Old Dominion University Research Foundation

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

GEOMETRIC MODELING FOR COMPUTER AIDED DESIGN

By

James L. Schwing, Principal Investigator

Progress Report
For the period ended January 1, 1993

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

Under
Research Grant NCC1-99
Lawrence F. Rowell, Technical Monitor
SSD-Vehicle Analysis Branch

April 1993

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

GEOMETRIC MODELING FOR COMPUTER AIDED DESIGN

By

James L. Schwing, Principal Investigator

Progress Report
For the period ended January 1, 1993

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

Under
Research Grant NCC1-99
Lawrence F. Rowell, Technical Monitor
SSD-Vehicle Analysis Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369

April 1993

Geometric Modeling for Computer Aided Design

Progress Report - January 1993

1. Introduction

Over the past several years, it has been the primary goal of this grant to design and implementation software to be used in the conceptual design of aerospace vehicles. The work carried out under this grant has been carried out jointly with members of the Vehicle Analysis Branch (VAB) of NASA Langley, Computer Sciences Corp. and Vigyan Corp. This has resulted in the development of several packages and design studies. Primary among these are1 the interactive geometric modeling tool, SMART, the Solid Modeling Aerospace Research Tool and the integration and execution tools provided by EASIE, the Environment for Application Software Integration and Execution. In addition, it is the purpose of the personnel of this grant to provide consultation in the areas of structural design and algorithm development, and software development and implementation, particularly in the areas of computer aided design, geometric surface representation and parallel algorithms.

During the last year, the investigators of the grant specifically proposed to consider the following areas.

- Provide general consulting on the use and development of aerospace structural analysis codes; in particular, design and implement a new Structures Module for SMART.
- Establish a database interface for POST that allows easier definition of data and helps perform data consistency checks for the model.
- Research the possibility of providing a general data integrity checker for EASIE.
- Build a prototype X-Window interface for EASIE.

- Develop methods for the creation of smoother, "faired" surfaces for SMART to allow the transfer of SMART geometry to structural grid generation programs and other analysis programs requiring surfaces to meet more exacting requirements.
- Develop algorithms for the efficient use of parallel and distributed computing.

This report refers to a number of Master's projects, conference presentations and journal papers. Copies of the Master's projects and an AIAA paper have been appended to the end of this report since they may be difficult to otherwise obtain. The other papers and conference articles appear as cited in the open literature and are therefore not included here.

2. Use and Development of Structural Codes

2.1 SMART Extensions

SMART provides conceptual designers with a rapid prototyping capability and additionally provides initial mass property analysis. In addition, SMART has a carefully engineered user interface that makes it easy to learn and use. Given the type of vehicle analysis that is conducted in VAB, it has become a priority to extend the capabilities of the SMART analysis support tools into the areas of aerodynamic and structural analysis. These enhancements have proven to be of interest to number of other groups at NASA, notably the HISAIR project. Recently, a new design team was formed with the purpose of soliciting requirements from all concerned engineering groups. The requirements were collected and synthesized by the design team and since have gone through a formal review process. The design team is presently completing design documents for these extensions to SMART.

A major effort of the work here centered on taking the approved system requirements documents and developing first implementation plans and continuing on to the coding and debugging phase. The personnel on this grant aided in the development of the implementation plans for the structural analysis portion of the codes including consulting on the general development of data transfer algorithms. Implementation of the actual code for the final portion of these system improvements has been given to programmers with Computer Sciences Corporation. A portion of this code, that used to develop interior structure for fuselages, was developed as a Master's project by Ms. S. Schwartz.

- "Surface Generation and Editing Operations Applied ato Structural Support of Aerospace Vehicle Fuselages", S. Schwartz, ODU Master's Project, Department of Computer Science, 1992.

2.2 General Structural Analysis Consulting

In addition, continuing research under this grant has been focused on the design and implementation of computer aided design tools to support conceptual level aerospace design. This has included the use of a number of finite element design and analysis codes involved in several design studies currently underway in the VAB.

Of these, the primary task has been the development of a finite element model of the primary structure of the air-breathing first stage of a two stage launch vehicle. The model includes a lifting-body configuration fuselage, a discrete wing and internal multi-bubble tanks. Development of this model involved converting sections from hardcopy with linear and quadratic scaling between the sections. At present the vehicle is changing and the analytic model needs to be changed to meet these changes.

Consultation on these and other structural codes and analysis have been provided through this grant by Mr. James Robinson. Some of the primary results of this work, resulted in the following publication.

- "Structural and Loads Analysis of a Two-Stage Fully Reusable Advanced Launch System", J.C. Robinson and D.O. Stanley, Fourth Symposium on Multidisciplinary Analysis and Optimizations, September 1992, AIAA paper # AIAA-92-4774.

2.3 Smooth Surfaces

Recently there has been an increasing interest in applying computational fluid dynamics (CFD) analysis to models at the conceptual level. Currently, none of the software systems which generate CFD grids and provide the corresponding analysis, provide tools for model generation.

As a first stage of this, personnel on this grant investigated the smoothing of data sets while reducing the size of the data. In essence algorithms were developed to find non-uniform rational bicublc B-spline approximations to given data sets. NURB's were chosen as the basic building blocks since they exhibit properties necessary to carry out the desired smoothness conditions needed for future surfaces. This work resulted in a Master's project by Ms. Carol Macri.

- "Implementation of an Algorithm for Data Reduction using Cubic-Rational B-Splines", C. Macri, ODU Master's Project, Department of Computer Science, 1992.

3. Enhancements for EASIE

3.1 Database Interface to POST

POST is an event driven program, the input to which falls into the above categories. POST is batch oriented taking input data from an ascii 'event' file. The flexibility of POST leads to a high degree of interdependence in the definition of data items. For example, when an alternate method of guidance is selected, a completely different set of input parameters must be specified. POST provides no tools for the definition of such input.

Current research has designed and implemented the prior work of Schwing and Grimm into a prototype which applies these techniques to the parameter variables of POST. User reaction has been extremely favorable. The personnel on this grant are currently engaged in the extension of the prototype to tabular variables, the last piece necessary for a phase one release of a new POST user interface. This work is currently being carried out by graduate students, Ms. Hima Gurla and Mr. Vasu Bokka. At the same time, work has also commenced on integrating data management directly with the database and stripping the current namelist data entry requirements of POST. This will allow the application of the data integrity checking previously described.

3.2 X-Windows and EASIE

EASIE provides a set of interactive utilities that simplify the task of building and executing computer aided design systems consisting of diverse, stand-alone, analysis codes. Resulting in a streamlining of the exchange of data between programs reducing errors and improving the efficiency. EASIE provides both a methodology and a collection of software tools to ease the task of coordinating engineering design and analysis codes.

Now that version one of EASIE has been released to the public the importance of the menu driven aspect of EASIE has been emphasized. Currently, this user interface is designed for simple ascii terminals and does not take advantage of recent advances in technology for presenting the user interface. On the forefront of these advances is the windowing system for the Athena project at MIT, X-Windows. Most of the software in this system is in the public domain and hardware in the form of X-servers and X-terminals is rapidly becoming available. To do some crystal gazing, it would seem that this combination of public domain software and low-cost hardware will lead to the next revolution of the user interface.

The research carried out in this area resulted in two master's projects. The effort was divided since EASIE can operate in two highly different modes. These are the complete command environment, CCE and the application derived environment. The work was completed by Ms. Chia-Lin Tsai and Ms. Ya-Chen Kao respectively.

- "User Interface Design for EASIE", C-L. Tsai, ODU Master's Project, Department of Computer Science, 1992.
- "Application Driven Interface Generation for EASIE", Y-C Kao, ODU Master's Project, Department of Computer Science, 1992.

4. Parallel Algorithms

It has become clear that much of the future improvements in computing power will arise in the use of parallel and/or distributed computing environments. Indeed, this can be seen in the new IRIS computers that have been brought in to support the VAB analysis and design programs. They are all multi-processor machines. While these machines can and do provide a certain amount of automatic algorithm adjustment to take advantage of this environment, true efficient use of any parallel or distributed environment requires careful investigation of the algorithms being developed. Algorithms initially developed for sequential single processor machines may not perform anywhere near optimally under automated conversion.

It is the belief of the principal investigators of this grant that the next major impact on the development of analysis programs will come via the proper utilization of parallel and distributed processing. Further, this can only occur if the proper ground work is developed. The personnel on this grant have been extremely productive in providing both the basis for understanding algorithm development on a number of exciting new architectures and also in a number of application areas that will have direct impact on research being conducted by Mr. Joe Rehder of VAB. This work is contained in the following publications.

- Optimal Parallel Algorithms for Problems Modeled by a Family of Intervals, (S. Olariu, J. Schwing and J. Zhang), *IEEE Transactions of Parallel and Distributed Systems*, v.3 no. 3, (1992), 364 - 374. A preliminary version was published in *Proc. 28-th Annual Allerton Conf. on Communication, Control, and Computing*, 1990, 282-291.
- On the Power of Two-Dimensional Processor Arrays with a Reconfigurable Bus System, (S. Olariu, J. Schwing and J. Zhang), *Parallel Processing Letters*, **1**, (1992) 29-34.
- Optimal Parallel Encoding and Decoding Algorithms for Trees, (S. Olariu, J. Schwing and J. Zhang), *International Journal of Foundations of Computer Science*, v. **3** no. **1**,

(1992), 1 - 10. A preliminary version of this paper appeared in *Proc. 1991 ACM Computer Science Conference*, San Antonio, Texas, March 1991, 1 - 10.

- Integer Problems on Reconfigurable Meshes, with Applications, (S. Olariu, J. Schwing and J. Zhang), *Journal of Computer and Software Engineering*, accepted for publication. A preliminary version has appeared in *Proceedings of the 29th Annual Allerton Conference on Communications, Control, and Computing*, 821-830, 1991.
- A Constant-time Channel Assignment Algorithm for Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *BIT*, v. 32, (1992) 586 - 597.
- Fast Computer Vision Algorithms on Reconfigurable Meshes, (S Olariu, J. Schwing and J. Zhang), *Image and Vision Computing Journal*, **v.10 no. 9**, (1992) 610 - 616. A preliminary version of this work has appeared in *Proceeding of the 6th International Parallel Processing Symposium*, Beverly Hills, 1992.
- Selection on Meshes with Multiple Broadcasting, (D. Bhagavathi, P. Looges, S. Olariu, J. Schwing, and J. Zhang), *BIT*, accepted for publication.
- Simulating Enhanced Meshes with Applications, (R. Lin, S. Olariu, J. Schwing, and J. Zhang), *Parallel Processing Letters*, accepted for publication.
- Applications of Reconfigurable Meshes to Constant-time Computations, (S. Olariu, J. Schwing and J. Zhang), *Parallel Computing*, accepted for publication. A preliminary version of this paper appeared as "Constant Time Integer Sorting on an nxn Reconfigurable Mesh" in *Proc. of the International Phoenix Conf. on Computers and Communications*, Scottsdale, Arizona, 1992, 480-484.
- A Simple Selection Algorithm for Reconfigurable Meshes, (S. Olariu, J. Schwing, W. Shen, L. Wilson, and J. Zhang), *Parallel Algorithms and Applications*, accepted for publication. A preliminary version of this work appeared in *Proc ISMM Conference on Parallel and Distributed Systems*, Pittsburgh, October 1992, 257 - 261.
- Fast Mid-level Vision Algorithms on Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *Parallel Computing: From Theory to Sound Practice, Proceedings of EWPC'92*, IOS Press, 1992, 188-191.
- Sorting in O(1) time on a Reconfigurable Mesh of Size nxn, (R. Lin, S.Olariu, J. Schwing and J. Zhang), *Parallel Computing: From Theory to Sound Practice, Proceedings of EWPC'92*, Plenary Address, IOS Press, 1992, 16-27.
- A Fast Selection Algorithm on Meshes with Multiple Broadcasting, (D. Bhagavathi, P. Looges, S. Olariu, J. Schwing, and J. Zhang) *Proc. International Conference on Parallel Processing*, St. Charles, Illinois, 1992, p. III-10 - III-17.
- Fast Component Labeling on Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *Computing and Information - Proc. International Conference on Computing and Information*, Toronto, 1992, 121 - 124.
- Efficient Image Processing Algorithms for Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *Proc. of Vision Interface, 1992*, Vancouver, British Columbia, May 1992.
- Computing the Hough Transform on Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *Proc. of Vision Interface, 1992*, Vancouver, British Columbia, May 1992.
- Time-Optimal Sorting and Applications on nxn Enhanced Meshes, (S. Olariu, J. Schwing and J. Zhang), *Proc. IEEE Internat. Conf. on Computer Systems and Software Engineering*, Comp Euro '92, The Hague, May 1992, 250 - 255.

- Interval-Related Problems on Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *Proc. ASAP '92*, Berkeley, August 1992, 445 - 455.
- Efficient Image Computations on Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *Proc. of CONPAR '92*, Lyon, France, September 1992.
- Convexity Problems on Meshes with Multiple Broadcasting, (S. Olariu, J. Schwing and J. Zhang), *Proc. 4th Annual Canadian Computational Geometry Conference*, St. John's, August 1992, 365 - 370.
- Convex Polygon Problems on Reconfigurable Meshes, (S. Olariu, J. Schwing and J. Zhang), *SPIE Conference on Vision Geometry*, Boston, November 1992.
- An Optimal Parallel vertex Cover Algorithm for Cographs, (R. Lin, S. Olariu, J. Schwing, W. Shen, and J. Zhang), *Proc ISCIS Conference*, Antalya, Turkey, November 1992, 49 - 55.
- Geometric Problems on Meshes with Multiple Broadcasting, (S. Olariu, J. Schwing and J. Zhang), *Proc ISCIS Conference*, Antalya, Turkey, November 1992, 41 - 47.

Appendix

Master's Project Reports and AIAA Conference Procedings

Surface Generation and Editing Operations Applied to Structural Support of Aerospace Vehicle Fuselages [1]

Susan K. Schwartz

A Master's Project
submitted to the
Computer Science Department
of
Old Dominion University
in partial satisfaction of
the requirements for the degree of
Master of Science

Project Advisor: Dr. James L. Schwing,
Associate Professor of Computer Science

April, 1992

[1]This work was supported under NASA grant NCC1-99.

Abstract

SMART, Solid Modeling Aerospace Research Tool, is the Vehicle Analysis Branch of NASA Langley Research Center's computer-aided design tool used in aerospace vehicle design. Modeling of structural components using SMART includes the representation of the transverse or cross-wise elements of a vehicle's fuselage, ringframes and bulkheads. Ringframes are placed along a vehicle's fuselage to provide structural support and maintain the shape of the fuselage. Bulkheads are also used to maintain shape but are placed at locations where substantial structural support is required.

Given a Bézier curve representation of a cross-sectional cut through a vehicle's fuselage and/or an interior tank, this project produces a first-guess Bézier patch representation of a ringframe or bulkhead at the cross-sectional position. The grid produced is later used in the structural analysis of the vehicle. The graphical display of the generated patches allows the user to edit patch control points in real time. Constraints considered in the patch generation include maintaining "square-like" patches and placement of longitudinal, or lengthwise along the fuselage, structural elements called longerons.

Contents

List of Figures

1 Introduction

"A model is a representation of some (not necessarily all) features of a concrete or abstract entity. The purpose of a model or an entity is to allow people to visualize and understand the structure or behavior of the entity, and to provide a convenient vehicle for 'experimentation' with and prediction of the effects of inputs or changes to the model [FOLEY, pp. 286–7]." In many instances, the model is the only means in which analysis can be performed to determine feasibility of an idea. Costs of creating an actual entity or the testing facility for a particular entity may be prohibitive and a model provides the simulation of the entity for experimentation and learning about a proposed system.

The cost of memory and computing time has decreased drastically in the past two decades and made the computer one of the most viable tools for modeling. In particular, graphics-based modeling tools are now used "to create and edit the model, to obtain values for its parameters, and to visualize its behavior and structure [FOLEY, p. 287]."

In the mid 1970's, the Vehicle Analysis Branch, VAB, of NASA Langley Research Center, LaRC, began development of its own solid modeling system. Numerous commercially produced systems were evaluated and determined not to meet the needs of the VAB. Thus, SMART, or Solid Modeling Aerospace Research Tool, was begun in the 1980's to provide the VAB with its own computer-aided design tool for aerospace vehicle design.

A primary method of modeling used by aerospace and structural engineers is based on the ability to create a "nice" grid on a surface. Finite element analysis and computational fluid dynamics both rely on known values at points relatively close to one another to predict values of quantities like stuctural stress at other points. Currently, the difficulties in producing suitable grids for these analyses slows the design process. Manual means of producing the grids are unsuitable and automating the process is the desired method.

The goal of this project has been to automate the Bézier patch generation of fuselage bulkheads and ringframes used in the structural analysis of aerospace vehicles. Sections two through five of this paper provide insight into the basics of SMART, aircraft structural design, the finite element analysis process, and the geometric representations used in the modeling process. Section six presents the algorithms developed to generate the desired patches. Snapshots of the SMART display showing the implementation of the algorithms are provided as Appendix A. Copies of the SMART structures requirements document and source code are provided as Appendices B and C, respectively.

2 Capabilities of SMART

SMART, written in the C programming language, was developed for use on the Silicon Graphics IRIS workstation, a computer which features custom graphics hardware and the UNIX operating system. The initial modeling requirements of the software included:

- ability to generate accurate 3-dimensional geometric descriptions of complex vehicle shapes quickly and easily;
- facilitate easy manipulation of the vehicle components using a hierarchial component grouping scheme;
- provide data from a single geometric representation to a variety of analysis programs; and
- real-time interaction with the user [MCMIL, p. 1].

The user interface of SMART was designed to accomodate "novice, occasional, and experienced users [MCMIL, p. 2]." The main features of the display, shown in Figure 1, are two large viewing windows or viewports, a small textport area, two horizontal main menus, and an area for displaying a variety of menus and slider bars pertinent to the given evolution. Most user input is accomplished by positioning the mouse over the desired menu, bar, or plotted geometric figure in the viewport and pressing an appropriate button.

Objects may be created from basic primitive shapes, that is, SMART-facilitated automatic generation of vehicle components, or by "free-hand" rendering with the mouse over the viewport. In particular, SMART "has an extensive capability for creating and modifying cross-section capability to create completely arbitrary shapes [MCMIL, p. 3]." The cross-sections are represented by either "Bézier cubic curves or a series of points connected

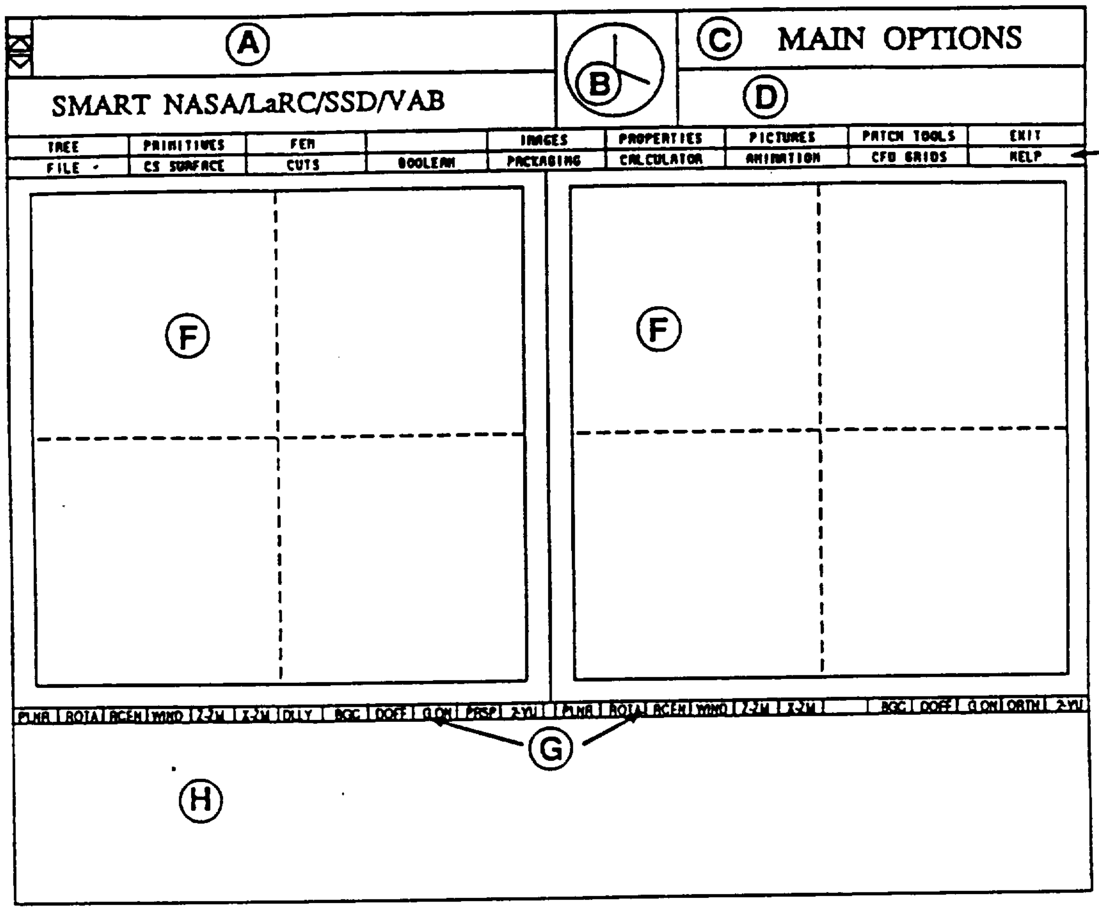


Figure 2-1
The Layout of the SMART Screen

A. Textport	**E. Mode Menubar**
B. Clock	**F. View Windows**
C. Function Name Area	**G. View Option Menubars**
D. Information Display Area	**H. Menu Display Area**

Figure 1: The Layout of the SMART Screen [SMART, p. 2-1]

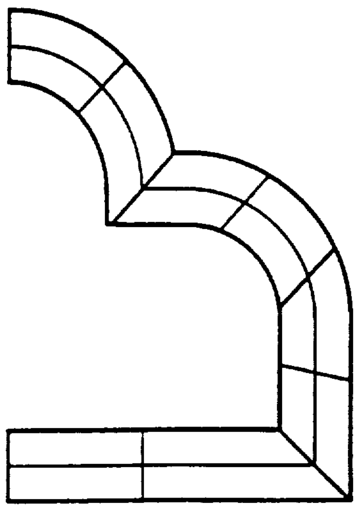


Figure 2: Ringframe from an offset curve [REHDER, p. 3]

by straight lines, referred to as Cartesian cross-sections [MCMIL, p. 3]." A discussion of Bézier curves is presented in Section 5 of this paper.

Once a geometric component is created, the component is accessed for a variety of processes. A capability, currently being developed, is the consolidation of the model generation process for structural analysis. New requirements specifications have been written and this project represents the fulfillment of many of the ringframe and bulkhead generation and longeron placement requirements. See [SOFT, pp. 24-28], provided in Appendix B of this paper, for the pertinent portion of the requirements document. [REHDER, pp. 3–4] describes the technique applied to constructing the model of these components. A planar surface is generated between two curves: one of the curves is formed by the outer surface of the fuselage; the other is a scaled offset from the fuselage curve, creating a ringframe, as in Figure 2, or a separate curve representing the cross-section of a tank interior to the fuselage, creating a bulkhead, as in Figure 3.

The planar surface, represented by Bézier bicubic patches, may be stored in several different types of files. In particular, SMART has the capability of writing an ASCII text file of the patch data for an entire vehicle in the format known as a "neutral file." This file may then be "read" by the PATRAN structural analysis program [PATRAN] and this geometry is used as

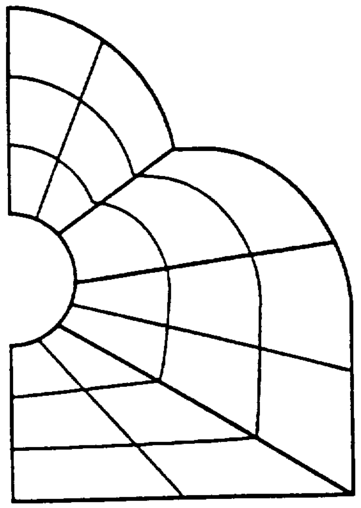


Figure 3: Bulkhead between fuselage and internal tank [REHDER, p. 4]

a template to create a suitable grid and then perform finite element analysis on that grid for various pressure and stress loadings.

3 Aircraft Structural Design

3.1 Design Considerations

The design of an aircraft requires the combined efforts of both the aerodynamics engineer and the structural engineer. The aerodynamicist considers the vehicle as an aerodynamic shape and analyzes the reaction of the surrounding air to the presence of the "envelope of specially shaped airframe surfaces [STIN66, p. 190]." This envelope must distribute the loads to the surrounding air. The airframe must also protect the items within, such as the payload, fuel, and engines. Given an accurate distribution of the air-loads of the vehicle, the structural engineer's job is to produce a sound structure. Because there is great difficulty in accurately predicting these loads at each point on the structure's surface, the structural engineer considers the "most critical design cases—which often run into thousands— arising from the various combinations of speed, attitude and weight throughout the flight [STIN66, p. 190]."

"Structural design affects the achievable flight envelope, stability and control, the operational role and the development potential of an aeroplane [STIN66, p. 192]." There are many considerations for structural design and each deserves to be fully explored. However, full explanations are beyond the scope of this paper, and each will be given at most, a cursory explanation:

- The outer skin must remain reasonably wrinkle-free and smooth in 1-g flight, which is different from an unloaded vehicle on the ground.

- The fabrication material must have a high strength-to-weight ratio, particularly at high temperatures, and high specific stiffness.

- The study of loads on a material is of major concern and both the way in which the load is applied and the area over which it is applied must be considered. When a material is loaded in a particular way, it is said to be stressed. There are three types of stress: tensile, compressive or bearing, and shear. Tensile stress is caused by tension across a cross-sectional element. Compressive stress is the reverse of tensile stress. Shear stress occurs tangential to the surface. The material's shape multidimensionally changes when it is stressed. Shear strain is defined as the angular displacement caused by shear stress. Similar strain definitions apply to tensile and compressive stress. Although a simplistic approach, it should be noted that stress causes strain and strain causes stress.

- Heat is also a consideration. The boundary layer of air surrounding a high-speed aircraft becomes heated and raises the temperature of the skin of the aircraft. External radiant heat may also be a factor.

- The elasticity/plasticity of a material is an important factor. If the strain caused by a stress completely disappears when the stress is removed, the material is said to be wholly elastic. If the strain has not disappeared, the material is said to have a permanent "set" and plasticity has occurred. "A structure is designed so that the working range of any component does not exceed its elastic limit. It is now possible to study stress-patterns established in structural components by various applied loads.... A useful general law, known as Hooke's Law, states that within elastic limits of a material the strain produced is proportional to the stress producing it [STIN66, p. 197]."

- Bending and torsion or twisting must also be accounted for. Bending takes place when a load is applied to a point on the flexural axis of a structural member and the reaction is at another point on the axis. Torsion will also occur if the reaction is offset from the flexural axis.

- Fatigue is also studied. It occurs when repeated stresses, each much lower than maximum tensile stress allowable, cause the cracking of structural members.

None of these items can be taken in isolation and generally combinations are considered simultaneously. "An important aid in structural analysis is the Principle of Superposition: that the total strain caused by a load-system may be considered as the sum of the individual strains caused by the various load components, taken in isolation [STIN66, p. 197]."

"The analysis of stress and strain in advanced aircraft structures has forced the development of very elegant and complicated mathematical techniques. The structural engineer must relate the effects of weights, aerodynamic inputs, elastic responses and stress distributions throughout the structure as one whole, for a wide variety of different shapes. Fortunately, the grid-like construction allows accurate analyses to be made and translated into mathematical statements that can be handled by computers [STIN66, p. 213]."

3.2 The Actual Design

The airplane has three basic parts, the fuselage, the wings, and the tail. This paper will only address the fuselage, parts of which are the focus for this project.

"The fuselage is the body to which the wings and the tail unit of an airplane are attached and which provides space for the crew, passengers, cargo, controls, and other items, depending upon the size and design of the airplane. It should have the smallest streamline form consistent with desired capacity and aerodynamic qualities of the airplane ... The main structure of a spacecraft or missile may be called a fuselage but is more commonly called the body or tank [MCKIN, p. 140]."

The modern aircraft's fuselage is of a semi-monocoque construction, as seen in Figures 4 and 5 . This means that the fuselage has a framework

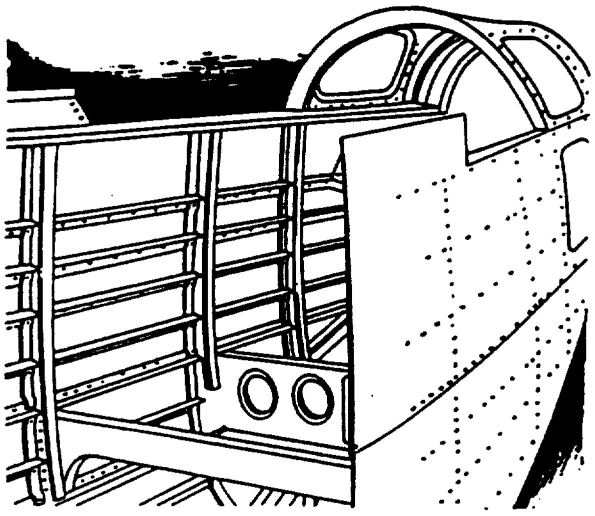


Figure 4: Semimonocoque construction [MCKIN, p. 144]

which supports an external skin which must withstand most of the stresses placed on the fuselage. The framework consists of several types of structural elements. The vertical or transverse elements of the fuselage support are called bulkheads, frames, and formers or rings. A bulkhead is a substantially constructed cross-section cutting across a fuselage, perpendicular to the fuselage's longitudinal beam, as in Figure 6. A bulkhead is placed at points of concentrated loads, and helps to distribute the loads over the skin and allows little radial expansion. There may be cut-out areas for doorways and holes, but doors and plates are used to maintain the structural requirement, as seen in Figure 7.

A frame serves primarily to maintain the shape of the body and has the outline of the cross-section of the vehicle, which can be seen in Figure 8. The loads at the frames are smaller and construction of the frames can be lighter than that of the bulkheads. Formers or rings have the same outline as the frame but are lighter and are used to maintain a uniform shape of the skin. This paper refers to all of these as ringframes.

