Development of a CAD System for Automated Conceptual Design of Supersonic Aircraft

by

Steven Glenn Wampler

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Mechanical Engineering

APPROVED:

Arvid Myklebust, Chairman

J. Robert Mahan

Charles F. Reinholtz

May 3, 1988

Blacksburg, Virginia

Development of a CAD System for Automated Conceptual Design of Supersonic Aircraft

by

Steven Glenn Wampler

Arvid Myklebust, Chairman

Mechanical Engineering

(ABSTRACT)

Development of a conceptual aircraft design system based on ACSYNT, an aircraft synthesis program written by the NASA Ames Research Center; is discussed. The system, named ACSYNT/VPI, was written using the PHIGS graphics standard for machine independence and designed based on top-down principles and standards. A functional description of ACSYNT is presented as well as detailed software requirements for ACSYNT/VPI. The software's design is covered in depth including design philosophies and software functional specifications. Program output and design results are presented in conjunction with project recommendations. The appendices include supporting design and development information.

Acknowledgements

I would like to thank my wife for her love and patience. She helped me keep a proper perspective on life and encouraged me during my graduate work.

I am also thankful to my parents for their support throughout my college education.

In addition, I would like to thank Dr. Myklebust who provided the opportunities for me to develop my interest and skills in writing CAD software, Dr. Mahan for his insight into engineering, Dr. Reinholtz for encouraging me to go to graduate school, whose work made my thesis possible, and who put up with me being a perfectionist.

Finally, I thank God for giving my life purpose by giving His life on the cross.

Table of Contents

1.0 Introduction **1**
Overview and Project History 1
Project Objectives 2
Thesis Organization 2

2.0 Literature Review **4**
Overview of Available Literature 4
Currently Available Conceptual Design Systems 4

3.0 Description of ACSYNT **6**
ACSYNT Overview 6
Functional Description of ACSYNT 6
ACSYNT/VPI Enhancements 8

4.0 Software Development Methods **9**
Overview 9
Work Methods 9
Design Methods 11
Documentation 22
Change Control Methods 24

5.0 Software Development Standards **29**
Overview 29
ACSYNT/VPI Design Standards 30
ACSYNT/VPI Coding Standards 38

6.0 Software Requirements **43**
Overview 43
Management Information 43
System Requirements 46
Project Policies 51
Project Constraints 52
Functional Requirements 53

7.0 Software Design **62**
Overview 62
Design Philosophies 63
User Interface Design 68
Menu Modules 76
Utility Routines 79
Data-flow and Geometric Modeling 103

8.0 Results and Recommendations 145

9.0 References 159

Appendix A. Glossary 161

Appendix B. Example ACSYNT Input File 164

Appendix C. Example ACSYNT Output File 166

Appendix D. graPHIGS Information 183
D.1 Functional Enhancements 183
D.2 Omissions 184
D.3 Differences 185
D.4 Language Binding 185

Appendix E. FORTRAN 77 Extension Used in ACSYNT/VPI Code 187

Appendix F. Example ACSYNT/VPI Menu Module 189

Appendix G. Example ACSYNT/VPI Geometry Data File 203

Appendix H. Example ACSYNT Geometry Common Block Data File 211

Appendix I. ACSYNT/VPI Menu Tier Charts 217

Appendix J. Functional Descriptions for ACSYNT/VPI Subroutines 225
J.1 AAMBL NASA 225
J.2 ACFRM NASA 225
J.3 ACMFG NASA 226
J.4 ACPCR NASA 226
J.5 ADISP NASA 226
J.6 AEYEF NASA 226
J.7 AGEOM NASA 227
J.8 AHSRF NASA 227
J.9 ALTCR NASA 227
J.10 ALTSR NASA 228
J.11 AMAIN NASA 228
J.12 AMDFY NASA 229
J.13 ANLTS NASA 229
J.14 ARAMB NASA 230
J.15 ARETL NASA 230
J.16 ARSCC NASA 230
J.17 ARSLC NASA 231
J.18 ARSLS NASA 231
J.19 ARSNL NASA 231
J.20 ARSRT NASA 231
J.21 ARSSC NASA 231
J.22 ARTAL NASA 231
J.23 ASACC NASA 232
J.24 ASCLR NASA 232
J.25 ASHAD NASA 232
J.26 ASHOW NASA 233
J.27 ASHRS NASA 233
J.28 ASRFC NASA 233
J.29 ASTCL NASA 233

J.30 ASUNF NASA 234
J.31 ASWAL NASA 234
J.32 ASWFP NASA 234
J.33 AWFRT NASA 234
J.34 AWIND NASA 234
J.35 AWISO NASA 235
J.36 AWMLT NASA 235
J.37 AWPRJ NASA 235
J.38 AWRES NASA 236
J.39 AWROT NASA 236
J.40 AWSCL NASA 236
J.41 AWSID NASA 237
J.42 AWSNG NASA 237
J.43 AWSTR NASA 237
J.44 AWTOP NASA 238
J.45 AWTRN NASA 238
J.46 CENTAN NASA 239
J.47 CLGEOM NASA 239
J.48 CMPCLR NASA 239
J.49 GEOMDB NASA 239
J.50 GMTRY NASA 240
J.51 GMVWDB NASA 241
J.52 GTGMPK NASA 242
J.53 LIST NASA 242
J.54 MENU NASA 242
J.55 MKTILE NASA 243
J.56 OPGEOM NASA 244
J.57 RETILE NASA 246
J.58 SHADDB NASA 246
J.59 SHDHDE NASA 247
J.60 STGMPK NASA 247
J.61 TILEIT NASA 247
J.62 UTILTY NASA 248
J.63 VECMAT NASA 249
J.64 XFRMTL NASA 250
J.65 XSSECT NASA 250

Appendix K. List of ACSYNT/VPI Utility Routines 251

Appendix L. ACSYNT/VPI Database Definitions 254

Appendix M. Software Development Library User's Guide 257
M.1 Overview 257
M.2 SDL Functional Description 257
M.2.1 Functional Overview 257
M.3 Operation 258
M.3.1 NAPCONS 260
M.3.2 NAPMSG 260
M.3.3 NAPVMCF 260
M.3.4 NAPREAD 261
M.3.5 NAPWAKE 262
M.4 General User Information 263
M.4.1 Overview 263
M.4.2 Sending Messages to NASASDL 263
M.4.3 Sending Commands to NASASDL 264
M.4.4 Checking Out Files from NASASDL 264

M.4.5 Sending Files to NASASDL 266
M.4.6 Inquiring and Logging File Part Numbers 267

Appendix N. Software Development Tools User's Guide 268
N.1 COMMENT EXEC and COMMENT XEDIT 268
N.2 MOD EXEC 269
N.3 GRAB XEDIT 269
N.4 MATH XEDIT and MATH EXEC 270
N.5 COMP EXEC 271
N.6 FORT XEDIT 271
N.7 BACKUP EXEC 272
N.8 TRASH EXEC 273
N.9 SEARCH EXEC 273

Appendix O. ACSYNT/VPI User's Guide 275
O.1 Overview 275
O.2 Functional Capabilities of ACSYNT/VPI Version X1.0 275
O.3 ACSYNT/VPI Run-time Requirements 276
O.3.1 Requirements Overview 276
O.3.2 Userid Requirements 277
O.3.2.1 Memory Requirement 277
O.3.2.2 Real Timer Requirement 278
O.3.2.3 Loader Table Requirement 278
O.3.3 File Requirements 279
O.3.4 Hardware Requirement 279
O.4 Running ACSYNT/VPI 280
O.5 ACSYNT/VPI Functions 282
O.5.1 Overview 282
O.5.2 Reading ACSYNT Geometry Common Block Files 283
O.5.3 Reading ACSYNT/VPI Geometry Files 284
O.5.4 Displaying the Aircraft's Geometry 284
O.5.5 Performing Three Dimensional Viewing Transformations 285
O.5.6 Changing Aircraft Component Colors 286
O.5.7 Changing the Number of Surface Tiles 287
O.5.8 No-showing Aircraft Components 288
O.5.9 Producing a Shaded Image 288
O.5.10 Change the Shading Ambient Light Fraction 290
O.5.11 Changing the Shading Surface Characteristics 290
O.5.12 Switching Sun Model Shading On and Off 291
O.5.13 Switching the Light Source Eye Flag On and Off 292
O.5.14 Controlling the Number of Shading Light Sources 293
O.5.15 Specifying the Color of All Shading Light Sources 293
O.5.16 Specifying Shading Light Parameters 294
O.5.17 Resetting the Shading Parameters to Default Values 296
O.5.18 Displaying the Geometry in Multiple Windows 297
O.5.19 Displaying the Geometry in a Single View 298
O.5.20 Storing Window Parameters 298
O.5.21 Restoring Window Parameters 299
O.5.22 Switching Between Parallel and Perspective Projections 300
O.5.23 Producing Top, Front, Side, and Isometric Views 300
O.5.24 Specifying View Rotation Values 301
O.5.25 Specifying View Scale Factors 302
O.5.26 Changing Workstation Colors 303
O.5.27 Exiting ACSYNT/VPI 304

Vita 305

List of Illustrations

Figure 1. Centralized testing of ACSYNT/VPI software. 10
Figure 2. Top down development activities 12
Figure 3. Top-down development flowchart 14
Figure 4. Example ACSYNT/VPI tier chart. 15
Figure 5. Five types of module coupling 17
Figure 6. Example ACSYNT/VPI flowchart. 18
Figure 7. Example of Dewey-Decimal system used to number flowcharts. 20
Figure 8. Documents generated during software development. 23
Figure 9. Program costs versus documentation level 25
Figure 10. Basic flowchart symbols 31
Figure 11. Specialized flowchart symbols 32
Figure 12. Programming constructs. 34
Figure 13. Programming constructs for FORTRAN "DO LOOP". 35
Figure 14. ACSYNT/VPI milestone and review schedule. 45
Figure 15. ACSYNT/VPI development environment. 47
Figure 16. ACSYNT/VPI target system environment. 49
Figure 17. Example ACSYNT/VPI workstation environment. 50
Figure 18. Finite state diagram for PHIGS 65
Figure 19. ACSYNT/VPI screen layout. 69
Figure 20. Example menu routine flowchart. 80
Figure 21. Example open function flowchart. 81

Figure 22. Example menu driver flowchart. 82
Figure 23. Example execute menu flowchart. 83
Figure 24. Example menu input flowchart. 84
Figure 25. Example process menu input flowchart. 85
Figure 26. Example process menu item flowchart. 86
Figure 27. Example close function flowchart. 87
Figure 28. ACSYNT/VPI data-flow diagram. 105
Figure 29. ACSYNT/VPI geometric modeling cycle. 106
Figure 30. ACSYNT fuselage definition. 107
Figure 31. ACSYNT engine pod definition. 108
Figure 32. ACSYNT airfoil definition. 109
Figure 33. ACSYNT/VPI fuselage definition. 111
Figure 34. ACSYNT/VPI store and engine pod definition. 112
Figure 35. ACSYNT/VPI airfoil definition. 113
Figure 36. Definition of aircraft components by cross-sections. 116
Figure 37. Cross section type definitions. 117
Figure 38. Example cross-section definition using nose component equations. 120
Figure 39. Surface points and tangent vectors for lofted surfaces. 122
Figure 40. Geometric definition of a bicubic patch. 124
Figure 41. ACSYNT/VPI component surface database. 127
Figure 42. ACSYNT/VPI geometry structure network. 129
Figure 43. Structure network for a single aircraft component. 130
Figure 44. Component symmetry example. 131
Figure 45. Polyline grid drawn on a surface patch. 135
Figure 46. Tiling of a component surface patch showing right hand generation of vertex points. 138
Figure 47. Definition of polygon front and back faces by vector cross product. 140
Figure 48. Example of one object obscuring the view of another. 142
Figure 49. ACSYNT/VPI Main Menu Screen 147

Figure 50. ACSYNT/VPI geometry menu showing F16 wireframe geometry. 148
Figure 51. Modified component colors and number of approximation curves. 149
Figure 52. Shaded image of gray aircraft using white light sources. 151
Figure 53. Front view of gray aircraft using yellow light sources. 152
Figure 54. Yellow sun model shading on a red aircraft. 153
Figure 55. Multi-window display, selective shading, and color input. 154
Figure 56. Hidden surface image of aircraft with swept forward wings and V-tail. 156
Figure 57. Shaded image of aircraft showing variability of wing component. 157
Figure 58. Menu structure tier chart. 218
Figure 59. Menu structure tier chart (continued). 219
Figure 60. Menu structure tier chart (continued). 220
Figure 61. Menu structure tier chart (continued). 221
Figure 62. Menu structure tier chart (continued). 222
Figure 63. Menu structure tier chart (continued). 223
Figure 64. Menu structure tier chart (continued). 224

1.0 INTRODUCTION

Overview and Project History

This thesis describes the development of a conceptual design system for supersonic aircraft. The system is based on ACSYNT, an aircraft design code written at the NASA Ames Research center, Moffett Field California [1]. ACSYNT which stands for AirCraft SYNThesis, allows aircraft designers to analyze and optimize preliminary designs of aircraft including fighters, bombers, transports, and missiles. Through a grant from the NASA Ames Research Center, Drs. A. Myklebust and J. R. Mahan and graduate students of Virginia Tech's Mechanical Engineering Department are undertaking the task of enhancing ACSYNT's design and analysis capabilities. The end result will be a highly interactive conceptual aircraft design system called ACSYNT/VPI.

The multi-year ACSYNT/VPI project began during the Spring of 1987 with the formation of a software development team consisting of three graduate students, Jayaram Sankar, Michele Grieshaber, and Steve Wampler and the two above named faculty members from the Virginia Tech Mechanical Engineering Department. The preliminary work included generating support software, creating a software development library, and writing software requirements. In addition, a set of subroutines were written to allow the ACSYNT/VPI software to be developed using the PHIGS graphics standard.

During the week of August 2, 1987, the ACSYNT software was installed by P. Gelhausen of the NASA Ames Research Center on a MicroVAX II at Virginia Tech. After verifying the MicroVAX II installation, the ACSYNT code was converted to run on an IBM 4341 mainframe and again

verified using output supplied by NASA. The IBM 4341 is dedicated to Virginia Tech's CAD/CAM laboratory and was chosen as the primary system for development of the ACSYNT/VPI software.

Project Objectives

In the early stages of the ACSYNT/VPI project, the software development team identified the following two overall objectives:

1. To produce a highly interactive conceptual aircraft design system based on ACSYNT.
2. To enhance ACSYNT's design and analysis capabilities.

The work described in this thesis was primarily designed to address the first objective but lays the foundation for achieving both. Specific design objectives can be found in "Software Requirements" on page 43.

Thesis Organization

Each chapter of this thesis is intended to be read selectively depending upon the reader's needs. "Literature Review" on page 4 and "Description of ACSYNT" on page 6 contain a discussion of recently developed conceptual aircraft design systems and a functional overview of ACSYNT, respectively. The main body documents the development of ACSYNT/VPI and is intended for those readers who will continue to develop and maintain the ACSYNT/VPI project. The last chapter contains a discussion of the project results and recommendations. Finally, for those readers re-

quiring more detailed information, the appendices contain a glossary, flowcharts, example code, example data files, and user's guides.

2.0 LITERATURE REVIEW

Overview of Available Literature

As stated in the "Introduction", ACSYNT's main purpose is to aid in the preliminary or conceptual design of advanced aircraft. The American Heritage Dictionary defines "concept" as "a thought or idea" and "design" as "a project or plan" [2]. Therefore, conceptual design can be defined as "the process of forming a project or plan from a thought or idea". Computer aided conceptual design, such as that performed by ACSYNT, enables a designer to take advantage of a computer's speed, memory, and processing ability early in the design process. The computer gives a designer the ability to analyze a wide range of ideas before proceeding on to the more expensive process of final design. This chapter contains an overview of currently available conceptual design systems, especially those related to aircraft design.

Currently Available Conceptual Design Systems

Of the available commercial conceptual design systems, most are not specifically used for aircraft design. Companies such as Cognition, a start-up company based in Billerica, Massachusetts, are actively involved in the development of conceptual design systems for mechanical engineering. For example, Cognition's "Mechanical Advantage" system allows engineers to sketch ideas on a graphics screen and perform preliminary engineering and cost analysis in order to select the best

design [3]. While such system may be useful in conceptual aircraft design, they are not capable of predicting the overall performance of an aircraft.

Although many proprietary systems may exist, the literature contains only a few examples of conceptual aircraft design systems. One such system, called HESCOMP, is used for helicopter sizing and performance evaluations. HESCOMP was written by S. J. Davis, et al. at Boeing Vertol under NASA and NAVY contracts and is currently in use by the Naval Air Development Center [4]. HESCOMP is very similar to VASCOMP II, a V/STOL aircraft sizing and performance program also written at Boeing Vertol. VASCOMP and HESCOMP can be used together to compare helicopter-V/STOL aircraft designs [5].

HESCOMP was enhanced by the addition of an interactive CAD interface by Lu and Myklebust in 1985. The interface is called HESCAD and allows for easier creation and modification of HESCOMP helicopter models [6].

CDS (Aircraft Configuration System) is a program written by Rockwell International which permits three-dimensional design and lofting of external skin and inlet ducts and arrangement of internal components. With CDS the designer can perform conceptual analysis of wave drag, weight and balance, aerodynamic center, and other parameters. Although CDS is not intended for final design, CDS has the ability to produce complex aircraft models. After the preliminary design is performed with CDS, final design is carried out on a Computervision CAD system using the preliminary design data [7].

3.0 DESCRIPTION OF ACSYNT

ACSYNT Overview

ACSYNT consists of approximately 30,000 lines of FORTRAN code developed at the NASA Ames Research Center. The program is capable of analyzing a wide range of aircraft from "small remotely piloted vehicles to giant transports" including short takeoff vertical landing (STOVL) fighters. In addition, ACSYNT has the potential for "predicting foreign aircraft performance and future designs" [1]. This chapter contains a functional description of ACSYNT and a list of proposed ACSYNT/VPI enhancements.

