately at the point (u,v). This parametric area element is mapped onto a surface element whose area is approximately

$$AEF(u,v)\ du\ dv$$

In the formal language of differential geometry, AEF is the *discriminant* of the *first fundamental form* on the surface. Intuitively, it measures the "average parametric velocity" in all directions.

A plot of AEF on a surface is useful in checking for parametric discontinuities between surface patches. As a simple example, consider a surface composed of bicubic patches of differing size. The CAGD system must construct an overall parameterization for the whole surface, and this is typically done by dividing the unit square into rectangles, each of which is mapped onto one surface patch. A simpleminded approach is to divide the overall parameter space into rectangles of uniform size, but this generally produces parametric discontinuities at patch boundaries if the patches are not identical. Illustration 6 shows the what can happen to the area expansion factor when a CAGD system does not divide the surface's overall parameter space intelligently among the surface patches. This situation occurs surprisingly often in practice, especially when geometry is imported from

another CAGD system, since mathematical translations may be performed which do not preserve parametric continuity.

The motivation for checking the quality of a surface's parameterization is essentially the same as for curves: geometric processing techniques are sensitive to the continuity and smoothness of surface parameterization. For example, suppose a surface is "cut" with planes at several locations, and that a grid is generated by distributing points by arc length along each of the cut curves. Each cut curve will inherit its parameterization from the parent surface, and by the arguments made in section 3.1, the gridding process will be more accurate and less expensive if the surface is smoothly parameterized. Again, further research is needed to produce automated, quantitative parametric quality metrics.

3.3. Surfaces Which Cannot be Properly Parameterized. As described in section 17.5 of [1], the method commonly used to construct bicubic spline surfaces interpolating an array of data points only works when the data points are arranged in a "nice" rectangular grid. This is because one set of parameter breaks is used for all isoparametric curves in the u direction, and another set is used for all curves in the v direction. If the data points are arranged in a way which demands a different parameterization on the various isoparametric curves, there is no hope of constructing a surface without unwanted folds or ripples. These defects are obvious in extreme cases such as the one shown in Illustration 7, but more subtle ripples can be hard to find.

Since small geometric features can affect the solutions produced by fluid flow simulation codes, it is important to provide the designer with a means of detecting them. A very effective detection tool was recently implemented in the AGPS system, based on this observation: where an unwanted surface feature exists, the arc length on the surface along an isoparametric curve is different from the arc length on a curve fit through the single "row" or "column" of data points. In other words, the surface is either too "loose" or too "tight," due to the compromise made when selecting the overall u and v parameter breaks. Since a single curve through a subset of the data points needs no such compromise, it is a good standard for comparison.

Using the AGPS system's geometric programming language, for example, it was not difficult to write a procedure which takes an arbitrary surface and checks the arc length of its isoparametric curves. A graphic display is produced showing the surface "painted" with colors corresponding to the degree of discrepancy found, making unwanted features stand out clearly. Illustration 8 shows this technique applied to the gross example used in Illustration 7, and Illustration 9 shows an aircraft body surface containing a subtle defect which is difficult to find using any other technique.

4. Summary and Conclusions. I have taken the viewpoint that Computer Aided Geometric Design is a process whose output is curves and surfaces, and expressed the need to measure the quality of these products throughout the process. Emphasizing the distinction between geometric and parametric properties, I claim that geometric quality must be the user's responsibility, but that CAGD system designers should take responsibility for parametric quality. Experienced engineers develop an understanding of the relationship between geometric properties and quality as defined in their application, and several types of graphical tools can be very helpful in focusing engineering judgement. Parametric quality, although not as commonly appreciated, can also be efficiently evaluated using graphical techniques. Since geometric processing and engineering analysis are relatively expensive, a large cost saving can be realized by early detection of defects which could cause subsequent computations to fail; thus I recommend addition of a quality-control step to the geometric design process.

Automated inspection and repair of geometry is certainly a worthwhile goal, but the current state of the art is based upon engineering judgement. The inspection process requires the user to make qualitative judgements from color graphic images, and the repair process is entirely ad hoc. I believe that a larger payoff will come from automating the inspection process rather than the repair process, since many curves and surfaces need to be inspected, but relatively few will need repair. Automated, quantitative geometric inspection remains an important topic for research.

REFERENCES

[1] G. FARIN, *Curves and Surfaces for Geometric Design*, Academic Press, San Diego, California, 1990.

[2] C. HOFFMANN, *Geometric & Solid Modeling: An Introduction*, Morgan Kaufmann, San Mateo, California, 1989.

[3] D. SNEPP AND R. POMEROY, *A Geometry System for Aerodynamic Design*, AIAA/AHS/ASEE Aircraft Design, Systems and Operations Meeting, St. Louis, Missouri, September 1987.

[4] W. CAPRON AND K. SMIT, *Advanced Aerodynamic Applications of an Interactive Geometry and Visualization System*, 29th Aerospace Sciences Meeting, Reno, Nevada, January 1991.

[5] H. HAGEN, TH. SCHREIBER, AND E. GSCHWIND, *Methods for Surface Interrogation*, Proceedings of the First IEEE Conference on Visualization, San Francisco, California, October 1990.

[6] R. SHEVELL, *Fundamentals of Flight*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[7] R. BARNHILL, *Geometry Processing: Curvature Analysis and Surface-Surface Intersection*, Mathematical Methods in Computer Aided Geometric Design, Academic Press, San Diego, CA, 1989.

[8] J. DILL, *An Application of Color Graphics to the Display of Surface Curvature*, Computer Graphics, Vol. 15, No. 3, August 1981.

[9] J. FOLEY, A. VAN DAM, S. FEINER, J. HUGHES, *Computer Graphics*, Addison-Wesley, Reading, MA, 1990.

[10] M. MORTENSON, *Geometric Modeling*, John Wiley & Sons, New York, 1985.

[11] J. THOMPSON, Z. WARSI, C MASTIN, *Numerical Grid Generation*, North-Holland, Amsterdam, 1985

[12] R. BARNHILL, G. FARIN, M. JORDAN, B. PIPER, *Surface / surface intersection*, Computer Aided Geometric Design 4, 3-16.

Curvature Plot for an Airfoil

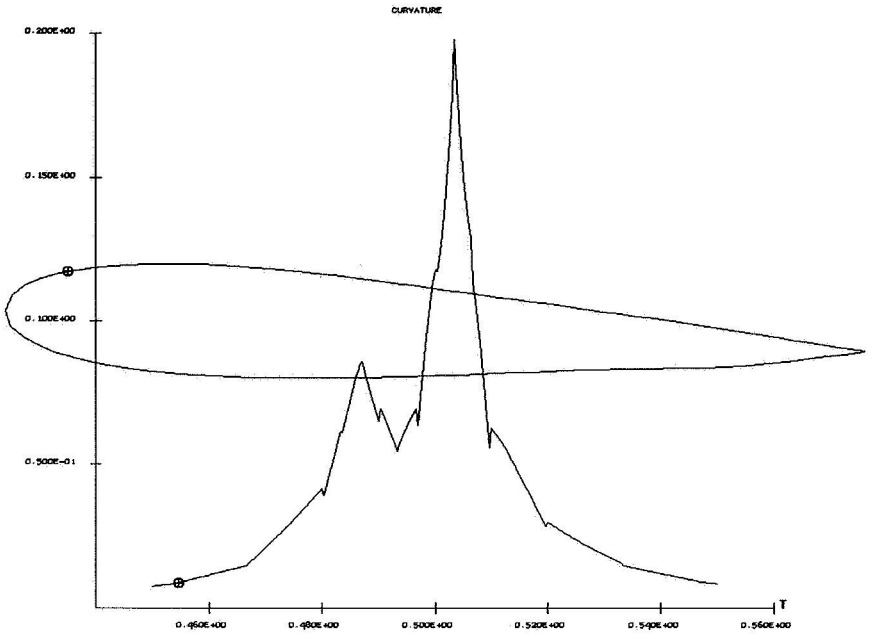


Illustration 1

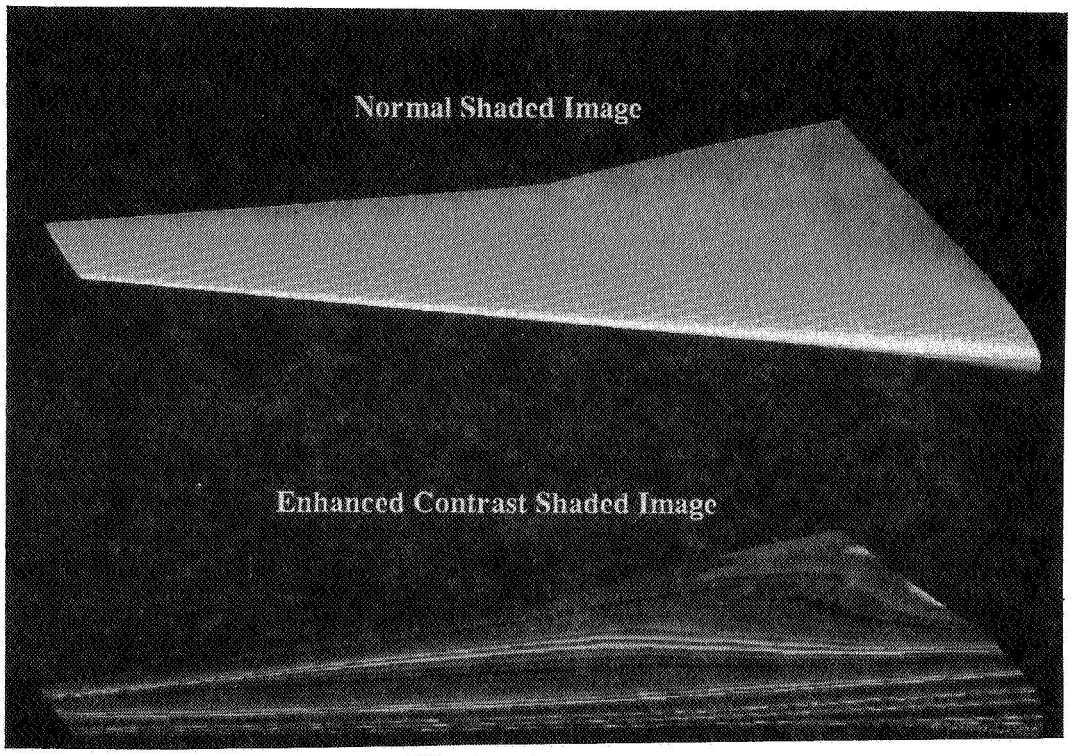
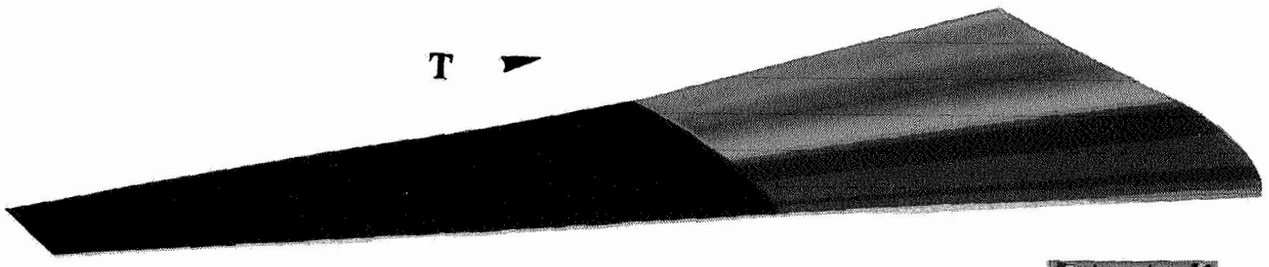


Illustration 2

Curvature in the T Direction on a Wing Surface



(discontinuity due to composite surface)

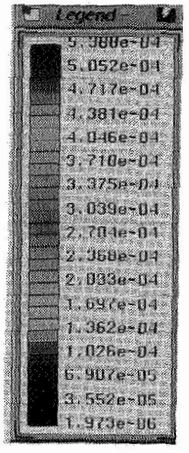


Illustration 3

Normal Vectors on a Wing Surface

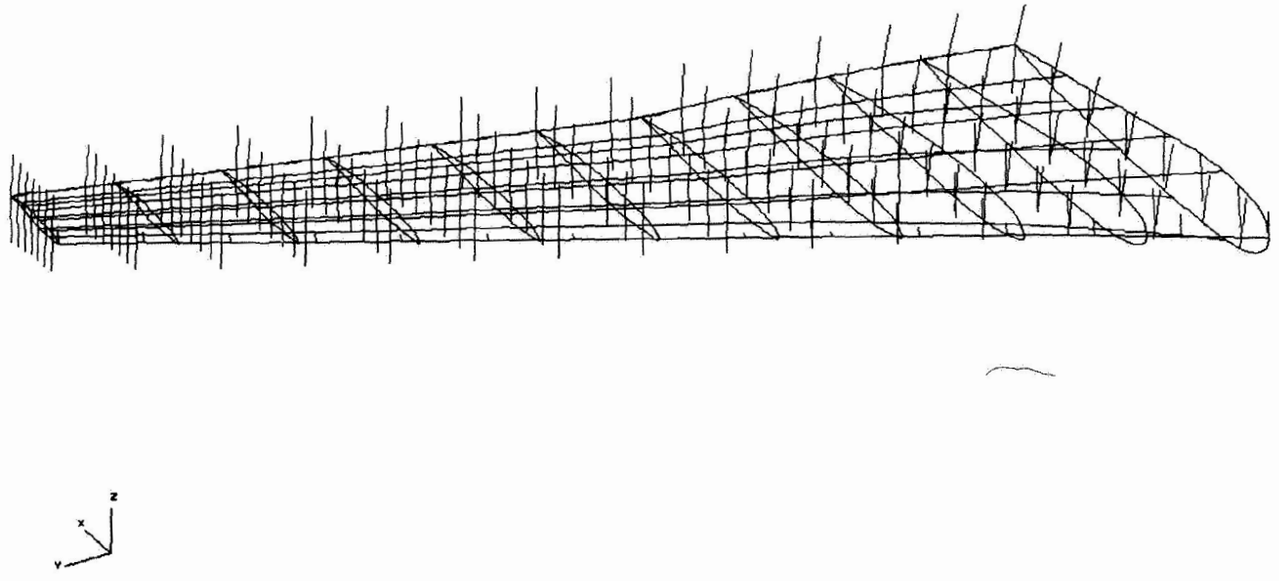


Illustration 4

Parametric Velocity Plot for Airfoil

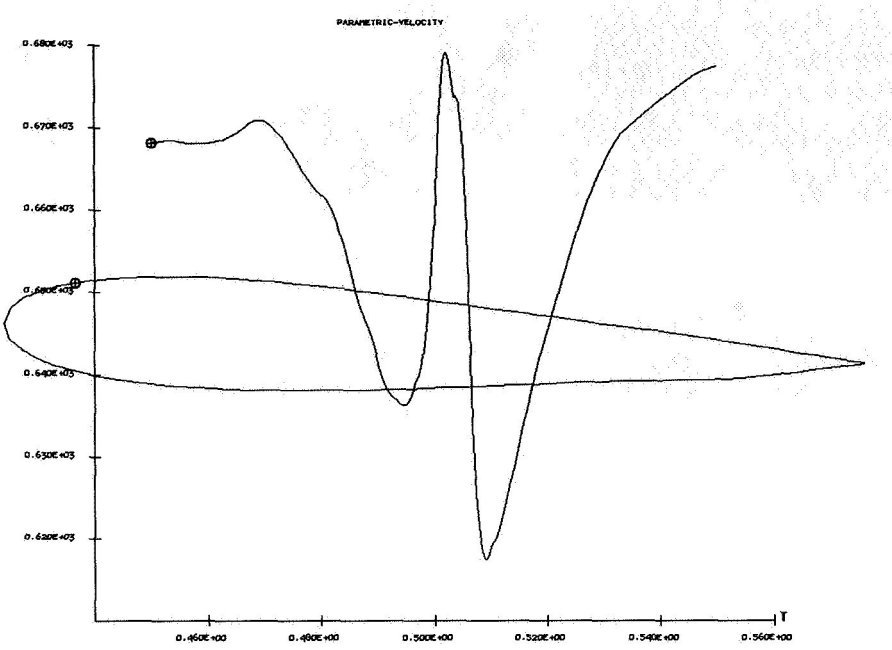


Illustration 5

Area Expansion Factor

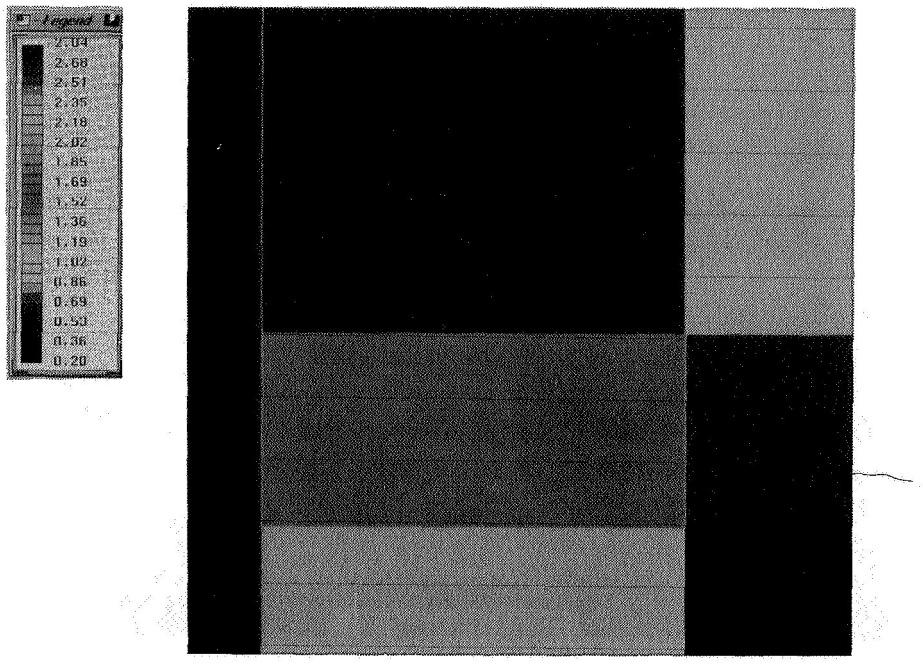


Illustration 6

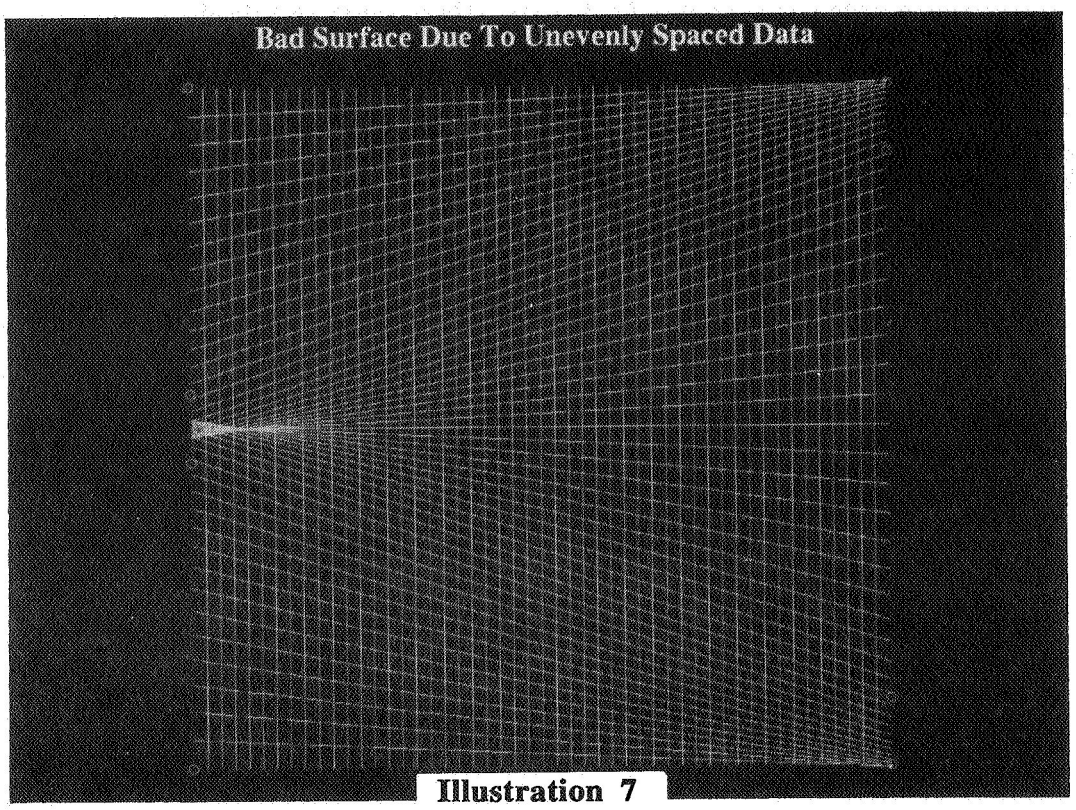


Illustration 7

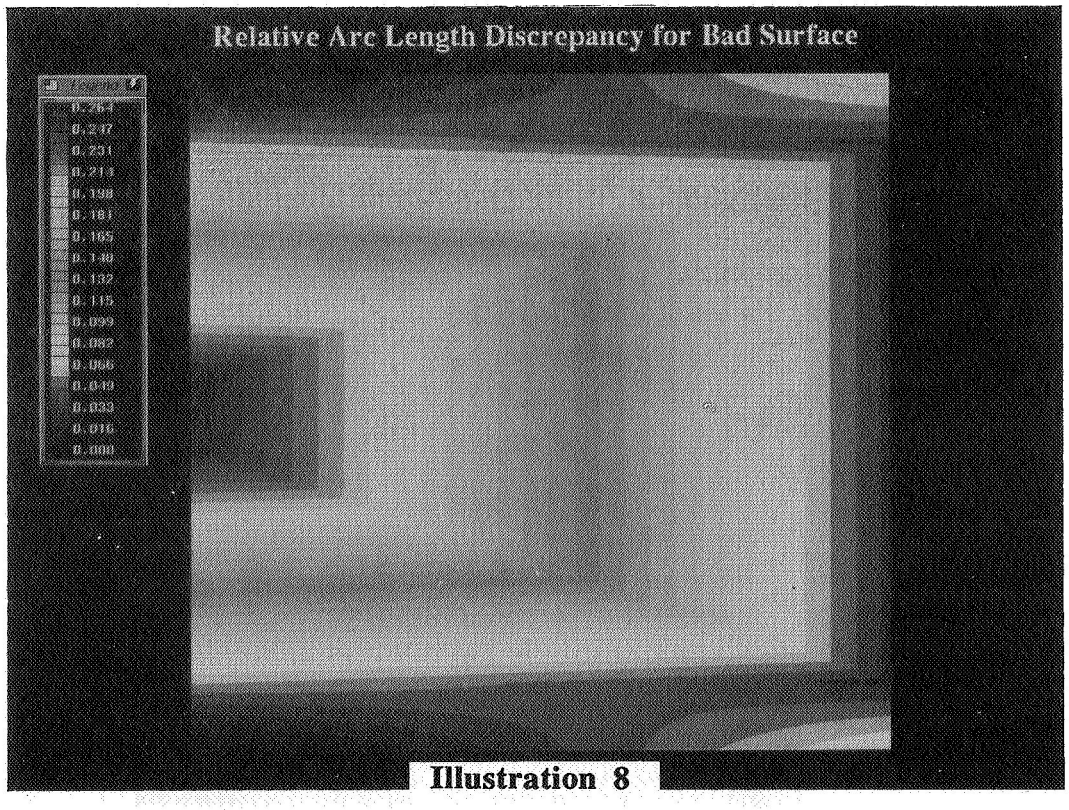


Illustration 8

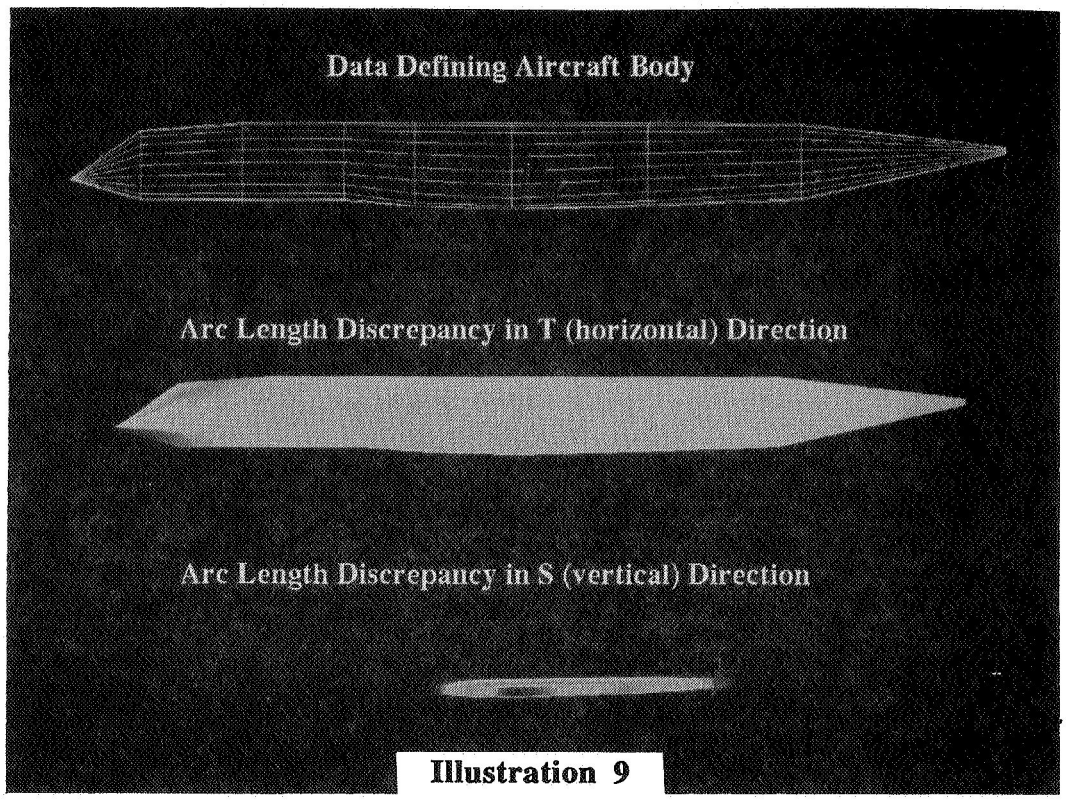


Illustration 9

THREE-DIMENSIONAL SURFACE GRID GENERATION FOR CALCULATION OF THERMAL RADIATION SHAPE FACTORS

Hany M. Aly
Senior Member of Technical Staff
Aerophysics Department
TRW Ballistic Missiles Division
San Bernardino, CA 92402

SUMMARY

The present paper describes a technique to generate three-dimensional surface grids suitable for calculating shape factors for thermal radiative heat transfer. The surface under consideration is approximated by finite triangular elements generated in a special manner. The grid is generated by dividing the surface into a two-dimensional array of nodes. Each node is defined by its coordinates. Each set of four adjacent nodes is used to construct two triangular elements. Each triangular element is characterized by the vector representation of its vertices. Vector algebra is utilized to calculate all desired geometric properties of grid elements. The properties are used to determine the shape factors between the element and an area element in space. The generated grid can be graphically displayed using any software with 3-dimensional features. In the present paper, DISSPLA was used to view the grids

INTRODUCTION

The thermal radiation shape or configuration factor between two surfaces has been the subject of many investigations. The determination of this factor is important in thermal radiation heat transfer applications which are encountered in a wide variety of engineering systems. Analytical derivation of shape factors, even for simple configurations, is very complex. Therefore, numerous tables have been generated and tabulated in the literature for basic geometries [e.g., see Hamilton and Morgan (reference 1) and Howell (reference 2)]. Despite the existence of these tables, different configurations are often needed so that it is not practical to tabulate all possible geometries. Furthermore, cases involving complex geometries may result in analytically non-integrable expressions. Therefore, numerical methods become more attractive to use in such cases, especially when incorporated in a computer code which handles the heat transfer calculations.

This paper describes a grid generation technique which was developed to aid in the numerical calculation of the shape factor between an area element and an arbitrary three-dimensional surface in space. In the present scheme, a given three-dimensional area is discretized into finite triangular elements (FTE) using the surface nodes in a special manner. Each triangular element is characterized by the vector representation of its vertices. The differential shape factor between a differential area element whose location and orientation

are fixed and each triangular element is calculated using the geometric properties of the triangle. The shape factor between an area element and the entire surface is obtained by summation of the differential shape factors for all triangles. This method is applicable to generalized three-dimensional areas without restrictions. However, blockage or shadowing effects due to other parts of the surface must be taken into account when the shape factor calculations are performed. Grids for selected sample cases were generated and the corresponding numerically calculated shape factors were compared with the analytical solutions in order to validate this technique.

SHAPE FACTOR EVALUATION

The shape factor between two differential planer area elements dA_1 and dA_2, shown in Figure 1, is defined (e.g., reference 1) as

$$F_{dA1-dA2} = \frac{\cos\theta_1 \cos\theta_2}{\pi S^2} dA_2 \tag{1}$$

where $F_{dA1\text{-}dA2}$ is the shape factor representing the fraction of the energy leaving the area element dA_1 that is arriving at the area element dA_2; θ_1 and θ_2 are the angles from normal for the surfaces 1 and 2, respectively; and S is the distance between the two surfaces.

In practical applications, it is usually desired to determine the radiant heat transfer between a planer differential area and a finite surface. Thus, the corresponding shape factor, $F_{dA1\text{-}A2}$, is obtained by integrating Equation 1 over A_2 as follows:

$$F_{dA1-dA2} = \int_{A_2} \frac{\cos\theta_1 \cos\theta_2}{\pi S^2} dA_2 \tag{2}$$

Equations 1 and 2 are the basic equations governing the view of one surface relative to another in space. Equation 2 can be integrated to define an overall shape factor between two finite areas as outlined by Siegel and Howell (reference 3). The mathematical treatment of such equations is tedious. In addition, Equation 2 is integrable only for relatively simple geometries. Although numerous tables were generated and published in the literature (e.g., references 1 and 2), different geometries may be encountered in practical applications. Thus, approximate or numerical methods must be used in these cases.

A numerical integration of Equation 2 can be achieved for an arbitrary surface, A_2, by dividing the surface into discrete elements. An elemental shape factor for an element, ΔA_2, is expressed by replacing Equation 1 by its equivalent in finite form as

$$F_{dA1-\Delta A2} = \left(\frac{\cos\theta_1 \cos\theta_2}{\pi S^2}\right) \Delta A_2 \tag{3}$$

Then Equation 2 can be evaluated by the following summation:

$$F_{dA1-A2} = \frac{1}{\pi} \sum_{i=1}^{n} \left(\frac{\cos\theta_1 \cos\theta_2}{S^2}\right)_i \Delta A_i \tag{4}$$

where n is the total number of elements generated and i is an index representing the ith element. A special grid generation method has been developed to construct a mesh of finite triangular elements (FTE). The elements are used to calculate the shape factor expressed in Equation 4. Details of the calculations of the terms in Equation 4 are beyond the scope of this paper.

GRID GENERATION METHOD

The present grid generation scheme was developed for an arbitrary three-dimensional surface in space. The surface under consideration can be defined by intersecting lines, as demonstrated by Figure 2. The points of intersection of these lines are the nodes used to construct the triangular elements as outlined below.

Consider a node b generated from the intersection of the ith α line with the jth β line as shown in Figure 2. This node is defined by its coordinates. For each node b (i,j), three additional neighboring points, c(i + 1,j), e(i,j + 1), and f(i + 1,j + 1), are generated in a similar manner. Two planer triangles are constructed using these four adjacent nodes. Each triangle is defined by the coordinates of its vertices. Using vector representation of the nodal points b, c, and f, triangle 1 is defined by vectors u and v, while triangle 2 is described by vectors v and w for a typical (i,j) node, as illustrated in Figure 3.

For the purpose of shape factor calculations, the terms in Equation 4 require, for each element, the values of the area, location, and direction of its normal vector. The centroid of a triangular area was chosen to represent its location in space.

The scheme described above is general and is applicable to any arbitrary surface. For a given geometry, one must define the surface of interest in terms of its nodal points before using this scheme. Some simple configurations were included herein for demonstration purposes. The shape factors calculated using the present method were compared with tabulated formulas for examples discussed in the following section. The objective of the present paper is to test and demonstrate the grid generation method and, therefore, no attempt was made to optimize the grid or to perform an error analysis. For visualization purposes, the generated grids were plotted using the DISSPLA graphics system.

EXAMPLES

In order to test the present scheme, comparisons with tabulated shape factors were made for some example cases. In each case, the geometry for the area A_2 was used to generate the nodal points covering the surface under consideration. A general FTE grid generator using this nodal information was used to calculate the geometric properties of the triangular elements. These properties were used to calculate the shape factor defined in Equation 4. The following four configurations were considered to test the present scheme: (1) rectangular area parallel to an element adjacent to one corner (Figure 3), (2) circular area parallel to an element (Figure 4), (3) cylinder and an element parallel to its axis with its normal passing

through one end (Figure 5), and (4) sphere and an element with its normal passing through the center (Figure 6). These test configurations, chosen for their simplicity, do not imply any limitation on the present scheme and were intended for validation only.

The results for Configuration 1 are summarized in Table I for various values of A/C and B/C (see Figure 3). Table I also includes the exact results obtained from the closed form solution reported by Hamilton and Morgan (reference 1) for comparison. All runs were made for a grid size of 25 x 20. Thus, the total number of triangular elements generated was 1000 in each run. The results are in excellent agreement with the exact solution for this grid. In general, the accuracy of the results depends on the grid size and the location of the element relative to the finite area. The grid can be plotted using any three-dimensional plot system to view the surface considered. Figure 7 is a computer-generated plot of the triangular elements constructed for a rectangle with A/B = 2. The plot shown was produced by the DISSPLA graphics system.

Table II summarizes the results for Configuration 2 (Figure 4) for various values of R/C and C/A. In this case, the nodal points were generated by choosing the α lines in the radial direction and the β lines in the tangential direction. The center of the circle can be avoided numerically by using a small concentric circle with a radius approaching zero. The results were compared with the exact solution reported by Hamilton and Morgan (reference 1) for a 25 x 25 grid. Figure 8 shows the plotted geometry.

Configuration 3 is an example of a three-dimensional surface. The surface nodes were generated from the intersection of the lines parallel to the cylinder axis (α lines) and circular lines in the tangential direct (β lines). The results summarized in Table III were obtained using a 20 x 50 grid and were compared with the exact solution reported by Hamilton and Morgan (reference 1). The grid is displayed in Figure 9 for A/R = 5.

The shape factor between a sphere and a differential element has been treated extensively in the literature, and formulas were derived for various cases involving the location and orientation of the element relative to the sphere. These formulas were summarized by Howell (reference 2). Configuration 4 is the simplest case in which the normal to the element passes through the center of the sphere. Table IV compares the results of the numerical calculations with the exact solution for various values of C/R and a 40 x 40 grid. Figure 10 is a DISSPLA output of the spherical grid.

CONCLUSION

This paper demonstrates the use of a special grid generator for calculating thermal radiation shape factors. The present method is based on finite triangular elements which are used to integrate the shape factor equation numerically. It has the advantage of being applicable to arbitrary three-dimensional surfaces without restrictions in addition to simplicity. Comparisons were made with tabulated results to validate the method.

REFERENCES

1. Hamilton, D. C.; and Morgan, W. R.: "Radiant-Interchange Configuration Factors," NASA TN-2836, 1952.

2 Howell, John R.: *A Catalog of Radiation Configuration Factors*, McGraw-Hill, New York, 1982.

3. Siegel, Robert; and Howell, John R.: *Thermal Radiation Heat Transfer*, Second ed, McGraw-Hill, New York, 1981.

Table I. Results of Configuration 1 for Various Values of A/C and B/C

A/C	B/C	EXACT SOLUTION	PRESENT METHOD
0.1	0.1	0.003141	0.003141
0.1	10	0.024865	0.024864
2	1	0.167375	0.167361
2	10	0.223400	0.223046
10	10	0.247971	0.246073

Table II. Results of Configuration 2 for Various Values of R/C and C/A

R/C	C/A	EXACT SOLUTION	PRESENT METHOD
0.5	1	0.06588	0.06515
0.5	10	0.1975	0.1959
1	1	0.2764	0.2736
1	10	0.4975	0.4957
5	1	0.9585	0.9608
5	10	0.9615	0.9622

Table III. Results of Configuration 3 for Various Values of A/R and C/R

A/R	C/R	EXACT SOLUTION	PRESENT METHOD
0.1	5	0.003050	0.003046
2	5	0.053239	0.053162
2	10	0.013430	0.013423
5	5	0.086983	0.086870
5	10	0.029211	0.029199
10	10	0.042189	0.042174

Table IV. Results of Configuration 4 for Various Values of C/R

C/R	EXACT SOLUTION	NUMERICAL
1.5	0.4444	0.4427
2	0.2500	0.2490
10	0.0100	0.00996
100	0.0001	0.0001

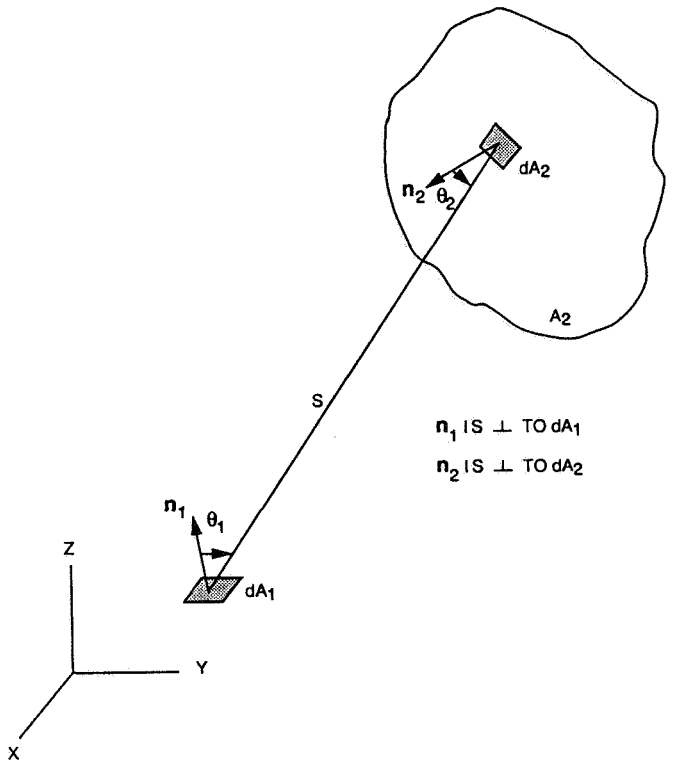


Figure 1. Geometric representation of shape factor between two differential elements.

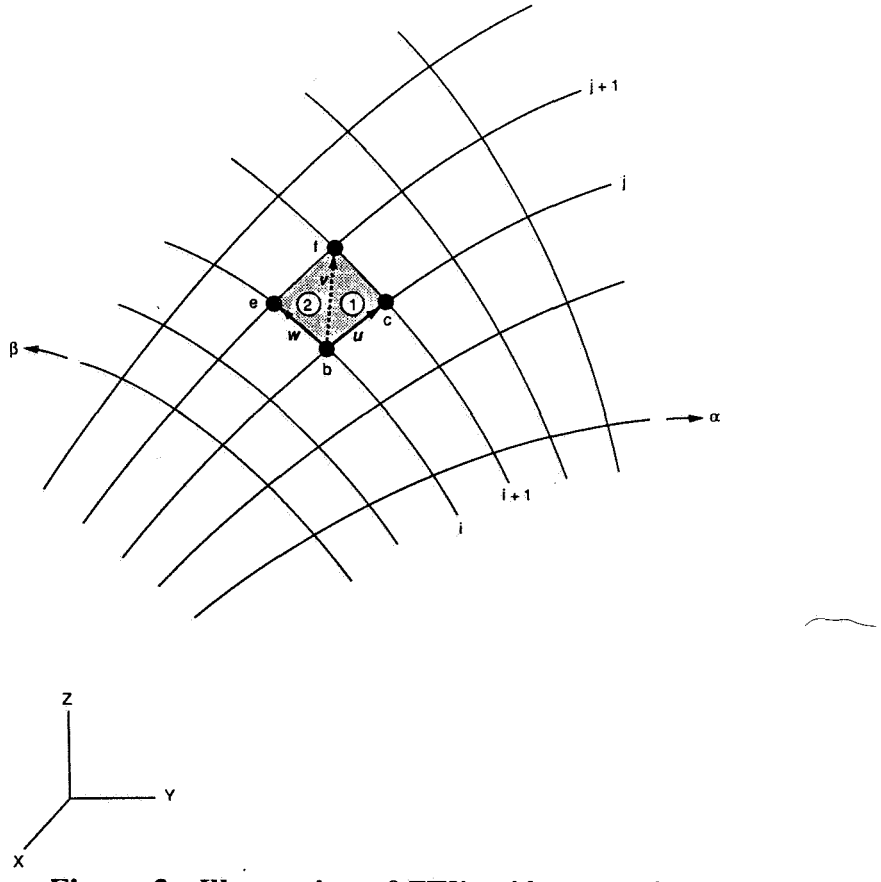


Figure 2. Illustration of FTE grid generation.

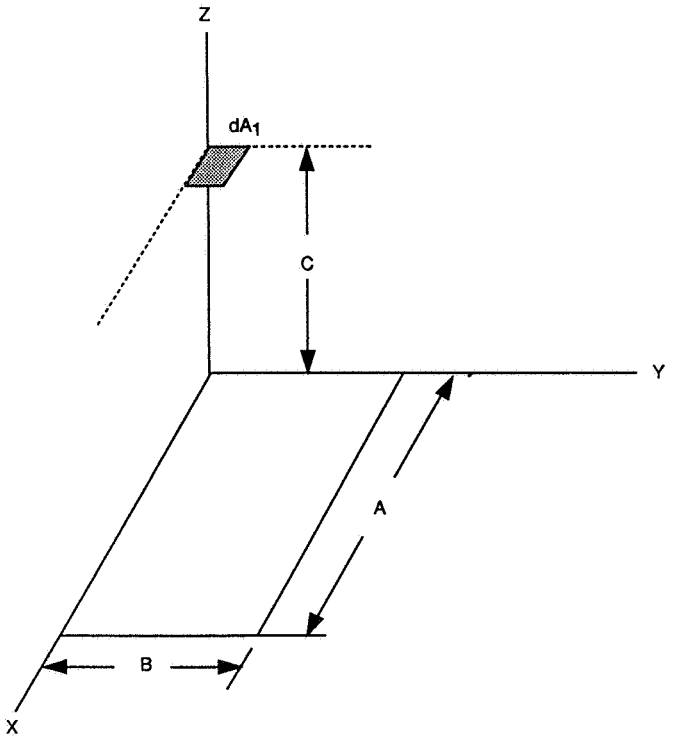


Figure 3. Geometry and coordinate system for Configuration 1.

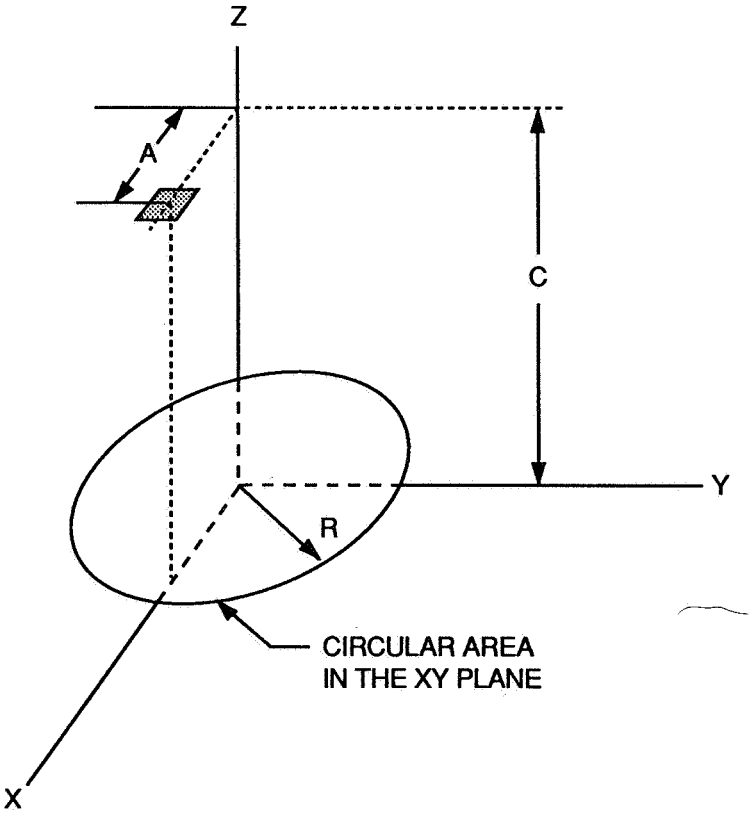


Figure 4. Geometry and coordinate system for Configuration 2.

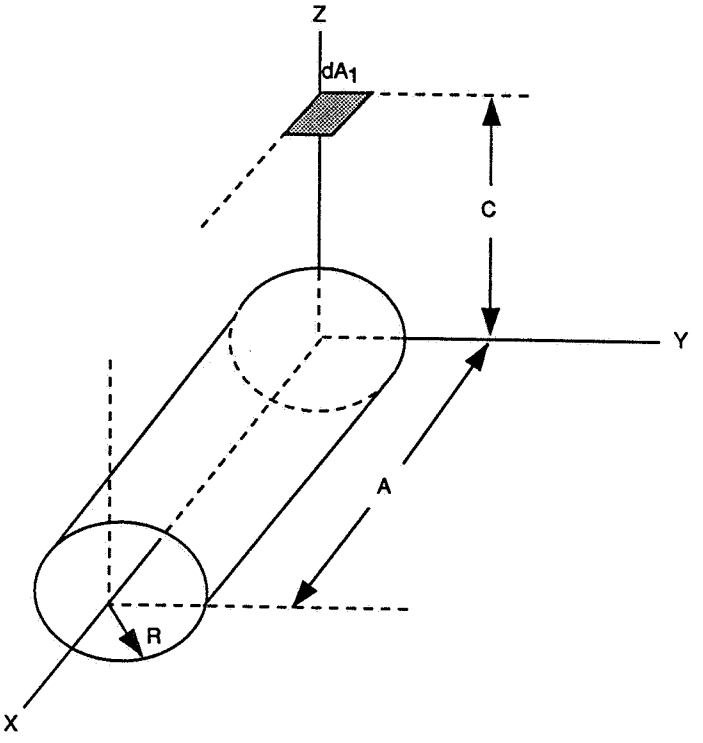


Figure 5. Geometry and coordinate system for Configuration 3.

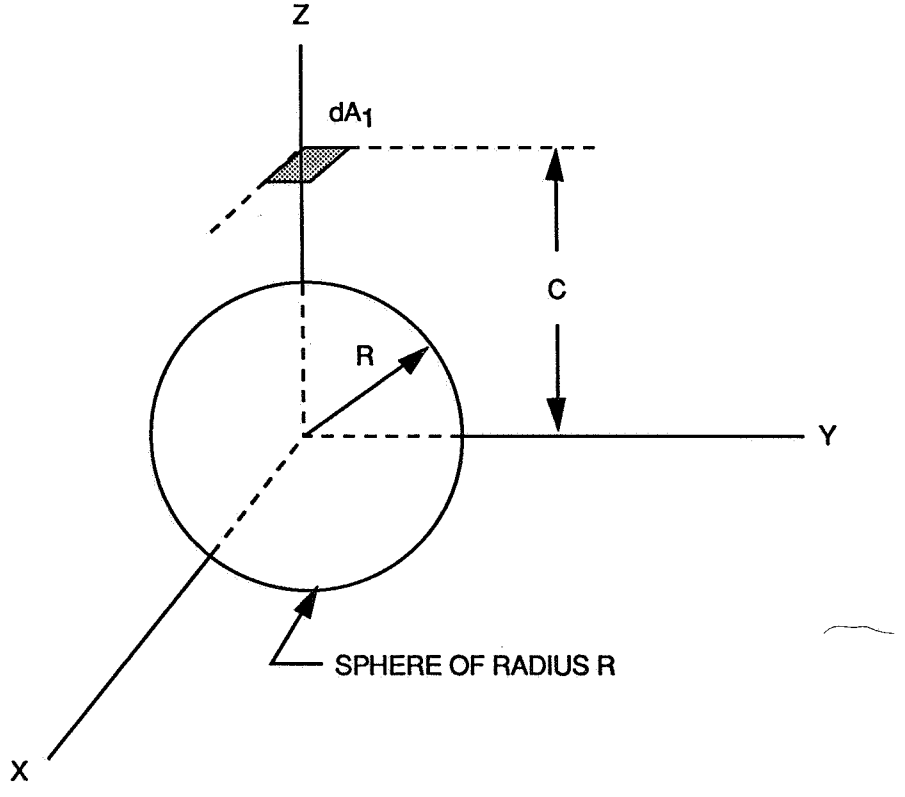


Figure 6. Geometry and coordinate system for Configuration 4.

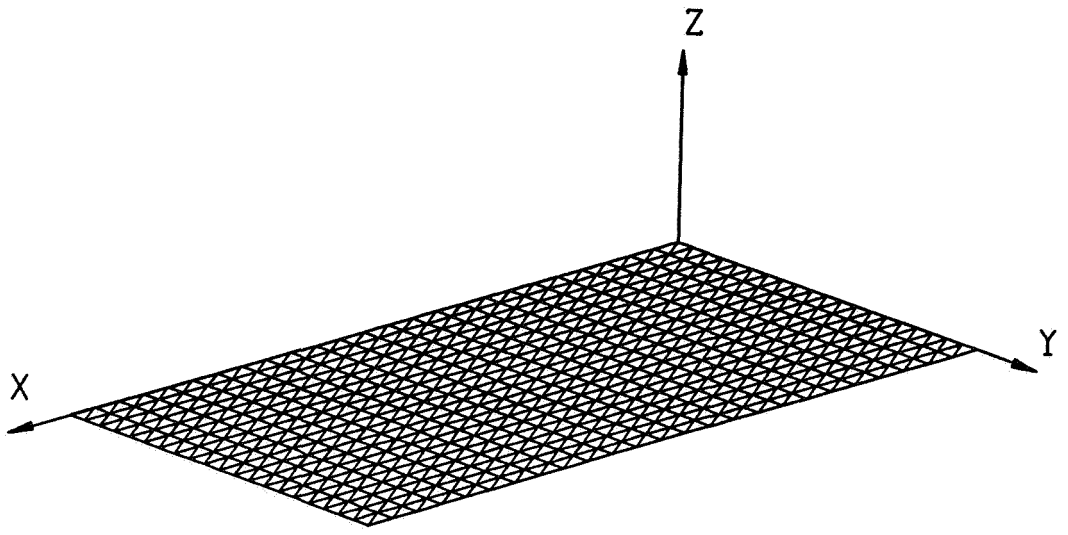


Figure 7. FTE grid for a rectangle (A/B = 2).

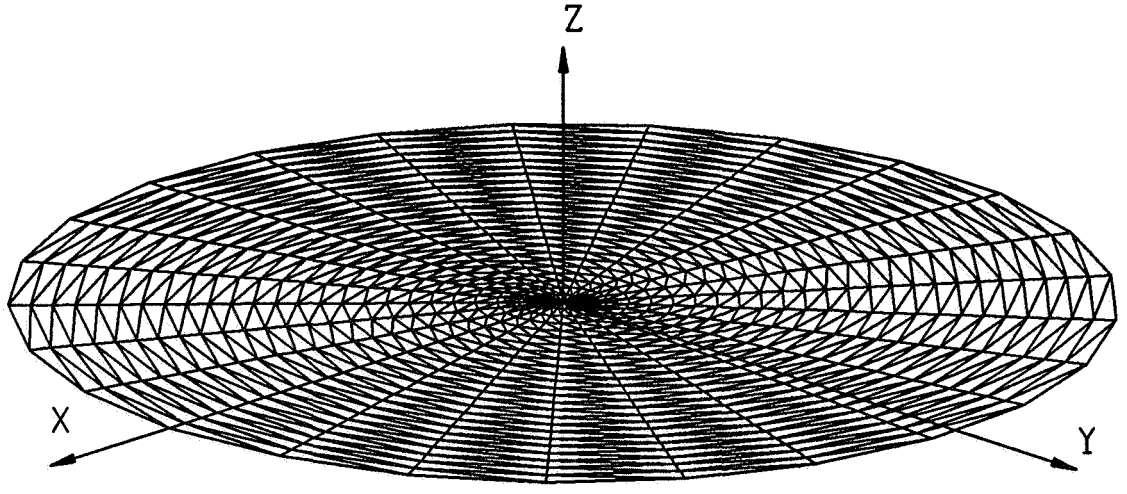


Figure 8. FTE grid for a circular area.

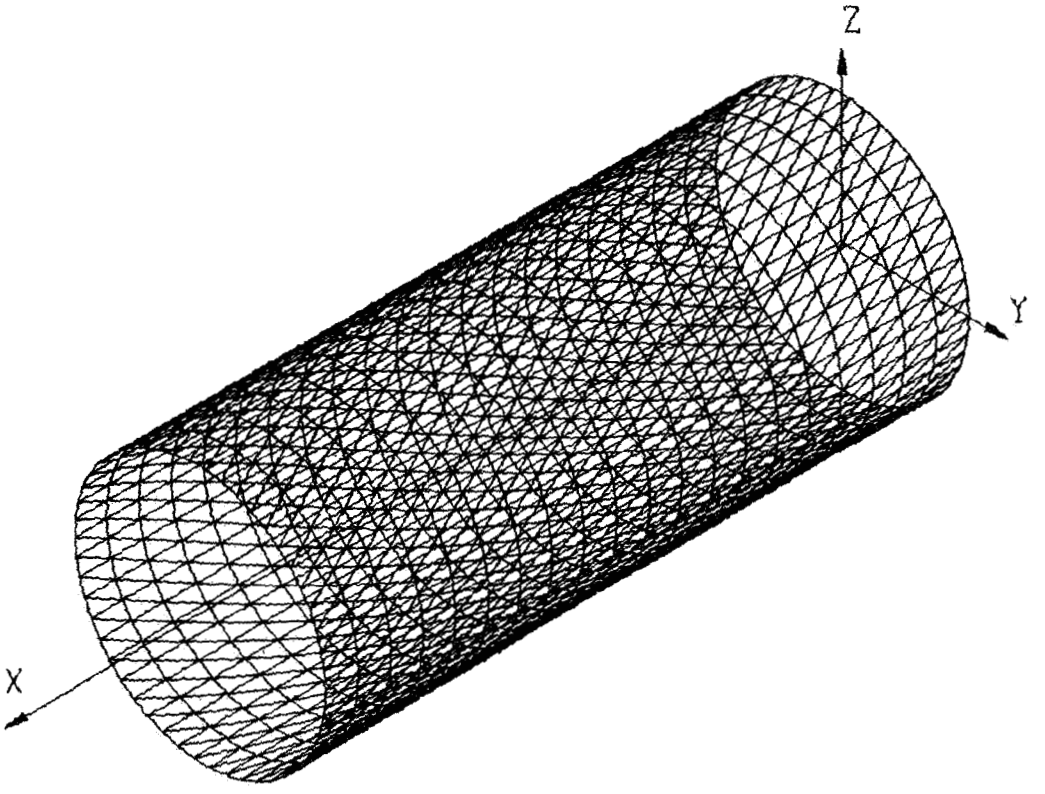


Figure 9. FTE grid for a cylinder (A/R ≈ 5).

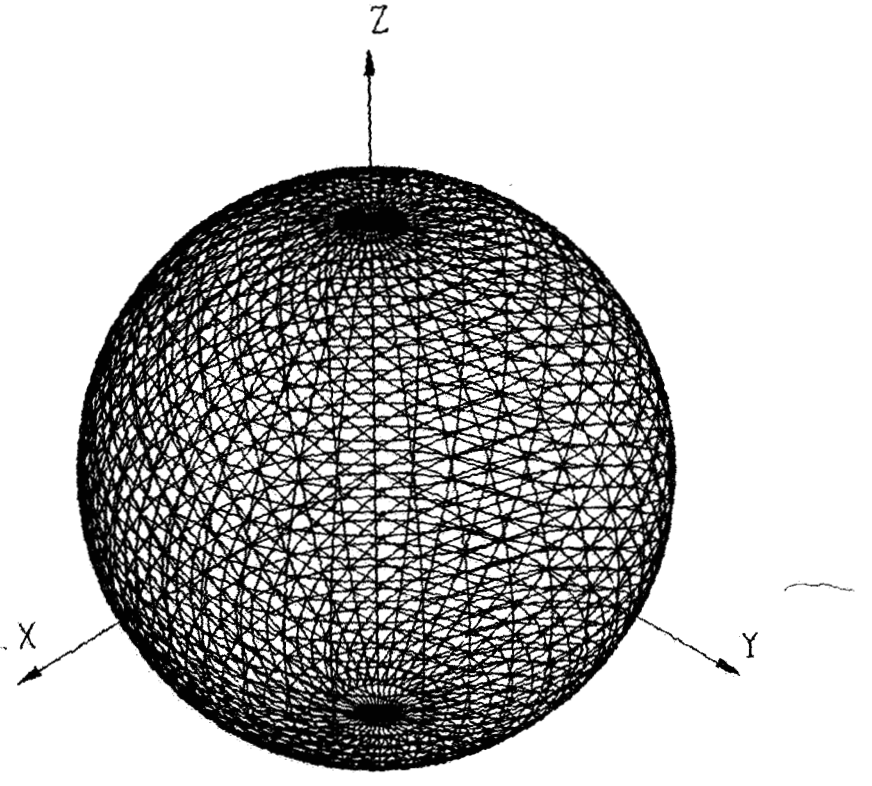


Figure 10. FTE grid for a sphere.

INTEGRATED GEOMETRY AND GRID GENERATION SYSTEM FOR COMPLEX CONFIGURATIONS

Vedat Akdag*, Armin Wulf*
Control Data Corporation
Irvine, CA 92714

SUMMARY

A grid generation system has been developed that enables grid generation for complex configurations. The system called ICEM/CFD is described and its role in computational fluid dynamics (CFD) applications is presented. The capabilities of the system include full computer aided design (CAD), grid generation on the actual CAD geometry definition using robust surface projection algorithms, interfacing easily with known CAD packages through common file formats for geometry transfer, grid quality evaluation of the volume grid, coupling boundary condition set-up for block faces with grid topology generation, multi-block grid generation with or without point continuity and block to block interface requirement, and generating grid files directly compatible with known flow solvers. The interactive and integrated approach to the problem of computational grid generation not only substantially reduces manpower time but also increases the flexibility of later grid modifications and enhancements which is required in an environment where CFD is integrated into a product design cycle.

INTRODUCTION

The basic techniques in numerical grid generation involve geometric modeling of an object and generating a three-dimensional grid of points that surround the object. In recent years, major advances have been made in flow solvers and visualization software for post processing. The major pre-processing task which is grid generation has not benefitted from the technology gains of the recent past.

The state of the art of Computational Fluid Dynamics has taken rapid strides in recent years with the development and application of unified, robust and efficient methods. Low speed subsonic flows to hypersonic flows around various configurations from internal flows to external flows with various flow physics can be calculated using Navier-Stokes Computational Fluid Dynamics codes. Recent codes avaliable in the industry are constructed using a synergistic approach of many solution methodologies. A multizonal structured grid can be employed to treat complex geometric topologies with ease.

---

* Consultant

The area of computational grid definition in CFD continues to require a high level of time and manpower and therefore has become widely recognized as a primary bottleneck in CFD analysis. The problem arises from the fact that grid generation takes on a more difficult character with increasing complexity in the geometry. Given the computational grid, today's engineers can simulate fluid flow around very complicated geometries. Efforts in geometry manipulation, surface grid generation and volume grid generation for a typical research project become primary schedule drivers. The result is an unacceptably long turnaround time for complex CFD problems. Previously, a typical pre-processing task from geometry manipulation to actual computational grid generation is shown in Figure 1.

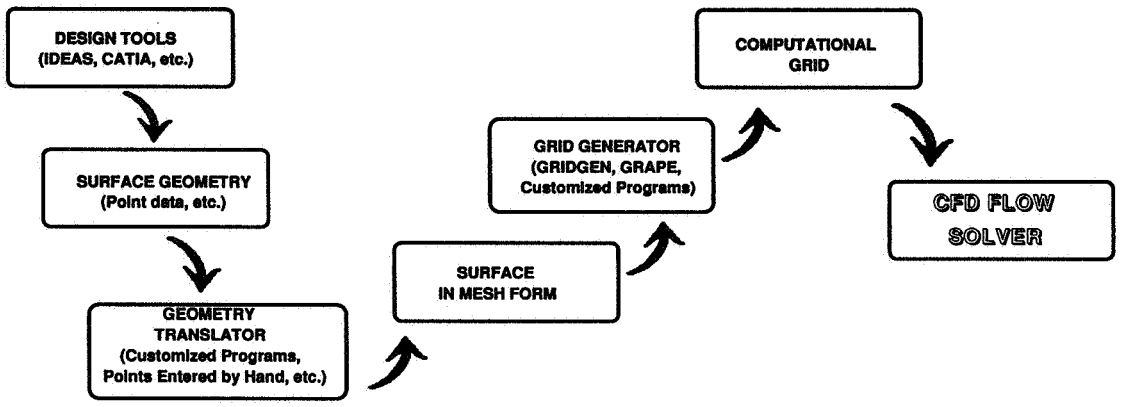


Figure 1: Traditional CFD Grid Generation

Despite considerable attention by researchers in recent years, geometry definition and grid generation still consumes a fair amount of time in CFD analysis cycle time. For complex configurations, this cumbersome procedure can take up to 80% of the effort spent on initial CFD analysis tasks (Figure 2).

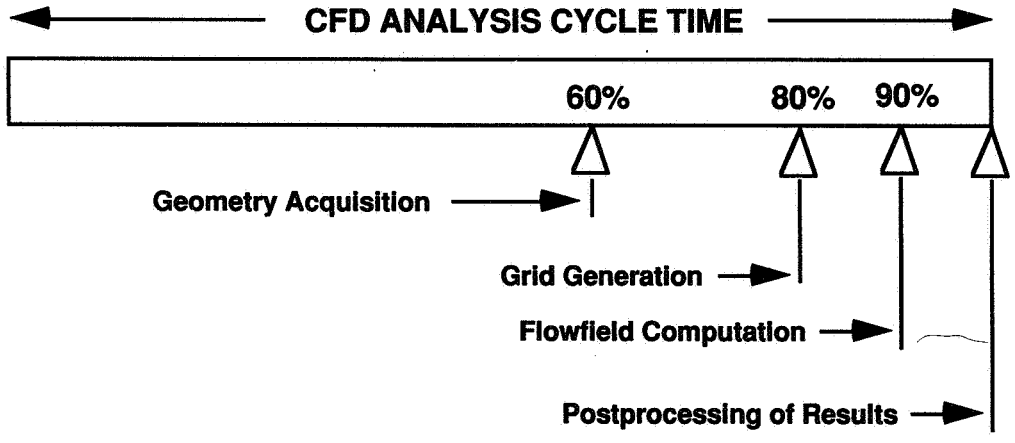


Figure 2: Traditional CFD Analysis Cycle

In many cases surface points are generated by custom programs or entered into a file for grid generators by hand. This can produce unsatisfactory results because the actual geometry definition will be altered as the computational grid is generated. Currently the majority of CFD application engineers do not have the capability of accepting CAD models in standards used by the industry (i.e. IGES CAD geometry definition). This can cause delays in schedules when a design change occurs and CFD analysis cannot be completed in a timely manner. To overcome this problem and to insure the quality of analysis, an integrated CAD and Grid generation package is needed. This approach can shorten the pre-processing procedure by a considerable amount (Figure 3).

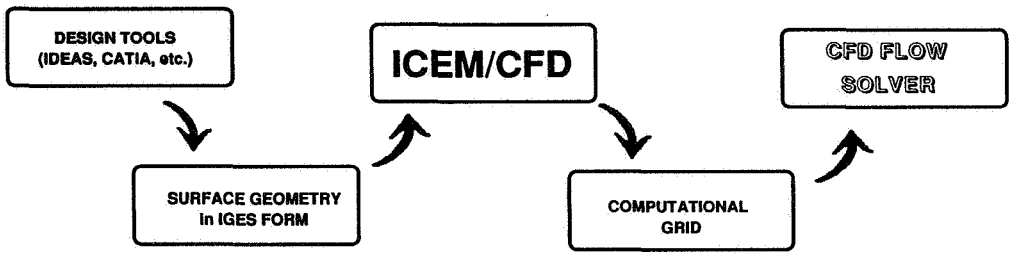


Figure 3: CFD Grid Generation with ICEM/CFD

CAD geometry can be taken from design tools and decomposed into meshable pieces while the association between the computational grid and the CAD model are maintained. This computational grid then can be fed directly into CFD flow codes. The main criteria used to select the grid generator are summarized below.[1]

- Minimize duplicate model generation
- Use direct path from/to master CAD model
- Robust algorithms that generate grids quickly
- Body fitted quality grids on master or deformed model
- Assured grid accuracy for Navier-Stokes analysis
- Reflect changes to master model quickly
- Managed CFD environment integrated into product design cycle

It is obvious that considering the recent advancements in the computer hardware and software technology, the amount of computer flow simulation data that will go into future aircraft design should greatly increase, as should the overall impact of CFD on design process. CFD analysis will become one of primary functions in product design. Today many companies are moving towards concurrent engineering concepts (Figure 4).

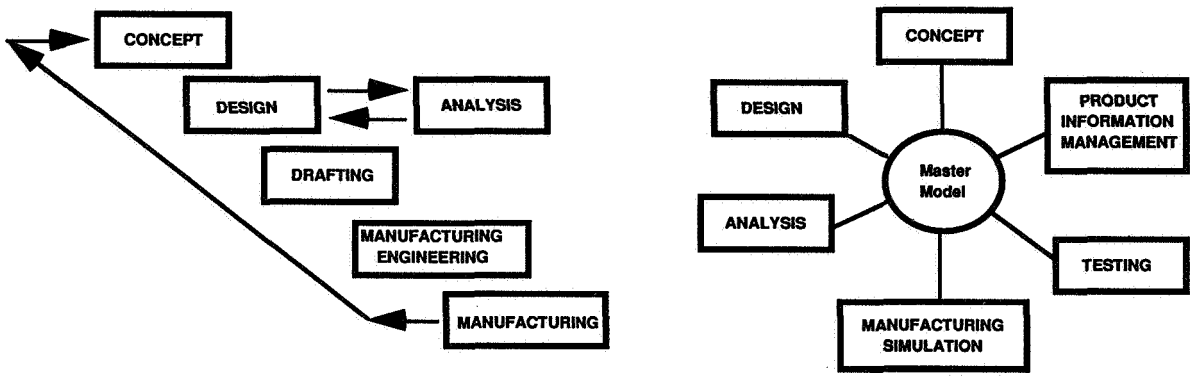


Figure 4: Product Development, Traditional versus Concurrent Engineering

Today, many companies achieve this goal through standardization on a CAD system for conceptual design, detailed design, analysis and manufacturing where master model concept is maintained throughout the process of getting the product to the customer. The change from traditional product development to a concurrent engineering approach requires a master model concept. The master model is used by different departments involved in model analysis as a means of transferring results from one department to another (Figure 5). The associated results based on the master model transferred between departments. In cases where the chosen CAD system is inadequate for performing computational grid generation for analysis, the requirement of data exchange (mainly geometry) between the grid generation tools and the CAD system becomes mandatory. CAD systems define geometry by using parametric curves and surfaces, whereas analysis environment represents a body using a set of discrete points. Acquiring continious discrete body points (surface in mesh form) from CAD geometry requires robust surface projection algorithms, where gaps between surfaces, overlaps, and discontinuities on the surface structures do not create problems.

Currently structured zonal grid generation softwares, operate on the surface mesh in point data format.[2,3,4,5]. They do not possess the advantages of ICEM/CFD's full CAD capability integrated with 3D grid generation functionality. There have been some efforts in utilizing CAD systems to enhance surface grid generation capabilities. This approach requires a considerable investment in time and money. The main problem is not in CAD capabilities but in the absence of adequate means for converting CAD data to CFD formats effectively [6]. Given an integrated tool like ICEM/CFD engineers can devote more time on the actual analysis of the flow solver results rather than tedious geometry acquisition and definition, and grid generation issues.

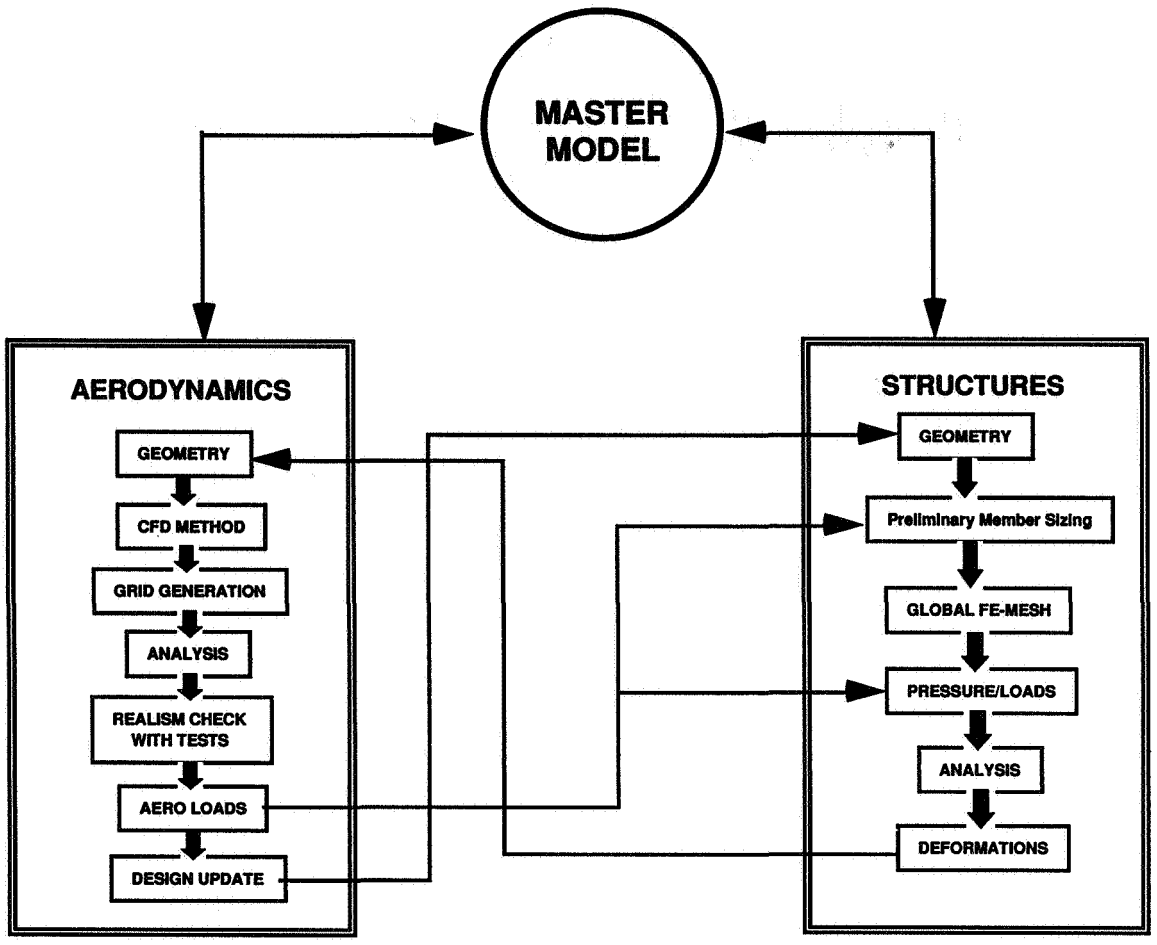


Figure 5: Integrated analysis environment into product design.

USE OF INTEGRATED GRID GENERATION TOOL

ICEM/CFD's excellent CAD capabilities support the creation of wireframes, 3D curves, regular surfaces, and multiple segment NURBS surfaces. It can generate and receive standard open CAD formats (i.e. IGES, SET, VDAFS, DXF) for data exchange with other CAD packages. After the vehicle geometry is read from other CAD packages, depending on the geometry configuration, a CFD application engineer can create blocking for completely new configurations with ease. An interactive user interface reduces the man-hours required to create, modify, and verify a grided configuration. It provides excellent tools for interactive topology generation for multiple block structured grid configurations. It can generate multi-block structured grids with or without point continuity and block to block interface requirement. When the point continuity required between blocks the system automatically checks the inter-block connectivity. Boundary conditions and inter-block connectivity information are generated and output automatically. The geometry can be represented using B-spline surface patches and the grid points on the surface are calculated by projection. Computational grid is not dictated by

surface patch structure. This allows the underlying geometry to be modified with no impact on the topology of the grid structure. Robust surface projection algorithms tolerate gaps, overlapping and sharp edges of surface geometry definitions. After the topology and connectivity information is generated, surface and volume grid generation is automatically performed. Computational grids required by viscous fluid flow solvers can be generated using ICEM/CFD, since it embodies the capability of generating grid points in double precision with grid relaxation and adequate bunching algorithms.

ICEM/CFD APPLICATIONS

A grid generation system is measured not only by its algorithms, but also by the multi-block and complex grids that it can generate. One also has to consider the CFD application process as a part of the product design cycle, where geometries of complex configurations are expected to be analyzed. Current ICEM/CFD applications vary from internal flow grids to external flow applications that cover aerodynamic and aeroheating applications of automotive and aerospace configurations. The blocking strategy for an application may be approached in several different ways. The user must make grid topology decisions based on his prior experience and flow solver capabilities. These decisions can be made without being limited to the grid generation tools since very complicated geometries can be grided using ICEM/CFD.

Multibody Configuration: To demonstrate the ICEM/CFD's capability in handling complex configurations, a multi-body geometry system, consisting of a core body with two strap on boosters of a launch vehicle is generated. The main body consists of a spherical nose cap, a conical section, a bulbous cylinder, and a backward facing step (boattail) followed by a long cylinder. The strap-on solid rocket boosters exhibit geometry similarities with the main body (Figure 6). In order to save computational time for the flow field calculations the front section of the geometry is grided using a single zone structure where the computational grid behind the boattail is constructed using five blocks. To generate the 3 dimensional grid around the nose section , a two dimensional surface grid is constructed. Utilizing the tools in ICEM/CFD, the 2D surface grid is rotated around the axis of symmetry to generate the 3D dimensional grid. The edge entities required for the 3D grid are constructed automatically, and the connectivity information between the block face entities is checked in the background by ICEM/CFD. The total flowfield calculation can be carried out by interpolating the flowfield between the single block nose grid and the multi-block grid structures.

Personal Launch System: A multiblock grid configuration of the PLS (Personnel Launch System) / HL-20 transatmospheric vehicle was generated using ICEM/CFD. The geometry of the PLS (Figure 5) was constructed by NASA Langley researchers using SMART, a solid modelling package developed at NASA Langley for Silicon Graphics Iris 4D workstations. Since SMART output was available in the form of PATRAN neutral files, the PLS geometry was converted into IGES format using PATRAN. The surface geometry, as B-spline surfaces was then read into ICEM. An orange peel type C grid was constructed around the nose area for improved stability of the flow solver at high Mach number and high angle of attack flight conditions. The construction of O type grid blocks around the wing and the vertical tail allowed grid point clustering around these surfaces (Figure 7).

B-2 like aircraft: Figure 8 shows a B-2 like aircraft, where a single block grid structure is used to grid the upper portion of the airplane. The surface grid points are projected on the underlying NURBS geometry. The tools available in ICEM/CFD, provide visualization of the face grid structure of the computational blocks before the full 3D grid is calculated.

Turbine Blade: The turbine blade configuration is grided using multi-block grid topology with an O-type grid around the near region of the blade where H-type grids fill the regions between consecutive blades. During the grid point clustering along block edges, the grid distribution on the block faces can be displayed for fine tuning. The output of this grid structure can be written in various flow solver formats currently used in the industry. Boundary condition on the block faces can also be attached to the grid points for output.

FUTURE WORK

CFD will continue to be applied to numerous interesting and challenging problems that will push the state of the art in the discipline. A considerable number of these problems exist for which CFD will be beneficial and sometimes absolutely necessary.

One of these challenges is to establish a new computational environment that is intended to support a multi-disciplinary approach to the design of advanced aerospace configurations. Designers of such vehicles can no longer rely on single discipline research tools to identify and understand undesirable design features. A multi-disciplinary design environment that integrates fluid dynamics, aeroelasticity and structural dynamics, electromagnetics and controls can minimize or eliminate additional design iterations taken during the single discipline design process. Most of these disciplines require various types of geometry manipulation and structured or unstructured computational grid generation capability.

The recent advancement in integrated geometry and grid packages, such as ICEM/CFD, will help the integration process for the multi-disciplinary computational environment to achieve synergism in design of advanced vehicles. ICEM/CFD can serve as the center of the master model concept for geometry, engineering analysis, detailed design, and manufacturing. Since integrated packages of geometry and grid generation, such as ICEM/CFD have become available to engineers, more time can be spent on the actual analysis of the calculations rather than tedious grid generation.

ACKNOWLEDGMENTS

The authors wish to acknowledge the efforts of Control Data, ICEM Systems and Aircraft Division of Aerospatiale, France in development of ICEM/CFD and related software.

REFERENCES

1. Rainsberger, R., Wulf, A., *Priorities for CFD-Grid Generation in a Master Model Environment*, 1991 UTECA Conference, Springfield, MA.

2. Visich, M., Boyd, C. N., Widhopf, G. F., Coons, R. E., S-C Huang, Oliphant, P. H., Staal, D. J., Than, P. T., *Advanced Interactive Grid Generation Using RAMBO-4G*, Paper AIAA-91-079, 29th Aerospace Sciences Meeting, January 7-10, 1991/Reno, Nevada

3. Steinbrenner, J. P., Chawner, J. R., and Fouts, C. L., *The GRIDGEN 3D Multiple Block Grid Generation System*, WRDC-TR-90-3022, Vol. I, II, July 1990.

4. Gatzke, T., LaBozzetta, W., Finfrock, G., Johnson, J., Romer, W., *MACGS: A zonal grid generation system for complex aero-propulsion configurations*, Paper AIAA-91-2156, 29th Aerospace Sciences Meeting, January 7-10, 1991/Reno, Nevada

5. Sorenson, R. L., McCann, K. M., *A method for interactive specification of multiple-block topologies*, Paper AIAA-91-0147,29th Aerospace Sciences Meeting, January 7-10, 1991/Reno, Nevada

6. Ching-Chung Luh, R., Pierce, L. E., Yip, D., *Interactive Surface Grid Generation,* AIAA-91-0796, 29th Aerospace Sciences Meeting, January 7-10, 1991/Reno, Nevada

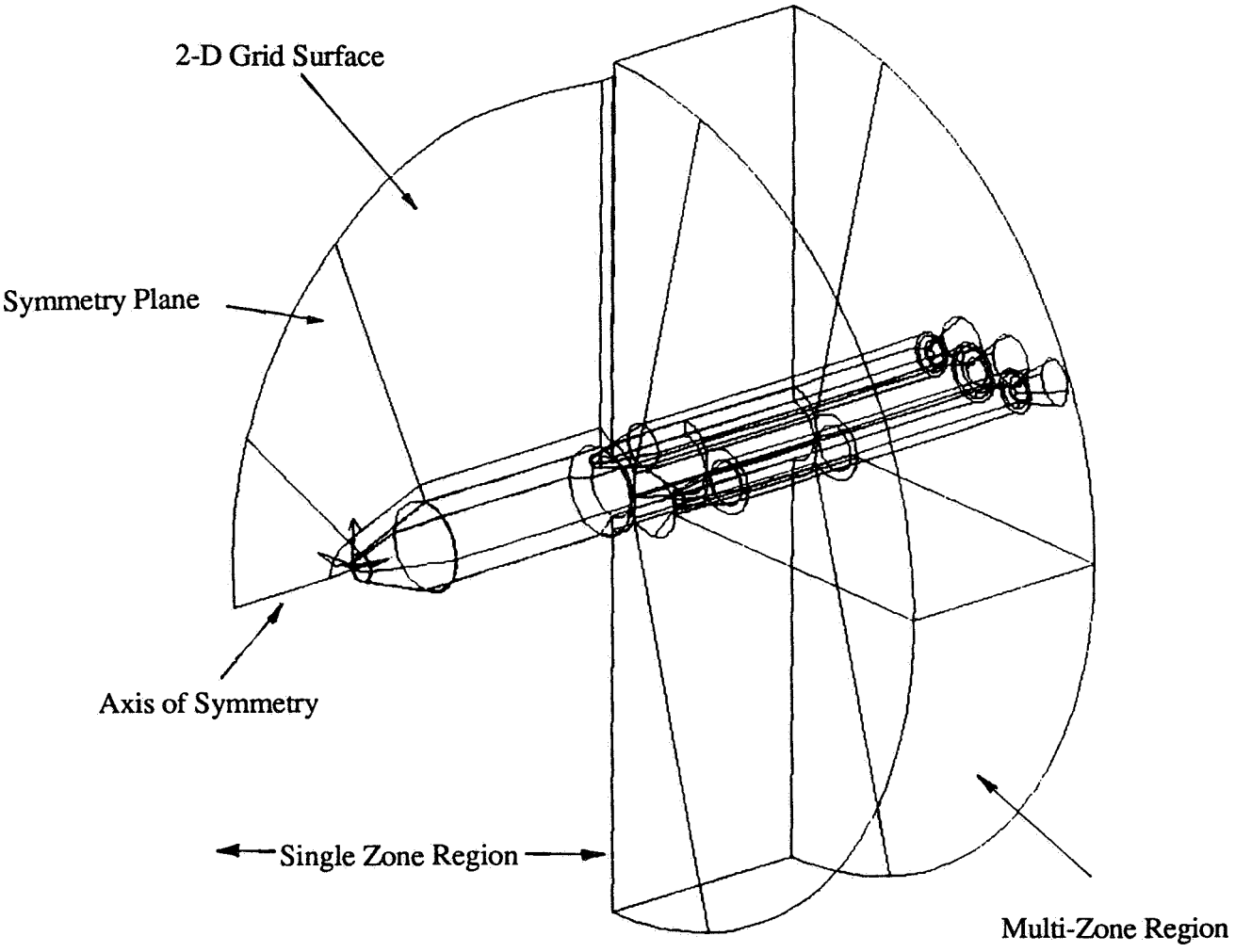
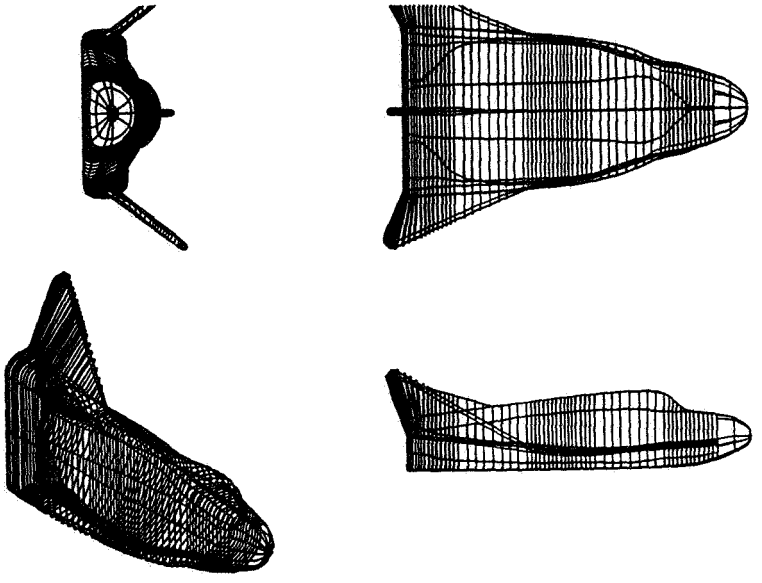

2-D Grid Surface
Symmetry Plane
Axis of Symmetry
Single Zone Region
Multi-Zone Region

Figure 6



9418 Surface Grid Points, 245282 Flow Field Grid Points

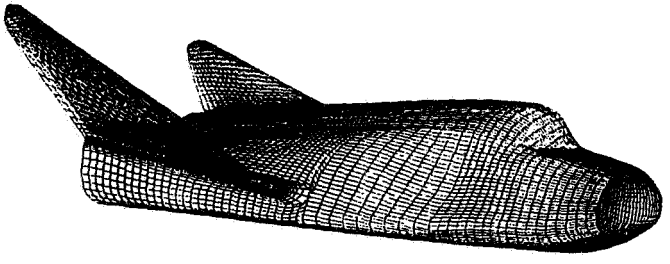


Figure 7

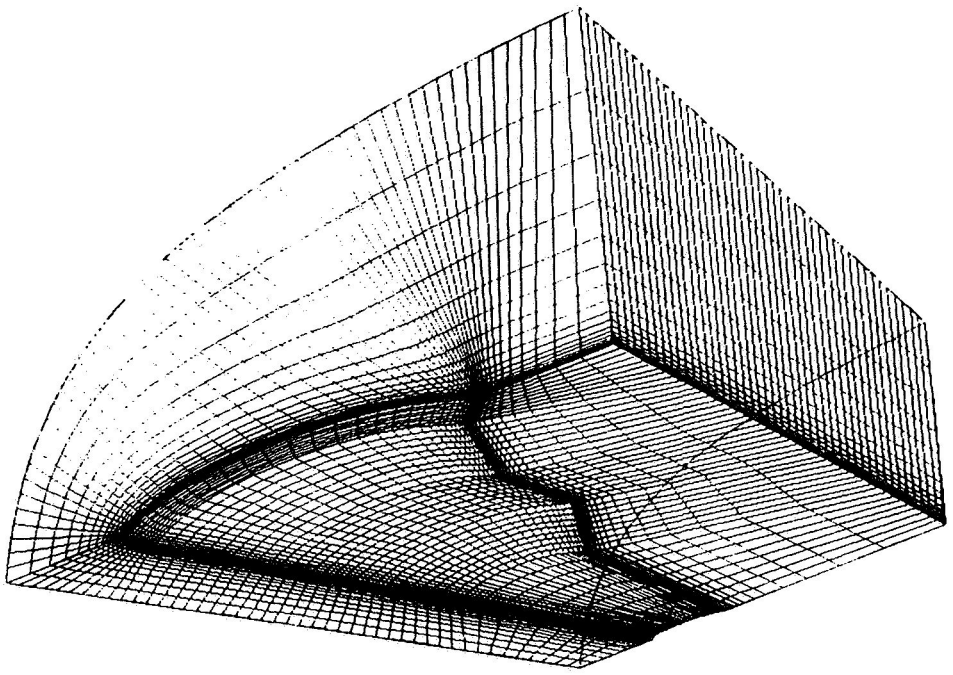


Figure 8

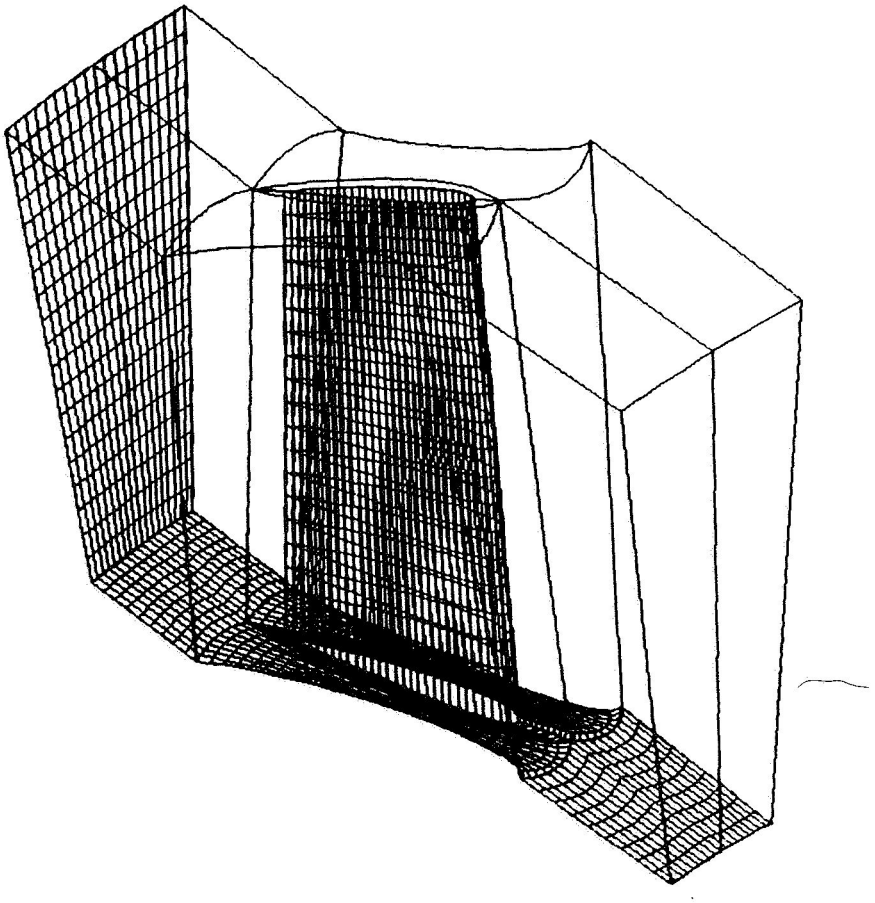


Figure 9

GRIDMAN: A GRID MANIPULATION SYSTEM

Peter R. Eiseman and Zhu Wang
Program Development Corporation
300 Hamilton Avenue, Suite 409
White Plains, NY 10601

INTRODUCTION

GridMan is an interactive grid manipulation system. It operates on grids to produce new grids which conform to user demands. The input grids are not constrained to come from any particular source. They may be generated by algebraic methods, elliptic methods, hyperbolic methods, parabolic methods, or some combination of methods. The methods are included in the various available structured grid generation codes. These codes perform the basic assembly function for the various elements of the initial grid. For block structured grids, the assembly can be quite complex due to a large number of block corners, edges, and faces for which various connections and orientations must be properly identified. The grid generation codes are distinguished among themselves by their balance between interactive and automatic actions and by their modest variations in control. The basic form of **GridMan** provides a much more substantial level of grid control and will take its input from any of the structured grid generation codes. The communication link to the outside codes is a data file which contains the grid or section of grid.

The grid, whether imported or not, is like a sculptor's clay and the **GridMan** user is like the sculptor who employs his tools to transform his clay into a masterpiece. Like the actions of the sculptor, the manipulations are dynamic. They appear visually as real time grid deformations that occur in dynamic response to local grid controls. As the user interactively applies the controls in various regions, the grid alterations successively and dynamically change the character and shape of the grid.

The typical grid alterations include the injection or removal of pointwise **clusters**, the **angle changes** between coordinate curves, and the free-form **geometric modeling** of region boundaries. In the **geometric modeling**, the actual bodies in the field are changed to accommodate variations in the design. Since the grid format does not change, successive numerical simulations can be set up rather quickly to show the effects of altered body geometry. The pointwise **clusters** are usually injected at sections along boundaries where attached boundary layers are expected and are removed in regions away from bodies where clustering is not needed and is wasteful. More generally, the clusters can be placed at any desired location in the field so that a user can, for example, interactively adapt the grid to the results of a previous simulation in order to gain accuracy in a subsequent simulation. The **angle changes** are primarily applied at the boundaries to produce coordinate orthogonality. As with the clustering, the creation of grid orthogonality is not restricted to the boundary locations. It can be established at any location.

VERSION SB3000

Version **sb3000** is the basic system which operates on **single block three dimensional grids** regardless of the source of blocks. In a very direct sense, it can be applied to a multiblock grid in a **roving** manner whereby a block section can be taken,

operated upon, and then returned to its original place. The block section can be a whole block, a sub-block, or can cut across a number of distinct blocks.

The three coordinate directions in the block are identified by the colors red, green, and blue. The local controls appear in the form of a sparse control net which is distinguished by the color yellow, by fatter lines than appear in the grid, and by a spacing discontinuity adjacent to the boundaries. The number of control points in the red, green, and blue directions are chosen by the user and are determined by how much control the user wishes to exercise. While the decision does not depend upon the number of grid points, it is assumed to be much less in number for each direction.

With the grid block placed in the transitional data file, the user calls **GridMan** into action. He is then prompted to choose the type of session among the possibilities of demonstration cases, restarts, or new grid files. Once the choice is made, the next prompt is to specify the number of control points for the three directions. After those numbers are given, the **GridMan** window comes into view. The window is carved into two primary spaces. The left hand side is the largest sector and is occupied by the three-dimensional image of the grid, control net, and block frame. The right hand side is a strip that contains a header identifying the program name below which are various combinations of graphic buttons.

The Buttons

The buttons represent the various actions for changing the view of the three dimensional object, altering the sections of the volume to be seen, selecting a control point for motion, determining which type of motion, asking for help, and quitting the program. While the buttons are labeled to identify their purpose, that labeling is quite brief due to the allotted screen space. To compensate for this space limitation, as the cursor moves over a button, the header on the strip changes to display a paragraph that gives a full description of the button's function. This process is dynamic and does not require the user to request help or push that button. It merely supplies information which can add clarity should there be some confusion. Its position at the header is sufficiently removed to not be disturbing to an experienced user who does not need such dynamic help. To push a button, the user pushes down and releases the left mouse key. In response, the button graphic displays a depressed key. A repeat click on the button or other action will turn off its function and the graphic button image will appear as an undepressed key. In this way, the user will immediately know the status of which functions are on and off.

The Scroll Bar

When a button is pushed for a change of view or a control point motion, a scroll bar is activated. The scroll bar appears below the three dimensional object on the left hand portion of the window. Upon the push of a button, the title on the scroll bar changes to reflect the mode indicated by the button. When the cursor is brought onto the scroll bar itself, the scroll bar title disappears and is replaced by a ruler that is numbered. The numbers appear at each end to define the range and at the current ruler location indicated by a red vertical arrow that points upward. By placing the cursor on top of the arrow, it can be dragged to a new location on the ruler. The drag is executed by depressing the left mouse key, rolling the mouse until the arrow is at the desired ruler location, and then undepressing the left mouse key. The result of the action appears as soon as the mouse key is undepressed.

While the dragging of the vertical arrow represents a discrete response, a continuous response is available by using the small buttons which appear at each end of the scroll bar. The innermost buttons are labeled by triangles that point away from the bar in their respective positive and negative directions. By depressing one of these buttons, there is a continuous motion in the corresponding positive or negative direction. This motion is seen in the scroll bar by the continuously changing position of the vertical arrow. The smoothly changing values from the scroll bar are fed to the program to perform the smooth changes that correspond to the currently chosen operation. This motion is stopped when the scroll bar button is released. The speed of the operation can also be changed to accommodate distinct needs. A fast motion may first be desired to get points or views quickly into some sort of general position and then be followed by a slower motion to more accurately alter such position. This option for speed adjustment is offered by the outermost buttons on the scroll bar. Successive clicks on the left outermost button will successively slow down the motion. The slow down is indicated by a down arrow on the button. Likewise, the right outermost button is labeled with an up arrow and will cause an increase in speed with each click.

While the scroll bar offers both discrete and continuous motion with speed control, the range of its operation has been given by a ruler of a specified initial length. This length can also be changed by using the middle buttons of the set of three which appear on each end of the scroll bar. The other two have been seen to define the continuous motion and its speed. The middle button on the left is labeled by two triangles that point towards themselves. Successive clicks on it will shrink the ruler length. Likewise, the middle button on the right hand side will expand the ruler length. It is labeled with two triangles that point away from each other.

The Three Dimensional Display

The three-dimensional display consists of up to four constitutive elements. These are (1) the Cartesian x, y, and z axes of the world space, (2) the grid block frame which shows the block edges, (3) the grid, and (4) the control net. The top three buttons allow the user to display any combination of grid, net and frame. The flexibility offered here is that one can simultaneously view these three items to take note of the relative positions of each or can remove some of them to more clearly inspect a particular part of interest.

The Cartesian x, y, and z axes provide the frame of reference for changes in view. They define the space in which the object of our actions is being taken. The block frame gives a view of the entire block volume in a manner which does not obstruct the view of the current elements within that volume. The frame is built up from the edges of the grid block. Aside from the edges themselves, a plot of the boundary grid points adjacent to those edges can appear. These points are connected to their corresponding edge points and to each other in their successive order. In addition, there is a meaning to their appearance or non-appearance. When the first layer of grid on a face is in view, **GridMan** considers that boundary to be specified on a grid point by grid point basis. Otherwise, when only the face edges appear, **GridMan** generates the face from the corresponding sheet of boundary control points. For this selection, there is a toggle button (entitled **Toggle boundary Form**) to switch between the specified boundary and the one which is opened for free-form manipulation. There is also a toggle button entitled **Frame** to switch on and off the display of the frame and world space axis combination.

The grid is displayed by the graph of one sheet at a time. This appears in the chosen color code. In particular, a constant value in the red direction is plotted as a red coordinate sheet; a constant value of green, a green sheet; and a constant value of blue, a blue sheet.

The view of the grid is toggled on and off with the key entitled **View Grid.** There is also the option to display additional images of the grid sheet that are reflected about the origin in the respective Cartesian world space directions. The mirror images are activated by the three toggle buttons entitled **Mirror X, Y** , and **Z** . These appear immediately below **View Grid.** In addition, there is a button entitled **Smooth** which smooths the grid lines to remove the stair case effect of plotting lines that are not aligned with the rows or columns of pixels on the screen. A final grid display button is provided to examine the grid surface for its geometric features. It is entitled **Render** and shows the grid surface as a solidly colored object under light sources. The **L key** on the key board returns the user to the previous mode.

Like the grid, the control net is displayed by a graph of one sheet at a time. That sheet appears with fatter lines and is displayed in the color yellow. Since the the actions are executed by control point movements, it is important to also display the effect of such motion relative to the current net sheet. A control point is identified within a sheet by the intersection of the two other net sheets. The intersection is identified by a change from the color yellow to that of the intersecting sheet direction and by still even fatter lines. For example, if the current net sheet corresponds to the grids plotted in blue, then the intersecting ones are represented in red and green respectively. Thus, the red intersection appears in red and the green in green. In addition, this intersection represents a point on a control net curve that intersects the current net sheet. To witness the most immediate effect of a motion without too much screen clutter, this net curve is plotted only up to the first control net point on either side of the current net. On the boundaries, there is only one side for this plot while in the interior there are two. It is distinguished by the color turquois. The net sheet display is toggled on and off by the button entitled **Net.** Unlike the grid, the net is not given the mirror or render option. Because of the fatness of the lines, the smoothing option is not needed.

The last display button is entitled **Depth.** It is used to clarify graphically which part of an object is in front of another object or part of an object.This is necessary to see items as they naturally appear in three-dimensions. Otherwise, something in the inside could be plotted over something on the outside. This operation is known as depth queuing and is well known. In the present case of Cartesian axes, the frame, the net, and the grid; the improvements from the depth queuing are modest. With solidly colored objects, the improvement is much more substantial.

Change of View

Given a three-dimensional object in the display, the position from which it is viewed may not be appropriate and must be changed for a better inspection. This is accomplished with the buttons under the subheading **Change View.** The basic rigid body motions are attached to the world space coordinate axes. The respective rotations are about the point center of mass of the entire grid block. This is computed in a pointwise rather than volumetric sense so that the emphasis is given to the point densities that are viewed. The center components are then just the separate averages over the total number of block points in the respective x, y and z directions. The rotations appear as a row of three buttons entitled **Rotate X** , **Y** , and **Z** . The respective translations are given by the next row of three which are entitled **Transl X, Y** , and **Z** . Relative to the point center of mass, the graphic image can be expanded or contracted with a zooming operation. The operation is activated with the **Zoom** button. Once activated for rotate, translate, or zoom, the scroll bar is dynamically labeled to reflect the choice. The user then moves the cursor to the scroll bar and performs the labeled task. As mentioned in the scroll bar discussion, there are the

options for continuous motion with adjustable speed and range as well as the discrete motion by use of the ruler position arrow.

While the center of gravity position generally keeps the graphic object in view for the rotate and zoom operations, the zoom to inspect a particular part of the grid may fall off of the screen. This would then require the user to apply a translate to bring it back into view. Rather than be forced into this situation, an alternative operation is offered. The operation is entitled Focus. When the **Focus** button is pushed, the user brings the cursor to a position on the screen, depresses the left mouse key, drags the cursor in a diagonal manner, and releases the mouse key. This dynamically puts a rectangular box in screen space about the region to be expanded. When the left mouse button is released, the expanded image appears. While such expansions can be done any number of times in succession, there is an undo command that allows the user to step back up to eight times. Beyond eight, the previous stages are not stored. Each step back to a previous stage is accomplished by clicking on the **Un-do** button. The last button is entitled **Reset Graphics** and brings back the original view of the object from the very beginning of the session.

The Selection of Net and Grid Sheet Displays

The display of the net and grid is given by sheets that can be seen in detail without the clutter that results should a three dimensional volume be displayed. The sheets are just the coordinate or natural net surfaces that are given by holding fixed one of the three curvilinear coordinates. The three dimensional volumetric view then comes from the dynamics of shifting through the collection of sheets.

The motion of the net sheet is given by pressing the key entitled **Scroll Net Sheet**. With each click of this button, the net sheet is scrolled in the current direction in the current plus or minus orientation. This scrolling action will go up to a boundary, will go to the opposite boundary, and will continue back towards the first boundary. Along with the scroll of the net, the grid display will also scroll should it be in view. This occurs because **GridMan** automatically chooses to display the closest grid sheet in the same direction. This automatic selection is made because the grid sheet closest to the net reflects most closely the character of the net as compared to sheets that are further away. Thus, when a control point is to be moved, its major effect on the grid can be witnessed most directly in a dynamic manner.

To change directions or the orientation in a given direction, there is a row of buttons just below the **Scroll Net Sheet** button. These are labeled **I** , **J** , **K** , and +/- . In keeping with the red, green, blue convention there is a one-to-one correspondence of (I, J, K) with (red, green, blue). In fact, the I appears in red; the J, in green; and the K, in blue. As an alternative, the scroll in the current direction can also be executed with the **I** , **J**, or **K** button with the same color as the grid display should it be present. When the color is different, the sheet is flipped into one with the color of the pressed button. The sheet which appears from the flip is the one which passes through the current control point that was highlighted on the previous sheet. That same point is also highlighted on the flipped sheet which indicates that no scrolling action occurred on the press of the given button. Further presses on the same button or on the **Scroll Net Sheet** button will give the scrolling action. This can be continued any number of times. To reverse the orientation the plus/minus button +/- must be pressed.

Aside from the grid display as it appears in the scroll of the control net, there is a separate means to view it. This is important so that a user can view the entire volume of the

grid. This is accomplished with the press of the button entitled **Scroll Grid Sheet.** Once pressed, the scroll bar is labeled accordingly and the user then moves the cursor to the scroll bar to perform the operation. With this operation, the grid sheets in the current direction are scrolled from the original position. The motion is either discrete or continuous. In the continuous case, the view appears in a dynamic form like a movie. This temporal approach gives a real feel of how the grid behaves in a volumetric sense. This motion is independent of the net. Should the net be still in view, it will appear statically in its original position. To switch directions, the user must return to the buttons on the right hand side to change the net direction and then to scroll the grid again.

The Selection of a Control Point

A control point is identified by its displayed control net sheet and the highlighted intersection of the two sheets in the directions of the other colors. Within the yellow net sheet, the current control point position is where the two control point curves of different colors intersect. In the case of a blue direction, the displayed grid if it appears is blue and is near the mostly yellow net sheet. In the red direction (constant sheets of red), there is a red net curve through the current control point. Likewise, there is a green net curve in the green direction. The intersection of red and green identifies the current control point. In addition, the current point is identified geometrically with the net curve that extends transversely piercing the sheet. The extent of that curve is limited to connect the net points just above and below the current net point. It appears in a turquois color.

With the scrolling operation for net sheets, a sheet in which the point lies can be selected. The next operation is to move the highlights to the points that we wish to identify. This is achieved by scrolling within the current net sheet. The action is taken by clicking the buttons which appear under the sub-heading entitled **Choose Control Point**. Again there is a sequence of **I** , **J** , **K** , and +/- buttons. Like the previous buttons for the net sheet scrolling, the same functionality and color codes apply. The only distinction is that nothing happens if one presses a button of the same color as the current net sheet direction. This, however, is just common sense because there are only the other two colors left for the scrolling action within the net sheet. That scrolling action is exactly the same as with previous case of the net sheets themselves.

The Motion of a Control Point

Once a desired control net point is identified to be the current net point, there are two options provided for movement. It can be moved within the net sheet surface or along the transverse links to the control points just above and below the net sheet surface.

To move within the net sheet, the button entitled **Move On The Plane** must be pressed. The cursor then appears at the center of the screen and a movement of the mouse will cause a motion of the control point. In effect, the motion of the mouse is mapped into a corresponding motion of the control point in its net surface. Along with this control point motion, the three dimensional grid is dynamically altered. This dynamic is seen as a smooth continuous motion of the grid in real time with the control point. Because the closest grid sheet to the net is in view, the most prominent changes in the grid are seen in this dynamic fashion. To see the effect on the full three dimensional grid, the user must apply the scroll grid sheet options in the various directions. During the movement, there are several options that can be exercised. One is a switch between the quadrants about the original position of the current control point. This is activated by pressing the **right mouse** key. The other option is to adjust the mapping to accommodate an interchange of

the vertical and horizontal motions of the mouse. The is activated by a press of the **center mouse** key. The motion is stopped and control is returned to the button array with a press of the **left mouse** key.

The speed of the motion within the net sheet can be adjusted. The controls appear in the buttons immediately below the one for **Move On The Plane**. To speed up the motion, press the **Fast Moving** button. The intensity of increase is given by the number of times the button is pressed. To slow down the motion, press the **Slow** button a desired number of times for the appropriate intensity. With the adjustments for speed, the next time the user activates the movement in the net sheet, the selected speed of response appears. The speed adjustment is useful to quickly deform the grid into a desired general shape and then to be able to slow it down to permit slower more accurate movements.

The motion along the transverse links that pierce the control net sheet is activated by pressing the key entitled **Move Off The Plane**. Once pressed the scroll bar becomes appropriately labeled and the user moves the cursor to operate the scroll bar. This already contains the speed adjustments. When this motion is occuring the user sees the geometric changes in the nearest surface appear in real time. This is particularly convenient for the case when the boundary is to be deformed.

With the two types of motion, the application to the boundary is special. When it is left in the specified mode, the control point changes have no effect. In order to model the boundary in a free form manner, the strict adherence to the boundary grid must be released. This is accomplished by first moving the current control point to the desired boundary face and then by pressing the toggle button entitled **Toggle Boundary Form.** The toggle is a switch between its rigid specification and its definition in terms of control points. Once in the free-form state, the motion of any control point on that boundary will change it. The user can tell if a boundary face is opened up for free-form manipulation by seeing if the frame graphic does or does not have its fringe of grid points plotted in the first interior layer.

The End of a Session

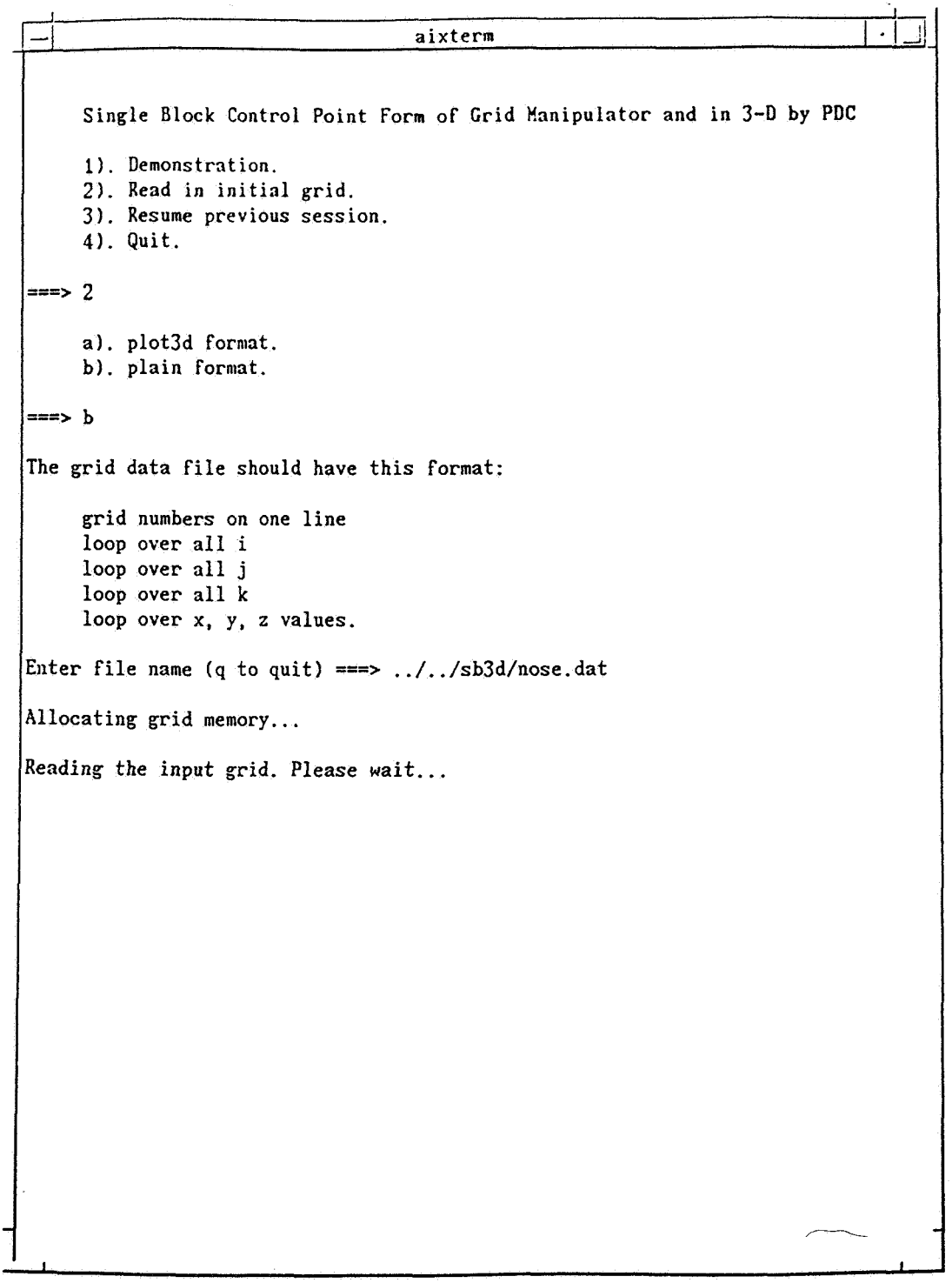
The session is ended by two quit commands. The first given by the button entitled **Quit Without Save** and the second is given by the button entitled **Save Grid Then Quit**. When the grid is to be saved, the user is queried for a file name.

Utilities

The general utility features are a request for more detailed help and the options to use the key board in certain cases. For example, the **I, J, K** keys operate with the same meaning as the corresponding graphic buttons. The detailed help is activated with the button entitled **CPF Help**. The CPF refers to the technical structure of the algorithm.

THE DYNAMIC MANIPULATION OF GRIDS

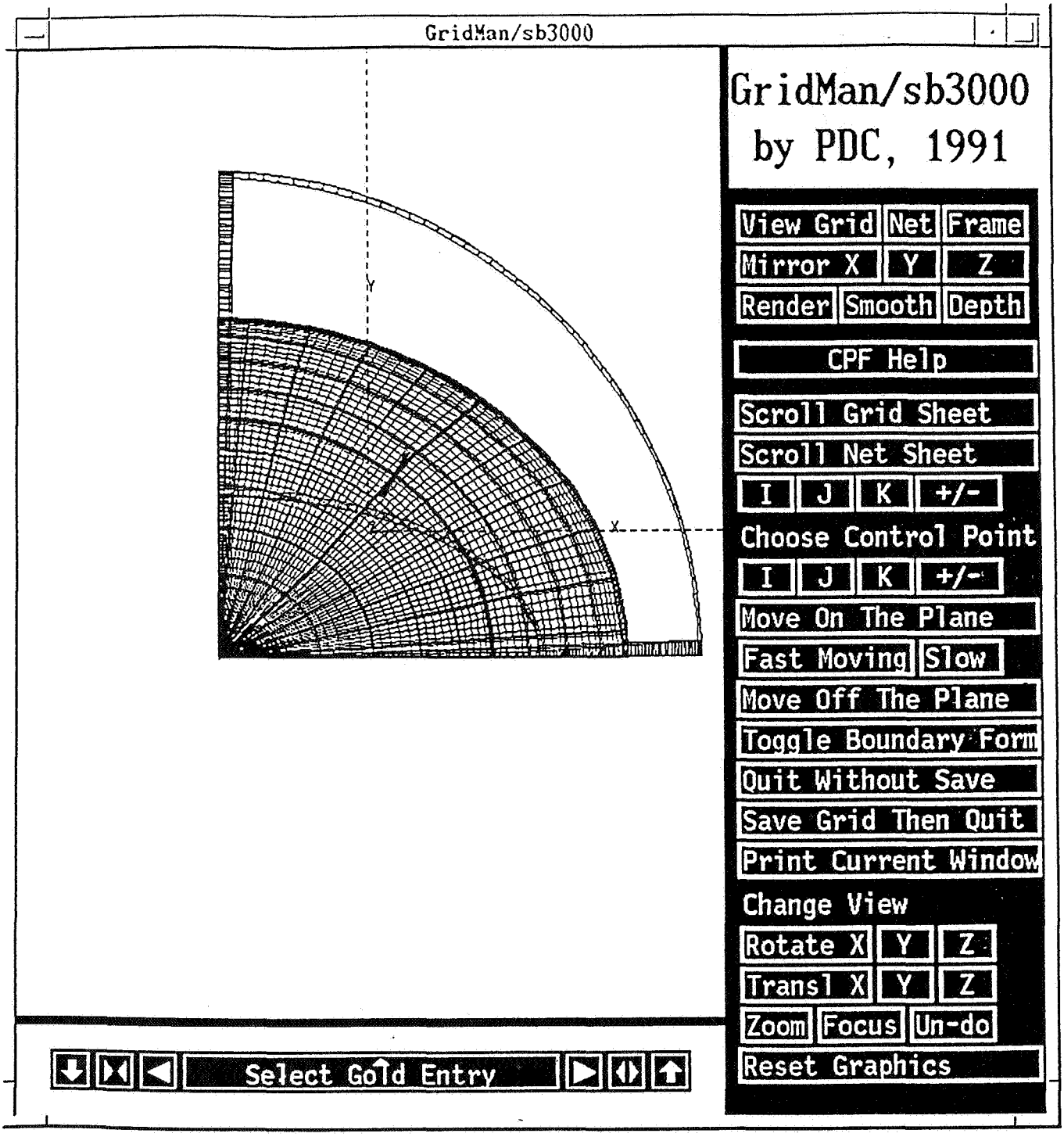
As an illustration of how the various active elements of the code are applied, an example session is provided below. This will show the use of the viewing features and the changes of the graphic object. It will allow one to see how one shifts the various parts of the three dimensional display about to show the parts of current interest. With the various configurational steps, the movement actions will be taken. This will demonstrate the various types of control point motion and their effects.



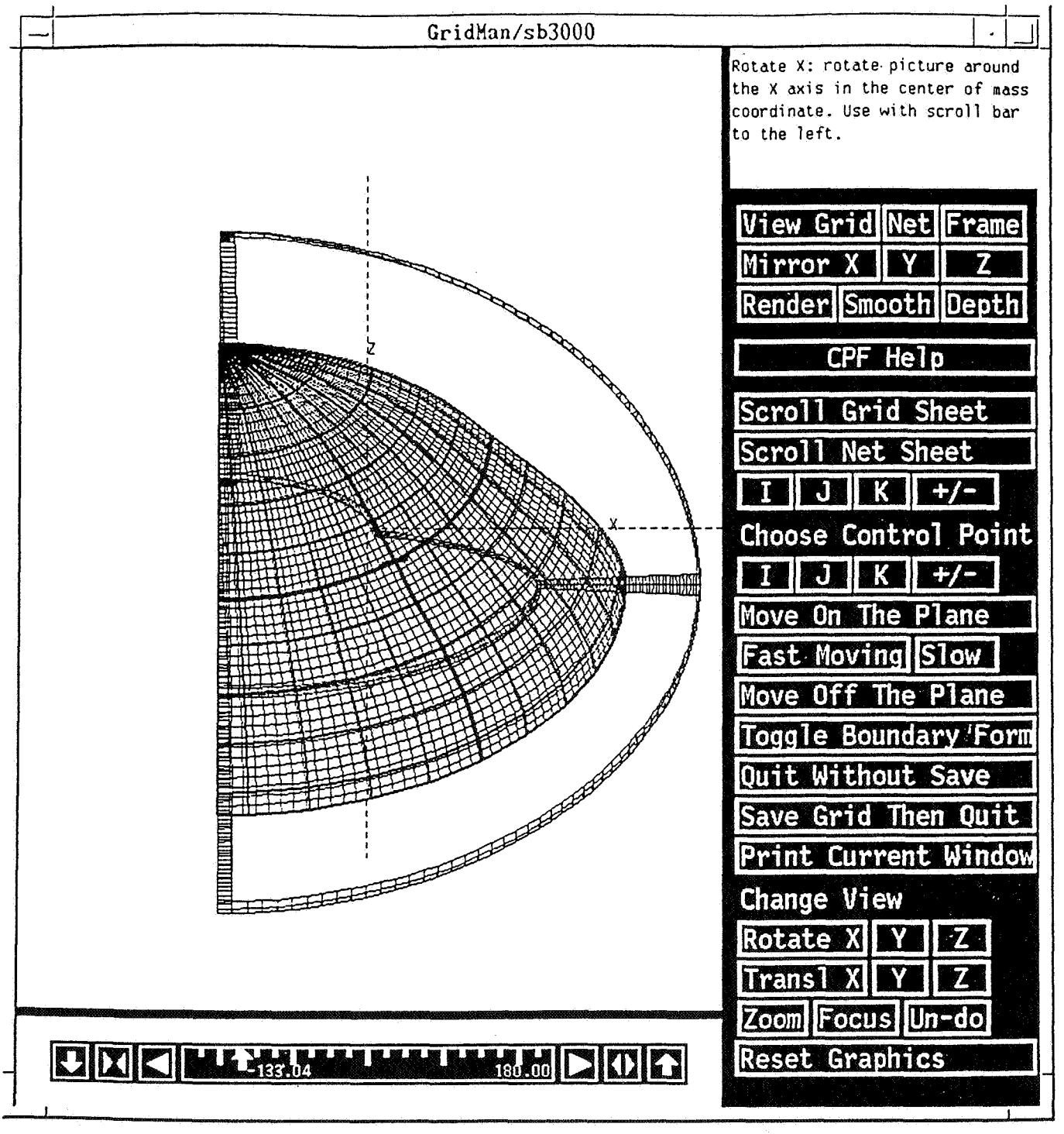
1. This is basically self-explanatory. You are prompted for different options step by step. For example, at the first input prompt (===>), we choose 2 to read in a grid. At the second prompt, we choose b for plain input format. Then for the third prompt, the file name is entered. Note that since we use dynamic memory allocation, the size of the grid can be as large as your system can take. If you have insufficient memory, however, you should remove the -DMAXMEM option in the *Makefile* and type make again to re-generate *gridman.sb3000*.

```
aixterm
     grid numbers on one line
     loop over all i
     loop over all j
     loop over all k
     loop over x, y, z values.

Enter file name (q to quit) ===> ../../sb3d/nose.dat

Allocating grid memory...

Reading the input grid. Please wait...

Calculating center of mass...

Shifting the grid to the center of mass coordinate...

The grid indices are from (0 0 0) to (49 49 49):
Type <return> to apply CPF to the whole block;
Type 0 to disable CPF (i.e., viewing the input grid only).
You may also enter a sub-block where CPF would be applied.
Remember CPF requires at least 3 grid points in a direction.
You may also enter, for example, 0 0 5 49 49 5, to apply CPF
only to k = 5 grid sheet for a surface manipulation.

===>

CPF block will be from (0 0 0) to (49 49 49).

Enter control numbers in I, J, K directions.
<return> for the default 1:4 control/grid ratio.
Note in CPF the minimum number of controls in one
direction is 5. If this is a CPF surface, put 0 in
the normal direction.

===>

Control numbers are 12 12 12.

Allocating system memory for CPF...

Computing blending functions...

Attaching control net to the grid...

Loading the specific boundaries...

Reconstructing the grid with CPF. Please wait...
```

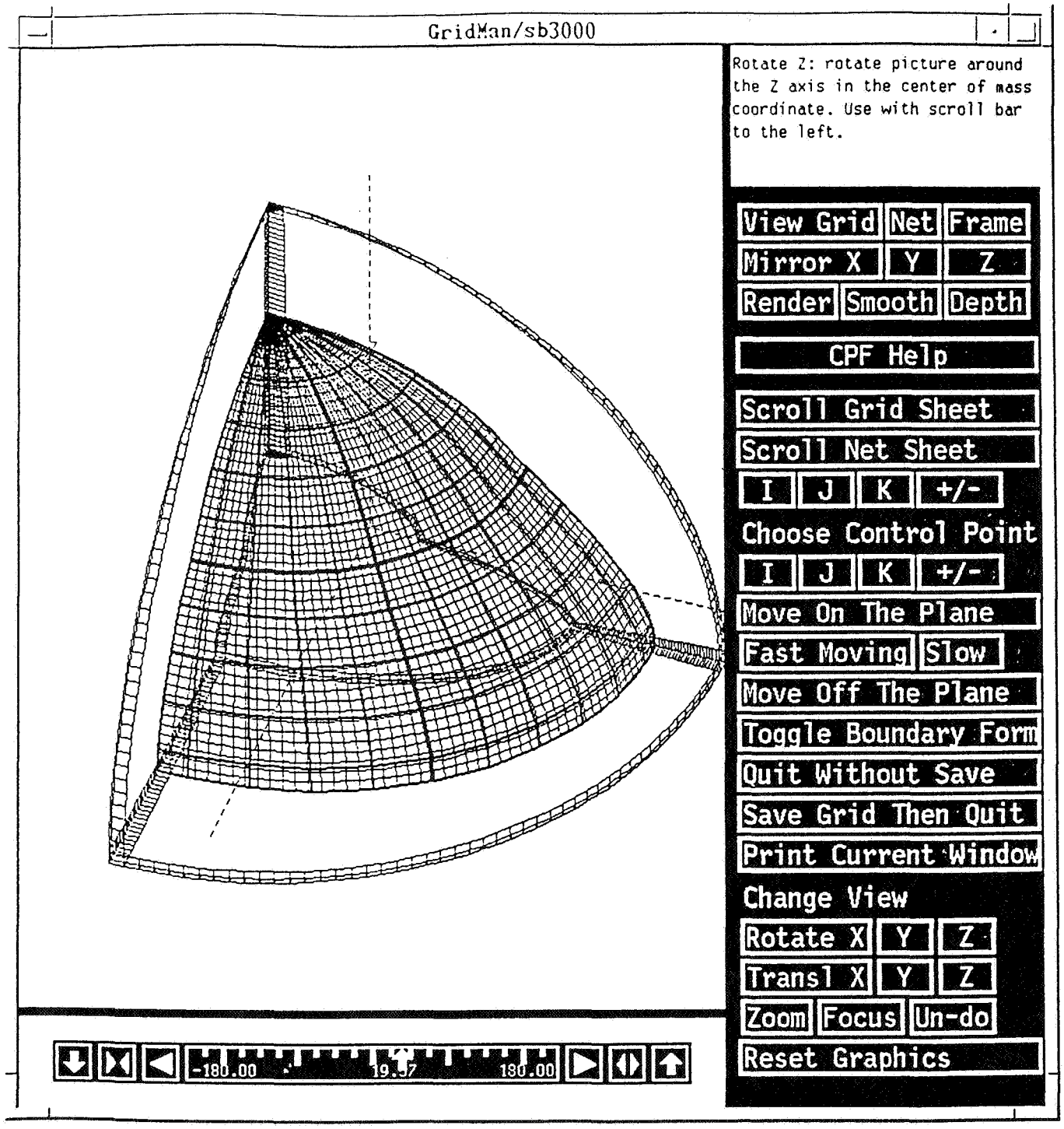
2. The first prompt on this picture is really the third prompt on the previous picture. The second prompt here asks where to apply CPF (Control Point Form). Here we simply type <Enter> to take default action which is to apply CPF to the whole grid block. The third prompt asks for the degrees of control you would like in each direction. The larger the number is, the more local control you have in that direction. Here we use the default action.



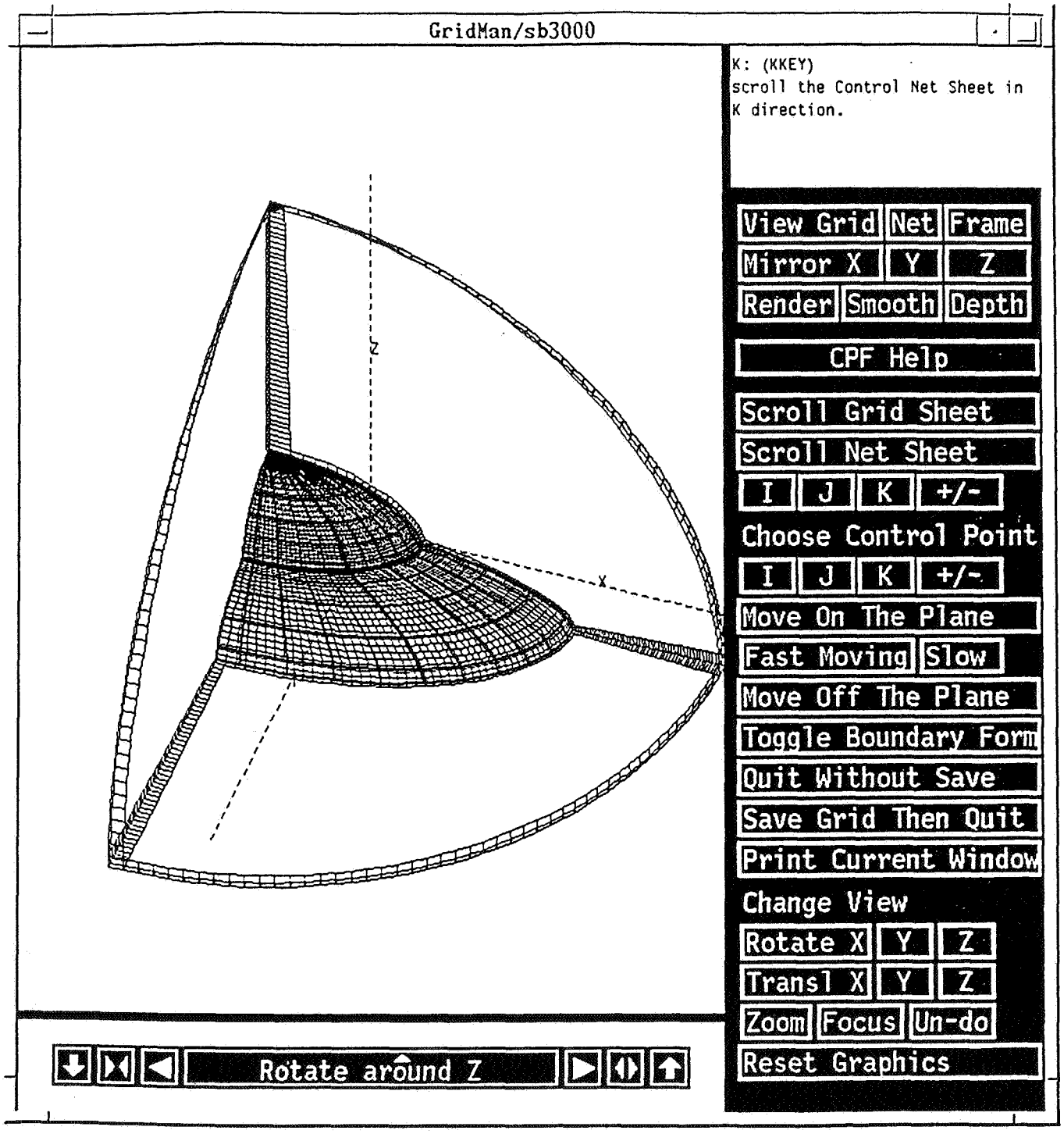
3. After a few moments, you will see the graphics display which has mainly three parts: 1). grid and net and frame display; 2). scroll bar; 3). menu bar. You generally use leftmousebutton to choose from menu bar or to use scroll bar. And some of the menu buttons have their keyboard equivalent. A dynamic help is available whenever a cursor falls on a button. General help is also available via CPF Help button.



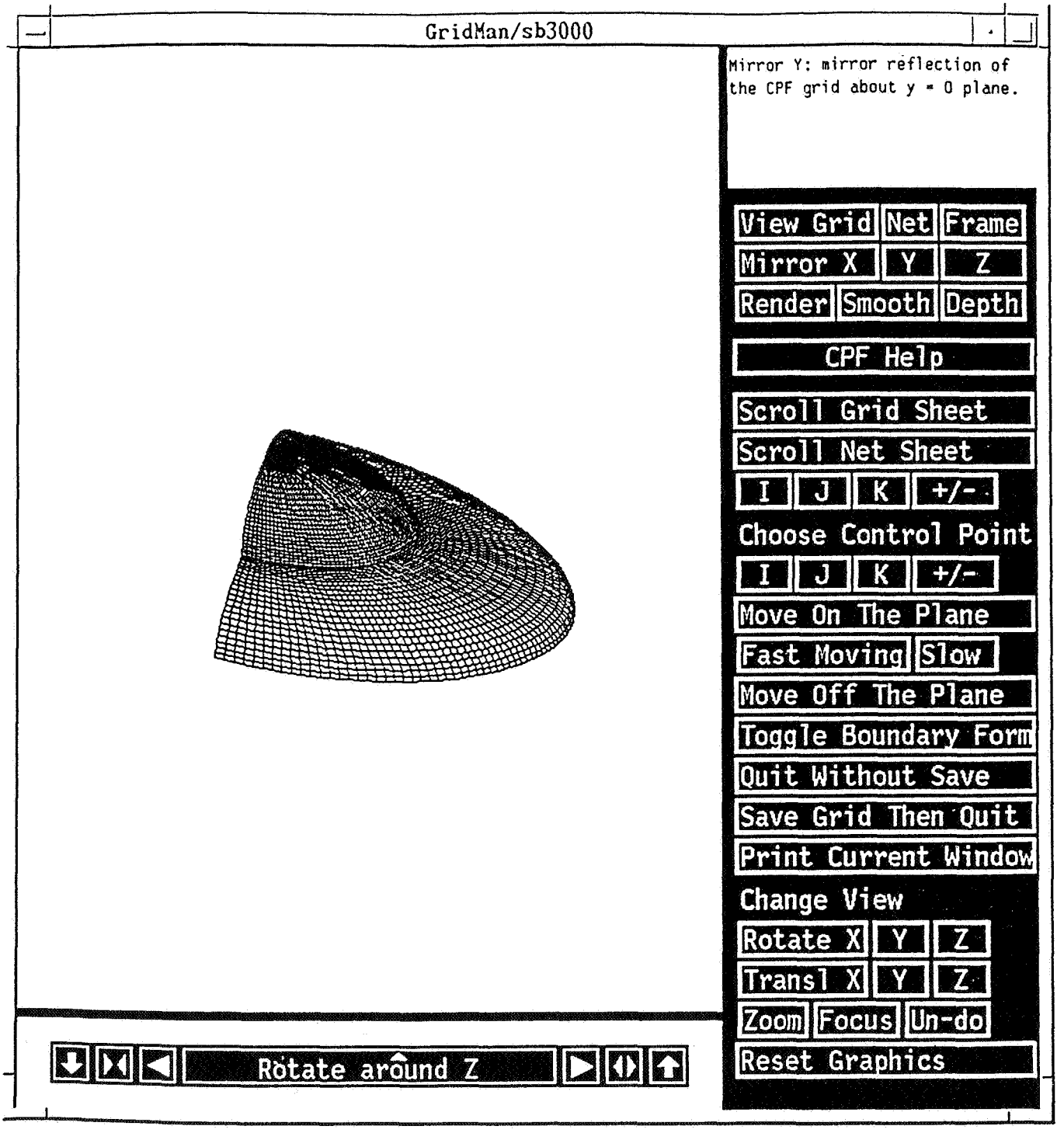
4. We first use modeling commands to get a better view of the graphical objects. Here we use scroll bar to rotate them around X axis. Note that we are using the center of mass coordinate which is the default. To do so, we first click on the Rotate X button, then click and keep pressing the ◁ icon on the scroll bar to continuously rotate around X axis.