The longitudinal components are longerons and stringers. They are supported by the bulkheads and frames and support the outer skin to prevent bulging due to severe stresses. They also are used to carry the axial loads

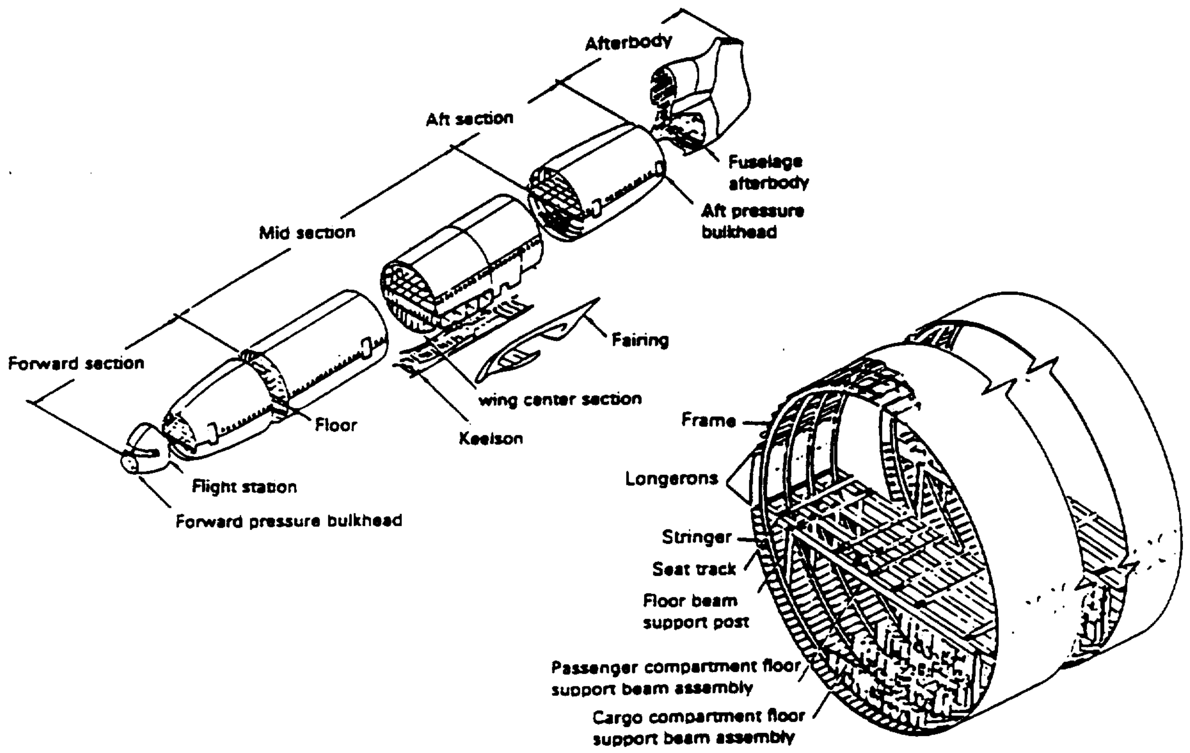


Figure 5: Typical semi-monocoque stiffened shell—L-1011 [NIU, p. 376]

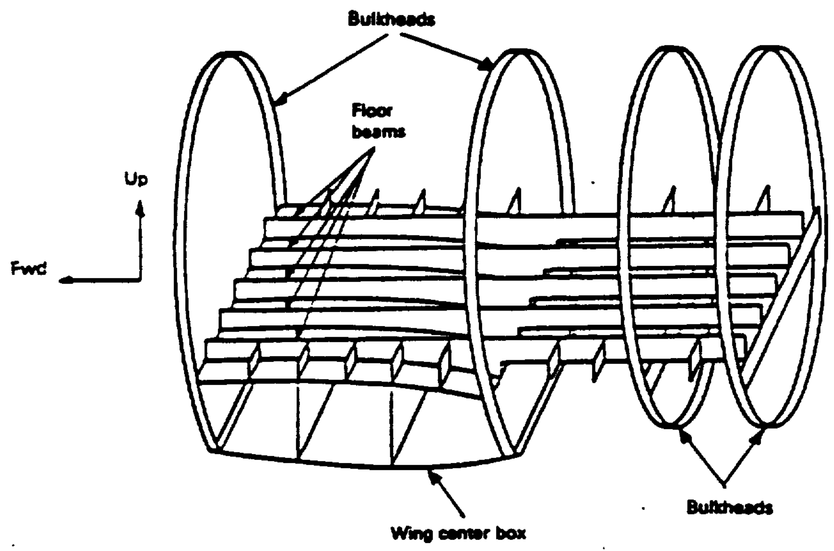


Figure 6: Typical transport fuselage center section floor beams arrangement. [NIU, p. 396]

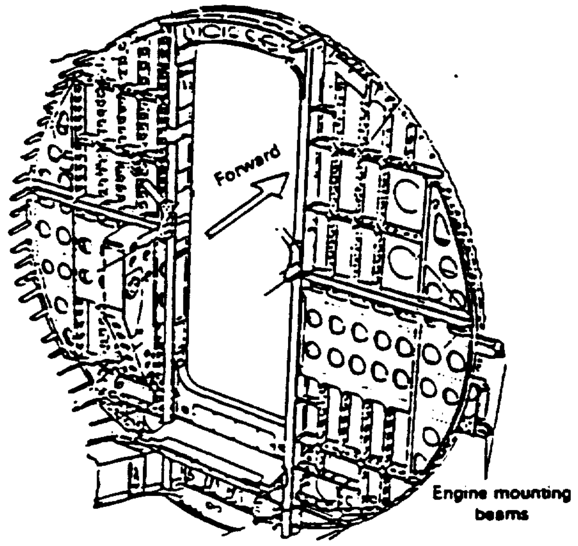


Figure 7: Typical pressure flat bulkhead [NIU, p. 398]

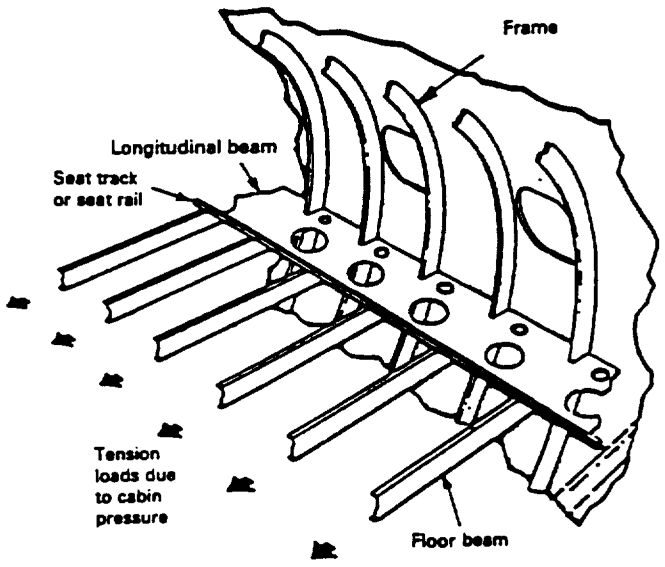


Figure 8: Fail-safe design by using longitudinal beam along side of fuselage. [NIU,p. 391]

caused by bending. Longerons are especially designed to take the end loads fore and aft of the vehicle and run the length of the fuselage. Stringers are shorter and of lighter construction. See Figure 5.

The external skin is formed from metal sheets which are attached to the frames and bulkheads by riveting or welding. It carries the loads of sheer stress and cabin pressure. Figure 9 shows the combined features mentioned above.

The semi-monocoque structure is considered to be "very efficient, i.e., it has a high strength to weight ratio, and it is well suited for unusual load combinations and locations. It has design flexibility and can withstand local failure without total failure through load redistribution [NIU, p. 377]."

4 Overview of Finite Element Analysis

Finite element analysis is defined to be a "group of numerical methods for approximating the governing equations of any continuous system [BARAN, p. 1]". Originally developed for the study of stresses in complex airframe

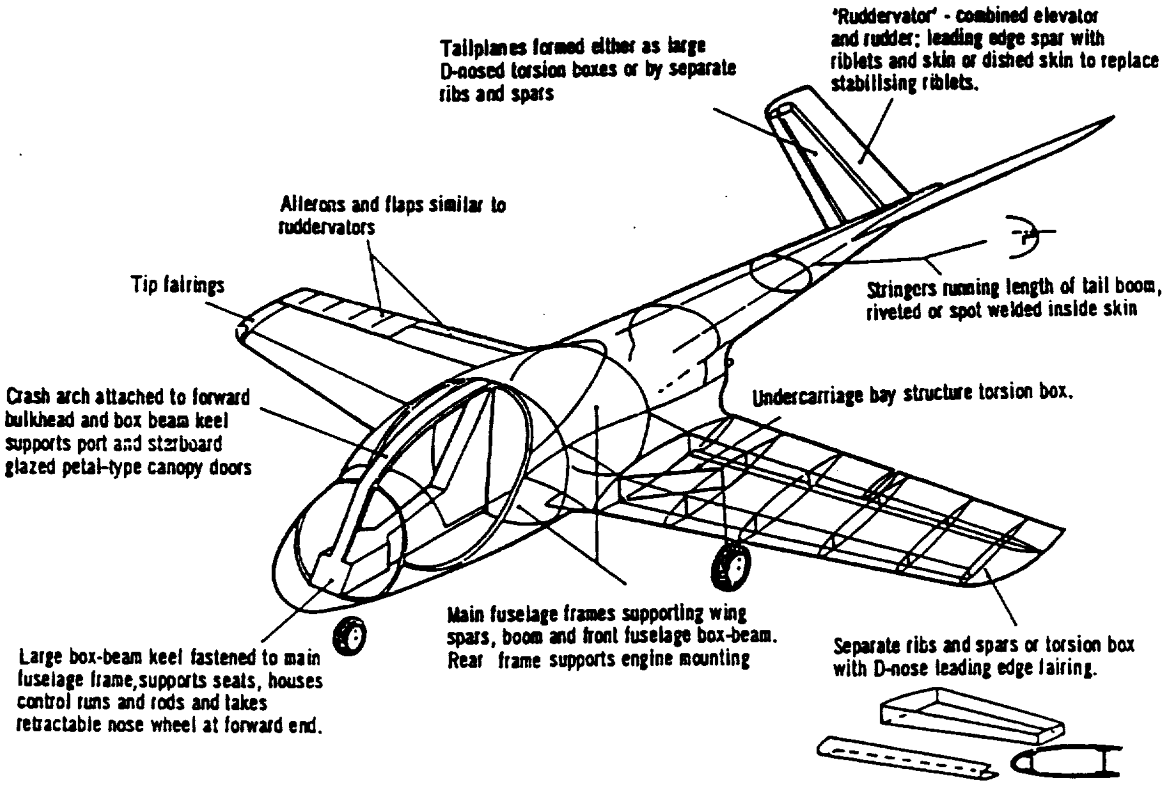

'Ruddervator' - combined elevator and rudder; leading edge spar with riblets and skin or dished skin to replace stabilising riblets.
Tailplanes formed either as large D-nosed torsion boxes or by separate ribs and spars
Ailerons and flaps similar to ruddervators
Tip fairings
Stringers running length of tail boom, riveted or spot welded inside skin
Crash arch attached to forward bulkhead and box beam keel supports port and starboard glazed petal-type canopy doors
Undercarriage bay structure torsion box.
Main fuselage frames supporting wing spars, boom and front fuselage box-beam. Rear frame supports engine mounting
Separate ribs and spars or torsion box with D-nose leading edge fairing.
Large box-beam keel fastened to main fuselage frame, supports seats, houses control runs and rods and takes retractable nose wheel at forward end.

Figure 9: Sketch of main details of aeroplane structure [STIN66, p. 205]

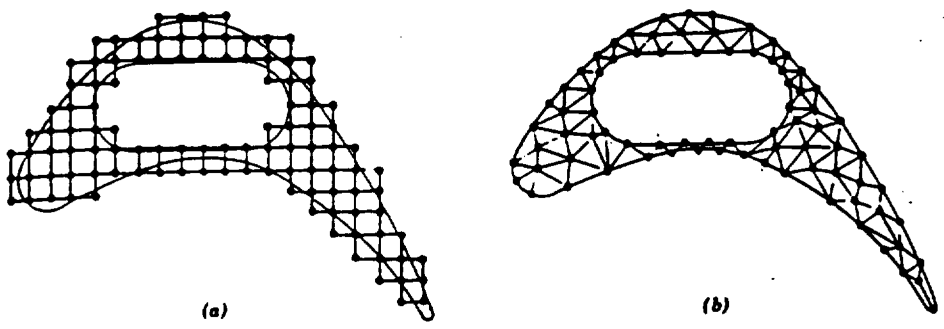


Figure 10: Finite difference and finite element discretizations of a turbine blade profile. (a) Typical finite difference model. (b) Typical finite element model. [HUEB, p. 5]

structures [HUEB, p. 3], the finite element method today is also used in a variety of engineering disciplines. It is particularly effective for problems with complex geometries. Until recently, finite element analysis was restricted to expensive mainframe computers, but the significant declines in hardware and processing costs have made this process available to virtually all engineers and scientists. Civil and aerospace engineers remain the most frequent users of this method.

The difficulty of a continuous system or structure is the infinitely many values of the unknown quantity being evaluated at each point of the structure. The objective of finite element analysis is to approximate the governing differential equation of the system or structure at selected points with a sufficient degree of accuracy. A mathematical model of the physical system is created. The points or nodes, when connected, define the elements of the model. This process of creating nodes and elements is called discretization and is illustrated in Figure 10. Simplifying assumptions are made to create approximating functions, from the original differential equations, which are then applied to the specified nodes of the model. Solutions are created for individual elements and then combined to represent a solution for the entire problem. The size and number of elements and simplifying assumptions determine the accuaracy of the analysis.

4.1 Steps in the Finite Element Method

Finite element analysis can be performed in a sequence of five steps, each of which has its own difficulties and time requirements. They are summarized as follows:

1. Perform the discretization. Dividing the physical structure into elements is the most important phase because this will greatly affect the accuracy of the analysis. Elements may take various shapes depending on the nature of the problem. This is discussed in greater detail in the next section.

2. Define the geometric properties of each element and any material properties and boundary or loading conditions pertinent to the analysis.

3. Formulate interpolation equations for each element. These are often polynomial in nature because of the ease in integrating and differentiating them. The interpolation functions in these equations give "an analytical expression for the displacement at any point inside the element" [BARAN, p. 4]. The value of the equation at any point in an element is a function of the nodes bounding the element. See Figure 11.

4. Assemble the system equations, accounting for properties outlined in Step 2 above, and solve the equations.

5. Make additional calculations, if necessary. The solution of the system of equations may be used to calculate other parameters. For example, in structural analysis, nodal values represent body displacements. These values are then used to calculate strains and stresses in the elements.

4.2 Creating the Mesh

There are two basic categories of planar elements: line and area. Beam and spring elements are examples of line elements. Beam elements are used in a variety of engineering problems to represent parts whose lengths are much greater than the cross-sectional depth or width. Area elements include flat plate or shell elements. The plate elements have a thickness much smaller than their other dimensions and are usually represented by three or four

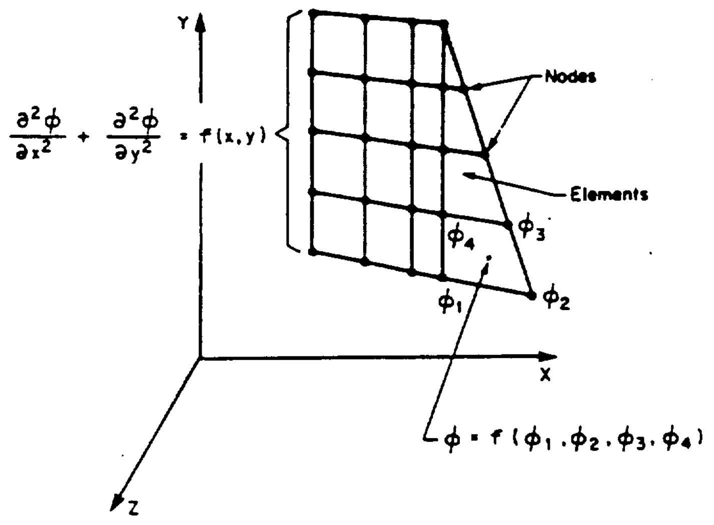

Y
$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = f(x,y)$
Nodes
Elements
ϕ_3
ϕ_4
ϕ_1
ϕ_2
X
$\phi = f(\phi_1, \phi_2, \phi_3, \phi_4)$
Z

Figure 11: An arbitrary shape divided into nodes and elements. The shape is governed by the partial differential equation shown. The value of this equation at any point in an element is a function of the values of the nodes Φ_i bounding the element. [BARAN, p. 3]

nodes. Solid or volume elements are a third type of element used to account for parts whose thickness is significant compared to other dimensions.

The model being created is an idealization of the actual physical structure being analyzed. By understanding the physical problem, the regions of the structure most likely to be stressed are determined. A coarse mesh is created, placing nodes at stress, support, and load points. Finer meshes can be created from this initial mesh, if necessary. Huebner quotes John M.Biggs as saying that it is a "waste of time to employ methods having precision greater than that of the input of the analysis [HUEB, p. 88]."

The shape and element pattern of the finite element model is determined by the location of the nodes. Other significant locations of nodes include structure corners and discontinuities. The model should closely approximate the shape of the actual structure. The size of the model can be reduced by accounting for the structure's symmetry, as in Figure 12. Establishing a coordinate system with an origin on an axis of symmetry allows easier definition of nodes and elements.

Ultimately, the choice of nodes and elements depends on the type of

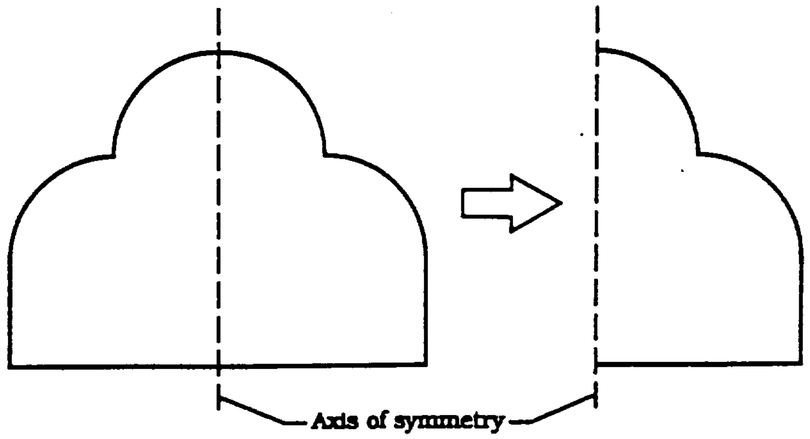


Figure 12: Model reduction due to structure symmetry

finite element analysis being performed and the accuracy required. Huebner suggests the following as rules for finite element modeling [HUEB, pp. 94-99]:

- If the problem involves concentrated loads and/or geometric discontinuities, minimum dimensions and areas requiring a refined mesh should be determined using St. Venant's principle. This states that "localized loads or geometric discontinuities cause stresses and strains only in the immediate vicinity of the load or discontinuity [HUEB, p. 99]."
- Stress analysis requires a more refined mesh than displacement analysis.
- Nodes should be placed at supports, load points, and other locations where information, such as displacements or temperatures, is required.
- Uniform mesh spacing should be used, if possible. If it is necessary to transition from coarse to fine meshes, the dimensions of adjacent elements should not differ by more than a factor of two. The transition should be made across a series of elements.
- When using plate or axisymmetric elements, quadrilaterals are the preferred shape because they are more accurate than triangles. Triangular elements should be used only when required by the geometry or for transitions.

- The aspect, or length-to-width, ratio of triangular or quadrilateral elements should be as close to unity as possible. Aspect ratios as large as 5.0 are permissible, but below 3.0 is preferrable.

- In triangular and quadrilateral elements, no extremely obtuse or acute angles should be used. The optimum is the equilateral triangle, where all angles are 60 degrees, or right angles in the quadrilateral, but deviations of up to 30 degrees is permissible.

- Curved surfaces should be modeled with flat elements whose nodes are all in one plane. The angle subtended by the surface and the plane should be less than 15 degrees.

- Poisson's ratio, should be less than 0.5. An elastic material elongates in the direction of an applied tension while its cross-section contracts perpendicular to the tension direction. During simple compression, the material contracts in the tension direction and expands perpendicularly to the tension. Poisson's ratio is the ratio of the resultant perpendicular strains to the parallel strains. Most metallic materials have a value of 0.25-0.3 and an assumed value of 0.3 is used. It is also assumed that Poisson's ratio approaches 0.5 as the stresses reach a maximum for the material [NILES, pp. 151-152].

- Lengths and areas of line and area elements must be non-zero. Values of zero may produce unpredictable results.

- Elements should not extend across discontinuities or changes in thickness. This tends to cause numerical errors and inaccurate results. Additional nodes and smaller elements should be used.

- It is assumed that flat plate elements have no in-plane rotational stiffness. If in-plane twisting is allowed, plate elements do not accurately represent the model's flat plates.

5 Geometric Representation

Vehicles are drawn using curves and surfaces which approximate the desired shape of the vehicle. There are numerous ways to represent such curves

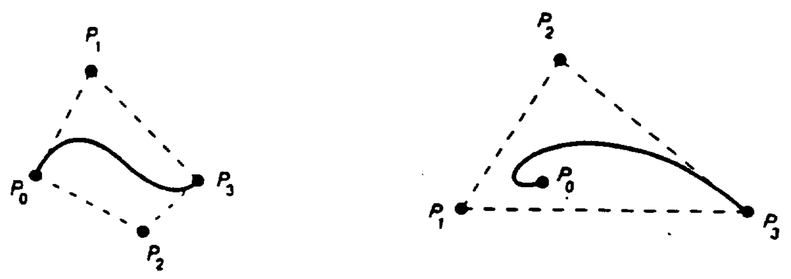


Figure 13: Two Bézier curves and their control points [FOLEY, p. 488]

and surfaces. As surface representations are a generalization of curve representation, this section will first consider working with curves. Often, a parameterization of curves, where each coordinate, x, y, and z, is a function of a parameter, t, i.e., $x = x(t), y = y(t), z = z(t)$, is used to avoid problems occuring with explicit and implicit equations used to describe geometric figures. For specifics, see [FOLEY, p. 478].

The predominant method used in SMART is the Bézier form of the parametric cubic polynomial curve segment. This consists of 4 points, P_0, P_1, P_2, and P_3 where P_0 and P_3 are endpoints of the curve segment and P_1 and P_2 are additional control points. Generally not on the curve segment, points P_1 and P_2 indirectly specify the tangent vectors to the curve at P_0 and P_3. Specifically, the direction of the tangent vector at P_0 is determined by P_0P_1 and the direction of the tangent vector at P_3 is determined by P_3P_2. See Figure 13.

To determine a point P on the curve segment, the parameterization of the domain is set up so that at parameter $t = 0$, $P = P_0$, and at $t = 1$, $P = P_3$. The weighting factors for each point, known as the Bernstein polynomials, are:

$$\begin{aligned} B_0^3(t) &= (1-t)^3 \\ B_1^3(t) &= 3t(1-t)^2 \\ B_2^3(t) &= 3t^2(1-t) \\ B_3^3(t) &= t^3 \end{aligned}$$

The resultant equation to evaluate P is:

$$P(t) = \sum_{j=0}^{3} P_j \times B_j^3(t)$$

Note that $P(t)$ is guaranteed to be cubic in t because it is a linear combination of cubic polynomials. The sum is computed for each coordinate, x and y in 2-D; x, y, and z in 3-D).

There are several advantages inherent to this representation:

- Cubic curves do not "wiggle" as much as higher order polynomials and give a relatively smooth approximation of the desired shape. Note that a cubic curve is the lowest degree polynomial to interpolate to four requirements: the two endpoints and the specified derivatives at each endpoint [FOLEY].

- The resultant curve segment is contained by the convex hull of its representative points. This guarantees that the curve segment is planar.

- Calculation of the derivative at any point on the curve segment, most notably at the endpoints, is easy. For a given t, $P'(t)$ is calculated as the linear combination of the derivatives of the Bernstein polynomials.

- The storage requirements for a curve segment are minimal—the four points and possibly, information about slope continuity with adjoining segments.

- Another way to compute $P(t)$ involves successive linear interpolations of pairs of the given four points. This linearity allows the Bézier representation to inherit the property of affine invariance. That is, when applying an affine transformation, scaling, rotation, shearing, or translation, to a Bézier curve, the result is the same whether the transformation is applied to the original four points, followed by the curve generation, or if the curve is generated from the points, followed by the transformation. Therefore, these viewing transformations need only be applied to the four control points of the segment, which minimizes computation time.

To represent a given shape, successive Bézier curves are placed end-to-end. Continuity of segments is guaranteed if P_3 of one curve is set to P_0 of the next curve. If slope continuity from one segment to the next is required, then P_2 and P_3 of the first curve and P_0 and P_1 of the second curve must remain collinear. Moving P_2 or P_1 "controls" the slope at the endpoint(s).

Sometimes a Bézier curve needs to be split into two pieces. If the parameter t is normally defined over the interval $[0,1]$, a value of t in this interval can be specified to represent a certain percentage c along the curve, or the place where the curve should be split. In essence, the first piece of the curve would be exactly the original curve over the parameter's interval $[0,c]$ and the second curve is the piece corresponding to $[c,1]$. [FOLEY, pp. 507–510] and [FARIN, pp. 75-77] describe this process using the geometric construction technique developed by F. de Casteljau in 1959. As in Figure 14, "the point on the curve for a parameter value of t is found by drawing the construction line L_2H so that it divides P_1P_2 and P_2P_3 in the ratio of $t:(1-t)$, HR_3 so that it similarly divides P_2P_3 and P_3P_4 and L_3R_2 to likewise divide L_2H and HR_3. The point L_4 (which is also R_1) divides L_3R_2 by the same ratio and gives the point $Q(t)$ [FOLEY, p. 508]," the value of the Bézier curve at parameter t. The points L_1, L_2, L_3, and L_4 are the control points for the first curve and R_1, R_2, R_3, and R_4 are for the second curve.

Bézier representation can be extended to surfaces. Bézier bicubic patches are determined by sixteen control points, positioned in a 4×4 gridlike pattern. The four points on a side of the patch form a Bézier curve segment. The center four points control slopes of the surface. The parameterization requires two variables, s and t, and a point $P(s,t)$ is calculated by:

$$\begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \times M_B \times \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{04} & P_{05} & P_{06} & P_{07} \\ P_{08} & P_{09} & P_{10} & P_{11} \\ P_{12} & P_{13} & P_{14} & P_{15} \end{bmatrix} \times M_B^T \times \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

where M_B is the coefficient matrix for the Bernstein polynomials and the P_i's are the control points of the patch. Slope continuity between two patches is achieved by maintaining collinearity of a control point on the border between them and the control points on either side of the border. Additionally, the ratio of distances between control points on either side of the boundary and the boundary control points must be consistent along the edge. Calculation

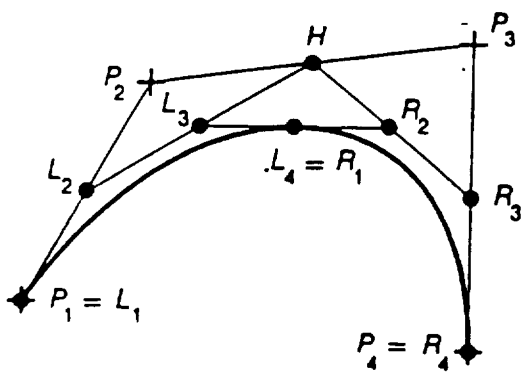


Figure 14: The Bézier curve defined by the points P_i is divided at $t = \frac{1}{2}$ into a left curve defined by the points L_i and a right curve defined by the points R_i. [FOLEY, p. 508]

of the slope at a given point on a patch is achieved by partial derivatives with respect to parameters s and/or t. See Figure 15.

All surfaces in SMART are represented with Bézier bicubic patches; however, sometimes two other representations for curves are used, each of which is equivalent to the Bézier representation. The first is the one-third point representation which requires using the coordinate values of the points on the Bézier curve at parameter values of $t = 0, \frac{1}{3}, \frac{2}{3}$, and 1. Note that the endpoints of the curve for both representations are the same. The one-third points are used in this project to place "control-like" points directly on the curves for clarity in editing multiple, closely spaced Bézier curves.

The Hermite representation is the other method. Often the slopes of the tangent vectors to each endpoint are known. The Hermite representation utilizes the two endpoints and the two tangent vectors to represent the curve. There are matrices which allow easy conversion from one representation to the other, which are included in Appendix C in the matrices2.h file.

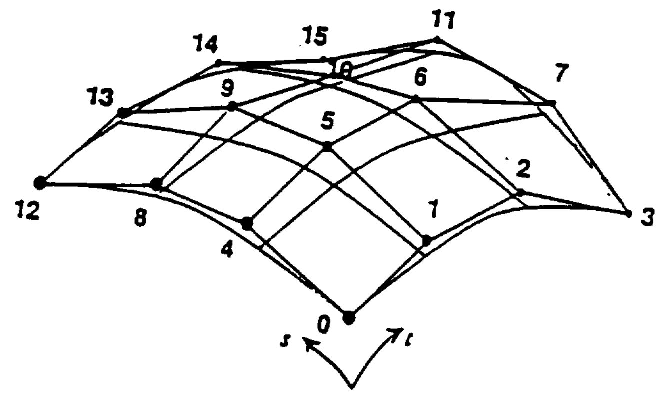


Figure 15: Bicubic Bézier Patch

6 Algorithms for Generating Bulkheads and Ringframes

6.1 Capabilities Developed for SMART Prior to the Project

As with many graphics programs, the image on the screen is constantly redrawn at speeds which fool the human eye into believing that the image has remained continuously on the screen. This is generally accomplished with a looping routine in the software. The main loop checks the mouse location and based on its present coordinates, determines whether the user had placed the mouse over a menu, a bar, or over a viewport on the screen. Based on the mouse's position, certain calculations are accomplished or editing capabilities are available. The display is refreshed each time through the loop, regardless of the function being performed. The loop is exited by explicit menu choices. The structure of the main loop in this portion of SMART is:

```
while (true)
begin
    if choosing a main-menu option then
        exit main loop and redisplay;

    else if over a menu then
    begin
```

```
        if over *main-cross-section-menu* then
            exit main loop and redisplay;
        else if over *store-patches-menu* then
            store patches in SMART data structure and redisplay;
        else if over *type-of-growth-menu* then
            redisplay patches using Bézier or linear format;
        else if over *growth-direction-menu* then
            redisplay with patches interior or exterior to cross-
                section;
        else if over *new-edge-menu* then
            store partial patches, begin new calculations from old
                leading edge, and redisplay;
    end

    else if over a bar then
    begin
        if over *patch-growing-bar* then
            calculate partial patches to given percentage and
                redisplay;
        else if over *radius-bar* then
            calculate new patches given new radius length and
                redisplay;
        else if over *centerline-bar* then
            calculate new patches given new centerpoint position
                and redisplay;
        else if over *ringframe-bar* then
            calculate new patches for ringframe value and redisplay
                with ringframe patches;
    end

    else if over the *right-viewport* then
        edit control points and redisplay;
end
```

The initial algorithms for creating bulkhead patches centered on a given fuselage cross-section represented by a linked list of Bézier curves. Due to the symmetry of the cross-section about a vertical axis of symmetry, the cross-section representation is actually one half of the complete cross-section, as in Figure 12. For the remainder of this paper, reference to a cross-section will imply the "half-cross-section" unless explicit indication to the contrary.

Because many interior tanks of a vehicle are spherical or multi-bubble spherical in shape, the software created a first-guess semicircular cross-section of a tank, interior to the fuselage cross-section, with its endpoints on the on the axis of symmetry of the fuselage cross-section. The points on the tank cross-section were generated around the semicircle to correspond to the percent of arclength of the one-third points of the curves of the fuselage cross-section. The centerpoint of the semicircle was placed at the calculated midpoint between fuselage cross-section endpoints, and the default radius was half the minimum distance from the centerpoint to any one-third point on the fuselage cross-section. The analogous representation using a circle external to the cross-section has also been developed and may potentially be used by aerodynamicists for computational fluid dynamics.

The patches generated between the given cross-section and the semi-circle represented a structural bulkhead between the fuselage and the tank. The percent-of-arclength guide for generating tank points enabled the patches to have reasonable wedge-like shape, which is as close to square-like patches as possible.

The original algorithm was as follows:

```
procedure bulkhead-first-guess (cross-section, centerpt, radius)

begin
    for each curve in cross-section do
    begin
        calculate 1/3 pts on curve;
        calculate inward pointing normal vectors to each 1/3 pt;
        calculate normalized vectors from centerpt in direction
            of each 1/3 point;
        calculate tank-points at length radius from centerpt in
            direction of normalized vectors;

        comment: The two points and two vectors comprise
            the Hermite representation of the curve, as seen
            in Figure 16.
```

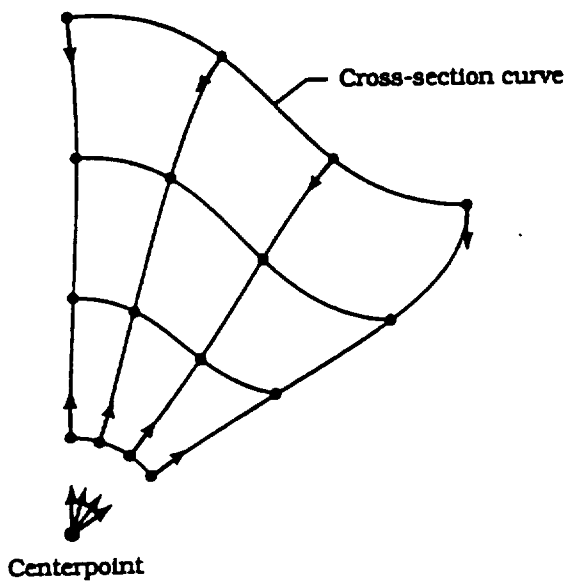


Figure 16: Vectors and points used to calculate a patch

```
        calculate Bézier curves between corresponding
            1/3 points and tank-points;
        calculate Bézier patch from 4 Bézier curves
        place patch on linked list;
    end
end
```

Using established SMART routines, graphical bars and menus were created to enable the user to change parameters. A bar is used to change the radius of the interior tank, allowing growth until the tank cross-section is at most tangent to the fuselage cross-section. The radius is also allowed to decrease to zero to represent a position in the fuselage where there is no interior tank and only a bulkhead. Another bar allows the centerpoint of the semi-circle to move along the axis of symmetry until the tank cross-section is tangent to the fuselage cross-section. Menus are used to allow choice of linear or Bézier curve patch growth between the cross-sections.

Editing of control points is important to allow smoothing of patch wedges. Because the points on the tank were generated according to a given radius, these points can be "dragged" with the mouse around the semicircle by determining the change in arclength and recalculating the actual point on the circle. Movement is restricted to tank-points which are patch "corner" points and one-third points on either side of the corner point are then recalculated. Patch corner points on the axis of symmetry are required to remain on the axis.

The eight patch control points not on either cross-section may also be "dragged" with the mouse to smooth the interior shape of the patches. The change in mouse position is used to calculate the new one-third point position. Points on the fuselage may not be edited in order to preserve the previously determined shape based on aerodynamic and structural constraints. Due to the speed of the Silicon Graphics processor, changes in patches are redisplayed in real time.

6.2 New Results

The specific tasking of this project was to allow automatic generation of a bulkhead or ringframe for a given cross-section(s). The bulkhead would be drawn between two given cross-sections, one representing the fuselage and the other representing the interior tank. This allows the interior tank to have any predetermined shape and not be limited to being circular. The ringframe would be drawn interior to the fuselage cross-section at a default width which could be edited.