Functional Description of ACSYNT

As described in the ACSYNT User's Manual, ACSYNT is an "interdisciplinary aircraft synthesis program" which consists of the following modules [1]:

GEOMETRY: calculates aircraft surface areas and volumes

TRAJECTORY: calculates the time and amount of fuel used for each segment of the specified mission

AERODYNAMICS: calculates minimum drag, lift, and induced drag

PROPULSION: calculates the design and off-design performance of a turbojet or turbofan engine

WEIGHTS: assigns initial weight values and component weight multiplying factors

TAKEOFF: predicts the minimum takeoff ground roll, and for short takeoff type aircraft nozzle angles and thrust split.

Additional modules include STRUCTURES, CARGO, GLOBAL (I/O), ECONOMICS, SUMMARY OUTPUT, GRAPHICS FILE, and NAVY but are not part of the Virginia Tech version of the code. Each of the above modules are called from a control program called COPES which stands for "Control Program for Engineering Synthesis" [1].

Input to ACSYNT consists of a user created data file containing the initial configuration, mission profile, and estimated gross weight of the aircraft. The actual gross weight is calculated by iteration until the difference between the new and previous weights are within a specified tolerance. Iteration is necessary because the fuel required to fly a specified mission is a function of the aircraft's gross weight of which fuel is a major component.

In addition to calculating the gross weight of an aircraft, ACSYNT is also capable of performing optimization and sensitivity analysis. Optimization requires the user to specify a variable to be optimized and a set of design constraints. For example, wing area can be optimized with respect to gross weight, turn rate, and excess power. Sensitivity analysis involves determining the effect of design changes on the gross weight of the aircraft. The ACSYNT User's Manual defines sensitivity as follows:

$$S = \frac{\delta W / W}{\delta P / P_n} \tag{3.1}$$

Where:

S = sensitivity of model to design changes
δW = change in gross weight
W = gross weight
δP = change in design parameter
P_n = nominal value of design parameter

If the change in the design parameter is positive ($\delta P > 0.0$) and the sensitivity is negative ($S < 0.0$) then the gross weight (W) can be reduced by reducing the design parameter (P). The same is true

if the change in the design parameter is negative ($\delta P < 0.0$) and the sensitivity is positive ($s > 0.0$) [1].

ACSYNT/VPI Enhancements

The final version of ACSYNT/VPI will contain many enhancements to the current noninteractive version of ACSYNT. For example, the user will be able to perform the following:

- create, edit and delete parametric aircraft components interactively without the use of data files
- rotate and translate the aircraft in three dimensions for viewing purposes
- produce hidden surface and shaded images of the aircraft
- produce mission profiles interactively
- detect and correct mistakes before the model is processed by ACSYNT
- apply optimization changes suggested by ACSYNT automatically
- perform additional design and analysis not yet available in ACSYNT.

Detailed software requirements and design specifications for ACSYNT/VPI can be found in "Software Requirements" on page 43 and "Software Design" on page 62. The following chapter describes the software development methods used throughout the project.

4.0 SOFTWARE DEVELOPMENT METHODS

Overview

This chapter describes the work, design, documentation, and change control methods used to develop the ACSYNT/VPI software and is recommended for those readers who will continue the development effort.

Work Methods

All of the ACSYNT/VPI software was developed on an IBM 4341 mainframe running IBM's VM/CMS operating system. Each member of the development team had a CMS userid with access to text editors, compilers, printers, and a wide range of computer aided design software. To speed the development of ACSYNT/VPI, each member worked on his or her own userid but performed testing from a central userid set up for the project. Figure 1 on page 10 illustrates the function of the testing userid in the ACSYNT/VPI project. Each time the software was tested, the testing userid would access and execute the latest version of development code.

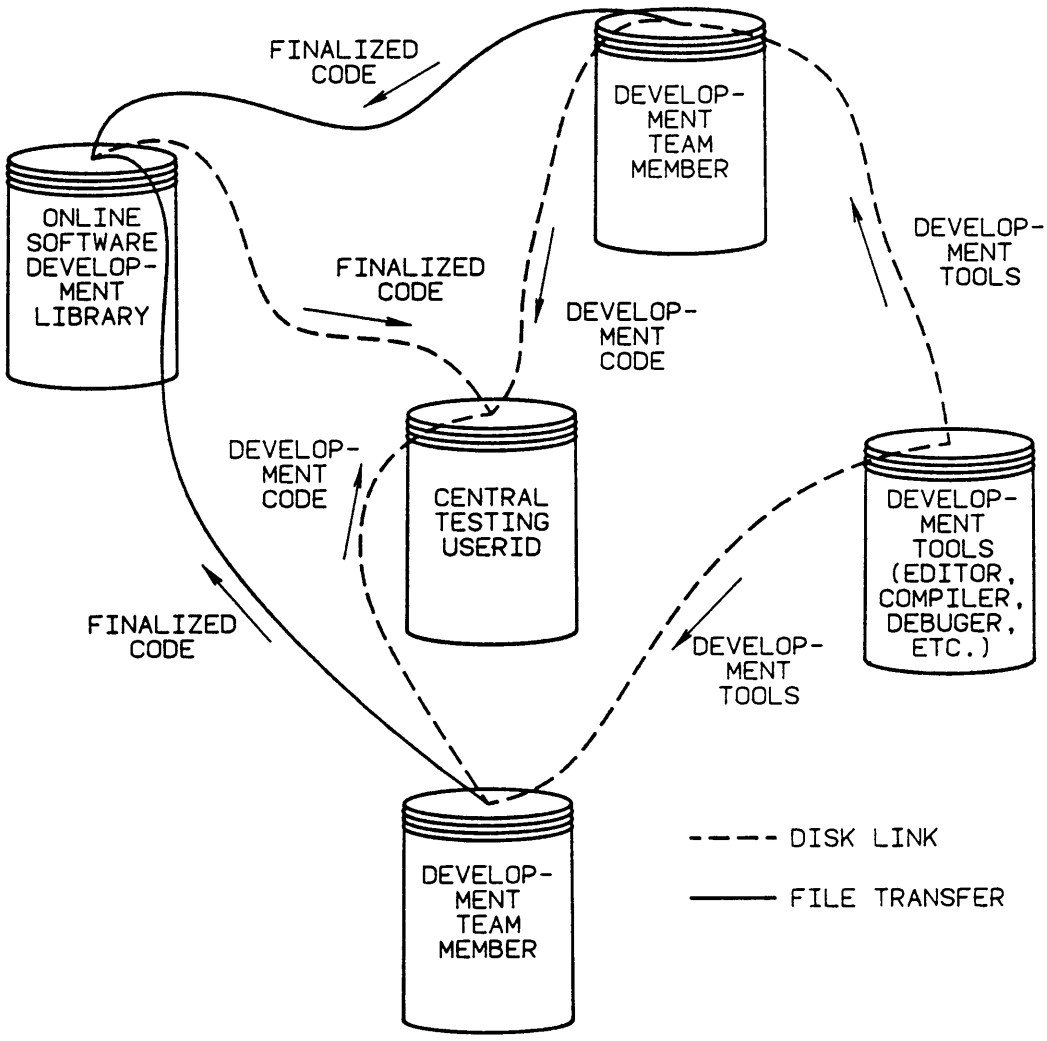


Figure 1. Centralized testing of ACSYNT/VPI software.

Design Methods

Overview

Throughout the ACSYNT/VPI project, the "top-down" method of software design has been used. Top down design is characterized by a logical progression from an abstract problem statement to concrete programming solution. At each level of the design process, a portion of the overall problem is solved resulting in a well defined program which solves the stated problem. Figure 2 on page 12 illustrates the activities involved in top down software development.

Requirements Generation

Part of defining the problem that a software project is intended to solve involves generating a set of software requirements. Software requirements serve two functions. First, they define in detail the functions the software must perform, and second, they provide a means of judging the progress of the development effort. At the beginning of the ACSYNT/VPI project, the following two types of software requirements were generated:

user: requirements imposed on the software from the user's point of view

functional: requirements imposed on the software from the designer's point of view, typically containing a list of database, hardware, and operational requirements.

"Software Requirements" on page 43 contains a complete set of requirements generated for the ACSYNT/VPI project.

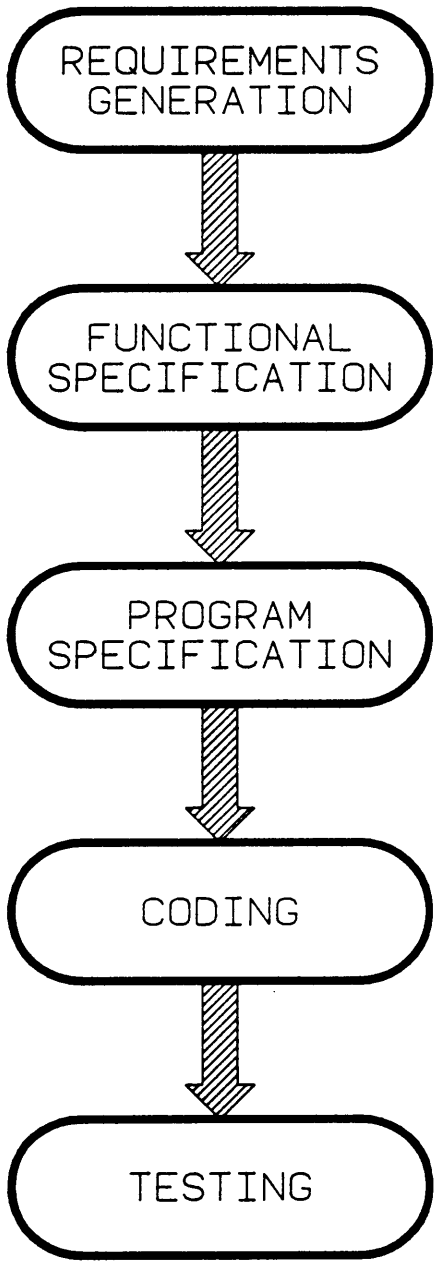


Figure 2. Top down development activities [10] .

Design Process

As shown in Figure 2 on page 12, the program definition, design, production, and verification activities follow the generation of software requirements. Figure 3 on page 14 illustrates how each activity is incorporated into what Tausworthe [10] calls a "design phase". The product of a design phase is one portion of the overall program. The following sections describe a typical design phase in terms of functional, program, code, and test specifications.

SOFTWARE FUNCTIONAL SPECIFICATIONS

Software functional specifications define the functions the software will perform and form the basis for both the program and test specifications. For the ACSYNT/VPI project, functional specification were written and illustrated using tier charts. An example tier chart is shown in Figure 4 on page 15 where each block represents an ACSYNT/VPI program module. In programming terms, a module is a set of one or more procedures which perform a specified function.

A good functional specification will produce "functionally cohesive modules" meaning that all of the components of each module are related directly to the performance of a single function [10]. Functionally cohesive modules tend to be easy to understand, code, debug, and modify. The other forms of module cohesion are communicational, procedural, temporal, logical, and coincidental and are defined in detail by reference 10.

Another concern addressed during the functional specification of the ACSYNT/VPI code was that of module coupling. Module coupling is defined in reference 10 as "a measure of data connectivity between a module and the program in which it is imbedded." As shown in Figure 5 on page 17, the preferred means of module coupling is called "data coupling" which involves passing data between modules only as arguments. Data coupling has been shown to be the lowest form of module

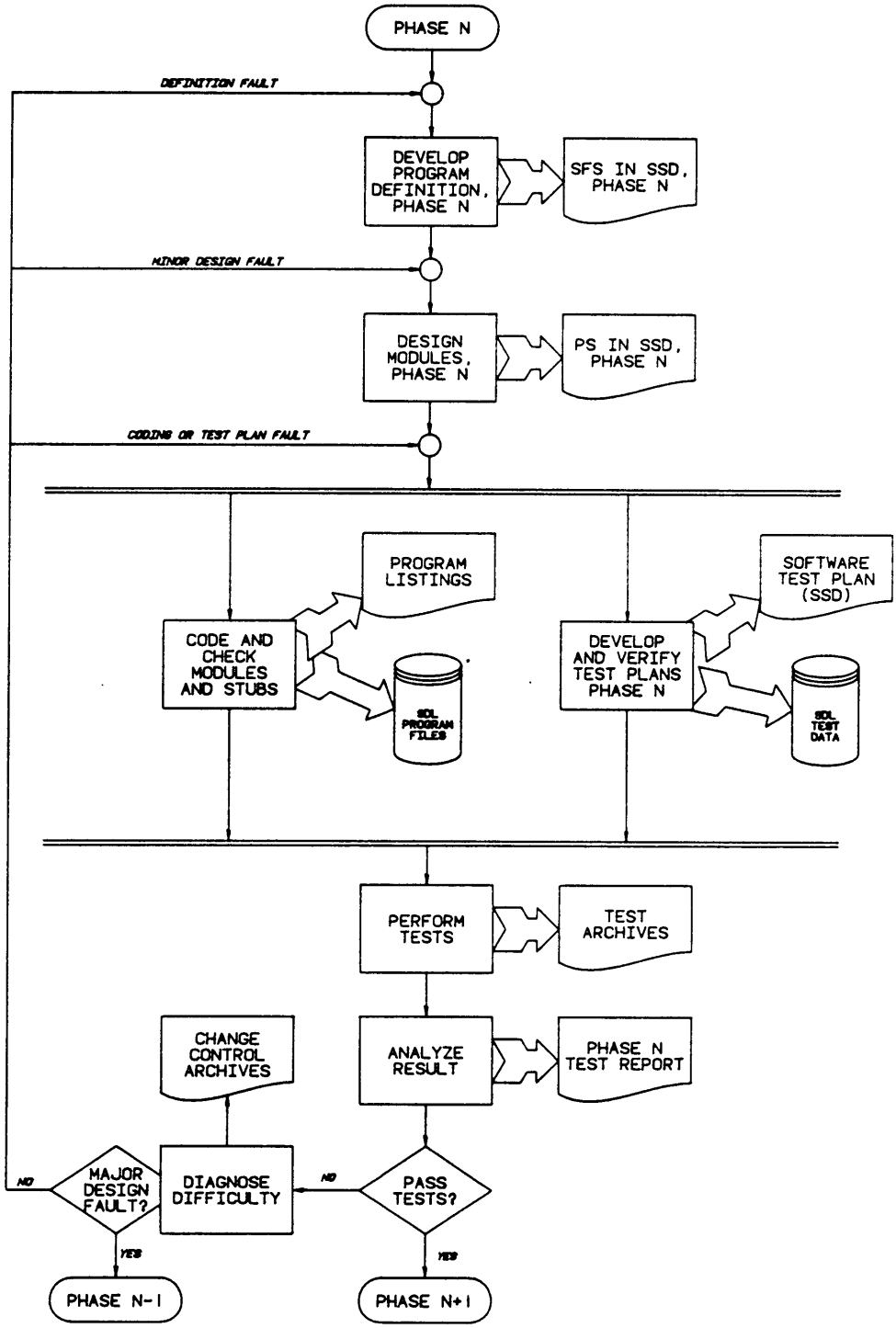


Figure 3. Top-down development flowchart [10].

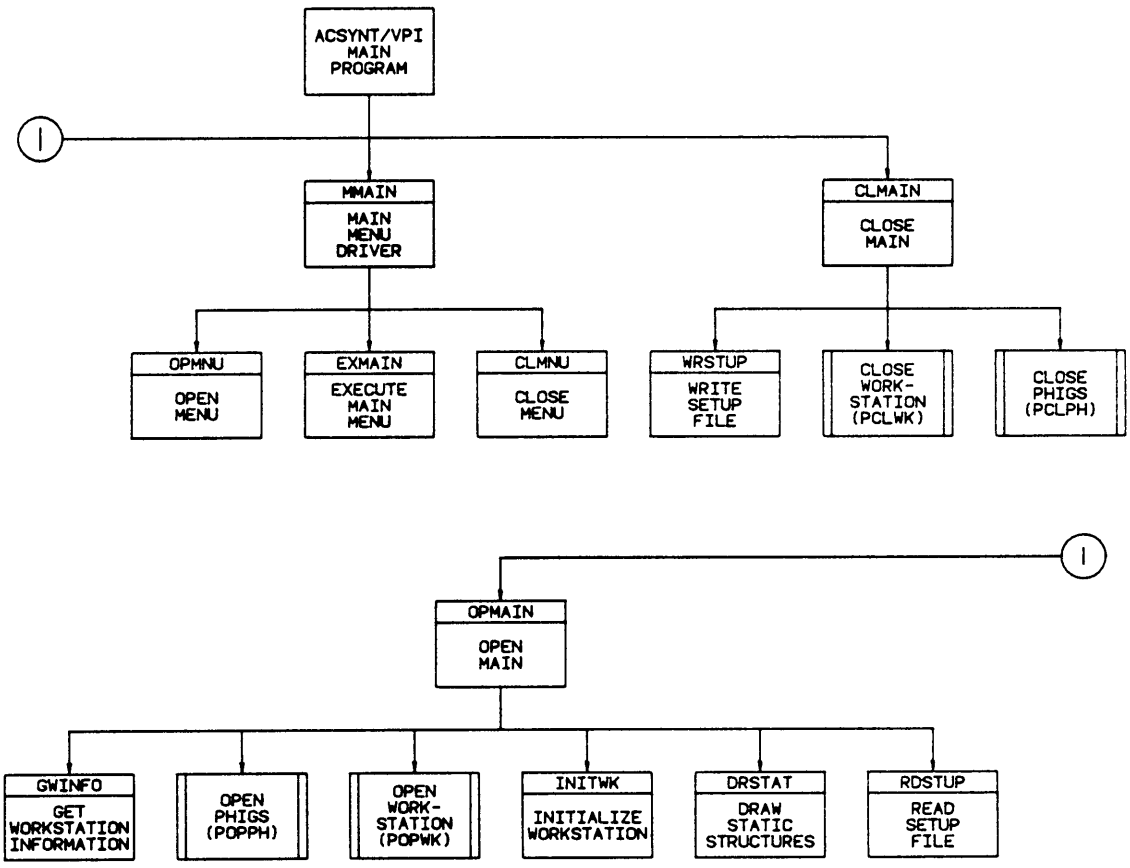


Figure 4. Example ACSYNT/VPI tier chart.

coupling and sufficient for any program, but from a practical point of view, other forms of module coupling are many times necessary [10].