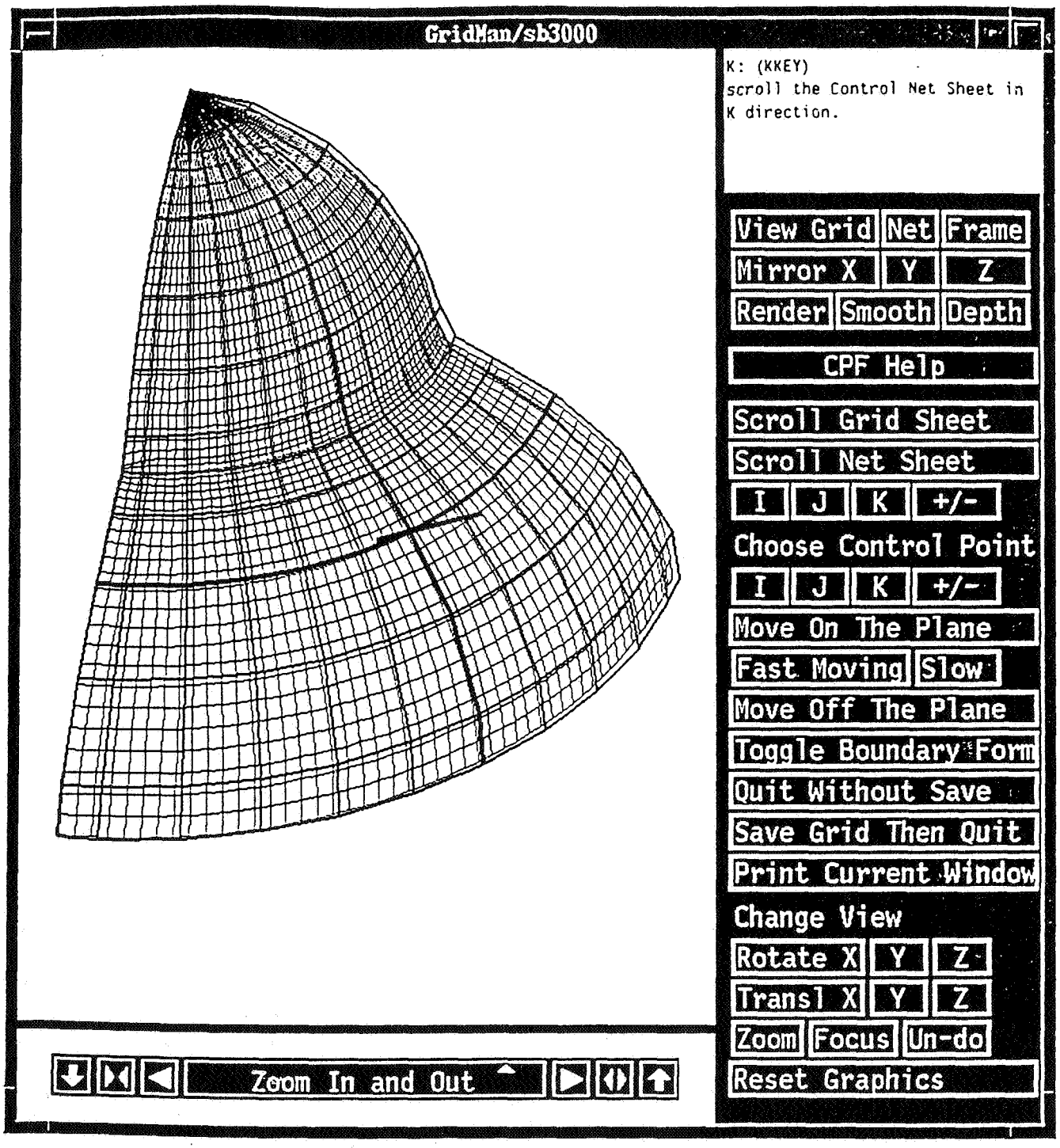
5. Now we click on the Z button (next to Rotate X button) followed by ▷ of the scroll bar to rotate about Z-axis. To see which axis is which, refer to the coordinate frame object (you see it clearly on previous page) which has dotted lines for axes and letters X, Y, Z for directions.



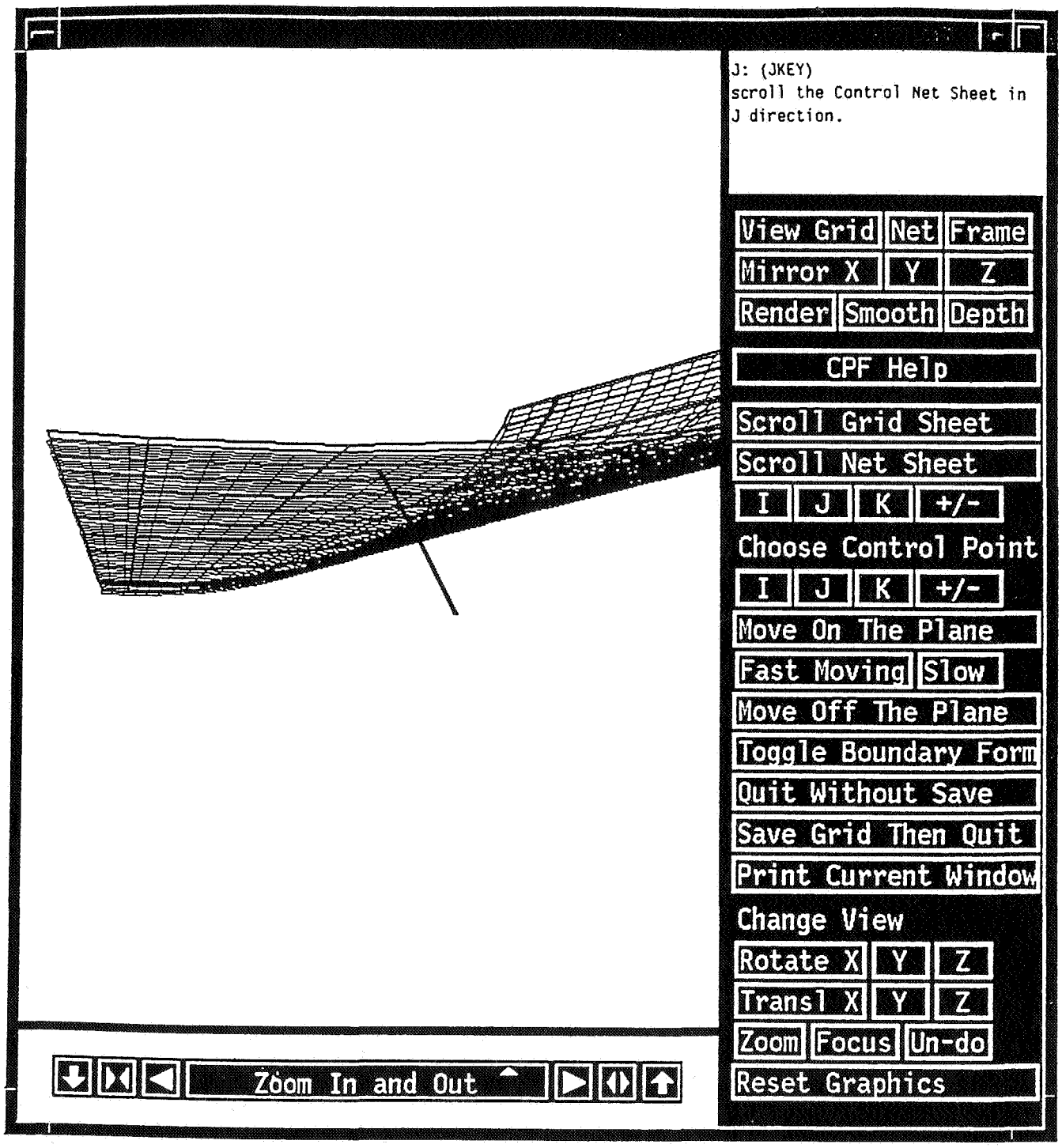
6. The initial grid and net sheets viewed, by default are the middle sheets in K direction. One may be interested in the boundary surfaces in that direction. The simplest way to do that is clicking on K button (below Scroll Net Sheet button) several times. You will find that when the net sheet is scrolled, the closest grid sheet to the net sheet in that direction is displayed. If you go over, use +/- button to reverse scrolling direction.



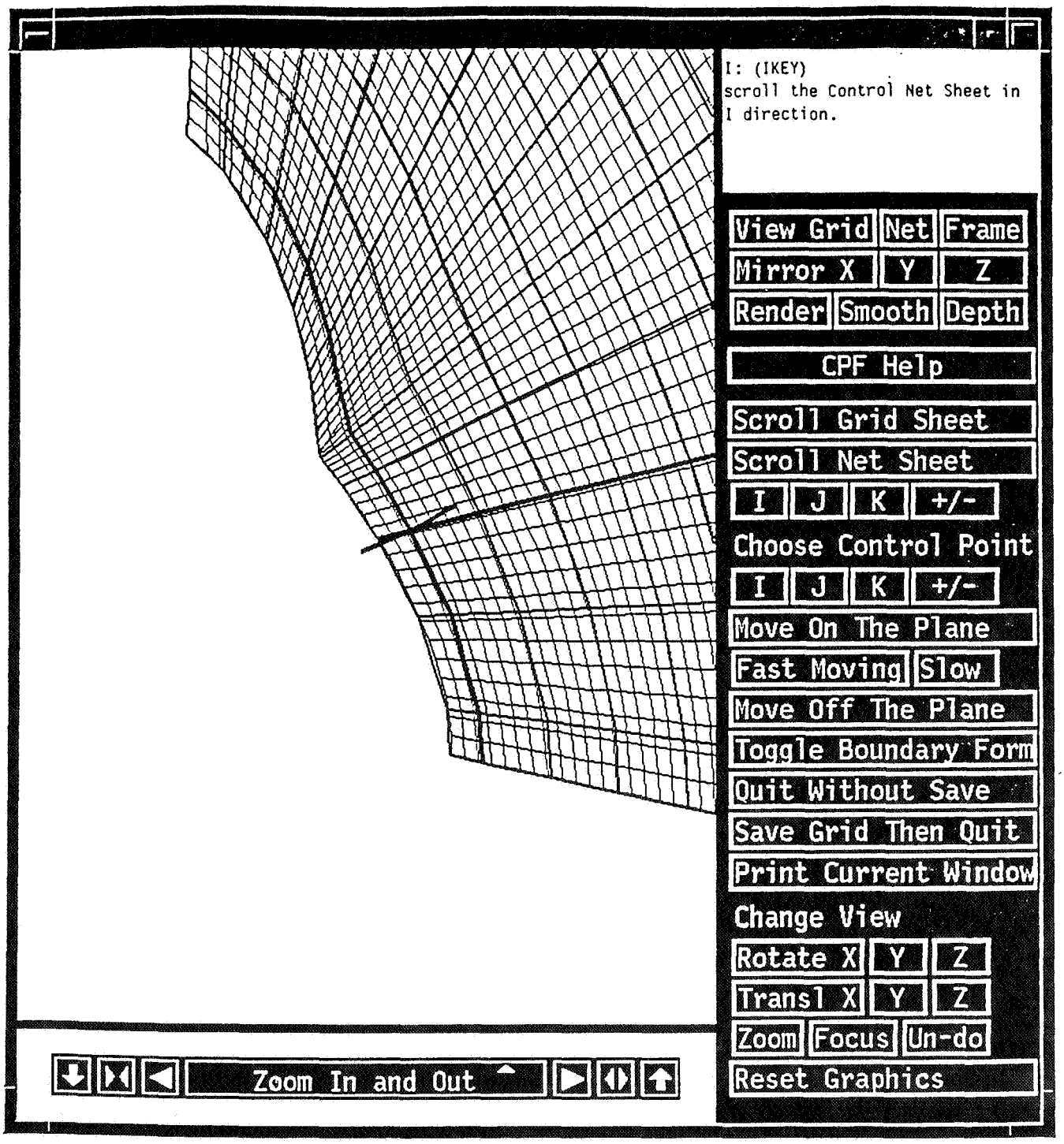
7. Now we click on the Net and Frame buttons (next to View Grid button) to toggle off the viewing of net and frame. Also since this particular grid is symmetric in Y direction, one can click on the Y button (next to the Mirror X button) to get a full view of the grid without distraction of the net and the frame.



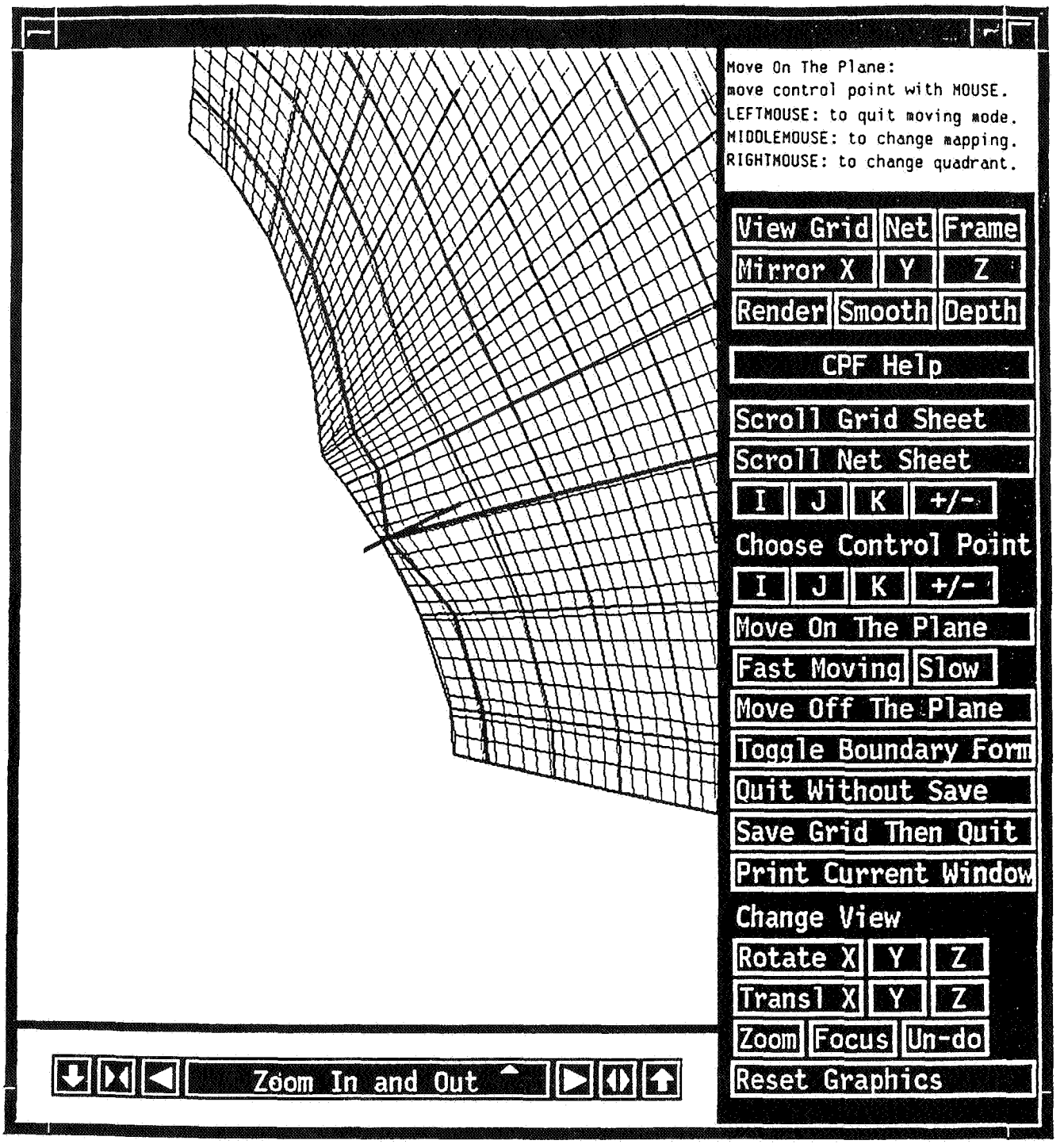
8. Now we would like to manipulate the grid with control points. We first use K button (below Scroll Net Sheet button) a couple of times to move out of the boundary surface just showed, then click on Net (next to the View Grid button) to put the control net display back on again.



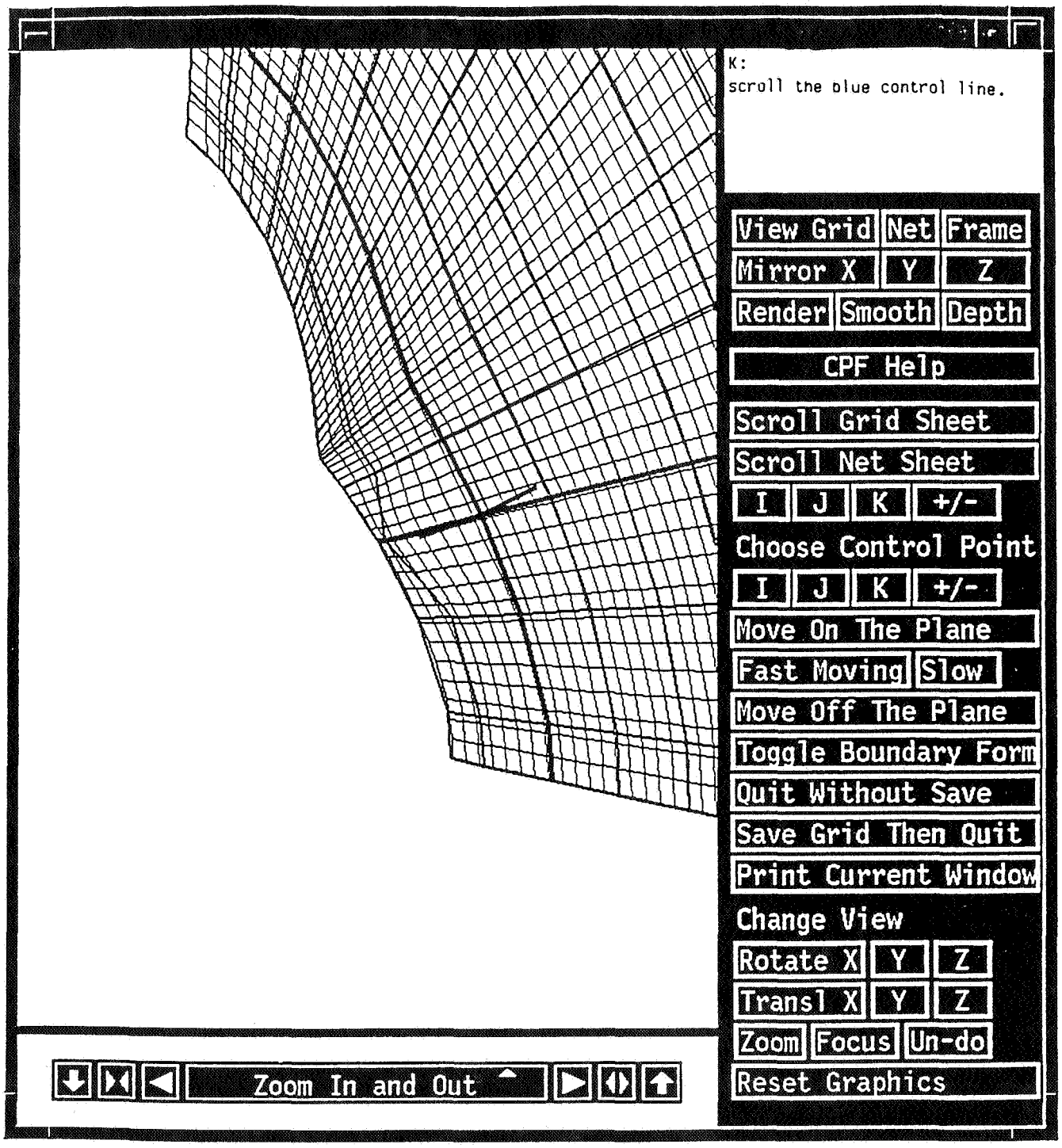
9. By clicking on J button (below Scroll Net Sheet), one gets a view of the net and grid in J direction. The dark line in the middle is the 3-point transverse vector for the current control sheet that can be used to guide Move Off The Plane operation.



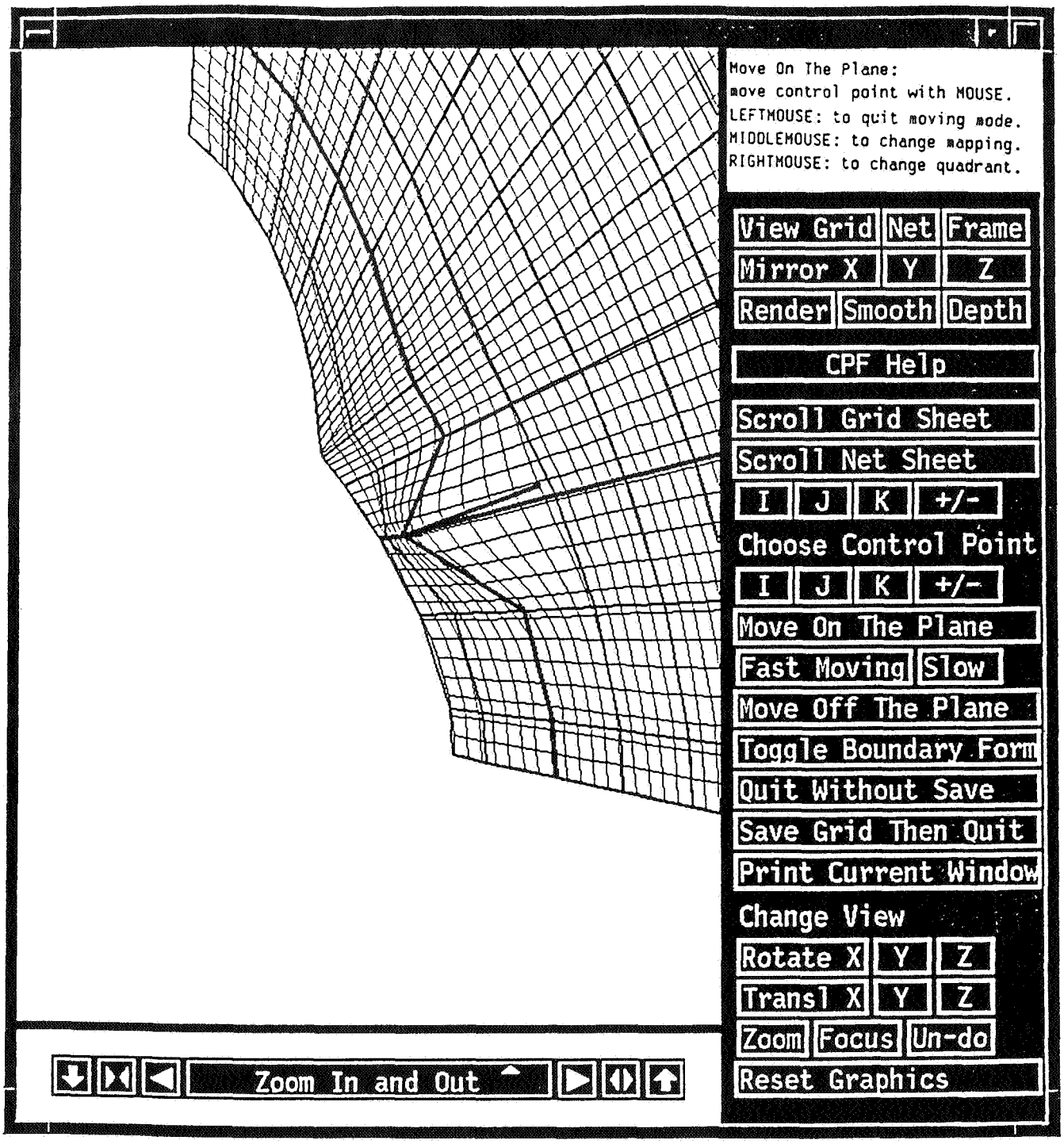
10. Same as above except now we are viewing the objects in I direction. Notice the two thick lines on this plane whose cross point is where the transverse line goes through. This is the control point we can dynamically move, either by Move On the Plane or Move off the Plane button.



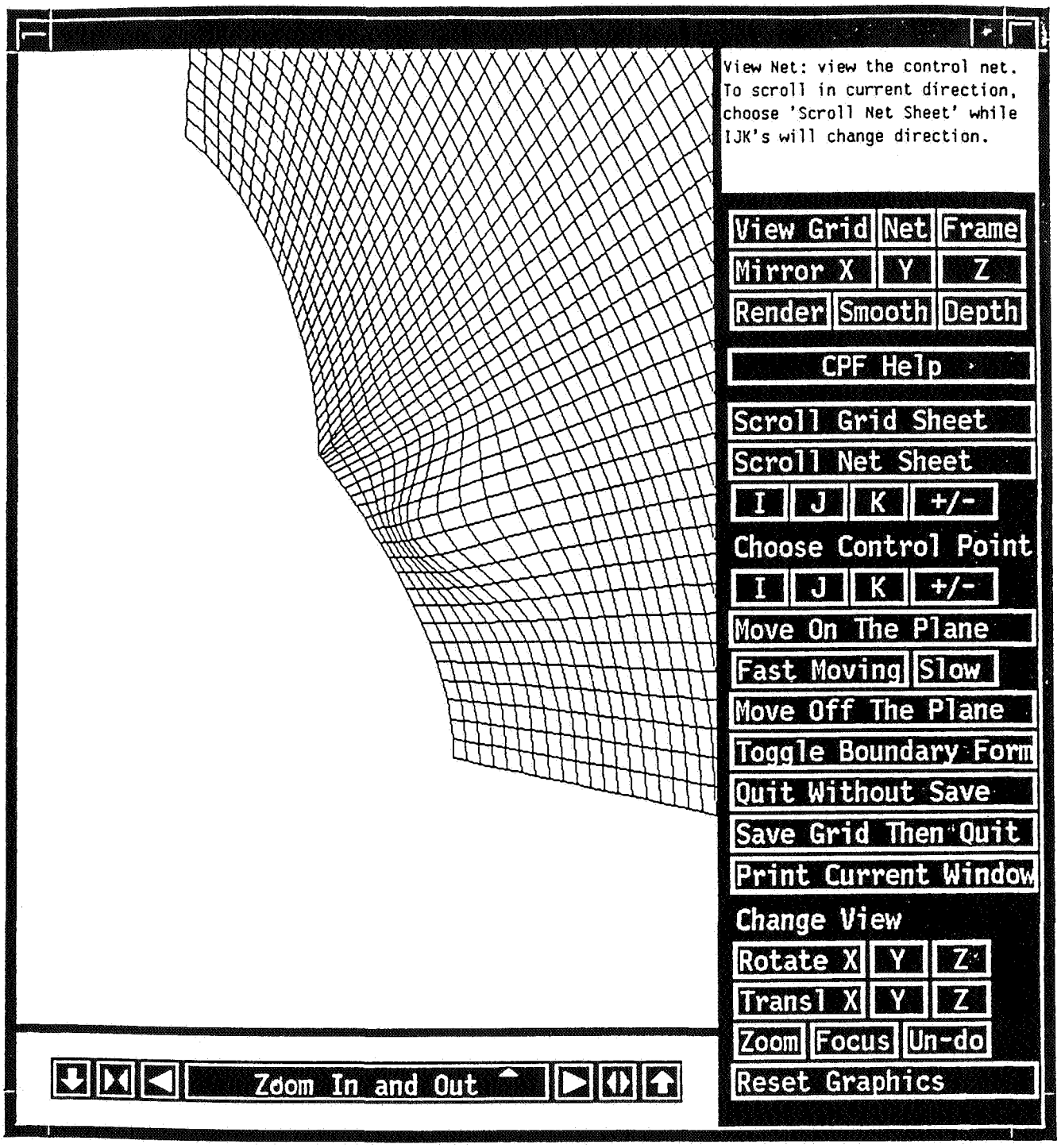
11. For now, we choose Move On the Plane. Here you use your MOUSE to control the movement of that control point on the plane shown. While in this mode, you may click middlemousebutton to change the mapping (you need this if you move your MOUSE horizontally but the control point moves as if vertically). Clicking the rightmousebutton would change the quadrant. If you want to move faster or slower, you must first quit moving by clicking leftmousebutton, then click on the Fast Moving or the Slow button once or more, then choose Move On the Plane to move again.



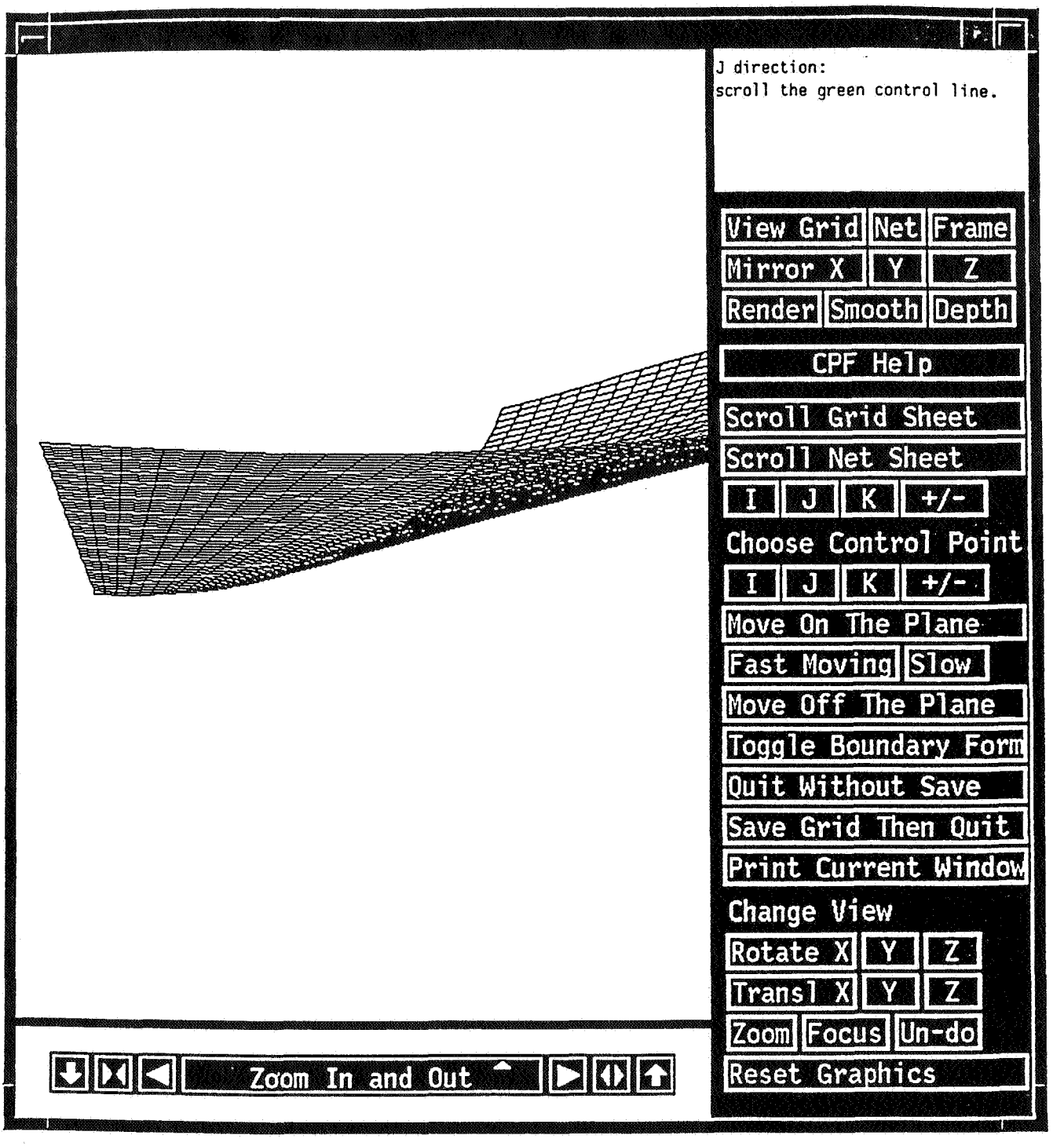
12. We now want to move one more point (on the same control plane). To do so, one would first buttons under Choose Control Point to scroll the thick lines. In this example, we only move thick line in K direction once.



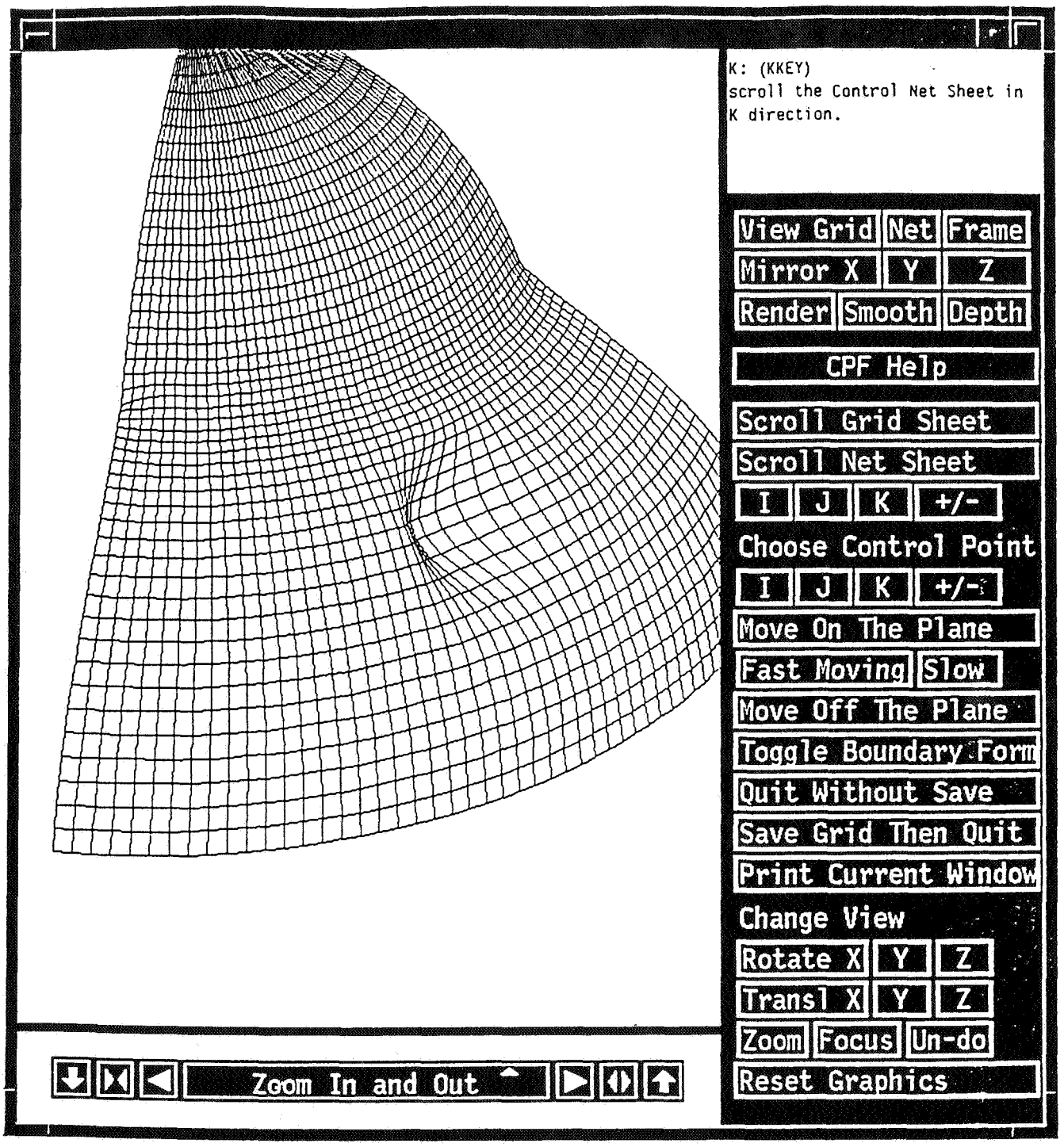
13. We choose Move On the Plane again. Here you use your MOUSE to control the movement of that control point on the plane shown. While in this mode, you may click middlemousebutton to change the mapping (you need this if you move your MOUSE horizontally but the control point moves as if vertically). Clicking the rightmousebutton would change the quadrant. If you want to move faster or slower, you must first quit moving by clicking leftmousebutton, then click on the Fast Moving or the Slow button once or more, then choose Move On the Plane to move again



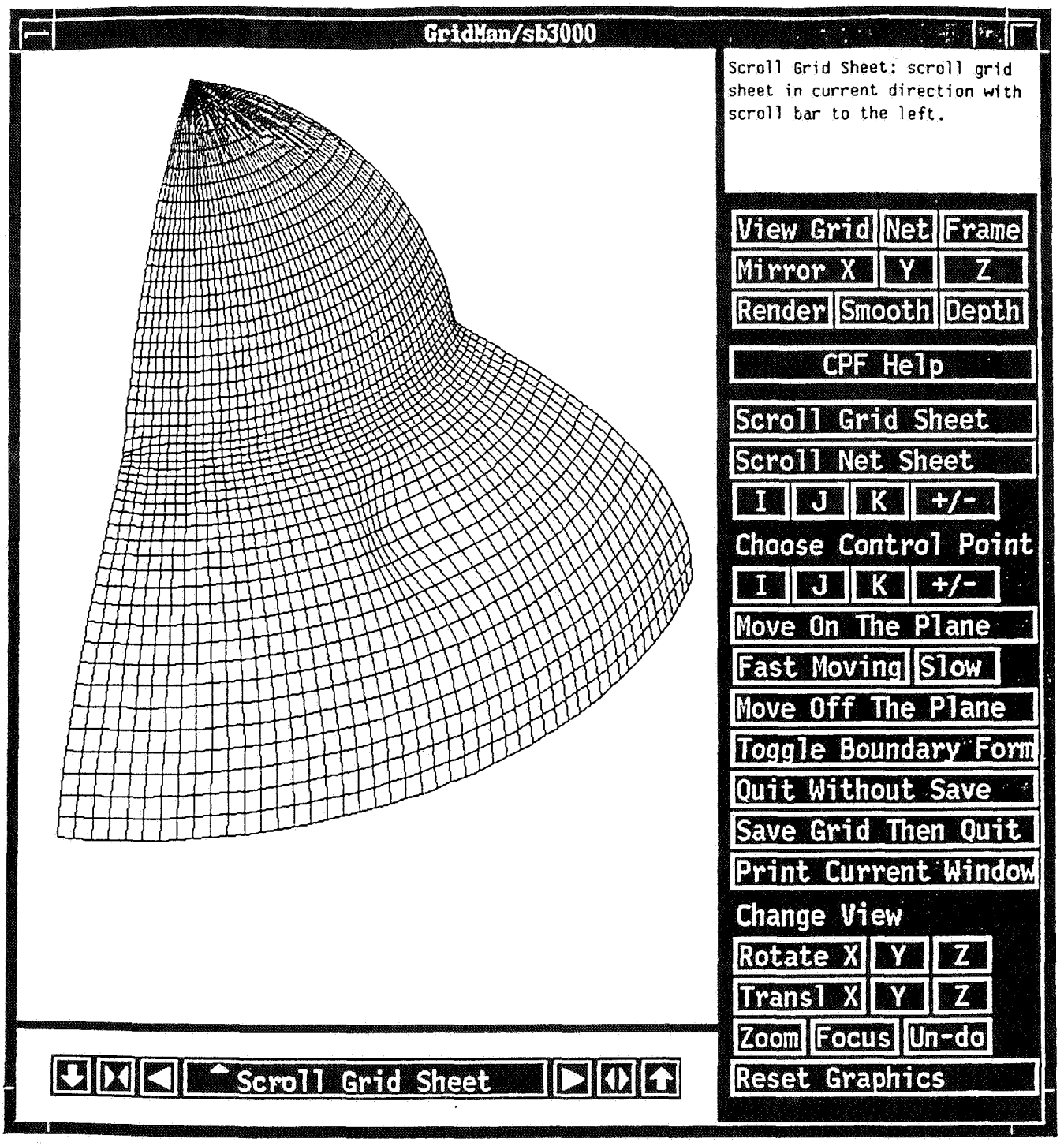
14. We take off the net by clicking on Net button to see the grid after we have moved those two control points.



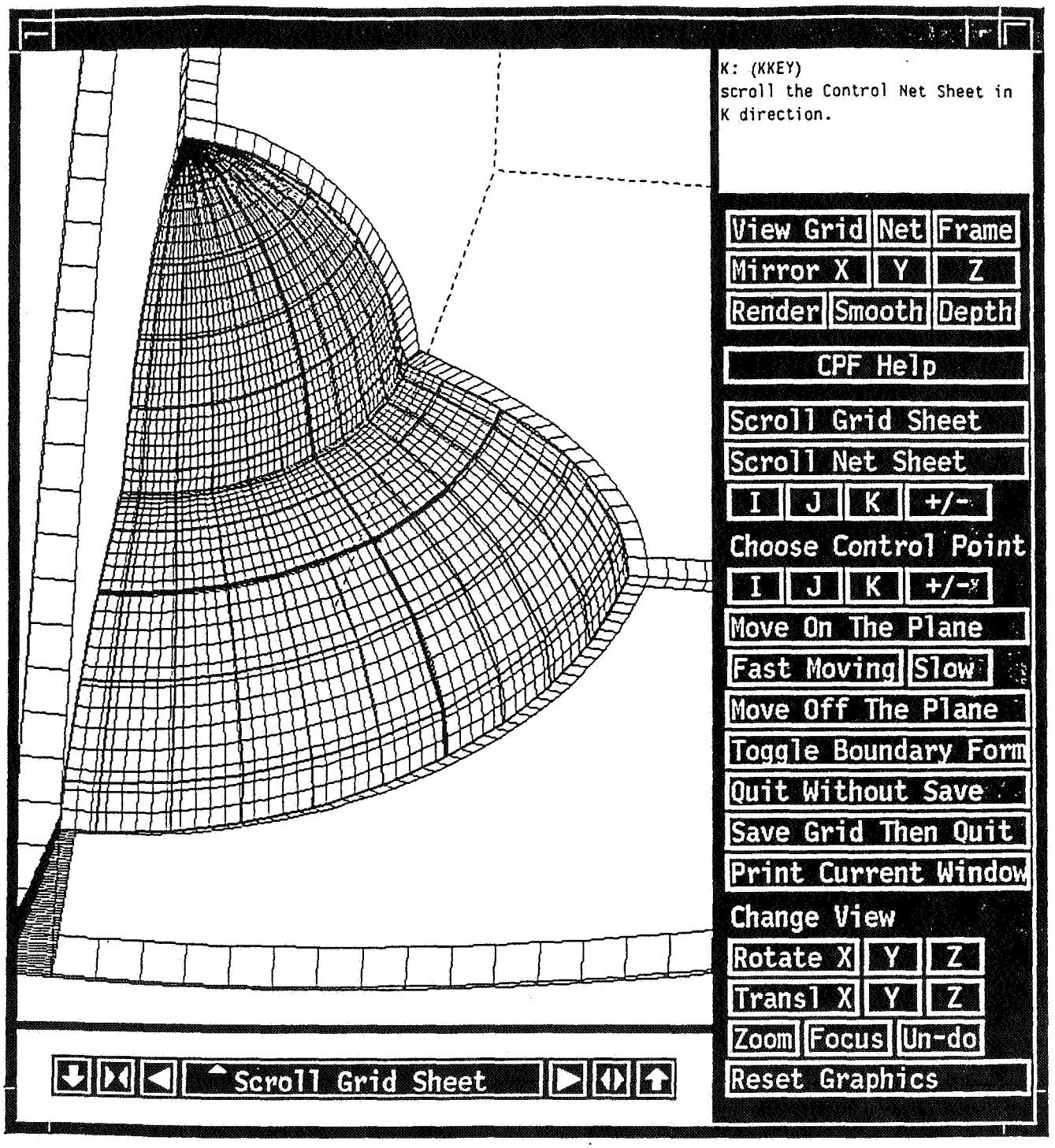
15. Click on J button (below Scroll Net Sheet) to view the grid and net in J direction (although net is off).



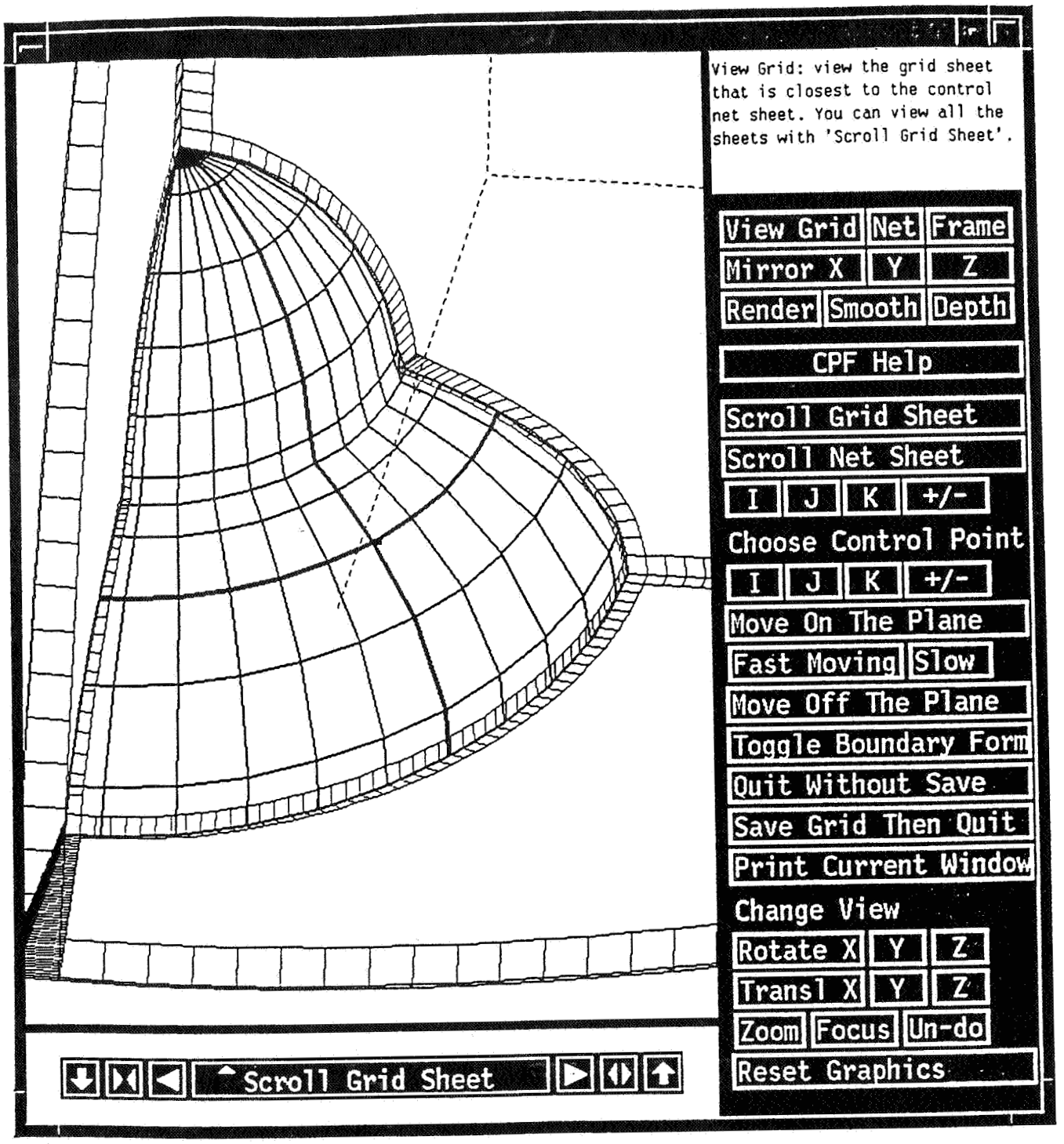
16. Now view the grid in K direction.



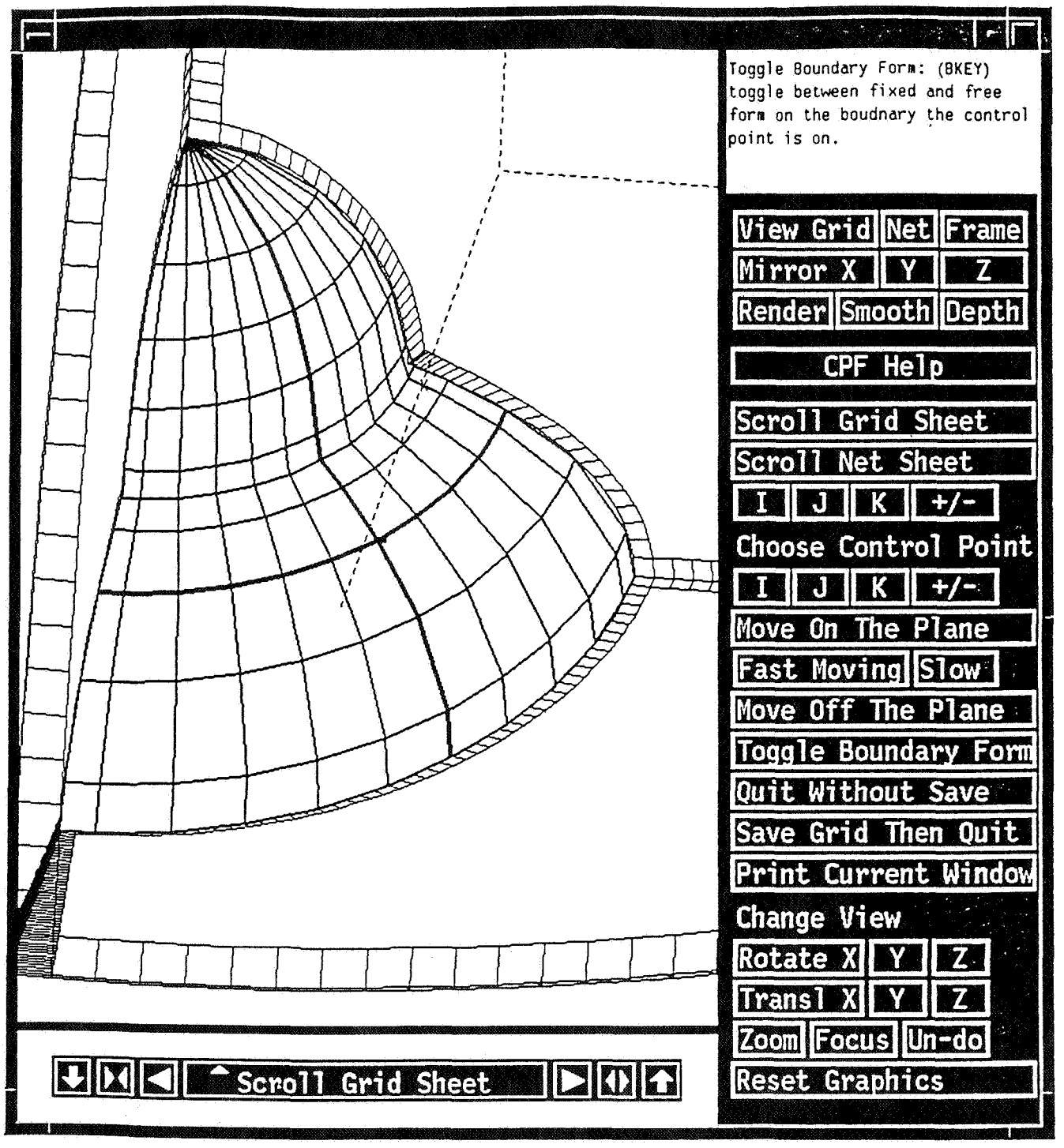
17. To examine grid changes in other sheets, use Scroll Grid Sheet along with the scroll bar. Here, we scroll to a sheet that is below the sheet just shown . As you can see, the changes are smaller.



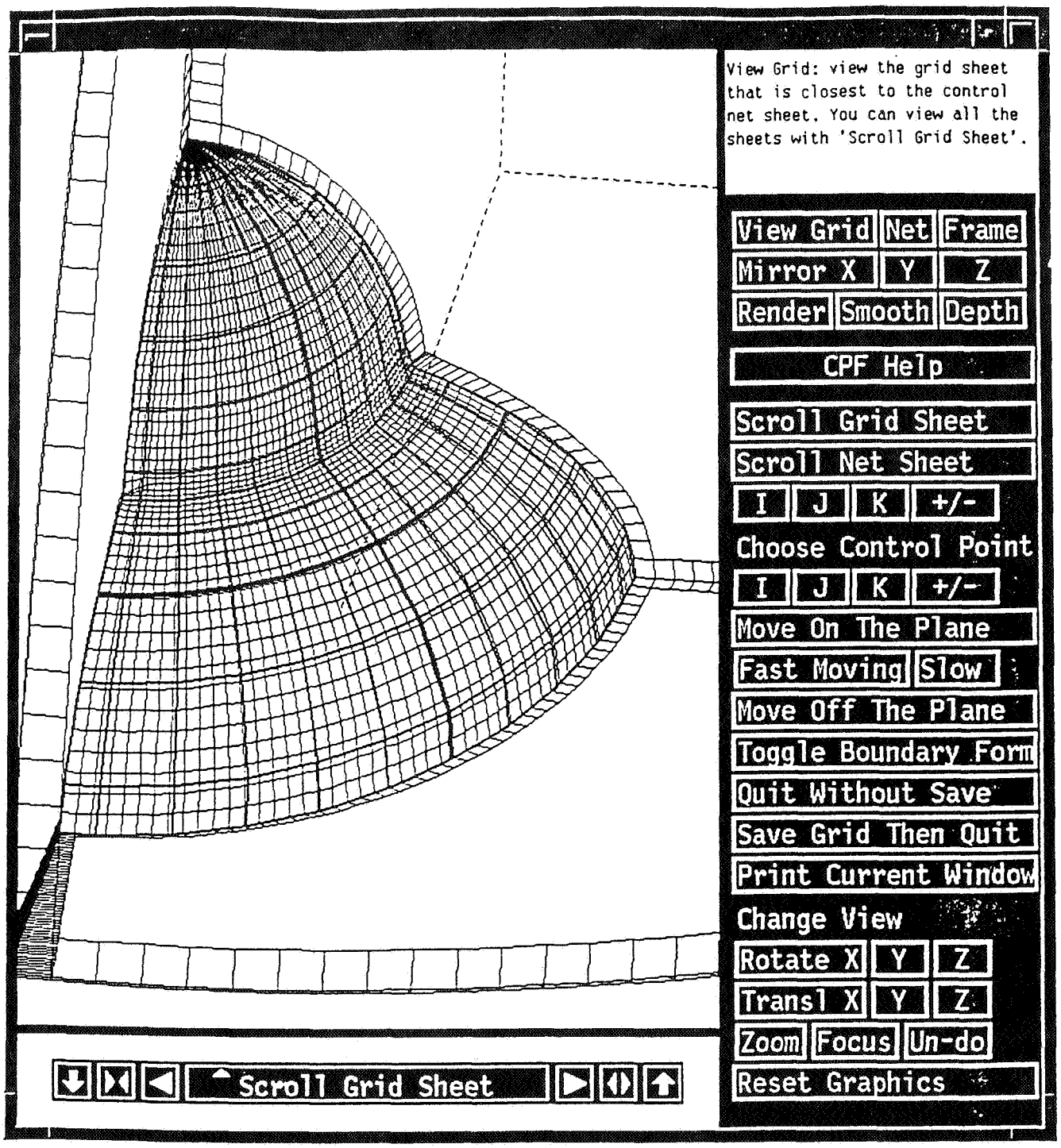
19. We would now like to show the options at the boundary. By clicking on K button (below Scroll Net Sheet) several times, one again reaches one of the boundary surfaces in K direction.



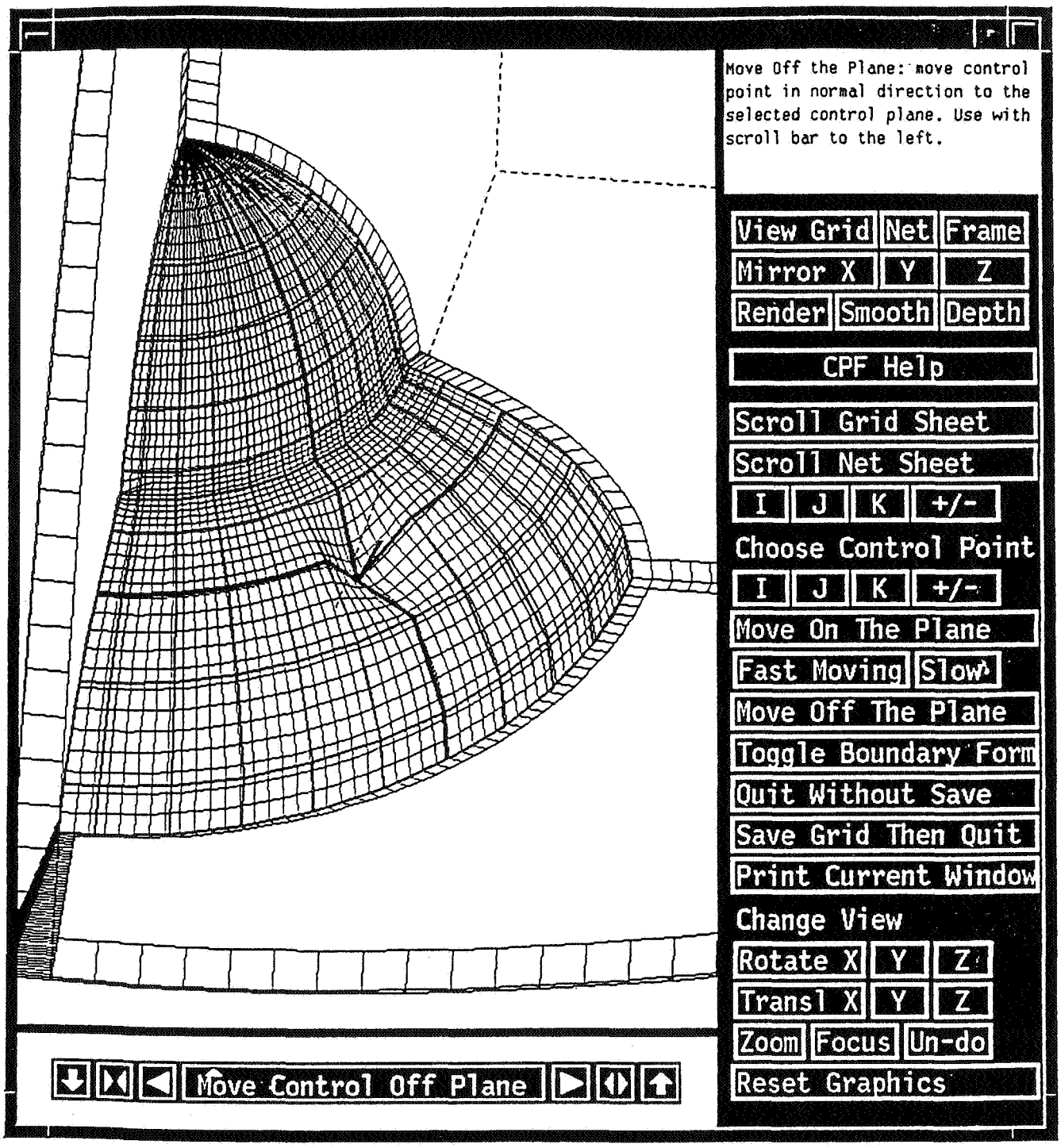
20. Now we toggle off the grid display and begin to examine the frame. You can see first layers of the grid on all 3 boundary surfaces. They not only tell you the grid spacing, but also the boundary type: if you see the first layers, then it is the fixed type. Otherwise it is free form. In this picture, the boundary surface for the K direction is of fixed type.



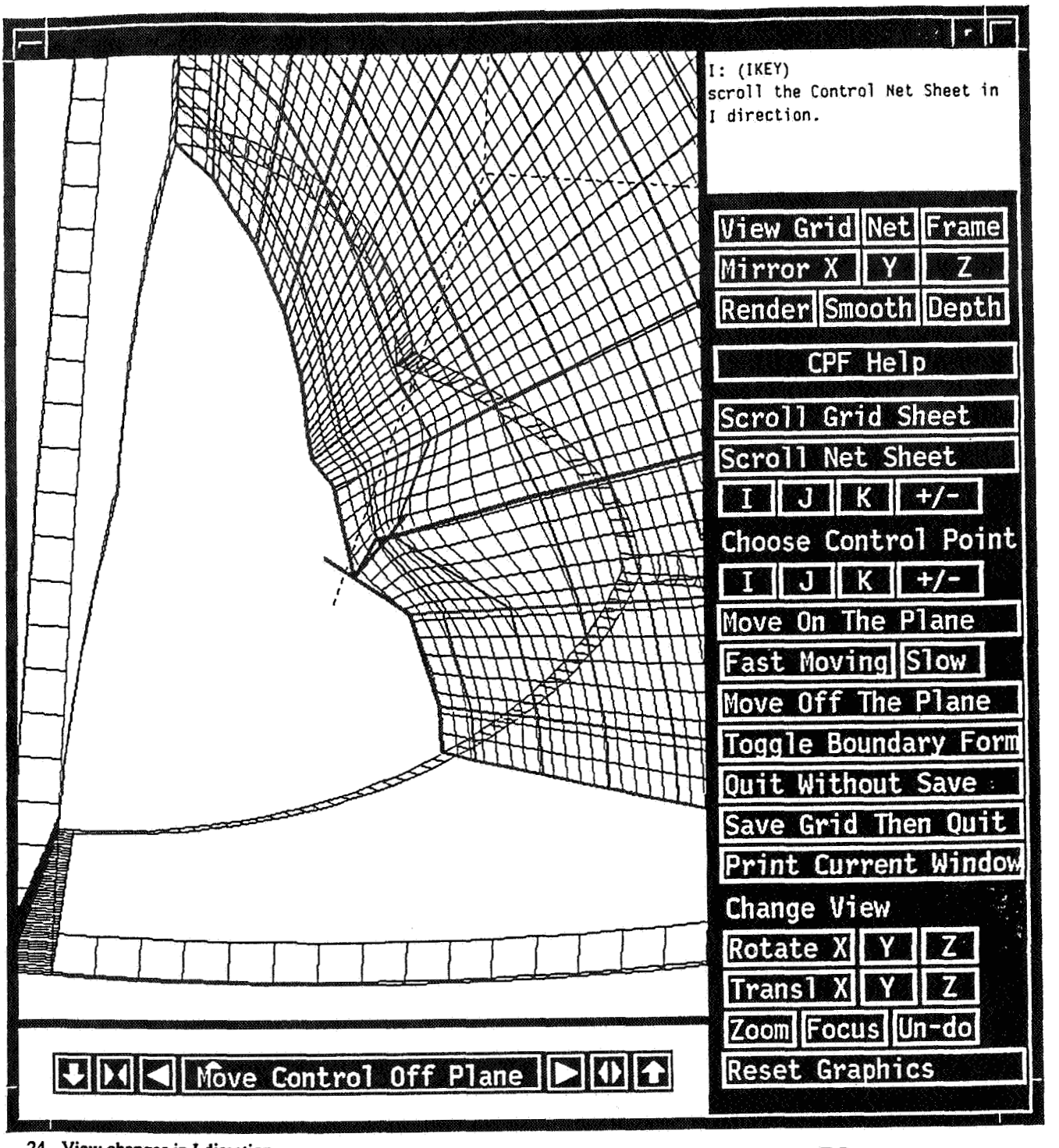
21. Now click on Toggle Boundary Form to change the boundary type (the boundary where the control net is on!). You can see now that the first layers on this boundary are off i.e., it is free boundary now.



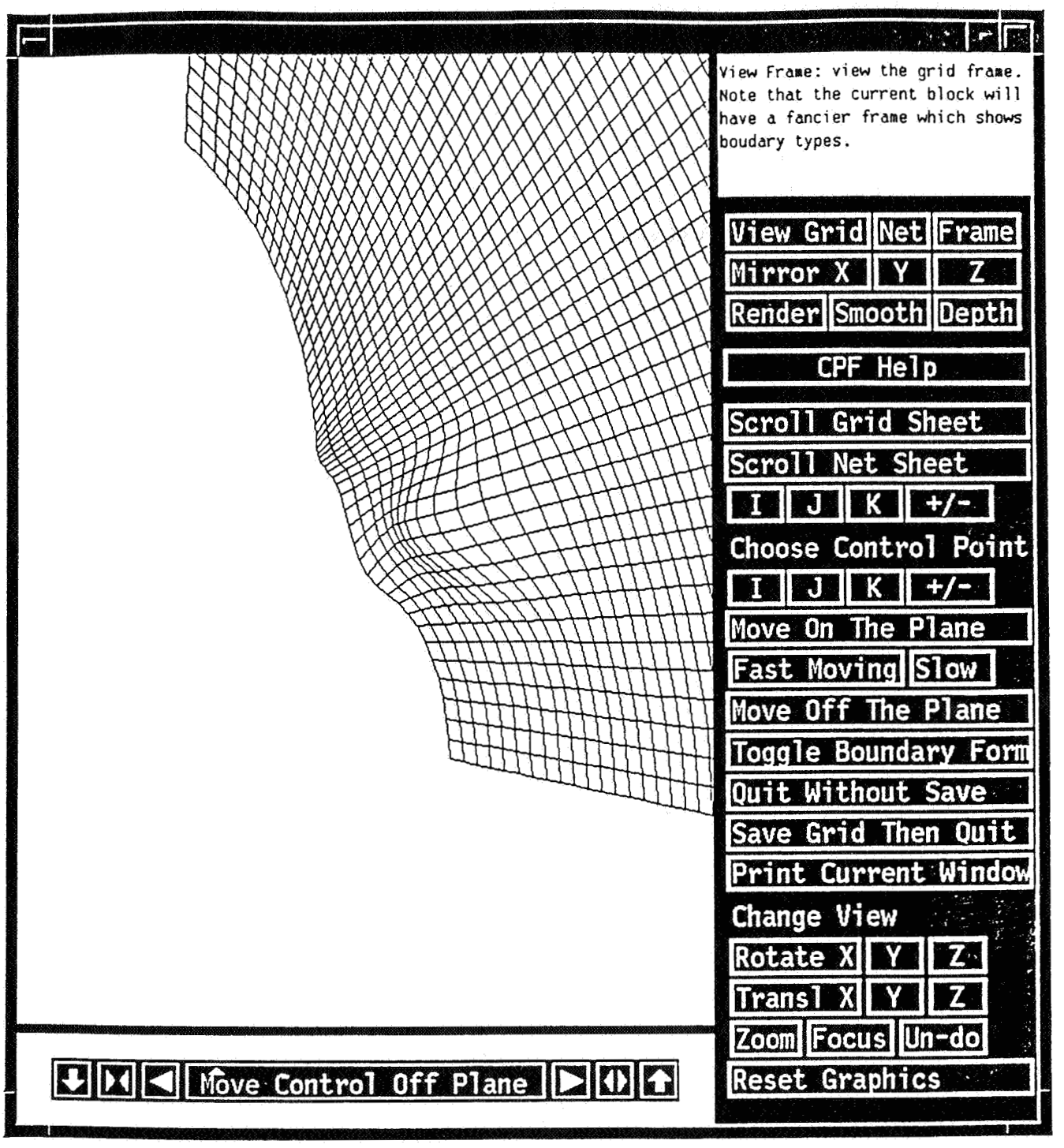
22. We put the grid display on to see the new grid on that boundary surface. The changes from the fixed one should be minimal (compared with Picture No.19).



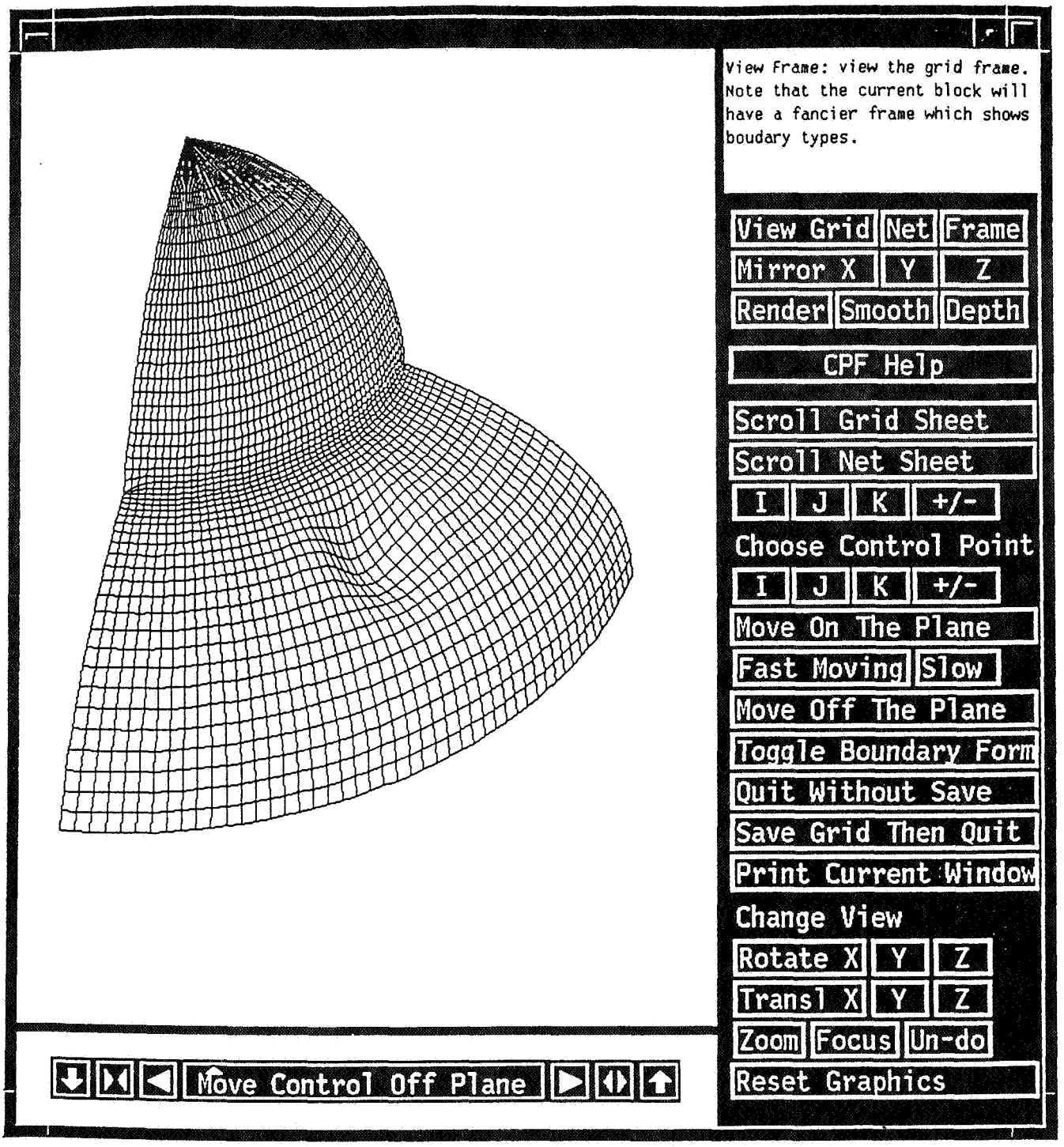
23. We now exercise the Move Off the Plane that will utilize the scroll bar. As you can see, we push the control point inside (remember this surface is free-form now).



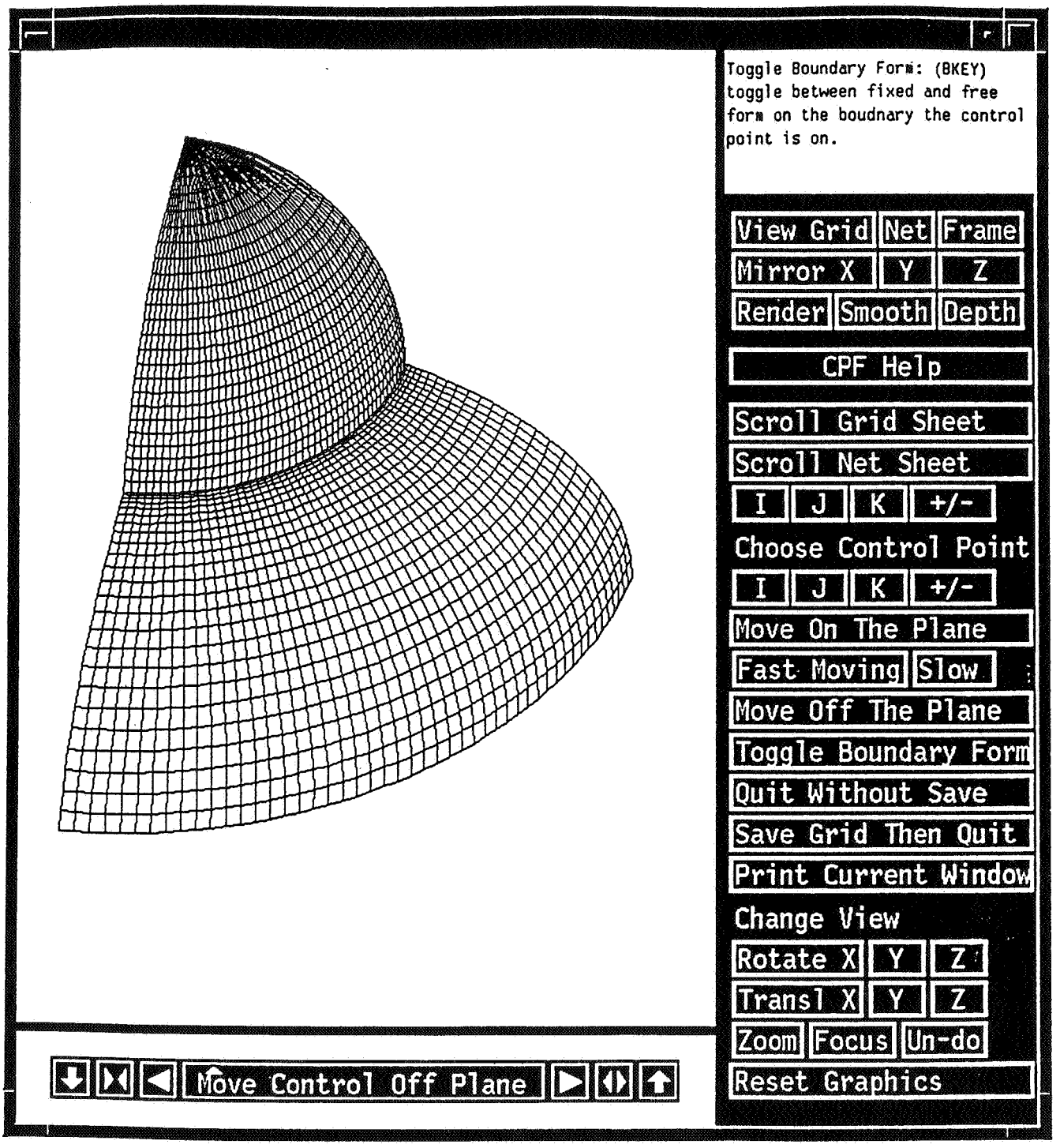
24. View changes in I direction.



25. View grid sheet in I direction only.



26. View grid sheet in K direction only.



27. Click on Toggle Boundary Form again to switch the boundary form back to specified (i.e., fixed type). As you can see, the grid surface is back to the original one.

Domain Modeling and Grid Generation for Multi-block Structured Grids with Application to Aerodynamic and Hydrodynamic Configurations*

S.P. Spekreijse
J.W. Boerstoel
National Aerospace Laboratory NLR
P.O. Box 90502, 1006 BM Amsterdam, The Netherlands
phone 31-5274-8361, fax 31-5274-8210

P.L. Vitagliano
Alenia Aeronautica, Direzione Tecnica Trasporto/Ufficio di Aerodinamica
80038 Pomigliano d'Arco (Napoli), Italy

J.L. Kuyvenhoven
Fokker Aircraft B.V.
P.O. Box 7600, 1117 ZJ Schiphol-Oost, The Netherlands.

Abstract

About 5 years ago NLR , Alenia/Gat and Fokker started jointly the development of a flow simulation system for engine/airframe integration studies on propeller as well as jet aircraft. The initial system was based on the Euler equations and made operational for industrial aerodynamic design work. The system consists of three major components: – a domain modeller, for the graphical interactive subdivision of flow domains into an unstructured collection of blocks, – a grid generator, for the graphical interactive computation of structured grids in blocks, – a flow solver, for the computation of flows on multi-block grids. The industrial partners of the collaboration and NLR have demonstrated that the domain modeller, grid generator and flow solver can be applied to simulate Euler flows around complete aircraft, including propulsion system simulation. Extension to Navier-Stokes flows is in progress. Delft Hydraulics has shown that both the domain modeller and grid generator can also be applied successfully for hydrodynamic configurations. In this paper, an overview is given about the main aspects of both domain modelling and grid generation.

*This investigation was partly performed under contract 01604N with the Netherlands Agency for Aerospace Programs (NIVR).

1 Introduction

The underlying idea of the widely used and accepted multi-block approach is to subdivide a geometrical complex flow domain region into several smaller, more manageable regions, referred to as blocks. Typically there are several individual blocks in a given flow domain, each block having three computational coordinates. On each block there are six block-faces with two computational coordinates. On each face there are four face-edges, containing only one computational coordinate. Furthermore, each edge has two vertices.

A grid in a block is represented by a number of discrete grid points, ordered in a three dimensional array.

In general, the flow domain may be subdivided into any conceivable structure provided that cell to cell matching on block boundaries is maintained. This does not require that one block-face of a given block match exactly with a block-face of another block, only that each cell on an interior block-face match with a cell of another interior block-face. In our approach, it is possible to subdivide any block-face into two subfaces. A face which consists of two subfaces is called a compound face. Each subface of a compound face is also allowed to be compound again. A face which is not compound is called elementary. Thus it is possible to subdivide a block-face into any number of elementary subfaces. With the concept of compound faces, two individual blocks may be connected to each other by a face which is a subface of one of the six block-faces of both blocks. This feature is called " partial block boundary interfacing", i.e. any part of one block-face may be connected to any compatible part of another block-face [1],[2]. In most applications this flexibility allows the user to define much fewer and larger blocks than would be the case with "complete boundary interfacing". An illustration of this effect is given by a simple 2D example (Fig. 1).

The use of compound faces also requires the use of compound edges. A compound edge is an edge which consists of two subedges. Each subedge is also allowed to be compound again. An edge which is not compound is called elementary. Thus it is possible to subdivide a face-edge into any number of elementary subedges. It is clear that a compound face must have at least two opposite compound face-edges.

A multi-block system requires domain decomposition, i.e. the subdivision of the flow domain into suitable blocks. In our approach, domain decomposition is done by a so called domain modeller which is a graphical interactive code operational on Silicon Graphics workstations. During domain decomposition, a user may interactively create vertices, edges, faces and blocks. Tools are available to create edges and faces with a user-defined geometrical shape (such as interior block-interfaces and block-edges), and edges and faces which are constrained to a certain surface shape (particularly those conforming to the geometry).

Output of the domain modeller are a topology and geometry file. The topology file describes the arrangement of the blocks and the geometry file describes the geometrical position of the vertices and the geometrical shape of edges and faces. The topology and geometry files are the main input files of the grid generator which is also a graphical interactive code operational on Silicon Graphics workstations. Output of the grid generator

is a grid file which contains the grid points in the blocks and the topological information about the arrangement of the blocks. The grid file is the main input file for a flow solver.

In this paper, an overview is given about the main aspects of domain decomposition and grid generation. The usability of the applied domain decomposition and grid generation techniques is demonstrated for both aerodynamic and hydrodynamic configurations. A general overview of the complete system (including flow simulation) may be found in [3].

2 Domain decomposition

Computer aided design (CAD) has become a standard tool in the aerospace industry for geometrically defining developing configurations, and the majority of advanced designs which are used in CFD studies have indeed been created on CAD systems. The differences in the various CAD systems, however, have made it necessary to establish one universal format for direct use during interactive domain modelling. Therefore, an entire configuration surface is represented by a set of input configuration surfaces, each surface containing a two dimensional ordered array of physical points. The input configuration surfaces are input for the domain modeller.

The goal of domain modelling is the creation of a topology and geometry file, which are required input files for the grid generator. The topology file defines the topology of a multi-block system and is described in subsection 2.1. The geometry file defines the geometrical position of the vertices, and the geometrical shape of the edges and faces, and is described in subsection 2.2. The creation of the topology and geometry file during a graphical interactive domain decomposition is described in subsection 2.3.

2.1 Topology definition

The blocks,faces,edges and vertices in a particular multi-block system are represented as $\{B\},\{F\},\{E\},\{V\}$ which are so called " label sets ". The label sets are subsets of $\mathcal{N}$: the set of positive natural numbers.

The topology of a multi-block system describes the arrangement of blocks. The arrangement of the blocks is defined by connectivity relations between the label sets. Only five connectivity relations define the complete arrangement of the blocks in a particular multi-block system.

The first one relates each block B to an *ordered* set of six block-faces:

$$\forall\, B \in \{B\} :\ B \mapsto (F_1, F_2, F_3, F_4, F_5, F_6). \tag{1}$$

The second one relates each face F to an *ordered* set of four face-edges:

$$\forall\, F \in \{F\} :\ F \mapsto (E_1, E_2, E_3, E_4). \tag{2}$$

The third one relates each edge E to an *ordered* set of two edge-vertices:

$$\forall\, E \in \{E\} :\ E \mapsto (V_1, V_2). \tag{3}$$

Compound faces and compound edges are introduced in order to allow partial block boundary interfacing. A compound face consists of two subfaces. Each subface of a compound face is also allowed to be compound again. A face which is not compound is elementary. The set of compound and elementary faces are denoted as $\{F^c\}$ and $\{F^e\}$. Thus $\{F\} = \{F^e\} \cup \{F^c\}$. The fourth connectivity relation

$$\forall\, F \in \{F^c\} :\ F \mapsto (F_1, F_2). \tag{4}$$

gives the two subfaces of each compound face.

Similarly, a compound edge consists of two subedges. The set of compound and elementary edges are denoted as $\{E^c\}$ and $\{E^e\}$. Thus $\{E\} = \{E^e\} \cup \{E^c\}$, and the mapping

$$\forall\, E \in \{E^c\} :\ E \mapsto (E_1, E_2). \tag{5}$$

gives the two subedges of each compound edge.

For a particular multi-block system, the topology file, which is an output file of the domain decomposer and an input file of the grid generator, contains nothing else as the definition of these five connectivity relations.

The five connectivity relations should obey certain rules such that the topology is meaningful. For instance, a block should contain exactly twelve edges and eight vertices, and a face should contain exactly four vertices. The rules are such that the block-edges, block-vertices and face-vertices are uniquely defined.

The ordering in the relations (1),(2) and (3) are also used to define the orientation of all blocks, faces and edges in a particular multi-block system. In a block, the first coordinate runs from face F_1 to face F_2, the second coordinate runs from face F_3 to face F_4, the third coordinate runs from face F_5 to face F_6. Thus the three pairs of opposite faces in a block are (F_1, F_2),(F_3, F_4) and (F_5, F_6). In a face, the first coordinate runs from edge E_1 to edge E_2, the second coordinate runs from edge E_3 to edge E_4. Thus (E_1, E_2) and (E_3, E_4) are the two pairs of opposite edges in a face. Finally, the coordinate direction of an edge is from vertex V_1 to vertex V_2. The ordering in relations (4) and (5) is of no importance.

The relation (1),(2) and (3) are also used to identify all kinds of geometrical degenerations. For instance, an edge E with two equal vertex labels, i.e. $E \mapsto (V_1, V_2)$ and $V_1 = V_2$, is an edge whose curve-shape is degenerated to a point. This is consistent because the orientation of an edge with two equal vertex labels is undefined. A face with two equal opposite edges, i.e. $F \mapsto (E_1, E_2, E_3, E_4)$ and $E_1 = E_2$ or $E_3 = E_4$, is face with a surface shape degenerated to a curve. If both pairs of opposite edges are equal then the surface shape is a point.

2.2 Geometry definition

The geometrical shape of a vertex is a physical point. The geometrical position of all vertices in a particular multi-block system are stored on the geometry file.

The geometrical shape of an elementary edge is a curve. A default elementary edge is an edge of which the edge-curve shape is a straight line segment between the two edge-vertices.

A non-default elementary edge is an edge of which the edge-curve shape is described by an ordered one dimensional array of physical (control) points. The ordering is defined by the topology relation (3): the first control point is close to the position of vertex V_1, the last control point is close to the position of vertex V_2. Cubic Hermite interpolation between the control points is used to describe the curve continuously. A correction procedure is automatically performed such that the curve matches the two edge-vertices exactly. The control points of the non-default elementary edges in a particular multi-block system are also stored on the geometry file.

The geometrical shape of an elementary face is a surface. A default elementary face is a face of which the face-surface shape is defined by the geometrical curve shape of the four face-edges only (by bilinear transfinite interpolation in which arc length scaled coordinates are used). A non-default elementary face is a face of which the geometrical surface shape is described by a well ordered two dimensional array of physical (control) points. Note that the geometrical representation of a non-default elementary face is the same as for the input configuration surfaces. The topology is again used to define the ordering of the control points: relation (2) is used for faces. Bicubic Hermite interpolation between the control points is used to describe the surface continuously. A correction procedure is automatically performed such that the surface matches the four face-edge curves exactly. The control points of the non-default elementary faces in a particular multi-block system are stored on the geometry file.

2.3 Interactive domain decomposition

The purpose of the domain modeller is to give users the capability to produce, inspect and modify the topology and geometry file. Hence, the topology and geometry files are input and/or output files of the domain modeller.

The domain modeller allows the content of the topology and geometry file not to be a complete topology, nor a single structure: this gives the users the capability to interrupt the work-session and to restart it at any time. The user can also create temporary dummy entities for reference or to help the building of more complex structures. When the user delivers the topology and geometry files to the grid generator, the not necessary faces, edges and vertices must be removed.

The functions of the domain modeller can be grouped into four sets: the block decomposition functions, the topological object management functions, the input functions and the view manipulation functions.

Conceptually, the only actions performed by the domain modeller are creations, killings and inspections of topological entities. Many different ways to do it are provided by the block decomposition functions.

The user usually starts with a reference geometry, which may be delivered as a set of input configuration surfaces and which is converted into a set of vertices, edges and faces by the domain modeller. The reference geometry can be manipulated in several ways. The user can take the input faces as final block-faces, or he can reface them by creating the block faces as close as possible to the reference geometry, and then projecting them onto

it.

During a typical block decomposition session, vertices and non-default edges and faces are created (or copied from external files). A new block is created by indicating (by mouse) eight block-vertices. The required edges and faces of the new block are then automatically detected and, when they do not already exist, created by default procedures. Usually only a few edges and faces have to be created explicitly; the vast majority is produced by default procedures. Some specific functions help to create new entities by geometrical manipulation of existing ones.

The user can read several sets of topology and geometry files during a work-session: the resulting topology will be the union of the contents of each set. Separate sets of entities can be connected by replacing entities of one set with entities of the other.

Because a complete topology may consist of hundreds of blocks, and it is pratically impossible to handle more than 10 to 20 blocks at once, it is necessary to give the users the capability to operate with sub-sets of topological entities within the domain modeler. Those sub-sets are called "topological objects". A topological object is a topologically consistent set of entities: it means that each entity contained in a topological object is defined by entities which belong to the same object. For example, if an edge is present, also its two vertices must be there.

With the topological object management functions the user can copy and delete entities into and from an object, and he can move entities from an object to another one. When new entities are created, they are automatically copied into a pre-defined object (the so-called "default object"). The topological objects can be set visible or not visible on the graphical screen. The user can select the default object, can assign a name to an object, can list the contents of an object on the screen, and can visualise, with different interpolation modes, the edges and the faces contained in an object. Since the content of each single topological object is a consistent topology, it is possible to write and read the topology and geometry file related to one single object only.

The concept of a topological object as a set of entities is meaningful even if no graphic screen is available. The topological objects are also used by some functions as input sets of entities.

To execute a block-decomposition or a topological object management function, it is necessary to prepare a set of input data. This set contains a code number, which identifies the command to be executed, and usually some other values, depending on the command itself. In order to provide a standard way to input data, whatever command has to be executed, a set of so-called "input functions" was designed. Those functions represent the interface between the user (the keyboard) and domain modeller. It is possible to check the input data before the command is actually executed, and to correct any value, when it is wrong, without re-entering the complete set of data. The command code itself can be changed, without re-entering the other data, when they are correct.

The input functions give the user a complete freedom to choose the sequence of input. The sequence of input functions selected by the user is listed in a so called history file. The history file records a complete block decomposition session and it may therefore be used to replay the block decomposition.

Finally, the view manipulation functions give the capability to visualize the entities on the graphic screen by rotating, translating, zooming, etc.

More details about domain decomposition may be found in [4].

3 Grid generation

The topology and geometry files are input files for the grid generator. Two other files are also important during grid generation: – a grid dimension file which contains the information for the specification of the grid dimensions of the multi-block grid, and – a grid control file which contains the grid control parameters for tuning of the grid. These two files are input and/or output files of the interactive grid generator.

A batch version of the grid generator is also available. The batch version is operational on a supercomputer (NEC-SX3) and is especially useful to create fine grids. The general way of working is as follows. Coarse or medium grids are generated during an interactive grid generation session. The generated grids are defined according to the user specified grid dimensions and grid control parameters. When the user is satisfied about the grid quality of the created grids, then the grid dimensions and grid control parameters are written to the grid dimension file and grid control file which are then output files of the interactive grid generator. Next, the grid dimensions are enlarged by a constant factor (the user has to modify, by an editor, only one number on the grid dimension file), and the complete set of four input files (topology file, geometry file, grid dimension file and grid control file) is sent to the super computer where a fine grid is generated by the batch version of the grid generator. This way of working is successful because of the fact that all grid control parameters have a relative meaning with respect to the grid dimensions.

In subsection 3.1 it is described how the grid dimensions of a particular multi-block system are easily defined. The next three subsections describe the main grid generation procedures for, respectively, edges, faces and blocks. Local grid refinement is described in subsection 3.5. The general way of working during interactive grid generation is described in subsection 3.6.

3.1 Grid dimension specification

The specification of the grid dimensions; i.e. the number of grid cells, of all blocks, faces and edges in a particular multi-block system requires the grid dimension specification of only a few suitable chosen edges. This is due to the constraining effect of the requirement that each grid line in each block must be continuous over any face that the block has in common with any adjacent block. These constraint relations depend only on the topology: each two opposite edges in a face must have the same grid dimension, and each four opposite edges in a block must have the same grid dimension. This observation makes it possible to subdivide the set of edges $\{E\}$ into disjunct sets (called groups) with the property that the grid dimension of all edges in the same group must be the same, while the grid dimensions of two edges in different groups are generally different. Furthermore,

simple sum relations between the grid dimensions of different groups may exist due to the existence of compound edges.If, for instance, a compound edge E with subedges E_1, E_2 belongs to the groups G and G_1, G_2, respectively, then it is clear that the grid dimension of group G is equal to the sum of the grid dimensions of groups G_1 and G_2.

The groups and the sum relations between the groups are automatically generated from the topology. Suppose that a particular multi-block system contains N groups with M sum relations between the groups. Then there are only $N - M$ independent grid dimensions, and the user has to specify the grid dimensions of only $N - M$ suitable chosen edges in order to define the grid dimensions of all groups, and consequently, of all edges, faces and blocks.

3.2 Grid generation in elementary edges

In subsection 2.2 it is described how a smooth curve is constructed for each elementary edge. Such a smooth curve is parameterized according to

$$P_E : s \in [0,1] \mapsto (x,y,z) \in \mathcal{R}^3, \tag{6}$$

where s is the scaled arc length. The orientation is defined by the topology: s runs from vertex V_1 to vertex V_2. The function P_E is called the geometrical shape function of an edge. A grid control function G_E of the form:

$$G_E : \xi \in [0,1] \mapsto s \in [0,1], \tag{7}$$

maps the computational ξ space onto the s space. The orientation of the computational ξ coordinate is the same as for the scaled arc length coordinate s. A grid distribution function maps the computational ξ space onto the edge curve and is defined as the composite mapping $P_E o G_E(\xi) = P_E(G_E(\xi))$. Thus the grid points of an edge with N grid cells are computed according to

$$P_E o G_E(\xi_i),\ \xi_i = i/N,\ i = 0 \ldots N. \tag{8}$$

A grid which is equally distributed along an edge is obtained by taking $s = G_E(\xi) = \xi$. The general form of the function $G_E(\xi)$ is taken as

$$G_E(\xi) = \int_0^{\xi} \exp(\sum_{i=0}^{4} \alpha_i \eta^i) d\eta, \tag{9}$$

where the five coefficients $\alpha_i, i = 0 \ldots 4$ are constants. The chosen form of the function G_E implies that the corresponding stretching function, defined as G_E''/G_E', is a polynomial function. At an elementary edge a user may specify two boundary conditions at each vertex, so that at most four boundary conditions exist. These four boundary conditions, together with the constraint

$$\int_0^1 \exp(\sum_{i=0}^{4} \alpha_i \eta^i) d\eta = 1, \tag{10}$$

are used to compute the five coefficients $\alpha_i, i = 0 \ldots 4$. When the number of boundary conditions is less than four, then the five coefficients are still uniquely determined by demanding that the degree of the polynomial stretching function is as low as possible.

3.3 Grid generation in elementary faces

The surface shape of an elementary face is parameterized by a geometrical shape function of the form:

$$P_F : (s,t) \in [0,1]^2 \mapsto (x,y,z) \in \mathcal{R}^3. \tag{11}$$

The orientation of s and t is defined by the topology: s runs from edge E_1 to E_2 and t runs from edge E_3 to E_4. At the boundary of the unit square in the (s,t) space, the function P_F coincides with the geometrical shape functions of the four face-edges:

$$\begin{aligned} P_F(s,0) &= P_{E_3}(s) \quad , \quad P_F(s,1) = P_{E_4}(s), \\ P_F(0,t) &= P_{E_1}(t) \quad , \quad P_F(1,t) = P_{E_2}(t). \end{aligned} \tag{12}$$

Thus s and t are scaled arc lengths along the boundary of the surface.

Similarly as for elementary edges, a grid control function is used to map the computational (ξ,η) space onto the (s,t) space:

$$G_F : (\xi,\eta) \in [0,1]^2 \mapsto (s,t) \in [0,1]^2. \tag{13}$$

The grid distribution function $P_F o G_F$ maps the computational space onto the surface. The orientation of the (ξ,η) space is the same as for the (s,t) space. The grid points of a face with $N \times M$ grid cells are found by

$$P_F o G_F(\xi_i,\eta_j),\ \xi_i = i/N,\ \eta_j = j/M,\ i = 0 \ldots N,\ j = 0 \ldots M. \tag{14}$$

The grid generation in an elementary face is preceded by the grid generation in the four face-edges. Thus the grid points along the four face-edges are known and also their corresponding s and t values. What remains is the computation of the grid points in the interior of the face.

The computation of the grid control function G_F is equivalent with the computation of the two functions

$$s = s(\xi,\eta)\ ,\ t = t(\xi,\eta). \tag{15}$$

These two functions are known at the boundary of the unit square in the computational domain:

$$\begin{aligned} s(\xi,0) &= s_{E_3}(\xi) \quad , \quad s(0,\eta) = 0, \\ s(\xi,1) &= s_{E_4}(\xi) \quad , \quad s(1,\eta) = 1, \\ t(0,\eta) &= t_{E_1}(\eta) \quad , \quad t(\xi,0) = 0, \\ t(1,\eta) &= t_{E_2}(\eta) \quad , \quad t(\xi,1) = 1. \end{aligned} \tag{16}$$

Note that the functions $s_{E_3}, s_{E_4}, t_{E_1}, t_{E_2}$ are edge grid control functions of the form of Eq. (9) and are thus monotonously increasing.

A simple and robust way to compute (s,t) for values of (ξ,η) in the interior of the unit square is provided by next two equations:

$$s = s_{E_3}(\xi)(1-t) + s_{E_4}(\xi)t, \tag{17}$$

$$t = t_{E_1}(\eta)(1-s) + t_{E_2}(\eta)s. \tag{18}$$

Eq. (17) implies that a grid line ξ = const. is mapped to the unit square in the (s,t) space as a straight line: s is a linear function of t. Eq. (18) implies that a grid line η = const. is also mapped to the unit square in the (s,t) space as a straight line: t is a linear function of s. For given values of ξ and η the corresponding s and t values are found as the intersection point of the two straight lines. It can be easily verified that the grid control function which corresponds to this system has a positive Jacobian i.e. $J = s_\xi t_\eta - s_\eta t_\xi > 0$.

However, the system (17),(18) is completely determined by the grid point distribution along the four face-edges and provides no grid control about the grid line slopes and first grid cell lengths. In order to have more grid control, the system is extended by

$$s = s_{E_3}(\xi)(1-t) + s_{E_4}(\xi)t + f(\xi,t), \tag{19}$$

$$t = t_{E_1}(\eta)(1-s) + t_{E_2}(\eta)s + g(\eta,s). \tag{20}$$

In this grid generation system, Eq. (19) implies that a grid line ξ = const. is mapped to the unit square in the (s,t) space as a curve which can be described as: s is a function of t, and Eq. (20) implies that a grid line η = const. is mapped to the unit square in the (s,t) space as a curve which can be described as: t is a function of s. Again, for given values of ξ and η the corresponding s and t values are found as the intersection point of these two curves.

The functions f and g are correction functions with respect to the straight lines of system (17),(18). Hence, f and g are identical zero at the boundary of the unit squares in respectively the (ξ,t) and (η,s) space. The normal derivatives of f and g at the boundary may be used to define the grid line slopes and first grid cell lengths.

The values of $\partial f/\partial t(\xi,0)$ and $\partial f/\partial t(\xi,1)$ may be used to define the grid line slopes of the grid lines ξ = const. along the face-edges E_3 and E_4, respectively. The values of $\partial f/\partial \xi(0,t)$ and $\partial f/\partial \xi(1,t)$ may be used to control the first grid cell lengths of the grid lines η = const. along the face-edges E_1 and E_2. Similar remarks can be made for the derivatives of the function g. More details about the relation between the normal derivatives of f and g at the boundary of the unit squares and the corresponding grid line slopes and first grid cell lengths at the boundary of the face in physical space may be found in [5].

When the normal derivatives of f and g are defined, then it remains to compute f and g in the interior of the unit squares. This is done by solving the biharmonic equations [6]:

$$\triangle\triangle f = 0, \tag{21}$$

on the unit square $(\xi,t) \in [0,1]^2$, and

$$\triangle\triangle g = 0, \tag{22}$$

on the unit square $(\eta,s) \in [0,1]^2$.

The user control about the grid line slopes is especially useful at singularities. An example is shown in Fig. 2. Such O-type meshes are needed at trailing edges of airfoils, wings, exhaust outlets, etc.

For both grid generation systems (17),(18) and (19),(20), the two families of grid lines in the (s,t) space are determined independently and a grid point is found as the intersection

point of two individual grid lines of each family.This approach is especially successful for Navier-Stokes grids (boundary layers) where the characteristics of the two families of the grid lines are totally different.

3.4 Grid generation in blocks

The grid generation in a block is preceded by grid generation in the six block-faces. Thus, the grid point distribution on the six block faces is known, and what is left to be done is the computation of the grid in the interior of the block. Two methods are used to compute the interior grid points in a block.

The first method is based on trilinear transfinite interpolation in the computational (ξ, η, ζ) space. Thus linear blending functions are used: $\xi, (1-\xi), \eta, (1-\eta), \zeta, (1-\zeta)$.

The second method is an extension of grid generation system (17),(18) for faces. A geometrical shape function is constructed which defines the volume shape of a block:

$$P_B : (s,t,u) \in [0,1]^3 \mapsto (x,y,z) \in \mathcal{R}^3. \tag{23}$$

where the orientation of (s,t,u) is again defined by the the topology. At the boundary of the unit cube in (s,t,u) space, the function P_B coincides with the geometrical shape functions of the six block faces. In fact, the function P_B is constructed by trilinear transfinite interpolation in (s,t,u) space.

A mapping from computational (ξ, η, ζ) onto the (s,t,u) is defined as

$$\begin{aligned}
s &= s_{E_1}(\xi)(1-t)(1-u) + s_{E_2}(\xi)t(1-u) + s_{E_3}(\xi)(1-t)u + s_{E_4}(\xi)tu, && (24)\\
t &= t_{E_5}(\eta)(1-s)(1-u) + t_{E_6}(\eta)s(1-u) + t_{E_7}(\eta)(1-s)u + t_{E_8}(\eta)su, && (25)\\
u &= u_{E_9}(\zeta)(1-s)(1-t) + u_{E_{10}}(\zeta)s(1-t) + u_{E_{11}}(\zeta)(1-s)t + u_{E_{12}}(\zeta)st, && (26)
\end{aligned}$$

where the edges (E_1, E_2, E_3, E_4),(E_5, E_6, E_7, E_8), and $(E_9, E_{10}, E_{11}, E_{12})$ are the sets of four block-edges in respectively the $s-$,$t-$ and $u-$ direction. The functions $s_{E_1}, \ldots, u_{E_{12}}$ are the corresponding edge grid control functions. For given values of ξ,η and ζ, the corresponding s,t and u values are computed by solving Eqs. (24),(25),(26) simultaneously. It can be easily verified that the grid control function which corresponds to this system has a positive Jacobian. Finally, the grid points in the interior of a block are computed by the grid distribution function $P_B o G_B$.

Both methods usually provide a good degree of clustering throughout the grid, but local regions of crossed grid lines, corresponding to negative values of cell volumes sometimes result. However, it appears that negative cell volumes occur much more seldomly when the second method is applied (then grid folding is caused by the geometrical shape function P_B).

3.5 Local Grid refinement

A multi-block grid with the property that each grid line in a block is continuous over any face that the block has in common with any adjacent block is called a basic multi-block

grid. The grid dimensions of a basic multi-block grid are defined in a way as described in subsection 3.1.

However, it is very useful that the grid in a particular block can be refined (coarsened) without changing the grid in the surrounding blocks, so that refined (coarsened) grids can be used in blocks located in regions where large (small) flow gradients are expected.

The grid refinement (coarsening) in a particular block is user-specified by three grid refinement/coarsening factors in each computational direction of that block. Of course, if grid coarsening is applied in a certain computational direction then the grid dimension in that direction must be dividable by the corresponding coarsening factor.

There is one restriction about the way local grid refinement is applied. At a particular internal block-face, there are in general two different grids which belong to the two blocks which have this block-face in common. The restriction is that one of the two grids in the block-face is coarse with respect to the other, so that a grid cell in the coarse grid may be connected to a number of fine grid cells in the fine grid. This property facilitates a flow solver to remain conservative across block-faces: the flux through a coarse grid cell-face is given by the sum of the fluxes through the corresponding fine-grid cell-faces [7]. An example of the application of local grid refinement is shown in Fig. 3.

3.6 Interactive grid generation

The interactive grid generator is operational on the Silicon Graphics Iris 4D workstation family.

During an interactive session, the first action of a user is to read the topology and geometry file of a particular multi-block system. After that, a selection panel is available to visualize the topology and geometry. The panel contains all block labels and the user may select (by mouse) a number of blocks from the panel. The result is a screen which depicts the selected blocks. The topology of each block is represented by the six block-face labels, all elementary and compound face labels of which a block-face may consist of, all corresponding elementary and compound edge labels, and all vertex labels (the total number of vertices is in general more than eight). The geometry of each block is represented only by the geometrical shape of all elementary edges on the block boundary (the total number of elementary edges on the block boundary is more than twelve in general).

Next, the grid dimensions are specified. The mouse is used to select an edge from the screen, and a number is given (by keyboard) which defines the number of grid cells along the edge. With the group concept described in subsection 3.1, the program automatically identifies those edges which obtain the same grid dimension value. If there are compound edges, then the sum relations between the groups are used to check if it is possible to compute the dimensions of other edges. If an edge dimension is known then the colour of the edge on the screen is changed and the grid dimension value appears at the middle of the edge.

This process is repeated until the dimensions of all edges are known. The user may then write the grid dimension file which defines the grid dimensions of the multi-block system.

If a grid dimension file already exists then the process of specifying the grid dimensions may be skipped and the user can simply read the grid dimension file.

When the grid dimensions are known, a second selection panel becomes available which contains the i, j, k values of all grid planes in each block. If the user selects some i, j or k values of some blocks then the corresponding grid planes in the blocks are shown on the screen. Thus the second selection panel is used to specify which grid planes must be visualized while the first selection panel is used to specify which blocks must be visualized topologically. In the interactive grid generator there are no explicit commands to compute a grid in some edges, faces or blocks, but instead of that, the visualization of grid planes in blocks is automatically converted to commands to compute the grids in the corresponding block-faces (if a grid plane in a block corresponds with a block-face) or blocks (if a grid plane is an interior grid plane).

Next, the grid tuning process may start. Along elementary edges, grid control is available only at the two vertices of the edges. The user may specify a grid density ρ at a vertex of an elementary edge by selecting the edge and vertex (via their labels) from the screen by mouse and by defining ρ by keyboard. The result is that the first grid cell length along the edge at the vertex becomes $\rho L/N$ where L is the lenth of the edge and N is the number of grid cells along the edge. It is also possible to specify a stretching value R at a vertex of an elementary edge. The result is that the ratio between the second and first grid cell length at the vertex becomes $1 + (R/N)$. Finally, the user may "connect" edges. If an edge E_2 at vertex V_2 is connected to an edge E_1 at vertex V_1 then the first grid cell length along edge E_2 at V_2 becomes equal to the first grid cell length along edge E_1 at V_1. In this way large chains of connected edges may be constructed, and if the grid in the "mother" edge is changed, the grid in all other edges in the chain are then also automatically changed. The program automatically takes care that a chain is not closed. The connection of edges is very useful to construct smooth grids across elementary edges.

Grid generation in an elementary face is preceded by grid generation in the four face-edges (the program automatically computes the grid in the four face-edges before it computes the grid in the interior of a face). Thus the grid points along the four face-edges are known. Grid generation in elementary faces may be based on system (17),(18) or system (19),(20). If system (17),(18) is used then the grid in an elementary face is determined by the grid point distribution along the four face-edges only. If system (19),(20) is used, then the grid in an elementary face is also determined by the grid line slopes and first grid cell lengths along the four face edges. At the corners of the face, the angle between the two face-edges is known and also the first grid cell lengths along the two face-edges. Default, these angles and grid cell lengths at the corners are interpolated (using arclengths) to define the grid line slopes and first grid cell lengths on the four face edges. Then the boundary conditions are sufficient to solve system (19),(20). However, the user may also specify explicitly the grid line slope and first grid cell length at certain locations along the four face-edges (which is for instance necessary at singularities). In that case, the mouse is used to identify the location along a face-edge, and the grid line slope angle and first grid cell length are specified by keyboard.

Grid generation in blocks is preceded by grid generation in the six block-faces (the

program automatically computes the grid in the six block-faces before it computes the grid in the interior of the block). Thus the grid points on the six block-faces are known. Grid generation in a block may be based on trilinear transfinite interpolation in computational or arclength scaled space. For both methods the grid in the block is determined by the grid point distribution on the six block-faces only. However, grid folding occurs much more seldomly when trilinear transfinite interpolation in arclength scaled space is applied.

During the interactive session the user may change at any moment the grid dimensions or apply local grid refinement. Local grid refinement in a block is performed by specifying the grid refinement/coarsening factors in the three computational directions of a block. It is not necessary to change the values of the grid tuning parameters when the grid dimensions are changed because their values are relative with respect to the grid dimensions.

One interactive session is in general not sufficient to tune the complete grid for a complex configuration. Therefore at the end of a session the user may write the grid dimension and grid control file (which contains all grid tuning parameters) and also a grid file which contains the partially constructed grid. Then, in a next session, the user can read these files and continue the grid generation process.

There are of course more additional "tools" in the interactive grid generator. For example, for grid inspection it is necessary to zoom, rotate and translate. Furthermore, it is possible at any moment to switch on and of the labels of the blocks, faces, edges and vertices which are selected via the first selection panel.

4 Applications

The usability of the applied domain decomposition and grid generation techniques is demonstrated for both aerodynamic and hydrodynamic configurations.

Complex aerodynamic configurations are shown in Figs. 4 and 5. Some results about the numerical flow simulation about the Fokker 50 and Fokker 100 may be found in [8]. Fig. 5a shows the aerodynamic surface of the Alenia transport aircraft G222 and Fig. 5b shows the corresponding grid.

Hydrodynamic configurations are shown in Figs. 6 and 7. Fig. 6 shows the grid in a part of the river Rhine. High grid density is not required at the boundary but in the interior of the river due to a minor bed which is a typical hydraulic feature. The topology contains five blocks. Fig. 7 is an example of a complex hydraulic structure to be built in the river Maas. The resulting topology is shown in Fig. 7a, the surface grid is shown in Fig. 7b. Note that local grid refinement has been applied in one block. Fig. 7c shows the grid in an interior circular plane. Note that such a plane contains four O-type singularities. More details about block decomposition and grid generation for hydrodynamic configurations may be found in [9].

Finally, Fig. 8 illustrates that it is possible to generate Navier-Stokes grids with the existing grid generation techniques.

5 References

[1] J.W. Boerstoel, J.M.J.W. Jacobs, A. Kassies, A. Amendola, R. Tognaccini, P.L. Vitagliano, Design and testing of a multiblock grid generation procedure for aircraft design and research. Applications of Mesh Generation to Complex 3-D Configurations, AGARD-CP-464,1990.

[2] L.E. Eriksson, E. Orbekk, Algebraic Block-Structured Grid Generation Based on a Macro-Block Concept. Applications of Mesh Generation to Complex 3-D Configurations, AGARD-CP-464,1990.

[3] J.W. Boerstoel, S.P. Spekreijse, An Information System for the Numerical Simulation of 3D Euler Flows around Aircraft. Computer Methods in Applied Mechanics and Engineering 89, pp. 237-257, 1991.

[4] P.L. Vitagliano, J.W. Boerstoel, Introduction and Guide to the Use of EDOMO for the Graphical Interactive Decomposition of Flow Domains into Blocks. NLR CR 90265, 1991.

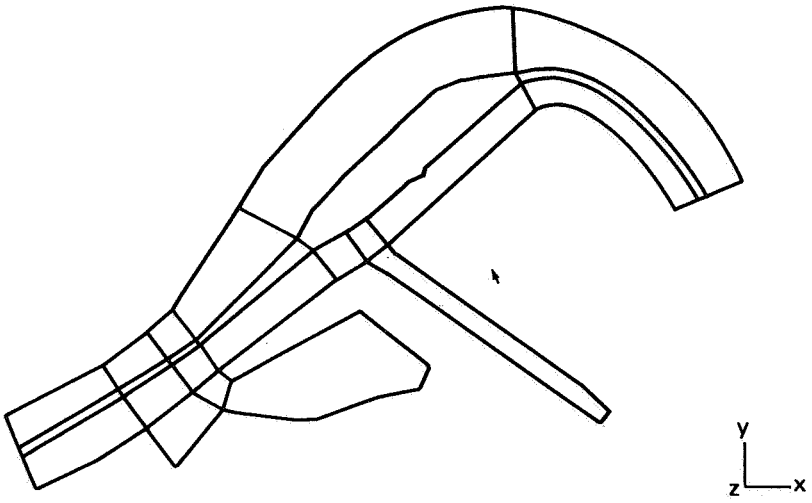
[5] S.P. Spekreijse, J.W. Boerstoel, New Concepts for Multi-Block Grid Generation for Flow Domains Around Complex Aerodynamic Configurations. Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, ed. A.S. Arcilla e.a., Elsevier North-Holland,1991.

[6] P.E. Bjorstad, Numerical Solution of the Biharmonic Equation. Ph.D.dissertation, Stanford University, 1980.

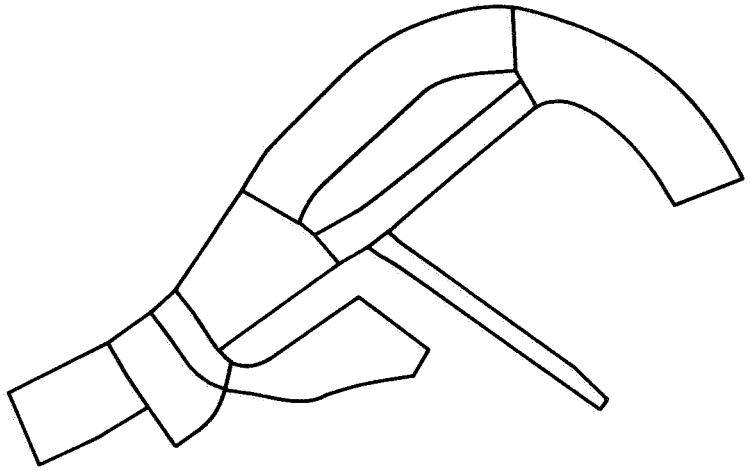
[7] A. Kassies, R. Tognaccini, Boundary Conditions for Euler Equations at Internal Block Faces of Multiblock Domains Using Local Grid Refinement, AIAA Paper 90-1590, 1990.

[8] J.L. Kuyvenhoven, J.W. Boerstoel, 3D Euler Flows Around Modern Airplanes. Proc. Eight GAMM-Conference on Numerical Methods in Fluid Dynamics, Notes on Numerical Fluid Mechanics, 29, ed. P. Wesseling. Vieweg 1989.

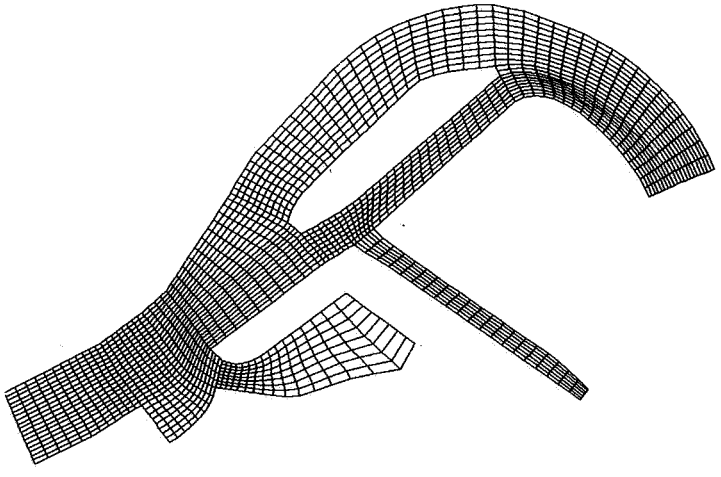
[9] M.J. van der Marel, Evaluation of ENGRID and EDOMO. Delft Hydraulics report Q997,1991.



a) Complete block boundary interfacing



b) Partial block boundary interfacing



c) Grid

Fig. 1 Part of harbour of Rotterdam near Noordereiland

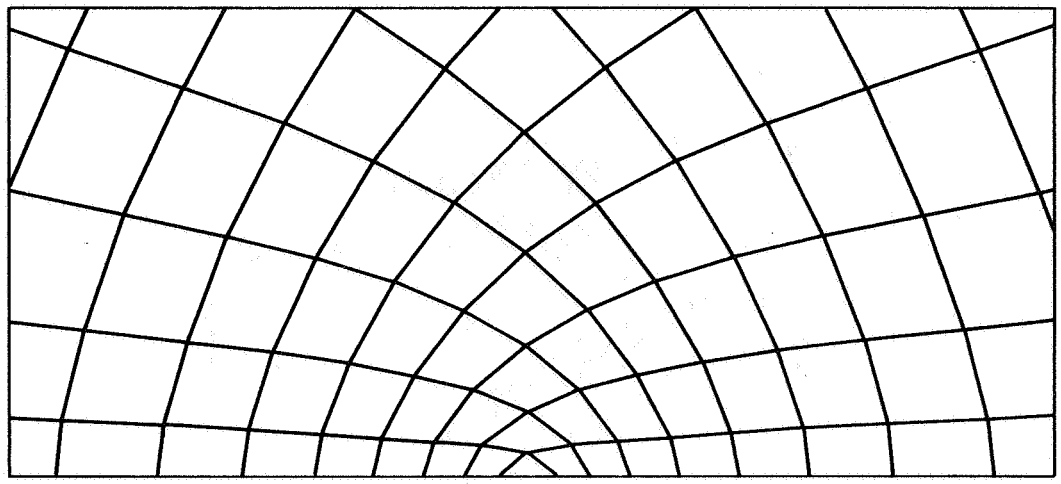


Fig. 2 Grid near O-type singularity

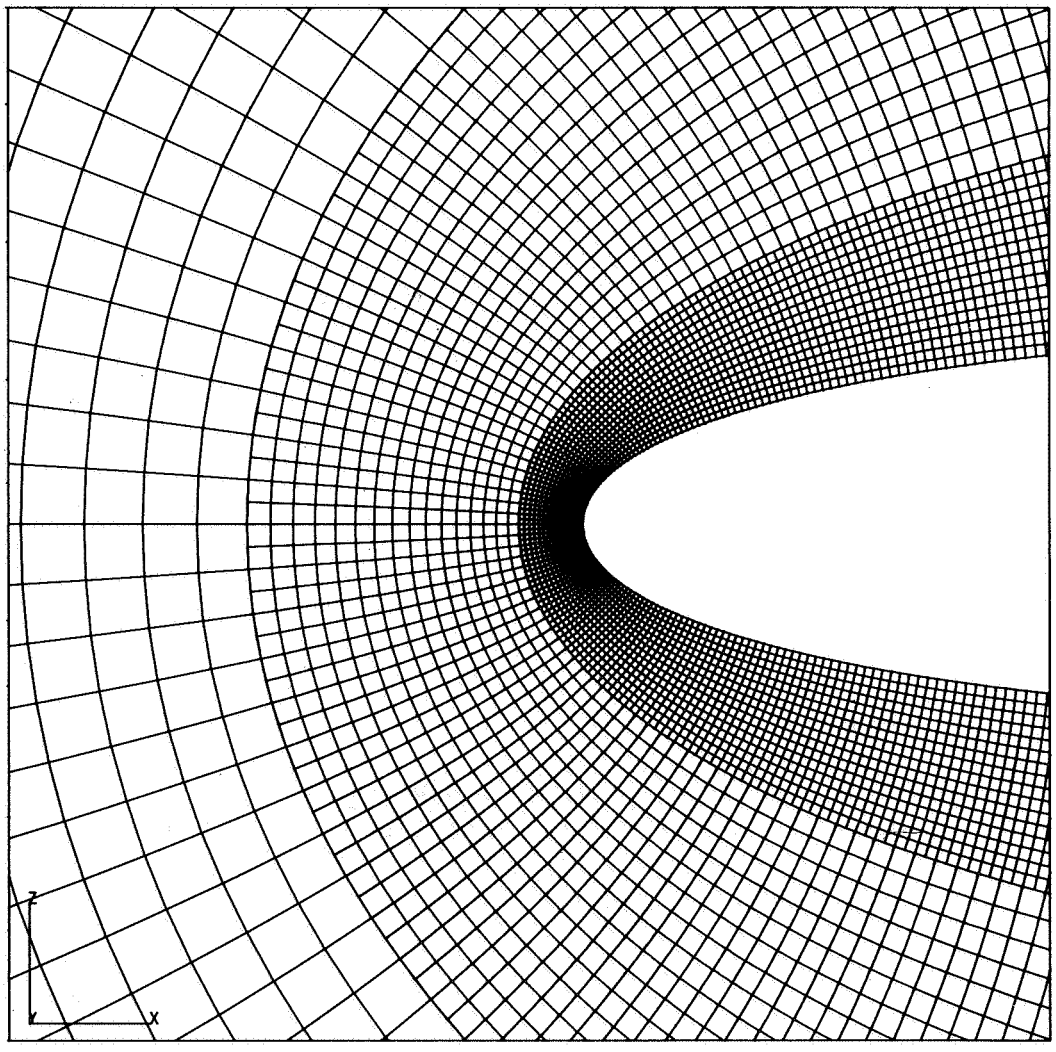


Fig. 3 Leading edge region of 3-block fine grid

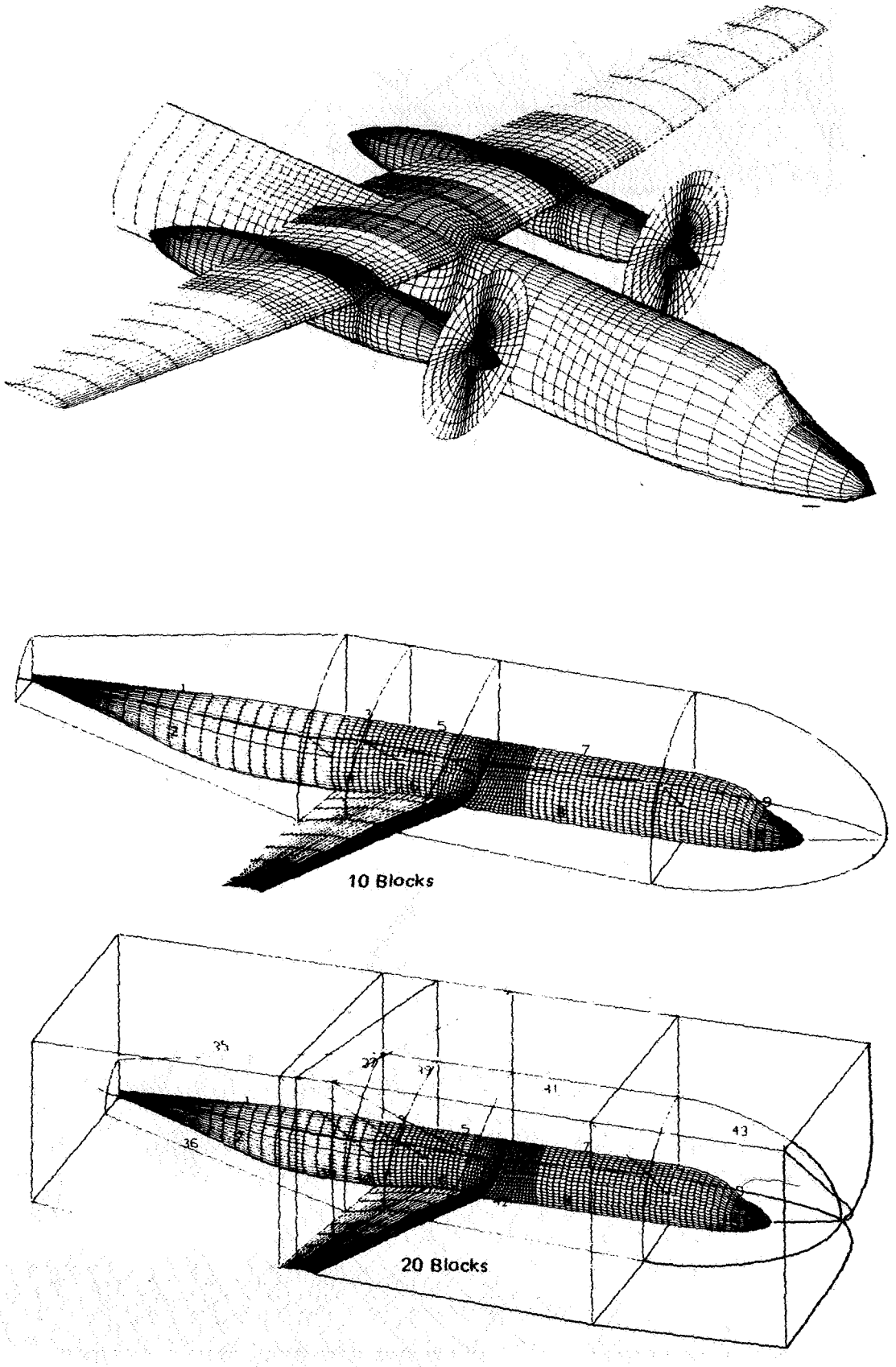


Fig. 4 Fokker 50 and Fokker 100 configurations

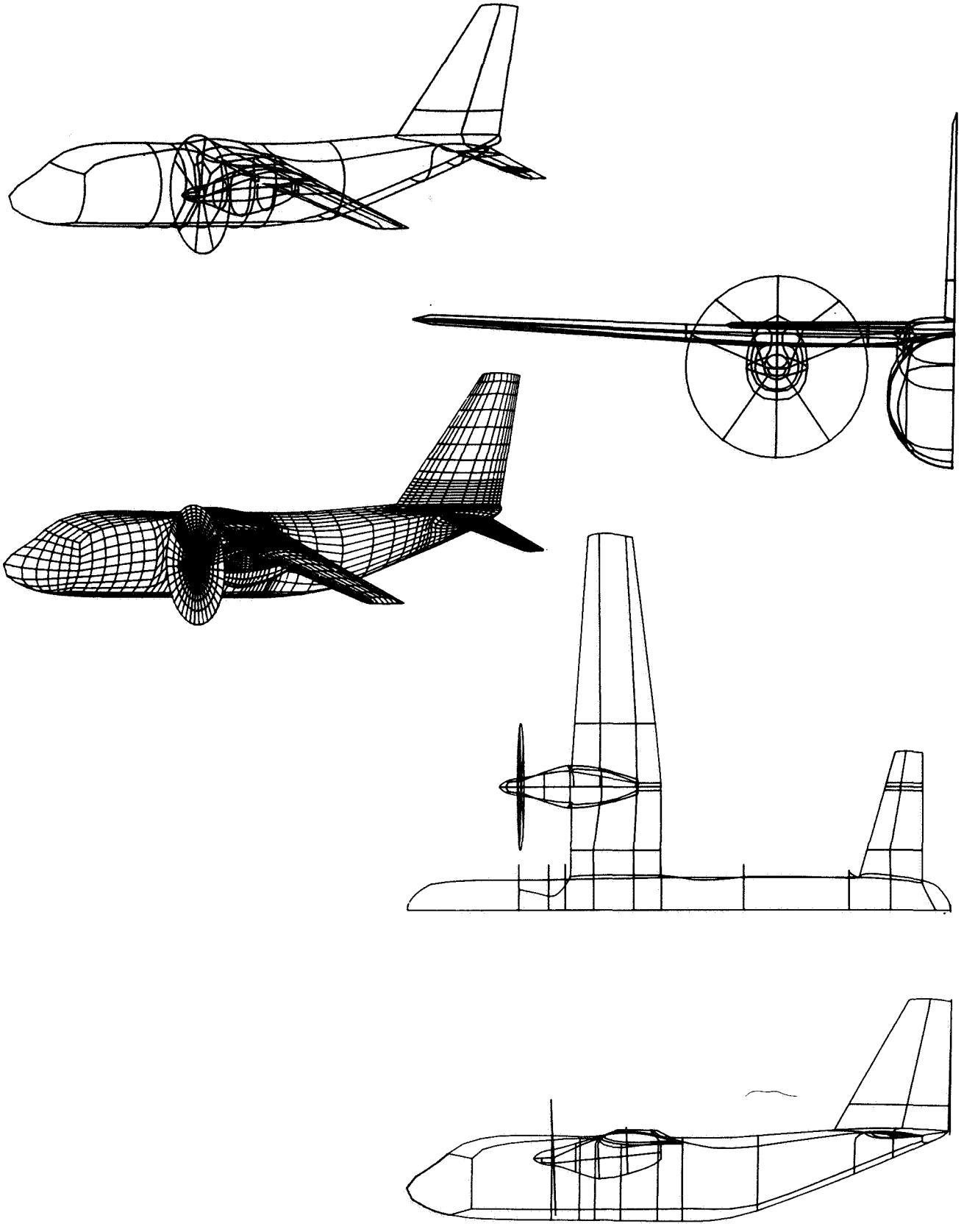


Fig. 5a Aerodynamic surface of Alenia transport aircraft G222

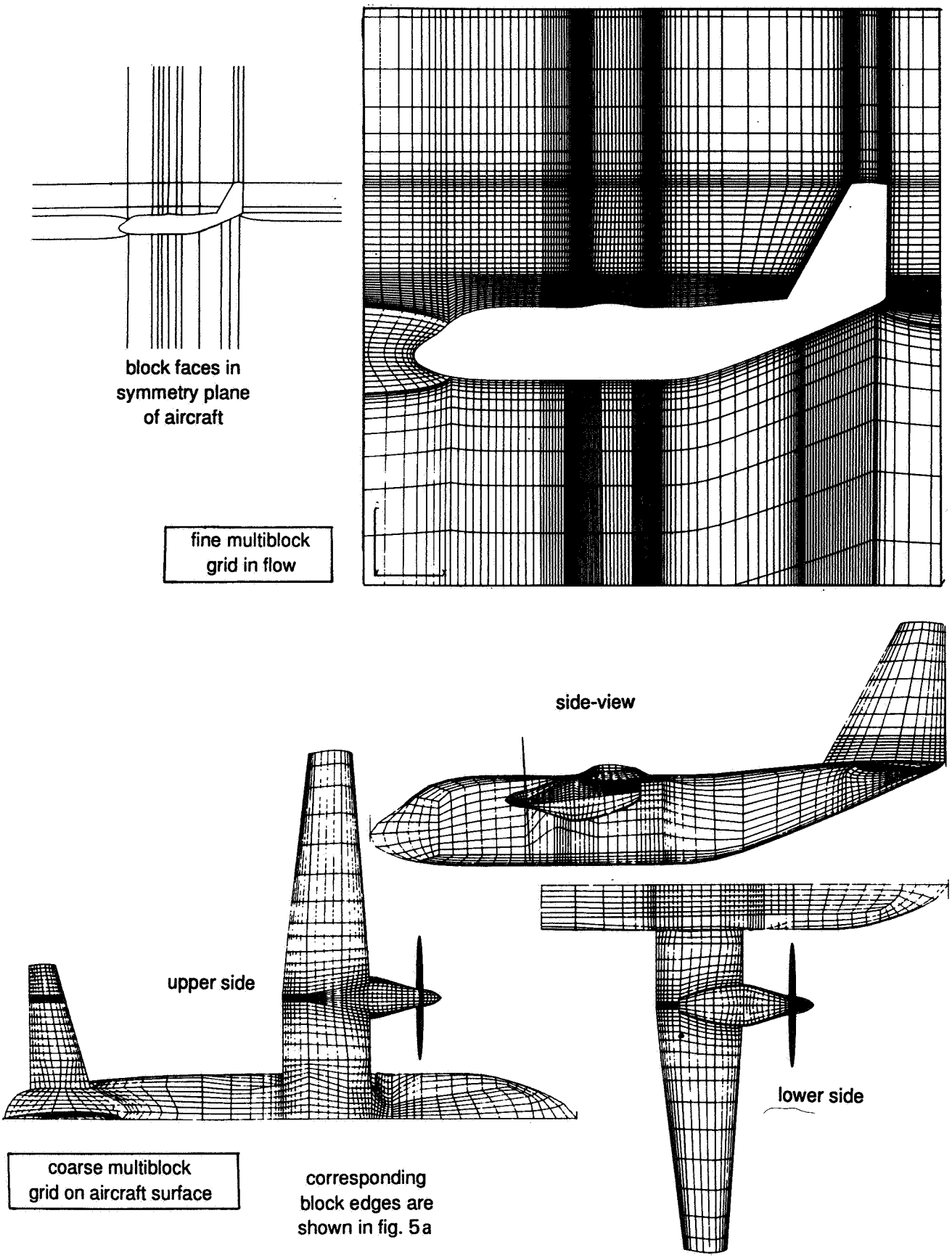


Fig. 5b Corresponding multi-block grid on aircraft surface and symmetry plane

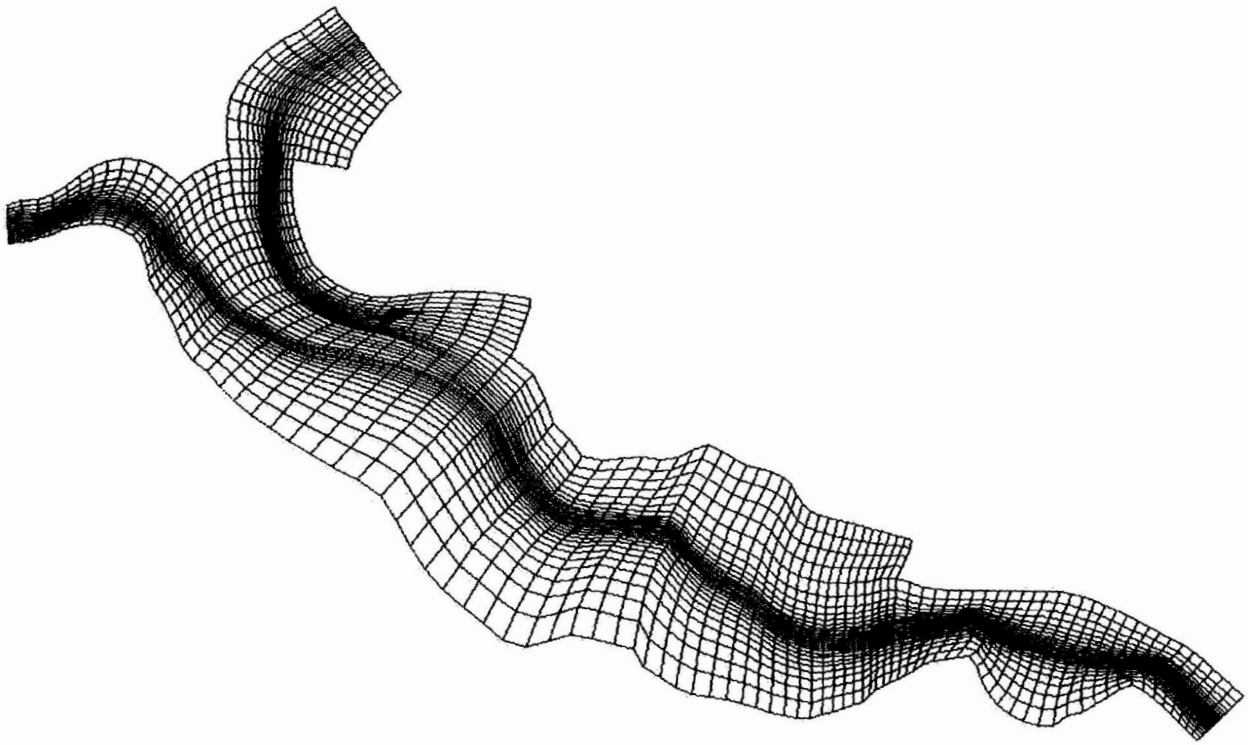
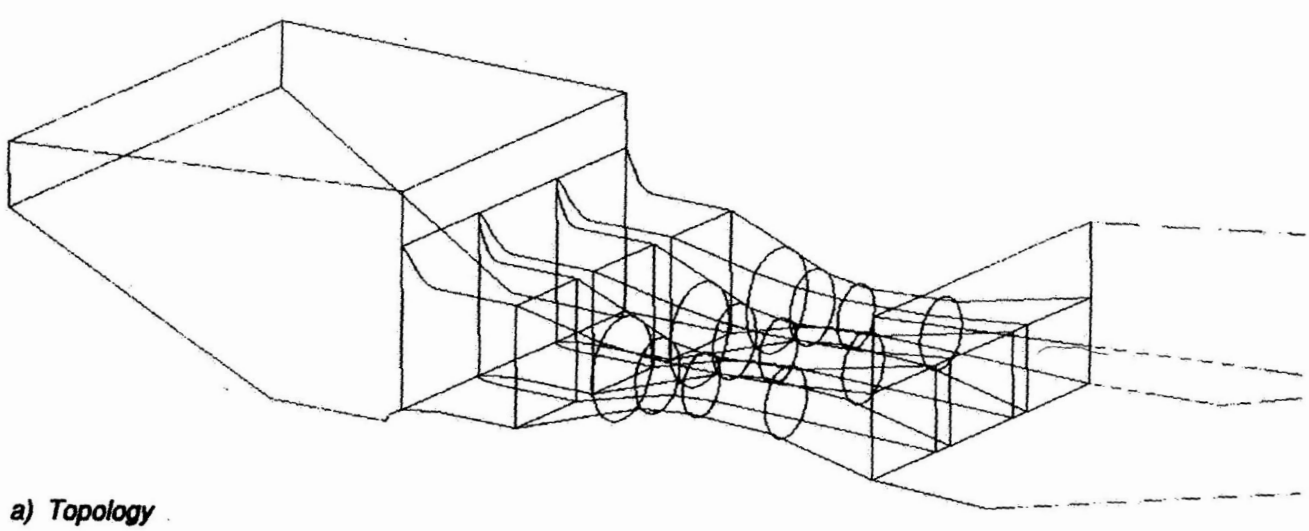
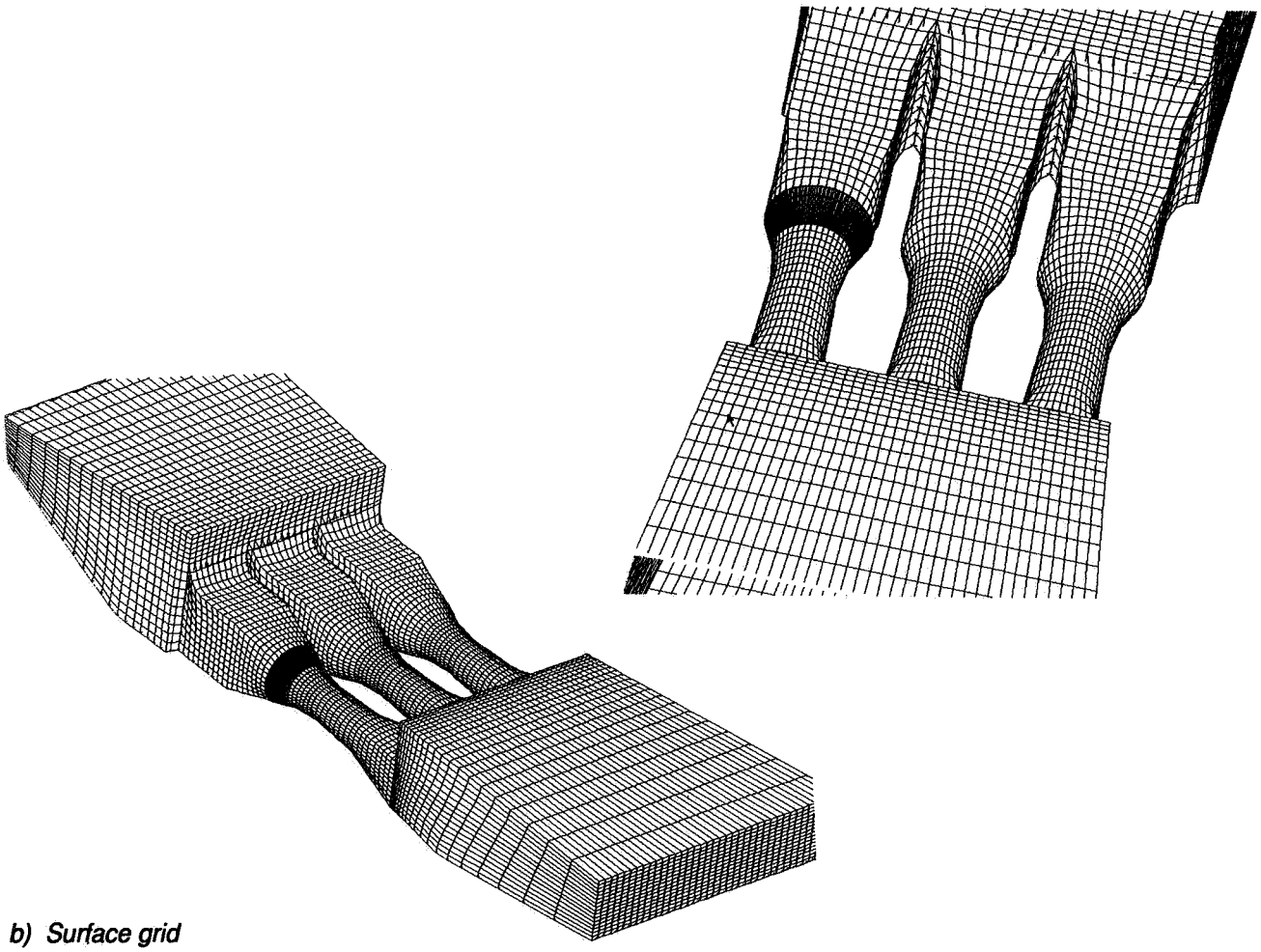


Fig. 6 Grid in part of the river Rhine

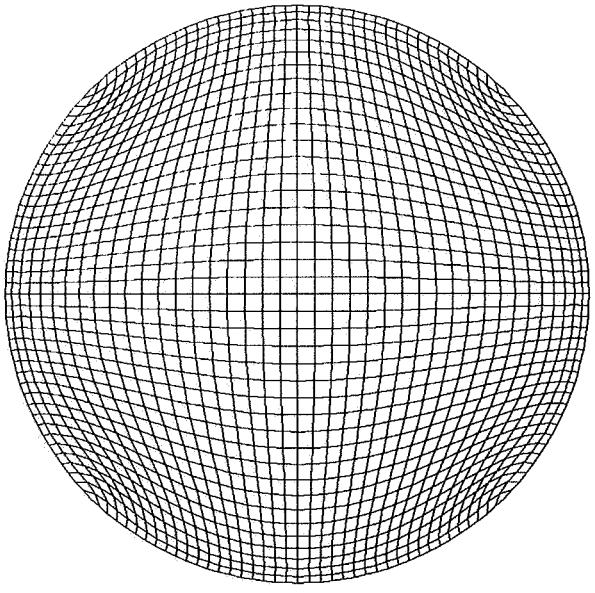


a) Topology

Fig. 7 Water power station



b) Surface grid



c) Grid in interior circular plane

Fig. 7 Water power station (continued)

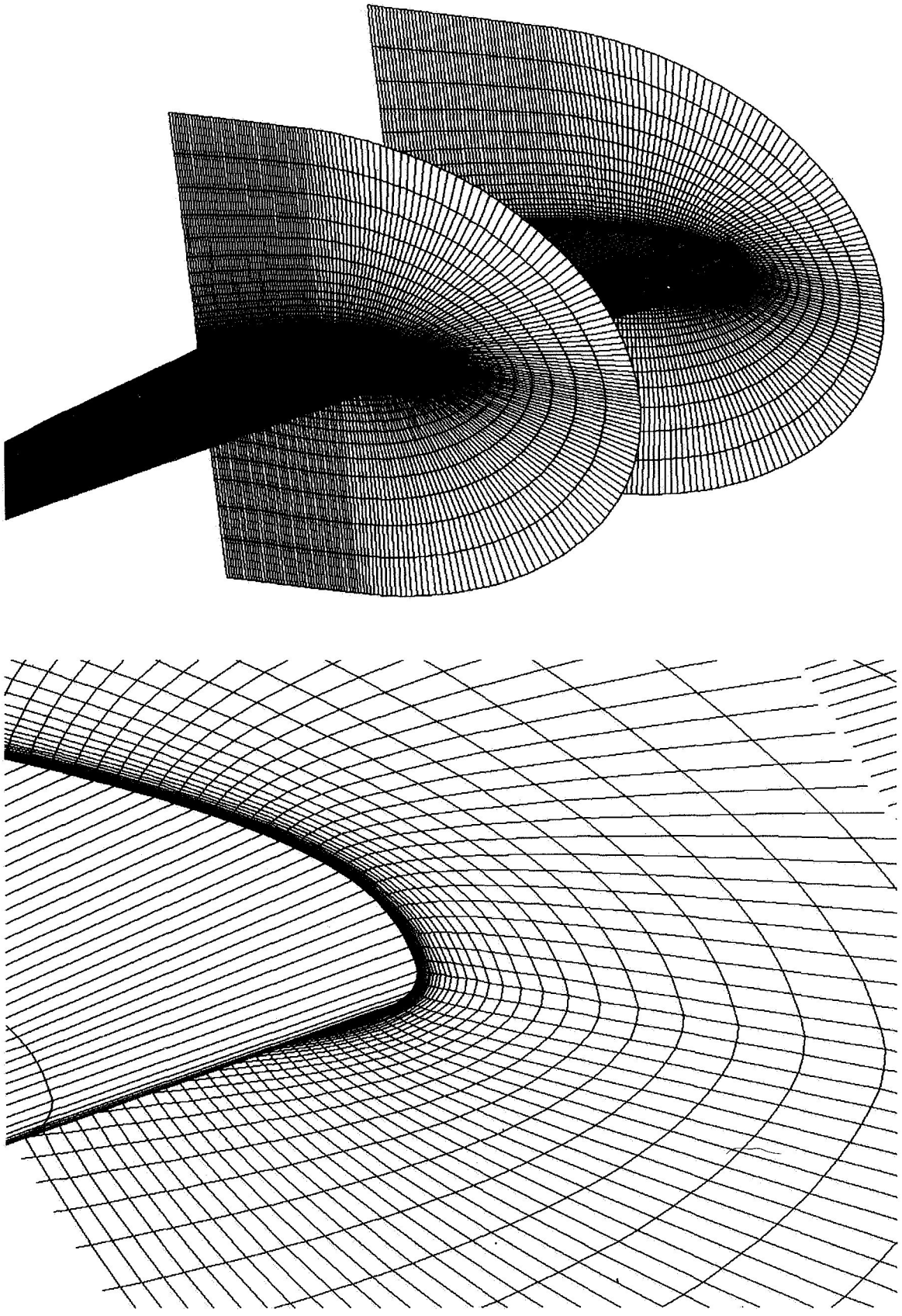


Fig. 8 Navier-Stokes grid around a wing

Towards a Theory of Automated Elliptic Mesh Generation

J.Q. Cordova
Visual Computing Incorporated
Mountain View, CA 94043

Abstract

The theory of elliptic mesh generation is reviewed and the fundamental problem of constructing computational space is discussed. It is argued that the construction of computational space is an NP-Complete problem and therefore requires a non-standard approach for its solution. This leads to the development of graph-theoretic, combinatorial optimization and integer programming algorithms. Methods for the construction of 2-D computational space are presented.