6.2.1 Bulkheads

There were several problems to consider in creating patches for the bulkhead between the two cross-sections. The requirement to have "square-like" patches supports the current method of calculating each patch using corresponding curve points of the two cross-sections. The most obvious problem is that both cross-sections may not have the same number of Bézier curve segments. Even if the number of curves is the same, their respective arclengths may not pair up in a fashion to create "nicely" shaped patches. These problems were solved with the following algorithm which compared arclengths of successive curves on each cross-section, splitting curves into two curves when differences in arclength was greater than a predetermined percentage. Locations where splits are made are internally stored and create an additional editing capability, explained in further detail below. The algorithm is as follows:

```
procedure match-curve-arclengths (fuselage-cross-section,
                                  tank-cross-section)
begin
    calculate percent of arclength of each curve in
        fuselage-cross-section;
    calculate percent of arclength of each curve in
        tank-cross-section;
    determine value where curve percents of arclength
        are close enough;

    look at first curves in each cross-section;

    while there is another curve in the fuselage-cross-section
        and another curve in the tank-cross-section do
    begin
        if difference in percents of arclength of current
            curves in each cross-section is greater than
            close-enough-value then
        begin
            split curve with larger percent of arclength (pal):
                first curve will have same pal as smaller curve;
            look at second curve of split curve (other piece
                of larger curve, farther along the cross-section)
                and the next curve on the other cross-section;
        end

        else
            look at the next curves on both cross-sections;
    end
end
```

This algorithm accomplishes two things: both cross-sections end up with the same number of Bézier curves and corresponding curves have near-equal percents of arclength, within an agreed-upon factor. As mentioned above, information is stored as to which curve endpoints were created by splitting original curves. Although the percent of arclength is a reasonable way to line up corresponding curves, it is sometimes preferrable to move the curve endpoints to straighten the patch wedges. "New" endpoints can be "dragged" with the mouse: the change in mouse position is translated into the change in percent of arclength of the split in the original curve and the original curve is resplit with the new percent. The subdivision of a Bézier curve is accomplished by finding control points of the curve as represented by a higher degree polynomial. Each piece of the curve will represent the same cubic polynomial on its own interval domain, as explained in Section 5 of this paper on Bézier curves or [FARIN, pp. 75–6]. Therefore, each new curve is an exact duplicate of the corresponding piece of the original curve. By returning to the original curve each time, the original shape of the cross-section is preserved, but editing of at least some of the curve endpoints is now also a feature of the software.

The other problem that needed consideration was the placement of longerons in the longitudinal structural design. The places where these longerons intersect the fuselage cross-section needed to be at "corner" points of the patches for later structural analysis, as explained in sections 3 and 4 of this paper on aerospace vehicle structure and finite element analysis. The shape of the vehicle in many instances reflects only aerodynamic requirements, and curve endpoints in the fuselage cross-section are usually not in the locations of longeron placement.

The guidance from engineers at NASA Langley Research Center's Vehicle Analysis Branch can be summarized: longerons are ideally spaced equally around the fuselage, but must especially be placed at discontinuity points, or curve endpoints where successive curves are not slope continuous with one another. Thus, a percentage of the longerons to be placed on the cross-section, equal to the percent of arclength of the portion of the cross-section between discontinuity points, should be equally spaced between the discontinuity points. If the desired placement of the longeron is too close to an already existing curve endpoint, a very narrow patch, which is undesirable, might be created. To resolve this, if a desired longeron position is within a curve-length, from the curve endpoint, corresponding to less than twenty-five percent of the equal spacing curve-length for that section of the cross-section between discontinuities, the longeron could be placed at the endpoint.

The resulting algorithm is shown in two parts. The first is the computation of a comparison value used to determine if the placement of the longeron requires the splitting of an existing curve or if it will be placed on an existing curve endpoint:

```
procedure compute-compare-value (equal-spacing-length,
    push-up-length, back-up-length, length-not-yet-included)

comment: Because longerons may be placed at curve endpoints
    and not exactly at the equal-spacing-length, the quantities
    push-up-length, or the curve-length difference of the
    positioning point located past the end of equal-spacing-length,
    back-up-length, or the curve-length difference of the
    positioning point located before the end of equal-spacing-length,
    and length-not-yet-included, or the curve-length total
    from previous curves which did not total to equal-spacing-length
    yet, keep track of differences in the calculated and actual
    position of the previously-placed longeron. The use of
    the term "section"in the following algorithms refers to
    the current portion of the fuselage cross-section between
    discontinuities.

begin
    if first curve in section then
        compare-value = equal-spacing-length;

    else
    begin
        if push-up-length and back-up-length are both zero then
            compare-value = equal-spacing-length;
        else if push-up-length > 0 then
            compare-value = equal-spacing-length − push-up-length;
        else if back-up-length > 0 then
            compare-value = equal-spacing-length + back-up-length;
        if longeron not placed on previous curve then
            compare-value = compare-value − length-not-yet-placed;
    end
end
```

The actual algorithm for placing longerons is:

```
procedure place-longerons(fuselage-cross-section,
                                    number-longerons-to-place)

comment: The first endpoint of a curve is the one closest to the
    beginning of the cross-section, the second endpoint is
    further along the cross-section.

begin
    for each section of fuselage-cross-section between discontinuities do
    begin
        number-longerons-for-section =
                    (number-longerons-to-place) × (pal-of-section);

        if number-longerons-for-section > 0 then
        begin
            calculate equal-spacing-length;

            comment: equal-spacing-length = pal-of-section divided
                by (number-longerons-for-section + 1)

            look at first curve of section;

            while all longerons not placed in section do
            begin
                compute-compare-value;

                if pal-current-curve = compare-value then
                begin
                    place longeron at second endpoint of curve;
                    look at next curve in section;
                end

                else if pal-current-curve > compare-value then
                begin
                    if not first curve of section and
                        longeron was not placed on previous curve and
                compare-value < 25% of equal-spacing-length then
            begin
                place longeron at first endpoint of curve;
                back-up-length = compare-value;
            end

            else if pal-current-curve and compare-value
                differ by > 25% of equal-spacing-length then
            begin
                split current curve (wrt compare-value);
                place longeron at split point;
                look at curve beginning at split point;
            end

            else if pal-current-curve and compare-value
                differ by ≤ 25% of equal-spacing-length then
            begin
                place longeron at second endpoint of curve;
                push-ahead-length = difference of
                    pal-current-curve and compare-value;
                look at next curve in section;
            end
        end

        else (pal-current-current < compare-value)
        begin
            increase length-not-yet-included by
                                        pal-current-curve;
            look at next curve in section;
        end
    end

    place longeron at last endpoint of section;
  end
 end
end
```

This algorithm would be applied to the fuselage cross-section prior to the match-curve-arclengths algorithm to ensure that the interior tank cross-section matches the fuselage cross-section for which the longerons have been considered.

6.2.2 Ringframes

The original algorithms enabled constant percentage ringframes, i.e., those ringframes whose width at each one-third point of the fuselage cross-section was a given percentage of the length of the Bézier curve from the one-third point to the corresponding tank cross-section point, to be created. However, in actual aerospace vehicle design, the requirement for ringframes is constant width and not constant percentage, although constant width is a misnomer. At points of discontinuity, the width of the ringframe is usually a little wider, the leading edge of the ringframe maintaining the basic shape of the cross-section at a place of greater structural stress.

To create a realistic width for the ringframe at points of discontinuity, the following algorithm was used to change the calculated "normal" to the cross-section at the discontinuity point. When normal vectors to each one-third point are calculated, because the tangent to each curve at the discontinuity is different, the curves would have a different normal vector emanating from the same point. This algorithm provides an alternative to just averaging the two normals at the discontinuity point:

```
procedure normal-at-discontinuity

begin
    for each discontinuity point do
    begin
        calculate normal vector to second endpoint of
            first curve meeting at discontinuity;
        calculate normal vector to first endpoint of
            second curve meeting at discontinuity;
        compute points corresponding to tails of two
            normal vectors;
        compute tangent vectors to each curve at discontinuity;
        compute intersection point between two lines through
```

```
                respective points at the head of the normal
                vectors in the direction of the tangent vectors;
            new-normal = vector from discontinuity to intersection
        end
    end
```

The ringframe patches are then constructed as follows:

```
    procedure ringframe-patches

    begin
        for each curve in fuselage-cross- section do
        begin
            calculate 4 inward Bézier curves using Hermite
                representation of one-third point on fuselage-cross-
                section curve, point in direction of normal at
                ringframe-width, two vectors of ringframe-width
                length in direction of normal;
            calculate patch from 4 curves;
            place patch in linked list;
        end
    end
```

7 Conclusion

The algorithms just described have been implemented in the current cross-sections portion of SMART and preliminary feedback from the previously mentioned NASA engineers has been extremely positive. The final implementation will be placed within the currently being developed structures portion of SMART. Actual "snapshots" of the SMART display showing these results are provided in Appendix A of this paper.

The development of software can be a very long and sometimes difficult evolution. Getting a user to specify his or her requirements such that they truly reflect the needs of the user can be extremely frustrating. The specifications may represent a simple concept yet the implementation may be very complex, and the reverse is also often true. This project has added a new dimesion to SMART and should enable the designing and testing of the design phases of aerospace vehicle research and development to be accomplished more expediently in the future.

MASTER PROJECT

Generating The Complete Control Environment Inferface for EASIE

by
Chia-Lin Tsai

Project Advisor:
Dr. James L. Schwing
Associate Professor of CS

Computer Science Department
Old Dominion University
Norfolk, VA 23529

April 1992

ABSTRACT

The Environment for Application Software Integration and Execution, EASIE, was designed to meet the needs of conceptual design engineers that face the task of integrating the results of many stand-alone engineering analysis programs. EASIE is a set of utility programs which supports rapid integration and execution of programs about a central relational database, and it provides users with two basic modes of executing operations: Application-Derived Executive (ADE), a menu-driven execution mode which provides users with sufficient guidance to quickly review data, select menu action items, and execute application programs, and Complete Control Executive (CCE), which provides a full executive interface allowing users in-depth control of the design process. Users can switch between these modes as needed. This project will consider the CCE mode interface.

Two objectives of this project are to redesign the selecting menus by using a windowing system and to reorganize the selecting structures of the selecting menus. The project will be implemented in the X window system, OSF/Motif version.

CONTENTS

LIST OF FIGURES

1. AN INTRODUCTION OF THE EASIE SYSTEM

1.1 WHY EASIE WAS DEVELOPED

The Environment for Application Software Integration and Execution, EASIE, was designed to meet the needs of conceptual design engineers that face the task of integrating the results of many stand-alone engineering analysis programs [REF 9]. The need for such techniques and tools has stemmed from the computer aided design and engineering activities with Langley Research Center's Space Systems Division (SSD).

1.2 WHAT EASIE WAS

EASIE provides access to the programs via a quick, uniform interface. The most predominant system design methodology uses the iterative technique. One progresses to a final solution through successive application of analysis techniques to increasingly refined data. EASIE facilitates this process.

In addition, EASIE is a set of utility programs which supports rapid integration and execution of programs about a central relational database. EASIE provides utilities which aid in the execution of the following tasks: selection of application programs, modification and review of program data, automatic definition and coordination of data files during program execution and a logging of steps executed throughout a design. Therefore, EASIE provides both a methodology and a set of software utility programs to ease the task of coordinating engineering design and analysis codes.

1.3 TWO OPERATION MODES OF EASIE

EASIE provides users with two basic modes of executing operations. The first, Application-Derived Executive (ADE), is a menu-driven execution mode which provides users with sufficient guidance to quickly review data, select menu action items, and execute application programs. The second mode of execution, Complete Control Executive (CCE), which provides a full executive interface allowing users in-depth control of the design process. For example, when using CCE, techniques are provided which allow the user to establish a design sequence and then automatically re-execute the sequence. This allows the engineer to refine input iteratively and review the results with minimum interaction. Users can switch between these modes as needed. This project will consider redesigning the CCE-mode interface.

1.4 WHAT CCE MODE WAS

The CCE-mode interface provides the flexibility of an operating system without requiring the user to track a multitude of files, directories, or data. In CCE, commands can be issued via menu selections or typed in via a command line. Various levels of menus, display, and help text are available.

2. A COMPARISON BETWEEN THE CURRENT EASIE SYSTEM AND THE DESIGN PRINCIPLES

To design a good human interface, we have to consider a number of design principles which are to help ensure good human factors in a design: be consistent, provide feedback, minimize error possibilities, provide error recovery, accommodate multiple skill levels, and minimize memorization. These principles are discussed more fully in [REF 8].

As described above, EASIE provides significant functionality; however, this utility is buried in the current user interface. To see these problems, let us consider each of the factors above with respect to the EASIE interface.

2.1 BE CONSISTENT

First, the EASIE interface is consistent. The conceptual model, functionality, sequencing, and hardware binding in EASIE have been uniform. For example, in the output portion of EASIE, the menu items are always displayed in the same relative position within the menu, system-status messages are shown at a logically fixed place, and the same codings are always employed. In addition, when considering the input portion of EASIE, keyboard characters always have the same function and can be used whenever text is being input, global commands such as *Help*, *Status*, and *Cancel* can be invoked at any time, and generic commands such as *Move*, *Copy*, and *Delete* are provided and can be applied to any type of object in the EASIE system.

2.2 PROVIDE FEEDBACK

Feedback can be given at three possible levels, corresponding to the hardware-binding, sequencing, and functional levels of user-interface design. Currently, the EASIE interface is restricted to keyboard input, thus hardware-binding is trivially satisfied. EASIE provides some sequencing feedback such as when each word of the input language (command, position, object, etc.) is accepted by the system. However, EASIE does not provide functional feedback, for example, there is no acknowledgement communicated to the user when an operation is processing.

2.3 MINIMIZE ERROR POSSIBILITIES

Users will make input errors in any system, and it is the job of the user interface to minimize error possibilities. The system tries to minimize the errors as possible. No matter how, there may be some error occurred in the future.

2.4 PROVIDE ERROR RECOVERY

It is important to provide error recovery: *Undo*, *Abort*, *Cancel*, and *Correct*. Unfortunately, EASIE currently only provides the *Cancel* feature.

2.5 ACCOMMODATE MULTIPLE SKILL LEVELS

User interface methods which can be used to help accommodate multiple skill levels are accelerators, prompts, help, extensibility, and hiding complexity. EASIE, however, does not provide accelerators which are faster interaction techniques that replace slower ones. Secondly, it provides some prompts which is to suggest what to do next, but these are not generally sufficient. Thirdly, the EASIE interface does not offer a sufficiently detailed help facility. For example, the EASIE interface does not give a full explanation about how to use commands. EASIE does offer a primitive extensibility which means letting the user add additional functionality to the interface by defining new commands as combinations of existing commands. Finally, the EASIE interface does not provide complexity hiding which can allow new users to learn basic commands and to start doing productive work without becoming bogged down with specifying options, learning infrequently used specialized commands, or going through complicated start-up procedures.

2.6 MINIMIZE MEMORIZATION

The final principle of user interface design is to minimize memorization. The original configuration of the EASIE system seems to be redundant. A new user has to read commands on a complicated menu to get what is needed. It is not economic.

3. TWO OBJECTIVES OF THIS PROJECT

There are two objectives of this project: redesign the selecting menus by using a windowing system, and reorganize the selecting structures according to the design principles outlined above.

3.1 REDESIGN THE SELECTING MENUS

Redesign of the menus of the Complete Control Executive (CCE) mode will be implemented in the X window system, OSF/Motif version. Since the initial menus of CCE mode are alphabetical with numerical selection, we want to redesign those menus to be windows. This will allow the accommodation of skill level in the EASIE system: hide complexity from the user.

Window-based user interfaces [REF 7] have become a common feature of most computer systems, and users are beginning to expect all applications to have polished user-friendly interfaces. The X window System, developed at Massachusetts Institute of Technology (MIT), is an industry-standard software system that allows programmers to develop sophisticated user interfaces that are portable to any system that supports the X protocol. In addition, X allows programs to display windows containing text and graphics on any hardware that supports the X protocol without modifying, recompiling, or relinking the application. X is based on a network-transparent client-server mode. The X server creates and manipulates windows in response to requests from clients, and sends events to notify clients of user input or changes in a window's state. One important different between X and many other window systems is that X does not define any particular user interface style. X also provides a device-independent layer that serves as a base for a variety of interface styles.

The OSF/Motif version of the X window system [REF 2] is a graphical user interface combining a toolkit, presentation description language, window manager, and style guide. First, the OSF/Motif toolkit is a rich and varied collection of widgets and gadgets for building OSF/Motif applications. The toolkit provides a standard graphical interface upon which the window manager is based. Second, the OSF/Motif presentation description language allows application developers and interface designers to create simple text files that describe the visual properties and initial states of interface components. Third, The window manager works with the toolkit to manage the operation of windows on the screen. The window manager provides functions for moving and resizing windows, reducing windows to icons, restoring windows from icons, and arranging windows on the workspace. Finally, the style guide describes the standard for window manager and toolkit behavior. It is a guide to usage, providing application writers with guidelines for using toolkit widgets, widget writers with guidelines for designing new widgets, and window manager writers with guidelines for designing new or customized window managers. Together, these four elements provide the OSF/Motif to be a standard of user interface behavior for applications.

3.2 REORGANIZE THE SELECTING STRUCTURES

The second objective is the reorganization of the interface with respect to the previously mentioned design principles. Version 1.0 of the EASIE interface has seven different standard menus in addition to a "Permanent" Menu of commands. They are Utility Selection Menu, Workspace Control Menu, Data Review/Modification Menu, Application Execution Menu, Procedure Execution Menu, Procedure Building Menu, Template Building Menu, and the "Permanent" Menu mentioned above. We find that the old ones are to be redundant and ineffective. One objective of the reorganization will be the minimization of memorization. Let us take an example of Workspace Control Menu shown in Figure 1 on the next page.

There are twenty-eight choices. It is difficult for users to select their choices. They have to read all the selections, then make their decisions. Therefore, we want to reorganize those menus to be more efficient. Let us have an example. If your selection concerning the WORKSPACE, then there will be only six choices: READ DESCRIPTION, NEW, COPY, ACTIVATE, SAVE TEMPLATE, and REMOVE FROM UFD. It will be easier for users to choose what they need. In addition, the user can not enter some commands with file name or with path if he/she does not know or make sure about the file names or paths. There is no way for the user to get the information of the file name or path he/she needs.

```
WORKSPACE CONTROL
                                                     COMMAND FORMAT
 1 - READ DESCRIPTION   -  WORKSPACE                 RD WS     <name>
 2 -                    -  CONFIGURATION             RD CFG    <name>
 3 -                    -  TEMPLATE                  RD TPL    <name>
 4 -                    -  APPL. PROG.               RD APPL   <name>
 5 -                    -  PROCEDURE                 RD PROC   <name>
 6 - CLEAR LOG OF OLD INFORMATION                    CL
 7 - TYPE               -  COMMAND LOG               TY LOG    <name>
 8 -                    -  PROCEDURE                 TY PROC   <name>
 9 - NEW                -  WORKSPACE                 N WS
10 -                    -  CONFIGURATION             N CFG     <name>
11 - COPY               -  WORKSPACE                 CP WS     <f,to>
12 -                    -  PROCEDURE                 CP PROC   <f,to>
13 - ACTIVATE           -  WORKSPACE                 ACT WS    <name>
14 -                    -  CONFIGURATION             ACT CFG   <name>
15 -                    -  TEMPLATE                  ACT TPL   <name>
16 -                    -  APPL. PROG.               ACT APPL  <name>
17 -                    -  UTILITY                   ACT UTL   <menu>
18 -                    -  INPUT TEMPL               ACT ITPL
19 -                    -  OUTPUT TEMPL              ACT OTPL
20 -                    -  PROCEDURE                 ACT PROC  <name>
21 -                    -  PROGRAM UFD               ACT PUFD  <path>
22 - SAVE TEMPORARY     -  WORKSPACE                 SA WS     <name>
23 -                    -  PROCEDURE                 SA PROC   <name>
24 - REMOVE FROM UFD    -  WORKSPACE                 RM WS     <name>
25 -                    -  CONFIGURATION             RM CFG    <name>
26 -                    -  TEMPLATE                  RM TPL    <name>
27 -                    -  PROCEDURE                 RM PROC   <name>
28 - SET USER LOGIN CHARACTERISTICS                  SLOG

ENTER COMMAND:
```

Figure 1. *WorkSpace Control menu*

4. AN OUTLINE OF THIS PROJECT

4.1 A GENERAL VIEW

The main purpose of this project is not to change the existing EASIE system, but to design a nice-looking, window selection menu for EASIE. The work of this project is to provide a front end to EASIE for users which handles basic menu processing and passes some commands as necessary to the EASIE command processor. Thus, some processing and error checking will be provided by the front end. EASIE commands are written into a file called easie.file in the user or home directory where they are available to the EASIE command processor, and they are also shown in the window for the user. Error messages and warnings will also be displayed in this window.

When the user starts EASIE, the user may enter a file name as an argument. Alternatively, the system will use a default file name called easie.input. The contents of this file define the basic operating environment for EASIE and include basic filenames and default directories. They are WorkSpace (.WS), Configuration (.CFG), Application (.APPL), Template (.TPL), Procedure (.PROC), home directory, program directory, and base directory. For format purpose, a blank line is entered if there is no corresponding file name or directory for a particular environment. The contents may be changed after being executed by the system. The following is an example format of the easie.input.

```
/tmp_mnt/home/tsai_c/project/ws.WS
/tmp_mnt/home/tsai_c/project/cfg.CFG

/tmp_mnt/home/tsai_c/project/tpl.TPL

/tmp_mnt/home/tsai_c/project
/tmp_mnt/home/tsai_c/project/program
/tmp_mnt/home/tsai_c/project/base
```

Figure 2. *Basic Environment File, easie.input*

The state diagrams in Appendix A define the operation implemented for the improved EASIE interface. Appendix B gives a user manual for the new CCE mode interface of the EASIE system. What follows is a description of the new CCE interface.

Upon initialization, the CCE mode interface will pop up a window with eight basic selections in a main menu bar and the current status or operating environment in the working area. These eight selections are Tools, Open, Retrieve, Update, Organize, Execute, Print, and List. The user can use a mouse to choose any of these selections. The main menu is further organized in a hierarchy which is a pull-down for the first level of sub-menu and a pull-right for further levels of sub-menu.

The working area will show the current status which includes the file names of WorkSpace, Configuration, Application, Template, Procedure, the home, program, and base directories. These data are read from the default file, easie.input, or the filename which the user entered as an argument. If there is no such file name or if the file does not specify that environment variables, the system will display **<null>** on the corresponding position in the working area. The current status will be updated during executing the system. Before exiting the system, the user will be asked whether to save the current status or not. If the answer is "OK", the updated status will be saved; otherwise, the updated status will not be saved, and the status will be kept as same as the first time the user logged in.

4.2 IMPROVEMENTS

First, we have given the user a windowing system for selecting choices. Thus, it is simpler for the user to select his/her choice and memorization and confusion of the previous system minimized. Second, we offer on-line help to assist the user. The user can get the on-line help whenever he/she pushes the help buttons. Third, we provide enhanced utility to the user, for example, the user can change his/her program or base directory as he/she needs. We also provide List selection for the user. The system provides a list of all appropriate files for a given situation, again, limiting the memorization and confusion factors. Fourth, we offer the current status. The current status indicates the current operating environment of EASIE for the user. Fifth, the user is provided with a file list for selecting when he/she needs to enter some file names. Finally, we remove some unnecessary and confusing menu choices, for example, Toggle the display mode (EASIE command T), Return to Previous Menu (EASIE command R), Quit this sequence of menus and return to the utility selection menu (EASIE command Q), Zero: cancel a command sequence (EASIE command O).

4.3 SAMPLE SESSION USING THE CCE MODE

The following EASIE session is included as a sample for the CCE mode user to follow. The screens given in Appendix C were recorded during the session. References to screens in Appendix C will be denoted by Screen n where n represents the screen number. This sample session has been put together to highlight the capabilities of the EASIE system using the CCE mode environment. To initialize EASIE, we type "easie". As described above this uses the file "easie.input". For reference, the contents of this file have been given previously in Figure 2.

Screen 1 is the general log-in screen presented to users who log in using default log-in characteristics. It includes a menu bar with eight selections, and a working area shown the current status. The user can resize the screen 1 by using the mouse device. Screen 2 is the resizing window. We will use screen 2 to present the main window in the following examples. Screen 3 shows the contents of the input file, easie.input under the /tmp_mnt/home/tsai_c/project directory, and it just shows the user the contents of the input file before executing the program.

The menu is organized in a hierarchy which is a pull-down for the first level of sub-menu, and a pull-right for further levels of sub-menu. The first level in the main menu is the selections of the main menu bar which are Tools, Open, Retrieve, Update, Organize, Execute, Print, and List. Now we choose the Tools selection, pull the sub-menu down, and select the General Concept from the pull-right sub-menu. Note that if there are pull-right sub-menus for the choice, there is a triangle after that choice. Screen 4 shows the condition above. Screen 5 shows the pop-up window after pushing the General Concept choice. The user can push the OK button in the pop-up help window. The pop-up help window will be closed.

Screen 6 shows that we choose the System command. A System Command widget will be popped up. Screen 7 is the pop-up widget. The user can type the system command in the widget, and push the Ok button or strike the Enter key. The EASIE command will be generated and written into one specific file, easie.file. Pushing the Clear button will erase the contents which the user typed in. The Help button will pop a help widget up and show the on-line information for that widget. Screen 8 is the pop-up on-line help widget pushed by the Help button. If the user pushes the Close button in Screen 7, the System Command widget and the on-line help widget will be closed. The situation for the Comment choice under the Tools selection is similar to the System command.

Screen 9 shows the results when the user pushes the clear Log choice. There are two sub-choices for the clear Log. An appropriate EASIE command will be generated by pushing each of these sub-choices.

Screen 10 represents the results of pushing the Open selection. Next we pushed the HOME sub-choice which means we want to activate an application from the home directory. An ACTIVATE-Application-HOME dialogue widget will be popped up. In this widget, all files with an .APPL extension will be appeared. The functions of the buttons on the bottom of this widget are similar to the buttons described above, except for the Filter button. Screen 11 shows how the user chose the file name he/she wants. The chosen file name will appear in the Selection column. The Filter button is a way to change the directory. Select the directory the user want to change to, and then push the Filter button. The list of file names will be modified and shown for the new directory. Screen 12 shows the widget described above. Screen 13 shows the on-line help information for the user by pushing the Help button.

Screen 14 shows the result when we chose the WorkSpace sub-choice under the New pull-down sub-menu. New means that we want to clear the WorkSpace filename in the current status. Screen 15 is presented the result.

The Retrieve selection and the Update selection are similar to the Open selection. Notice that the sub-choice

Directory under the Update selection is an improvement of the modified CCE mode. Let us take a look of this choice. Screen 16 shows that we pushed the BASE sub-choice of the Directory under the Update selection. It means that we want to update the directory for base programs and configurations. A Change-Directory widget will be popped up, and the default base directory will be shown in this widget. Screen 17 is the pop-up widget. The functions of this widget are similar to the System command's.

Now we take a look of the Organize selection. There are four choices: Copy, Remove, Save, and ReName. Screen 18 shows the result when we chose the Application sub-choice of the Copy. When the Application sub-choice under the Copy pull-down menu selected, this result is in Screen 19. Screen 19 presents all the file names with .APPL extension under the directory. The functions of this widget are same as the ACTIVATE-Application-HOME widget. After selecting a file name, the system will pop a COPY-to widget for the user to enter the copy-to file name. Screen 20 demonstrate the COPY-to widget. The functions of this widget is similar to that of the pop-up widget of the System command choice. The functions of the Remove, Save, and ReName sub-choices under the Organize selection, the Execute selection, the Print selection, and the List selection are similar to the functions mentioned above.

Next we consider how to exit the EASIE system. We select the Quit EASIE choice under the Tools selection. Screen 21 shows the choice. A question widget will be popped up. Screen 22 is the pop-up question widget. It will ask the user whether to save the current status or not. Pushing the Ok button means to save the modified status. Pushing the Cancel button means to keep the original status as the first time the user logged in. Screen 23 presents the result when the user pushes the Ok button for saving the modified status. Screen 24 shows an example of EASIE commands generated during executing the system. These generated EASIE commands will be written into a file called easie.file, and will be sent to the EASIE command processor.

4.4 COMMAND SUMMARY USING THE CCE MODE

The following section summarizes and collects the EASIE command information of the CCE mode interface as it is organized in this project. There are eight selections in the menu bar of the main window. They are Tools, Open, Retrieve, Update, Organize, Execute, Print, and List. In what follows, we distribute EASIE commands under each of these choices. It should be noted that the user is no longer responsible for knowing the structure of these commands. The new interface automatically provides this information. Some selections do not have the EASIE commands since the functions of those selections can be performed by the modified CCE mode interface. For example, Help choice under the Tools selection, and the List selection. As described above, some functions have been added to the modified CCE mode interface, for example, changing the base or the program directory.

<u>Tools Selection</u>

S - System command
  Used to pass a command to the operation system.
  Form: S <system command>
  Example: S ls

C - Comment
  Used to place a comment in the command log. This allows notes to be inserted in the log for later reference and clarity.
  Form: C <comment>
  Example: C enter today's date

CL - Clear Log
  Used to remove prior information from a cluttered command log or clear the log completely.
  Form: CL <type>
  Example: CL D
  Allowable object types: D - prior to a given date
  T - total, a new log started

L - Log out
  Used to give an orderly closeout of the EASIE system, and return the user to the computer's operation system. Before exiting the EASIE, the system will pop up a question widget, and ask the user: "Save Current Status?". If the answer is "OK", the system will save the current status into a file called easie.input; otherwise, it will not save the updated status, and it will keep the original status. After that, the system will close all the windows which the user opened during executing the system.
  Form: L
  Example: L

<u>Open Selection</u>

ACT - Activate
  Used to associate the indicated object with the user's workspace.
  Form: ACT <type> <filename>
  Example: ACT CFG /tmp_mnt/home/tsai_c/project/cfg.CFG
  Allowable object types: APPL, CFG, ITPL, OTPL, PROC, TPL, WS

N - New
  Used to create a new object or get a fresh object.

Form: N <type>
Example: N WS
Allowable object types: WS, CFG, TPL, PROC

Retrieve Selection

TY - Type
Used to type the indicated file at the terminal.
Form: TY <type> <filename>
Example: TY PROC /tmp_mnt/home/tsai_c/project/proc.PROC
Allowable object types: LOG, PROC, BAT, FILE

RVU - Review
Used to review data from the configuration database. This command invokes the interactive "REVIEWER" program, and will display for possible modification a "view" of a configuration database. A view of a database is defined as the collection of variables defined by a data template.
Form: RVU <type>
Example: RVU IDB
Allowable object types: IDB, ODB, TPL

RD - Read Description
Used to read a file description associated with any workspace, program procedure, template, or database.
Form: RD <type> <filename>
Example: RD APPL /tmp_mnt/home/tsai_c/project/appl.APPL
Allowable object types: APPL, CFG, ITPL, OTPL, PROC, TPL, WS

Update Selection

ED - Edit
Used to invoke a system editor for certain operations.
Form: ED <type> <filename>
Example: ED PROC /tmp_mnt/home/tsai_c/project/proc.PROC
Allowable object types: LOG, PROC, TPL

CD - Change Description
Used to change a file description of the indicated object by using the system editor.
Form: CD <type> <filename>
Example: CD TPL /tmp_mnt/home/tsai_c/project/tpl.TPL
Allowable object types: APPL, CFG, ITPL, OTPL, PROC, TPL, WS

Organize Selection

CP - Copy
Used to copy one file to another.
Form: CP <type> <filename> <filename>

Example: CP CFG /tmp_mnt/home/tsai_c/project/cfg.CFG /tmp_mnt/home/tsai_c/project/configuration.CFG
Allowable object types: APPL, CFG, PROC, TPL, WS, FILE

RM - Remove
Used to remove a file from the user's file directory.
Form: RM <type> <filename>
Example: RM CFG /tmp_mnt/home/tsai_c/project/cfg.CFG
Allowable object types: APPL, CFG, PROC, TPL, WS, FILE

SA - Save
Used to save the indicated object for the later work.
Form: SA <type> <filename>
Example: SA PROC /tmp_mnt/home/tsai_c/project/proc.PROC
Allowable object types: PROC, WS

CN - Change Name
Used to change the name of a file as indicated.
Form: CN <type> <old filename> <new filename>
Example: CN TPL /tmp_mnt/home/tsai_c/project/tpl.TPL /tmp_mnt/home/tsai_c/project/template.TPL
Allowable object types: APPL, CFG, PROC, TPL, WS, FILE

Execute Selection

EX - Execute
Used to execute an indicated application program or procedure command file.
Form: EX <type> <filename>
Example: EX APPL /tmp_mnt/home/tsai_c/project/appl.APPL
Allowable object types: APPL, PROC

SUB -Submit
Used submit a job for batch processing.
Form: SUB <type> <filename>
Example: SUB APPL /tmp_mnt/home/tsai_c/project/appl.APPL
Allowable object types: APPL

Print Selection

PR - Print
Used to print an indicated file at a local hard copy printer.
Form: PR <type> <filename>
Example: PR LOG /tmp_mnt/home/tsai_c/project/log.LOG
Allowable object types: LOG, PROC, BAT, FILE

PRVU - Print Review
Used to print a template or a view of the database.
Form: PRVU <type>
Example: PRVU IDB
Allowable object types: IDB, ODB, TPL

5. CONCLUSION

EASIE is consisted of a set of utility programs to meet the needs of conceptual design engineers who needs many stand-alone engineering analysis programs. Since the selecting menu of the original EASIE interface are the alphanumerical menu selection, and the structures of the selecting menu are not well-organized. Thus, the main purpose of this project is to give a front end to EASIE for the users. This project is considered in CCE mode, and is implemented in the X window system, OSF/Motif version.

