Old Dominion University Research Foundation

136P

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

95594
P-136

GEOMETRIC MODELING FOR COMPUTER AIDED DESIGN

By

James L. Schwing, Principal Investigator

Progress Report
For the period ended March 31, 1992

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665

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

June 1992

ANALYTIC REVIEW FORM

ER
(Initials)

Primary Record IPS # 95594

___ Document should not receive analytic treatment

SUBSIDIARY ADD

Document page range	___	to	___	New subsidiary # ___
Document page range	___	to	___	New subsidiary # ___
Document page range	___	to	___	New subsidiary # ___

SUBSIDIARY DELETE/CORRECTION

Subsidiary #_____

(IPS#____________)

___ Delete

Reason: ___ limited technical content
___ no separate authorship
___ context dependent

___ Adjust paging
___ Other ____________________

____

Subsidiary #_____

(IPS#____________)

___ Delete

Reason: ___ limited technical content
___ no separate authorship
___ context dependent

___ Adjust paging
___ Other ____________________

____

Subsidiary #_____

(IPS#____________)

___ Delete

Reason: ___ limited technical content
___ no separate authorship
___ context dependent

___ Adjust paging
___ Other ____________________

____

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 March 31, 1992

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665

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**

June 1992

1. Introduction

The primary goal of this grant has been the design and implementation of 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 and Computer Sciences Corp. This has resulted in the development of several packages and design studies. These include two software tools currently used in the conceptual level design of aerospace vehicles. These tools are SMART, the Solid Modeling Aerospace Research Tool, and EASIE, the Environment for Software Integration and Execution. Work under this grant also includes contributions to the design studies of orbital vehicles, specifically, the HL-20.

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. A more detailed review of the capabilities of SMART can be found in [1]. This describes the necessary characteristics built into SMART which allow it to be used efficiently as a front end geometry processor for other analysis packages.

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. A more detailed review of the characteristics of the EASIE system can be found in [2].

As conceptual aerospace design move to consider future vehicles several factors stand out. First, conceptual designers face a pressing need to enhance their analysis capabilities as traditional formulae and historical data are exceded by new conditions and requirements. The effort to generate new formulae and tables will proceed by the application of higher order analysis packages such as EAL and PATRAN. Geometric input for structural development in such packages is tedious at best.

Secondly, many modern analysis codes, such as POST, provide an excellent analysis capability with a high degree of flexibility. Flexibility is further added to the design system by providing design engineers with tools to interface analysis codes together (the tools of EASIE for example). The price for this high degree of flexibility is two fold. Within a given program the high degree of flexibility leads to complex data structures. The user of the program is made responsible for the creation and proper formatting of the input data file. The user is also responsible for tracking the definition of the appropriate sets of input parameters. In addition, allowing users to freely define execution sequences of analysis programs adds a high degree of interdependence in the definition of data items. Consistency and integrity of the data is currently the responsibility of the user.

In what follows, this report will cover in further detail the advances made in each of the following areas.

- General consulting on the use and development of aerospace structural analysis codes.
- Continued the development of SMART routines for the generation of structural element generation and automatic grid generation preprocessing.
- Enhancing the design of a database interface for POST that allows easier definition of data and helps perform data consistency checks for the model.
- Build a prototype X-Window interface for EASIE.
- Initial work on the development of basic parallel algorithms.

ORIGINAL PAGE IS
OF POOR QUALITY

2. Use and Development of Structural Codes

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.

In order to provide the detailed understanding necessary to design and implement the integration of SMART and advanced structural analysis programs as well as providing insight into the analysis programs themselves, the services of an expert are required. This support has been supplied by Mr. James C. Robinson. The remainder of this section consist of Mr. Robinson's report on the project work he has condicted during this period.

The primary task during this period has been the analysis of the Rockwell developed configuration of the HL-20 vehicle. The existing HL-20 finite-element model was modified to conform to the Rockwell concept of the HL-20. This required the addition of a partial circular cylindircal pressure shell and a flat floor area. The existing upper part of the vehicle exterior forms the remainder of the pressure shell. Existing frames in the previous conformal shell model were removed from the access doors. Frames were modeled for the new pressure shell structure. The remainder of the conformal model was converted into access panels (large doors) on the upper surface and a frame stiffened, heatshield structure on the lower surface.

THe model was analyzed and resized for five loading conditions. Two loading conditions controlled the sizing of most of the structure. The first is the internal pressure case plus the 3-G axial acceleration of a normal launch. The second is an abort condition which subjects the vehicle to an 8-G axioal acceleration and a 10 psi over-pressure due to the explosion of the launch vehicle. The Rockwell concept with doors exterior to the pressure shell causes the external and internal pressure loadings to be supported by two different load paths. The considerable pressure loads in the vehicle exterior caused by the abort condition required that part of hte structure to be resized to resist over-all buckling of the vehicle. This capability was not present in the resizing program.

The creation and implementation of an algorithm to size a finite element model to prevent over-all buckling was the second task for this period. Rigorous mathematical programming methods exist that may be used to resize a structure for buckling but application at a level consistent with the strength resizing (approximately 3000 elements and five load cases) is not practical for preliminary design efforts. The method implemented uses EAL generated strain-energy-densities for the critical buckling mode shapes and several small AWK scripts to calculate new element sizes. While the method is heuristic in nature, it appears to provide a "hands-off" solution to the problem.

3. Enhancements for SMART

3.1 Determination of the Structural Requirements

One of the major achievments of this period of research was the determination of those requirements that structural engineers across the NASA-Langley base need in order to more readily carry out their model preparation. The requirements are contained in the document "System Software Requirements for SMART - Vehicle Structure Modules." The document is included in Appendix A. This subsection will conclude with a summary of the document.

This document specifies the functional requirements for software components which address the geometric and data modelling needs of the aerospace structural engineer. The modules are to be included as part of the

(SMART) package. Hereafter, these software components will be referred to as SMART - Vehicle Structure Module (VSM). The purpose of this document is to state precisely WHAT the SMART Structures modules will do, without consideration as to HOW it will be done.

The software requirements document is intended to be used by the following groups: the SMART Development Team, structural engineers in VAB, other interested structural engineers and SMART users at NASA LaRC, and the design and implementation group at Old Dominion University. SMART provides conceptual designers with a rapid vehicle geometry prototyping capability. Of current interest is the definition and implementation of those characteristics which would provide the design engineer with a more effective and efficient tool for building structural models. Construction of such models is currently a bottleneck toward carrying out the analysis process. The goal of SMART VSM is to address this bottleneck.

The SMART VSM modules will be a set of software tools designed to aid in the development of geometric and data models to be used in the structural analysis of aerospace vehicles. SMART VSM will provide the following general capabilities:

- creating and editing structural elements for the wing and fuselage of a given aerospace vehicle,
- integrating wing and fuselage structural assemblies,
- integrating tail and fuselage structural assemblies,
- remapping of aerodynamic loads data in a manner consistent with the developed structural model,
- applying point and area based loads to the model, and
- preparing loads data for visual presentation.

The requirements listed in this document are currently in the process of implementation. Completion of the implementation for testing will be by the end of June, 1992, with final implementation expected by the end of the summer.

3.2 Design of Frames and Bulkheads

During the most recent term of this project, one of the enhancements made in the structural modeling capabilities of SMART is that of the desing of frame and bulkhead components. This work was carried out by Susan Schwartz and served as her Master's project at Old Dominion. The text of her project is provided in Appendix B.

3.3 SMART Related Publications

During the period of this grant, joint work with members of the VAB and the principal investigator lead to the following publication.

- with W. Engelund and C. Cruz, "Conceptual Level Aerodyanmic Heating Predictions Using the Aerodynamic Preliminary Analysis System (APAS)" - *Proceedings of AIAA Aircraft Design Conference*, AIAA-91-3087, September 1991.

4. Enhancements for EASIE

4.1 Database Interface for POST

This section describes the work during this period on the database interface to POST. As described in the introduction, many modern analysis codes provide an excellent analysis capability with a high degree of flexibility. Within a given program the high degree of flexibility leads to complex data structures. The user of the program is made responsible for the creation and proper formatting of the input data file. The user is also responsible for tracking the definition of the appropriate sets of input parameters.

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 [3] into a proto-type which applies these techniques to the parameter variables of POST. User reaction has been extremely favorable. The next stage has been to adapt this proto-type for use with tabular variables and to enhance the data interdependence factors described above. The work is being condicted by students Shawn Casey and William Denny. The enhancements are projected for completion by the end of the summer 1992.

4.2 X-Windows and EASIE

Now that version 1 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.

With this in mind, work under this grant has developed a new user interface for EASIE. This work has been carried out as Master's projects by students Ya-Chen Kao and Chia-Lin Tsai. They developed MOTIF based interfaces for the ADE and CCE modes of execution. The text of their project reports is included in Appendicies C and D.

4.3 EASIE Related Publications

During the period of this report, work with members of VAB and ACD and the principal investigator lead to the following publication.

- with K. Jones, L. Rowell and A. Wilhite, "Environment for Application Software Integration and Execution" - *Procedings of the 7th ASCE Conference on Computing and Database*, May 1991.

5. 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 algoritm 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.

During the period of this grant, the principal investigators and a doctoral student, Jingyuan Zhang, have continued to be remarkably successful in the development of such baseline algorithms. Below is a list of refereed publications related to this work.

- "On the Power of Two-Dimensional Processor Arrays with Reconfigurable Bus Systems" - *Parallel Processing Letters* September 1991, v 1, no 1, pp 29 - 34.
- "An Optimal Encoding and Decoding Algorithm for Trees" - accepted by *International Journal of Foundation of Computer Science*; preliminary version - *Procedings of the 19th Annual ACM-CSC*, Mar. 1991, pp 1-10.
- "Integer Problems on Reconfiguratble Meshes" - accepted by *Journal of Computer ans Software Engineering*; preliminary version - *Proceedings of the 29th annual Allerton Conference*, Oct. 1991, pp. 811 - 820.
- "Fundamental Algorithms on Reconfigurable Meshes" - *Proceedings of the 29th Annual Allerton Conference*, Oct. 1991, pp 821-830.
- "A fast Adaptive Convex Hull Algorithm on Two-Dimensional Processory Arrays with a Reconfigurable Bus Systems" - *Proceedings of the 3rd NASA Symposium on VLSI Design*, Nov. 1991, pp. 13.2.1-13.2.9 .
- "Sorting in O(1) Time on a Reconfigurable Mesh of Size nxn" - *Parallel Computing: From Theory to Sound Practice, Procedings of EWPC'92*, Plenary Address, IOS Press, pp. 16 - 27, 1992.
- "Fast Mid-level Vision Algorithms on Reconfigurable Meshes" - *Parallel Computing: From Theory to Sound Practice, Proceedings of EWPC'92*, IOS Press, pp. 188 - 191, 1992.

6. References

1. *A Solid Modeler For Aerospace Vehicle Preliminary Design*, M.L. McMillan, J.J. Rehder, A.W. Wilhite J.L. Schwing, J.L. Spangler, and J.C.Mills, presented to AIAA conference on CAD Modeling, August 1987.

2. *Software Tools for the Integration and Executions of Multidisciplinary Analysis Programs*, L Rowell, J. Schwing, and K. Jones, AIAA-88-4448, Sept., 1988.

3. *Data Management Interface for POST*, J. Schwing, final report for NASA Task NAS1-18584-50, July 1989.

Appendix A

61

N92-27830

p. 38

System Requirements Specification for SMART Structures Mode

1. Introduction

1.1 Purpose

This document specifies the functional and informational requirements for software modules which address the geometric and data modelling needs of the aerospace structural engineer. The modules are to be included as part of the Solid Modeling Aerospace Research Tool (SMART) package developed for the Vehicle Analysis Branch (VAB) at NASA Langley Research Center (LaRC). Hereafter, these modules will be referred to as SMART Structures. The purpose of this document is to state precisely WHAT the SMART Structures modules will do, without consideration as to HOW it will be done. Each requirement is numbered for reference in development and testing.

1.2 Scope

This software requirements document is intended to be used by the following groups: the SMART Development Team, structural engineers in VAB, other interested structural engineers and SMART users at NASA LaRC, and the design and implementation group at Old Dominion University. SMART provides conceptual designers with a rapid vehicle geometry prototyping capability. Of current interest is the definition and implementation of those characteristics which would provide the design engineer with a more effective and efficient tool for building structural models. Construction of such models is currently a bottleneck toward carrying out the analysis process. The goal of SMART Structures is to address this bottleneck.

The SMART Structures modules will be a set of software tools designed to aid in the development of geometric and data models to be used in the structural analysis of aerospace vehicles. SMART Structures WILL provide the following general capabilities:

- creating and editing structural elements for the wing and fuselage of a given aerospace vehicle,
- integrating wing and fuselage structural assemblies,
- integrating tail and fuselage structural assemblies,
- remapping of aerodynamic loads data in a manner consistent with the developed structural model,
- applying point and area based loads to the model, and
- preparing loads data for visual presentation.

SMART Structures WILL NOT provide the following capability:

- generation of the geometric surfaces defining any of the internal or external assemblies for wing, fuselage, or tank surfaces since such capabilities are either currently or shortly will be supplied by other SMART modules.

1.3 Terminology and References

Technical terms used in this document are common terms used in the engineering design of aerospace vehicles. Complete definitions can be found in any standard text on the topic, for example [2,3]. Figures 1-4 below illustrate the typical naming and placement of structural elements in aerospace vehicles.