PROGRAM SPECIFICATIONS

A program specification defines how the software will be coded to implement the functions defined in the software functional specification. For the ACSYNT/VPI project, flowcharts where used to document the program specification in order to pass the program's design on to future design team members. The objective was to prevent future designers from having to read through the source code in order to understand the function and flow of the program.

Reference 10 defines a flowchart as "a graphical representation for the definition, analysis, or solution of a problem." For software development, flowcharts generally refer to drawings which describe the logical sequence of operations in a program. An example ACSYNT/VPI flowchart is shown in Figure 6 on page 18. Referring to the example figure, flowcharting symbols with horizontal stripes represent subroutines which are defined on other flowcharting pages. Symbols with vertical stripes represent subroutines defined external to the ACSYNT/VPI code such as system subroutines graphics subroutines. The other flowcharting symbols used in ACSYNT/VPI flowcharts are described in reference 11 and also in "Software Development Standards" on page 29. Organization of the ACSYNT/VPI flowcharts and resulting code followed the Dewey-Decimal system described below and in detail in reference 10.

As shown in Figure 7 on page 20, each symbol in a flowchart is given an integer number starting with the top symbol and moving down and to the right. According to the Dewey-Decimal system, the flowchart for each of the horizontally striped symbols will be numbered "m.n", where "m" is the number of the current flowchart and "n" is the symbol number within flowchart "m". For example, suppose a flowchart's identification number is "1.2" (n = 1.2). If the fifth symbol (m

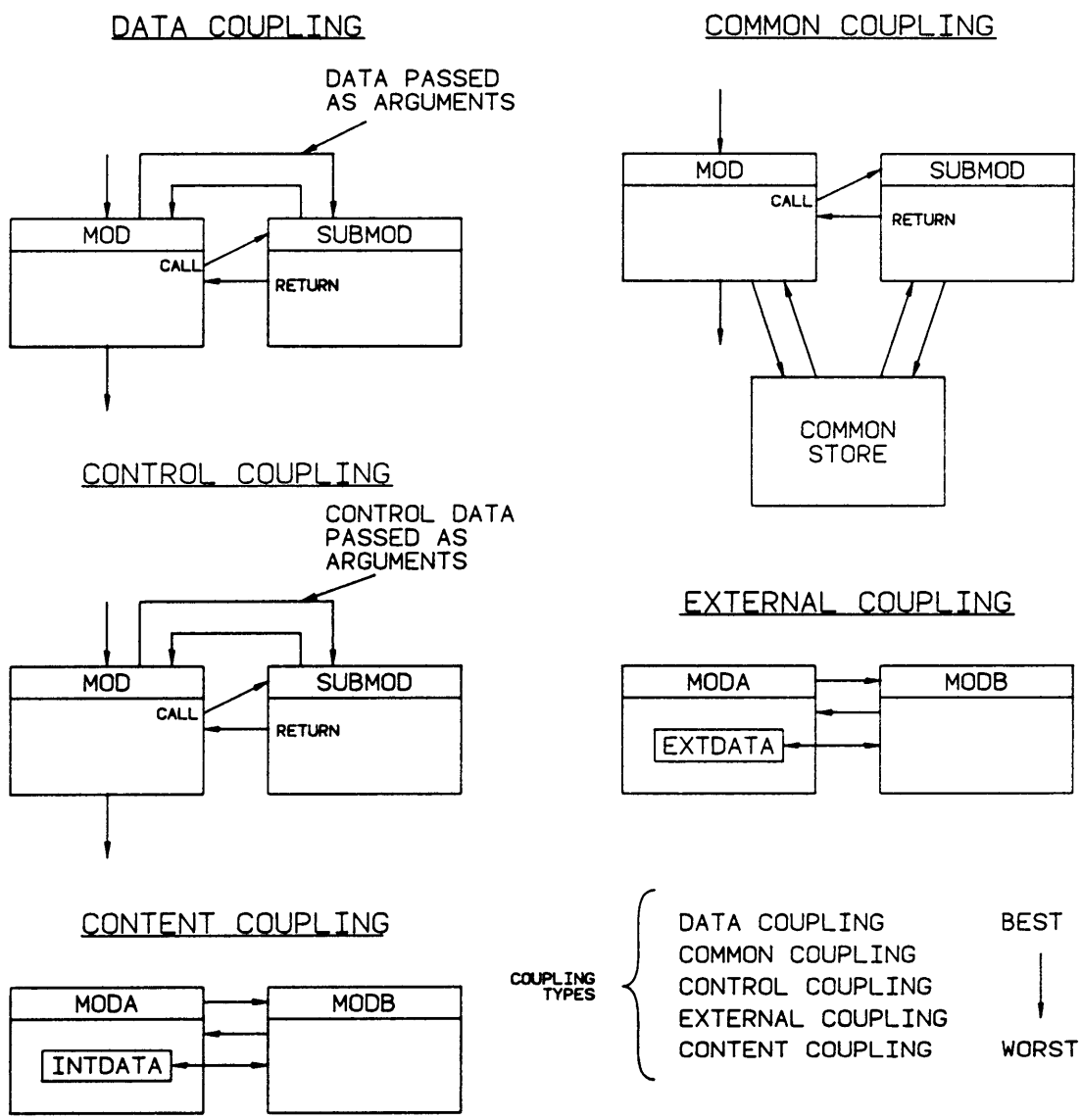


Figure 5. Five types of module coupling [10].

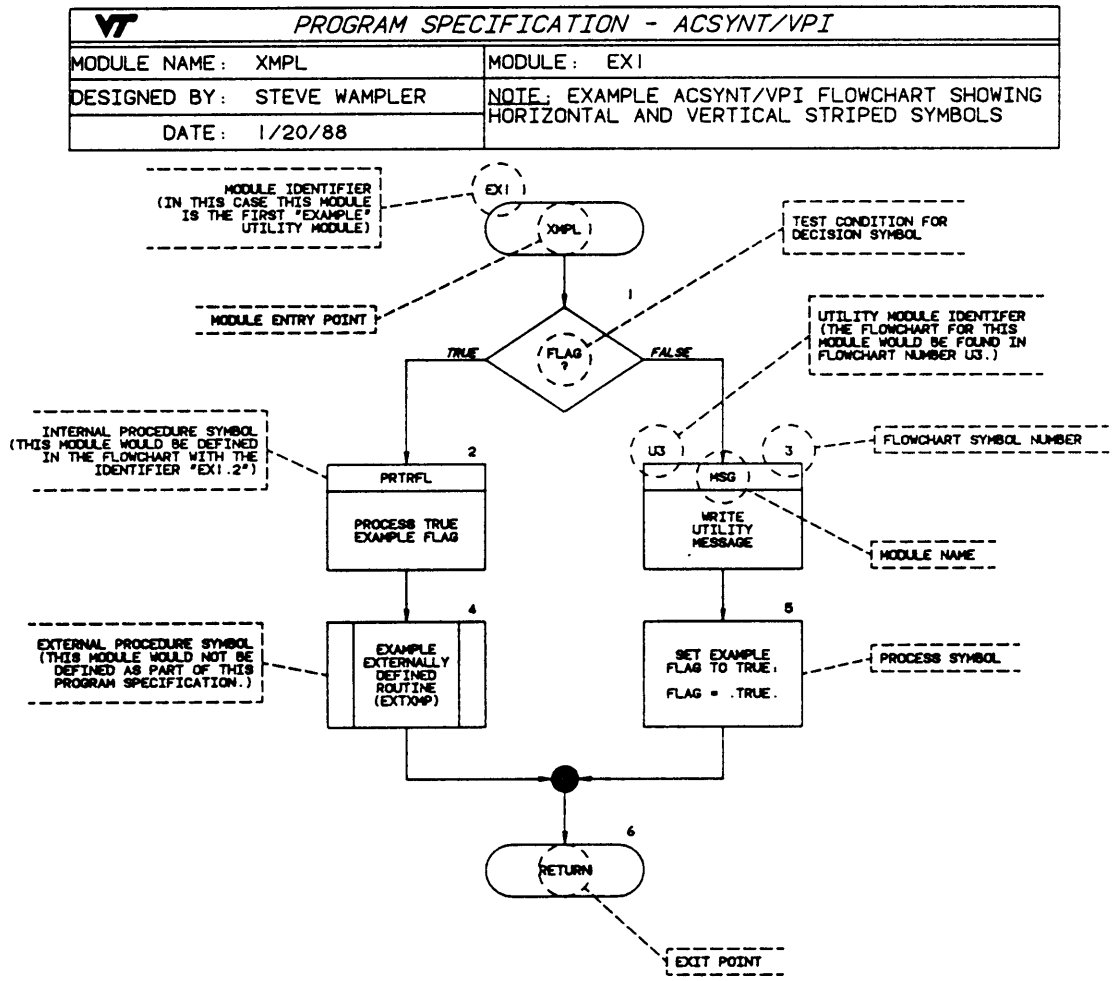


Figure 6. Example ACSYNT/VPI flowchart.

= 5) in flowchart "1.2" is horizontally striped, then the flowchart for the fifth symbol will be contained in flowchart number "1.2.5.".

For subroutines called from more than one point in the code, the subroutine's flowchart is given a unique alphanumeric identifier. For example, the flowchart for a subroutine used to normalize a vector might be given the unique identifier "V1" as shown in Figure 6 on page 18. The "V" indicates that the subroutine is used for vector type operations and the "1" indicates that it is the first vector subroutine to be defined. The subsequent vector subroutines would be given the identifiers "V2", "V3", "V4", etc.

To define an ACSYNT/VPI subroutine, a flowchart was first sketched on paper then used to produce the source code. After testing the code and making any necessary changes, the flowchart was drawn using CADAM (Computer Augmented Design And Manufacturing) and placed in the project notebook. The June 1988 ACSYNT/VPI progress report will contain approximately 400 flowcharts for version X1.0.0 of the ACSYNT/VPI code [8, 9].

TEST SPECIFICATIONS

As shown in Figure 3 on page 14 after producing a program specification, the source code is generated in parallel with test specifications. Test specifications detail how the code will be tested for correctness. A correct program is one which performs as specified in the program's software functional specifications [10].

Because each individual module of the ACSYNT/VPI code was designed, coded, and tested by the same design team member, no formal software testing specifications were created. Instead, as each module was added to the program, the module was reviewed and tested by the designer and other members of the development team. In particular, modules were checked for the following [12].

- Module preamble:

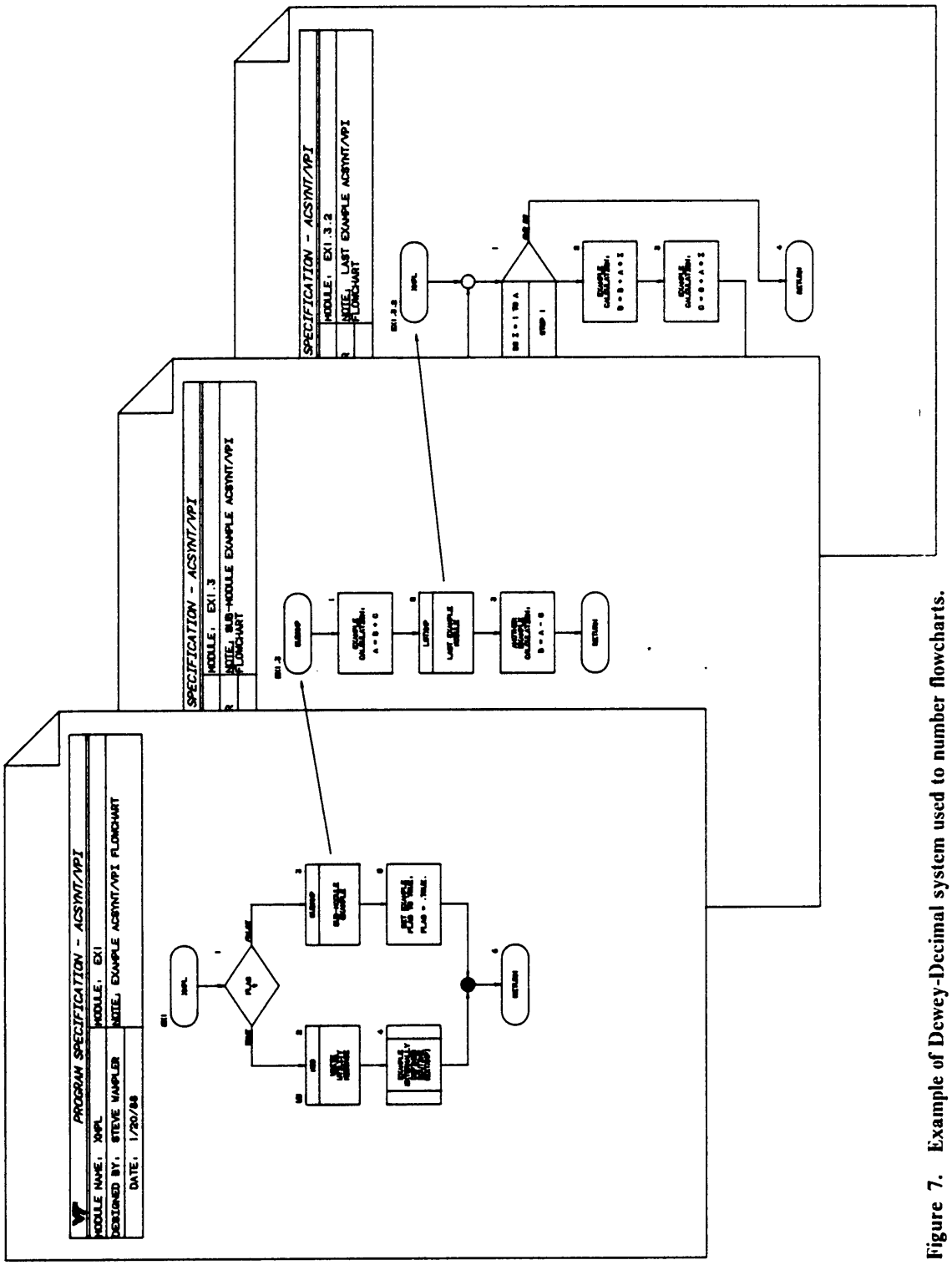


Figure 7. Example of Dewey-Decimal system used to number flowcharts.

    - name, part number, and version level included?
    - functional description accurate and unambiguous?
    - all global variables defined adequately?
    - key local variables defined adequately?
    - calling sequence and parameters adequately defined?
    - method/algorithm explained as appropriate?
    - exceptional exits/conditions noted?
- Consistent with specifications?
- Well-structured module design?
- Efficient module design?
- Quality and placement of commenting?
- Effective usage of library routines?
- Compliance with standards?
- Good programming style?
- Are character string delimiters/terminators correct?
- Are all variables set before use? Used after set?
- Will logical tests operate as intended? For example, should a test be for "greater than" or "greater than or equal to"? In comparing strings, is the correct number of characters tested starting at the correct offset, if any?
- Are all parameters to a subroutine call set properly?
- Are calling references correct as to purpose, number, and types of arguments? Is an erroneous attempt made to set constants passed to a routine?
- When returning from a routine, especially prematurely, is "cleanup" complete? Buffers freed? Files closed?
- Are all errors valid? What happens if a second error is encountered while processing the first one?
- If a variable is tested for several values, are tests made for all possible values? Is an "otherwise" case provided?
- Are there absolute constants where there should be symbolic?
- Are all loop variables initialized properly before entry?
- Are ELSE blocks provided as needed?
- Is code graphics device independent in terms of:
    - colors
    - screen size
    - graphical input device capabilities

    - primitive attribute capabilities?
- Do input routines handle:
    - values out of bounds
    - invalid characters in the input
    - invalid file names?
- Are graphical input devices reset after an input error occurs?
- Does the use of a logical graphical input device interfere with another logical graphical input device?
- If an error does occur can the module recover without loss of data?
- Does the module clean up after itself once its function is complete such as releasing input devices and screen space?
- Does the module have any adverse effects on the function of any other modules?

The testing of ACSYNT/VPI was facilitated by the fact that the program is menu driven, which allows selective testing of new portions of the code. "Testability" was also considered in the design of the code.

Documentation

Program documentation is a by-product of top down design. During each phase of development, documents such as functional and program specifications are generated which fully define the resulting code. If the final code does not match the documentation, then the documentation is rendered obsolete. Figure 8 on page 23 illustrates the flow of information between documents from the beginning of a software project to the time of release. Many of the documents shown took the form of design team notes and conversations and did not need to be formally generated during the first year of the ACSYNT/VPI project [10].