This paper is organized by the introduction of the EASIE system, a comparison between the current EASIE system and the design principles, two objectives of this project, and an outline of this project.

At the beginning, this paper gives the reader a general concept about the EASIE system. By comparing to the design principles, we found that the current EASIE got some flaws. Therefore, the two objectives of this project are to redesign the selecting menus and reorganize the selecting structures. To redesign the selecting menus by using a windowing system is to *hide complexity* from the user. To reorganize the selecting structures is to *minimize memorization*. Finally, we give the reader a general concept about the modified EASIE system in CCE mode and some improvements we did.

Although we enhance some functionality to the current

EASIE system in CCE mode, there are still some potential bugs in this project. First, the input file must be in the correct format; otherwise, the system will not perform well. This project does not provide file-existence checking. Second, we suggest that we can minimize the levels of pull-right menus. It may be more organizing if we put the choices in the second level of pull-right menus to be some buttons in the pop-up widget as pushing the choice of the first level of pull-right menu.

REFERENCES

[REF 1] ......, by the Staff of O'Reilly and Associates, Inc., X Toolkit Intrinsics Reference Manual, second edition for X11, release 4, Volume five, O'Reilly & Associates, Inc., 1990

[REF 2] ......, OSF/Motif Style Guide, Open Software Foundation, Prentice-Hall, Inc., New Jersey, 1988

[REF 3] Al Kelley and Ira Pohl, A Book On C Programming in C, second edition, The Benjamin/Cummings Publishing Company, Inc., California, 1990

[REF 4] Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, second edition, Prentice-Hall Inc., New Jersey, 1988

[REF 5] Brian W. Kernighan and Rob Pike, The UNIX Programming Environment, Prentice-Hall, Inc., New Jersey, 1984

[REF 6] Dan Heller, Motif Programming Manual For OSF/MOTIF Version, Volume Six, Motif edition, O'Reilly & Associates, Inc., 1991

[REF 7] Douglas A. Young, The X Window System Programming and Applications with Xt, OSF/MOTIF edition, Prentice-Hall, Inc., New Jersey, 1990

[REF 8] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, Computer Graphics: Principles and Practice, second edition, Addison-Wesley Publishing Company, U.S.A., 1990

[REF 9] James L. Schwing, Lawrence F. Rowell, and Russell E. Criste, The Environment for Application Software Integration and Execution (EASIE) Version 1.0 Volume III Program Execution Guide, NASA TM-100575, National Aeronautics and Space Administration (NASA) Langley Research Center, Hampton, Virginia, April 1988

[REF 10] Joseph S. Dumas, Designing User Interfaces for Software, Prentice Hall, New Jersey, 1988

[REF 11] Samul P. Harbison and Guy L. Steele Jr., A Reference Manual, 3rd edition, Prentice-Hall, Inc., New Jersey, 1991

[REF 12] William M. Newman and Robert F. Sproull, Principles of Interactive Computer Graphics, second edition, McGraw-Hill Book Company, U.S.A., 1979

Master's Project Report

Application Driven Interface Generation for EASIE

by

Ya-Chen Kao

Advisor : Dr. James L. Schwing

April 28, 1992

Department of Computer Science
Old Dominion University
Norfolk, VA 23529 – 0162

ABSTRACT

The Environment for Application Software Integration and Execution, EASIE, provides a user interface and a set of utility programs which support the rapid integration and execution of analysis programs about a central relational database. EASIE provides users with two basic modes of execution. One of them is a menu–driven execution mode, called Application–Driven Execution (ADE), which provides with sufficient guidance to review data, select a menu action–item, and execute an application program. The other mode of execution, called Complete Control Execution (CCE), provides an extended executive interface which allows in depth control of the design process.

Currently, the EASIE system is based alphanumeric interaction techniques only. It is the purpose of this project to extend the flexibility of the EASIE system in the ADE mode by implementing it in a window system. Secondly, a set of utilities will be developed to assist the experienced engineer in the generation of an ADE application.

Table of Contents

1. Introduction

The Environment for Application Software Integration and Execution, EASIE, which developed for NASA by Old Dominion University, Computer Sciences Corporation and Vehicle Analysis Branch of NASA Langley, provides with a methodology and a set of software utility program to ease the task of coordinating engineering design and analysis codes.

EASIE provides a user interface and a set of utility programs which support the rapid integration and execution of analysis programs about a central relational database [1]. EASIE provides users with two basic modes of execution. One of them is a menu–driven execution mode, called Application–Driven Execution (ADE), which provides users with sufficient guidance to review data, select a menu action–item, and execute an application program. The other mode of execution, called Complete Control Execution (CCE), provides an extended executive interface which allows in depth control of the design process. In CCE, commands can be issued via menu selection or directly typed. Although CCE provides the flexibility of an operating system, it also is complicated to use like an operating system. Most users currently access the EASIE system via the menu–driven mode known as ADE.

In general, the EASIE system addresses the needs of two different classes of users who be involved in the buildup and use of an engineering design system.

The first classification represents the engineer/designer/analyst. This group conducts the design study through the execution of modeling and analysis programs and the generation of data required to this evaluate the design against its objectives. EASIE documentation will refer to this group as "EASIE system users" or, more often, as "designers". In general, these users are only interested in executing programs already installed into an EASIE design system [1].

A second group aided by EASIE will be referred to as "application programers" or "experts". These programers/engineers are responsible for the development and improvement of modeling and analysis programs used in the engineering design process. EASIE documentation will refer to this group as "experienced engineers". They are the experts with respect to particular application programs and can defines its input and output variables [1].

2. EASIE : ADE–mode Considerations

The predominant design method used by engineers is the iterative technique. One processes to a final solution through successive applications of analysis techniques to increasingly refined data. EASIE provides a basic user tools which support the selection and execution of application programs, viewing, and editing of program data. EASIE also provides tools for a design team to easily manage the design environment by providing the ability to quickly integrate new analysis programs and data with the existing environment.

2.1 Concepts of EASIE system

<u>Configuration Data:</u>

Configuration data is stored in a system–managed database. An advantage of the EASIE user interface is that data held in the database are automatically communicated to either a user or an application program in an appropriate format. Once the basic data definitions and values have been made, a copy of this "master" database is placed in a controlled project directory. Access to this database is provided on a "read only" basis. That is, the users may display the configuration data for review, or they may make copy of the database for their personal files. Updates to the master database can be entered only by the design manager [1].

<u>Reviewer:</u>

The EASIE software interface provide a program called the "REVIEWER" which can access any data. Based upon an indicated analysis program or other dataset in the database for a designer, the REVIEWER, using information contained in the database, can then make the appropriate selection to retrieve the necessary input and present that data at the terminal.

<u>Data Templates:</u>

The software screen forms used to control the flow of data to and from the database are called data templates. A data template is basically a list of all data required for input (or supplied as output) by a given program along with their required data formats. Since data templates are generated by EASIE utility program. Finally, access to these data templates is used in conjunction with the REVIEWER to directly modify the variables in the database when presented during the Review process [1].

<u>Formatter:</u>

A final utility called the Formatter uses the data templates to enable the automatic generation of FORTRAN subroutine source code, called Formatter code, which can be placed in the application program allowing it to retrieve data or store data into the database during program execution.

2.2 Sample Session for ADE mode

Menu displayed during an ADE session are typically created by experienced engineers to guide new users through the proper sequence of steps to conduct some particular design activity. Given such an interface, an introductory user can easily learn to manipulate data and execute programs in the Application Derived Executive (ADE) mode. Now we look at an example to describe an interaction with EASIE for a given application. This example illustrates capabilities of the EASIE system. It consists of four short programs that define and draw a box. Figure 1 represents the basic relationship among these programs and their data.

Within the concepts of the EASIE system, we can realize this figure in a straight forward manner.

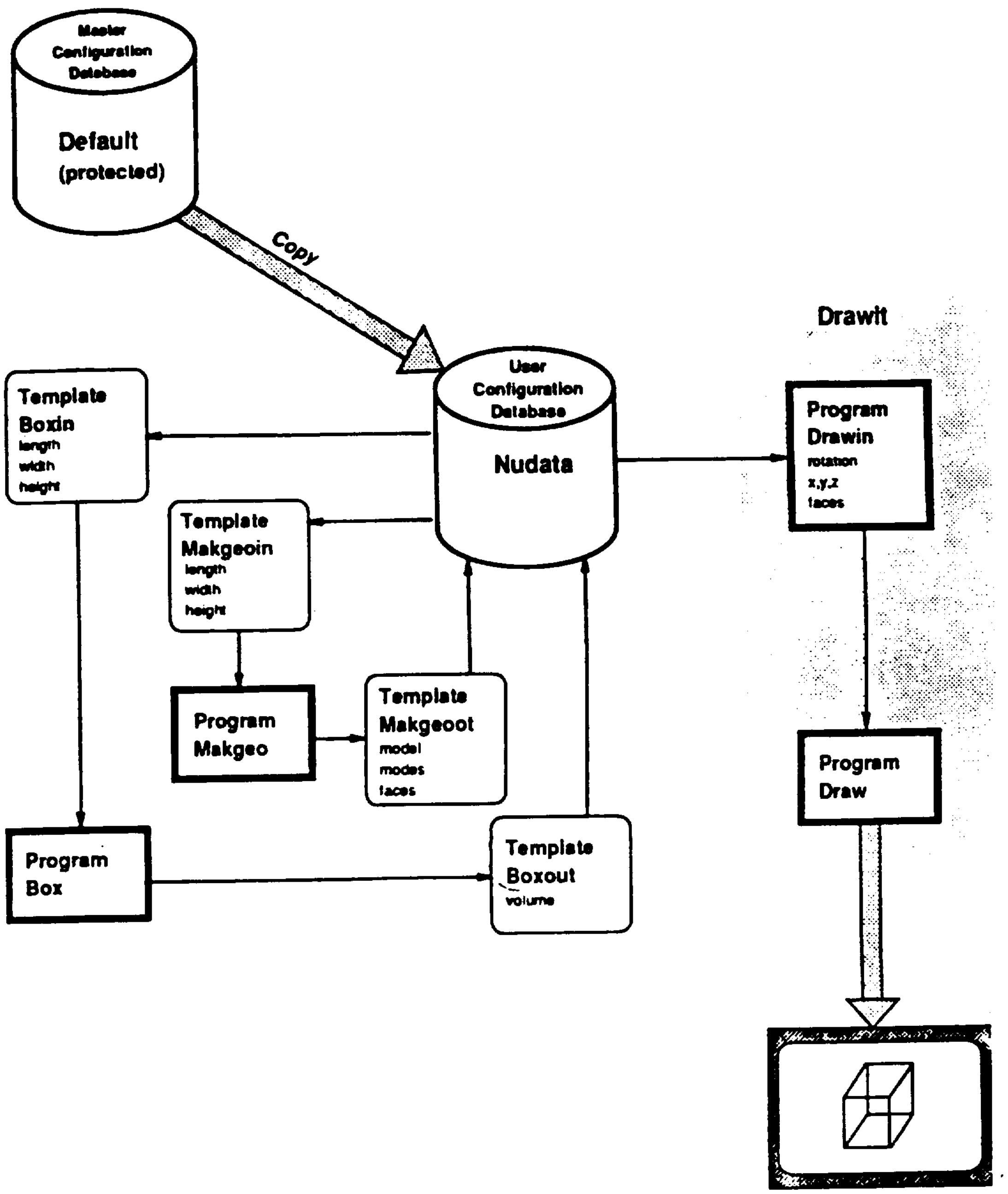


Figure 1. Flow Diagram For Sample Session Using EASIE

The Box program extracts dimensional data from the database, calculates physical properties(volume), and stores it in the database. The MAKGEO program extracts the dimensional data from the database, create a geometric boundary representation for the box, and stores that data in the database. The object of the DRAW program is to display the box geometry that exists in the database.

The session commences with the user entering the following command (underlined text represents user input) .

```
$ exmenu
    CS - SELECT A CONFIGURATION
    DC - DELETE A USER CONFIGURATION
    CD - EDIT A CONFIGURATION DESCRIPTION FILE
    R  - REVIEW PROGRAM INPUT
    E  - EXECUTE A PROGRAM
    P  - PRINT OUTPUT FILES
    X  - EXIT
  Input: label – menu choice, <CR> – reprint menu – CS
```

SCREEN 1

Screen 1, the first screen presented, provides a menu of the commands available for basic interaction: selection, deletion, editing , and review of configuration data, program execution, and printed out. From figure 1, the first choice from this menu would be CS for the selection of a configuration database.

```
 MASTER CONFIGURATIONS
DEFAULT

 USER CONFIGURATIONS
SHOULD A NEW CONFIGURATION BE CREATED ( Y = yes )
Y

COPY SOURCE CONFIGURATION ( FOR DEFAULT VALUES )
TO THE DESTINATION CONFIGURATION

ENTER SOURCE CONFIGURATION ( "1" TO LIST ) :
DEFAULT

ENTER DESTINATION CONFIGURATION ( "1" TO LIST ) :
NUDATA
```

COMMENCING DATABASE COPY

SCREEN 2

Screen 2 is the result of that selection. The first four lines displayed indicate the existence only the master configuration database DEFAULT. Since EASIE users may changed only personal database. Then, the following steps are the copy of configuration. In ADE mode, the menu provides basic selections available for designers, then designers use the keyboard for alphanumeric input to the program. Successive menu choices allow designers to complete a special execution of programs. Details of the sample session described above can be found in the EASIE Volume III – Program Execution Guide.

Control of the system during ADE is governed by a command procedure designed by the experienced engineer who , as it will be seen, must also be an expert on the EASIE system. The next section demonstrates how procedures are structured in EASIE. Thus the construction of such procedures should be considered a priority in order to fit the needs of designers who will use the ADE mode.

2.3 Menu Manipulation and Construction in ADE mode

Since design is generally iterative in its nature, the procedures controlling the EASIE sessions for ADE users should have the ability to jump and loop when needed. During the execution of a procedure, EASIE will keep track of its position via a procedure counter (pc). The procedure counter may be reset by jumping to a labeled statement within a procedure. Labels are placed in a procedure with a comment statement of the form below:

```
C LABEL  :   < label_id >
```

Consider this example.

```
GET JMPL THERE
.
.
.
C LABEL : THERE
```

Menus can be presented to the ADE user via the "GET MENU" command. The format for this command is :

```
GET MENU   < n >        where < n > represents an associated menu number.
```

Menus are stored in separated files, whose format is detailed below. The combination of this command, along with the ability to jump and loop within a procedure, provides EASIE with the flexibility to make the ADE interface work. The procedure file to be executed is linked to a particular choice of USER–ID and is automatically executed when EASIE is initiated with that ID. The following illustrates the procedure and its associated menu files [1].

```
C LABEL : MM
GET MENU 1
C LABEL : R
GET MENU 2
C LABEL : E
GET MENU 3
C LABEL : CS
GET CFG
GET JMPL MM
C LABEL : DC
RM CFG
GET JMPL MM
C LABEL : CD
CD CFG –
GET JMPL MM
C LABEL : BIR
RVU BOXIN
GET JMPL R
C LABEL : BOR
RVU BOXOUT
GET JMPL R
C LABEL : MR
RVU MAKGEOIN
GET JMPL R
C LABEL : DR
RVU DRAWIN
GET JMPL R
C LABEL : BX
EX APPL BOX
GET JMPL E
C LABEL : MX
EX APPL MAKGEO
GET JMPL E
C LABEL : DX
EX APPL DRAWIT
GET JMPL E
C LABEL : X
L
N
```

Figure 2. A printout of the procedure file

```
CS   CS   SELECT A CONFIGURATION
DC   DC   DELETE A USER CONFIGURATION
CD   C    EDIT A CONFIGURATION DESCRIPTION FILE
R    R    REVIEW PROGRAM INPUT
E    E    EXECUTE A PROGRAM
P         PRINT OUTPUT FILES
X    X    EXIT
```

Figure 3a. EXMENU.PROC_1.

```
BIR   BI   REVIEW INPUT FOR BOX
BOR   BO   REVIEW OUTPUT FOR BOX
MR    M    REVIEW INPUT FOR MAKGEO
DR    D    REVIEW INPUT FOR DRAWIT
MM    R    RETURN TO MAIN MENU
```

Figure 3b. EXMENU.PROC_2.

```
BX   B   EXECUTE BOX
MX   M   EXECUTE  MAKGEO
DX   D   EXECUTE  DRAWIT
MM   R   RETURN TO MAIN MENU
```

Figure 3c. EXMENU.PROC_3.

A review of the commands in figure 2 procedure reveals the use of the GET MENU command three times — namely, command 1, 2 and 3 . Since each of these has a different number, it refers to each of the menus listed on figure 3. For example, GET MENU 2 refers to the menu contained in file EXMENU.PROC_2. In general, the use of the statement GET MENU < n > in a procedure with the name <proc_id> requires the existence of a menu file.

In general, when a procedure is activate, command are being sent to the EASIE command processor from the procedure, and thus are not expecting feedback from a user. It is clear from the above example that construction of an ADE–mode procedure is a nontrivial operation and requires an expert on the EASIE system. Unfortunately, EASIE does not provide the analyst with utilities to create a predefined procedure and associated menu files.

As a final note, EASIE user file directory will contain a large variety of files. Though an explanation of each of these files is attracted, there would generally be little reason for a general user to become involved with any of the details or naming conventions used in these files. Such details can and generally should be left for the EASIE system to monitor. Although most designers prefer to access EASIE via the ADE mode interface, many have voiced disappointment over the lack of a modern interface. In addition to the preceding section makes it clear construction of such an interface is difficult at best. This has led us to identify two major problems with the current EASIE ADE mode interface.

2.4 The Drawbacks of EASIE in ADE mode

- The EASIE system is based alphanumeric interaction.
- Control of the system during ADE mode is governed by command procedure. The construction of such procedures are designed by an experienced analyst.

3. Principles of Interface Design

Most computer users feel that computer systems are unfriendly, uncooperative and that it takes too much time and effort to get something done. They feel dependent on specialists, and they notice that "software is not soft". Users use computers as tools for achieving tasks of particular problem domains such as text processing,financial planning, or computer–aided design. It is too much to ask users to learn about something as complex as a large computer program by direct observation of what the program does. Therefore, the overall goal of the design methodology is to help programmers deliver their designs, not only by reducing the complexity of the delivery process, but also by helping to ensure that the delivered system provides a good interface for users.

The quality of the user interface often determines whether users enjoy or despise a system and ultimately whether the system is even used. The following will describe five principles of interface design [4] [5].

3.1. Put the User in Control

An effective interface allows users to form an accurate and detailed cognitive representation of the structure of the software and to learn quickly how to operate it. A poor interface frustrates and confuses users placing them in constant doubt about where they are in application; it makes users unsure that they can predict how the software will respond to their direction; it creates difficulties in operating the software; and it makes it easy to make errors but not to recover from them.

In order to solve this problem, different interface construction techniques have been proposed:

- Provide online help that informs the user about the structure and operation of the application.
- Provide effective prompts and status messages that guide the user through procedures and keep

them informed about program status.

- Provide error messages that allow the user to understand both what went wrong and how to smoothly recover from the error.
- Provide the user with the means to move freely within and between screens and the ability to move easily to major menu items and to quickly exit from the application.
- Provide consistency in the use of words, formats and procedure.

3.2. Address the User's Level of Skill and Experience

One of the most difficult problem for you as a software developer is overcoming this gap between your skills and the skills of most users. If the application you are developing will be used by people with no computer experience, then your design must favor these users over the more experience ones. In order to solve this problem, different interface construction techniques have been proposed.

- Avoid jargon .

  All computer terms and other technical jargon not familiar to the users must be eliminated from the interface or explained to the users. The design must be subjected to ensure that potential users understand the words contained in menus, messages, help text and tutorials.
- Use appropriate transaction control procedures.

  New users will be most comfortable with menu or simple question–and–answer dialogue. Experienced users can use these methods, but they may want to be able to string together sequences of commands and use function keys to speed up the operation of an application.
- Provide several levels of detail for error and help messages.

  Experienced users need error and help messages to remind them of what they already know. New users, however, need step–by–step procedures and examples that instruct them in the operation of the application. The needs of both these groups can be met by providing more than one level of help and error message.

3.3. Be consistent in wording, formats, and procedures

Consistency is an important feature that should be built into every interface and it should be maintained across applications. Consistency helps the user to learn an application more easily, to use it more easily, and to recover more easily when there is a problem.

3.4. Protect the user from inner working of the hardware and software that is behind the interface

One of the characteristics of a poor interface is that it displays information about the internal workings of the software that the typical end user cannot understand. For example, displaying a message such as "FORTRAN END" may tell you that the software is operating normally, but it may be meaningless to the end user. In addition, many new users are very sensitive about their lack of knowledge of computer hardware and software. As a consequence, they are immediately upset when words and phrases that describe the internal workings of the software are displayed on the screen. A good interface will protect the user from having to know about the inner working of hardware and software tools.

3.5. Minimize the burden on the user's memory

Human beings are poor at recalling detailed information but are remarkably good at recognizing it. A good interface design should minimize the need for the user to memorize and later recall information. Whenever possible, users should be able to choose from lists and be allowed to use their recognition memory rather than their recall memory. Here different interface construction techniques have been proposed.

- Be Consistency in your use of words, formats, and procedures. Consistency reduces the user's need to learn and remember new information.
- Display status messages that remind users where they are in an application and what options are in an application and what operations are in effect.
- Provide online help that is designed as an aid to memory.
- Use memory joggers in prompts and data entry captions. For example, tell users how to format dates, such as (mm/dd/yy).

4. Modification of EASIE in ADE mode

4.1 Window system : OSF's MOTIF

Almost all modern user interface are window–based. Windows allow the user to interact with multiple source of information at the same time. Window techniques allow a relatively rapid access to more information than is possible with a single frame of the same screen size. The window system provides many of important features of the modern interface, for example, applications that show results in different area of display, the ability to resize the screen areas in which those applications are executing, pop–up and pull–down menus and dialog boxes.

Currently, the EASIE system is based alphanumeric interaction techniques only. The goal in the design of any menu should be to facilitate the user's ability to make a choice quickly and accurately. It is the purpose of this project to extend the flexibility of EASIE system in the ADE mode by implementing it in a window system. The user–interface, with its windows and pulldown menus, is popular because it is easy to learn and requires little typing skill. The windowing system chosen to implement EASIE is OSF/MOTIF. What follows is a brief description of Motif [2] [3].

OSF/Motif is a graphical user–interface toolkit, window manager, style guide, and user–interface language. Motif's graphical interface is based on the X window system from MIT. This underlying technology provides you with a network–based graphical user interface. Motif is composed of a style guide, window manager, interface toolkit and presentation description language.

- **Style guide**

  The style guide describes a standard behavior and a set of connections for applications, to ensure a consistent feel on multiple applications. The style guide includes extensions for powerful network–based workstation. Its "look" is based on the HP–three dimensional screen–button appearance.

- **Window manager**

  The window manager lets you manipulate multiple applications on the screen and plays a principle role in enforcing the style guide.

- **Interface toolkit**

  The OSF/Motif toolkit is based on the X windows intrinsics, a toolkit framework provided with MIT's X window system. The intrinsics use an object–oriented model to create graphical objects known as widgets or gadgets. The specified widgets maintain consistency between applications.

- **Presentation description language**

  This language enables application developers to describe the presentation characteristics of the application interface independent of the actual application code. The separation between application and interface lets you make many changes to the overall appearance and layout of an application without having to modify, recompiler, or relink the application itself.

4.2 Design Considerations for ADE Facilitator

Menus displayed during ADE mode are typically created by experienced engineers to guide other designers in the proper sequence of steps to conduct some particular design activity. In this project, in addition to implementing ADE mode in a window system, a set of utilities are developed to assist the experienced engineer in the generation of an ADE application. It is assumed that an experienced engineer has sufficient knowledge of the desired application to develop an organized approach to use of that application. The ADE facilitator has been developed to capture this information in a way that automatically includes a number of good interface design principles. Thus we have designed the ADE facilitator to overcome the problem listed in section 2.4. In addition, the ADE facilitator provides the engineers a simple environment to generate the ADE application easily. Experienced engineers are not required to have any knowledge of principles of interface design or OSF/MOTIF

. They make use of the ADE facilitator to build up an application–dependent hierarchy menu in any desired format.

In order to develop ADE facilitator as a good interface, there are a lot of issues we considered.

- **The layout of menus**

When the user interface makes use of graphical objects such as window and menus, it is called a graphical user interface(GUI). Compared with the nongraphical application, interactive graphicals makes menu selection such simpler and faster. The menu is displayed on the screen, the user points to a selection with a graphical input device, like mouse. This menu can facilitate the user's ability to make a choice quickly and accurately.

In application programs with commands or many different operands, the size and complexity of the interface can become a serious problem. A simple solution is to use a multilevel menu. With a hierarchical menu, the user first selects from the choices set at the top of the hierarchy, which causes a second choice set to be available. The process is repeated until a leaf node of the hierarchy tree is selected. Since the ADE facilitator captures the organization of an experienced engineer, a natural task decomposition is obtained.

- **Feedback**

Feedback is as essential in conversation with a computer. A selected objected or menu command is highlighted, so the user can know that action has been accepted. In this project, when the application designer travels the menus being built, information about the current level of menu hierarchy is displayed in a list window. This list window provides good feedback for the application designer. In addition, the full feedback facilities of OSF/MOTIF are automatically provided for the final ADE application.

- **Error Recovery**

A poorly design interface gives the user no choice but to proceed with the command. A well–designed interface lets the user back out of such situation with a cancel command. With good error recovery, the user is free to explore unlearned system facilities without " fear of failure ". In a less serious type of error, the user may want to correct one of units of information needed for a command. The dialogue style in use determines how easy to make such corrections are. Command–language input can be corrected by multiple backspaces to the item in error, followed by reentry of the corrected information and all the information that was deleted. This project provides these capability automatically through its OSF/MOTIF interface.

- **Be Consistent**

Consistency reduces the user's need to learn and remember new information. For example, when the select procedure is used on all menu, a user has to learn it only once and it is easier to remember. In this project, we provide the capability for selecting menu by pushing first button of the mouse, and pointing a special item before doing insertion with second button of the mouse. Again these are capabilities provided automatically through the OSF/MOTIF interface.

- **Provide online documentation to help the user to understand how to operate the application**

In this project, we have provided a help menu that gives the user a brief overview of how to use this application.

4.3 Demonstration of the ADE Facilitator

In windows 1~28, we demonstrate the use of the ADE facilitator. These windows also illustrate those characteristics we mentioned above that lead to a well designed interface. For this example we develop an interface for the sample problem stated in section 2.2. This interface is developed with the following steps.

Window 1 is an original ADE facilitator, there is no menus with it.

Step 1. Push the "Add menubar Item" button.

Step 2. Type in the name of new menubar item.

Step 3. Click on the "ok" button.

Windows 2 ~7 show the procedure of creating new items on the menubar using step1~step3 recursively.

Step 4. Point the pulldown menu of a menubar item using first button of the mouse.

Step 5. Push the "Add Item with subItem" button.

Step 6. Type in the name of a new item.

Step 7. Click on the "ok" button.

Windows 8~11 show the procedure of creating a new cascading item using step 4~step 7.

Step 8. Point the pullright menu of a branch item.

Step 9. Push "Add Menu Item" button.

Step 10. Click on the "ok" button.

Step 11. Type in the EASIE command.

Step 12. Click the "ok" button.

Windows 12~14 show the procedure of creating a new leaf item using step 8 ~ step 12.

Window 15 is the resulting of built menu of first item at the menubar.

Step 15. Select an item on list window using the first button of the mouse.

Step 16. Release the button.

Windows 16 ~17 show the procedure of deleting an item from the menu using step 15 ~ step 16.

Window 18 ~23 show the procedure of building the menus of second item at the menubar.

Window 24 shows the menu of third item at the menubar.

Step 17. Push "Delete Menubar Item" button.

Step 18. Type in the name of the item existing on the menubar.

Step 19. Click on the "ok" button.

Windows 25 ~26 show the procedure of deleting a menubar item.

Window 27 is the dialog box for "save" button.

Window 28 is the dialog box for "Exit" button.

Windows 29 ~34 show the EASIE user interface built by ADE facilitator. This user interface helps the EASIE user make a choice quickly and accurately. When a leaf node of a hierarchy menu is selected, a special command will be showed up on the list window and being sent to the EASIE command processor at the same time.

5. The general Structure of the Solution

As seen in the previous chapter, presentation of an ADE facilitator menu is best carried out in a hierarchical manner. This hierarchy is easily described by the tree data structure shown below.

5.1 Data Structure

```
typedef structure menu {
    structure info data;
    structure menu *sub_menu;
    structure menu *next;
  } *node;
```

Since the purpose of the ADE facilitator and the creation of an ADE application, the menu structure cannot be known ahead of time and therefore must be dynamic. The usual approach to declare a space large enough to hold the maximum amount of data we could logically expect cannot work. Thus tree components are created only as they are needed. Each component contains information about the location the next components. Such a tree can expand or contract as the ADE facilitator is executed and we use this dynamic data structure to hold the structure of the newly created menus.

In general, each node of this structure contain information related to the menu choice it represents as well as two information of locations. The first location is that of next menu item at same level of hierarchy, and the other is the location of the first choice for a submenu item.

Figure 4 demonstrates the structure of menus.

5.2 Mechanisms

We provide two operations on the data structure mentioned above.

(i) Add an item to the menu.

(ii) Delete an item from the menu.

Each item has its own ID. We use a binary expression to represent the location of an item or subitem in the hierarchy of the menu to which it belongs.

```
ID = 0 0 0 0 1 | 0 0 0 0 1 = 33
          |         |
   second level   first level
```

figure 5.

The first five bits from right side stand for the first level of hierarchy in the menu, the second five bits stand for the second level of hierarchy and so on. For example, consider the ID number 33 in figure 5. This binary expression represents the first subitem of the first item at the first level of hierarchy in the menu. Thus the current implementation use a five bits to stand for each level, and therefore there are 32 items at most for each level of hierarchy in the menu.

We use bitwise operators to deal with the problem from adding or deleting a menu or submenu item. Except for the ID of any item at the first level of hierarchy menu, simple addition or subtraction operations on the ID are not sufficient to find the ID of the next submenu item.

```
ID1 = 0 0 0 0 1 | 0 0 0 0 1 = 33
ID2 = 0 0 0 1 0 | 0 0 0 0 1 = 65
```

figure 6 .

In figure 6 ID1 represents the first submenu item of first item located at the first level of hierarchy. ID2 represents the second submenu item of first item located at the first level of hierarchy. Using the bitwise operators, we can easily realize the relationship between ID1 and ID2 as follows.

```
num = ( ID1 & 01740 ) >> 5;
num ++ ;
numtemp = ( ID1 | 01740 ) ;
num = ~ ( ( ~ num << 5 );
ID2 = num & numtemp;
```

Based upon this encoding the menu hierarchy can easily be stored in file format. The resulting menu tree needs to be stored for both later editing or using in an ADE session by a design engineer. Currently we differentiate leaf nodes in the tree from branch nodes as follows.

(i) Format for an item with submenu (branch node) :

```
ID   name   :
```

(ii) Format for an item without subitem (leaf node) :

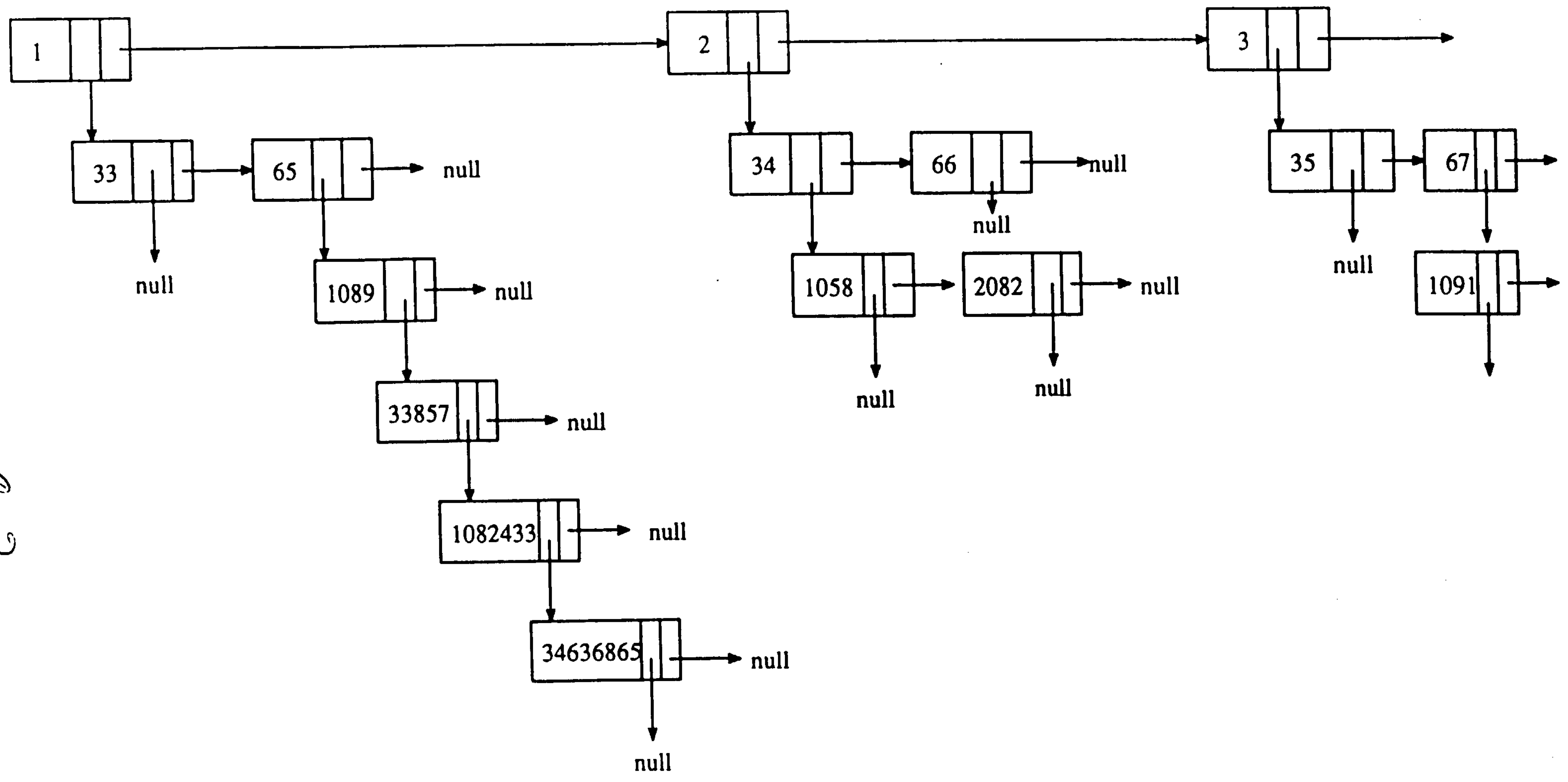


Figure 4. A structure of menu tree.