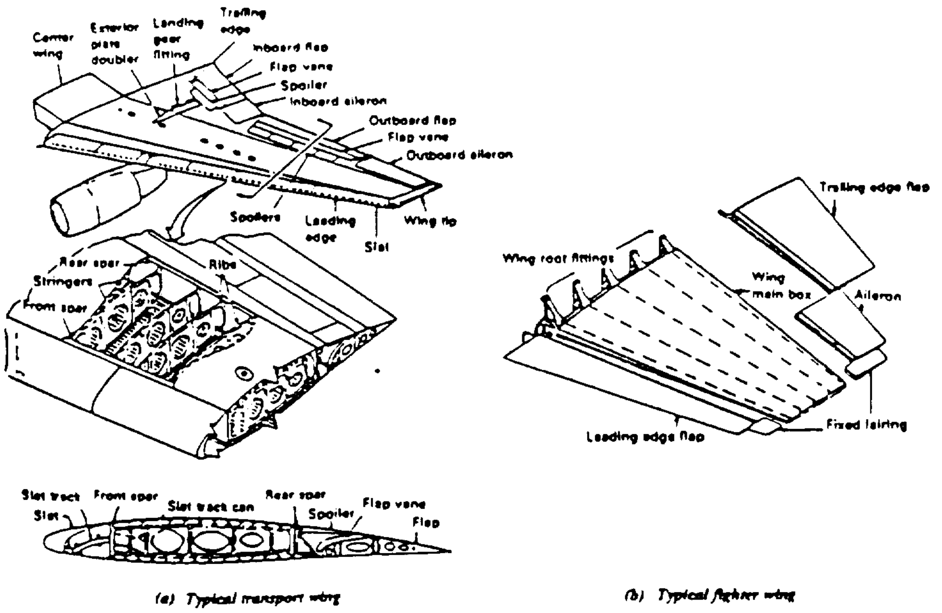


Figure 1
Wing Structural Components

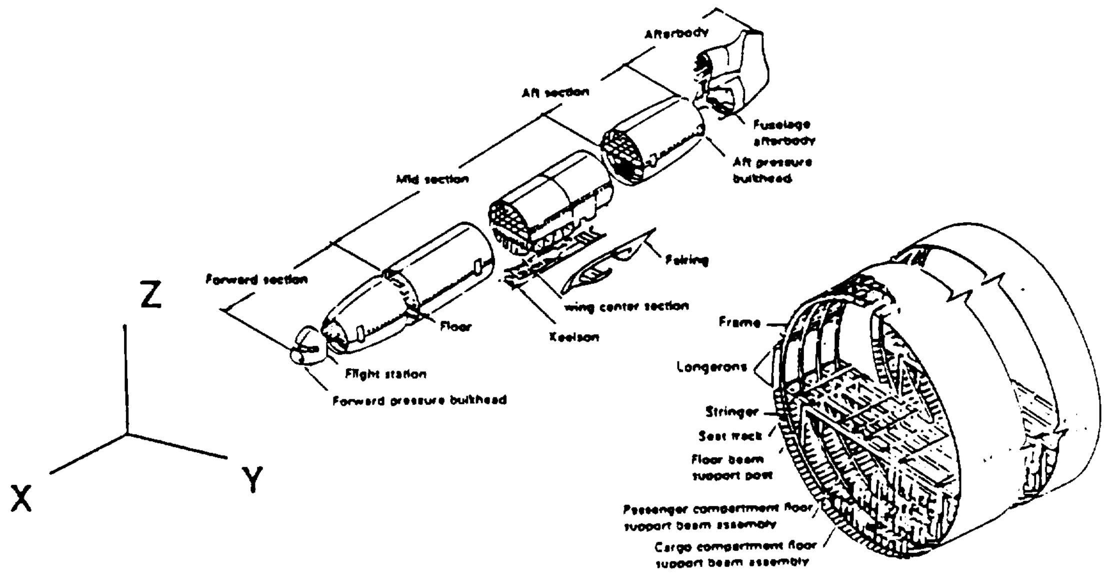


Figure 2
Fuselage Structural Components

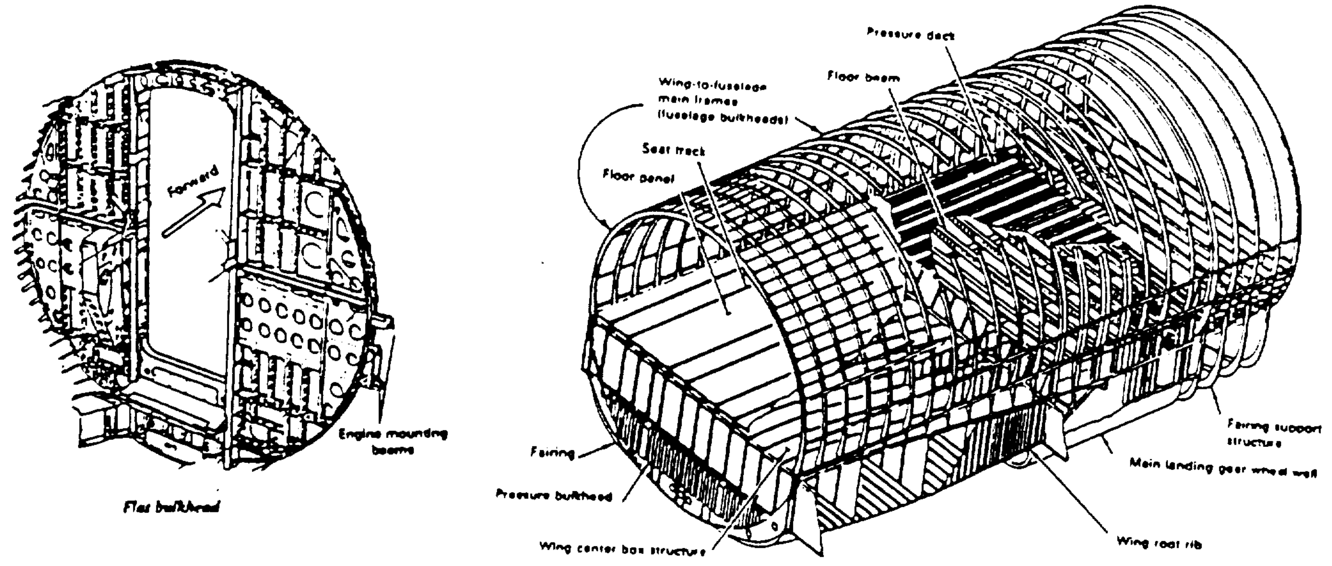

Engine mounting beams
Flat bulkhead
Forward
Pressure deck
Floor beam
Wing-to-fuselage main frames (fuselage bulkheads)
Seat track
Floor panel
Fairing
Pressure bulkhead
Wing center box structure
Fairing support structure
Main landing gear wheel well
Wing root rib

Figure 3
Aircraft Frame Structure

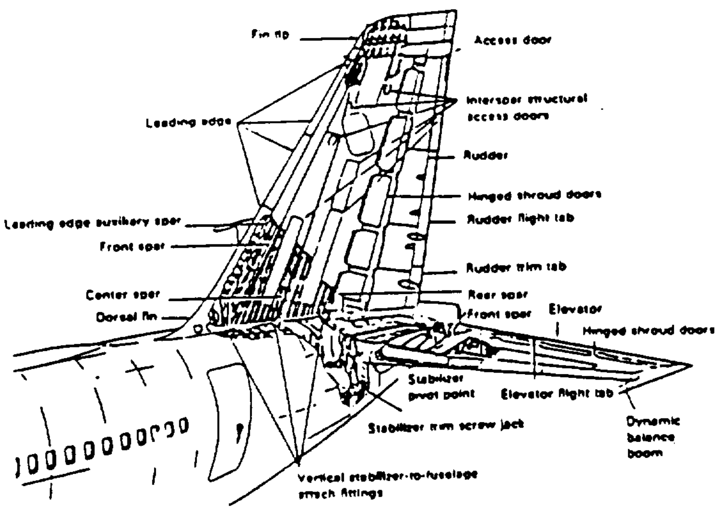


Figure 4
Tail Structural Components

Additional terminology used in this document.

- *consistent* - when applied to surface geometry and data patches relative to the structural model means that the edges of the patches have been defined to correspond to structural elements immediately beneath the surface.
- *patch* - a bicubic functional representation of a geometric or data surface using two parameters.
- *grid point* - the 1/3 points of the patches that define the geometry.
- *surface patch divider* - a non-structural entity used to separate surface patches as structural entities such as ribs and spars separate patches.

The following references are provided as background in basic aerospace design and in the software packages SMART, PATRAN and APAS.

[1] *Aerodynamic Preliminary Analysis System II (APAS), Part II -- User Manual*, Rockwell International Corporation, April 1981.

[2] *Basic Science for Aerospace Vehicles - 4th Edition*, Northrup Institute for Technology, McGraw-Hill, 1972.

[3] Colvin and Colvin, *Aircraft Handbook*, McGraw-Hill, 1929.

[4] McMillan, Rehder, Wilhite, Schwing, Spangler and Mills, *A Solid Modeler for Aerospace Vehicle Preliminary Design*, AIAA paper 87-2901.

[5] *PATRAN Plus User Manual, Vols. I & II*, PDA Engineering, 1988.

[6] *SMART User Reference Guide*, Vehicle Analysis Branch, NASA Langley, 1991.

1.4 *Overview of the Software Requirements Document*

Section 2 of this document consists of a general overview of SMART Structures and includes the following subsections:

- a perspective of how SMART Structures relates to the existing and proposed development and analysis process in VAB,
- a general look a SMART Structures functions,
- user characteristics,
- general constraints, and
- assumptions and dependencies.

Section 3 of this document consists of a detailed listing of the requirements for SMART Structures. Where applicable, every requirement will address each of the following areas:

- introduction,
- inputs,
- processing, and
- outputs.

Finally, Section 3 will end with a listing of the external interface requirements including user interface and software interface requirements.

2. General Description

2.1 SMART Structures: A Perspective

The SMART Structures mode will be a assembly of the SMART geometric design system. The main objectives of SMART Structures are the definition and editing of structural elements for aerospace vehicles and, for further structural analysis, the preparation of geometric and data models consisten: ·ith the structural elements. Initial geometric surface descriptions of the aerospace vehicles will be prepared through other design modules in SMART [4,6]. The basic interface to the rest of SMART is through the SMART data tree. It is assumed that the aerospace vehicle assemblies, to which structure will be added, have been defined in SMART and have bicubic surface definitions as described by [4,6]. Structural elements prepared by SMART Structures will follow this same format and will be inserted into the SMART data tree and will thus be available to SMART modules such as visualization.

SMART Structures will provide tools for the definition and editing of both point and area load data. In addition, SMART Structures will be able to read data files created by APAS which contain loads and other aerodynamic analysis information, such as heating. Once geometry and data surfaces have been prepared that correspond to the underlying structural elements, SMART Structures will be able to write the results to PATRAN neutral files. Figure 5 below shows the interaction of various SMART modules and the APAS and PATRAN analysis programs while figure 6 illustrates the proposed data flow for the structural analysis process in VAB. SMART Files Primitives represents those functions in SMART available for defining geometric surfaces, both general(box, sphere, etc.) and aero-specific (wing, tank, etc.). SMART Files represents those functions in SMART that aid in the ·ading and writing of various types of data files.

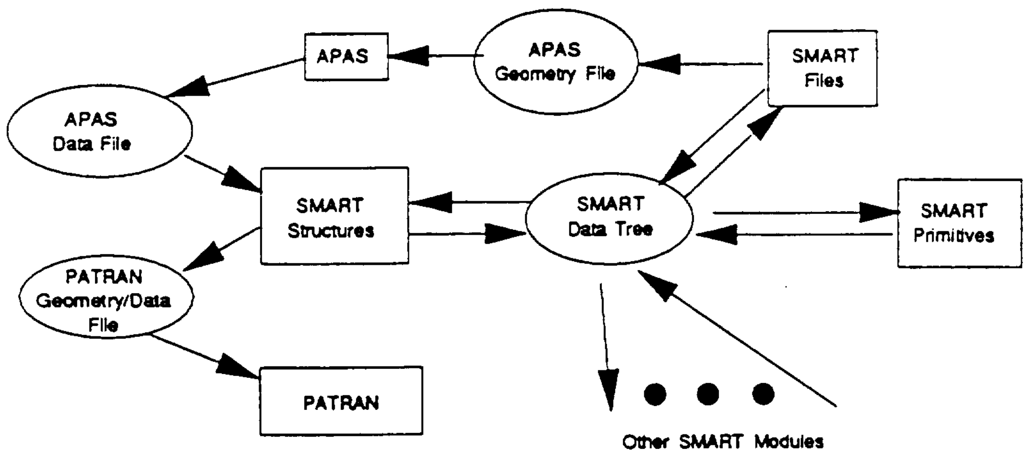


Figure 5
Data Flow Relative to Smart Structures

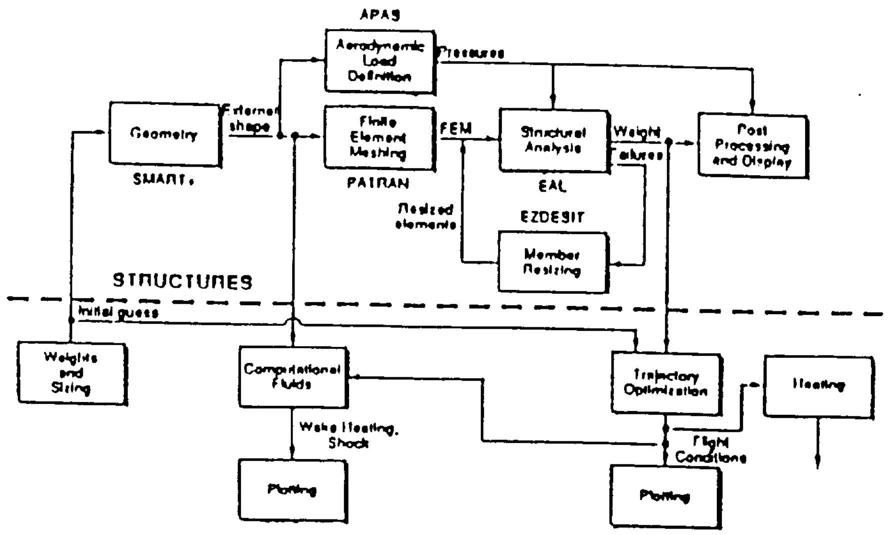


Figure 6
Proposed Data Flow for Conceptual Vehicle Analysis in VAB

2.2 *SMART Structures Functions*

SMART Structures functionality is defined by the following groups of requirements:

Wing-type Elements - Definition and Editing
Note: wing-type includes consideration of tail, pylon, and other wing-like structures.

1. Wing Box - Leading Edge Requirements
2. Wing Box - Trailing Edge Requirements
3. Wing Box - Root and Tip Rib Requirements
4. Rib Requirements
5. Spar Requirements
6. Multiple Wing Assembly Integration Requirements
7. Wing Section Cutout Requirements
8. Wing Output Requirements

Fuselage-type Elements - Definition and Editing
Note: fuselage-type include consideration of tank and other fuselage-like structures.

9. Fuselage Assembly Generation Requirements
10. Cross Section Generation Requirements
11. Ring Frame Requirements
12. Bulkhead Requirements
13. Longeron, Keel, and Beam Requirements
14. Fuselage Output Requirements

Assembly Placement and Integration

15. Wing - Fuselage Placement and Integration
16. Wing - Tail placement and Integration

Load Definition and Visualization
17. Point Load Requirements
18. Path and Area Load Requirements
19. Analysis-Generated Load Requirements
20. Load Output Requirements

External Interfaces
21. User Interface Requirements
22. Software Interface Requirements

2.3 *User Characteristics*

Users of the SMART system are aerospace design engineers. All users will be familiar with SMART as a geometric design tool. Further, users of SMART Structures will have knowledge of structural design and analysis programs. This includes knowledge of design parameters and input and output variables for associated analysis programs such as APAS and PATRAN.

2.4 *General Constraints*

SMART Structures will be written in the "C" language, using the SMART graphics user interface developed for the Silicon Graphics (SGI) IRIS 4D Workstation using SGI's GL graphics library. As graphics will be used, the SMART Structures module will only run from the console. Note that this is consistent with the current execution of SMART. It is further noted that this also implies the existence of a pointing device (the mouse). No other hardware will be needed.

2.5 *Assumptions and Dependencies*

Software developed for SMART Structures is dependent on the operating system and libraries provided by SGI with their IRIS Workstation products and graphics display and GUI modules already developed for SMART. Changes in these underlying systems may result in changes in the operation or in the appearance of SMART Structures.

SMART Structures will provide software tools to define and edit structural elements for all possible SMART-produced wing geometries. At this time, the full range of wing definitions supported by SMART is yet to be determined. It is not essential for the purpose of this document that all such geometries be currently defined; however, it is assumed that any NEW wing geometries will satisfy the following characteristics.

1. Upper and lower surfaces will be defined by an array of bicubic patches consistent with the SMART geometry format.
2. The SMART data tree will contain sufficient information so that a planform view of the wing can be determined.
3. When previously defined by the SMART Wing modules, the SMART data tree will contain sufficient information to determine the placement of flaps and other cutouts in the wing surface.

It is assumed that fuselage geometries will satisfy the following characteristics:

1. There may be several fuselage assemblies defined, for example, fore, center, and aft assemblies.
2. Surfaces are defined by an array of bicubic patches consistent with the SMART geometry format [6].

3. Functional Requirements

1. Wing Box - Leading Edge Requirements

1. General Requirements

The *wing box* is made up of the leading edge spar, the trailing edge spar and the root and tip ribs. This section and the two which follow describe the requirements for defining and editing these structural elements and the region immediately preceding the leading edge spar and following the trailing edge spar.

The purpose of this function is the generation of the geometry for the major structural element in each wing assembly behind the leading edge usually referred to as the *leading edge spar*. In addition, geometry for the leading edge rib elements within the leading edge is also generated. Figure 7 illustrates typical positions for the leading edge spar and leading edge ribs.