As shown in Figure 8, a great deal of documentation could be generated for a given software project. Unfortunately, a difficult balance must be struck between too little documentation and too

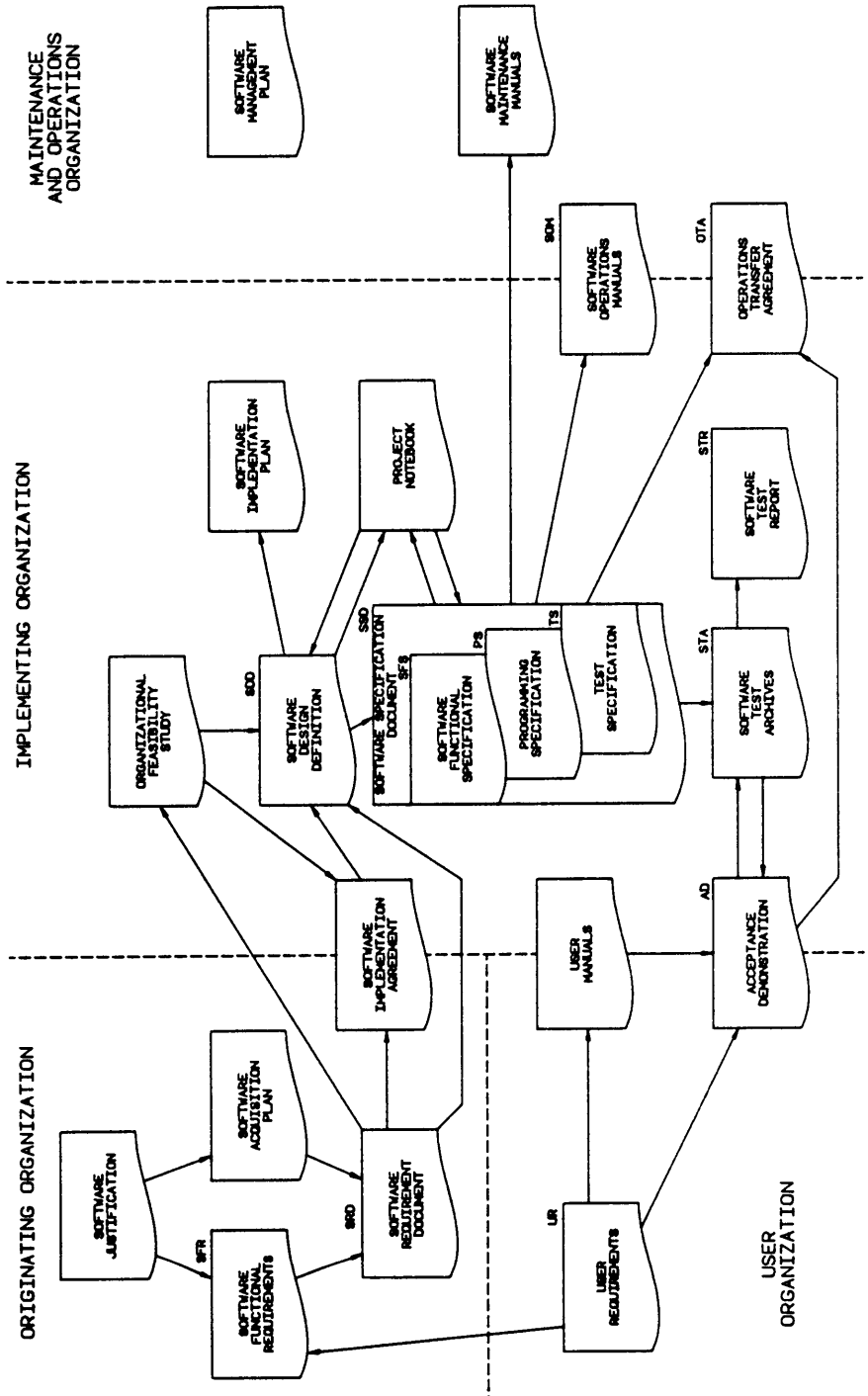


Figure 8. Documents generated during software development.

much documentation. Figure 9 on page 25 shows that the level of documentation can greatly effect the cost of development and maintenance. Reference [10] states the following concerning the optimum level of documentation:

> The optimum level of documentation depends on several factors. When the costs involved cover the developmental phase only--by a fixed-man team--then a low level of documentation suffices. But when there is the possibility that design team personnel can change during the project, or when later maintainability of the program is stated as a documentation requirement, then a higher level is needed. Then too, if humans must draft flowcharts, type narrative, and then maintain these in an error-free condition, there is quite a different cost associated with documentation than when automatic documentation facilities are available.

Because the ACSYNT/VPI design team may have new members as often as once a year, detailed software requirements and program specifications were generated to document the design of ACSYNT/VPI. The other forms of documentation tended to be low due to limited personnel and time. Recommendations for improving the project documentation are included in the chapter titled "Results and Recommendations" on page 145.

All documents created for the project including the program source code can be found in the project's software development library (SDL) or project notebook. The SDL and project notebook are discussed in detail in the following section "Change Control Methods".

Change Control Methods

Overview

Anytime a development team works on the same software project, methods should be developed to control changes made to the documents and source code. Change control methods help to prevent the following:

- two or more people changing a document or source file at the same time

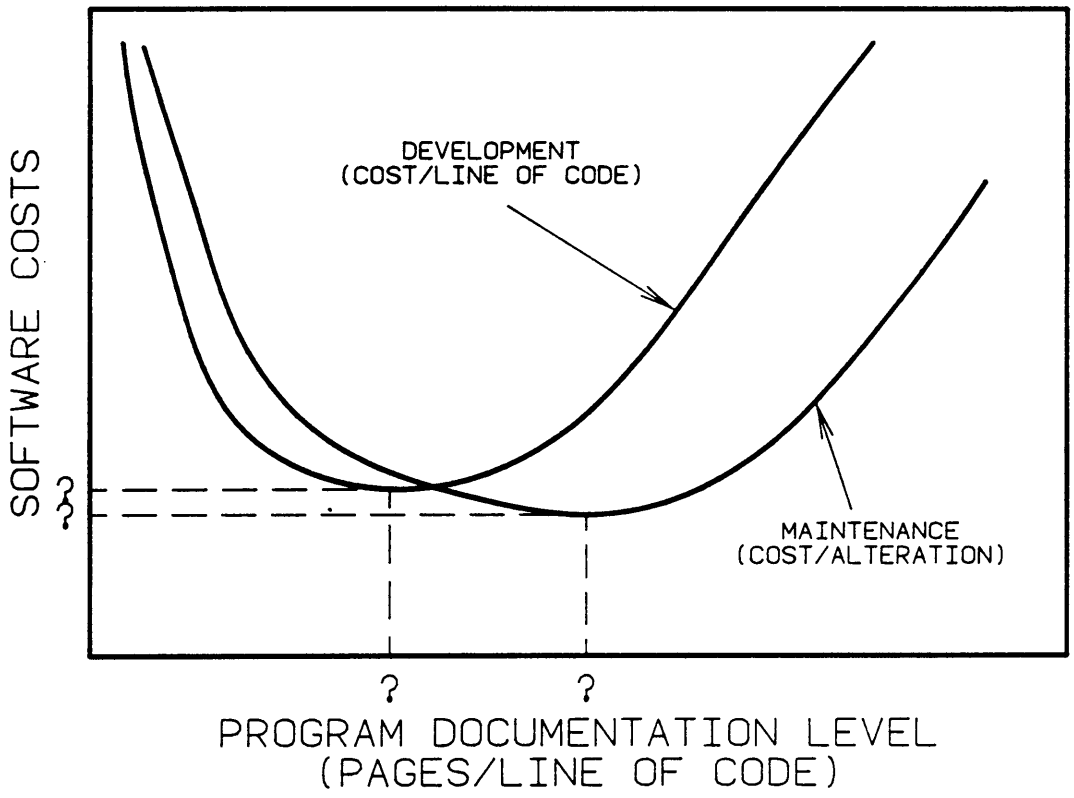


Figure 9. Program costs versus documentation level [10].

- a master file from being erased or overwritten by accident
- unauthorized release of unreleased code
- mixing of old and new versions of the code at time of release

For the ACSYNT/VPI project, change control is accomplished three ways, through a software development library (SDL), project notebook, and code version numbers.

Software Development Library

After creation and error checking, all documents, source code, object code and data files are placed in the ACSYNT/VPI software development library (SDL). All files contained in the SDL are treated as master copies and can only be changed by authorized development team members. All files residing outside of the SDL are considered under change control, under development, or obsolete as defined below:

change control file:
any file which has been formally checked out of the SDL. After a file has been checked out, no other team member can access the file until the original has been checked back in.

development file:
any file which has not yet been submitted to the SDL but is part of the development project.

obsolete file:
any file which is a copy or old version of a file contained in the SDL but has not been offically checked out of the SDL.

"Appendix M. Software Development Library User's Guide" on page 257 contains instructions on how to use the ACSYNT/VPI online software development library.

Project Notebook

The ACSYNT/VPI project notebook serves as a reference source for all members of the software development team. The notebook contains the following items:

- Software Requirements
- Research Material
- Description of Project's Software Development Tools
- Software Discrepancy Reports
- Code Review Checklists
- PHIGS Information
- ACSYNT Input Examples
- ACSYNT Output Examples
- ACSYNT User's Manual
- Program Specification Flowcharts

The program specification flowcharts are the most important documents stored in the project notebook because they represent the current state of the project code. Although the flowcharts are also stored in CADAM drawing files, the plotted flowcharts in the project notebook are considered to be the master copies. Any changes made to the program are first made by hand to the flowcharts in the project notebook and then to the source code. The CADAM flowcharts are then updated and replotted at a convenient time. By using the above method, the flowcharts in the project notebook always reflect the current state of the source code.

Version Numbers

Version numbers provide a means of controlling the release of software. The method of reference 12 has been modified to give the following format used to denote version numbers for the ACSYNT/VPI software:

```
Cn.m.l
```

where:

C is one of the following letters:

V for a standard release piece of software
S for a special version of released software
X for unreleased software

- **n** is the major version number
- **m** is the minor version number
- **l** is the release modification number

A detailed discussion of software configuration control can be found in reference 12.

The following chapter discusses how the development methods explained above were adopted in terms of software development standards.

5.0 SOFTWARE DEVELOPMENT STANDARDS

Overview

Reference 10 states that the goal of software development standards is to reduce the time spent debugging code and to increase the ability of those who come in contact with the code to understand the code's function and operation. Good development standards facilitate the operation and alteration of a program as the program evolves and have been shown to improve overall project productivity by as much as 50 percent [10]. For the ACSYNT/VPI project, standards have also helped ensure consistency between the code written by different design team members.

This chapter outlines the software development standards of special importance to the ACSYNT/VPI project. Specifically, ACSYNT/VPI project standards for flowcharting and graphics are discussed under the heading "ACSYNT/VPI Design Standards" on page 30, and FORTRAN and commenting standards under the heading "ACSYNT/VPI Coding Standards" on page 38. The standards presented are not intended to hinder the design process but to enhance the final software product. A much more detailed discussion of generalized software development standards can be found in reference 11.

ACSYNT/VPI Design Standards

The following American National Standards were applied to the design of the ACSYNT/VPI software:

- ***American National Standard Flowchart Symbols and Their Usage in Information Processing***, ANSI standard number X3.5-1970, Sept. 1, 1970.
- ***Draft Proposed American National Standard Programmer's Hierarchical Interactive Graphics System (PHIGS)***, ANSI standard number X3.144-198x, Oct. 23, 1986.
- ***Draft Proposed American National Standard for the FORTRAN Language Binding of the Programmer's Hierarchical Interactive Graphics Standard (PHIGS)***, ANSI standard number X3.144.1-198x, Feb. 20, 1986.

Each of the standards listed above are described in light of the ACSYNT/VPI project in the following sections.

ACSYNT/VPI Flowcharting Standards

FLOWCHARTING SYMBOLS

During the program specification stage of the ACSYNT/VPI project, flowcharts were generated as discussed in "Software Development Methods" on page 9. The flowchart symbols used in the ACSYNT/VPI project closely followed to the ANSI flowcharting standard referenced above although some symbols were created or modified to promote structured programming. A description of the flowchart symbols used to define ACSYNT/VPI are given in Figure 10 on page 31 and Figure 11 on page 32.

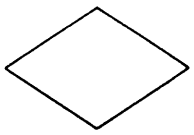
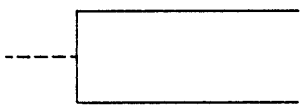
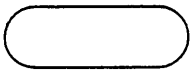
SYMBOL	DESCRIPTION
	INPUT/OUTPUT SYMBOL: THIS SYMBOL REPRESENTS AN I/O MEDIUM OR FUNCTION, SUCH AS MAKING AVAILABLE INFORMATION FOR PROCESSING (INPUT), OR THE RECORDING OF PROCESSED INFORMATION (OUTPUT).
	PROCESS SYMBOL: THIS SYMBOL REPRESENTS ANY KIND OF PROCESSING; FOR EXAMPLE, THE PROCESS OF EXECUTING A DEFINED OPERATION OR GROUP OF OPERATIONS RESULTING IN A CHANGE IN VALUE, FORM, OR LOCATION OF INFORMATION.
	DECISION SYMBOL: THIS SYMBOL REPRESENTS A SPECIFIC DECISION OR SWITCH OPERATION THAT DETERMINES WHICH OF A NUMBER OF ALTERNATE PATHS IS TO BE FOLLOWED. THIS SYMBOL MAY NOT BE STRIPED.
	COMMENT OR ANNOTATION: THIS SYMBOL IS USED TO ENCLOSE DESCRIPTIVE COMMENTS OR EXPLANATORY NOTES AS CLARIFICATION. THE BROKEN LINE IS CONNECTED TO ANY SYMBOL WHERE THE ANNOTATION IS MEANINGFUL.
	SEQUENCE, CONTROL FLOW: SEQUENTIAL PROGRAM FLOW IS INDICATED BY SINGLE-LINE ARROWS CONNECTING SYMBOLS. ARROWHEADS ARE NECESSARY TO SHOW DIRECTION. USE ONLY ONE ARROWHEAD PER FLOWLINE, AT ITS END.
	LOOP COLLECTING NODE: THE SMALL OPEN CIRCLE SHOWN REPRESENTS THE ITERATION POINT IN A LOOPING OPERATION.
	DECISION COLLECTING NODE: THE SMALL BLACKED CIRCLE SHOWN REPRESENTS THE MERGIN OF ALTERNATIVE FLOW PATHS IN A PROGRAM.
	TERMINAL SYMBOL: THE SYMBOL SHOWN REPRESENTS THE ENTRY OR EXIT POINT OF A FLOWCHART.

Figure 10. Basic flowchart symbols [11].

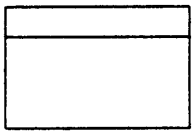
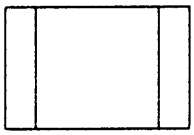
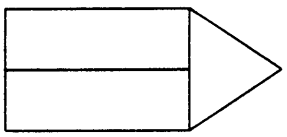
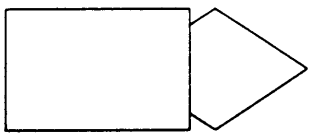
SYMBOL	DESCRIPTION
	INTERNAL PROCEDURE: THE SYMBOL SHOWN REPRESENTS A NAMED PROCEDURE (SUBPROGRAM OR SUBROUTINE) MODULE THAT HAS A MORE DETAILED REPRESENTATION ELSEWHERE IN THE SAME SET OF FLOWCHARTS.
	EXTERNAL PROCEDURE: THE SYMBOL SHOWN REPRESETS A NAMED PROCEDURE (SUBROUTINE) MODULE OR LOGICAL UNIT THAT IS NOT DETAILED IN THIS SAME SET OF FLOWCHARTS.
	INDEXED LOOPING: THE SYMBOL SHOWN REPRESENTS LOOP INITIALIZATION, PREDICATE TESTING, AND UPDATE FUNCTIONS. TESTING ALWAYS FOLLOWS EVERY INITIALIZATION AND UPDATE.
	NON-NORMAL EXIT: THE SYMBOL SHOWN REPRESENTS THE EXIT FROM A PROCESS DUE TO ABNORMAL OR PARANORMAL EVENTS.

Figure 11. Specialized flowchart symbols [11].

FLOWCHARTING CONSTRUCTS

In order to promote top down structured programming, all flowcharts were generated by combining the programming constructs shown in Figure 12 on page 34. The constructs are described as follows [10]:

- DO f THEN g
- IF c THEN f ELSE g
- WHILE c DO f
- DO f WHILE c
- CASE

Where:

- f and g are "proper programs" meaning that they have only one entry and one exit point
- c is a test condition
- IF, THEN, ELSE, WHILE, DO, and CASE are logical operators.

In addition, a special construct was used to denote a FORTRAN "DO LOOP" as shown in Figure 13 on page 35.

By nesting the proper program constructs listed above, the resulting structure will also be a proper program having only one entry and one exit point with no flowlines intersecting. The final result is software which is structured, easy to understand, implement, and modify.

ACSYNT/VPI Graphics Standards

OVERVIEW

A large part of the ACSYNT/VPI project consists of creating a graphical user interface for ACSYNT. A graphical interface allows the aircraft designer to interact with a model and an image