ID name

command name

Of course, design engineers do not need to have this knowledge of the format of a file, it is handled automatically for them. When they develop an application–dependent menu, the ADE facilitator is going to help them store the menu tree.

5.3 Capabilities and Limitations of ADE facilitator

We note the following capabilities of designed into the ADE mode facilitator.

- Add an item with submenu into the pulldown or pullright menu at any special position.
- Add an item without submenu into the pulldown or pullright menu at any special position.
- Delete an item with submenu from the pulldown or pullright menu.
- Delete an item without submenu from the pulldowm or pullright menu.
- Add an item into the menubar at any special position.
- Delete an item from the menubar.

We also note two cautions for the current implementation.

- We can create six levels of menus at most.
- Each level of menu could have 32 items at most.

6. Conclusions

In this project, we used OSF/MOTIF toolkit based on the X window system to implement new ADE mode. In addition, we designed an interface with facilities to help the design manager easily build the application–dependent menu, called the ADE facilitator. With this, an EASIE design manager can quickly develop an application–dependent menu to any desired format, and the EASIE user can make a choice quickly and accurately.

7. References

1. James L. Schwing, Lawrence F. Rowell E. Criste, The Environment for Application Software Integration and Execution (EASIE), Volume III, NASA Technical Memoradum, April 1988.
2. Douglas A. Young, The X Window System Programming and Application with Xt, OSF/MOTIF edition, Prentice Hall.
3. Dan Heller, Motif Programming Manual, O'Reilly & Associates , Inc, 1991.
4. William M. Newman, Robert F. Sproull, Principles of Interactive Computer Graphics, second edition, Mcgraw–Hill Book Company, 1979.
5. Joseph S. Dumas, Designing User Interfaces for Software, Prentice Hall, 1988.

IMPLEMENTATION OF AN ALGORITHM FOR DATA REDUCTION USING CUBIC-RATIONAL B-SPLINES

Computer Science 698 - Master's Project

Caroline E. Macri

10 December 1992

Acknowledgments

I would like thank several people for their contributions toward this project

- Dr. James Schwing for providing the idea for the topic of this project and for helping me to understand the area of curve and surface design.
- The employees of the NASA Langley Vehicle Analysis Branch for the generous use of their books, software and time.
- My husband Steve for his emotional and financial support

IMPLEMENTATION OF AN ALGORITHM FOR DATA REDUCTION USING CUBIC RATIONAL B-SPLINES

ABSTRACT

A master's project implementing and evaluating a data reduction algorithm using cubic rational B-splines is presented. The method emphasizes the geometric characteristics of curve construction. A general overview of the mathematics of Bezier and B-spline curve representations is presented. The algorithm itself is then briefly summarized. The large library of geometric operations developed in support of the implementation are described. Some of the more important geometric issues encountered - including ambiguity, data validity, and the meaning of error in the context of this fitting method are discussed. Finally, an evaluation of the data reduction obtained with the final program, using several representative data sets, is presented.

INTRODUCTION

The purpose of this master's project was to implement and evaluate the data reduction algorithm outlined by Chou and Piegl [CHOU92]. A copy of their paper is provided in Appendix A. Compared to other curve fitting schemes, this method "provide[s] a sufficiently clear insight into the geometric characteristics of curve construction." The authors' describe the general technique, but not the details required to develop a correct, robust program. Much of this project was spent investigating these details. The result was in an even greater depth of understanding about curve geometry. These insights constitute the bulk of this report.

First, however, curve fitting and its relevance to a significant class of practical problems is explained. Some of the mathematics of Bezier curves and B-splines are detailed since both are used in the Chou/Piegl algorithm. Next, the algorithm itself is briefly summarized. The large library of geometric operations which were developed in support of the implementation are described. Some of the more important geometric issues encountered - including ambiguity, data validity, and the meaning of error in the context of this fitting method are documented. Finally, an evaluation of the data reduction attained with the final program, using several representative data sets, is presented.

CURVE FITTING

Curve fitting and surface design constitute a very general class of problems with broad application to engineering. There are many methods to model objects for engineering analysis. The 'best' curve or surface with which to do this is a function of the analysis to be performed as well as a compromise based on available computer resources. For the most part, curve and surface modeling require extensive use of graphics, both to verify the accuracy of the model and for proper visualization and interpretation of analysis results.

Most of this work was performed under a research grant at NASA Langley's Vehicle Analysis Branch (VAB). At VAB, engineers and computer scientists are currently applying surface fitting methodologies to aerospace vehicle design in a system called Solid Modeling Aerospace Research Tool (SMART). SMART allows a user to interactively create a model made of surface patches. It has automated the manipulations necessary to convert this model into descriptions suitable for use by pre-existing analysis packages. Surfaces in SMART are represented using Bezier patches (see Theoretical Background) however some consideration is being given to other techniques. The Chou/Piegl algorithm is one such alternative.

THEORETICAL BACKGROUND

Curve fitting is the process of defining a set of piecewise polynomials to approximate some set of discrete data points. A polynomial representation is compact - one segment can approximate several data points and interpolate infinitely many others - and provides the functional description required for many types of engineering analysis.

Cubic parametric polynomials are quite prevalent in curve fitting applications. Cubics are preferred over lower order polynomials because they permit slope continuity and interpolation at endpoints or, alternatively, they can be manipulated to achieve second derivative continuity in a relatively simple fashion. Practical experience has shown that higher order polynomials tend to oscillate undesirably between interpolation conditions [KREY79].

In the context of curve representation, parametric means the location of a point $\{x,y,z\}$ is defined as a function of an additional variable, called a parameter. Typically, such a variable is denoted by t and is specified over a closed, real interval. The segments, or individual pieces which are combined to form a curve, are each described parametrically as $\{X(t),Y(t),Z(t)\}$. This eliminates interdependencies between x, y and z and provides great flexibility in the types of curves which can be represented. Any non-parametric form can be easily converted to parametric notation. The computational stability of calculations involving parametric curves can be directly influenced by the choice of the parameter variable's interval. A common interval is $0 \leq t \leq 1$.

It is not probable that a single cubic curve will accurately approximate a set of data points. In a piecewise representation, cubics are fitted through subsets of the data. These segments must meet some sort of interpolation conditions at their end points in order to join smoothly with adjacent segments. An n^{th}-degree polynomial has n+1 coefficients which can be determined by solving n+1 simultaneous equations. In practice, this means n+1 interpolation conditions for a segment are required to calculate its polynomial form. This data can be n+1 points, two endpoint slopes and n-1 points, or various combinations of points, slopes and higher derivatives.

Parameterization has a subtle effect on the concept of continuity between segments. The result is a distinction between derivative (C^n) and geometric (G^n) continuity [FOLE90]. C^n means that all derivatives with respect to t, up to and including the n^{th} derivative, are equal between segments. Geometric continuity is more clearly demonstrated by visualizing a curve segment's end point tangent vector. Two segments are G^1 continuous if their tangent vectors are scalar multiples where they join. The parameter, t, is not shown and the resulting curve plotted in x, y, and z exhibits no change in slope between segments. C^n implies G^n except for the case where the n^{th} derivative vanishes. Geometric continuity is often less computationally expensive than derivative continuity. It is used in applications where visually pleasing results are the major requirement. Many engineering analyses are based on an underlying parametric model which demands derivative continuity.

There are many ways to describe a cubic polynomial segment. One method, which will be utilized in this project, was developed by Pierre Bezier [FARI88]. It uses four known points, $P_0 .. P_3$, to calculate cubic polynomial coefficients. A Bezier segment interpolates P_0 and P_3. P_1 and P_2 are called control points because they define the direction and magnitude of the endpoint tangents but are not themselves interpolated. Since the endpoint slopes are defined, it is possible to construct a piecewise Bezier curve with C^1 continuity. A cubic Bezier parameterized on $0 < t < 1$ has the form

$$C(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3.$$

The coefficients of the four points are known as the Bernstein polynomials and sum to one for any value of t. Each point on a Bezier segment is, in fact, a weighted average of its defining points. This results in a very useful property, the convex hull property. A Bezier curve is completely enclosed by the shape formed by stretching a surface around its control points (Figure 1.)

Rational Bezier representations introduce weighting factors which fine tune the impact each control point has on the curve. The cubic form is

$$C(t) = \frac{(1-t)^3 w_0 P_0 + 3t(1-t)^2 w_1 P_1 + 3t^2(1-t) w_2 P_2 + t^3 w_3 P_3}{(1-t)^3 w_0 + 3t(1-t)^2 w_1 + 3t^2(1-t) w_2 + t^3 w_3}$$

Again, it is clear that each point on the curve is a weighted average of the control points. The weight magnitudes affect the relative influence of each control point as demonstrated in Figure 2.

Rational cubic Beziers satisfy the shape invariance property [PIEG87]. Providing the two shape invariance factors

$$S1 = \frac{w_0 w_2}{w_1^2} \qquad S2 = \frac{w_1 w_3}{w_2^2}$$

remain constant, the weights can take on any value and still produce the same curve. As a simple example, consider the case where all points are equally weighted. Regardless of the magnitude, all versions are equivalent. The standard representation for rational Beziers takes advantage of shape invariance factors and sets endpoint weights (w_0, w_3) to 1.

Cubic B-spline representations are also used in this project. A series of m control points, parameterized over a set of m+3 knots, is used to define a set of m-3 segments with C^2 continuity. Any particular segment is influenced by, but does not necessarily interpolate, four consecutive control points. Conversely, a control point impacts only four adjacent segments, effectively localizing its influence. The flexibility of the B-spline representation can be increased by duplicating knots. This forces point interpolation at the expense of geometric continuity. The form of a cubic B-spline [PIEG87] is

$$C(t) = \sum_{i=1}^{n-3} N_{i,3}(t)P_i \qquad 0 \leq t \leq 1$$

where

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \text{ and } t_i < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and for $p \geq 1$

$$N_{i,p}(t) = \frac{t-t_i}{t_{i+p}-t_i} * N_{i,p-1}(t) + \frac{t_{i+p+1}-t}{t_{i+p+1}-t_{i+1}} * N_{i+1,p-1}(t) .$$

Like rational Beziers, rational cubic B-splines introduce weighting parameters to fine tune the impact each control point has on each segment's shape. This improves a user's ability to manipulate the curve. Its form

$$C(t) = \sum_{i=1}^{n-3} R_{i,3}(t)P_i$$

where

$$R_{i,p}(t) = \frac{N_{i,p}(t)w_i}{\sum_{j=1}^{n-3} N_{j,p}(t)w_j}$$

is similar to that of a cubic B-spline with the addition of the weighting factors [PIEG87].

OVERVIEW OF ALGORITHM

The Chou/Piegl algorithm fits a rational Bezier curve, within a specified tolerance, to a set of points and end tangents. More specifically, using the end conditions, the control points P_1 and P_2 of an interpolating non-rational Bezier are determined for each interior point. These will vary and must be averaged to produce a single approximating Bezier.

Next, the curve is converted to a rational form with endpoint weights of 1. For every interior point, interpolating weights based on the average P_1 and P_2 are calculated. The variation of the weights indicates the goodness of fit. If most interior points are close to the average curve, the weights will be similar. The opposite is true when the points vary greatly in their distance from the approximating curve.

In the cubic form, the calculated w_1 and w_2 both contribute to forcing the rational form through a specific point. This coupling makes it difficult to determine variation solely due to the w_1's or to the w_2's. Using DeCasteljau's algorithm, it is possible to decompose the cubic into two rational quadratics and thereby isolate the weights.

For every point, tolerance criteria is converted to an acceptable range of weights for the quadratics. A fit with acceptable error is achieved when there is a non-empty intersection of the weight ranges. 'Average' weights are chosen from this intersection and are used to combine the quadratics into a cubic.

It is unlikely that one curve will adequately fit a large set of points. When tolerance is not achieved, the curve must be recursively subdivided into segments and refit. To preserve G^1 continuity, the first tangent direction of a segment is set equal to that of the last tangent for the preceding segment. The final G^1 representation merely joins the Bezier's with triple knots and reparameterizes based on chord length.

A C^1 representation takes advantage of cubic shape invariance factors. The weights can be adusted to produce equal tangents between adjacent segments. Though still a piecewise Bezier, the curve is a C^1 parameterization in cubic B-spline form.

GEOMETRIC OPERATIONS

This algorithm emphasizes the geometric aspects of curve fitting. It was necessary to develop a large library of supporting point, vector, line and plane operations for this implementation. Complete documentation is located in the program listing [Appendix B]. During this process, parametric representation proved to be a very powerful and useful tool when dealing with lines in three space.

Consider a point, vector representation of a line, (P_0,V), where each is a 3x1 matrix with values for x, y and z. Given two lines in this form, it is not readily apparent how to determine an intersection. Now, consider the parametric form for the same lines (Figure 3.)

$$P_1 + V_1*t = P \text{ and}$$

$$P_2 + V_2*s = P.$$

These must be equal at the intersection - that is

$$P_1 + V_1*t = P_2 + V_2*s.$$

Picking any two of the x,y and z components yields two equations and two unknowns

$$P_1x + V_1x*t = P_2x + V_2x*s$$

$$P_1y + V_1y*t = P_2y + V_2y*s$$

which can be readily solved.

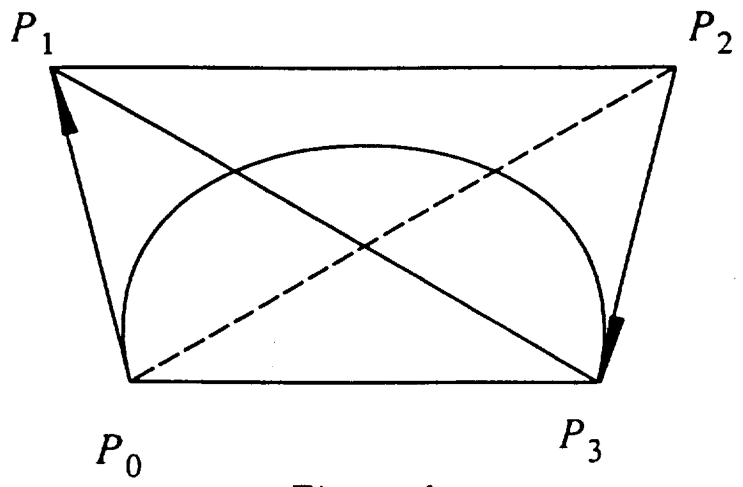


Figure 1.

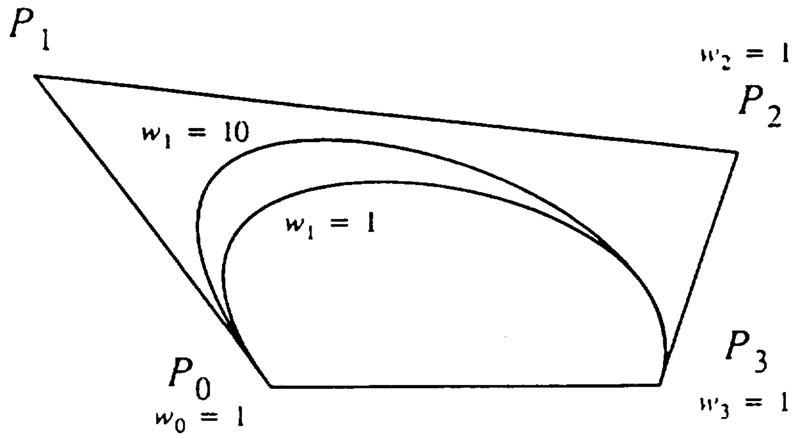


Figure 2.

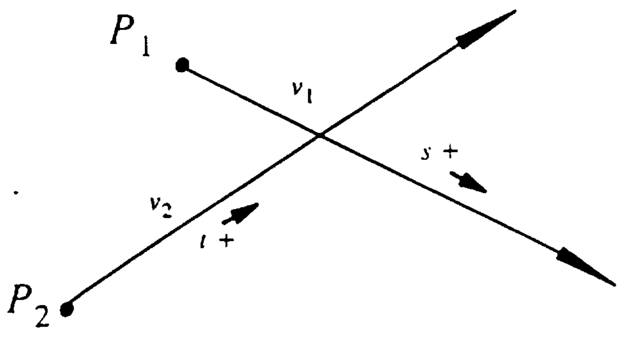


Figure 3.

Once s and t are known, it is simple matter to determine the Cartesian coordinates. Often, though, the parameter value is more useful in subsequent operations. A planar situation, where a determination of P's containment in the 'cup' defined by P_0P_3, T_0 and T_1 must be made, is shown in Figure 4. First, a line defined by P and a vector in the direction of P_0P_3 is constructed and parameterized on s. Sides are created from P_0 and T_0, and P_3 and T_1 and are parameterized on t. For containment, P's line must intersect one side with positive s and the other with negative s. If it intersects both with negative s, P is to the 'right' of the cup and if both are positive, it is to the 'left'. It is also necessary to decide if P is above P_0P_3. For this, construct P_0P_3, paramaterize PT_1 on t and intersect the two lines. $T < 0$ indicates containment while an $t > 0$ would mean P is below. Throughout the project, many situations were encountered which could be simply and directly handled with the parametric representation.

END TANGENT CALCULATIONS

The program accepts a set of data points as input. It must therefore make some assumptions about endpoint tangents. The case shown in Figure 5. subdivided the data set into three segments and requires tangent estimates at points 0, 4, 8, and 11. Farin [FARI88] details several methods for tangent generation. A Bessel function, which calculates the slope for a quadratic passing through the point and it's two closest neighbors, was chosen for this

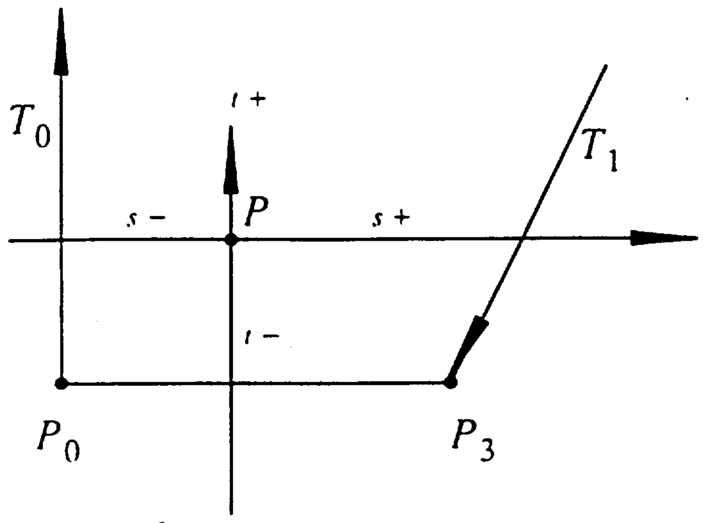


Figure 4.

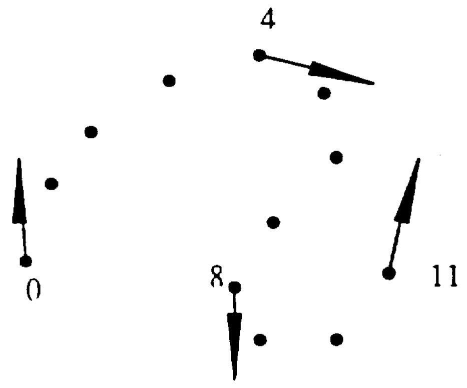


Figure 5.

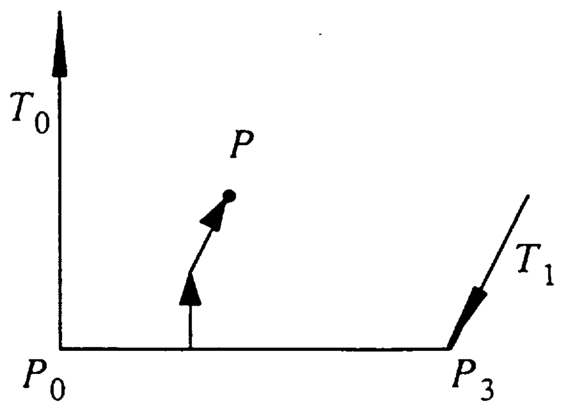


Figure 6.

implementation. This is not necessarily the best option. For greater flexibility, the user could be given a choice of methods and the ability to specify directions at the first and last data points.

CONTAINMENT

One important discovery, made while testing the program, is that some sets of points cannot be represented with a Bezier curve. A data set with planar tangent vectors illustrates the point. The convex hull property forces the curve to lie in the plane defined by the tangents, therefore out-of-plane interior points cannot be interpolated. This forced a closer examination of what exactly constitutes a valid set of data.

The first portion of the Piegl and Chou paper shows that any point of a non-rational Bezier curve is actually the result of calculating a point on P_0P_3 at some t, and traversing from this point for some distance in the direction of T_0, and then for some distance in the direction of $-T_1$ (Figure 6.) The exact distances travelled depend on the positions of P_1 and P_2. This operation can be reversed to determine the P_1 and P_2 locations and t value for an arbitrary point. Tangent length is undefined at the beginning of this process, so the valid point space must allow a convex hull with P_1 and P_2 infinitely far from P_0P_3 (Figure 7.) This results in two semi-infinite strips of width $|P_0P_3|$, emanating from P_0P_3 and extending out to infinity in the direction of T_0 and T_1 respectively. A

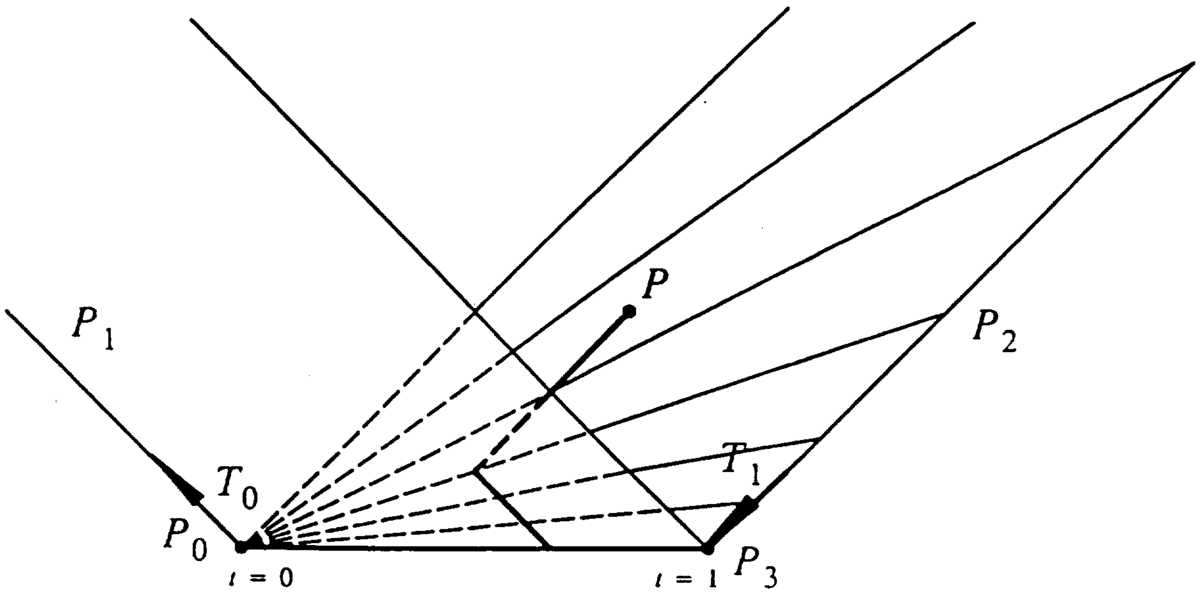


Figure 7:

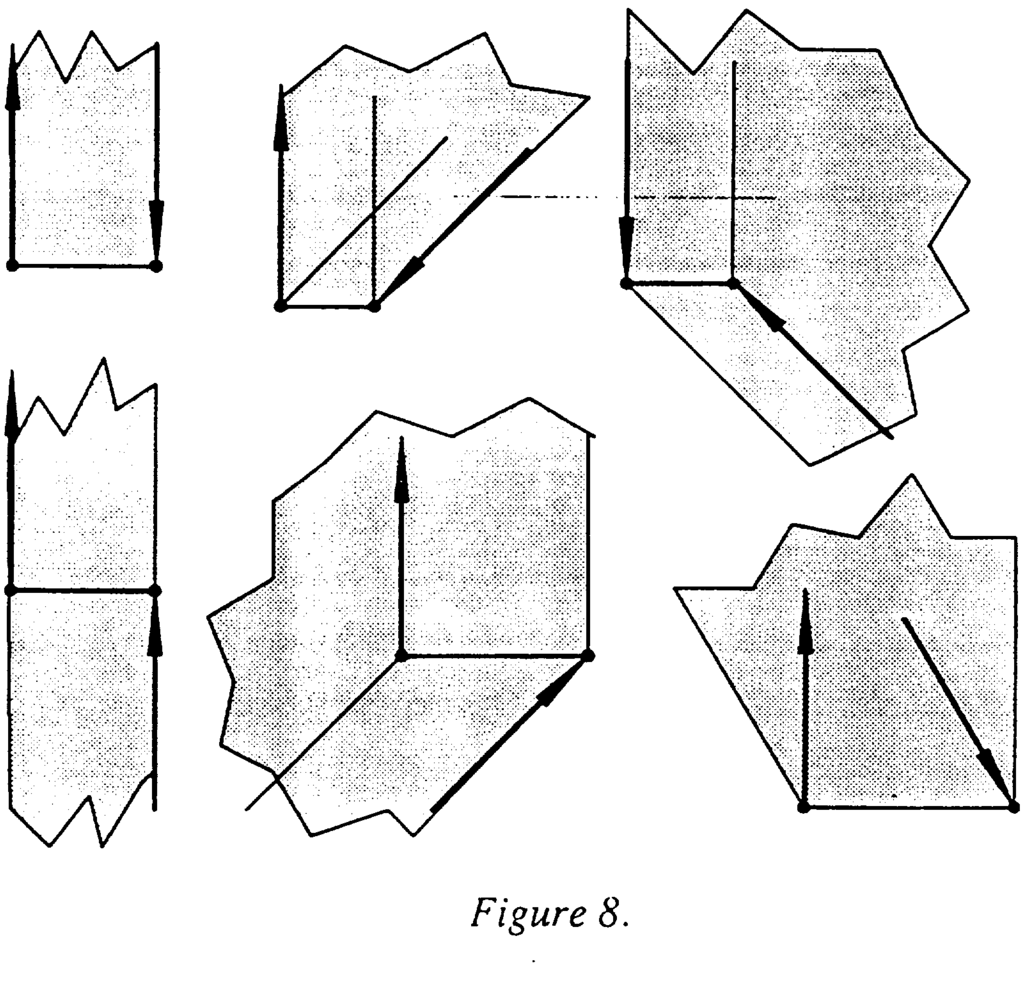


Figure 8.

point must be able to project onto P_0P_3 via T_1 and $-T_0$. For example, first project point P onto the $P_0P_3T_0$ plane along the direction T_1, and then to the segment P_0P_3 along the direction $-T_0$. The valid state space is a wedge bounded by the two strips.

In the planar case, the analysis is slightly different. The wedge collapses onto a subsection of the plane. Figure 8. illustrates the shape of this area as the tangents change their relative positions. Containment can be verified with the parametric line operations described in the Geometric Operations section. This analysis provides some useful insight into curve behavior, especially for users of systems which allow interactive manipulations of Beziers.
For the case where tangents are collinear, this implementation considers only those points lying on the line segment P_0P_3 to be valid.

A second containment check is required prior to calculating weights for a rational Bezier. Figure 9. illustrates how the averaging process for P1 and P2 may cause some points to violate the convex hull property.
This can be detected in the 3-dimensional case using the scalar triple product for vectors ($a \cdot (b \times c)$). The physical interpretation of the triple product is six times the volume defined by the three vectors (Figure 10.) The sign of this volume depends on the order (left- or righthand) of the operation. Any point can be combined with each of the four faces of the tetrahedron. The four volumes

generated from the scalar triple products, if performed in a consistent hand, will sum to the volume of the original convex hull. Interior points will produce individual volumes whose signs are consistent with the total volume. Some of the volumes generated using an exterior point will have the opposite sign.

Again, the planar case was somewhat more difficult to handle. Checking for containment in a four-sided polygon had to be reduced to checking for containment in its four constituent triangles. This provided correct detection even in some of the degenerate planar convex hull shapes such at that shown in Figure 11. As long as the point is inside one triangle, it is in the convex hull. Inclusion in a triangle is easily verified using parametric line representation. Consider Figure 12. If PP_0 is parameterized on s and P_1P_2 on t, the intersection must satisfy $s < 0$ and $0 < t < 1$ for containment.

A set of points which violates containment will cause that set to be subdivided. As seen in Figure 13., where B cannot be contained between A and C, a set of three points may be invalid. It is subdivided into two data sets with two points each. However, two points do not provide sufficient information to locate P_1 and P_2 so some assumptions must be made. In this case, the algorithm positions P_1 and P_2 at $.4*|P_0P_3|$ to avoid overly long or crossing tangents (Figure 13.) Farin [FARI88] describes several other methods.

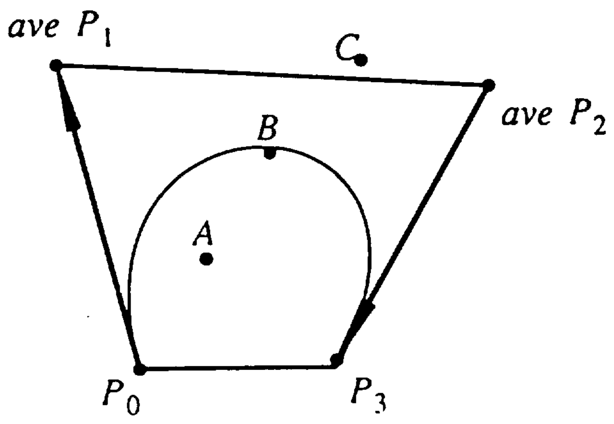


Figure 9.

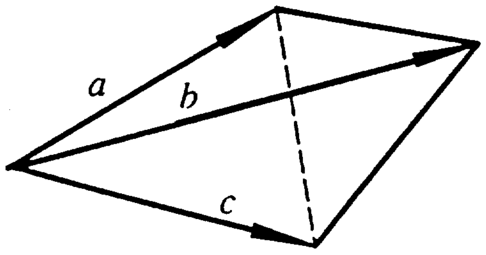


Figure 10.

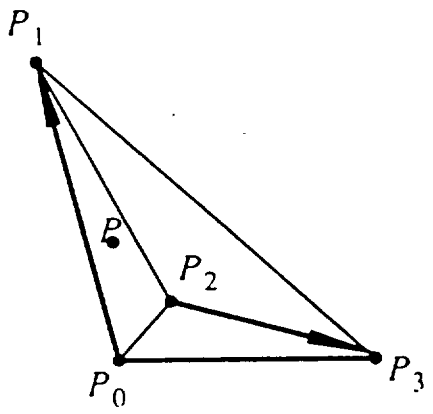


Figure 11.

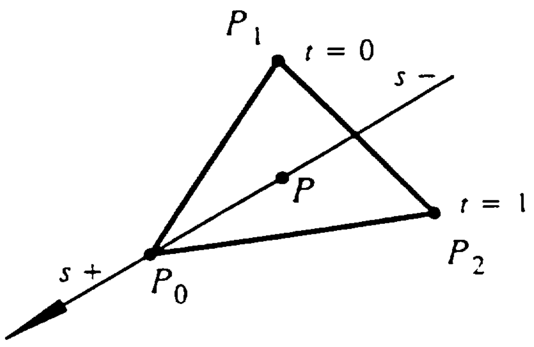


Figure 12.