Introduction

Since the introduction of elliptic mesh generation as an all-purpose technology, there have been various attempts to codify the blocking process that must preceed the solving of elliptic equations. From these attempts, three distinct approaches have emerged: interactive, algorithmic, and knowledge based [5], [1], [20]. It is probably fair to say that this ordering also reflects the relative success of each approach. Why should this be? Could it be that human intervention is necessary for solving the blocking problem? Note that human interaction is minimal for unstructured mesh generation. Therefore, why should it be needed for structured mesh generation? This line of inquiry leads to the consideration of combining structured and unstructured mesh generation within a generalized theory.

Structured mesh generation (SMG) and unstructured mesh generation (UMG) belong to the general discipline of computational geometry (CG). Computational geometry is concerned with the systematic study of geometric algorithms [8], [16], [17]. A typical sample of CG problems might include geometric searching, computation of convex hulls, surface-surface intersection, visibility detection, and motion planning. Mesh generation is concerned with the problem of decomposing complex shapes into simpler pieces. The simplest definitions of UMG and SMG as CG problems are as follows:

(UMG) Decompose a region bounded by algebraic surfaces of degree 1 into tetrahedra.
(SMG) Decompose a region bounded by algebraic surfaces of degree ≥ 1 into deformed hexahedra.

While these definitions are not complete, they serve to emphasize the main relationship between unstructured and structured mesh generation. This is that UMG is a linear problem while SMG is a nonlinear problem. In particular, UMG is a special case of SMG since it is easy to construct a tetrahedral decomposition for each hexahedron.

The view that SMG should be developed within the framework of non-linear computational geometry can be motivated in another way. Namely, that it represents a natural generalization of previous theories. This is presented next as an evolution of algorithms A -> B -> C.

The standard formula for elliptic mesh generation is based on a four step procedure:

Algorithm A
I. Generate curves
II. Assemble curves into block faces
III. Set boundary point numbering and point distribution
IV. Solve elliptic equations in each block

The subtle difficulty with this approach is that topological and geometric constructions are intermixed. For example, Step I serves to simultaneously identify an edge in computational space and the mapping of that edge into a curve in physical space. Thus, "block" is used to reference a region in computational space as well as its image in physical space. This shortcoming was realized by Allwright [1], who proposed a different procedure:

Algorithm B
I. Approximate configuration using hypercubes
II. Add surrounding hypercubes to form a cube
III. Set boundary conditions on hypercube faces
IV. Solve elliptic equations globally over computational space

The fundamental idea behind Allwright's method is that it is possible to construct a suitable computational space for a configuration using only "simple" geometric information. Then, given a computational space, the positions of zonal boundaries are to be determined by solving the elliptic equations globally. Thus, with topological and geometric operations separated, a large part of the elliptic grid generation process can be automated.

The theory and algorithms for automated elliptic mesh generation presented in this paper are based on a five step procedure:

Algorithm C
I. Approximate configuration with polyhedra
II. Decompose surrounding space into convex polyhedra
III. Construct computational space
IV. Set boundary conditions on block boundaries
V. Solve elliptic equations globally over computational space

Thus, while B uses a piecewise constant approximation to the configuration, C uses a piecewise linear approximation. And, while B adds only exterior blocks, C considers both the addition of interior and exterior blocks as defined by a general decomposition problem. In addition, these blocks may be convex polyhedra instead of cubes. Finally, the relation between a blocking and computational space is addressed in C.

In summary, SMG is a non-linear problem and therefore one approach for constructing solutions is via appropriate linearization schemes. This leads directly into computational geometry and to the relationship between structured and unstructured mesh generation.

The remainder of this paper is organized around providing a clearer description of steps II-III in C and discussing algorithms for their solution in two dimensions. A central theme of this paper is that some problems are simply "hard" to solve. This idea is developed in the first section on complexity and algorithms. Section two is concerned with the principal "hard" problem of zone generation and its reformulation as a polygon decomposition problem. Algorithms for polygon decomposition are then outlined. The encoding of a blocking topology from a set of intersecting curves is presented in the third section. This establishes an identification between physical space and an incomplete computational space. To complete the construction of computational space, block coordinates must be resolved. This is discussed in the section on boundary point numbering.

Complexity and Algorithms

The complexity of an algorithm can be measured by counting the number of arithmetic operations it takes to process its input. Of particular interest is the asymptotic value of the operations count as the input size becomes large. The complexity of several important algorithms is listed in the table below.

Algorithm	Input Size	Operations Count
Polygon Triangulation	N vertices	O(N)
Sorting	N keys	O(N logN)
Hidden Surface Removal	N polygons	$O(N^2)$
Gaussian elimination	N unknowns	$O(N^3)$

These algorithms have the property that they solve a certain problem in polynomial time. That is, the operations count is a polynomial function of the input size. Such problems are said to be in the class P. There are many problems for which polynomial time algorithms are not known but for which a solution can be verified in polynomial time. A classic example is the traveling salesman problem: given a set of cities, and distances between all pairs, find a tour of all the cities of distance less than some value. Problems of this kind are said to be in the class NP. In summary then:

P: The class of problems which can be solved in polynomial time.
NP: The class of problems for which a solution can be verified in polynomial time.

Clearly all problems in P are also in NP but it is not known if P = NP. This is one of the outstanding problems in computer science. Among the class of NP problems, many of practical importance have a special property: if they could be solved in polynomial time, then *all* problems in NP could be solved in polynomial time. These problems are said to be NP-complete. To date, no one has been able to find efficient algorithms for any NP-complete problem and therefore it is generally believed that $P \neq NP$. This is usually interpreted as saying that NP-complete problems are harder than those in P. What about problems that are not NP-complete or are not in NP? Any problem which can be transformed to an NP-complete problem will have the property that it cannot be solved in polynomial time unless P = NP. Such problems are said to be NP-hard because they are at least as hard as the NP-complete problems. A more thorough discussion of these issues can be found in [9].

Many important problems in combinatorial optimization and computational geometry are either NP-complete or NP-hard. The only known algorithm guaranteed to produce a solution is exhaustive search, which has exponential complexity. As will be shown, several problems associated with blocking and the construction of computational space are NP-complete. This observation will be important in guiding a search for algorithms which are suitable for such intrinsically difficult problems.

Polygon Decomposition

The problem of constructing a zonal decomposition of a 2-D region bounded by smooth curves has been investigated previously [1], [7], [19], [20]. From these investigations there has emerged a weak understanding that the zonal decomposition problem consists of several sub-problems:

- When should two curves be connected by a zonal curve?
- How many connections should there be between connected curves?
- Where should connections intersect the curves they connect?
- What shapes should the connections have?

As noted in the introduction, this is a non-linear problem and local solutions may be obtained via a linearization process.

> Linearized Zonal Decomposition Problem: Given a polygonal region (a polygon with polygonal holes), decompose it into an optimal number of "quad-like" polygons.

Approximation by Polygons

The problem of polygonal decomposition of polygonal regions is central to both structured and unstructured mesh generation. When the polygon is a triangle, a comprehensive theory can be developed and this leads to the well known triangulation algorithms used in unstructured mesh generation [8], [17]. The general problem of n-gon decomposition is much more difficult and in several cases is known to be NP-complete or NP-hard [3], [10]. However, it is precisely the problem of 4-gon decomposition that is important to elliptic grid generation. It is interesting and significant that it is possible to isolate the essential difficulties of 4-gon decomposition. As discussed in [16], the quadrilateralization problem becomes difficult when the region is multiply connected. The following table summarizes these observations on approximation by polygons:

Problem	Classification	Algorithm
3-gon decomposition of polygonal regions with holes.	P	Constrained Delaunay triangulation [6].
Convex 4-gon decomposition of polygonal regions without holes.	P	Lubiw's dynamic programming algorithm [11].
Convex 4-gon decomposition of polygonal regions with holes.	NP-complete	?

The intrinsic difficulty of 4-gon decomposition is associated with deciding how to reduce the connectivity of a region. We interpret this to mean that more than geometric information is needed to decide whether a blocking of a multiply connected region is acceptable. This point of view is supported by examining the role of physics in blocking. As an example, consider the multiply connected region defined by the exterior of a 4-element airfoil geometry. For fluid dynamicists, it is natural to add wake-cuts to each airfoil and thereby reduce the connectivity of the region. However, this region might also represent a plate with 4 holes and the problem under investigation could be to simulate crack propogation. A structural dynamicist might therefore choose to add cuts between the holes. In either case, block boundaries are positioned so as to control grid density and therefore the quality of the physics calculation. More generally, block boundaries represent "initial guesses" for an adaptive meshing procedure. From this viewpoint, connectivity reduction becomes an intrinsically difficult problem for both structured and unstructured mesh generation.

The decomposition of polygonal regions into convex quadrilaterals is not always possible. For example, a pentagon cannot be quadrilateralized unless interior vertices are allowed. These interior vertices are known as Steiner points and are also necessary to achieve optimal decompositions [3].In practice, it may be convenient to add not only interior vertices, but interior line segments as well. The general decomposition problem relevant to elliptic mesh generation is therefore as follows:

> Convex Decomposition Problem: Construct a convex decomposition of the interior of a simple polygon containing obstacles of the form: simple polygon, line segment, point.

Of course, the main interest is in having a convex 4-gon decomposition. However, for the purpose of generating a blocking, n-gons with 4 distinguished corners are acceptable. A novel algorithm for constructing such a decomposition will be briefly discussed. More complete details are given in [4].

4-gon Decomposition Algorithm

The algorithm used to solve the convex decomposition problem has the following simple structure:

(A) Construct a constrained Delaunay triangulation of the region.
(B) Remove edges in the triangulation to generate convex 4-gons.

Efficient algorithms for (A) are well known [6] and will not be discussed here. Step B is a problem in combinatorial optimization. Moreover, it must be NP-complete or NP-hard because step A is in the class P but (A)-(B) solves the intrinsically difficult problem of 4-gon decomposition. Recently, a new class of algorithms (genetic algorithms) have been successfully applied to similar difficult combinatorial optimization problems [14]. These genetic algorithms are based on the principles of reproduction, crossover, and mutation as applied to a pool of near-optimal solutions. While the theory of genetic algorithms is in its infancy, they are proving to be of practical value on a growing number of problems. The development of such an algorithm for the solution of B is described in a separate paper [4].

In practice, the application of this algorithm results in a decomposition consisting of n-gons where n=4,5 and further optimization may produce polygons with even more edges. A data-structure for encoding the topology of such a mixed decomposition is discussed next.

Graphs and 2-D Blocking

Graphs and graph theory can be used to understand the topology of a blocking. In the simplest description, a blocking consists of a set of curves C (block boundaries) which intersect only at their endpoints P. The topology of a blocking can be described by a directed graph G = (V,E) through the correspondence {vertices V <-> points P}, {edges E <-> curves C}, and {edge directions <-> curve orientations}. This correspondence is realized by means of an embedding of the digraph into the surface.

$$(V,E) \rightarrow \mathbb{R}^2$$
$$V \mapsto P$$
$$E \mapsto C$$

Although a blocking determines a unique digraph and digraph embedding, recovering a blocking from a digraph requires additional structure.This structure is known as a rotation system [12]. A rotation system for a graph is defined via a rotation system for each vertex. A rotation system at a vertex is an ordering of edges incident to that vertex. This definition must be generalized for digraphs because edges may be incident to or incident from a vertex.

Definition: A rotation system for a digraph (V,E) is a subset of signed edges E_v and a rotation operator R_v which maps an edge in E_v to its successor. The set E_v contains e or -e depending on whether e is incident to or incident from v respectively.

With this definition, each vertex is assigned a unique subset of signed edges E_v, each edge in E_v is incident *to* v, and the union of E_v as v ranges over V is $\pm E$. Now, starting with an edge e_i, the rotation system can be used to construct a successor edge $e_{i+1} = -R\, e_i$. Continuing in this manner, a directed path through the digraph vertex set can be built. This leads to the following definition:

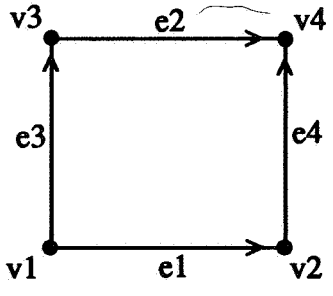
Definition: An R-path is a sequence of signed edges $(e_1, e_2, \ldots e_n)$ such that $e_{i+1} = -R\, e_i$ for some rotation operator R. Edges e_1 and e_2 are said to be R-path connected if they are members of the same R-path.

When an R-path is closed, it is called an R-cycle. More generally, the following definition holds:

Definition: An R-cycle is an R-path in which every edge has a successor.

An example helps to clarify the definitions given above. Consider a blocking which consists of a single square with curves oriented as shown below. A diagraph and rotation system for this blocking are

e1 = (v1,v2) E_{v1} = (-e1,-e3)
e2 = (v3,v4) E_{v2} = (e1,-e4)
e3 = (v1,v3) E_{v3} = (e3,-e2)
e4 = (v2,v4) E_{v4} = (e2,e4)



Note that because there are only two signed edges in E_v, the rotation operator R_v is trivially defined.

Starting at edge e3, the R-cycle (e3, e2, -e4, -e1) is computed as

$$
\begin{aligned}
&e3 \\
&e2 = -R_{v3}\, e3 \\
&-e4 = -R_{v4}\, e2 \\
&-e1 = -R_{v2}\, -e4
\end{aligned}
$$

and starting at e4, the R-cycle (e4, -e2, -e3, e1) is computed as

$$
\begin{aligned}
&e4 \\
&-e2 = -R_{v4}\, e4 \\
&-e3 = -R_{v3}\, -e2 \\
&e1 = -R_{v1}\, -e3
\end{aligned}
$$

It is easy to verify that there are no other R-cycles induced by this rotation system. In this case, all of the R-cycles have been found by computing the maximal R-paths. In fact, this can be proved in general.

Theorem: There is a 1-1 correspondence between maximal R-path components and R-cycles.

Proof: It is first shown that every maximal R-path component E is an R-cycle. If not, then there is some edge e in E that does not have a successor so that e' = -R e is not in E. But (e,e') is an R-path containing e which implies that E is not maximal. Similarly, if C is an R-cycle that is not a maximal R-path, then some edge e in C must have a successor e' = -R e that is not in C. But, C is R-path connected so that e' must be in C.

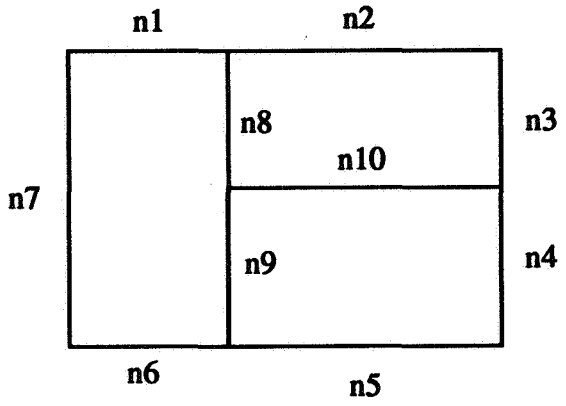
The maximal R-path components provide a partitioning of ±E into disjoint R-cycles. Intuitively, these R-cycles correspond to block boundaries. In the example above, the two R-cycles correspond to the two blocks representing the inside of the square and the outside of the square. In general, the R-cycles associated with an arbitrary rotation system do not correspond to block boundaries.

In summary, a digraph with certain vertex-edge operators can be used to encode the topology of a blocking. In practice, this may be used to reconstruct a blocking from a set of intersecting curves. This concept can be extended to higher dimensions where it can be used to encode the topology of a 3-D blocking [15].

Boundary Point Numbering

The density of grid points in a computational mesh is strongly influenced by the density of points on the corresponding boundary curves. In practice, the density of points is controlled by specifying the curve parametrization and the total number of points on the curve. While the selection of a parametrization can be quite arbitrary, the numbering problem is generally constrained. For example, multi-block point continuous meshes must have a consistent numbering across block boundaries. A blocking together with a such a numbering constitutes a description of computational space. In this section it is shown that the boundary point numbering problem is equivalent to computing integer solutions to a linear programming problem.

Theboundary point numbering problem is easier to discuss with a simple example in mind. Thus, consider the blocking shown in the figure below.



The problem under consideration is to number each curve so that opposing block faces have an equal number of points. Let (n1, n2, ..., n10) be the unknown values for the curve numberings. Then, the following system of 6 linear equations in 10 unknowns must hold:

$$\begin{aligned} n1 - n6 &= 0 \\ n2 - n10 &= 0 \\ n10 - n5 &= 0 \\ n7 - (n8+n9) &= 0 \\ n8 - n3 &= 0 \\ n9 - n4 &= 0 \end{aligned}$$

In general, for a blocking with E edges and B blocks, this will lead to a 2B x E linear system:

$$\mathbf{Ln = 0.} \qquad (1)$$

The linear system (1) is underdetermined and in this case it is easy to discover the extra degrees of freedom (n1, n2, n3, n4 for example). In practice, it may be desirable to fix certain values and have inequality constraints on others. For example:

$$\begin{aligned} n1 &= 10 \\ n2 &= 10 \\ n3 &= 10 \\ n7 &\geq 20. \end{aligned}$$

As a final constraint, the solutions to (1) should all be positive integers and it is desired to find the "smallest" such solution. After a simple transformation of variables, the edge numbering problem can be posed as follows:

Given integer valued **L** and **m**, find integer solutions to the problem:

$$\text{Minimize } \| \mathbf{n} \|_1$$

subject to:

$$\mathbf{L}\,\mathbf{n} = \mathbf{m} \qquad (2)$$

$$\mathbf{n} \geq \mathbf{0}$$

This problem is well known as an integer linear programming (ILP) problem [13]. Integer programming is another NP-Complete problem [9] and so it appears that the edge numbering problem is intrinsically difficult.

The simplex algorithm [2] is a well known iterative technique for solving linear programming (LP) problems. There is no guarantee that application of the simplex algorithm to (2) will produce an optimal integer solution. In other words, optimal solutions to LP problems are not necessarily integer. As discussed in [13], a necessary and sufficient condition for an LP with integer data to have an optimal solution that is integer is that the matrix **L** be totally unimodular. This means that every square, nonsingular submatrix of **L** has an integer inverse. It is interesting that the matrix **L** in (2) has this property and therefore that the simplex algorithm can be used to solve (2). The proof makes use of the following theorem.

Theorem: An integer matrix **L** with $L_{ij} = 0, 1, -1$ for all i and j, is totally unimodular if

1. No more than two nonzero entries appear in each column.
2. The rows can be partitioned into two subsets Q_1 and Q_2 such that
   (a) If a column contains two nonzero entries with the same sign, one entry is in each of the subsets.
   (b) If a column contains two nonzero elements of opposite sign, both elements are in the same subset.

Condition (1) is met because each curve is counted at most two times in the blocking equations. For curves i counted exactly twice (zonal curves), it is usually possible to arrange that n_i appear in the blocking equations with both plus and minus signs. In that case, set Q_1 = {index set of the rows} and Q_2 = {empty set}.

Summary

A theory of structured mesh generation has been developed within the framework of computational geometry. In this theory, the blocking step of structured mesh generation is identified with the problem of hexahedral decomposition. In the plane, this reduces to quadrilateralizing a polygonal region. The fact that this problem is NP-complete accounts for the empirically observed result that blocking is hard. Knowing that a problem is NP-complete is useful in choosing appropriate algorithms for its solution. Novel iterative algorithms based on genetic search have been developed to solve the blocking problem. The relationship between a block decomposition and computational space has been investigated using methods of topological graph theory. This has resulted in the definition of a blocking as a digraph with rotation system. Computational space is then a blocking together with block coordinates. The problem of computing block coordinates is shown to be equivalent to an integer linear programming problem. The special structure of the boundary point numbering equations allows a standard solution to this problem using the simplex method.

References

[1] Allwright, S.E., *Techniques in Multiblock Domain Decomposition and Surface Grid Generation*, in Numerical Grid Generation in Computational Fluid Dynamics, 1988.

[2] Brickman, L., *Mathematical Introduction to Linear Programming and Game Theory*, Springer Verlag, 1988.

[3] Chazelle, B. and Dobkin, D.P., *Optimal Convex Decompositions*, in Computational Geometry, G. Toussaint Ed., Elsevier Science, 1985.

[4] Cordova, J.Q., *Computational Geometric Aspects of Automated Mesh Generation*, Int. J. Comp. Geometry & Applications (to appear), 1992.

[5] Cordova, J.Q., *VisualGrid: A Software Package for Interactive Grid Generation*, AIAA 90-1607, 21st Fluid Dynamics Conference, Seattle, 1990.

[6] Chew, P.L., *Constrained Delaunay Triangulations*, Proc. 3rd Symposium on Computational Geometry, ACM, 1987.

[7] Dannenhoffer, J.F., *A Block-Structuring Technique for General Geometries*, AIAA 91-0145, 29th Aerospace Sciences Meeting, Reno, 1991.

[8] Edelsbrunner, H., *Algorithms in Combinatorial Geometry*, Springer Verlag, 1987.

[9] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

[10] Keil, M.J. and Sack, J.R., *Minimum Decomposition of Polygonal Objects*, in Computational Geometry, G. Toussaint Ed., Elsevier Science, 1985.

[11] Lubiw, A. , *Decomposing Polygonal Regions Into Convex Quadrilaterals*, Proc. 1st Symposium on Computational Geometry, ACM, 1985.

[12] Gross, J.L. and Tucker, J.L., *Topological Graph Theory*, Wiley Interscience, 1987.

[13] Garfinkel, R.S. and Nemhauser, G.L., *Integer Programming*, Wiley Interscience, 1972.

[14] Goldberg, D.E., *Genetic Algorithms*, Addison Wesley, 1989.

[15] Noble, S.S. and Cordova, J.Q., *Blocking Algorithms for Structured Mesh Generation*, AIAA 92-0659, 30th Aerospace Sciences Meeting, Reno, 1992.

[16] O'Rourke, J., *Art Gallery Theorems and Algorithms*, Oxford Univ. Press, 1987.

[17] Preparata, F.P. and Shamos, M.I., *Computational Geometry*, Springer Verlag, 1985.

[18] Sedgewick, R., *Algorithms*, Addison Wesley, 1983.

[19] Steinbrenner, J.P., Chawner, J.R., and Fouts, C.L., *The GRIDGEN 3-D Multiple Block Grid Generation System*, WRDC-TR-90-3022, July, 1990.

[20] Vogel, A.A., *A Knowledge-Based Approach to Automated Flow-Field Zoning for Computional Fluid Dynamics*, NASA TM 101072, April, 1989.

EAGLEView: A SURFACE AND GRID GENERATION PROGRAM AND ITS DATA MANAGMENT

M. G. Remotigue, E.T. Hart, and M. L. Stokes
Mississippi State University Engineering Research Center
Mississippi State, MS

ABSTRACT

An old and proven grid generation code, the EAGLE grid generation package by Joe Thompson, is given an added dimension of a graphical interface and a real-time database manager. The Numerical Aerodynamic Simulation (NAS) Panel Library is used for the graphical user interface. Through the panels, EAGLEView constructs the EAGLE script command and sends it to EAGLE to be processed. After the object is created, the script is saved in a mini-buffer which can be edited and/or saved and reinterpreted.

The graphical objects are set-up in a linked-list and can be selected or queried by pointing and clicking the mouse. The added graphical enhancement to the EAGLE system emphasizes the unique capability to construct field points around complex geometry and visualize the construction every step of the way.

INTRODUCTION

EAGLEView[1] is interactive surface and grid generation software developed to reduce the amount of time spent on the surface definition and refinement process so integral to the solution of the computational field simulation problems.

EAGLEView is a tool for the construction of two- and three- dimensional structured and unstructured surface geometries, and block-structured and unstructured volume meshes. EAGLEView is based on the EAGLE grid generation system developed by Joe Thompson.[2-4] EAGLE system is comprised of two programs: one defines the boundary surfaces[3] and the other generates and smooths the points within the field.[4]

EAGLEView combines the EAGLE surface and grid generation codes under one graphical program. The user can define his geometry, compute the volume grid, visualize the results, and make changes if necessary without having to execute a different program.

The NAS Panel Library[5] is used as the user interface to EAGLEView. The EAGLE commands most often used are available through data-entry panels, which are accessed by pull-down menus. The user enters the appropriate information into the panels; the EAGLE command is generated from this information and then submitted to the EAGLE batch code. The script is continuously

displayed in a mini-buffer located in the *Command Panel.* The user may save the contents of the buffer in order to restart the session at a later date.

Although familiarity with EAGLE is helpful, the engineer does not have to know the exact syntax of each command; EAGLEView generates the script. Associated with each panel is a help utility which explains the fields and prompts the user if necessary. Inquiries about both EAGLEView operation and EAGLE commands are available on-line through this option. Because of the features, both those engineers who are new to and those who are experienced with EAGLE will be able to use this software productively in just a few hours.

The user can define geometry either by reading in IGES-formatted files, or by using EAGLE-View's CAD-like commands, which include B-Spline curve and surface generation. Geometries are constructed in EAGLEView in an object-oriented manner: points are used to create curves, which are used to create surfaces, which are used to create grids. Embedded in EAGLEView is a point-and-click interface in which every point, curve, surface, or grid may be accessed and/or queried using the mouse.

After the algebraic surface is created, it may be refined using elliptic methods. Three dimensional elliptic smoothing is a planned addition. Three-dimensional elliptic smoothing is currently available in the batch EAGLE code; however, it is a planned addition to EAGLEView. Also grid quality measures are to be implemented so the user can check the meshes throughout the creation process.

EAGLEView gives the user a variety of different ways he may view the objects that are created. He may represent his surfaces as wireframe or as flat- or Gouraud-shaded. The grid surface can be viewed while each of the interior planes is highlighted manually or automatically by using the animate buttons.

In the next sections, the data structures and the functionality of the panels of EAGLEView will be discussed, and the construction of curves, surfaces, and grids will be reviewed.

DATA MANAGMENT

To build upon the foundation of the EAGLE grid code, a graphical interface developed from the NAS Panel Library was overlaid on top of the fortran source code. These panels are used to collect information that EAGLE needs to create the object. As will be discussed later, each entry in the panel needs not to be entered. If any pertinent information is missing, an error message indicates the field to be entered. Otherwise, the EAGLE command is constructed and sent to EAGLE to be parsed and executed. Upon completion, the object is displayed in the viewing window and the script is saved in the **Journal Script** buffer in the *Command Panel.*

As the graphical objects are created, an entry is added in a linked-list. Objects such as points, lines, surfaces, grids, vectors, and axes are pointed to by the graphical object list. The list is set-up using a C structure that points to the next object in the list. The final object will have a terminating null as the pointer. A schematic can be seen in Figure 1.

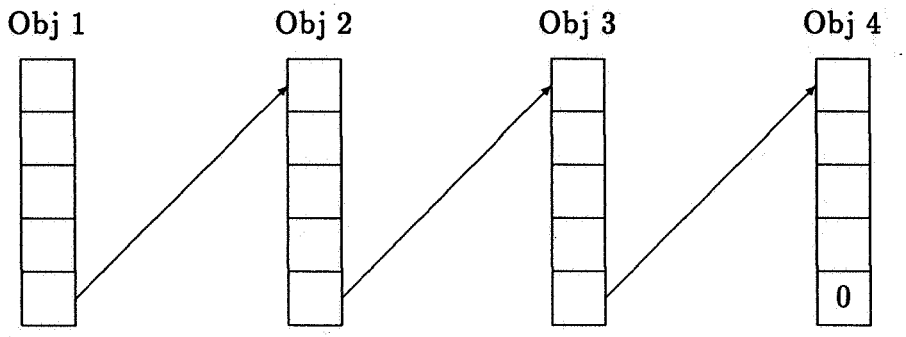


Figure 1: Graphical data structure.

Each object as it is created is stored and allocated memory in its own data structure that is pointed to by the graphical object list. This allows the flexibility to add and delete objects from the display. The objects can also be toggled visible or not. If the object is desired to be visible, a drawing function is called. Only the grids are not redrawn, the graphical representation is in the form of a display list. This only allows one grid in core at any one time. However, large surfaces do have a tendency to slow the graphical responsiveness.

THE PANELS OF EAGLEView

The graphical interface of EAGLEView is developed from the NAS Panel Library. These panels act as data collectors for the various options of EAGLE or as the control of the visual display. The user enters the information, toggles the appropriate buttons, or manipulates sliders, discs, or menus. The operation of main display panels, shown in Figure 2, and the panels that construct the primitive entities are discussed in detail in the following section.

Command Panel

The Command Panel contains the functions necessary for EAGLE script file execution and manipulation.

The EAGLEView mini-buffer may be edited by pressing the **Edit** menu button. A window will appear at the mouse cursor containing the editor specified by the user in the EAGLEView Execution script. Upon termination of the edit session, control will be returned to EAGLEView and the edited EAGLE script will appear in the mini-buffer.

EAGLE script files may be read from disk using the **Read** option. When users click on this button, the *Read Panel* appears. A list of files in the current directory appears in the mini-buffer. The user may click on one of these files, or may type the file name in the typein above it. When the correct file name has been entered, the user should click on the **Accept** key, which causes the contents of the file to appear in the mini buffer in the *Command Panel.*

These commands may be executed in part or full by pressing the **Execute** menu button and

dragging the cursor over the desired option. To highlight a portion of script in the mini-buffer to be executed, the user should position the cursor over the beginning of the command he wants to start with and then drag the cursor past the last command to be executed.

The **Save** menu option works identically to the **Read** menu option. When the *Read Panel* appears, the user should click on the appropriate file name in the mini-buffer, or type the file name into the typein. He should then click the **Accept** button, at which time the script will be written to the file specified.

The **Clear** menu option allows the user to begin a new EAGLEView session without terminating the program. When this button is selected, the EAGLEView database will be purged, the EAGLE script mini-buffer will be cleared, and previously created objects will disappear from the viewing window.

Clicking on the **Help** button causes the EAGLEView *Help Panel* to be invoked. The user can access both EAGLEView and EAGLE commands by section and/or by phrase.

Viewing Window

The *Viewing Window* is the central EAGLEView window. In it all graphics are displayed. The user can easily manipulate and/or pick all objects in the window using the mouse. To rotate the objects, the user should hold down the left mouse button and sweep the cursor in the direction of rotation. The scene will rotate continuously while the left mouse button is down at the speed with which the cursor is moved. Zooming is accomplished by holding the middle mouse button down and pushing up to zoom out or down to zoom in. To translate the objects, the user should hold the right mouse button down and drag the cursor to the desired position. The objects in the scene will follow the movement of the cursor.

Manipulation Panel

The objects in the viewing window may also be rotated and/or translated by using the sliders and dials in the *Manipulation Panel.* By manipulating the dials, the user can rotate the objects about a specific axis. When the mouse button is released, the dial then returns back to the zero position, and the objects retain their new position. Using the sliders enables the user to translate the objects along the specified axis. The objects will move until the mouse button is released. As with the rotations, upon release of the mouse button, the slider value returns to the zero position.

Display Panel

The buttons and menus in the *Display Panel* allow the user to inquire about and modify the characteristics of objects in the viewing window.

The menus under the **Select** heading contain three entries: On, Off, and Toggle. The user can individually turn on, turn off, or toggle points, vectors, axes, curves, surfaces, and grids.

Complex geometries slow manipulation of the *Viewing Window* considerably. EAGLEView's answer to this problem is the **Speed Draw** button. Setting Speed Draw = ON changes which objects are rendered in the *Viewing Window* during rotation, translation, and zoom. Lines are reduced to points, surfaces are reduced to lines, and just grid boundaries are shown while the mouse button is depressed. The objects are rendered in full, however, when the user releases the mouse button.

The **Select Curve** and **Select Surface** buttons enable the user to inquire about a visible curve or surface. When the user "picks" a curve or surface in the viewing window with the right mouse button, the panel used to create that entity will appear on the screen showing the appropriate information.

Construction Panel

The user builds or creates the display objects in the *Construction Panel.* In these panels, points, lines, surfaces, grids, vectors, and axes can be created and manipulated. A description of the most often used panels and the entries will be provided. Descriptions of the other panels can be found in the online help file. The **Script** button should be depressed after all needed information has been provided to the panels. Then, the objects are displayed on the screen, unless otherwise indicated.

Point Panel

A point will be used to attach vectors and as end points to curves. The user can specify "construction" points in the field. This can be done by entering the triad in the **X, Y, Z** typein space provided or by selecting a point on a displayed line or surface. Selection of a point on a line or a surface is done with the right mouse button. New points can be added by selecting the up arrow near the **Point** number display. Previous points can be queried by clicking the down arrow. Even though the points are displayed on the screen, the **Script** button should be depressed so that the entry can be saved in the script buffer.

Vector Panel

The vector is used in the construction of splined curves, surfaces created by a rotated curve, grids created by a rotated surface, and in translations. A vector can be created numerous ways. The user can use the rotation dials or the typeins above the dials for a particular angle around a specified axis, or specify the components of the vector, or use existing points on the screen to calculate a normal from three points or a tangent from two. The vector does not become visible until it is attached to a point. The **Attach Pt** is unecessary, but the vectors are unpickable if they are not on the screen. The **Normalize** and **Reverse** buttons are self explanatory. To create another vector, click the up arrow. To querie a previous vector, click the down arrow.

Axes Panel

A subordinate axis is used in the *Translate/Rotate Panel* under *Utilities.* The axis is an entity of EAGLEView, not of EAGLE. When the axis is used, the translations and rotations of the axis are used to construct the origin and Eulerian rotations applied to lines, surfaces, and grids. A scaling factor can also be applied to the axis, or scaling can be done in the *Translate/Rotate Panel.*

Line/Scurve Panel

This is found as a sub-menu of Curves in the *Construction Panel.* Lines or spline curves are used to define the edges of surfaces and grids or used as axis curves. A straight line can be created by either entering the triads of the end points in **R1** or **R2** or by selecting the points displayed in the viewing window. For reference, **R1** is the first end point and **R2** is the last end point. The number of points on the curve is entered in the **Point** field. The points can also be selected from a previous curve by selecting the curve in the viewing window. The spacing at either end can also be specified by selecting **R1 Spacing** and/or **R2 Spacing**. Spacing can also be typed in or picked from a curve. Hyperbolic tangent spacing is the default used. If no spacing is specified, the line is equally spaced. A splined curve is created similarly, except that the slope vectors for the corresponding end points are selected or typed into the **T1** and **T2** fields.

The curve can also be splined onto a forming surface, by selecting a surface for the **Form** field. The selected surface is defined as a bi-cubic surface spline. The curve is then mapped onto the splined surface.

It should be noted that entering negative values for the points and spacing still conforms to the EAGLE logic of using SETVAL and SETNUM. This option is handy for future batch runs when the spacing or the number of points may only be changed.

Planar Conics Panel

This is another panel for specific conic shapes like a circle, an ellipse, a hyperbola, and a parabola. These conics are created in the XY plane. The user enters the appropriate data for each entity: radius for a circle, or minor and major axis for an ellipse, and so forth. The user can control the portion of the conic created by specifying the bounding angles. The conic will be created from **Angle1** to **Angle2**. Negative angles can be entered so that the conic will be created in the correct sense. The conic can always be reversed at a latter time in the *Switch Panel.*

Intersection Panel

This panel is actually found in the *Surfaces Menu*, but the result is a curve. This is a useful feature of EAGLE. This produces a curve at the intersection of two surfaces. The only criterion is that the second running index or the j index of the protruding or **male** surface runs into the **female** surface. This operation is iterative but reliable. The intersection curve will have the same number of points as the j index of the **male** surface. This implies that the **male** surface be completely bounded by the **female** surface at the intersection.

Blend Panel

This panel creats surfaces using algebraic techniques. If only the **L1** and **U1**, for lower i constant curve and upper i constant curve, and the number of interior **Curves** are specified, then a ruled surface is indicated. If all edges are specified, transfinite interpolation (TFI)[7] with arclength interpolants[8] is used to calculate the interior points. Consequently, **L2** and **U2** represent the lower j constant curve and the upper j constant curve respectively. A forming surface can be selected to spline the new surface onto. The ability to select between linear and arclength interpolants and a polar TFI will be available in a future version.

Stack Panel

The stack option creates a surface by progressing a curve along an **Axis line**. The **Axis line** could be any curve in space. Another bounding curve can be used to blend between. This option is very particular on the orientation of the curves and seems to work best if the bounding curves are in the XY plane and the **Axis Line** is out of the XY plane.

Rotate Panel

The *Rotate Panel* allows the user to select a curve and one of the primary axis vectors, which needs to be created in the *Vector Panel*, to rotate about to create a surface. The bounding angles can be specified in the **Start** and **End** fields.

Generic Grid Panel

This panel is used as a container for the grids that can no longer be associated with the forming surfaces. Grids become generic if they have been transformed, scaled, rotated, or extracted. Only actions that can be performed in this panel are the ability to toggle the grid visible or not or to delete the grid from memory.

Grid Blend/TFI Panel

Algebraic grids can be created from blending between two bounding surfaces or by specifying all six bounding surfaces of a volume. Either **Blend** or **TFI** is selected. The appropriate number of bounding surface fields will appear. The same nomenclature used in the *Surface Panel* is utilized here. Where **L1**, **U1**, **L2**, **U2**, **L3**, and **U3** represent the different bounding surfaces. The **Blend** option requires that the number of **Layers** or stacked surfaces be specified. Currently, the linear interpolants are used to calculate the interior points for the TFI option, arclength interpolants and a polar TFI option is underway.

Grid Rotate Panel

This panel is similar to the **Rotate Panel** to create a surface, except that a surface is selected.

The user still needs to create an axis vector in the *Vector Panel* and specify the number of **Points** or surfaces in the rotated direction. The same control on the bounding angles is also available.

Unstructured Panels

The *Unstructured Panels* consist of a *Read Panel*, a *Surface Panel*, and a *Grid Panel* (Not implemented at publication). The ability to construct unstructured surfaces and grids is under development by Nigel Weatherhill.[9,10] This capability will allow the construction of hybrid structured/unstructured grids using either point-to-point or overlapped block boundaries.

Utility Panels

The *Utility* menu of the *Construction Panel* is a container of the vast number of utility functions. These utilities include the ability to (a) redistribute points or spacing on curves or surfaces, (b) extract lines, surfaces, or grids from other lines, surfaces, or grids, (c) concatenate lines, surfaces, or grids, (d) translate, scale, or rotate lines, surfaces, or grids, (e) reverse and swap the direction of the indices on lines, surfaces, or grids, and (f) assemble points or lines into lines or surfaces.

THE FUTURE

The future of EAGLEView will take several independent paths. A modified version will be included as an integrated module in FAST.[6] This is a natural merging in that both FAST and EAGLEView use the NAS Panel Library, and both have a similarity designed visual database, allowing the use or the surfer module in FAST for the rendering in EAGLEView. EAGLEView will have an independent development path at the Engineering Research Center for Computational Field Simulation with the emphasis on integrating field simulation programs directly into the interface. This process would allow a user to set boundary conditions and initial field properties interactively, then submit the request to solve the problem to either the local workstation or a remote computer with full interactive control of the process. Initially, flow codes will be supported, but support for other field simulations will be added, providing for such capabilities as the simultaneous solution of both fluid as well as structural fields. The combined capabilities of EAGLEView and FAST will provide the researcher with a fully integrated system: from the design process to system simulation and visualization. Additionally, other research organizations have shown interest in expanding the capabilities of EAGLEView for their own purposes under the agreement that the changes be made available to the public without charge.

REFERENCES

1. Soni, B. K.; Thompson, J. F.; Stokes, M.; and Shih M.-S.: GENIE^{++}, EAGLEView, and TIGER: General and Special Purpose Graphically Interactive Grid Systems. AIAA-92-0071, 1992.

2. Thompson, J. F.; and Gatlin, B.: Program EAGLE User's Manual Volume I — Introduction and Grid Applications. AFATL-TR-88-117, September 1988.

3. Thompson, J. F.; and Gatlin, B.: Program EAGLE User's Manual Volume II — Surface Generation Code. AFATL-TR-88-117, September 1988.

4. Thompson, J. F.; and Gatlin, B.: Program EAGLE User's Manual Volume III — Grid Generation Code. AFATL-TR-88-117, September 1988.

5. Tristram, D. A.; Walatka, P. P.; Raible, E. L.; and Hultquis, J.: Panel Library Programmer's Manual: Version 9.7. Report RNR-90-006, December 1990.

6. Bancroft, G. V.; Merritt, F. J.; Plessel, T. C.; Kelaita, P. G.; McCabe, R. K.; and Globus, A.: FAST: A Multi-Processed Environment for Visualization of Computational Fluid Dynamics. *Proceeding of the IEEE Visualization '90 Conference.* October 1990.

7. Thompson, J. F.; Warsi, Z. U. A.; and Mastin, C. W.: *Numerical Grid Generation, Foundations, and Applications.* Elsevier Science Publishing Company, New York, 1985.

8. Soni, B. K.: Two- and Three-Dimensional Grid Generation for Internal Flow Applications of Computational Fluid Dynamics. AIAA-85-1526, 1985.

9. Weatherhill, N. P.; and Soni, B. K.: Grid Adaption-Refinement in Structured-unstructured Algorithms. *Proceedings if the Third International Conference of Numerical Grid Generation in CFD.* Barcelona, Spain, June 1991.

10. Weatherhill, N. P.: A Method for Generating Irregular Computational Grids in Multiply Connecting Planar Domains. *International Journal for Numerical Methods in Fluids.* Vol. 8, p. 181, 1988.

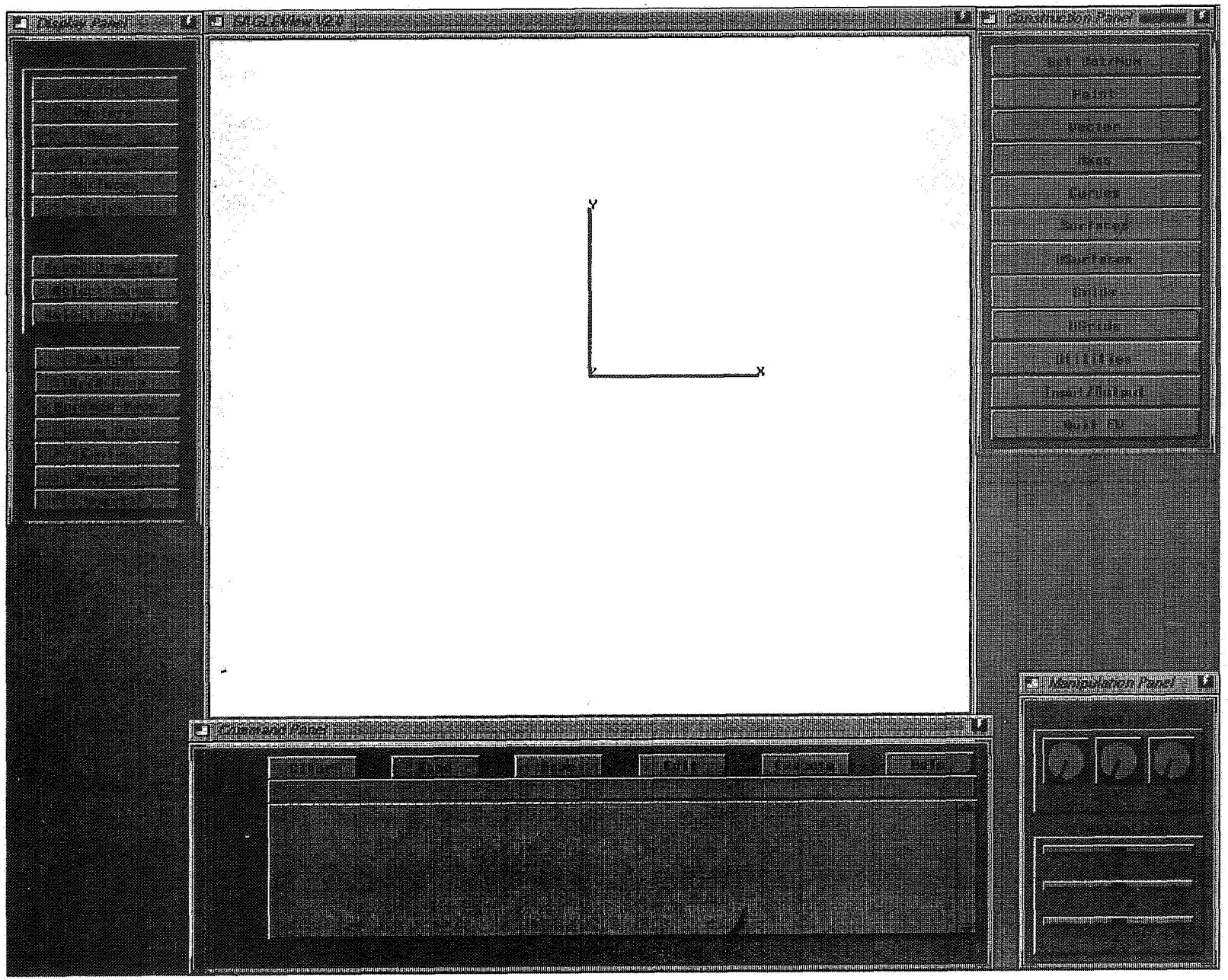


Figure 2: Main display panels of EAGLEView.

RECENT ENHANCEMENTS TO THE GRIDGEN STRUCTURED GRID GENERATION SYSTEM †

John P. Steinbrenner and John R. Chawner
MDA Engineering, Inc.
Arlington, TX

ABSTRACT

Significant enhancements are being implemented into the GRIDGEN 3D, multiple block, structured grid generation software. Automatic, point-to-point, interblock connectivity will be possible through the addition of the domain entity to GRIDBLOCK's block construction process. Also, the unification of GRIDGEN2D and GRIDBLOCK has begun with the addition of edge grid point distribution capability to GRIDBLOCK. The geometric accuracy of surface grids and the ease with which databases may be obtained is being improved by adding support for standard CAD file formats (e.g., PATRAN Neutral and IGES files). Finally, volume grid quality has been improved through addition of new SOR algorithm features and the new hybrid control function type to GRIDGEN3D.

LIST OF SYMBOLS

$\vec{r} = [\ x\ y\ z\ ]^T$ Cartesian coordinate vector
(ξ, η, ζ) computational coordinates
(i, j, k) computational indices
(Φ, Ψ, Ω) control functions

INTRODUCTION

Grid generation is perhaps the most manhour intensive task in the process of applying computational fluid dynamics (CFD) methods to complex configurations. This conclusion is supported in the findings of two separate CFD committees: [1] [2]. Not surprisingly, CFD research in the past few years has produced several suites of grid generation software, each aimed at reducing the so-called grid generation bottleneck. This software follows a marked trend towards development of interactive, graphical tools [3] [4] [5]. The trend is quite logical if one considers the speed at which workstation technology is advancing and the fact that grid generation is a highly visual-oriented, geometry-based technology.

†Work done on subcontract to Computer Sciences Corp. (CSC-ATD-ER-92-B-117) on contract to NASA Langley Research Center (NAS1-19038)

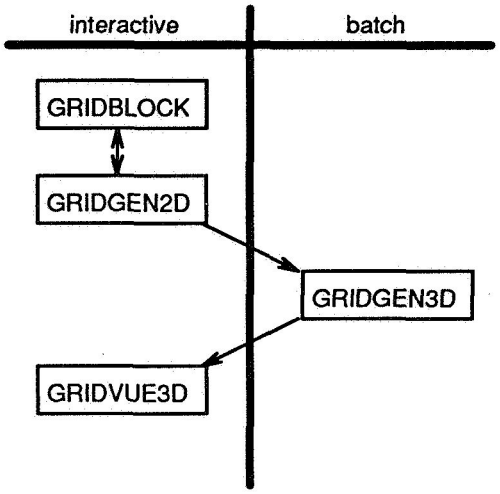


Figure 1: Schematic of the GRIDGEN process.

One particular example of a graphical, interactive software system developed for the generation of multiple block, structured grids is GRIDGEN [9]. The GRIDGEN system consists of four separate codes which accomplish distinct tasks of the grid generation process in the order indicated in Figure 1. The process begins with a user supplied database, which consists of any number of discrete $M \times N$ networks of points. The collection of networks is used by GRIDGEN only to define the 3D surface shape of the configuration. The first code, GRIDBLOCK (Figure 2), is used to decompose the domain around the database into a multiple block structure and to assign a computational coordinate system to each block. GRIDGEN2D (Figure 3), the second code, is then used to generate, using algebraic and elliptic partial differential equation (PDE) methods, grid points on the twelve edges and six faces of each block. The third code, GRIDGEN3D, uses algebraic and elliptic PDE methods to generate the grid points within each block. Finally, GRIDVUE3D is used for visual examination of the completed multiple block volume grid. GRIDBLOCK, GRIDGEN2D, and GRIDVUE3D are interactive graphics codes written specifically for the Silicon Graphics, Inc. IRIS workstations (they also run on IBM RS/6000 workstations with the GL Graphics software), while GRIDGEN3D is a batch code written for a Cray X/MP supercomputer.

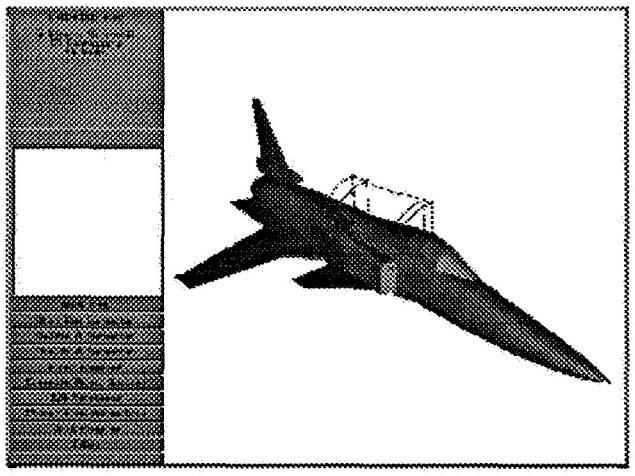


Figure 2: A typical GRIDBLOCK screen.

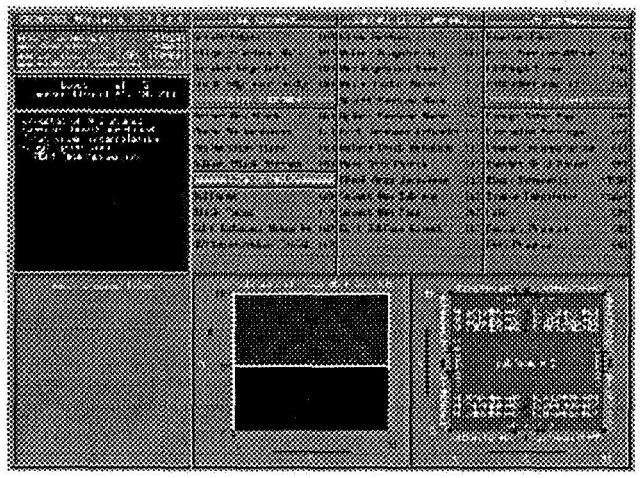


Figure 3: A typical GRIDGEN2D screen.

While the initial release of GRIDGEN (Version 6) has been very successful, there remains plenty of room for improvement. Therefore, several changes are being made to GRIDGEN that will eventually result in Version 8. These tasks may be organized in three categories: those designed to increase user efficiency, those designed to improve geometric accuracy, and those designed to improve grid quality. First, user efficiency has been improved through addition of edge grid point generation capability to GRIDBLOCK, creation of the domain entity in GRIDBLOCK, and addition of low-level automation to GRIDBLOCK and GRIDGEN2D. Second, surface grid geometric accuracy and the ease with which database files can be obtained is being improved through supporting the popular CAD file formats PATRAN Neutral File and IGES. Finally, grid quality has been improved through upgrades to the control function types and numerical algorithm used with the PDE solver in both GRIDGEN2D and GRIDGEN3D. The remaining sections of this paper describe the rationale behind and benefits of each of these tasks.

IMPROVED USER EFFICIENCY

One of the overall goals of the GRIDGEN software is to make the user a more efficient grid generator. This is attempted through the use of interactive computer graphics such that the user has immediate graphical feedback on each step of the generation process. One path toward improved efficiency is to consolidate GRIDBLOCK and GRIDGEN2D into a single code, thereby eliminating much user confusion. Part of this long term goal is accomplished within Version 8 by adding GRIDGEN2D's edge point distribution capability to GRIDBLOCK. Further improvements are made in the blocking process by adding the domain entity to GRIDBLOCK such that the software can determine all point-to-point interblock connections automatically. Finally, the users workload is reduced through the addition of tools to perform several grid generation functions automatically.

Edge Point Distribution in GRIDBLOCK

Once the database defining the shape of the configuration has been loaded into GRIDBLOCK and visually inspected, the user begins decomposing the domain by drawing the lines that will

eventually make up the block edges. These 3D lines are known as connectors. Connectors may be made up of any number of segments. Segments are the primitive line types available and include (see Figure 4): straight lines, elliptical arcs, splined curves, and lines constrained to the database. The connectors may be drawn as though the user were sketching them with pencil and paper; no regard need be given to the order or direction that they are drawn, nor what block they'll be in. The example in Figure 4 shows the connectors that define one block of a system.

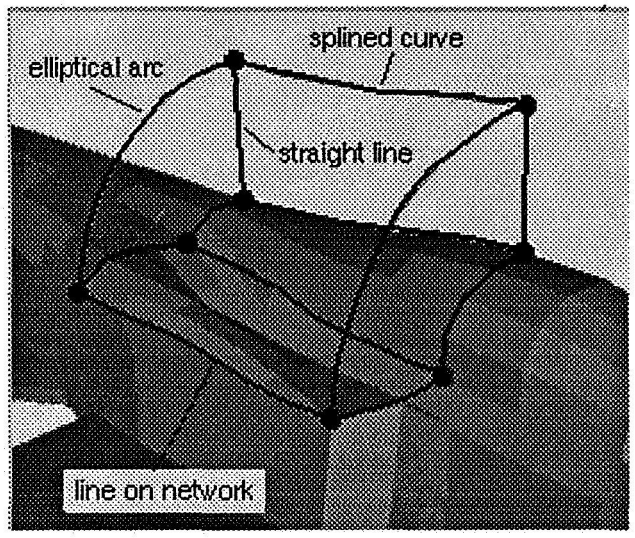


Figure 4: Segment types in GRIDBLOCK.

Block construction in GRIDBLOCK Version 6 proceeded by grouping a number of connectors into a block, and then by specifying orientation of the computational coordinate system within the block. Computational dimensions (i.e., number of grid points in each computational direction) would then be assigned, and finally interblock connections and flow boundary conditions (BC's) would be explicitly set by the user. Generation of actual grid points began within GRIDGEN2D, first by distribution of points in 1D along block edges and then in 2D on block faces.

The separation of the tasks of choosing the number of points in a block (in GRIDBLOCK) and distributing the grid points on edges (in GRIDGEN2D) could sometimes cause problems for the user. Only upon distributing points along edges in GRIDGEN2D did it become apparent whether the number of points chosen in GRIDBLOCK was appropriate. In cases where the user decided that too few or too many points had been specified he or she would exit GRIDGEN2D, re-run GRIDBLOCK, and change the block size. Unfortunately, changing the block size in GRIDBLOCK also required the user to manually re-set all interblock connections and BC's. This would affect most every other block in the system resulting in a lot of re-work.

The solution to this problem was to add the capability to distribute grid points on connectors to GRIDBLOCK Version 8. This allows the user to immediately determine whether or not sufficient points have been chosen for the distribution function, thereby eliminating the need for tedious rework. Part of this new capability consists of adding new segment types to GRIDBLOCK: a surface cubic segment, a piecewise cubic polynomial spline with Bessel interpolants constrained to the database; and a user defined segment read from a formatted ASCII file.

Connectors are assigned a computational dimension via type-in or by copying the dimension of another connector. Grid point distribution is then controlled interactively using tools that combine the functionality of the edge subdivision and grid point distribution menus of GRIDGEN2D Version 6. This combined interface provides a more intuitive and considerably streamlined way of distributing grid points on the connector. GRIDGEN2D's distribution functions (2-sided Vinokur, 1-sided hyperbolic sine and hyperbolic tangent, 1-sided geometric, and equal spacing) have been added to GRIDBLOCK. One new distribution function is based on Monotonic Rational Quadratic Splines (MRQS) [6], whereby a grid point may be placed at a specific location with the stipulation that the grid point distribution vary smoothly across it.

In Version 8, connectors may be redimensioned at any time with minimal effort, and all features of the previous grid point distribution will automatically be applied to the new grid points (Figure 5). Relative grid point distributions are also preserved after the shape of the connector is modified (Figure 6). This is a significant improvement over the current process in GRIDGEN2D Version 6, where the grid points must be redistributed manually if the edge shape is changed and where the edge shape must be redefined if the distribution of points is to be changed.

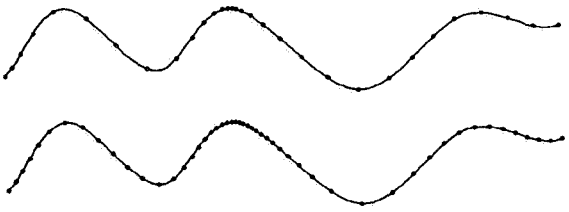


Figure 5: In GRIDBLOCK Version 8 the relative grid point clustering is maintained after the number of points is changed.

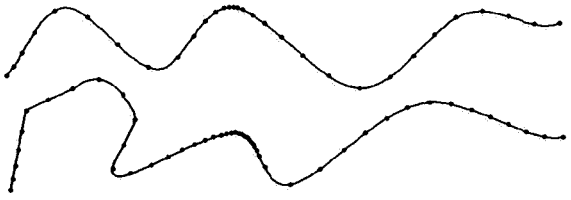


Figure 6: In GRIDBLOCK Version 8 the relative grid point clustering is maintained after the connector shape is modified.

An additional feature of GRIDBLOCK Version 8 is that database constrained segment types are maintained in parametric coordinates. This means that when the segment is edited in GRIDBLOCK the points will remain on the database, whereas in Version 6 they could be moved off the surface. This feature also implies that edge shapes will not have to be re-drawn in GRIDGEN2D in order to use the database parametric elliptic PDE solver.

Development of the Domain Entity

The second major addition to GRIDBLOCK Version 8 is the development of a new methodology for block construction. The impetus for this methodology comes largely from the fact that

GRIDGEN allows non-full face interblock connections and flow boundary conditions, a general boundary condition type supported by a number of flow solvers [7],[8]. Earlier it was mentioned that GRIDBLOCK Version 6 defined blocks by the connectors that formed the twelve physical edges of the block. No geometrical information pertaining to regions interior to a block's face (such as the horizontal connector between blocks B and C that also lies in the face of Block A in Figure 7) was associated with that block. Hence, it was impossible to determine inter-block connections automatically. The user was responsible for setting all connections manually by typing in block numbers, face numbers, and index ranges.[‡] This error-prone procedure was the most confusing step in GRIDBLOCK Version 6. Further, the fact that dimensions were assigned independently on the block level allowed inconsistent block dimensions to go undetected until the user attempted to set an impossible connection such as the one in Figure 8. Problems of this sort are being eliminated in GRIDBLOCK Version 8 with the introduction of a new entity, called a domain, which fits between connectors and blocks.

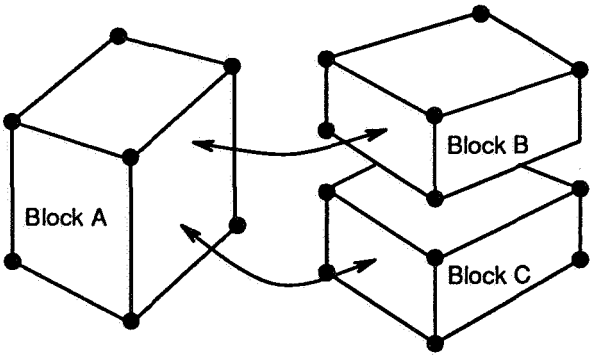


Figure 7: In GRIDBLOCK Version 6 blocks were represented by connectors and nodes only.

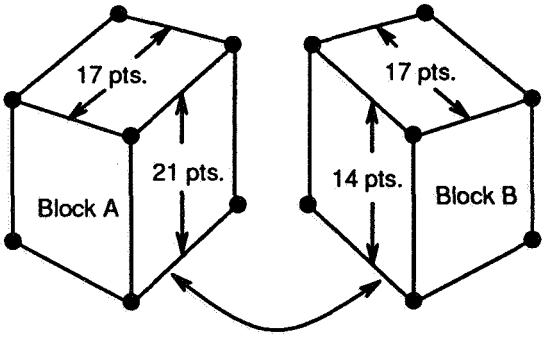


Figure 8: In GRIDBLOCK Version 6 inconsistencies in dimensioning could lead to impossible connections.

A domain is a 2D region on the face of a block. It is analogous to a subface in GRIDGEN2D terminology. A domain is created by defining a closed loop of connectors that outlines a region of a face of a block. This may either be a region of a particular BC, a surface on which two blocks connect, or a subset or superset of either. The sole requirement is that the domain represent a

[‡]Note that if GRIDGEN were limited to full face to face connections, the necessary connections could be determined automatically.

computationally rectangular region of the computational domain. A single surface represented by numerous domains is shown in Figure 9.

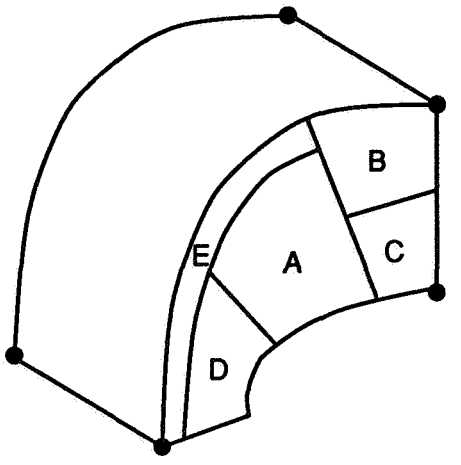


Figure 9: In GRIDBLOCK Version 8 a single block face may be represented by several domains.

Dimensional inconsistencies are now caught by GRIDBLOCK Version 8 during domain construction rather than in connection specification. This is possible since connectors have a number of grid points associated with them. When a domain is completed, it is compared to all other domains to insure uniqueness. A check is also made to see if two adjacent edges in the domain overlap with any other domain. If so, the user may have inadvertently defined the same physical surface by two differing domains, a mistake likely to cause problems for the remainder of the process (see Figure 10). Alternately, two domains with coincident adjacent edges may indeed represent different surfaces, as illustrated in Figure 11. The wing in this figure with a cusped trailing edge has domain B defined on the upper wing and domain A defined on the lower wing. The trailing edge connectors are coincident, as are the short connectors adjacent to the trailing edge at the wing root and tip. A blind description of the two domains might suggest that they lie on the same surface when in fact they do not. GRIDBLOCK will warn the user to exercise care under such circumstances. User generation of unique domains is expected to be a source of confusion for the novice user, and so additional tools are being added to GRIDBLOCK to ease confusion. For example, domain splitting and concatenation routines are included.

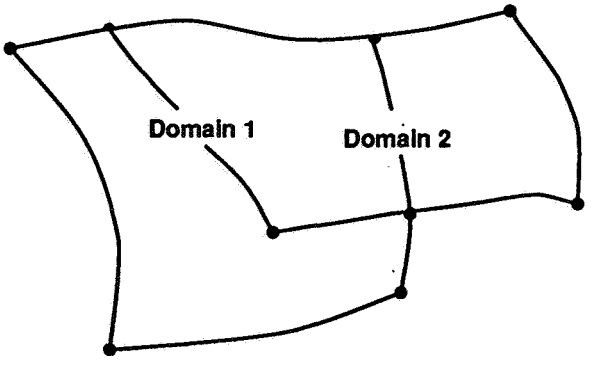


Figure 10: An example of overlapping domains.

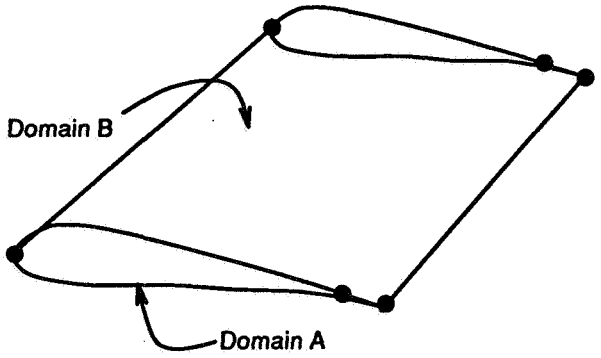


Figure 11: An example of non-overlapping domains with coincident edges.

In GRIDBLOCK Version 8, a block is constructed by interactively grouping domains into the six faces of the block. A face may consist of a single domain. However, the more interesting case is when the face is comprised of several domains (see Figure 9). Here, the user builds the face by systematically selecting the component domains that make up the face. The algorithm developed for face construction is order dependent, and as such allows faces to be constructed from an ambiguous set of domains in an unambiguous way. For example, the C-type face in Figure 12 representing a wing and wake region by four domains may be defined unambiguously by connecting the domains in the order A, B, C and A again. Without any order, it would be impossible to ascertain whether the domains connect at the leading edge, the wing trailing edge, the downstream connector, or at the inboard or outboard connectors on domain A.

The face construction algorithm developed for GRIDBLOCK will detect errors in face dimensioning as well as connections which result in a non-rectangular face. When the final face is assigned to a given block, dimensional and geometrical consistency is automatically checked. Block construction is then completed by assigning computational directions ξ, η, ζ at a corner of the block. This too is done graphically, in a manner similar to GRIDBLOCK Version 6. Since a block's computational orientation is the final step in block construction, later reorientation of a block is trivial. Hence, what formerly required a complete respecification of the block shape, dimensions, and interblock connections and BC's may now be achieved by a single, simple command.

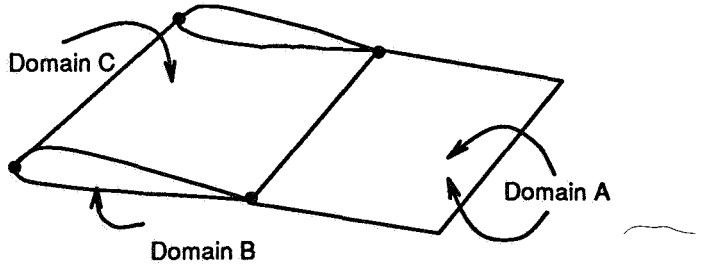


Figure 12: A face represented by the four domains A,B,C,A.

If the blocks have been defined without domain overlap, each domain will be assigned to a block exactly zero, one or two times.[§] If a domain is not assigned to any block, it is a superfluous domain

[§]A domain defined more than twice signifies an impossible blocking structure, usually detectable by the user.

and is not part of the developing blocking system. A domain assigned once in the block structure represents an external boundary of the composite block structure. Domains of this type normally correspond to block surface regions on which BC's would be set, such as solid surfaces, planes of symmetry, farfield conditions, etc. The user may set these BC's by picking the domain from the screen and the BC type from a flow solver-specific menu similar to GRIDBLOCK Version 6. Finally, domains assigned twice in the structure represent interblock connections. Since such domains exist in two places, the GRIDBLOCK Version 8 will automatically be able to determine all regions of point to point connections in the multiple block structure. This is a significant improvement over GRIDBLOCK Version 6, which as described above forced manual establishment of all connections. Hence, this new capability will eliminate both the time needed to set connections and the error associated with it. The instances where doubly defined domains in a blocking arrangement are not intended to define a flow-through condition (such as a flowfield obstruction with zero thickness) are easily handled by setting a BC on each occurrence of the domain, thereby circumventing the need for GRIDBLOCK to find a connection at that domain.

Automation Tools

Introduction of the domain entity and edge point distribution in GRIDBLOCK Version 8 also has a beneficial side effect that falls into the category of automation. Since domains (essentially subfaces) have been defined in terms of four edges, and since grid points have been distributed on the connectors that make up the edges it will be possible for GRIDGEN2D Version 8 to automatically initialize surface grid points within each domain using algebraic techniques. This is true even in cases where the connectors have been defined in terms of the database and the domain is intended to maintain the database shape since the parametric coordinates are now stored with the connector. Hence, the role of GRIDGEN2D Version 8 will be one of a surface grid refinement tool rather than a surface grid generation tool.

The changes to GRIDBLOCK Version 8 will also allow a user's change in the number of points on a connector to be propagated throughout the entire multiple block structure semi-automatically. In this utility, the user would select a connector, enter the new dimension, and all affected domains, faces and blocks would be updated semi-automatically to maintain dimensional consistency. New grid point distributions would be calculated automatically, and a new surface grid system reflecting the new blocking dimensions could be initialized in GRIDGEN2D. In this way, a low-level change to the blocking system would be enforced on the entire grid in a nearly automatic fashion.

The term "semi-automatically" used above to describe dimensional updating indicates what might happen in a complicated blocking structure. If a face is defined by a single domain with single connectors on each of the four edges, it is easy to see how changes in the number of points on one connector (edge) can be propagated to the opposite face edge. However, Figure 13 depicts a block face defined by domains numbered 1-6. In this example, if the dimension of connector A is changed, the connectors on the opposite side of domain 1 (connectors B, C and D) must be redimensioned such that the sum of their dimensions equals the new dimension of connector A. This in turn requires a redimensioning of connectors E, J, K, F, G and H. Clearly then, the redimensioning of a single connector can throw the dimensional consistency of an entire multiple block system out of balance. Notice that there exists no pre-definable manner in which the connectors in Figure 13 should be redimensioned to maintain consistency. In GRIDBLOCK

Version 8 the user will be led through the areas of ambiguity, being asked to make decisions along the way, until the entire block system is resized correctly.

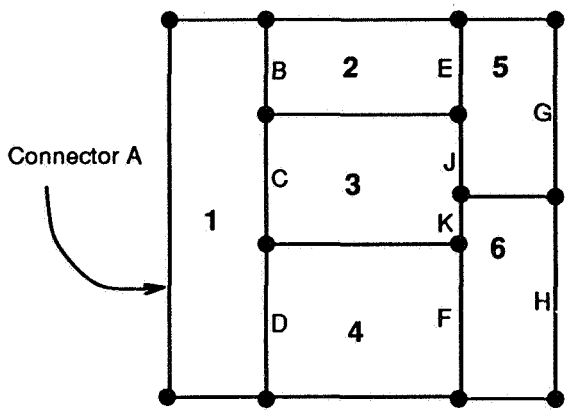


Figure 13: An example of an ambiguity in redimensioning a connector.

User Customization of GRIDGEN2D

Two tools were added to GRIDGEN2D to decrease the amount of user input required to generate surface grids: a verbosity setting and preferencing.

Verbosity setting refers to the ability to decrease the number of text prompts and menu options that the user of GRIDGEN2D Version 6 currently has to deal with. By reducing this, novice users will find GRIDGEN2D Version 8 easier to use because they won't be inundated with queries for lesser used options that may obscure the grid generation tasks. Also, experienced users will be able to quickly move through certain commands with fewer keystrokes. The verbosity setting may be either normal or terse. GRIDGEN2D with normal verbosity is the default setting, and it appears to the user as it always has. In terse mode, GRIDGEN2D hides certain menu buttons from the user (e.g., the button for Change Spline Fit from the edge shape definition menu) and eliminates seldom used options from text prompts. An example of the latter is the Algebraic Solver main menu command which is shown here in normal verbosity:

```
Interpolate       Method
-------------------------------------------------------
x,y,z(i,j)     1. arclength based (Soni) TFI (default)
               2. linear (original) TFI
               3. polar coordinate TFI
               4. TFI with boundary orthogonality
               5. TFI  stretching along i = const. lines
               6. TFI  stretching along j = const. lines
               7. TANH stretching along i = const. lines
               8. TANH stretching along j = const. lines
x,y,z(u,v)     9. fit grid to parametric surface shape
u,v(i,j)      10. interp. u,v and fit to par. surface
x,y,z(u,v)
```

```
-----------------------------------------------------
Select interpolation scheme 1-9. (default = 1)
```

and here in terse verbosity:

```
Interpolate       Method
-----------------------------------------------------
x,y,z(i,j)     1. arclength based (Soni) TFI (default)
               3. polar coordinate TFI
-----------------------------------------------------
Select interpolation scheme 1,3. (default = 1)
```

Preferencing refers to the ability of an experienced GRIDGEN user to specify ahead of time the types of certain grid generation options to be used such that the related text prompts or menu buttons don't appear when the code is run. For example, it is possible to set arclength based TFI as a preference such that when the Algebraic Solver main menu command is invoked, no prompts for the specific method appear (as shown above) and arclength based TFI is run automatically. Preferences may be set for various edge generation options (edge shape, distribution function, etc.), algebraic solver (method type), and elliptic PDE solver (control function, relaxation factor, etc.).

Using both terse verbosity and preferencing, it is now possible to start the elliptic PDE solver running simply by invoking the main menu command, rather than having to answer the dozen text prompts displayed by Version 6. It is also possible to have grid points distributed on an edge in the same automated fashion.

The verbosity setting, preferencing and several other customizable features can be set in any of three ways: interactively, through the new Customization main menu command; through environment variables at runtime; or through a run commands file called `.ggrc` that contains a namelist of the various options.

IMPROVED GEOMETRIC INTERFACING

It is well understood in the CFD community that the transition from geometric definition of a configuration (on a CAD system) to a computational grid is one of the major hurdles in the overall CFD process [2]. Two approaches for smoothing this transition immediately come to mind. On one hand, grid generation methods could be added as modules to existing CAD packages [4]. On the other hand, CAD-like functionality could be added to existing grid generation software. An advantage to the latter approach is that grid generation users wouldn't have to learn to operate the CAD system, which, as technology stands today, is much more complex than the grid generator. This latter approach is the one adopted for the GRIDGEN system.

GRIDGEN Version 6 required geometry models ordered in a discrete point array form, so that a connected rendering of the model (database, in GRIDGEN terminology) would appear as a wireframe model. A GRIDGEN database for an F-15 SMTD aircraft is displayed in Figure 14.

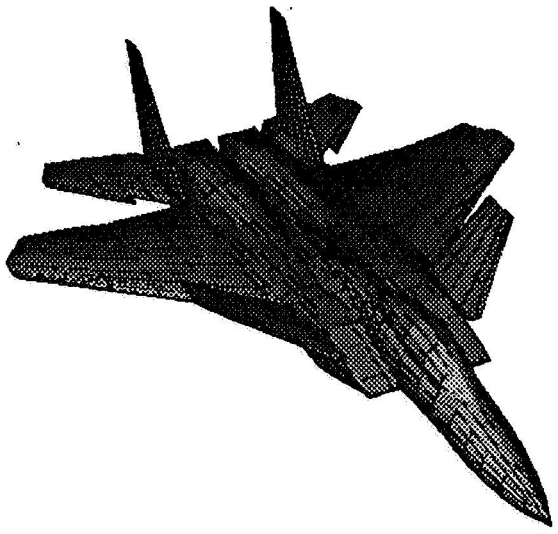


Figure 14: A discrete database for the F-15 fighter.

Unfortunately, a number of limitations are inherent in a discrete database. First, a discrete approximation to a mathematically continuous model is often a poor representation of the geometry, particularly when the database is either sparsely discretized or if the model has regions of large surface curvature. GRIDGEN places grid points on the faceted model of a continuous geometry. Secondly, a discrete database is not readily output by standard CAD systems, since most use a mathematical surface form more sophisticated than simple point data. This often forced the CAD operator to generate a database by piecing together a collection of discretized cross sectional cuts, a tedious and error-prone process.

GRIDGEN Version 8 will support entities written in the PATRAN Neutral File format. PATRAN [10] is a general engineering software system originally designed for solid mechanics applications, and the neutral file is a format intended for access by other computer hardware and software. The entities to be supported in this format are cubic curves for edge shape definition, and bi-cubic patches and trimmed surface models for surface shape definition.

GRIDGEN Version 8 will also support entities written in the Initial Graphics Exchange Specification [11] (IGES) format supported by the National Institute of Standards and Technology (formerly the National Bureau of Standards). This format, supported by most CAD packages, will provide a direct link between the CAD software and the GRIDGEN codes. Seven IGES entity types are immediately useful in GRIDGEN, including the parametric curves and surfaces, the rational B-spline curves and surfaces, and trimmed surface entities. Implementation of the rational B-spline entities will be done through extensive use of the DT_NURBS library [12], a library of Fortran subroutine B-spline utilities developed for the Navy's David Taylor Research Center.

Implementation of the improved geometric interface will begin with GRIDBLOCK and GRIDGEN2D extensions to read and interpret surface and curve geometries in either IGES or PATRAN formats. From there several new connector segment shapes will be added to GRIDBLOCK, including a free curve segment based on B-splines, a conic section segment based on rational B-splines (which will complement the current ellipse segment), and surface-constrained

curve segments allowing general curves to be drawn directly on the IGES and/or PATRAN surfaces. In addition, an interface will be developed which uses the DT_NURBS library for calculating intersection curves on two NURBS surfaces.

The two main surface grid generation algorithms in GRIDGEN2D will also require modification to accept the IGES and PATRAN formats. In the conventional surface solver, only two of the three Cartesian coordinates of surface grid points are calculated via a surface form of Poisson's equation, and the third is calculated by interpolating from the geometric surface. If the defining surface is in IGES or PATRAN form (rather than the GRIDGEN database format), ray tracing routines from the DT_NURBS library will be accessed to interpolate the third physical coordinate from the surface. In the parametric surface solver, surface grid points are calculated as a solution to Poisson's equation transformed into the surface's intrinsic (parametric) coordinates. The resulting parametric coordinate solution is then transformed back to physical coordinates via the geometry data. Hence, GRIDGEN2D's parametric solvers will be upgraded to read both IGES and PATRAN data as the parametric surface representation.

ALGORITHMIC IMPROVEMENTS

This section describes enhancements to the volume grid generation software, GRIDGEN3D, that are primarily designed to improve the quality of the resulting grid.

Control Functions

GRIDGEN3D is a batch code that takes as input the surface grids generated by GRIDGEN2D (see Figure 1) and creates the grid on the interior of each block in the system using algebraic and PDE methods. Determining the quality of the resulting grid is a nebulous task at best. However, the attributes of smoothness, clustering, and orthogonality are qualitative measures that are often cited as desirable. In GRIDGEN3D, these qualities are obtained by the PDE methods through the use of several control function formulations as shown in Figure 15 [13],[14]. The reader is referred to Reference [9] for a more detailed explanation of this material. It is unfortunate, however, that each of these control functions concentrates on a single quality measure. A new control function type was developed for use in GRIDGEN3D that is a hybrid of the types described above. This hybrid control function is also available in GRIDGEN2D.

The concept of the hybrid control functions involves applying two control functions simultaneously. One, the background control function, tends to affect the grid near the block interior. The other, the foreground control function, tends to affect the grid near the block surfaces. As applied in GRIDGEN3D, the background control function may be either LaPlace, Thomas & Middlecoff, or Fixed Grid (see Figure 15). The foreground control function is always Sorenson.

By way of example, consider the Thomas & Middlecoff background control functions. The goal of these control functions is to cluster grid points on the interior of the block based on the clustering on the six faces. They are derived based on the assumption that grid lines transverse to a face are locally straight and intersect the face orthogonally. The calculation proceeds by computing control function components on each face as follows:

control function	effect on grid
LaPlace	smooth variation of cell volume, tending toward evenly distributed points, throughout the interior
Thomas & Middlecoff	clustering on block interior is based on clustering on faces
Fixed Grid	kinks in the original grid are removed
Sorenson	orthogonality of grid lines with faces, tight clustering at faces, LaPlace effect on interior

Figure 15: Effect of control functions on the volume grid.

$$\Phi_b = \frac{\vec{r}_\xi \cdot \vec{r}_{\xi\xi}}{\vec{r}_\xi \cdot \vec{r}_\xi} \quad \text{on j} = 1,\ \text{j} = \text{J},\ \text{k} = 1,\ \text{k} = \text{K} \tag{1}$$

$$\Psi_b = \frac{\vec{r}_\eta \cdot \vec{r}_{\eta\eta}}{\vec{r}_\eta \cdot \vec{r}_\eta} \quad \text{on k} = 1,\ \text{k} = \text{K},\ \text{i} = 1,\ \text{i} = \text{I} \tag{2}$$

$$\Omega_b = \frac{\vec{r}_\zeta \cdot \vec{r}_{\zeta\zeta}}{\vec{r}_\zeta \cdot \vec{r}_\zeta} \quad \text{on i} = 1,\ \text{i} = \text{I},\ \text{j} = 1,\ \text{j} = \text{J} \tag{3}$$

Two-dimensional TFI is then used to interpolate Φ_b onto each i constant plane, Ψ_b onto each j constant plane, and Ω_b onto each k constant plane. By themselves, the Thomas & Middlecoff control functions are very robust and result in good grids. A plane from a volume grid computed using Thomas & Middlecoff control functions is shown in Figure 17b. Note, however, that the assumption of orthogonality at the faces used in the derivation, does not result in practice.

On the other hand, Sorenson control functions (used in the foreground) are computed to enforce a specific clustering and transverse angle at each face. Given any grid, one can compute the control functions at each point based on the computational derivatives of the Cartesian coordinates. Sorenson's method uses this relationship to compute the control functions on each of the six faces but only after the computational derivatives transverse to the face have been altered to enforce orthogonality and clustering. Consider computation of Φ_f on the $\zeta = \zeta_{min}$ face. At each point on the face one can compute the derivatives $\vec{r}_\xi$, $\vec{r}_{\xi\xi}$, $\vec{r}_\eta$, $\vec{r}_{\eta\eta}$, and $\vec{r}_{\xi\eta}$ using finite differences and the known grid point coordinates on the face. The derivative transverse to the face $\vec{r}_\zeta$ is not known. Once it is, however, it will be possible to compute $\vec{r}_{\zeta\zeta}$ (from $\vec{r}_\zeta$ and the transient volume grid) and $\vec{r}_{\eta\zeta}$ and $\vec{r}_{\zeta\xi}$ (by differencing $\vec{r}_\zeta$). The problem is then reduced to determining appropriate values for $\vec{r}_\zeta$ to enforce orthogonality.

$\vec{r}_\zeta$ is computed at each point on the face based on user-specified angle and spacing constraints, which in turn are calculated from the known angle and spacing data on the four edges of the face. Referring to Figure 16, the grid point's coordinates on adjacent faces allow us to compute the arclength spacing Δs, and the two intersection angles θ and δ on four edges using

$$\Delta s = \mid \vec{r}_\zeta \mid \tag{4}$$

$$\cos\theta = \frac{\vec{r}_\zeta \cdot \vec{r}_\xi}{|\vec{r}_\zeta||\vec{r}_\xi|} \tag{5}$$

$$\cos\delta = \frac{\vec{r}_\zeta \cdot \vec{r}_\eta}{|\vec{r}_\zeta||\vec{r}_\eta|} \tag{6}$$

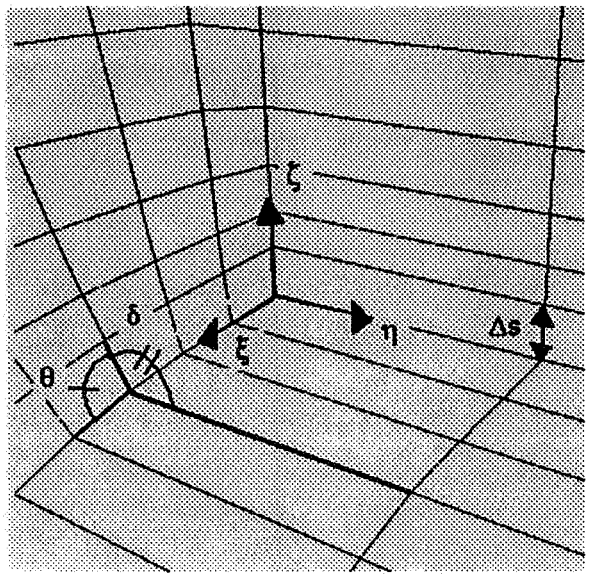


Figure 16: Computation of angle and spacing constraints for the Sorenson control functions.

These constraints are then interpolated from the four edges onto the interior of the face using TFI. The spacing Δs is interpolated using computational based LaGrange blending functions and the angles θ and δ are interpolated using computational based exponential blending functions that blend the edge angles to $\frac{\pi}{2}$ on the face interior. Then, the angle and spacing constraints are used to compute $\vec{r}_\eta$ on the face interior followed by computing Φ_f on the face.

In the standard Sorenson scheme, the control functions are computed on each of the six faces and then interpolated onto the block interior using TFI with computational based exponential blending functions that decay the control functions to zero on the block interior. An example of this method is shown in Figure 17c. One can see that the grid is indeed clustered and orthogonal to the faces yet points in the interior are smoothly distributed, where the governing PDE is locally reduced from Poisson's to LaPlace's equation.

The best features of the Sorenson and Thomas and Middlecoff control functions are maintained with the hybrid control functions calculated as follows. The background control functions Φ_b, Ψ_b, and Ω_b are computed throughout the entire block and the foreground control functions Φ_f, Ψ_f, and Ω_f are computed on only the six block faces. First, the difference of background and foreground control functions are formed on the six faces as

$$\begin{aligned} \Phi_d &= \Phi_f - \Phi_b \\ \Psi_d &= \Psi_f - \Psi_b \\ \Omega_d &= \Omega_f - \Omega_b \end{aligned} \tag{7}$$

Next, the control function differences are distributed from the faces to the block interior using TFI with exponentially decaying blending functions such that the differences vanish far into the block interior. Finally, hybrid control functions are computed by adding the background functions to the difference functions. The resulting functions will then have $\Phi \rightarrow \Phi_f$ near the block faces, and $\Phi \rightarrow \Phi_b$ near the block interior, with equivalent relations for the Ψ and Ω control functions. Thus, the hybrid control functions locally assume the form of the background or foreground control functions in the regions where the particular control functions are best suited.

$$\begin{aligned} \Phi &= \Phi_d + \Phi_b \\ \Psi &= \Psi_d + \Psi_b \\ \Omega &= \Omega_d + \Omega_b \end{aligned} \tag{8}$$

An example of the hybrid combination of Thomas & Middlecoff and Sorenson control functions is shown in Figure 17d. One can see that not only are the angle and spacing constraints enforced at the faces but the grid clustering is carried through the block interior.

Numerical Method

Solution of Poisson's Equation in GRIDGEN3D is performed using an explicit, pointwise SOR algorithm. The robustness and efficiency of this algorithm was improved by the addition of the following features.[¶]

1. Variable sweep direction. Each iteration begins the SOR sweep through the i,j,k indices in a different corner of the block to prevent biasing of the grid in any one computational direction.
2. One sided differencing. Forward or backward differencing of the coordinate derivatives based on the sign of the control function improves numerical stability by making the system of equations more diagonally dominant.
3. Grid sequencing. In grid sequencing, the PDEs are first solved on a coarse grid consisting of a sparse subset of the full grid. The solution on the coarse grid converges more rapidly in the PDE solver. At some point the coarse grid solution is stopped and the grid point coordinates and control functions on the full grid are interpolated from the coarse grid and the PDE solution proceeds on the full grid. The net computer time required to converge the grid is drastically reduced.

CONCLUSION

The improvements discussed in this paper are expected to move the GRIDGEN system to a new level of user efficiency and geometric and computational accuracy. Initial release of the new

[¶]These three features are also available in GRIDGEN2D.

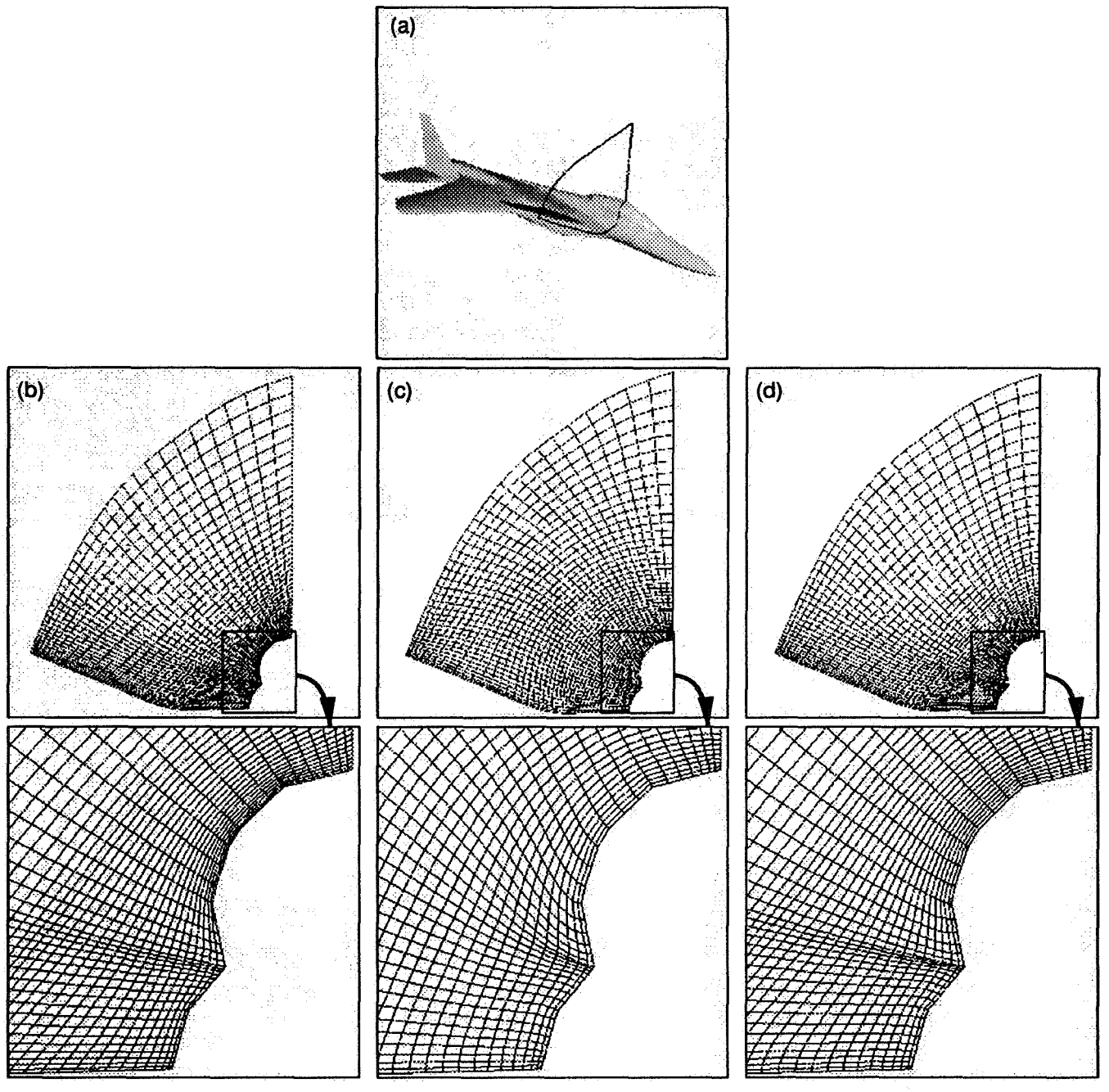


Figure 17: For a plane from a 3D grid (a) three control function formulations are compared: Thomas & Middlecoff (b), Sorenson (c), and hybrid Thomas & Middlecoff and Sorenson (d).

software, which will remain in government contractor domain, is anticipated near mid-year 1992. Long-term plans for GRIDGEN beyond this date include further consolidation of the four component codes, increased grid automation and rework elimination through the new geometry hierarchical structure, further development of CAD system functionality, development and incorporation of improved grid quality procedures such as grid adaption, and expansion of the GRIDGEN umbrella to unstructured grid techniques.

ACKNOWLEDGEMENT

GRIDGEN upgrades are being performed under subcontract to the Computer Sciences Corporation, and under contract to the Air Force Wright Laboratory at Eglin Air Force Base. The authors would like to acknowledge Dr. Robert Smith of NASA Langley Research Center and Mr. Robert Ames of David Taylor Research Center and their respective agencies for initiating and sponsoring these efforts. The authors also acknowledge Dr. Donald Kinsey of the Air Force Wright Laboratories at Wright Patterson AFB for sponsoring GRIDGEN's initial development.

REFERENCES

1. Steger, J.L.: *Technical Evaluation Report on the Fluid Dynamics Panel Specialists' Meeting on Application of Mesh Generation to Complex 3-D Configurations.* ed. by Schmidt, W., AGARD-AR-268, AGARD, France, March 1991.

2. Smith, R.; Choo, Y.; Van Dalsem, W.; and Bircklebaw, L.: *Directions for Surface Modeling/Grid Generation in NASA.* 1991 NASA CFD Conference, NASA Ames Research Center, Mar. 1991.

3. Thompson, J.F; et al.: 'Program EAGLE Numerical Grid Generation System Users Manual." AFATL-TR-87-15, Vols. I-III, Air Force Armament Laboratory, March 1987.

4. Seibert, W.: "A Graphic-Iterative Program-System to Generate Composite Grids for General Configurations." *Numerical Grid Generation in Computational Fluid Mechanics '88*, edited by S. Sengupta et al., Pineridge Press Ltd., Swansea, U.K., 1988.

5. Amdahl, D.J.: "Interactive Multi-Block Grid Generation." *Numerical Grid Generation in Computational Fluid Mechanics '88*, ed. by S. Sengupta et al., Pineridge Press Ltd., Swansea, U.K., 1988.

6. Abolhassani, J.; Sadrehaghighi, I.; Smith, R.E.; and Tiwari, S.N.: "Application of Lagrangian Blending Functions for Grid Generation Around Airplane Geometries." *Journal of Aircraft*, Vol. 27, No. 10, October, 1990, pp. 873-877.

7. Raj, P.; et al.: "Three-Dimensional Euler Aerodynamic Method (TEAM)." AFWAL-TR-87-3074, Vols. I-III, Flight Dynamics Laboratory, Wright Research and Development Center, 1987.

8. Cooper, G.K.; and Sirbaugh, J.R.: "PARC Code: Theory and Usage." AEDC-TR-89-15, Arnold Engineering Development Center, Arnold Air Force Base, 1989.

9. Steinbrenner, J.P.; Chawner J.R.; and Fouts, C.L.: "The GRIDGEN 3D, Multiple Block Grid Generation System." WRDC-TR-90-3022, Vols. I and II, Wright Patterson Air Force Base, 1990.

10. *PATRAN Plus User Manual.* PDA Engineering, July, 1987.

11. Smith, B.; et. al.: *Initial Graphics Exchange Specification (IGES) Version 4.0.* U.S. Department of Commerce NBSIR 88-3813, June 1988.

12. *David Taylor Research Center Spline Geometry Library DT_NURBS Users Manual.* Boeing Computer Services, February, 1990.

13. Thomas, P.D.; and Middlecoff, J.F.: "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations." *AIAA Journal*, Vol. 18, 1979, pp. 652-656.

14. Sorenson, R.L.: "The 3DGRAPE Book: Theory, Users Manual, Examples." NASA TM-102224, NASA Ames Research Center, July, 1989.

Cell Volume Control at a Surface for Three-Dimensional Grid Generation Packages

Stephen J. Alter*,
Lockheed Engineering & Sciences Company, Hampton, Virginia
and
Kenneth J. Weilmuenster†
NASA Langley Research Center, Hampton, Virginia

Abstract

This paper presents an alternate method of calculating the cell size for orthogonality control in the solution of Poisson's three-dimensional space equations. The method provides the capability to enforce a better initial guess for the grid distribution required for boundary layer resolution. This grid point distribution is accomplished by enforcing grid spacing from a grid block boundary where orthogonality is required. The actual grid spacing or cell size for that boundary is determined by the two or four adjacent boundaries in the grid block definition, which are two dimensional grids. These two dimensional grids are in turn defined by the user using insight into the flow field and boundary layer characteristics. The adjoining boundaries are extended using a multi-functional blending scheme, with user control of the blending and interpolating functions to be used. This grid generation procedure results in an enhanced Computational Fluid Dynamics calculation by allowing a quicker resolution of the configuration's boundary layer and flow field and by limiting the number of grid re-adaptions. The cell size specification calculation has been applied to a variety of configurations ranging from axisymmetric to complex three-dimensional configurations. Representative grids will be shown for the Space Shuttle and the Langley Lifting Body (HL-20).

Introduction

Design and development of aerodynamic configurations in industry is primarily being done with inviscid solvers as well as intricate panel methods. [1, 2, 3] These simplified solvers are being employed as methods of homing in on optimum designs for a variety of contracts

*Aeronautical and Astronautical Engineer
†Senior Research Engineer, Aerothermodynamics Branch, Space Systems Division, Member AIAA

and developmental programs. Yet most contractors find it necessary to further optimize the designs through detailed studies and gathering of experimental data [1].

Although the design processes employed by industry optimize configurations based on designed mission profiles, the capabilities for acquiring experimental data at these design points are limited. As a result, Computational Fluid Dynamics (CFD) is being used as an alternative source of detailed evaluation data. Most users in the CFD community would prefer to use CFD codes modeling viscous effects in the design process because they model the non-linear effects of the flow characteristics [4]. By utilizing this type of CFD code, a more competitive environment is established for design optimization, resulting in better configuration designs. However, current CFD codes can not be used as design tools, simply because they are still too time consuming to be used in parametric studies.

Current research and development in the CFD arena has been focusing on accurate simulation of viscous flows [5, 6, 7]. The area of most development has been in the generation of Navier-Stokes solvers, with emphasis on the algorithms employed as opposed to the resolution of the grid for numerical accuracy. As progress has been made in the generation of these viscous flow solvers, more complicated configurations are being envisioned and designed, especially in the area of high speed aerodynamics. Some typical complex configurations include the National Aerospace Plane [8] to the advanced fighter designs [9]. The increased complexity of configurations has led to more complex solution algorithms in order to gain numerical accuracy.

There are several different methods for obtaining numerical accuracy based on the grid. One way is to re-adapt the grid being used based on flow characteristics. This type of procedure will allow for most efficient use of the available grid points. With the emergence of the SAGE code [10] to the shock alignment procedure that is used in the Langley Aerothermodynamic Upwind Relaxation Algorithm (LAURA) code [11], numerical accuracy via grid re-adaption is being made possible. By obtaining numerical accuracy in this fashion, the number of grid points necessary to solve the flow field calculations can be reduced with a corresponding rapid resolution of the flow characteristics. Furthermore, these re-adaption codes assume a solution has been generated and grid adaptation is done to improve numerical accuracy and convergence.

However, the original grids used by the solvers to obtain an initial solution for grid adaptation have grids that are coarse near the surface, or wall. The coarse spacing typically results from the methods used to determine the cell sizes necessary to calculate the forcing functions of the elliptic solvers. The most commonly used method for determining cell sizes for orthogonality control is Trans Finite Interpolation (TFI). TFI is based on maintaining the overall grid distribution as defined by the boundaries of the grid block flow domain or surface edges. Yet, as a result of the dependency on the opposing flow domain boundaries, the grid lines that result may have a highly skewed apperance (figure 1). This skewness of grid lines from the surface will cause the overall cell size to be much larger as opposed to the grid line being orthogonal to the surface. This leads to excessive grid re-adaptations to resolve the boundary layer.

In order to generate a quality grid for a Navier-Stokes CFD calculation, the grid distribution must allow for the accurate resolution of the viscous forces at a solid boundary. The grid spacing is controlled in elliptic solvers by the specification of the first cell sizes off of the configuration's surface and the decay rate of the forcing functions for orthogonality; this pa-

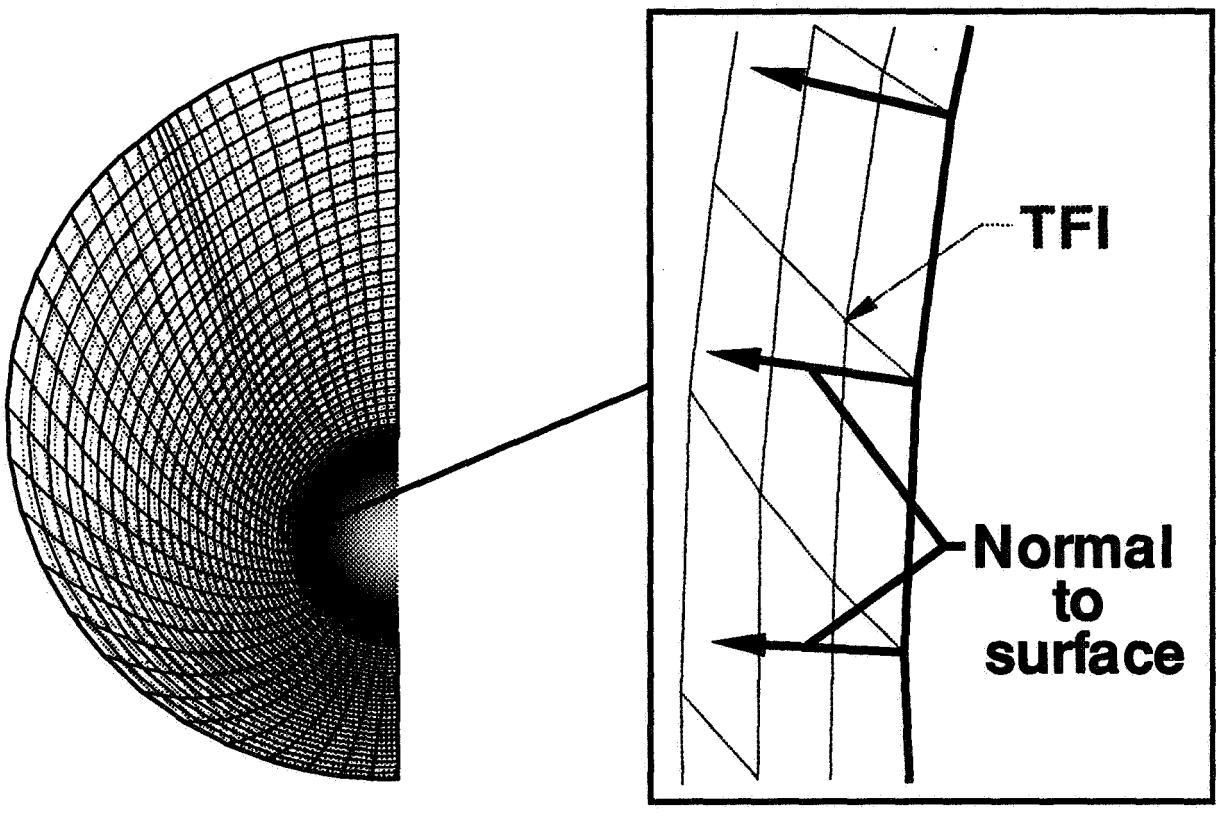


Figure 1: Grid line skewness produced by TFI.

per will only consider the latter. Thus a better guess as to the surface cell sizes necessary to model the boundary layer flow characteristics, with the use of local grid densities is needed.

The areas that require high grid densities and small cell sizes can be identified using several methods. One might be able to use current databases of experimental data for configurations having similar geometric characteristics and evaluated under similar flight conditions. Another might be to use linear panel theory. Nonetheless, the general areas of concern for proper boundary layer modeling can be determined and by having proper surface and near-surface grid densities, accuracy and solution convergence rates can be maximized.

The purpose of this paper is to present an alternate method that links the problem of a better initial grid to the solving of Poisson's three dimensional space equations. The method allows the user to use insight to determine the "best" guess of the local cell size for different computational regions and define the cell sizes on the surface domain with a multi-functional blending scheme. This blending is done by smearing the Local ARc length Cell Sizes (LARCS) based on the configuration's surface edges, into two distinct planes that represent the first interior points from the configuration's surface. Then a continuous hyperbolic function is used to combine these interior planes to generate a single plane representing the cell sizes off the configuration's surface. The blending will result in a smooth cell size transition between all defining surface edges and will remove the dependency on the opposing grid block boundary.

Method

The overall goal of this method is to take the best guess of the cell size to be used in given areas of a configuration for boundary layer resolution and propagate the cell sizes onto the configuration surface. The resulting blend will be smooth from computational region to computational region. This process begins by determining the local cell sizes of the grid to use on the defining boundaries of the grid blocking structure (figure 2).

The local wall cell size Δs_w, given in equation 1

$$\Delta s_w = \frac{Re_w \mu_w}{\rho_w a_w} \tag{1}$$

is based on the cell Reynold's Number at the surface. By setting the local cell Reynold's number to a value of 2.0 or less [11], the proper cell size can be determined (figure 3). This Reynold's number and the resulting cell size will enable a CFD code to capture the viscous forces in the boundary layer [15, 16]. By accomplishing this capture early in the CFD calculation of the viscous flow field, enhanced convergence of the boundary layer will occur resulting in a reduced number of successive grid re-adaptions. Hence, the flow resolution should be expedited.

The other parameters required to calculate the cell sizes include the local temperature (T_w) and density (ρ_w). These parameters can be determined by any number of methods. Some of these include comparing experimental or computational results of a similar configuration, using some of the relationships for ideal or perfect gases [12] or a simple impact theory program such as APAS [3]. In any case the overall goal is to determine a local density, and temperature for a given region on the surface of the configuration.

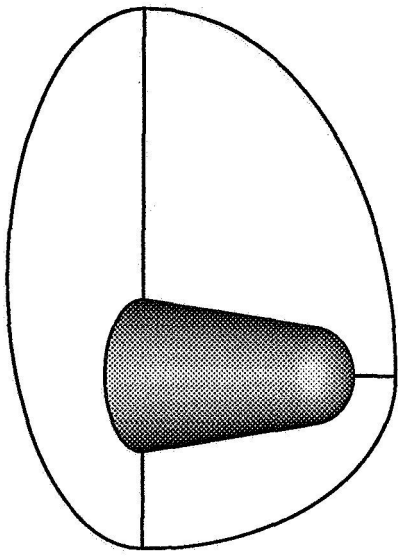


Figure 2: Block structure to be utilized in defining flow field domain.

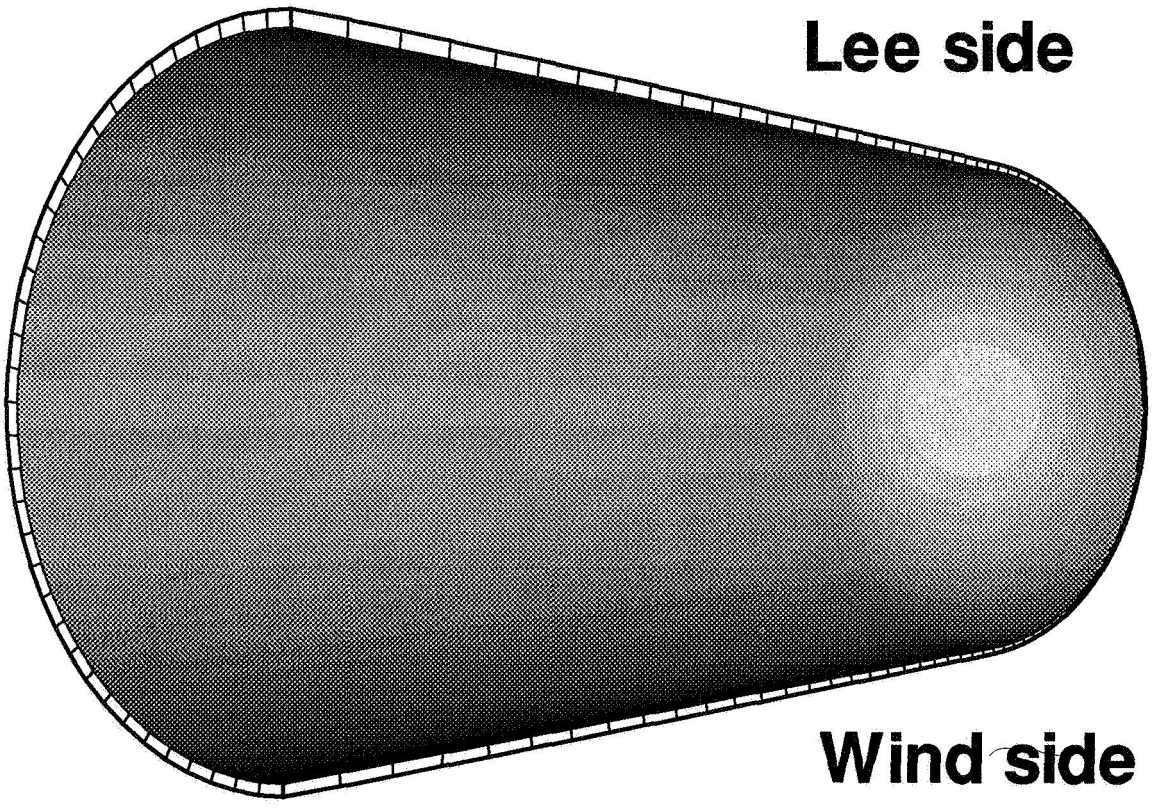


Figure 3: Cell size on a surface based on an initial guess with large cell sizes on the leeside decreasing towards the windside.

The temperature is used to calculate two other needed parameters. The first is the local speed of sound (equation 2)

$$a_w = \sqrt{\gamma R T_w} \tag{2}$$

used in the cell Reynold's number equation. The second parameter that is derived from temperature is the viscosity. It can be determined by using Sutherland's Law [13], equation 3 or reference [14] for those cases where temperatures are above 2000deg K.

$$\mu_s = 1.4584 \times 10^{-5} \frac{(T^oK)^{3/2}}{(T^oK) + 110.33} \tag{3}$$

Thus the initial cell sizing parameter calculation is dependent on the user's discretion, but the resulting cell size should represent the type of flow characteristics expected. For example, if a cone was being evaluated at an angle of attack of 40 degrees, one would expect cell sizes on the leeward side to be large compared to the windward side. (figure 3.)

After determining the cell sizes to be utilized in the different regions of the block boundaries, a grid generator such as GRIDGEN [17], PATRAN[18], etc., is used to define the two-dimensional grids for each face of the grid block (figure 4). Then three-dimensional Poisson solvers [19] are employed to create the volume grid for computational analysis. In order to solve these equations to obtain orthogonality on the block boundary surfaces of choice, the cell size, computed by equation 4,

$$\Delta s = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{4}$$

has to be known. It is this cell size that LARCS is used to determine.

The determination of the cell sizes emanating from any given point on a configuration's surface by LARCS begins with the functionalization of the cell sizes of the adjacent boundaries with respect to their parametric coordinates. For example, if cell sizes for the surface of the configuration in figure 5 are being determined, the defining boundaries are the leeside and the windward side symmetry planes (faces 3 and 4 respectively), the pole boundary (face 1) and the exhaust plane (face 2). Next the cell size as a function of parametric coordinate is determined for each edge (figure 6.) using the standard arc-length equation 4.

From the functionalization, two distinct planes representing cell sizes off of the configuration's surface are created. The first plane is generated by blending from the cell size functionalization on face 1 to that of face 2, in the ξ direction. Similarily, the second plane is a blend between face 3 and face 4 in the η direction. These planes can be generated utilizing linear or parabolic blending. For the linear blending of faces 1 and 2 (figure 5), the basic blending coefficient calculated via equation 5

$$\sigma_\xi = \frac{\xi_{max} - \xi}{\xi_{max} - 1} \tag{5}$$

with the condition in equation 6

$$\sigma_{\xi_{1\to 2}} + \sigma_{\xi_{2\to 1}} = 1.0 \tag{6}$$

forms the blending coefficient of face 1 to face 2, equation 7

$$\sigma_{\xi_{1\to 2}} = \frac{\xi_{max} - \xi}{\xi_{max} - 1} \tag{7}$$

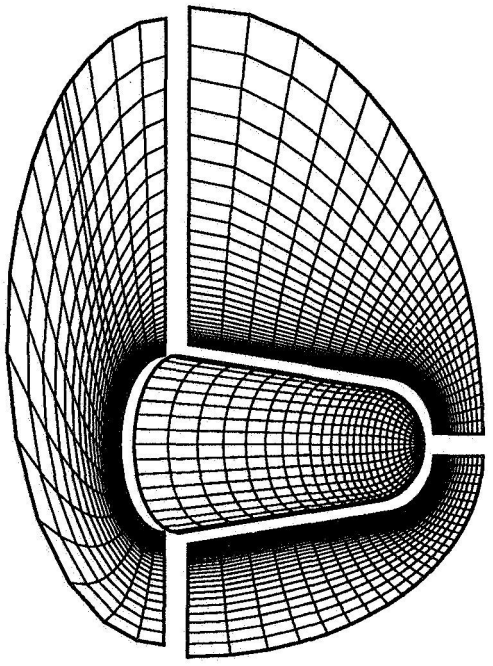


Figure 4: Two dimensional grid definitions on flow domain block boundaries.

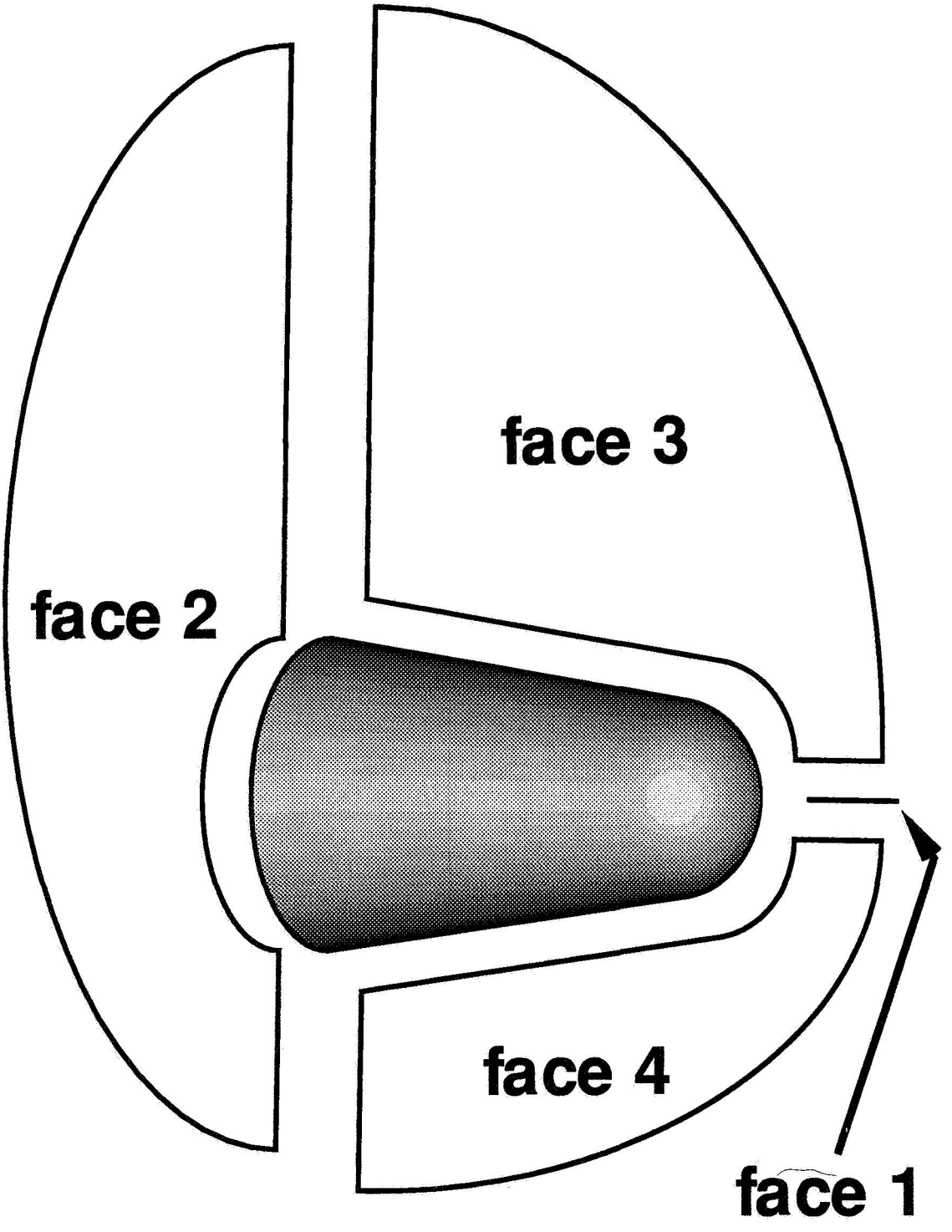


Figure 5: Simple cone for cell sizing calculation.