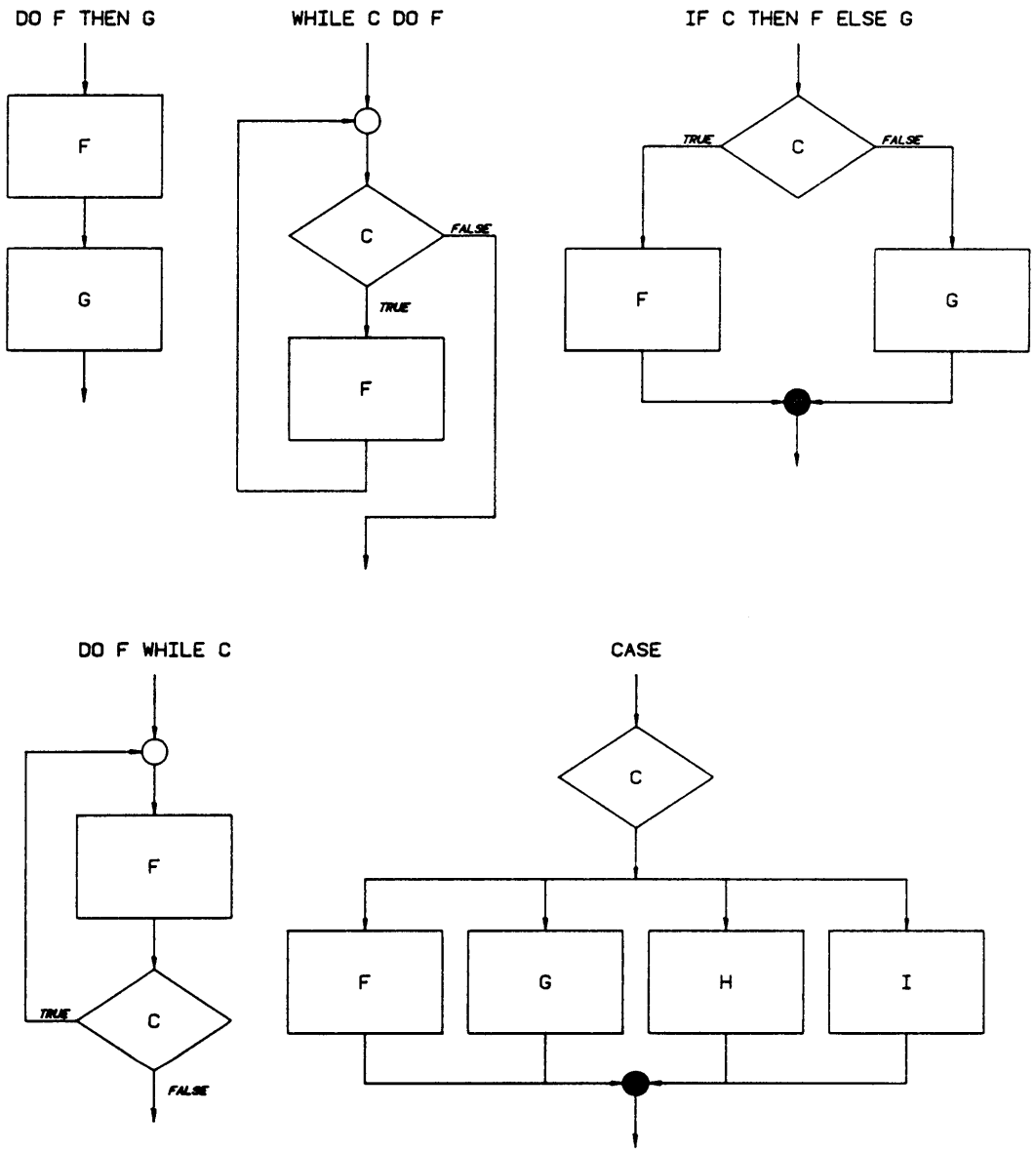
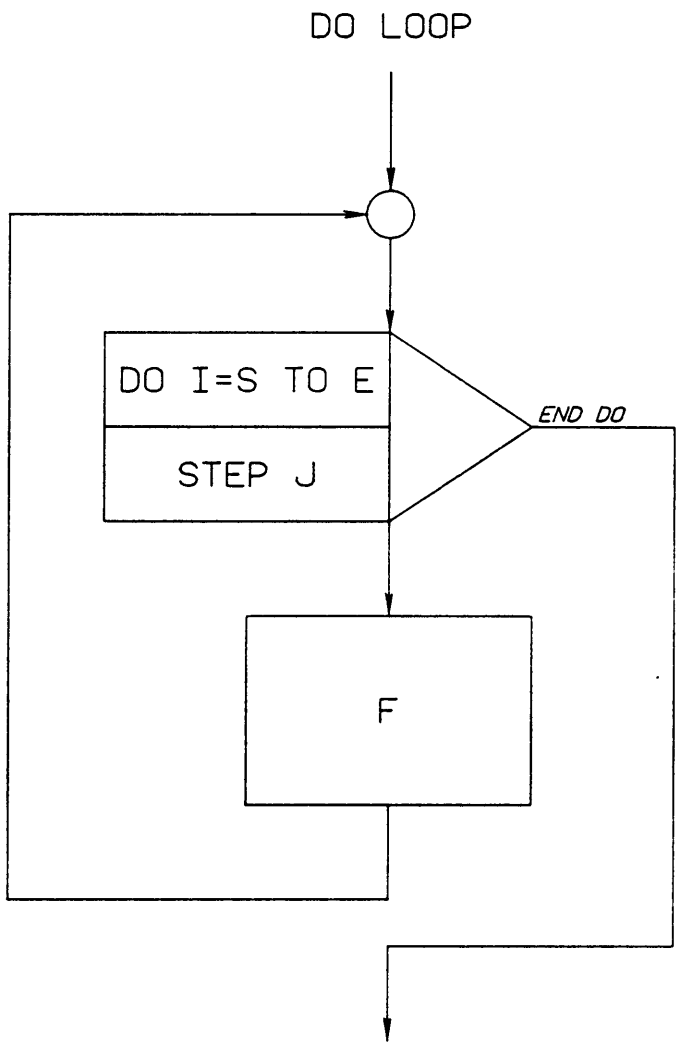


Figure 12. Programming constructs.



WHERE:

I = LOOP COUNTER
S = STARTING LOOP INDEX
E = ENDING LOOP INDEX
J = LOOP INDEX INCREMENT

Figure 13. Programming constructs for FORTRAN "DO LOOP".

of the aircraft he or she is designing. The interactive software generates the image by calling graphics subroutines to control the designer's graphics terminal. The following is an sample list of functions that a typical set of graphics routines should be able to perform:

- draw graphics primitives such as:
  - lines
  - markers (small points, stars, boxes, etc.)
  - text
  - polygons (filled with color or a pattern, empty, etc.)
- set primitive attributes such as:
  - color
  - size
  - type (line type, marker type, text font, etc.)
- get user input from devices such as a keyboard, light pen, cursor tablet, dial, or button box.

THE PHIGS GRAPHICS STANDARD

For the ACSYNT/VPI project, a standardized set of graphics routines called PHIGS were used to perform all graphical input and output. PHIGS, which stands for "Programmer's Hierarchical Interactive Graphics System" is currently the proposed American National Standard for three dimensional computer graphics. PHIGS was chosen to help ensure that the ACSYNT/VPI software is "device independent." Device independence means that the ACSYNT/VPI software will be able to run on a wide range of graphics workstations with little or no modification.

Not only does PHIGS help promote device independence, PHIGS also frees the programmer from having to develop low level graphics routines allowing him or her to concentrate on the graphics application. The PHIGS standard is described in reference 13 as follows:

- allows graphics application programs to be easily transported between installations
- aids graphics applications programmers in understanding and using graphics methods

- guides device manufacturers on useful graphics capabilities
- performs many functions currently performed by graphics applications; thus, off loading the graphics application development effort.

PHIGS subroutines can perform the following [13]:

- output graphical primitives
- control appearance of graphical primitives with attributes
- control multiple graphical workstations
- control 2D and 3D transformations and coordinate systems
- generate, modify, and control groups of primitives called structures
- modify the hierarchical relationship of structures
- obtain graphical input throught logical graphical input devices
- archive and retrieve structures and structure hierarchies
- inquire the capabilities and states of the graphics system
- handle errors.

Also, extensions to the PHIGS standard called "PHIGS+" have been proposed which will add the ability to generate curves, surfaces, and shaded images. The ACSYNT/VPI project will greatly benefit from PHIGS+ by providing parallel but faster curve, surface, and shading calculations. PHIGS+ will permit ACSYNT/VPI to take advantage of the rendering capabilities of high performance graphics workstations and still be device independent.

In order to ensure device independence, the ACSYNT/VPI software had to follow the PHIGS standard very closely. Unfortunately, IBM's implementation of PHIGS, called "graPHIGS", has many nonstandard features. For example, the PHIGS standard specifies the exact FORTRAN subroutine name for each PHIGS function which is called the "PHIGS FORTRAN 'binding'" . But graPHIGS uses a "generic language binding" which allows the PHIGS functions to be called from FORTRAN, C, and PASCAL. IBM has also added extensions to their PHIGS implementation such as circle and ellipse primitives [14]. If the graPHIGS binding or extensions were used in the ACSYNT/VPI code, the code would not be portable to non-IBM systems. "Appendix D. graPHIGS Information" on page 183 contains a list of non-standard graPHIGS features.

To permit the development of ACSYNT/VPI to occur on Virginia Tech's IBM 4341 system, a set of PHIGS to graPHIGS conversion routines were written to ensure ACSYNT/VPI's adherence to the PHIGS standard. The conversion routines follow the FORTRAN binding specified in the PHIGS standard but actually call graPHIGS routines to perform the required function. When installing the ACSYNT/VPI code on a non-IBM system, the conversion and graPHIGS routines will be replaced by the standard PHIGS routines for that system.

ACSYNT/VPI Coding Standards

Overview

Coding standards were applied to the ACSYNT/VPI project to promote code portability, programming consistency, and thorough documentation. The following sections discuss the language and commenting standards adopted for the ACSYNT/VPI project.

Language Standards

Language standards, like graphics standards, help ensure code portability between computer systems. For the ACSYNT/VPI project, the FORTRAN 77 standard was chosen as the programming language for the following reasons:

- FORTRAN 77's extensive computational capabilities
- the FORTRAN 77 standard is well supported on all ACSYNT/VPI target computer systems
- ACSYNT is written in FORTRAN and will be merged with the ACSYNT/VPI code
- the ACSYNT/VPI design team's capability to write and debug FORTRAN 77 code

The FORTRAN 77 standard was accepted by the American National Standards Institute in 1978 [16]. A functionally identical standard was accepted by the International Standards Organization in 1980 [17].

Many FORTRAN 77 implementations, such as DEC's VMS FORTRAN and IBM's VS FORTRAN, contain nonstandard extensions to the FORTRAN 77 standard [18]. For example, IBM's FORTRAN implementation allows the source code to be in upper or lower case but the standard specifies all upper case [19]. DEC's FORTRAN permits variable names to be longer then the standard length of six characters.

For the ACSYNT/VPI code, only one FORTRAN extension was used which simplifies the reading of data files and keyboard input. "Appendix E. FORTRAN 77 Extension Used in ACSYNT/VPI Code" on page 187 contains a discussion of the extension in light of the ACSYNT/VPI code.

Code Commenting Standards

OVERVIEW

The purpose of commenting source code is to make the code more understandable to those who must read it [12]. Code comments are not intended to replace but to supplement program specifications as described in "Software Development Methods" on page 9. This sections outlines the commenting standards applied to the ACSYNT/VPI code in terms of module header blocks and line comments.

MODULE HEADER BLOCKS

Header block comments provide a means of throughly documenting the code produced by a programmer. An example header block is listed below:

```
C=====================================================================
C                        M O D U L E   A S H O W
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0
C   PART #:  VT-A045
C MODULE #:  ASHOW
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL ASHOW(WKID)
C=====================================================================
C INPUT PARAMETERS:
C     WKID   - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     NONE
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     ALLOWS USER TO SHOW AND NO SHOW AIRCRAFT COMPONENTS
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     MSHOW - NO SHOW MENU DRIVER
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  11/30/87 (12:24:46)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================
```

As shown in the example, a header block contains the following information:

module name: clearly identifies module for quick reference

project title: identifies the software development project

version number: identifies the release level of the module

part number: used to keep track of source code files

module number: used to reference the module's flowchart

compiler name: identifies the compiler used during development of the code

call sequence: describes proper syntax for accessing the module

input parameters: describes the input parameters in terms of data type and purpose

output parameters: describes the output parameters in terms of data type and purpose

common inputs: describes the common block inputs in terms of data type and purpose

common outputs: describes the common block outputs in terms of data type and purpose

local variables: describes variables defined internal to the module in terms of data type and purpose

functional description: describes the function of the module

example(s): helps clarify the function of the module through an example

methods/algorithms: further defines the operation of the module

modules called: lists the modules called from the current module and their function

coded by: lists the programmer who created the code and the date the code was written

development site: specifies where the code was developed

Typically, header blocks document decisions made by the programmer which are not defined in the programming specifications, such as variable names, part numbers, date of creation, etc. Automatic generation of header blocks is discussed in "Appendix N. Software Development Tools User's Guide" on page 268.

LINE COMMENTS

In addition to header blocks, most of the ACSYNT/VPI source code lines were proceded by a descriptive comment statement. Although not specifically a standard, indentation and blank lines were also used to make the code more readable as shown in the following example:

```
      DO 100 I = 1, NLINES

C       GET ITH INPUT LINE
        CALL GTINLN(INLUN, INLINE)

C       PARSE INPUT LINE RETURNING VALUES IN "VALUES" ARRAY
        CALL PARSE(INLINE, VALUES, ERRIND)

C       CHECK ERROR INDICATOR RETURNED BY PARSER
        IF (ERRIND .GT. 0) THEN
C         WRITE ERROR MESSAGE
          CALL MSG(WKID, 'ERROR --> INVALID INPUT')
        ELSE
C         RESET INPUT STRING
          CALL RSTINP(INLINE)
        ENDIF

  100   CONTINUE
```

The commenting standard applied to the ACSYNT/VPI project stated that, on average, one out of every three lines of source code (33 percent comments) must be a comment line excluding blank lines. On average, the ACSYNT/VPI code actually has one comment line for every executable FORTRAN statement (50 percent comments).

6.0 SOFTWARE REQUIREMENTS

Overview

The purpose of software requirements is to clearly define the problem or problems a software product is intended to solve. Requirements provide a checklist of items which must be realized in order for the software to be considered complete. In addition, software requirements prevent stray development of nonrequired code and were therefore completed before the ACSYNT/VPI software was designed. This chapter presents, in detail, the requirements initially imposed on the ACSYNT/VPI software.

Management Information

This section presents the ACSYNT/VPI project requirements from a project management point of view and includes a discussion of the software production method, first year milestone and review schedule, and items to be delivered with a production version of the ACSYNT/VPI code.

Production Method

The ACSYNT/VPI software will be produced in house at Virginia Tech's Computer Aided Design Laboratory. With access to high performance computer, graphics, and software development facilities, Virginia Tech faculty and students have the resources necessary to develop interactive design software, such as ACSYNT/VPI, without the foreseeable need for outside assistance. The only software used in the ACSYNT/VPI code but not developed at Virginia Tech will be the PHIGS routines which perform graphical input and output. The PHIGS graphics standard is described in "Software Development Standards" on page 29.

Milestone and Review Schedule

Figure 14 on page 45 contains the original milestone and review schedule for the first year of the ACSYNT/VPI project. The figure shows how the different phases of the project overlapped and indicates when code and project reviews should take place. Additional schedules should be generated before the beginning of each project year.

Software Deliverables

A typical software release for ACSYNT/VPI should include the following items:

- release notice
- functional description of ACSYNT/VPI and enhancements list
- ACSYNT/VPI executable program module for the target computer system
- training agreement
- maintenance agreement

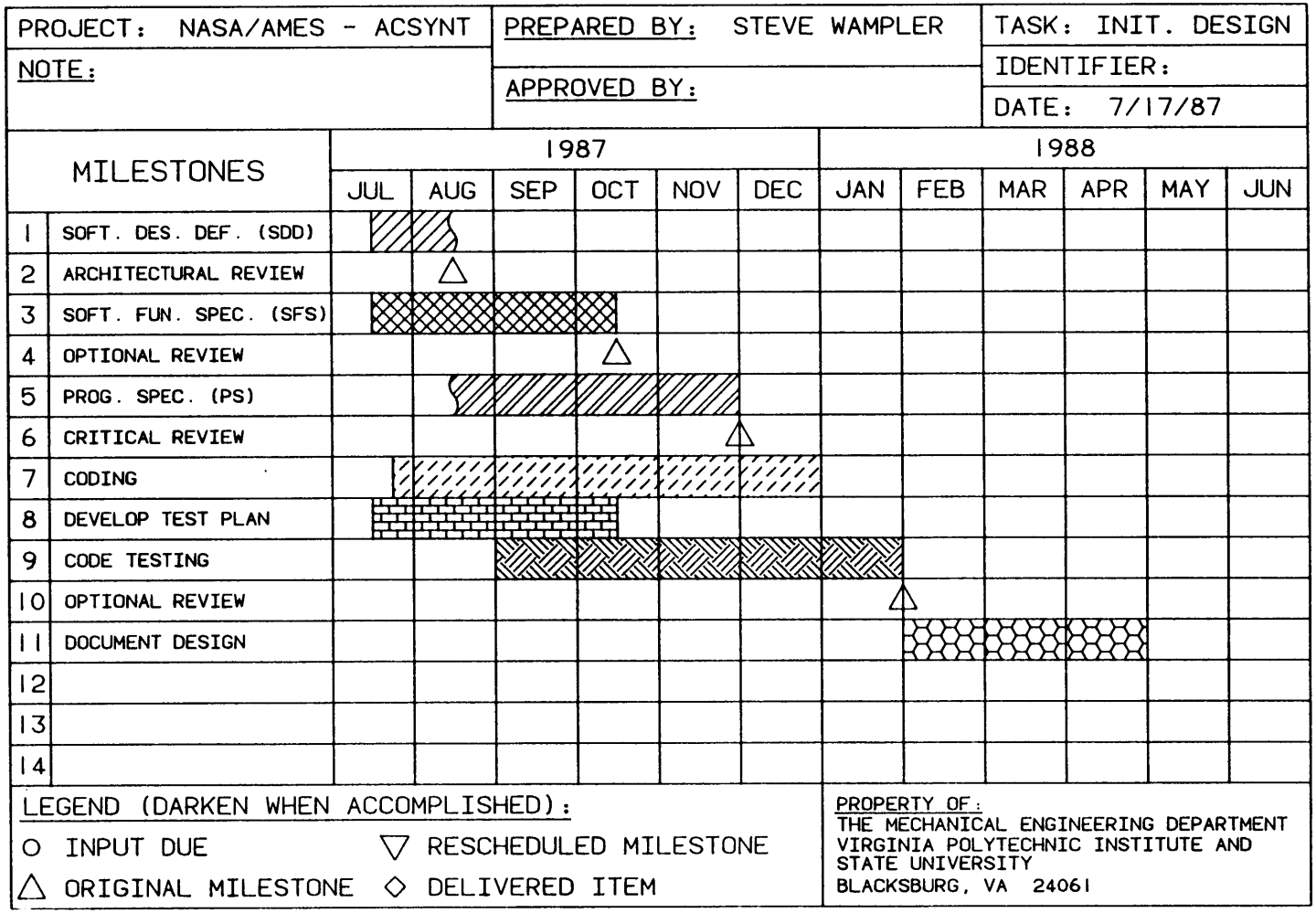


Figure 14. ACSYNT/VPI milestone and review schedule.

- installation manuals
- training manuals
- user's manuals
- problem report instructions and forms
- suggestion report forms
- information concerning the ACSYNT/VPI users' consortium

System Requirements

The following section contains a discussion of requirements imposed on any system targeted to run the ACSYNT/VPI software including host system requirements and graphics workstation requirements.

Host System Requirements

The current developmental version of ACSYNT/VPI runs on an IBM 4341 mainframe under IBM's VM/CMS operating system. Virginia Tech's IBM 4341 has 16 megabytes of real memory which can be shared by multiple users each having a "virtual machine" with access to a maximum of 12 megabytes of "virtual memory." In the above context, the terms "virtual machine" and "virtual memory" mean that each user appears to have access to his or her own 12 megabyte computer. Figure 15 on page 47 illustrates the current ACSYNT/VPI development environment.