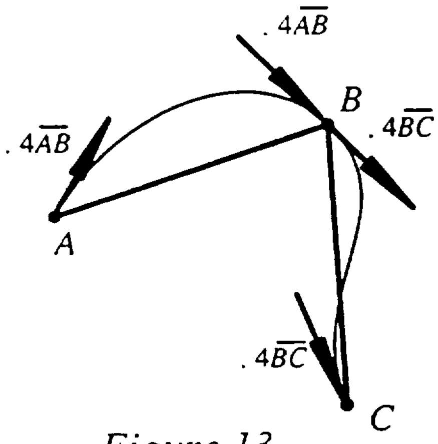


Figure 13.

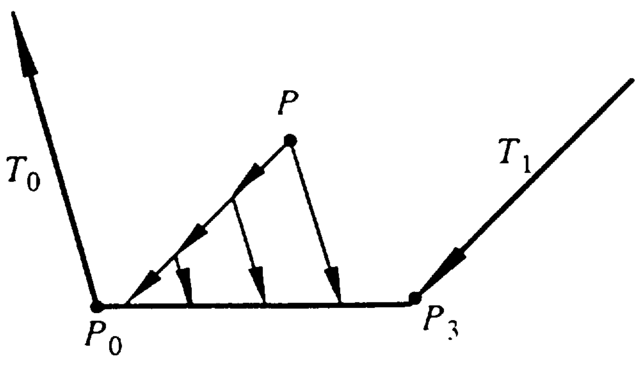


Figure 14.

PLANAR AMBIGUITIES

Figure 7. depicts the process used to determine tangent lengths and the value of t for a point P. For the three dimensional Bezier curve, this is a unique solution. The planar situation is shown in Figure 14. There are several paths from P along the directions of $-T_0$ and T_1, each landing at different locations along P_0P_3. The extreme cases have a zero length component for one direction. They define the bounding values for valid t along P_0P_3. In some instances, this range extends outside the interval $0 < t < 1$ which is also necessary for a valid representation. Chou and Piegl recommend using a chord length parameterization:

$$t = \frac{|P_0P|}{|P_0P| + |PP_3|}$$

but it was found that this could generate erroneous values for t. This implementation, instead, calculates the acceptable range and places t at the midpoint, guaranteeing valid t and preventing zero length tangents.

A similar problem was encountered when calculating weights for rational Bezier (Figure 15.) In the three dimensional case, M is uniquely identified by the intersection of P_0P_3 with the plane PP_1P_2. This construct is not possible in the planar case, however, P must be barycentric with respect to (that is, within the triangle defined by) P_1, M and P_2. An acceptable range for t can be defined by shifting M along P_0P_3. These bounds do not necessarily correspond those determined for the non-rational

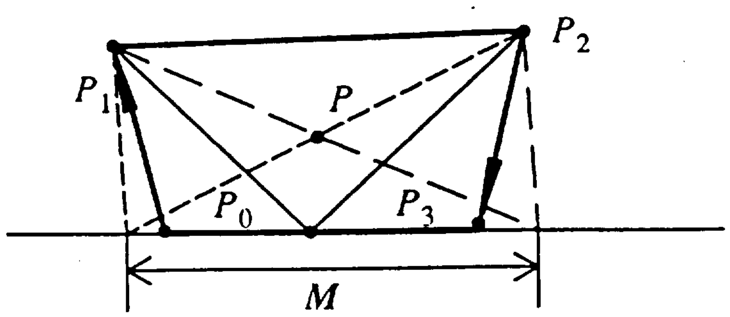


Figure 15.

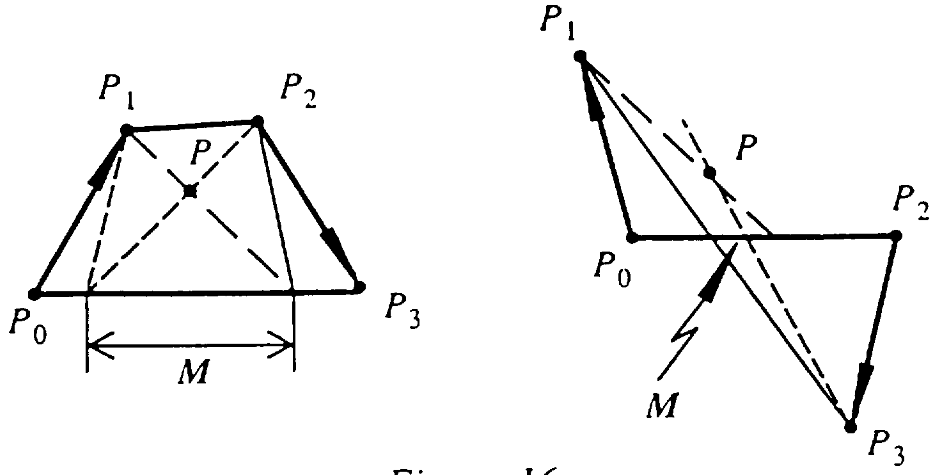


Figure 16.

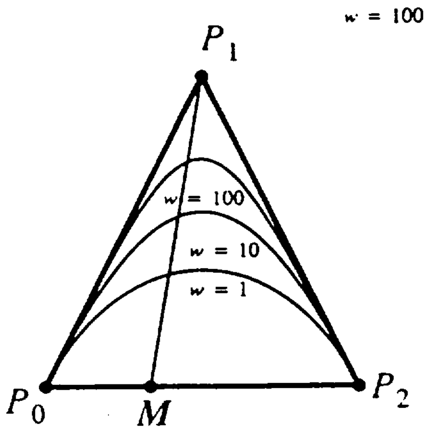


Figure 17.

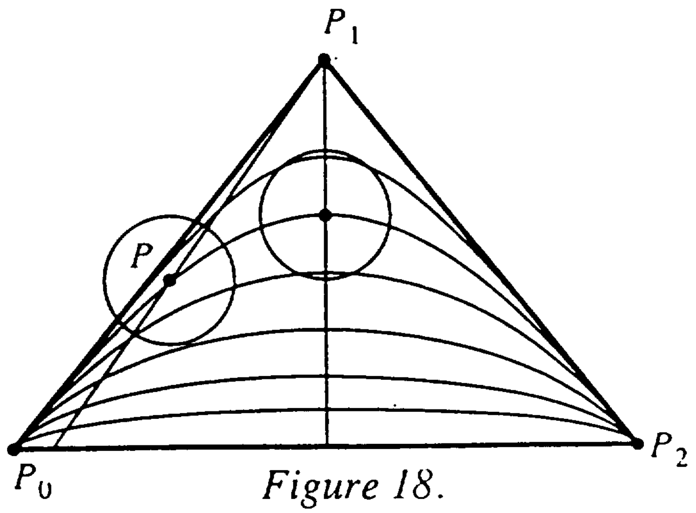


Figure 18.

Bezier. Again, this program sets t to the midpoint of the acceptable range.

In the collinear case, w_1 and w_2 are set to zero thereby eliminating any influence of P_1 and P_2 on the shape of the curve.

Handling these ambiguities turned out to be rather involved. Figure 16. demonstrates how different lengths and orientations for P_0P_1 and P_2P_3 require subtly different interpretations for valid t ranges. In fact, the majority of time for this project was devoted to understanding these relationships to guarantee proper special case handling. Planar and linear curves are frequently encountered in aerodynamic vehicle design, and an acceptable implementation must correctly deal with them.

TOLERANCE CHECKING

A tolerance check is performed only after a set of data points has met the containment criteria. In their paper, Chou and Piegl detail the shoulder point method of error control, noting that it provides a very tight fit, especially for points near the ends of a segment. They briefly mention an epsilon disk method and observe that it involves more work, but results in fewer segments. This program allows either option.

Some interesting properties were discovered during analysis of the epsilon disk method. A rational quadratic Bezier has the standard form

$$C(t) = \frac{(1-t)^2P_0 + 2t(1-t)w_1P_1 + t_2P_2}{(1-t)^2w_0 + 2t(1-t)w_1 + t^2w_2}$$

For a constant t, this can be restated in the form

$$C(t) = M(1-v) + P_1v$$

where

$$M = \frac{(1-t)^2P_0 + t^2P_2}{(1-t)^2 + t^2} \text{ and } v = \frac{2(1-t)tw_1}{(1-t)^2 + 2w_1(1-t)t + t^2}$$

Figure 17 illustrates the implications. A straight line connecting a point on P_0P_2 to P_1 corresponds to a constant t and the distance along that line (represented by v) corresponds to the weight of the intersecting curve. This realization of the epsilon disk method does not necessarily find the maximum and minimum weights, but rather exploits the simplicity of the above relations to approximate a weight range. First, it calculates points at the user specified distance (epsilon) from P along a line parallel to P_0P_2. The weights at these locations define the allowable weight range. Figure 18 demonstrates why the epsilon disk method is less restrictive than the shoulder point method. The latter technique measures tolerance at t = .5 where the distance between curves of differing weights is greatest.

The usual notion of tolerance is that a point falls within some specified distance of the approximating curve. This algorithm checks fit while the curve is decomposed into conics, so it is not apparent if the cubic satisfies the common interpretation of tolerance. To obtain a better feel for the implications of choosing a particular epsilon value are, the program was run with several types of data. Figure 19. summarizes the results.

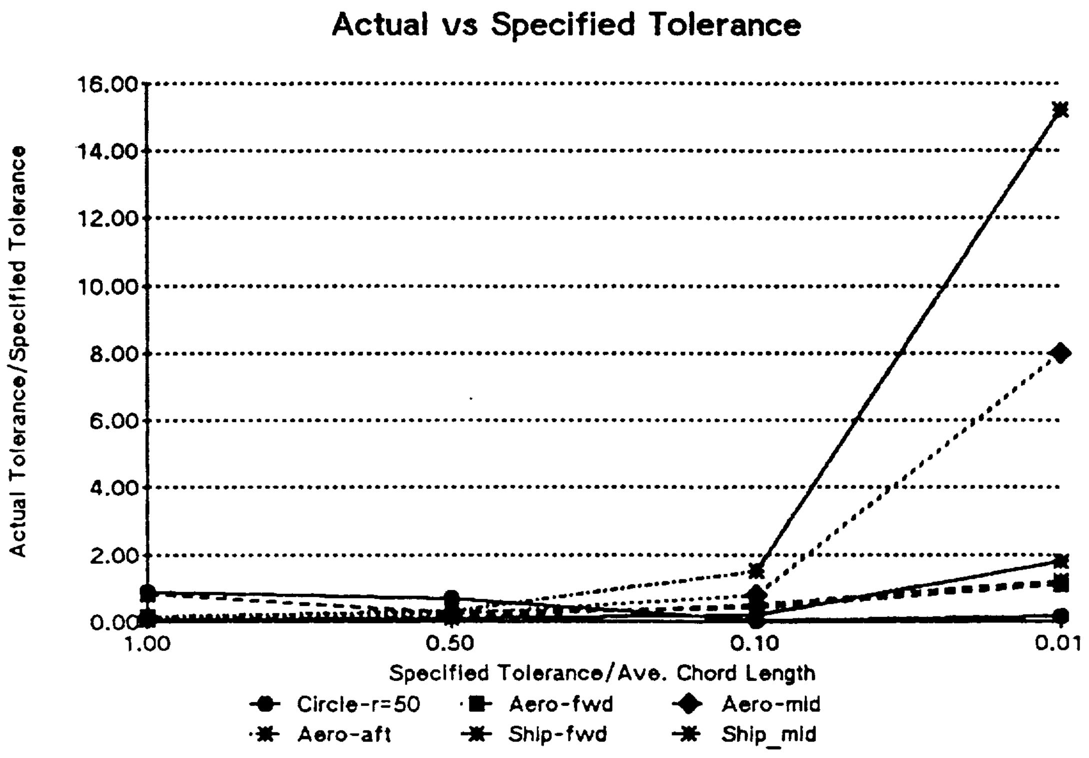


Figure 19.

The tolerances in this graph are normalized to average chord length for the particular data set. It appears that the actual tolerance is approximately equal to the specified tolerance for epsilons down to about one-tenth of the average chord length. As tolerance requirements become tighter, it becomes harder to correlate specified and actual values. It is important to note, however, that very stringent requirements on tolerance are counter to the desire of fitting many points with minimal segments.

SEGMENTATION

Piegl and Chou suggest a binary search to subdivide the data set when a fit within tolerance is not possible. For this implementation, some preprocessing was added. Since space vehicles may have corners, this program allows the user to define a maximum corner angle. The data set is processed by calculating the angle between vectors formed by three consecutive points. When the angle falls below the maximum, the data set is separated in two.

Like many other curve fitting schemes, this method does not always handle inflection satisfactorally. Test runs (Figure 20.) showed that when an interval with an inflection narrowed to three points, an unacceptably 'wiggly' curve resulted. Fortuantely, this algorithm provides a simple method to identify inflection. End tangents orientation which cause a Bezier to cross P_0P_3 (Figure 22.) signify inflection. To handle this situation, the program successively scans each set of three adjacent point and splits the data set when inflection is found. Figure 21. shows the improvement in the quality of the curve fit as compared to Figure 20.

G^1 AND C^1 CURVE FORMS

This implementation generates a C^1 curve, calculating the G^1 form as an intermediate step. As suggested by Piegl and Chou, this is accomplished by piecing the Beziers with

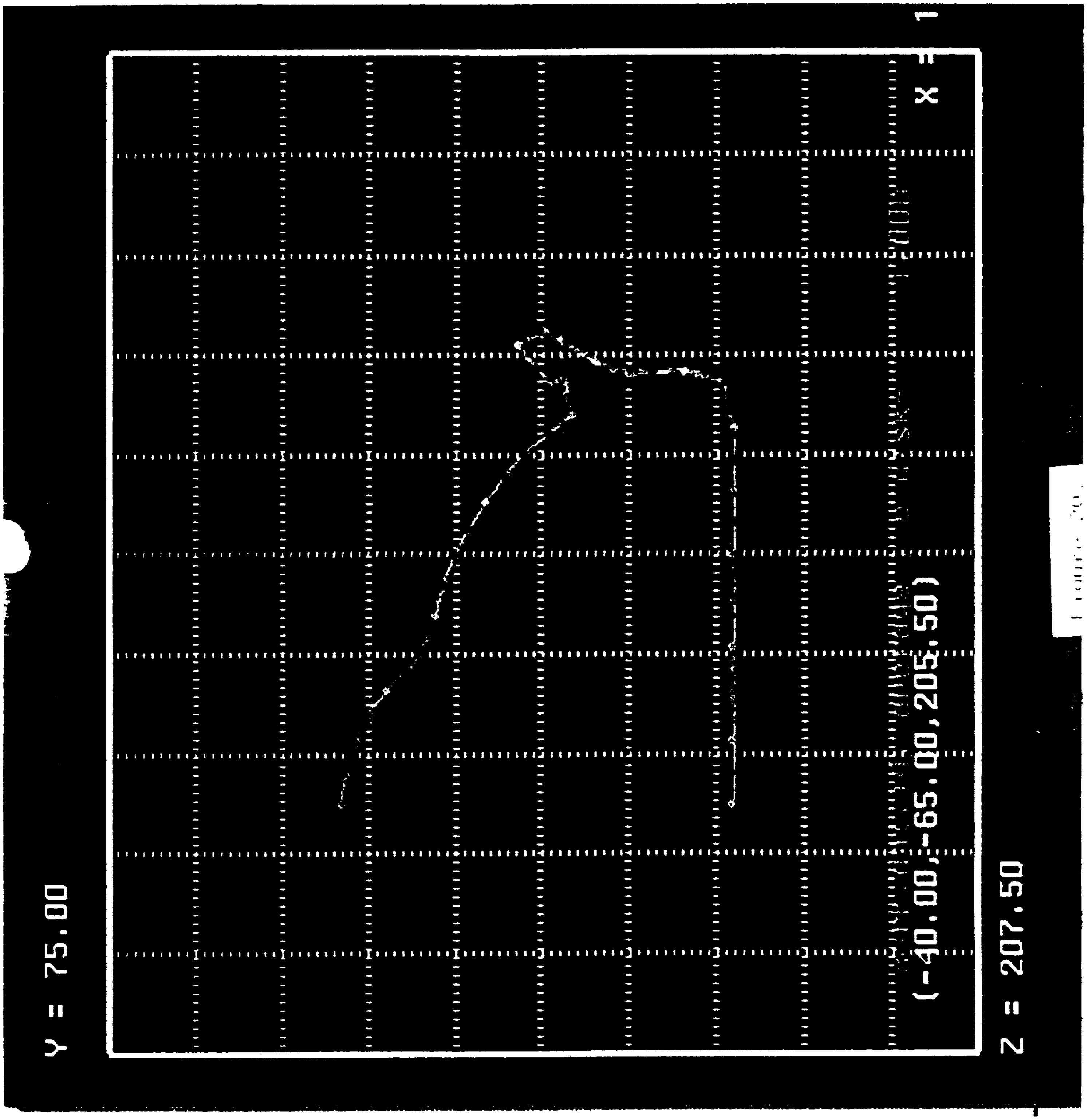


Figure 20.

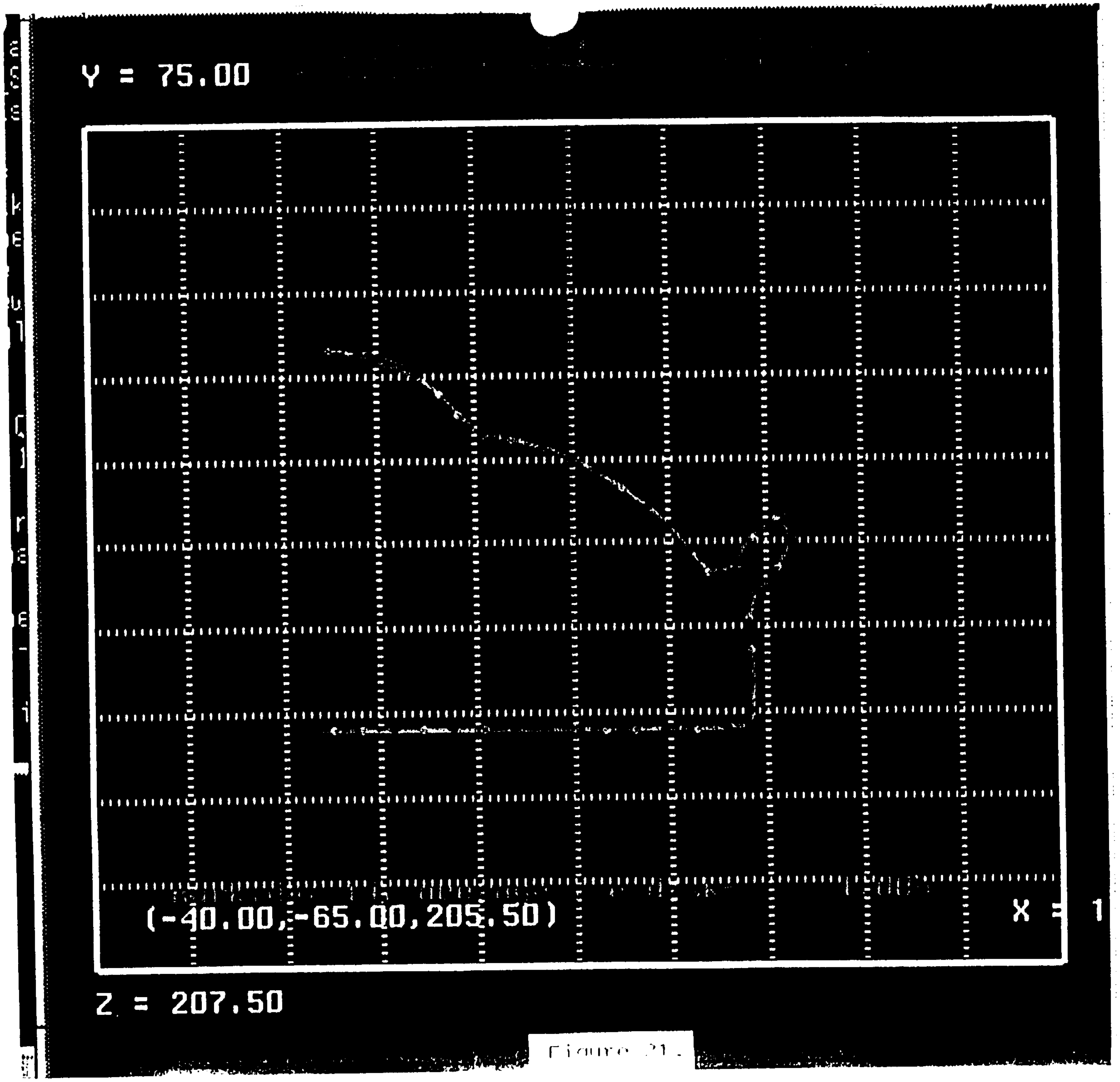


Figure 21.

triple knots parameterized on chord length at the join points (Figure 23.) To obtain the C^1 form, the weights are adjusted to match tangents at segment boundaries. The authors present the necessary algebraic manipulations to perform this, but the following analysis provides a more intuitive understanding of the process. Consider the first segment in Figure 23. Any weight changes must satisy the cubic shape invariance factors

$$S_{left} = \frac{w_0 w_2}{w_1^2} \qquad S_{right} = \frac{w_1 w_3}{w_2^2} .$$

In fact, no weight adjustments are necessary for the first segment. W_4 is determined from the tangent matching condition which, when based on a chord length parameterization evaluates to

$$w_4 = w_2 * \frac{|P_2P_3|}{|P_0P_3|} \Big/ \frac{|P_3P_4|}{|P_3P_6|}$$

On the next segment, w_5, and then w_6 are determined using the shape invariance factors. W_7 is adjusted to match the tangents. The process proceeds in a similar fashion for the remaining segments.

EVALUATION OF THE ALGORITHM

This algorithm's purpose is data reduction. Several measures of merit were developed to help quantify its performance. The first, m/m_c, measures data reduction. It compares number of segments generated (m) to number of segments if a piecewise cubic were fitted to the data (m_c).

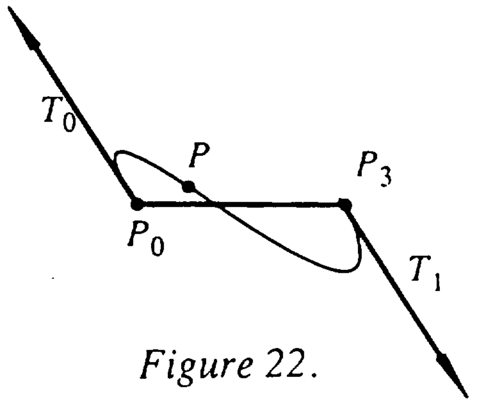


Figure 22.

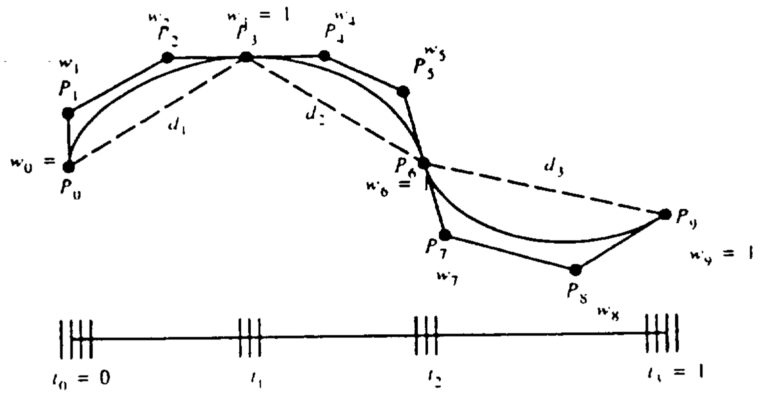


Figure 23.

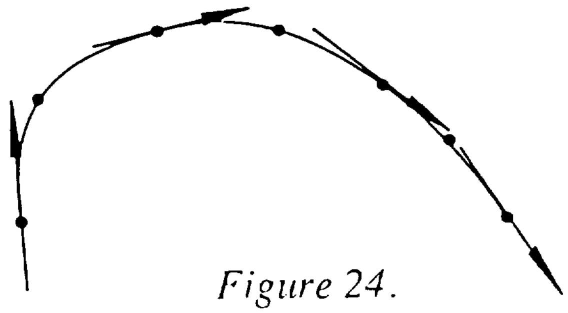


Figure 24.

C1 continuity is obtained with a cubic fit in the following manner:

1. Define endpoint tangent for initial point
2. Using this tangent and three points to calculate a cubic
3. Calculate resulting tangent at third point
4. Repeat steps 2. and 3. until done.

The result, as seen in Figure 24., is $m_c = (n-1)/2$ where n is the total number of points. Since this method would require preprocessing for corners, appropriate adjustments have been made in any comparisons. M/m_c will vary with epsilon; greater tolerance will require fewer segments. Figure 25. shows data reduction for a variety of curve types and tolerances.

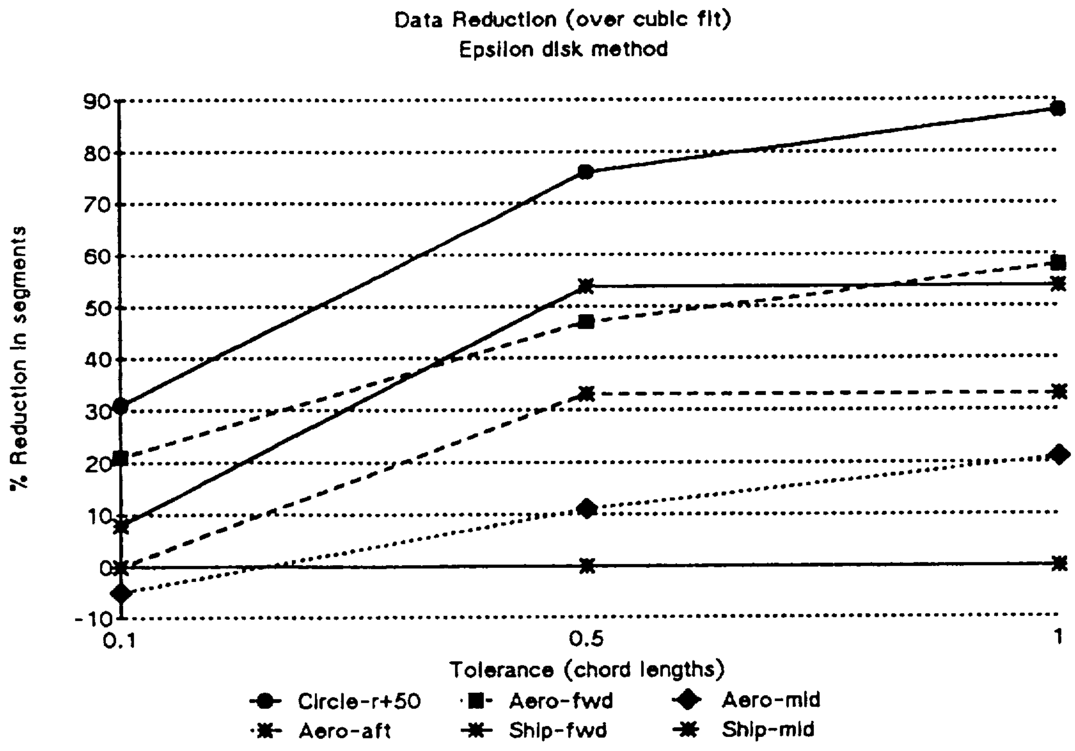


Figure 25.

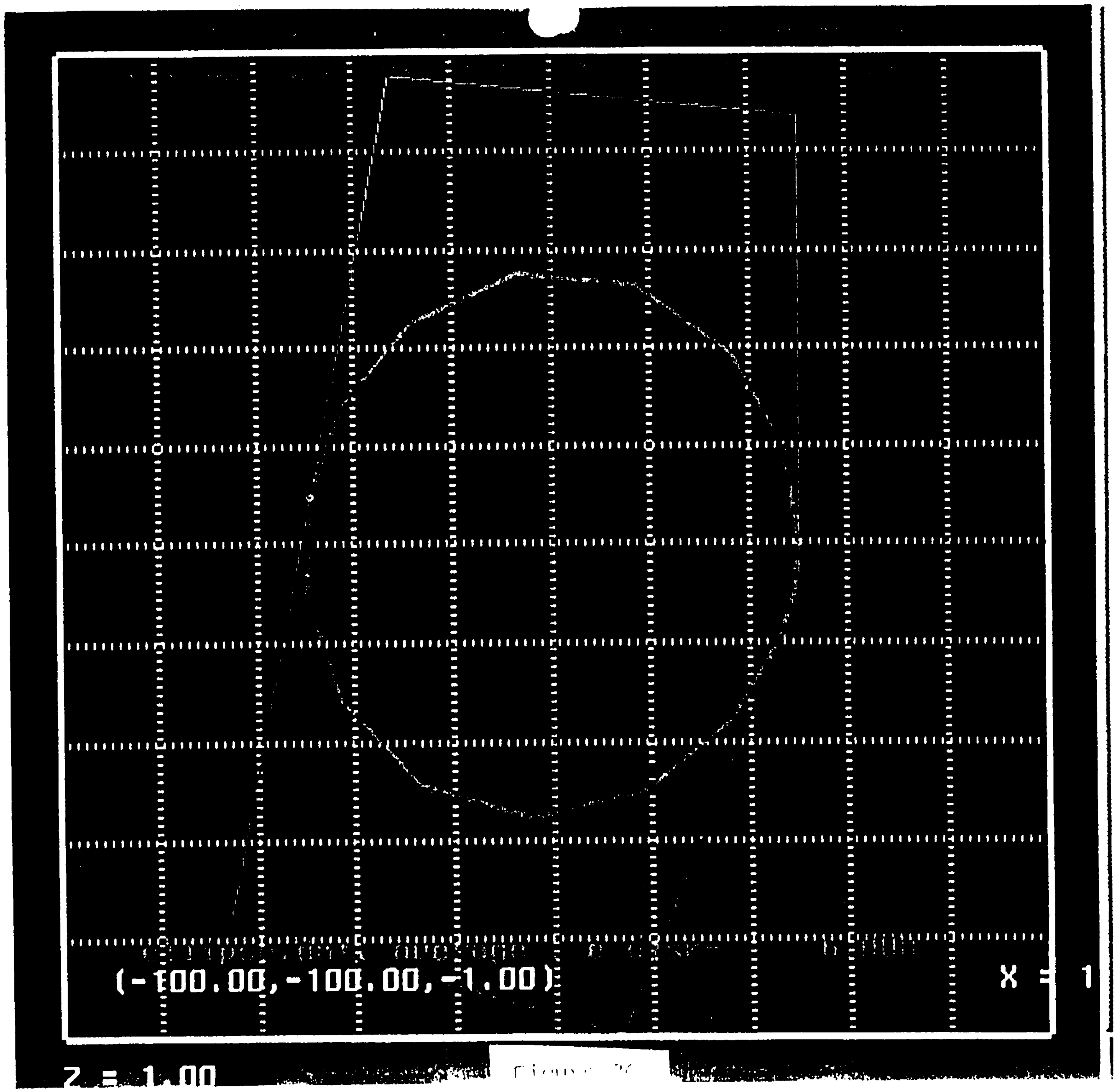
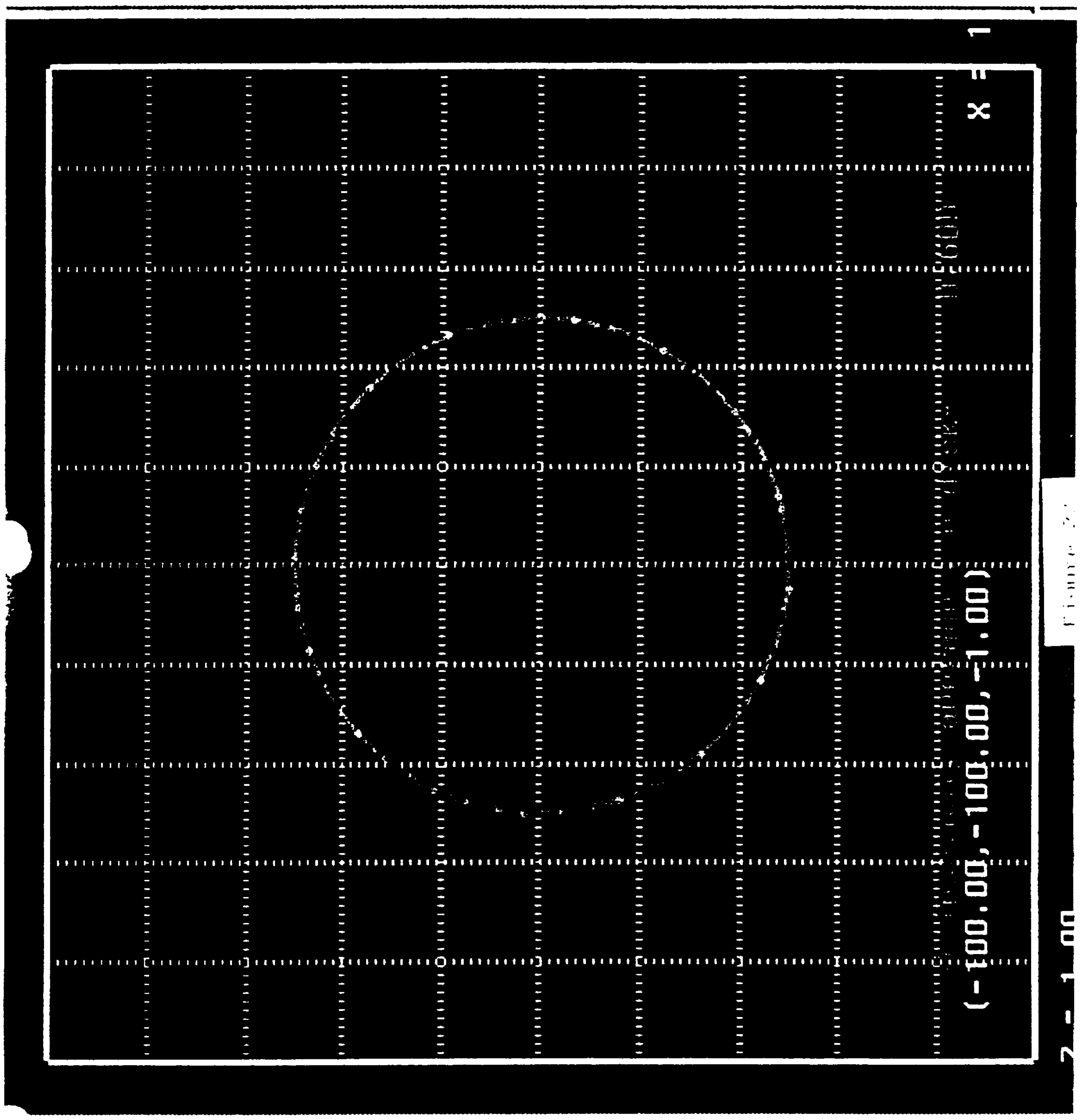


Figure [illegible]


X = 1
(-100.00,-100.00,-1.00)
Z = 1.00

For a relative tolerance of .1 chord length, it appears that the number of segments needed to describe the curve can be reduced anywhere from 0 to 30% over a simple cubic fit. Figures 26. and 27. illustrate the tradeoff between tolerance and data reduction. Upon closer, inspection of the data, the curves which had minimal data reduction tended to have many inflection points. The small data reduction is actually the result of a tradeoff for curve 'quality' in the form of better inflection handling. The quality of the same curves fit with consecutive cubics was not investigated, but they will most likely exhibit oscillation at inflection points.