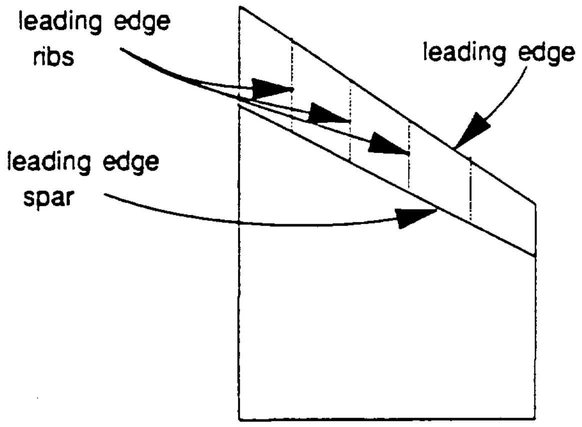


Figure 7
Leading Edge Spar and Rib Components

R1.1 The leading edge spar must be generated perpendicular to the wing planform.

R1.2 The extent of the leading edge spar must be defined by where it intersects the root and tip ribs and the upper and lower surfaces of the wing.

R1.3 The extents of the leading edge ribs must be defined by where they intersect the leading edge, the leading edge spar, and the upper and lower surfaces.

R1.4 The placement of the leading edge spar must be checked relative to the placement of the trailing edge spar, the two spars must not be allowed to intersect.

2. Input Requirements

R1.5 The leading edge spar may be DEFINED as parallel to the leading edge with the distance back from the leading edge determined by either

a. a percentage length of the root chord – 0% at the leading edge and 100% at the trailing edge, or

b. a measured distance behind the leading edge measured along the root chord which may be input either by value or by pointing.

R1.6 The leading edge spar may be DEFINED as skew to the leading edge, requires two inputs to fix its position behind the leading edge, determined by either

a. percentage of length of both the root and tip chords -- 0% at the leading edge and 100% at the trailing edge, or

b. measured distances behind the leading edge along the root and tip chords which may be input either by value or by pointing.

R1.7 The leading edge spar may be EDITED as parallel to the leading edge by updating current values as follows:

a. a percentage length of the root chord,

b. a measured distance behind the leading edge measured along the root chord which may be input either by value or by pointing.

R1.8 The leading edge spar may be EDITED as skew to the leading edge by allowing updates of either of two inputs to fix its position as follows:

a. percentage of length of either the root or tip chords,

b. measured distances behind the leading edge along either the root or tip chords which may be input either by value or by pointing.

If the leading edge spar was initially input as skew to the leading edge and later edited as parallel to the leading edge, it is first made parallel to the leading edge, intersecting the root chord at the same place as the initial skew leading edge spar.

R1.9 The leading edge spar may be DELETED.

Since much of the other internal structure depends upon the placement of the leading edge spar, editing the leading edge spar requires deleting this structure and redefining it. Since this may require extensive data reentry on the part of the user, the user will be notified of the destructive nature of this choice and asked to verify the intention to make this change.

R1.10 The leading edge rib elements may be DEFINED as parallel to the root chord and specified by:

a. equal spacing and the number of required ribs,

b. entering the value of the position of each desired rib,

c. pointing at the position of each desired rib.

R1.11 The leading edge rib elements may be DEFINED as perpendicular to the leading edge and specified by:

a. equal spacing and the number of required ribs,

b. entering the value of the position of each desired rib,

c. pointing at the position of each desired rib.

R1.12 The leading edge rib elements may be EDITED as parallel to the root chord and specified by:

a. equal spacing and updating the number of required ribs,

b. changing the value of the position of each desired rib,

c. dragging any desired rib to a new position.

R1.13 The leading edge rib elements may be EDITED as perpendicular to the leading edge and specified by:

a. equal spacing and updating the number of required ribs,

b. changing the value of the position of each desired rib,

c. dragging any desired rib to a new position.

Editing leading edge ribs initially defined as parallel to the root chord as perpendicular to the leading edge (or alternately initially defined as perpendicular to the leading edge and edited as parallel to the root chord) causes the ribs to be redefined immediately.

Editing non-equally spaced ribs as equally spaced causes the given number of ribs to be redistributed equally before editing.

R1.14 All leading edge rib elements may be DELETED by choosing a delete all function.

R1.15 The leading edge rib elements may be DELETED by individually pointing at each rib to be deleted.

3. Processing Requirements

R1.16 Surfaces will be generated to define a complete, separate leading edge assembly.

R1.17 Results are presented real-time on a both a planform view of the wing and a side view of the wing.

R1.18 Since users must define the leading edge spar prior to defining the leading edge ribs, this module notifies users of an attempt to define rib elements out of sequence.

R1.19 Surface patch dividers may be defined, edited and deleted like leading edge ribs.

2. Wing Box - Trailing Edge Requirements

1. General Requirements

The purpose of this function is the generation of the geometry for the major structural element in each wing assembly preceding the trailing edge usually referred to as the *trailing edge spar*. In addition, geometry for the trailing edge rib elements within the trailing edge is generated. Figure 8 illusrtates typical positions for the trailing edge spar and trailing edge ribs.

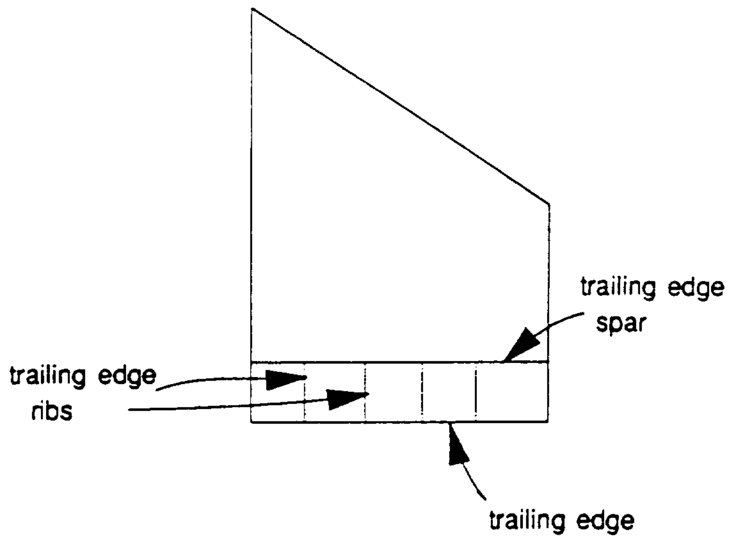


Figure 8
Trailing Edge Spar and Rib Components

R2.1 When previously defined, the wing planform will be drawn with outlines of flaps or other trailing edge aero-surfaces so that the placement of the trailing edge spar may be as accurate as possible.

R2.2 The trailing edge spar must be generated perpendicular to the wing planform.

R2.3 The trailing edge spar must be defined by where it intersects the root and tip chords and the upper and lower surfaces of the wing.

R2.4 The extents of the trailing edge ribs must be defined by where they intersect the trailing edge, the trailing edge spar, and the upper and lower surfaces.

R2.5 The placement of the trailing edge spar must be checked relative to the placement of the leading edge spar, the two spars must not be allowed to intersect.

2. Input Requirements

R2.6 The trailing edge spar may be DEFINED as parallel to the trailing edge with the distance back from the trailing edge determined by either

a. a percentage length of the root chord – 0% at the leading edge and 100% at the trailing edge, or

b. a measured distance preceding the trailing edge measured along the root chord which may be input either by value or by pointing.

R2.7 The trailing edge spar may be DEFINED as skew to the trailing edge, and requires two inputs to fix its position preceding the trailing edge, determined by either

a. percentage of length of both the root and tip chords -- 0% at the leading edge and 100% at the trailing edge, or

b. measured distances preceding the trailing edge along the root and tip chords which may be input either by value or by pointing.

R2.8 The trailing edge spar may be EDITED as parallel to the trailing edge by updating current values as follows:

a. a percentage length of the root chord,

b. a measured distance preceding the trailing edge measured along the root chord which may be input either by value or by pointing.

R2.9 The trailing edge spar may be EDITED as skew to the trailing edge by allowing updates of either of two inputs to fix its position as follows:

a. percentage of length of either the root or tip chords,

b. measured distances preceding the trailing edge along either the root or tip chords which may be input either by value or by pointing.

If the trailing edge spar was initially input as skew to the trailing edge, and later edited as parallel to the trailing edge, it is first made parallel to the trailing edge, intersecting the root chord at the same place as the initial skew trailing edge spar.

R2.10 The trailing edge spar may be DELETED.

Since much of the other internal structure depends upon the placement of the trailing edge spar, editing the trailing edge spar requires deleting this structure and redefining it. Since this may require extensive data reentry on the part of the designer, the designer will be notified of the destructive nature of this choice and asked to verify the intention to make this change.

R2.11 The trailing edge rib elements may be DEFINED as parallel to the root chord and specified by:

a. equal spacing and the number of required ribs,

b. entering the value of the position of each desired rib,

c. pointing at the position of each desired rib.

R2.12 The trailing edge rib elements may be DEFINED as perpendicular to the trailing edge and specified by:

a. entering the value of the position of each desired rib,

b. equal spacing and the number of required ribs,

c. pointing at the position of each desired rib.

R2.13 The trailing edge rib elements may be EDITED as parallel to the root chord and specified by:

a. equal spacing and updating the number of required ribs,

b. changing the value of the position of each desired rib,

c. dragging any desired rib to a new position.

R2.14 The trailing edge rib elements may be EDITED as perpendicular to the trailing edge and specified by:

a. equal spacing and updating the number of required ribs,

b. changing the value of the position of each desired rib,

c. dragging any desired rib to a new position.

Editing ribs initially defined as parallel to the root chord as perpendicular to the trailing edge (or alternately initially defined as perpendicular to the trailing edge and edited as parallel to the root chord) causes the ribs to be redefined immediately.

Editing non-equally spaced ribs as equally spaced causes the given number of ribs to be redistributed equally before editing.

R2.15 All trailing edge rib elements may be DELETED by choosing a delete all function.

R2.16 The trailing edge rib elements may be DELETED by individually pointing at each rib to be deleted.

3. Processing Requirements

R2.17 Surfaces will be generated to define a complete, separate trailing edge assembly.

R2.18 Results are presented real-time on a planform view of the wing and on a side view of the wing.

R2.19 Since users must define the trailing edge spar prior to defining the trailing edge ribs, this module notifies users of an attempt to define rib elements out of sequence.

R2.20 Surface patch dividers may be defined, edited and deleted like trailing edge ribs.

3. Wing Box - Root and Tip Rib Requirements

1. General Requirements

The purpose of this function is the generation of the geometry for the other major structural elements of any wing assembly, the root and tip ribs. Figure 9 illustrates a typical position for the root and tip ribs.

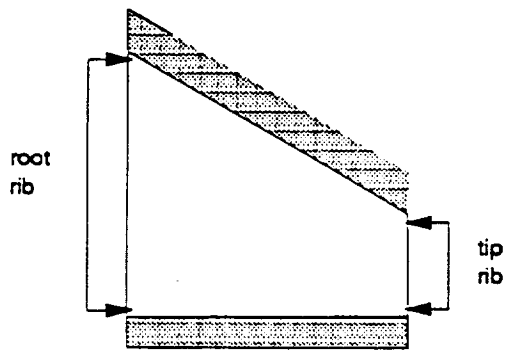


Figure 9
Root and Tip Rib Components

R3.1 The tip rib must be generated perpendicular to the plane of the wing planform.

R3.2 If the wing assembly being considered attaches to another assembly of the wing, the root rib must be generated perpendicular to the plane of the wing planform.

R3.3 If the wing assembly attaches to the fuselage, the root rib must be initially generated in a plane parallel to the plane of symmetry for the vehicle being defined. This angle can be found by adding the dihedral angle to the perpendicular of the wing planform.

It should be noted that this initial definition of the root rib may be modified by the requirements of wing-fuselage integration discussed below in the Processing Requirements of the wing-fuseslage section.

R3.4 Both of these ribs must contain the line segments in the wing planform defined by the root and tip chords.

R3.5 The extents of the root and tip ribs are defined by where they intersect the leading and trailing edges and the upper and lower surfaces.

2. Input Requirements

The requirements made for defining the root and tip ribs completely specify these elements, therefore no user input is required.

3. Processing Requirements

R3.6 Any processing required must be completed automatically once both the leading and trailing edge spars are defined.

4. Rib Requirements

1. General Requirements

The purpose of this function is the generation of the geometry for rib elements in the wing. Figures 10 and 11 illustrate typical positions for the placement of ribs in the wing box.

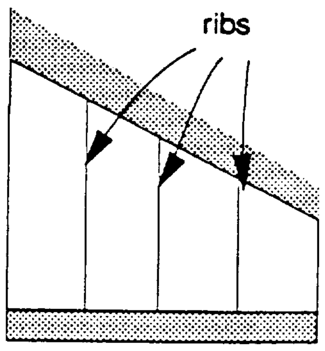


Figure 10
Ribs Defined Parallel to Root Rib

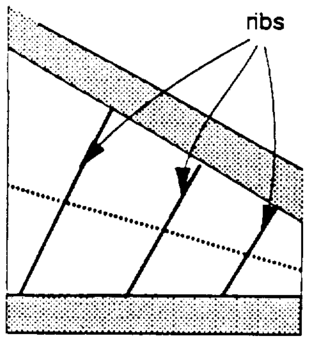


Figure 11
Ribs Defined perpendicular to the 50% Chord

R4.1 Rib structural elements must be generated perpendicular to the wing planform.

R4.2 The extents of the ribs are defined by where they intersect the wing box and by the upper and lower surface of the wing.

R4.3 When the spars are defined, the rib elements are subdivided where they intersect spar elements.

2. Input Requirements

R4.4 The rib elements may be DEFINED as parallel to the root chord and may be specified by

a. equal spacing and the number of required ribs,

b. pointing at the position of each desired rib.

R4.5 The rib elements may be DEFINED as perpendicular to a user-specified percentage chord and further specified by

a. equal spacing and the number of required ribs,

b. pointing at the position of each desired rib.

R4.6 The rib elements may be DEFINED by pointing to the wing box at the endpoints of the desired rib; this method will be referred to as "freehand."

R4.7 The rib elements may be DEFINED by entering the value of the endpoints of each desired rib.

R4.8 The rib elements may be EDITED as parallel to the root chord by

a. equal spacing and updating the number of required ribs,

b. dragging any desired rib to a new position.

R4.9 The rib elements may be EDITED as perpendicular to a potentially updated user-specified percentage chord by

a. equal spacing and updating the number of required ribs,

b. dragging any desired rib to a new position.

R4.10 The rib elements may be EDITED by dragging any endpoint of a rib to a new position.

R4.11 The rib elements may be EDITED by changing the value of any endpoint of a rib.

Editing ribs as parallel to the root rib (alternately as perpendicular to a percentage chord) not initially defined in that format causes the ribs to be redefined immediately in the format chosen for editing, i.e. the same number of ribs are redistributed equally spaced and parallel to the root rib (alternately perpendicular to the percentage chord).

R4.12 All rib elements may be DELETED by choosing a delete all function.

R4.13 The rib elements may be DELETED by individually pointing at each rib to be deleted.

3. Processing Requirements

R4.14 Results are presented real-time on a planform view of the wing.

R4.15 Validity of "freehand" input for ribs is checked to insure that ribs do not intersect except perhaps at the boundaries of the wing box.

The definition of rib and spar elements has a direct implication on the representation of elements for the wing surfaces as follows:

R4.16 Geometry elements in the wing surfaces will be redefined so that the boundaries of the surface patches matches to the boundary of an underlying rib or spar patch.

This geometry data is to be provided as input to analysis programs such as PATRAN [5]. Elements input into such analysis programs are either in the form of three- or four-sided elements.* The process of subdividing the wing box into ribs and spars can lead to elements with five or more sides which are not acceptable for further analysis.