Although ACSYNT/VPI is being developed in a mainframe environment, the code is targeted for a stand alone system capable of running the industry standard UNIX operating system. Figure 16 on page 49 illustrates the stand alone target system environment. The target system should also

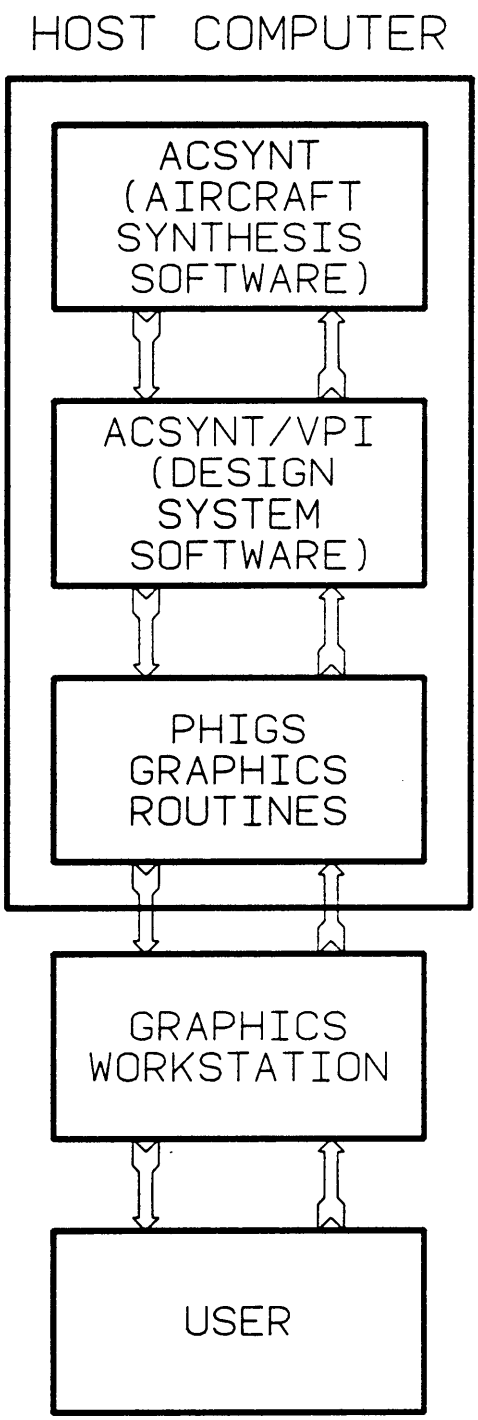


Figure 15. ACSYNT/VPI development environment.

be fast enough to allow ACSYNT calculations to be performed in real time, preventing the need for batch processing of aircraft models.

Graphics Workstation Requirements

Figure 17 on page 50 shows the minimal graphics workstation environment in which ACSYNT/VPI will operate. For the first project year, the graphics workstation must have, as a minimum, the capabilities of an IBM 5080 graphics workstation. The IBM 5080 has the following features [20].

- high screen resolution (1024 X 1024 pixels)
- 256 displayable colors out of a pallet of 4096
- local area fill
- local transformations
- local character generator
- maximum of 1.1 megabytes of display list memory
- alphanumeric keyboard
- lighted program function keyboard (LPFK)
- tablet
- dials
- graphics application support with IBM graPHIGS

During the second year of the ACSYNT/VPI project, a high performance graphics workstation will be used having all of the features of an IBM 5080 plus the following:

- z-buffer for hidden surface removal
- PHIGS and PHIGS+ support of:
  - real time 3D transformations
  - real time shading
  - real time surface intersections based on execution of PHIGS+ instructions (for 6000 or more polygons)
- 10-12 mips (million instructions per second) workstation speed

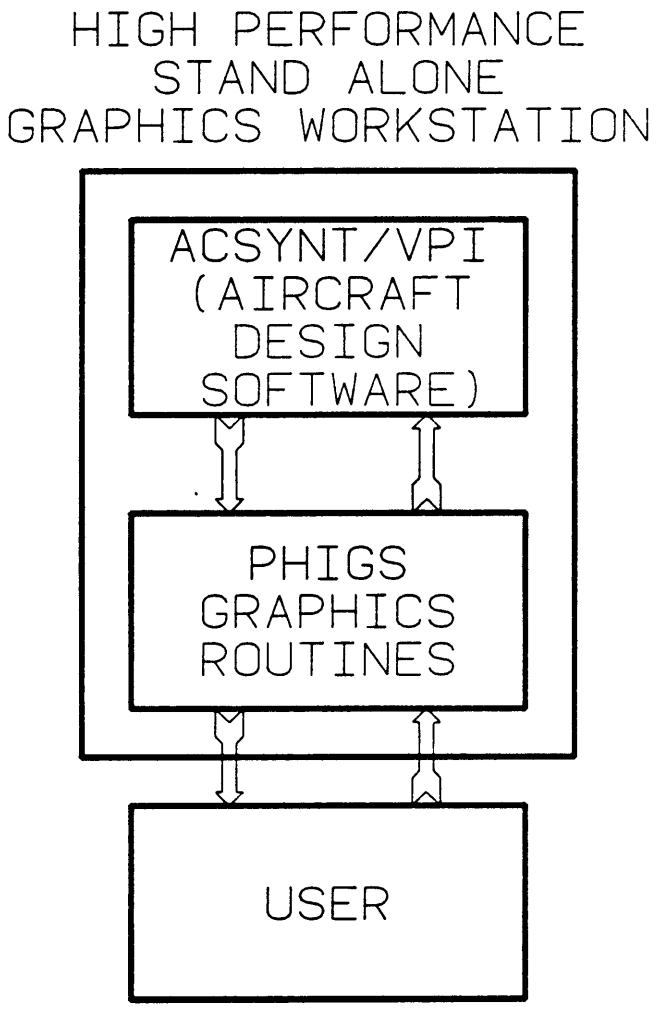


Figure 16. ACSYNT/VPI target system environment.

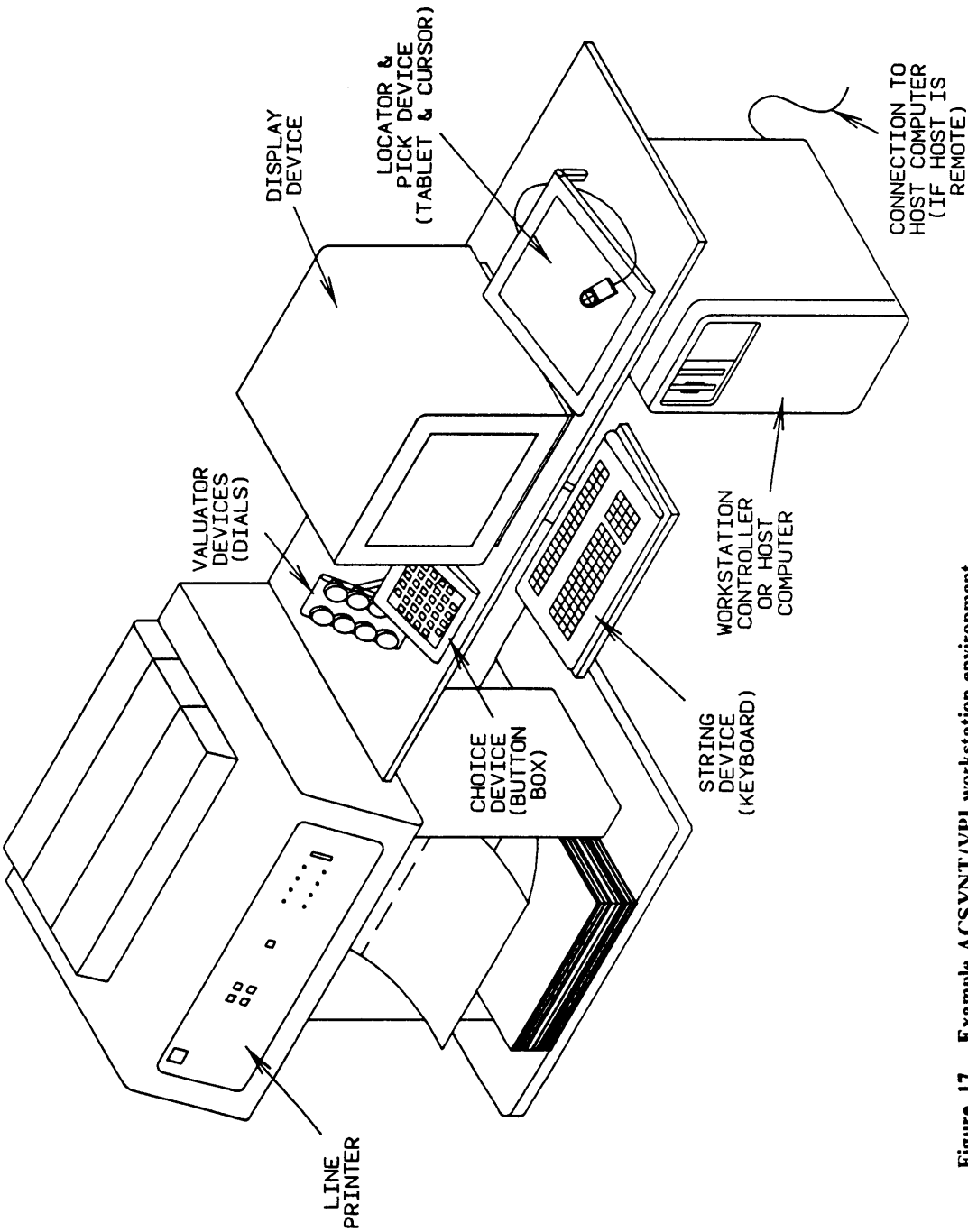


Figure 17. Example ACSYNT/VPI workstation environment.

- at least 4096 active colors out of a pallet of 16.7 million

Silicon Graphics 4D/70 and Raster Technology GX4000 are examples of workstations capable of the features listed above.

Project Policies

This section outlines the policies applied to the ACSYNT/VPI project concerning project development and documentation.

Development Policies

The top down method of software development will be used throughout the life of the ACSYNT/VPI project. References 10 and 11 contain details concerning the proper application of top down design and "Software Development Methods" on page 9 describes the use of top down design for the ACSYNT/VPI project.

Documentation Policies

Documentation can be of varying quality and detail. For the ACSYNT/VPI project, documentation will take the following two forms;

- in-house documentation used to record the development of the ACSYNT/VPI code
- external documentation used to report progress to NASA.

In-house documentation will range from notes recorded in research notebooks to high quality programming flowcharts used to document the design of ACSYNT/VPI.

Project Constraints

Overview

This section describes development and priority constraints imposed on the ACSYNT/VPI project.

Development Constraints

Due to the need for portable software, ACSYNT/VPI must be written using the FORTRAN 77 and PHIGS standards. Both standards are discussed in "Software Development Standards" on page 29. Following the FORTRAN 77 and PHIGS standards will help insure that the ACSYNT/VPI code will run, with few or no modifications, on any high performance graphics workstation.

Competing Design Characteristics

The following is a list of competing design characteristics ranked according to importance in meeting the requirements discussed in this chapter:

1. development time
2. development cost
3. portability and machine independence
4. expandability
5. maintainability
6. documentation readability
7. vulnerability to operator or input error
8. execution speed
9. training time
10. operation cost
11. program size

Functional Requirements

Overview

The remainder of this chapter contains the functional requirements which must be satisfied by ACSYNT/VPI software. The section includes:

- user requirements
- software functional requirements,
- input, processing, output requirements
- database requirements.

All of the requirements listed were developed to further clarify ACSYNT/VPI's purpose and to guide the ACSYNT/VPI design team during the design of the software.

User Requirements

The following list specifies the assumed needs of those who will use the ACSYNT/VPI software. This list does not address how the software will meet the user's needs, but only lists the needs to aid in the design of the software.

NOTE: Many of the items listed below will not be included in the first production version of the software, but are included for design planning purposes only.

The software should allow the user to perform the following:

- specify and respecify input data for ACSYNT interactively
- invoke/execute the ACSYNT software
- invoke/execute analysis software to compute:
  - cross sectional areas
  - wave drag cone areas
  - volumes
  - surface areas
  - other (user written)
- invoke/execute additional design software
  - kinematic synthesis
  - design aided by an expert system
  - aeroelasticity analysis
  - other (user written)
- view output from ACSYNT, analysis routines and other design software
  - interactive graphics output
    - ▲ geometric representation of aircraft and components
    - ▲ shaded images
    - ▲ plotted graphs
    - ▲ numerical results (on graphics screen)
  - hardcopy output
    - ▲ picture files

        - numerical results
        - plotted graphs
- access detailed online help information which is easy to use and read
    - in context
    - help index

Note that the user requirements stated above may change as the ACSYNT/VPI project develops.

Software Functional Requirements

The goal of the ACSYNT/VPI project is to produce a highly interactive conceptual design system based on ACSYNT. The code should be device-independent and written following the proposed PHIGS standard for 3-D graphics. Specifically, the program should, over the long term, be able to perform the functions listed below based on ACSYNT computations and/or user interaction.

NOTE: Many of the items listed below will not be included in the first production version of the software, but are included for design purposes only.

Software functional requirements for the first project year are in **bold print** in the list below:

- create models representing:
    - **fighters**
    - transports
    - missiles
    - bombers
- **display interactive menus for viewing geometrical output**
- **move components separately and interactively**
- **modify parametric components**
- compute:
    - **cross sectional areas**
    - mass properties
    - wetted areas

    - wave drag cone areas
- link interactively to commercial CAD/CAM systems.
- **produce output files for later retrieval**
- produce IGES output
- automatically and/or interactively define standard components:
    - **fuselage** including:
        - **canopy**
        - **nose**
        - **inlets**
        - **nozzle**
    - airfoils including:
        - **wings**
        - **wing tips**
        - **vertical stabilizers**
        - **horizontal stabilizers**
        - ailerons
        - flaps
        - **strakes**
        - **canards**
    - bays including:
        - crew compartment
        - electronics bay
        - fuel bay
        - cargo bay
        - engine bay
        - passenger compartment
    - miscellaneous including:
        - **engine nacelles**
        - **pylons**
        - landing gear
        - speed brakes
- automatic and/or user assisted interactive filleting

- animate moving parts
- design mechanisms for landing gear, flaps, and speed brakes
- **produce wire frame surface models**
- render models using:
  - **hidden surface elimination**
  - **constant shading**
  - **smooth shading**
  - **movable sun model**
  - cross sections
  - interactive color selection
  - shadows
  - **reflections (specularity)**
- invoke/execute ACSYNT design code and process ACSYNT input data using:
  - menu screens for numerical input data entry
  - default starting models for input data
  - graphical entry of input data
- invoke/execute expert system to aid in design
- provide display for aeroelasticity computations

Note that the requirements stated above may change as the ACSYNT/VPI project develops.

Input, Processing, and Output Requirements

INPUT REQUIREMENTS

All input to ACSYNT/VPI will be in the form of user supplied input at a graphics workstation or input data files read by ACSYNT/VPI at the user's request. User and file input are discussed in the following paragraphs.

User Input

User input will involve user interaction with ACSYNT/VPI through the logical input devices string, pick, locator, choice, and valuator. User interaction will consist of ACSYNT/VPI prompting the user for required input and then waiting for the user to respond. In some cases, the user should have the option of entering data in many different ways. For example, if a numeric value is required, the user should be able to enter the data from the keyboard or by dialing in the number using input dials. PHIGS subroutines, discussed in "Software Development Standards" on page 29, will be used to control all graphical input and output.

The user will be responsible for generating or acquiring any data needed by ACSYNT/VPI, but whenever possible ACSYNT/VPI should supply the user with appropriate default information. User's manuals, examples, and online help screens should also be available to provide any information the user may need in order to generate appropriate input. Also, the user should always have the option of exiting from an input mode without specifying any input.

File Input

An ACSYNT/VPI data file should include all of the data required to described the aircraft's geometry, mission profile, weight, aerodynamics, etc. ACSYNT/VPI data files should also include viewing information such as window locations and three dimensional view transformation values. In general, input files will be generated by ACSYNT/VPI whenever the user chooses to save the current design session. Also, a start-up data file should be generated whenever the user choses to exit ACSYNT/VPI. The next time the user runs ACSYNT/VPI, the start-up file should automatically be read to restore the previous modeling session. Start-up files could also be automatically written at a specified interval during a design session in case the session is abnormally terminated.

PROCESSING REQUIREMENTS

After the user enters data, ACSYNT/VPI should check to see if the input is valid and, if so, process the input as required. If the input is invalid, a descriptive error message should be generated and the input requested again. For example, if the user enters the name of an input file which does not exist, ACSYNT/VPI should issue an error message such as:

```
ERROR (GETFN) --> FILE <JUNK> DOES NOT EXIST.  PLEASE REENTER:
```

Error messages should be short, descriptive and written in nontechnical terms and perhaps suggest a means of correcting or preventing the error.

Output Requirements

As with input, all of ACSYNT/VPI's output should be handled by standard PHIGS subroutine calls. The only output not produced with PHIGS will be data file and nonuser generated error messages. File output will consist of ACSYNT/VPI data files discussed above under "Input Requirements". Nonuser generated errors are errors not directly caused by the user such as a database routine trying to process invalid data. Such error messages will be output using standard FORTRAN write statements.

Database Requirements

ACSYNT/VPI will be required to process and store a large amount of data to fully describe an aircraft model. For a typical model, the following data will be required:

- geometry data including:
  - number of aircraft components
  - component parameters
  - component locations and orientations

- component connectivity data
- component attributes including:
  - component label
  - existence flag
  - colors and line type data
  - type identifier
  - PHIGS structure identifiers
  - number of cross sections
  - surface approximation values
  - weight data
- cross section parameters
- surface data
- shading and hidden surface parameters
- shading and hidden surface work areas
- ACSYNT common block data

- aerodynamics data
- stability data
- trajectory data including
  - mission profile(s)
  - takeoff and landing data
- propulsion data
- COPES control data
- PHIGS view data including:
  - number of defined views
  - view attributes including
    - view identification label
    - view active flag
    - projection type
    - location of viewer
    - location of projection plane
    - location of clipping planes
    - clipping indicators

- view screen location
- view transformation values

Ideally, a database system should be written for the ACSYNT/VPI project to organize the above data. An example of such a system is "WORM", a data structuring system for FORTRAN described in reference 21.

7.0 SOFTWARE DESIGN

Overview

This chapter contains an overview of ACSYNT/VPI's design and should be read by those people who will continue the developing the ACSYNT/VPI code. The reader is assumed to have a functional knowledge of the FORTRAN 77 standard and PHIGS graphics standard as discussed in references 16 and 13 respectively.