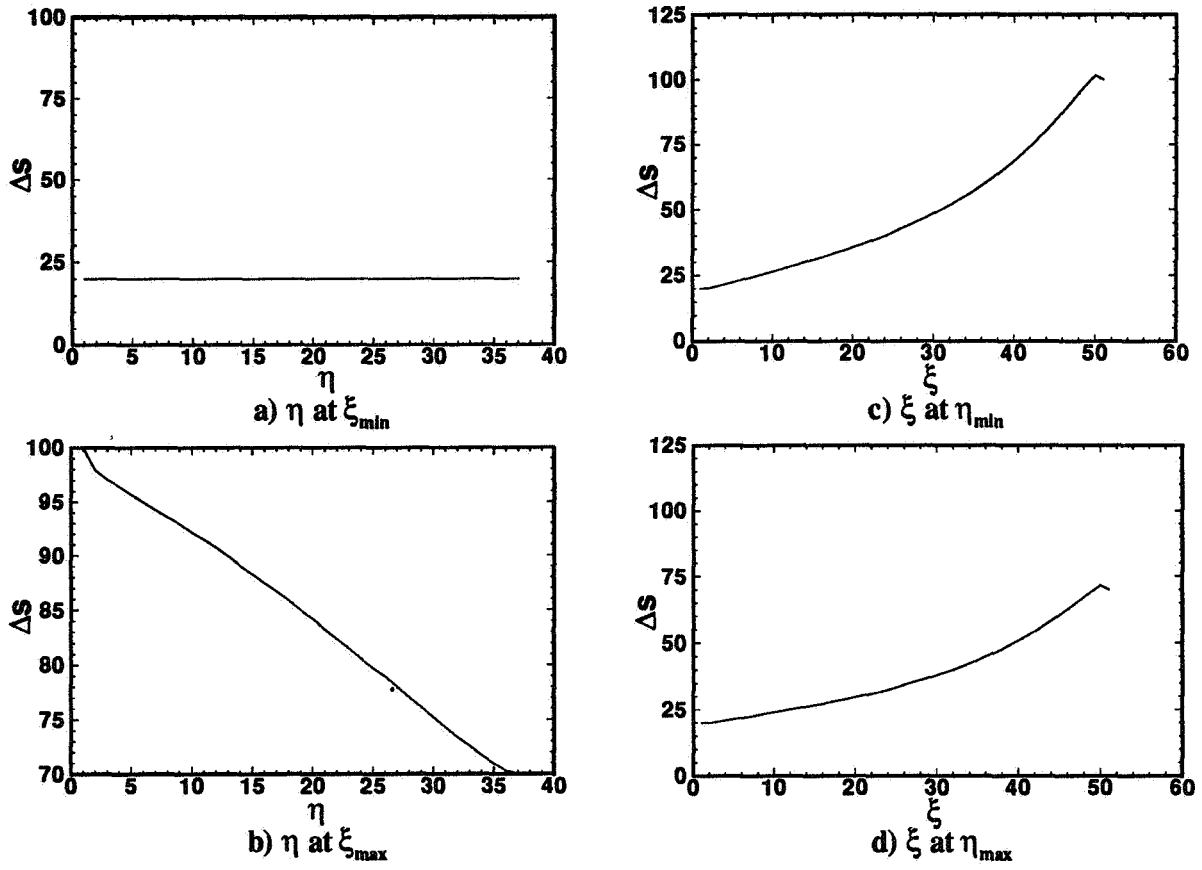


Figure 6: Cell sizing functionalization based on parametric coordinate for each edge of the configuration's surface.

and the blending coefficient of face 2 to 1, equation 8.

$$\sigma_{\xi_{2\to 1}} = 1 - \frac{\xi_{max} - \xi}{\xi_{max} - 1} \tag{8}$$

Using the linear blending, equation 9

$$\Delta s_{\xi,\eta} = \sigma_{\xi_{1\to 2}} \Delta s_{\xi_{min}} + \sigma_{\xi_{2\to 1}} \Delta s_{\xi_{max}} \tag{9}$$

substituting in equations 7 & 8, and combining like terms, equation 10 is used to blend from opposing faces 1 and 2.

$$\Delta s_{\xi,\eta_{1\to 2}} = \left(\frac{\xi_{max} - \xi}{\xi_{max} - 1}\right)(\Delta s_{\xi_{min}} - \Delta s_{\xi_{max}}) + \Delta s_{\xi_{max}} \tag{10}$$

Likewise, linear blending of opposing faces 3 and 4 can be written as in equation 11 (figure 7).

$$\Delta s_{\xi,\eta_{3\to 4}} = \left(\frac{\eta_{max} - \eta}{\eta_{max} - 1}\right)(\Delta s_{\eta_{min}} - \Delta s_{\eta_{max}}) + \Delta s_{\eta_{max}} \tag{11}$$

The parabolic blending is somewhat more complicated but can be derived the same way with the basic blending coefficient as described in equation 5 and the condition stated in equation 6. However, the blending coefficients $\sigma_{\xi_{1\to 2}}$ and $\sigma_{\xi_{2\to 1}}$, are different due to their parabolic nature; $\sigma_{\xi_{1\to 2}}$ becomes:

$$\sigma_{\xi_{1\to 2}} = \frac{\sigma_\xi^2}{1 - 2\sigma_\xi + 2\sigma_\xi^2} \tag{12}$$

and $\sigma_{\xi_{2\to 1}}$ becomes:

$$\sigma_{\xi_{2\to 1}} = \frac{1 - 2\sigma_\xi + \sigma_\xi^2}{1 - 2\sigma_\xi + 2\sigma_\xi^2} \tag{13}$$

Thus, the parabolic blending for face 1 to 2, is given by equation 14,

$$\Delta s_{\xi,\eta_{1\to 2}} = \frac{\sigma_\xi^2 \Delta s_{\xi_{min}} + (1 - \sigma_\xi)^2 \Delta s_{\xi_{max}}}{1 - 2\sigma_\xi + 2\sigma_\xi^2} \tag{14}$$

and the parabolic blending from faces 3 to 4 (figure 8) is given by equation 15.

$$\Delta s_{\xi,\eta_{3\to 4}} = \frac{\sigma_\eta^2 \Delta s_{\eta_{min}} + (1 - \sigma_\eta)^2 \Delta s_{\eta_{max}}}{1 - 2\sigma_\eta + 2\sigma_\eta^2} \tag{15}$$

In either case, the resulting surfaces from the linear blending, figures 9 and 10, as well as the surfaces from the parabolic blending, figures 11 and 12 represent two unique surfaces for specifying cell sizes for orthogonality control. For most symmetry planes this may indeed be true because the user may only want blending between the configuration surface and the outer flow domain. Yet for the block surfaces that depend more heavily on all four edges, these two artificial surfaces have to be combined. This will enable the attributes of each edge to affect the overall cell sizes used in the specification of orthogonality.

To generate a surface which has attributes of each of the four functionalized edges, the two artificial surfaces are combined with a hyperbolic blending scheme. This scheme becomes evident when viewing the configuration surface domain in parametric space. Consider figure

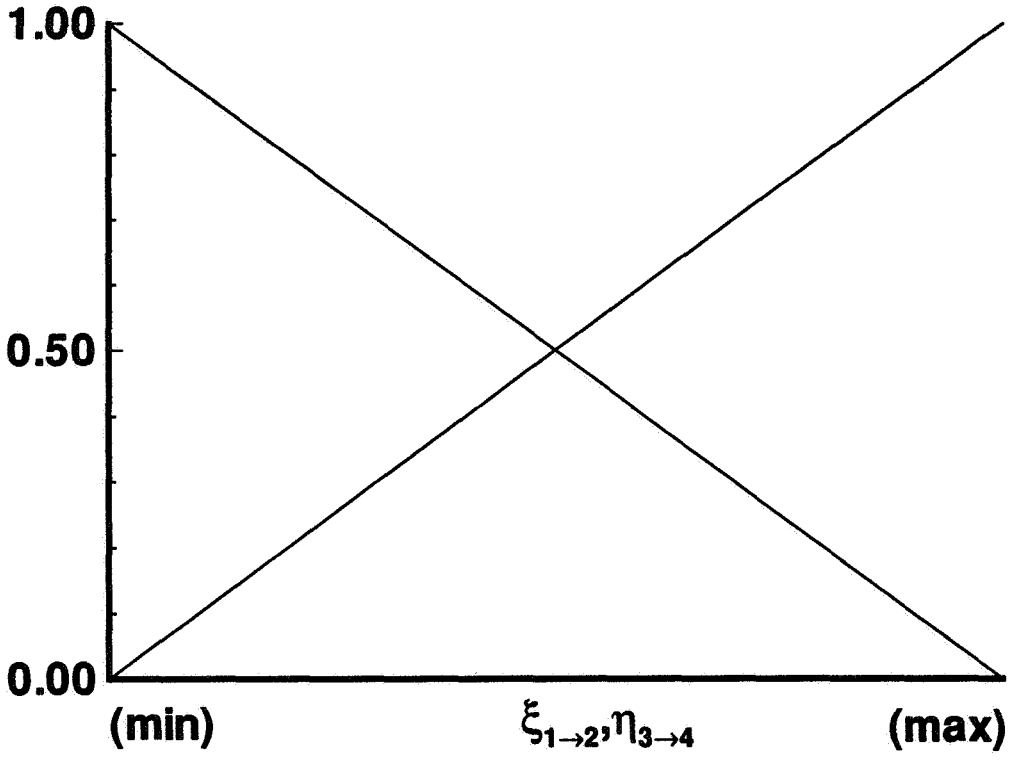


Figure 7: Linear blending functions between opposing faces 1 &2 and 3 & 4.

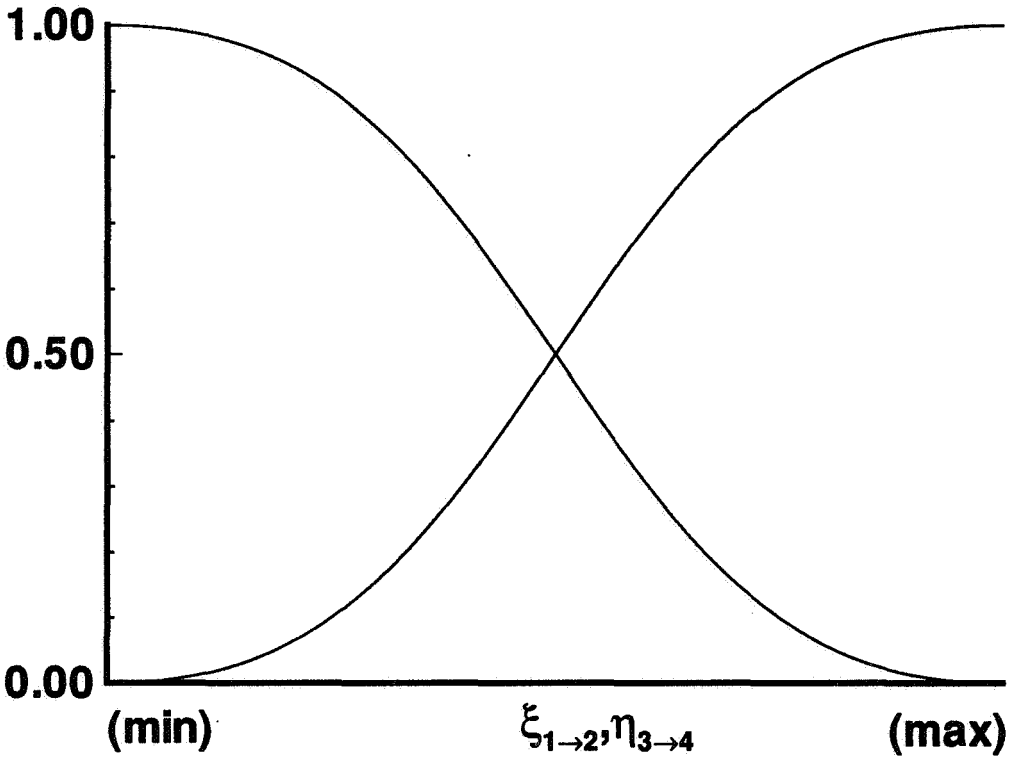


Figure 8: Parabolic blending functions between opposing faces 1 &2 and 3 & 4.

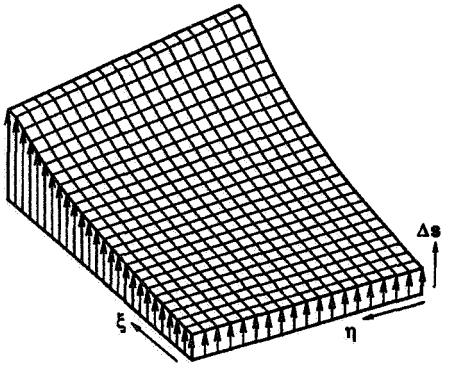


Figure 9: Artificial surface results from linearly blending face 1 to 2.

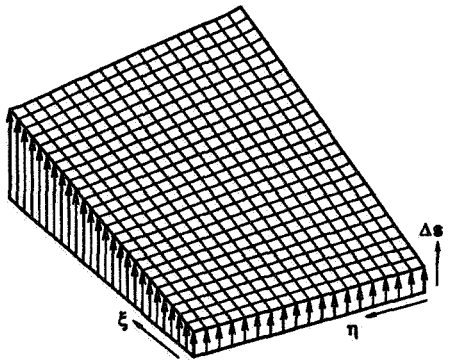


Figure 10: Artificial surface results from linearly blending face 3 to 4.

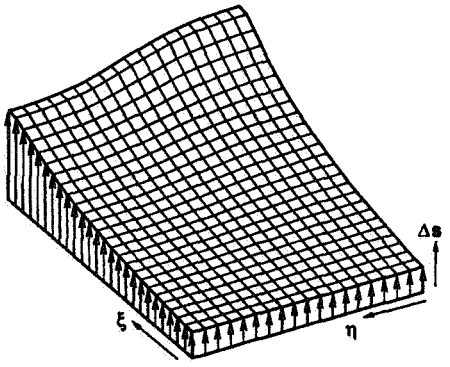


Figure 11: Artificial surface results from parabolically blending face 1 to 2.

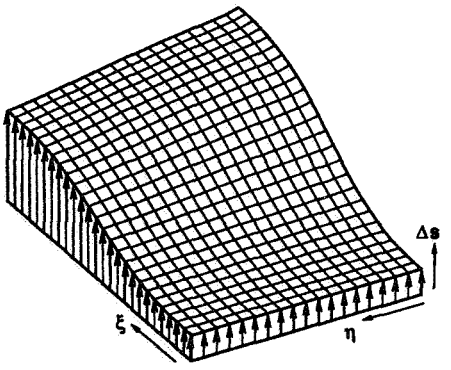


Figure 12: Artificial surface results from parabolically blending face 3 to 4.

13, where the percentages of the four edge functions are to be used to calculate the overall cell sizes. For the corners of the domain, the resulting cell sizing surface is made up of 50% of each function that meets at the respective corner, or the two individual artificial surfaces.

In addition, the center of the domain has 50% of each artificial surface. Also notice that at the middle of each edge, there is either a 100% or a 0% of the surface to be utilized. This arises merely from the initial linear and/or parabolic blending of the opposing face edge functions. For the surface that contains the actual edge function, the dominant edge, 100% of that function is to be use on that edge. Likewise, the other surface, the non-dominant edge function is to have no effect on the resulting hyperbolically blended surface, because it is either a line or a result of the initial parabolic blend.

By connecting the 50% regions with straight lines in the parametric domain the hyperbolic structure can be seen. Equations 16 and 17 represent the blending functions for face 1 to 2 and face 3 to 4 respectively, as shown in figure 14.

$$\delta_\xi = \frac{1}{2}|\alpha^2 - \beta^2 - 1| \tag{16}$$

$$\delta_\eta = \frac{1}{2}|\beta^2 - \alpha^2 - 1| \tag{17}$$

Here,

$$\alpha = 2\left(\frac{\xi_{max} - \xi}{\xi_{max} - 1}\right) - 1$$

and,

$$\beta = 2\left(\frac{\eta_{max} - \eta}{\eta_{max} - 1}\right) - 1 \quad .$$

Combining these blending coefficients, and hence the initial artificial surfaces, equation 18 results.

$$\Delta s_{\xi,\eta} = \delta_\xi \Delta s_{\xi,\eta_{1\to 2}} + \delta_\eta \Delta s_{\xi,\eta_{3\to 4}} \tag{18}$$

In addition, the plot of the percentages in parametric space represent two distinct blending function surfaces, one for each opposing face pair. Thus for any parametric point on the surface, certain percentages of each surface are used,and the combined percentages sum to 100%. Due to the smoothness of the blending function, the effects of all four edges are smeared throughout the resulting blended surface's domain.

By utilizing this function, the overall cell size necessary for orthogonality specification can be calculated and is based on the defining edges of the configuration's surface domain. The resulting blended surface as shown in figure 15 is a better description for cell size because it now limits the maximum distance a grid point can be from the configuration surface and is not as dependent on surface curvature as other methods used. In effect, a multi-functional blending scheme is used to determine the cell sizes. Thus, by utilizing the cell sizes determined by the LARCS method, the forcing functions necessary to control orthogonality within elliptic volume grid generators are more dependent on the users best guess. In doing so, the control of grid spacing near the wall is more related to the problem at hand, as opposed to the rapid solution of the elliptic Poisson's equations.

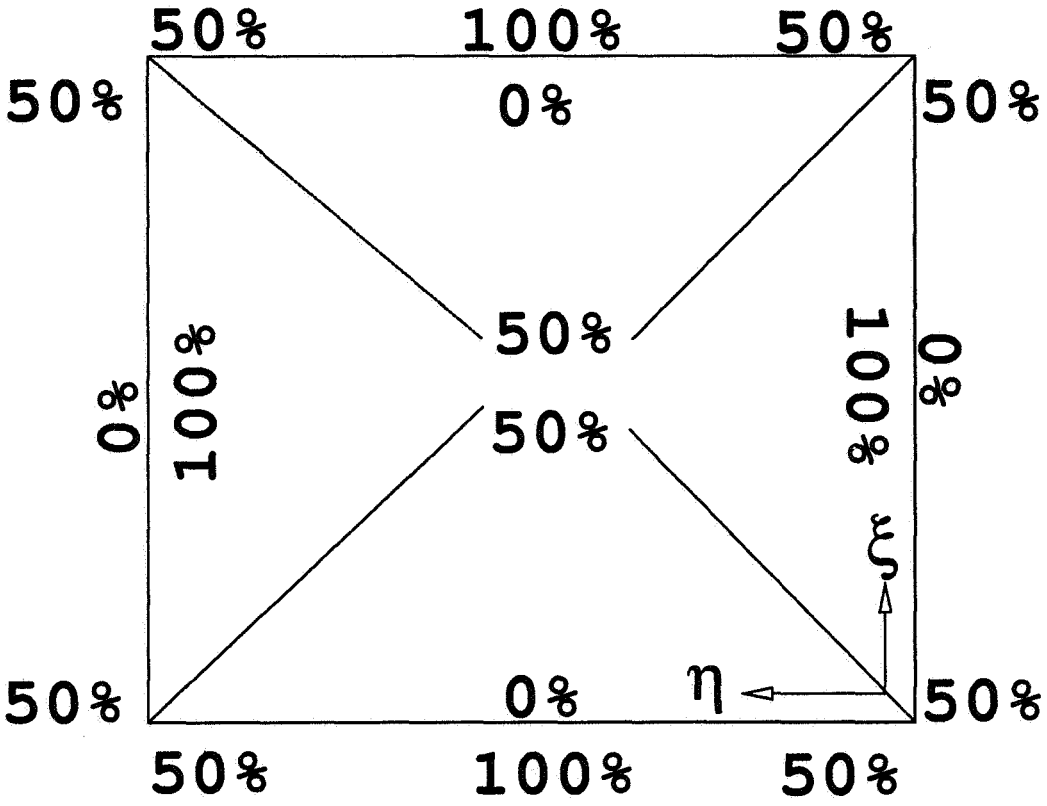


Figure 13: Percentages of surface functions in the parametric domain.

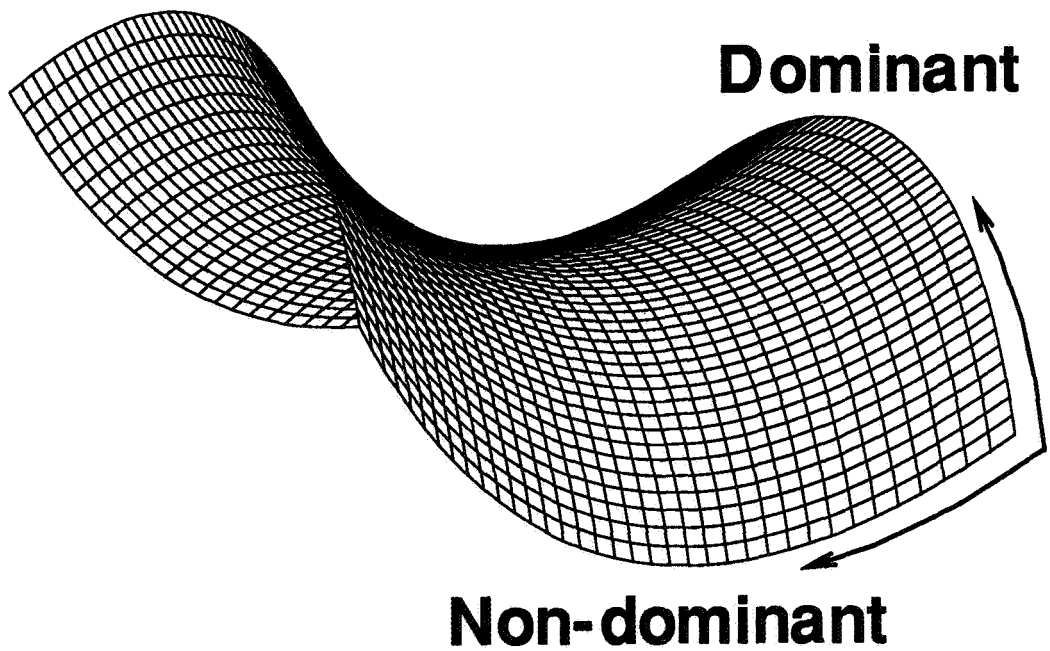


Figure 14: Hyperbolic blending function in parametric space for the dominant and non-dominant edge cell sizing functions.

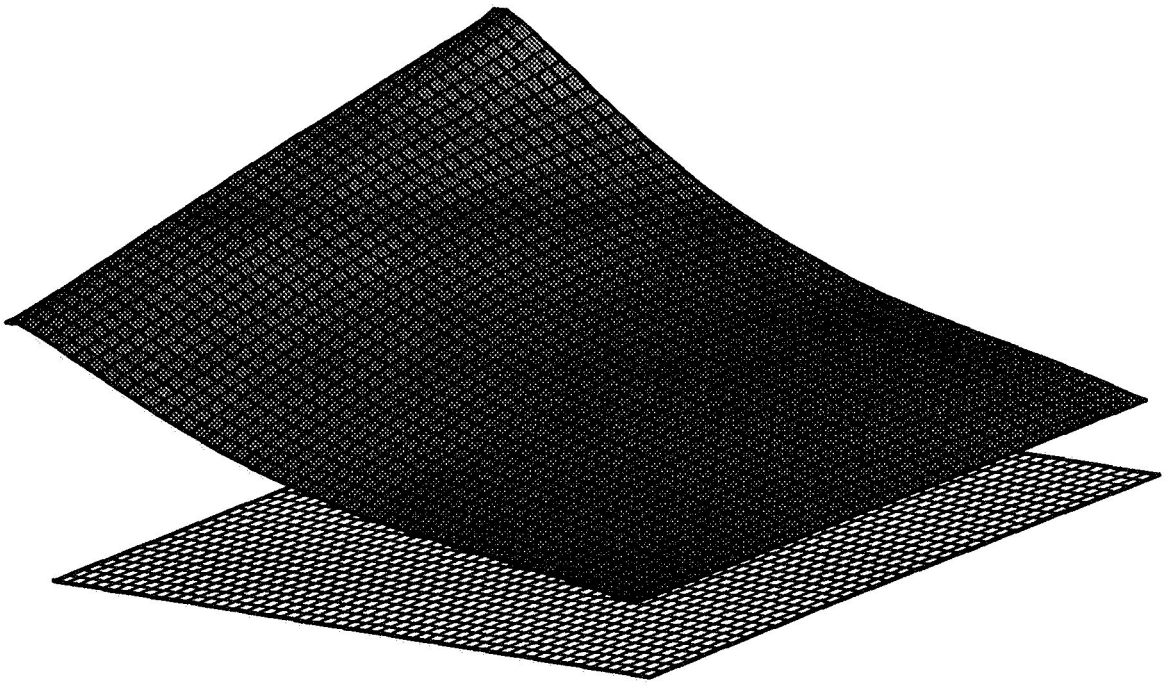


Figure 15: Fully blended cell sizing surface based on defining edges in parametric space.

Method Comparison

As stated in the introduction, the most common way of calculating the cell size to use for orthogonality specification is TFI. There are two different TFI's that can be utilized. The first is two dimensional in parametric space with $(\xi, \eta, \Delta s)$ being the three variables. Although this seems straight forward, the effects of the edge functions are not smeared and a smooth blending does not result (figure 16).

The second type of TFI that can be used is in three dimensional parametric space. For this type of interpolation, it was stated earlier that if TFI is used, due to its dependency on grid distributions, grid lines emanating from the surface may be skewed. Because of these grid lines being skewed, the cell sizes based on the standard arc-length calculation, equation 4, will be much larger than if the grid line is orthogonal to the surface (figure 17). In turn this produces a much larger cell size because the method is trying to stretch a grid line between two opposing faces with different distributions which arise from trying to model the flow field characteristics.

On the other hand, if the dependency on the grid distribution is removed and a smooth blending function is used between the defining cell sizes on the surface's edges, smaller initial grid spacings result (figures 16 and 17). In addition, smaller cell sizing gradients are produced across the surface domain. This occurs simply due to the fact that the orthogonality is assumed in the arc-length calculation. Therefore to obtain the best distribution of the cell sizes over the surface of the configuration, the LARCS method should be the method of choice. In doing so, the grid spacing near the surface will be smaller, lending to better and quicker resolution of the boundary layer in the evolution of the CFD solution.

As an example of the LARCS method, two different configurations are considered. The first configuration is the Space Shuttle (figure 18). By viewing the comparison of cell sizes in the contoured plots, the areas of highest cell sizes are the discontinuous regions. More specifically the wing root and fuselage juncture and the Orbital Manuevering System (OMS) pod. This is also evident when viewing the resolved grid using both methods (figure 19). For the TFI grid, at the first grid point from the wing root and fuselage juncture, the LARCS method has placed 3 more grid points in the same relative distance. These extra grid points placed by the LARCS method and the elliptic solution, shows just how much more of the boundary layer can be resolved.

Also consider the Langley lifting body, the HL-20 (figure 20). Here the cell sizes, shown in figure 21, are similar between the two different methods. This can be attributed to the fact that the surface has no discontinuous regions. Yet, when considered more closely, at those regions of high curvature gradients, such as the wing root and fuselage juncture, the surface cell sizes do differ slightly. The sensitivity to the surface curvature is another condition that is neglected by the LARCS method. Furthermore, due to the smooth blending of the LARCS method, the gradients found on the fuselage forward of the wing for the three dimensional TFI, are not apparent for the LARCS method of distribution. Thus, the cell sizes based on the LARCS method are more likely to result in better resolution of the boundary layer, and not cause instability in the CFD solver, due to the smooth transitions between all cell sizing edge functions.

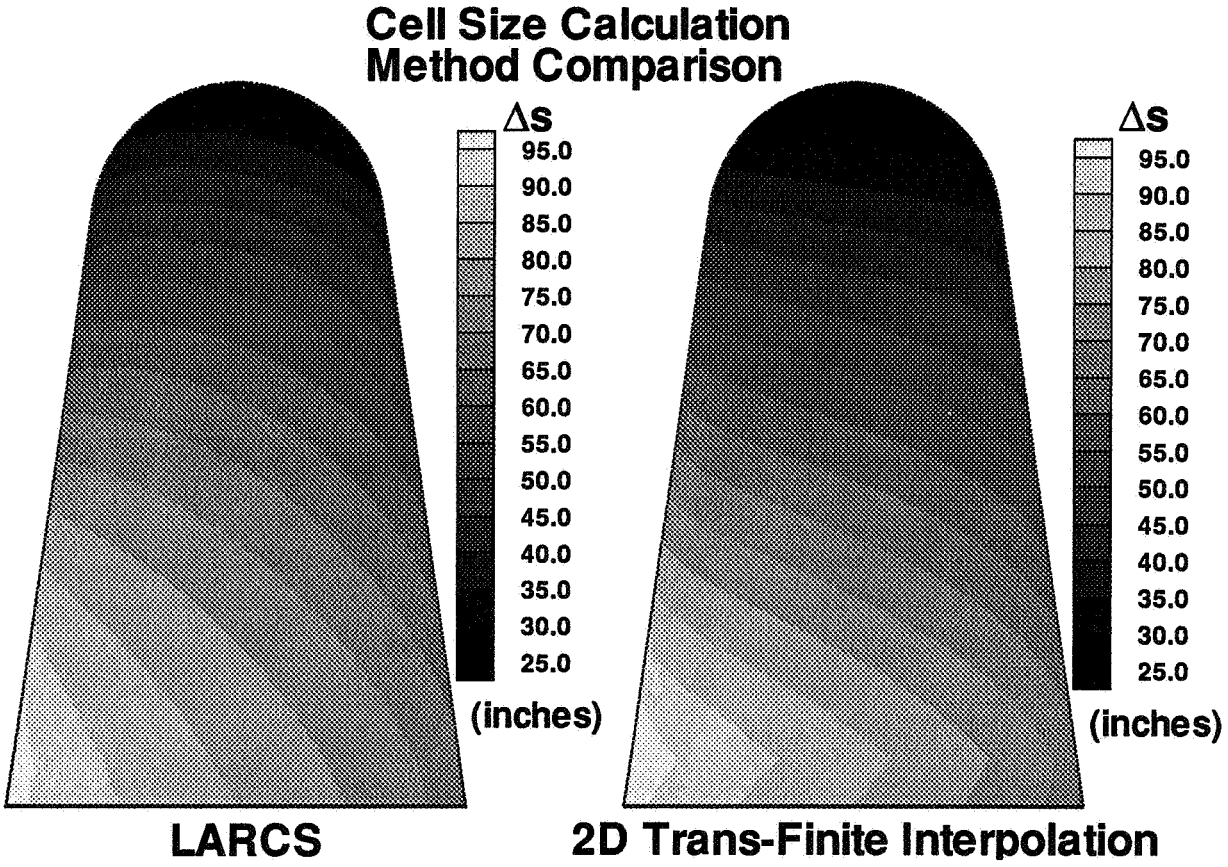


Figure 16: Cell sizing comparison of 2D TFI and the LARCS methods.

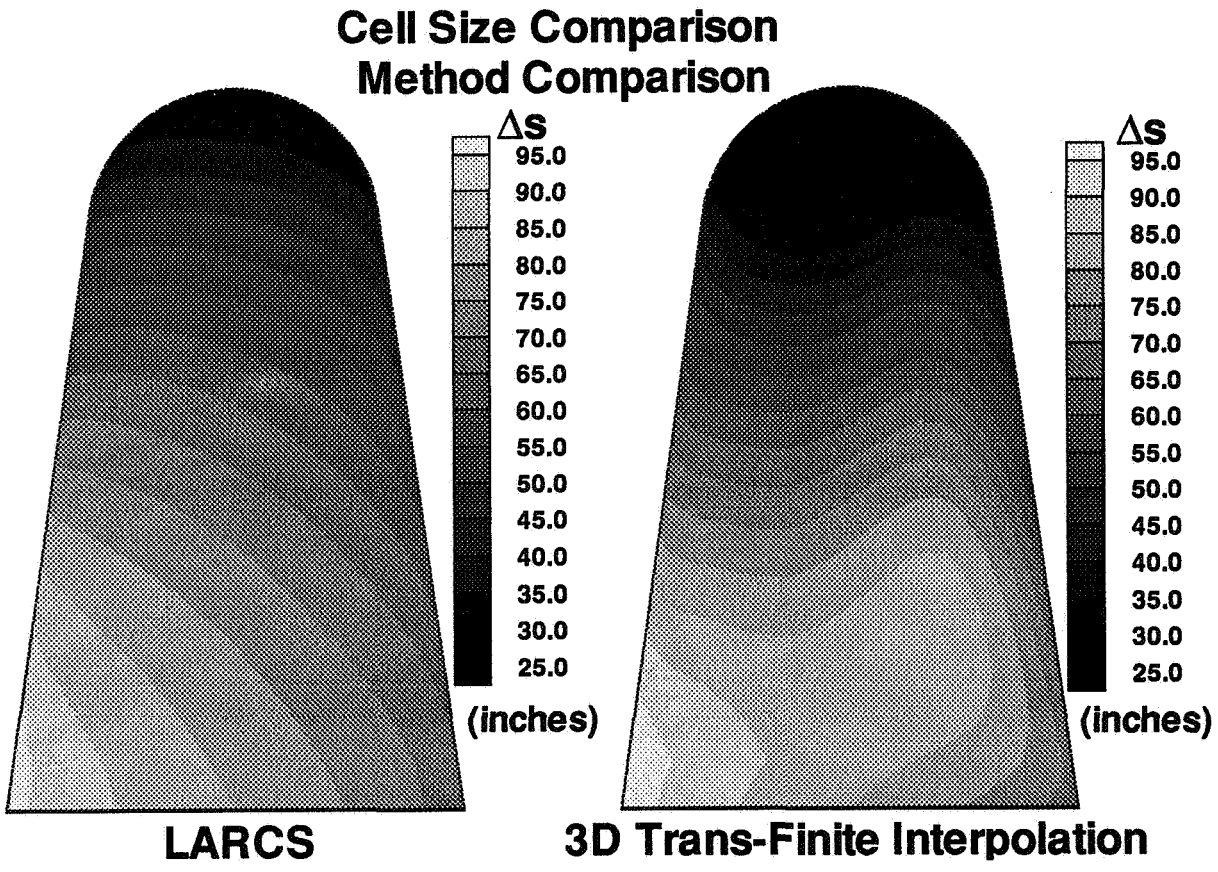


Figure 17: Cell sizing comparison of 3D TFI and the LARCS methods.

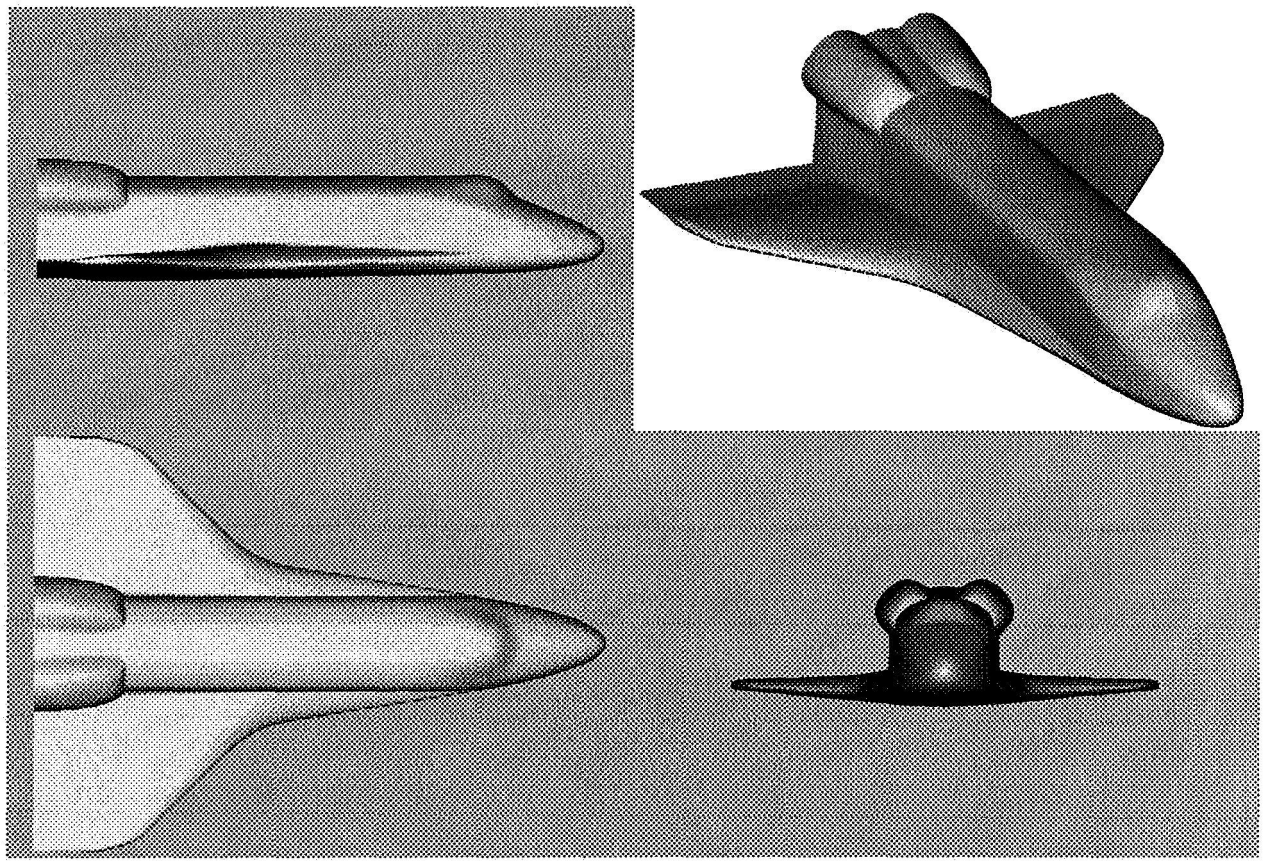


Figure 18: Space Shuttle surface geometry.

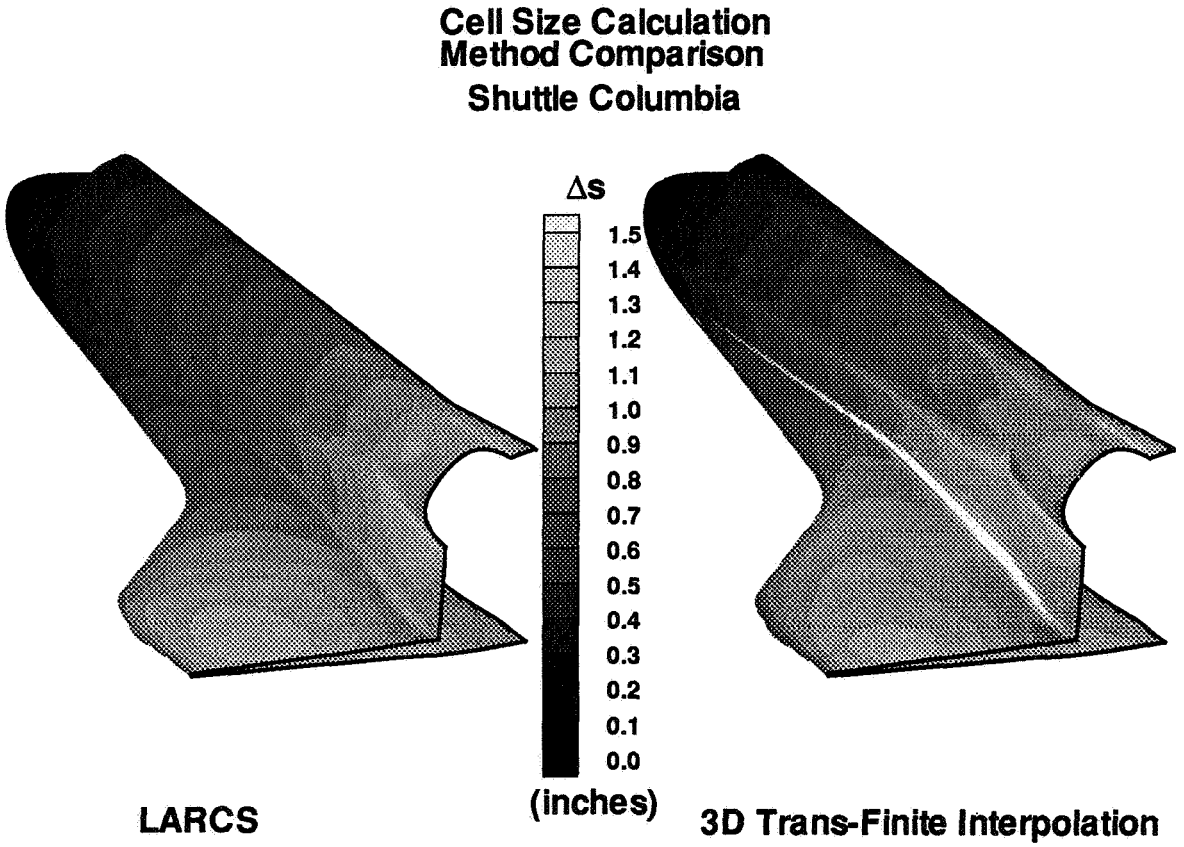


Figure 19: Cell sizing comparison of 3D TFI and LARCS for the Space Shuttle.

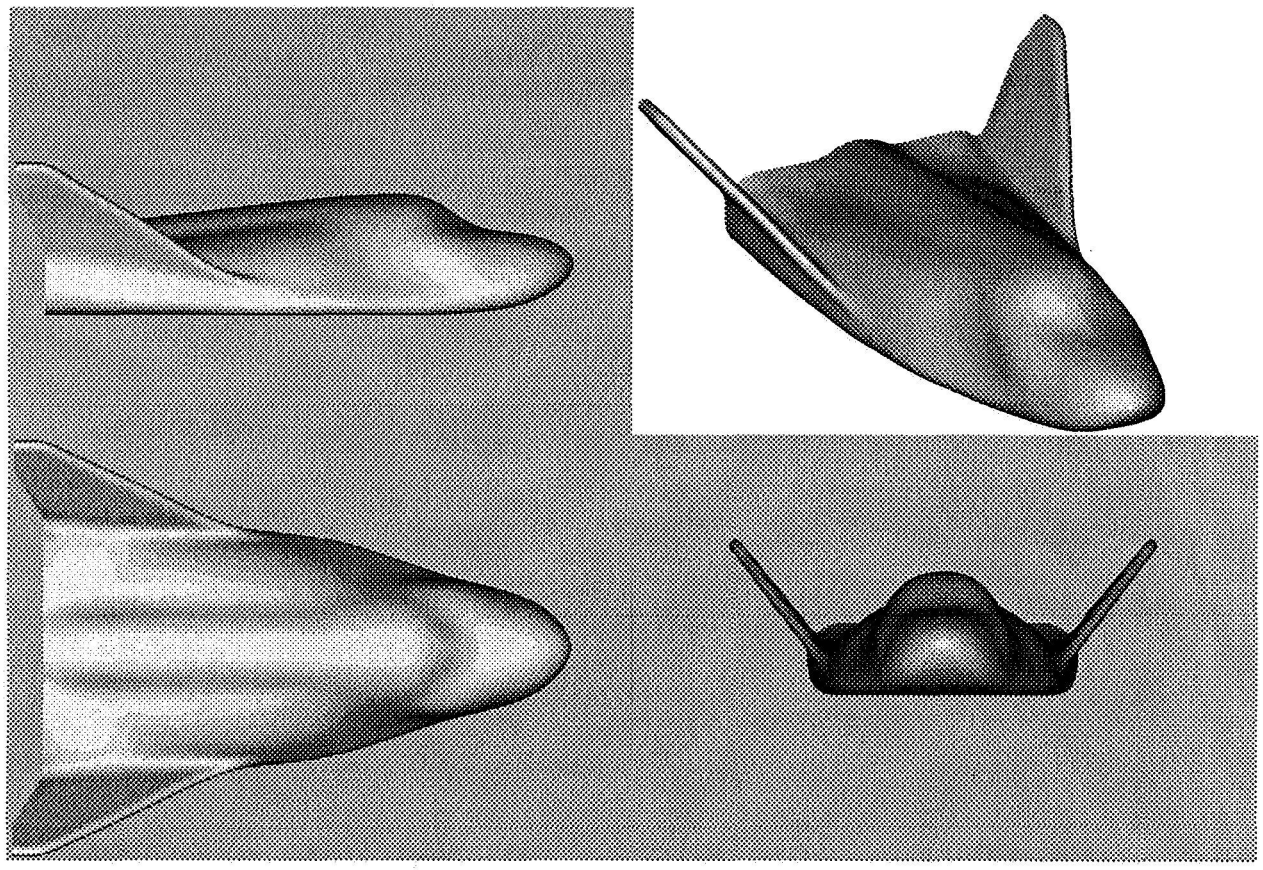


Figure 20: Langley Lifting Body (HL-20).

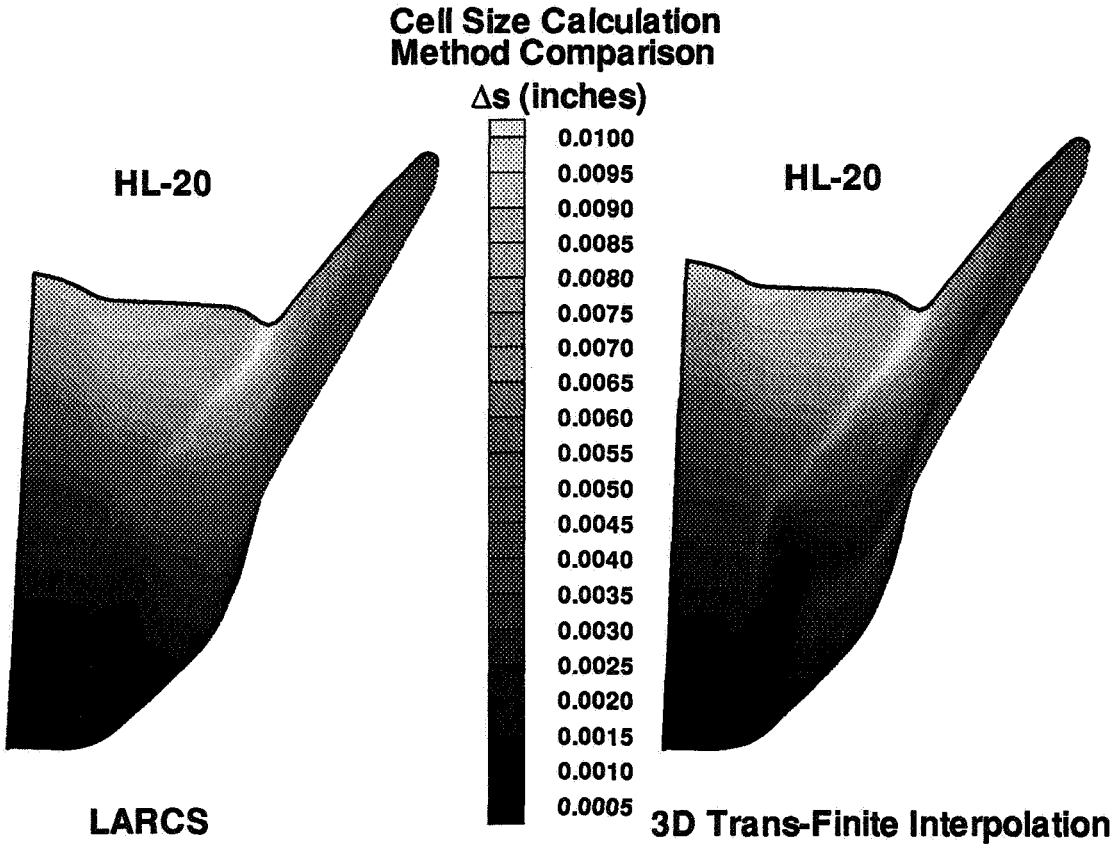


Figure 21: Cell sizing comparison of 3D TFI and LARCS for the HL-20.

Summary

An alternate method for calculating cell sizes for the explicit control of orthogonality in solving Poisson's three dimensional space equations has been presented. This method inherently builds a better link from the user's "best" guess for proper cell sizes to the specification of the cell sizes throughout the domain of the configuration's surface. The resulting cell sizes are much better than those generated by other methods, simply because LARCS eliminates any dependency on grid distributions, and is only concerned with the cell sizing distributions on the defining edges of the configuration's surface. In addition, the resulting cell sizes have smoother distributions on the surface of the configuration, which tends to reduce CFD solver inaccuracy, as opposed to large cell sizing gradients. Furthermore, by utilizing the LARCS method, the number of points near the surface will tend to be higher, and the flow solvers used will have a better chance of resolving the boundary layer. Finally, by having better resolution of the viscous forces before the first grid adaptation, the number of successive re-adaptations necessary to obtain a converged solution should be reduced. Thus, the user can obtain a quicker resolution of the boundary layer and the resulting converged solution, due to better initial grid spacing from the surface.

References

[1] Reding, P. J. , Kudlick, D. A., "Aerodynamic Design Methods for Advanced Maneuvering Reentry Bodies," AIAA Missile Sciences Conference, November 13-15, 1990, Monterey, Ca.

[2] Danberg, J. E., Sigal, A., Celmins, I., "Aerodynamic Characteristics of a Family of Cone-Cylinder-Flare Projectiles," AIAA paper 89-3371, August 1989.

[3] Cruz, C. I., Engelund, W. C., "Enhancements to the High Speed Convective Heating and Viscous Drag Prediction Techniques of the Aerodynamic Preliminary Analysis System (APAS)," AIAA paper 91-1435, June 24-26, 1991, Honolulu, Hawaii.

[4] Steger, J. L., "Application of Mesh Generation to Complex 3-D Configurations" AGARD Advisory Report AR-268, March 1991.

[5] Blottner, F. G., "Accurate Navier-Stokes Results for the Hypersonic Flow over a Spherical Nosetip," AIAA paper 89-1671, June 1989.

[6] Edwards, T. A., Flores, J., " Computational Fluid Dynamics Nose-to-Tail Capability: Hypersonic Unsteady Navier-Stokes Code Validation," AIAA paper 89-1672, June 1989.

[7] Bhutta, B. A., Lewis, C. H., "Comparison of Hypersonic Experiments and PNS Predictions Parts I & II." *Journal of Spacecraft and Rockets*, Vol. 28, July-August 1991, pp. 376-393

[8] Williams, R. M., "National Aerospace Plane: Technology for America's Future," *Aerospace America*, Vol. 24, No. 11, 1986, pp. 18-22

[9] Olson, L. E., "Applied Aerodynamics," *Aerospace America*, Vol. 29, No. 12, pp. 60

[10] Davies, C. and Venkatapathy, E., "A Simplified Self-Adaptive Grid Method, SAGE," NASA TM-102198

[11] Gnoffo, P. A., "An Upwind-Biased, Point-Implicit Relaxation Algorithm for Viscous, Compressible Perfect-Gas Flows," NASA TP-2953

[12] Anderson, J. D. Jr., "Modern Compressible Flow with Historical Perspective," McGraw-Hill, 1982, pp. 84-113, 248-259

[13] Gupta, R. N., Lee, K., Thompson, R. A., Yos, J. M.,"Calculations and Curve Fits of Thermodynamic and Transport Properties for Equilibrium Air to 30,000 K," NASA Reference Publication 1260, October 1991, pp. 14

[14] Gupta, R. N., Lee, K., Thompson, R. A., Yos, J. M.,"Calculations and Curve Fits of Thermodynamic and Transport Properties for Equilibrium Air to 30,000 K," NASA Reference Publication 1260, October 1991, pp. 59

[15] Gnoffo, P. A., "An Upwind-Biased, Point-Implicit Relaxation Algorithm for Viscous, Compressible Perfect-Gas Flows," NASA TP-2953 pp. 15

[16] Anderson, D. A., Tannehill, J. C., Pletcher, R. H.,"Computational Fluid Mechanics and Heat Transfer," McGraw-Hill, 1984, pp. 343-346

[17] Steinbrenner, J. P., Chawner, J. R., Fouts, C. L., "The GRIDGEN 3D Multiple Block Grid Generation System," Wright Research & Development Center, WRDC-TR-90-3022, October 1989.

[18] Product Specification, PATRAN Plus, PDA Engineering, 1991

[19] Sorenson, R. L., "The 3DGRAPE Book: Theory, User's Manual, Examples," NASA TM-102224, pp. 75-79

SINGULARITY CLASSIFICATION AS A DESIGN TOOL FOR MULTIBLOCK GRIDS

Alan K. Jones
Boeing Computer Services
Bellevue, WA

INTRODUCTION

A major stumbling block in interactive design of three-dimensional multiblock grids is the difficulty of visualizing the design as a whole. One way to make this visualization task easier is to focus, at least in early design stages, on an aspect of the grid which is inherently easy to present graphically, and to conceptualize mentally, namely the nature and location of singularities in the grid. The topological behavior of a multiblock grid design is determined by what happens at its edges and vertices. Only a few of these will be in any way exceptional. These exceptional behaviors lie along a singularity graph, which is a one-dimensional construct embedded in three-dimensional space. The varieties of singular behavior are limited enough to make useful symbology on a graphics device possible. Furthermore, some forms of block design manipulation that appear appropriate to the early conceptual-modelling phase can be accomplished on this level of abstraction.

The present work is an overview of a proposed singularity classification scheme and selected examples of corresponding manipulation techniques, drawn mostly from (ref. 1). Mathematical details can be found in (ref. 2) or (ref. 3). It will not treat practical issues of how a computer interface based on such principles would look. Nor will it consider in detail the correspondence between singularity graphs and the resulting block designs.

SINGULARITY CLASSIFICATION AS A DESIGN TOOL FOR MULTIBLOCK GRIDS

For the purposes of this discussion, a multiblock grid is a decomposition of a region in space into blocks, each of which is the image of a cube in parameter space under a smooth, invertible coordinate map. Neighboring cubes intersect only in complete faces, edges or vertices. This last condition is by no means universal in multiblock applications, but it will simplify the analysis greatly.

For numerical applications, decompositions should ideally be conformal in the sense that the blocks meeting at each vertex can be assembled into a consistent local coordinate system. This is analogous to the condition in two dimensional gridding that four blocks, each subtending a right angle in parameter space, should meet to fill out the 360° angle at each vertex. Special numerical techniques are generally required at points which are not conformal in this sense. See, for example, Section IV.3 of (ref. 4). In three dimensions, a conformal vertex is one where eight blocks meet, in an arrangement equivalent to the standard configuration of one block per octant. Similarly, four blocks should meet along each interior edge. Any failure of conformality is called a combinatorial singularity. In many cases, it is possible to remove combinatorial singularities by relaxing the conditions on the coordinate maps. For example, an entire face of a block in computational space may be collapsed to a point or a line in real space, or two adjacent faces may be unfolded to become coplanar. Such analytical singularites are really just an alternative realization of the same underlying phenomenon. However we choose to represent them, singularities are an unavoidable fact of life. Many interesting domains, in both two and three dimensions, simply do not admit a completely nonsingular blocking. A discussion of commonly used three-dimensional grid designs and their singularities, and the computational consequences of these singularities, can be found in (ref. 5).

An interior edge is singular if other than four blocks meet along it. Since there is in effect only one degree of freedom in the location of the blocks, it is enough to count them.The singularity graph is is just the union of all the singular edges. With appropriate bookkeeping conventions, it is always possible to represent the singularity graph as the union of a set of pentagonal curves (strings of edges along which five blocks meet) and a set of triangular curves (strings of edges along which three blocks meet). These curves do not terminate in the interior of a domain; they must either be closed cycles or open curves, with endpoints on the domain boundary. However, the curves need not be disjoint. For example, two pentagonal curves may coincide to give a string of edges along which six blocks meet. Where singularity curves cross, necessarily at vertices, we get complicated behavior. Thus, a simple but useful display of the topological nature of a block design would be to draw the singularity graph, with each edge color coded according to its type, and intersections clearly noted.

We can make the display more useful by encoding information about what happens at singularity curve intersections. Behavior at an interior vertex can be visualized by drawing a certain convex polyhedron, which is in effect just a picture of a small neighborhood of the vertex. To see this, fix a vertex v, and consider all the blocks that meet at v. Together, these form a neighborhood of v which we call U, topologically equivalent to a three-dimensional ball. Unfortunately, to understand the blocking of this neighborhood, it is not enough just to count the blocks. We need to draw its face adjacency graph, denoted $G(v)$. The nodes of G are the blocks that meet v. An edge of G connects two nodes if and only if their corresponding blocks share a face. The graph G has a very concrete geometrical meaning. It is the one-dimensional skeleton of the bounding surface of U. For example,

the graph at a non-singular vertex is just the graph of edges of the 3-cube, and the neighborhood U does in fact look just like a 3-cube.

Figure 1 shows the six most common polyhedra, corresponding to singular internal vertices adjacent to at most 12 blocks. Two faces of the N=12 case are shaded to aid in visualization. Not shown is the only non-singular case, the 8-block polyhedron which is an ordinary cube. (It is an interesting fact that N, the number of blocks, must be even.) From now on, we adopt the convention that curves are labelled with their total net singularity, i.e., b-4, where b is the number of blocks meeting along the curve. Curves with negative net singularity will be shown in grey, while curves with positive net singularity are shown in black.

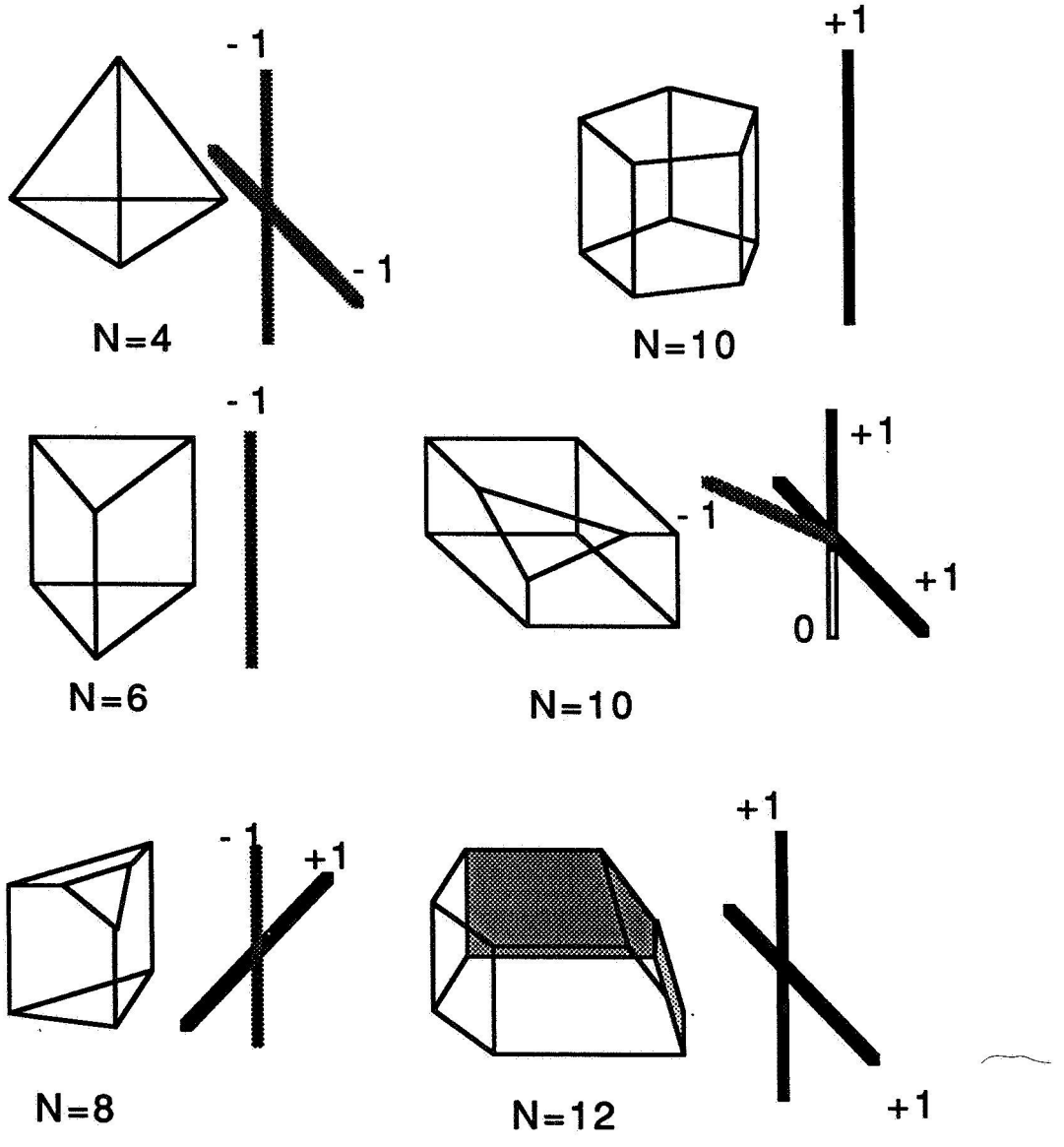


Figure 1. Singular polyhedra corresponding to common crossing patterns.

We will discuss two types of grid manipulations that can be done at the level of singularity graphs. The first type allows us to avoid the more exotic behaviors cataloged in figure 1. (The second type has to do with behavior on the boundary, and we will treat that later.) For example, figure 2 shows how a blocking scheme can be modified to break the connection between two intersecting curves of triangular singularities, by introducing new blocks so that a polyhedral neighborhood that began as a tetrahedron becomes instead a union of two triangular prisms. Figure 3 is the analogous construction for two pentagonal singularity curves.

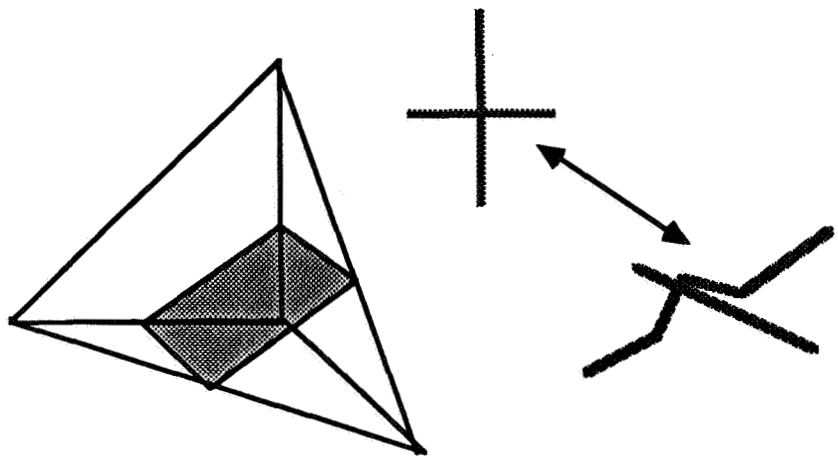


Figure 2. Removing an intersection between triangular curves.

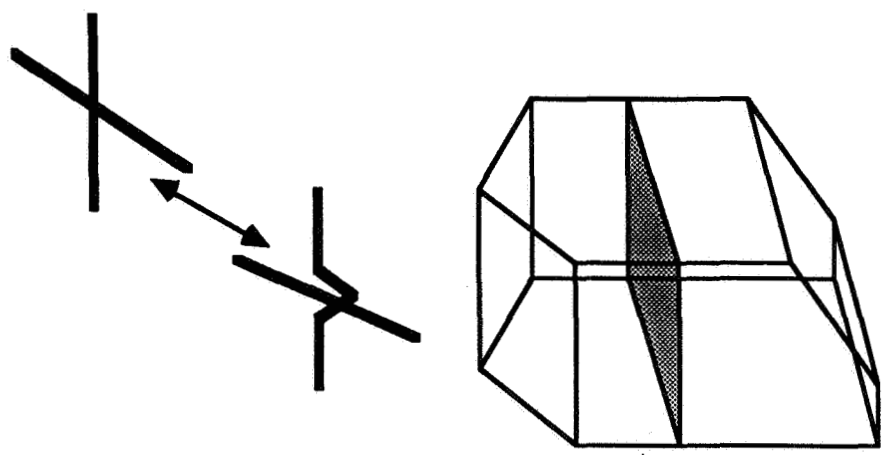


Figure 3. Removing the intersection of two pentagonal curves.

Similarly, figure 4 shows how to remove the intersection of a pentagonal and a triangular curve. The situation in figure 5 is more delicate. The most useful interpretation of this diagram is that one triangular and one pentagonal curve can coalesce when they cross a second pentagonal curve and leave the intersection as a non-singular curve, i.e., one along which four blocks meet. This construction will turn out to be crucial in the manipulation of boundary creases and corners.

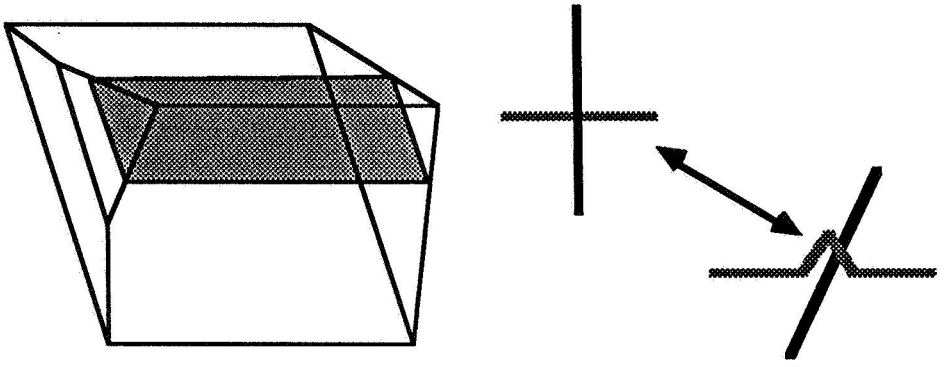


Figure 4. Removing an intersection between curves of mixed type.

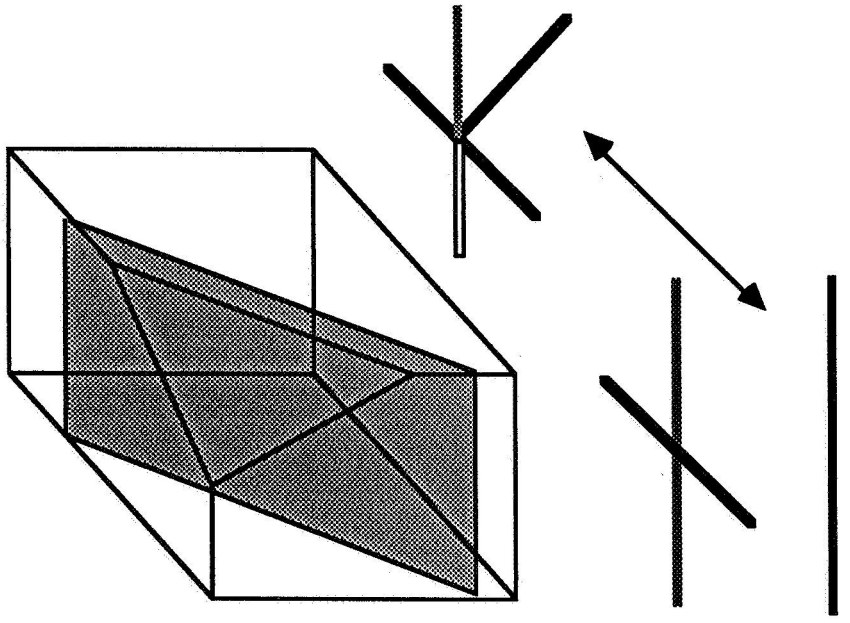


Figure 5. Coalescing curves of opposite type at a pentagonal curve.

Now we wish to put these constructions to use in constructing block designs. Consider first the simplest case, a computational domain which is a solid ball with smooth boundary. Each octant of the ball can be parametrized smoothly by a tetrahedron, which has the singularity graph shown in figure 6. Pasting eight of these graphs together gives the complete singularity graph of the solid ball, as shown in figure 7.

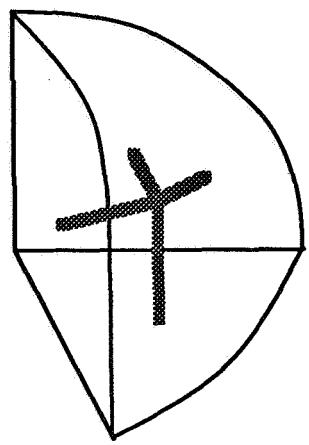


Figure 6. The singularity graph of one octant of a ball.

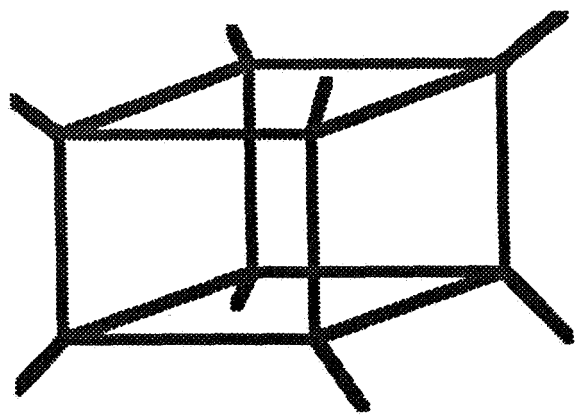


Figure 7. The complete singularity graph of the solid ball.

Figure 8 shows four alternative ways of viewing the singularity graph of a ball. In the upper left hand corner, the graph from figure 7 is reproduced. This graph is connected but is not a manifold. In the upper right hand corner, the four vertical curves have been pulled off the graph, leaving two closed cycles and four open curves. In the lower left hand corner, four different open curves have been pulled off, leaving only one cycle. Finally, on the lower right, we have four open curves and no cycles at all. Evidently, simple counting is not enough to characterize this topology. The only obvious invariant is the total number of open singularity curve ends that terminate on the boundary of the region. For a smooth manifold, this is always -4χ, where χ is the Euler characteristic. Thus, for a smooth spherical boundary, we must have a net count of -8 singularity curve ends. See (ref. 3) for details.

The first two alternatives are probably the most useful in practice because they reflect the symmetries of real computational domains. The connected arrangement of figure 7 will be referred to as the local version because in it the singularites can be kept arbitrarily close to the boundary. The other arrangements are referred to as global versions, because the open singularity curves are typically threaded through the whole length of the domain.

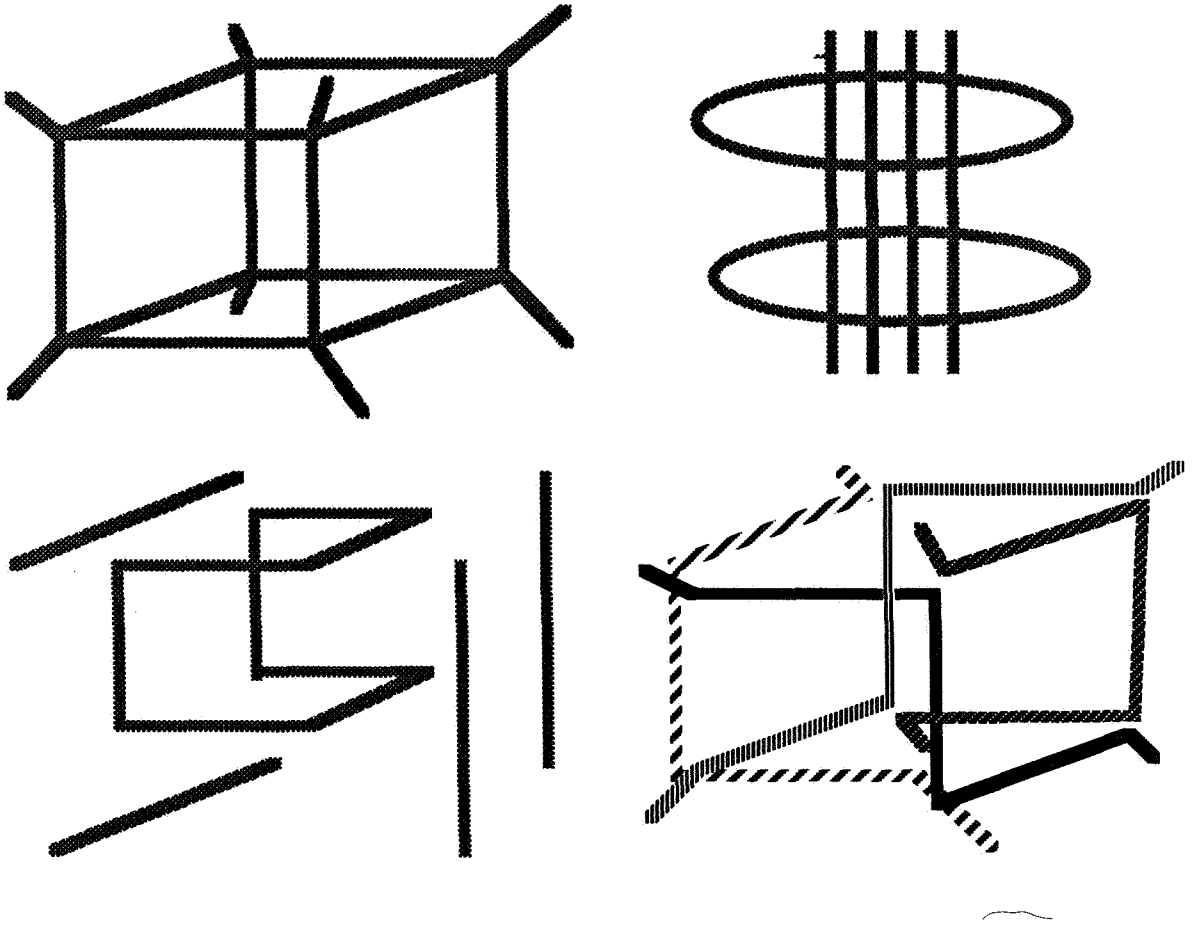


Figure 8. Separating a singularity graph into disjoint curves.

Next, we consider the dual case of a spherical void inside a cubical domain. One octant of this domain looks like the 10-sided polyhedron shown in figure 5. Pasting eight of these together gives figure 9, which is the standard local scheme for a smooth, simply connected void. Using the subdivision of figure 5 allows us to separate the four vertical pentagonal curves from the singularity graph and arrive at the global scheme of figure 10. In this case, we have a total of four pentagonal curves connecting the outer boundary to itself, and a total of eight triangular curves connecting the outer boundary to the inner boundary. This is consistent with the total singularity count of -8 for the smooth spherical inner boundary, and 0 for the outer boundary, which is assumed to be a non-smooth rectangular prism. See (ref. 1) or (ref. 3) for details.

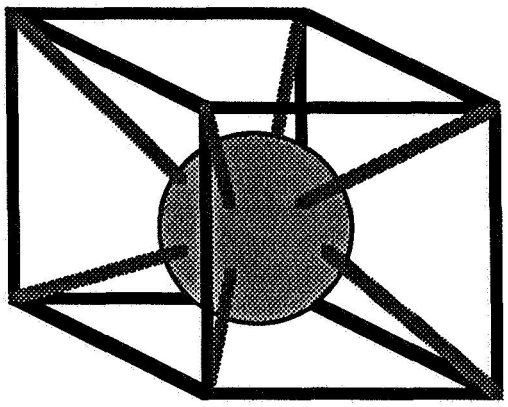


Figure 9. The complete local form for the complement of a ball.

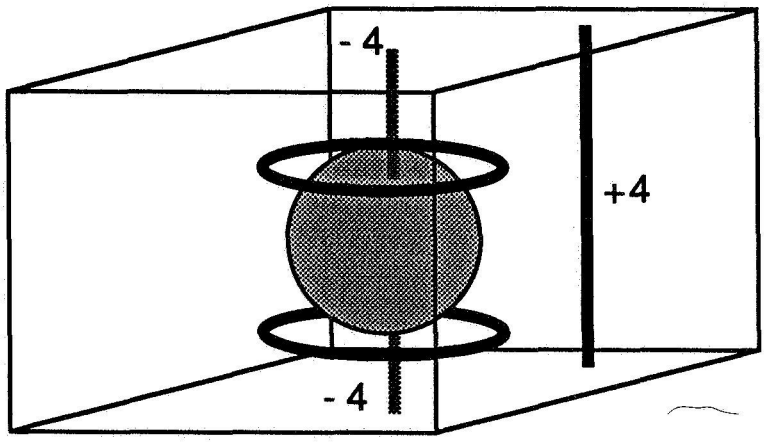


Figure 10. A global form for the complement of a ball.

The shapes of the singularity graphs in figures 7 and 9 are suggestive. In each case, what we see is the edge graph of the 3-cube, connected to the smooth boundary of the region by triangular edge "stingers" at all eight corners. The only difference is that, for the smooth outer boundary of figure 7 the 3-cube graph consists of triangular curves, while for the smooth inner boundary of figure 9 it consists of pentagonal curves. This is not accidental. In each case, we may start with a boundary which is just a rectangular prism, and smooth away its corners and creases. (A crease is a boundary edge along which the coordinate species of the outward pointing normal changes. A corner is where three or more creases meet.) This is the second major class of singularity manipulation techniques. The process is shown in figure 11. The operation of removing a crease consists of adding an extra block along each boundary face (the shaded faces in figure 11). Blocks corresponding to edge-adjacent boundary faces are made face-adjacent. (In figure 11, these faces are crosshatched.) Thus, a convex crease, such as we find on an external boundary, corresponds to a triangular singularity curve, and a concave crease, such as we find on an internal boundary, to a pentagonal curve.

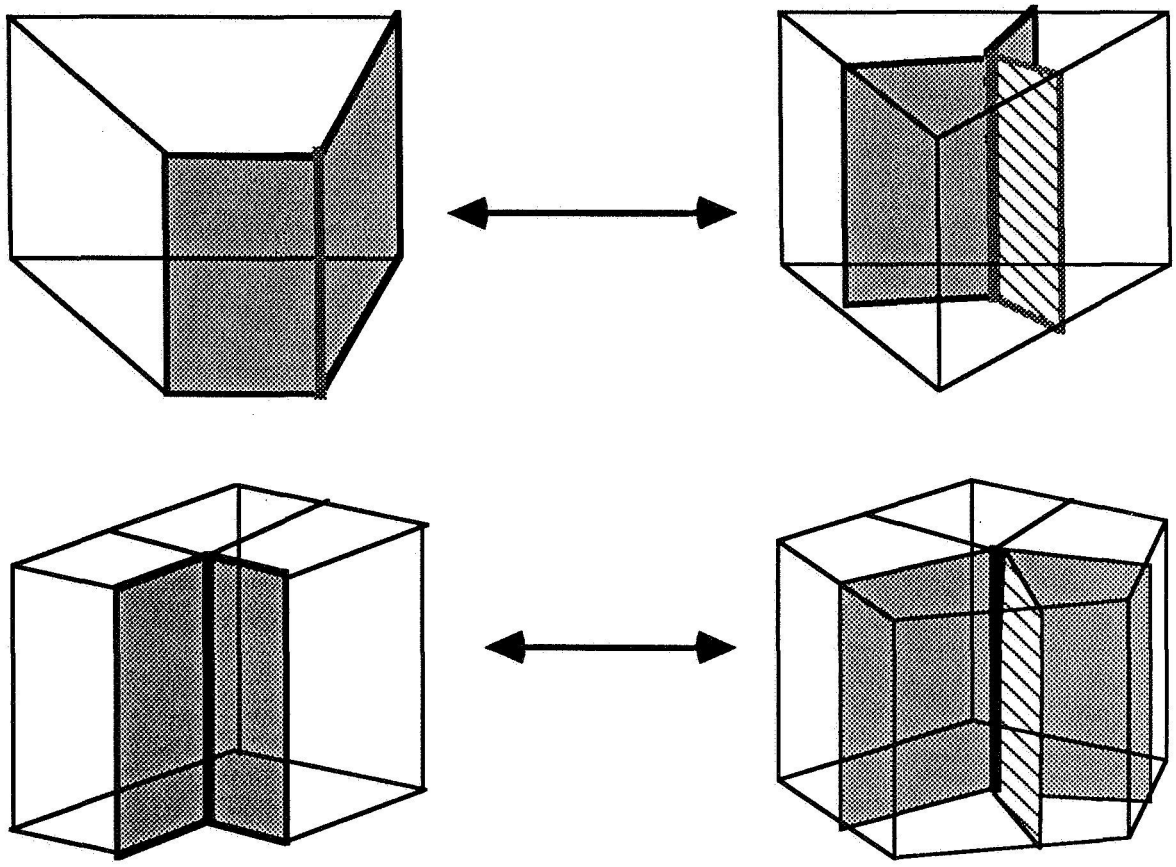


Figure 11. Equivalence of boundary creases and internal singularity curves.

The situation at corners is somewhat more delicate, as shown in figure 12. In the left hand column are typical arrangements of blocks on the boundary of a domain, with boundary faces shaded, and the corner of interest marked by a black dot. In the right hand column are the corresponding polyhedra. The new blocks added along shaded faces correspond to the vertices marked with open squares in the polyhedra; the existing blocks correspond to vertices marked with shaded circles. We see that a convex corner, like that on an outer boundary, corresponds to a tetrahedral singularity, and a concave corner, like that on an inner boundary, to the exceptional N=10 case shown in figure 5. In either case, the internal singularity is connected to the new boundary by a triangular singularity curve because the polygonal face formed by the new boundary blocks is triangular. The third row in figure 12 shows the behavior at a saddle corner, which we would find, for example, at a wing-body join. The internal singularity is the same as for a concave corner, but it is now joined to the boundary by a pentagonal curve, rather than a triangular one.

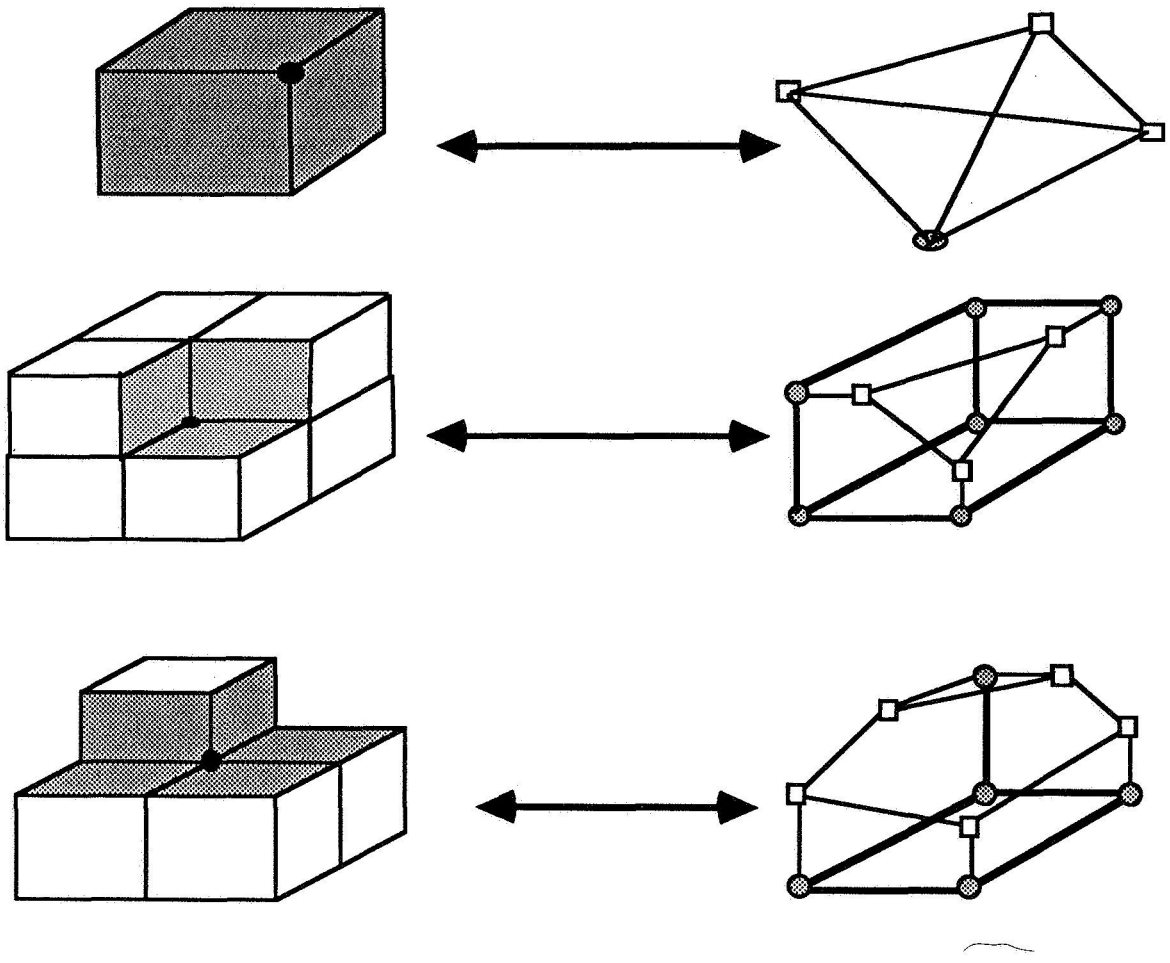


Figure 12. Equivalence of boundary corners and internal singularity vertices.

As a practical illustration of this smoothing procedure, consider blocking in the neighborhood of a rounded wingtip, as shown in figure 13. Here, for the sake of visual clarity, we have changed conventions. It is the four unshaded faces of the square wingtip which are smoothed. The resulting pair of triangular curves that terminate on the wingtip surface corresponds to a parabolic singularity in the terminology of (ref. 5).

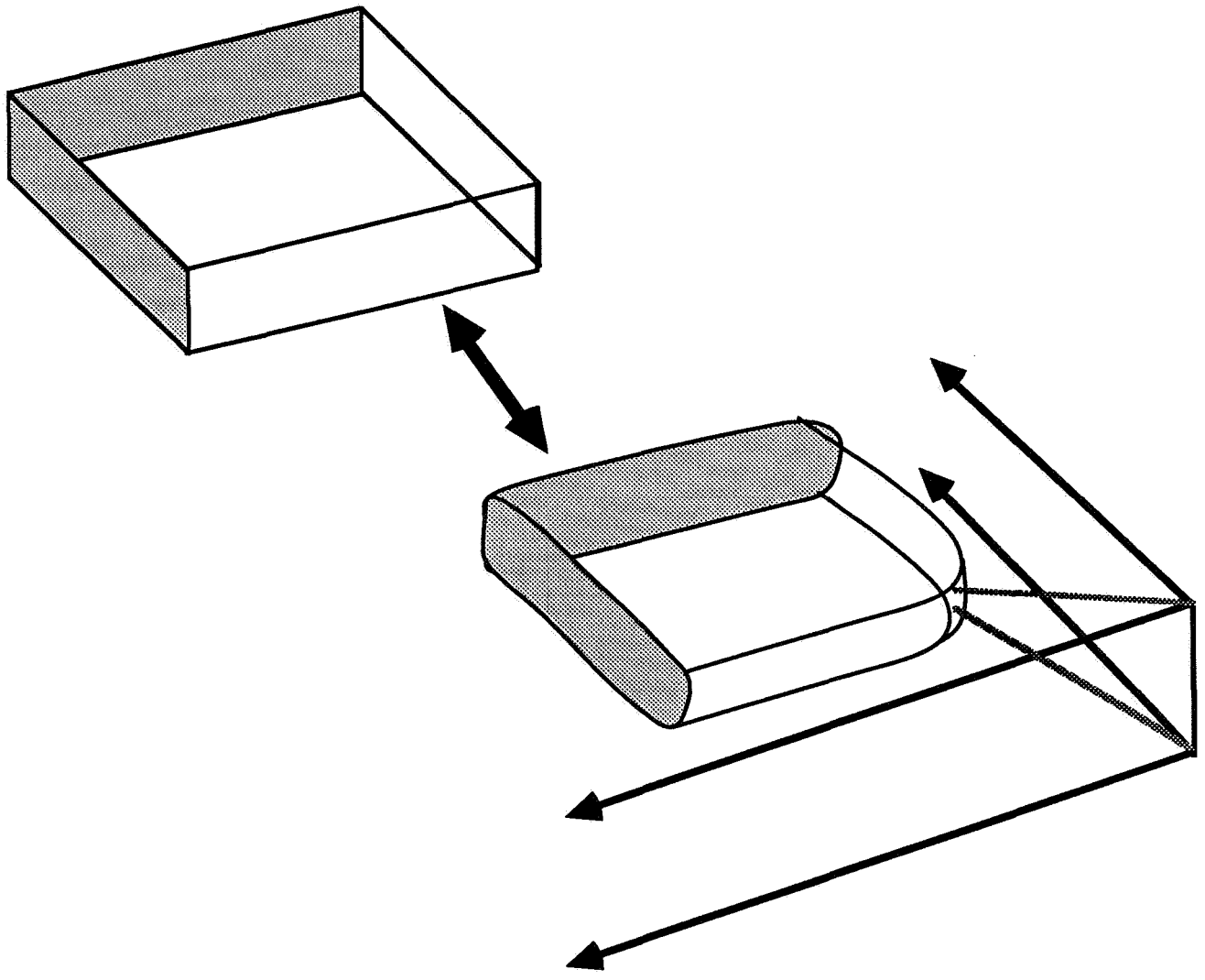


Figure 13. Blocking near a rounded wingtip.

For a second, example, consider how the singularity graphs behave under a basic operation of constructive solid modeling. View a landing gear assembly as the union of two cylinders, as shown in figure 14. We want a block design for the complement of the union of these cylinders. The first step is to find a blocking for the complement of one cylinder. We start with the block design of figure 15 for one octant of this complement, and, as usual, paste eight copies together to get the singularity graph of figure 16. For clarity, note that the individual singularity curves are drawn as though collapsed into groups of four each, although this would probably not be done in practice. For concreteness, figure 16 also shows in top view the block boundaries that would be seen with this design.

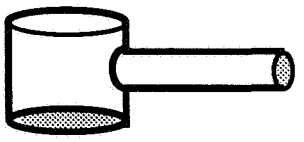


Figure 14. Landing gear modelled as the union of two cylinders.

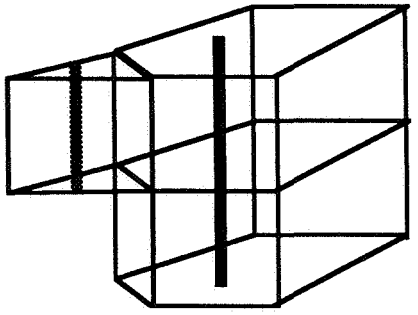


Figure 15. Blocking around one octant of a finite cylinder.

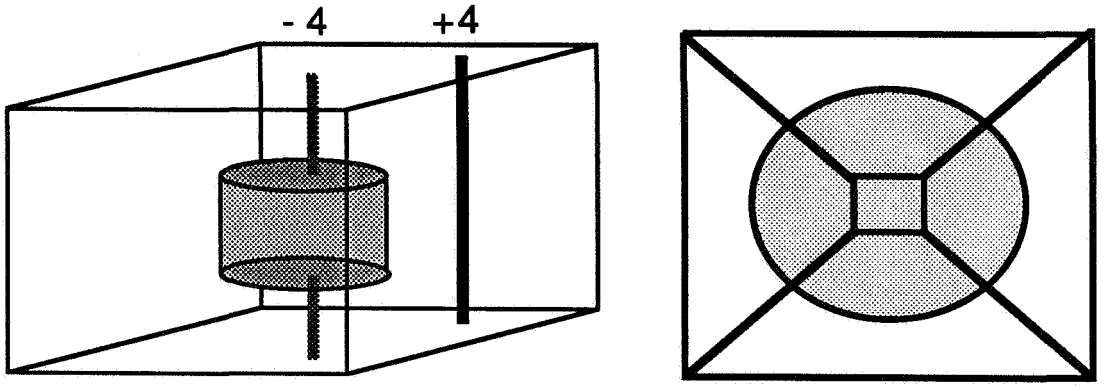


Figure 16. Singularity graph and top view of blocking around a finite cylinder,

If the two cylinders were disjoint, then the singularity graph we seek would be two disjoint copies of figure 16. However, they do intersect, and in a non-symmetric way, with one cylinder pierced by the other. The effect of this is that one set of pentagonal singularity curves now terminates on the surface of the pierced cylinder, where two-dimensional pentagonal singular vertices are needed to block around the circular hole cut by the piercing cylinder. See (ref. 3) for details. This is shown in figure 17.

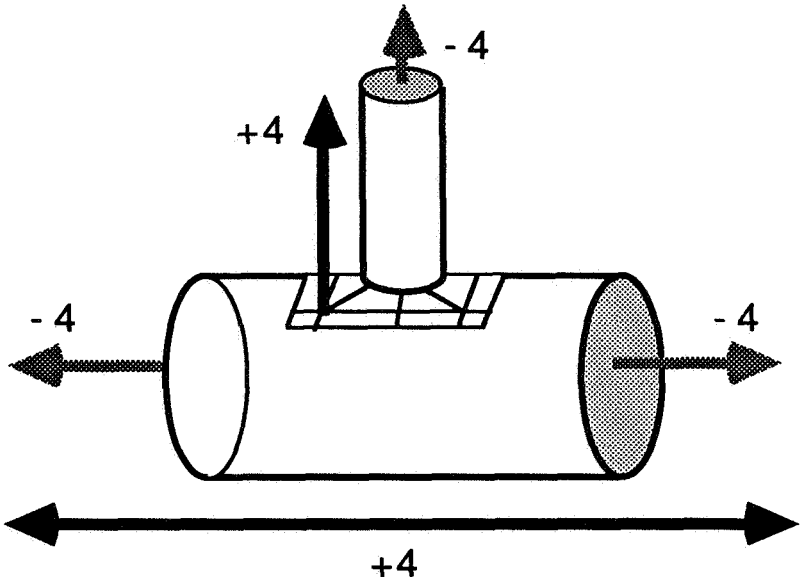


Figure 17. Singularity graph around the union of two cylinders.

When intuition fails in cases like this, the best algorithmic approach is to replace all solids by rectilinear forms, as in figure 18, construct a blocking around them, which can always be made non-singular, and then pull creases and corners off the boundary until the smooth case is recovered. The pentagonal curves terminating on the landing gear surface are then seen to be consequences of the saddle corners contained in the curve of intersection between the two rectangular prisms, marked with black dots in figure 18. Recall the third row of figure 12.

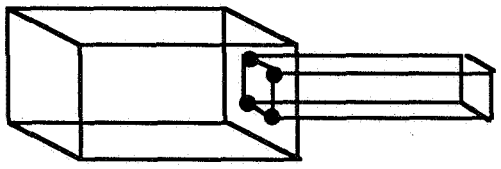


Figure 18. The rectilinear analog of figure 14, showing four saddle vertices.

Finally, consider a complete half-body-wing-pylon-nacelle assembly. A schematic section view is shown in figure 19, along with a typical two-dimensional block arrangement. The large black dots mark two-dimensional vertex singularities, vertices where other than four blocks meet. There are six pentagonal vertices in this scheme, and indeed it can be shown (ref. 2) that any two-dimensional blocking of this domain must have a net singularity count of +6 . Sweeping this two-dimensional design along the aircraft's longitudinal axis gives the set of three-dimensional singularity curves shown in figure 15. Note that, as in figure 16, if the cylindrical body and nacelle do not extend to infinity, triangular singularity curves arise as shown, connecting the ends to the boundary of the computational domain. At this stage, both the wing and the pylon have been left as rectilinear slabs.

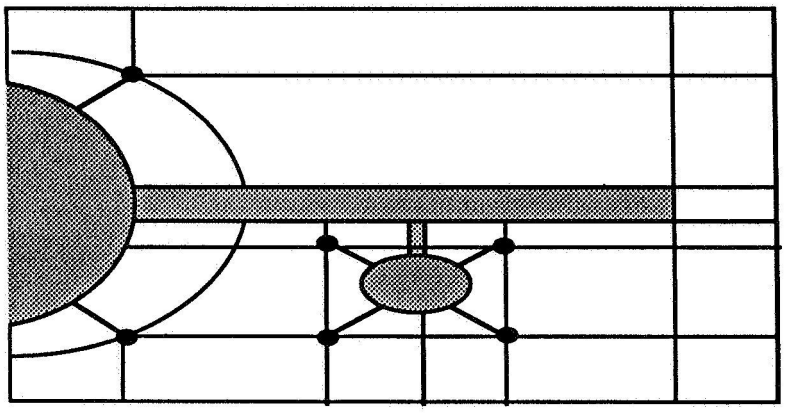


Figure 19. Two dimensional block design for half-body-wing-pylon-nacelle.

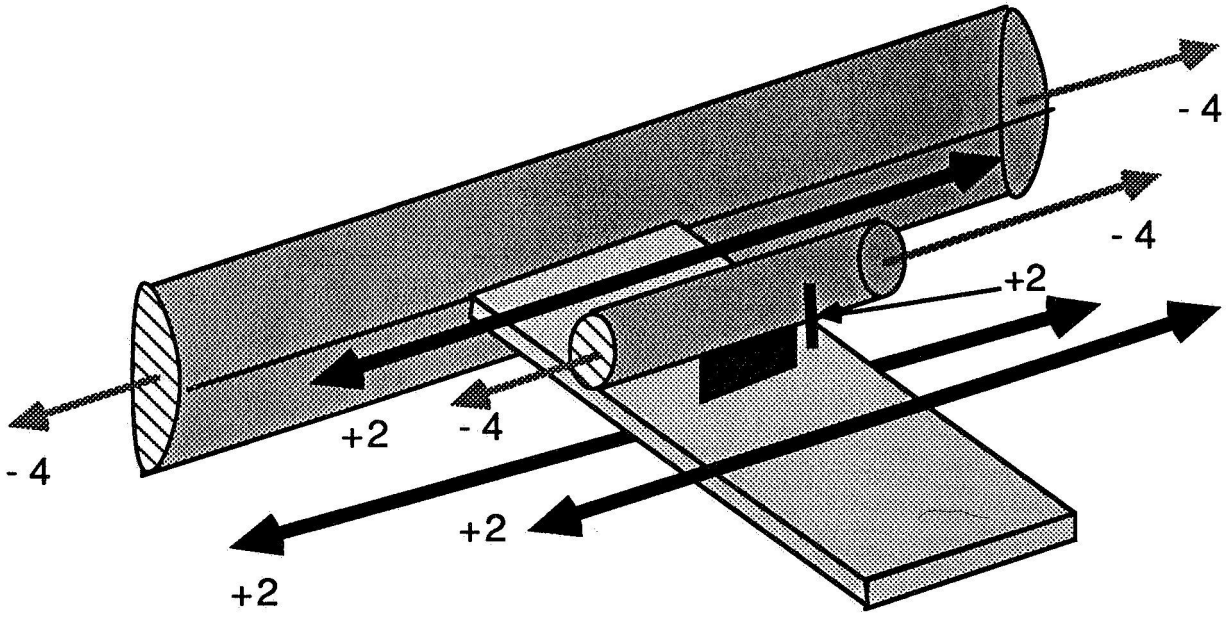


Figure 20 Three-dimensional singularity graph from sweeping figure 14.

Now we selectively pull creases and corners off of the airplane surface into the interior of the domain. If the nose and tail of the body and the leading edge of the wing and pylon are rounded, the result is the graph shown in figure 21. Additional modifications could be performed. Consider, for example, rounding the wingtip as in figure 13, or adding a flow-through nacelle.

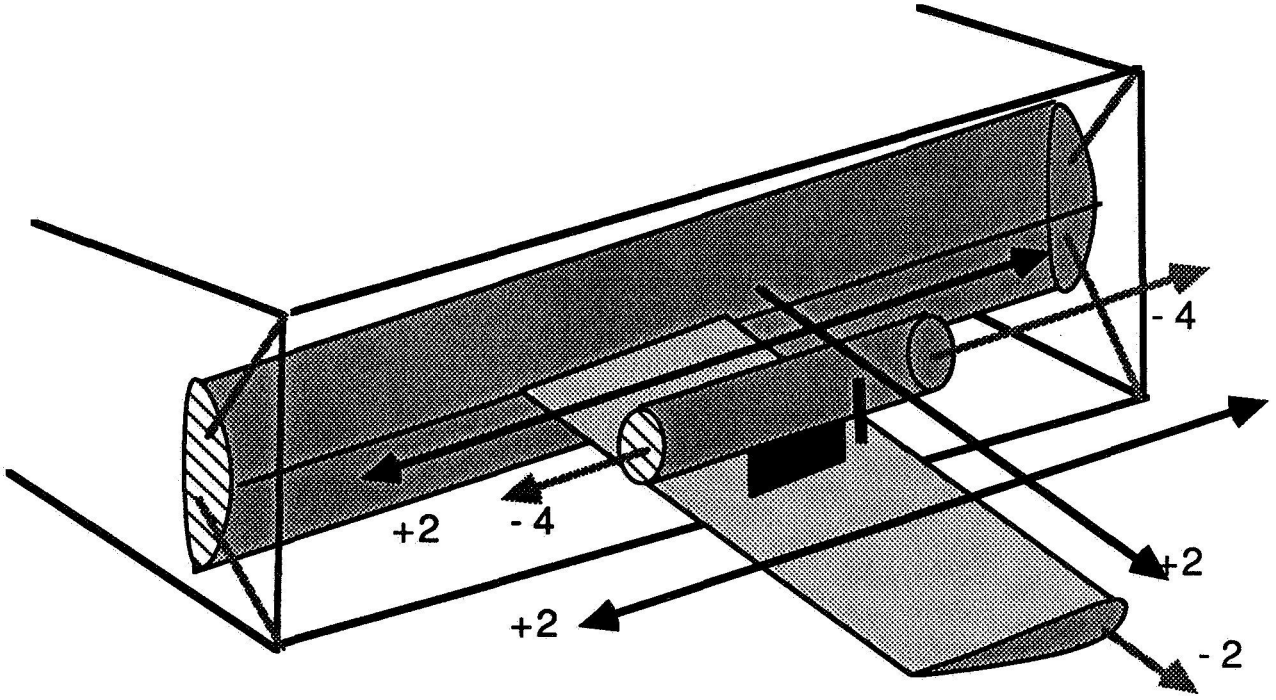


Figure 21. Selective rounding of inner boundary edges.

REFERENCES

1. Jones, Alan K.: Mathematical tools for 3D multiblock grid design. Tech. Report ETA-TR-121, Boeing Computer Services, June 1989.

2. Jones, Alan K.: Grid singularities in the plane. Tech. Report ETA-TR-82, Boeing Computer Services, March 1988.

3. Jones, Alan K.: Grid singularities in three-dimensional domains. Tech. Report ETA-TR-83, Boeing Computer Services, March 1988.

4. Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W.: *Numerical Grid Generation,* North-Holland, 1985.

5. Eriksson, L.E.: Practical three-dimensional mesh generation using transfinite interpolation. *SIAM J Sci Stat Comp,* vol. 6, 1985, pp. 712-741.

THE NASA-IGES GEOMETRY DATA VISUALIZER*

Matthew W. Blake
NASA Ames Research Center
Moffett Field, CA

Lonhyn Jasinskyj
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA

Jin J. Chou
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA

SUMMARY

This paper describes NIGESview, an interactive software tool for reading, viewing, and translating geometry data available in the Initial Graphics Exchange Specification (IGES) format (ref. 1). NIGESview is designed to read a variety of IGES entities, translate some of the entities, graphically view the data, and output a file in a specific IGES format. The software provides a modern graphical user interface and is designed in a modular fashion so developers can utilize all or part of the code in their grid generation software for Computational Fluid Dynamics (CFD).

INTRODUCTION

The NASA Research Centers have started a coordinated effort to standardize the exchange of geometry, grid, and solution data used in the analysis of computational aero-physics problems. The first phase of this effort is to provide rapid and accurate geometry data exchange between Computer Aided Design (CAD) systems and grid generation software. A draft Technical Specification has been written titled "NASA-IGES Geometry Data Exchange Specification" (ref. 2). The three NASA Research Centers, Ames, Langley, and Lewis are currently developing grid generation software capable of utilizing data as specified in this NASA-IGES Geometry Data Exchange Specification.

---

* Mr. Jasinskyj and Dr. Chou's participation is provided under NASA Contract NAS 2-12961

Personnel supporting the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center are developing a general NASA-IGES data visualization tool called NIGESview. The NIGESview program is designed to assist in transferring complex geometry between various software systems. The NIGESview program can read IGES files and display the geometry data from the file in a three dimensional color graphics display. The user is provided with several tools to visually inspect the geometry and perform some modification. Many advanced capabilities are planned. The intended data flow between CAD systems and grid generation programs is depicted in figure 1.

This paper describes the program design and important features as well as current and future planned capabilities.

PROGRAM DESIGN AND FEATURES

The NIGESview program is implemented with a highly interactive user friendly interface utilizing mouse and menu driven input and advanced color graphics. The program is written in the C++ language and is designed to run on a Silicon Graphics workstation.

The NIGESview program is designed to read in CAD geometry data in either IGES or NASA-IGES format, remove non-geometric entities that are not needed, translate some geometric entities to more desirable forms, display and analyze the geometry, and output IGES files with the desired set of NASA-IGES characteristics. The user is provided with visual and statistical tools to aid in verification of the data and to provide a capability for minor modifications. A key design goal is to retain the heirarchical object structure of geometry data and to be able to graphically interact with the data. The NIGESview software functionality is shown in figure 2. A list of some important features planned for the NIGESview software is presented below.

- Input capability:
  - NASA-IGES compliant files
  - Most Standard IGES files
  - Statistics provided to user on input file processing
  - Window up/down through directories to select files

- Conversion capabilities:
  - Reject non-geometric, non-NASA-IGES entities
  - All non-NASA-IGES entities converted to B-Spline based entities
  - All conversion options are individually user selectable

- Output capabilities:
  - NASA-IGES files
  - NASA-IGES NURBS only files

- Viewing and manipulation features:
  - Rotate, translate, zoom, change center of focus, etc.
  - Change color, adjust light source, wire frame or solid,
  - Select object by number or picking on screen with mouse, turn on/off objects
  - Delete an object (IGES entity)

CURRENT AND FUTURE EFFORT

Initial versions of the software will read in NASA-IGES format files, display the geometry in various ways, and output a NASA-IGES compliant file. Additional capabilities will be added incrementally.

An Alpha version of the software is currently available. This version can read in and display most NASA-IGES entities and write out a NASA-IGES data file. Version 1 of the software will be completed in late summer of 1992 and will include a robust NASA-IGES file input and output capability as well as several object viewing options and file information displays. Most of the remaining planned capabilities should be available by early 1993, including complete IGES input and conversion to NASA-IGES format.

NIGESview should provide an excellent base program for developing grid generation capabilities. Some potential enhancements include surface grid generation and domain decomposition, display and processing of solid models, and different output formats. In addition to the complete NIGESview program, many of the individual modules should be of use to surface modeling and grid generation developers.

Both the executable software and the individual modules will be made available to U. S. government, industry, and academia through normal NASA channels. This will provide developers with the option of utilizing all or part of this software in their specific grid generation application.

REFERENCES

1. Initial Graphics Exchange Specification (IGES) Version 5.1, distributed by the National Computer Graphics Association (NCGA) Technical Services and Standards, IPO Administrator, 2722 Merrilee Drive, Suite 200, Fairfax, VA 22031, telephone 703-698-9600 extension 325

2. Blake, Matthew W.; Chou, Jin J.; Kerr, Patricia A.; Thorp, Scott A.: The NASA-IGES Geometry Data Exchange Standard. *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA CP-3143, April 1992.

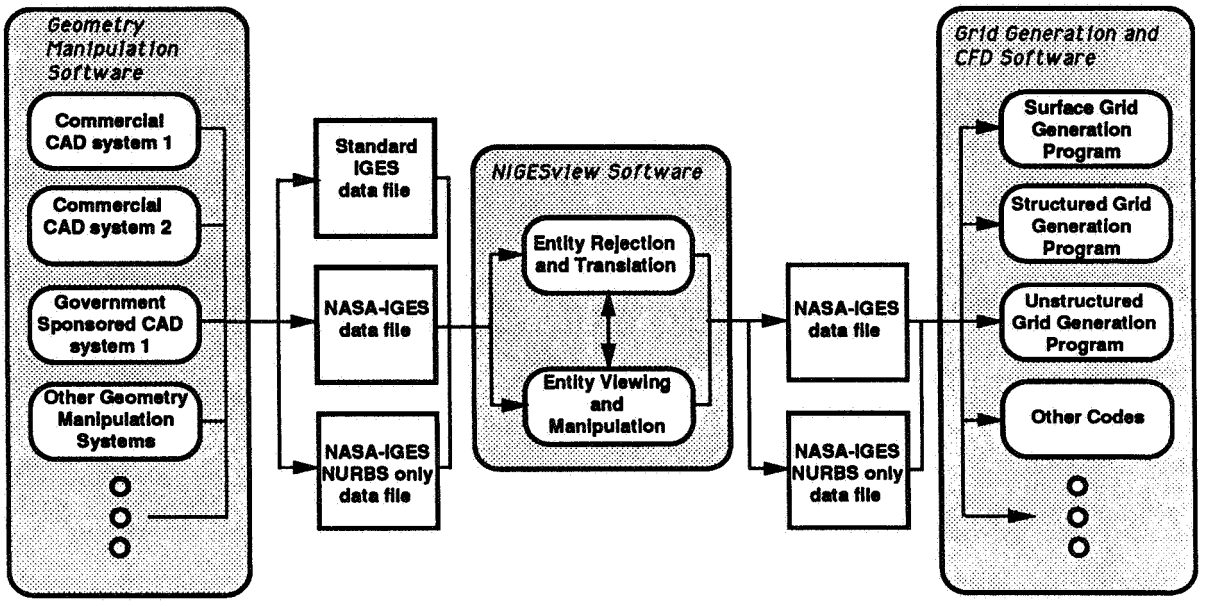


figure 1. NASA-IGES Data Flow

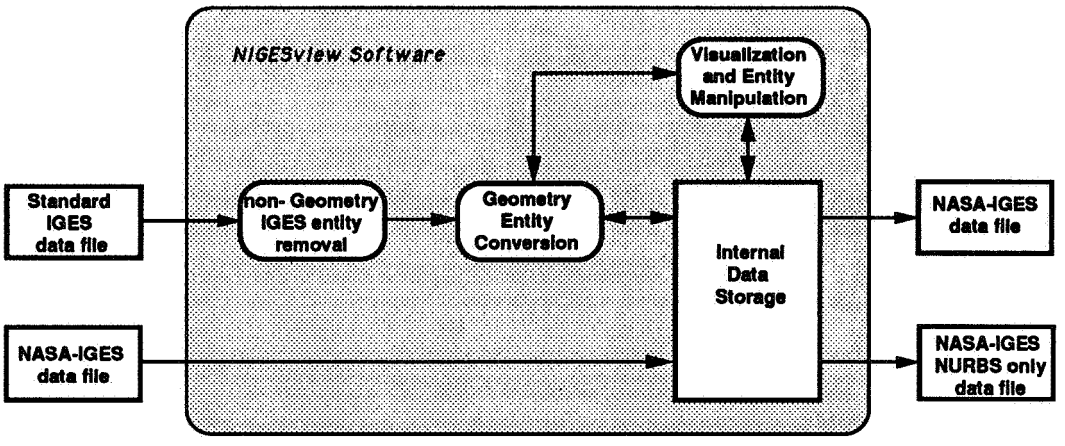


figure 2. NIGESview Software Functionality

GRAPEVINE: GRIDS ABOUT ANYTHING BY POISSON'S EQUATION IN A VISUALLY INTERACTIVE NETWORKING ENVIRONMENT*

Reese L. Sorenson
NASA Ames Research Center
Moffett Field, CA

Karen McCann
Sterling Software
Palo Alto, CA

SUMMARY

A proven three-dimensional multiple-block elliptic grid generator, designed to run in "batch mode" on a supercomputer, is improved by the creation of a modern graphical user interface (GUI) running on a workstation. The two parts are connected in real time by a network. The resultant system offers a significant speedup in the process of preparing and formatting input data and the ability to watch the grid solution converge by re-plotting the grid at each iteration step. The result is a reduction in user time and CPU time required to generate the grid and an enhanced understanding of the elliptic solution process. This software system, called GRAPEVINE, is described, and certain observations are made concerning the creation of such software.

INTRODUCTION

Mathematical algorithms for grid generation, especially structured grid generation, are well advanced. But even so it can take six man-months to generate a useable multiple-block grid about a real airplane or any other complex real-world shape, with that effort split between surface modeling, surface gridding, and volume gridding. A major reduction of that discrepancy between algorithms and performance, in the area of volume grid generation, is the subject of this paper.

3DGRAPE[1,2] is an example of grid generation software which is algorithmically mature, but the use of which can account for a generous share of those six man-months. The user's time goes to collecting and formatting input data and to actually running the grid generation program. Input data for such a grid generator (EAGLE[3,4] is another example of the genre) can extend to thousands (or even tens of thousands) of lines for a grid about a realistic shape. A reasonable approach to reducing that burden is to use the graphical power of a workstation to speed-up the processes of collecting and formatting the input data, and of running the program. GRAPEVINE is a software system which attempts to do that.

---

* Ms. McCann's participation in this project was under contract to Sterling Software, NAS2-13210.

GRAPEVINE consists of the five code modules illustrated in Figure 1. First is a newly-written user-friendly GUI, running on a workstation, which controls the process. Second is 3DECANT, a code module which displays the grid. The third code module is 3DGRAPE, the elliptic multi-block grid generator program. The fourth code module is 3DPREP[5], a recently-written workstation program to speed the process of collecting and formatting the input data for 3DGRAPE. 3DPREP is not yet fully integrated with the other code modules, and so is shown as separated from them in the Figure.

While the GUI, 3DECANT, and 3DPREP are written in C and naturally reside on a workstation, the elliptic solver is written in FORTRAN and must run on a supercomputer due to the large amount of calculation required. Hence the GRAPEVINE system operates across a network, with the C modules running on a workstation and the elliptic solver running concurrently on a supercomputer. The code to effect that communication and control across the network is the fifth module making up GRAPEVINE.

The result is a system which will work as shown in the flowchart in Figure 2. The user first prepares input data in a graphically enhanced and accelerated fashion. The user then decides just what grid surfaces should be viewed during the iteration process and how they should be viewed, and then chooses to do the numerical processing on either the workstation or a supercomputer. He sends that data to the numerical processor and starts it running. The numerical processor notes what very small subset of all the grid data is required each iteration to update the grid plotting, and sends that data back to the workstation each iteration. Thus the grid plot is updated every iteration, allowing the user to watch it as it converges. By using these features the user can allow the solution process to continue, or stop it and change some data, or abort it and start anew. The result is a greatly enhanced understanding of the elliptic grid generation process, and significant savings in CPU time and the user's time. At present the step labeled "Create control scalars," shown in the upper left-hand corner of the Figure, is done in the separate program 3DPREP, or in an editor.

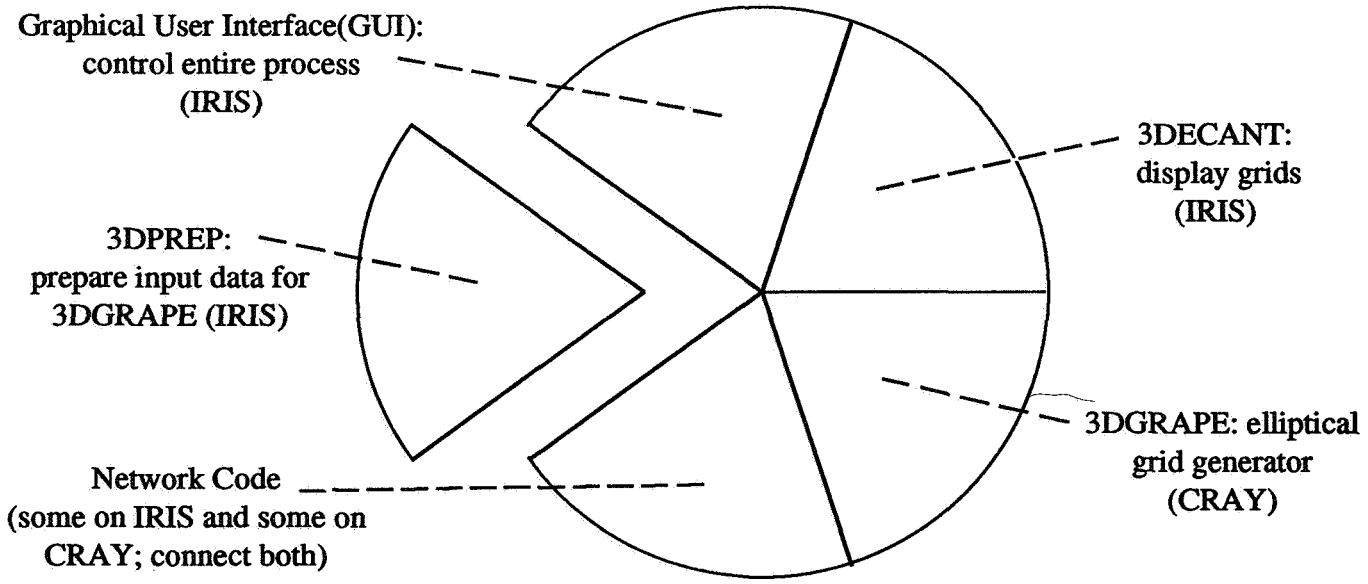


Figure 1. Modules comprising GRAPEVINE.

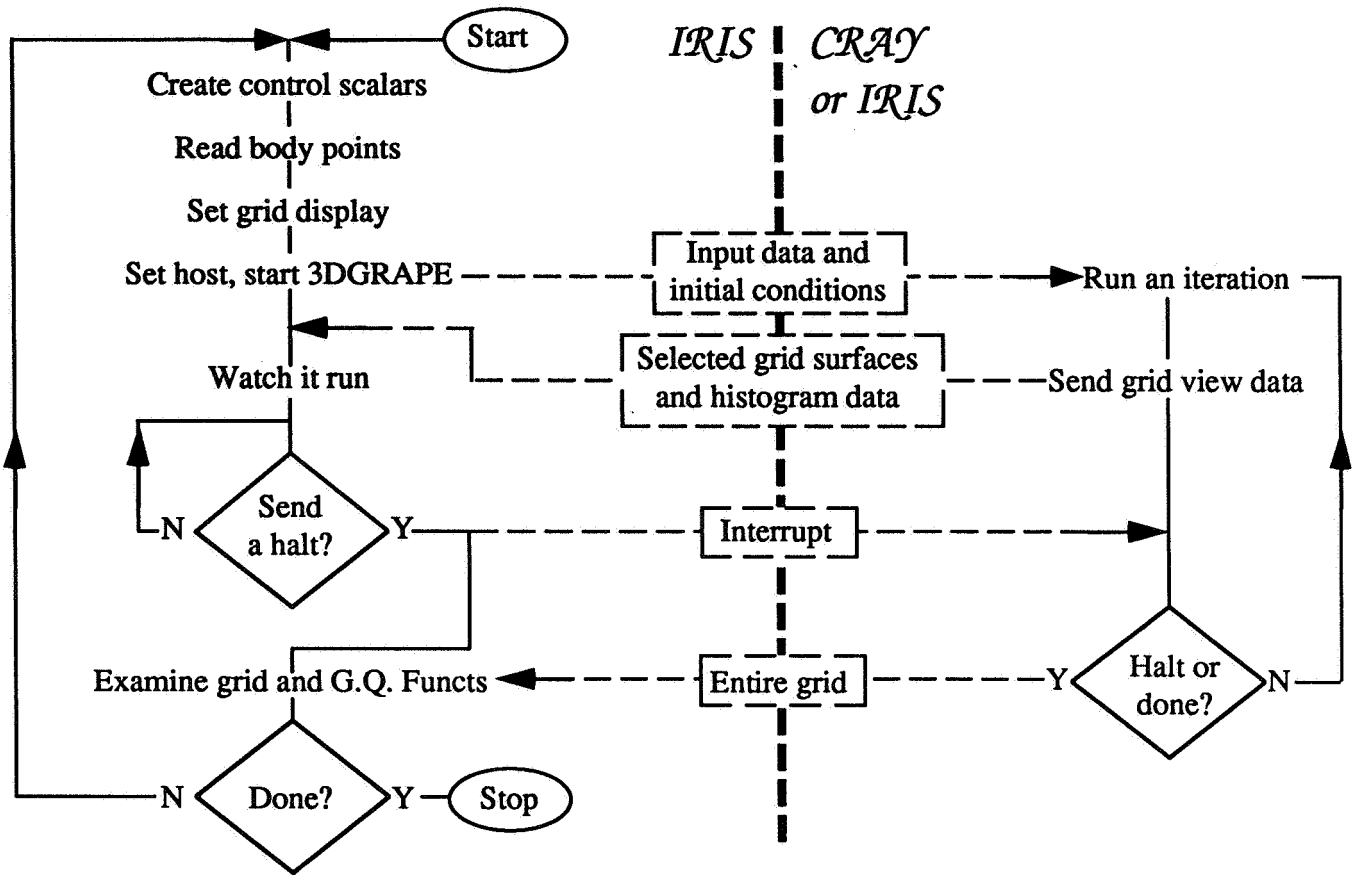


Figure 2. Flow chart summarizing GRAPEVINE operation.

CODE MODULES

GRAPEVINE is a combination of several code modules, some new and some old.

The 3DPREP Module

This module is currently a separate program, running on a workstation, which helps the user collect and format input data for the 3DGRAPE program. It consists of a multiple-screen graphical user interface. It obtains its input data by asking the user questions, or allowing the user to enter input data in a random-access fashion, or allowing the user to read in an old case and then selectively modify it. The module outputs two files (called file10 and file11, file names and file structures which are familiar to users of 3DGRAPE). These files are then used as input data for the 3DGRAPE grid generator. This code module, 3DPREP, is discussed in detail in Ref. 5. Future plans call for integrating its capabilities into GRAPEVINE.

The 3DECANT Module

This module has been fully integrated into GRAPEVINE. It consists of graphics code and transformation algorithms. It allows the user to select the grid surfaces, taken from any block or blocks, which are to be plotted. The user can color the surfaces arbitrarily or by grid quality functions, and do any combination of translation, rotation, and zooming, about either the body's axes or the screen's axes, with the center of rotation being the centroid of that which is being viewed. Various other display options, such as hidden-surface-removal, are included. This module is also discussed in Ref. 5.

The 3DGRAPE Module

This code is the proven three-dimensional multiple-block elliptic grid generator. Given the user's requirement for cell height on boundary surfaces, numerical values for inhomogeneous terms are found iteratively by the code. The result is local near-orthogonality and controlled cell height on boundary surfaces, with those controlling influences decaying exponentially toward the interior of the blocks. It is a multiple-block volume grid generator wherein the block-to-block boundary surfaces are found automatically by the code, alleviating the need for the user to define them before starting. This module is discussed in detail in Refs. 1 and 2. As used in GRAPEVINE the code is improved by better initial conditions, and by an improved treatment of sharp corners in the middle of block faces.

The Graphical User Interface Module

The graphical user interface and the network code are the principal contributions presented in this paper. The graphical user interface consists of several screens.

The Data Input/Output (I/O) Screen

GRAPEVINE begins in the Data I/O Panel, as shown in Figure 3 (all the screen Figures in this paper have been reduced to grayscale, but generous use of color is made in the program). Default file names for input and output are given. The user has the opportunity to overwrite those file names. Upon exit, the current file names are written into an "environment file" which is read upon the succeeding startup. Thus the user's chosen file names in the current run become the default file names for the next run, simplifying operation. The input files include the file10 and file11 input files which are created by 3DPREP. Once the user has entered the necessary file names, he clicks on either the "read start files" button or the "read restart files" button. Those simple operations give GRAPEVINE all the information it needs to generate the grid.

However, the program needs one more input data file, this one telling it how to plot the grid. The creation of this file is discussed below, but once it has been created it is typically just read in and used as

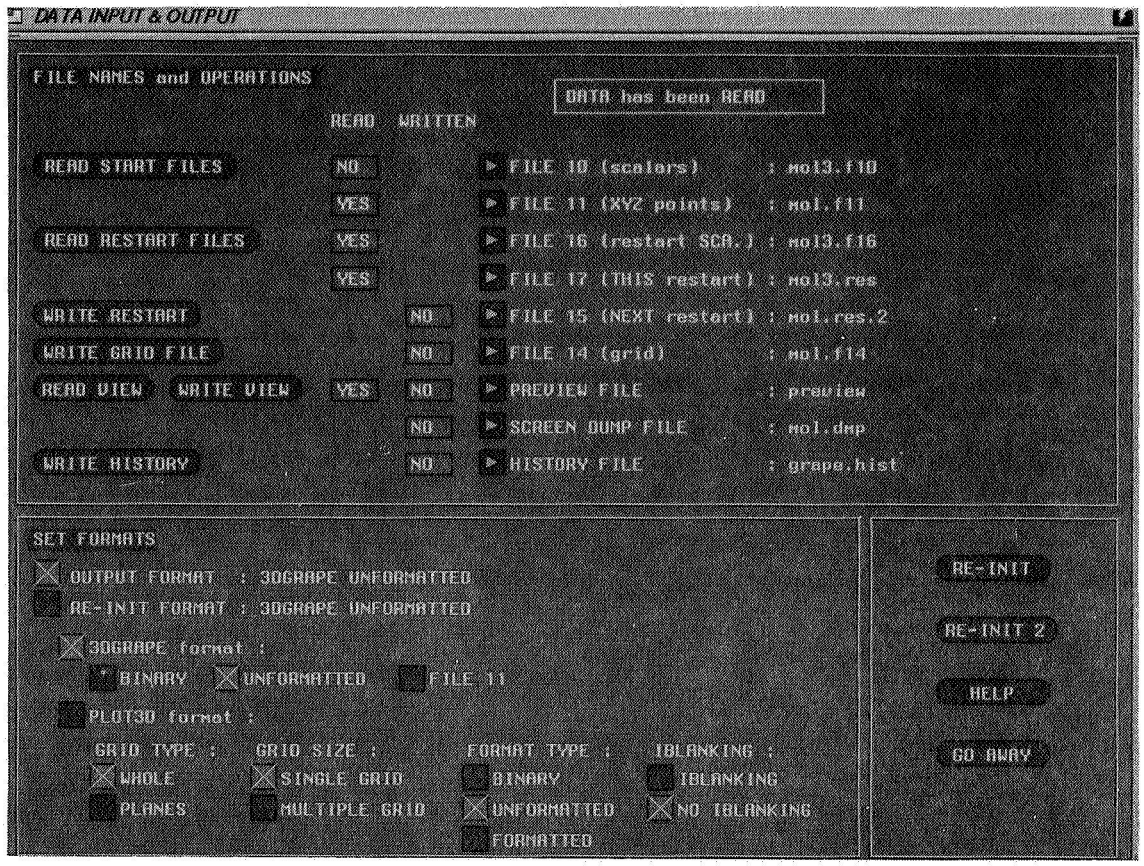


Figure 3. Data I/O panel.

in the previous run. Doing this requires clicking on the "read view" button. Thus after clicking on two buttons, the program is in most cases ready to begin calculating.

The lower part of the Data I/O Panel shows the different file type options available for outputting the grid. It can write in the 3DGRAPE or PLOT3D formats, with several options for each format. 3DGRAPE has the ability to do both new starts and restarts; this capability is preserved in GRAPEVINE.

Upon exiting the program this panel comes up again. It shows the user not only what files have been read but also what files have been written, in the small boxes containing the words "yes" or "no." If the user desires to write a file which was not requested in the input data, such as a restart file, it may be done from this panel.

The Create Surface Panel

This panel, shown in Figure 4, is where the user specifies what surfaces are to be plotted and how they are to be colored. Any collection of surfaces or portions thereof viewed together make a "surface set." The user may specify any number of surface sets. When viewing the user may cycle through the

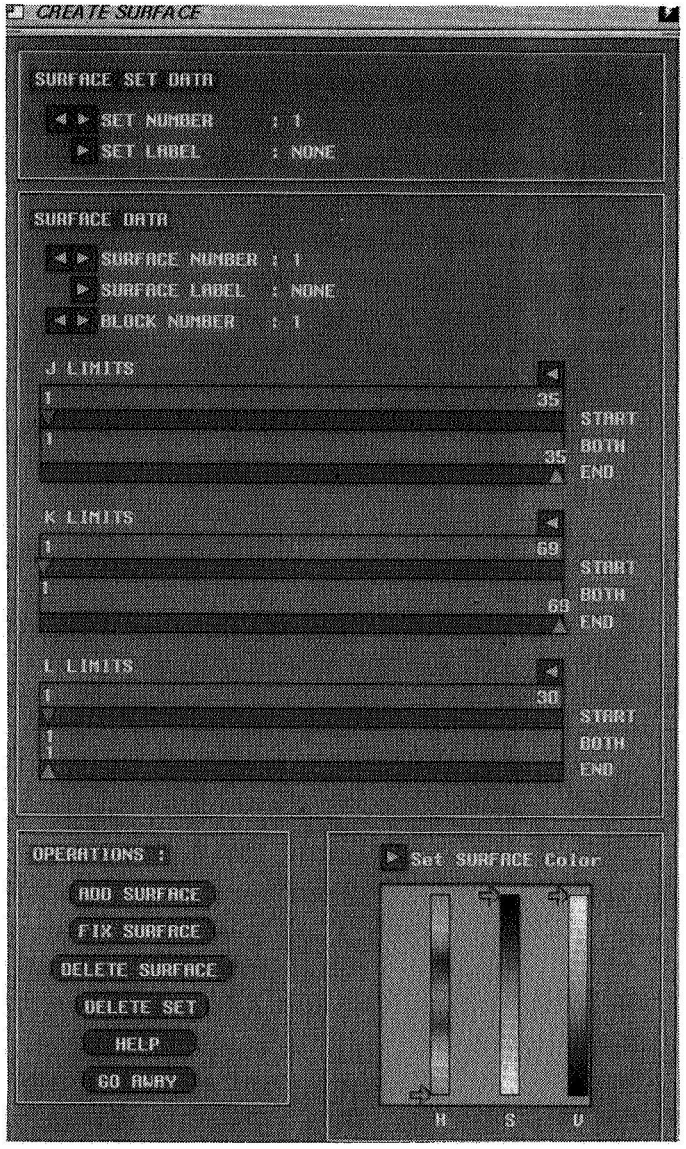


Figure 4. Create surface panel

given surface sets, viewing each separately.

Sliders shown in the center of the Create Surface Panel are used to specify the starting and ending index values which, together with a block number, specify each surface or portion thereof (the user could even specify a solid region for a surface, such as an entire wing including both upper- and lower-surfaces). The lower-right-hand corner of the Create Surface Panel gives three vertical sliders which are used to specify the color in which the surface is to be plotted. The color is specified by the hue-saturation-value method.

Plotting transformations, discussed above, are used to translate, rotate, and zoom the plots of the surface-sets. At the user's option a different orientation for each surface set may be used, or the same orientation may apply to all surface sets.

The data specified in this panel — block numbers, index limits, color codes — together with the orientation matrices which are the result of operating the transformations, are written at the conclusion of the run and are read in the next time as the "view file." This file is simple text, and can be viewed and modified with an editor. This is an example of an attempt, also made in 3DGRAPE, to keep the input data file types accessible to the user. This is much appreciated if, for whatever reason, things go wrong.

The Main Iteration Screen

There is one task which the user will wish to do before beginning the iteration process, and that is to specify upon which host processor the numerical calculations for solving the Poisson equations will be done. For small cases the user may wish to perform the calculations in "standalone" mode on the workstation. In most cases, however, a supercomputer is required. The "remote host" panel, shown in Fig. 5, effects this choice. The computers available are, of course, installation-dependent. At the Numerical Aerodynamic Simulation (NAS) facility at NASA Ames Research Center the choice is between the user's

IRIS workstation (a product of Silicon Graphics, Inc., also referred to as SGI), a CRAY-2 called Navier, and a CRAY-YMP called Reynolds (both CRAYs are products of CRAY Research, Inc.).

The main iteration screen, which the user watches during the iteration process, is shown in Fig. 6. The large plotting region on the lower left shows the grid at each time step. Before starting the iteration the user cycles through the given surface sets, choosing the one to be viewed during the iteration, and positions it as

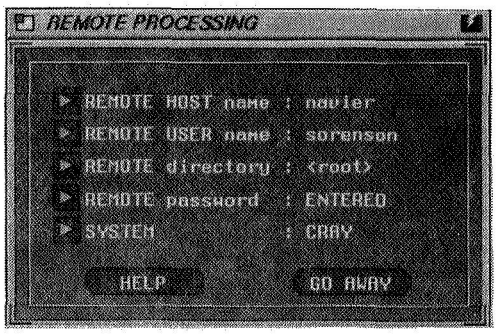


Figure 5. Remote host panel

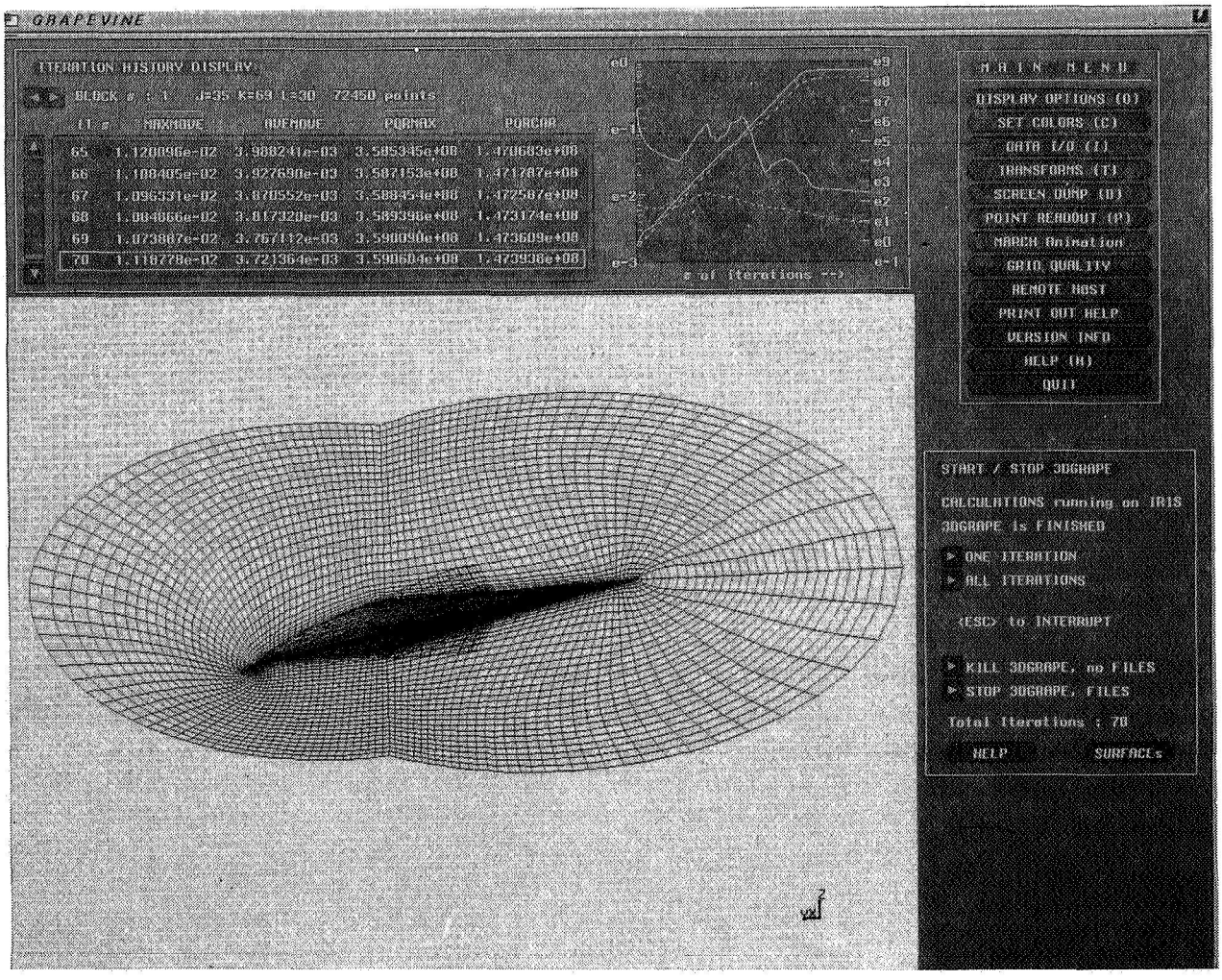


Figure 6. Main iteration screen

desired by using the transformations. At any time during the iteration process the user may interrupt, reposition the viewed set, or choose another surface set, and continue.