R4.17 Once the process of defining ribs and spars is finished, each surface element must be reviewed for the number of sides generated. If five or more-sided elements exist, they must be identified to the user. The user must be placed in a mode that allows editing the rib or spar elements to correct this situation.**

R4.18 Surface patch dividers may be defined, edited and deleted like ribs. In addition, any section of a rib (between two spars) may be designated as a section of a surface patch divider. Conversely, any section of a surface patch divider may be designated as a rib section.

* Note: elements are *not* polygons; edges may be curved.

** Since the order of definition of ribs and spars is left to the user requirements R4.16 and 4.17 are placed with the processing requirements of both ribs (here) and spars (requirements R5.18 and R5.19).

5. Spar Requirements

1. General Requirements

The purpose of this function is the generation of the geometry for spar elements in the wing. Figures 12 and 13 illustrate typical positions for the placement of spars in the wing box.

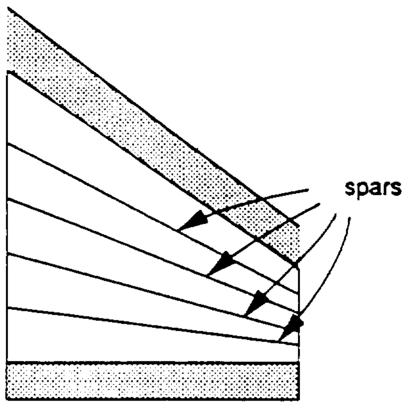


Figure 12
Spars Defined at Equal Percentages of the Root and Tip Chords

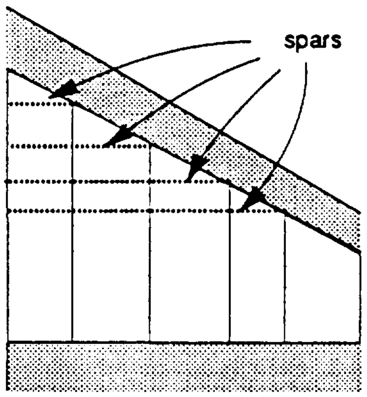


Figure 13
Spars Defined Perpendicular to Existing Ribs

R5.1 Spar structural elements must be generated perpendicular to the wing planform.

R5.2 The extents of the spars are defined by where they intersect the wing box and by the upper and lower surface of the wing.

R5.3 When the ribs are defined, the spar elements are subdivided where they intersect rib elements.

2. Input Requirements

R5.4 The spar elements may be DEFINED as endpoints placed at equally calculated percentages of the root and tip chords and further specified by the number of spars.

R5.5 The spar elements may be DEFINED as parallel to a user-specified percentage chord and further specified by

a. equal spacing and the number of required spars,

b. pointing at the position of each desired spar.

R5.6 The spar elements may be DEFINED as perpendicular to existing rib elements and further specified by

a. equal spacing and the number of required spars,

b. pointing at the position of each desired spar.

The choice of whether to define ribs or spars first is left to the user. Thus, if the user decides to define spars first, this technique is not appropriate and the user is asked to use another technique.

R5.7 The spar elements may be DEFINED by pointing at the endpoints of the desired spar; this method is referred to as "freehand."

R5.8 The spar elements may be DEFINED by entering the value of the endpoints of each desired spar.

R5.9 The spar elements may be EDITED as endpoints placed at equally spaced percentages of the root and tip chords and by updating the number of spars.

R5.10 The spar elements may be EDITED as parallel to a potentially updated user-specified percentage spar chord and further specified by

a. equal spacing and updating the number of required spars,

b. pointing at the position of each additional spar.

R5.11 The spar elements may be EDITED as perpendicular to existing rib elements and further specified by

a. equal spacing and updating the number of required spars,

b. pointing at the position of each additional spar.

R5.12 The spar elements may be EDITED by dragging any endpoint of a spar to a new position.

R5.13 The spar elements may be EDITED by changing the value of any endpoint of a spar.

Editing spars as endpoints at equally spaced percentages of the root and tip ribs (alternately either as parallel to a percentage chord or as perpendicular to existing ribs) not initially defined in that format causes the spars to be redefined immediately in the format chosen for editing, i.e. the same number of spars are redistributed with endpoints at equally spaced percentages of the root and tip ribs (alternately either as parallel to the percentage chord or as perpendicular to existing ribs).

R5.14 All spar elements may be DELETED by choosing a delete all function.

R5.15 The spar elements may be DELETED by individually pointing at each spar to be deleted.

3. Processing Requirements

R5.16 Results are presented real-time on a planform view of the wing.

R5.17 Validity of "freehand" input for spars is checked to insure that spars do not intersect except perhaps at the boundary of the wing box.

The definition of rib and spar elements has a direct implication on the representation of elements for the wing surfaces as follows:

R5.18 Geometry elements in the wing surfaces will be redefined so that the boundaries of the surface patches matches to the boundary of an underlying rib or spar patch.

This geometry data is to be provided as input to analysis programs such as PATRAN [5]. Elements input into such analysis programs are either in the form of three- or four-sided elements. The process of subdividing the wing box into ribs and spars can lead to elements with five or more sides which are not acceptable for further analysis.

R5.19 Once the process of defining ribs and spars is finished, each surface element must be reviewed for the number of sides generated. If five or more sided elements exist, they must be identified to the user. The user must be placed in a mode that allows editing of the rib or spar elements to correct this situation.*

R5.20 Surface patch dividers may be defined, edited and deleted like spars. In addition, any section of a spar (between two ribs) may be designated as a section of a surface patch divider. Conversely, any section of a surface patch divider may be designated as a spar section.

6. Multiple Wing Assembly Integration Requirements

1. General Comments

The purpose of this function is to allow the user to edit the structural elements of any wing assembly as necessary to produce a model that has elements consistent with the requirements for input to analysis programs such as PATRAN [5]. A summary of these requirements follow. Two assemblies which share a common rib with a common airfoil (or upper and lower surface), must have the same number of spar elements ending at the common boundary. In addition, corresponding elements from each of the assemblies must intersect at a single point on the common boundary. Figure 14 illustrates this condition.

* Since the order of definition of ribs and spars is left to the user, requirements R5.18 and R5.19 are placed with the processing requirements of both spars (here) and ribs (requirements R4.16 & R4.17).

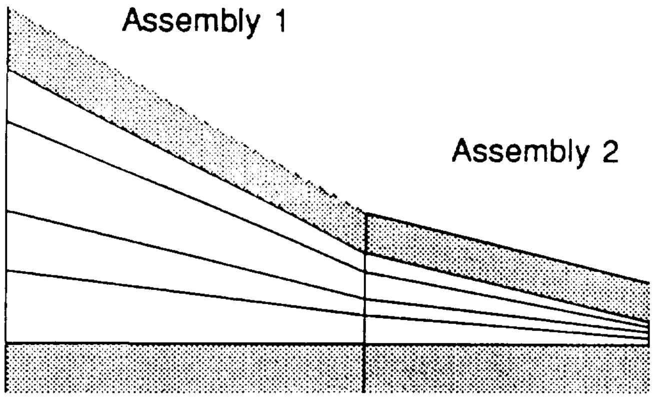


Figure 14
Integration of Wing Assemblies

It is NOT the function of SMART Structures to provide facilities to edit the chord elements of the two assemblies or their airfoils at intersection to obtain a match of these elements. It is assumed that this match is provided by the SMART Wing Generation module(s). It is the function of this requirement to provide editing facilities in order to facilitate the matching of spar elements at the common chord.

R6.1 It may be necessary for the designer to define assemblies in addition to those provided to SMART Structures. Such assemblies would be separated by a plane, defined perpendicular to the wing planfrom. The plane will be represented by a line segment drawn where this plane intersects the wing planform.

2. Input Requirements

R6.2 New assemblies may be defined by

a. selecting an existing rib or spar as the assembly boundary,

b. entering the values of the endpoints of the defining line segment on the boundary of the wing box,

c. pointing to the position of the endpoints of the defining line segment on the boundary of the wing box.

R6.3 Any assembly may be named or renamed.

R6.4 For MATCHING assemblies at a common boundary, identify a assembly for editing and use any of the spar or rib editing techniques described in the sections on Spar or Rib Requirements.

R6.5 Point to a spar or rib endpoint at the common chord and move that point so that it matches one of the spar or rib endpoints from the other assembly.

3. Processing Requirements

R6.6 Results are presented real-time on a planform view of the wing.

R6.7 Operations for the integration of wing assemblies require both the existence of two wing assemblies and the existence of structural elements for both of the assemblies. The existence of this information is checked and the user informed of the need to provide additional inputs when necessary.

7. Wing Section Cutout Requirements

1. General Comments

The purpose of this function is to allow the user to edit the structural elements of the wing assemblies as necessary to define a region of the wing to be cut out for special purposes such as a landing gear box. Let a spar (alternately rib) section be that section of a spar (rib) between two rib (spar) elements. It is assumed that the desired cutout will be bounded by spar and rib elements. Required editing may include the insertion or deletion of spar or rib elements.

2. Input Requirements

R7.1 Point at rib or spar section for deletion.

R7.2 Point to existing structural elements to define the endpoints of an additional rib or spar sections; this method will be referred to as "freehand."

7.3 Surface patches may be designated as missing or holes in the surface.

3. Processing Requirements

R7.4 Results are presented real-time on a planform view of the wing.

R7.5 Operations for the definition of structural elements for a wing cutout require the existence of structural elements in the wing. The existence of this information is checked and the user informed of the need to provide additional inputs when necessary.

R7.6 Validity of "freehand" input is checked to insure that inserted rib and spar section intersect existing ribs and spars only at the designated endpoints.

8. Wing Output Requirements

1. General Requirements

Since processing in SMART Structures is a time-consuming procedure, user input is captured and stored until the entire wing structure is defined. Upon finishing, one and two-dimensional geometry elements are created for both the wing surface and the user-defined structural elements. Elements in the surface are reformulated from the original SMART geometry so that the new surface patches are defined by the boundaries of the underlying structural elements.

R8.1 The resulting elements must be added back to the SMART data tree for use by other SMART modules, including visualization, and may be output as a PATRAN neutral file. Sufficient information is output to the SMART data tree so that the wing structures can be recalled, edited by SMART Structures and output again in edited form.

2. Input Requirements

R8.2 The user indicates that the editing of the structural model of the wing is complete.

3. Processing Requirements

Two types of elements need to be calculated, one- and two-dimensional.

R8.3 Two-dimensional elements are represented by bicubic surfaces.

R8.4 One-dimensional elements are represented by cubic curves.

R8.5 Both one- and two-dimensional elements are translated into a format appropriate for both the SMART data tree [6] and a PATRAN neutral file [5]. Information for the PATRAN neutral file is written at the user's request.

R8.6 Both one- and two-dimensional elements are generated for all rib and spar structural elements. The user may indicate which type of element is to be saved for output.

R8.7 SMART geometry surface elements are reformulated so that each boundary of the new surface elements correspond to the boundary of one of the underlying structural elements.

Reformulation of the geometry surface elements may cause some deviation from the original geometry surface elements.

9. Fuselage Assembly Definition Requirements

1. General Requirements

The purpose of this function is the identification of fuselage assemblies. Generally, the placement of structural elements in the fuselage follows differing principles depending upon the position along the axis of the fuselage and the structures and loads being supported there. Figure 15 illustrates a typical placement of fuselage assemblies.

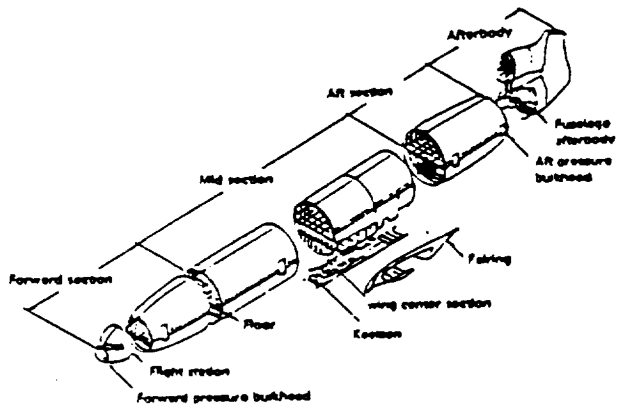


Figure 15
Fuselage Components

It is assumed that basic surface geometry for the fuselage has been provided in the SMART data tree, along with sufficient information to determine top, front and side views.

R9.1 Stations separating assemblies are placed perpendicular to the x-axis.

2. Input Requirements

R9.2 The assembly divisions may be DEFINED by pointing to a top view of the fuselage at the position assemblies are to be divided.

R9.3 The assembly divisions may be DEFINED by numeric input of the fuselage station value which is to divide the assemblies.

R9.4 The assembly divisions may be EDITED by pointing to a top view of the fuselage moving the position of the indicator dividing the assemblies.

R9.5 The assembly divisions may be EDITED by updating the numeric value of the station which divides the assemblies.

R9.6 A assembly division may be DELETED by pointing at the division which is to be removed.

Since much of the other internal structure depends upon the placement of the assembly divisions, editing the assembly divisions requires deleting this structure and redefining it.

3. Processing Requirements

R9.7 Results are presented real-time in a top view of the fuselage.

10. Cross Section Generation Requirements

1. General Requirements

The purpose of this function is the identification of fuselage stations where frames and bulkheads are defined. Figure 16 illustrates the typical placement of stations within a fuselage assembly.

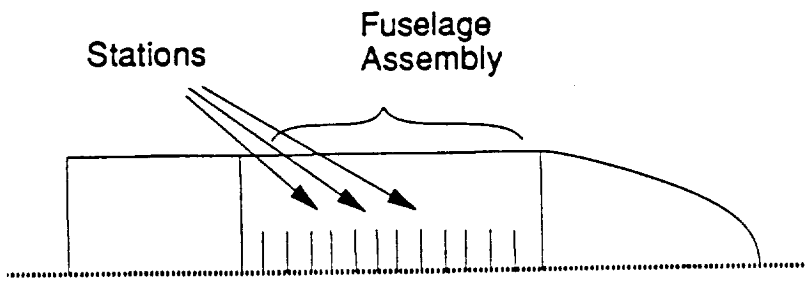


Figure 16
Cross Section Placement

R10.1 Stations are placed perpendicular to the axis of the fuselage.

R10.2 Each fuselage assembly is treated separately.

2. Input Requirements

R10.3 The assembly for which stations are to be generated is designated by pointing at a top view of the fuselage.

R10.4 The station positions may be DEFINED by designating that stations should occur at a uniform interval and a numeric value for that interval, for example, every 20 inches.

R10.5 The station positions may be DEFINED by pointing at a top view of the fuselage at the position a station is to be inserted.

R10.6 The station positions may be DEFINED by numeric input of the desired location of the station.

R10.7 The assembly divisions may be EDITED by updating the value for the size of the interval at which stations should occur.

R10.8 The assembly divisions may be EDITED by pointing at a top view of the fuselage and moving the station indicator to a new position.

R10.9 The assembly divisions may be EDITED by updating the numeric value for the position of the station.

R10.10 Editing non-equally spaced stations as equally spaced causes the given number of stations to be redistributed equally before editing. Since this may delete user supplied data, the user will be notified prior to this change.

R10.11 All stations may be DELETED by choosing a delete all function.

R10.12 A station may be DELETED by pointing individually at the station to be deleted.

3. Processing

R10.13 Results are presented real-time in both top and side views of the fuselage.

11. Ring Frame Requirements

1. General Requirements

The purpose of this function is the designation of a given frame station and the generation of structural elements corresponding to that frame. Figure 17 illustrates a typical frame station.

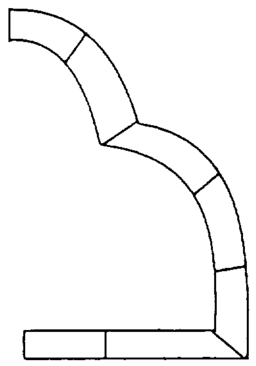


Figure 17
Frame Structure

R11.1 Ring frames are initially generated to have a uniform depth.