CONCLUSIONS

Regarding the implementation, for curves without a lot of inflection, it offers the potential for significant data reduction. With tolerances of 10% of the average chord length, data reductions of 15% over a simple cubic fit are realistic.

The purpose of a master's project is to provide practical software engineering experience. Several important conclusions can be drawn based on the experience gained during the implementation of this algorithm including

1. Special case and error handling comprise a large portion of developing practical software. Dealing with these issues constituted the bulk of this project. This program was by no means piece of commercial quality

software, but to be even marginally useful to the engineers at NASA, errors and special cases had to be considered.

2. Curve and surface fitting is somewhat of an art. Often, a visibly pleasing fit is an indication of the quality of fit, and there are many methods available to manipulate a curve into a visually pleasing form. This does not negate the need to understand the mathematics behind these methods however. Familiarity with a variety of curve and surface fitting techniques provides a programmer with the ability to intelligently adapt his or her application to the problem at hand.

3. Many of the relationships between points, weights, and a resultant curve can be arrived at through algebraic manipulation of the parametric curve equations. However, these relations are often much more intuitive when approached from a geometric standpoint. They are also easier to program using geometric operations.

RECOMMENDATIONS FOR FUTURE WORK

1. Extend this algorithm to surfaces. This will require defining one or several methodolgies for how the surface will be divided into patches. It is not likely that several consecutive curves will be fit with equal numbers of equally sized segments. Perhaps a sort of

reverse stategy, where the user defines the size and location of patches and the program returns actual tolerances, may be necessary.

2. Analyze a different curve representation, (NURBS, as opposed to cubic Bezier) for data reduction potential in a similarly geometric fashion. This would entail an extensive literature search to see if similar work has already been done. A data reduction method using a NURBS rep would have C^2 derivative continuity which is useful in aerodynamic vehicle analysis.

REFERENCES

CHOU92 Chou, J. J. and L. A. Piegl, "Data Reduction Using Cubic Rational B-splines," IEEE Computer Graphics & Applications, Vol. 12, No. 3, May 1992, pp 60-68.

FARI88 Farin, G., Curves and Surfaces for Computer-Aided Design, Academic Press, New York, 1988.

FOLE90 Foley, J.D., A. van Dam, S.K. Feiner and J.F. Hughes, Computer Graphics, Principles and Practice, Addison-Wesley, Massachussetts, 1990.

KREY79 Kreyszig, E., Advanced Engineering Mathematics, John Wiley & Sons, New York, 1979.

PIEG87 Piegl, L. and W. Tiller, "Curve and surface construction using rational B-splines," Computer Aided Design, Vol. 19, No. 9, Nov 1987, pp 485-498.

AIAA

AIAA-92-4774
Structural and Loads Analysis of a Two-Stage Fully Reusable Advanced Manned Launch System

James C. Robinson
Old Dominion University
Research Foundation
Norfolk, VA
Douglas O. Stanley
NASA Langley Research Center
Hampton, VA

Fourth
AIAA/USAF/NASA/OAI
Symposium on Multidisplinary Analysis and Optimizations
September 21-23, 1992 / Cleveland, OH

STRUCTURAL AND LOADS ANALYSIS OF A TWO-STAGE FULLY REUSABLE ADVANCED MANNED LAUNCH SYSTEM

James C. Robinson*
Old Dominion University Research Foundation
Norfolk, Virginia 23529

Douglas O. Stanley**
NASA Langley Research Center
Hampton, Virginia 23681-0001

Abstract

The conceptual design of a rocket-powered, two-stage fully reusable launch vehicle has been performed as a part of NASA's Advanced Manned Launch System (AMLS) study. This paper summarizes the structural design and loads analysis of this two-stage fully reusable launch vehicle. The method used to determine the structural weights consists of generating a finite-element model for each vehicle, selecting a set of critical loading conditions, determining the loads on the model caused by those conditions, determining the model response and changing the sizes of individual elements to obtain a safe structure. The integrated loads on the two-stage vehicle were obtained from a three-degrees-of-freedom trajectory analysis.

Nomenclature

Al/Li	aluminum lithium
AMLS	Advanced Manned Launch System
APAS	Aerodynamic Preliminary Analysis System
DOF	degrees of freedom
EAL	Engineering Analysis Language
ET	external tank
FEM	finite-element model
g	acceleration of gravity at Earth's surface (32.2 ft/sec^2)
Gr/Pi	graphite polyimide
KSC	Kennedy Space Center
LH2	liquid hydrogen (at 4.43 lb/ft^3)
LO2	liquid oxygen (at 71.2 lb/ft^3)
PLS	Personnel Launch System
POST	Program to Optimize Simulated Trajectories
q	dynamic pressure, lb/ft^2
SMART	Solid Modeling Aerospace Research Tool
T/W	thrust-to-weight ratio
α	angle of attack, degrees
ΔV	incremental velocity, ft/sec

*Aerospace Engineer, Associate Fellow AIAA.

***Aerospace Engineer, Space Systems Division, Member AIAA.*

Introduction

In recent years, NASA has begun studies to define options for the next manned space transportation system. The goals of this broad NASA effort are to define systems that meet future mission requirements of transporting personnel and payloads requiring a manned presence, while emphasizing improved cost-effectiveness, increased vehicle reliability and personnel safety, and large operational margins. Three approaches are being examined for satisfying future manned launch needs. One approach is the evolution of the current Space Shuttle. Another is the definition of a small Personnel Launch System (PLS) for carrying people and small amounts of cargo to and from space. The third approach is that of a new, more operationally efficient Advanced Manned Launch System (AMLS) to replace the present Space Shuttle.[1]

The goals of the AMLS study are to examine systems that provide routine, lower-cost manned access to space. Technologies and system approaches are being studied that will contribute to significant reductions in operating costs relative to current systems. A wide variety of vehicle types and propulsion systems has been examined in the conceptual and preliminary design of next-generation manned launch systems as a part of the AMLS study. These include single-stage and two-stage systems, systems utilizing rocket and airbreathing propulsion, and systems with varying degrees of reusability.[2] For the assumed flight rate, payload class, and technology readiness, a rocket-powered, two-stage fully reusable system was selected for detailed study. This rocket-powered, two-stage vehicle would be expected to have a 2005-2010 initial operating capability in order to gradually replace an aging

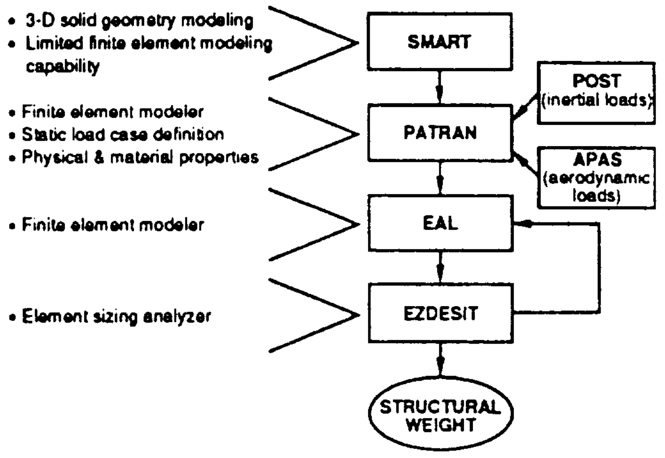
Shuttle fleet. Hence, a 1995-2000 technology readiness date has been assumed to represent normal growth (evolutionary) technology advancements in vehicle structure, propulsion, and subsystems. Although many of these assumed technological advancements contribute to significant weight savings in the vehicle, a portion of this weight savings has been applied to aspects of vehicle design that enhance the operations, reliability, and safety factors of the system.

This paper summarizes the structural design and loads analysis of this two-stage fully reusable launch vehicle. The method used to determine the structural weights consists of generating a finite-element model for each vehicle, selecting a set of critical loading conditions, determining the loads on the model caused by those conditions, determining the model response and changing the sizes of individual elements to obtain a safe structure. The integrated loads on the two-stage vehicle were obtained from an optimal three-degrees-of-freedom (3-DOF) trajectory analysis.

Analysis Methodology

The integrated structural design of next-generation rocket-powered launch systems requires proper consideration of the effects of the vehicle geometry, trajectory, and aerodynamics. All of the geometry and subsystem packaging is performed using the NASA-developed Solid Modeling Aerospace Research Tool (SMART) geometry package. SMART is a menu-driven interactive computer program for generating three-dimensional Bezier surface representations of aerospace vehicles for use in aerodynamic and structural analysis.[3] All of the trajectory analysis is performed using the 3-DOF version of the Program to Optimize Simulated Trajectories (POST). POST is a generalized point mass, discrete parameter targeting and optimization program which allows the user to target and optimize point mass trajectories for a powered or unpowered vehicle near an arbitrary rotating, oblate planet.[4] The Acrodynamic Preliminary Analysis System (APAS) is used to determine vehicle aerodynamics. In the subsonic and low supersonic speed regimes, APAS utilizes slender body theory, viscous and wave drag empirical techniques, and source and vortex panel distributions to estimate the vehicle aerodynamics. At high supersonic and hypersonic speeds, a non-interference finite-element model of the vehicle is analyzed using empirical impact pressure methods and approximate skin-friction methods.[5]

The method used for structural analysis and weights determination includes geometry modeling, finite-element modeling, loads generation and application, finite-element analysis, element sizing to meet loading conditions and structural criteria, and structural element weight summation, organized into an iterative process. This process is illustrated in Fig. 1. The external shape of a vehicle configuration can be modeled by discretizing a SMART geometry into a finite-element model (FEM) through the use of PATRAN.[6] For relatively simple geometries, a FEM can be constructed directly. Experience is then used to initially determine and model the internal structure of the vehicle. Physical and material properties of the structure are included in the FEM of the vehicle. Mission static-load cases are assembled from critical POST inertial loads and APAS aerodynamic loads. The completed finite-element structural model with physical and material properties, external loading, and structural arrangement is then ready for analysis.



• 3-D solid geometry modeling
• Limited finite element modeling capability
SMART
POST (inertial loads)
• Finite element modeler
• Static load case definition
• Physical & material properties
PATRAN
APAS (aerodynamic loads)
• Finite element modeler
EAL
• Element sizing analyzer
EZDESIT
STRUCTURAL WEIGHT

Figure 1. Structural analysis methodology.

The Engineering Analysis Language (EAL)[7] is used for the finite-element analysis. The finite-element analysis produces resultant structural loads due to the loading conditions for each element. The responses determined by EAL for the loading conditions include displacements and reactions, stresses, buckling loads and strain-energy densities. The resultant loads are indicative of the load paths of the vehicle structure. These loads are applied to the EZDESIT program[8] to size the finite-elements (bars, planar beams, and plate elements) to withstand the loading conditions as shown in Fig. 2. The cross-sectional areas of bar elements are sized. The cap cross-sectional areas and web height and thickness are sized for planar beams. The plate element design variables depend on the type of construction chosen. Isotropic and composite honeycomb, hat-stiffened, and membrane panels along with corrugated web elements can be sized by the code. For each element, a stiffness matrix and a construction geometry (lamina gage, honeycomb core height, etc.) are specified, and each element has an initial thickness equal to the minimum

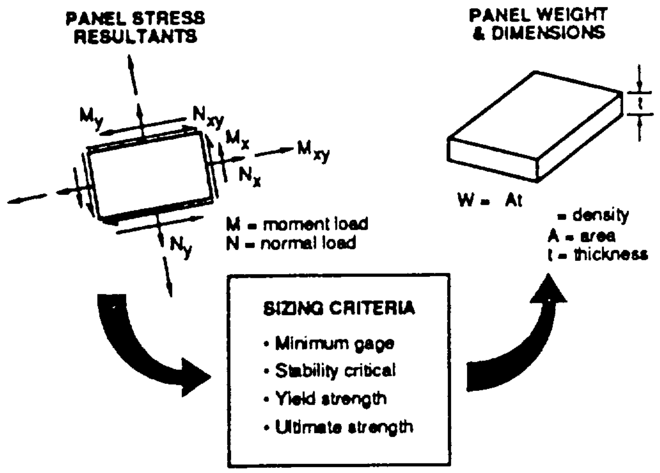


Figure 2. Panel sizing methodology.

gage value. The elements are sequentially checked for failure due to panel buckling, yield, and ultimate modes for each loading case. If failure occurs, the element dimensions are increased until the indicated failure mode is satisfied. The geometric sizing of the panel alters the stiffness properties. Thus, the finite-element analysis and geometry sizing are iterated until convergence is achieved.

Resizing of structural elements to increase the global buckling strength of the structure is accomplished after completion of the strength sizing. Local buckling constraints are satisfied in the strength sizing in EZDESIT. The method uses EAL-generated element strain-energy densities to calculate scaling factors that are applied to EZDESIT element dimension files. The global-buckling sizing method is described in Appendix A. Scaling factors are calculated using values for the first buckling mode of the loading conditions which caused global buckling in the strength-sized structure below ultimate load. This method provides a satisfactory structure but does not optimize the solution.

The dominant load case for each element is determined, and its corresponding dimensions, weight, and failure mode are obtained. The results of the sizing can be reviewed in two different manners. The resulting weights can be grouped by failure mode, element type, load case, and component, or the EZDESIT output file can be read into PATRAN and the element properties displayed on the model. Those properties include internal loads, dominant load case, failure modes, and unit weights. Highly stressed areas may indicate a need for an alternative structural design. Resultant loads are reviewed by the structural designer, and the necessary changes to the structural arrangement are made by altering the FEM and reanalyzing the structure.

Vehicle Concept

Mission and Guidelines

The design reference mission for the two-stage fully reusable AMLS vehicle calls for the delivery and return of up to 40,000 lb of payload from Kennedy Space Center (KSC) to Space Station Freedom (220 nmi, 28.5° inclination) along with a crew of ten (eight passengers and a two-person flight crew). A three-day flight duration with an in-flight margin was budgeted (35 man-days). The payload bay dimensional requirements were a 15-ft diameter by 30-ft length. On-board propellant would provide an incremental velocity (ΔV) of 1350 ft/sec following launch insertion into a 50 x 100 nmi orbit. Landing would nominally be at the KSC launch site.

The AMLS vehicle was designed with a crew escape capability characterized by the jettisoning of the crew module using high-impulse solid rocket motors with inflight stabilization followed by the deployment of a parachute system for landing. In addition, both the booster and orbiter have single-engine-out capability from lift-off for added reliability and mission success. A 15-percent dry weight growth margin was also allocated. The orbiter was required to have a 1100-nmi crossrange capability to allow once-around abort for launch to a polar orbit and to increase daily landing opportunities to selected landing sites. All trajectories for this vehicle have maximum acceleration limits of 3 g and normal load constraints on the wings equivalent to a 2.5-g subsonic pull-up maneuver.

Vehicle Configuration

The AMLS vehicle, shown in Fig. 3, is a two-stage, parallel-burn design that consists of a manned orbiter and an unmanned winged booster that stages at a Mach number of 3 and glides back to the launch site. Propellants are crossfed from the booster to the orbiter during the boost phase so that the orbiter's propellant tanks are full at staging. Both the booster and orbiter use liquid hydrogen and liquid oxygen as propellants. The orbiter also employs a detachable payload canister concept to allow off-line processing of payloads and rapid payload integration. Both the booster and orbiter are control configured and employ wing tip fins for lateral control. Integral, reusable cryogenic propellant tanks are used on both the booster and orbiter. Dual-lobed tanks are used on the orbiter to allow the external payload canister to be easily integrated and for aerodynamic and reentry heating considerations. As shown in the figure, the total vehicle dry weight is 343,000 lb, and the gross weight is 2,604,000 lb. The total liftoff thrust-to-weight ratio (T/W) of the vehicle is 1.3. The reference AMLS orbiter utilizes five light-weight derivatives of the Space Shuttle Main Engine for main propulsion, whereas the reference booster uses five of the same engines with a lower area ratio nozzle. These engines are throttled to 80 percent of rated thrust for normal operation to provide single-engine-out capability on each stage and to increase individual engine life. Both the booster and orbiter utilize an internally stiffened ring-frame construction with carrier panels and durable metallic thermal protection system tile sections attached to insulated standoffs where appropriate.

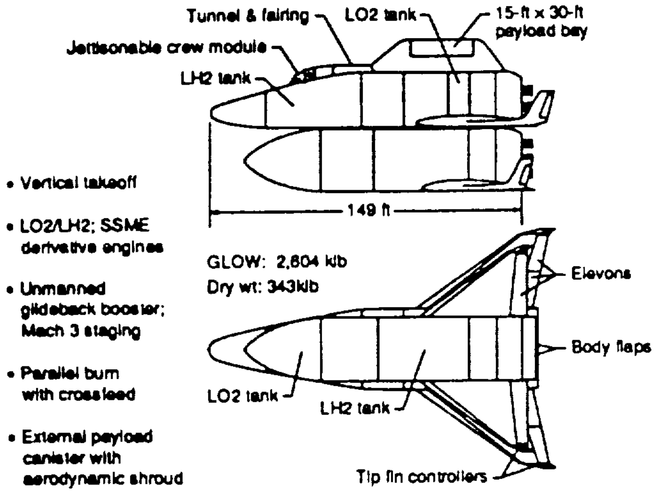

Tunnel & fairing
LO2 tank
15-ft x 30-ft payload bay
Jettisonable crew module
LH2 tank
• Vertical takeoff
• LO2/LH2; SSME derivative engines
• Unmanned glideback booster; Mach 3 staging
• Parallel burn with crossfeed
• External payload canister with aerodynamic shroud
149 ft
GLOW: 2,804 klb
Dry wt: 343klb
Elevons
Body flaps
LO2 tank
LH2 tank
Tip fin controllers

Figure 3. AMLS two-stage vehicle configuration.

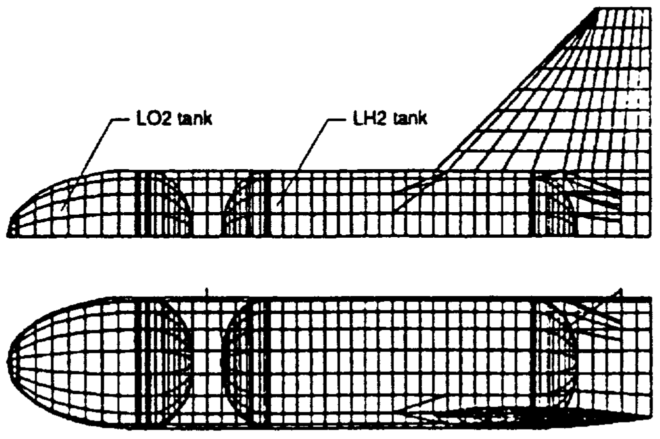


Figure 4. Booster finite element model.

Structural Configuration

The construction of the orbiter and booster of the AMLS utilizes near-term technology. The main propellant tanks were assumed to be welded aluminum-lithium (Al/Li) 2095 structures that were machined from plate. External foam insulation is used on the surfaces requiring insulation because internal cryogenic insulation was judged to be higher-risk. The cryogenic tanks and wing carry-through structure are separated as much as possible to maintain relative simplicity in the tank construction. Because of the lack of experience in the use of reusable cryogenic tanks, the AMLS design philosophy maintains simple structural arrangements, stress patterns and temperature distributions that may contribute to improved tank life.

Booster Description. The structural arrangement of the booster is shown in Fig. 4. The tank construction of the booster is similar to that of the External Tank (ET) of the Space Shuttle system except for the use of the more advanced Al/Li 2095 material. The liquid oxygen (LO2) tank has a flat front bulkhead, an ogive forward section, a cylindrical body and an ellipsoidal aft dome similar to the ET LO2 tank. The Al/Li 2095 intertank section is dry and contains the forward landing gear and support for the forward connection to the orbiter. The liquid hydrogen (LH2) tank is cylindrical with fore and aft ellipsoidal domes. It requires more internal stiffening than the ET LH2 tank because it is a primary compression (and bending) load path on the launch pad. The cryogenic tanks and intertank section are stiffened with internal Al/Li 2095 ring frames and stringers. The cylindrical section behind the LH2 tank is denoted as the aft skirt. It is constructed of a short section of Al/Li 2095 to alleviate thermal distortion problems and a longer section of graphite-polyimide (Gr/Pi) honeycomb that will withstand booster operating temperatures without insulation. It supports the thrust structure of the propulsion system and contains the primary wing attachments. The wing is also constructed of uninsulated Gr/Pi honeycomb with a non-structural titanium leading edge. The primary landing gear support is attached to the wing. The wing carry-through is attached to heavy frames in the aft skirt at the front and aft spars. The front of the wing root rib is attached to the fuselage by a vertical link. The Al/Li thrust structure is a conical shell in the upper portion with a flat shelf or beam supporting the three lower engines without attachment to the wing box or aft LH2 tank dome. There is a non-structural Gr/Pi aerodynamic fairing between the wing and the fuselage.

Orbiter Description. The orbiter (shown in Fig. 5) differs structurally from the booster in that it employs dual-lobed main propellant tanks, a long, canted conical nose and an aft-located LO2 tank. The nose of the orbiter is carbon-carbon supported by an insulated Gr/Pi shell. The forward landing gear is in the Gr/Pi honeycomb nose section. The LH2 tank is a welded Al/Li 2095 structure with Al/Li 2095 extensions on either end to provide thermal compatibility with Gr/Pi struc-

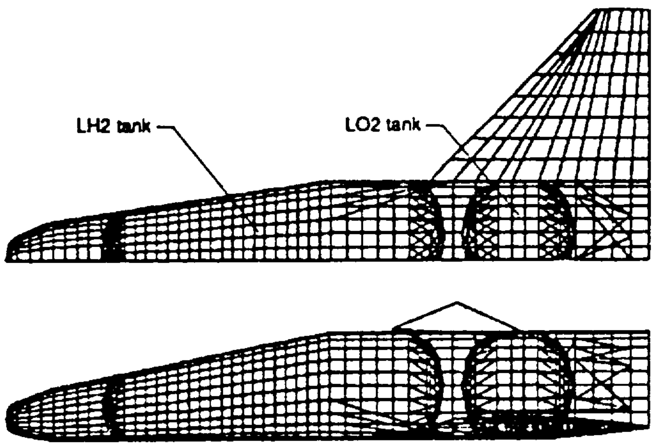


Figure 5. Orbiter finite element model.

ture. The forward attachment between the orbiter and the booster is contained in the LH2 tank. The intertank structure is constructed of Gr/Pi honeycomb. The LO2 tank is constructed of Al/Li and carries some primary thrust loads. The Al/Li thrust structure in the orbiter is a truss structure design. Supports for the truss are attached to the longerons in the LO2 tank at the top and bottom of the web that divides the two lobes and a heavy longeron that reacts the thrust loads transmitted from the booster and extends into the LO2 tank lower skin. The fuselage section behind the LO2 tank is again denoted as the aft skirt. It is constructed of a short section of Al/Li 2095 to alleviate thermal distortion problems and a longer section of graphite-polyimide honeycomb. It supports the thrust structure of the propulsion system and contains the primary wing attachments. The orbiter wing is also constructed of Gr/Pi honeycomb with a non-structural advanced carbon-carbon leading edge. The primary landing gear support is attached to the wing. The wing attachment is similar to that in the booster except that there is a forward beam through the intertank area. The orbiter also has non-structural Gr/Pi aerodynamic fairings between the wing and fuselage and between the lobes of the LO2 and LH2 tanks. These fairings support the thermal protection system but must be attached to the tanks in a manner that will absorb the thermal shrinkage of the cryogenic tanks while maintaining the integrity of the thermal protection system.

Results and Discussion

Trajectory Analysis

The nominal POST ascent trajectory for the two-stage fully reusable vehicle is presented in Fig. 6. As shown in the figure, the initial T/W is about 1.3. As propellant is burned, the vehicle accelerates until it enters the transonic flight regime at high dynamic pressure (maximum of 700 psf) at about 60 sec. The large increase in drag at this point causes the rate of acceleration to decrease for a short period of time. The vehicle then accelerates until staging occurs at Mach 3 at an altitude of 71,000 ft. Vehicle trim and control during ascent is provided by gimballing of the main engines of both vehicles. Because the booster is empty of propellant and the orbiter is fully loaded at staging, engine gimbal angles of up to 10 degrees are required to trim the configuration. This leads to large thrust loads (up to 2.0 Mlb) being transferred between the vehicles and complicates the vehicle structural design. The unmanned booster then separates from the orbiter and performs an unpowered glide back to the launch site. These staging and glide-back maneuvers are described in more detail in reference 9. The orbiter continues to accelerate until the longitudinal acceleration limit of 3 g is encountered at 300 sec. The engines are throttled to maintain this limit until orbital insertion occurs at 420 sec into a transfer orbit with a 50-nmi perigee and 100-nmi apogee. Further details on ascent trajectories for the AMLS vehicle are contained in reference 10.

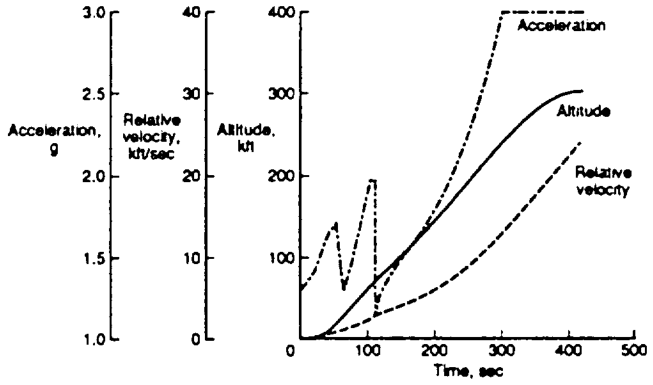


Figure 6. AMLS ascent trajectory.

The nominal POST entry trajectory for the fully reusable orbiter is presented in Fig. 7. After performing a deorbit burn, the vehicle reaches nominal atmospheric interface (altitude of 300,000 ft) at a relative flight path angle of -1° and an angle of attack of 30°. Throughout the majority of the entry profile, the angle of attack of the orbiter remains between 25° and 30° to allow hypersonic trim, maximize lift-to-drag ratio, and minimize lee-side heating. POST was employed to determine a trajectory that minimized the maximum stagnation-point heat rate during entry while still achieving sufficient crossrange (1100 nmi) to allow for a once-around abort from a polar orbit. At an altitude of 260,000 ft, the equilibrium stagnation point heat rate (based on a reference sphere with a 1-ft. radius) reaches 65 Btu/ft^2-sec. The bank angle of the vehicle is then modulated between 0° and 90° for about 1700 sec to hold the heat rate below

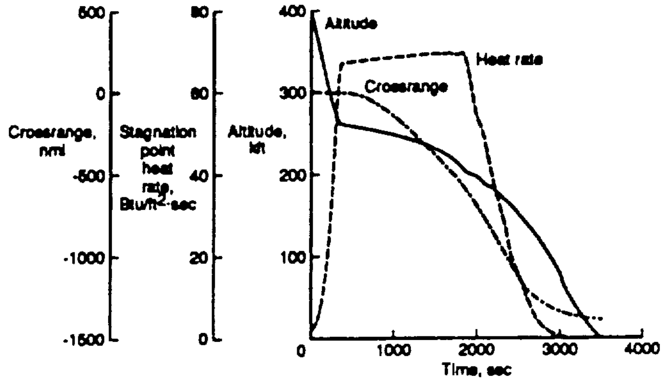


Figure 7. AMLS entry trajectory.

69 Btu/ft^2-sec. This was found to be the the minimum value that the maximum stagnation-point heat rate could be limited to and still achieve the desired crossrange. When an altitude of 200,000 ft is reached, the bank angle gradually decreases, and the vehicle prepares for terminal energy management maneuvers. Using this approach, the orbiter is capable of about 1300 nmi of crossrange. This entry analysis is used to size the nonstructural thermal protection system, which is sized to assure that the underlying vehicle structure remains within reasonable temperature ranges to ensure adequate material strength.

Finite-Element Model Construction

The results from the APAS aerodynamic analysis (vehicle lift coefficient, drag coefficient, and pitching moment coefficient variation with Mach number and angle of attack) served as inputs to the POST trajectory analysis. The integrated loads on the vehicle due to thrust, gravity, and aerodynamic forces were then obtained from POST for a number of critical loading conditions. APAS was also used to help determine the distribution of the aerodynamic loads on the vehicles. The critical loading conditions selected include:

1) a lift-off condition with impact due to the sudden release of vehicle hold-down restraints
2) a pre-launch condition without internal tank pressurization to simulate on-the-pad conditions with ground winds
3) a maximum qα (dynamic pressure multiplied by angle of attack) condition on the booster
4) a condition just prior to staging when the maximum thrust is transmitted to the orbiter
5) a Mach-3 separation maneuver on the orbiter
6) a 2.5-g subsonic pull-up maneuver
7) a 2.0-g landing impact
8) a material minimum-gage condition

The LO2 tank limit pressure utilized in the study was 22 psia, with an ultimate pressure of 33 psia. The LH2 tank limit pressure utilized in the study was 34 psia, with an ultimate pressure of 51 psia. In general, design ultimate loads were utilized of 1.5 times the applied loads. However, to increase vehicle reliability and provide longer life, it might be desirable to increase the operational load margins further. Increasing operational load margins could allow lifetime vehicle certification, similar to commercial airline practices, which would greatly reduce ground operations manpower requirements.

Symmetric finite-element models of both the orbiter and booster vehicles were constructed for the study. The model of the booster (Fig. 4) has approximately 1250 joints and 1700 elements. The model of the orbiter (Fig. 5) has approximately 1600 joints and 2800 elements. The models were analyzed separately with constraints located at the attachment points on the orbiter. The connection between the vehicles is almost statically determinate, and reactions on the models were checked against inertial forces calculated from the POST trajectory program.

Each skin was modeled with a two-dimensional element having honeycomb sandwich properties, which is the most reliable two-dimensional element with membrane and bending stiffness in EZDESIT. The unstiffened skin in the tank domes was modeled with a very shallow core structure. Because all of the skin thickness in a sandwich is effective in both directions, the stiffened skin in the tank walls was modeled with a core structure of sufficient depth to provide equivalent weight for the necessary longitudinal stiffeners in a T-stiffened skin.

Small frames were modeled with planar beam elements. Since the skin elements are large (~30 in. long), and skin curvature reduces the effective skin width actually contributing to the flange area, these elements were centered on the skin joints to prevent large amounts of skin area from contributing to the outer flange stiffness of the frame. Large frames were modeled with quadrilateral elements for webs and with rods for interior flanges. Frames are located at every skin joint in the tank and, consequently, contribute to hoop strength as efficiently as the skin in the model. Additional core weight was included to compensate for this situation.

Tank pressure loads were modeled as distributed loads on the interior of the tanks. An approximation was made in the modeling of liquid-pressure-head loads by applying the loads for partially full conditions as pressures caused by a reduced-density liquid acting on the complete interior of the tank. This produces the correct maximum pressure but produces somewhat higher pressures in other areas.