Above the plotting region is the iteration history display. For a chosen block four parameters are given in numerical form at each iteration. These parameters, familiar to users of 3DGRAPE, are MAXMOVE, the distance moved by the point which moved the farthest, AVEMOVE, the average of the distances moved by all interior points, PQRMAX, the maximum absolute value of the forcing functions at each point, and PQRCOR, the maximum absolute value of the corrections applied to the forcing functions. These same four parameters, all functions of iteration count, are also shown as plots. In the actual program color is used to differentiate between the different function plots.

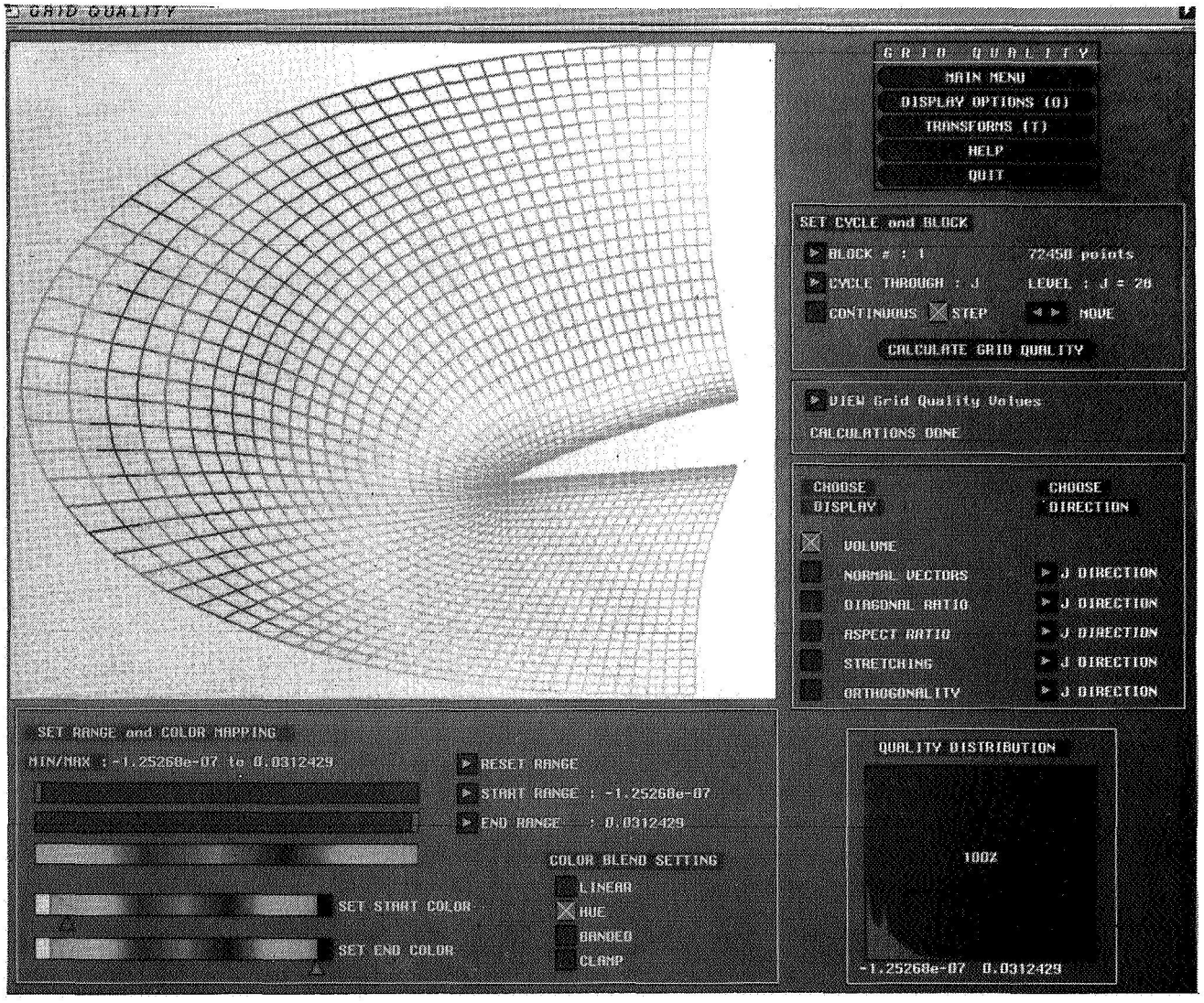


Figure 7. Grid quality screen.

The main menu is given on the upper right, and the panel on the lower right controls starting and stopping the iteration process.

The Grid Quality Screen

The last remaining screen, the grid quality screen, shown in Figure 7, is a graphical implementation of a grid quality program originally written for batch operation in FORTRAN by K. Chawla. For a chosen block it gives the user the option of calculating the cell volume, and up to five other grid quality functions, with each of the other five functions in any of the three coordinate directions. Those grid quality functions are: (1) the angle between neighboring normal vectors, (2) the ratio between the lengths of the two diagonals in a cell face, (3) the aspect ratio of the cells, (4) the ratio between the heights of adjacent cells, called the stretching ratio, and (5) the orthogonality given by dot-products. Volume or any one of the other functions can be shown at any time. The chosen function is used to color grid surfaces as the program marches through the grid by incrementing or decrementing a chosen index. The marching may be manual or automatic. Various options are given to map the calculated function values into colorations. A distribution display shows how many of the points have various values of the chosen function. The user may rotate, translate, and zoom the displayed surfaces.

The Network Code Module

As mentioned above, the user may choose to do the mathematical calculations (the numerical solution of the Poisson equations) on either the IRIS upon which the remainder of the program is running, or on some remote host processor (typically a supercomputer such as a CRAY). If the choice is the IRIS, no networking is needed; the 3DGRAPE program is simply linked with the other code and called.

But if a remote host is chosen, a modified version of 3DGRAPE is run on the CRAY. That operation requires several steps:

1. The input data files (file10 in the case of a new start or file16 in the case of a restart, and file11) are sent from the IRIS to the CRAY. This is done by using the C function "system" to call the UNIX function "rcp."

2. The UNIX function "rexec" is used to start the remote process on the CRAY.

3. The C header code on the CRAY reads the input files to determine what dimension sizes are required, and allocates storage on the CRAY accordingly.

4. The IRIS sends to the CRAY a list of the surfaces which the user has chosen to display during the iterative process.

5. 3DGRAPE in FORTRAN on the CRAY re-reads the input data, initializes its variables, and begins iterating.

6. After each iteration the x,y,z values for the displayed surfaces are sent to the IRIS, along with convergence history information which is to be plotted as a function of iteration count. The IRIS display is then updated.

7. When the iteration procedure is completed, or when an exit interrupt is received by the CRAY, the entire grid is sent to the IRIS.

Three different IRIS-to-CRAY interrupts were found to be necessary: *kill, exit,* and *pause-continue*. A *kill* interrupt causes the CRAY process to halt, possibly in the middle of an iteration, and not send the grid back. This is useful when the grid being generated is obviously unsatisfactory, and the most expedient halt is desired. An *exit* interrupt causes the CRAY to complete the current iteration, send the grid back, and stop. This is used when the grid appears to be good enough to use without further iteration. A *pause* interrupt causes the CRAY to finish the current iteration, send the entire grid back to the IRIS, and then halt without releasing CRAY memory or terminating the run. The user can then examine the grid graphically on the IRIS by cycling through other surface sets; do translation, rotation, and zoom operations on those plots; and plot the grid with coloration by grid quality functions with marching through all grid points. Once development of the program is finished, and the functionality of 3DPREP is fully integrated into the code, the user will be able at this point to modify the input parameters. After performing these tasks during the pause, the user can issue the *continue* command and re-start the CRAY process.

Network functions "socket" and "bind" were used to open sockets on both the IRIS and CRAY sides of the network. All communication is done over the sockets, with the exception of the *kill* interrupt which is sent at the system level. Routines from the CRAY library "binconv" were used to change floating point and integer numbers between CRAY and IRIS formats. Algorithms used were partially modeled on the sample code "chat" provided by SGI on IRIS systems, and the networking library "dlib" written by M. Yamasaki. All network functions are error-trapped; this is necessary because hardware interruptions from both the CRAY and the network may occur at any time.

The purpose of distributing the code over a network is to accelerate the iterative solution of the Poisson equations, relative to the speed at which they would be solved running on the IRIS alone. The speed at which the code runs when distributed is dependent upon many factors. First is the IRIS, which slows due to a lack of free memory or too many users. Second is the network, which slows under heavy use. Third is the CRAY which, when many users are logged on, can cause a particular task to wait for as much as sixty seconds before getting another time slice. For these reasons it is difficult to give a definitive statistic concerning the degree to which the code speeds up when distributed, but experience indicates a speedup by a factor of ten to fifteen.

LESSONS LEARNED

It should be recognized that distributed processing, at least in the CFD context, is in its relative infancy. There have been networks for many years, over which data and news are shared. But there are few examples of systems wherein the *processing* of data is shared over more than one node of a network.

Doing so allows specialization wherein individual processors are assigned those tasks at which they excel. The present work is one example of truly distributed processing. The workstation excels at graphical interfacing, while the supercomputer excels at "number crunching."

When this project was begun approximately two years ago CRAY computers (together, possibly, with the emerging Japanese supercomputers), held a unique position in the computational scheme of things, widely accepted as the only real practical supercomputers. And so at that time, due to their position in the industry and the fact of the availability of several of them at NASA Ames Research Center, CRAYs were the obvious choice for the numerical processor for this project. It is doubtful that networking was high on the priority scheme during their design (one might argue that CRAYs are good at number crunching *because* they do *not* excel at networking). But that lack of emphasis has revealed itself in certain difficulties faced by this design team. CRAYs tend, it would seem, to favor long time slices per user, causing the interactive user to have to wait as long as sixty seconds for the processing of an interrupt. And it was the experience of this design team that the CRAYs crashed several times per day (the IRIS workstations are up for weeks at a time). And because they are so attractive for number crunching, CRAYs tend to be heavily loaded, further slowing things. Also, it was found that certain networking functions worked differently on different host computers. CRAYs and IRISes have different internal data formats, giving rise to the need for conversion before transfer. These difficulties took their toll on this development effort. CRAY's position in the scheme of things is now threatened by ultra-high-end multiple-processor workstations, which have a clear price/performance advantage, and by massively parallel machines, which have the potential for much higher mflops rates. Ultra-high-end workstations and massively parallel machines should be considered as candidates for the numerical processor in the design of any such distributed system in the future.

At the outset of this project there was debate over how much of the pre-existing 3DGRAPE code should be translated into C. Opinion ranged from "all" to "none." As it turned out, a header program was written in C which reads the input data to determine array dimension requirements, allocates arrays accordingly, processes iterrupts, and does data output for transfer back to the IRIS. This header calls various parts of 3DGRAPE in FORTRAN to re-read the input and initialize variables. And the C header contains the "main iteration loop" in which other parts of 3DGRAPE in FORTRAN are called to actually perform an iteration. If these authors had it to do over again, more of the FORTRAN would be translated into C, principally the remainder of the data I/O, and the variable initialization, leaving only the contents of the main iteration loop in FORTRAN.

It is obvious that the graphical capabilities of modern workstations can have a great and beneficial effect upon productivity of scientists and engineers in general, and on practitioners of CFD in particular. But they bring about a situation with respect to programming languages which is analogous to that which argues for distributed computing. Just as distributed computing allows different computers to do that part of the job at which they excel, so does multi-linguistic programming allow different languages to do what they do best. C (or C++) is the language of choice for graphical interfaces for many reasons including versatility in data structures, versatility in performing I/O, and compatibility with graphical libraries.

FORTRAN (especially as implemented on CRAY computers) vectorizes best, and is still the language of choice among the scientists and engineers who develop the technology implemented by these systems. And so the paradigm suggested above, wherein the graphical user interface, the networking, array allocation, initialization of variables, and I/O are done in C (or C++), and the contents of the "main iteration loop" remains in FORTRAN, seems at present to be a workable compromise.

CONCLUSIONS

A working software system for generating structured multiple-block grids, distributed over a network between a supercomputer and a workstation, has been developed and tested. Because the code is distributed its user interface is interactive on a workstation, and its numerical processing is performed on a supercomputer. Thus a great reduction is seen in user time required to collect and format the input data and to run the code, and the numerical processing is performed much faster than if it were done on a workstation.

It was the experience of these developers that CRAY supercomputers, though powerful at numerical processing, are not ideally suited for interactive networking applications. Developers of similar distributed software systems in the future should instead consider the use of ultra-high-end workstations or massively parallel computers for role of numerical processor. It is further the conclusion of these developers that when coding a graphical user interface in C for a pre-existing numerical application in FORTRAN, everything except the contents of the "main iteration loop" should be translated into C.

ACKNOWLEDGEMENTS

The authors are grateful to Michael J. Yamasaki and Matthew W. Blake of NASA Ames Research Center, and John R. Campbell of Computer Science Corp. for their help in writing the networking code. The authors are grateful to Kalpana Chawla of the MCAT Institute for supplying a grid quality code. And the authors thank the NAS Applied Research Branch, the NAS Systems Development Branch, and the Office of Technology Utilization at NASA Ames Research Center for partial financial support.

REFERENCES

1. Sorenson, R. L.: Three-Dimensional Zonal Grids About Arbitrary Shapes by Poisson's Equation. Appears in Sengupta, S., Häuser, J., Eiseman, P. R., and Taylor, C., eds.: *Numerical Grid Generation in Computational Fluid Mechanics.* Pineridge Press Ltd., 1988.

2. Sorenson, R. L.: *The 3DGRAPE Book: Theory, Users' Manual, Examples,* NASA TM-10224, 1989.

3. Thompson, J.F.: A Composite Grid Generation Code for General 3D Regions — the EAGLE Code. *AIAA Journal,* vol. 26, no. 3, March, 1988, p. 271.

4. Thompson, J.F. and Lijewski, L.E.: Composite Grid Generation for Aircraft Configurations with the EAGLE Code. Appears in Thompson, J.F. and Steger, J. L., eds.: Three Dimensional Grid Gen*eration for Complex Configurations — Recent Progress,* AGARD-AG-309, 1988.

5. Sorenson, R. L., and McCann, K. M.: A Method for Interactive Specification of Multiple-Block Topologies. AIAA 91-0147, Jan., 1991.

AN INTERACTIVE MULTI-BLOCK GRID GENERATION SYSTEM

T. J. Kao and T. Y. Su
Aerodynamics Research, Engineering Division
Boeing Commercial Airplane Group
Seattle, WA

Ruth Appleby
CFD Computing
Boeing Computer Services
Seattle, WA

SUMMARY

A grid generation procedure combining interactive and batch grid generation programs has been put together to generate multi-block grids for complex aircraft configurations. The interactive section provides the tools for 3D geometry manipulation, surface grid extraction, boundary domain construction for 3D volume grid generation, and block-block relationships and boundary conditions for flow solvers. The procedure improves the flexibility and quality of grid generation to meet the design/analysis requirements.

INTRODUCTION

The process for conducting a computational fluid dynamics (CFD) analysis usually follows four steps:

- Geometry modeling
- Model discretization (2D and 3D)
- Flow analysis
- Result interpretation.

As the flow codes become more reliable and easy to use, the flowtime for acquiring a solution becomes relatively small when compared to the pre-processing step of geometry modeling and discretization, the most difficult and time-consuming aspects of the process.

This paper describes a multiblock grid generation approach in which geometry continuity is maintained and the required surface grids and block definition are prepared using the Boeing-developed Aero Grid and Paneling System (AGPS)[1,2], and the block-block relationship bookkeeping and boundary conditions are achieved using an interactive graphics interface program, BCON[3].

A unique feature of AGPS is that it is a *geometry programming language*. It allows users to create command files to perform specific tasks. This has allowed us to develop a collection of command files specifically for grid generation problems. This grid generation package contains geometry manipulation tools, various gridding functions, and graphics utilities for generating surface grids and defining block topology. And most importantly, the surface grids in AGPS are based on the actual mathematical surfaces, not approximations.

BCON is a menu-driven graphics interface program. BCON input consists of strings or arrays of points generated from AGPS or another CAD tool/surface geometry source. It generates input files that contain the block definitions, the block relationships and the block boundary conditions required for a multi-block volume grid generation and flow solver.

The combination of AGPS and BCON provides an effective means of producing surface and field grids for arbitrary three-dimensional configuration. In the following sections, we discuss some of the results in applying current procedures.

METHOD/APPROACH

The current approach is to use AGPS to define lines, curves, and surfaces at the block or sub-block boundaries and then to extract the edge and surface grids on these lines, curves, and surfaces. After the edge and surface grids are defined, BCON interactively combines those edge and surface grids into blocks, then writes out block data for 3D volume grid generation and block-block relationships with boundary conditions for an Euler or Navier-Stokes flow solver [4]. We are currently generating 3D volume grids with the Eglin Arbitrary Geometry implicit Euler (EAGLE), a batch-oriented algebraic/elliptical grid code[5]. Figure 1 shows the relationships of each step. It should be noted that this approach is not limited to a particular 3D grid code.

AGPS CAPABILITY

A 3D surface geometry system, AGPS was designed by aerodynamicists primarily for aerodynamics applications. AGPS is a useful tool during the preprocessing and postprocessing steps of the CFD process because of its ability to model both general and complex geometric shapes. It allows for double-valued curves and surfaces and has a variety of surface and curve types, including bicubic, biquintic, conic, interpolated, non-uniform rational B-spline (NURBS), and ruled. It generates surface intersections easily and accurately and has a unique *subrange* surface capability that allows surfaces to be broken or trimmed into smaller regions while keeping the exact mathematical definition used in the original. AGPS-generated surface lofts can be checked thoroughly for flaws using shaded graphics, curvature plots, and various other methods, helping the user achieve a surface with the desired characteristics.

Most importantly, AGPS is more like a geometry programming language than a program itself. The command files make it easy to repeat tasks. In addition, a playback feature is available to interactively record the steps followed.

With the AGPS user-accessible menu (Figure 2) and command file structures, we have developed a grid generation *package* that includes data I/O, geometry manipulation, various gridding functions, and graphics utilities. Inexperienced users need only limited CAD or geometry training and do not need to understand internal data structure or remember a multitude of object names during the operations. With some experience, users can tailor the package to fit their particular needs.

Currently, the primary function of the grid generation package is to define block edges and faces for a multiblock structured grid. As shown in Figure 2, the package allows users to

1. Input AGPS geometry and extracted grids
2. Manipulate curve and surface geometry
3. Extract curve and surface grids
4. Output grids in a specified format
5. Manipulate graphics displays and a geometry database

These functions give engineers a set of highly interactive graphics tools in a single environment to generate geometry, distribute points, and interface with a particular 3D grid generator and/or flow solver.

After preparing the block layout topology, including the number blocks, shape and size of each block, and location of the blocks, the user then creates the block layout interactively by defining lines,curves, and surfaces at the block and sub-block boundaries. Next, the edge and surface grids are generated. The

points are extracted directly from the curves and surfaces and lie precisely on the original mathematical definitions.The user can also control grid distribution and perform grid manipulations such as merging and splitting.

BCON CAPABILITY

BCON is a menu/mouse-driven, interactive graphics interface program written in C and using NASA-Ames Research Center's PANEL Library for a graphical user interface on the Silicon Graphics Inc.'s IRIS graphics workstations. The user first prepares the geometry surface definition and decides on the block topology, as well as surface grids, with AGPS. With BCON, the user then defines blocks, block-block relationships, and boundary conditions for input to the volume grid code (currently, EAGLE) and the flow solver. BCON accepts input in the form of strings or arrays of points from AGPS or other geometry/surface grid generation sources. It is highly modular, which allows new features to be incorporated easily.

Major functions BCON provides are

1. Input
2. Define block
3. Define i, j, k index system for each block
4. Impose boundary conditions
5. Write EAGLE run stream and input data deck
6. Write block-block relationship and boundary conditions input files for flow solvers

To simplify the data preparation and reduce redundant data, unique edge/face is required and each block should be represented by as many edges/sub-edges as possible. In cases of original geometry, or when it is necessary to preserve the distribution of interface points, blocks are represented by faces/subfaces. Once a face is defined, no edge associated with that face needs to be prepared; BCON code can identify the edge from the related face.

Each block can be formed by a combination of edges/sub-edges and faces/sub-faces.For a simple face, only two edges need to be defined. For a simple block, only two opposite faces need to be defined. If a block has complex faces, these faces need to be defined first. Similarly, if a face has complex edges, the edges need to be defined first . BCON picks up the remaining simple edges/faces automatically and forms the block.

BCON lets the user interactively select the origin from one of the block vertices and define the directions of the local indices i, j, and k for the first block. The code will propagate these indices throughout all the blocks defined in the blocking process. The block-block relationship is established during this process. Once the blocking process is finished, only the faces without neighbor blocks are displayed; on these faces, the user can impose a variety of boundary conditions, i.e. solid wall, symmetry plane, inlet, exhaust, and far-field. Output files include the EAGLE run stream and input data deck for 3D volume grid generation, as well as block-block relationships and boundary condition input files for flow solvers.

APPLICATIONS

To demonstrate the capability of the present procedure, we have studied several test cases, from simple two blocks wing/body to full configuration four-engine transport airplane. Figure 3 shows the surface grid of a wing/nacelle/strut configuration. The complete flow field consists of 20 blocks (Figure 4) with a total number of 60,000 grid points. An H-type grid is used for all external flow field, and a cylindrical grid is used for nacelle inlet and exhaust flow. Figure 5 shows the isobars of Euler analysis on the configuration surfaces.

Figure 6 shows surface grids for the body/wing/strut/nacelle twin-engine transport model. The volume grid contained approximately 1.2 million grid points and was composed of 26 to 32 blocks,

depending on the nacelle simulation. The entire flow through nacelle configuration (with pylon, wing, and fuselage) is modelled with 32 blocks (Figure 7). Fifteen of these blocks are used for the flow-through core and fan cowls. Figure 8 shows constant K plane field grid with part of the surface grid of a fan cowl. Figure 9 shows a comparison of computed wing Cp with wing tunnel data for nacelle installed at eta=0.34, at M=0.77 with Cl=0.55.

The third case is the high speed civil transport (HSCT). Figures 10 shows a symmetry plane and surface grid for cruise configuration with vertical tail. Figure 11 shows the surface grids for high-lift configuration without a vertical tail. The general block layout for high-lift and cruise configuration is shown in Figure 12. It used 10 blocks for a cruise case and 18 blocks for a high-lift case. Figure 13 shows the surface streamlines for both cruise and high-lift cases.

The last case is a four-engine 747-200 full configuration for Navier-Stokes analysis. The complete flow field consists of approximately 45 blocks with over 2 million grid points. Figure 14 shows the complete geometry surface grids. Figure 15 shows a layout of some blocks near the configuration. For this case, the viscous calculation was limited to the wing. It took 14 blocks to cover the boundary layer region. Figure 16 shows viscous blocks near a wing/body junction.

CONCLUSION

The combination of the AGPS Grid Generation Package and the BCON program significantly reduces the flowtime required for generating and pre-processing 3D multiblock grids. The long term goal of this effort is to merge these two capabilities with 3D volume grid generation into one integrated environment, giving users timely, accurate 3D grid generation.

REFERENCES:

1. D.K.Snapp and R. C. Pomeroy, "A Geometry System for Aerodynamic Design," AIAA/AHS/ASEE Aircraft Design, System, and Operations Meeting, AIAA-87-2902, September 14-16, 1987.

2. W. K. Capron and K. L. Smit, "Advanced Aerodynamic Applications of an Interactive Geometry and Visualization System," 29th Aerospace Sciences Meeting, AIAA-91-0800, January 7-10, 1991.

3. T. Y. Su, R. A. Appleby, and H. C. Chen, *A General Multiblock Euler Code for Propulsion Integration, Volume II: User Guide for BCON, Preprocessor for Grid Generation and GMBE*, NASA CR187484, May 1991.

4. N. J. Yu, H. C. Chen, T. Y. Su, and T. J. Kao, "Development of a General Multiblock Flow Solver for Complex Configuration," Proceedings of the Eighth GAMM Conference on Numerical Methods in Fluid Mechanics, 1990.

5. J. F. Thompson, *Program EAGLE User's Manual*, AFATL-TR-88-117, Vol. III, September, 1988.

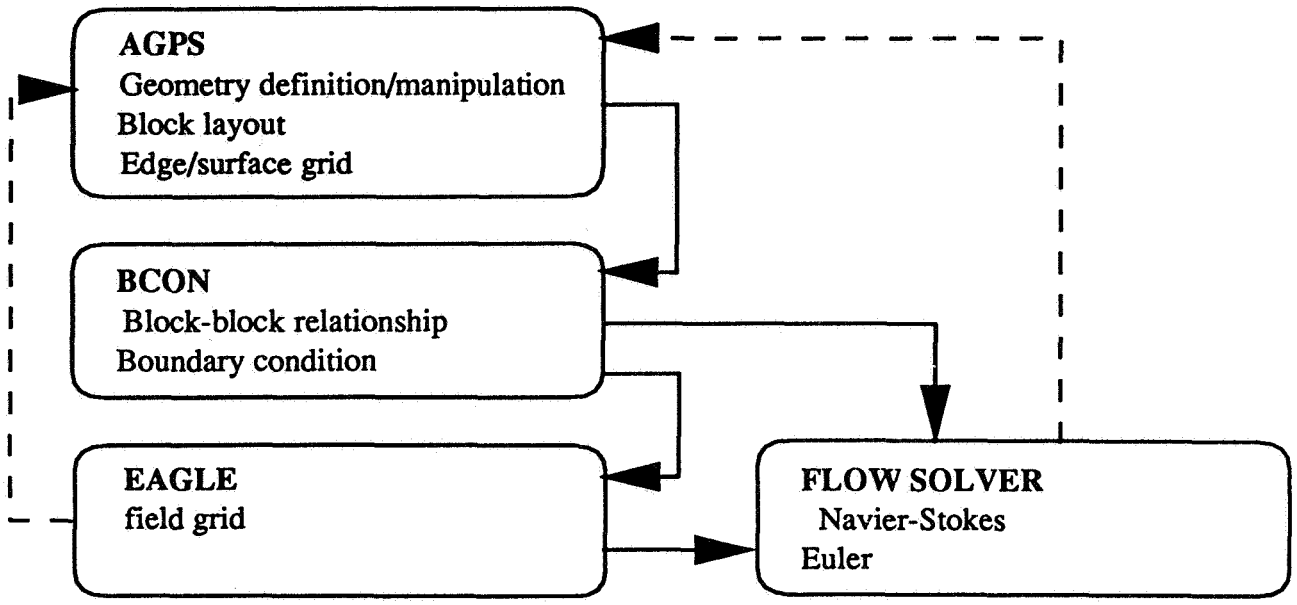


Figure 1. Multiblock grid generation process.

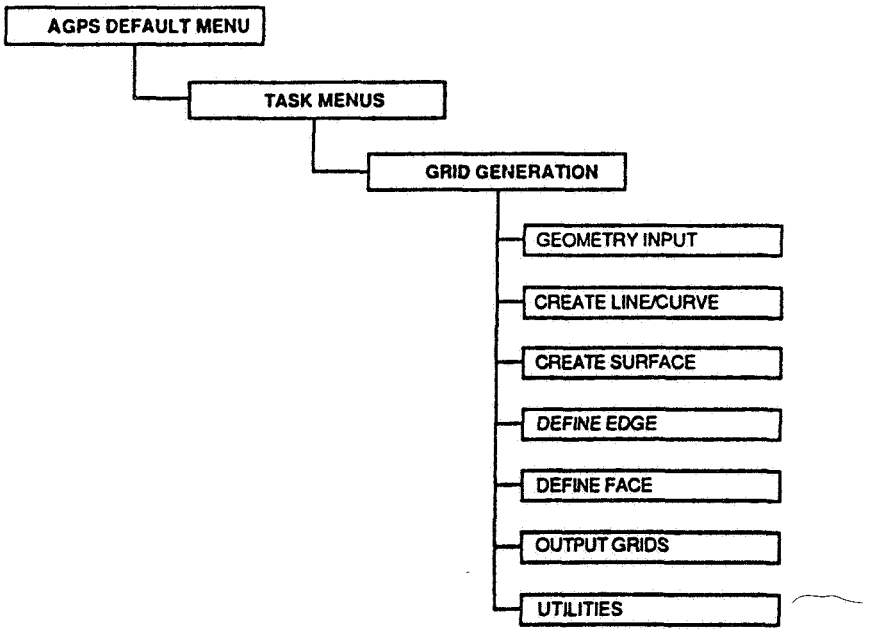


Figure 2. AGPS user-accessible menu.

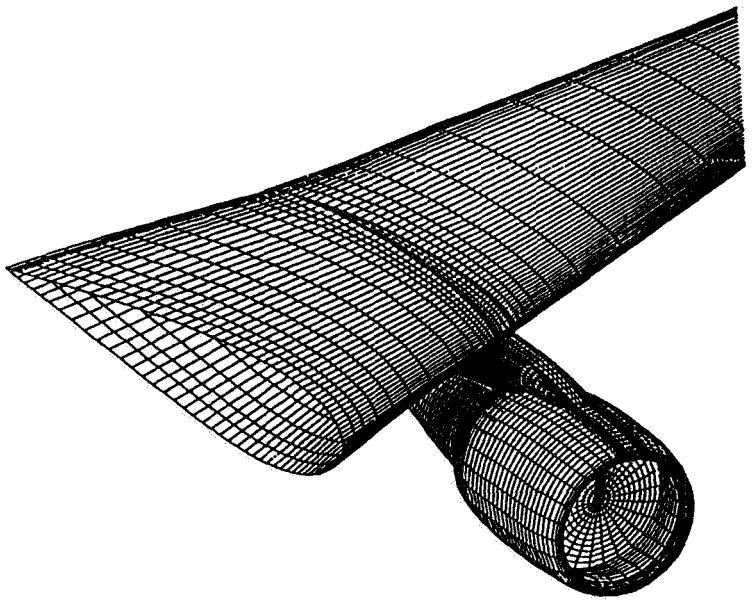


Figure 3. Surface grids for a wing/nacelle/strut.

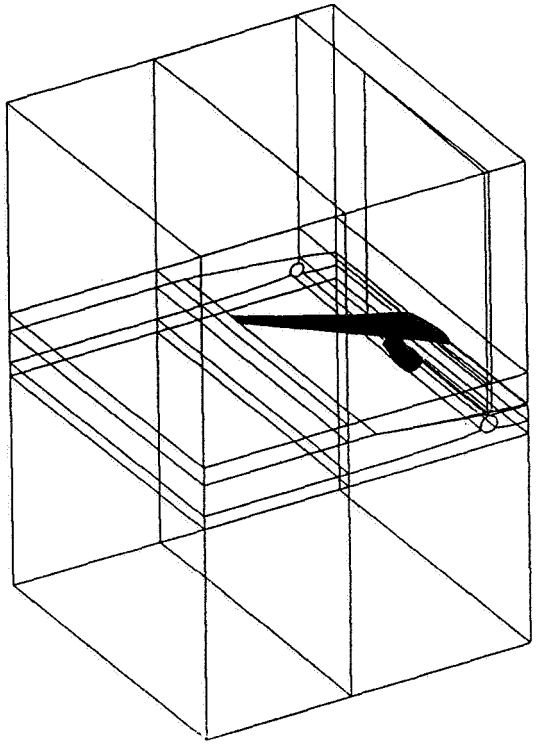


Figure 4. Blocks layout for a wing/nacelle/strut.

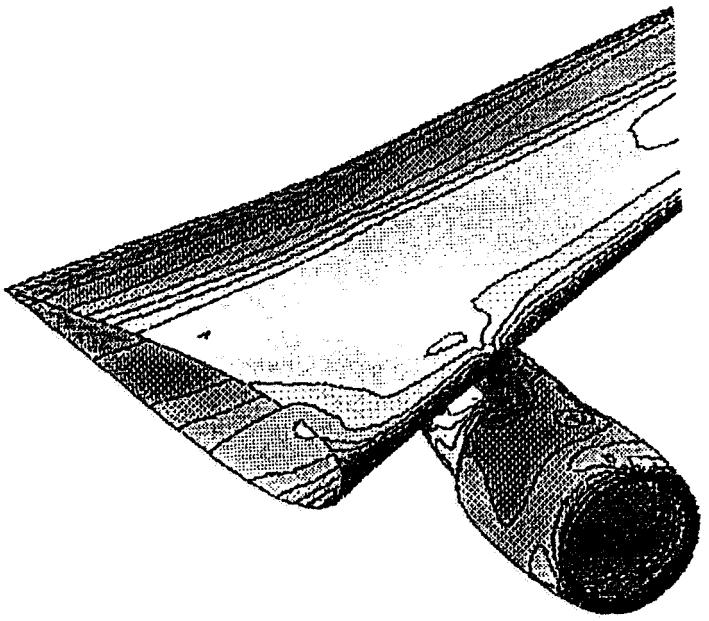


Figure 5. Isobar for a wing/nacelle/strut configuration at Mach=0.8, Alpha=2.0.

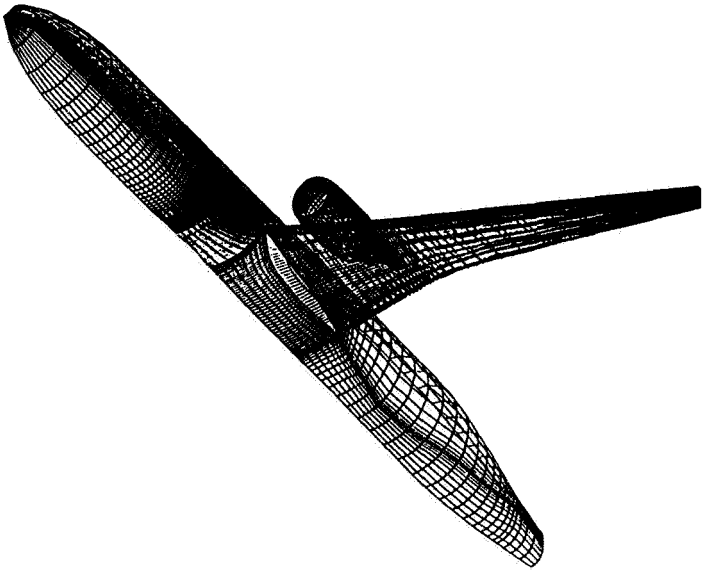


Figure 6. Surface grids for a body/wing/nacelle/strut.

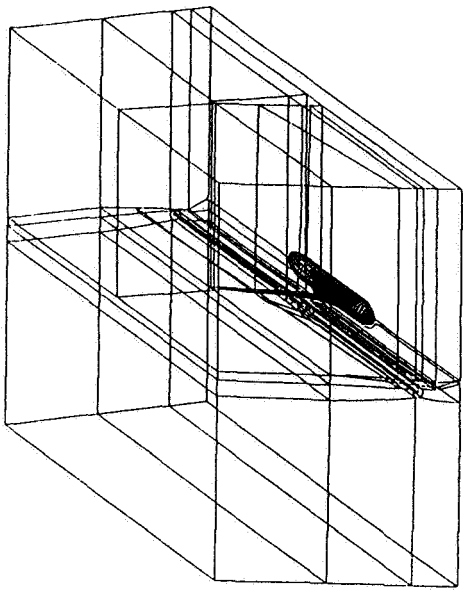


Figure 7. Blocks layout for a body/wing/nacelle/strut.

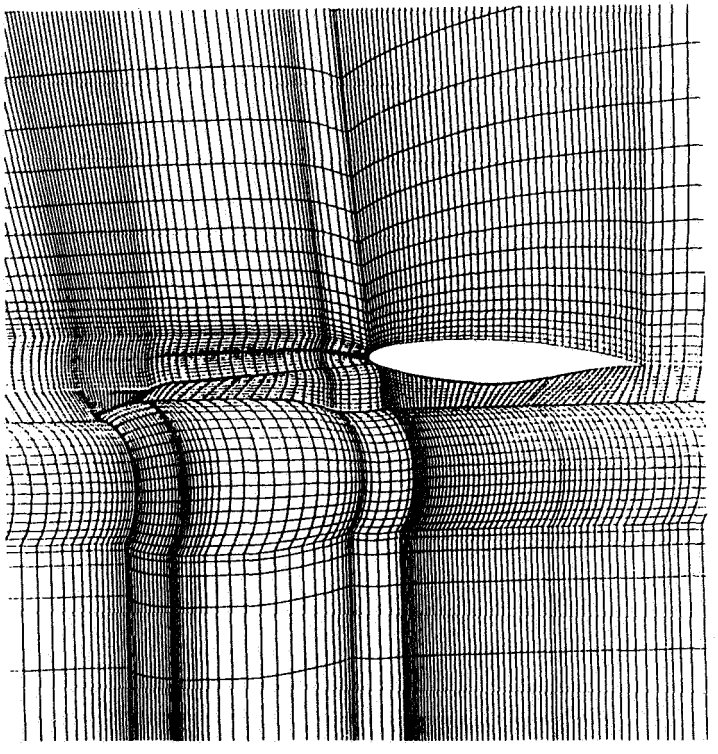


Figure 8. Constant K-plane field grid with fan cowl surface grids.

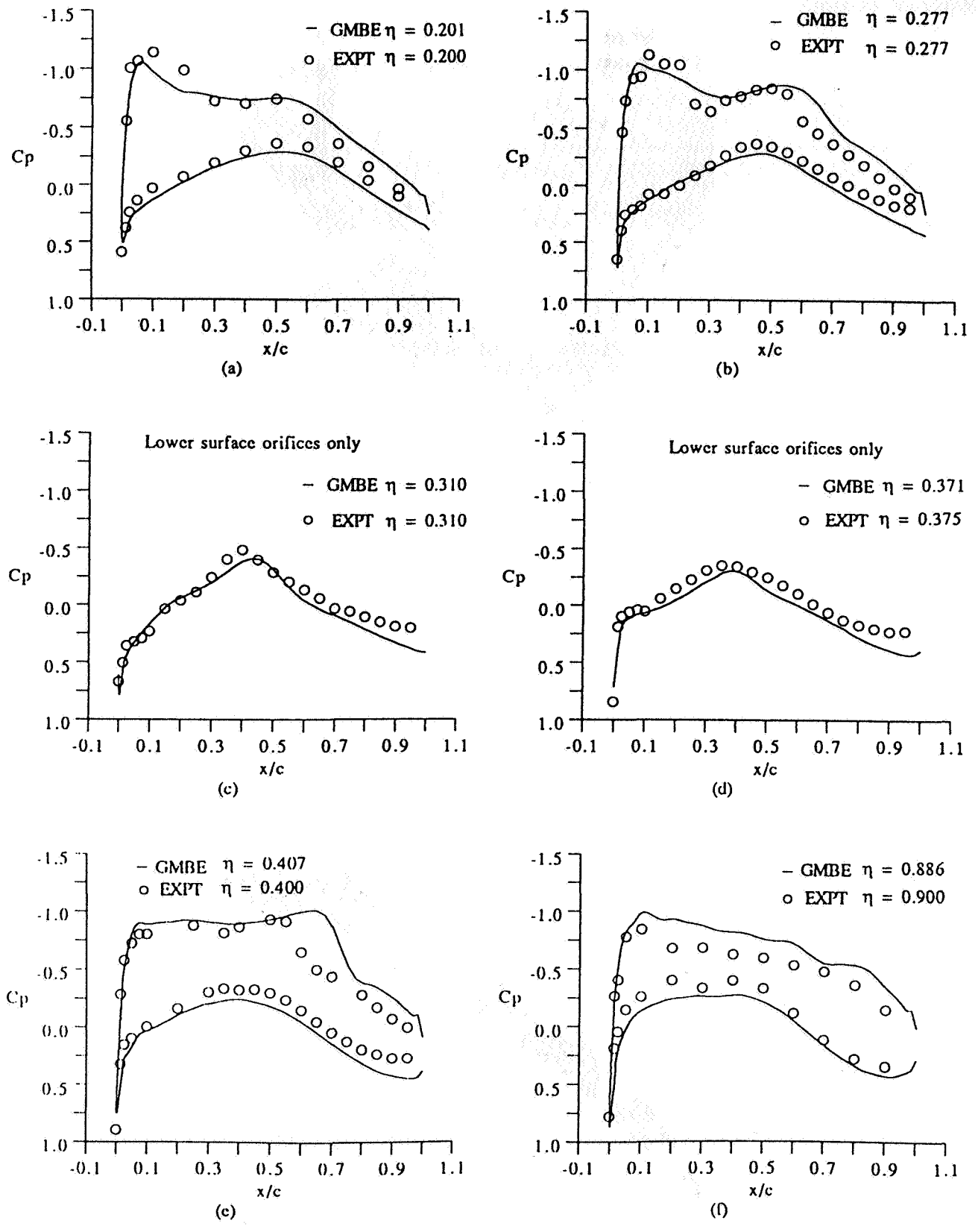


Figure 9. Comparison of computed wing Cp with wing-tunnel data. for nacelle installed at η=0.34, Mach=0.77 with Cl=0.55.

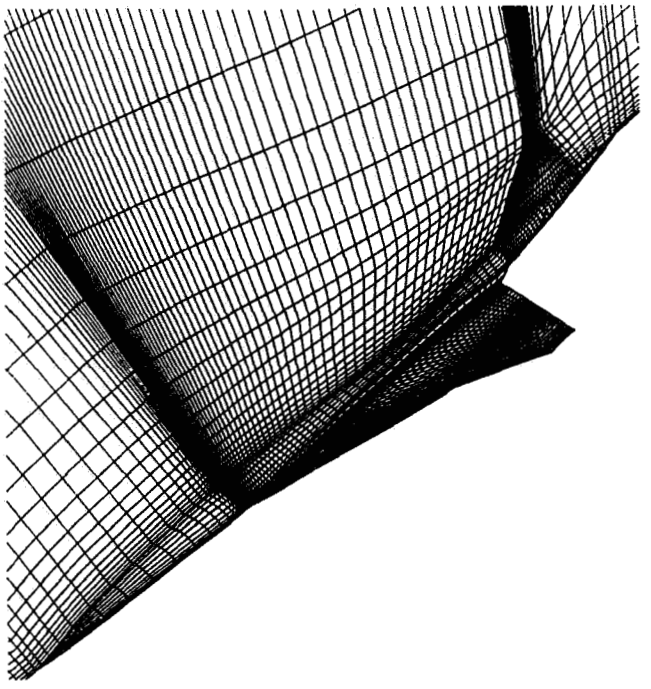


Figure 10. Surface grid and symmetry plane for HSCT cruise configuration.

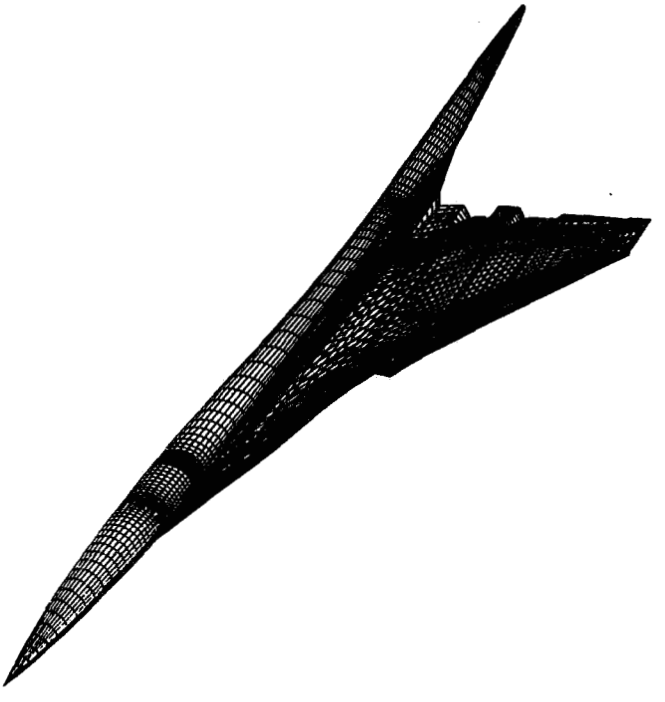


Figure 11. Surface grid for HSCT high lift configuration.

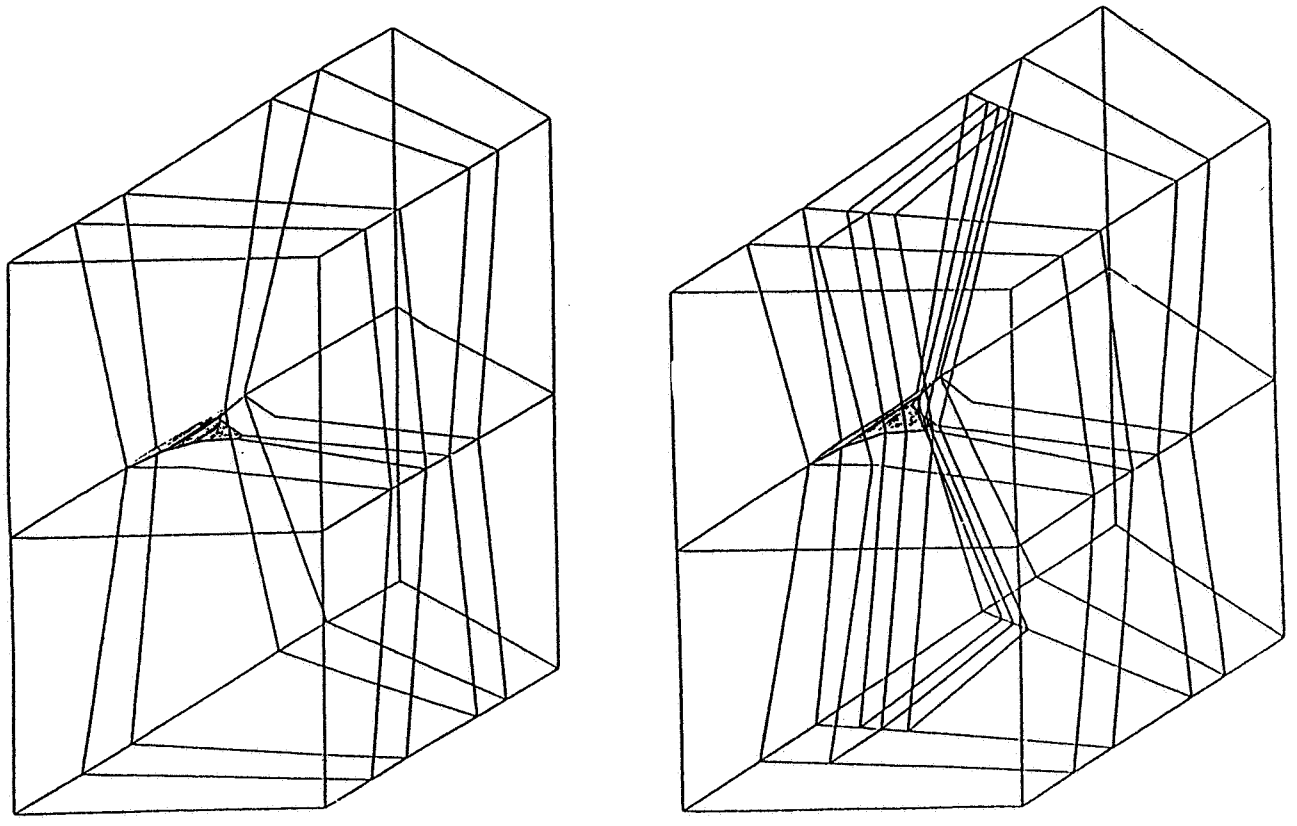


Figure 12. Blocks layout for HSCT cruise (10 blocks) and high lift (18 blocks) configurations.

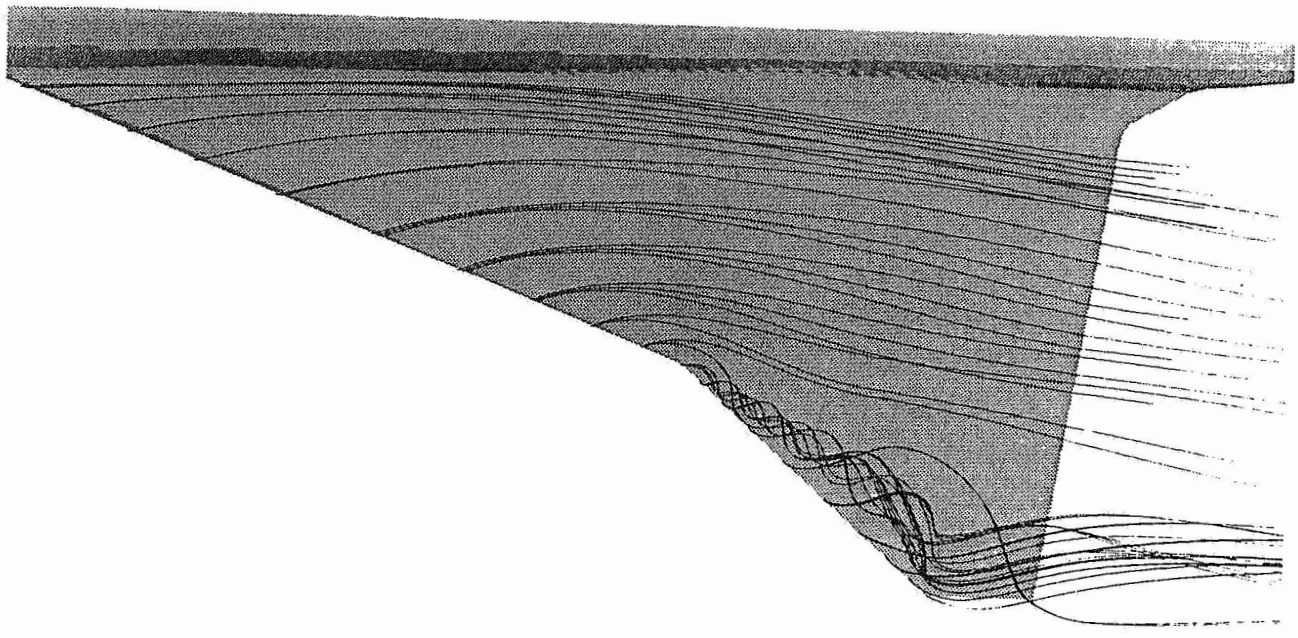


Figure 13a. Streamline traces for HSCT cruise configuration.

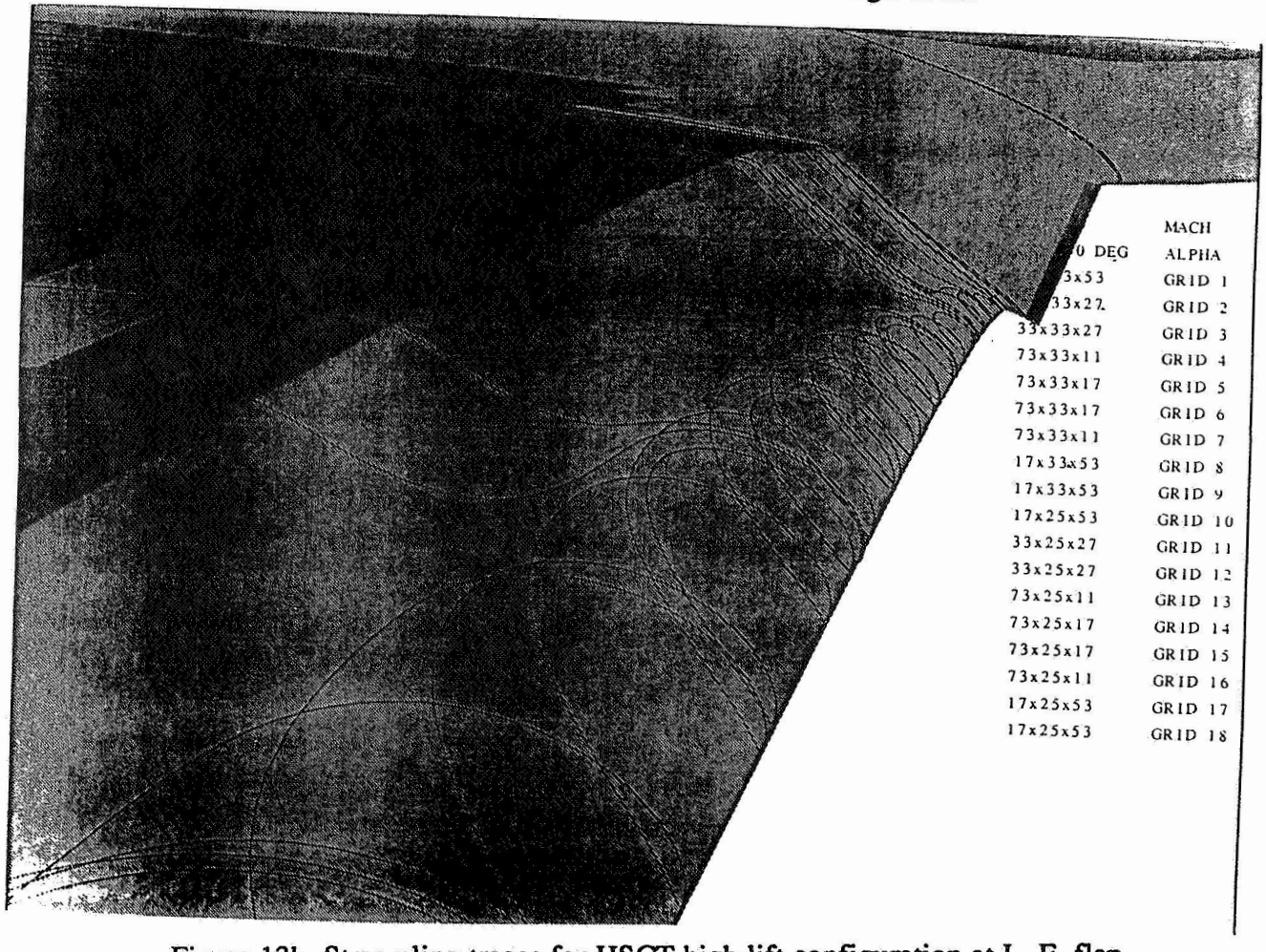


Figure 13b. Streamline traces for HSCT high lift configuration at L. E. flap.

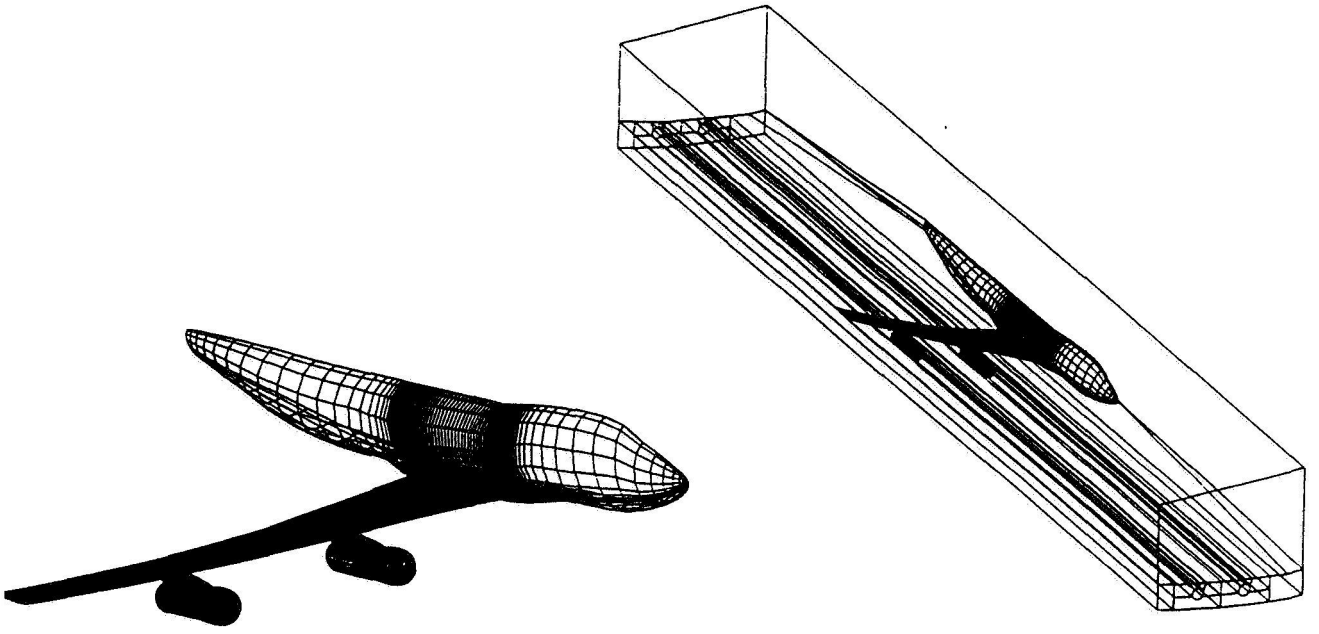


Figure 15. Near field blocks layout for four engines 747-200

Figure 14. Complete surface grids for 747-200.

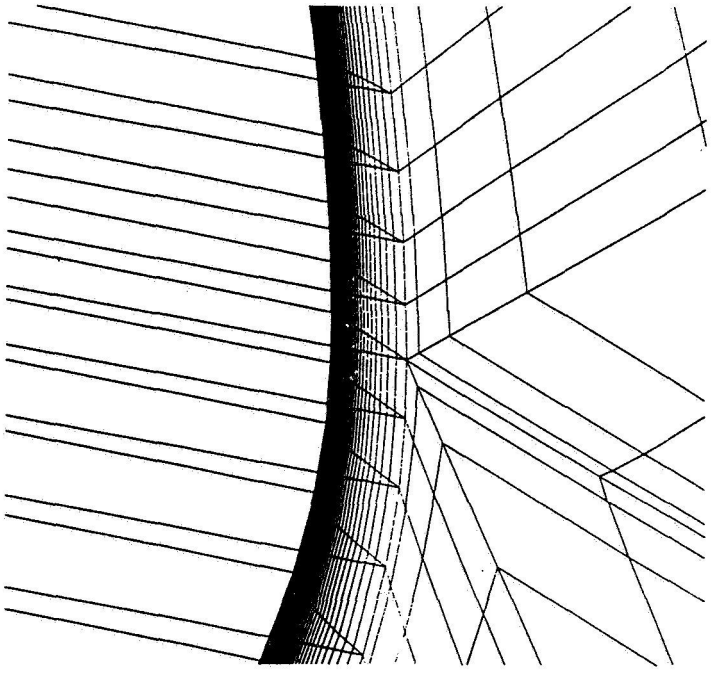


Figure 16. Viscous blocks near a wing/body junction.

INTERACTIVE SOLUTION-ADAPTIVE GRID GENERATION

Yung K. Choo
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

Todd L. Henderson*
University of Illinois
Urbana, Illinois 61801

SUMMARY

TURBO-AD is an interactive solution-adaptive grid generation program under development. The program combines an interactive algebraic grid generation technique and a solution-adaptive grid generation technique into a single interactive solution-adaptive grid generation package. The control point form uses a sparse collection of control points to algebraically generate a field grid. This technique provides local grid control capability and is well suited to interactive work due to its speed and efficiency. A mapping from the physical domain to a parametric domain was used to improve difficulties that had been encountered near outwardly concave boundaries in the control point technique. Therefore, all grid modifications are performed on a unit square in the parametric domain, and the new adapted grid in the parametric domain is then mapped back to the physical domain. The grid adaptation is achieved by first adapting the control points to a numerical solution in the parametric domain using control sources obtained from flow properties. Then a new modified grid is generated from the adapted control net. This solution-adaptive grid generation process is efficient because the number of control points is much less than the number of grid points and the generation of a new grid from the adapted control net is an efficient algebraic process. TURBO-AD provides the user with both local and global grid controls.

INTRODUCTION

Quality grids are needed to obtain accurate numerical simulations of complex flows. But generating quality grids is difficult for complex problems because the grids must be provided without a prior knowledge of the flow details. Precise local and global grid controls are often difficult even when details of flow are known. A grid adaptation process (refs. 1 to 3) that couples the grid with the evolving flow solution can provide a means of obtaining proper distribution of grid points. This adaptation process is combined with the control point form of algebraic grid generation (refs. 4 to 6)

*Summer student intern at NASA Lewis Research Center.

to improve the efficiency of the adaptation process and to provide explicit local/global grid control capability.

TURBO-AD under development starts with a given initial grid and a given initial solution, and it redistributes the grid points to provide denser grid in high gradient areas of a solution while having sparse grid in low gradient areas. New flow solution obtained from the adapted grid is able to capture more significant flow features in the high gradient areas. The program runs on the IRIS 4D series workstations interactively. Interactive process of the code allows the user to immediately see the results and modify the grid as desired along the way.

In TURBO-AD, all grid modifications are performed in a parametric domain. The grid adaptation is obtained through adapting the control points to a source distribution, which is defined by the gradients of a numerical flow solution. The new adapted grid is then computed from the new adapted control points. The control point adaptation offers advantages over the direct grid adaptation. It increases the efficiency of the solution-adaptive procedure because the number of control points is significantly fewer than the number of grid points. It also provides smooth grids resulting from the control point form. The solution adaptation of a grid does not always result in a smooth grid for many adaptation techniques. But, when a nonsmooth set of control points is used to generate a grid the resulting grid is much smoother. The grid modifications in the square parametric domain offers another advantage. It allows the control points and grid points on the boundary to freely move with very little alteration of the given boundary geometry. It also prevents grids from tangling that can happen in concave regions with the control point formulation since there are no concave boundaries in the parametric domain. This paper briefly discusses the control point form of algebraic grid generation and solution-adaptive grid generation using parametric mappings. Then interactive solution-adaptive grid generation procedure is discussed with an example.

ALGEBRAIC CONTROL POINT GRID GENERATION

The control point form of algebraic grid generation and its application are presented in references 4 to 6 in detail and are not repeated in this paper. This section only contains a brief sketch of the control point form. The control point array is a sparse grid-type arrangement of locations in space with an index for each direction. A two-dimensional control point array, $(C_{i,j})$ is shown in figure 1. As an algebraic method, the control point form provides explicit control of the grid shape and spacing through the movement of the control points.

A fundamental part of the control point formulation is the construction of curves. This construction represents algebraic coordinate generation in a single direction between two opposing boundaries. These boundaries are the fixed endpoints of the curves. The remaining control points are in the interior of the sequence and are used to control the shape of the curve.

From any one control point, the two neighboring points on each side form local segments within the entire piecewise linear curve. The local segments about each control point define a change in

direction. The rate of change is determined by the two linear connections attached to the given control point as the curve assumes the respective directions between each pair of control points.

A continuous direction field is obtained in a smooth manner by interpolation. The independent variable for the interpolation is the curve parameterization. The interpolated result defines the field of vectors that are tangent to the desired curve. Simply stated it is an interpolation of first parametric derivatives. This determines a smooth first derivative of the entire curve. The desired curve is then obtained by a parametric integration. The integration here is taken so that the curve connects the specified endpoints.

To obtain local grid controls, local interpolation functions are used. With local functions, the movement of a control point results in an alteration of the constructed curve that is restricted to a local region about the point. The remaining regions are unaltered. The simplest local interpolants are the piecewise linear functions that do not vanish over at most two intervals. Example curves constructed from left to right boundaries are $\mathbf{E}_2(\xi)$ and $\mathbf{E}_j(\xi)$ in figure 1. Similarly, a curve $F_i(\eta)$, can be constructed for the other direction from bottom to top boundaries. The tensor product form, $T(\xi,\eta)$, depends only upon $\mathbf{C}_{ij}$. The tenor product matches $\mathbf{E}_j$ or $\mathbf{F}_i$ at the extremities of i and j (i.e., at corner points in fig. 1 for instance).

When boundaries are to be specified, the corresponding data appear at the extremities of the values for ξ and η. Since the coordinate transformations are generally expressed in the form of a vector for the desired positions of all points in physical space, it is convenient to express the boundary specifications in terms of the position vector. Thus, the boundaries are denoted by $\mathbf{P}(1,\eta)$, $\mathbf{P}(N-1,\eta)$, $\mathbf{P}(\xi,1)$, and $\mathbf{P}(\xi,M-1)$. To include the boundaries, the multisurface transformation is performed again as above, but now with the actual boundaries inserted. This results in a modification of $\mathbf{T}$ for the ξ and η directions, respectively. In each such directional construction, the actual boundaries appear as end conditions for the corresponding variable while the remaining boundaries are generated by the control points. Thus, by subtracting $\mathbf{T}$ from the sum of both directional constructions, the actual boundaries become end conditions for each variable. This process follows a Boolean sum format and upon simplification becomes

$$\begin{aligned} Q(\xi,\eta) = T(\xi,\eta) &+ \gamma_1[1 - G_1(\xi)][P(1,\eta) - F_1(\eta)] \\ &+ \gamma_2 G_{N-1}(\xi)[P(N-1,\eta) - F_N(\eta)] \\ &+ \gamma_3[1 - H_1(\eta)][P(\xi,1) - E_1(\xi)] \\ &+ \gamma_4[H_{M-1}(\eta)][P(\xi,M-1) - E_M(\xi)] \end{aligned} \tag{1}$$

where G and H are integral of the local interpolation functions, and each of the four terms following the tensor product $\mathbf{T}(\xi,\eta)$ represents a transfinite conformity to a boundary when each boundary on-off switch, γ, is 1. By setting any γ_i to 0, the corresponding boundary becomes available for free form modeling by means of the control points. In the order of appearance, the boundaries are for $\xi = 1$, $\xi = N-1$, $\eta = 1$, and $\eta = M-1$. Further details of the control point form are presented in references 4 and 5.

SOLUTION-ADAPTIVE CONTROL POINT GENERATION

The solution-adaptation of grids is usually accomplished by directly adapting all the grid points. This can be time consuming for many adaptation algorithms. To increase the grid adaptation speed, the procedure described in reference 1 is used to adapt the control net instead of the grid. This new approach can easily be more than ten (10) times faster in two-dimensional than the usual direct grid adaptation procedure of reference 1. The new grid generation from the adapted control net is an efficient algebraic process.

The adaptation procedure using the control points is illustrated in figure 2. It starts from the initial control net in the physical space (x,y) as shown in figure 2(a). The position of the initial control points is determined by the distribution of the initial grid points, from which the control net is constructed by "attachment" (ref. 5), and the definition of the local interpolation functions. In order to control the grid structure locally, local interpolation functions are chosen for TURBO-AD. The interpolation functions are nonzero over one interval about the first and last partition points on the opposite boundaries, while the interpolation functions over the internal partition points are nonzero over two intervals. Resulting from this is the first fine control net meshes around the boundary as seen in figure 2(a). The procedure then defines a control net in the parametric space (s,t) as shown in figure 2(b). This control net is mapped into another parametric space (u,v) to establish a uniform control net as shown in figure 2(c). The initial grid is also mapped into the parametric space (u,v) using the same procedure. Then two control-net control sources are defined at every grid point in u and v directions. The sources are defined by a linear combination of the first and second derivatives of one or more monitor functions, ϕ_n, in each parametric direction as shown in equations (2a) and (2b)

$$\sigma^{u}_{k\ell} = \sum_{n=1}^{M} \left\{ w^{u}_{1n} \left| \frac{\partial \phi_n}{\partial u} \right| + w^{u}_{2n} \left| \frac{\partial^2 \phi_n}{\partial u^2} \right| \right\} \tag{2a}$$

$$\sigma^{v}_{k\ell} = \sum_{n=1}^{N} \left\{ w^{v}_{1n} \left| \frac{\partial \phi_n}{\partial v} \right| + w^{v}_{2n} \left| \frac{\partial^2 \phi_n}{\partial v^2} \right| \right\} \tag{2b}$$

where k and ℓ are the indices of the grid point where the source is located. The w's are weighting parameters that allow the user to control the contribution from each term. The monitor functions, ϕ, can be any flow property (i.e., density, pressure, Mach number, etc.).

The control point in the parametric domain, (u,v), is now modified using equations (3a) and (3b).

$$u'_{ij} = u_{ij} + \sum_{k,\ell} K_{ijk\ell}\sigma_{k\ell} \tag{3a}$$

$$v'_{ij} = v_{ij} + \sum_{k,\ell} K_{ijk\ell}\sigma_{k\ell} \tag{3b}$$

where $K_{ijk\ell}$ is the influence coefficients. The influence coefficient controls the effects of a source at a grid location, (k,ℓ), on a control point, (i,j). The (u',v') domain is normalized to go from zero to one. The modified new control net is shown in figure 2(d). The computation of the summation in equation (3) can be very costly if the influence coefficient K has to control the effects of the source on all grid points instead of the control points. This will be evident in a sample interactive case as the number of control points is significantly less than grid points. The coefficients are defined by equations (4a) and (4b).

$$K^u_{ijk\ell} = \frac{u_{ij} - u_{k\ell}}{d} \exp(-\alpha_u d) \tag{4a}$$

$$K^v_{ijk\ell} = \frac{v_{ij} - v_{k\ell}}{d} \exp(-\alpha_v d) \tag{4b}$$

where

$$d = \sqrt{(u_{ij} - u_{k\ell})^2 + (v_{ij} - v_{k\ell})^2}$$

and the α's are decay parameters for each of the parametric directions. The degree of adaptation is controlled by the user's input of the w's and α's.

In order to improve the efficiency, a cut-off value has been implemented such that the summation in equation (3) is terminated for sources where the $K_{ijk\ell}$'s decay below a user specified value. The input for this cut-off parameter, β, ranges between zero and one. At $\beta = 0$, the summation includes all points, and at $\beta = 1$, the summation would include no points.

Also, the sources are mirrored across each boundary to improve the orthogonality of the adapted control net to the boundary. Orthogonality of the control net lines can also be enforced by making the control net orthogonal to the boundary as described in reference 5.

The new control net points in the (u',v') domain have been repelled by the strong sources. This can be seen in figure 2(d). Now a new uniform control net is mapped back to the (s,t) domain using the mapping from the (u',v') domain to the (u,v) domain. This results in a clustering of points near the strong sources in the original parametric domain as shown in figure 2(e). At this point the new control net points are used to calculate a new parametric grid which is then mapped into the physical domain with the control net. The adapted control net in the physical domain is shown in figure 2(f).

INTERACTIVE SOLUTION-ADAPTIVE GRID GENERATION, SAMPLE CASE

The following example demonstrates main features available in TURBO-AD. The case to be considered is a ramp/expansion with an inflow Mach number of 2.0. The initial control net, grid, and Mach number distribution can be seen in figure 3. TURBO-AD uses the workstation mouse for the primary input. The right button either exits a current operation or displays the current menu. The menu selection is accomplished by highlighting the appropriate choice and then releasing the right button. The left and center mouse buttons are typically used for controls within the operations.

TURBO-AD can be executed by typing 'turboad.' The program then ask the user to type in the grid file name followed by the solution file name. The solution file can contain one to five scalar functions. The solution file for the sample case contains density, pressure, temperature, Mach number, and entropy, in that order. The solution file is optional. If a 'return' is entered at the request for a solution file, the user has access the interactive grid modification features discussed in reference 6 about TURBO-I.

Once TURBO-AD has been started, it will create the parametric mapping in memory and then display the initial grid and control net. The user can then bring up the home menu by pressing the right mouse button and holding it down. This menu is the top level or main menu. The user can select "MODIFY VIEW" which will take the user to the viewing control menu. Then by pressing the right mouse and holding it down, the user will see the "MODIFY VIEW" menu choices. The input function distribution can be displayed by selecting "Contour Function." This is a pull down menu, so the user should move the mouse to the right side of the selection. This will bring down a listing of functions that can be contoured. By selecting the fourth function the user can see the Mach number contour for the example case. It should be noted that the first function in the users's solution file is "function 1," the second is "function 2," etc. The "Scroll View" option allows the user to look at the control net alone, the grid alone, or many combinations. For this example, the user should see the control net, grid, and Mach number distribution if "Scroll View" is selected several times.

The user is now ready to prepare for the control net (CN) adaptation. The "ADAPTATION CONTROL" menu option can now be selected from the main menu or the "MODIFY VIEW" menu. Under the "ADAPTATION CONTROL" menu the user should use the mouse to pull down the "Active Functions" menu. This menu is to control which functions are used for adaptation. The default is for all functions to be turned off. The user can toggle the functions on or off by selecting the appropriate one. For this example the fourth function is selected. Please note that there is no correlation between the function being viewed and the functions selected under "Active Functions" for the CN adaptation.

The 'adapt.par' file contains the user definable parameters, w's and α's, shown in equations (2) and (4). The user is now ready to calculate the sources discussed in equations (2a) and (2b). This can be done by selecting the menu option "Calculate Source Terms." When the calculation of the sources is finished, the user will succeed in bringing up the menu. The calculation of the sources has now produced two new functions, σ^{ξ} and σ^{η}, in the users list of functions. At this point, the user could view the source distribution using the built in color contouring available under "MODIFY VIEW." If the user contours the sixth function for this example case, using the 'adapt.par'

file, the user should see the source distribution in the s direction shown in figure 4. On the workstation screen, the blue colors show the lower source strength and the red show the higher source strength. The locations where high values exist is where the adapted control net lines should cluster. The source distribution in the t direction has been set to zero using the input file in this example. The user can now return to the adaptation menus.

The control net adaptation can now be started. To do this the user should select "Adapt CN" entry in the "ADAPTATION CONTROL" menu. This operation will take about 20 to 90 sec depending on which IRIS 4D series workstation is being used. The completion of this operation will be obvious if the control net is displayed. The user will see the new adapted control net pop up. The new control net should look like the one shown in figure 5. The user now has two choices, is the control net acceptable, or should the user select "restore CN and Grid" to bring back the initial control net? The control net that now exists will capture the strong shock, but may not capture the reflected weak shock adequately.

To capture the weaker reflected shock, the user recalculates the grid with the new control net by selecting the "ReCalculate Grid" entry, and then selects "Redefine BLOCK" entry under the HOME menu, allowing grid adaptation only in the weaker shock region without the effect of the stronger shock. When the "redefine BLOCK" entry is selected, TURBO-AD will instruct the user to select the i_{min}, j_{min} point using the left mouse button and the i_{max}, j_{max} point using the center button. The left button should be used to select a point near the expansion corner. The user can determine what region is selected by the color changes of the grid. The region can be changed over and over while in this option. To exit, the user should press the right mouse button. A new control net will be created and attached to the new local region as shown in figure 6(a). At this time the user should return to the adaptation menu and select "Calculate Sources" entry. This will only calculate sources in the new region as shown in figure 6(b). The user can then use "Adapt CN" and "ReCalculate Grid" to adapt to the weaker shock. The process should be repeated two or three times starting with the calculation of the sources. The resulting grid should look similar to the one shown in figure 7. The user should note that there are discontinuities along the local block boundaries. This will be smoothed out when the user returns to "Redefine BLOCK" and chooses the entire grid and does a grid calculation. The resulting grid is shown in figure 8.

Further improvements can still be made by using local control point manipulations as discussed in reference 6. The final adapted control net, grid, and the new solution are shown in figure 9. The new solution was obtained using the adapted grid from a separate Euler code. The strong shocks have both become crisp and discernable.

CONCLUDING REMARKS

Two technologies are combined to develop an interactive solution-adaptive grid generation program, TURBO-AD. One is the solution-adaptation technique that uses the parametric mapping with control sources that are derived from the flow solutions. The other is the control point form of

algebraic grid generation technique that provides precise local control. The control point form technique is efficient as an algebraic method. Significant computational savings are attained by adapting the control net to solutions instead of the usual direct grid adaptation to solutions. Further savings in the computational time is achieved in TURBO-AD by cutting off the computation of the influence function K at grid points where K decays below a user specified value.

TURBO-AD is being developed on the IRIS 4-D series workstations. It can display the grid, solution contour, and the control net separately or can overlay one over the other for examination. With the improvement in computational speed, two-dimensional solution-adaptive grid generation is now a viable option on graphics workstations. As shown with an illustration, TURBO-AD has global grid control over the entire domain, control over subregions, as well as local grid control capability.

TURBO-AD has had limited testing. Further testing and enhancements are needed to improve robustness and efficiency of the code. Also needed are guidelines for the selection of adaptation parameters.

REFERENCES

1. Lee, K.D.; Loellbach, J.M.; and Pierce, T.R.: Solution Adaptive Grid Generation Using a Parametric Mapping. Numerical Grid Generation in Computational Fluid Mechanics '88: Proceedings of the Second International Conference, S. Sengupta, et al., eds., Pineridge Press, Ltd, Swansea, Wales, 1988, pp. 455-464.

2. Lee, K.D.; and Loellbach, J.M.: Geometry-Adaptive Surface Grid Generation Using a Parametric Projection. J. Aircraft, vol. 26, Feb. 1989, pp. 162-167.

3. Lee, K.D.; Henderson, T.L.; and Choo, Y.K.: Grid Quality Improvement by a Grid Adaptation Technique. Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, A.S.-Arcilla, et al., eds., North-Holland, 1991, pp. 597-606.

4. Eiseman, P.R.: A control Point Form of Algebraic Grid Generation. Int. J. Num. Methods Fluids, vol. 8, Oct. 1988, pp. 1165-1181.

5. Eiseman, P.R.; Snyder, A.; Choo, Y.K.: Dynamics of Local Grid Manipulations for Internal Flow Problems, Computational Fluid Dynamics Symposium on Aeropropulsion, NASA CP-10045, pp. 30-26, 1990.

6. Choo, Y.K.; Miller, D.P.; and Reno, C.: Implementation of Control Point form of Algebraic Grid Generation Technique, Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, ed. by A.S.-Arcilla, et al. North-Holland, pp. 331-341, 1991.

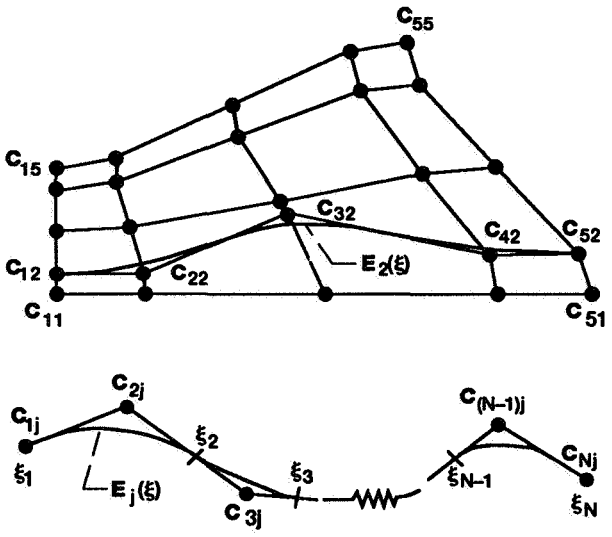
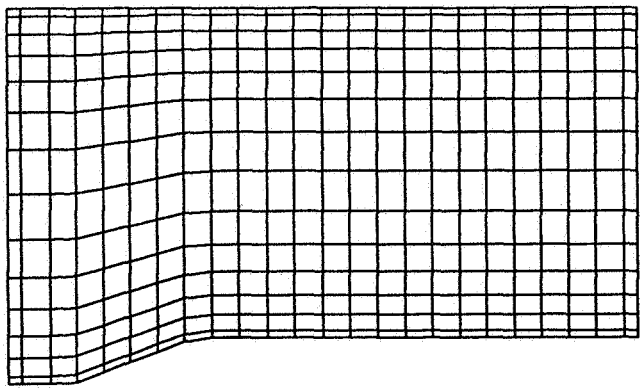
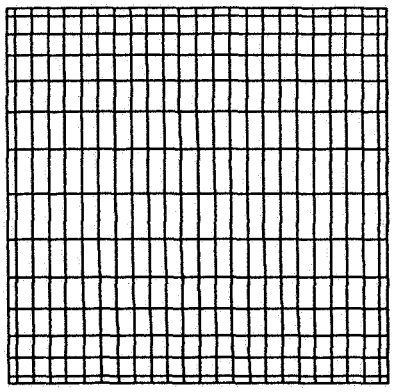


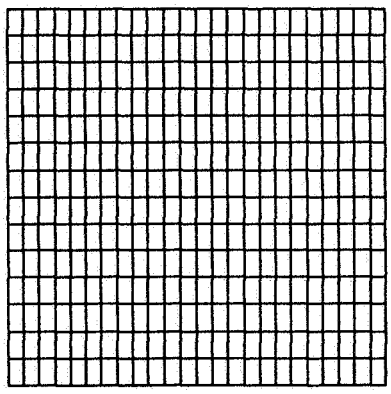
Figure 1.—Control net and construction of a curve.



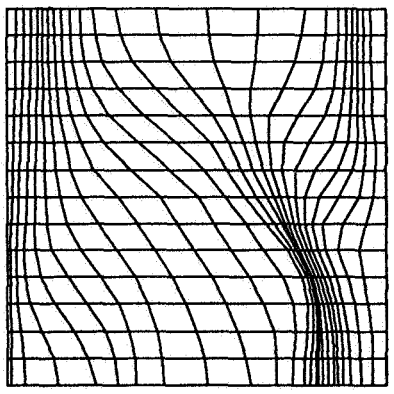
(a) Initial control net in physical space (x, y).



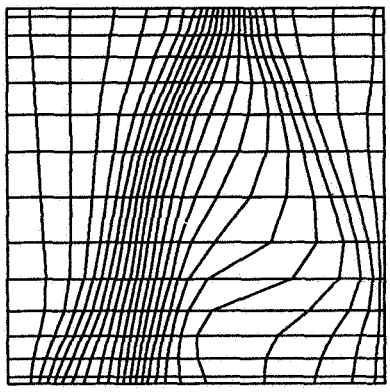
(b) Initial control net in parametric space (s, t).



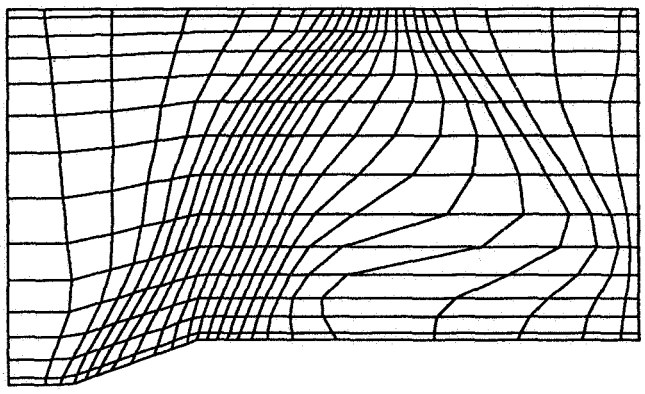
(c) Uniform control net in parametric space (u, v).



(d) Adapted control net in parametric space (u', v').

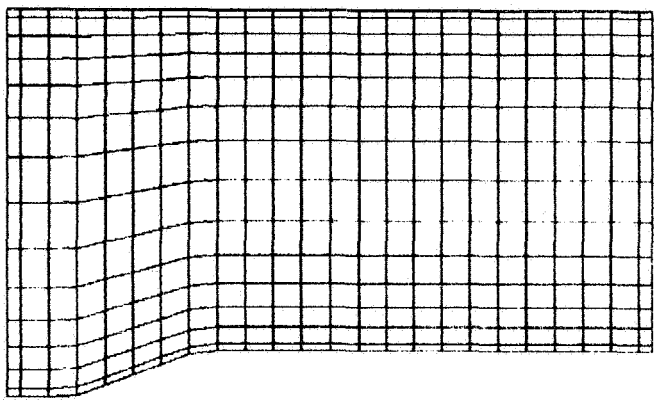


(e) Adapted control net in parametric space (s, t).

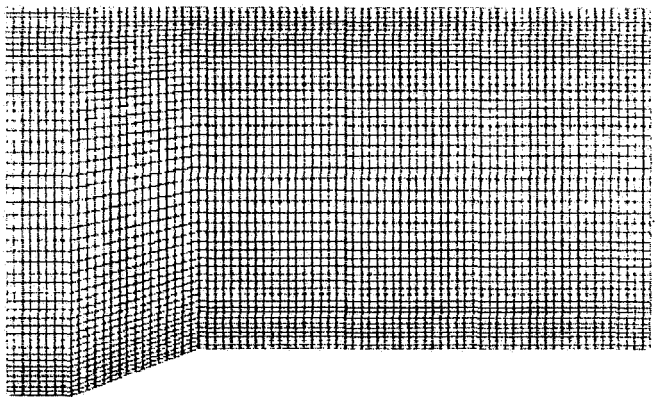


(f) Adapted control net in physical space (x, y).

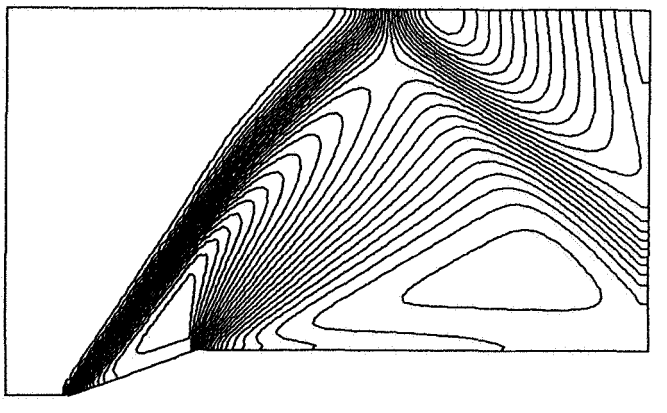
Figure 2.—The adaptation process for the control net.



(a) Control net.



(b) Grid.



(c) Mach number distribution.

Figure 3.—Initial results.

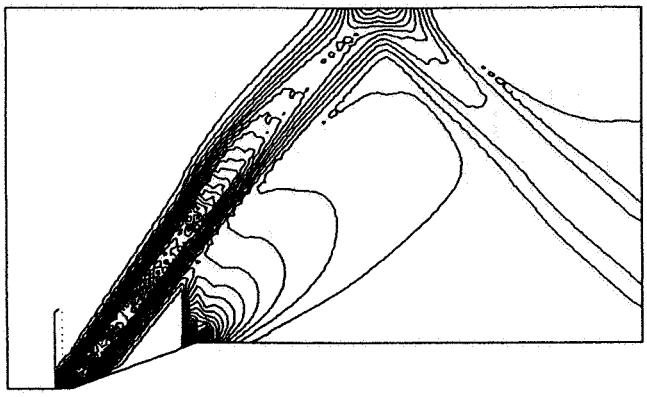


Figure 4.—Source distribution in S-direction.

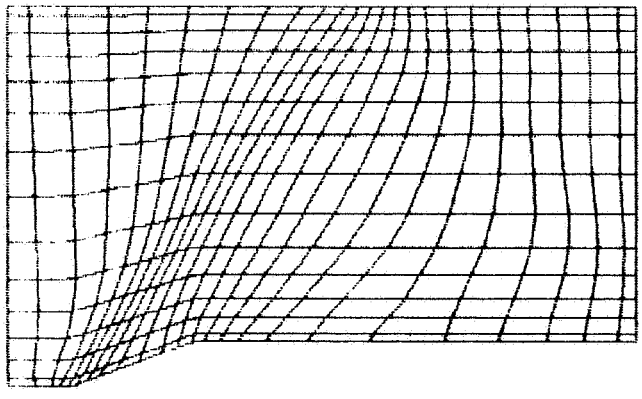
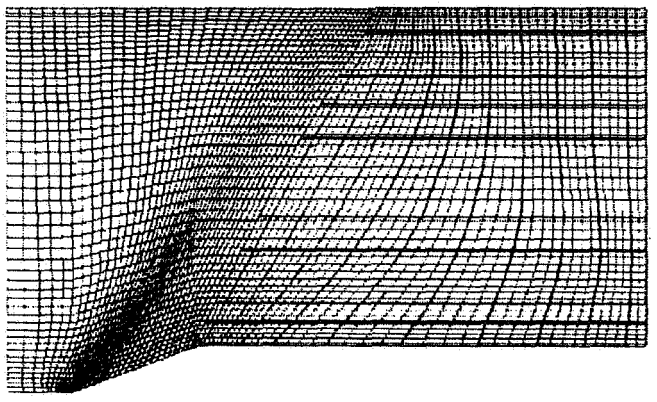
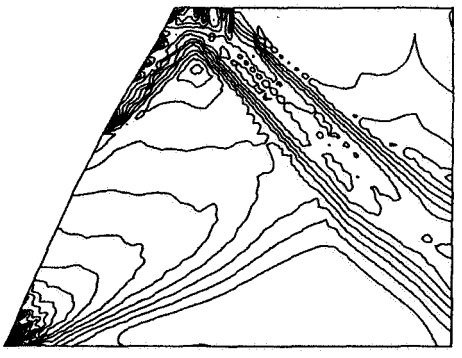


Figure 5.—Adapted control net.



(a) Sub-block control net.



(b) Local source distribution.

Figure 6.—Sub-block used for local adaptation to resolve the weak reflected shock.

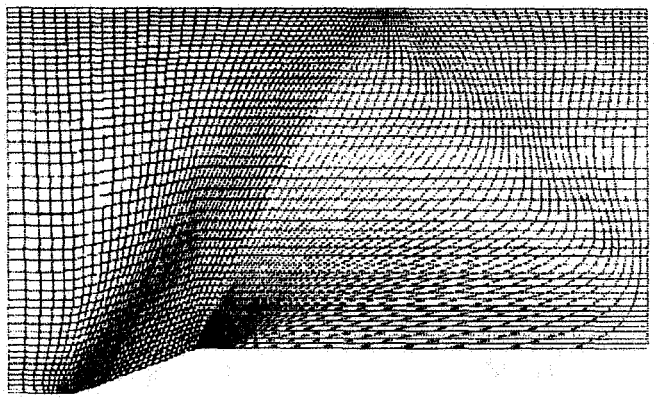


Figure 7.—Solution adapted grid after local adaptation.

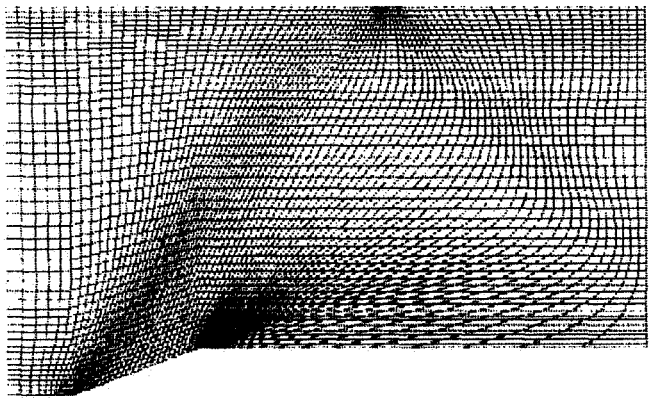
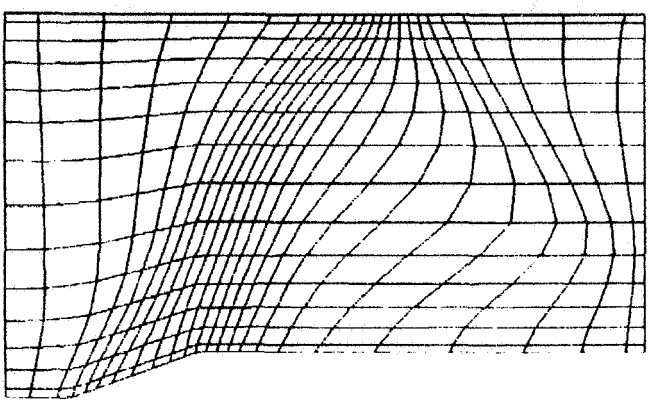
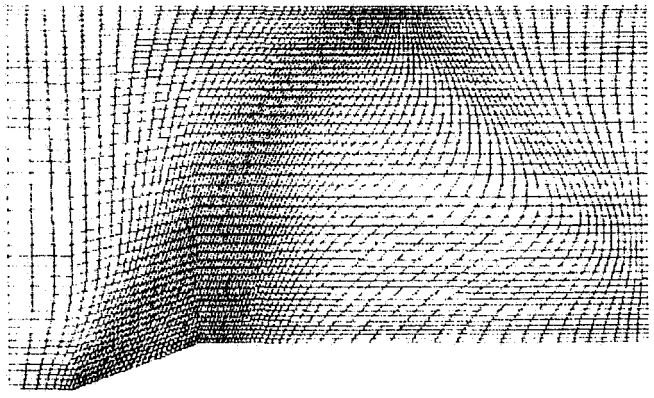


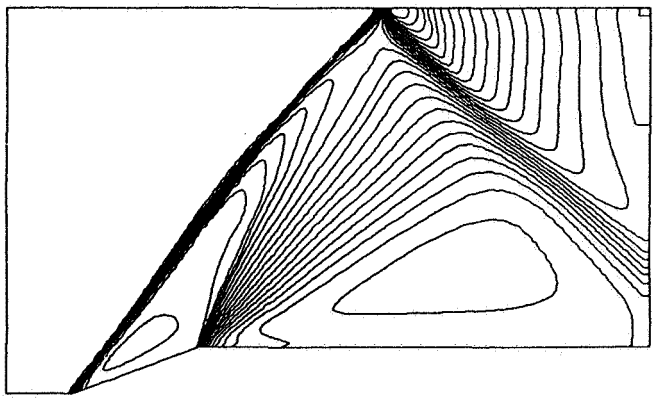
Figure 8.—Solution adapted grid after recalculation of grid using entire domain.



(a) Control net.



(b) Grid.



(c) Mach number distribution.

Figure 9.—Final results.

MAG3D and Its Application to Internal Flowfield Analysis

K. D. Lee[1], T. L. Henderson[2]
University of Illinois
Urbana, Illinois 61801

Y. K. Choo[3]
NASA Lewis Research Center
Cleveland, Ohio 44135

SUMMARY

MAG3D (Multi-block Adaptive Grid, 3D) is a three-dimensional solution-adaptive grid generation code which redistributes grid points to improve the accuracy of a flow solution without increasing the number of grid points. The code is applicable to structured grids with a multi-block topology. It is independent of the original grid generator and the flow solver. The code uses the coordinates of an initial grid and the flow solution on that grid to produce the coordinates of a solution-adapted grid and the flow solution interpolated onto the new grid. MAG3D uses a numerical mapping and potential theory to modify the grid distribution based on properties of the flow solution on the initial grid. In this paper, the adaptation technique is discussed, and the capability of MAG3D is demonstrated with several internal flow examples. Advantages of using solution-adaptive grids are also shown by comparing flow solutions on adapted grids with those on initial grids.

INTRODUCTION

The need for quality grids is increasing as the complexity in geometry and flow physics increases. The quality of a flow solution can be greatly enhanced by using a grid adapted to geometrical and physical features of the flow problem. Although some desirable grid characteristics can be accommodated in the initial grid generation process, it is difficult to generate a quality grid in a single step. In general, the grid must be supplied without *a priori* knowledge of the flow solution. This introduces the need for a solution-adaptive grid technique which couples the grid generation process with the flow solution process. Two approaches are available in generating solution-adaptive grids. One

---

[1]Associate Professor, Department of Aeronautical and Astronautical Engineering
[2]Graduate Research Assistant, Department of Mechanical Engineering
[3]Research Scientist, Internal Fluid Mechanics Division

approach is grid refinement which divides grid cells in regions of rapidly varying flow conditions into smaller cells, increasing the number of grid points. The other is grid redistribution which relocates grid points in an effort to distribute the truncation error more uniformly. Each method has advantages and disadvantages. The former seems more flexible but requires a provision in the flow solver to handle the grid change. The latter does not require any change to the flow solver but may degrade the grid quality. The MAG3D code is based on the grid redistribution technique.

The primary task of the solution-adaptive grid is to reduce the truncation error for improved solution accuracy. This is often achieved by controlling the cell size. However, there are other grid properties which strongly influence flow calculations. Therefore, a successful solution-adaptive grid should satisfy the following grid quality issues. It should not generate highly skewed cells, sharply kinked gridlines, or large cell aspect ratios, which tend to deteriorate the convergence rate and solution accuracy. Severe grid stretching and sudden change in cell size are also not desirable because they reduce the order of truncation. Better resolution can also be achieved if gridlines are aligned with flow characteristic lines such as a shock, a wake, a vortex core, and a contact discontinuity. Grid orthogonality is particularly important near the solid boundary for an efficient implementation of boundary conditions. In addition, there are other concerns: the grid adaptation process should be inexpensive compared to the flow calculation; it should be robust enough to handle different geometries and flow conditions; and, it should provide simple and predictable grid control options for the degree of adaptation and for the choice of grid quality.

METHOD

The adaptation method used in MAG3D is based on a numerical mapping and potential theory. It starts by defining numerical mapping functions of the initial grid into a parametric domain. The solution-adaptive grid generation is then a procedure of altering the mapping functions so that the flow gradients in the parametric domain become as uniform as possible. The mapping functions are modified by the influence of grid control sources whose strengths are defined from the distribution of flow variables on the initial grid. In two dimensions, the distribution of a flow variable over a grid forms a three-dimensional solution surface. Hence, the solution-adaptive grid generation in two dimensions can be compared to the surface grid generation on a three-dimensional surface patch. The difference lies in the source definition; the latter from the geometry and the former from the flow solution. Earlier, this technique was developed for geometry-adaptive surface grid generation [1]. It was later applied to solution-adaptive grid generation in two dimensions [2]. The present MAG3D code is an extension of the technique to three dimensions.

The present method of generating a solution-adaptive grid is as follows. First, the parametric representation of the initial grid is obtained by normalizing its computational coordinates, or indices. The result is a uniformly discretized cube in parametric coordinates, s_i, where i = 1,2,3 for each parametric direction. This mapping contains information about the initial grid which may already be tailored to the geometry and predicted flow

characteristics. Adaptation to the flow solution is performed by another mapping, which modifies the mapping functions using the influence of grid control sources. Grid control sources are defined separately for each parametric coordinate at the center of each cell in the parametric space. Source strengths are defined to reflect local solution characteristics on the initial grid. A monitor function, ϕ, is defined to be a linear combination of flow variables. In MAG3D, the source strengths are defined as a linear combination of the monitor function and its first- and second- derivative in each parametric coordinate. That is,

$$\sigma_i(Q) = w_{0_i}|\phi| + w_{1_i}\left|\frac{\partial\phi}{\partial s_i}\right| + w_{2_i}\left|\frac{\partial^2\phi}{\partial s_i^2}\right| \qquad (1)$$

where Q stands for the source point. The w's are input parameters which allow for different weights to be placed on the various derivatives of ϕ in each parametric coordinate.

The second mapping includes the effects of all sources at every grid point in the parametric space. The source effects are defined as displacements, creating a modified set of parametric coordinates, $\hat{s}_i$,

$$\hat{s}_i(P) = s_i(P) + \sum_Q K_i(P,Q)\ \sigma_i(Q) \qquad (2)$$

where $K_i(P,Q)$ is the influence coefficient that determines the effect of a grid control source at a source point, Q, to a field point, P. It is defined from the potential theory as a function of the distance between the source point and the field point, given by

$$K_i(P,Q) = \left\{\sum_{i=1}^{3}\alpha_i(s_i(P)-s_i(Q))^2\right\}^{-1/2} \qquad (3)$$

where α_i is the weighting parameter in each parametric coordinate. The choice of weighting parameters determines the decay characteristics of the influence coefficient. The combination of the weights in (1) and the parameters in (3) defines the degree and character of the adaptation. The mapping also accounts for the effects of image sources which are defined as a reflection of the field sources across the domain boundaries. The use of image sources enforces the Newmann condition along a boundary, making the gridlines nearly orthogonal at the boundary. In a multi-block application, the image sources are obtained from the neighboring block in order to maintain the grid continuity across the block boundary.

The second mapping repels grid points away from strong sources. Therefore, cells containing strong sources become larger than those with weaker sources. Next, the modified parametric space is uniformly discretized. The points of this new discretization are then mapped back to the physical space by interpolating their locations from the known locations of points in the initial mapping. Since the modified parametric space is expanded

in regions of strong sources, the uniform rediscretization produces relatively fine grid spacing in these regions. In addition to the basic technology discussed here, MAG3D is equipped with many cost-saving features such as influence cut-off distance, group source influence, coarse grid adaptation, and multi-level searching algorithm.

The present adaptation technique has many advantages which stem from the use of parametric mapping and grid control source. For instance, basic characteristics of the initial grid can be retained while the grid is adapted to the flow solution. Linear potential theory allows for combined grid controls based on the superposition principle. The grid can be adapted to multiple flow quantities through a combined source definition. The adaptation can be performed only in a partial domain by defining the source strengths only in the region. The source formulation guarantees smoothness in the resulting grid, although sources are distributed irregularly. The method is not restricted to specific grid generators or flow solvers, and hence can be incorporated with many available grid and flow codes without modifications. The adaptation process can be repeated if a satisfactory result is not achieved after a single application.

DEMONSTRATIONS

The capability and merits of MAG3D are demonstrated with three flow problems shown in Figure 1: a supersonic corner flow, a supersonic injector nozzle flow, and a supersonic flow around a blunt fin mounted on flat plate. All flowfield calculations are performed using the CFL3D computer code acquired from the NASA Langley Research Center [3]. It solves the thin-layer, laminar, time-dependent, compressible, Reynolds-averaged Navier-Stokes equations using finite-volume, upwind-inviscid and centered-viscous discretization, factored-implicit integration, and multi-grid acceleration. The test cases are selected because the flowfields contain many flow features which are suitable for grid adaptation, such as shocks, mixing layers, shock-shock and shock-boundary-layer interactions. The sensitivity of these flow features to grid quality demonstrates the effectiveness of the solution-adaptive grid in improving solution accuracy. In all three cases, the grid control sources are defined as a combination of gradients of the Mach number and pressure on the initial grid. This choice for the monitor function performs well in adapting the grid to both shocks and boundary layers.

The first example is the supersonic, symmetric corner flow bounded by the intersection of two wedges with equal angles of 9.5 degrees as shown in Figure 1 [4]. A no-slip condition is imposed on the wedge surfaces on the bottom and right sides and a symmetric condition is used on the other two sides. Laminar flow is considered, with a free stream Mach number of 3.0 and a Reynolds number of 3.07×10^6 per unit length. The calculations are performed on 33x65x65 grids. In order to observe the capability of MAG3D, the adaptations are performed with directional applications by the choice of w's in (1). The grids and flow solutions are shown in Figures 2 to 5 for the initial grid and 1D, 2D, and 3D adaptations, respectively. In grid pictures, every other gridline is shown for clarity. Here,

the 1D adaptation means that the grid is adapted only in one parametric coordinate direction. The 2D adaptation stands for the adaptation in two coordinate directions. However, the source influences are always accounted for in all three directions. These tests demonstrate the controllability of MAG3D. More global adaptation is possible by changing α_i in (3). It is noteworthy that the gridlines become near orthogonal to the boundaries due to the use of image sources. As expected, the adapted grids produce a much better shock structure and a better resolution for the shock-induced recirculation zone in the boundary layer.

The second problem is a supersonic mixing nozzle flow where a sonic transverse jet is injected into a Mach 4 freestream [5]. Only the perfect gas model is considered; no chemistry is included. The geometry and the flow conditions are shown in Figure 1. No-slip and adiabatic conditions are imposed on the top wall where the jet is located. Symmetric boundary conditions are used at the bottom wall and both side walls. At the exit, the no flow-gradient condition is applied. The grid size is 49x33x33. The initial grid is already clustered to resolve the jet and the wall boundary layer. The grid and the flow solutions are compared for the initial and solution-adapted grids in Figures 6 and 7, respectively. Again, only every other line is shown for clarity. It can be seen that the gridlines of the adapted grid follow the bow shock and the mixing layer. Improved resolution can be observed with the solution-adaptive grid: the bow shock becomes sharper, and the temperature mixing layer is captured more clearly.

The third example is the supersonic flow field around a blunt fin mounted on a flat plate as shown in Figure 1 [6]. This flow contains a complex three-dimensional shock-wave and boundary-layer interaction. The free stream conditions are Mach number of 2.95 and Reynolds number of 3 million based on the fin width. The grid size is 65x33x33. The initial grid is clustered in the boundary layers along the flat plate and the fin surface. The boundary layer on the plate is assumed to start at the upstream far-field boundary. Figures 8 and 9 show the grids and flow solutions for the initial and adapted grid cases respectively. The solution-adaptive grid shows the grid clustering in the shock and shear layer regions. Therefore, sharper shocks and shear layers can be observed in front of the blunt fin.

In order to compare the convergence rate, flow calculations on the solution-adaptive grids are started from the free stream condition instead of the interpolated solutions from MAG3D. In all three cases, the solution-adaptive grids do not significantly deteriorate the convergence rate of the flow solver. The solution-adaptive grids demonstrate a significant improvement in the resolution of major flow features obtained with the initial grids. The solution-adaptive grid not only improves the sharpness of the flow solution, but also captures important secondary flow structures that are not presented in the initial solutions.

CONCLUDING REMARKS

MAG3D is shown as an effective tool to generate solution-adaptive grids for various three-dimensional flow problems. The presented adaptation technique is an efficient

method to improve certain grid properties. The method is convenient because it does not require any modifications to grid generators and flow solvers. Also, it can preserve the basic grid characteristics of the initial grid, which is often desirable in practical applications. The degree and locality of adaptation can be controlled in each direction separately through input parameters. Repeated adaptation can be performed when single adaptation is not satisfactory. The presented examples show that the solution-adapted grid significantly improves the flow resolution with little penalty in convergence and stability in the flow calculation.

In some applications, solution-adaptive grids may possess undesirable grid properties such as severe grid distortion and sudden grid stretching. Therefore, a grid-quality assessment and manipulation process will be incorporated into MAG3D to help prevent undesirable grid properties. The same adaptation technique can be utilized for the grid quality enhancement [7]. In this case, the grid control sources are extracted from the distribution of grid quality measures such as skewness and cell aspect ratio.

REFERENCES

1. Lee, K. D. and Loellbach, J. M., "Geometry-Adaptive Surface Grid Generation Using Parametric Projection," Journal of Aircraft, Vol. 26, No. 2, pp. 162-167, February 1989.

2. Lee, K. D. And Loellbach, J. M., "A Mapping Technique for Solution-Adaptive Grid Control," Proceedings AIAA 7th Aerodynamics Conference, Seattle, WA, pp. 129-139, AIAA Paper 89-2178, 1989.

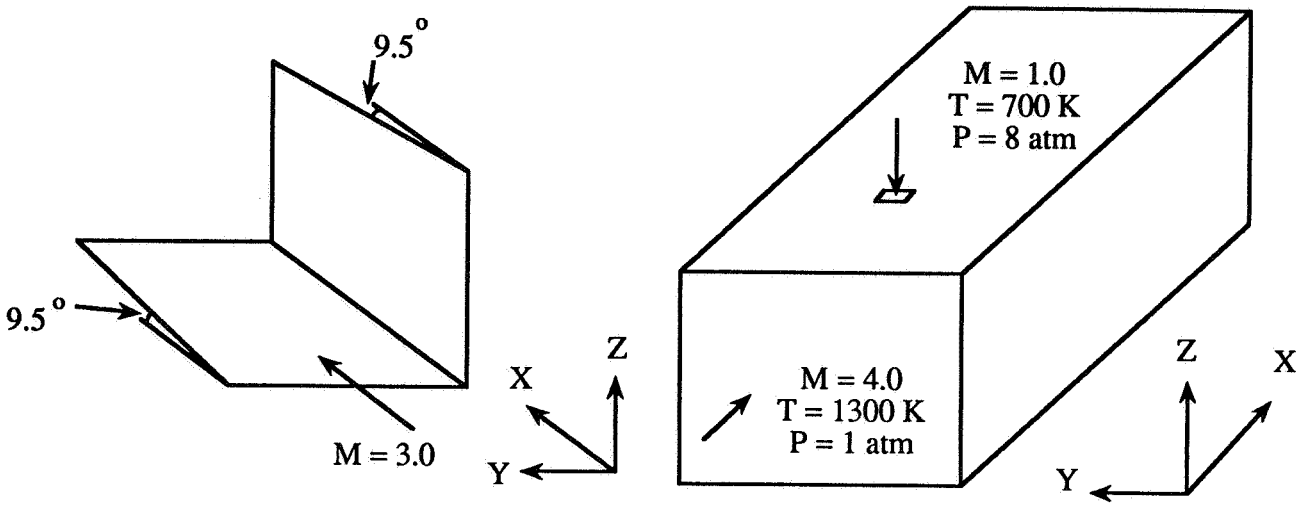
3. Vatsa, V., Thomas, J. L. and Wedan, B. W., "Navier-Stokes Computations of Prolate Spheroids at Angle of Attack," AIAA Paper 87-2627-CP, 1987.

4. West, J. E. and Korkegi, R. H., "Supersonic Interaction in the Corner of Intersecting Wedges at High Reynolds Numbers," AIAA Journal, Vol. 10, No. 5, pp. 652-656, May 1972.

5. Yu, S. T., Tsai, Y. P., and Shuen, J. S., "Three-Dimensional Calculation of Supersonic Reacting Flows Using an LU Scheme," AIAA Paper 89-0391, 1989.

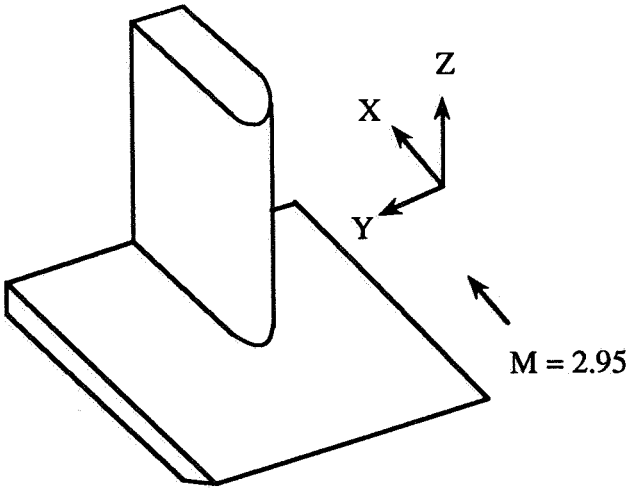
6. Nakahashi, K. and Deiwert, G. S., "A Three-Dimensional Adaptive Grid Method," AIAA Paper 85-0486, 1985.

7. Lee, K. D., Henderson, T. L., and Choo, Y. K., "Grid Quality Improvement by a Grid Adaptation Technique," Proceedings 3rd International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Barcelona, Spain, pp. 597-606, June 1991.



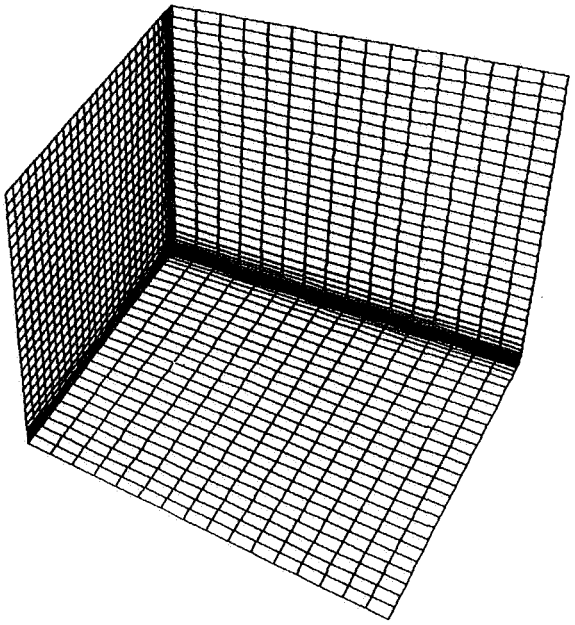
(a) Supersonic 3D corner flow.

(b) Supersonic mixing with transverse jet.



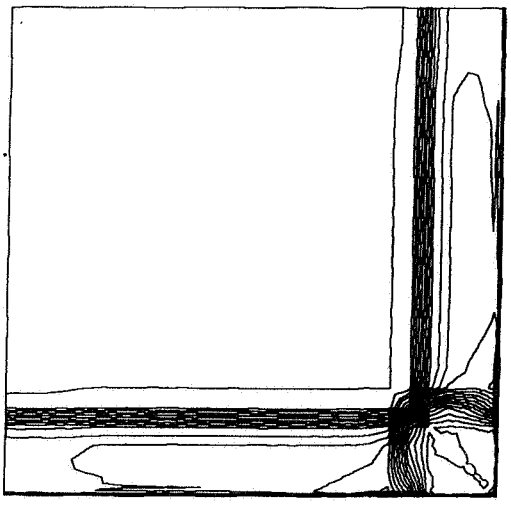
(c) Supersonic flow over blunt fin on flat plate.

Figure 1. Schematic flow configurations of the model problems.

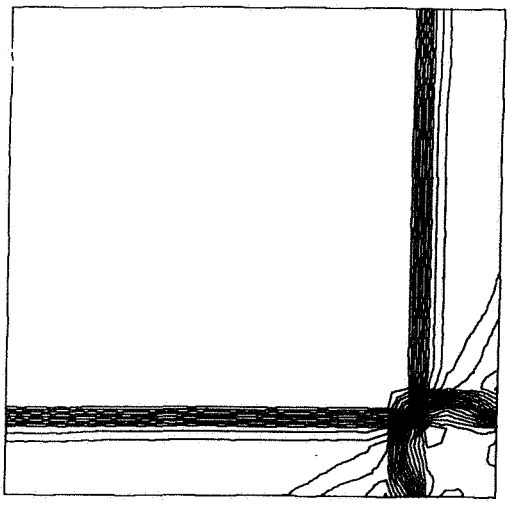


(a) Initial grid.

(b) Density contours on i-planes.

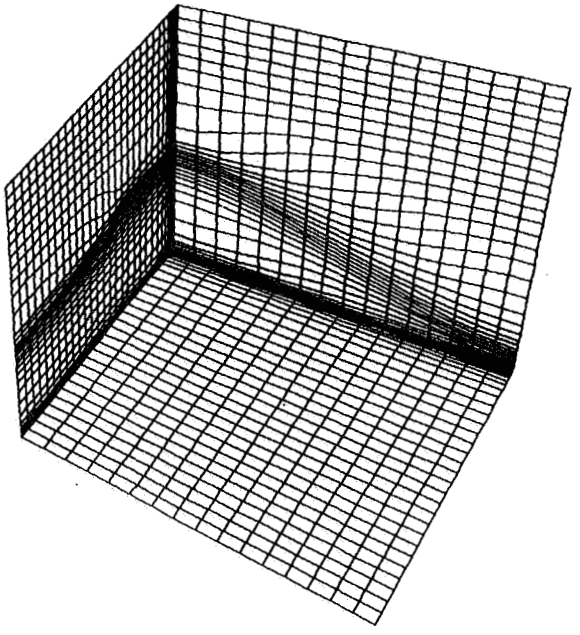


(c) Density contours at an i-plane.

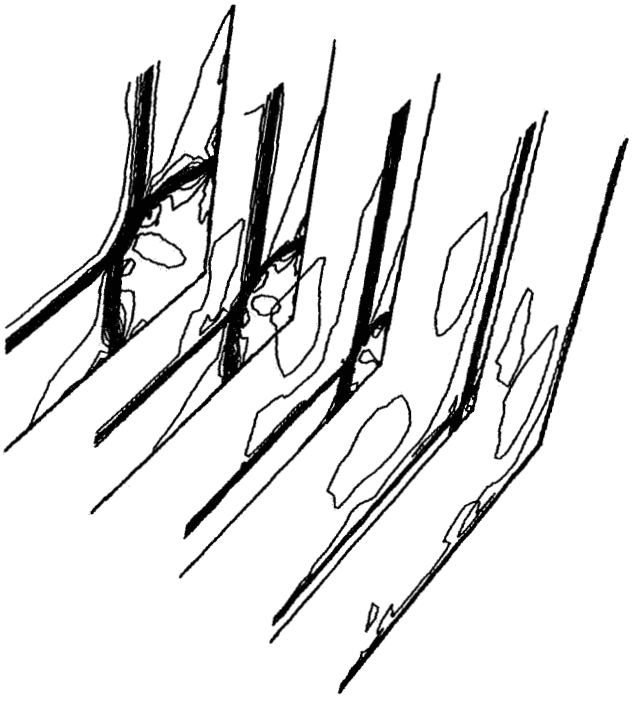


(d) Pressure contours at an i-plane.

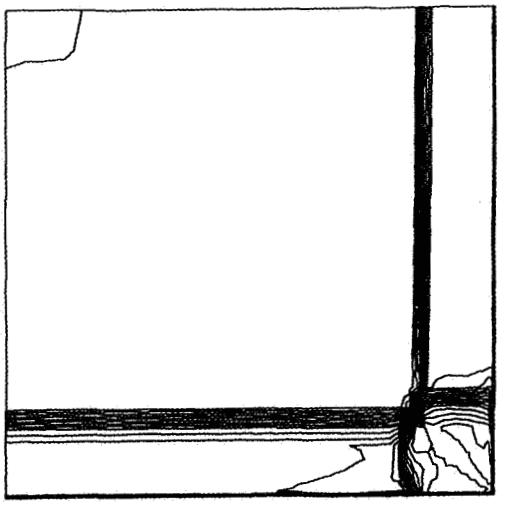
Figure 2. Initial grid and its flow solutions - supersonic 3D corner flow.



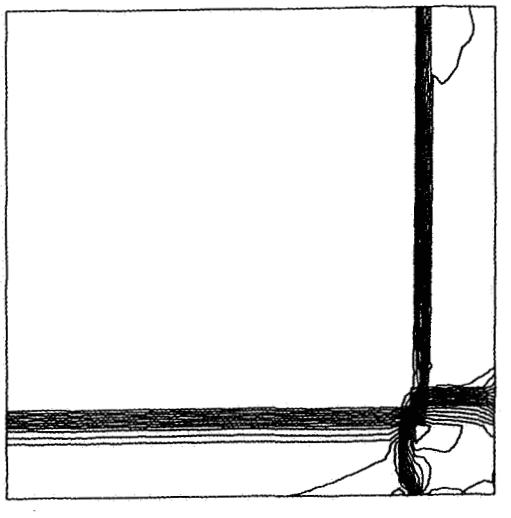
(a) One-direction adapted grid.



(b) Density contours on i-planes.

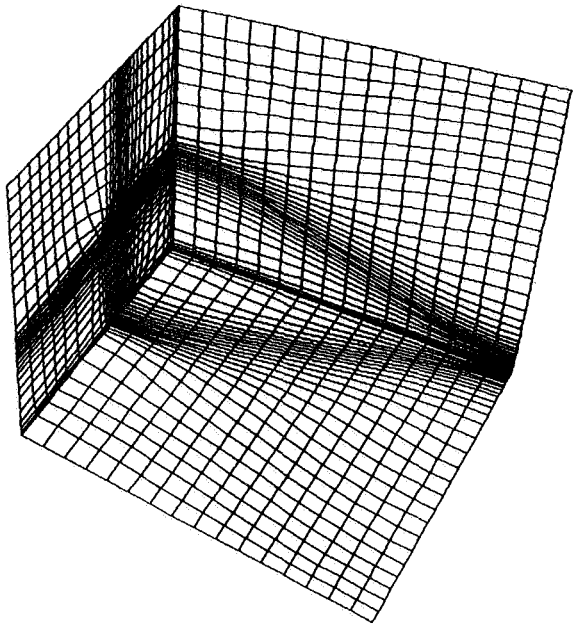


(c) Density contours at an i-plane.

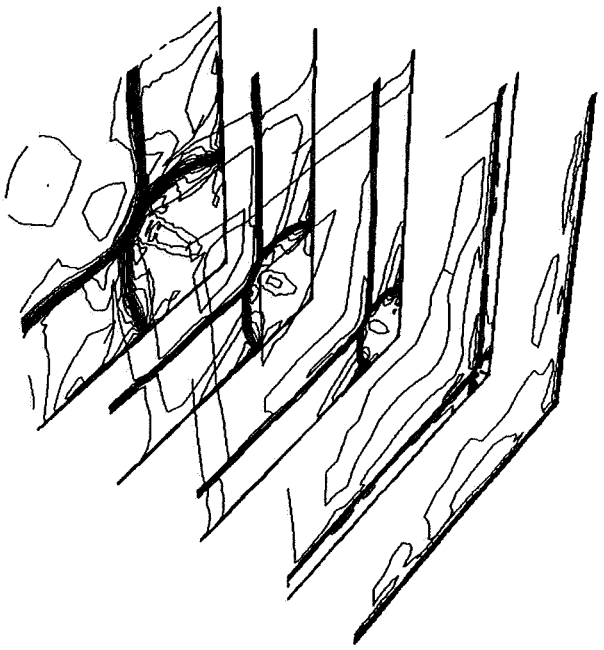


(d) Pressure contours at an i-plane.

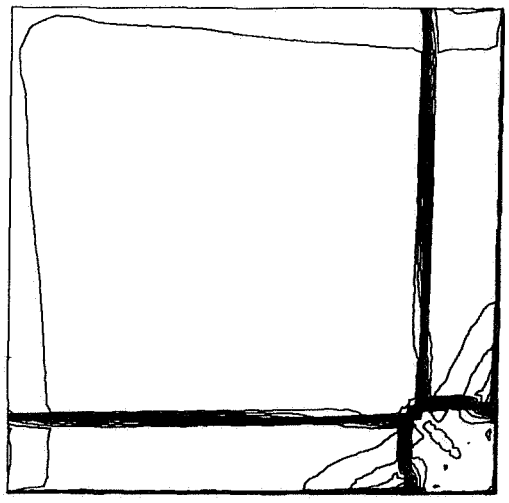
Figure 3. One-direction adapted grid and its flow solutions - supersonic 3D corner flow.



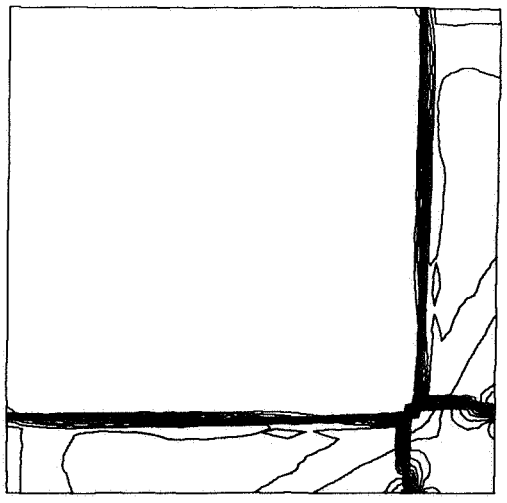
(a) Two-direction adapted grid.



(b) Density contours on i-planes.

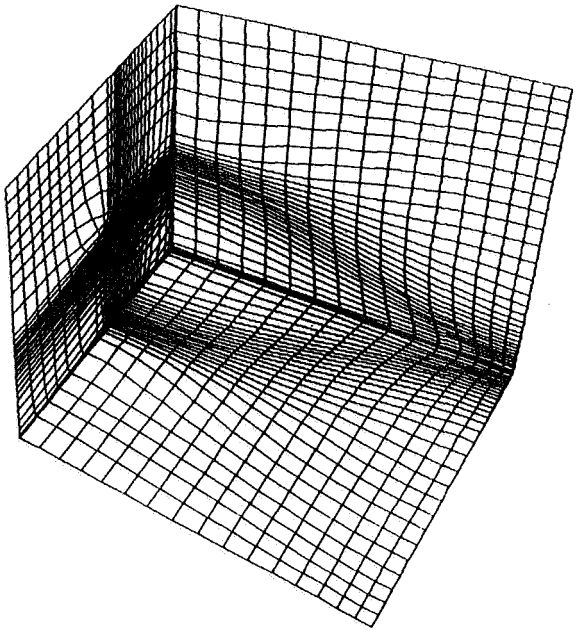


(c) Density contours at an i-plane.

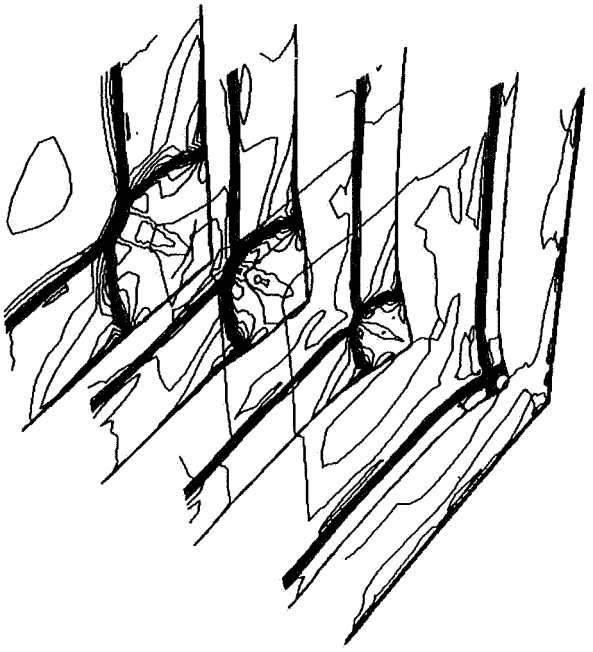


(d) Pressure contours at an i-plane.

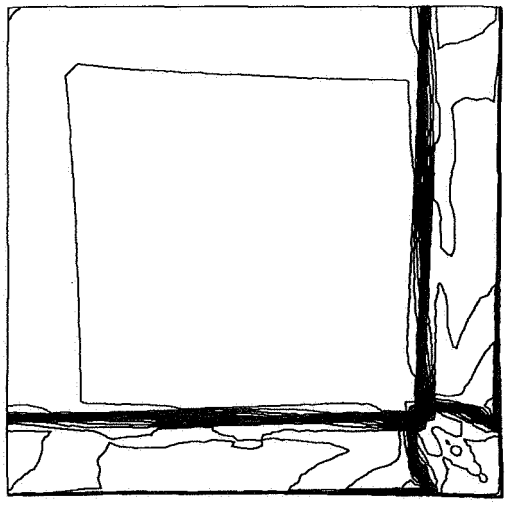
Figure 4. Two-direction adapted grid and its flow solutions - supersonic 3D corner flow.



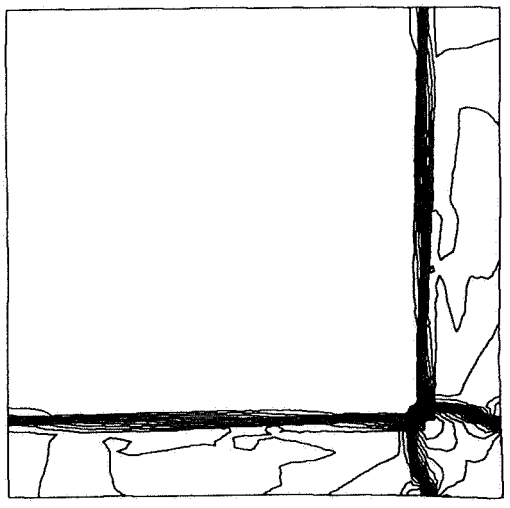
(a) Three-direction adapted grid.



(b) Density contours on i-planes.

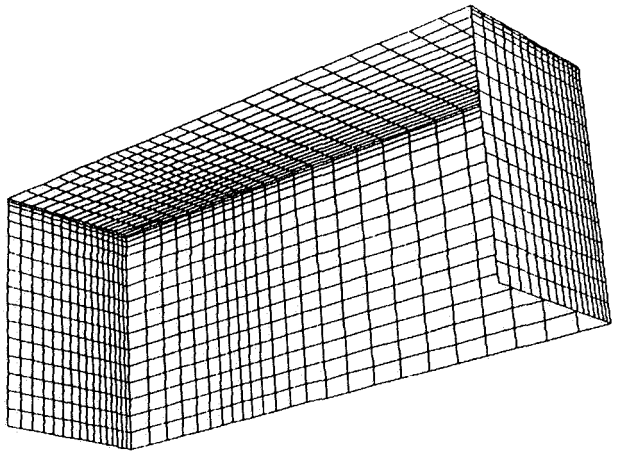


(c) Density contours at an i-plane.

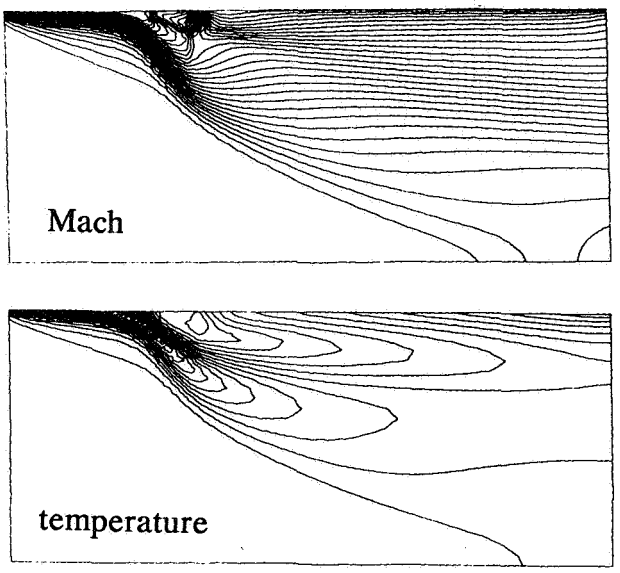


(d) Pressure contours at an i-plane.

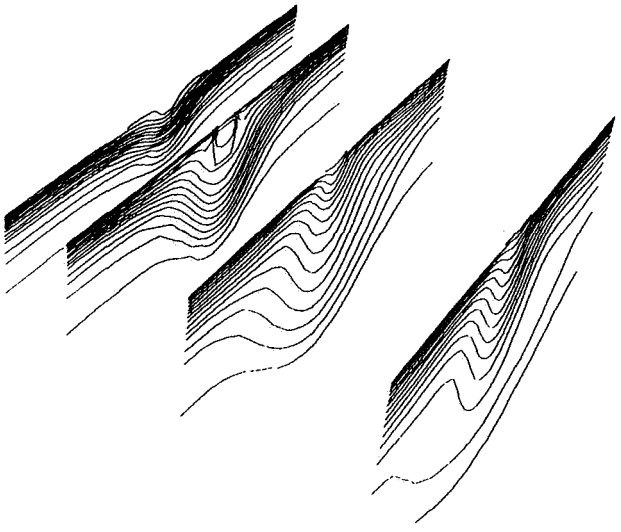
Figure 5. Three-direction adapted grid and its flow solutions - supersonic 3D corner flow.



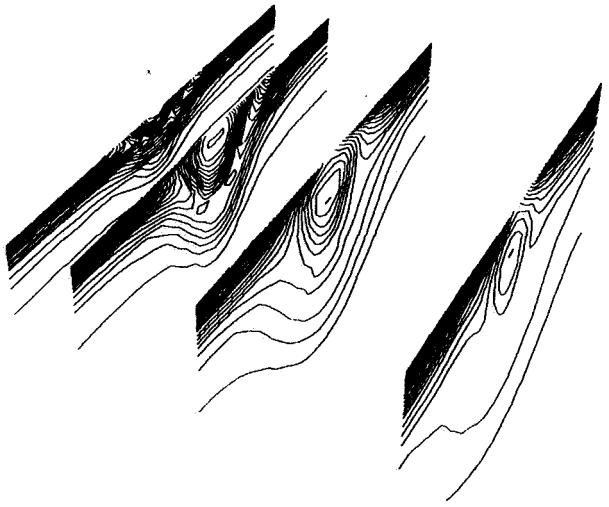
(a) Initial grid.



(b) Mach and temperature contours on the symmetry plane.

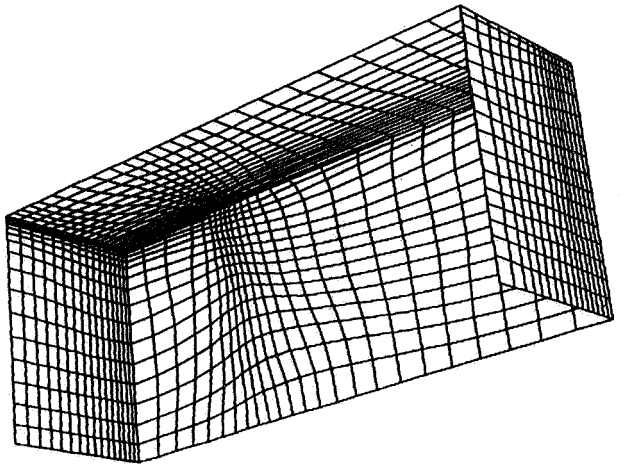


(c) Mach contours at i-planes.