R11.2 Structural elements representing the frame are generated as both one- and two-dimensional.

R11.3 Only one ring frame is generated at the boundary of two assemblies.

2. Input Requirements

R11.4 In a given fuselage assembly, all station stations may be DESIGNATED as the sites for ring frames.

R11.5 In a given fuselage assembly, individual station stations may be DESIGNATED as the site for a ring frame by pointing at that station.

R11.6 Ring frames are initially DEFINED to have a constant depth by a user input value which is then used as a default for all subsequent ring frames generated.

R11.7 The default value for the depth of a ring frame may be EDITED at any time updating the depth of frames subsequently generated.

R11.8 The value for the depth of a user-specified ring frame may be EDITED regenerating that ring frame with the specified depth.

R11.9 Since ring frames are represented by patches, ring frames patches may be EDITED by changing the value of depth at either edge of the patch.

R11.10 The depths along the edges of patches in one ring frame can be used as a template for the depths along the edges of patches in another ring frame. That is, corresponding edges of patches will have the same depth.

R11.11 Since structural elements representing the frames are bicubic patches, ring frames are available to be EDITED by updating any of their patch control parameters.

3. Processing

R11.12 Results are presented real-time in a front view of the station for which the frame is defined.

R11.13 Editing structural elements of the frame as patches follows SMART interface conventions for patch editing [5].

12. Bulkhead Requirements

1. General Requirements

The purpose of this function is the designation of a given station station as a bulkhead and the generation of structural elements corresponding to that bulkhead. Figure 18 illustrates a typical bulkhead.

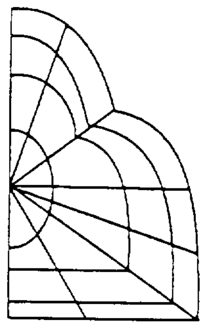


Figure 18
Bulkhead Structure

R12.1 Bulkheads are initially generated from the outer skin to inner structure, if any, such as a tank.

R12.2 Structural elements representing the bulkhead are generated as both one- and two-dimensional.

R12.3 Only one bulkhead is generated at the boundary of two assemblies.

2. Input Requirements

R12.4 In a given fuselage assembly, all station stations may be DESIGNATED as the sites for bulkheads.

R12.5 In a given fuselage assembly, individual station stations may be DESIGNATED as the site for a bulkhead by pointing at that station.

R12.6 Since structural elements representing the bulkheads are bicubic patches, they are available to be EDITED by updating any of their control parameters.

3. Processing Requirements

R12.7 Results are presented real-time in a front view of the station for which the bulkhead is being defined.

R12.8 Editing of structural elements representing the bulkhead follows SMART interface conventions for patch editing.

13. Longeron, Keel, and Beam Requirements

1. General Requirements

The purpose of this function is the generation of structural elements representing longerons, keels, and other beams. Figures 19 and 20 illustrate typical placements for longerons, keel and other beams.

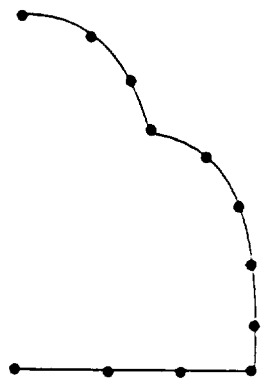


Figure 19
Placement of Longerons

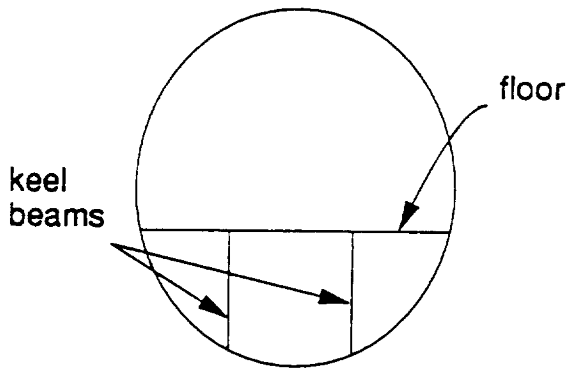


Figure 20
Keel Beam and Floor Structure

R13.1 Longeron and beam elements are generated along the outer skin of the fuselage parallel to the axis of the fuselage (x-axis).

R13.2 Longeron elements are represented by one-dimensional elements.

R13.3 Keel, floor and other beam elements are represented by both one and two dimensional elements.

2. Input Requirements

Generally, each station station intersects the same number of longeron, keel or other beam elements.

R13.4 Elements will initially be DESIGNATED as longerons.

R13.5 The initial positions of longerons are DEFINED by entering the number of longerons required for a given station.

R13.6 Additional longeron positions may be DEFINED by pointing to the desired locations.

R13.7 Additional longeron positions may be DEFINED by entering values for the desired locations.

R13.8 The value representing the number of longeron locations may be EDITED.

R13.9 A longeron location may be EDITED by dragging it to the desired new position.

R13.10 A longeron location may be EDITED by changing the values representing its position.

R13.11 All longeron locations may be DELETED by choosing a delete all function.

R13.12 Single longeron locations may be DELETED by pointing at them individually.

R13.13 Since successive stations will have the same number of longeron positions, one station can be used as a template for another. That is, corresponding edges of patches will used on successive station stations.

R13.14 Positions for keel and other beams may be DESIGNATED by pointing at the desired locations. By default, keel beam elements will be initialized with a depth equal to the frame depth at that position and a direction perpendicular to the fuselage.

R13.15 The depth of beam elements may be EDITED by changing its value.

R13.16 The direction of beam elements may be EDITED by dragging the endpoint of the beam to the desired direction.

R13.17 The direction of beam elements may be EDITED by changing the values of the endpoint of the beam.

In case the number of longerons is not fixed between a pair of station stations, the following editing functions are necessary:

R13.18 A longeron element may be ADDED by pointing at its endpoints in successive stations; this method is referred to as "freehand."

R13.19 A longeron section may be DELETED by pointing to it in a given station.

3. Processing Requirements

R13.20 Results are presented real-time on both a front view and a side view of a given station.

R13.21 The default longeron positions will be calculated using the following rule. The cross section will be divided into segments based upon points of discontinuity of the fuselage curve. Arc lengths of each segment will be computed. Longerons will be positioned at the points of discontinuity between sections. The remaining longerons will be assigned to each segment in a percentage proportional to the relative arc length of that segment. The longerons in each segment will be placed uniformly along that segment.

R13.22 Validity of "freehand" input for longerons is checked to insure that they do not intersect other longerons.

14. Fuselage Output Requirements

1. General Requirements

Since processing in SMART Structures is a time-consuming procedure, user input is captured and stored until the entire fuselage structure is defined. Upon finishing, one and two-dimensional geometry elements are created for both the fuselage surface and the frame, bulkhead and beam elements. One-dimensional elements are created for the longeron elements. Elements in the surface are reformulated from the original SMART geometry, so that the output patches are defined by the boundaries of the underlying structural elements.

R14.1 The resulting elements must be added back to the SMART data tree for use by other SMART modules, including visualization, and may be output as a PATRAN neutral file. Sufficient information is output to the SMART data tree so that the fuselage structures can be recalled, edited and output again in edited form.

2. Input Requirements

R14.2 The user indicates that the editing of the structural model of the fuselage is complete.

3. Processing Requirements

Two types of elements need to be calculated, one- and two-dimensional.

R14.3 Two-dimensional elements are represented by bicubic surfaces.

R14.4 One-dimensional elements are represented by cubic curves.

R14.5 Both one- and two-dimensional elements are translated into a format appropriate for both the SMART data tree [6] and a PATRAN neutral file [5]. Information for the PATRAN neutral file is written at the user's request.

R14.6 Both one- and two-dimensional elements are generated for all frame, bulkhead and beam structural elements. The user may indicate which types of elements are to be saved for output.

R14.7 One-dimensional elements are created for the longerons.

R14.8 SMART geometry surface elements are reformulated so that each boundary of the new surface elements corresponds to the boundary of one of the underlying structural elements.

Reformulation of the geometry surface elements may cause some deviation from the original geometry surface elements.

15. Wing - Fuselage Placement and Integration Requirements

1. General Comments

The purpose of this function is to provide for the relative placement of the wing and fuselage and for the integration of major structural elements in the wing and fuselage to build an integrated structural model. It is assumed that the wing and fuselage structural models have been completed by SMART Structures. Figure 21 illustrates several different carry-through structures for wing - fuselage integration. Figure 22 illustrates several different techniques for joining wing-spar structure to fuselage-ring frame structure.

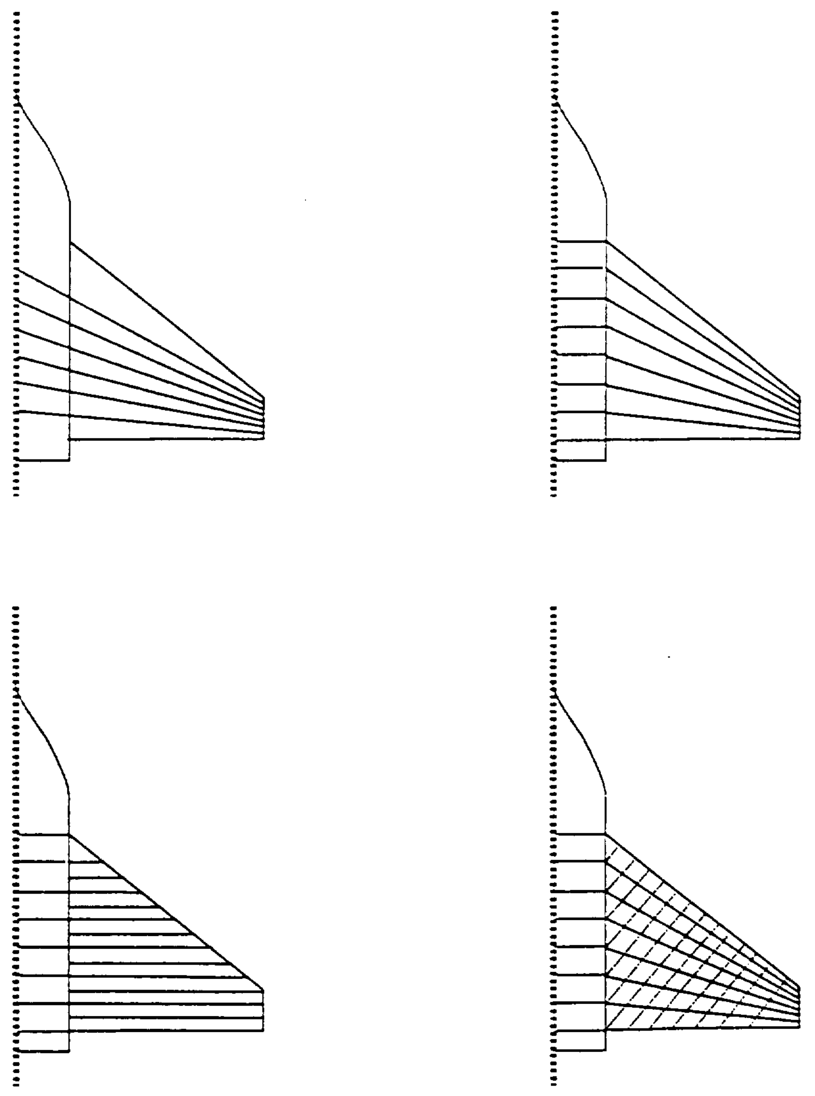


Figure 21
Carry-Through Structures

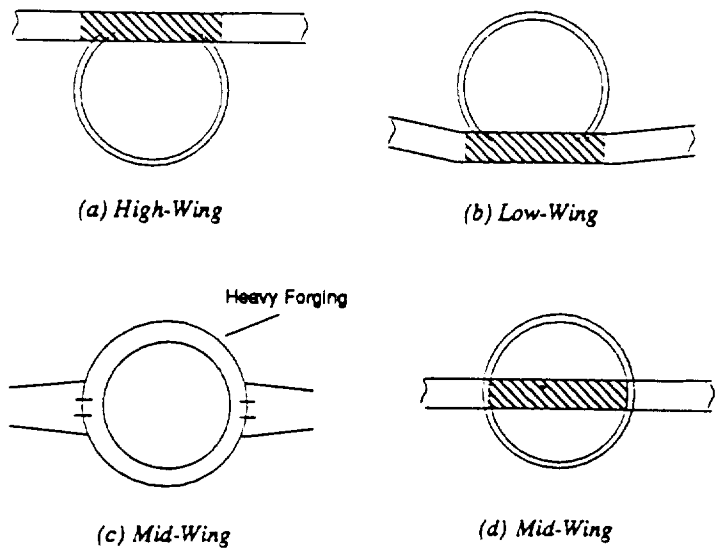


Figure 22
Joining Spar and Ring Frame Structures

R15.1 It is assumed that the wing will be integrated to only one of the assemblies of the fuselage.

2. Input Requirements

R15.2 The right wing is presented in planform view together with a top view of the fuselage. Symmetry defines the placement of the left wing. A side view is also presented to allow for height alignment.

R15.3 The assembly of the fuselage with which the wing is to be integrated is IDENTIFIED by pointing.

R15.4 The wing may be PLACED relative to this assembly by pointing at the desired position.

R15.5 The wing may be PLACED relative to this assembly by numerical input of the fuselage station at which the leading edge - root rib intersection is to be placed.

R15.6 The placement may be EDITED by dragging the wing to a new position.

R15.7 The placement may be EDITED by updating the fuselage station value at which the leading edge - root rib intersection is to be placed.

R15.8 The user may SELECT any one of the methods of carry-through illustrated in figure 21. When possible ring frame and bulkhead stations will be moved in the x-direction to match spar positions for the method of carry-through selected.

R15.9 Ring frames and bulkhead stations may be EDITED to change their x-location by dragging the station to a corresponding spar position.

R15.10 Ring frames and bulkhead stations may be EDITED to integrate wing fuselage structure by adding a new frame or bulkhead station using the techniques of section 10, above.

R15.11 Ring frames and bulkhead stations may be EDITED to integrate wing fuselage structure by deleting a frame or bulkhead station.

R15.12 The user may SELECT any one of the methods for joining spars to ring frames illustrated in figure 22.

R15.13 Since structural elements representing the joining of spars to ring frames and bulkheads are bicubic patches, they are available to be EDITED by updating any of their control parameters.

3. Processing Requirements

Since the bulk of processing consists of editing frame and bulkhead stations, processing proceeds as in requirements R11.9-R11.10 and R12.6-R12.7.

R15.14 The root rib of the wing must be made to intersect with frame elements in the fuselage. This may require that the root rib no longer be parallel to the plane of symmetry of the vehicle.

R15.15 Editing the root rib necessarily requires updating the spar elements that intersect it.

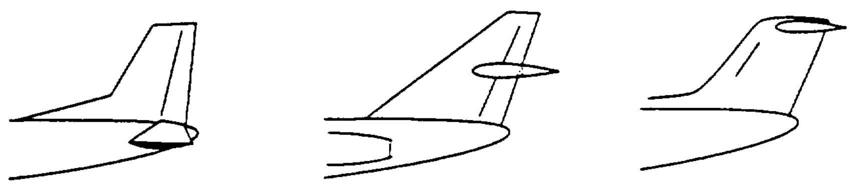
4. Fuselage Integration Output

R15.16 Since the positioning and definition of elements in both the wing-type object and fuselage-type object may change due to this operation, all elements are verified to determine the need for recalculation. When necessary both the SMART data tree and PATRAN neutral file output must be updated.