The body of this chapter is divided into five sections; "Design Philosophies", "User Interface Design", "Menu Modules", "Utility Routines", and "Data-flow and Geometric Modeling". Each section presents ACSYNT/VPI's design in increasing detail, starting with an abstract discussion of design philosophies and ending with a detailed discussion of data-flow and geometric modeling which includes ACSYNT/VPI's rendering of the aircraft's image.

Design Philosophies

Overview

This section contains an abstract discussion of the philosophies applied to the design of the ACSYNT/VPI code. The discussion includes "finite state machines" and "menu based design" as applied to the ACSYNT/VPI project. The philosophies presented form the basis for the design discussed in the rest of the chapter.

Finite State Machines

OVERVIEW

A "finite state machine" is defined as an apparatus or process which performs a task under a limited number of possible operating conditions [2]. An example of a finite state machine is provided by the PHIGS graphics standard discussed below.

PHIGS AS A FINITE STATE MACHINE

The PHIGS graphics standard described in "Software Development Standards" on page 29 and in references 13 and 14, was designed to be a finite state machine. PHIGS manages four state variables each having two possible values; either "open" or "closed". All PHIGS states are by default "closed" before PHIGS is invoked, and many PHIGS functions can only be accessed when PHIGS is in the proper state. The four state variables are defined below [22]:

PHIGS system state:
defines whether or not PHIGS has been activated or deactivated using the "OPEN PHIGS" or "CLOSE PHIGS" functions respectively. No other PHIGS functions can be accessed until PHIGS is activated.

workstation state:
defines whether or not a workstation has been activated or deactivated using the "OPEN WORKSTATION" or "CLOSE WORKSTATION" functions respectively. PHIGS structure display operations can only be used if a workstation is active.

structure state:
defines whether or not a PHIGS display structure is open and able to be modified, or closed and unavailable for modification. A structure is opened and closed with the "OPEN STRUCTURE" and "CLOSE STRUCTURE" functions respectively. PHIGS graphics primitives and attributes can only be created if the structure state is "open".

archive state:
defines whether or not a structure archive file has been opened or closed using the "OPEN ARCHIVE FILE" or "CLOSE ARCHIVE FILE" functions respectively.

PHIGS can only operate under the following set of states:

```
{PHCL, WSCL, STCL, ARCL}
{PHOP, WSCL, STCL, ARCL}
{PHOP, WSOP, STCL, ARCL}
{PHOP, WSOP, STOP, ARCL}
{PHOP, WSOP, STOP, AROP}
{PHOP, WSOP, STCL, AROP}
{PHOP, WSCL, STOP, ARCL}
{PHOP, WSCL, STOP, AROP}
{PHOP, WSCL, STCL, AROP}
```

where:

PHOP = PHIGS system state open
PHCL = PHIGS system state closed
WSOP = workstation state open
WSCL = workstation state closed
STOP = structure state open
STCL = structure state closed
AROP = archive state open
ARCL = archive state closed

Figure 18 on page 65 contains the finite state diagram for PHIGS showing the functions which cause PHIGS to change states. The archive state has been excluded from the figure for clarity.

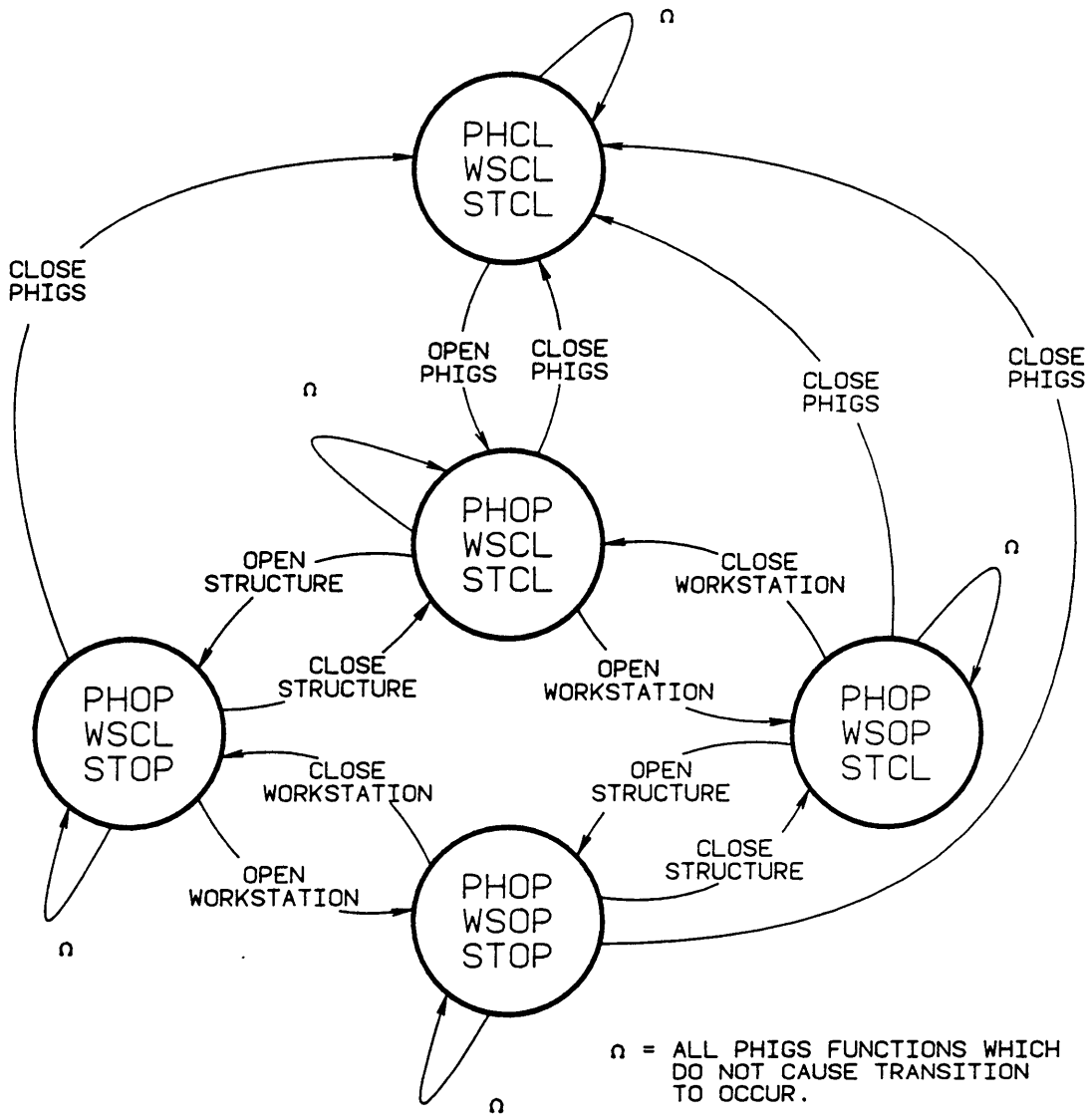


Figure 18. Finite state diagram for PHIGS [22] .

ACSYNT/VPI AS A FINITE STATE MACHINE

Like PHIGS, the ACSYNT/VPI software was designed using the concept of a finite state machine. For example, ACSYNT/VPI menus are displayed with an "OPEN MENU" routine, and removed from the screen with a "CLOSE MENU" routine. "Open" and "close" routines are also used to control the display of aircraft geometry, shaded images, and hidden surface images. In fact, most ACSYNT/VPI menu driven functions were implemented as "open" and "close" routines. Typically, "open" routines acquire use of resources such as screen space and input devices and "close" routines release resources for use by another portion of the program. For example, when user supplied numeric input is required, an "OPEN NUMBER INPUT" routine is called to prepare ACSYNT/VPI to accept input from the keyboard. When numeric input is no longer needed, the "CLOSE NUMBER INPUT" routine is called to free the keyboard device for use by other routines. The following is a list of ACSYNT/VPI "open" and "close" functions:

OPEN MAIN:
performs initialization for ACSYNT/VPI

CLOSE MAIN:
shuts down ACSYNT/VPI

OPEN MENU:
draws a menu on the screen

CLOSE MENU:
removes a menu from the screen

OPEN NUMBER INPUT:
prepares for user supplied numeric input

CLOSE NUMBER INPUT:
removes numeric input echo from the screen

OPEN GEOMETRY:
displays the current ACSYNT aircraft geometry on the screen

CLOSE GEOMETRY:
removes the current ACSYNT aircraft geometry from the screen

OPEN SHADING:
prepares for shading the aircraft geometry

CLOSE SHADING:
removes shaded images from the screen

OPEN HIDDEN SURFACE:
prepares for hidden surface elimination

CLOSE HIDDEN SURFACE:
removes a hidden surface image from the screen

Menu Based Design

Menu based design refers to the ACSYNT/VPI philosophy that all pickable items on the screen, including the aircraft geometry and views, are treated as menus. For the current version of ACSYNT/VPI, the following four menu types have been defined:

command: a text menu which changes as the user moves up and down the menu structure

standard: a text menu which does not change and contains commonly used functions such as "HELP", "HARDCOPY", etc.

view: consists of the borders and labels associated with each PHIGS viewport and is used to specify a view for functions such as windowing or shading

geometry: consists of the individual aircraft components such as a wing, fuselage, canopy, etc. and is used to specify a particular aircraft component when needed by the program.

For example, when the user wishes to shade an image of the aircraft model, he or she can pick an item from any one of the four menu types defined above. If the user selects an aircraft component, the component will be highlighted and added to a list of components to be shaded. If a view is then picked, the aircraft components in the shading list will be shaded in the selected view. At any time the user can select an item from the command or standard menus to perform any available function including exiting the shading menu by selecting the "RETURN" menu item.

The use of menu based design has proven to greatly simplify the overall design of the ACSYNT/VPI code by providing a unified means of acquiring user input. A more detailed discussion of ACSYNT/VPI menus is contained in "Menu Modules" on page 76.

User Interface Design

Overview

Mink [24] states that "graphics programs require careful attention to what the screen will look like at every point in the program both in terms of human factors and interface design." This section presents the design of ACSYNT/VPI's user interface including the screen layout, menu structure, and interactive input methods.

Screen Layout

The ACSYNT/VPI design team originally planned to use pop-up windows as the primary means of displaying data on the ACSYNT/VPI screen but later decided on the more conventional screen layout illustrated in Figure 19 on page 69. The layout shown was chosen for the following reasons:

- met all requirement specified in "Software Requirements" on page 43
- would not require the time consuming development of window management software
- would not slow down ACSYNT/VPI's response time due to the window management and PHIGS overhead associated with pop-up windows
- easy to design and implement
- conservative, not likely to cause problems in the future
- simple and uncluttered

The ACSYNT/VPI screen was designed using the guidelines listed below [24]:

- allocate screen space and use color to lead user's attention to what is most important.
- keep menu layouts simple, uncluttered and compact
- minimize hand and eye movement

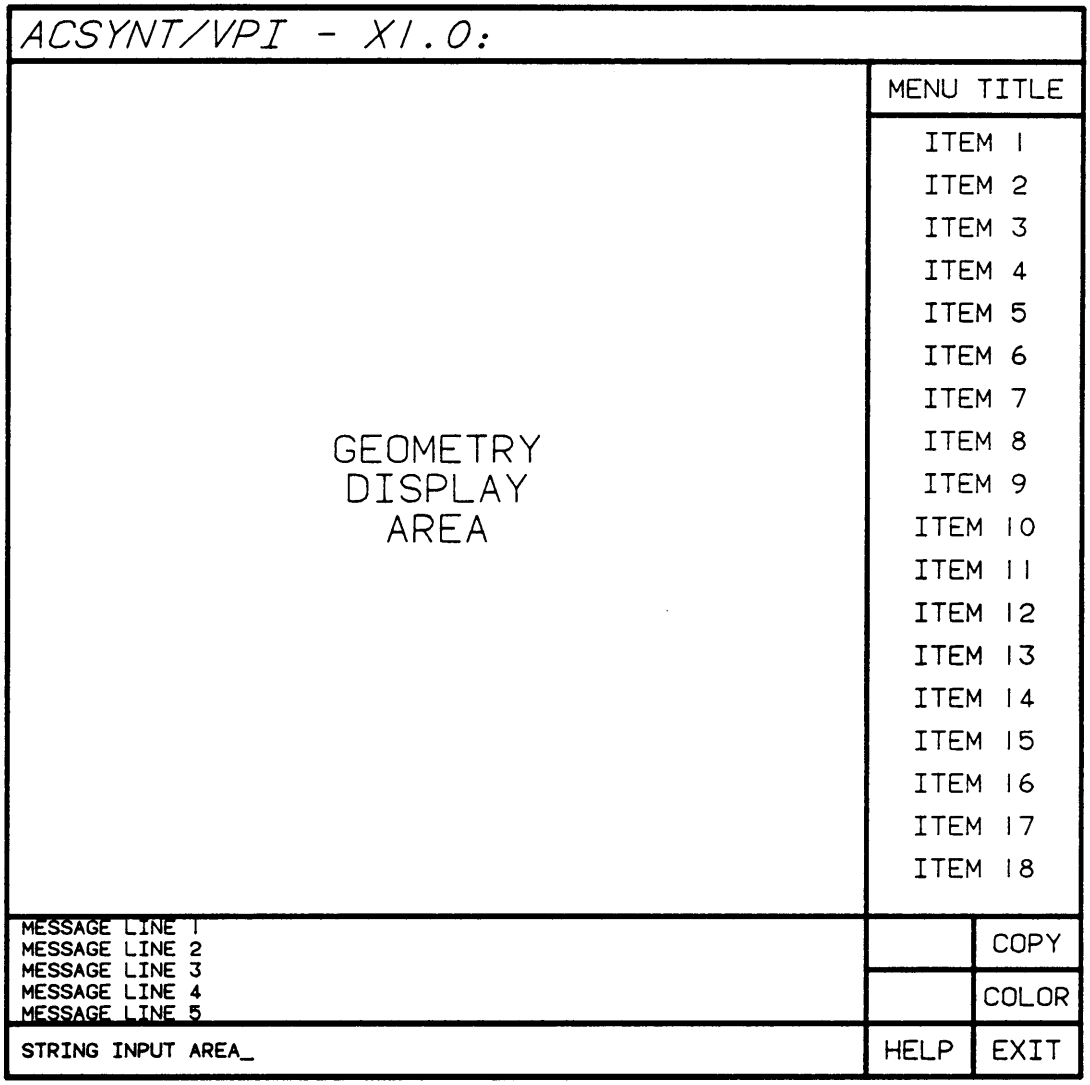


Figure 19. ACSYNT/VPI screen layout.

- be sure all text is readable
- use blues, grays, and greens for background colors
- use bright colors sparingly
- use red to indicate alarm, yellow to indicate caution
- use color changes to attract attention
- make use of contrast.

Command Menu Structure

ACSYNT/VPI's command menu structure is based on the design and analysis functions available in ACSYNT such as "geometry", "trajectory", "propulsion", etc. "Appendix I. ACSYNT/VPI Menu Tier Charts" on page 217 contains the tier charts showing the hierarchy of ACSYNT/VPI's menu structure. ACSYNT/VPI's menu structure was designed using the following guidelines [24]:

- menu items should be easy to find and logically located in the menu structure
- menu items should correspond to functions performed by the code
- menus should be shallow and broad, meaning that menu items should not be buried deep within the menu structure

Interactive Input Methods

This portion of the chapter contains a general discussion of the interactive input methods developed for the ACSYNT/VPI project. Specific routines designed to implement the input methods are described in "Utility Routines" on page 79.

All of ACSYNT/VPI's interactive user input is accomplished through pick, valuator, and string input devices. All devices operate in event mode and are initialized at the beginning of the program. Below is a discussion of how each input device is used within ACSYNT/VPI.

PICK DEVICE

The pick device is used to select menu items from any one of the four ACSYNT/VPI menu types. Each menu consists of at least two PHIGS structures, one root structure and one sub-structure. The root structure contains a PHIGS pick identifier which indicates the menu's type. The sub-structure contains a set of pick identifiers, one for each menu item. After triggering a pick device event, the PHIGS "pick path" will contain pick identifiers for both the menu type and menu item of the selected menu. Event mode input and PHIGS pick paths are discussed in detail in reference 14.

STRING DEVICE

A PHIGS string device allows character string data to be input from a typewriter-style keyboard. The string device echoes each keystroke on the workstation display in a specified echo area. For ACSYNT/VPI, the echo area is at the bottom of the screen as shown in Figure 19 on page 69.

When initializing the string device, a default string can be provided which the user may either type over or accept by pressing the enter (or return) key. The length of the default string, specified in the string's FORTRAN "CHARACTER" definition, defines the number of characters the user can enter. The following is an example of the ACSYNT/VPI string input area with an initial string of "ACSYNT".

ACSYNT

Note that the cursor is initially under the first character in the initial string. Below is a discussion of the specific use of string input for specifying ACSYNT/VPI filenames and numeric input.

File Name Input Using the String Device

File identifiers for the current IBM version of ACSYNT/VPI take the following form:

```
fn ft fm
```

where:

fn is an IBM CMS filename which must be "FILE" and is therefore not specified by the ACSYNT/VPI user.

ft is an IBM CMS filetype which must be seven characters or less when the file is used as a FORTRAN data file.

fm is an IBM CMS filemode letter which must be "A" indicating that the file resides in the user's own file directory.

As an example, a typical ACSYNT/VPI data file might have a CMS file identifier of "FILE GEOM A". An ACSYNT/VPI user would only specify the filetype of "GEOM" because the filename and filemode must be "FILE" and "A" as stated above.

NOTE: Non-IBM versions of ACSYNT/VPI may have to be modified to allow input of file identifiers in the formats other than IBM.

Numeric Input Using the String Device

Below is an example of ACSYNT/VPI's echo area when used for numeric input:

```
_       1,    1.00,    0.94,    0.00
```

Depending upon the data required, up to eight real or integer values can be entered by the user, separated by blanks or commas. When the user presses the enter (or return) key the string input device returns the character string shown in the string echo area. ACSYNT/VPI's numeric input routines check the input line to determine if the line is valid. A valid input line contains only the characters "0123456789. ,+-" and does not contain multiple decimal points or minus signs which

are next to each other. For example, the following input lines are invalid due to the dollar sign ("$") and double minus signs ("--") respectively:

```
1, 0.0, $, 4.5
2, --0.5, 0.0, -.45
```

Values are transferred to and from a numeric input string using FORTRAN internal writes and reads.