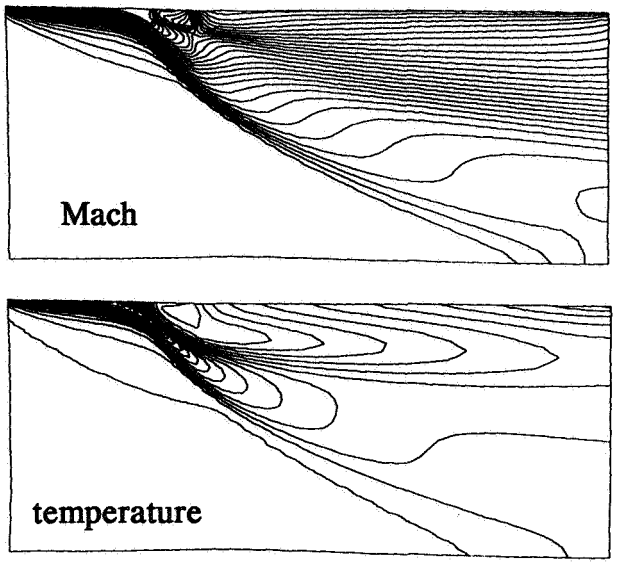


(d) Temperature contours at i-planes.

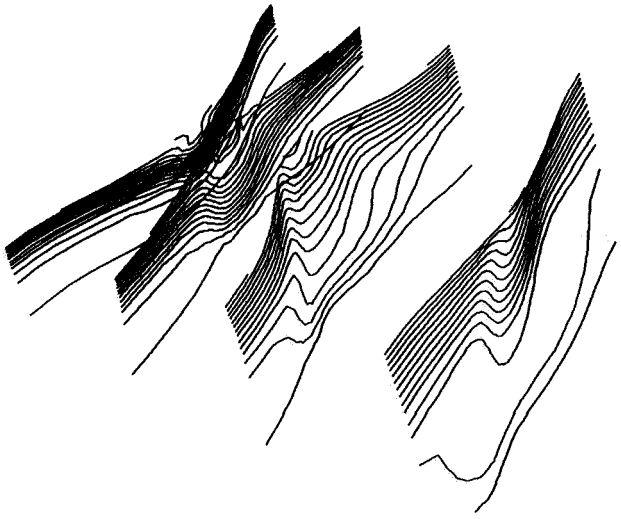
Figure 6. Initial grid and its flow solutions - supersonic mixing with a transverse jet.



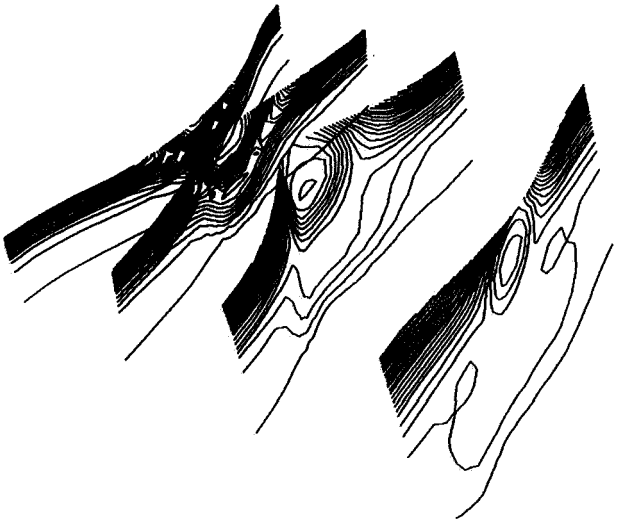
(a) Adapted grid.



(b) Mach and temperature contours on the symmetry plane.

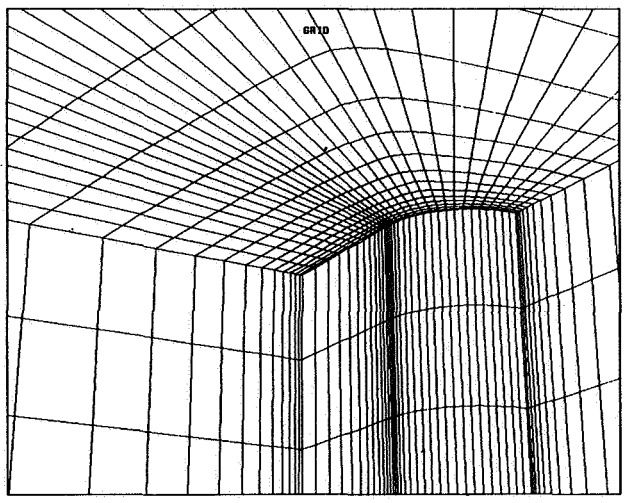


(c) Mach contours at i-planes.

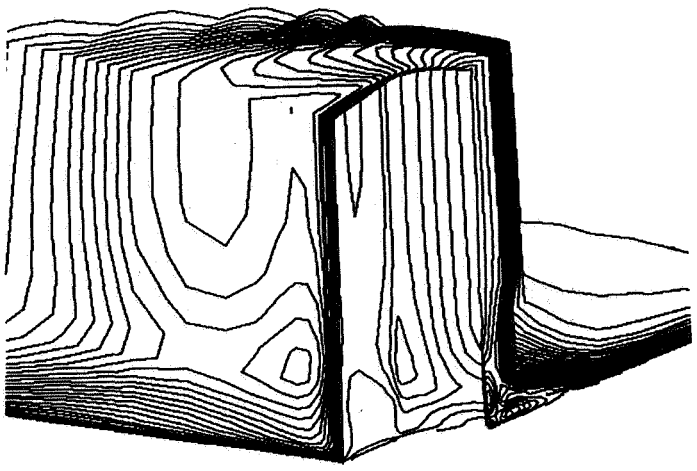


(d) Temperature contours at i-planes.

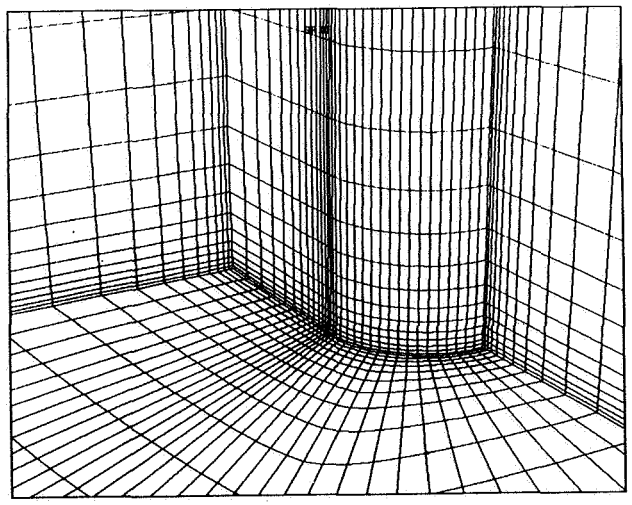
Figure 7. Adapted grid and its flow solutions - supersonic mixing with a transverse jet.



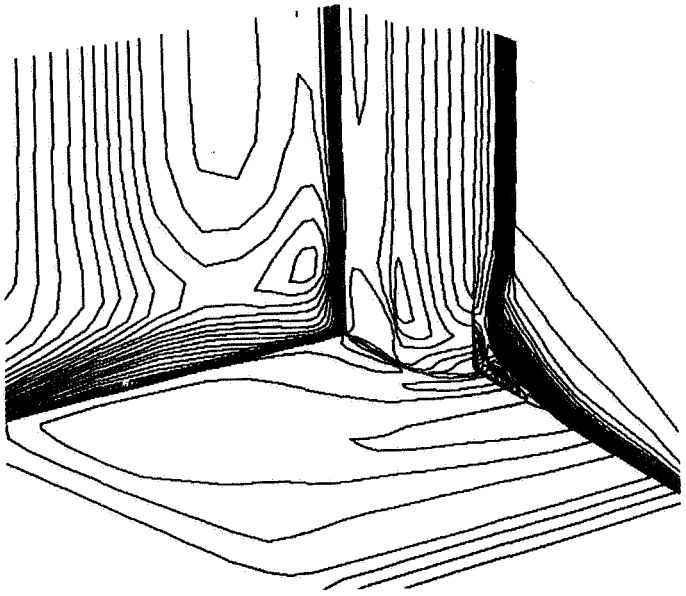
(a) Initial grid - upper side.



(b) Mach contours on grid in (a).

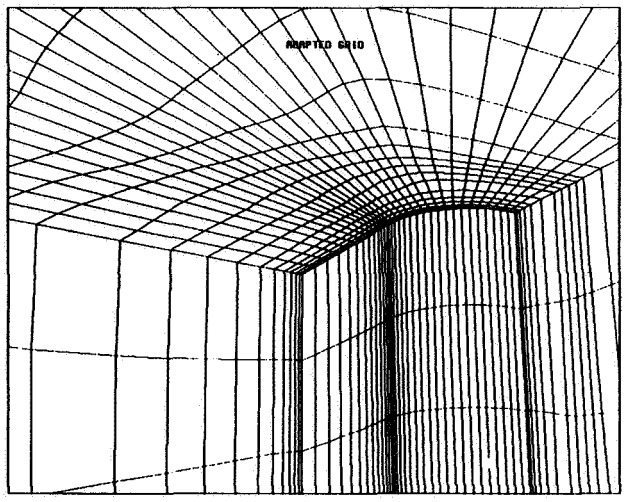


(c) Initial grid - lower side.

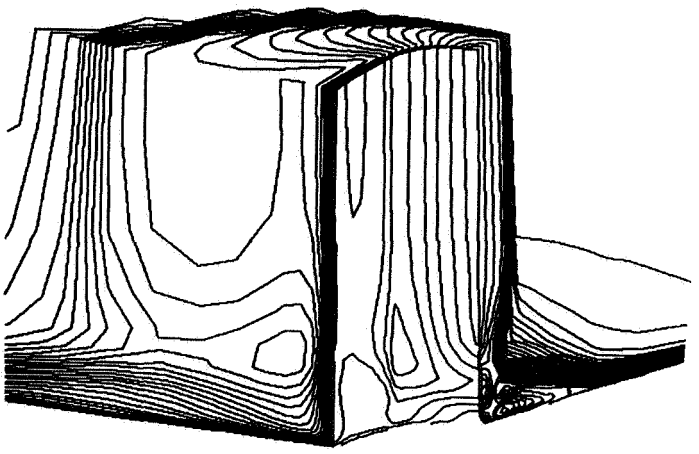


(d) Mach contours on grid in (c).

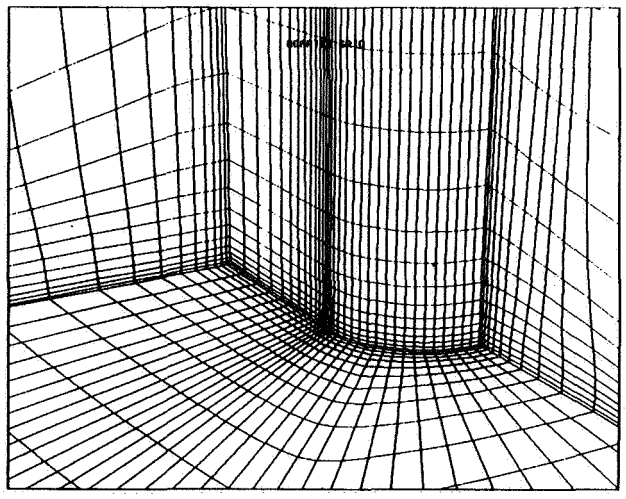
Figure 8. Initial grid and its flow solutions - supersonic flow over a blunt fin on a flat plate.



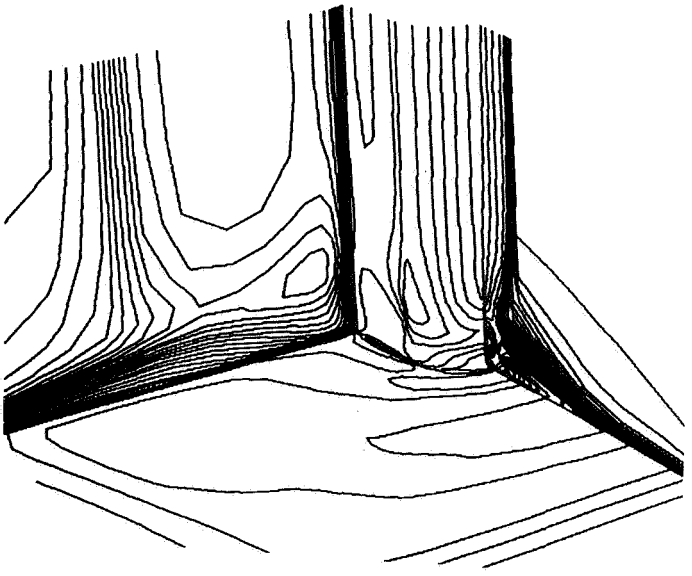
(a) Adapted grid - upper side.



(b) Mach contours on grid in (a).



(c) Adapted grid - lower side.



(d) Mach contours on grid in (c).

Figure 9. Solution-adaptive grid and its flow solutions - supersonic flow over a blunt fin on a flat plate.

ALGEBRAIC GRID ADAPTATION METHOD USING NON-UNIFORM RATIONAL B-SPLINE SURFACE MODELING+

Jiann-Cherng Yang* and B. K. Soni**
NSF Engineering Research Center and
Department of Aerospace Engineering
Mississippi State University
Mississippi State, MS 39762

ABSTRACT

An algebraic adaptive grid system based on equidistribution law and utilized by the Non-Uniform Rational B-Spline (NURBS) surface for re-distribution is presented. A weight function, utilizing a properly weighted boolean sum of various flow field characteristics is developed. Computational examples are presented to demonstrate the success of this technique.

INTRODUCTION

Grid adaptive schemes are divided into two basic categories: differential and algebraic. The differential method is based on a variational approach where a function which contains a measure of grid smoothness, orthogonality, and volume variation is minimized by using a variational principle. This approach provides a solid mathematical basis for the adaptive method, but the Euler-Lagrange equations must be solved in addition to the original governing equations. On the other hand, the algebraic method requires much less computational effort, but the grid may not be smooth. The algebraic techniques are based on devising an algorithm where the grid movement is governing by estimates of the local error in the numerical solution. This is achieved by requiring the points in the large error regions to attract other points and points in the low error region to repel other points. One of the disadvantages of algebraic method is that it generates the unsmooth grid. The utilization of NURBS will generate the well-distributed smooth grid (Fig. 1).

An adaptive grid is a grid that controls the placement of grid points automatically based on the solution of the physical problem under consideration and allows optimal grid redistribution as the solution progresses. The accuracy of the numerical algorithm depends not only on the formal order of approximation but also on the distribution of grid points in the computational domain. The adaptive grid is one of the methods that can make the numerical algorithm more accurate. The grid adaptive scheme, presented in this paper, is based on the equidistribution principle.

+ Work partially funded by a research on a grant from Teledyne Brown Engineering Corp and AFOSR.

* Graduate Student, Aerospace Engineering, Student Member AIAA.

** Associate Professor, Aerospace Engineering, Mississippi State University, Sr. Member AIAA.

The development of the adaptive algorithm for the structured flow simulation is accomplished as a two step process. The first step is to define an adaptive weighting mesh (distribution mesh) (Ref. 1–2) on the basis of the equidistribution law applied to the flow field solution. The second, and probably the most crucial step, is to redistribute grid points in the computational domain according to the aforementioned weighting mesh. Adaptive weighting (distribution mesh) provides the information on the desired concentration of points to the grid redistribution scheme. The evaluation of weighting mesh is accomplished by utilizing the weight function representing the solution variation and the equidistribution law (Ref. 3). The selection of the weight function plays a key role in grid adaptation (Ref. 4–5). A new weight function utilizing a properly weighted boolean sum of various flowfield characteristics is defined. The redistribution scheme is developed by utilizing Non–Uniform Rational B–Spline Surface (NURBS) representation (Ref 6). The application of NURBS representation results in a well–distributed smooth grid by maintaing the fidelity of the geometry associated with boundary curves. Various computational examples are presented to demonstrate the success of these methods.

GRID ADAPTATION METHOD

The basis of the grid adaptation is the equidistribution law in one dimension (Ref. 8) applied to the flow field. The equidistribution law is represented as

$$x_\xi \, w = constant \tag{1}$$

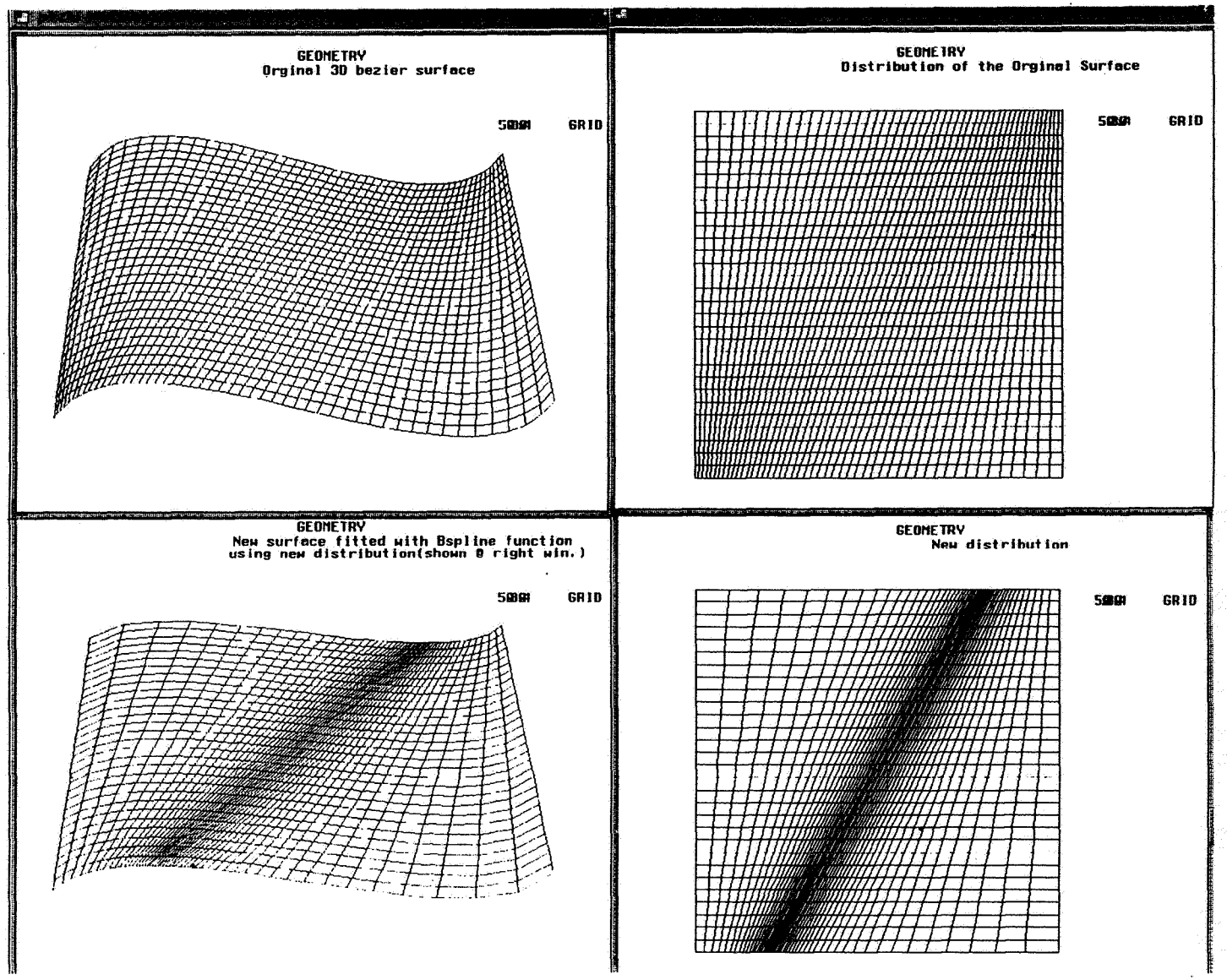
With the weight function w taken as a function of ξ, this is just the Euler equation for the minimization of the integral

$$I = \int_0^1 w(\xi)\, x_\xi^2 \, d\xi \tag{2}$$

Applying the spring analogy form to the above integral, the point location x_i can be defined by (Ref. 8)

$$x_i = L \frac{\int_1^i \frac{d\xi}{w(\xi)}}{\int_1^N \frac{d\xi}{w(\xi)}} \quad (i=2, 3 \ldots\ldots, N-1) \tag{3}$$

The integral in the denominator depends on the point distribution, amounting to a sum of $1/w$ over the points. However $\Delta\xi = 1$ by construction regardless of the distribution. Therefore, the integral in (3) is evaluated iteratively. The integral in the numerator is then also re–evaluated for each point, thus changing the point distribution again. This process must be continued until convergence before the final new distribution is obtained.



Physical domain Computational domain

Figure 1.
Use NURBS surface to generate smooth grid.

ALGORITHM

A self–explanatory, pictorial view of the algorithm of algebraic adaptive grid generation is shown in Fig. 1.

The grid and solution will be transferred from the flow solver if the solution is good enough to apply grid adaptation. The criterion of good solution can be the pressure L^2 residue or another property of solution that can be used as the convergent indication.

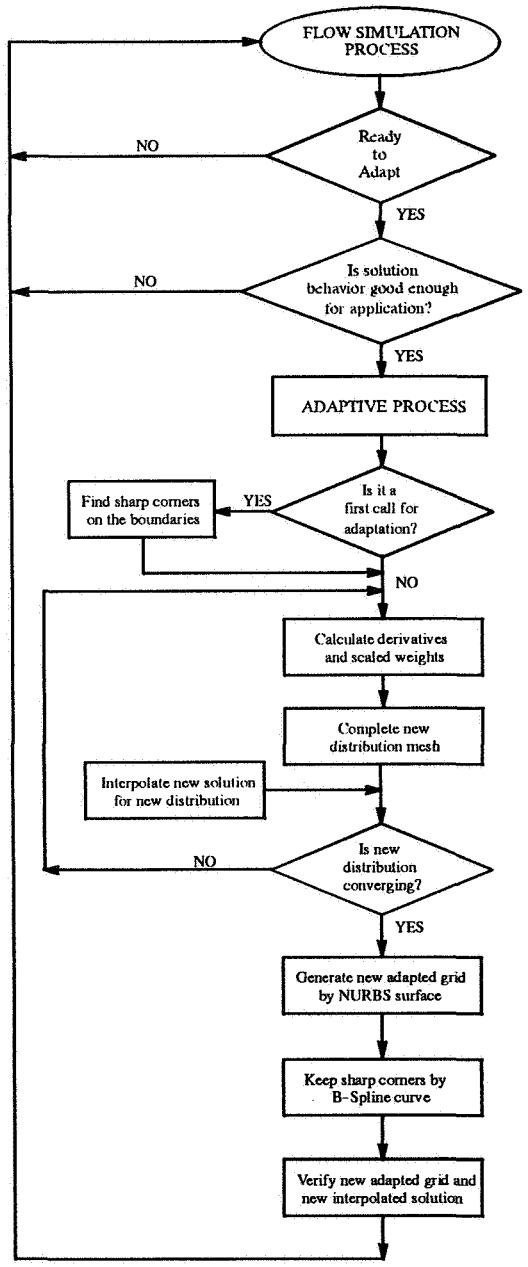


Figure 2.
Algorithm for 2–D Algebraic Adaptive Grid Generation System

WEIGHT FUNCTION

The weight function is a very important part in the adaptive grid system. Here we introduce a new weight function applicable to various flow field characteristics. The weight functions are computed in both the ξ and η computational directions, and then coupled adaptation is applied. A linear combination

$$1 + \sum_{j=1}^{N} \lambda_j w_j , \qquad with \sum \lambda_j = 1 \qquad (4)$$

where N – number of flow parameter (e.g. pressure, temperature, density, etc.)

λ_j – weighting factor associated with flow parameter $\lambda_j \geq 0$

$w_j - \alpha_j q_j \oplus \beta_j k_j = \alpha_j q_j + \beta_j k_j - (\alpha_j + \beta_j - 1) q_j k_j$

q_j – scaled gradient of the flow variable j such that $0 \leq q_j \leq 1$

k_j – scaled curvature values of the flow variable j such that $0 \leq k_j \leq 1$

$0 \leq \alpha_j \leq 1$, $0 \leq \beta_j \leq 1$, – weight factors assigned to gradients and curvatures is developed as the weight function utilizing the Boolean sum of contributions from scaled gradients and curvatures. The Boolean sum allows an appropriate weight to the gradients as well as the curvatures. For example, in the case of the j^{th} flow parameter, the influence of weights can be presented as follows:

q_j	k_j	α_j	β_j	w_j
1	0	1	β	1
1	0	α	β	α
0	1	α	1	1
0	1	α	β	β
1	1	1	1	1
1	1	α	1	1
1	1	1	β	1
1	1	α	β	1
0	0	α	β	0

Note that the value of the weight contribution is controlled by the weight factors and is at a maximum when gradients and/or curvature values are at a maximum. The gradients and curvatures associated with the flow characteristics under consideration are smoothed using a four point diffusion process.

SCALING SCHEME FOR α & β

Dwyer has developed a method to determine the percentage change in a dependent variable as a priori. (Ref. 9). This method is enhanced here to determine the suitable scaling α and β at each grid point. (Ref. 3) The adaptation technique can be described in the generalized coordinates as

$$\xi = \frac{\alpha\int_0^x q d\bar{x} + \beta\int_0^x k d\bar{x} - (\alpha+\beta-1)\int_0^x q d\bar{x}\int_0^x k d\bar{x}}{\alpha\int_0^1 q dx + \beta\int_0^1 k dx - (\alpha+\beta-1)\int_0^1 q dx\int_0^1 k dx} \tag{5}$$

where x – normalized arc length, q – flow parameter gradient, k – flow parameter curvature, α & β – weight factor assigned to gradients and curvatures. so that

$$\Delta\xi = \frac{[\,\alpha q + \beta k - (\alpha+\beta-1)qk\,]\,\Delta x}{\alpha\int_0^1 q dx + \beta\int_0^1 k dx - (\alpha+\beta-1)\int_0^1 q dx\int_0^1 k dx} \tag{6}$$

Then with R_1 defined as

$$R_1 = \frac{\alpha\int_0^1 q dx}{\alpha\int_0^1 q dx + \beta\int_0^1 k dx - (\alpha+\beta-1)\int_0^1 q dx\int_0^1 k dx} \tag{7}$$

we have the maximum percentage change in the solution over a grid interval,

$$r = \frac{q\Delta x}{\int_0^1 q dx} = \frac{\Delta\xi}{R_1} - \frac{[(1-q)\beta\int_0^1 k dx + (1-\alpha)q\int_0^1 k dx]\Delta x}{\alpha\int_0^1 q dx}$$

$$\leq \frac{\Delta\xi}{R_1} = \frac{1}{NR_1} \tag{8}$$

since $\Delta\xi = 1/N$ and N+1 is the number of points on the coordinate line. If we take an equality in Eq. (8) with R_1 from Eq. (7) , we have

$$r = \frac{1}{NR_1}$$

$$= \frac{\alpha(1-\int_0^1 k dx)\int_0^1 q dx + \int_0^1 q dx\int_0^1 k dx + \beta(1-\int_0^1 q dx)\int_0^1 k dx}{N\alpha\int_0^1 q dx} \tag{9}$$

Ignoring the effect of the β term:

$$r = \frac{(1-\int_0^1 k dx)\alpha\int_0^1 q dx + \int_0^1 q dx\int_0^1 k dx}{N\alpha\int_0^1 q dx} \tag{10}$$

$$\Rightarrow \alpha = \frac{\int_0^1 q dx \int_0^1 k dx}{\int_0^1 q dx \ (rN - 1 + \int_0^1 k dx \ )} \tag{11}$$

Similarly with R_2 defined as

$$R_2 = \frac{\beta \int_0^1 k dx}{\alpha \int_0^1 q dx + \beta \int_0^1 k dx - (\alpha + \beta - 1) \int_0^1 q dx \int_0^1 k dx} \tag{12}$$

we can get

$$\beta = \frac{R_2 \int_0^1 q dx \ [\ \alpha (1 - \int_0^1 k dx) + \int_0^1 k dx \ ]}{\int_0^1 k dx \ (\ 1 - R_2 + R_2 \int_0^1 q dx \ )} \tag{13}$$

With Boolean sum operation, we found that the choice of $R_1 = 1$ and $R_2 = 1$ still results in a very good adaptive grid.

GENERATE REDISTRIBUTED ALGEBRAIC GRID

All of the adaptive grids displayed in this paper are generated by the Non-Uniform Rational B-spline in surface (NURBS) or reparametrized uniform B-spline surface. The reparametrized uniform B-spline curve is applied to move the boundary points in order to keep the sharp corners (Fig.3). In general, the degree m NURBS surface (3-D) is defined as the projection of a tensor product B-spline in 4-D (Ref. 5-6):

$$\underset{\sim}{r}(\xi,\eta) = \frac{\sum_{i=0}^{NI} \sum_{j=0}^{NJ} B_{ij}^{m}(\xi,\eta) \ H_{ij} \ Q_{ij}}{\sum_{i=0}^{NI} \sum_{j=0}^{NJ} B_{ij}^{m}(\xi,\eta) \ H_{ij}} \tag{14}$$

where B_{ij}^{m} are the mth order bivariate B-spline basis functions. H_{ij} and Q_{ij} are the associated latice of weights and control points. In the grid redistribution process, a matrix form of bi-cubic NURBS was applied which can be formulated as (Ref. 7):

$$\underset{\sim}{r}(\xi,\eta) = [1 \ \xi \ \xi^2 \ \xi^3] \ M_{bl}^{3} \ [HQ] \ [M_{bl}^{3}]^T \ [1 \ \eta \ \eta^2 \ \eta^3]^T \tag{15}$$

$$\xi = \frac{t - t_i}{t_{i+1} - t_i} \quad , \quad \eta = \frac{s - s_i}{s_{i+1} - s_i} \tag{16}$$

where control points and weights [H Q] are as follow:

$$[HQ] = \begin{bmatrix} (HQ)_{i-1,j-1} & (HQ)_{i-1,j} & (HQ)_{i-1,j+1} & (HQ)_{i-1,j+2} \\ (HQ)_{i,j-1} & (HQ)_{i,j} & (HQ)_{i,j+1} & (HQ)_{i,j+2} \\ (HQ)_{i+1,j-1} & (HQ)_{i+1,j} & (HQ)_{i+1,j+1} & (HQ)_{i+1,j+2} \\ (HQ)_{i+2,j-1} & (HQ)_{i+2,j} & (HQ)_{i+2,j+1} & (HQ)_{i+2,j+2} \end{bmatrix} \tag{17}$$

The matrix M^3_{bl} is defined as

$$M^3_{bl} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \tag{18}$$

$$b_{11} = \frac{(t - t_{i+1})^2}{(t_{i+1} - t_{i-2})\,(t_{i+1} - t_{i-1})}, \quad b_{12} = 1 - (b_{11} + b_{13})$$

$$b_{21} = -3b_{11}, \quad b_{22} = -(b_{21} + b_{23})$$

$$b_{31} = 3b_{11}, \quad b_{32} = -(b_{31} + b_{33})$$

$$b_{41} = -b_{11}, \quad b_{42} = -(b_{41} + b_{43} + b_{44})$$

$$b_{13} = \frac{(t_{i-1} - t_i)^2}{(t_{i-1} - t_{i+2})\,(t_{i-1} - t_{i+1})}, \quad b_{14} = 0$$

$$b_{23} = \frac{3(t_{i+1} - t_i)\,(t_i - t_{i-1})}{(t_{i-1} - t_{i+2})\,(t_{i-1} - t_{i+1})}, \quad b_{24} = 0$$

$$b_{33} = \frac{3(t_{i+1} - t_i)^2}{(t_{i-1} - t_{i+2})\,(t_{i-1} - t_{i+1})}, \quad b_{34} = 0$$

$$b_{44} = \frac{(t_{i+1} - t_i)^2}{(t_i - t_{i+3})\,(t_i - t_{i+2})}$$

$$b_{43} = -(t_{i+1} - t_i)^2 \left[\frac{1}{(t_{i-1} - t_{i+1})\,(t_{i-1} - t_{i+2})} + \frac{1}{(t_i - t_{i+2})\,(t_i - t_{i+3})} + \frac{1}{(t_{i+2} - t_{i-1})\,(t_{i+2} - t_i)}\right]$$

The redistributed algebraic grid is generated by utilizing distribution mesh as parameter space. The convex hull, local support, and variation diminishing properties (Ref. 7) of B-spline functions contribute to the generation of the well-distributed smooth grid. The application of inverse NURBS formulation (Ref. 6) allows reevaluation of control points which influences the fidelity of surface geometry during the redistribution process. The redistributed grid is evaluated for positive areas and smoothness. If there is any negative area, the further more grid adaptation will not be executed.

APPLICATIONS

The initial grids were generated by (GENIE–2D–3D) (Ref. 1). The basic flow simulation codes is the computer code PARC2D (Ref. 10) which are algorithmically based on the Beam–Warming implicit finite difference scheme using approximate factorization. The routines for grid adaptation are incorporated in the PARC2D system with proper data flow.

(1) Wedge Problem with a 5 Degree Incline Angle at the Leading Edge. Using the theory developed for supersonic flow, it is possible to compute the exact behavior of the shock waves generated by single and double wedge in supersonic flow. In the wedge problem, the Mach number is 1.9, and the Reynolds number is 1.0×10^{10} with a grid size at 80 x 60. There is one shock and one reflected shock in the control volume. Without grid adaptation, the shock and the reflected shock are as thick as two small tapes. Even with a significant number of iterations (e.g. 10000 iterations)(Fig. 4). of the flow solver, the solution shows very little improvement. After grid adaptation, the two shocks appear thinner. Extremely thin shocks can be achieved by increasing grid adaptation numbers (Fig. 5). The grid adaptation was performed at every 300 iterations. The redistributed grids and their influence on the solution are presented in Figure 5. It can be seen that at every adaptation stage the pressure contours have indicated a crisper shock. A closer look at the changing grid behavior near the shock region is presented in Figure 6–7. In spite of extremely skewed grid lines, the code was able to execute and provide a better solution.

(2) Double Wedge Case with Grid Size 103 x 27 and 2.0 Mach Number. As expected, the shocks are thinner with grid adaptations, and the outlet near the mixture area of the second shock and the third shock is more clear (Fig. 8).

(3) Cylinder Case with Grid Size 80 x 100, Mach Number 3.0,and Zero Angle of Attack. The bow shock in front of the cylinder is captured better in the case of the adapted grid as shown in figure 9–10. Although the bow shock is a curve, the adaptive grid also bends along the shock wave.

It is generally well known that either rapidly varying the step size or the skew grid angle can lead to poor accuracy when using finite difference approximations. Dwyer has shown by numerical experiments that variable grids coupled with adaptive techniques are accurate. (Ref. 9) In this paper we also observed in several cases that high skew angle grids coupled with adaptive techniques can excellently resolve the shock (Fig. 5–10). We also acknowledge that if the orientation of the grid line is aligned with the shock wave, the shock wave is captured exactly (Fig. 11). Thus, not only the cluster of the grid points near the shock is necessary but also the alignment of the grid points with the shock wave is important. We found that during the adaptation, the alignment of the grid points with shock wave will be accomplished by forming the skewed grid lines, according to the flow solutions. It is hoped that mathematical analysis could be developed to prove this observation, but the present investigators have not proceeded in this pursuit. We hope it will be accomplished in the near future.

(4) Airfoil NACA 0012 with Grid Size is 180 x 160, Mach Number 0.799, and the Angle of Attack 1.25. The upper surface shock is thinner with the grid adaptation. The lower surface has no weak shock if no grid adaptation is applied, but with grid adaptation, the weak shock was captured (Fig. 12). Comparing the C_p vs x/c charts, it is clear that the adaptive grid caught both shocks much better than the initial grid did (Fig. 13), but the adaptive grid loses the grid concentration at the leading edge and the tail. This is because the control field is so large that the effect of concentrating grid lines near the high gradient and/or the high curvature is spread to the whole field. The strategies that can improve this kind of condition are modifying the weight function by controlling the distribution near the body surface or subdividing the whole field into several blocks with some smaller blocks near

the surfaces of the airfoil and applying the local grid adaptation. The second strategy needs the development of a multi-block adaptive grid system. Both strategies are currently under investigation.

CONCLUSION

With a new weight function formula, a new method that applies the NURBS surface in the algebraic adaptive grid generation system is introduced. The effect of this new adaptive grid system is pretty good in capturing the shock wave for some smaller control field. The grid adaptation strategies that are applied to catch the high gradient and/or high curvature area are used not only to cluster grid points near those areas but also to move the grid points to align along the high solution derivative areas. For complicated control fields, the multi-blocks adaptive grid system may be required. The weight function that is suitable for laminar or turbulance flow simulation which need very small cells near body surface, also needs to be investigated. The extension of this 2-D general adaptive grid system to a three dimensional problem and its adaptation to a multiblock environment is currently underway.

References

1. Soni, B. K., "GENIE: GENeration of Computational Geometry-Grids for Internal Flow Configurations", *proceedings of Numerical Grid Generation in Computational Fluid Mechanics '88*, Miami, FL, 1988.
2. Soni, B. K., "Grid Generation for Internal Flow Configurations", *Journal of Computers and Mathematics with Applications, 1991.*
3. Thompson, J. F., "A Survey of Dynamically-Adaptive Grids in The Numerical Solution of Partial Differential Equations", *Applied Numerical Mathematics 1*, p. 3, North-Holland, 1985.
4. Connett, W. C., Agarwal, R. K., and Schwartz, A. L., "An Adaptive Grid-Generation Scheme for Flow Field Calculations", AIAA-87-0199.
5. Abolhassani, J. S., Smith, R. E., and Surendra, N. T., "Grid Adaptation for Hypersonic Flow", AIAA-87-1169.
6. Yamaguchi, F., "Curves and Surfaces in Computer Aided Geometric Design", Springer-Verlag, 1988.
7. Yoon, Y.-H., "Enhancements and Extensions of EAGLE Grid Generation System", Ph. D. Dissertation, Mississippi State University, Mississippi, MS,1991.
8. Thompson, J. F., Warsi, Z. U. A., Mastin, C. W., Adaptive Grids, in: *Numerical Grid Generation Foundations and Applications* (North-Holland, 1985), p. 370-378.
9. Dwyer, H. A., "Grid Adaption for Problem in Fluid Dynamics", AIAA Journal Vol. 22. No. 12, December 1984.
10. Cooper, G. K., Sirbough, J. R., "PARC Code: Theory and Usage", AEDC-TR-89-15, Arnold Engineering Development Center, Arnold AFB.

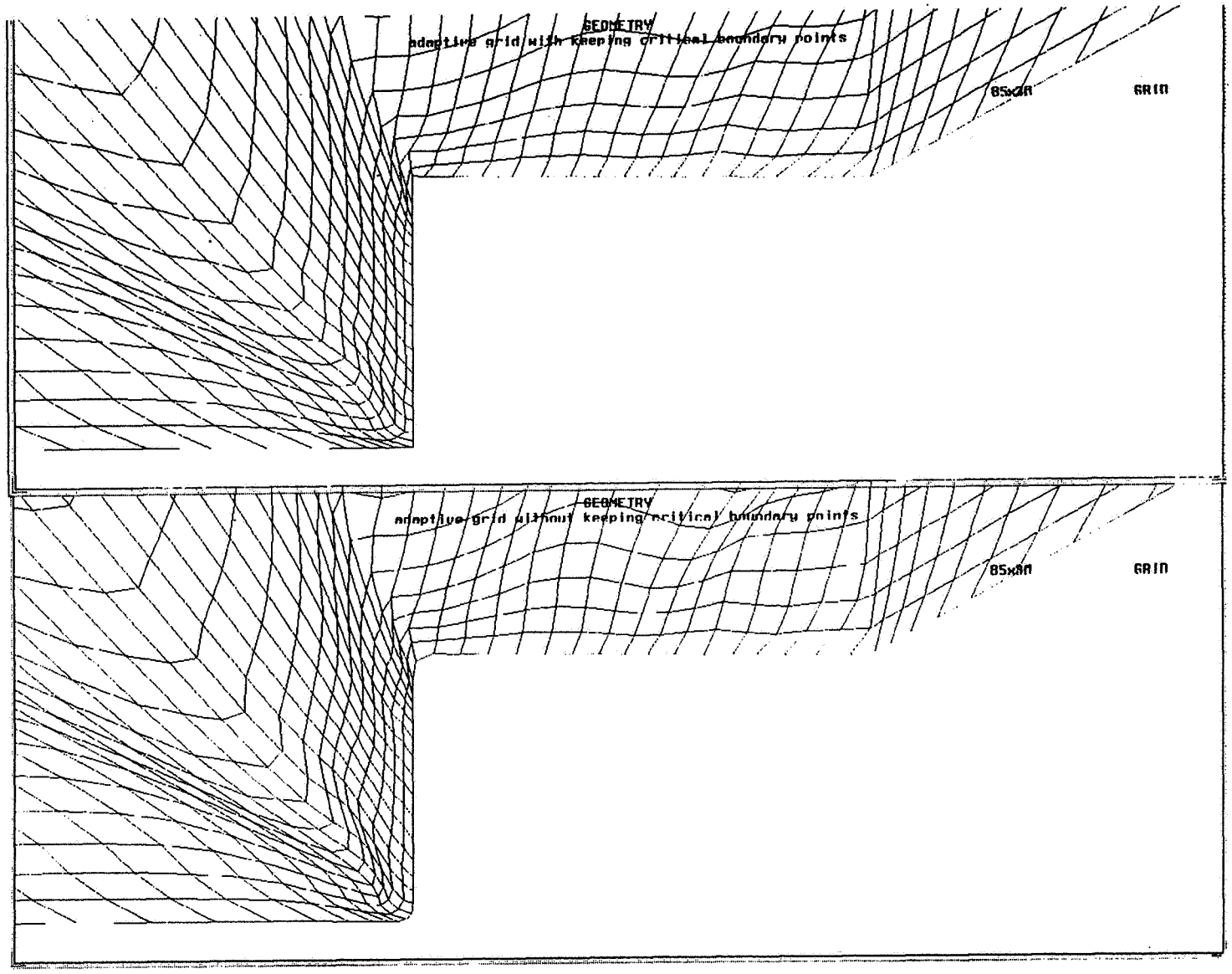


Figure 3.
Use B-Spline curve to keep sharp corners.

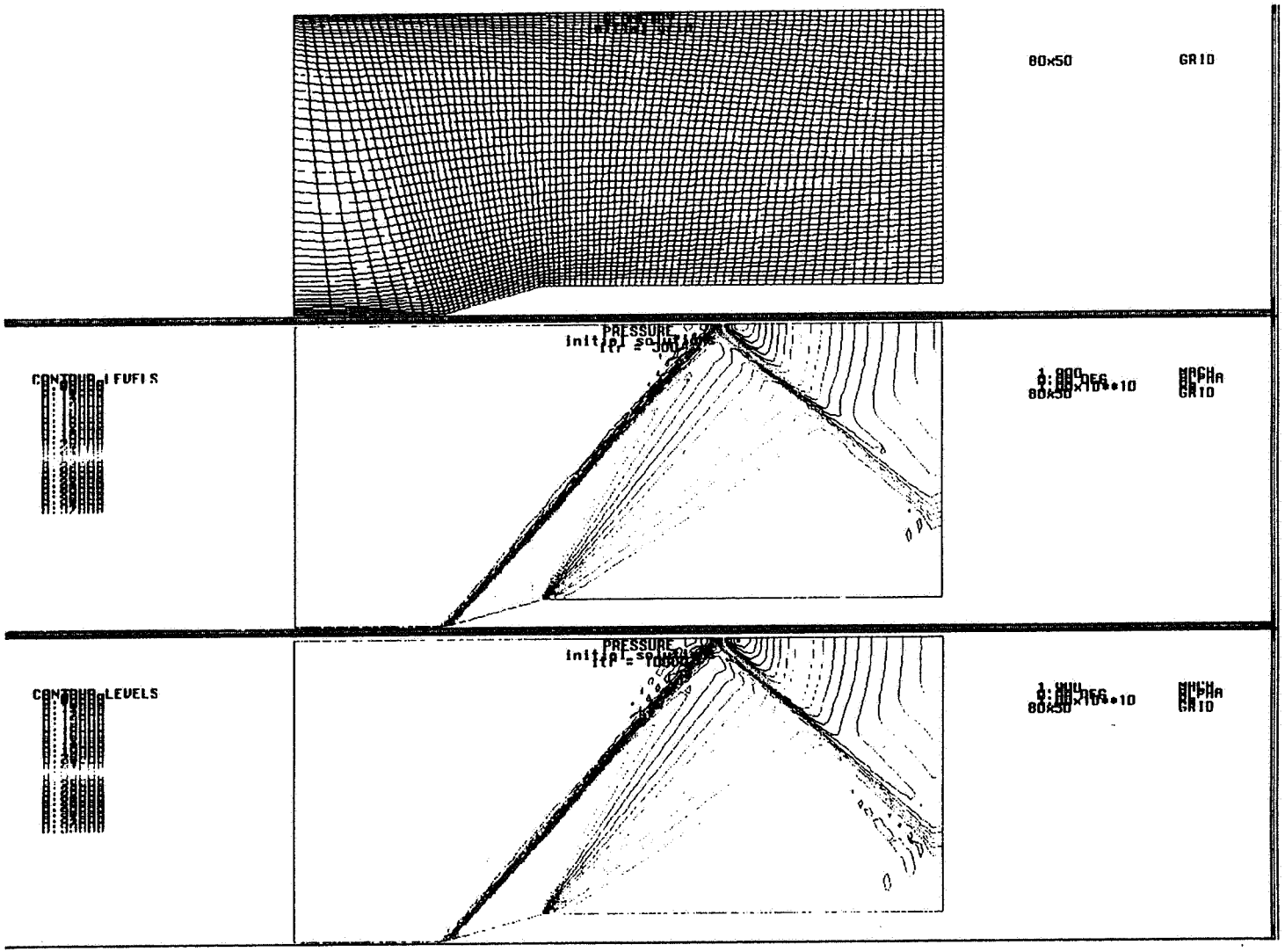
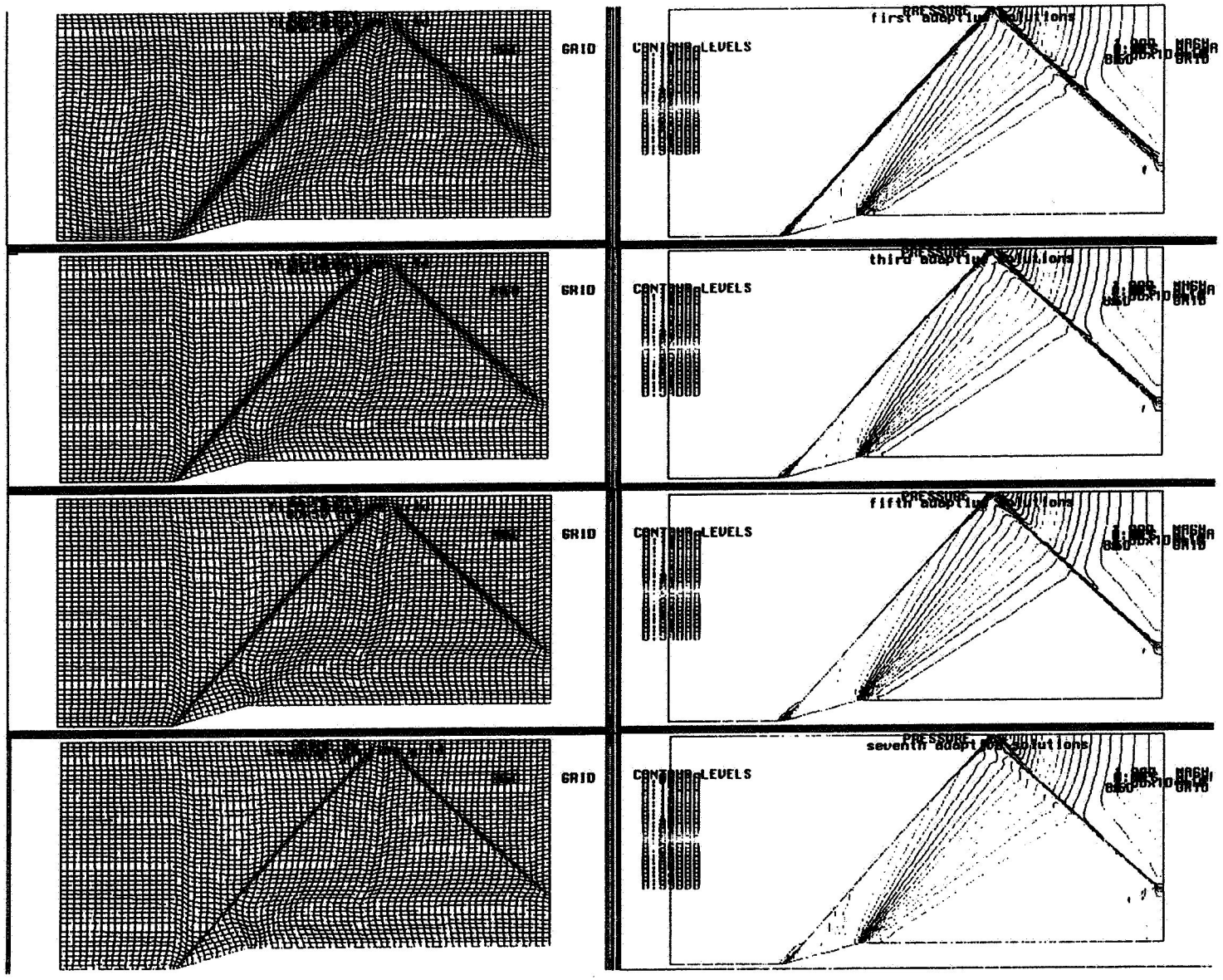


Figure 4.
Single wedge initial grids & solutions after 300 & 10000 iterations.



Grids

Pressure contours

Figure 5.
Single wedge grid adaptation process.

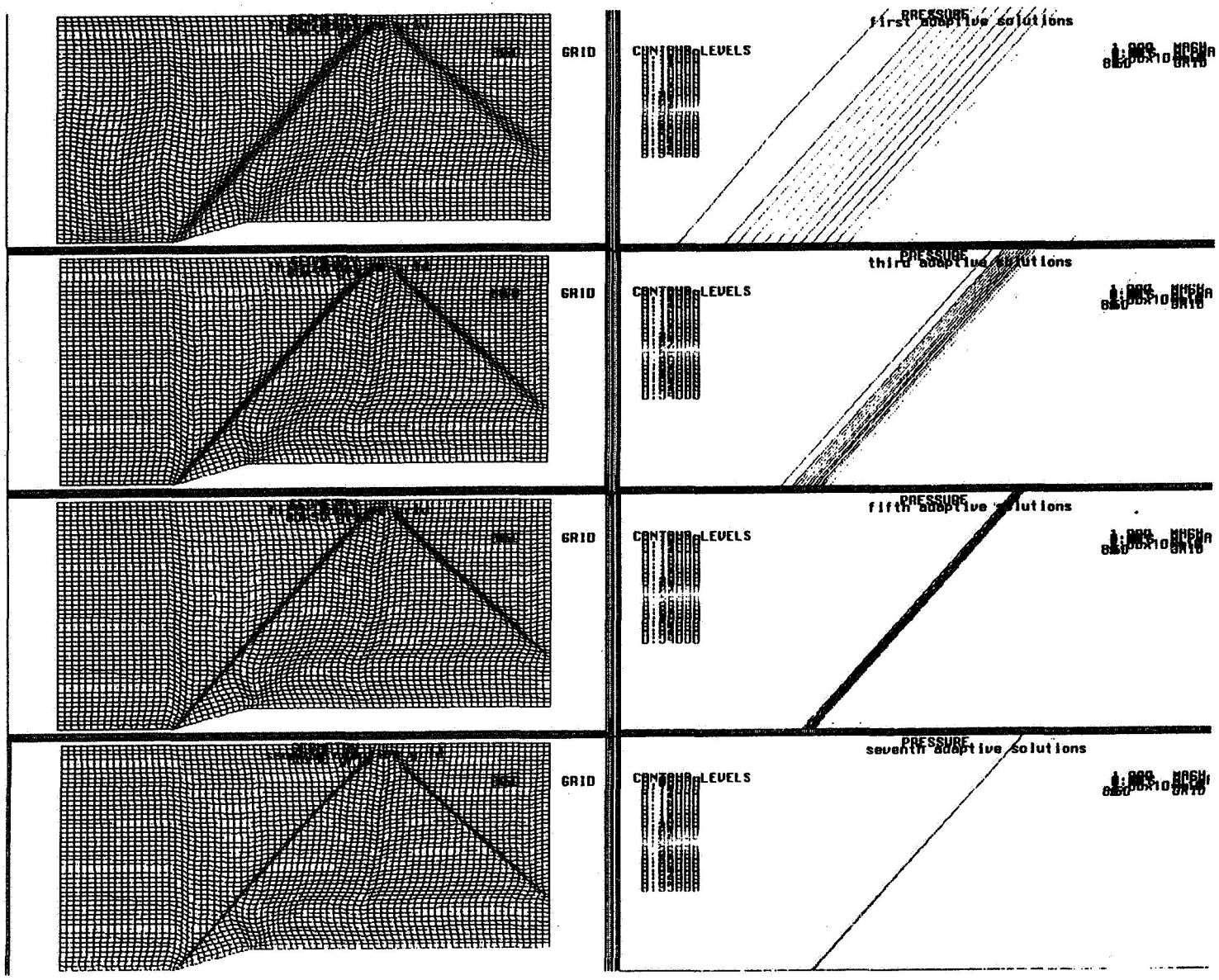


Figure 6.
Single wedge, closer pressure contour look near shock region.

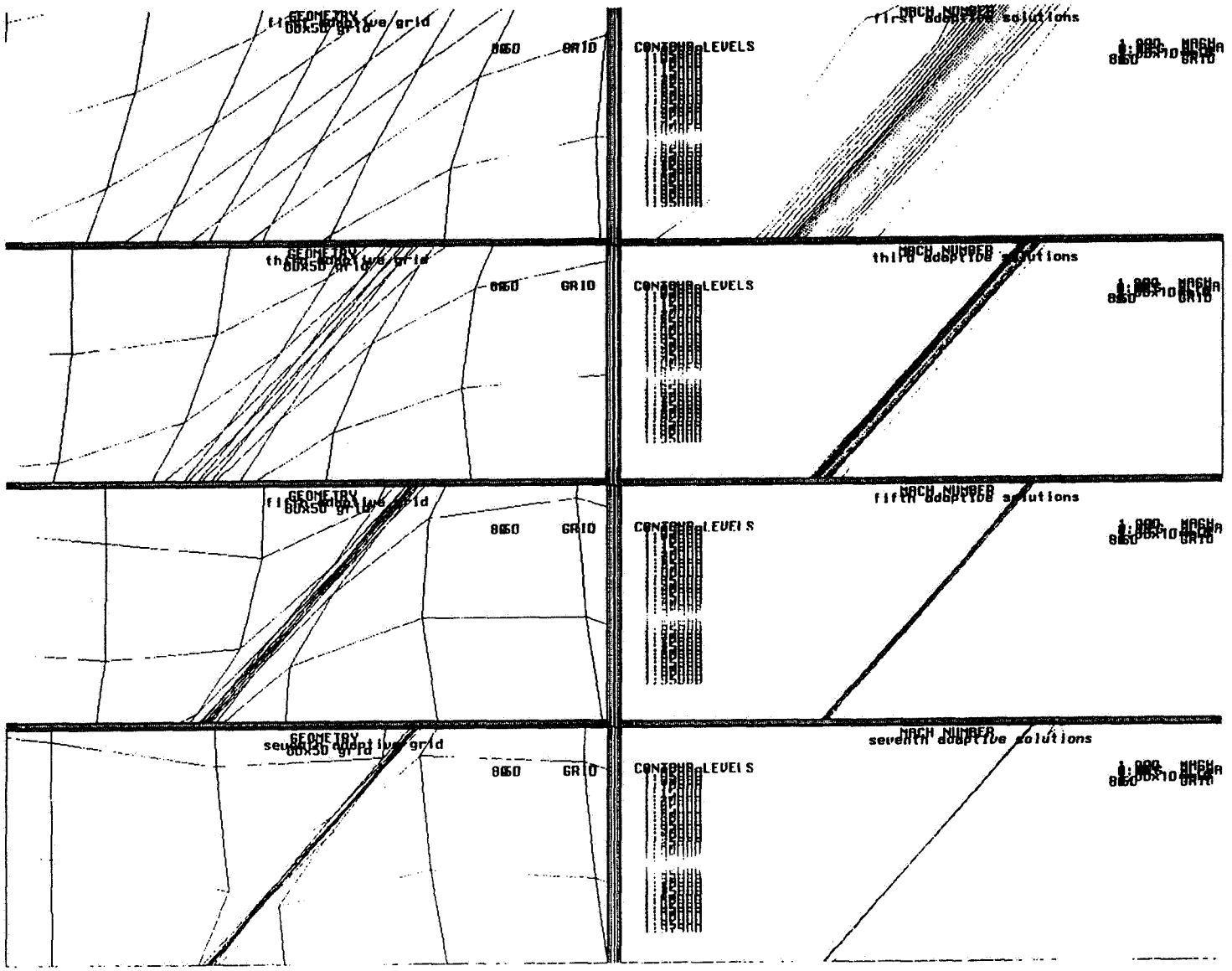


Figure 7.
Single wedge, closer Mach number contour look near shock region.

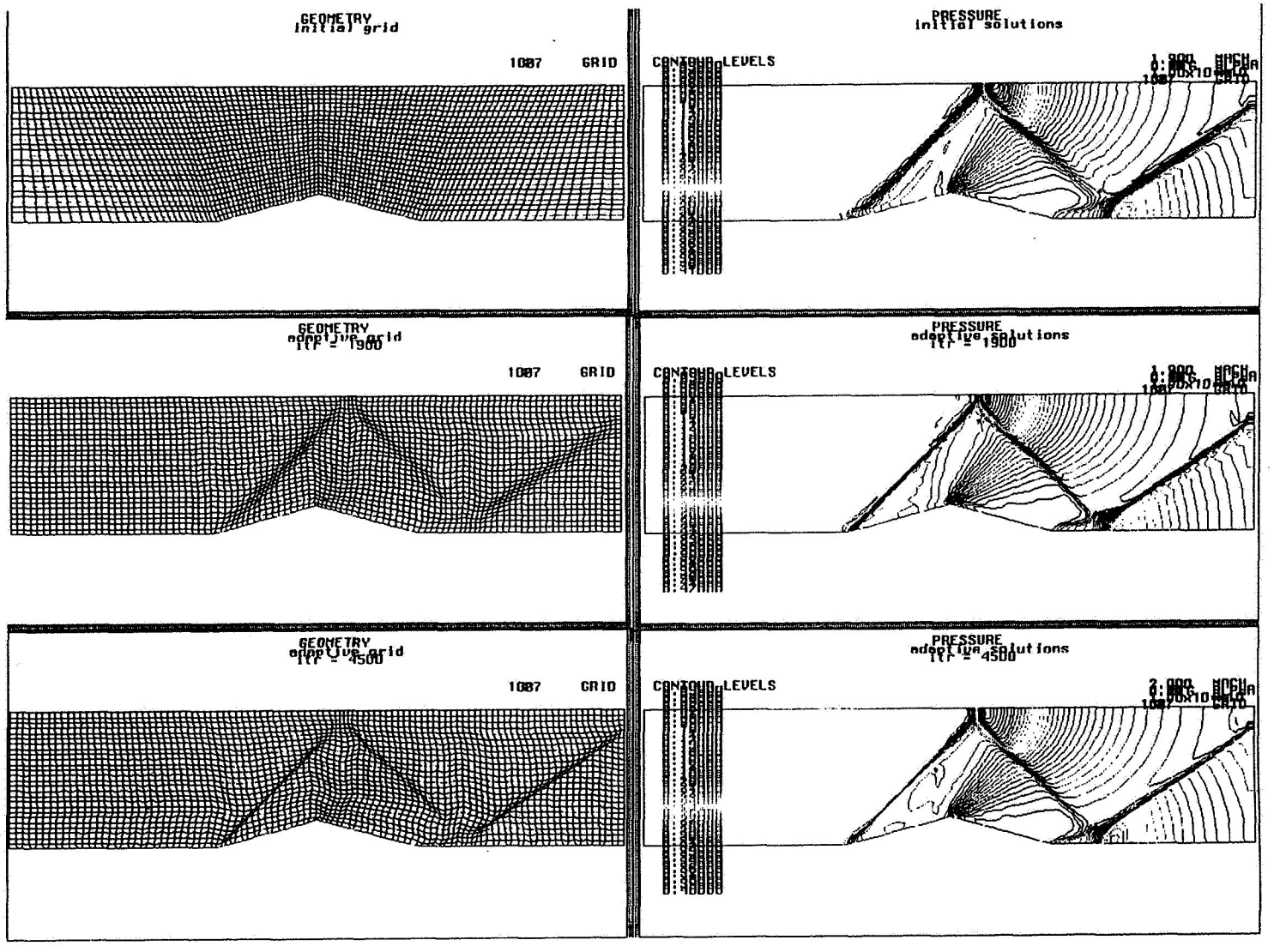
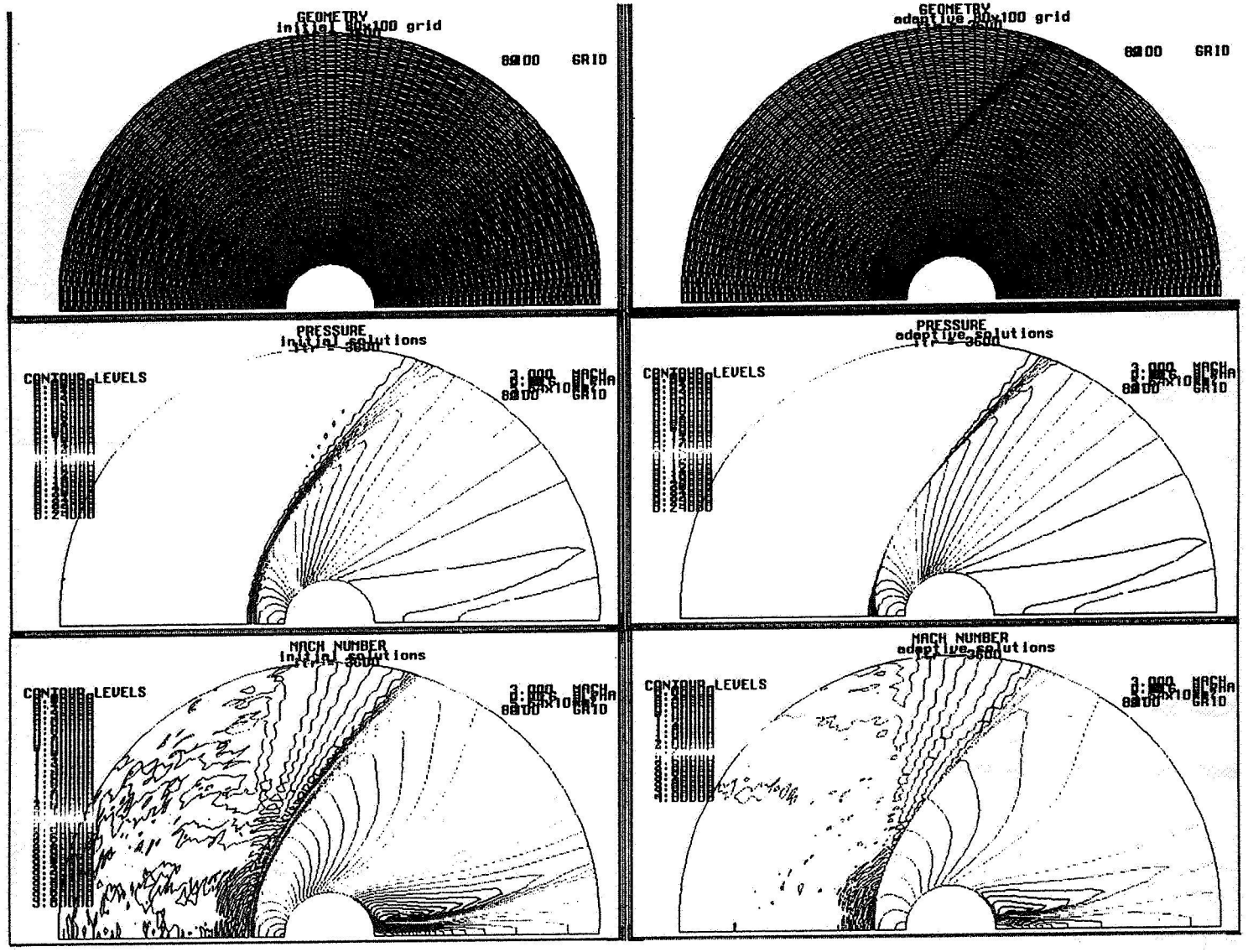


Figure 8.
Double wedge case non-adaptive/adaptive simulation.



Initial grid simulation

Adaptive grid simulation

Figure 9.
Flow around cylinder initial/adaptive cases.

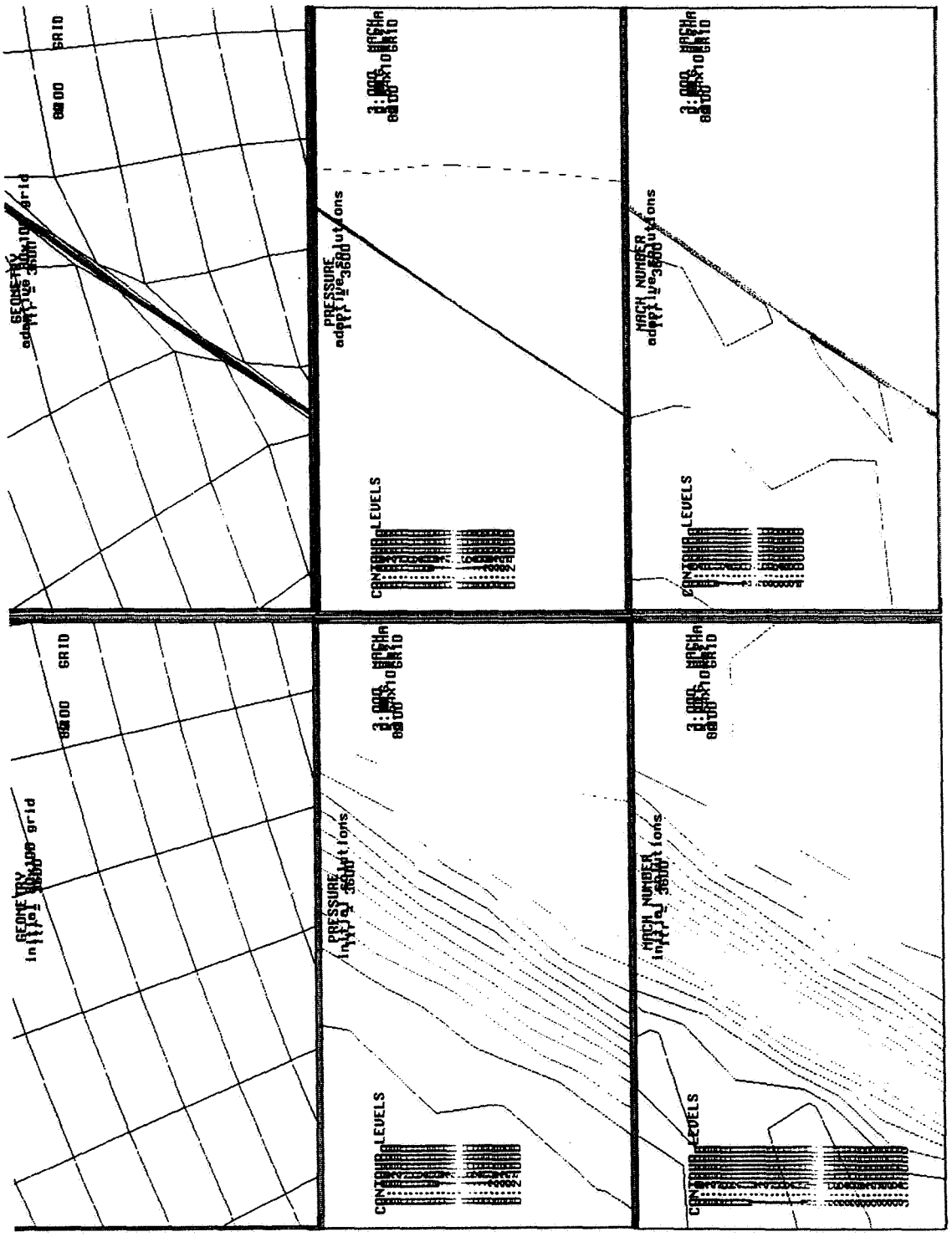


Figure 10.
Flow around cylinder, closer look near shock region.

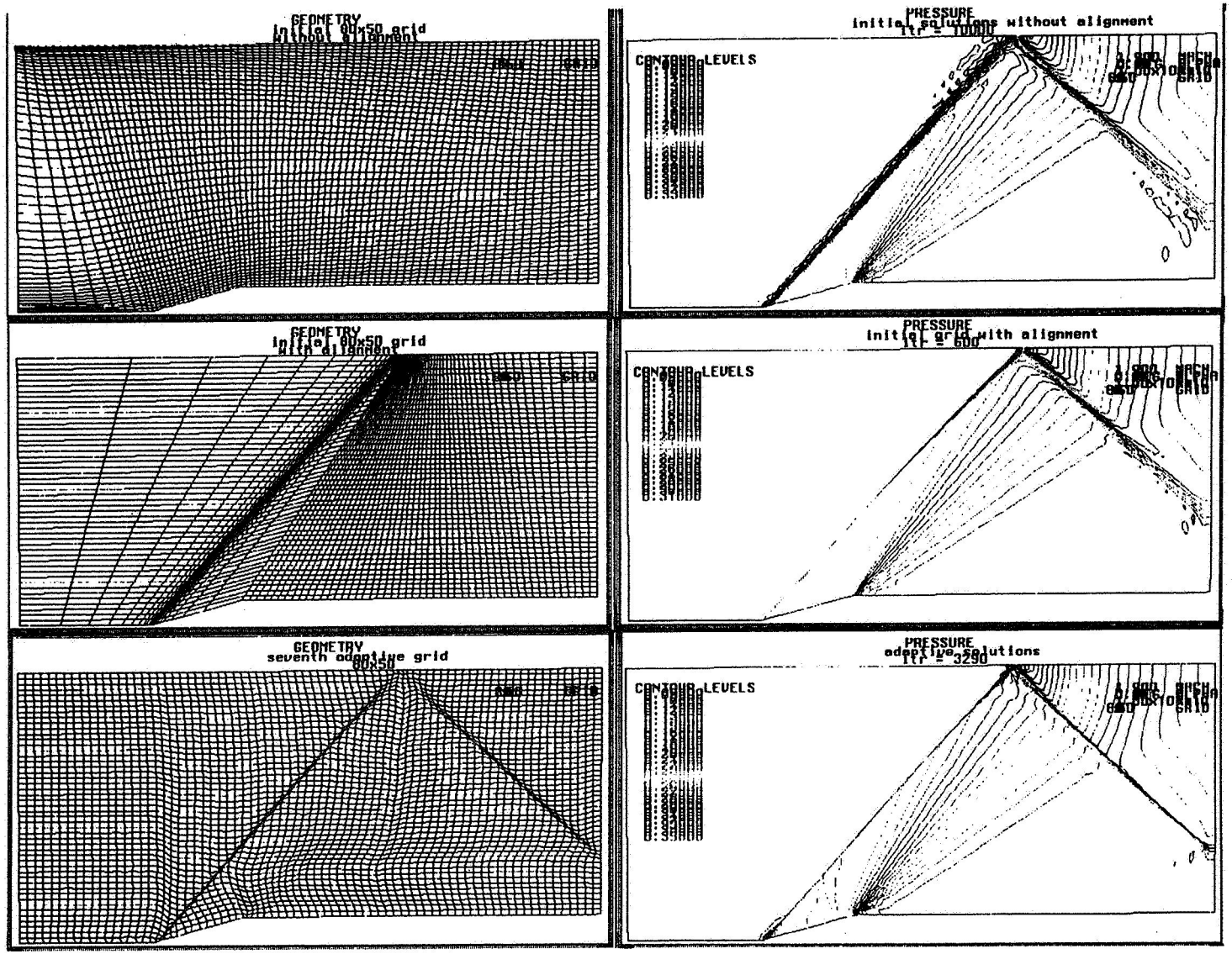


Figure 11. .
Grid lines align with the shock wave capture shock exactly.

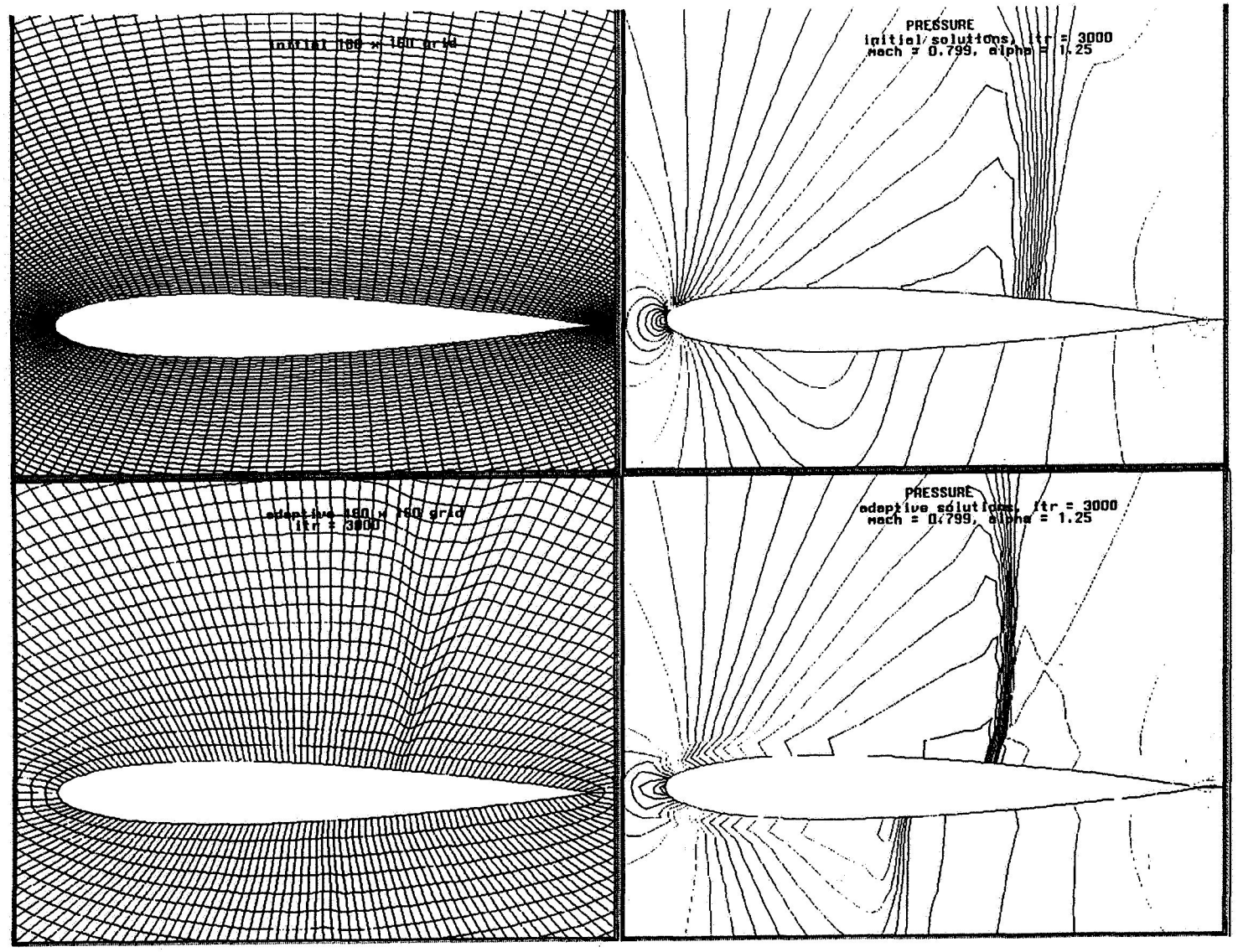


Figure 12.
Airfoil NACA0012 case non-adaptive/adaptive grid/solution.

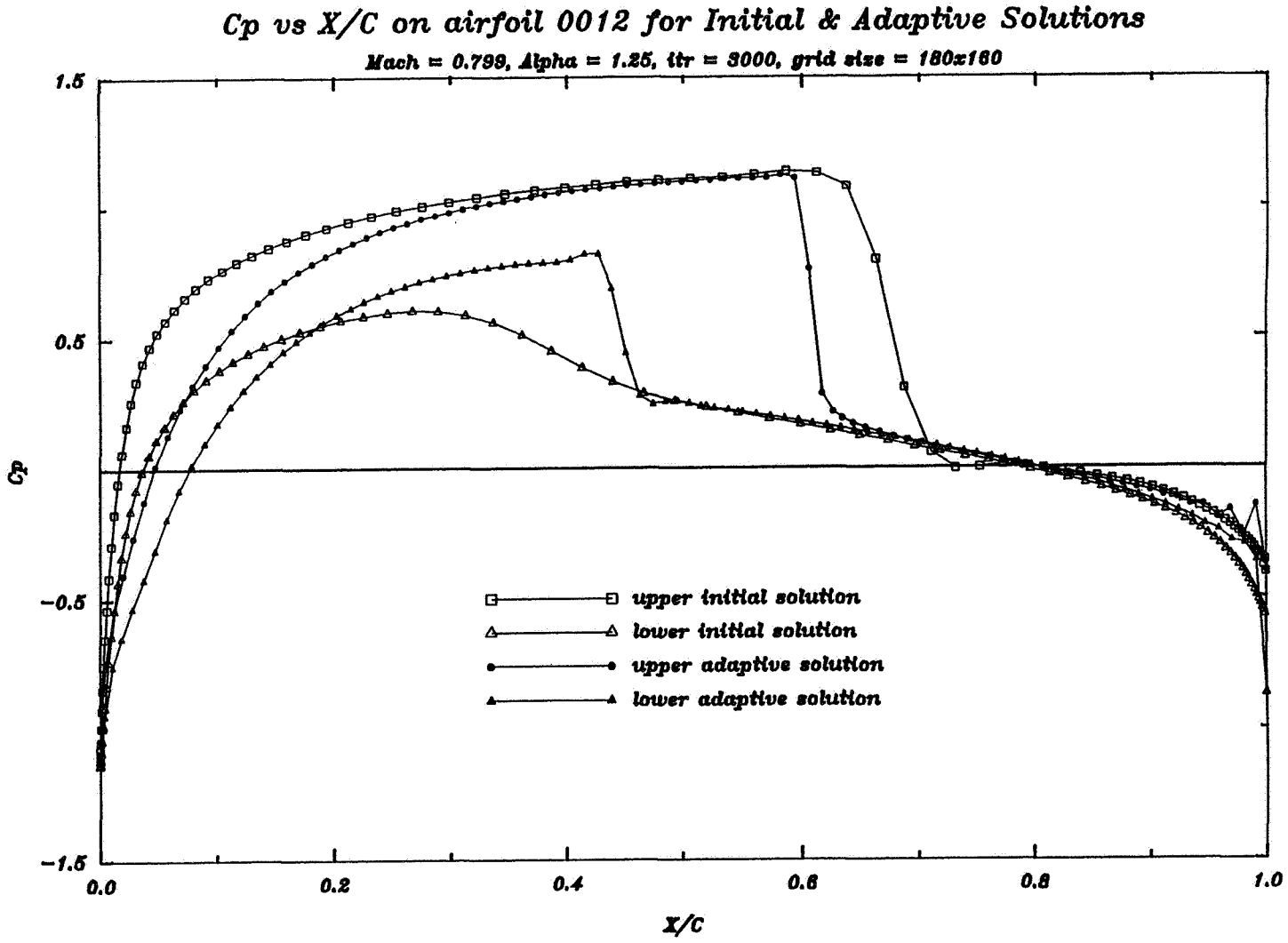


Figure 13.
Airfoil NACA0012 case Cp vs x/c plot.

Using Artificial Intelligence to Control Fluid Flow Computations

Andrew Gelsey
Computer Science Department
Rutgers University
New Brunswick, NJ 08903
gelsey@cs.rutgers.edu

Abstract

Computational simulation is an essential tool for the prediction of fluid flow. Many powerful simulation programs exist today. However, using these programs to reliably analyze fluid flow and other physical situations requires considerable human effort and expertise to set up a simulation, determine whether the output makes sense, and repeatedly run the simulation with different inputs until a satisfactory result is achieved. Automating this process is not only of considerable practical importance but will also significantly advance basic AI research in reasoning about the physical world.

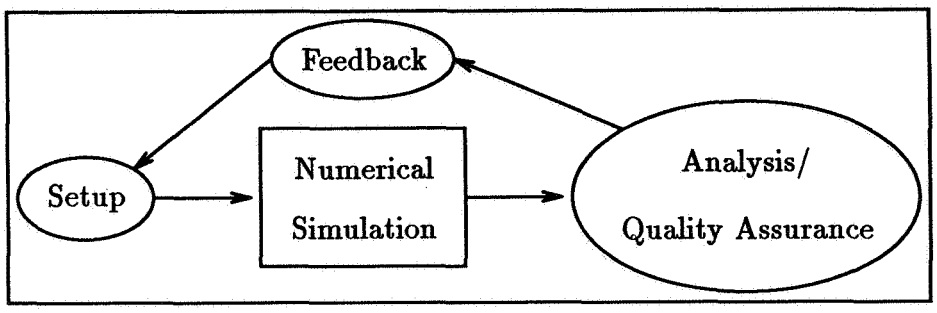


Figure 1: Analyzing a physical situation

1 Introduction

Numerical simulation is an important tool for predicting the behavior of physical systems. Many powerful numerical simulation programs exist today. However, as illustrated in Figure 1, using these programs to reliably analyze a physical situation requires considerable human effort and expertise to

- set up the simulation by transforming a description of the physical situation into a representation the numerical simulation program can successfully process,
- analyze the output of the simulation program to extract desired information and in particular to
- determine whether the output makes sense and how accurate it is likely to be, and if the output is not acceptable, to
- determine how to change the simulation program's input so that it will produce better output.

As a result, these numerical simulation programs typically can't be run successfully by inexperienced users. Perhaps more importantly, these simulation programs can't be reliably invoked by other programs. For example, an automated system for designing complex objects could not easily include a numerical simulation as part of the process it uses to evaluate new designs.

In this paper I will address these problems of setup, analysis, quality assurance, and feedback for numerical simulation.

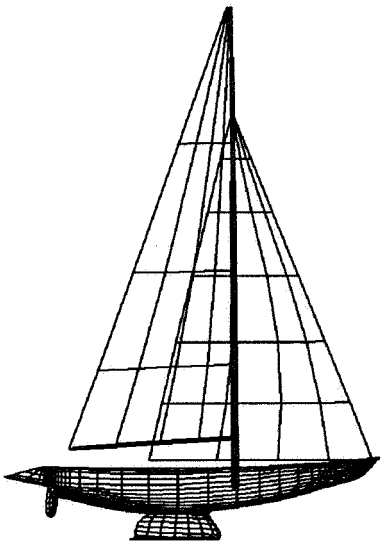
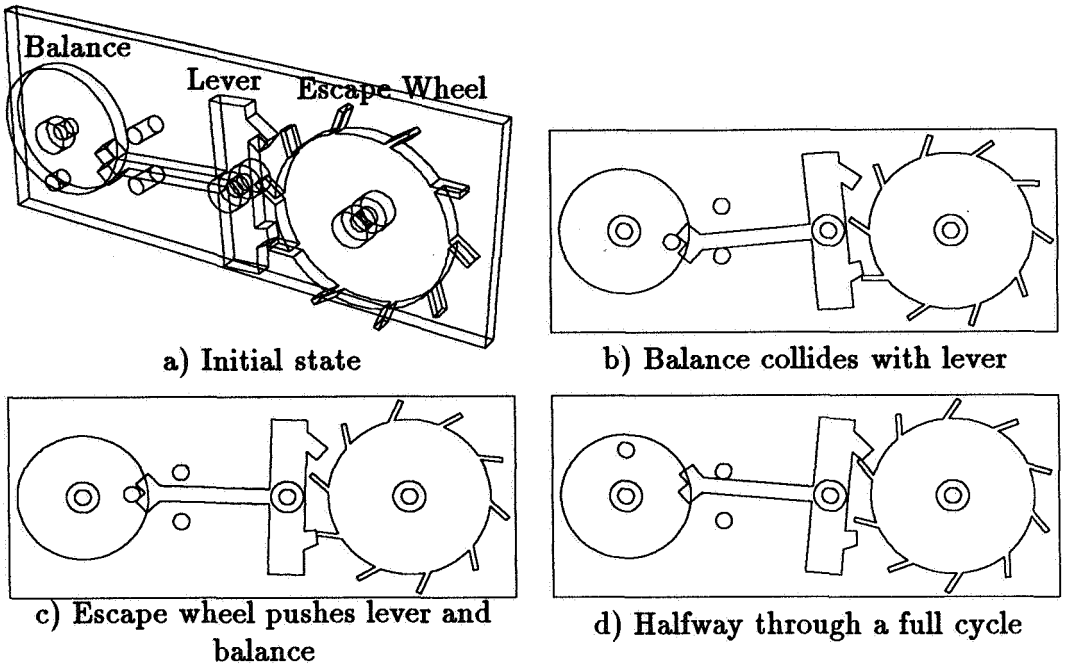


Figure 2: *Stars & Stripes*, winner of the 1987 America's Cup competition

2 Example 1: Racing Yachts

Much of the work on automated interfaces to numerical simulators that I will discuss in this paper has been done in the context of a larger system for automated design of racing yachts like the one in Figure 2. In order to evaluate a possible yacht design, the design system must determine how much time the yacht will need to traverse a specified race course, which will depend on the various forces acting on the yacht. Some of these forces can be computed quite accurately with simple formulas: for example, much of the drag on the yacht is due to an effect known as skin friction, which is directly proportional to the surface area of the yacht. Other forces, however, can only be computed with sufficient accuracy by using powerful numerical simulators. One force in this category is *lift-induced drag*, the main example I will discuss in this paper.

The force the wind exerts on a yacht can be decomposed into two components, one oriented in the direction the yacht is moving and the other perpendicular to the direction of motion. The perpendicular force must be balanced by a force from the yacht's keel, which acts as a lifting surface just like an airplane's wing does, except that for the yacht the lift force is horizontal instead of vertical. The physical effect which generates this lift force also necessarily generates a corresponding drag force, lift-induced drag, which can significantly



a) Initial state

b) Balance collides with lever

c) Escape wheel pushes lever and balance

d) Halfway through a full cycle

Figure 3: Clock or watch escapement mechanism

affect a yacht's performance.

In order to tractably compute this force, a number of modeling assumptions must be made. The key assumption is that viscosity may be neglected when computing lift-induced drag, because viscous contributions to drag will be modeled in other ways, for example as skin friction. If viscosity is neglected, the general nonlinear Navier-Stokes equations of fluid flow may be reduced to a linear partial differential equation, Laplace's equation for velocity potential

$$\nabla^2 \phi = 0$$

The velocity vector of the fluid at any point is given by the gradient of the scalar velocity potential $\phi(x, y, z)$.

3 Example 2: Clockwork Mechanisms

The escapement mechanism in Figure 3 keeps the average speed of a clock or watch constant by allowing the escape wheel, which is pushed clockwise by a strong spring, to advance by only one tooth for each oscillation of the balance.

I've discussed this mechanism in prior publications [Gelsey 1987, Gelsey 1989, Gelsey 1990, Gelsey 1991]. Modeling the behavior of this mechanism requires a series of assumptions. The first assumption is that Newton's laws of motion are adequate for this domain. Newton's laws assume forces are computed using other methods, and the remaining modeling assumptions are models for the various forces. In particular, I've explored two alternative ways of modeling collision forces: a microscopic model in which solid objects are allowed to overlap slightly in space, generating a repulsive force proportional to their depth of overlap (the depth of overlap approximates the slight distortion of the part that is the actual source of the contact force), and a macroscopic model in which moving parts are assumed to collide inelastically and move under geometric constraints from then on. Both models predict similar behaviors for the escapement mechanism.

4 Setting up a Numerical Simulation

A model of a physical situation suitable for numerical simulation must include a set of state variables, quantities whose values represent the state of the physical system. The model must also include a set of differential equations describing the relations between the values of the various state variables. The particular equations for a specific situation must be instantiated from general template equations like Newton's laws or Laplace's equation. The instantiated equations will refer to a specific set of state variables, for example the positions and velocities of the three moving parts of the escapement mechanism in Figure 3, and therefore may be solved by a numerical method which will then compute the values of the state variables for the specific situation being modeled.

I've discussed the problem of identifying state variables and differential equations for clockwork mechanisms extensively in prior publications [Gelsey 1987, Gelsey 1989, Gelsey 1990], so I'll focus here on the fluid flow domain. The state of the fluid is represented by the value of the velocity potential $\phi(x, y, z)$ at every point. For a computer simulation, however, ϕ must be represented by its value at a finite set of discrete points. Picking these points is the principal problem of setting up a numerical simulation of fluid flow. PMARC[1], the numerical simulation program we use to compute lift-induced drag, reformulates Laplace's equation as an integral equation, so instead of

[1]Panel Method Ames Research Center

having to choose points throughout the volume of fluid surrounding the racing yacht, it is only necessary to chose points on the surface of the yacht, by partitioning the surface into a set of pieces called panels.

It is important to emphasize that the initial description of a physical situation to be analyzed typically does not identify state variables. In the case both of mechanical devices and fluid flow, the input consists primarily of a purely geometrical description supplemented by information about various other physical properties like masses of parts in a mechanical device or density of fluid in a flow problem. Currently we represent mechanical device geometry using CAD/CAM Constructive Solid Geometry techniques [Requicha 1980], while ship geometry is represented using B-spline surfaces [Rogers and Adams 1990]. These geometrical descriptions must then be partitioned into pieces corresponding to different state variables.

We have received considerable advice about how to create "good" panelizations of a surface from the domain experts with whom we are working, Drs. Fritts, Salvesen, and Letcher, all members of the design team for the *Stars & Stripes* yacht shown in Figure 2. The principle guidelines we have been given include

- Resolve stagnation points and other areas of rapidly changing potential. The fluid flow hitting a solid body must change its direction in order to pass around one side of the body or the other side. As a result, there are *stagnation points* on the surface of the body where fluid velocity is zero. Since the velocity potential changes rapidly near points like these, small panels must be used in those areas to properly represent the state of the fluid flow.

- Attempt to maintain a nearly orthogonal mesh of panels. Panels should resemble rectangles rather than long, thin diamonds.

- Attempt to keep panel aspect ratio low. Panels should not be far longer than they are wide.

- Control panel expansion ratio. Bound the ratio of the areas of adjacent panels so that very small panels won't have very large neighbors.

- Attempt to orient panel boundaries along streamlines of fluid flow.

Managing the tradeoffs between these goals can be a complex reasoning problem. For example, if velocity potential near stagnation points changes rapidly

in one direction but remains fairly constant in the other, resolution of the stagnation point may best be achieved with long, thin panels which tend to violate aspect ratio constraints. In some cases constraints need to be applied more and less stringently in different areas of the body's surface. For example, a rounded body can't have an extremely orthogonal mesh everywhere, but orthogonality should be maintained as widely as possible.

5 Quality Assurance

A major difficulty in using numerical simulation programs, especially in an automated manner, is that it is generally difficult to tell whether or not to trust the output of the program. A human expert building a model of physics on which to base a numerical simulator makes use of a body of knowledge and assumptions which does not end up in the simulation program itself. For example, PMARC is based on the assumption that viscosity safely can be neglected. This assumption tends to hold much better for long, thin shapes than for short, fat ones. However, PMARC can be run to predict flow around a short, fat shape, and to an inexperienced user the results will look perfectly reasonable, until compared with experimental data, if any is available.

An automated solution to this "quality assurance" problem requires representing and using this body of knowledge and assumptions, which can be broadly classified into the following categories

- Modeling assumptions, e.g. PMARC assumes zero viscosity — these assumptions may or may not be directly testable. An example of a testable assumption from my microscopic model of collision forces in mechanisms is the assumption that the volumes of moving parts will overlap only slightly. If a large overlap occurred during a simulation that would indicate that the simulation output was not trustworthy.

- Expectations about input, e.g. long, thin body — more directly testable properties of the input typically derived from the simplifying assumptions, often by a fairly complex chain of reasoning.

- Expectations about output

  - ranges of plausible output values, e.g. velocities within fluid flowing around a body shouldn't be greater than several times the body's velocity with respect to the surrounding fluid.

  - consistency requirements for output values, e.g. the sum of the kinetic and potential energy in a clockwork mechanism should not increase as time passes
  - qualitative relationships among output values, e.g. fluid flow around a solid body should have a stagnation point where the flow hits the body, then should speed up as the fluid passes around the body again and then slow down again as the fluid merges together and leaves the body.
  - convergence behavior — as space and time discretizations are refined, differential equation solutions should settle down to asymptotic values
  - sensitivity to minor input perturbations — typically, small perturbations should generate small responses
  - response to extreme input values — may be computable from overall trends
  - trusted output results from similar physical configurations
- Test cases with known solutions
- Associated simpler models useful for coarse checking of output
- Associated deeper models to use if satisfactory output cannot be achieved
- Alternative numerical methods that might be appropriate

Some of these criteria could be better applied given a database of past experience. For example, values of lift and drag forces for similar sailing yachts could be used to reject implausible values resulting from a bad simulation.

6 Feedback

Successful analysis of a physical situation will typically require running a numerical simulator several times and intelligently modifying the simulator's input between each run. In the case of fluid flow, the heuristic nature of the guidelines for producing a good panelization make it likely that the first attempt will yield an unacceptable solution. Quality assurance knowledge must then be applied to recognize problems in the solution and attempt to diagnose and correct flaws in the initial panelization that led to these problems. In the

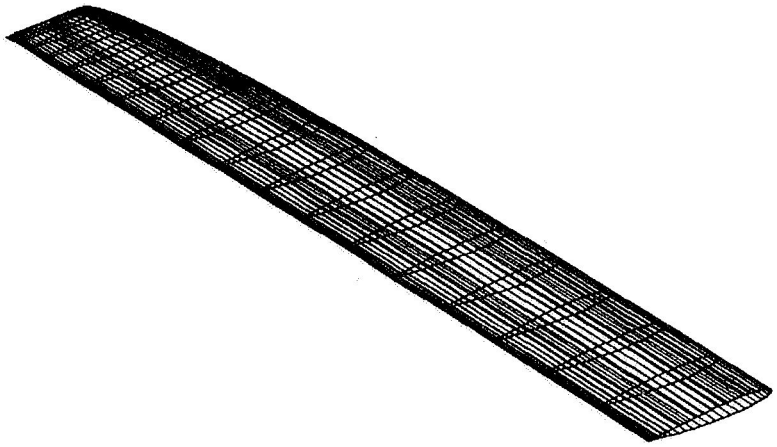


Figure 4: NACA 0012 wing

mechanics domain, feedback between simulation runs is needed not so much to change state variables as to change initial simulation conditions in order to more fully understand the physical situation (see [Gelsey 1991]).

7 Experimental Results

In the fluid flow domain, my colleagues and I have implemented initial versions of automated setup, quality assurance, and feedback for the PMARC numerical simulator mentioned earlier. The resulting system will accept a description of a solid body's geometry and automatically panelize the body's surface and run PMARC on it. However, this initial system does not directly apply the panelization guidelines discussed earlier; instead, it generates panelizations based on a distribution scheme called cosine spacing which tends to create panelizations following the guidelines, at least for fairly simple shapes. At present we have implemented quality assurance checks for most of the panelization guidelines and are in the process of investigating where on a body's surface the various guidelines tend to be most important and how tradeoffs between the guidelines should be handled.

Our automated simulation controller is capable of running a series of nu-

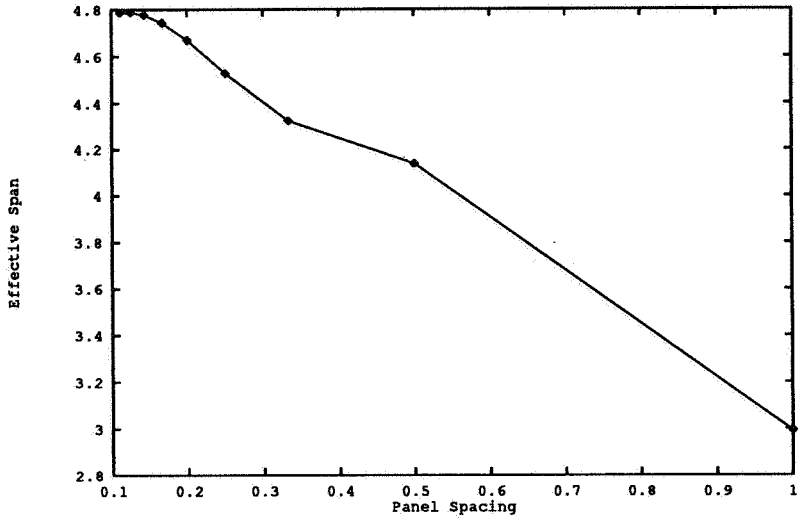


Figure 5: PMARC output for various panelizations

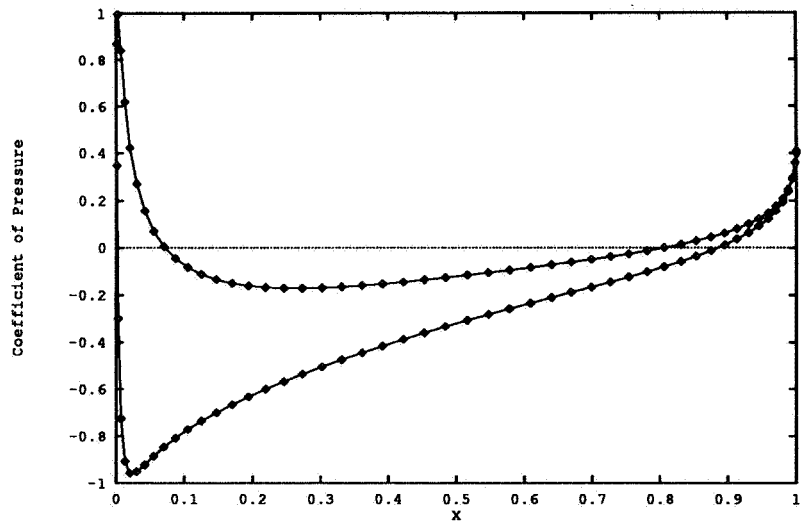


Figure 6: Pressure distribution for fluid flowing in the positive x direction. (Upper curve shows pressure on bottom of wing, lower curve shows pressure on top of wing.)

merical simulations with different panelizations. For example, Figure 5 plots PMARC output as a function of panel spacing for the simple test body in Figure 4. The largest spacing, normalized to 1, was an experiment with only 30 panels on the entire wing, while the smallest spacing, 0.1, covered the wing with 3000 panels. Note that as panel spacing decreases, the output converges to a particular value, as one would expect a reasonable solution to do. The output value here is effective span, the span of an idealized wing with similar performance. The wing in Figure 4 has a span of 5 meters, and the effective span in Figure 5 converges to a slightly lower value, as one would expect because the losses due to spillover between the upper and lower wing surfaces at the wing tip would not be present in an idealized wing. Figure 6 shows the pressure distribution along near the center of the wing as computed by the simulation with finest spacing. After Laplace's equation is solved to compute velocity potential, the coefficient of pressure plotted in Figure 6 is computed using Bernoulli's equation

$$c_p = 1 - \left(\frac{v}{v_{\text{free stream}}}\right)^2$$

Note that the pressure coefficient rises to 1, its maximum possible value, at the leading edge, indicating a stagnation point, and that the qualitative shape of the pressure distribution follows the quality assurance guidelines.

We are experimenting with various feedback schemes. We have tried simple schemes for redistributing panels to refine solution values, but have found that these don't work well. We are in the process of implementing a more sophisticated scheme which will make a more active use of panelization guidelines and quality assurance knowledge.

In the mechanical device domain I've implemented programs to automatically set up and intelligently control numerical simulations of clockwork mechanisms. [Gelsey 1987, Gelsey 1989, Gelsey 1990, Gelsey 1991]. I am currently focusing on the problem of quality assurance for this domain.

8 Related Work

Jambunathan *et al.*[1991] and Andrews[1988] discuss the use of expert systems technology to augment more traditional computational fluid dynamics programs. Most other artificial intelligence research concerning reasoning about physical systems has focused on qualitative rather than numerical simulation. [Weld and de Kleer 1990] Exceptions are the work of Sacks[1991] and Yip[1991];

however, they have focused on numerical simulators for ordinary differential equations and have not addressed the issue of quality assurance. Forbus and Falkenhainer[1990] discusses the use of qualitative simulation to check the quality of numerical simulation results; however, the approach described appears limited to physical situations modeled by ordinary differential equations.

9 Conclusion

While the problems of setup, analysis, quality assurance, and feedback for numerical simulation are clearly of considerable practical importance, their solution should also lead to significant insights into some basic AI problems. Successful use of numerical simulators appears to require a combination of spatial reasoning and reasoning about physics which has so far received little attention from AI researchers. The first step in classifying and understanding these reasoning methods is to collect a set of concrete examples of them, and the only way to be sure the examples fully capture the reasoning used in to incorporate them in programs which can successfully apply the reasoning to the control of numerical simulations. This is the goal of the research discussed in this paper.

10 Acknowledgments

The research on automated use of PMARC was done with fellow Rutgers Computer Science Dept. faculty member Gerard Richter and graduate student Ke-Thia Yao. We worked with hydrodynamicists Martin Fritts and Nils Salvesen of Science Applications International Corp., and John Letcher of Aero-Hydro Inc. Our research was supported by the Defense Advanced Research Projects Agency and the National Aeronautics and Space Administration under NASA grant NAG2-645.

References

[Andrews 1988] Andrews, Alison E. 1988. Progress and challenges in the application of artificial intelligence to computational fluid dynamics. *AIAA Jounal* 26(1):40–46.

[Forbus and Falkenhainer 1990] Forbus, Kenneth D. and Falkenhainer, Brian 1990. Self-explanatory simulations: An integration of qualitative and quantitative knowledge. In *Proceedings, Eighth National Conference on Artificial Intelligence*, Boston, MA. AAAI-90.

[Gelsey 1987] Gelsey, Andrew 1987. Automated reasoning about machine geometry and kinematics. In *Proceedings of the Third IEEE Conference on Artificial Intelligence Applications*, Orlando, Florida. Also appears in Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann, 1990.

[Gelsey 1989] Gelsey, Andrew 1989. Automated physical modeling. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit, Michigan USA.

[Gelsey 1990] Gelsey, Andrew 1990. *Automated Reasoning about Machines.* Ph.D. Dissertation, Yale University. YALEU/CSD/RR#785.

[Gelsey 1991] Gelsey, Andrew 1991. Using intelligently controlled simulation to predict a machine's long-term behavior. In *Proceedings, Ninth National Conference on Artificial Intelligence.*

[Jambunathan *et al.* 1991] Jambunathan, K.; Lai, E.; Hartle, S. L.; and Button, B. L. 1991. Development of an intelligent front-end for a computational fluid dynamics package. *Artificial Intelligence in Engineering* 6(1):27–35.

[Requicha 1980] Requicha, Aristides A. G. 1980. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys* 12:437–464.

[Rogers and Adams 1990] Rogers, David F. and Adams, J. Alan 1990. *Mathematical elements for computer graphics.* McGraw-Hill, 2nd edition.

[Sacks 1991] Sacks, Elisha P. 1991. Automatic analysis of one-parameter ordinary differential equations by intelligent numeric simulation. *Artificial Intelligence* 48(1).

[Weld and de Kleer 1990] Weld, Daniel S. and Kleer, Johande, editors 1990. *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, San Mateo, California.

[Yip 1991] Yip, Kenneth 1991. Understanding complex dynamics by visual and symbolic reasoning. *Artificial Intelligence* 51(1–3).

ON CONSTRUCTING THREE DIMENSIONAL OVERLAPPING GRIDS WITH CMPGRD

William D. Henshaw
Geoffrey Chesshire
Michael E. Henderson
IBM Research Division
Thomas J. Watson Research Centre
Yorktown Heights, NY 10598

ABSTRACT

We describe some techniques for the construction of three-dimensional composite overlapping grids using the grid construction program CMPGRD. The overlapping approach can be used to generate grids for regions of complicated geometry. The grids can be constructed to be smooth and free from coordinate singularities. The ability to create smooth grids for complicated regions is an important first step towards the accurate numerical solution of partial differential equations. We describe how to create grids for surfaces defined by cross-sections such as an airplane wing. We also describe how the patched surfaces generated from a CAD package can be used within the CMPGRD program and how grids can be created in regions where surfaces intersect.

INTRODUCTION

We describe some techniques for the construction of three-dimensional composite overlapping grids using the grid construction program CMPGRD. A composite overlapping grid consists of a set of logically rectangular component grids which cover a region and overlap where they meet. Interpolation conditions connect solution values defined on the grid. In the paper *Composite Meshes for the Solution of Partial Differential Equations* [1] we described CMPGRD and discussed techniques for the solution of elliptic and hyperbolic partial differential equations (PDEs) on overlapping grids. Our examples were limited to two-dimensional problems. In this paper we emphasize the extensions made to the code for the generation of three-dimensional grids. The overlapping approach can be used to generate grids for regions of complicated geometry. The region can be divided into a number of sub-domains for which a component grid can be more easily created. The component grids can be constructed to be smooth and free from coordinate singularities. Once all component grids have been created the CMPGRD program will automatically determine the interpolation conditions which connect the grids. CMPGRD has a very general algorithm for doing this which supports any number of component grids overlapping in any order. CMPGRD can generate the overlapping grids appropriate for higher order interpolation, higher order discretization, cell-centred or cell-vertex grids and the sequence of grids which can be used for the multigrid algorithm [2]. Some other references which describe overlapping grids and the solution of PDEs thereon include [3] [4] [5] [6] [7].

The problem of grid construction can be made simpler by using the composite overlapping grid technique. However, the task of creating component grids, especially in three-dimensions, can still

be very difficult. In this paper we discuss some ways to create three-dimensional component grids suitable for use with CMPGRD. The emphasis here is on techniques which are closely linked to the overlapping approach and to methods which avoid coordinate singularities.

We first describe how to create grids for surfaces defined by cross-sections, such as an ellipsoid or wing, where the cross-sections may converge to a point at one or both ends. The basic underlying problem here is the creation of a composite grid about a sphere which is free from singularities. A solution to this problem is to cover the sphere with more than one patch. We use an orthographic projection to create grids over the north and south poles of the sphere. This approach for a sphere can be generalized to more general surfaces which are defined by cross-sections: a separate patch is placed on the end of the surface where the cross-sections converge to a point.

A complicated object such as an airplane consists of multiple surfaces which connect to each other, such as when a wing joins a fuselage. Using the cross-section technique component grids can be created for the wing and fuselage but there remains the problem of connecting the wing to the fuselage. The connection may either be smooth, in which case a fillet grid is appropriate, or the connection may be a corner in which case the curve of intersection between the two surfaces should be an edge of a grid. We consider the latter case here. One way to create a grid for the wing which matches to the fuselage is to project the end face of the wing grid onto the surface of the fuselage. We present an example of this technique. We have developed another, more general, approach for creating grids in the region where surfaces intersect. This method begins by computing the curve of intersection between the surfaces and then reparameterizing the surface using the intersecting curves to define the portion of the surface to use. We give several examples to illustrate this approach.

For the description of complicated three-dimensional regions there are many advantages to using a computer aided design (CAD) package. We show how the patched surfaces generated from a CAD package can be used within the CMPGRD program. Although the patched surface may be smooth, the parameterization of the surface may not be smooth and it is thus necessary to reparameterize the surface. These reparameterized surfaces can then be used in CMPGRD to create overlapping grids. We show an example of a grid for a wing connected to an engine nacelle by a pylon.

CREATING THREE-DIMENSIONAL OVERLAPPING GRIDS

A composite overlapping grid consists of a set of component grids. Each component grid covers a portion of the computational region. Component grids overlap where they meet. Solution values are matched by interpolation at the overlapping boundary. In order to determine how to interpolate between grids CMPGRD requires knowledge of the component grid mapping not only at grid points but also at all intermediate positions. Thus the component grid must be defined as a continuous mapping.

Component grid mapping: For the purposes of CMPGRD a component grid is defined as a smooth mapping from a unit cube, $\mathbf{r}$, to the computational domain $\mathbf{x} \in R^3$. CMPGRD needs to evaluate this mapping at any point $\mathbf{r}$ in the unit cube and requires both the image of the mapping, $\mathbf{x}(\mathbf{r})$, and its derivatives $\partial \mathbf{x}(\mathbf{r})/\partial \mathbf{r}$:

$$\text{Component Grid Mapping}: \mathbf{r} \rightarrow (\mathbf{x}, \frac{\partial \mathbf{x}}{\partial \mathbf{r}}).$$

A component grid has a number of characteristics associated with it. For example, each face of the

component grid has a boundary condition. The face may be part of a true boundary, or the grid may be periodic, or the derivative of the grid may be periodic or the face may be used for interpolation.

CMPGRD provides various features for creating two-dimensional and three-dimensional component grids. One approach is for the user to provide a complete description of the component grid which is supplied to CMPGRD as an *externally* defined grid (i.e. Fortran subroutine). In this case one must supply the mapping from the unit cube to the computational domain and the derivatives of this mapping. This approach is useful if a grid has been created by some other package. If the component grid mapping is only known at a set of grid points then the mapping can be defined everywhere by using interpolation. CMPGRD has such a component grid interpolation routine.

Another way to define a three-dimensional grid is to first define a surface in three-dimensions as an externally defined curve. This surface can then be automatically extended in the normal direction to create a three-dimensional grid. Alternatively two offset surfaces can be blended to form a space filling grid.

Sphere in a Box: As a first example of creating a three-dimensional composite grid we consider the problem of generating a smooth grid for the region exterior to a sphere and interior to a box in which the sphere sits. We describe two ways to create a composite grid for this problem.

A grid around the sphere can be created using a mapping defined by the standard spherical-polar coordinate transformation:

$$\begin{aligned} \text{Sphere} : (r_1, r_2, r_3) &\rightarrow (x_1, x_2, x_3) \\ (\theta, \phi, R) &= (2\pi r_1, \pi(r_2 - .5), R_a + (R_b - R_a) r_3) \\ (x_1, x_2, x_3) &= (R\cos(\theta)\cos(\phi), R\sin(\theta)\cos(\phi), R\sin(\phi)) \end{aligned}$$

This is not the recommended method to create a grid since the mapping has singularities at the north and south poles. These singularities will spell trouble when one wants to solve a PDE on the grid. In fact CMPGRD must know which grids are singular in order for its algorithm to work properly.

By using multiple patches it is possible to cover the surface of a sphere with grid transformations so that there are no coordinate singularities. At each pole we create a grid defined by an orthographic projection, see figure (1). Consider the set of points $(s_1, s_2, -R)$ on the plane which is tangent to the sphere (radius R) at the south pole. The orthographic projection associates a point $(s_1, s_2, -R)$ on this plane with a point $\mathbf{x} = (x_1, x_2, x_3)$ on the sphere as the intersection between the sphere and the line through the north pole and the point $(s_1, s_2, -R)$. This point of intersection is given by

$$\begin{aligned} \text{Orthographic} : (s_1, s_2) &\rightarrow (x_1, x_2, x_3) \\ (x_1, x_2, x_3) &= (\rho\cos(\theta), \rho\sin(\theta), -\zeta) \\ &= \left(\frac{\alpha^2 s_1}{(\alpha^2 + s^2)}, \frac{\alpha^2 s_2}{(\alpha^2 + s^2)}, \alpha\frac{(\alpha^2 - s^2)}{(\alpha^2 + s^2)}\right) \end{aligned}$$

where

$$\alpha = 2R \quad , \quad s^2 = s_1^2 + s_2^2 \quad , \quad \cos(\theta) = \frac{s_1}{s} \quad , \quad \sin(\theta) = \frac{s_2}{s}$$

$$\rho = \alpha\frac{\alpha^2 s}{\alpha^2 + s^2} \quad , \quad \zeta = \alpha\frac{\alpha^2 - s^2}{\alpha^2 + s^2} \quad , \quad \zeta^2 + \rho^2 = R^2$$

To complete the definition of the grid transformation we define the relationship between the unit square coordinates (r_1, r_2) and the (s_1, s_2) coordinates,

$$s_1 = (r_1 - .5)\ s_a \quad , \quad s_2 = (r_2 - .5)\ s_b$$

where the constants s_a, s_b determine the extent of the orthographic patch. For later reference we note the derivatives of the orthographic transformation:

$$\frac{\partial \rho}{\partial s} = \frac{\rho}{s}\frac{\zeta}{R} \quad , \quad \frac{\partial \zeta}{\partial s} = -\frac{\rho^2}{R} \quad , \quad \frac{d\zeta}{d\rho} = \frac{-\rho}{\zeta} \tag{1}$$

$$\frac{\partial s}{\partial s_1} = \frac{s_1}{s} \quad , \quad \frac{\partial s}{\partial s_2} = \frac{s_2}{s} \quad , \quad \frac{\partial \theta}{\partial s_1} = -\frac{\sin(\theta)}{s} \quad , \quad \frac{\partial \theta}{\partial s_2} = \frac{\cos(\theta)}{s} \tag{2}$$

The full three-dimensional component grid is defined by extending each surface in the normal direction. The box is covered with a simple rectangular grid. The composite grid is shown in figures (3) and (4).

Surfaces defined by cross-sections: The method described in the previous section can be generalized to create grids for surfaces defined by cross sections. We assume that the surface is defined as a function of a periodic variable θ, $0 \le \theta \le 2\pi$, and an axial variable ζ, $-1 \le \zeta \le +1$:

$$\mathbf{x} = \mathbf{f}(\theta, \zeta)$$

When ζ is fixed $\mathbf{f}(\theta, \zeta = \zeta_0)$ will define a cross-sectional curve on the surface. The cross-sections should converge to a point at $\zeta = \pm 1$. We will also see that the cross-sections must also tend to an elliptical shape as $\zeta \to \pm 1$ in order that there be no singularity in the derivative of the mapping at the poles.

The cross-sectional surface will be covered by three patches: a central cylindrical patch and two end patches defined by orthographic projections. The central patch is defined by the mapping

$$\begin{aligned} C_2 : (r_1, r_2) &\to (x_1, x_2, x_3) \\ \mathbf{x} &= \mathbf{f}(\theta, \zeta) \\ (\frac{\partial \mathbf{x}}{\partial r_1}, \frac{\partial \mathbf{x}}{\partial r_2}) &= (2\pi \frac{\partial \mathbf{f}}{\partial \theta}, \zeta_c \frac{\partial \mathbf{f}}{\partial \zeta}) \\ \text{where } \theta = 2\pi\, r_1 \quad &, \quad \zeta = \zeta_c\,(r_2 - .5) \end{aligned}$$

The constant $\zeta_c \le 2$ determines the portion of the sphere covered by the central patch. When $\zeta_c = 2$ the central patch will extend all the way to each pole. The orthographic patch at the $\zeta = +1$ pole is defined by

$$C_1 : (r_1, r_2) \to (x_1, x_2, x_3) \tag{3}$$

$$\mathbf{x} = \mathbf{f}(\theta, \zeta) \tag{4}$$

$$\frac{\partial \mathbf{x}}{\partial r_1} = \frac{\rho}{s}\Big[-\sin(\theta)\{\frac{1}{\rho}\frac{\partial \mathbf{f}}{\partial \theta}\} - \frac{\cos(\theta)}{R}\{\rho \frac{\partial \mathbf{f}}{\partial \zeta}\}\Big] \tag{5}$$

$$\frac{\partial \mathbf{x}}{\partial r_2} = \frac{\rho}{s}\Big[\cos(\theta)\{\frac{1}{\rho}\frac{\partial \mathbf{f}}{\partial \theta}\} - \frac{\sin(\theta)}{R}\{\rho \frac{\partial \mathbf{f}}{\partial \zeta}\}\Big] \tag{6}$$

where ρ, s and θ are the variables associated with the orthographic patch:

$$s_1 = (r_1 - .5)\, s_a \quad , \quad s_2 = (r_2 - .5)\, s_b \quad , \quad s^2 = s_1^2 + s_2^2$$

$$\cos(\theta) = \frac{s_1}{s} \quad , \quad \sin(\theta) = \frac{s_2}{s} \quad , \quad \rho = \frac{\alpha s}{\alpha^2 + s^2} \quad , \quad \zeta = \alpha \frac{\alpha^2 - s^2}{\alpha^2 + s^2} \quad , \quad \rho^2 + \zeta^2 = R^2$$

The Jacobian derivatives of the mapping, equations (5), (6) are determined in the standard way using the chain rule

$$\begin{aligned}\frac{\partial \mathbf{x}}{\partial r_1} &= \frac{\partial \theta}{\partial r_1}\frac{\partial \mathbf{f}}{\partial \theta} + \frac{\partial \zeta}{\partial r_1}\frac{\partial \mathbf{f}}{\partial \zeta} \\ &= \frac{-\sin(\theta)}{s}\frac{\partial \mathbf{f}}{\partial \theta} + \frac{-\rho^2}{R}\cos(\theta)\frac{\partial \mathbf{f}}{\partial \zeta} .\end{aligned}$$

These derivatives are written in the special form of equations (5),(6). We assume that we have a subroutine which defines the cross-sections $\mathbf{f}(\theta, \zeta)$ and also certain derivatives of the cross-sections:

$$\text{Cross-section routine: } (\theta, \zeta) \rightarrow (\mathbf{f}(\theta, \zeta), \frac{1}{\rho}\frac{\partial \mathbf{f}}{\partial \theta}, \rho\frac{\partial \mathbf{f}}{\partial \zeta}) \tag{7}$$

We choose to have the routine return $(\frac{1}{\rho}\frac{\partial \mathbf{f}}{\partial \theta}, \rho\frac{\partial \mathbf{f}}{\partial \zeta})$ as opposed to $(\frac{\partial \mathbf{f}}{\partial \theta}, \frac{\partial \mathbf{f}}{\partial \zeta})$ in order to avoid inaccuracy in numerically computing the Jacobian. In fact, in order that the derivatives of the patches exist at the poles the cross-sectional function $\mathbf{f}$ must satisfy

$$\lim_{s\to 0} \left[\cos(\theta)\frac{\partial \mathbf{f}}{\partial \rho} - \frac{\sin(\theta)}{s}\frac{\partial \mathbf{f}}{\partial \theta}\right] = C_1 \tag{8}$$

$$\lim_{s\to 0} \left[\sin(\theta)\frac{\partial \mathbf{f}}{\partial \rho} + \frac{\cos(\theta)}{s}\frac{\partial \mathbf{f}}{\partial \theta}\right] = C_2 \tag{9}$$

where the constants C_1, C_2 must be independent of θ. These conditions will be satisfied provided that the cross-sections tend to an elliptical shape at the poles:

$$f_i(\theta, \zeta) \sim \rho(a\cos(\theta) + b\sin(\theta)) .$$

Condition (8), for example, follows from writing equations (5) in the form

$$\frac{\partial \mathbf{x}}{\partial r_1} = \left[\cos(\theta)\frac{\partial \mathbf{f}}{\partial \rho} - \frac{\sin(\theta)}{s}\frac{\partial \mathbf{f}}{\partial \theta}\right] + \frac{1}{s^2}\left(\frac{\rho}{s}\frac{\zeta}{R} - 1\right)(s\cos(\theta))\left(s\frac{\partial \mathbf{f}}{\partial \rho}\right)$$

and noting that

$$\lim_{\rho\to 0}\frac{1}{s^2}\left(\frac{\rho}{s}\frac{\zeta}{R} - 1\right)(s\cos(\theta)) = 0 .$$

Ellipsoid: A grid for an ellipsoid with principal axes of lengths (a, b, c) can be created using the cross-section function

$$\begin{aligned}\text{Ellipsoid} : (\theta, \zeta) &\rightarrow (x_1, x_2, x_3) \\ (x_1, x_2, x_3) &= (f_1, f_2, f_3) = (a\rho\cos(\theta), b\rho\sin(\theta), c\zeta)\end{aligned}$$

with Jacobian derivatives defined by

$$\begin{bmatrix} \frac{1}{\rho}\partial f_1/\partial\theta & \frac{1}{\rho}\partial f_2/\partial\theta & \frac{1}{\rho}\partial f_3/\partial\theta \\ \rho\partial f_1/\partial\zeta & \rho\partial f_2/\partial\zeta & \rho\partial f_3/\partial\zeta \end{bmatrix} = \begin{bmatrix} -a\sin(\theta) & b\cos(\theta) & 0 \\ -a\zeta\cos(\theta) & -b\zeta\sin(\theta) & c\rho \end{bmatrix}$$

The resulting grid is shown in figure (5).

Wing with Joukowsky Cross-Section: A slightly more complicated problem involves the generation of a grid around an wing which sits in a box. For simplicity we define the wing to have cross-sections which are Joukowsky airfoils. At the wing tip the cross-sections are forced to become elliptical since this is a necessary requirement for the grid to be smooth. The mappings are thus defined as

$$\begin{aligned} \text{Joukowsky wing} : (r_1, r_2) &\rightarrow (x_1, x_2, x_3) \\ (x_1, x_2, x_3) &= (f_1(\zeta), g(\rho) f_2(\theta, \zeta), g(\rho) f_3(\theta, \zeta)) \end{aligned}$$

The cross section functions f_2 and f_3 are defined as Joukowsky airfoils:

$$\begin{aligned} f_2 + i f_3 &= w + 1/w \\ w &= a e^{i\theta} + i d e^{i\delta} f_4(\rho) \\ f_4 &= g(\rho) - g'(1)\rho \end{aligned}$$

The function $g(\rho)$ (recall $\rho = \sqrt{1-\zeta^2}$) is chosen to be $g(\rho) = \tanh(\beta\rho)$ and causes the cross-sections to converge to a point. The function f_4 satisfies $f_4(\rho = 0) = 0$ causing the airfoil to become elliptical at the tip and $f_4'(\rho = 1) = 0$ causing the cross-section at $(\zeta, \rho) = (0, 1)$ to lie in the in the $x_2 - x_3$ plane. We make use of this latter result when we attach the wing to a fuselage in the next section. For the airfoil shown the free parameters defining the airfoil were taken as

$$a = .85 \quad , \quad d = .15 \quad , \quad \delta = 15^o = 15(\pi/180)^r \quad , \quad \beta = 2$$

In order to have sufficient grid lines in locations of highest curvature we stretch the grid lines at the leading and trailing edges.

Wing on a fuselage - first approach: In this section we consider the problem of attaching a wing to a fuselage. Here we describe a relatively simple procedure to accomplish the joining. Later we describe another solution to this problem. The procedure is illustrated in figure (7) and consists of the steps:

1. Create component grids for the wing and for the fuselage.
2. Deform the end of the wing grid to be the same shape as the piece of the fuselage it will attach to.
3. Position the deformed wing onto the fuselage.

For step 1 we use the wing grid as defined in the previous section. For simplicity we define the fuselage to be part of a cylinder:

$$\begin{aligned} \text{Cylinder} : (r_1, r_2) &\rightarrow (x_1, x_2) \\ \theta = \pi(r_1 - .5) \quad &, \quad R = R_1 + R_2 r_2 \\ (x_1, x_2, x_3) &= (R\cos(\theta), y_a + y_{ba} r_3, R\sin(\theta)) \ . \end{aligned}$$

To perform step 2 we deform the end of the wing to lie on a cylinder. For our geometry this is accomplished with the mapping

$$x_1 \leftarrow x_1 + R_1 - \left\{ R_1 - \sqrt{R_1^2 - x_3^2}\, e^{-\alpha x_1} \right\}$$

which transforms the end of the airfoil onto the cylinder. The mapping has an exponentially small effect on parts of the wing which are far from the end (figure (7b). Finally we move the wing to join the fuselage. The resulting composite grid will have the topology shown in figure (7c).

In general, to perform step 2 of deforming the end of the wing we will need to know how to map a plane (or the end face of the wing grid) onto the surface of the fuselage. It is also important to perform this deformation without changing the surface shape of the wing. The composite grid for a Joukowsky wing attached to a cylinder in a box is shown in figure (6). The figure shows a cut through the wing and cylinder. The interpolation points are shown.

This example illustrates some important advantages and disadvantages of composite grids. The advantage was gained because the grid generation problem was simplified by decomposing the domain into simpler problems. The disadvantage came when the grids had to be joined together along a physical boundary.

M6 wing: In this section we use the ideas of the previous section to create a grid for the ONERA M6 wing which is free from artificial coordinate singularities. The M6 wing does, however, have a sharp trailing edge. The geometry of the M6 wing is defined by a sequence of cross-sections. Each cross-section is defined by a set of points. We fit a cubic spline to each cross section, parameterized by pseudo-arclength. Thus if the cross-section at $\zeta = \zeta_i$ is defined by the points

$$(f_2(\theta, \zeta_i), f_3(\theta, \zeta_i)) \ : \ \{(y_j, z_j)\}_{j=1}^{n_j}$$

then we define the pseudo-arclength by $s_1 = 0$ and

$$s_j = s_{j-1} + C\sqrt{(y_j - y_{j-1})^2 + (z_j - z_{j-1})^2} \quad j = 2, 3, \ldots, n_j \ ,$$

where C is chosen so that $s_{n_j} = 1$. We fit cubic splines

$$\begin{aligned} Y_i(s) &= \text{spline}(\{(s_j, y_j)\}) \\ Z_i(s) &= \text{spline}(\{(s_j, z_j)\}) \end{aligned}$$

We use (linear) interpolation between cross-sections to define the surface of the wing everywhere:

$$\begin{aligned} f_2(\theta, \zeta) &= (1-\alpha)Y_i(s) + \alpha Y_{i+1}(s) \\ f_3(\theta, \zeta) &= (1-\alpha)Z_i(s) + \alpha Z_{i+1}(s) \\ s &= \theta/(2\pi) \\ \alpha &= (\zeta - \zeta_i)/(\zeta_{i+1} - \zeta_i) \ , \quad \zeta_i \le \zeta \le \zeta_{i+1} \end{aligned}$$

The cross-sections at the wing tip should converge to a point. Moreover in order that the tip be smooth the cross-sections should converge to an ellipse. To ensure these conditions we slightly deform the tip of the wing. The definition of the cross-sections for the wing are thus

$$\begin{aligned} \text{M6 wing} : (r_1, r_2) &\rightarrow (x_1, x_2, x_3) \\ (x_1, x_2, x_3) &= (L\zeta, \ \tanh(\beta\rho) f_2(\theta, \zeta), \ \tanh(\beta\rho) f_3(\theta, \zeta)) \end{aligned}$$

where L is the length of the wing and where β is chosen sufficiently large so as to not significantly change the shape of the wing tip. The composite grid for an M6 attached to a cylinder is shown in figure (8).

Component grids created from CAD Package: A common representation of surfaces in computer aided design (CAD) packages is that of a Coons patch [8], also known as transfinite interpolation [9]. The Coons patch is defined as a mapping from the unit square into three-space. In this representation the unit square (r_1, r_2) is divided into an array of sub-patches. There are $n_i \times n_j$ such subpatches. Each subpatch is defined as a polynomial, mapping the unit square (u_1, u_2) (of the subpatch) into $\mathbf{x} \in R^3$.

$$\text{Subpatch: } \mathbf{x}_{ij}(u_1, u_2) = \sum_{m=0}^{n_u-1} \sum_{n=1}^{n_v-1} \hat{\mathbf{x}}^{ij}_{mn} u_1^m u_2^n \tag{10}$$

where $\hat{\mathbf{x}}^{ij}_{mn}$ are constants. The subpatch boundaries, $r_1 = r_{1,i}$, $i = 1, ..., n_i - 1$ and $r_2 = r_{2,j}$, $j = 1, ..., n_j - 1$, are equally spaced in the (r_1, r_2) plane,

$$r_{1,i} = \frac{i}{n_1 + 1} \quad , \quad i = 1, ..., n_i \quad , \quad r_{2,i} = \frac{j}{n_2 + 1} \quad , \quad j = 1, ..., n_j \; .$$

The full mapping is defined as

$$\mathbf{x}(r_1, r_2) \;=\; \mathbf{x}_{ij}\left(\frac{r_1 - r_{1,i}}{r_{1,i+1} - r_{1,i}}, \frac{r_2 - r_{2,j}}{r_{2,j+1} - r_{2,j}}\right) \quad \text{for} \quad r_{1,i} \le r_1 \le r_{1,i+1} \quad \text{and} \quad r_{2,j} \le r_2 \le r_{2,j+1}$$

The smoothness of the multi-patch surface $\mathbf{x}(r_1, r_2)$ is obtained by placing constraints on the subpatch mappings at the boundaries of the sub-patches. Often patches are not parametrically smooth but only geometrically smooth: the derivatives with respect to r_1 or r_2 may not be continuous across different subpatches. In general we require parametric smoothness of surfaces if we wish to use the grid to solve a PDE problem. In this case we may want to reparameterize the patch to create a smooth parameterization. We are working on a variety of methods for this purpose. In figure (9) we show a composite grid for some surfaces created from the CATIA computer aided design package. This grid shows a wing connected to an engine nacelle by a pylon.

CREATING COMPONENT GRIDS FROM INTERSECTING SURFACES

Curves of Intersection: The first step in defining a grid in the region where two surfaces intersect is to define and parameterize the curve of intersection. Even assuming that the curve of intersection is well defined, consisting of one connected component, the parameterization of this curve is not well defined. This parameterization will be important when we wish to create a grid.

Suppose we have two surfaces, $\mathbf{S}_i : [0,1]^2 \rightarrow \mathbf{R}^3$, $i = 1, 2$ mapping the unit square into three-dimensional space,

$$\mathbf{S}_i \;=\; \{\mathbf{x} = \mathbf{S}_i(\mathbf{r}_i) \;:\; \mathbf{r}_i = (r_i, s_i) \in [0,1]^2\}$$

and suppose these surfaces intersect along some curve (figure (10)). Denote this curve of intersection by $\mathbf{C} : [0,1] \rightarrow \mathbf{R}^3$,

$$\mathbf{C} = \{\mathbf{x} = \mathbf{C}(s) \;:\; s \in [0,1]\}.$$

As the curve $\mathbf{C}(s)$ traces out a path on the surface $\mathbf{S}_i$ it will also trace out a path on the unit square which is mapped to the surface $\mathbf{S}_i$. Denote this two-dimensional curve on the unit square of $\mathbf{S}_i$ by $\mathbf{R}_i : [0,1] \rightarrow [0,1]^2$,

$$\mathbf{R}_i \;=\; \{\mathbf{r} = \mathbf{R}_i(s) \;:\; s \in [0,1]\}.$$

The curves $\mathbf{R}_i(s)$ satisfy

$$\mathbf{C}(s) = \mathbf{S}_i(\mathbf{R}_i(s)) \qquad s \in [0,1], \quad i = 1,2 \tag{11}$$

We see that the curve of intersection can be represented in three ways, $\mathbf{C}(s)$, $\mathbf{S}_1(\mathbf{R}_1(s))$ and $\mathbf{S}_2(\mathbf{R}_2(s))$. For consistency we want a numerical representation for these curves so that the defining conditions (11) are satisfied precisely (up to round off errors). This means that when we evaluate the curve we must solve the nonlinear equations (11), using Newton's method for example. That is given a value of s we determine $\{\mathbf{C}(s), \mathbf{R}_1(s), \mathbf{R}_2(s)\}$ satisfying (11). We will keep an accurate approximation to the curve to use as a starting guess for Newton. One reason we require such a precise definition for the curve of intersection is that the curve may be used to define a grid which has very small spacings between grid lines. In this case a less accurate representation of $\mathbf{C}$ or $\mathbf{R}_i$ may lead to erroneous results when interpolation points are computed for the overlapping grid.