16. Wing - Tail Placement and Integration Requirements

1. General Comments

The purpose of this function is to provide for the relative placement of the wing and tail (or other wing-like structure such as a wing tip) and for the integration of major structural elements to build an integrated structural model. It is assumed that the wing and tail structural models have been completed by SMART Structures. Figure 23 illustrates several typical methods for wing-tail integration.



(a) Conventional Tail (b) + Tail (c) T-Tail

Figure 23
Wing - Tail Integration

R16.1 It is assumed that the wing will be integrated to only one of the assemblies of the tail.

2. Input Requirements

R16.2 The right wing is presented in planform view together with a top view of the tail. Symmetry defines the placement of the left wing. A side view is also presented to allow for height alignment. Initially the wing and tail are assumed to be perpendicular to each other.

R16.3 The assembly of the tail with which the wing is to be integrated is IDENTIFIED by pointing.

R16.4 The wing may be PLACED relative to this assembly by pointing at the desired position.

R16.5 The placement may be EDITED by dragging the wing to a new position.

R16.6 When possible spars in the tail will be moved to match spar positions for the wing being attached.

R16.7 Spar locations in either the wing or tail may be edited as in wing-assembly integration described in Requirements Section 6 above.

3. Fuselage Integration Output

R16.8 Since the positioning and definition of elements in both the wing and fuselage may change due to this operation, all elements are verified to determine the need for recalculation. When necessary both the SMART data tree and PATRAN neutral file output must be updated.

17. Point Load Requirements

1. General Requirements

The purpose of this function is the placement and modification of structural loads associated with user-specified points in the wing and fuselage.

R17.1 Loads may be represented either by scalar or vector values corresponding to the load condition and an indicator of the grid point in the existing structural model at which the load should be applied.

R17.2 Multiple loads of various types may be applied at any given grid point in the structural model.

2. Input Requirements

R17.3 A load may be DEFINED for any existing grid point in the structural model of the wing or fuselage; such a load will be called a point load.

R17.4 Multiple point loads may be defined for any grid point in the existing structural model.

R17.5 Any point load may be DEFINED as either a scalar or vector value.

R17.6 All point loads may be simultaneously DISPLAYED on a visual representation of the structural model.

R17.7 Any point load may be EDITED by selecting the grid point at which the load is applied and updating the value of the associated scalar or vector.

R17.8 Any point load may be DELETED.

3. Processing Requirements

If the structural model is edited after the point loads have been applied, one of two situation exist. Either a given grid point at which a point load was applied remains in its original position on the structural model or the associated grid point has either been moved or deleted.

R17.9 If the point is in its original position, point loads should remain attached to the same grid point in the resulting structural model.

R17.10 If the grid point has been moved from its original position or deleted, the user is notified so that the load can be appropriately placed in the modified model. An initial default for the placement of the point load is that grid point in the modified model closest to the original grid point.

18. Path and Area Load Requirements

1. General Requirements

The purpose of this function is the placement and modification of structural loads associated with specified paths or areas in the wing and fuselage. For example loads associated with the placement of a heavy pipe (path load) or a thermal protection system (area load).

R18.1 Loads can be represented by scalar values at the grid points in the model representing the path or area over which the load is to be spread.

R18.2 Multiple loads of various types may be applied at any given path or area in the structural model.

2. Input Requirements

R18.3 A path may be DEFINED as any sequence of grid points in the existing structural model where successive grid points share a common edge with the previous grid point.

R18.4 An area may be DEFINED as a connected collection of grid points in the existing structural model.

R18.5 A load may be DEFINED for any existing path in the structural model of the wing or fuselage by specifying the total load which will be spread uniformly over the indicated path; such a load will be called a path load.

R18.6 A load may be DEFINED for any existing area in the structural model of the wing or fuselage by specifying the total load which will be spread uniformly over the indicated area; such a load will be called an area load.

R18.7 Multiple path or area loads may be DEFINED for any path or area in the existing structural model.

R18.8 All path and area loads may be DISPLAYED on a visual representation of the structural model.

R18.9 Any path or area load may be EDITED by selecting the path or area over which the load is applied and redefining the collection of grid points comprising that path or area.

R18.10 Any path or area load may be EDITED by selecting the path or area over which the load is applied and updating the value of the total associated load.

R18.11 Any path or area load may be DELETED.

3. Processing Requirements

If the structural model is edited after the path or area loads have been applied one of two situation exist. Either the grid points defining a given path or area remain in their original position on the structural model or they have either been moved or deleted.

R18.12 If the grid points defining the path or area remain in their original position, the loads should remain attached to the original path or area in the resulting structural model.

R18.13 If some of the grid points have been moved from their original position or deleted, the user is notified so that the path or area can be appropriately placed in the modified model. An initial default for the placement of the load is that set of grid points in the modified model closest to the original grid points.

R18.14 Data is kept on the patches with which the path and area loads are associated through their grid points.

19. Analysis-Generated Load Requirements

Accept as input loads generated by the APAS and other aerodynamic analysis programs and use these to generate data patches in a form compatible with the PATRAN neutral file and consistent with the structural model.

1. General Comments

As noted in section 2 above, models designed using SMART are subjected iteratively to several different analysis techniques. Part of this process involves using SMART-generated vehicle geometry in the aerodynamic analysis programs such as APAS. The various loads generated in this analysis are important to the structural analysis. It should be noted that although the geometric surfaces represented in both the aero and structures model are the same, the representations employed are different, each tailored to meet different needs. It is the purpose of this function to retrieve the aero data and the representation used for the aero geometry and map the data to a form consistent with the representation of the structural geometry defined above.

It is assumed that the input will be in the format of data files handled by APAS [1] and will contain parametric identifiers for the test cases run along with data for each surface panel that includes the centroid of the panel, the loading information at that centroid and other functional outputs. It is assumed that a "smoothed" bilinear interpolant to this data will provide sufficient accuracy for further structural analysis.

2. Input Requirements

R19.1 The module must be able to present all available local data files for selection by the user.

R19.2 The module must be able to present parameter information identifying all test cases contained in a specified data file for selection by the user.

R19.3 The module must be able to read the data file format.

3. Processing

R19.4 This module is responsible for creating a bicubic data surface representing a smoothed approximation of the bilinear interpolant to the data found in the data file.

R19.5 Data patches may be output to a PATRAN neutral file [5].

R19.6 Data patches are further processed by rescaling data values so that ranges fall approximately in the geometry range of the wing thickness and placed in the SMART data tree [6].

20. Load Output Requirements

Two types of elements need to be calculated at this stage, data patches for output to PATRAN and scaled data surface patches using the SMART format to aid in visualization of the surface.

R20.1 PATRAN data patches are output in the format described in [5].

R20.2 SMART data surface patches are output in the format described in [6].

21. User Interface Requirements

R21.1 SMART Structures must use the SMART graphics user interface developed for the Silicon Graphics (SGI) IRIS 4D Workstation using SGI's GL graphics library and described in [6].

In addition the following general user interface consideration must hold:

R21.2 Validity of keyboard inputs is checked; specifically, numeric input must be numeric and within acceptable ranges defined by the extents of the wing planform or fuselage.

R21.3 Pointer input on points and edges is modified to attach to an existing point or edge within 5 raster units if appropriate for the indicated operation.

R21.4 Since edit and delete operations must follow corresponding define operations, users are notified of an attempt to edit elements out of order.

R21.5 Users are notified of mistaken input immediately.

R21.6 Requests for deletion are confirmed with the user.

R21.7 An UNDO feature which should allow for the restoration of the state of the structural model just prior to the last user operation. This feature would apply to the immediately preceding operation as tracking an entire sequence of preceding operations and the information necessary to restore the model would be complicated and time consuming.

The following two points, while not requirements, would significantly improve the functionality of the SMART Structures modules form the user's point of view.

R21.8 An INTERMEDIATE SAVE feature which would allow the user to exit the SMART Structures modules saving the current state of the structural model generation so that it could be restored for completion at a later time.

R21.9 The ability to save and edit a SCRIPT or JOURNAL file of a SMART Structures session which would allow the user to replay a set of modifications exactly as entered or in an appropriately modified fashion.

22. *Software Interface Requirements*

The basic interface to the rest of SMART is through the SMART data tree.

R22.1 SMART Structures must be capable of reading any aerospace vehicle geometry generated by SMART.

R22.2 SMART Structures must be capable of generating structural and visualization surfaces in a format for inclusion in the SMART data tree.

R22.3 As detailed in sections 8, 14, 15, 19, SMART Structures must be capable of reading aerodynamic analysis data files and writing PATRAN neutral files.

Appendix B

N92-27831

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

1 Introduction 5

2 Capabilities of SMART 6

3 Aircraft Structural Design 9
- 3.1 Design Considerations 9
- 3.2 The Actual Design 11

4 Overview of Finite Element Analysis 15
- 4.1 Steps in the Finite Element Method 18
- 4.2 Creating the Mesh 18

5 Geometric Representation 21

6 Algorithms for Generating Bulkheads and Ringframes 26
- 6.1 Capabilities Developed for SMART Prior to the Project 26
- 6.2 New Results 30
  - 6.2.1 Bulkheads 30
  - 6.2.2 Ringframes 36

7 Conclusion 37

Appendix A: Implementation of the Algorithms: Snapshots of the SMART Display 39

Appendix B: Software Requirements for SMART Structures 65

Appendix C: SMART Code to Implement Algorithms 103

List of Figures

1 The Layout of the SMART Screen [SMART, p. 2-1] 7
2 Ringframe from an offset curve [REHDER, p. 3] 8
3 Bulkhead between fuselage and internal tank [REHDER, p. 4] 9
4 Semimonocoque construction [MCKIN, p. 144] 12
5 Typical semi-monocoque stiffened shell—L-1011 [NIU, p. 376] 13
6 Typical transport fuselage center section floor beams arrangement. [NIU, p. 396] 14
7 Typical pressure flat bulkhead [NIU, p. 398] 14
8 Fail-safe design by using longitudinal beam along side of fuselage. [NIU,p. 391] 15
9 Sketch of main details of aeroplane structure [STIN66, p. 205] 16
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] 17
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] 19
12 Model reduction due to structure symmetry 20
13 Two Bézier curves and their control points [FOLEY, p. 488] . 22
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] 25
15 Bicubic Bézier Patch 26
16 Vectors and points used to calculate a patch 29
17 A fuselage cross-section. 42
18 A fuselage cross-section with component cubic Bézier curve segments and control points. 43
19 A simple tank cross-section (simple implies convex shape) with component cubic Bézier curve segments and control points. . . 44
20 A fuselage cross-section with interior simple tank cross-section. 45
21 A multi-bubble tank cross-section with component cubic Bézier curve segments and control points. 46

22 A fuselage cross-section with interior multi-bubble tank cross-section. 47
23 A fuselage cross-section with generated interior circular tank with cubic Bézier curve segments showing the path of growth for bulkhead patches. 48
24 A fuselage cross-section with generated interior circular tank with first-guess bulkhead patches. 49
25 A fuselage cross-section with generated interior circular tank with control points (plotted as one-third points) edited to smoother patches. 50
26 A fuselage cross-section without interior tank with first-guess bulkhead patches based on cubic Bézier inward growth curves. 51
27 A fuselage cross-section without interior tank with first-guess bulkhead patches based on linear inward growth curves. 52
28 A fuselage cross-section with interior simple tank cross-section and cubic Bézier growth curves between corresponding points. Blue squares on the green cross-sections indicate locations where original given cross-sections were split to accomplish equal number of curves per cross-section, and corresponding curves of near-equal percents of arclength. 53
29 Initial bulkhead patches based on cubic Bézier growth curves between fuselage cross-section and simple tank cross-section. . 54
30 Bulkhead after editing of control points of patches between fuselage cross-section and simple tank cross-section. 55
31 A fuselage cross-section with interior multi-bubble tank cross-section and cubic Bézier growth curves between corresponding points. Blue squares on the green cross-sections indicate locations where original given cross-sections were split to accomplish equal number of curves per cross-section, and corresponding curves of near-equal percents of arclength. 56
32 Initial bulkhead patches based on cubic Bézier growth curves between fuselage cross-section and multi-bubble tank cross-section. 57
33 Bulkhead after editing of control points of patches between fuselage cross-section and multi-bubble tank cross-section. . . 58
34 Placement of twelve longerons on fuselage cross-section with interior multi-bubble tank cross-section. 59

35 Placement of nine longerons on fuselage cross-section with interior simple tank cross-section. 60
36 Placement of thirteen longerons on fuselage cross-section with interior simple tank cross-section. 61
37 Placement of thirty longerons on fuselage cross-section with interior simple tank cross-section. 62
38 A fuselage cross-section with ringframe patches generated with constant percent of growth along cubic Bézier curve segments. 63
39 A fuselage cross-section with constant width ringframe patches. 64

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

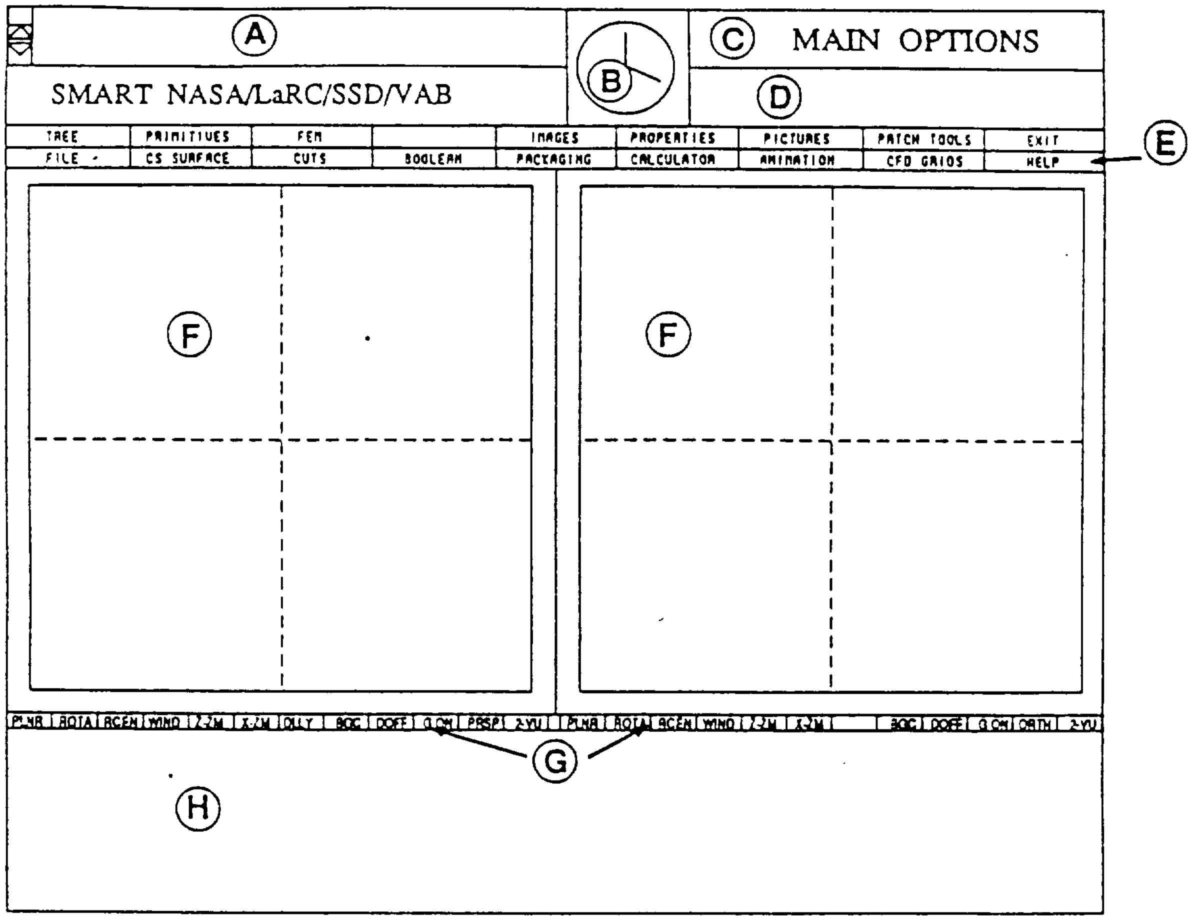


Figure 2-1
The Layout of the SMART Screen

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]