Finite-Element Model Analysis

The booster and orbiter finite-element models were analyzed using EAL for the previously mentioned load cases. The models were then resized using the EZDESIT program to satisfy strength criteria. Four iterations of the analysis and resizing procedure were carried out to obtain a satisfactory strength design. The weights calculated by EZDESIT contain a user-specified non-optimum factor or net-weight multiplier to approximate the weight of structural details not modeled in the finite-element model. The non-optimum factor used in this study is 1.5, thus increasing finite-element model weights by 50 percent. Linear bifurcation buckling analyses of both models showed that they buckled below design ultimate loads. The models were resized iteratively using the procedure discussed in Appendix A to increase the buckling strength so that it equaled or exceeded the design ultimate loads.

Booster Analysis. The strength-designed element weight-per-unit of area is shown in Fig. 8. Most of the vehicle surface skin weighs less than 2 psf. However, frame and longeron weights are not included in these weights. Two psf of Al/Li structure is equivalent to an average skin thickness of 0.099 in. of Al/Li plus the 50% non-optimum weight. In the LH2 tank, the ring frames add additional hoop material equivalent to an average skin thickness of 0.033 in. of material plus the 50% non-optimum weight. The value of internal pressure multiplied by the radius of the LH2 tank is approximately 8000 lb/in.

The sums of the strength-designed weights caused by the different load conditions are shown in Table 1, and their distribution is given in Fig. 9. The load case numbers correspond to those described above in the "Finite-Element Model Construction" section. Load case 1, lift-off with the thrust impact due to instantaneous hold-down release, sizes elements that contribute to over half of the weight. This loading condition instantaneously removes a restraining force equal to 30 percent of the vehicle's weight. Alleviation of this impact load may be beneficial, but no estimate of the resulting weight savings has been made.

Table 1. Booster weight controlled by load case.

Load Case	Weight, lb
Lift-off with impact	28,232
Unpressurized tanks on pad	270
Max qα	7500
Maximum thrust transfer	10,674
Subsonic pull-up maneuver	1,326
Runway bump	270

The sums of the strength-designed weights controlled by the different failure conditions are shown in Table 2. The small increment added to satisfy buckling constraints (not shown in Table 1) is also shown. The failure condition controlling the sizing of various elements is shown in Fig. 10. The minimum-gage-sized structure, which constitutes about 20 percent of the total weight, is primarily in the lightly loaded portions of the wing, LO2 tank, intertank and fairings. Overall buckling does not appear to be significant in the booster.

Table 2. Booster weight controlled by failure mode.

Failure Mode	Weight, lb
Minimum gage	11,122
Panel buckling	4,450
Compressive yield	3,374
Yield	8,716
Ultimate	20,610
Overall buckling	79

The calculated weights for the components of the strength-designed booster model are shown in Appendix B. Also shown are the estimated component weights from the Space Shuttle ET, where applicable. The slosh baffle weight for the booster oxygen tank was obtained from the scaled ET. The liquid hydrogen and oxygen tanks are lighter than the scaled ET weights, as expected, because the Al/Li 2095 alloy used on the booster is lighter, stronger and stiffer than the 2219 alloy used in the ET. The intertank section is considerably lighter than the scaled ET value because it is not subjected to the large thrust loads applied to the ET by the solid rocket boosters nor does it have the transverse beam and large cut-outs for the structure to support those loads.

Orbiter Analysis. The strength-designed weight-per-unit of area is shown in Fig. 11. Again, most of the vehicle surface skin weight is less than 2 psf. The webs between the two lobes of the tanks are not visible in the figure but constitute a rather large portion of the tank weights.

The sums of the strength-designed weights caused by the different load conditions are shown in Table 3, and their distribution is given in Fig. 12. The load case numbers corre-

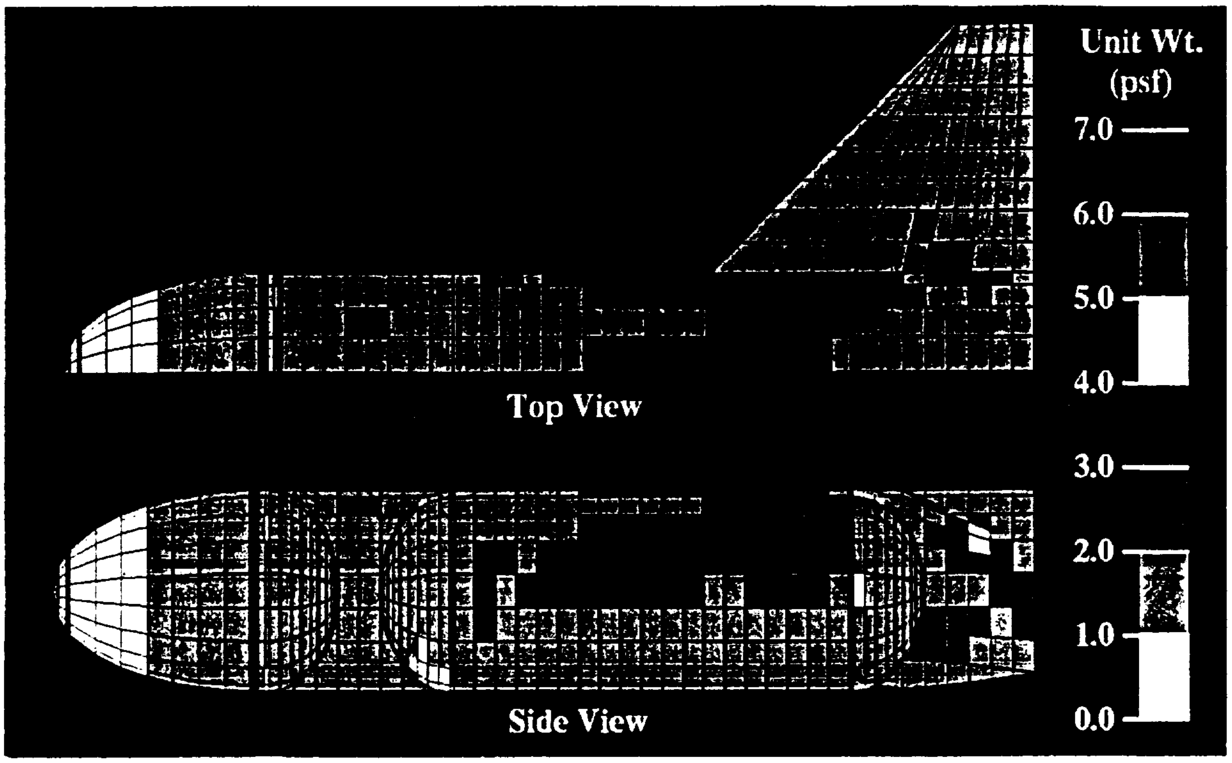


Figure 8. Booster shell elements weight per unit area.

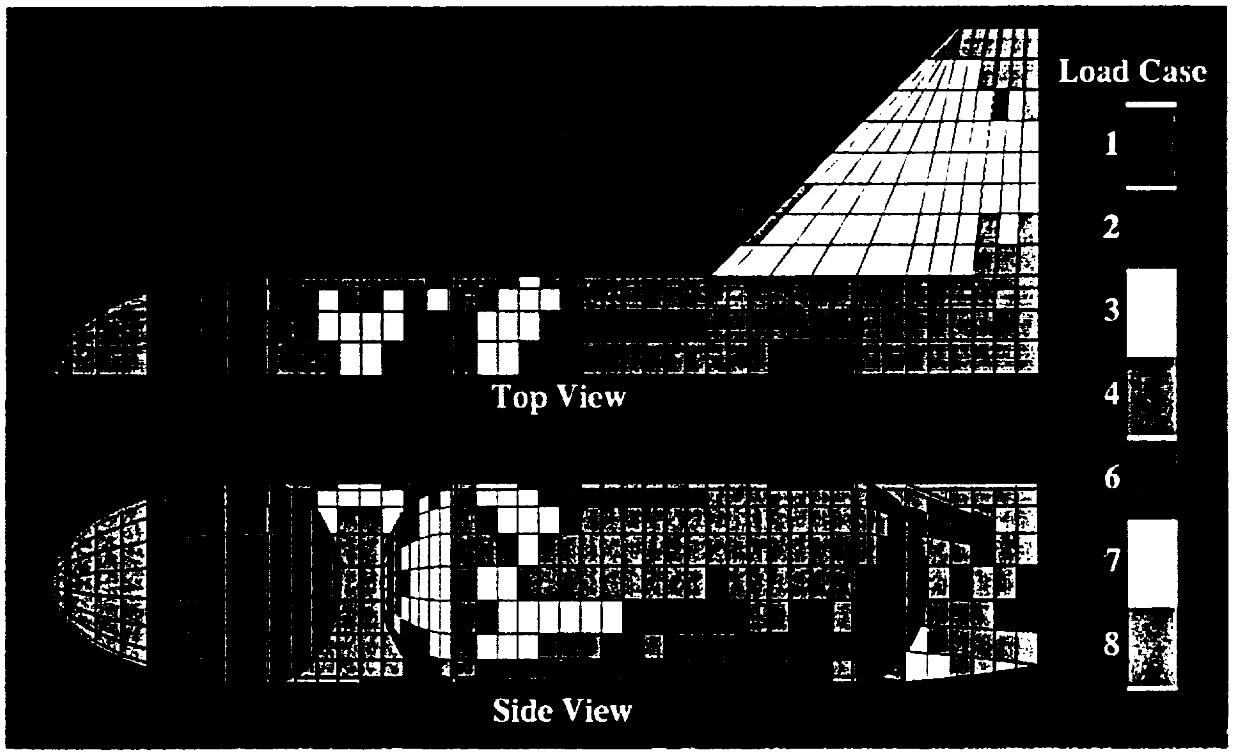


Figure 9. Booster load cases sizing shell elements.

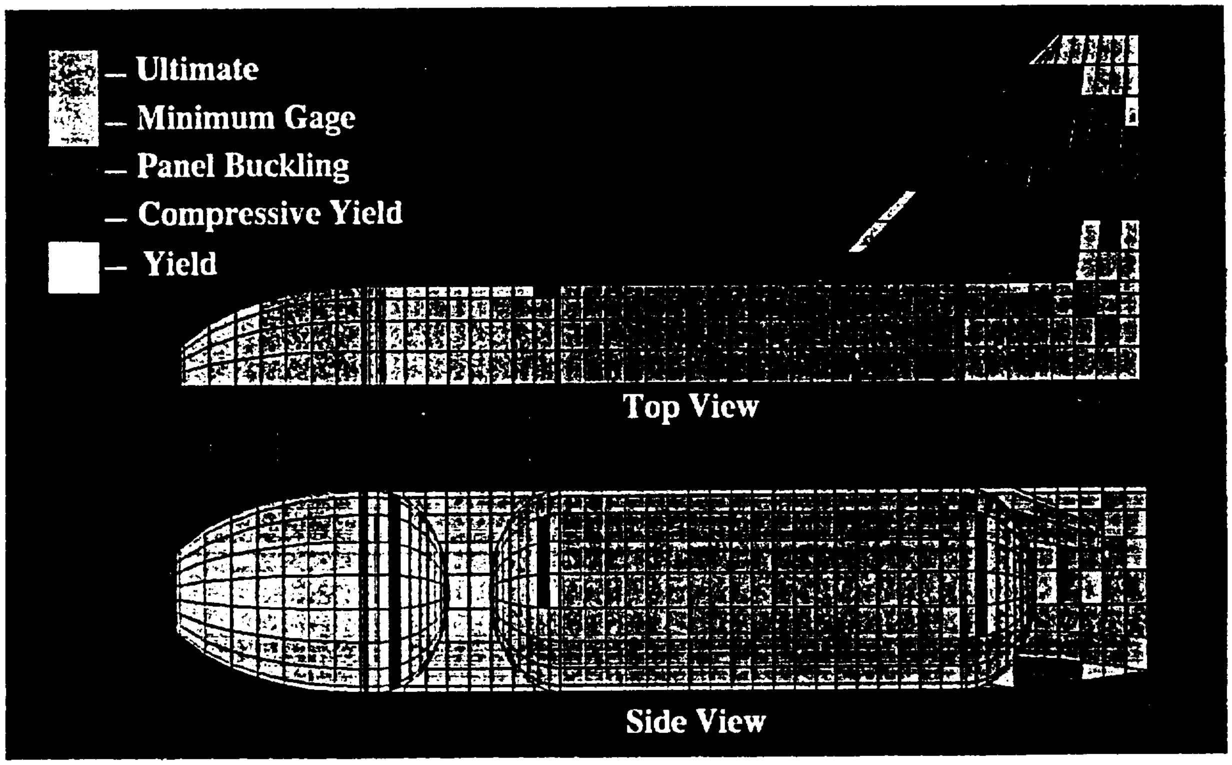


Figure 10. Booster failure conditions controlling shell element weights.

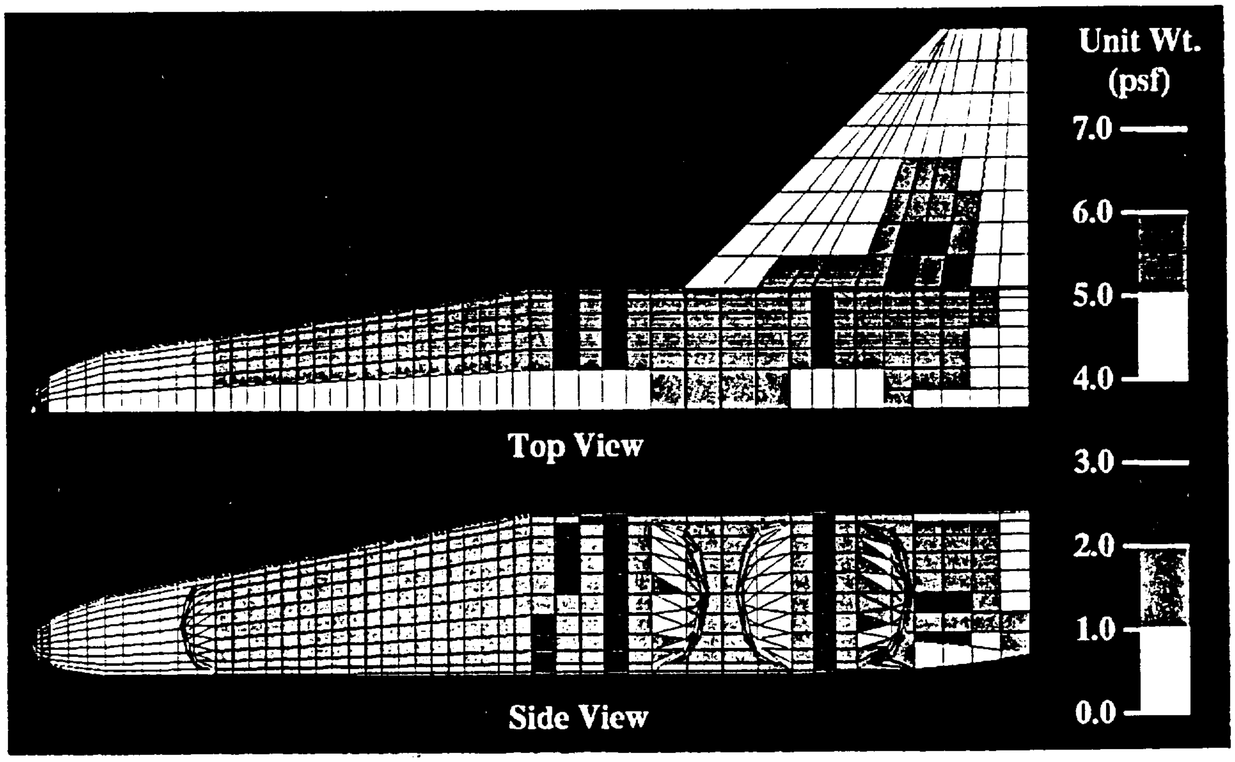


Figure 11. Orbiter shell elements weight per unit area.

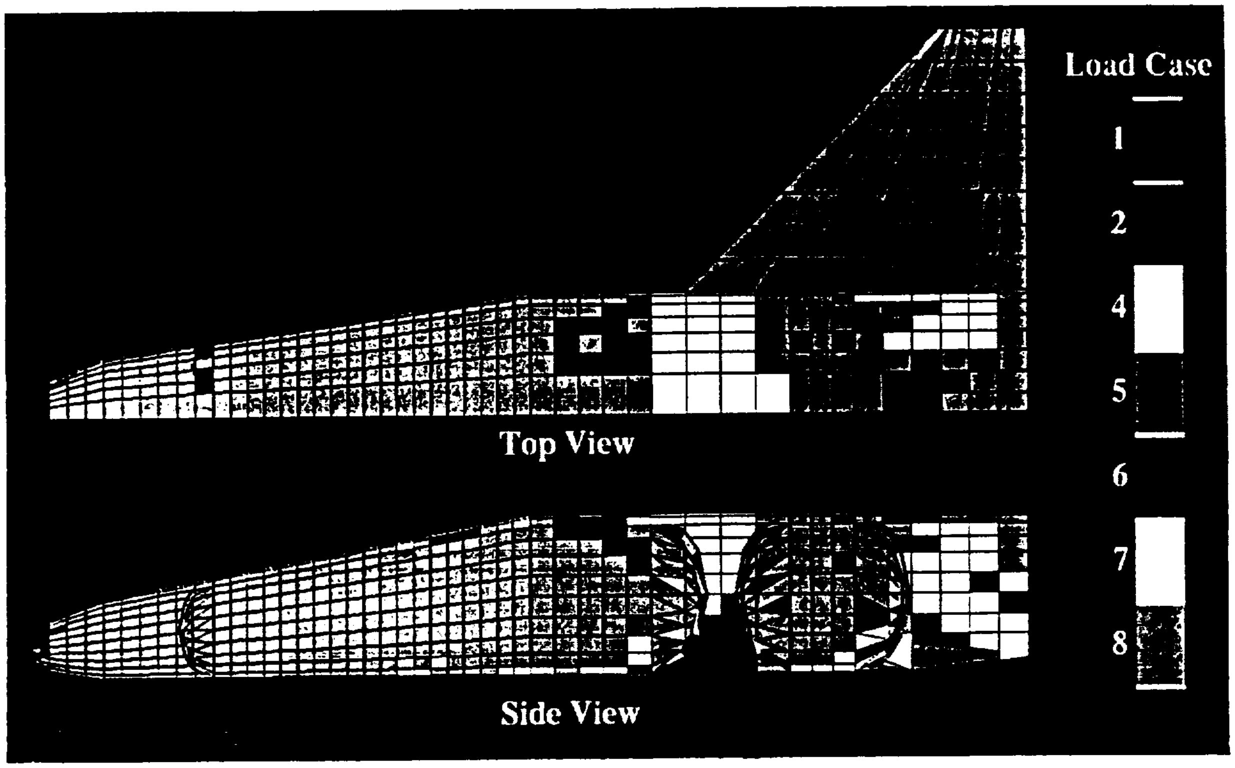


Figure 12. Orbiter load cases sizing shell elements.

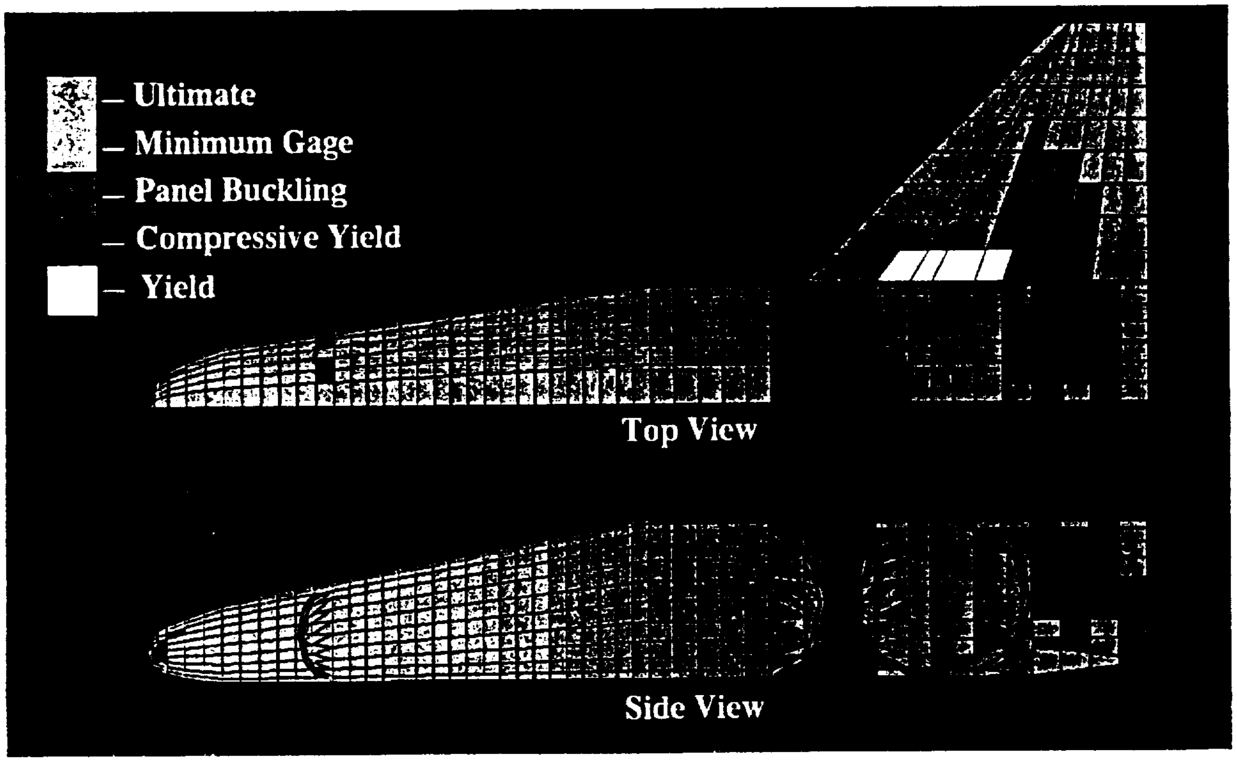


Figure 13. Orbiter failure conditions controlling shell element weights.

Table 3. Orbiter weight controlled by load case.

Load Case	Weight, lb
Lift-off with impact	33,202
Unpressurized tanks on pad	628
Maximum thrust transfer	8,234
Separation maneuver	9,666
Subsonic pull-up maneuver	4,694
Runway bump	2,060

spond to those described above in the "Finite-Element Model Construction" section. Load case 1, lift-off with the thrust impact due to instantaneous hold-down release, again sizes elements that contribute to over half of the weight.

The sums of the strength-designed weights controlled by the various failure conditions are shown in Table 4. The increment added to satisfy buckling constraints (not shown in Table 3) is shown as well. The failure condition controlling the sizing of various elements is shown in Fig. 13. The minimum-gage-sized structure, which constitutes about 40 percent of the total weight, is primarily in the lightly loaded portions of the wing, nose, fairings and the tank webs mentioned above. Overall buckling appears to be significant in the orbiter, causing a weight increase of 2800 lb. A large part of this weight increase is in the fairings over the valleys between the tank lobes. Remodeling these areas as fairings that only transmit applied pressure loads to the tanks would decrease the weight of the fairings but increase the axial load in the tanks. The effect of this change has not been investigated.

Table 4. Orbiter weight controlled by failure mode.

Failure Mode	Weight, lb
Minimum gage	22,920
Panel buckling	5,492
Compressive yield	7,228
Yield	8,336
Ultimate	14,510
Overall buckling	2,800

The calculated weights for components of the strength-designed orbiter model are shown in Appendix C. The unit weight of the orbiter LO2 tank is similar to that of the booster. The unit weight of the orbiter LH2 tank is 3 percent heavier than the booster LH2 tank despite the fact that it is lightly loaded relative to the booster LH2 tank, which must support the weight of the LO2 tank. The higher weight of the orbiter LH2 tank is largely due to the dual-lobe configuration that requires a center web structure between the lobes. The LH2 tank web is over 10 percent of the total tank weight. The intertank section is significantly lighter in unit weight (~50 percent) than the booster intertank because it supports only the LH2 tank, which is about one-fifth the weight of the LO2 tank, and because it is constructed of Gr/Pi rather than Al/Li. The weight of the fairings over the valleys between the two lobes of the propellant tanks is approximately 1900 lb without consideration for buckling and is a disadvantage of employing a dual-lobe tank shape.

Conclusions

A structural analysis of a two-stage fully reusable Advanced Manned Launch System (AMLS) vehicle has been made using near-term material technology. The study indicates that the concept is feasible. Because fully reusable launch vehicle concepts can be very sensitive to weight growth, it is very important to perform detailed structural analyses as early as possible in the design cycle to evaluate the effect of evolutionary material technologies and to reduce program risk.

The loading condition that sizes most of the structural weight is the impact caused by the sudden release of vehicle hold-downs that removes, instantaneously, a restraining force equal to 30 percent of the vehicle's weight. Elimination or relaxation of vehicle hold-down requirements could significantly reduce the vehicle structural weight but would complicate vehicle control requirements at lift-off. The effect of reducing the impact load has not been determined.

The loads causing buckling of the strength-sized structures, determined from global bifurcation buckling analyses, were less than the applied loads. A resizing procedure was developed and applied to increase the buckling strength to a satisfactory level. Global buckling was found to be insignificant on the booster; however, resizing the structure to consider global bucking impacts on the orbiter increased the total structural weight by 5 percent.

A large portion of both the booster and orbiter structure is lightly loaded. Twenty percent of the booster structure and 40 percent of the orbiter structure by weight uses minimum-gage materials.

The dual-lobe main propellant tanks employed on the orbiter added significant structural weight that would not be required for cylindrical tanks. The central web in the orbiter liquid hydrogen tank is over 10 percent of the total tank weight.

In addition the weight of the fairings required between the two lobes of each tank is approximately 1900 lb, constituting over 3 percent of the vehicle structure.

An assumption was made that simple structural arrangements, primarily separation of tank pressure containment and wing bending restraint, would benefit cryogenic tank life. Verification of this assumption and determination of other factors affecting cryogenic tank life are necessary before building such a concept.

References

1. Stone, H. W. and Piland W. M., "An Advanced Manned Launch System Concept," IAF Paper 92-0870, Sept. 1992.

2. Freemen, D. C.; Talay, T. A.; Stanley, D. O.;, and Wilhite A. W., "Design Options for Advanced Manned Launch Systems," AIAA Paper 90-3816, Sept. 1990.

3. McMillin, M. L., et al, "A Solid Modeler for Aerospace Vehicle Preliminary Design," AIAA Paper 87-2901, Sept. 1987.

4. Brauer, G. L.; Cornick, D. E.; and Stevenson, R., "Capabilities and Applications of the Program to Optimize Simulated Trajectories," NASA CR-2770, Feb. 1977.

5. Divan, P. E., "Aerodynamic Analysis System for Conceptual and Preliminary Analysis from Subsonic to Hypersonic Speeds," AIAA Paper 80-1897, Aug. 1980.

6. Anon., "PATRAN Plus User's Manual," Release 2.3 Pub. 2191020, PDA Engineering, Costa Mesa, CA., July 1988.

7. Whetstone, W. D., "Engineering Analysis Language Reference Manual," EISI, San Jose, CA., July 1983.

8. Cerro, J. A. and Shore, C. P., "EZDESIT, A Computer Program for Structural Element Sizing and Vehicle Weight Prediction," NASA TM-101649, 1990.

9. Naftel, J. C. and Powell, R. W., "Aerodynamic Separation and Glideback of a Mach 3 Staged Booster," AIAA Paper 90-0223, Jan. 1990.

10. Stanley, D. O., et al, "Conceptual Design of a Next-Generation, Fully Reusable Manned Launch System," AIAA Paper 91-0537, Jan. 1991.

APPENDIX A

Buckling Resizing Method

The finite-element solution of the bifurcation buckling problem is expressed as:

$$[K]\{\phi\} + \lambda [Kg]\{\phi\} = 0$$

where K is the global stiffness matrix, ϕ is the buckling mode shape (eigenvector), λ is the buckling eigenvalue and Kg is the global geometric stiffness matrix. While there are multiple eigenvalues and modes, usually it is desired to stiffen the structure so that the lowest buckling eigenvalue is greater than unity for the maximum applied load. If the equation is pre- and post-multiplied by a single mode shape,

$$\{\phi_1\}^T [K]\{\phi_1\} + \lambda_1 \{\phi_1\}^T [Kg]\{\phi_1\} = 0$$

using the usual normalization procedure where the $\{\phi_1\}^T [Kg] \{\phi_1\}$ term is -1.0, then the $\{\phi_1\}^T [K] \{\phi_1\}$ term becomes the eigenvalue. The $\{\phi_1\}^T [K] \{\phi_1\}$ term is also twice the strain energy (SE) caused by a displacement in that shape. This relationship may be used to develop a resizing algorithm by recognizing that in a simple column (a statically determinate structure with a fixed load distribution) the buckling load is directly proportional to the bending stiffness of the column.

The pre- and post-multiplied $\{\phi_1\}^T [K] \{\phi_1\}$ term is the summation of all the individual element stiffness terms (Ke), also pre- and post-multiplied, where n is the number of elements:

$$\{\phi_1\}^T [K]\{\phi_1\} = \sum_{i=1}^{n} \{\phi_1\}^T [Ke_i]\{\phi_1\}$$

hence, the elements having the largest strain energy for a given mode shape contribute the most to the corresponding eigenvalue. Because weight is the value to be minimized, it would appear that the elements having the largest strain energy-per-unit of weight will be most effective in increasing the buckling value for a given weight increase.

This procedure permits resizing of elements in the same way as fully stressed design does for strength-sizing, but the sizing increment depends on the summation of element changes. The approach used is to normalize the elemental strain-energy densities for a mode shape, producing a data set (N) having values from 0.0 to 1.0, and assume a distribution for modification based on the normalized values. The distribution used herein assumes that each element will be incremented in dimension in proportion to its normalized strain-energy density. To prevent elements with small contributions from being incremented, the distribution may be truncated by neglecting elements below a minimum value.

An estimate of the eigenvalue change due to the application of the increments is:

$$\Delta\lambda = 2\,(SE_{max}) \sum_{i=1}^{n} (N_i^2)$$

where SE_{max} is the strain energy of the element having the largest strain-energy density. Because the estimated eigenvalue change is probably not the same as the difference between the desired value (DV) and the existing value, a scaling factor must be applied to the increments.

The magnitudes of the elemental changes are scaled using a factor equal to the required eigenvalue change divided by the change caused by the unscaled increment:

$$\text{Scale factor} = (DV - \lambda_1) / \Delta\lambda$$

The required change is the difference between the desired value and the existing value. The desired value may be 1.0, or somewhat larger, because the resizing process causes the lower eigenvalues to become closely spaced and there may be a possibility of interaction. The magnitude of the maximum elemental change is limited by a pre-selected value (move limit) and other changes are scaled accordingly. Changes for several modes, caused by either a single loading with multiple eigenvalues less than 1.0 or several loadings with eigenvalues less than 1.0, may be made by summing the individual changes and imposing move limits on the summed changes.

The advantages of the method are that it is relatively simple, does not require additional optimization programs, considers individual elements, and requires only the extraction and scaling of the strain-energy densities after the buckling calculations. Furthermore, the distribution of strain-energy density between axial and bending components may be used to assist in element sizing. At the expense of a larger number of elements, individual ply orientations may be evaluated in a composite structure. The disadvantages of the method are that the analyst must decide how to apply it and interact with the solution process to determine when convergence has occurred. Also, because the process is sequential in nature, there is no guarantee of obtaining an optimum solution.

APPENDIX B

Booster Strength-Sized Weights

Component	**Study Weight, lb**	**Scaled ET Weight, lb**
Liquid oxygen tank	6,751	8,336
Shell structure	3,027	4,645
Aft ring	732	503
Aft dome	1,435	1,631
Slosh baffles	1,557	1,557
Intertank	4,667	6,506
Skin	3,436	4,194
Attachment flange	0	380
Frames	1,089	931
Connection structure	142	646
Miscellaneous	0	355
Liquid hydrogen tank	15,919	18,003
Forward dome	979	1,161
Barrel skin	8,801	11,675
Barrel frames	5,073	3,446
Aft dome	1,066	1,721
Wing	8,846	
Upper skin	2,409	
Lower skin	2,209	
Ribs and spars	2,170	
Center section	2,058	
Aft fuselage	12,199	
Skin	4,319	
Frames	2,672	
Thrust structure	3,474	
Wing fairing	1,734	
Booster/orbiter connection	1,445	
Total structure	**49,827**	

APPENDIX C

<u>Orbiter Strength-Sized Weights</u>

Component	**Study Weight, lb**
Liquid hydrogen tank	19,595
Forward dome	400
Barrel skin	11,504
Barrel frames	6,100
Aft dome	1,591
Intertank	3,063
Skin	2696
Frames	367
Liquid oxygen tank	8,127
Forward dome	1,025
Barrel skin	2,950
Barrel frames	1,058
Aft dome	1,537
Slosh baffles	1,557
Fairings	4,494
Liquid hydrogen tank	1,405
Liquid oxygen tank	488
Wing	2,601
Nose section	1,657
Nose cap	104
Front gear attachment	13
Skin	1,321
Frames	219
Wing	13,424
Upper skin	3,041
Lower skin	3,035
Ribs and spars	2,711
Center section	4,637
Aft fuselage	7,024
Skin	3,273
Frames	968
Thrust structure	2,783
Total structure	**57,384**