One way to choose the parameterization of $\mathbf{C}(s)$ is to use one of r_1, s_1, r_2, s_2 to parameterize the curve. For example if the curve $\mathbf{R}_1(s)$ is a single valued function of s_1, as in figure (10), then we may use $s = r_1$. The equations defining $\mathbf{C}(s)$ would be

$$\begin{aligned} \mathbf{C}(s) &= \mathbf{S}_i(\mathbf{r}_i) \quad i = 1,2 \\ s_1 &= s \end{aligned} \tag{12}$$

For given s there are 7 equations for the 7 unknowns $\mathbf{C}(s)$, $\mathbf{r}_i = \mathbf{R}_i(s)$, $i = 1,2$. These equations can be solved by bisection in the variable r_1. Another way to choose the parameterization of $\mathbf{C}(s)$ is to base the parameterization on another curve which is close to $\mathbf{C}$. Thus suppose we have another approximate intersection curve $\mathbf{C}_A(s)$. Given s we define $\mathbf{C}(s)$ to be the point on $\mathbf{C}$ which intersects the plane which is normal to the curve $\mathbf{C}_A$ at the point $\mathbf{C}_A(s)$, see figure (11).

$$\begin{aligned} \mathbf{C}(s) &= \mathbf{S}_i(\mathbf{r}_i) \quad i = 1,2 \\ \frac{\partial \mathbf{C}_A}{\partial s} \cdot (\mathbf{C}(s) - \mathbf{C}_A(s)) &= 0 \end{aligned}$$

These seven equations define the seven unknowns $\mathbf{C}(s)$, $\mathbf{r}_i = \mathbf{R}_i(s)$, $i = 1,2$.

Component grids defined using the curves of intersection: We consider a few examples showing how a component grid can be constructed making use of the curves of intersection. In these examples the curve of intersection, $\mathbf{C}(s)$, between the two surfaces will become the edge of a component grid. The faces of the component grid adjacent to this edge will lie on the respective surfaces. To parameterize these faces we will make use of the curves $\mathbf{R}_i(s)$, corresponding to $\mathbf{C}(s)$, which lie in the unit square coordinates. By construction, the grid we create will have faces that lie precisely (up to round off errors) on the surfaces.

The geometry of the first example is shown in figure (12) where two concentric cylinders intersect two planes. The goal is to create the grid $\mathbf{x} = \mathbf{G}(\mathbf{r})$ which has faces coming from the cylinders and planes as shown in figure (13). Figure (12) shows the 4 unit squares and the curves of intersection. To define the faces of $\mathbf{G}$ we use the portions of the surfaces which are bounded by the curves of intersection. In the unit square coordinates the faces will correspond to the regions bounded by the images of the curves of intersection. The mapping for the face, $\mathbf{G}_1(r_1, r_2, 0)$ corresponding to the end at $r_3 = 0$ of $\mathbf{G}$ can be defined as the composite transformation,

$$\mathbf{G}_1(r_1, r_2, 0) = \mathbf{S}_1(\mathbf{P}_1(r_1, r_2)),$$

where the mapping $\mathbf{P}_1$ in the $\mathbf{r}_1$ plane can be defined, for example, by transfinite interpolation

$$\mathbf{r} = \mathbf{P}_1(r_1, r_2) = (1 - r_2)\mathbf{R}_1(r_1) + r_2\ \mathbf{R}_2(r_1).$$

The other three faces are defined in a similar manner (figure 13). The grid $\mathbf{G}$ is then defined as a transfinite interpolation between these four faces

$$\begin{aligned}\mathbf{G}(r_1, r_2, r_3) \;=\; & (1 - r_1)\mathbf{G}_1(0, r_2, r_3) + r_1\mathbf{G}(1, r_2, r_3) + (1 - r_3)\mathbf{G}_1(r_1, r_2, 0) + r_3\mathbf{G}(r_1, r_2, 1) \\ & -\Big\{(1 - r_1)\Big[(1 - r_3)\mathbf{G}_1(0, r_2, 0) + r_3\mathbf{G}(0, r_2, 1)\Big] \\ & \quad +r_1\Big[(1 - r_3)\mathbf{G}_1(1, r_2, 0) + r_3\mathbf{G}(1, r_2, 1)\Big]\Big\}\end{aligned}$$

Figure (14) shows some surfaces and a grid that have been created using these techniques. Note that this parameterization of the grid $\mathbf{G}$ will only be smooth if the parameterizations of the curves of intersection, $\mathbf{C}_i(s)$ are defined in a consistent way. A major catastrophe occurs, for example, if opposite faces have parameterizations defined in opposite directions so the grid folds over on itself. In a less severe case the grid lines may become highly skewed. Even if the parameterization is still single valued we wish to avoid such cases since finite difference approximations are usually less accurate for skewed grids.

Grid for two intersecting spheres: In the second example we create a grid to be used in the region where two spheres (actually spherical shells) intersect. This *collar* grid will be used to join the spheres and looks something like a torus. Two of the faces of this collar grid lie on the the surfaces of the spheres. The other two faces lie on spherical shells which are offset from the spheres. The collar grid is shown in figure (15). The composite grid for the two spheres in shown in figure (16). Only the grids on the surfaces of the spheres are shown but the collar grid can be recognized.

Wing on a fuselage - approach 2: We use the techniques for creating grids from intersecting surfaces to connect a wing to a fuselage. We first define surfaces for a cylinder and a wing with Joukowsky cross-sections. Another wing-like surface is defined by offsetting the wing in the normal direction. The two wing surfaces are intersected with the cylindrical surface and a wing grid is defined in the region bounded by the intersections. The composite grid is shown in figure (17).

CONCLUSIONS

We have given a brief description of some techniques for the construction of overlapping grids in three space dimensions. We have shown how to create grids for surfaces that are defined by cross-sections. We have described some ways to create grids in the regions where surfaces intersect. These grids are defined by smooth transformations and are free from artificial coordinate singularities. The ability to create smooth grids for complicated regions is an important first step towards the accurate numerical solution of PDEs on such regions. The results presented in this paper show some of the potential advantages of using overlapping grids to create grids for three dimensional geometries. However, there is still much work to be done to make the grid construction problem easier and more automatic.

References

[1] G. Chesshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *J. Comp. Phys.*, 90(1):1–64, 1990.

[2] W. D. Henshaw and G. Chesshire. Multigrid on composite meshes. *SIAM J. Sci. Stat. Comput.*, 8(6):914–923, 1987.

[3] P. G. Buning, I. T. Chiu, S. Obayashi, Y. M. Rizk, and J. L. Steger. Numerical simulation of the integrated space shuttle vehicle in ascent. paper 88-4359-CP, AIAA, 1988.

[4] J. L. Steger and J. A. Benek. On the use of composite grid schemes in computational aerodynamics. *Computer Methods in Applied Mechanics and Engineering*, 64:301–320, 1987.

[5] D. L. Brown. A finite volume method for solving the Navier-Stokes equations on composite overlapping grids. In B. Engquist and B. Gustafsson, editors, *Third International Conference on Hyperbolic Problems*, pages 141–158. Chartwell-Bratt, 1991.

[6] G. Chesshire and W. D. Henshaw. Conservation on composite overlapping grids. IBM Research Report RC 16531, IBM Research Divison, Yorktown Heights, NY, 1991.

[7] D. L. Brown, G. Chesshire, and W. D. Henshaw. Getting started with CMPGRD, introductory user's guide and reference manual. report LA-UR-89-1294, Los Alamos National Laboratory, 1989.

[8] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, 1988.

[9] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin. *Numerical Grid Generation.* North-Holland, New York, 1985.

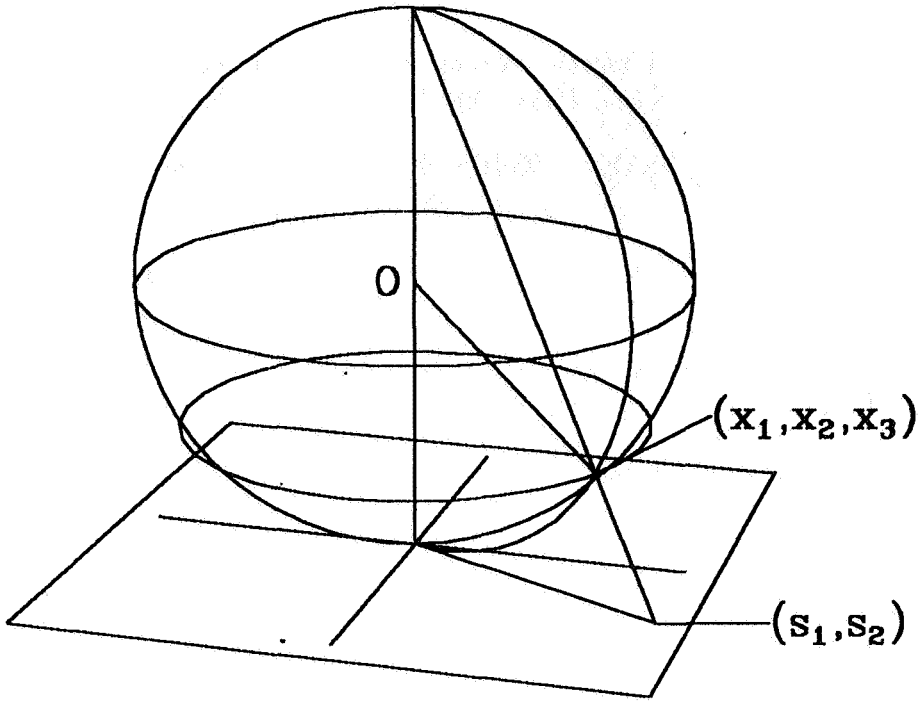


Figure 1: Orthographic projection

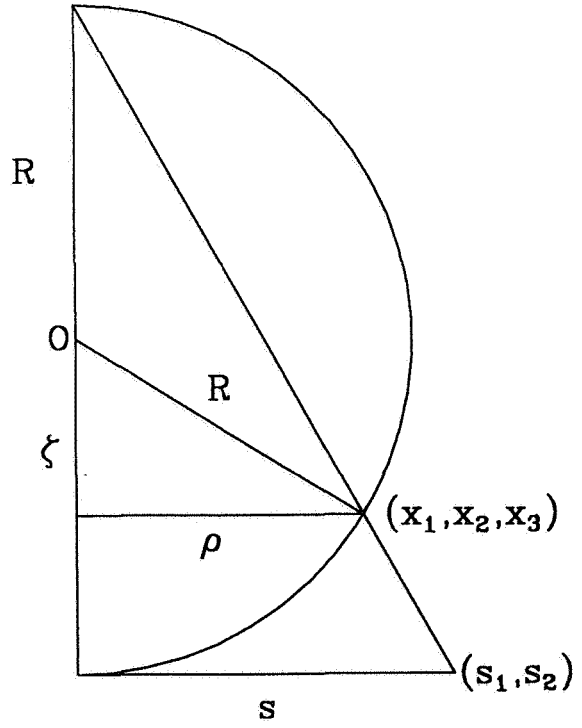


Figure 2: Variables associated with the orthographic projection

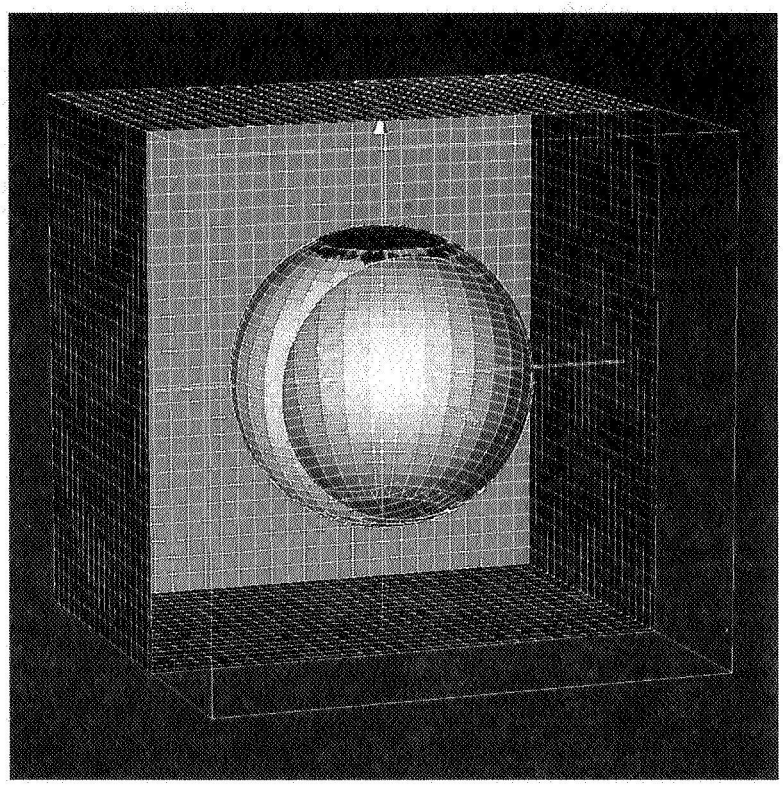


Figure 3: Overlapping grid for a sphere

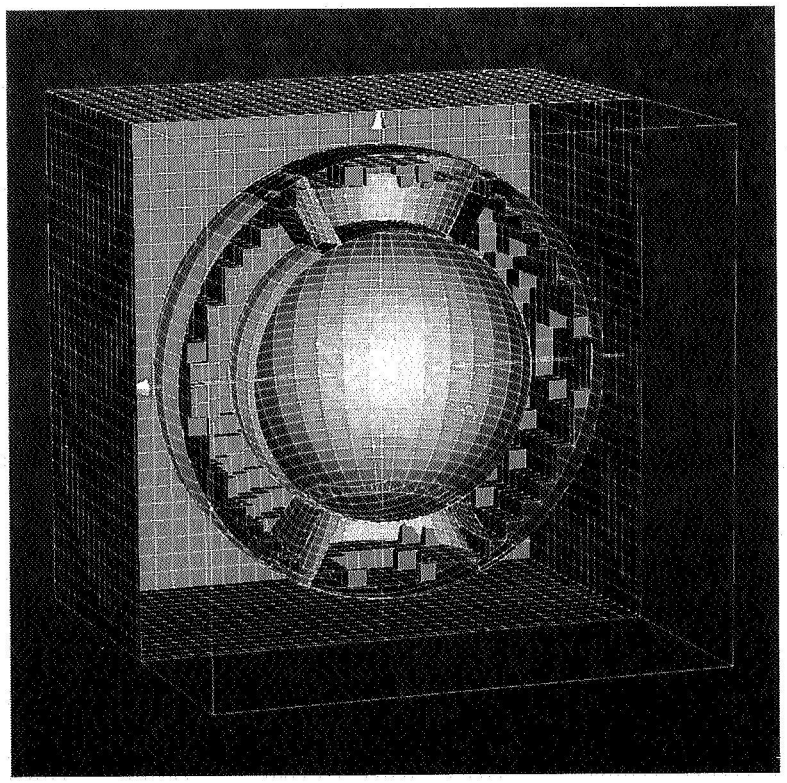


Figure 4: Overlapping grid for a sphere showing interpolation points

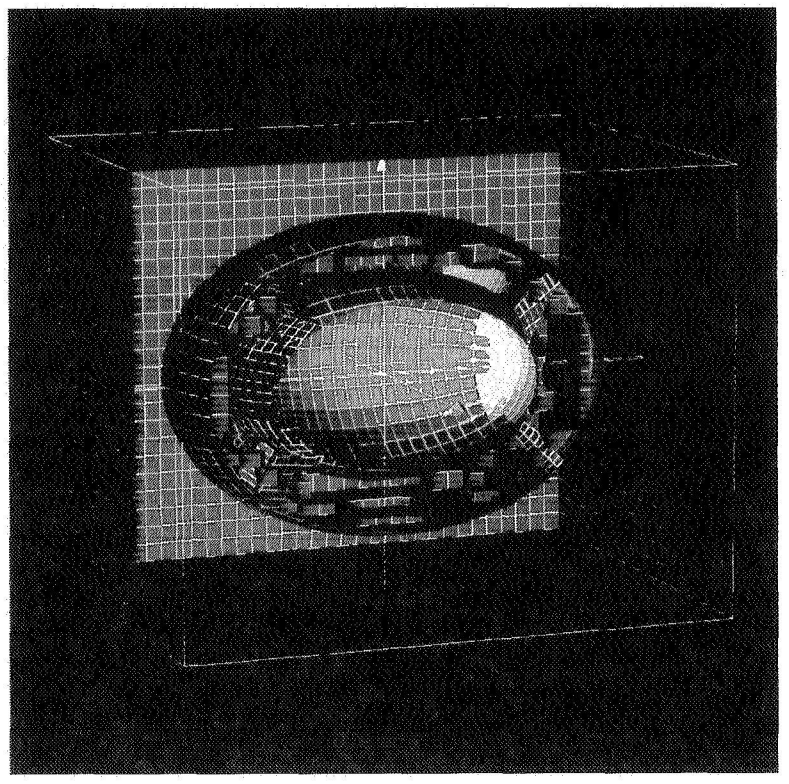


Figure 5: Overlapping grid for an ellipse showing interpolation points

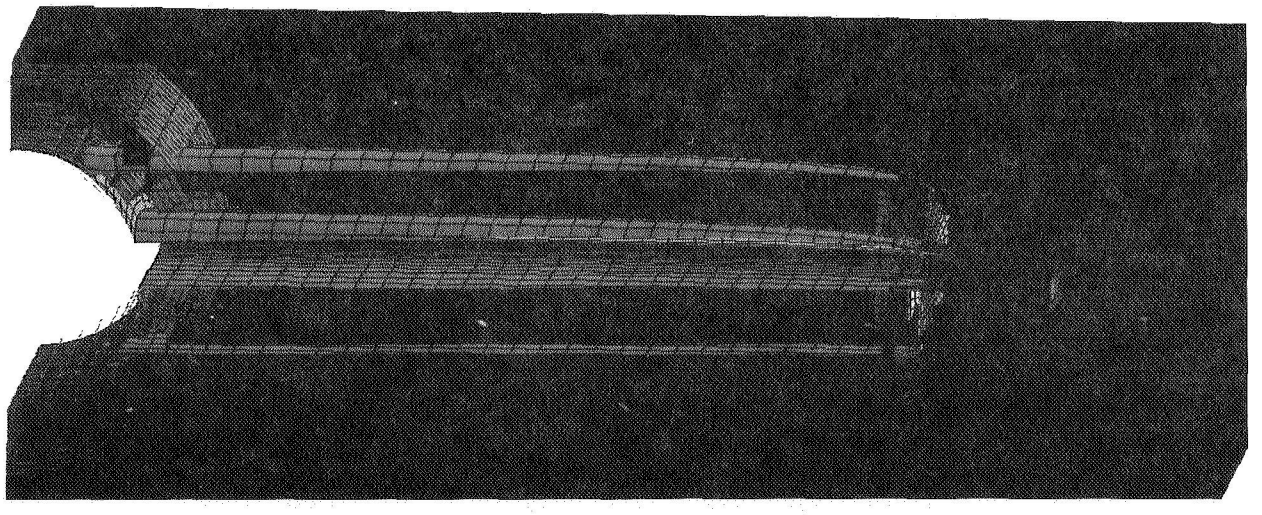


Figure 6: Overlapping grid for Joukowsky wing on a cylinder

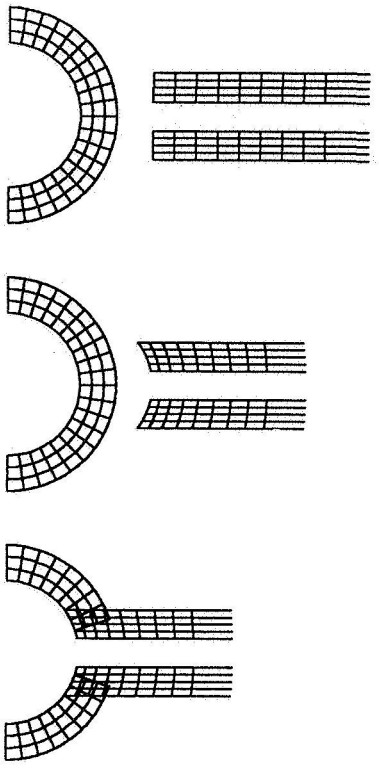


Figure 7: Attaching a wing to a fuselage - approach 1

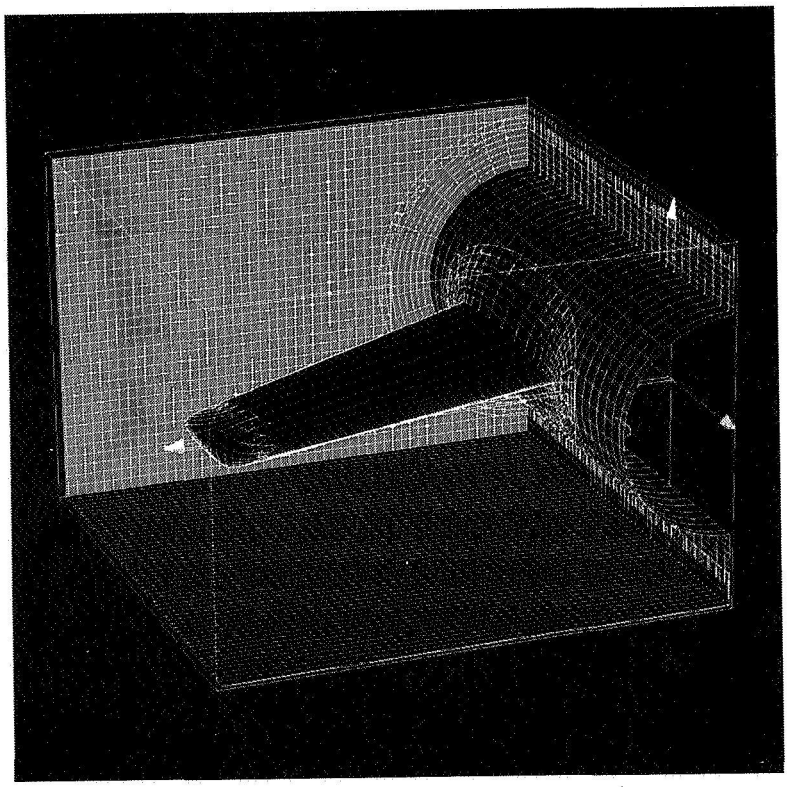


Figure 8: Overlapping grid for an M6 wing on a cylinder

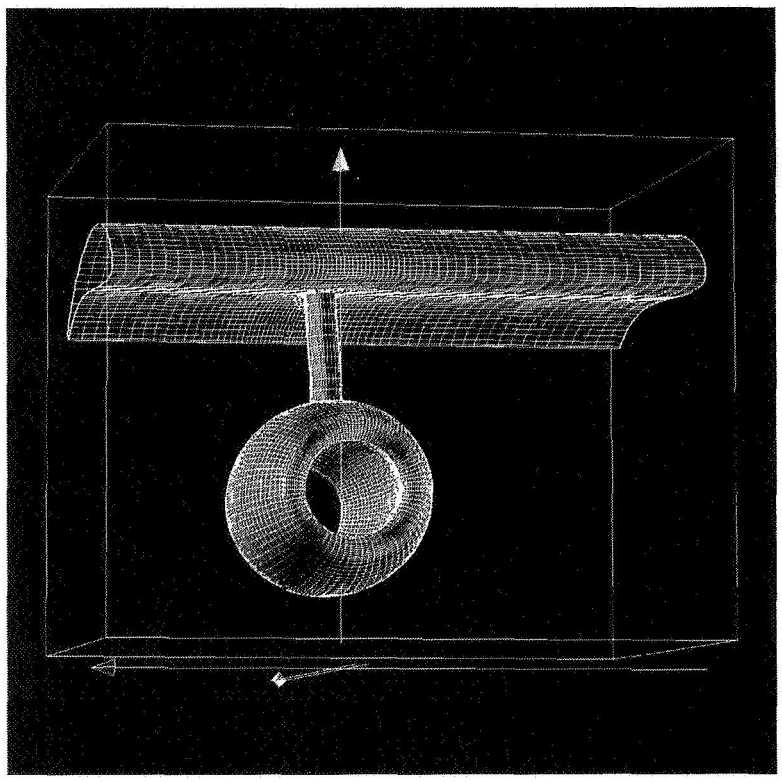


Figure 9: Overlapping grid for a wing, pylon and engine nacelle

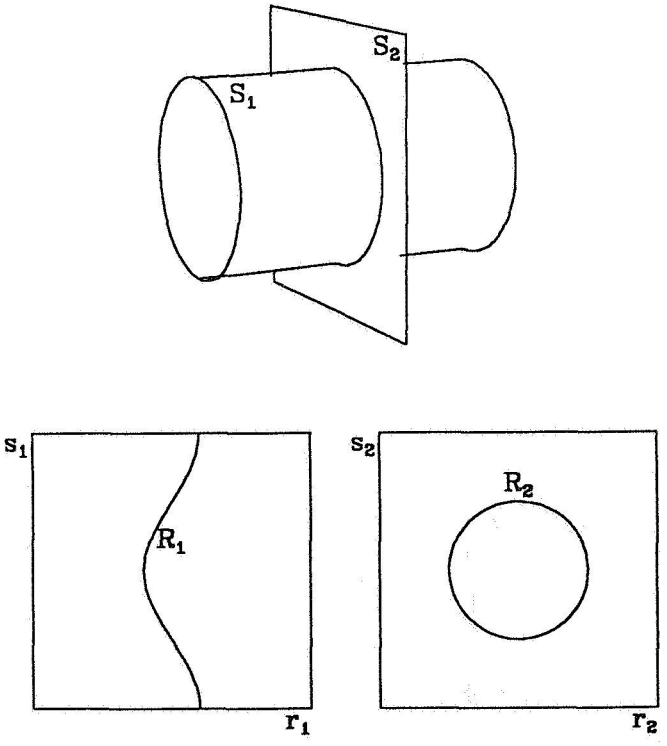


Figure 10: Curve of intersection for a cylinder and a plane

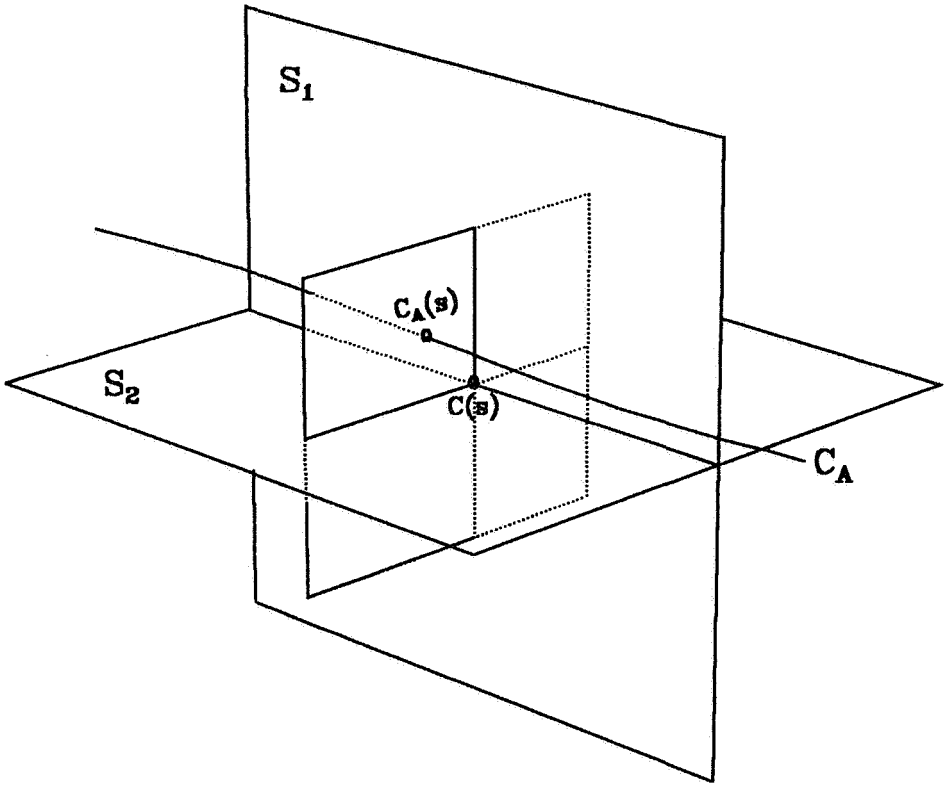


Figure 11: Parameterizing the curve of intersection with a nearby curve

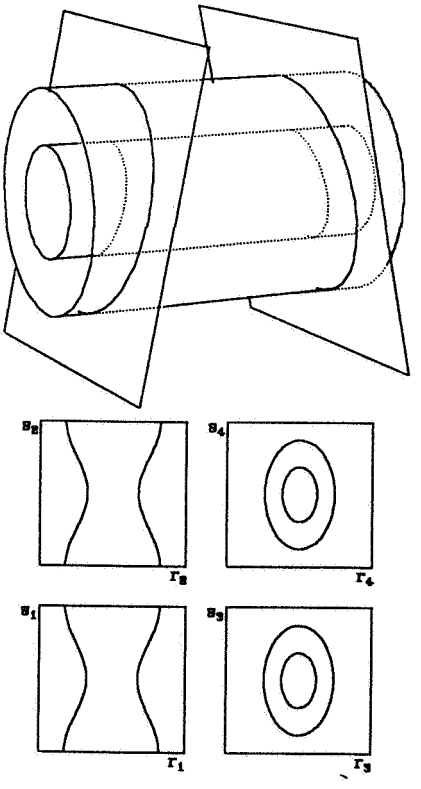


Figure 12: Curves of intersection for G_1

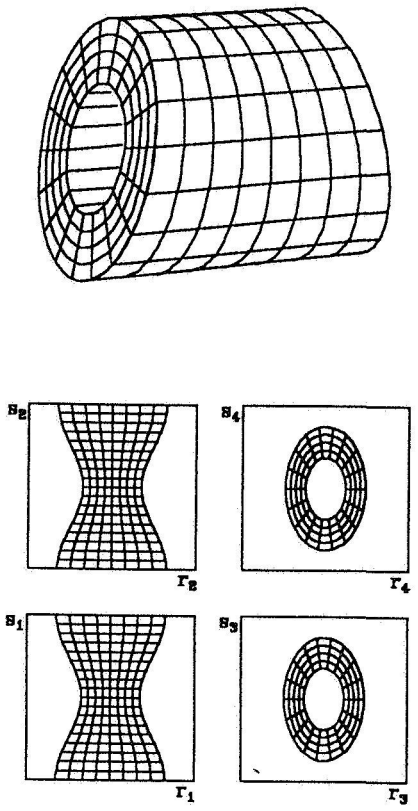


Figure 13: Grid G_1 and the images of the faces in the unit squares

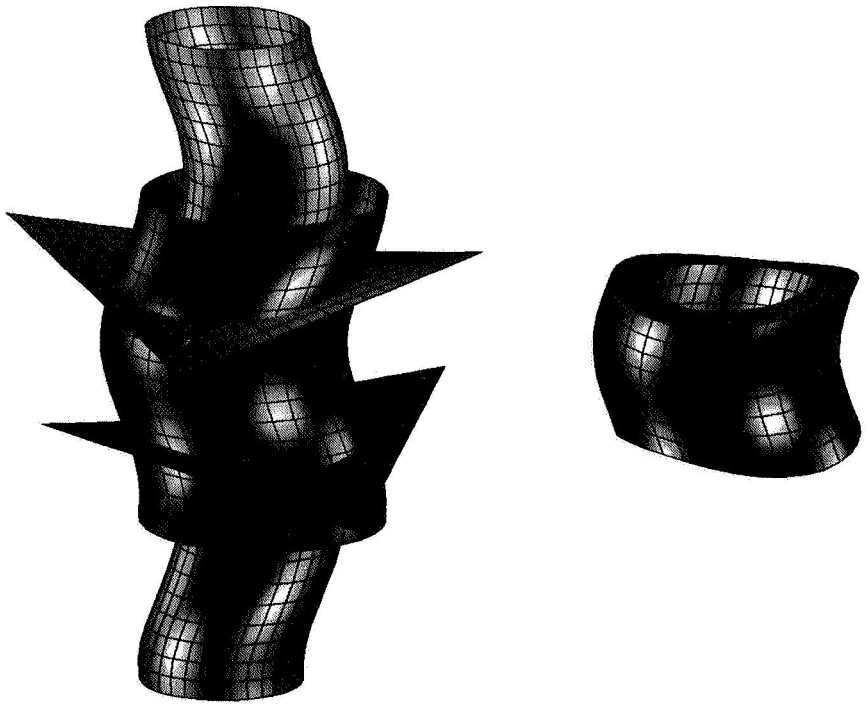


Figure 14: Creating a component grid by intersecting surfaces

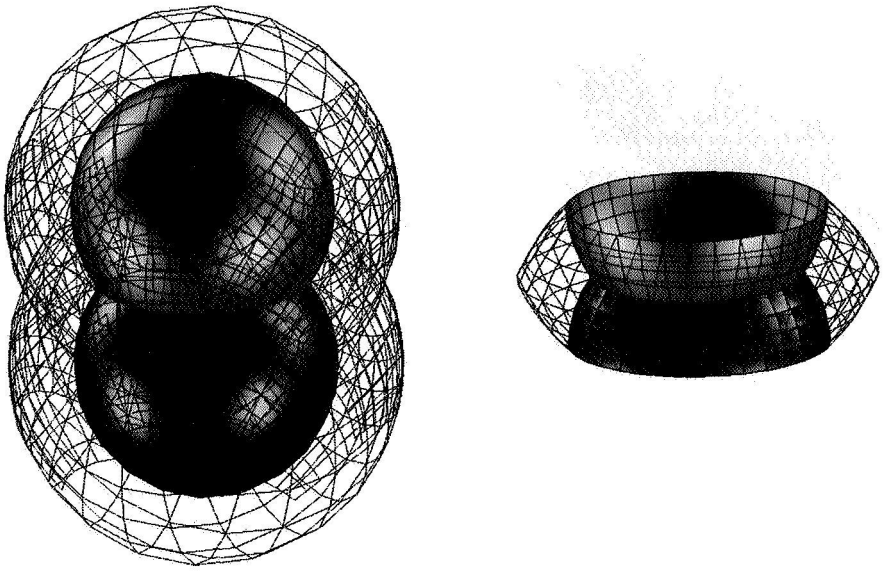


Figure 15: Collar grid between two intersecting spherical shells

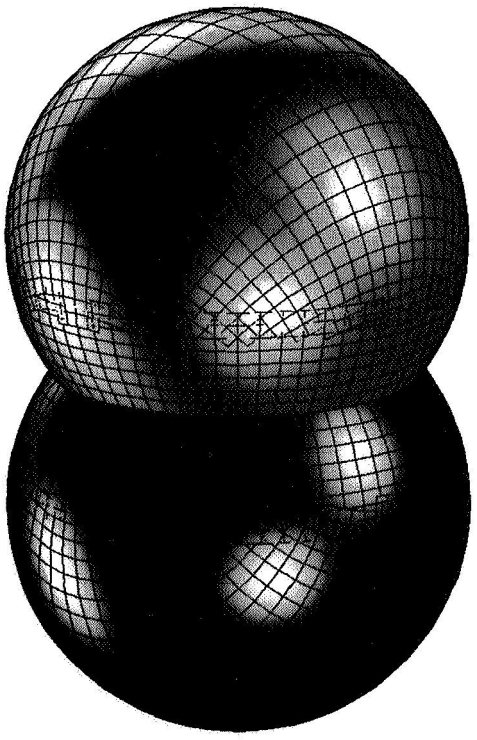


Figure 16: Overlapping grid for two intersecting spheres

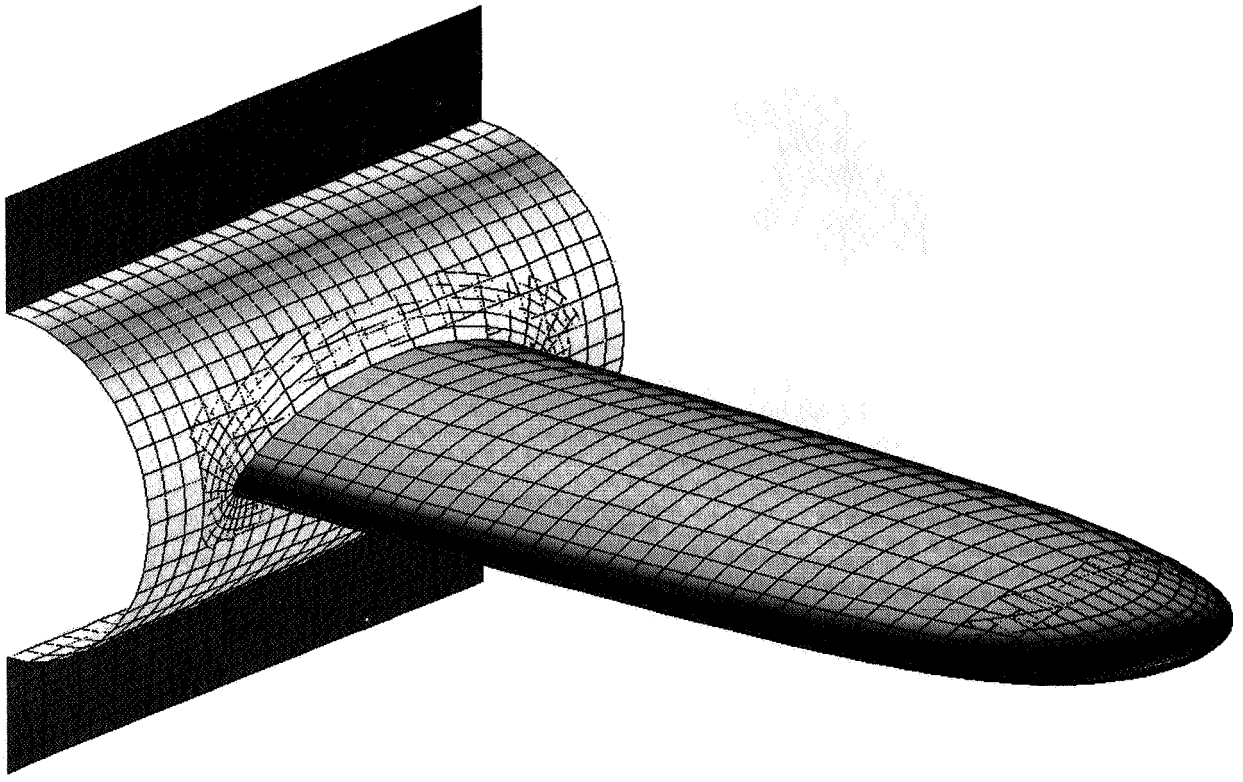


Figure 17: Overlapping grid created using intersecting surfaces

RECENT ADVANCES IN UNSTRUCTURED GRID GENERATION PROGRAM VGRID3D

Paresh Parikh and Shahyar Pirzadeh
ViGYAN, Inc., Hampton, Virginia

INTRODUCTION

VGRID3D is a program for generation of unstructured grids over complex configurations. The grid elements (triangles on the surfaces and tetrahedra in the field) are generated starting from the surface boundaries towards the interior of the computational domain using the Advancing Front Method.

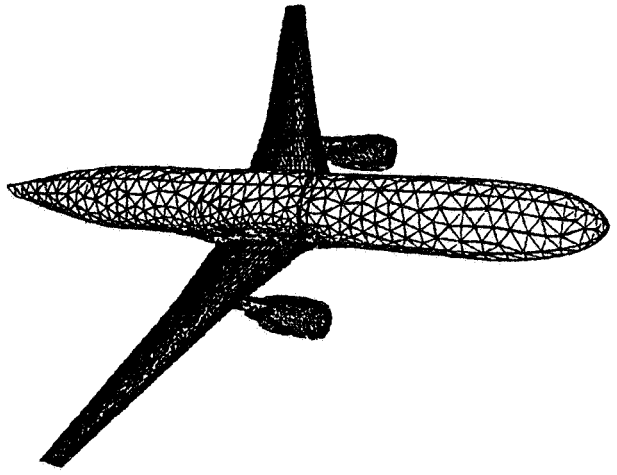
Over the past two years, many changes have been made and several new options are added to **VGRID3D**. Chief among these are:

- a restart capability that enables generation of a large, complex grid on a relatively small memory computer (like a workstation) in several restart runs,
- a menu-driven graphics version that helps a user visualize grid generation at every step during the process,
- a robust and user-friendly grid generator made possible by incorporating a wide experience base generated by the user community.

Several promising new developments are under way. These include

- structured background grid to provide a better control of the grid spacing, and
- development of an adaptive remeshing scheme that couples **VGRID3D** to an efficient Euler equation solver **USM3D**.

In this presentation, the grid generation process is briefly outlined first. This is followed by a description of the new developments augmented with examples.

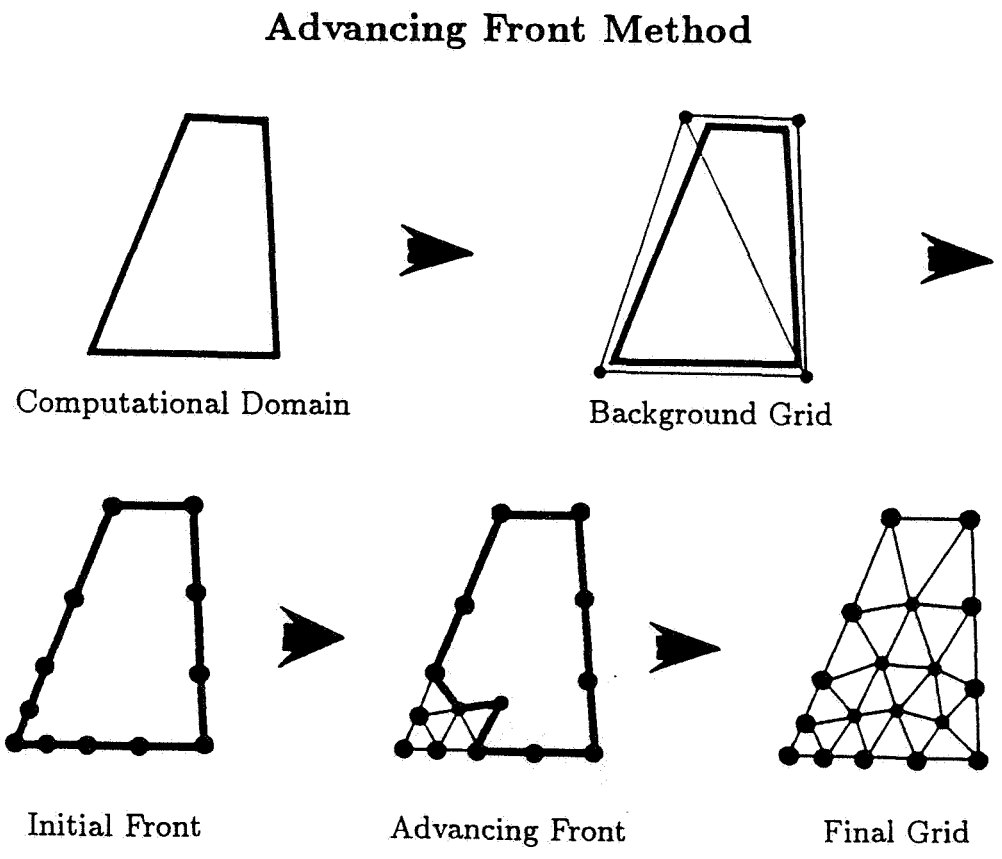


ADVANCING FRONT METHOD

The entire grid generation process using **VGRID3D** is summarized in the following main steps (Ref. 1,2):

1) The boundaries of the domain to be gridded are divided into a number of surface patches. These surfaces define the configuration of interest as well as the far-field boundaries of the computational domain.
2) A background grid is set up to define the local grid characteristics such as grid point spacing.
3) Each surface patch is, in turn, subdivided into a number of triangles to form the 'first' or 'initial' front.
4) The front is advanced in the field by introducing new points and forming tetrahedra until the entire computational domain if filled.

In the following figure the grid generation process is depicted in two dimensions.



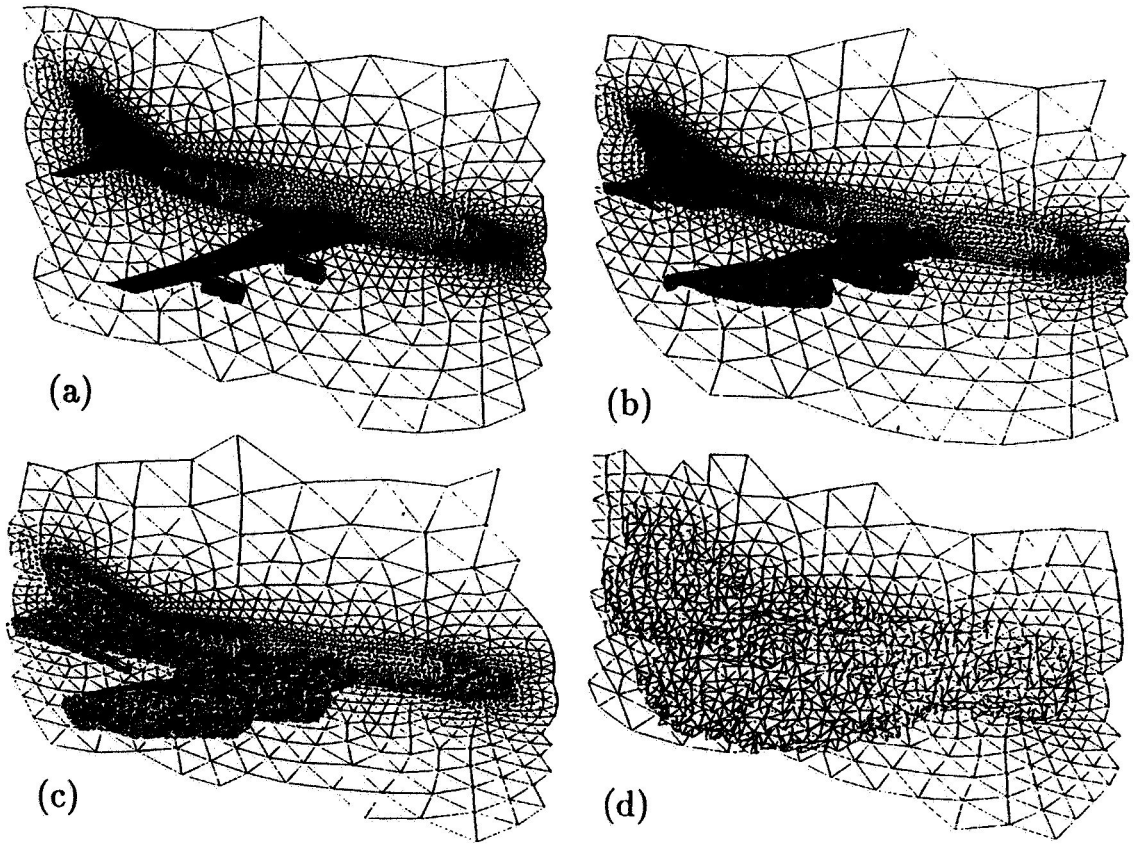
GRID RESTARTING

In the Advancing Front Method, the grid is generated in a marching fashion. Cells are formed on the current front by introducing new points in the field, which, in turn, reshape the front. Thus at any given time during the grid generation process, the computational region is divided into gridded and ungridded regions which are separated by the current front. Thus the grid generation process can be stopped and restarted without regard to the previously generated grid.

A grid restart capability has been incorporated in **VGRID3D**. The method is based on a local/global renumbering system. In each restart run, a user specified number of new grid points are introduced, the newly generated points and element connectivity are globally renumbered, the recent data are appended to the previously generated grid and the current front is locally renumbered for the next restart run.

The restart capability has resulted in a drastic reduction in the memory requirement for the grid generation, and now makes it possible to generate a large grid interactively on a large computer or in several restart runs on a smaller workstation. While conceptually simple, the restart capability has proven to be a powerful tool and has significantly enhanced the usability and productivity of **VGRID3D.**

The following picture shows several stages of an advancing front during the generation of grid around a Boeing 747 configuration using the restart capability.



INTERACTIVE GRAPHICS FOR VGRID3D

Interactive graphics has become an integral part of any three-dimensional grid generation process, whether for structured or unstructured grids. An interactive graphics version of **VGRID3D** has been prepared to help a user visually monitor each step of the grid generation process and intervene as necessary. Using this version, a user can examine the input to the grid generator, the background grid, the initial point distribution the surface triangulation for each patch comprising the configuration, and the selection and introduction of tetrahedra in the field. Although the interactive program is capable of generating a complete 3-D grid using a workstation, this task has been customarily left to a larger computer for speed and accuracy.

The interactive version currently runs on any Silicon Graphics IRIS 4D series of workstations, and is user-friendly and menu-driven. A version to run on SUN workstations is currently under development. In addition to the interactive version of **VGRID3D**, two other graphics pre-processor programs have been developed that help a user prepare the input to **VGRID3D**. The first of these is **PREGR1** which helps construct surface patch definition from geometry data given in either 'network' or 'cross-section' format. The second, **PREGR2**, helps construct and manipulate background grid which controls grid characteristics. Both **PREGR1** and **PREGR2** were originally written by Mr. Clyde Gumbert of NASA Langley Research Center. Current versions incorporate minor modifications that make them compatible with other programs in the unstructured grid software package. Availability of these graphics programs has greatly expedited the grid generation process.

GRID POST-PROCESSING

A grid with 'good' quality cells is essential for an accurate flow solution and for a stable convergence for steady-state problems. In the context of unstructured grids, a 'good' grid is one where the variation in size among the neighboring cells is smooth with a minimum number of distorted (high skewness) cells. Despite many precautions exercised in the grid generation process, the final grid almost inevitably contains some cells of undesirable geometric quality. For the advancing front grid generator, distorted cells can form either when the variation of grid spacing on the background grid is not sufficiently smooth, or when one or more fronts collide.

In order to alleviate this problem, a post-processing program called **POSTGR** has been developed. This program first provides an assessment of the grid quality based on the cell 'volume ratio' and 'included angles'. The volume ratio criterion is defined as the ratio of the actual cell volume to a corresponding 'ideal volume' of an equilateral cell which is based on an average of the actual cell edges. A volume ratio of less than 5 percent is considered distorted. The included angle is the angle between the adjacent faces of a tetrahedral cell. There are 6 such angles for any tetrahedra. Any cell with an included angle of less than 5° or greater than 175° is tagged as a distorted cell. The distorted cells are then removed along with an additional layer of surrounding cells, creating cavities in the volume grid. The cavities are next filled with new tetrahedra using the advancing front method. In practice several such cycles of alternately using the **POSTGR** and **VGRID3D** are used to remove most of the distorted cells.

This post-processing technique has proved to be very effective in improving grid quality and is being routinely used in the grid generation process.

NEW DEVELOPMENTS

Over the past two years, **VGRID3D** has been made robust and several options added to make it more user-friendly as described before. Further research is under way on several additional developments which should result in an overall improved grid quality, significantly reduced grid generation time, and expanded capabilities of **VGRID3D**. The developments include:

1) structured background grids which provide a better control of grid spacing with significantly less user interaction, and

2) an adaptive remeshing scheme to obtain the final solution more efficiently.

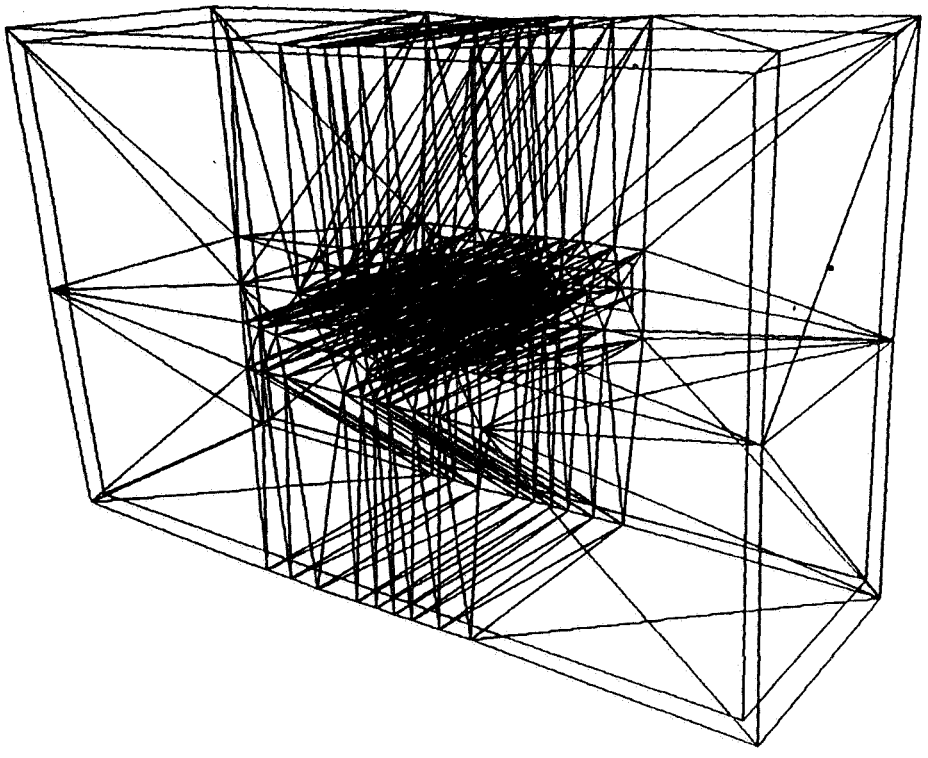
The above developments are at various stages of completion and have been successfully applied to focused applications which are presented here. The core technologies exist for both, but are not sufficiently mature to be installed in the production version of **VGRID3D**.

CONVENTIONAL BACKGROUND GRID IN VGRID3D

An important feature of a grid generation technique is its ability to distribute points smoothly throughout the computational domain with convenient user control. In the Advancing Front Method, the grid element size is controlled by parameters specified at the nodes of a secondary coarse grid called 'Background Grid'. The conventional background grid consists of a number of tetrahedral cells (an unstructured mesh in itself) which completely encloses the computational domain to be gridded. As the mesh front is advanced in the field, the information for the location of a new point is interpolated from spacing parameters prescribed at the nodes of a background cell which encloses the point.

This method of grid spacing control has several disadvantages. The quality of the final grid is dependent upon the smoothness in variation of the grid spacing from point to point in the background grid. An uneven variation in background grid spacing may result in many tangled fronts to the detriment of the grid generation process. Hence, a user needs to exercise caution in the construction of the background grid. Construction of the tetrahedral background grid is presently a manual process which is aided by a graphics pre-processor program **PREGR2**. Although **PREGR2** has simplified this procedure a great deal, there is currently no automatic way of modifying a background grid should the necessity arise.

Although the conventional method of grid spacing control using tetrahedral background grids has been successfully used to generate grids over several complex configurations, it is currently the most time consuming task in the entire grid generation process. A sample background grid is shown in the following figure for an ONERA M6 wing configuration and shows the complexity of the method of grid spacing control. For this reason research has been focused on alternatives. One such alternative is the so called 'structured' background grid which is described next.

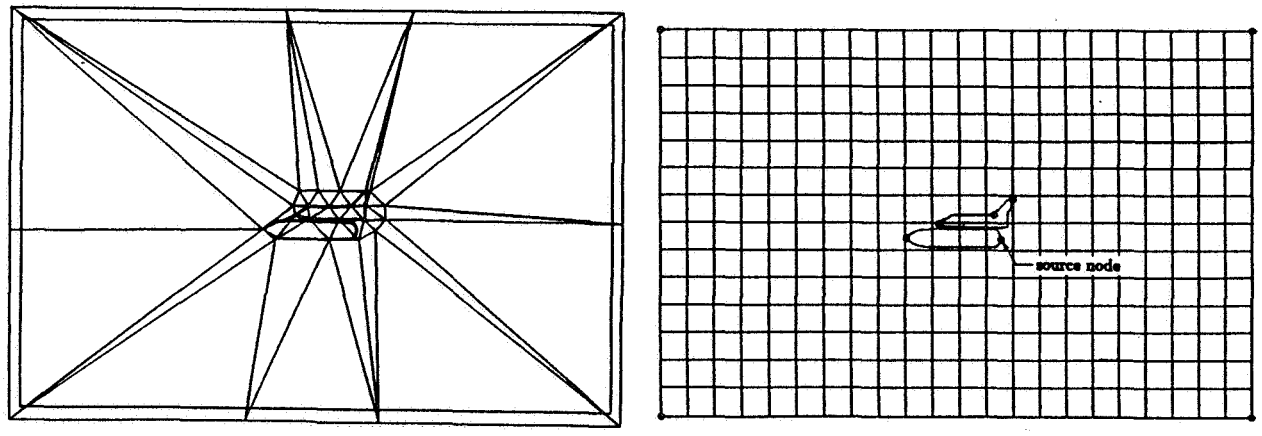


STRUCTURED BACKGROUND GRID

A new type of background grid has been introduced which resolves many of the problems and inconveniences associated with the conventional tetrahedral background grids (Ref.3). Some of the salient features of this new background grid are:

- The grid consists of uniformly spaced Cartesian cells, hence the name **structured**. Since a Cartesian grid can be generated automatically, the new method eliminates need for an involved graphics program.
- The desired grid spacing on the configuration is controlled by an arbitrary number of user specified 'source' elements. There is no restriction to the number and location of these source elements. Currently two types of elements are used: nodal and linear.
- The spatial variation of spacings at the nodes of the structured Cartesian grid is modeled as diffusion of 'heat' in a medium with 'source' elements acting as 'heat' sources, and solving a Poisson equation.
- During the grid generation process, the interpolation of spacing for any arbitrary location is now a simple matter of Cartesian interpolation, thus eliminating need for specialized data structure used in the conventional unstructured grid method.

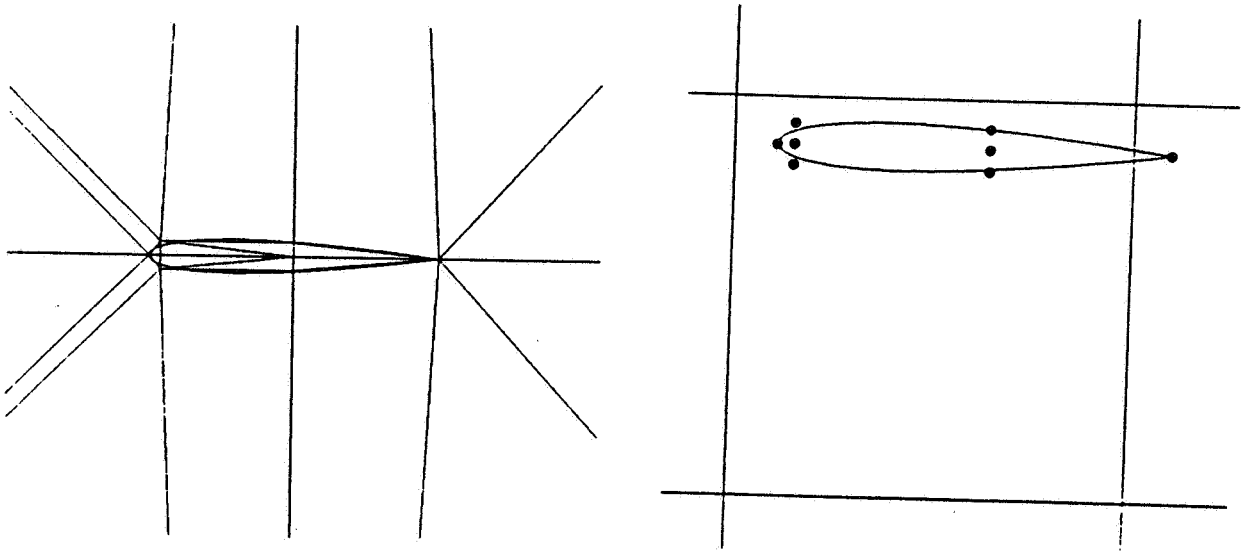
The figure below compares background grids obtained using the conventional and the new methods, in the symmetry plane of a Space Transportation System configuration. In the new method the grid spacing is controlled by specification of five source nodes on the body and four at the corners of the outer computational boundaries, while the conventional method required constructing 48 triangles and specifying grid control parameters at each of the nodes.

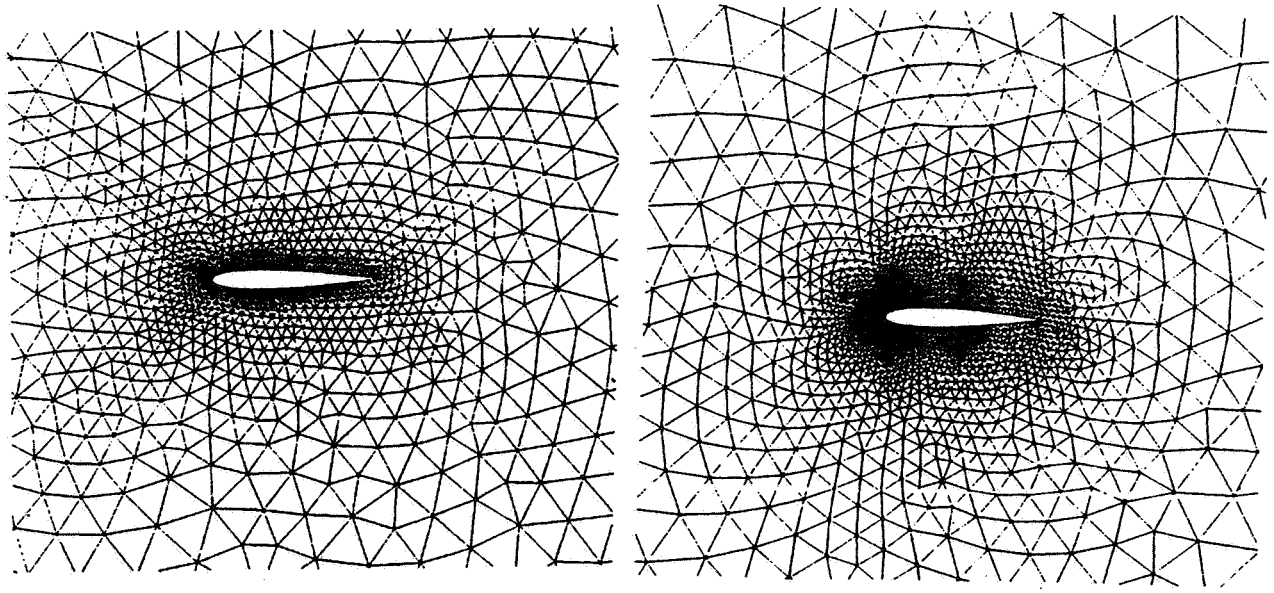


STRUCTURED BACKGROUND GRID - AN EXAMPLE

The use of structured background grid not only simplifies the preparation of the background grid information, but the quality of the final grid is significantly improved. The improved grid quality is illustrated in the following figure on a NACA 0012 airfoil. The top two frames depict partial views of the conventional triangular (left) and the new structured Cartesian (right) background grids are shown. The conventional background grid has 22 cells and 16 nodes. The structured grid is a 21 X 21 grid. The structured grid has 16 source elements for grid spacing control, 8 of which located near the airfoil are shown in the picture.

The corresponding field grids are shown in the bottom two pictures. These near field views clearly reveal the differences in grid quality near the airfoil. The conventionally generated grid lacks the desired smoothness in distribution, whereas the one generated with the new method exhibits an orderly progression of contours resembling concentric 'isotherms' around the airfoil.

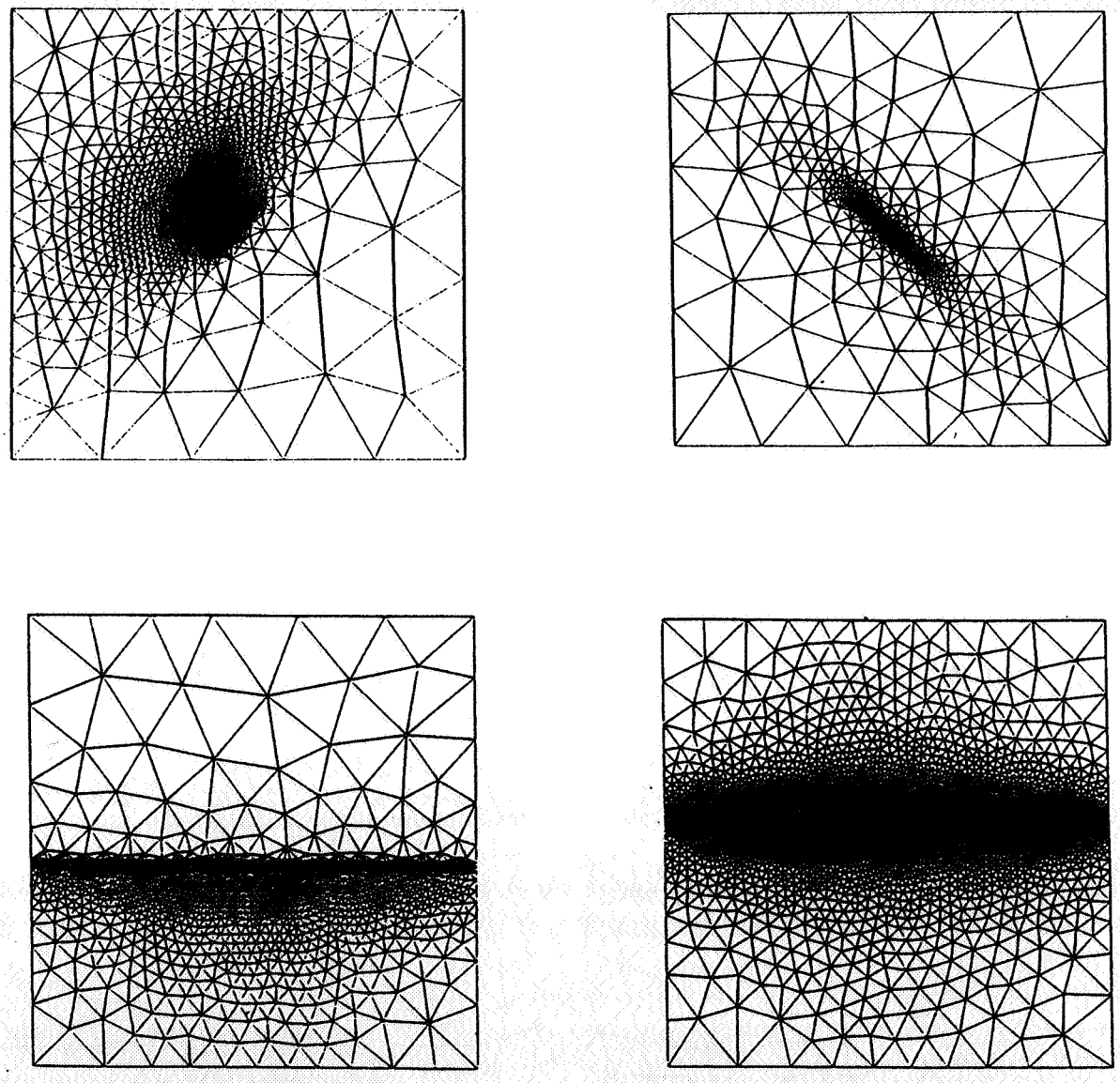




DIRECTIONAL CONTROL USING STRUCTURED BACKGROUND GRID

In addition to symmetrical propagation of spacing parameter, the new method is capable of controlling the directional intensity of the sources. This is done by limiting the source intensities to certain zones and directions.

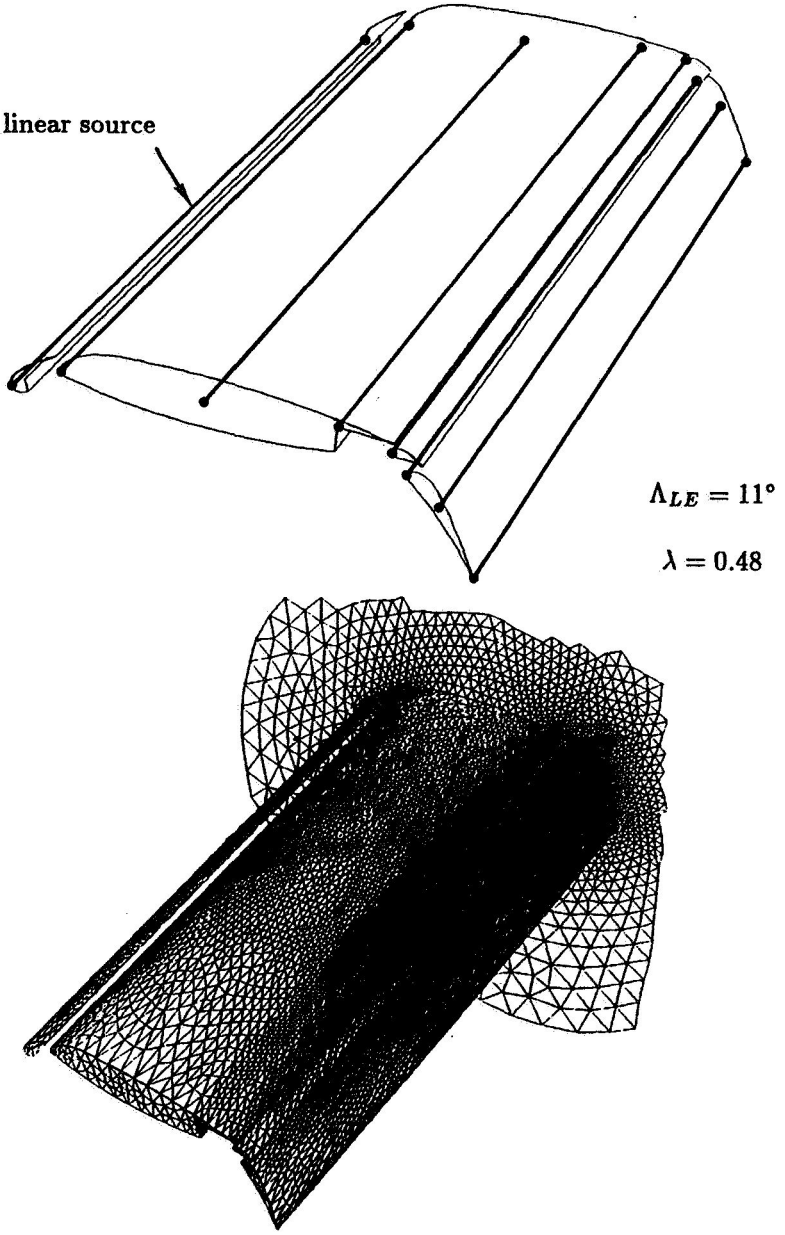
The figure below shows sample grids generated with various directional controls applied. The top two pictures show control exercised by controlling nodal sources, while the bottom two pictures show grids obtained by controlling propagation of linear sources.



STRUCTURED BACKGROUND GRID IN 3-D

The technique of structured background grid has been recently extended to three dimensions (Ref.4). The figure below shows a multi-element wing with a leading edge sweep of 11 degrees and a taper ratio of 0.48. A 21 X 16 X 10 Cartesian background grid for this configuration includes 8 linear source elements on the wing section (top picture) and 8 nodal elements on the corners of the outer computational boundaries. All source elements have directional intensities for better control of grid point clustering. The resulting surface triangulation on the wing is shown in the bottom picture.

The smooth distribution of grid element size variation and the intended clustering of grid points along the leading and trailing edges is evident. Experience has shown that a reproduction of a grid with this quality using the conventional background grid would require many tetrahedral cells and require a substantial amount of users' time.



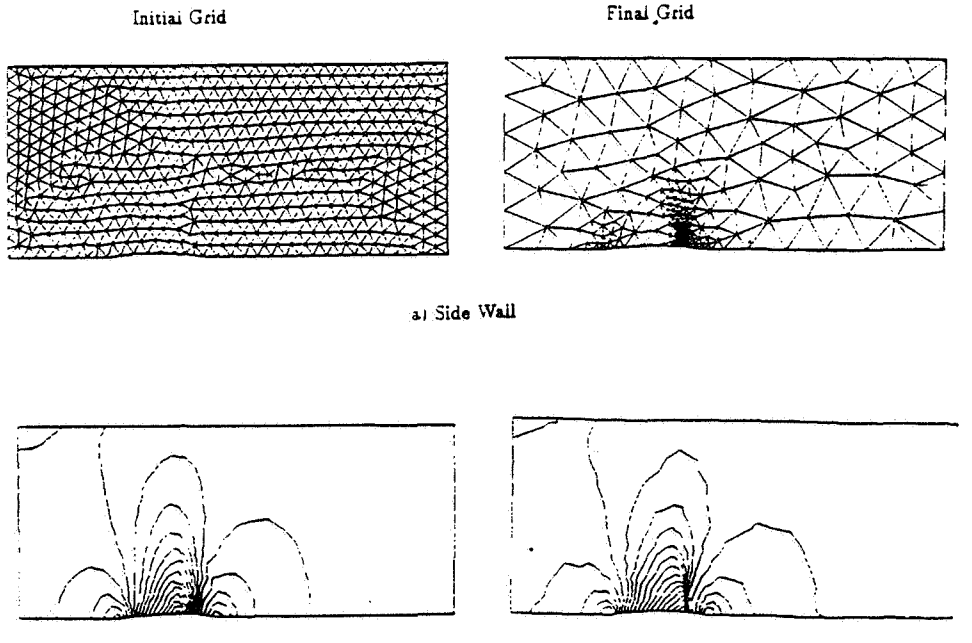
ADAPTIVE REMESHING PROCEDURE

Adaptation of the mesh to the dominant features of the flow offers the possibility of enhancing the solution quality while retaining control over the total number of grid points. Work is in progress in coupling **VGRID3D** to an efficient inviscid flow solver (Ref. 5), **USM3D**, in such a manner that the information produced by one is successively used by the other to allow the grid to adapt.

As mentioned earlier, in the Advancing Front Method the grid element size is determined by a user specified 'spacing parameter' on each node of a conventional secondary grid termed 'background' grid. In the adaptive procedure described here, the spacing parameter is calculated, instead, based on the local flow gradients. The steps involved for the adaptive remeshing procedure are (Ref. 6):

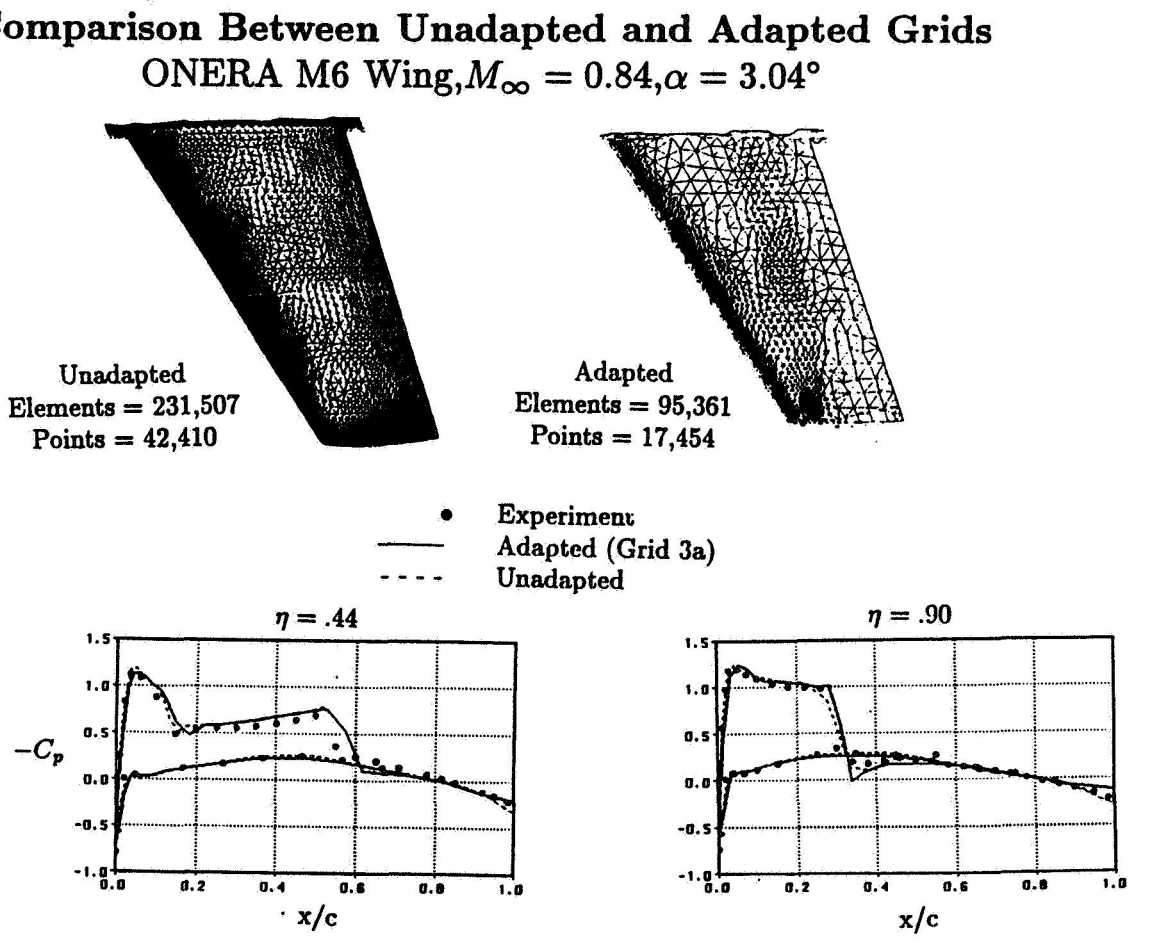
1) Obtain initial solution on a coarse grid.
2) Calculate the 'spacing parameter' at the nodes of the current grid, using the current solution as an error indicator.
3) Generate a new grid using current field grid as the background grid.
4) Obtain a new solution.
5) Repeat steps 1-4 until desired accuracy is obtained.

The adaptation procedure is demonstrated in the following figure on a circular arc bump in a channel at a $M_\infty = 0.85$. The top portion of the figure compares the initial and the final grids, while the bottom portion compares the C_p contours. A total of four remeshing cycles were applied. Clustering of grid points in the region of large flow gradients and coarsening of the grid in the smoother regions of the flowfield is evident in the adapted grid.



ADAPTIVE REMESHING ON ONERA M6 WING

To demonstrate a more complex example, flow over an ONERA M6 wing at a $M_\infty = 0.84$ and $\alpha = 3.06°$ is considered. The procedure was initiated with a coarse, almost uniform grid made up of 52,567 tetrahedral elements and 9,646 points. Three remeshing cycles were performed resulting in a grid with 95,361 elements and 17,454 points. The efficiency of the adaptive procedure is shown in the figure by comparing the results after the third remeshing cycle with an unadapted flow solution. The unadapted grid has 231,507 tetrahedral elements and 42,410 points. The top portion of the figure shows the upper surface triangulation for both the grids while in the bottom portion, the computed C_p distributions at two spanwise locations are compared with the experimental data. The adapted solution is in very good agreement with the unadapted one which has about 2.5 times the number of elements. Additional work is underway to quantify the efficiency of the procedure in terms of run times and solution quality.



REFERENCES

1. Parikh P., Pirzadeh, S. and Löhner, R.: A Package for 3-D Unstructured Grid Generation, Finite-Element Flow Solutions, and Flow-Field Visualization. NASA CR-182090, September 1990.

2. Löhner, R. and Parikh, P.: Three-Dimensional Grid Generation by the Advancing Front Method. *Int. J. Num. Meth. Fluids*, Vol. 8, 1988, pp. 1135-1149.

3. Pirzadeh, S.: Structured Background Grids for Generation of Unstructured Grids by Advancing Front Method. AIAA Paper 91-3233, 1991.

4. Pirzadeh, S.: Recent Progress in Unstructured Grid Generation. AIAA Paper 92-0445, 1992.

5. Frink, N. T., Parikh, P. and Pirzadeh, S.: A Fast Upwind Solver for the Euler Equations on Three-Dimensional Unstructured Grids. AIAA Paper 91-0102, 1991.

6. Parikh, P. and Frink, N. T.: An Adaptive Remeshing Procedure For Three-Dimensional Unstructured Grids. Paper presented at 4th International Symposium on Computational Fluid Dynamics, Davis, California, September 9-12, 1991.

THREE-DIMENSIONAL UNSTRUCTURED GRID GENERATION VIA INCREMENTAL INSERTION AND LOCAL OPTIMIZATION

Timothy J. Barth
and
N. Lyn Wiltberger
NASA Ames Research Center, Moffett Field, CA

Amar S. Gandhi
MCAT Institute, Moffett Field, CA

ABSTRACT

Algorithms for the generation of three-dimensional unstructured surface and volume grids are discussed. These algorithms are based on incremental insertion and local optimization. The present algorithms are very general and permit local grid optimization based on various measures of grid quality. This is very important; unlike the two-dimensional Delaunay triangulation, the 3-D Delaunay triangulation appears not to have a lexicographic characterization of angularity. (The Delaunay triangulation is known to minimize that maximum containment sphere, but unfortunately this is not true lexicographically). Consequently, Delaunay triangulations in three-space can result in poorly shaped tetrahedral elements. Using the present algorithms, three-dimensional meshes can be constructed which optimize a certain angle measure, albeit locally. We also discuss the combinatorial aspects of the algorithm as well as implementational details.

INTRODUCTION

Unstructured grids offer great advantages in the automatic generation of computational grids about complex geometries. In the present work we consider triangulation methods in two and three space dimensions. In order to minimize the amount of required user interaction, incremental triangulation algorithms are employed which permit automatic (adaptive) point placement. Several incremental algorithms exist for the triangulation of two and three dimensional domains. Most of these algorithms are designed to produce the Delaunay triangulation of specified sites which form the vertices of the mesh. In the present work, we have chosen an incremental triangulation algorithm based on point insertion and local edge swapping. The advantage of this algorithm is that the edge swapping can be used to optimize user specified grid quality measures. (This algorithm can be used to generate a Delaunay triangulation by proper choice of measure.) In addition, we produce Steiner triangulations by inserting new sites at triangle circumcenters of the existing triangulation to minimize the maximum triangle aspect ratio (or any other user specified criteria). An objective of the current research is to extend this two-dimensional algorithm to include the generation of surface grids or volume grids in three-dimensions. The algorithm based on edge swapping is well known in two dimensions [1] but the extension to surface grids and volume grids in three dimensions in not straightforward. In the present work, we show the extension of edge swapping to include the generation of surface grids and three dimensional volume grids. Differing edge swapping optimization criterion are considered. Steiner triangulations in three dimensions are produced by placing new sites at circumsphere centers of the existing triangulation.

Motivation:

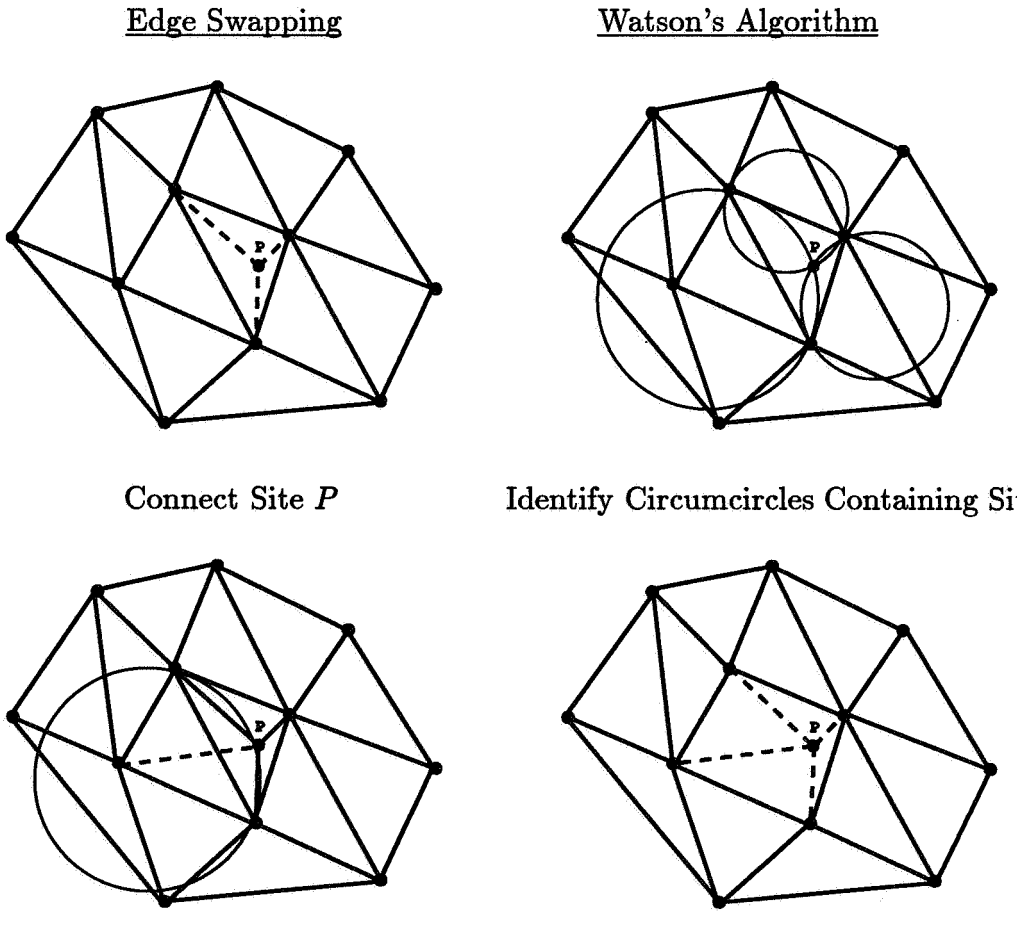
- Develop fast automatic three-dimensional unstructured grid generation capability for complex geometries.

Objectives:

- Generalize a 2-D unstructured grid generation algorithm based on incremental insertion and edge swapping.
  - Generation of surface grids on 2-manifolds in 3-D.
  - Automatic adaptation to maintain surface fidelity in 3-D.
- Extend incremental insertion and edge swapping to 3-D volume mesh generation.
  - Investigation of differing edge swapping optimization criterion.
  - Automatic point insertion (Steiner triangulations).

Edge Swapping or Watson's Algorithm?

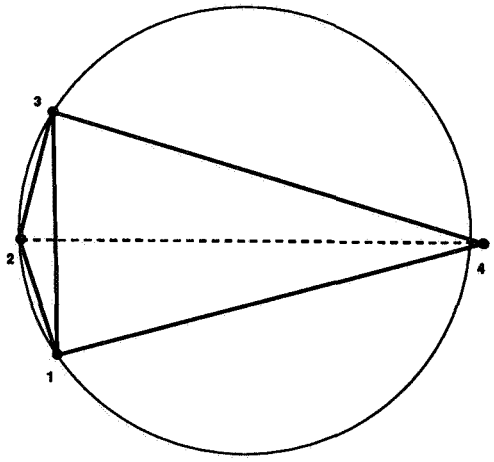
Conceptually speaking, the incremental algorithm based on edge swapping differs significantly from the incremental Watson algorithm [4] for the Delaunay triangulation. However, when edge swapping is used to generate a Delaunay triangulation, both methods have similar implementations. Both methods begin by inserting a new site P in the interior of an existing triangulation. The edge swapping algorithm begins by connecting site P to the existing triangulation. The union of all triangles incident to P forms a polygon surrounding this new site. The edges of this polygon must each pass the circumsphere test. Edges failing this test must be swapped. The process continues until all edges of the polygon pass. The algorithm is trivial to implement using recursive programming. Conceptually, Watson's algorithm finds all triangles with circumcircles containing site P. This identifies invalid edges that when removed reveal a polygon with vertices visible from P. The vertices of this polygon are then connected to P. The actual implementation of Watson's algorithm is similar to the edge swapping code. The first step is to find *any* triangle with its circumcircle containing site P. Once this triangle is found, edges are reconnected and neighboring triangles tested. This can also be easily programmed recursively. The resulting code is very similar to the edge swapping implementation.



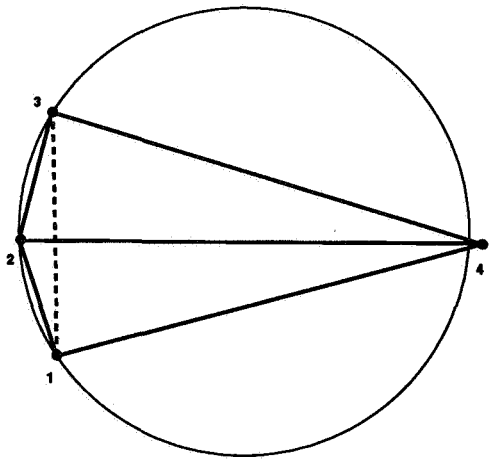
Min-Max and Max-Min Triangulations

The edge swapping algorithm easily extends to allow edge swapping based on criteria other than the circumsphere test. The circumsphere test is known to be equivalent to the angularity optimization that chooses the diagonal position that maximizes the minimum interior angle (Max-Min triangulation). Another popular method chooses a diagonal position for a given convex triangle pair that minimizes the maximum interior angle (the Min-Max triangulation). If the diagonal position is swapped using this measure then the four neighboring edges must be checked (the Max-Min version of this algorithm only requires that two edges be checked). It should also be noted that the Min-Max triangulation can produce triangulations that locally minimize the maximum angle but are not globally optimal with respect to the same measure.

- Advantage of incremental edge swapping is that it permits other types of local optimization.
  - Requires more sophisticated recursion implementation and usually only produces a locally optimum triangulation.
  - Typical local optimizations minimize the maximum interior angle (Min-Max) or maximize the minimum interior angle (Max-Min,Delaunay triangulation).



Delaunay (Max-Min) Configuration



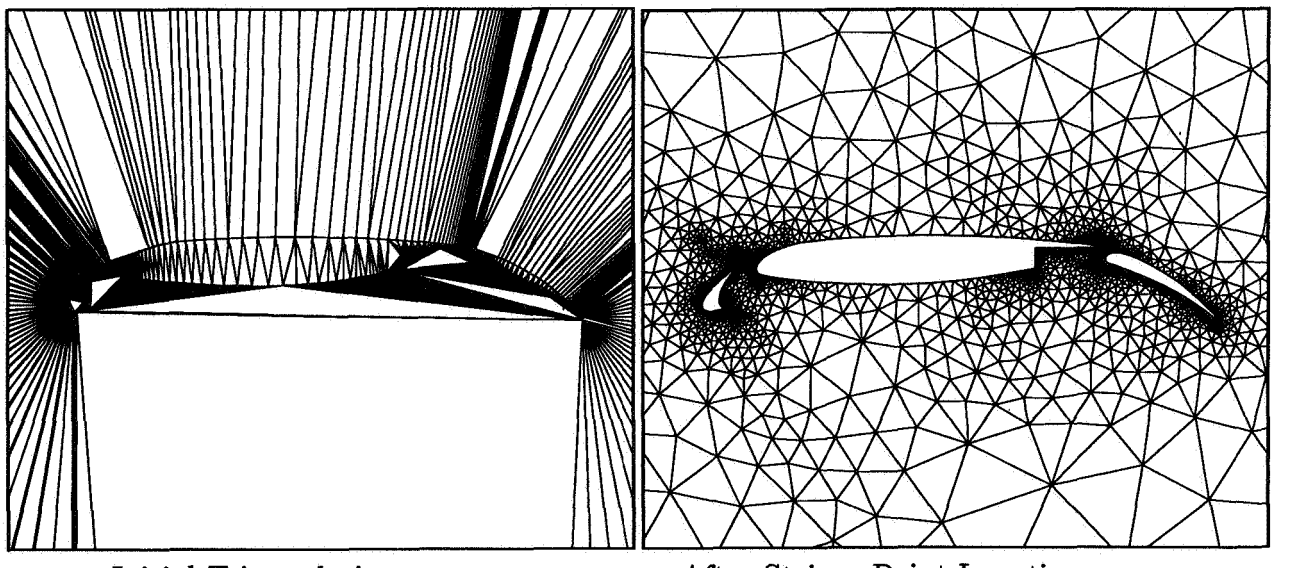
Min-Max Configuration

- Min-Max triangulation favors less obtuse triangles (important for highly stretched meshes).

Steiner Triangulations

A Steiner triangulation is any triangulation that adds additional sites into an existing triangulation to improve a specified measure of grid quality. For the present application, Delaunay triangulations are constructed which refine a given triangle by adding a new site at the circumcenter of the triangle. This technique is well known (Homes and Snyder [3], Warren et al [4]). The choice of circumcenter guarantees that no other point in the triangulation lies closer than the radius of this circle. The initial triangulation consists of a Delaunay triangulation of the boundary points (possibly edge swapped to maintain the boundary curve definitions). Points are inserted to minimize the maximum triangle aspect ratio using a dynamic heap data structure.

- Steiner Triangulation - insert additional sites into an existing triangulation to improve a specified measure of grid quality.



Initial Triangulation

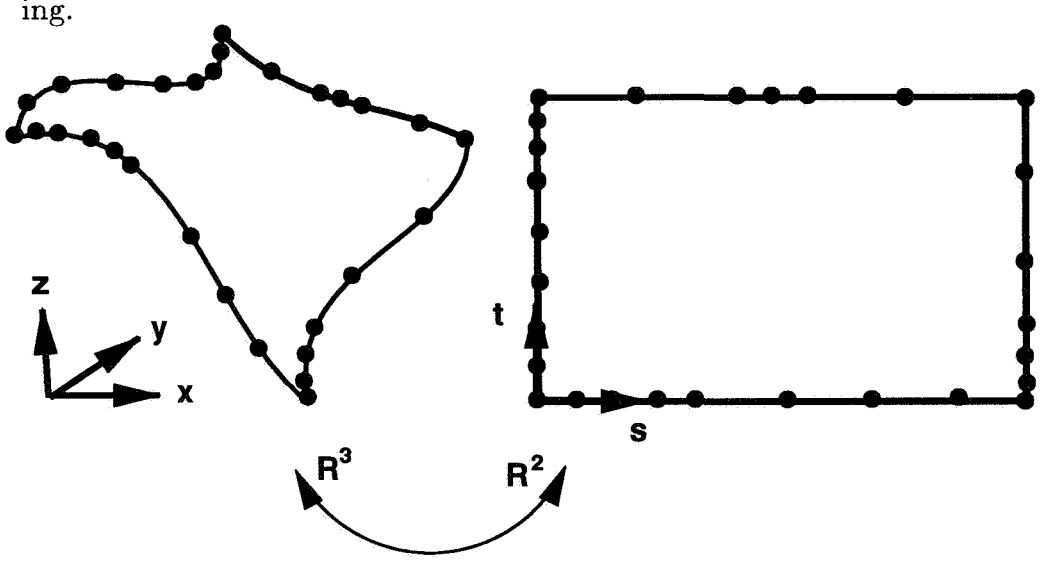
After Steiner Point Insertion and Interior Edge Removal

- Insert sites at circumcenters to minimize triangle aspect ratio.

3-D Surface Patch Projection

Three-dimensional surface grids are constructed from rectangular surface patches (assumed at least C^0 smooth) using a generalization of the 2-D Steiner triangulation scheme. Points are placed on the perimeter of each patch using an adaptive refinement strategy based on absolute error and curvature measures. The surface patches are projected onto the plane. Simple stretching of the rectangular patches permits the user to produce preferentially stretched meshes. (This is useful near the leading edge of a wing for example.)

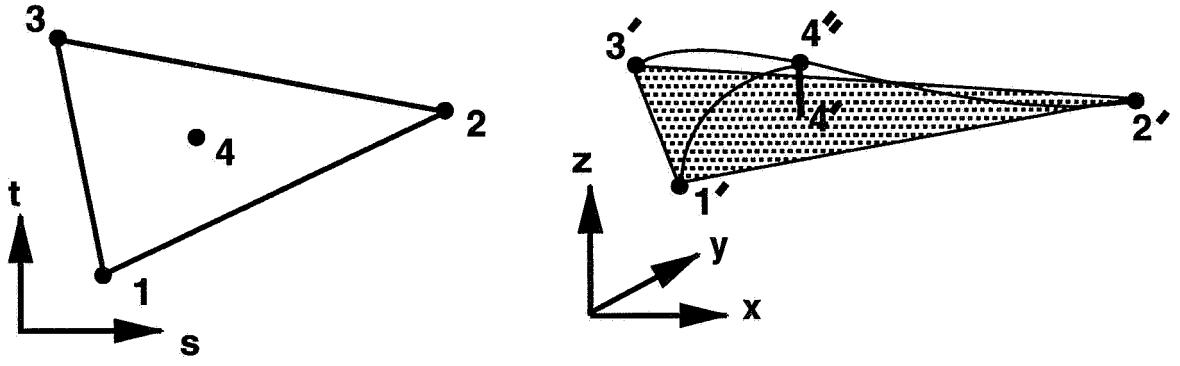
- Sites are distributed on the perimeter of each patch using 1-D adaptive subdivision.
- Curve adaptation criterion:
  - Absolute tolerance
  - Curvature tolerance
- Patches projected into 2-space for triangulation.
  - Mapping to rectangle permits simple stretching which results in directional biasing.



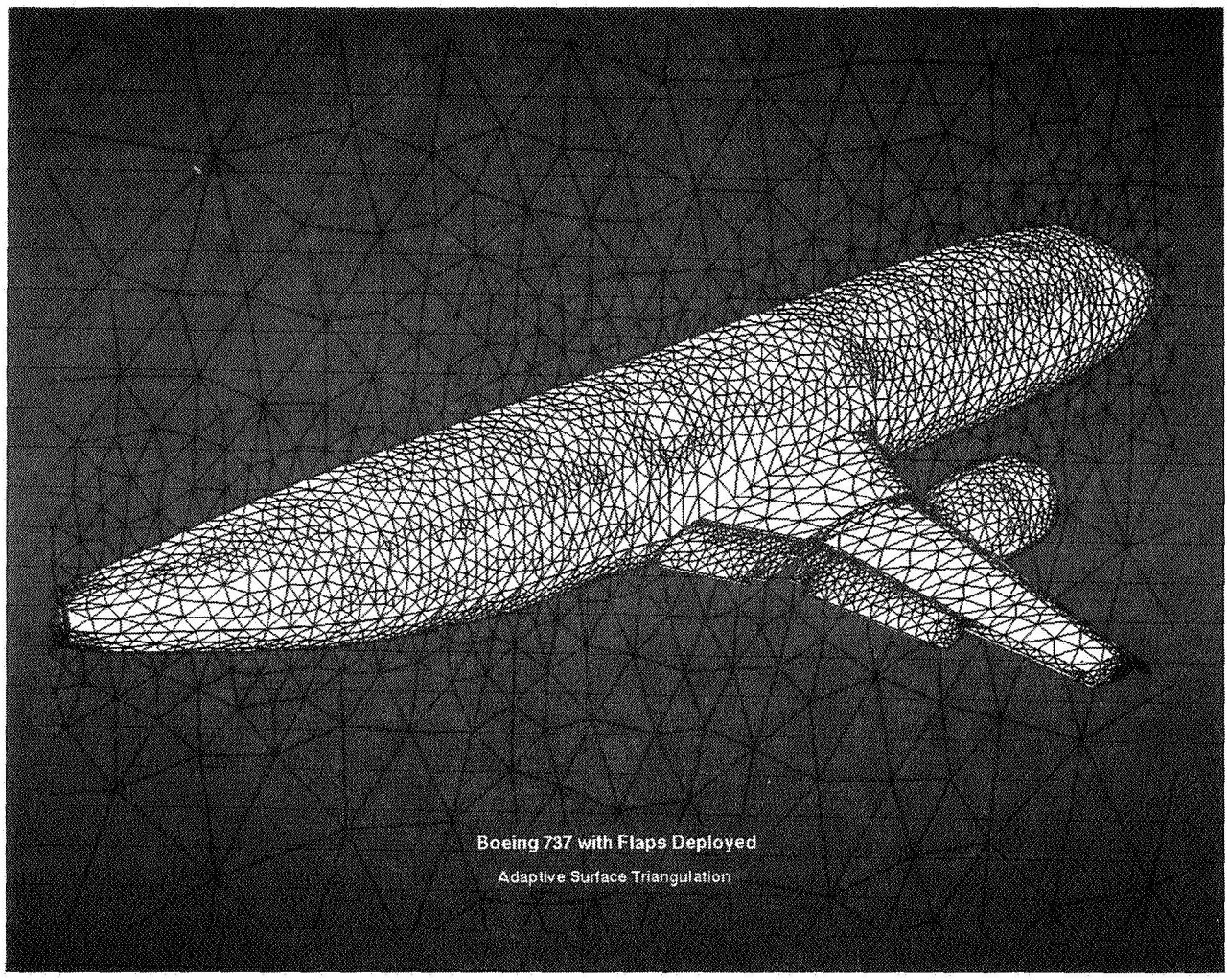
Adaptive Patch Triangulation

The triangulation takes place in the two dimensional (s,t) plane. The triangulation is adaptively refined using Steiner point insertion to minimize the maximum user specified absolute error and curvature tolerance on each patch. The absolute error is approximated by the perpendicular distance from the triangle centroid (projected back to 3-space) to the true surface. The user can further refine based on triangle aspect ratio in the (s,t) plane if desired.

- Adaptive Delaunay triangulation in (s,t) plane.
  - Adaptive insertion based on absolute error and maximum curvature tolerance.



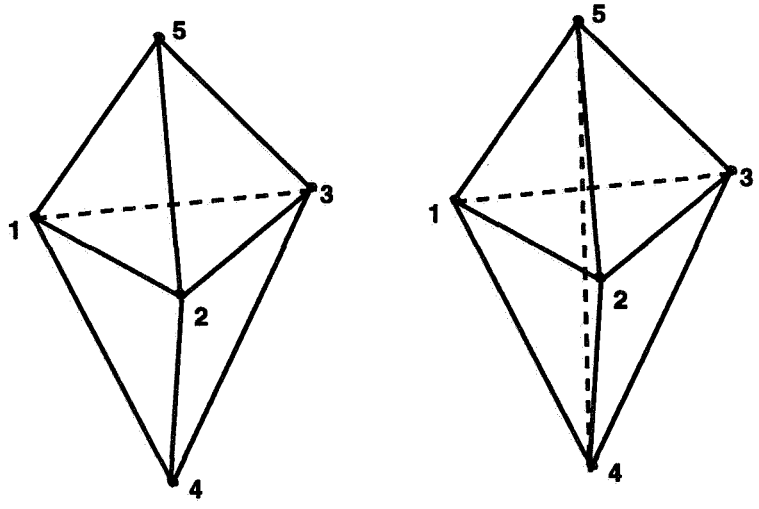
  - Calculation of triangulation absolute error by measurement of distance from face centroid to true surface.
  - Optional refinement based on triangle aspect ratio.


Boeing 737 with Flaps Deployed
Adaptive Surface Triangulation

3-D Incremental Insertion and Optimization

The 3-D edge swapping methodology is based on Lawson's result [5] that there exists at most two ways to triangulate $n+1$ points in R^n. It can be shown that if a configuration of 5 points in 3-D allows only one triangulation, it will satisfy the Delaunay circumsphere test. If a configuration of 5 points in 3-D allows two triangulations, then only one will satisfy the Delaunay circumsphere test. The edge swapping algorithm is based on locally swapping convex non-Delaunay triangulations of 5 points into its Delaunay counterpart. It has been shown that during the optimization process for a general mesh the procedure may get stuck in a local optimum. However, Rajan [1] proved that inserting a site into an existing Delaunay triangulation followed by edge swapping is guaranteed to recover the new Delaunay triangulation. This forms the basis of our 3-D Incremental Insertion and Optimization algorithm for 3-D Volume Mesh Generation.

- Combinatorial theorem (Lawson): There exist at most two ways to triangulate $n+1$ points in $\mathbf{R}^n$ (subject to convexity).



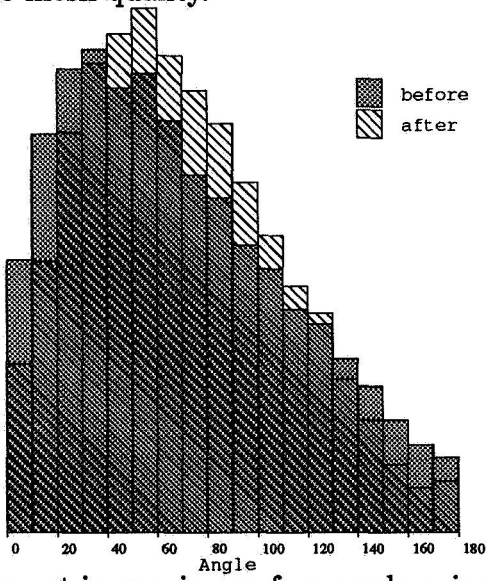
2 Tetrahedron Configuration 3 Tetrahedron Configuration

- Edge swapping an arbitrary 3-D triangulation (subject to circumsphere optimization) is not sufficient to guarantee that a Delaunay triangulation will be produced.
- Site insertion into an existing Delaunay triangulation followed by edge swapping will produce a new Delaunay triangulation (Rajan, 1991).

Local and Global Optimization

As long as a metric can be defined for the two triangulations of a configuration of 5 points in three dimensions, edge swapping locally to optimize that metric can be performed. In this way, edge swapping provides a general framework which allows for different criteria of judging mesh quality. This permits criteria other than the Delaunay circumsphere test. For instance, we can maximize the minimum face (or solid) angle, or minimize the maximum face (or solid) angle, or reduce the number of edges and volumes in the mesh. Local optimization is not guaranteed to yield the globally optimal mesh, but typically it produces significantly improved meshes.

- 3-D edge swapping permits local optimization based on many possible criterion such as:
  - Circumsphere
  - Maximize the minimum face (or solid) angle
  - Minimize the maximum face (or solid) angle
  - Minimize the total number of edges and volume
- Local optimization usually does not produce a globally optimal triangulation but can significantly improve the mesh quality.

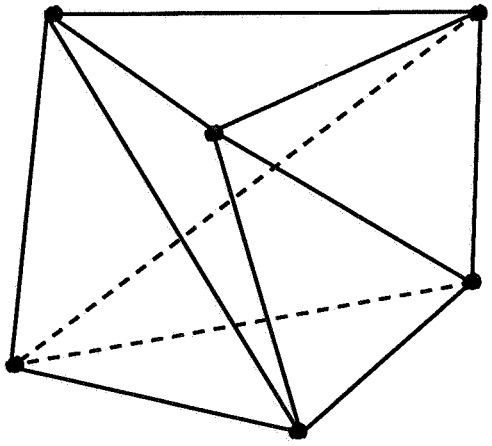


Histogram showing improvement in maximum face angle using edge swapping procedure on triangulation of 500 random sites in unit cube.

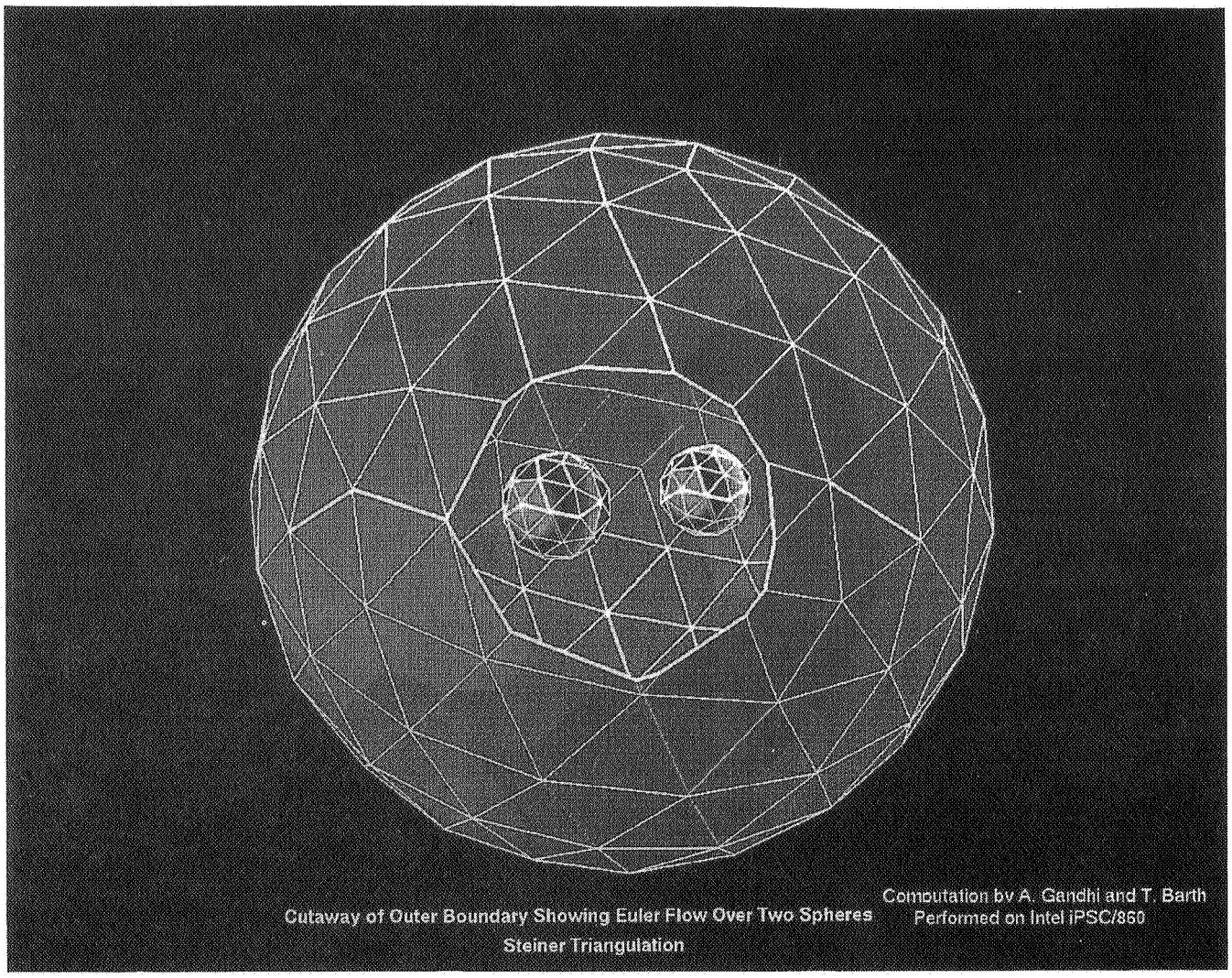
3-D Volume Triangulation

In order to perform our 3-D Incremental Insertion and Optimization algorithm for 3-D Volume Mesh Generation, we need a constrained Delaunay triangulation of the boundary in which we can incrementally add new sites. This Constrained Delaunay triangulation of the boundary sites is constructed using the Tanamura algorithm. Steiner points are introduced into this triangulation in a way that optimizes some criterion of mesh quality. We have been working with trying to minimize the maximum cell (tetrahedron) aspect ratio by introducing a site at the circumcenter of the tetrahedron. It should be noted that construction of constrained Delaunay triangulation is not always possible and we may need to add Steiner points to the interior of the untetrahedralizable volume to make it tetrahedralizable.

- Constrained Delaunay triangulation of boundary sites is constructed using Tanamura algorithm implemented by Marshal Merriam at NASA Ames
- Steiner points are inserted into this triangulation which minimize the maximum tetrahedron aspect ratio (or any other user specified criterion).
- Caveat Emptor: Construction of constrained Delaunay triangulation of boundary sites is not always possible unless Steiner points are added to the interior of the volume. Example:



Nontetrahedralizable twisted prism with reflex exterior faces



Cutaway of Outer Boundary Showing Euler Flow Over Two Spheres
Steiner Triangulation

Computation by A. Gandhi and T. Barth
Performed on Intel iPSC/860

CONCLUSIONS

We have demonstrated the feasibility of adaptive Steiner point insertion and edge swapping to generate surface meshes about complex geometries. Edge swapping algorithms have the advantage over certain other standard triangulation techniques because of the ability to improve mesh quality through local optimization. We have also shown how to extend the edge swapping idea to three dimensions using the combinatorial result of Lawson. The resulting algorithm permits adaptive Steiner triangulations about complex geometries.

- Demonstrated the feasibility of adaptive Steiner point insertion and edge swapping to generate surface meshes about complex geometries.
- Implemented an incremental insertion and edge swapping algorithm to generate 3-D volume meshes.
- Demonstrated the unique capability of edge swapping to improve mesh quality using various optimization criteria.

REFERENCES

1. Green, P.J. and Sibson, R.,"Computing the Dirichlet Tesselation in the Plane",The Computer Journal, Vol. 21, No. 2, 1977, pp.168–173.
2. Rajan, V.T. Optimality of the Delaunay Triangulation in R^d. In *Proceedings of the 7th Annual ACM Symposium on Computational Geometry*, 1991, pp. 357–372.
3. Holmes, G. and Snyder, D., "The Generation of Unstructured Triangular Meshes using Delaunay Triangulation," in *Numerical Grid Generation in CFD*, pp. 643-652, Pineridge Press, 1988.
4. Warren, G., Anderson, W.K., Thomas, J.L., and Krist, S.L.,"Grid Convergence for Adaptive Methods,", AIAA paper 91-1592-CP, Honolulu, Hawaii, June 24-27,1991.
5. Lawson, C. L., "Properties of n-dimensional Triangulations" CAGD, Vol. 3, April 1986, pp. 231–246.

Geometry Modeling and Multi-Block Grid Generation For Turbomachinery Configurations+

Ming H. Shih *

and

Bharat K. Soni **

National Science Foundation Engineering Research Center
For
Computational Field Simulation, Mississippi State University

ABSTRACT

An interactive three-dimensional grid generation code, TIGER (*T*urbomachinery *I*nteractive *G*rid gen*ER*ation) has been developed for general turbomachinery configurations. TIGER features the automatic generation of multi-block structured grids around multiple blade rows for either internal, external or internal-external turbomachinery flow fields. Utilization of the Bezier's curves achieves a smooth grid and better orthogonality. A graphic environment utilizing FORMS Library serves as the interface on the Silicon Graphics Inc. (SGI) IRIS 4D platforms. Based on the geometry information with its built-in pseudo-AI algorithm, TIGER generates the algebraic grid automatically. However, due to the large variation of turbomachinery configurations, this initial grid may not always be as good as desired. TIGER therefore provides graphical user interactions during the process which allow the user to design, modify, as well as manipulate the grid, including the capability of elliptic surface grid generation.

The computational mapping, geometry modeling, and the user-interactions are the the main procedures of TIGER. This presentation will cover these issues associated with general turbomachinery geometries. Various examples have been exercised to demonstrate the success of the developed algorithm.

+ This research is supported by a grant from NASA Lewis Research Center.

* Graduate Research Assistant, Ph.D. Student
Student member AIAA
Member Sigma Gamma Tau

** Associate Professor
Member AIAA

C-6

INTRODUCTION

During the last decade, computational fluid dynamics (CFD) has evolved as an essential technique for solving engineering problems associated with flow fields. CFD allows greater accuracy and improves the time-cost efficiency. Applications of CFD are now pratical on the flow field predictions for very complex geometries such as full scale aircrafts, counter-rotating propfans, and coastlines. CFD developments also help the numerical prediction on magnetic field and bio-chemistry. Great advances can be found in the flow field prediction associated with turbomachinery configurations. Acoustic prediction with CFD methods is currently an important issue in turbomachinery applications since environmental problems become the focus of the public. However, due to the complexity of the geometry, numerical grid generation associated with the flow field about turbomachinery systems is difficult and time-consuming. Even with the advances in the general purpose, interactive grid generation codes like GENIE[1,2,3,4] and EAGLEView[5,6,7], it is still a labor-intensive task to generate a grid for such applications which have large variations in design. As a consequence, there arises a strong demand for a grid generation tool to generate quality grids for a large variety of designs of the turbomachinery configurations in a time-efficient manner.

TIGER[7,8,9,10], an interactive grid generation code customized for turbomachinery applications, has been developed to meet this demand. The overall objective of TIGER was to develop an efficient and robust computational grid generation system tailored for complex turbomachinery applications that would be timely enough for engineering designs using computational fluid dynamics. It is written in both Fortran-77 and C languages, with Fortran routines doing most of the mathematical calculations and C routines driving the graphic interfaces. The graphic interface is an application of the FORMS Library[11] which is a graphical user interface toolkit applicable to the SGI IRIS 4D platforms. Figure 1-a is the main panel of the user interface with graphic window. Figure 1-b is an option panel for blade information inputs. TIGER is currently capable of generating structured grids for internal, external, and internal-external flow fields about turbomachinery systems. With a simple switch, the user is able to choose the preferred grid topology. TIGER automatically maps the physical configuration into the computational domain. It reads in various industrial geometry definitions for the solid entities, such as the blades, the hub/spinner and the duct/shroud from the data files in the form of discreted data points. Built-in geometry manipulators in TIGER will revolve, intersect, spline and generate the grid for these surfaces in sequence. Graphical user interactions allow the user to design and manipulate the grid mesh with mouse buttons and the graphic entities on the screen. After the boundary surfaces have all been generated, algebraically or with user's manipulation, the grid generation for each sub-block will take place with the default transfinite interpolation (TFI) or with the elliptic solver system.

CONSTRUCTING PROCEDURES

There are three major procedures in TIGER during the construction of the grid. They are (1) Computational Mapping, which controls the mapping from the physical domain into the

computational domain; (2) Geometry Modeling, which contains several geometry manipulation routines, such as body of revolution, intersection of two surfaces, and curve/surface spline; and (3) User Interactions, which allow the user to design, modify, or manipulate the grid mesh interactively.

Computational Mapping

The establishment of a mapping from the physical domain to the computational domain is the first step toward the generation of the numerical grid. This can be accomplished by dividing a complex 3D flow field into a collection of contiguous, simply connected blocks filled with discreted points. Each of the solid surfaces forms the full or partial sides in a computational block. Figure 2-a is a typical two-block propfan application, and figure 2-b is the computational mapping for such a system.

Various types of grid topology are available in TIGER. If we use the designation "CH" to indicate C-type grid for the overall configuration, and H-type grid for the grid around the blades, there are two types of grid topology available to date. They are HH and CH types. HC and CC types of grid are currently under development. With a comprehensive input from the graphical interface or through the journal file, the user can choose the preferred topology. Based on this input, TIGER automatically maps the physical domain into the computational domain, and requests the user to provide appropriate information through the interface or the journal file.

Geometry Modeling

Geometry modeling is considered as the most delicate part of the whole process. We must pay close attention to preserve the definitions of the geometry surfaces, such as the blades and the hub, during the manipulations of these surfaces. Cubic spline is the most common technique used to spline the curves or surfaces to the desired number of points and point distribution. Cubic spline usually gives a very satisfactory result for a smooth curve or surface. However, it suffers the defects of introducing wiggles for a curve with large curvature change. If the free-end boundary condition is applied to the cubic spline, it is difficult for the curve to keep its slope at the ends.

Inversed B-Spline[12,13] algorithms, developed by Yoon and Thompson and later modified by Yu and Soni, are used for splining curves and are going to be used for splining surfaces in TIGER to avoid the problems caused by the cubic spline. This spline technique calculates the control vertices of an existing curve/surface (inverse calculation), and allows the user to input a new number of points and a new distribution; hence it allows the user the ability to re-spline the curve/surface to the desired mesh without loosing the definition. It also keeps the 3rd order smoothness of the curve/surface, viz., it has continuous second derivatives.

Due to the geometrical characteristic of the turbomachinery being axisymmetric, it provides the advantage to simplify the transformation between the 3D revolved surfaces into 2D surfaces. A simple transformation law may be expressed as the following.

$$m = \sum_{i=2}^{i=NI} \alpha_i \qquad (1)$$

where $\alpha_i = \sqrt{(z_i - z_{i-1})^2 + (r_i - r_{i-1})^2}$

$$\sigma = \theta r_m \qquad (2)$$

where r_m is the reference radius

The inverse transformation from (m,σ) into (Z,R,θ) is done with the aid of the B-spline algorithm.

User Interactions

It is the goal of TIGER to generate the smooth algebraic grid automatically with very few user inputs. However, it is not practical to expect a grid generation code that is smart enough to generate the grid automatically without any need for modification for any design of the turbomachinery configurations, because that there are too many variations in terms of configuration design. Therefore, certain user interactions must be provided for the user to modify minor portions of the grid to achieve the favorable grid. With this in mind, TIGER features three interactive procedures, which give the user the freedom to design their favorable grids. These interactions are to (1) design the "ruler lines"; (2) design the "segment curves"; and (3) manipulate the surfaces. They will be discussed in detail later.

Bezier curve/surface formulation plays a very important role in these interactions. A bi-cubic Bezier's surface can be expressed as

$$r(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{i,j} B_i^m(u) B_j^n(v), \qquad (3)$$

where $B_i^m(u)$ is Bernstein Polynomial of degree m,

$$B_i^m(u) = \binom{m}{i} (1-u)^{m-i} u^i \qquad (4)$$

and m, n are the numbers of Bezier points in u,v direction , respectively

$0 \le u \le 1 \quad , 0 \le v \le 1$

Set n = 0, then Eq. (3) becomes

$$r(u) = \sum_{i=0}^{m} p_i B_i^m(u) \qquad (5)$$

The Bezier's algorithm in TIGER was programmed according to Eq. 3, viz., it is generalized for any degree of Bezier's surfaces. However, TIGER only applies Eq. 5 in its algorithm.

Bezier's curve is used heavily in TIGER due to two of its important properties, which makes it very useful in grid generation:

(1). $\sum_{i=0}^{m} B_i^m(u) = 1$ which implies that the Bezier curve is invariant under translation and rotation. In other words, the curve is independent of the choice of coordinate system.

(2). $\dot{r}(0) = \phi\ (p_1 - p_0)$ *and* $\dot{r}(1) = \psi\ (p_n - p_{n-1})$ where ϕ, ψ are scalars.

This property implies that the Bezier curve expresses the tangents at the end points in terms of difference with the Bezier control points p_1 and p_{n-1}, respectively, multiplied by constants ϕ and ψ.

Design of the "Ruler Lines"

A "ruler line", or "ruler" in short, is nothing but a grid line with constant J–indice. Graphically, TIGER allows the user to use the mouse buttons and graphic entities shown on the screen to "design" the rulers on the meridional surface, i.e. (Z,R) plane, with the Bezier's curve. Once this process is done, the grid generated later in the process will fall into this user–designed trend. TIGER keeps the design process executed by the user into a file in pseudo code format, which allows later reactment. In other words, the user does not need to go through the same process if nothing is changed in this part.

Design of the "Segment Lines"

Similar to the process of designing rulers, this step also allows the user to design the ruler line in segment with the graphical interaction provided by TIGER. It designs segments, however, in (m,σ) plane instead of (Z,R) plane. It therefore assigns the third coordinates θ to the ruler line on the segment basis. User also has to provide the point distribution information during this step for each of the segments.

Surface Manipulation

TIGER generates the surface grid with TFI for each of the surfaces. It may not, however, be the most favorable grid to the user. Therefore, TIGER features the interactive surface manipulation, which allows the user to manipulate the surface with Bezier's curves, elliptic solver, averaging relaxation, and other techniques. A graphic panel with buttons and counters is provided for the user to access those functions easily. To date, such surface manipulation is available for cascade surfaces; i.e. J-constant surfaces. A user may localize the manipulation to an interactively defined zone.

GRID GENERATION

TIGER generates the initial grid by default with the algebraic TFI technique for each surface patch and sub-block. The axiom for dividing the surface patches and volume blocks is decided automatically by an algorithm tracing the critical indices. TIGER converts all of the coordinates into cylindrical coordinates; it matters not if the geometry definition is expressed in Cartesian coordinates or cylindrical coordinates. The reason for keeping the coordinates in a cylindrical system is due to the physical axisymmetricity of the turbomachinery. To generate the surface patches on the hub, for example, if we do the transfinite surface interpolation based on the information from the four boundaries of the surface, we, very likely, are going to loose the geometry definition of the hub, and the surface will turn out to be a surface composed of various "dips" and "bumps". This is definitely not the one we are looking for. It takes additional procedures, such as projection, to bring the transfinite surface back to the axisymmetric hub surface. However, with the transfinite interpolation done in the cylindrical coordinates, the result is very satisfactory. As stated above in the surface manipulation, TIGER also carries the elliptic grid generation algorithm. This is done by transforming the grid from (Z,R,θ) into (m,σ), and, after the elliptic iterations, the grid is transformed inversly back to cylindrical coordinates.

Similar methodology applies to the volume grid generation. The default algebraic grid by TFI is also done in cylindrical coordinates. However, if the grid somehow shows negative Jacobians due to the complexity of the geometry design, 3D Laplace/Poisson elliptic iterations may be performed to smooth out the grid and eliminate the negative Jacobians.

EXAMPLES

To date, TIGER has generated grids for various configurations such as single rotation propfans, counter-rotation propfans, ducted propfans, rotors-stators, and marine propellers. Three configurations are presented in this abstract as examples for internal, external, and internal-external flow fields, respectively.

The first example is the NASA Lewis single-stage transonic axial flow ROTOR-67 configuration with 22 rotor blades and 33 stator blades. This is an example of an internal flow field. The flow field is decomposed into two blocks, with a 49x21x25 grid for the rotor block, and a 45x21x17 grid for the stator block. HH grid type is used due to the physical domain. Figure 3-a is the solid image for this geometry. Figure 3-b is the grid mesh behavior on K=1 surface. Figure 3-c is the J= Jmax surface, i.e. the grid mesh on the outer shroud. Note that there is discontinuity between the rotor blocks and the stator blocks. This is due to the fact that these blocks will be rotated against each other to simulate the physical rotation; it is not necessary to link the grid between these two blocks since the Euler flow solver TURBO[14] developed by Mississippi State University will link the grid lines at each time step.

The second example is an external flow field case, a GE counter-rotation propfan with F4-A4 blade design with 8x8 blade count. The physical domain is decomposed into two blocks with 61x36x16 grid points for the front blade block and 61x36x16 grid points for the rear blade block. Figure 4-a is the solid image for this configuration. Figure 4-b is the grid mesh behavior for K=1 surface. Figure 4-c is the surface grid on the hub, namely, J=1 surface.

The third example is the NASA Lewis 1.15 Pressure Ratio Fan with 12 rotor blades and 32 stator blades, which is an internal-external flow field. This geometry is decomposed into four blocks with an approximate grid size of 200,000 grid points for a passage. A narrow gap between the rotor and the lower surface of the duct is simulated with 4 grid cells in between. Figure 5-a is its solid image. Figure 5-b is the grid mesh for K=1 surface, with a closeup image in figure 5-c. Note that the axial grid lines between the rotor blade tip and the lower duct surface remain near the duct surface and spray out from the leading edge and trailing edge of the duct. The reason for keeping grid lines from spraying off the duct surface before they leave the duct is that it will be easy to generate a viscous grid without too much user interaction.

REFERENCES

1. Soni, B.K., "*GENIE: GENeration of Computational Geometry-Grids for Internal-External Flow Configurations*", Proceedings of "Numerical Grid Generation in Computational Fluid Machanics '88", Pineridge Press, pp. 915-924, 1988.

2. Soni, B.K., and Dorrell, E.W., "*INGRID Interactive Geometry-Grid Generation for Two Dimensional Applications*", AEDC-TR-86-49.

3. Dorrell, E.W., and McClure, M.D., "*3D INGRID: Interactive Three-Dimensional Grid Generation*", AEDC-TR-87-40.

4. Soni, B.K., McClure, M.D., and Mastin, C.W., "*Geometry and Generation in N+1 Easy Steps*", "*The First International Conference on Numerical Grid Generation in Computational Fluid Dynamics*", Landshut, W. Germany, July 1986.

5. Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W., *Numerical Grid Generation: Foundations and Applications*, North Holland, 1985.

6. Thompson, J.F., "*Program EAGLE-Numerical Grid Generation System User's Manual*", Vols. II, III, AFATL-TR-87-15, March 1987.

7. Soni, B.K., Thompson, J.F., Stokes, M., and Shih, M.H., "GENIE^{++}, EAGLEView and TIGER: General and Special Purpose Graphically Interacitve Grid Systems", 30th Aerospace Sciences Meeting & Exhibit, AIAA-92-0071, Reno, Nevada, January 1992.

8. Soni, B.K., and Shih, M.H., "TIGER: Turbomachinery Interactive Grid GenERation", Proceedings of the Third International Conference of Numerical Grid Generation in CFD, Barcelona, Spain, June 1991.

9. Soni, B.K., and Shih, M.H., "*TURBOGRID: Turbomachinery Applications of Grid Generation*", 26th AIAA/SAE/ASME Joint Propulsion Conference, AIAA-90-2242, Orlando, Florida, July 1990.

10. Shih, M.H., "*TIGER: Turbomachinery Interactive Grid genERation*", Master's Thesis, Mississippi State University, December 1989.

11. Overmars, M.H., "FORMS: A Graphical User Interface Toolkit for Silicon Graphics Workstations", Department of Computer Science, Utrecht University, Utrecht, Netherlands, 1991.

12. Yoon, Y.H., "Enhancements and Extensions of EAGLE Grid Generation System", PhD Dissertation, Mississippi State University, May 1991.

13. Mortenson, M.E., "Geometric Modeling", John Wiley & Sons, 1989.

14. Janus, J.M., "*Advanced 3-D CFD Algorithm for Turbomachinery*", PhD Dissertation, Mississippi State University, May 1989.