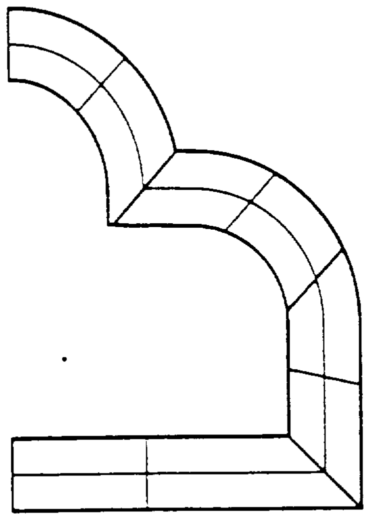


Figure 2: Ringframe from an offset curve [REHDER, p. 3]

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

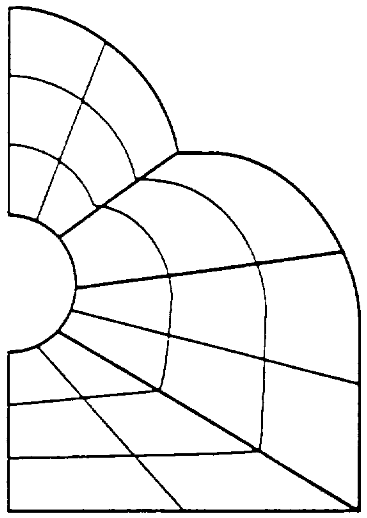


Figure 3: Bulkhead between fuselage and internal tank [REHDER, p. 4]

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

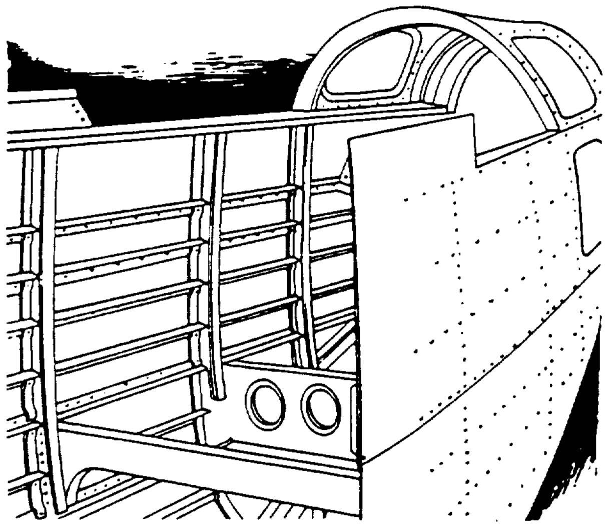


Figure 4: Semimonocoque construction [MCKIN, p. 144]

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

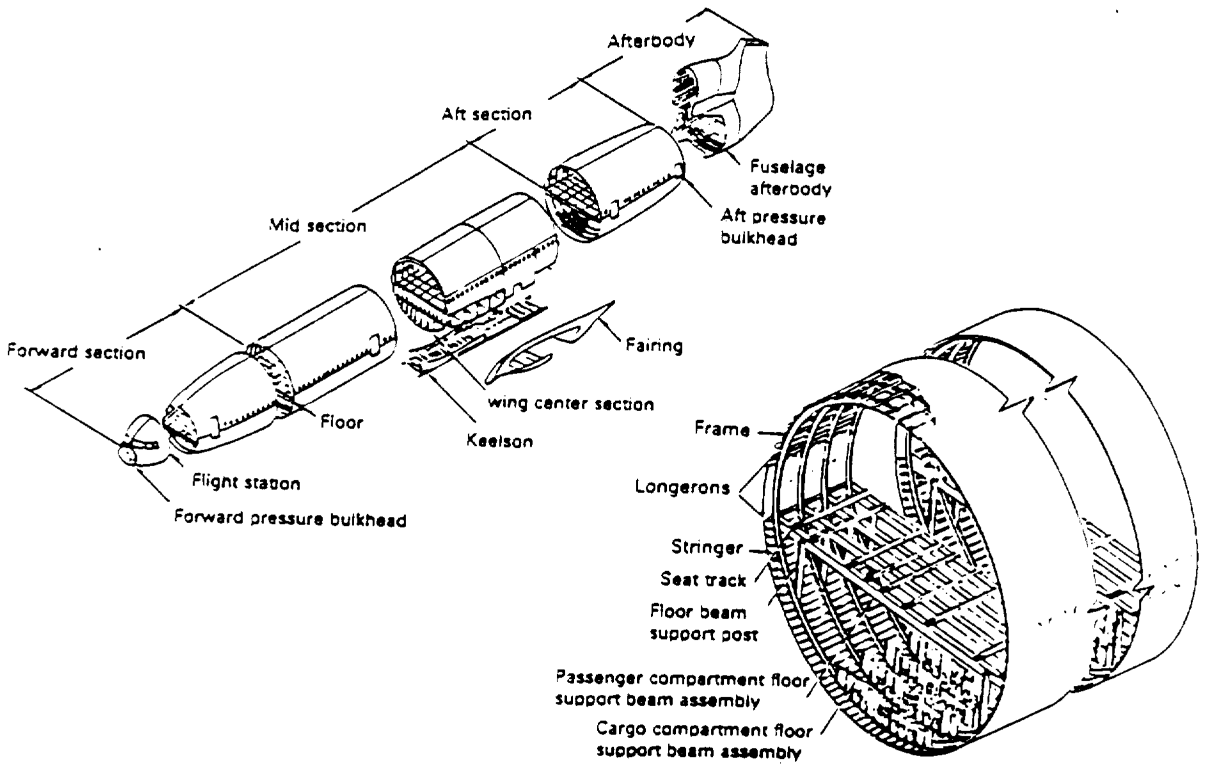


Figure 5: Typical semi-monocoque stiffened shell—L-1011 [NIU, p. 376]

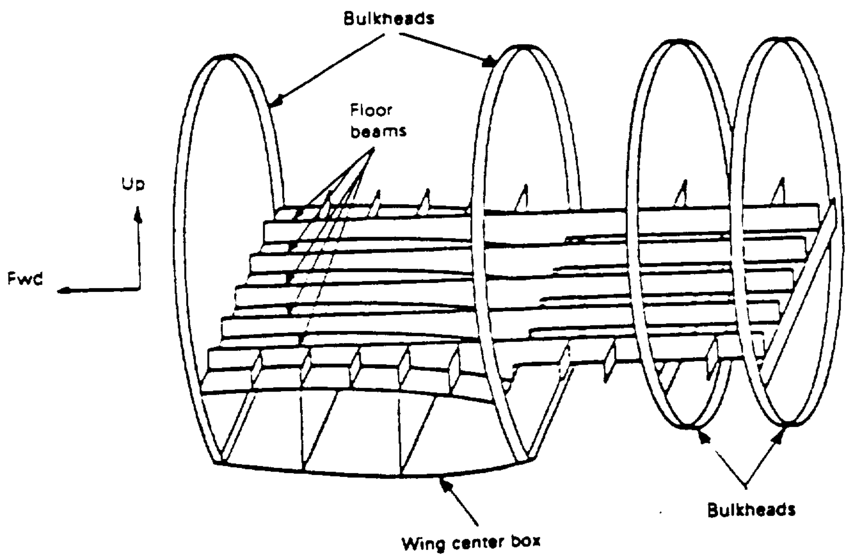


Figure 6: Typical transport fuselage center section floor beams arrangement. [NIU, p. 396]

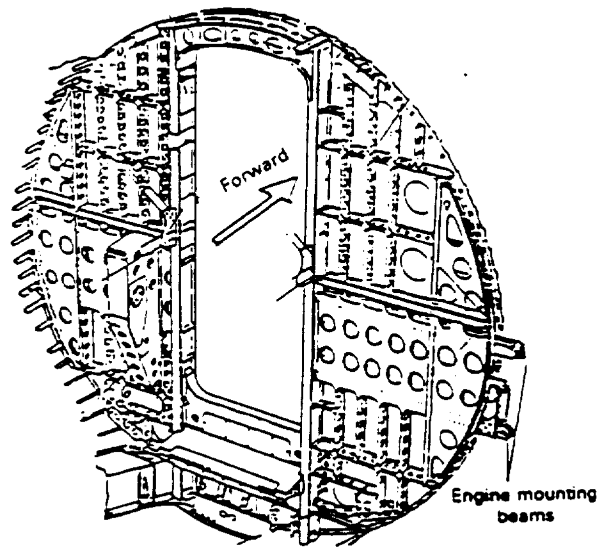


Figure 7: Typical pressure flat bulkhead [NIU, p. 398]

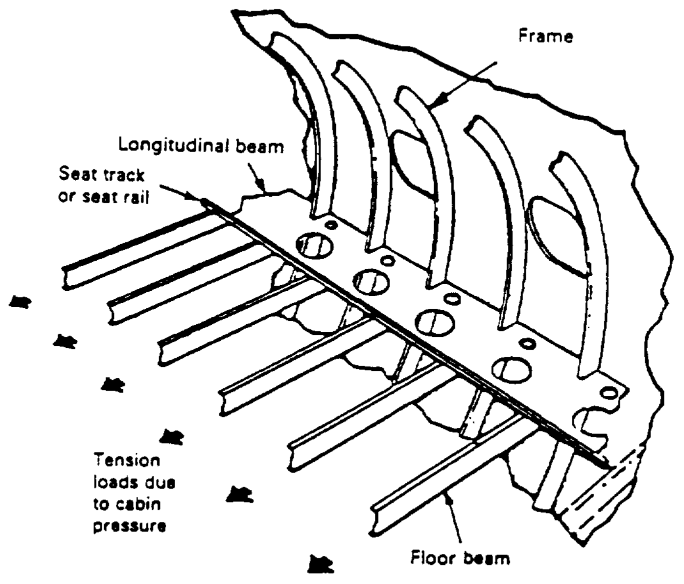


Figure 8: Fail-safe design by using longitudinal beam along side of fuselage. [NIU,p. 391]

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

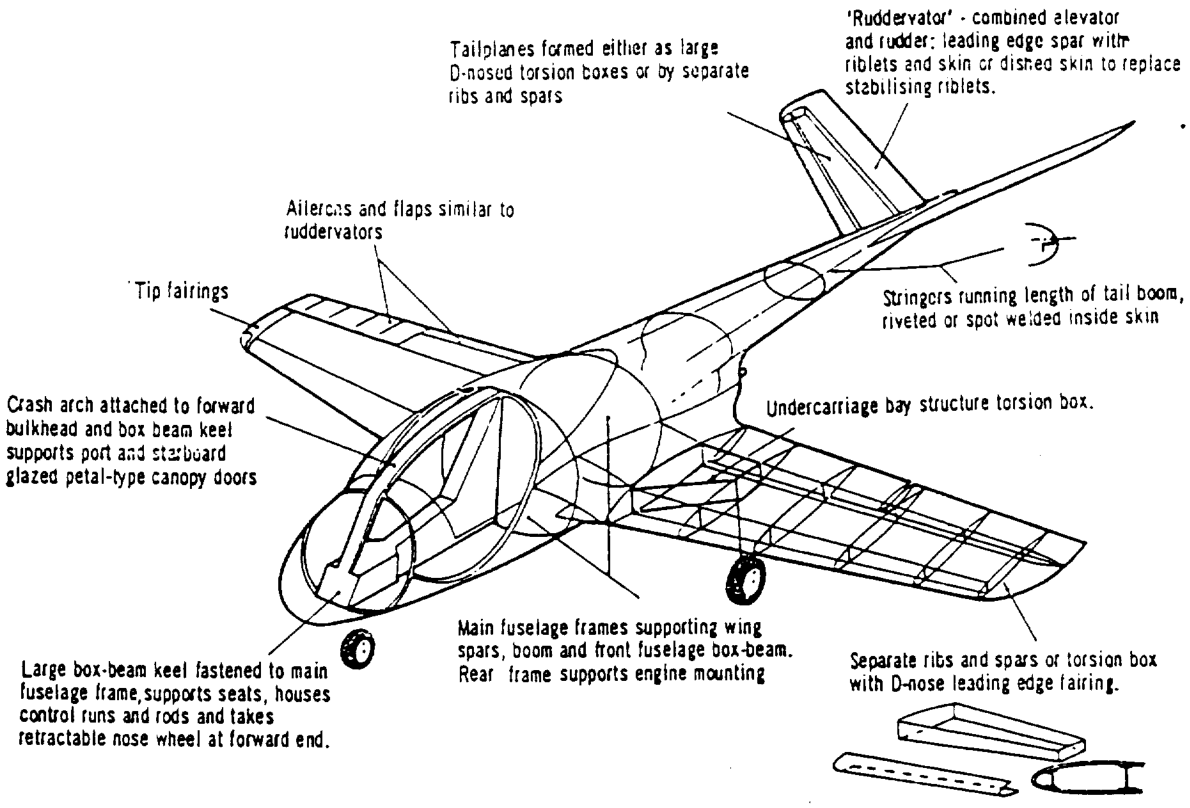

'Ruddervator' - combined elevator and rudder: leading edge spar with riblets and skin or dished skin to replace stabilising riblets.
Tailplanes formed either as large D-nosed torsion boxes or by separate ribs and spars
Ailerons and flaps similar to ruddervators
Tip fairings
Stringers running length of 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]

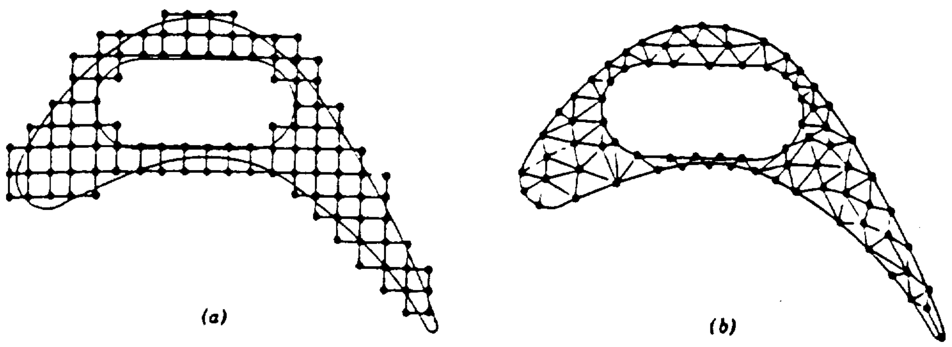


Figure 10: Finite difference and finite element discretizations of a turbine blade profile. (a) Typical finite difference model. (b) Typical finite element model. [HUEB, p. 5]

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 t' 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

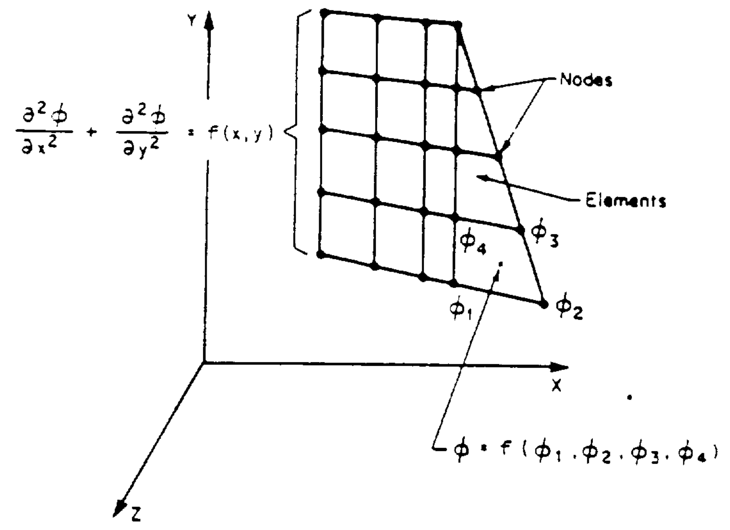

Y
$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f(x,y)$
Nodes
Elements
ϕ_4
ϕ_3
ϕ_1
ϕ_2
X
$\phi = f(\phi_1, \phi_2, \phi_3, \phi_4)$
Z