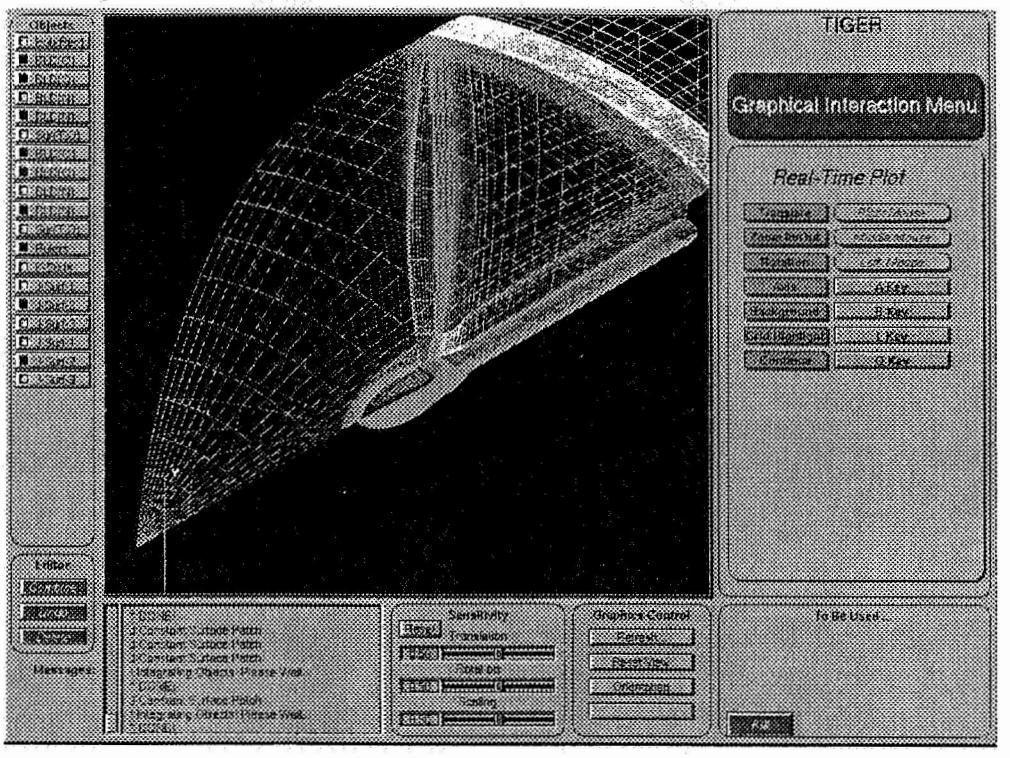


Figure 1–a TIGER's main panel

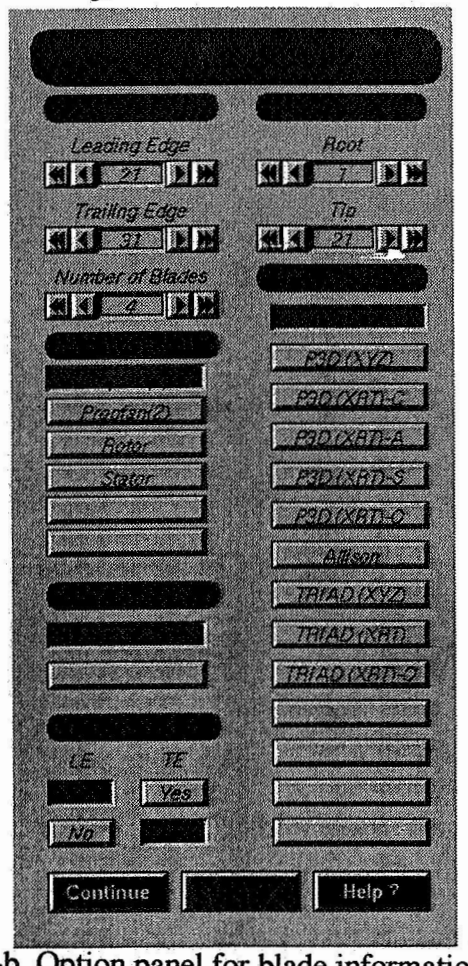


Figure 1–b Option panel for blade information inputs

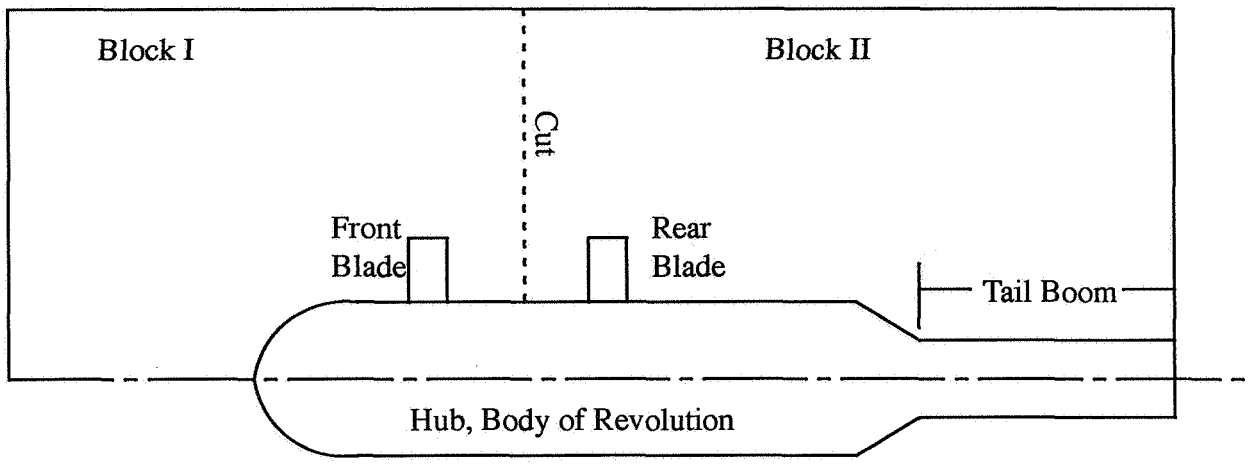


Figure 2–a Two–Block Propfan Application

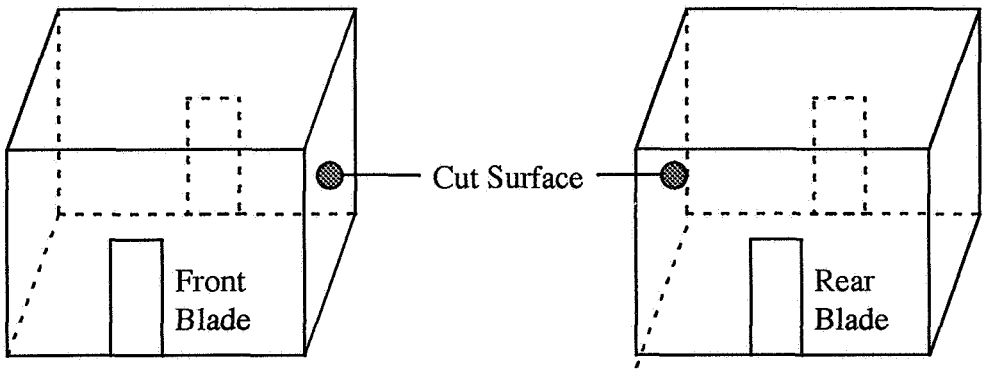


Figure 2–b Typical Computational Domain of a Two–Block Propfan Passage

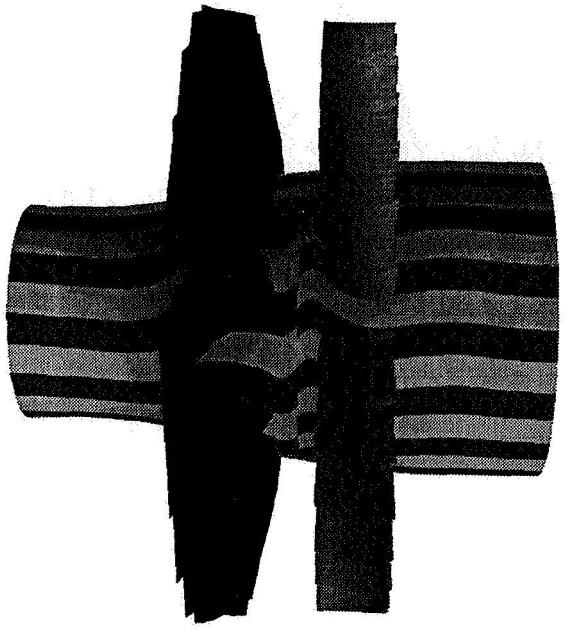


Figure 3–a ROTOR–67 Solid Image

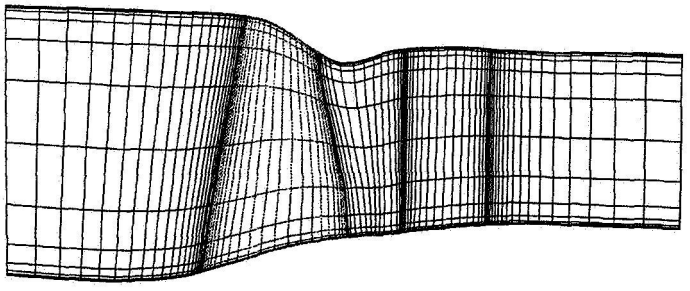


Figure 3–b K=1 Surface for ROTOR–67

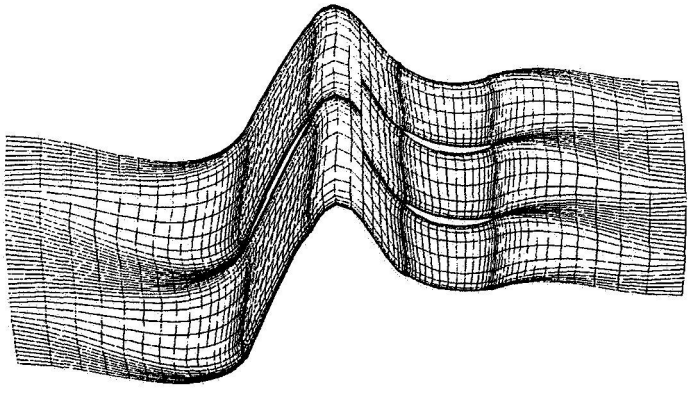


Figure 3–c Mesh on the Shroud

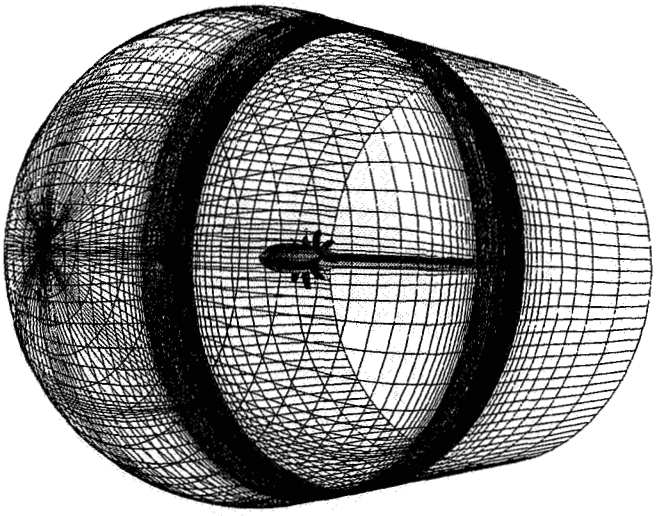


Figure 4–a Counter-Rotation Propfan

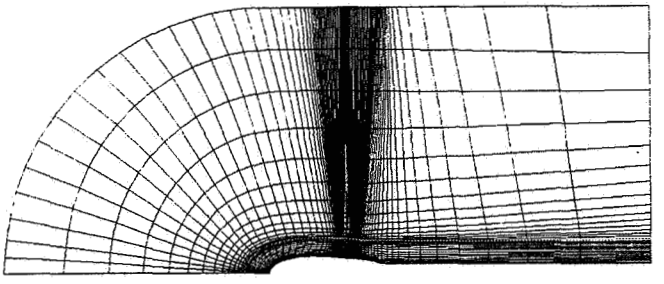


Figure 4–b Grid Surface on K=1 Surface

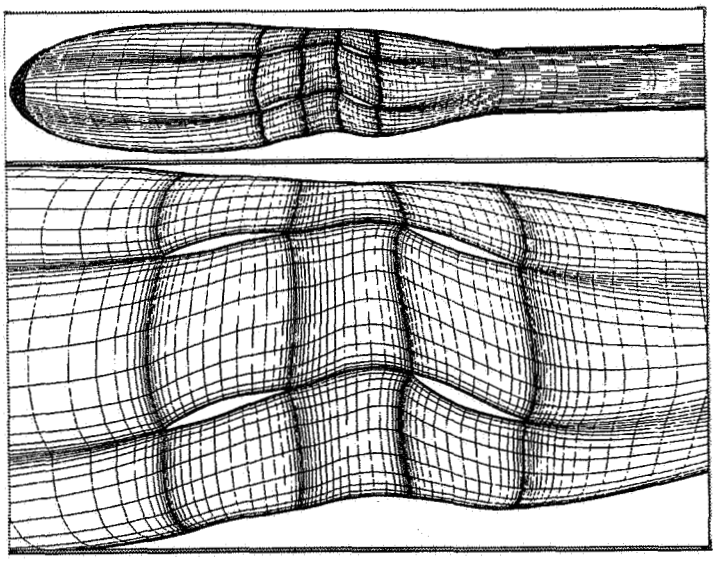


Figure 4–c Surface Grid on the Hub

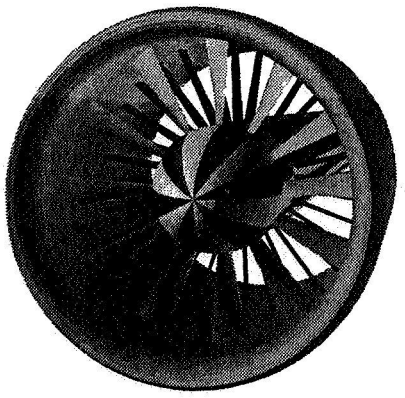


Figure 5–a 1.15 Pressure Ratio Fan

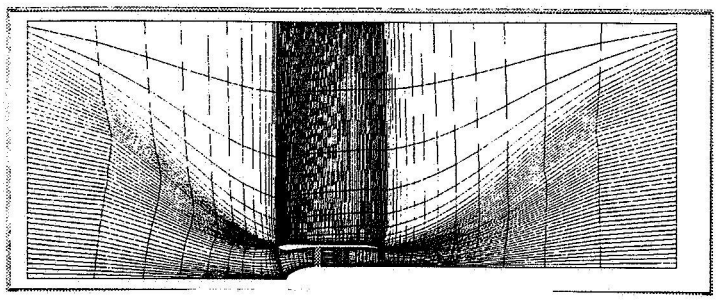


Figure 5–b K=1 Surface Grid

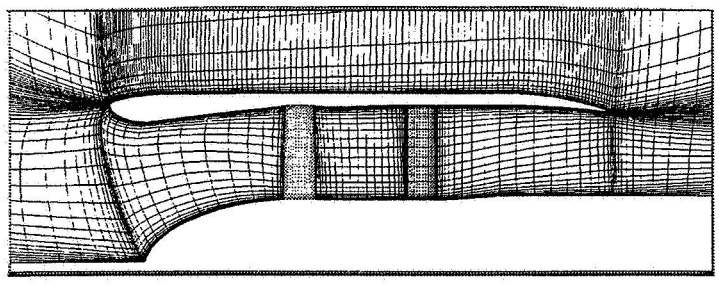


Figure 5–c Closeup Grid for K=1 Surface

A MULTIBLOCK GRID GENERATION TECHNIQUE APPLIED TO A JET ENGINE CONFIGURATION

Mark E. M. Stewart
Institute for Computational Mechanics in Propulsion
NASA Lewis Research Center
Cleveland, Ohio 44135

SUMMARY

Techniques are presented for quickly finding a multiblock grid for a two-dimensional geometrically complex domain from geometrical boundary data. An automated technique for determining a block decomposition of the domain is explained. Techniques for representing this domain decomposition and transforming it are also presented. Further, a linear optimization method may be used to solve the equations which determine grid dimensions within the block decomposition. These algorithms automate many stages in the domain decomposition and grid formation process and limit the need for human intervention and inputs. They are demonstrated for the meridional or throughflow geometry of a bladed jet engine configuration.

INTRODUCTION

A central issue in performing numerical simulations in complex geometries is determining a set of points on which the equations can be discretized, and then finding techniques which will numerically solve this set of discrete equations. There are many approaches to this problem, but the sampling points are either locally structured or unstructured. Structured grids have a trivial local relationship between neighbors while the neighbor relationships in unstructured grids must be stored in tables. This difference has implications for the algorithms which may be implemented, the methods of generating grids and ancillary issues. For example, matrix formulations of implicit schemes on unstructured grids lose the bandedness possible with structured grids. However, when using structured grids, the burden of dealing with complex geometries shifts to how the blocks interact with each other.

For these classes of grids, there are many specific approaches to complex geometry simulations. Euler solutions for aircraft configurations using unstructured grids have been presented by Jameson, Baker, and Weatherill[1] and Lohner and Parikh.[2] Buning et. al.[3] have simulated an ascending Space Shuttle using overset grids, and Chesshire and Henshaw[4] have demonstrated Navier-Stokes solutions with overlapping grids in two dimensions. Non-overlapping, structured multiblock grids and solutions for aircraft configurations have been found by Sawada and Takanashi[5] while Whitfield et. al.[6] have found Euler solutions for counter-rotating propfans where the structured grids are in relative motion.

Stewart[7,8] has used domain decomposition techniques and multiblock grids to find Euler solutions for multi-element airfoil sections and unbladed jet engine configurations.

The focus of this paper is on non-overlapping structured grids and techniques for quickly generating domain decompositions and grids for multiblock flow solvers. There have been several approaches to this problem. Steinbrenner et. al.[9] have demonstrated an interactive system for three-dimensional domain decomposition and grid generation. Vogel[10] has studied knowledge-based techniques for performing domain decomposition. The approach is to define operations acting on bodies in a two-dimensional domain which include enclosing a body with a region, connecting several bodies or dividing a region. The operations are organized in a depth-first search which is managed by a rule-based system incorporating expert knowledge. The rules are specialized to deal with the bodies within the domain. Allwright[11] has demonstrated a three-dimensional grid generation system which starts from an initial approximation to a decomposition and inserts sub-components into the decomposition. Dannenhoffer[12] has presented techniques for two-dimensional domain decomposition which include both the idea of an initial approximation to the topology and specialized rules in an expert system to find a suitable set of grid blocks.

In the following section, an algorithm for determining a domain decomposition is presented. Then, a representation of a decomposed domain is explained which allows rule-driven transformations to be applied. In the next section, an algorithm for dimensioning grids is explained. Lastly, these techniques are demonstrated for a jet engine meridional flow plane.

DOMAIN DECOMPOSITION

The decomposition of a two-dimensional domain may be performed using a search algorithm which finds boundary conforming regions in a two-dimensional domain. In the same way that the skin of a balloon will conform to the bounding walls when blown up in a confined space, this algorithm refines a trivial, coarse approximation to the perimeter of a region so that it conforms to any neighboring boundaries without excessive stretching. To do this, the algorithm uses both a directional probe to look for a bounding point above an interval and a set of rules to interpret the results. Based on the endpoints and which curves they lie on, the rules determine which intervals to probe, which bounding points to add or remove from the perimeter and hence the new subsegments of the perimeter. These subsections and searches are organized in a tree structure which may be significantly pruned by optimizations.

The algorithm is demonstrated in figure 1 by considering the simple case of an airfoil in a box, $PQRS$, and an initial, coarse approximation to the region below the airfoil, $ACRS$. To transform $ACRS$ to a region which conforms to the lower surface of the airfoil, the probe is applied to AB and determines the highest flat ceiling above the middle third of AB, EF, as in figure 1b). Limiting the depth of this probe to the height Search Depth prevents finding points on the segment PQ which would yield a perimeter with excessive stretching. The probe considers the curves which define the domain and finds the point D. The perimeter is then modified to $ADBC$ as in figure 1c), and two child segments, AD and DB, are created from AB.

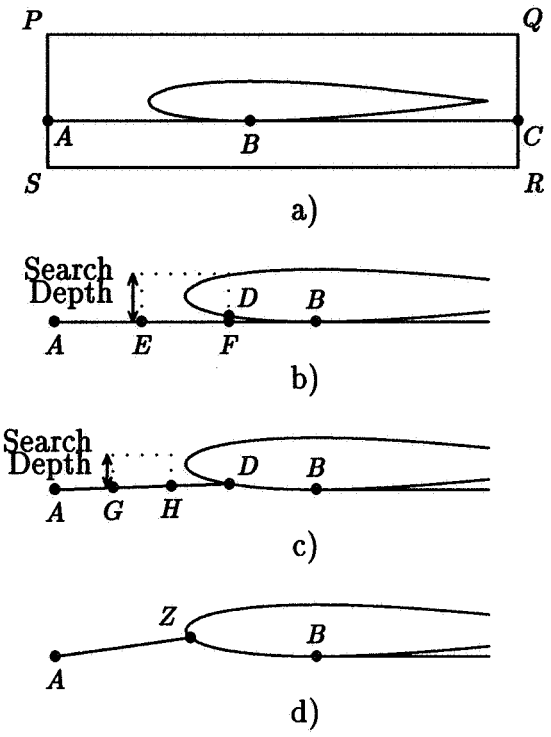


Figure 1. Steps in finding a boundary conforming region.

The refinement continues at the next finer scale by considering AD and DB for refinement. The refinement rules dictate that AB be refined since the endpoints, A and B, lie on different curves. DB is not refined since its endpoints lie on the same boundary defining curve. Probing above the middle third of AD, GH, yields no point within search depth and refinement rules require that the two outer segments, AG and HD, be refined. This failure to find a highest ceiling implies that there will be a transition between bodies in the perimeter. The refinement is extended in a tree structure by considering successively smaller sub-segments as above, and eventually yields the perimeter AZB of figure 1d).

The example demonstrates refinement in one direction on one side of a block, but the technique may be applied on four sides in the outward direction. This allows the perimeter to be extended on all sides. Further, by alternating the direction of search at corners, the directionality of the scheme is reduced.

Once a region is found, it may be removed from the domain to yield a reduced domain. The algorithm may then be applied again to the reduced domain. In this manner, non-overlapping, boundary conforming regions are successively found and removed until the domain is covered.

MANIPULATION OF A DECOMPOSITION

This decomposition strategy finds a decomposition of a complex two-dimensional geometry in terms of topologically rectangular regions. However, for manipulation of this decomposition, a simple and consistent representation is necessary. The decomposition is represented by representation entities linked together in a network to form a semantic network. Geometrical and logical information about the domain and its decomposition is encoded in variables and pointers bundled with each instance of an entity. Processes may then be constructed which traverse this network and decide how to supplement or transform the representation.

A structure is a "C" language programming construct which is used to represent the blocks, curve segments and other entities in the decomposition. A structure bundles together variables describing an entity. For example, a basic entity represented by a structure is the perimeter curve segment, seg, which is a section of the perimeter defining a block. Bundled in its structure are variables describing type and geometrical information and pointers to its neighbors in the perimeter and in adjacent blocks. The links defined by these pointers are shown conceptually in figure 2. By having segments point to neighboring curves in the perimeter, a circular linked list is formed which defines the block's perimeter. By having a segment point to a curve across a block interface in an adjacent block, local decomposition information is represented.

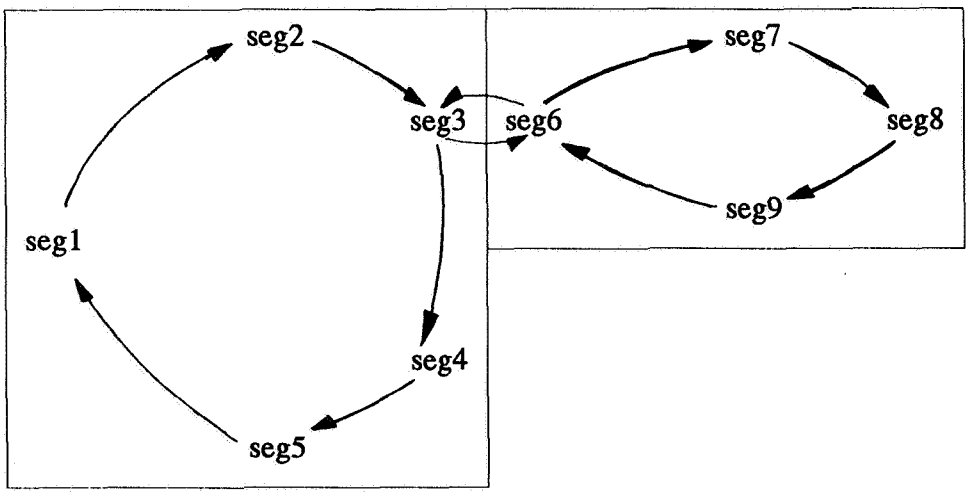


Figure 2. Stylized blocks from a decomposition showing the links both between adjacent blocks and between segments in a perimeter. Additional data is bundled in structures representing each segment.

Replication of these basic, generic units reduces a complex geometry and its decomposition to a set of similar objects (curve segments and blocks) and their important semantic associations. However, processes may then be written which interact with this representation of blocks and segments and supplement it or isomorphically transform it. For example, the perimeter of a topologically rectangular region has four corners. To find appropriate corner positions for a raw perimeter, geometrical information from the perimeter curve entities is interpreted by a set of rules in one process. Similarly, blocks in the decomposition may be straightened by using rules to interpret geometrical information

from the representation and decide where cuts should be made. Further, to balance the computational load of a simulation in parallel, another process uses rules which interpret grid size and topological features to decide where to merge and cut blocks and grids. Each of these processes work on the decomposition and yield rule driven transformations of the decomposition.

GRID DIMENSIONING AND FORMATION

One of the processes which acts on this data structure finds grid dimensions within each block of the decomposition. To have a structured grid within each block, it is clear that the number of cells on opposite sides of a block must be equal. For simplicity in the numerical scheme, it is assumed that normal coordinate lines match at block interfaces. To satisfy this condition, the number of cells on opposite sides of a block interface must be equal. These two constraints give two equations with integer coefficients for each block.

For example, figure 2 depicts two blocks from a decomposition. If the number of cells along a segment is denoted $seg_i \rightarrow n$, then the resultant equations are

$$seg_1 \rightarrow n = seg_3 \rightarrow n + seg_4 \rightarrow n, \tag{1}$$

$$seg_2 \rightarrow n = seg_5 \rightarrow n,$$

$$seg_7 \rightarrow n = seg_9 \rightarrow n,$$

$$seg_3 \rightarrow n = seg_8 \rightarrow n.$$

A system of equations is formed from the equations from all blocks, and this system is generally underconstrained.

The system of equations may be simplified by removing the trivial equations, giving a system where the k^{th} equation is

$$\sum_i a_{ki} n_i = 0, \tag{2}$$

where $seg_i \rightarrow n = n_i \in \mathbf{Z}^+$ and $a_{ki} \in \mathbf{Z}$ are coefficients. If we define z_k such that

$$\sum_i a_{ki} n_i = -z_k, \tag{3}$$

then optimizing the objective function

$$-\sum_k z_k = \sum_k \sum_i a_{ki} n_i \tag{4}$$

with the constraints (3) is a linear programming problem which may be solved using the simplex method.

If in the solution we have $z_k \neq 0$ for any k, then the constraint equations (2) are inconsistent and there is no solution. Since a_{ki} are integers and the simplex method involves operations which are

closed under the rational numbers, the solutions will be rational. Consequently, integer solutions are ensured by taking integer multiples of the dimension solutions.

Since the system of equations (2) is generally underconstrained, the solution is not unique. Consequently, the grid may be adapted to give appropriate resolution of the solution. If $\{n_i\}$ is the solution set, then an integer multiple, l, of the solution, $\{ln_i\}$, is also a solution. Further, the solution vector may also have independent subspaces where solutions, n_i, are not subject to all the constraints (2). In practice, many of the dimensions, n_i, are not constrained at all and may be freely varied. Consequently, adaptation of the solution is often straightforward.

Grids are formed in the multiblock grid based on the perimeter data and grid dimensions. Initial grids are found with a Coons patch followed by elliptic smoothing in the interior of each grid block and at common interfaces. Singularities in the grid must also be smoothed.

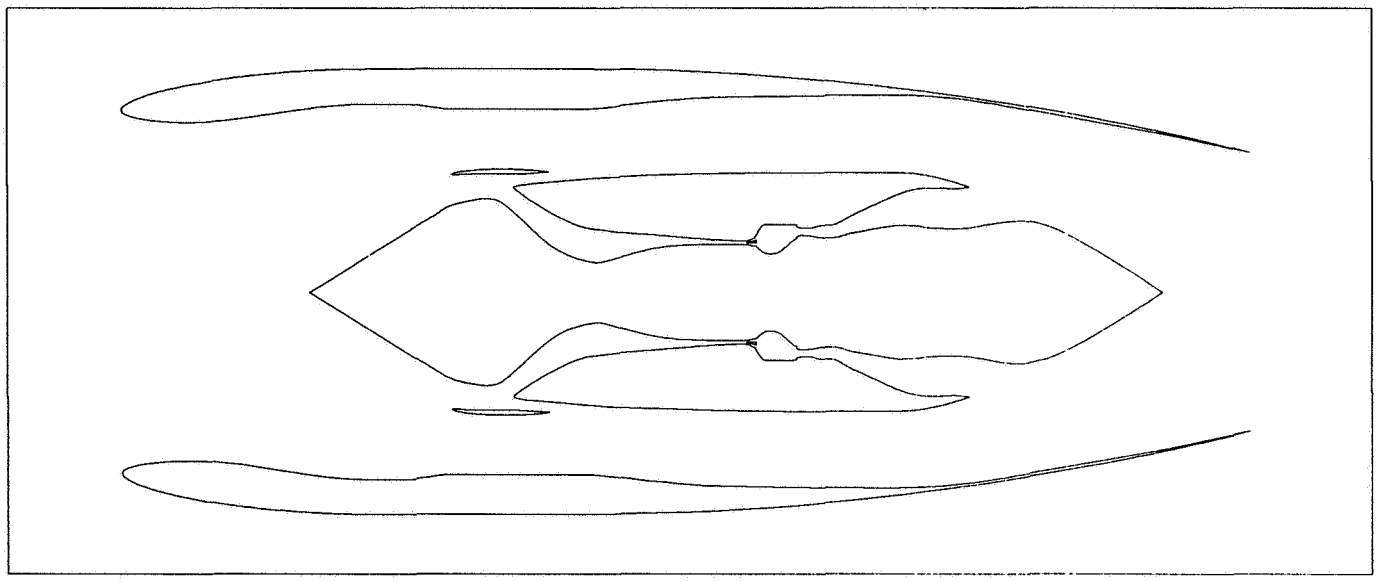
To simplify modeling blade effects in the engine meridional section, these grid smoothing techniques must be specialized to align coordinate lines with the leading and trailing edges of blade rows. The grid may be searched to find the appropriate coordinate line and align it with the leading or trailing edge.

RESULTS

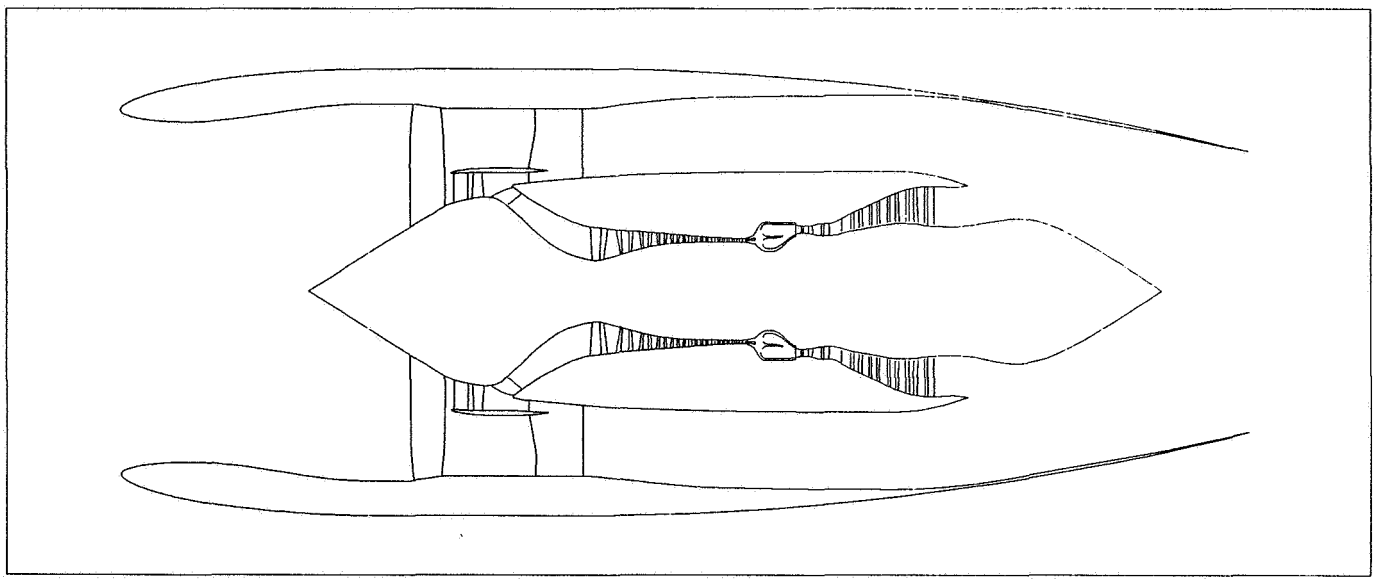
As a demonstration of these techniques, the meridional or throughflow plane of a jet engine is considered. The engine geometry, shown in figure 3, is based on the Energy Efficient Engine (E^3) which was built by General Electric Aircraft Engines under contract to NASA.[13] The engine flow involves external flow about the nacelle and internal flow through the inlet, bypass duct and core compressor, combustor and turbines which are shown in figure 3a. The throughflow geometry is axisymmetric about the centerline and both half planes are included in the domain for solution comparison. In the engine ducts there are alternating stationary and rotating blade rows which turn the flow and either do work by compressing the flow or extract energy as the flow expands. The profiles of these blade rows are included in the grid to facilitate modeling blade effects and they are shown in figure 3b.

The domain decomposition algorithm is applied to this geometry in the form of coordinates for the boundaries shown in figure 3a. As a specialization of the technique to accommodate blade models, block interfaces are forced at the blade leading and trailing edges shown in figure 3b. The resulting decomposition is shown in figure 4. The decomposition is then transformed by several processes traversing the representation. For example, the boundary layer blocks which ensure that grid singularities are off the surface are merged with adjacent blocks. Similarly, the placement of corners is changed. Some manual transformations have also been performed to simplify the decomposition.

The system of dimensioning equations for the final block decomposition (2) contains 76 non-trivial equations which relate 20 of the 230 solution variables, n_i, which represent grid dimensions. The remaining 210 solutions, n_i, are independent of the constraint equations (2). The independence of solution variables is demonstrated in any of the blocks which span a duct in figure 5. The constraint



a)



b)

Figure 3. a) Flowpath geometry for the engine meridional geometry and b) flowpath geometry with blade and combustor component profiles.

equations (1) require the number of cells along opposite walls to be equal and for topological reasons this dimension is independent of all other block dimensions. The dependencies between dimension solutions, n_i, are demonstrated in the transverse direction where the grid dimension couples with blocks throughout the engine.

After solving these equations for a coarse grid, the dimension solution, n_i, is refined and adapted by finding other solutions to the equations. Multigrid levels are added by multiplying all dimensions, n_i, by a power of two. Further adaptation is done by changing variables in the independent subspaces

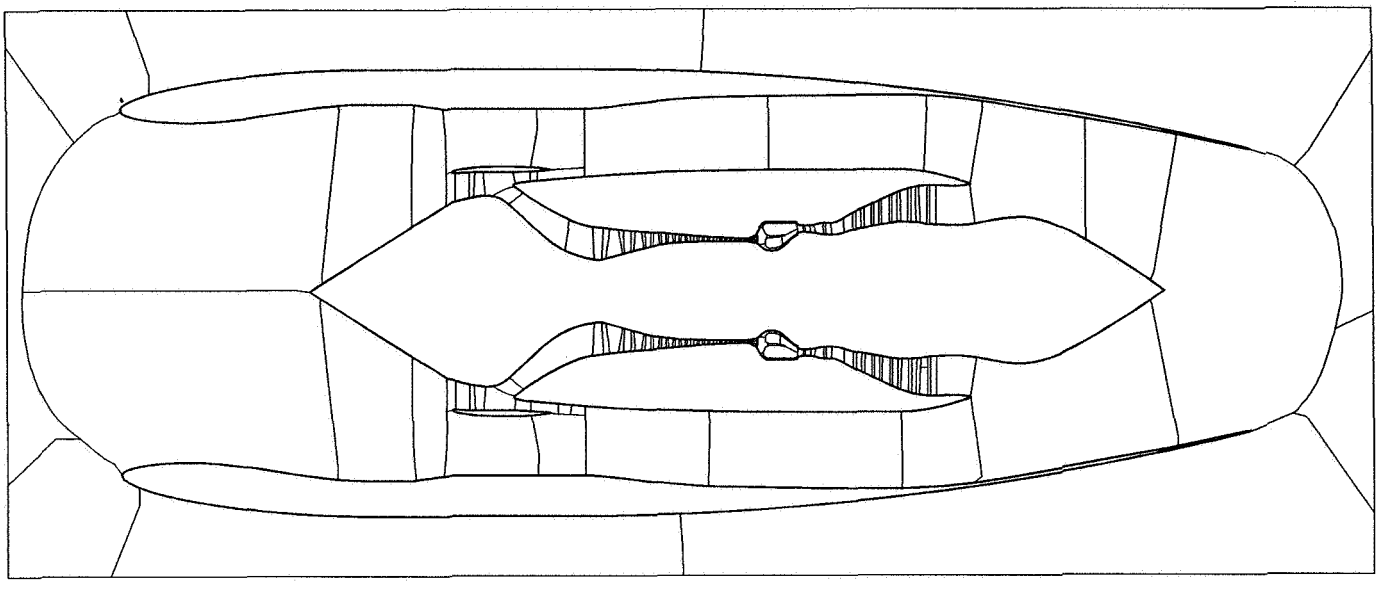


Figure 4. The domain decomposition produced by the decomposition algorithm. Block interfaces are forced to resolve the profiles of blades and other components. The decomposition consists of and is represented by blocks containing curve segments.

noted above. One grid permitted by the dimensioning equations is shown in figure 5. After merging grid blocks for vector architectures, the grid contains 26440 cells in 36 blocks which are indicated by the solid grid lines in the diagram. The grid extends to a circular boundary in the far field.

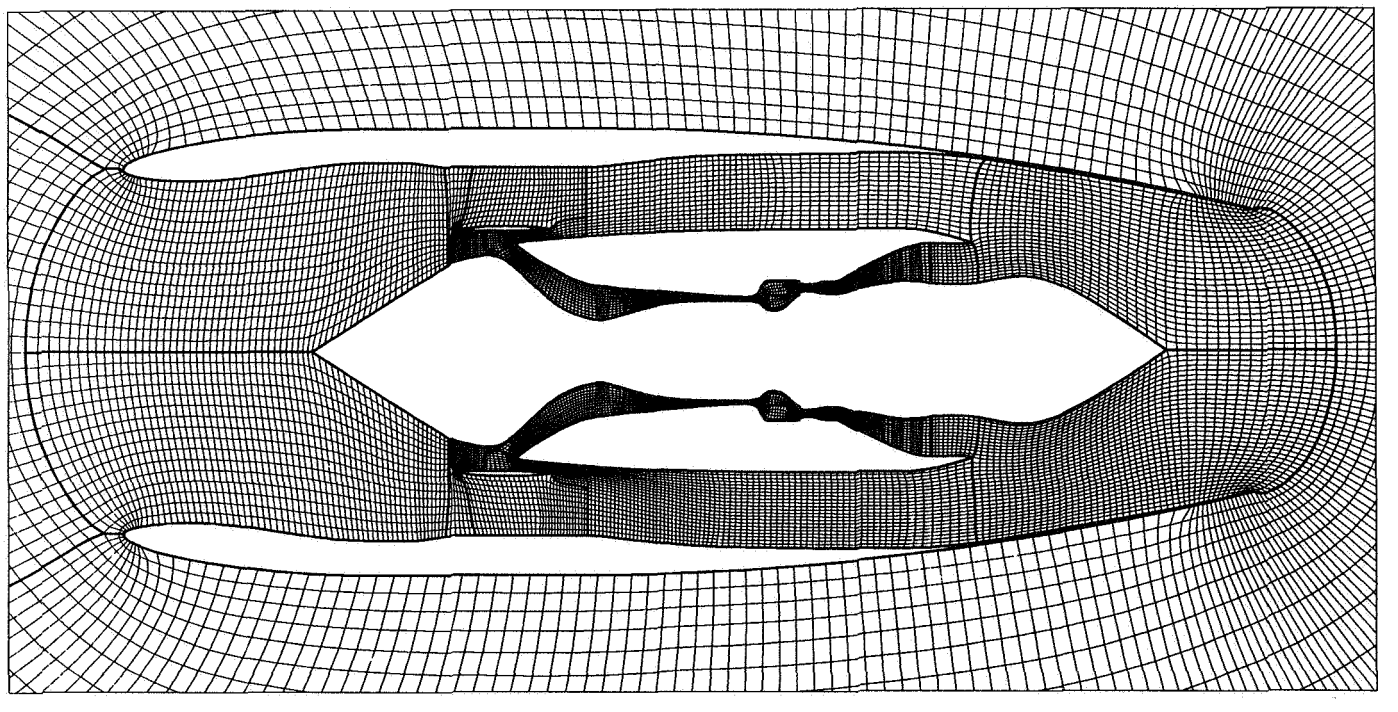


Figure 5. The inner region of the grid for the engine meridional plane. The grid contains 26440 cells in 36 blocks which are indicated by the bold grid lines. Grid lines are aligned with the profiles of blades and combustor components.

CONCLUSIONS

Techniques for generating multiblock grids have been presented and demonstrated for a jet engine meridional or throughflow section. In particular, an algorithm for determining a domain decomposition from boundary coordinate data is explained. Also, representation and transformation of this decomposition is noted. Techniques for dimensioning grids within the decomposition are also formulated. These algorithms automate many stages in the domain decomposition and grid formation process.

REFERENCES

[1] Jameson, A.; Baker, T. J.; and Weatherill, N. P.: Calculation of Inviscid Transonic Flow over a Complete Aircraft. AIAA-86-0103, Jan. 1986.

[2] Lohner, R.; and Parikh, P.: Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method. AIAA-88-0515, Jan. 1988.

[3] Buning, P.; Chiu, I.; Obayashi, S.; Rizk, Y.; and Steger, J.: Numerical Simulation of the Integrated Space Shuttle Vehicle in Ascent. AIAA-88-4359, Aug. 1988.

[4] Chesshire, G.; and Henshaw, W.: Composite Overlapping Meshes for the Solution of Partial Differential Equations. *Journal of Computational Physics*, Vol. 90, 1990, pp. 1-64.

[5] Sawada, K.; and Takanashi, S.: A Numerical Investigation on Wing/Nacelle Interferences of USB Configuration. AIAA-87-0455 Jan. 1987.

[6] Whitfield, D.; Swafford, T.; Mulac, R.; Belk, D.; and Janus, J.: Three-dimensional Unsteady Euler Solutions for Propfans and Counter-rotating Propfans in Transonic Flow. AIAA-87-1197 June 1987.

[7] Stewart, M.: A General Decomposition Algorithm Applied to Multi-Element Airfoil Grids. AIAA-90-1606, Seattle, June 1990.

[8] Stewart, M.: Euler Solutions for an Unbladed Jet Engine Configuration. AIAA-92-0544, Reno, January 1992.

[9] Steinbrenner, J.; Chawner, J.; Fouts, C.: A Structured Approach to Interactive Multiple Block Grid Generation. AGARD FDP meeting Leon, Norway, May 1989.

[10] Vogel, A.: *A Knowledge-Based Approach to Automated Flow-Field Zoning for Computational Fluid Dynamics*, NASA Tech. Memo 101072, April 1989.

[11] Allwright, S.: Techniques in Multiblock Domain Decomposition and Surface Grid Generation, Proceedings of 2nd International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Miami 1988.

[12] Dannenhoffer, J.: Computer-Aided Block-Structuring Through the Use of Optimization and Expert System Techniques. AIAA-91-1585, June 1991.

[13] Stearns, E. M.; *Energy Efficient Engine Integrated Core/Low Spool Design and Performance Report*, NASA CR-168211, Feb. 1985.

BATCH MODE GRID GENERATION: AN ENDANGERED SPECIES?*

David M. Schuster
Lockheed Engineering and Sciences Company
Hampton, VA 23666

SUMMARY

The title poses a rhetorical question which, given the rapid development of interactive surface modeling and grid generation techniques, may appear to have some foundation. In fact, non-interactive grid generation schemes should thrive as emphasis shifts from development of numerical analysis and design methods to application of these tools to real engineering problems. A strong case is presented for the continued development and application of non-interactive geometry modeling methods. Guidelines, strategies and techniques for developing and implementing these tools are presented using current non-interactive grid generation methods as examples. These schemes play an important role in the development of multidisciplinary analysis methods and some of these applications are also discussed.

INTRODUCTION

A great deal of emphasis has recently been placed on the development of interactive software tools for surface modeling and grid generation. This emphasis is motivated and justified by the rapid expansion in capability, and availability of three-dimensional graphics workstations and by the acute need for geometry modeling and grid generation tools capable of accurately describing general, complex configurations. Widely used programs such as GRIDGEN[1], EAGLE[2], and more recently, EAGLEView, have addressed these issues.

The availability of these interactive geometry modeling systems has made it convenient for applications researchers, who previously relied on outside sources for pre-generated grids, to generate their own grids. Thus, these researchers progressed from performing virtually no grid generation to using tools designed to model general, complex geometries using general grid topologies. Often overlooked in this progression is a large database of research and software development for grid generation schemes which operate in a non-interactive or batch mode.

Current interactive schemes provide an outstanding resource for modeling complex multicomponent geometries using arbitrary grid topologies. In many cases, they provide the only means by which some geometries, like engine inlet/airframe intersections, single and multiple stores, etc., can be accurately modeled. However, for a great number of problems, this level of modeling detail and flexibility is not required. Therefore, many researchers new to the field of grid generation needlessly spend countless man-hours developing surface geometry databases, distributing and redistributing points, and generating

* Work performed under NASA Contract NAS1–19000

surface, and ultimately, field grids using interactive programs to model relatively simple geometries.

This paper focuses on schemes designed to efficiently generate grids in one step, starting with a simple definition of the surface geometry of the configuration to be modeled. Specifically, it discusses the techniques and strategies used to develop the Complete Aircraft Mesh Program (CAMP)[3]. CAMP is a multi-block grid generation program which was originally developed to model complete fighter aircraft configurations including wing, fuselage, canard, horizontal stabilizer and symmetry plane mounted vertical fins. It has been extended to model aircraft with vertical fins mounted away from the symmetry plane. Grids have been generated for a wide range of fighter aircraft, transports, and most recently, a generic National Aerospace Plane (NASP) configuration. Examples of grids generated by CAMP and programs developed using various CAMP modules will be discussed to demonstrate the utility of non-interactive grid generation methods for modeling of relevant engineering problems.

The paper also addresses the importance of non-interactive grid generation methods in analyzing multidiscipline engineering problems. Unsteady aerodynamics, aeroelasticity and aeroservoelasticity are examples of disciplines that require deforming geometries to be efficiently modeled while simultaneously calculating the flowfield properties and aerodynamic loads. The strategies used to develop grid generation programs such as CAMP can be efficiently employed within an aerodynamic analysis program to dynamically deform the surface and field grids according to a specified or calculated motion. Examples of how these schemes have been implemented in a three-dimensional aeroelastic analysis method employing Euler/Navier-Stokes aerodynamics will be presented to amplify this point.

The discussion presented in this paper is applicable to the development of single and multiple zone structured grids. Unstructured grid generation presents a completely different set of challenges. The discussion is also primarily directed toward the applications researcher, rather than the algorithm developer. Applications researchers are typically required to model a wide variety of unique geometries and are usually the first to invalidate a given grid generation method. Thus, they typically are in the greatest need of new and more efficient ways to accomplish their work.

STRATEGIES AND TECHNIQUES USED IN THE DEVELOPMENT OF THE COMPLETE AIRCRAFT MESH PROGRAM (CAMP)

CAMP was originally written to provide numerical modeling support for a Euler/Navier-Stokes Three-Dimensional Aeroelastic (ENS3DAE) analysis program developed by the Georgia Division of the Lockheed Aeronautical Systems Company (LASC-GA) under Air Force Contract F33615–87–C-3209, "Flight Loads Prediction Methods for Fighter Aircraft." The main objective of developing the program was to provide a capability which does not require a large amount of user interaction or a high level of grid generation expertise to model complete, structurally flexible, fighter aircraft. The approach taken was to develop a single grid generation program which uses a fixed grid topology that is applicable to as wide a range of geometries as possible. Among the requirements for this development were that the program be made highly modular in structure, use a simple but general input structure and execute efficiently on a wide range of computer hardware. The result of this effort is a program which has been used to

model geometries as simple as an isolated wing to complex wing/fuselage/horizontal tail/vertical tail configurations. CAMP has been run on platforms ranging from supercomputers to low-level workstations.

CAMP exploits the inherently modular nature of aircraft geometries to obtain its high degree of efficiency and flexibility. The majority of fixed wing aircraft can be represented by one to ten major components. CAMP can currently model up to six major components: wing, fuselage, canard, horizontal tail and one or two vertical tails. The input data is arranged according to these major components using a combination of Namelist and free-format input as shown in Figure 1. The geometry input was designed to be as intuitive as possible so that surface data could be obtained from a designer using a Computer Aided Design (CAD) system and converted to the CAMP format with minimal effort. Wing, horizontal tail and canard geometries are input as a series of constant spanwise station cuts, fuselage data as constant axial station cuts and vertical tail data as constant vertical station cuts. Individual components can be added or deleted simply by changing a binary switch in the Grid Control and Index Specification input. Since the program was written primarily from an aerodynamicist's perspective, it assumes that all configurations have a wing, and this component is a required input for every configuration. However, through creative manipulation of the input data, fuselage alone and most fuselage/tail (missile) geometries can be readily modeled using CAMP.

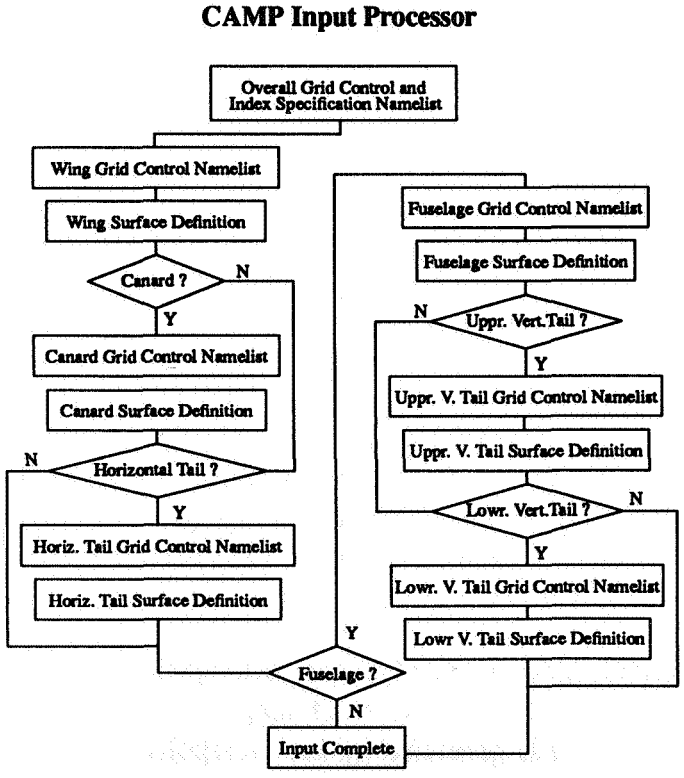


Figure 1. Modular input structure of CAMP.

Previous research indicated that a zonal H-H grid topology, which has the desirable property that the resulting grids are Cartesian like and closely aligned with the streamlines of the flow, could be used to accurately model a wide range of geometries and is especially suited to fighter and transport aircraft[4,5,6]. Thus, a zonal H-H grid topology, shown in Figure 2, was adopted for use in the program. At present,

a given grid is divided into a minimum of two and a maximum of six zones or blocks. All blocking is performed automatically within the code depending on the combination of components chosen to be modeled. Geometries consisting of combinations of wing, canard and/or horizontal tail only are modeled using two blocks, one for the upper section of the flowfield, and one for the lower section of the flowfield. Addition of a fuselage automatically ensures that at least two more blocks will be added, one for the region above the fuselage and one for the region below the fuselage. Symmetry plane mounted vertical tails can be added without additional blocks, but offset vertical tails, as encountered on many modern fighters, require that one or both of the fuselage blocks be split into two blocks.

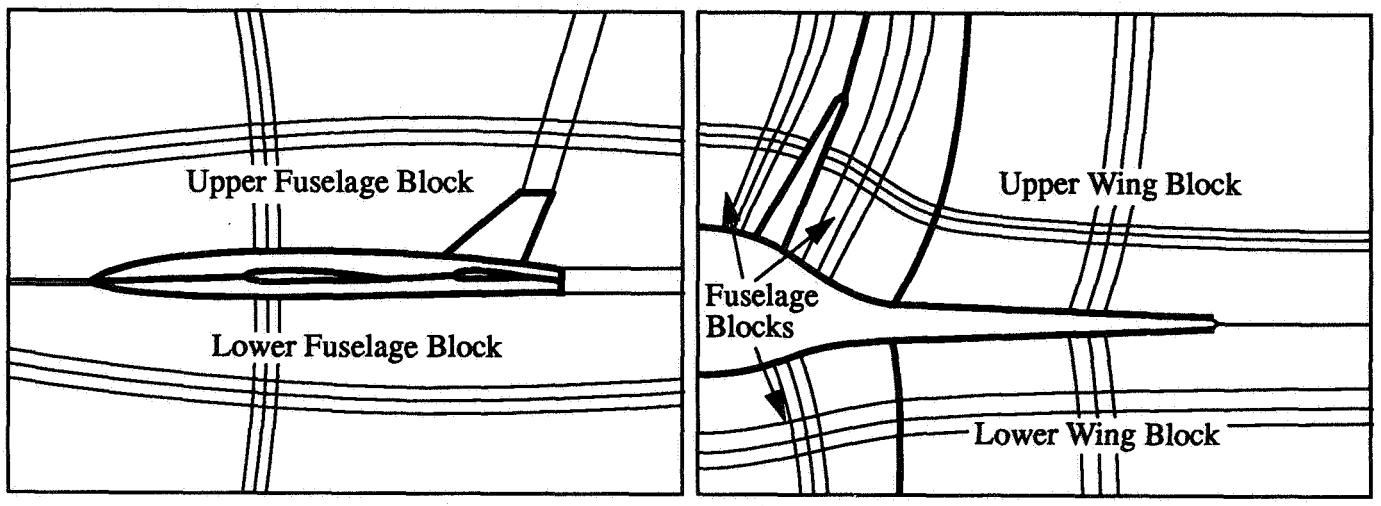


Figure 2. CAMP zonal H-grid topology.

The complete three-dimensional grid is generated in one, two, or three steps, depending on the complexity of the configuration. Like the input processor, the surface and field grid generation also proceeds in a modular fashion. The zones about the horizontal surfaces (wing, canard and horizontal tail) are always generated first. If a fuselage is to be modeled, its grid zones are generated next using the previously defined horizontal surface zones to guide the fuselage grid generation. Finally, the addition of vertical tail surfaces requires manipulation of the fuselage grid, which consists of a series of algebraic shearings and redistribution of points along existing grid lines. The grid generation can be stopped after any of the above steps, and a fully defined three-dimensional grid of the components processed to that point can be output. Therefore, generating grids about different combinations of components entails nothing more than toggling software switches in the input data.

CAMP generates all surface grids using algebraic techniques. The organization of the input data allows the surface grids to be defined using a series of simple, one-dimensional parametric interpolations. For instance, the wing surface grid is generated by first interpolating in the streamwise direction using a set of input grid point indices and spacings. This results in a uniform streamwise distribution of grid points at each defining station. This streamwise interpolated surface is then interpolated in the spanwise direction to obtain the complete surface grid. A similar procedure is performed for each of the remaining components.

An important feature of the CAMP surface grid generation scheme is that all interpolations are linear. Ironically, this is done to preserve the accuracy of the input surface data. Many realistic aircraft

configurations employ sharp edges, such as supersonic wing leading edges, forebody strakes and chines, etc., which play an important role in the aerodynamics of the vehicle. Use of higher order interpolation methods can result in spurious oscillations of the surface data near sharp edges. Some splines, such as tension, Bezier and B-splines, can be used to interpolate without oscillation near these points, but these either require additional user input to control the interpolation, or inordinately smooth the sharp edge. Linear interpolation guarantees that a sharp edge will be captured as long as there is a point on the edge. By using linear interpolation, CAMP shifts the burden of an accurate surface definition to the input data which is presumably coming from a CAD package or an analytical definition. It is generally a trivial exercise to vary the density of points describing the surface using these sources, and thus it is easier to control the accuracy of the surface definition external to the grid generation program than internally.

Field, or volume, grids are generated using a combination of algebraic and elliptic equation techniques. The fields above and below the horizontal surfaces are generated using a quasi-three-dimensional method where two-dimensional grids are defined at each spanwise station and stacked together to define the complete grid. The individual planar grids can be generated using an algebraic, parabolic[7], or fully elliptic[8,9] scheme. The algebraic scheme is always used to generate an initial grid for the parabolic and elliptic methods. It simply performs a linear interpolation between the wing/canard/horizontal tail surface and the outer boundary, with a specified clustering near the horizontal surfaces. The parabolic scheme combines an algebraic and hyperbolic equation grid generation scheme to march from the horizontal surface to the outer boundary. Along the way, it uses an elliptic equation grid generator to smooth the grid. The parabolic scheme has the desirable feature that near the initial horizontal surface, the grid lines transverse to the surface are nearly orthogonal and clustering is easily controlled. However, for geometries with very blunt leading edges, the parabolic grid generator can produce very artistic, but unusable, grids. To counteract this problem, an elliptic grid generator has been added as a post-processor to the parabolic scheme to further smooth the grid. If a large number of post processing iterations are specified, the entire grid generation scheme is equivalent to generating the grid planes using only the elliptic equation solver.

The field grids above and below the fuselage are generated using purely algebraic methods. The fuselage surface grid is comprised of a series of constant axial station cuts whose positions are defined by the axial locations of the grid points in the previously defined horizontal surface grid at its intersection with the fuselage. A series of algebraic shearings, linear interpolations and smoothings are then used to generate the field grid between each fuselage station cut and the outer boundary. Vertical tail surfaces are added by shearing and, where necessary, re-interpolating the fuselage grid. The horizontal surface zones are mated to the fuselage zones by algebraically shearing the wing grid in the spanwise direction.

At present, there is point to point continuity at the zonal interfaces, but the slopes and transverse grid spacings across the interfaces are discontinuous. The current aerodynamic analysis program supported by this grid generator does not require slope and spacing continuity across the interfaces since it does not use differences across the zones. Also, there is no full three-dimensional elliptic equation grid generation capability in the program. If grids with smooth zonal interfaces are required, CAMP could be easily modified to output its grid coordinates and zonal interface information in a format suitable for input into the 3–D elliptic grid generators associated with the GRIDGEN or EAGLE programs. In effect, CAMP would become a preprocessor for these codes. A three-dimensional grid generation module could also

be added to CAMP with minimal effort. Surface grids generated by CAMP have also been used to define surface databases for the GRIDGEN software.

EXAMPLES OF GRIDS GENERATED BY CAMP AND ITS DERIVATIVES

Numerous aircraft configurations have been modeled using CAMP. These geometries range from isolated wings to wing/fuselage/tail configurations. A generic fighter configuration, which possesses many of the features present on modern aircraft designs, will be used to demonstrate some of the capabilities of the program. The geometry consists of a wing, fuselage and twin, canted vertical tails, as well as a fuselage forebody chine. The surface grid generated by CAMP for this geometry is shown in Figure 3. The half-span surface of the configuration is modeled using a total of 13,275 points, 2,394 on the wing, 1,806 on the vertical tail and 9,075 on the fuselage. The full-span geometry shown in the figure is obtained by reflecting the half-span grid about the fuselage symmetry plane.

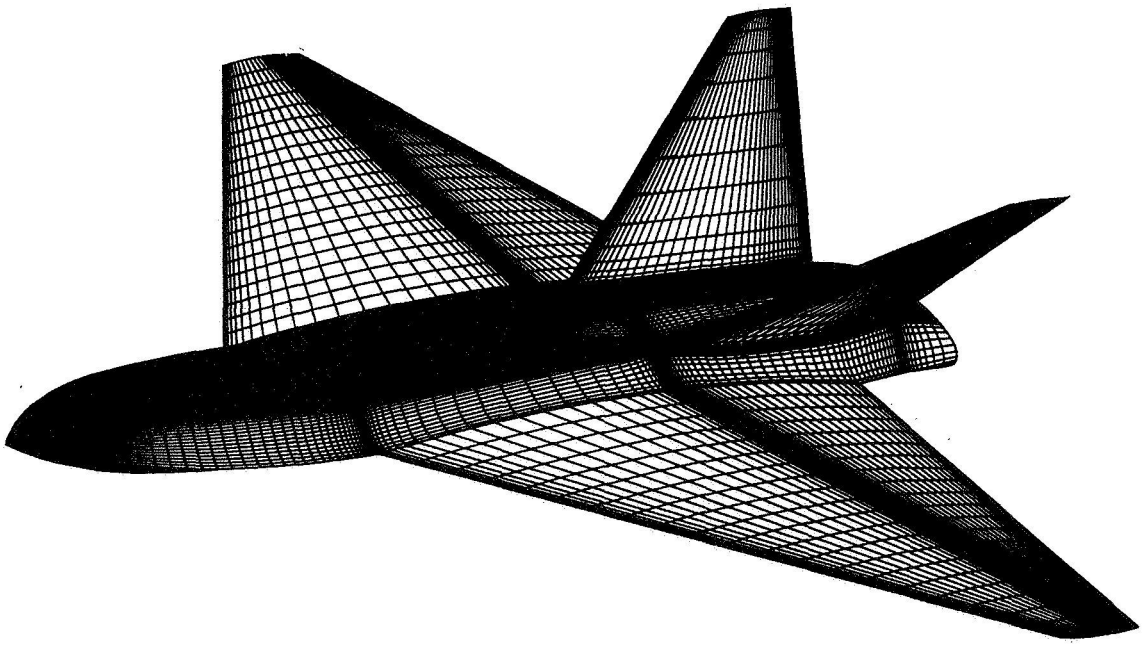


Figure 3. CAMP surface grid for a generic fighter configuration.

CAMP generates a total of five field grid zones for this configuration, one each modeling the flowfield about the upper wing, lower wing, and lower fuselage surfaces. The upper fuselage surface is broken into two zones, one inboard of the vertical tail and one outboard. The grid generated for this case has a total of 693,816 grid points for the half-span model. Each zone consists of 141 points in the streamwise direction and 37 points in the normal direction. All of the zones have 29 points in the spanwise direction except for the zone between the vertical tail and the wing grid which only has 17 spanwise points. All of the aircraft features are accurately modeled by this grid topology and point distribution.

Figure 4 shows a side view of a portion of the grid generated by CAMP at the wing mid-station. The figure shows the grids obtained using each of the three available grid generation schemes. The

algebraic grid consists of straight lines connecting the wing surface to the outer boundary , and the streamwise lines are clustered near the surface to capture the wing boundary layer. The parabolically generated grid maintains a high degree of orthogonality at the wing surface and along the wake lines ahead of and behind the wing. The elliptic grid trades surface orthogonality for a higher degree of smoothness in the interior grid.

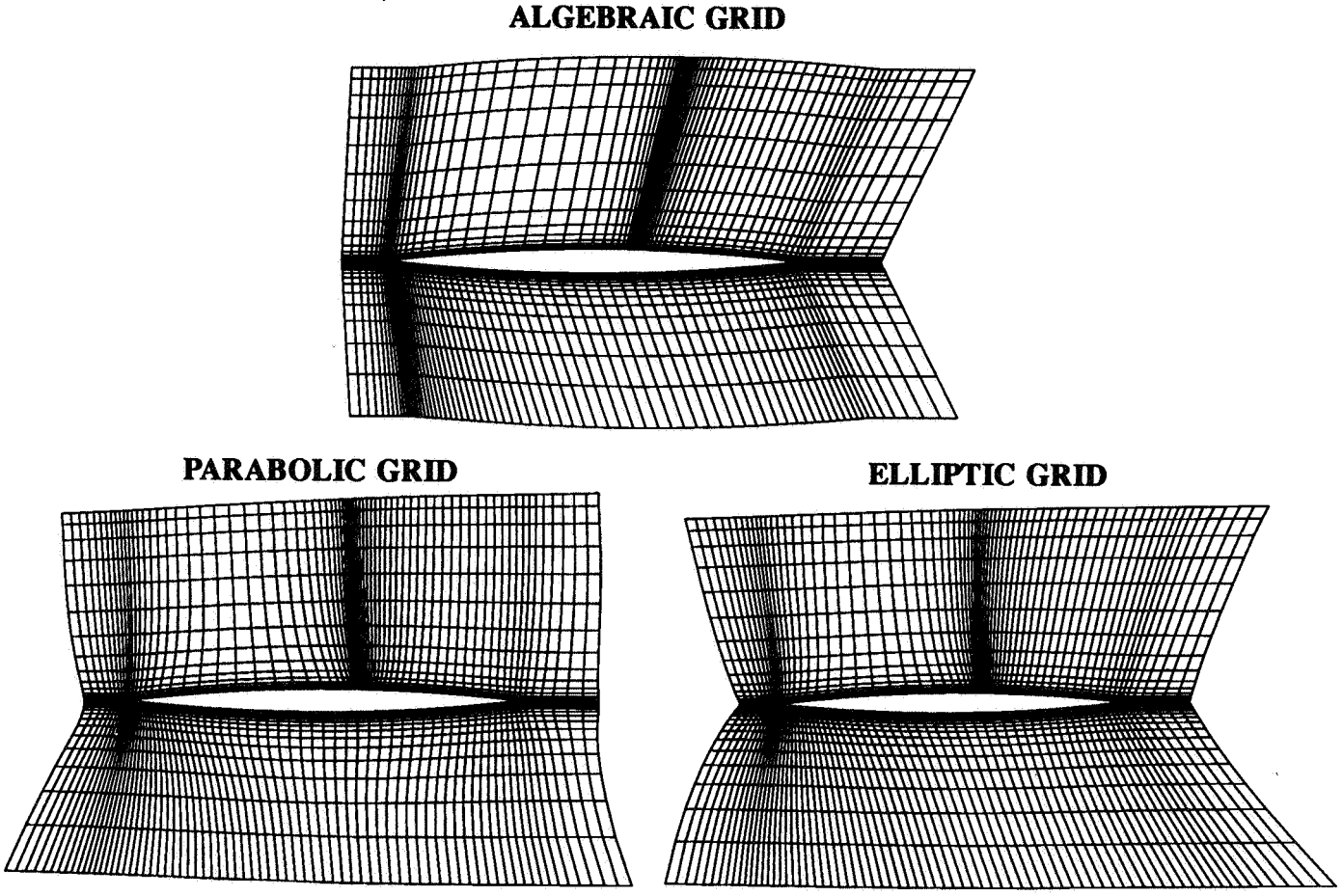


Figure 4. Side view of H-H grid generated about the wing midspan.

A side view of the fuselage grid at the plane which contains the vertical tail is shown in Figure 5. The vertical tail planform is highlighted by the dense clustering of lines near the aft end of the fuselage. The aircraft is modeled as if it were mounted on a sting, but the fuselage could also be closed off or another grid zone added in the sting area to simulate the engine exhaust plume. This grid is generated algebraically, and it accurately defines all areas where high flow gradients are anticipated, including the fuselage nose and boundary layer and the wing and vertical tail leading and trailing edges.

A front view of a section of the grid around the fuselage, vertical tail and wing is shown in Figure 6. In this figure, the interfaces between the various zones are clearly evident. The figure shows the discontinuity of slope and spacing across the zonal interfaces, but the grids within each zone are well behaved.

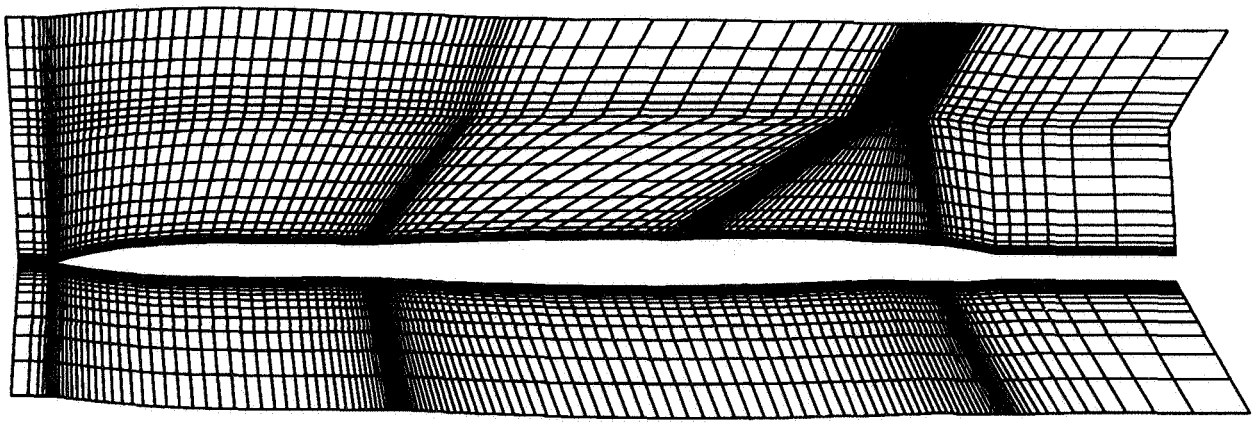


Figure 5. View of the algebraic fuselage grid through the vertical tail plane.

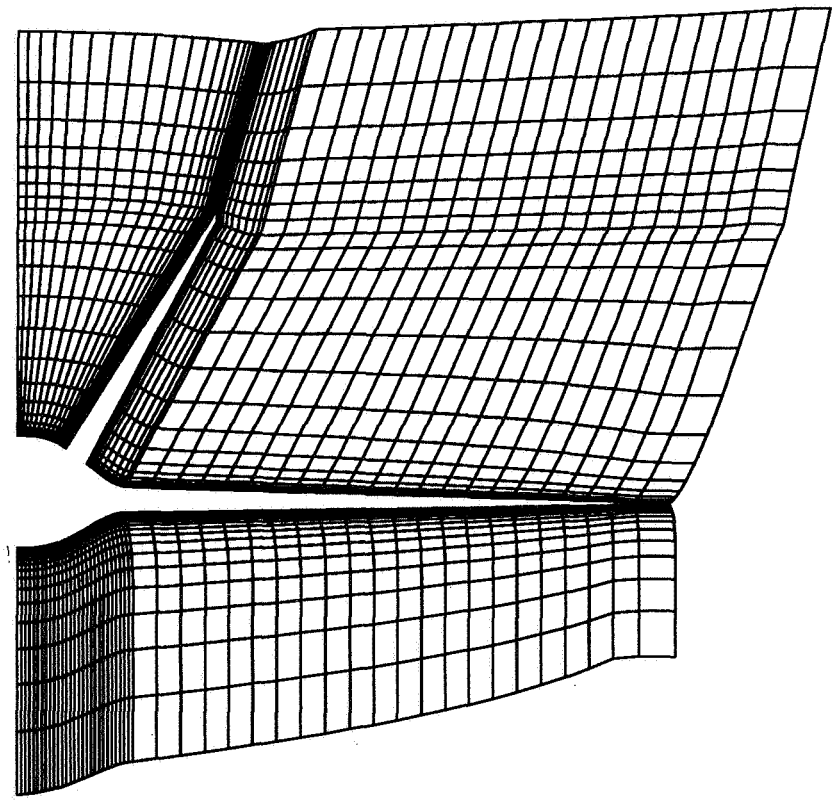


Figure 6. View of patched, zonal grid through the fuselage, vertical tail and wing.

The Cray 2 (Voyager) supercomputer at the NASA Langley Research Center was used to generate these examples. On this computer a grid suitable for aerodynamic computations was generated in less than 15 CPU (Central Processor Unit) seconds. Generation of the grid using the algebraic and parabolic methods are significantly more efficient than the elliptic equation scheme, with the algebraic grid requiring approximately 4.5 seconds and the parabolic grid requiring 14.2 seconds. The elliptic equation grid was defined using 50 iterations at each wing station and required 99 seconds. These times

represent a complete grid generation, including geometry input, surface grid, outer boundary and zonal interface specification and final generation of the full 3–D grid, demonstrating CAMP's high degree of computational efficiency.

It is not the intention of this paper to promote CAMP as the ultimate grid generation program for every application. Therefore, this section will be closed by presenting two examples of grids generated by other non-interactive grid generation programs which are based on the above grid generation strategies. These examples are used to illustrate a number of important points pertaining to the utility of non-interactive grid generation methods. Most importantly, the notion that adoption of a given non-interactive grid generation scheme forces the researcher to adhere to a specific grid topology is dispelled. The modular coding employed in the development of CAMP has been used to advantage in the development of a number of other non-interactive grid generation programs using other than an H-H topology. Both of the following configurations were modeled with single zone C-H grid topologies using a program based on many of the modules in CAMP. Some of these modules, such as the point distribution and clustering routines and the algebraic and elliptic equation grid generators were implemented as "black boxes" with virtually no changes in the original coding. Other routines required minor modifications to be suited to the new grid topology. The input formats for the definition of the various aerodynamic surfaces are identical to those used in CAMP.

Figure 7 shows the surface grid for a generic fighter wing/body developed using the above C-H topology grid generator. The geometry consists of many of the overall features associated with modern fighter aircraft, including a blended wing/fuselage and a highly swept fuselage forebody. This geometry could be easily modeled using CAMP, but the wing and fuselage are so highly blended that a single zone grid models this configuration much more efficiently than CAMP's multi-zone topology. In this case, the entire wing/fuselage configuration is modeled as a wing alone. This allows the geometry and its associated flowfield to be accurately described using a minimum number of grid points without compromising the salient geometric features of the model. Selection of this modeling technique reduced the grid generation effort for this vehicle to a relatively simple exercise and allowed the researchers to focus their attention on accurately predicting the complex flow physics associated with the operation of this type of aircraft at high angles of attack[10].

The applications researcher is often required to perform parametric studies involving a number of geometric changes to a given configuration. These may be as simple as minor changes to airfoil contours on a wing to more complex perturbations like control surface deflections. For these types of problems, each analysis requires that a new grid be generated. If an interactive grid generation method is used, the manpower required to model the various geometries can be formidable. Application of non-interactive methods can greatly relieve this effort. An example of such an application is presented in Figure 8. This surface grid represents a wing with multiple, deflected, leading and trailing edge control surfaces. The C-H grid generation program described above was used to generate the grid about this wing by modifying the input to allow the user to specify an arbitrary number of leading and trailing edge control surfaces and their respective deflections. Thus, by changing a handful of parameters in the program input, the researcher can efficiently generate a series of grids with various control surface sizes and deflections. This discussion leads to the final topic of this paper which addresses multidisciplinary engineering problems.

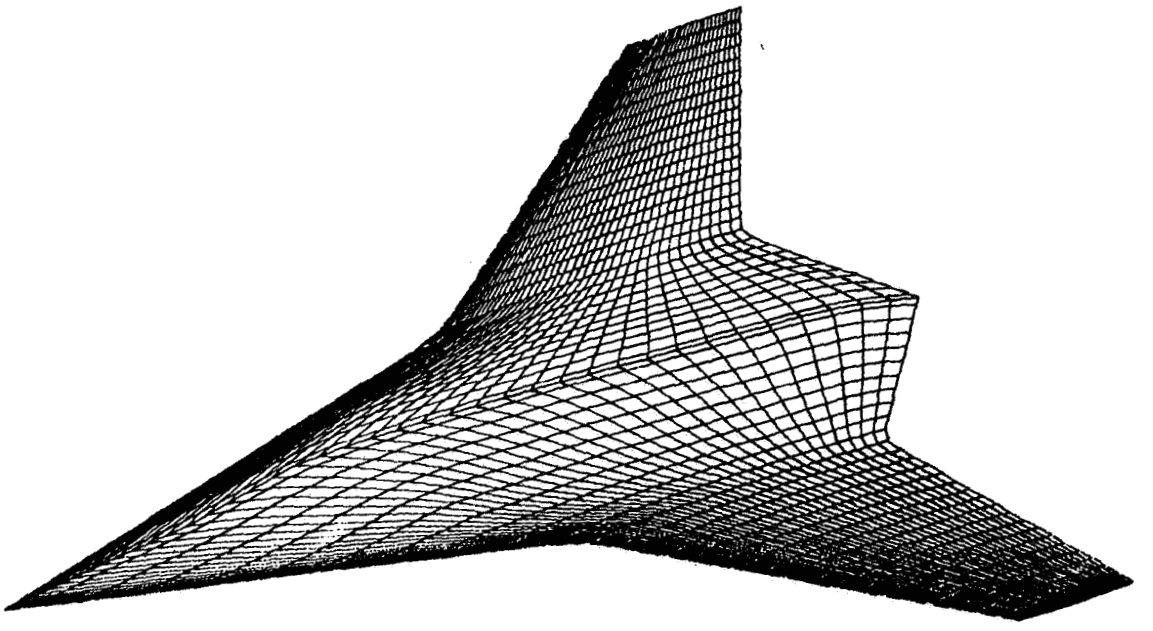


Figure 7. Surface grid for a blended wing/body fighter geometry.

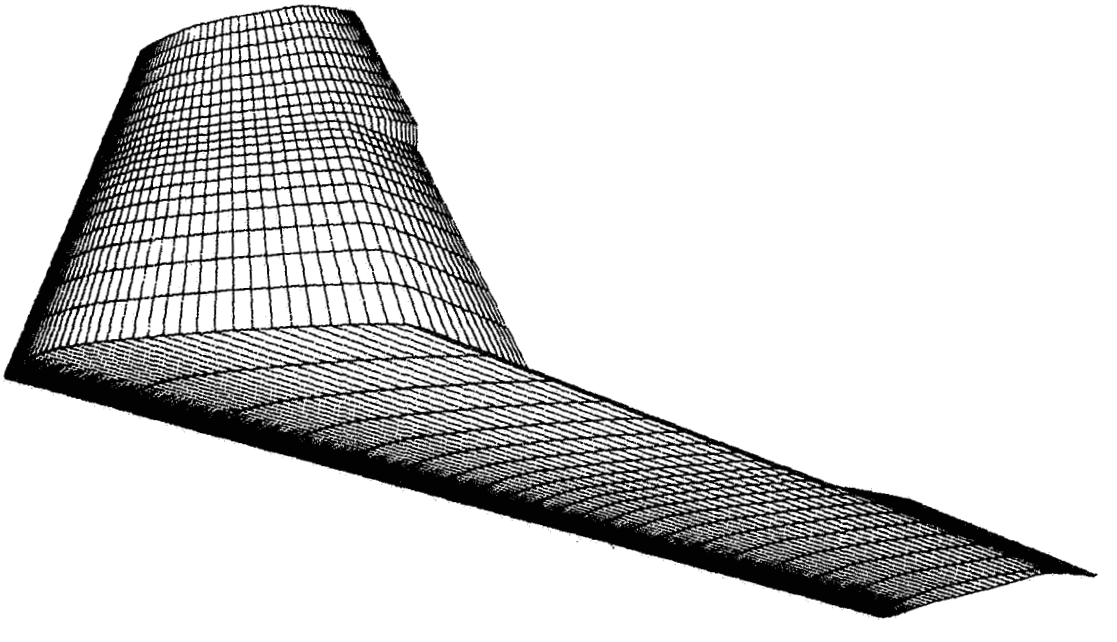


Figure 8. Surface grid for a wing with deflected leading and trailing edge control surfaces.

GRID GENERATION FOR MULTIDISCIPLINARY ENGINEERING PROBLEMS

The problems associated with grid generation for parametric analyses are elevated to an even higher level when one examines the modeling requirements of multidisciplinary engineering analyses. For these problems, the geometry is required to "react" to forces produced by the interaction of phenomena that are peculiar to several engineering disciplines. For instance, aeroelasticity deals with the interaction of the aerodynamics with the structure of a flexible vehicle. In this case, as the aerodynamic load changes, the structure deflects, which in turn changes the aerodynamic load. This process continues until a steady state load and structural deflection are achieved, or in many cases the cycle can continue indefinitely. Aeroservoelasticity is even more complex with aerodynamics, structures and controls all interacting. In each of these cases, the geometry of the vehicle changes, often thousands of times, throughout the course of the analysis. It is not practical to stop the calculation after each geometry perturbation to externally generate a new grid. Therefore, a grid generation capability must be included as an integral part of the multidisciplinary analysis method.

Until recently, researchers have used approximate methods to describe geometric deflections for these types of problems. Many of these involved perturbing the velocity boundary condition at the vehicle surface to simulate the deflection of the geometry. For small deflections, these methods were satisfactory, but many problems preclude the use of these methods. More recent developments have concentrated on physically deflecting the vehicle surface and likewise, the grid surrounding the vehicle. The ENS3DAE aeroelastic method uses a computationally efficient, robust algebraic redistribution algorithm, based on methods developed for CAMP, to update the grid surface geometry and field grids iteratively during an aeroelastic analysis. Batina[11] has employed a spring analogy which has physical basis and is easily applied to unstructured grids as well as structured grids. Kandil and Chuang[12] solve an unsteady equation, known as the Navier-Displacement equation, for the grid motion. Methods employing an elliptic equation grid generator for a small number of iterations to smooth the field grid after each perturbation of the surface geometry have also been suggested. Each method has its relative merits and shortcomings, typically involving a trade-off between computational efficiency and general applicability. Due to its relative simplicity and robust operation, the following paragraphs discuss the method used in the ENS3DAE aeroelastic analysis method. However, while this method is applicable to a wide range of problems, this area is experiencing a high degree of research activity, and the reader is encouraged to examine alternative methods as they are published.

The method employed in ENS3DAE deflects the grid using a simple algebraic shearing which preserves the initial quality of the grid. Originally, the grid was assumed to deflect only in the vertical (Y) direction, but the method has since been extended to accept deflections in all three directions. Figure 9 will be used to describe the original implementation of the algorithm. The basic concept is to update the interior grid such that points near the surface move with the surface, while points far from the surface do not move significantly. This allows grids initially clustered near the surface to resolve viscous flow phenomena, to remain clustered at the surface, while the outer boundary of the grid remains fixed in space. This is accomplished by computing a normalized arc-length distribution for each grid line connecting the deforming surface to the outer boundary. The deflection of each grid

point along that line is then computed by:

$$\Delta Y_K = \Delta Y_{AE}\left(1 - S_K/S_{MAX}\right) \tag{1}$$

where, ΔY_{AE} is the deflection at the surface, S_K is the arc-length along the grid line at point K and S_{MAX} is the total arc-length of the grid line. While this is a very simple and computationally efficient method for updating the grid, it is also quite effective, as illustrated in Figure 10. This figure shows a front view of a CAMP generated wing/body grid before and after an aeroelastic calculation using ENS3DAE. The deflection for this case is significant, and the overall features of the original grid, such as the clustering of grid lines near the vehicle surface is well preserved in the deflected grid. This grid was updated a total of 1000 times during the aeroelastic calculation with no apparent loss of resolution. The time required to perform the grid updates is negligible compared with that required to solve the flow equations and the structural equations of motion.

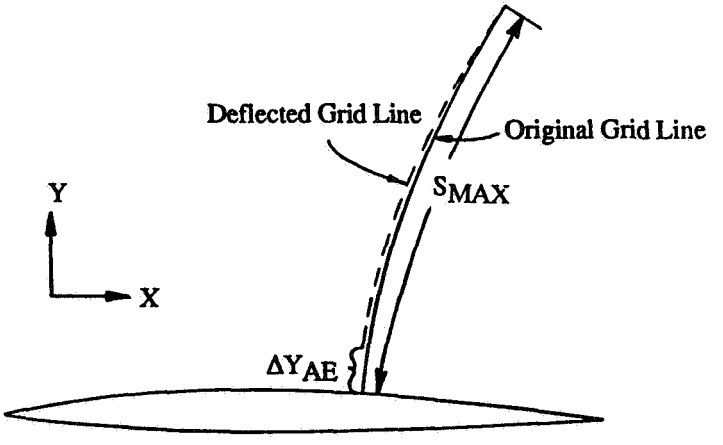


Figure 9. Algebraic grid redistribution method.

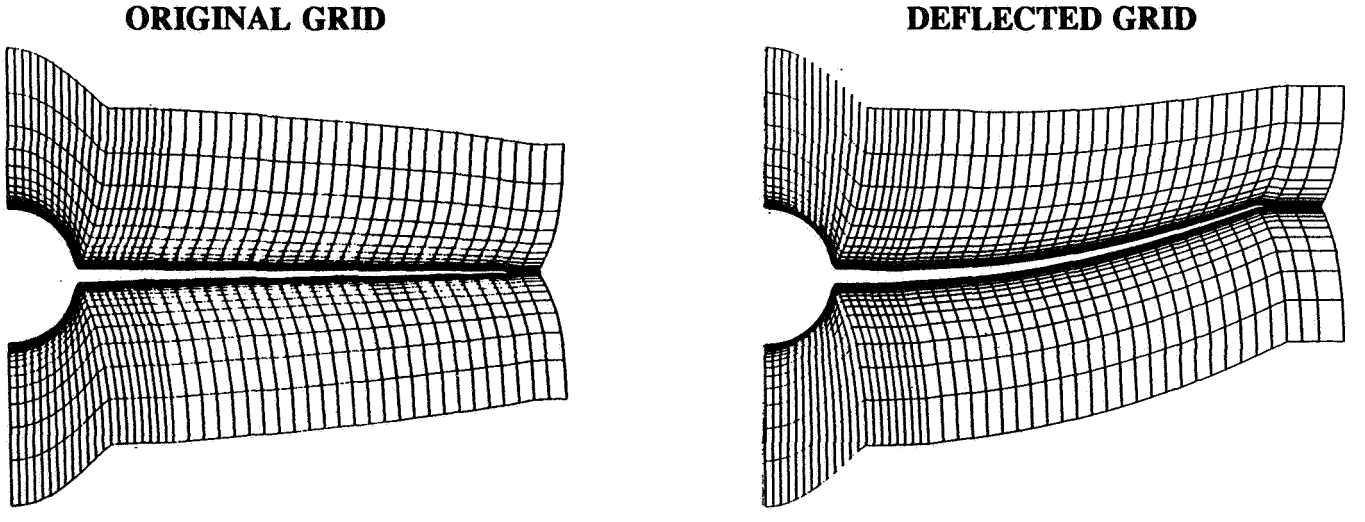


Figure 10. Original and deflected wing/body grid.

CONCLUSION

This paper has discussed the relative merits of geometry modeling and grid generation using methods based on a non-interactive, or batch, mode of program execution. Efficiency and improved productivity have been the primary motivation used to develop an existing non-interactive grid generation program known as CAMP. The overall features of the program, as well as its performance and examples of grids generated by the method have been presented. Methods for generating grids with alternative topologies, and implementation of some of these techniques into multidisciplinary engineering problems has been introduced.

The purpose of this paper is not to "sell" the reader on the CAMP grid generator or any other specific method, but rather to introduce techniques and concepts used to develop this and similar programs. The time and effort which must be devoted to geometry modeling and grid generation for complex configurations has been well documented in literature, and methods utilizing three-dimensional, interactive graphics are well positioned to greatly relieve this constraint. However, there are many problems, some involving relatively complex configurations, which can be effectively modeled using simple, efficient, non-interactive grid generation techniques. Application of these schemes can often reduce the burden of geometry modeling, and significantly improve productivity.

REFERENCES

1. Steinbrenner, J. P., J. R. Chawner and C. L. Fouts, "The GRIDGEN 3D Multiple Block Grid Generation System," WRDC-TR-90–3022, July, 1990.
2. Lijewski, L. E. , et al., "Program EAGLE User's Manual," AFATL-TR-88–117, September 1988.
3. Schuster, D. M. and E. H. Atta, "Flight Loads Prediction Methods for Fighter Aircraft, Volume II — Complete Aircraft Mesh program (CAMP) User's Guide," WRDC-TR-89–3104, November, 1989.
4. Atta, E., L. Birckelbaw, and K. Hall, "Zonal Grid Generation Method for Complex Configurations, " AIAA Paper 87–0276, 1987.
5. Birckelbaw, L. D., E. H. Atta, and C. S. Rubertus, "Euler Analysis and Correlations for Realistic Upper Surface Blowing Aircraft Configurations," AIAA Paper 87–2271, August, 1987.
6. Narain, J. P., "Inviscid Flow Predictions for Complex Aircraft Configurations Using Euler Equations," AIAA Paper 87–2616, August, 1987.
7. Noack, R. W. , and D. A. Anderson, "Solution Adaptive Grid Generation Using Parabolic Partial Differential Equations," AIAA Paper 88–0315, 1988.
8. Thompson, J. F., F. C. Thames, and C. W. Mastin, "Automatic Generation of Body-Fitted Curvilinear Coordinate Systems for Fields Containing Any Number of Arbitrary Two-Dimensional Bodies," *Journal of Computational Physics*, Vol. 15, 1974, pp.299–314.
9. Thomas, P. D., "Composite Three-Dimensional Grids Generated by Elliptic Systems," AIAA Paper 81–0996, 1981.
10. Vadyak, J. and D. M. Schuster, "Navier-Stokes Simulations of Burst Vortex Flowfields for Fighter Aircraft at High Incidence," *Journal of Aircraft*, Volume 28, Number 10, October, 1991, pp. 638–645

11. Batina, J. T., "Unsteady Euler Algorithm with Unstructured Dynamic Mesh for Complex-Aircraft Aerodynamic Analysis," AIAA Journal, Volume 29, Number 3, March 1991, pp. 327–333.

12. Kandil, O. A. and H. A. Chuang, "Dynamic Grid Deformation Using Navier-Displacement Equation for Deforming Wings," AIAA-89–1982–CP, AIAA 9th CFD Conference, June 1989.

TECHNIQUE FOR USING A GEOMETRY AND VISUALIZATION SYSTEM TO MONITOR AND MANIPULATE INFORMATION IN OTHER CODES

Thomas P. Dickens
Aerodynamics Configuration Computing
Boeing Computer Services
Seattle, WA

ABSTRACT

A technique has been developed to allow the Boeing Aero Grid and Paneling System (AGPS[1,2]), a Geometry and Visualization System, to be used as a dynamic real-time geometry monitor, manipulator, and interrogator for other codes. This technique involves the direct connection of AGPS with one or more external codes through the use of Unix pipes. AGPS has several commands that control communication with the external program. The external program uses several special subroutines that allow simple, direct communication with AGPS. The external program creates AGPS command lines and transmits the lines over the pipes or communicates on a subroutine level. AGPS executes the commands, displays graphics/geometry information, and transmits the required solutions back to the external program. The basic ideas discussed in this paper could easily be implemented in other graphics/geometry systems currently in use or under development.

INTRODUCTION

Many computational fluid dynamics (CFD) codes use iterative techniques, or at least a complex solution process that users would like to monitor to allow them to see convergence histories, perturbations to geometry, or other algorithmic processes. Considerable effort would be required to write graphical monitoring code for each such application. Also, many CFD analysis codes have geometry operations written inside the codes. This is frequently done without using the latest state-of-the-art in curve and surface mathematics and without the benefit of the trial and error of investigating variations on a method. These types of problems can be solved quickly with features provided within AGPS.

Programs monitored graphically with AGPS may provide early and detailed insight allowing refinement of algorithms in both precision and in performance. This will be illustrated by the use of AGPS connected to a supersonic marching code running in an inverse design mode.

Beyond passively monitoring codes, the technique allows AGPS to interact with codes, doing geometry tasks that have been perfected within AGPS. This allows rapid prototyping of advanced geometry manipulation and interrogation in other codes without the time-consuming process of building these capabilities from scratch.

OVERVIEW OF AGPS

While developed and used primarily for the preliminary design of aircraft, AGPS is used throughout The Boeing Company for a variety of tasks. The program is a surface geometry system in which virtually any shape can be modeled. The underlying mathematics include cubic and quintic polynomials and rational b-splines for representing curves, surfaces, and solids. The interface consists of a structured programming language with over 150 geometry-related commands, along with a mouse-menu driven interface. Higher-level command files can be constructed and used as a macro capability to do complex or repetitive tasks very simply. Collections of command files are used as high-level packages to allow users to accomplish complex tasks by following an interactive menu-driven session. Hundreds of existing command files are included in the AGPS release, which runs on a variety of machines including VAX, SGI, HP/Apollo, IBM, and Cray. AGPS has been coded to be machine/architecture independent and utilizes a dynamically allocated object data structure. For further information on AGPS, refer to A. E. Gentry's paper, "Requirements for a Geometry Programming Language for CFD Applications",[3] included in these proceedings.

CONCEPTUAL DESIGN OF NEW TECHNIQUE

A short time ago, techniques were developed within AGPS to have a small parent process launch AGPS using a pair of named pipes for communication. When AGPS needs to spawn a new process, the small launch process is signaled to do the task. This is necessary to get around the UNIX problem of using fork and exec, where fork will duplicate the current process' entire program space, then replace it with a new process. With a large number of objects in the data structure, duplicating AGPS would occasionally fail in the fork call. This technique inspired me to develop a new AGPS command, Connect to External Process (CXP).

The CXP command allows AGPS to be driven from an external process with all user input and output routed from and to the external process (Figure 1). The communication is routed through the AGPS launch process, since the communication protocol has already been established there. The simplest form of an external process is a terminal window which passes all strings typed into it to AGPS and echoes all strings which AGPS returns. From here, a user program can construct the individual AGPS commands to accomplish a requested task, providing an on-the-fly dynamic command file generation capability. In addition to command string communication, the external process has access to AGPS internals via (currently) 80+ "subroutine calls", covering areas of AGPS graphics, data structures, memory management, and objects. This is intended to provide enough hooks into AGPS to allow a large variety of tasks to be accomplished easily from within a host of user applications.

The CXP command handles the setup, initialization, and status of communication of an external user program to AGPS. CXP will do as little or as much as needed, from connecting to, or disconnecting from, existing pipes, to creating the required pipes and spawning the specified external process. Once AGPS has connected to the external process and it is active, all input and output is received from and routed to the external process. In other words, the external process takes control of AGPS user I/O.

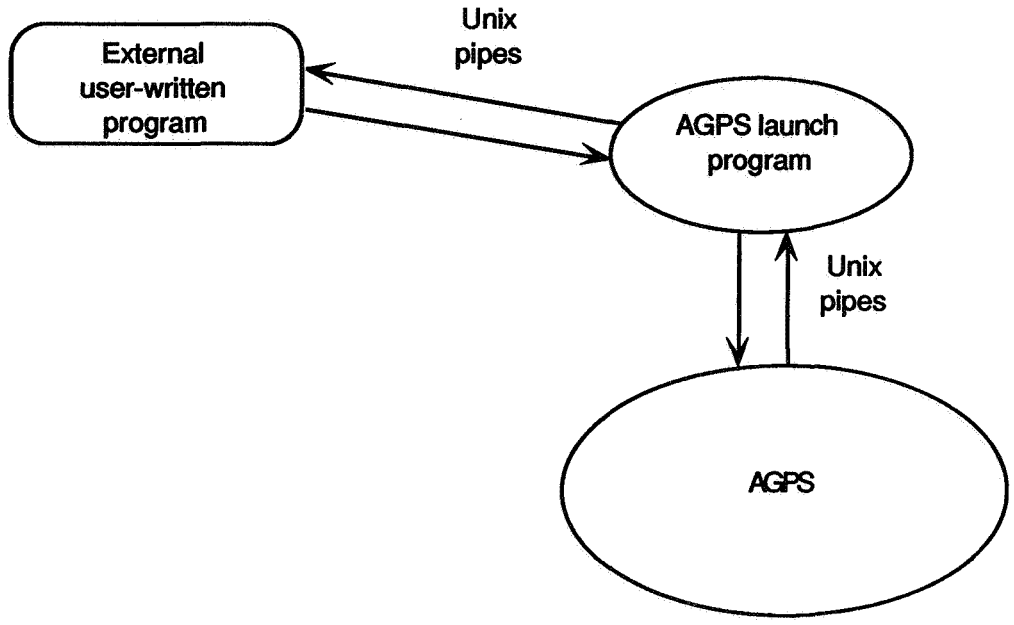


Figure 1. Connection of External Process to AGPS.

TECHNIQUES

Following suit with the techniques used for the AGPS launch process, the connection of AGPS to the external user process is done using Unix named pipes. The pipes are used according to methods outlined in many texts, such as "Unix System Programming",[4] and provides the communication path. The protocol for communication through the pipes was established using fixed-length packets containing command, mode, and data information. Using status commands, a handshaking dialog between AGPS and the external process can be achieved. This quickly became quite involved as more capabilities were explored. While I could get this newly developed subsystem to perform, I soon realized that AGPS users would not want to work at this level of detail. Therefore, I wrote higher-level support routines to do all of the low-level protocol work, allowing a user to simply add a few subroutine calls to their program. These calls took the following form:

connect_2_agps: Create two named pipes, then launch AGPS, instructing it to connect to the pipes and wait for instructions.
command_2_agps: Send the specified character string to AGPS as an AGPS command.
wait_4_agps: After sending a command to AGPS, wait for AGPS to respond. Resulting output from AGPS is also handled.
disconnect_agps: Terminate the AGPS session, disconnect and delete the pipes.

With these four subroutines, usable from either FORTRAN or C, programmers can connect their programs to AGPS and use it to do tasks for their own programs. This follows the AGPS philosophy of providing capabilities to users without restricting or needlessly focusing their use. We have found that users are very imaginative, and many of the currently advertised capabilities using AGPS have been user derived.

When using this technique, it is important to remember that even though your program and AGPS are talking, they are two separate processes and do not share any common memory. If your program has a geometric entity defined, that definition needs to be transmitted to AGPS before AGPS can operate on it, and any results from AGPS need to be transmitted back. To accomplish this, some of the file I/O commands in AGPS were modified to write through the pipes in addition to writing to files.

APPLICATION OF THE METHODS

One application of these methods is to have a user's program transmit some data in the form of geometric objects to AGPS and have AGPS display the objects graphically. This is useful to many scientific programmers who are experts at CFD work or other complex fields, but who don't have the time or the inclination to learn to program graphics on their systems. Using the four above-mentioned pipe commands, they can quickly generate graphics from their programs. During the alpha testing of the CXP capabilities, I used these methods to prototype new view rotation routines, which have been incorporated into the production version of AGPS. I then looked for an interesting program to further validate the method. That case study is highlighted here and represents a part time effort over a couple of weeks.

The program is a supersonic marching code being used for an inverse wing design. Since the flow is supersonic, the effects to changes in the geometry are only felt downstream, allowing the program to simply march downstream. The code was using NPSOL, a general-purpose, constrained, non-linear optimization code based on Stanford University's SOL/LSSOL, SOL/NPSOL, and SOL/QPSOL libraries. At each marching station we used NPSOL to perturb a small number of points in a cross-section of the wing, while attempting to minimize the difference from the current calculated pressure and a specified target pressure. We observed the results of NPSOL to be marginal and very slow (probably, the way we used NPSOL could explain some of this), so we decided to watch NPSOL at work. The marching code was modified to use the four subroutines mentioned above, and a special subroutine was written to construct AGPS commands to represent the desired data in an AGPS geometry format. The data, a composite showing the desired pressure, the current pressure, and the geometry deltas that were applied to the wing section, was displayed and updated in real time by AGPS as the marching code was running (Figure 2).

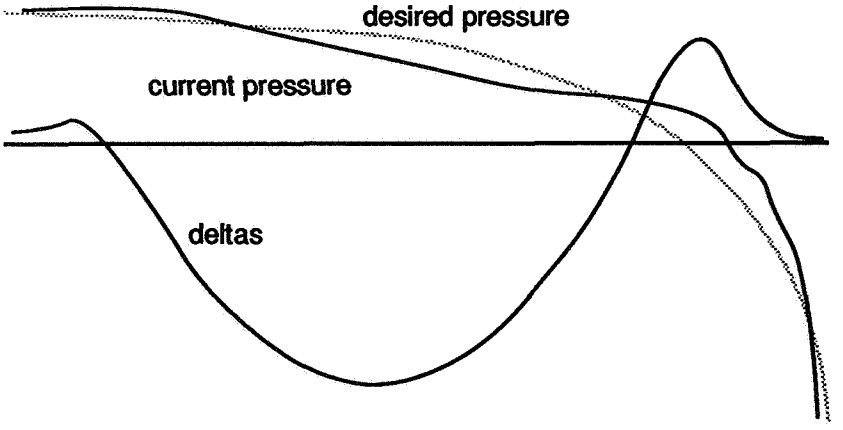


Figure 2. NPSOL Design Status at an Early Iteration.

Each iteration of NPSOL changed the geometry at one design point, then called the user-supplied routine to adjust the grid and recompute the pressure. The resulting plots were very interesting. The current pressure took wild swings from iteration to iteration (Figure 3). After 40 to 50 iterations, things stabilized, and after 120 iterations the objective function was satisfied and NPSOL finished (Figure 4). The slowness was obviously due to the high number of iterations involved, since a refitting of the grid and a pressure solution were needed for each of the 120 iterations for each design station. A higher number of independent design points would be required to achieve a better, smoother convergence to the desired pressure. This would have greatly increased the number of iterations required for each design station, causing the convergence time to be unacceptable.

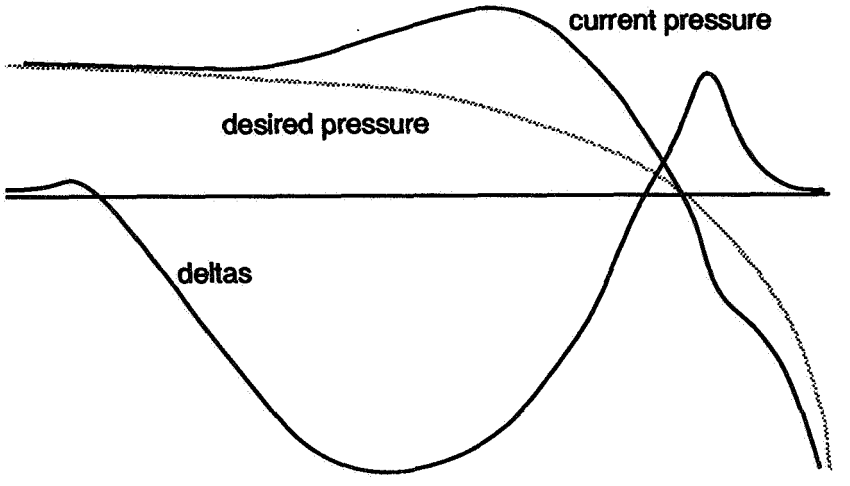


Figure 3. NPSOL Design Status at Iteration 7.

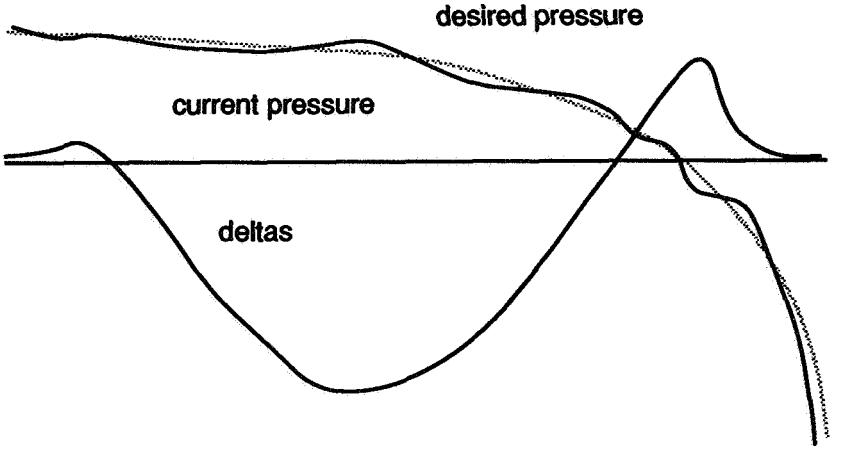


Figure 4. NPSOL Design Status When Completed.

The resulting pressure from the new geometry was similar to the desired pressure, but we observed many wobbles and kinks. When this was shown to an aerodynamicist on the project, he sketched in a number of vertical lines where the current pressure crossed the desired pressure. Remembering his elementary aerodynamics, he then sketched arrows showing the needed movement in the section of geometry to better achieve the desired pressure (Figure 5). Knowing that the flow was supersonic, all of the surface grid points could be intelligently perturbed at the same time, then a new grid could be fit and a pressure solution calculated for the station. Being a computer scientist and not an aerodynamicist, I trusted his observation on blind faith and concluded that I could easily write a subroutine to implement his assertion. I began coding a routine, initially ignoring the fact that the pressure computations occurred on cell centers, while I was applying calculated deltas to grid points. I also ignored the effects of the direction of the velocity vector at the cell midpoints. After trial and error adjustments on the function to calculate the weight deltas, I was able to achieve the desired pressure to within a tolerance much greater than the NPSOL method with fewer than six iterations. The resulting pressure was so close to the desired pressure that a plot of it here would show a single pressure curve.*

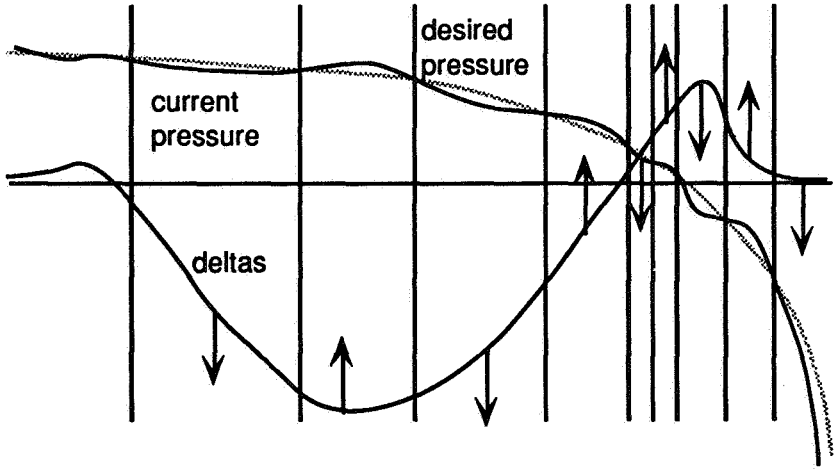


Figure 5. Modifications to the Geometry Deltas to Achieve the Desired Pressure.

We were pleased that in addition to demonstrating a successful application of the AGPS CXP command, we improved the supersonic marching code by more than an order of magnitude in both performance and quality. We were able to use AGPS as a microscope and dynamically dissect and improve this marching code. Now, what else could we do with this?

While we had reached the desired pressure in the design region, there was a problem fitting the new geometry into the old. We experimented in the marching code using crude feathering techniques, but wanted to use more sophisticated methods. We then thought of shipping the "before" and "after" geometry to AGPS, to use proven AGPS methods to blend and smooth the geometry, then ship the results back to the marching code. This was implemented with the smoothing logic residing in an AGPS command file, which could be edited live as the marching code was running. The changes would then be used and seen during the next iteration. The strengths of both codes, the graphics and geometry knowledge of AGPS and the flow-solving knowledge of the marching code, enabled rapid development of the algorithm.

* A short time after these experiments, a coworker in the lab showed us a 1991 paper by W. H. Mason[5] detailing this inverse method we had just implemented. Mason explores much more detail and theory than the implementation presented here.

LIMITATIONS AND FINDINGS

While experimenting with the design code, I timed various operations and tried various things to tune the system. Initially I would construct the geometry inside of AGPS by sending literally hundreds of commands through the pipe to define the desired points, strings, and curves, then by sending the commands to draw the objects. However, the bandwidth of the pipe communication created a bottleneck. I achieved better performance by writing the geometry commands into a file, then sending AGPS the command to read and execute the newly created command file. I admit that I prefer the idea of sending all the data through the pipes, but considering performance, intermediate files are a good solution.

I also investigated using named pipes with an NFS mount to communicate between two machines. After unsuccessful attempts and additional research,[6] I found that even though the named pipes exist on the disk, which allow them to be opened through the NFS mount, the actual communication occurs within the Unix kernel. The packets of data are routed within the kernel from process to process and cannot be passed to another machine.

I then looked into using sockets. The creation of sockets is different than pipes, but they are read from and written to in the same manner as pipes, allowing the same code to do the communication. With sockets however, I experienced performance degradation of two orders of magnitude. Therefore, I recommend using sockets only when the amount of data is small and the benefit of running on two machines is needed. I also experienced occasional sporadic results with sockets involving incomplete packets of data.

CONCLUSIONS

I have described a technique for graphically monitoring codes and performing geometric functions. The technique provides insight to the workings of the codes, and facilitates the task of developing algorithms for them. Additional development time is saved by having AGPS perform geometric tasks, rather than developing and coding these functions independently.

FUTURE WORK

I am interested in refining the use of sockets to offer distributed setups. The developed protocol allows multiple AGPS sessions to be used from a single user's code, which, when combined with sockets, would allow a central program to utilize multiple geometry tasks across a variety of machines. Enhancements to the protocol have been proposed to use an external program as a subsystem of AGPS rather than to control AGPS. Multiple external processes will be able to be dynamically connected to AGPS and accessed as AGPS commands. Subroutine access through the pipes will allow direct manipulation of the AGPS data structure and memory. This can also be used to prototype future AGPS commands and capabilities. As for the future of the inverse design code, ongoing work is being done, with AGPS now an integral part of the effort.

REFERENCES

1. D. K. Snepp and R. C. Pomeroy, "A Geometry System for Aerodynamic Design," AIAA/AHS/ASEE Aircraft Design, Systems, and Operations Meeting, AIAA-87-2902, September 14-16, 1987.

2. W. K. Capron and K. L Smit, "Advanced Aerodynamic Application of an Interactive Geometry and Visualization System," 29th Aerospace Sciences Meeting, AIAA-91-0800, January 7-10, 1991.

3. A. E. Gentry, "Requirements for a Geometry Programming Language for CFD Applications", NASA Workshop on Software Systems for Surface Modeling and Grid Generation, NASA CP- 3143, April 1992.

4. Keith Haviland and Ben Salama, "Unix System Programming", Addison-Wesley, 1987.

5. W. H. Mason, "A Three-Dimensional Inverse Method for Supersonic and Hypersonic Body Design", AIAA 9th Applied Aerodynamics Conference Volume 2, AIAA-91-3325-CP, September 23-25, 1991.

6. M. J. Bach, "The Design of the Unix Operating System", Prentice-Hall Software Series, 1986.

APPLICATION OF THREE-DIMENSIONAL BÉZIER PATCHES IN GRID GENERATION *

C. Wayne Mastin
Mississippi State University
Mississippi State, MS

SUMMARY

Bézier and B-spline patches are popular tools in surface modeling. With these methods, a surface is represented by the tensor product of univariate approximations. The extension of this concept to three-dimensions is obvious and can be applied to the problem of grid generation. This report will demonstrate how three-dimensional patches can be used in solid modeling and in the generation of grids. Examples will be given demonstrating the ability to generated three-dimensional grids directly from a wire frame without having to first set up the boundary surfaces. Many geometric grid properties can be imposed by the proper choice of the control net.

INTRODUCTION

Free-form modeling of surfaces has been the main objective in the field of computer aided geometric design. Rational Bézier curves and surfaces have been used extensively for surface modeling. With this method, as well as most other methods of curve and surface modeling, the geometric object is defined using a set of points and a set of basis functions. In the case of surfaces a tensor product basis is used to construct the basis functions for generating a surface patch. Precisely how the basis functions are used in the surface generation depends on the desired properties of the surface. In particular, it depends on whether the set of points that is used to describe the object is to be interpolated as a wire frame for building the surface or is to be a control net to describe the general shape of the surface. It is the first method, usually associated with Coons patches, that will be used to develop a solid modeling technique. However, either concept can be generalized to three dimensions. In

*Work supported by NASA Grant NSG 1577.

fact, the patches, which probably should be referred to as blocks in the three-dimensional analogue, are actually defined by constructing a control net.

The purpose of this report is to investigate the concept of free-form modeling of surfaces and apply the idea to the field of grid generation. In the following sections, the generalization of rational Bézier curves to three dimensions is developed and used in the construction of three-dimensional grids. The development is analogous to that used to generate surfaces and follows the terminology and notation of Farin [1]. No doubt the extension of one-dimensional basis functions to arbitrary tensor products in any dimension was known to exist from the earliest development of tensor product surfaces, however, the first application of the concept in solid modeling appeared to be in the paper of Saia *et al.* [2]. Another use of three-dimensional basis functions was also note by Farin.

Several examples of three-dimensional grids constructed from wire frame descriptions are presented. Although no detailed analysis of the quality of the generated grid has been done, it has been shown that in most cases the grid is very close to the grid which is constructed using transfinite interpolation.

RATIONAL BÉZIER PATCHES

A three-dimensional solid is given by a wire frame description. The blocks of the wire frame are filled in with patches and a grid is constructed in each patch. Three-dimensional grids are constructed using the tensor products of one-dimensional basis functions. The basis functions for the rational Bézier curve of degree n are given as

$$R_i^n(t) = \frac{w_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)}$$

where the B_i^n are the Bernstein polynomials defined as

$$B_i^n(t) = \begin{pmatrix} n \\ i \end{pmatrix} t^i (1-t)^{n-i}.$$

The weight parameters w_i can be used to cause grid clustering near points of the wire frame.

A three-dimensional patch can be constructed from a control net. If the basis functions are of degree l, m and n, then the points of the control net can be denoted as a three-dimensional array $\{\mathbf{b}_{i,j,k}\}, i = 0, \cdots, l,\ j = 0, \cdots, m,\ k = 0, \cdots, n$. The points of the patch are defined using the tensor product basis functions and the points of the control net. The

points of the patch are described by the parametric equation

$$\mathbf{b}(r,s,t) = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n}\mathbf{b}_{i,j,k}R_i^l(r)R_j^m(s)R_k^n(t)$$

As in the two dimensions, the case of most interest is when $l = m = n = 3$. For it is then that we have enough control points to ensure continuity of grid line slopes between patches or specify any other type of boundary condition for the grid line slopes. Therefore, in the grid construction in the next section, only rational tricubic Bézier basis functions are implemented.

GRID CONSTRUCTION

The grid construction begins with the input of the coordinates of a wire frame which describes the general shape of the three-dimensional region to be gridded. Along with the point coordinates, there is also input information describing the type of grid line continuity or discontinuity that is to be imposed at the wire frame points. This information is used to prescribe the location of the control points in each block of the wire frame. The description of a rational cubic Bézier curve requires four points, including the two end points of the curve. Therefore each block of the wire frame will require a total of 64 control points including the four vertices on the wire frame. These control points are chosen so that the grid lines have the desired continuity or direction property requested by the input information. So as not to generate an excessive amount of input, the directions of the grid lines were computed using differences of coordinates computed on the wire frame. By using various combinations of central and one-sided difference quotients, the grid lines can be constructed so that they have a discontinuous slope or pass smoothly through the points of the wire frame. Except for the boundary points, a control point is needed on each side of the input point in each of the coordinate directions. These control points indicate the direction of the grid line as it enters and leaves the point. The following list indicates the different types of conditions that have been implemented.

1. Weighted central differences used in both directions.
2. Forward difference used on both sides.
3. Backward difference used on both sides.
4. Backward used going toward point, forward used leaving point.

5. Forward or backward differences at a boundary point.

6. Periodic conditions.

7. Rounded corners.

8. Orthogonal grid lines at a symmetry plane.

The weighting referred to in condition 1 was found to give a better value for the slope. The usual central difference can be formulated as the average of a forward and a backward difference. Here a weighted average of a forward and a backward difference is used where the weighting depends on the distance to the two neighboring points on the wire frame.

Several examples are presented to demonstrate the use of the three-dimensional Bézier patches in grid generation. The first example is plotted in Figure 1. The wire frame model describes a simple duct with a varying cross section. The rounded corner option is used to change the duct from a rectangular shape to an oval shape. The plots of the interior grid surfaces show a uniform distribution of grid points. The weights in the basis functions were all set to one for this example. The second example is more interesting. A minimal number of points is used to describe a wing and part of a fuselage. The bulge below the wing is representative of a faired over engine inlet. Weighting of the basis functions has been used to cluster grid points near the wing/fuselage configuration. The symmetry plane was constructed through the center of the fuselage. Grid lines are orthogonal to the symmetry plane. Different control point directions at the wing tips resulted in the corners on the aircraft surface. The wire frame and several gridded surfaces are shown in Figure 2. The last example is another aircraft configuration. The same number of points as in the previous example is used to model a high speed aircraft. Only the basic wing fuselage geometry is modeled. The grid, plotted in Figure 3, demonstrates the capability of dealing with a polar type of axis and a periodic condition. The sparsity in the number of points which describes the geometry is evident in the wire frame.

There are many three-dimensional grid generation methods that can generate a volume grid from a given surface grid. Thus the question might be asked as to how the grids constructed above using the three-dimensional solid modeling technique differs from the interior grid which would be constructed from a set of bounding surface grids. In an attempt to answer this question, the boundary grids from Figure 3 were used to generate interior grids by transfinite interpolation. The grids constructed by the two methods are not identical, but as can be seen in Figure 4, they are nearly the same, and it would be difficult to say which method generates a better grid. Either grid could be improved by applying an elliptic method to reduce skewness. The grid distribution could also be controlled by adding interior

interpolation surfaces in the case of the transfinite interpolation, or by adding points to the wire frame representation in the case of the Bézier grid.

In all of these examples, a uniform knot spacing was used in each patch, however, nonuniform rational Bézier functions could be used or one could use rational B-splines. In any case the basic concept is a straightforward generalization of the surface construction procedure.

One point that is of both practical and mathematical interest has not been treated in this report. That is the three-dimensional analogue of the well-known twist problem in surface modeling. The continuity in the coordinate gradients is not sufficient to uniquely determine all of the control points, and the manner in which these points are chosen determines whether the surface has the correct shape near the vertices of the patch. In our computations, it has been assumed that the mixed derivatives vanish, which in surface modeling is known to result in flat regions near the corners. One of several twist models could have been used, but there would still be left the four interior control points of the block to be determined. Therefore, until there is a reasonable three-dimensional twist model available, it was decided to simply neglect the mixed derivative effects.

CONCLUSIONS

The results of this report show that solid modeling techniques using three-dimensional rational Bézier functions can be used to generate grids in a simple one-step procedure. Many different types of configurations and edge and face treatments can be included in the model. The complexity of the geometric model is only limited by the amount of input that is desired. This solid modeling technique is designed for free-form solids where a general shape or design is to be modeled rather than for constructing a solid with precisely defined edges or faces.

REFERENCES

1 Gerald Farin, *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide, Second Edition*, Academic Press, San Diego, 1990.

2 A. Saia, M. S. Bloor, and A. De Pennington, "Sculptured Solids in a CSG Based Geometric Modelling System", in *The Mathematics of Surfaces II*, R. R. Martin, Ed., Clarendon, Oxford, 1987.

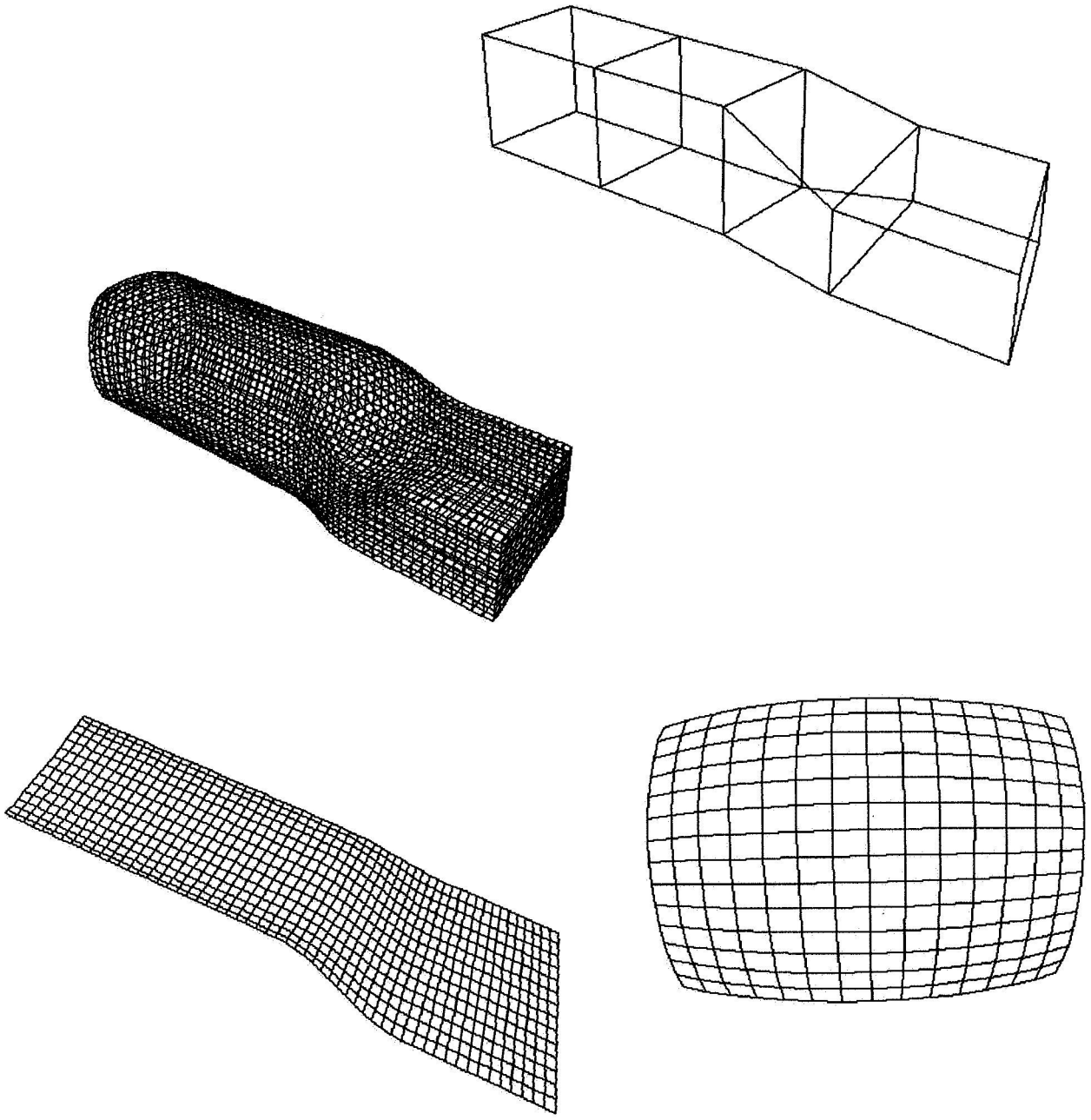


Figure 1. Wire frame and surface grids for a duct with a varying cross section.

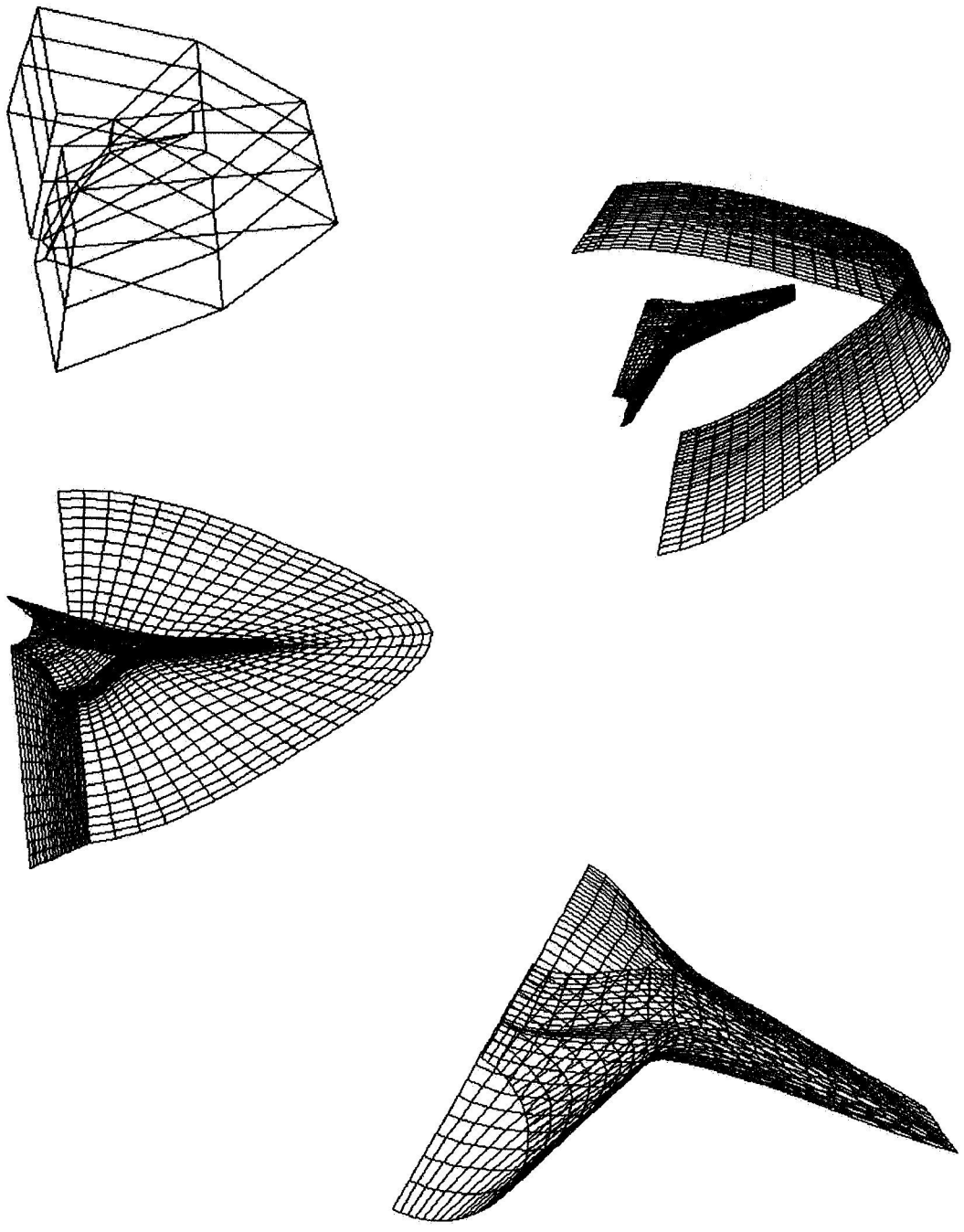


Figure 2. Wire frame and surface grids for a wing and part of a fuselage.

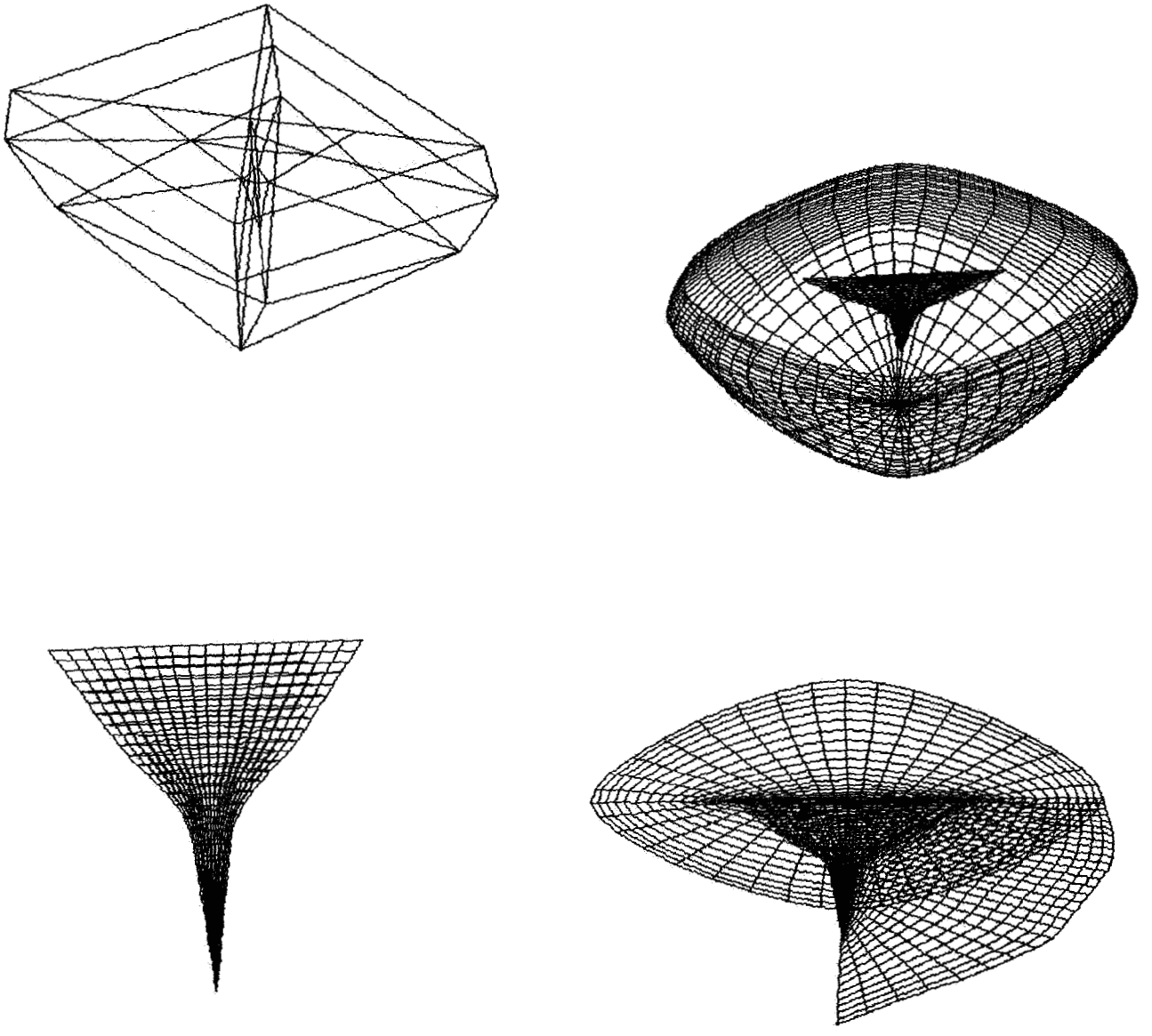


Figure 3. Wire frame and surface grids for a high speed aircraft.

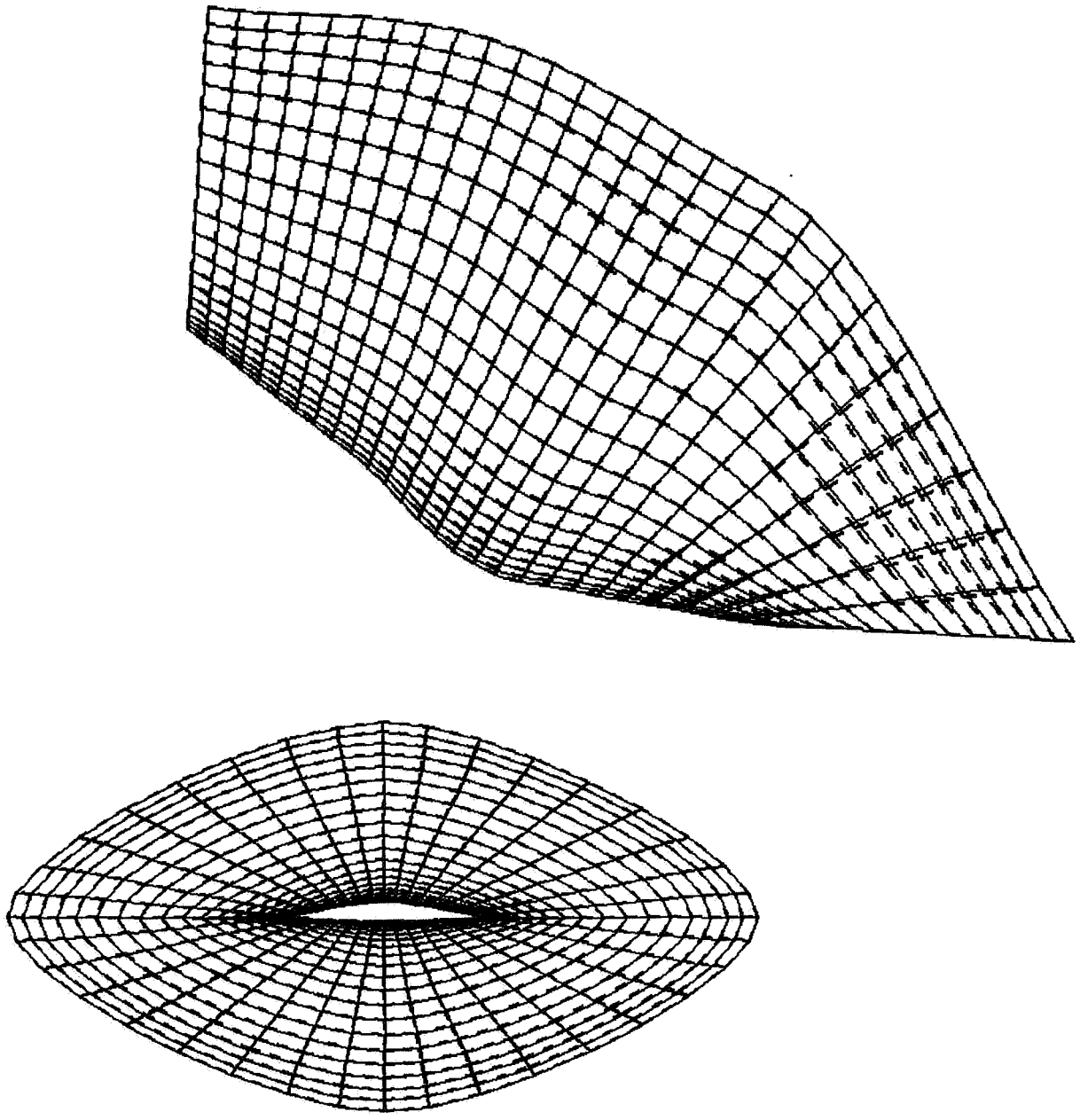


Figure 4. A comparison of 3D Bézier grids (solid lines) and transfinite interpolation grids (dash lines).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1992	Conference Publication

4. TITLE AND SUBTITLE
Software Systems for Surface Modeling and Grid Generation

5. FUNDING NUMBERS
505-90-53-02

6. AUTHOR(S)
Robert E. Smith, Editor
NASA Surface Modeling and Grid Generation Steering Committee

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23665-5225

8. PERFORMING ORGANIZATION REPORT NUMBER
L-17093

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CP-3143

11. SUPPLEMENTARY NOTES
Proceedings of the NASA Workshop on Software Systems for Surface Modeling and Grid Generation held at the Langley Research Center, April 28–30, 1992

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
It is a NASA objective to promote improvements in the capability and efficiency of computational fluid dynamics. Grid generation, the creation of a discrete representation of the solution domain, is an essential part of computational fluid dynamics. However, grid generation about complex boundaries requires sophisticated surface-model descriptions of the boundaries. The surface modeling and the associated computation of surface grids consume an extremely large percentage of the total time required for volume grid generation. Efficient and user-friendly software systems for surface modeling and grid generation are critical for computational fluid dynamics to reach its potential. As a part of meeting its objective, the NASA Workshop on Software Systems for Surface Modeling and Grid Generation is being held April 28–30, 1992. The papers and software demonstrations emanating from government laboratories, universities, and industries are being presented at the workshop. The proceedings of the workshop are presented herein. The papers represent the "state-of-the-art" in software systems for surface modeling and grid generation. Also, several papers describe improved techniques for grid generation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Surface modeling; Grid generation; Computer-aided design; Computational fluid dynamics	529
	16. PRICE CODE A23

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		

National Aeronautics and
Space Administration
Code JTT
Washington, D.C.
20546-0001

Official Business

Penalty for Private Use, $300

SPECIAL FOURTH-CLASS RATE
POSTAGE & FEES PAID
NASA
PERMIT No. G27

NASA

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return