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

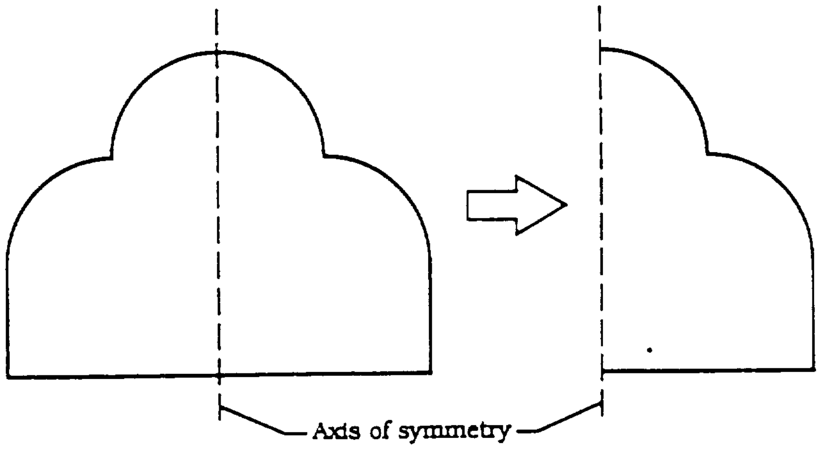


Figure 12: Model reduction due to structure symmetry

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 . :tio 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

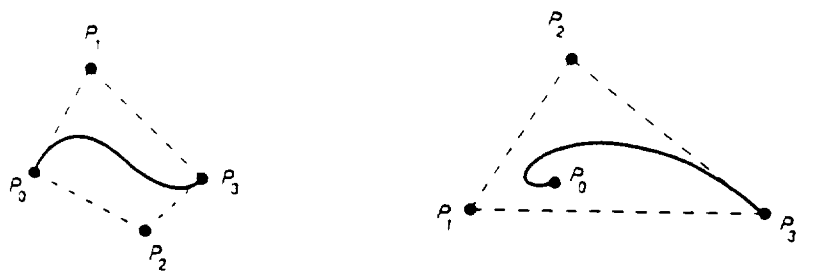


Figure 13: Two Bézier curves and their control points [FOLEY, p. 488]

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

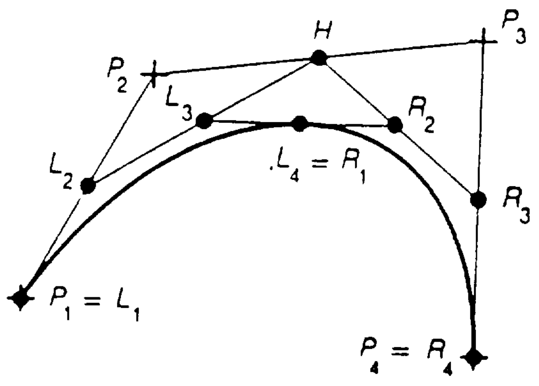


Figure 14: The Bézier curve defined by the points P_i is divided at $t = \frac{1}{2}$ into a left curve defined by the points L_i and a right curve defined by the points R_i. [FOLEY, p. 508]

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.

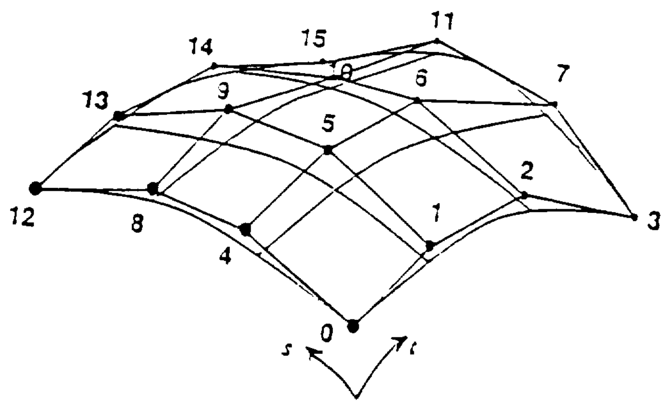


Figure 15: Bicubic Bézier Patch

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.
```

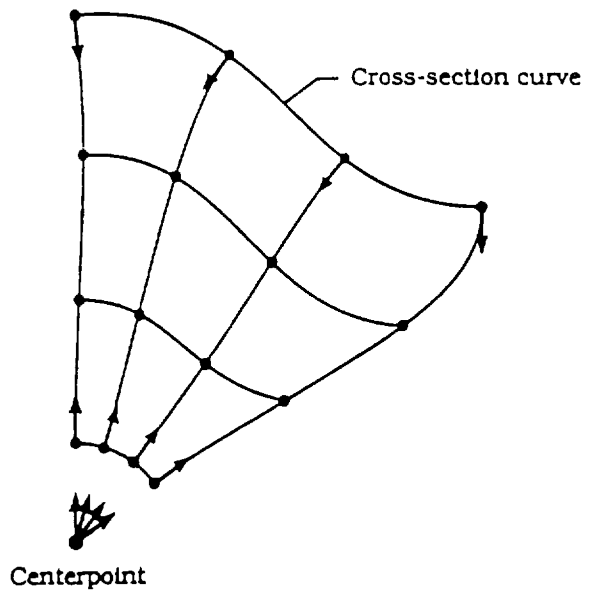


Figure 16: Vectors and points used to calculate a patch

```
        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 : 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 $\leq$ 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.

Appendix C

N92-27832

Master's Project Report

Application Driven Interface Generation for EASIE

[illegible]

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

2. EASIE: ADE–mode Considerations

 2.1 Concepts of EASIE system

 2.2 Sample Session for ADE mode

 2.3 Menu Manipulation and Construction in ADE mode

 2.4 The Drawbacks of EASIE in ADE mode

3. Principles of Interface Design

4. Modification of EASIE in ADE mode

 4.1 Window System: OSF's MOTIF

 4.2 Design Considerations for ADE Facilitator

 4.3 Demonstration of the ADE Facilitator

5. The General Structure of Solution

 5.1 Data Structure

 5.2 Mechanisms

 5.3 Capabilities and Limitations of ADE Facilitator

6. Conclusions

7. References

Appendix: program Listing

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.

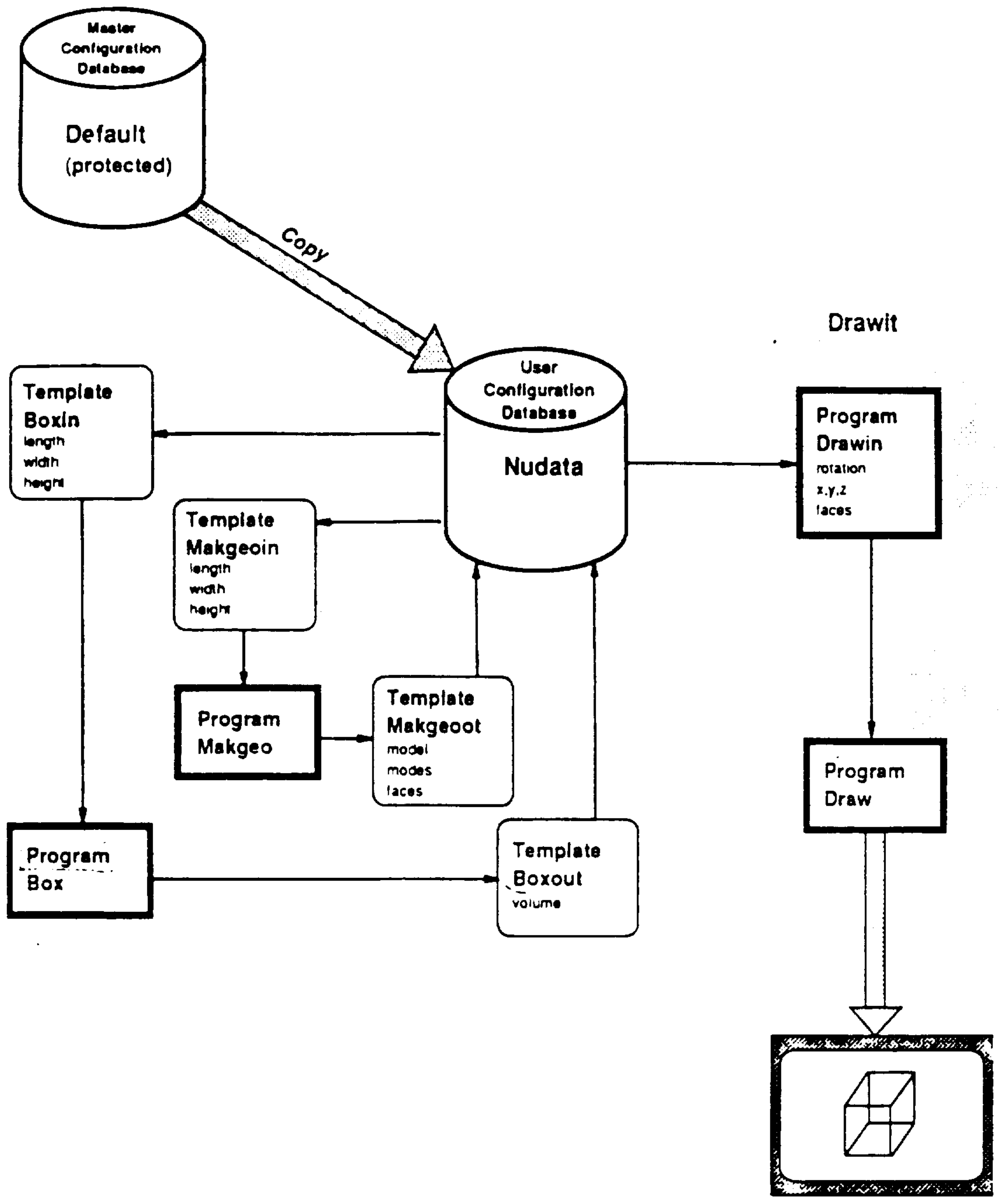


Figure 1. Flow Diagram For Sample Session Using EASIE

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.

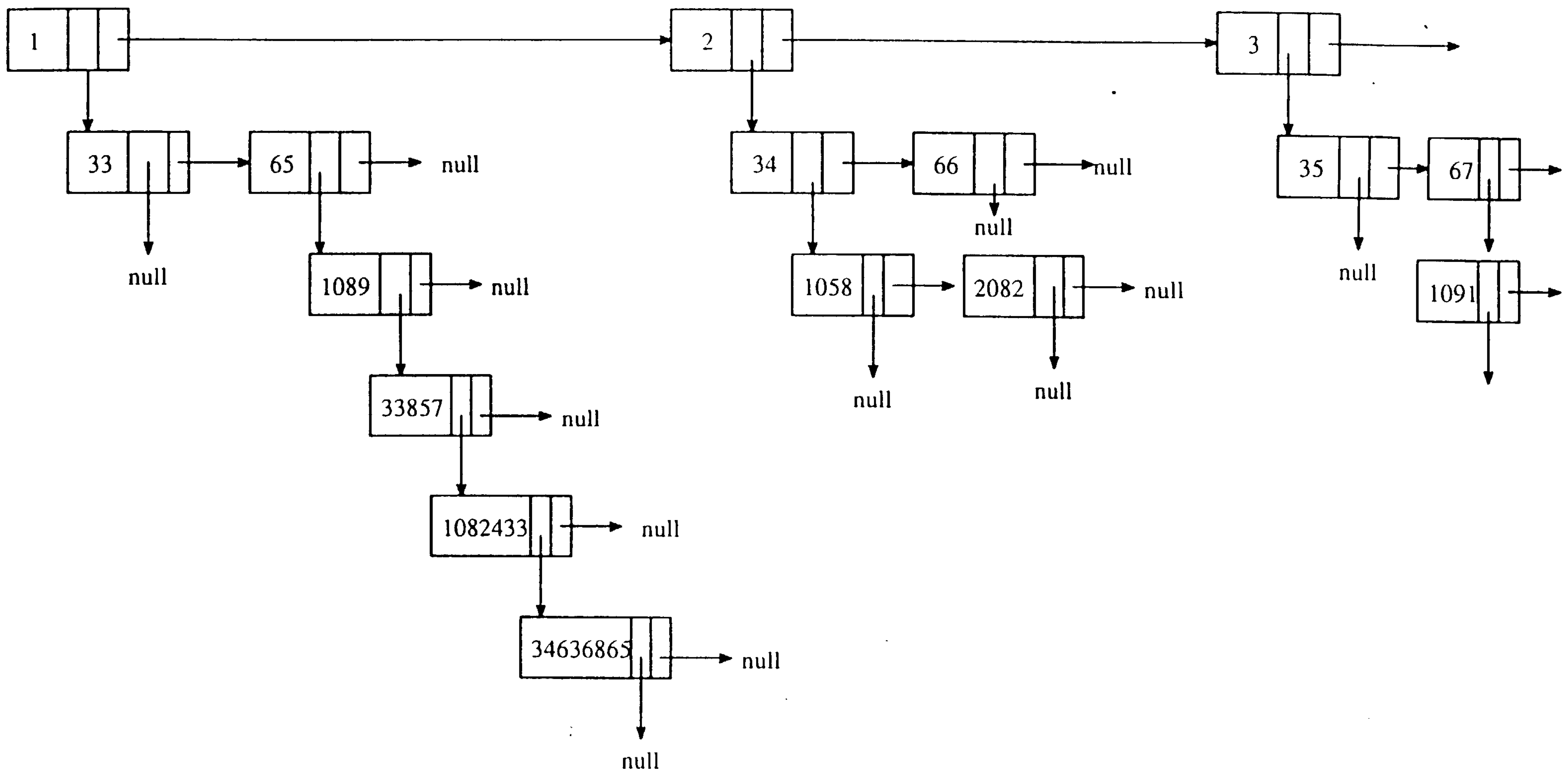


Figure 4. A structure of menu tree.

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) :

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.

Appendix D

N92-27833

p. 37

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

PAGE

ABSTRACT ii

CONTENTS iii

LIST OF FIGURES iv

1 AN INTRODUCTION TO THE EASIE SYSTEM 1
1.1 Why EASIE was developed 1
1.2 What EASIE was 1
1.3 What two operation modes of EASIE were 2
1.4 What CCE mode was 2

2 A COMPARISON BETWEEN THE CURRENT EASIE SYSTEM AND THE DESIGN PRINCIPLES 3
2.1 Be consistent 3
2.2 Provide feedback 4
2.3 Minimize error possibilities 4
2.4 Provide error recovery 4
2.5 Accommodate multiple skill levels 5
2.6 Minimize memorization 5

3 TWO OBJECTIVES OF THIS PROJECT 6
3.1 Redesign the selecting menus 6
3.2 Reorganize the selecting structures 8

4 AN OUTLINE OF THIS PROJECT 10
4.1 A general view 10
4.2 Improvements 12
4.3 A sample session using the CCE mode 13
4.4 A command summary using the CCE mode 17

5 CONCLUSION 21

REFERENCES

APPENDIX A: State-transition diagrams of this project

APPENDIX B: User manual of this project

APPENDIX C: Sample screens using the CCE mode of this project

APPENDIX D: Programs and files of this project

LIST OF FIGURES

Figure **PAGE**

1 *WorkSpace Control menu* 9

2. *Basic Environment File, easie.input* 11

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.

Tools Selection

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

Open Selection

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