VALUATOR DEVICE

Valuator devices provide a means of entering scalar values and are typically implemented as rotating dials. A valuator's value can be limited by specifying the valuator's maximum and minimum values when initializing the device. For example, a dial used as a valuator to enter a rotation angle might be limited to a range of -360.0 to +360.0 degrees. ACSYNT/VPI's design assumes that at least eight valuator devices are available for use at any one time.

Many portions of ACSYNT/VPI allow the use of valuator input. For example, valuators are used for view transformations and for entering screen color values and shading parameters. Each routine using valuator input would normally be required to reinitialize the valuator devices with different maximum and minimum values. But, for the ACSYNT/VPI project, a method was developed to prevent the need for reinitialization.

The method involves the use of normalized valuator input. Each valuator is initialized at the beginning of the program to have a range of 0.0 to 1.0 and an initial value of 0.5. When the user turns a valuator dial the program calculates a "non-normalized" value x using the following formula:

$$x_{new} = x_{old} + (v_{new} - v_{old}) \times (x_{\max} - x_{\min}) \tag{7.1}$$

where:

x_{new} is the new non-normalized value of x

x_{old} is the previous or initial value of x

v_{new} is the new normalized valuator value returned by the "GET VALUATOR" PHIGS function

v_{old} is the previous or initial normalized valuator value

x_{max} is the maximum non-normalized value of x

x_{max} is the minimum non-normalized value of x

After the above calculation is performed, v_{old} is set equal to v_{new}. A FORTRAN array is used to store the v_{old} values for each valuator. Also, each portion of the ACSYNT/VPI code which uses valuator input is responsible for maintaining an array of non-normalized x_{new} values and their allowable x_{max} and x_{min} limits. By forcing each routine to maintain a local array of non-normalized values, the valuator devices become independent of the applications for which they are used. Application independence results in the ability to transfer the use of the valuators from one routine to another and back again without having to reinitialize the valuators each time.

Another advantage of normalized valuator input is that the valuator's non-normalized value can be reinitialized to 0.5 at any time without effecting the function of any input routine. For example, each time a valuator's value changes, ACSYNT/VPI checks to see if the value has reached a limit of 0.0 or 1.0. If so, the valuator is automatically reinitialized to a value of 0.5. Due to Equation 7.1 on page 73, the effect is that the non-normalized values are not limited by the range of the valuator. Rotating an object about an axis is one case where unlimited valuator input is convenient.

Below is a discussion of the specific ACSYNT/VPI use of valuators for view transformations and parameter input.

View Transformation Input Using Valuators

Seven valuator devices are used to enter the following view transformation values:

1. view rotation about the x-axis (α)
2. view rotation about the y-axis (β)

3. view rotation about the z-axis (γ)
4. view translation along the x-axis (t_x)
5. view translation along the y-axis (t_y)
6. view translation along the z-axis (t_z)
7. view scale (s)

Each value is stored in an array which is used to calculate the following view transformation matrix:

$$[\mathbf{M}] = \begin{bmatrix} k_{11} & k_{12} & k_{13} & t_x \\ k_{21} & k_{22} & k_{23} & t_y \\ k_{31} & k_{32} & k_{33} & t_z \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \times s \tag{7.2}$$

where:

$[\mathbf{M}]$ = view matrix

$k_{11} = cos(\alpha) \times \cos(\beta)$

$k_{12} = cos(\alpha) \times \sin(\beta) \times \sin(\gamma) - \sin(\alpha) \times \cos(\gamma)$

$k_{13} = cos(\alpha) \times \sin(\beta) \times \cos(\gamma) + \sin(\alpha) \times \sin(\gamma)$

$k_{11} = sin(\alpha) \times \cos(\beta)$

$k_{12} = sin(\alpha) \times \sin(\beta) \times \sin(\gamma) + \cos(\alpha) \times \cos(\gamma)$

$k_{12} = sin(\alpha) \times \sin(\beta) \times \cos(\gamma) - \cos(\alpha) \times \sin(\gamma)$

$k_{31} = -\sin(\beta)$

$k_{32} = cos(\beta) \times \sin(\gamma)$

$k_{33} = cos(\beta) \times \cos(\gamma)$

t_i = the translation along the axis i

The matrix is applied to the current three-dimensional PHIGS viewport causing the geometry appearing on the screen to rotate, translate, or scale depending upon which of the seven valuators were turned.

Numeric input using valuators is directly related to the numeric input using the string device discussed previously. The string device echoes the current non-normalized values of each valuator device. When a valuator changes value, the string device is reinitialized to echo the current non-normalized value. For example, suppose the user is prompted to enter an integer number between 1 and 5 where the default value is 2. After initializing the string device, the string input area will contain the following:

To change the value the user has the option of turning valuator dial 1 or entering a new value. Either way, ACSYNT/VPI will update the string echo to reflect the new input value.

A functional description of the ACSYNT/VPI modules designed to implement the input methods discussed above can be found in the "Utility Routines" section of this chapter.

Menu Modules

Overview

The ACSYNT/VPI command menu structure was implemented as a set of chained "menu modules" each representing one item in the menu structure. Menu modules either perform the function associated with a particular menu item, display a submenu, or both. This section discusses the hierarchical relationship between menu modules and presents the typical menu module design.

Menu Module Hierarchy

The following page contains a list of all the menu modules which are currently part of ACSYNT/VPI. The list shows the hierarchical nesting of the modules. Note that each module name starts with the letter "A" indicating the module is an ACSYNT/VPI menu module. The "A" is followed by the module's four letter identification label.

Some of the menu module routines such as ASACC do not actually display a menu but perform a function whenever their corresponding menu item is selected. Also, many of the menu modules listed above are "dummy stubs" which mean they are currently undefined and do not perform any function.

Typical Menu Module Design

Most ACSYNT/VPI menu modules consist of up to eight subroutines which are identical in function from one module to another. The name of each subroutine starts with a one or two letter prefix and ends with a four letter menu identifier. The prefix indicates the function of the routine and the four letter identifier indicates which menu module contains the routine. Below is an example list of the eight menu routines which drive ACSYNT/VPI's main menu. A functional description and flowchart figure reference follows the name of each routine.

```
AMAIN - main menu module for ACSYNT/VPI which calls:
  AGEOM - aircraft geometry menu module which calls:
    AACTP - dummy stub for aircraft type
    AFUSE - dummy stub for defining the fuselage
    AAIRF - dummy stub for defining the airfoils
    ASTOR - dummy stub for wing stores
    AEPOD - dummy stub for engine pods
    AMDFY - geometry modification menu module which calls:
      ACPRM - dummy stub for modifying aircraft component parameters
      AXPRM - dummy stub for modifying aircraft cross-section parameters
      AFLLT - dummy stub for creating and modifying fillets
      ACPOS - dummy stub for modifying aircraft component positions
      ACROT - dummy stub for modifying aircraft component rotations
      ACERS - dummy stub for erasing aircraft components
      ACPCR - component color menu module which calls:
        ASACC - sets all component colors (no menu displayed)
        ARSCC - resets component color input values (no menu displayed)
      ARETL - component surface approximation (tiling) menu which calls:
        ARTAL - retiles all aircraft components (no menu displayed)
        ARSRT - resets retiling input value (no menu displayed)
    ADISP - geometry display menu module which calls:
      ARDRW - dummy stub for redrawing the aircraft geometry
      AADRW - dummy stub for setting an autodraw flag
      AMDRA - dummy stub for resetting an autodraw flag
      ASHOW - geometry show/no-show menu module which calls:
        ASHFP - swaps the component show and no show screens (no menu)
        ASHAL - shows all aircraft components (no menu displayed)
      APICK - dummy stub for turning on and off component pickability
    AHSRF - hidden surface menu module
    ASHAD - shading menu module which calls:
      AAMBL - shading ambient light menu module which calls:
        ARAMB - resets ambient light input values (no menu)
      ASRFC - shading surface characteristics menu module which calls:
        ARSSC - reset surface characteristics input values (no menu)
      ASUNF - switches on and off the shading sun model flag
      AEYEF - switches on and off the light source eye flag
      ANLTS - number of shading lights menu module which calls:
        ARSNL - resets the number of light sources input (no menu)
      ALTSR - light source parameters menu module which calls:
        ARSLS - resets light source input values (no menu)
      ALTCR - light source color menu module which calls:
        ARSLC - resets light source color input values (no menu)
      ASHRS - resets shading parameters to default (no menu)
    AWIND - window menu module which calls:
      AWMLT - switches display to multiple windows (no menu)
      AWSNG - switches display to single window
      AWSTR - stores a window's settings
      AWRES - restores a window's settings
      AWPRJ - switches a window's projection type (parallel/perspective)
      AWISO - changes a window to an isometric view
      AWTOP - changes a window to a top view
      AWFRT - changes a window to a front view
      AWSID - changes a window to a side view
      AWROT - changes a window's rotation values
      AWTRN - dummy stub used to change a window's translation values
      AWSCL - changes a window's scale
      AWBOX - dummy stub used to zoom a window
  ATRAJ - dummy stub for ACSYNT trajectory information
  AAERO - dummy stub for ACSYNT aerodynamics information
  APROP - dummy stub for ACSYNT propulsion information
  ASTAB - dummy stub for ACSYNT stability information
  AWGHT - dummy stub for ACSYNT weights information
  ATOL  - dummy stub for ACSYNT takeoff and landing information
  ADA   - dummy stub for accessing design and analysis code
  AFILE - dummy stub for reading and writing ACSYNT/VPI data files
```

Routine	*Description*
AMAIN	main menu module for ACSYNT/VPI (see Figure 20 on page 80)
OPMAIN	opens PHIGS and workstation, initializes workstation and draws screen (see Figure 21 on page 81)
MMAIN	main menu driver (see Figure 22 on page 82)
EXMAIN	executes main menu input loop (see Figure 23 on page 83)
INMAIN	gets and processes main menu input (see Figure 24 on page 84)
PMAIN	processes main menu input (see Figure 25 on page 85)
PIMAIN	processes a picked main menu item (see Figure 26 on page 86)
CLMAIN	closes workstation and PHIGS, writes setup file (see Figure 27 on page 87)

Of the eight routines listed above, only the OPMAIN and CLMAIN routines differ greatly from one menu module to another. Also, note that each of the above subroutine names end in the four letter identifier "MAIN" indicating that they all belong to the main menu module. A complete list and functional description of all of ACSYNT/VPI's subroutines is contained in "Appendix J. Functional Descriptions for ACSYNT/VPI Subroutines" on page 225.

Utility Routines

Overview

Utility routines are defined in ACSYNT/VPI as any routine which can be called from more than one point in the code. Typically, utility routines perform general purpose functions such as normalizing a vector or capitalizing a string of characters but can also perform complex tasks such as drawing the aircraft's geometry or displaying a new menu. The ACSYNT/VPI utility routines are categorized as follows:

VT	PROGRAM SPECIFICATION - ACSYNT/VPI
MODULE NAME: AMAIN	MODULE: AMAIN
DESIGNED BY: STEVE WAMPLER	NOTE: MAIN MENU MODULE FOR ACSYNT/VPI
DATE: 3/23/88	

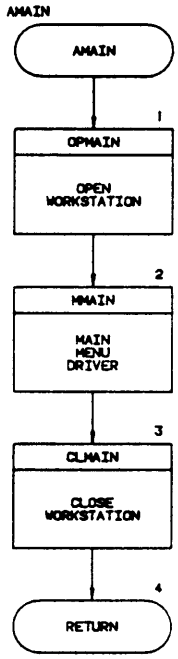


Figure 20. Example menu routine flowchart.

VT	PROGRAM SPECIFICATION - ACSYNT/VPI
MODULE NAME: OPMAIN	MODULE: AMAIN.1
DESIGNED BY: STEVE WAMPLER	NOTE: OPENS ACSYNT
DATE: 4/5/88	

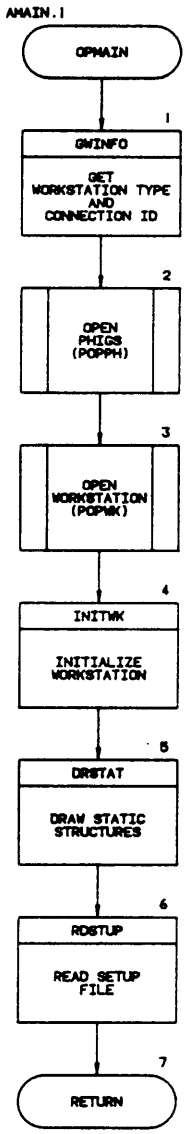


Figure 21. Example open function flowchart.

VT	PROGRAM SPECIFICATION - ACSYNT/VPI
MODULE NAME: MMAIN	MODULE: AMAIN.2
DESIGNED BY: STEVE WAMPLER	NOTE: MAIN MENU MODULE FOR ACSYNT/VPI
DATE: 3/23/88	

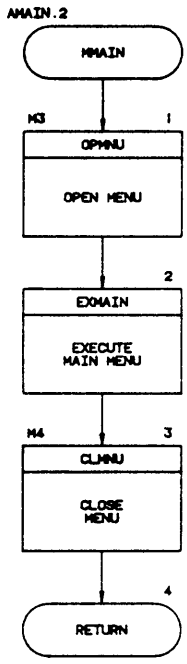


Figure 22. Example menu driver flowchart.

VT	PROGRAM SPECIFICATION - ACSYNT/VPI
MODULE NAME: EXMAIN	MODULE: AMAIN.2.2
DESIGNED BY: STEVE WAMPLER	NOTE: EXECUTES MAIN MENU
DATE: 4/6/88	

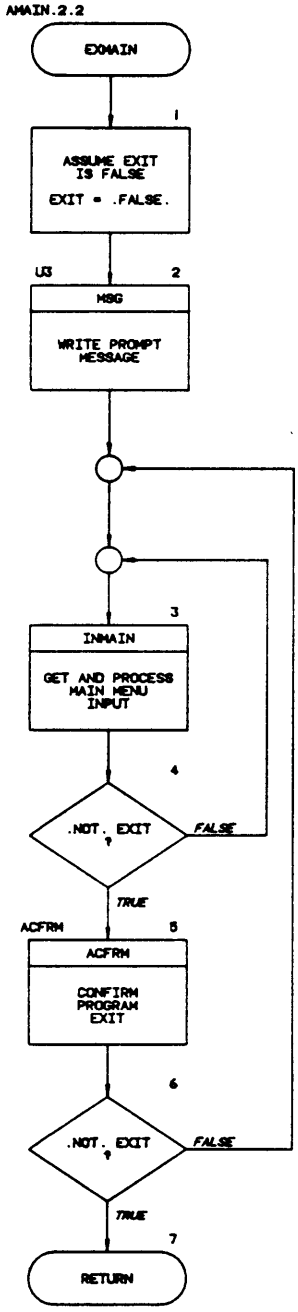


Figure 23. Example execute menu flowchart.

VT	PROGRAM SPECIFICATION - ACSYNT/VPI
MODULE NAME: INMAIN	MODULE: AMAIN.2.2.3
DESIGNED BY: STEVE WAMPLER	NOTE: GETS AND PROCESSES MAIN MENU INPUT
DATE: 4/5/88	

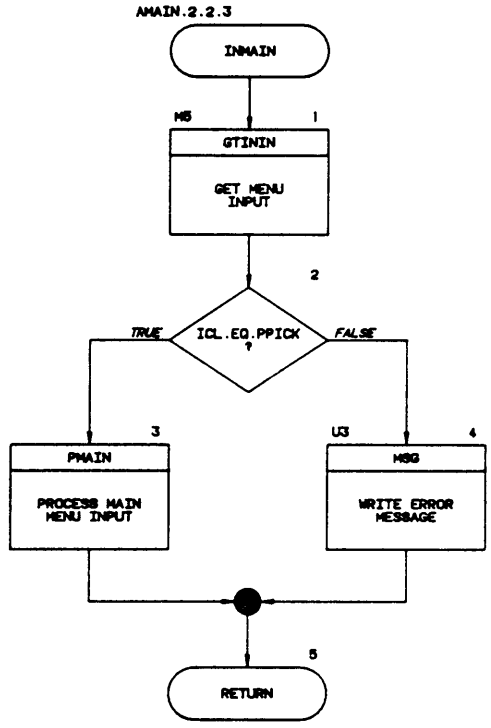


Figure 24. Example menu input flowchart.

PROGRAM SPECIFICATION - ACSYNT/VPI	
MODULE NAME: PMAIN	MODULE: AMAIN.2.2.3.3
DESIGNED BY: STEVE WAMPLER	NOTE: PROCESSES MAIN MENU INPUT
DATE: 3/24/88	

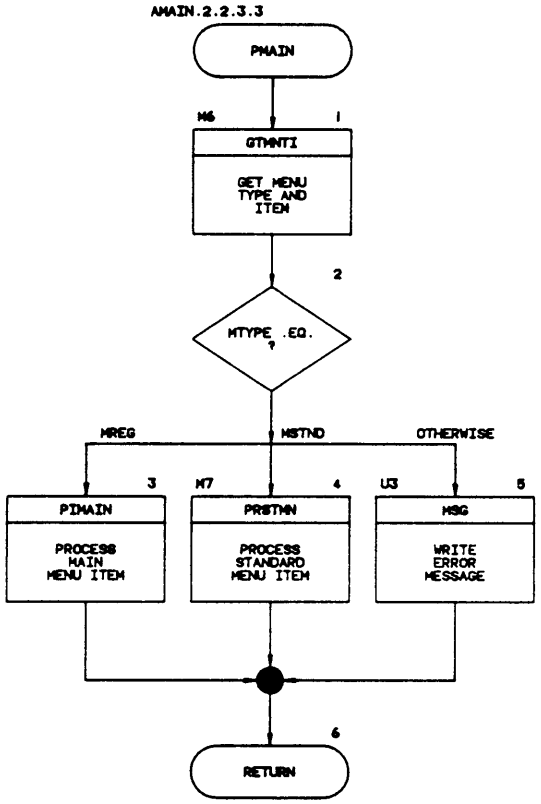


Figure 25. Example process menu input flowchart.

PROGRAM SPECIFICATION - ACSYNT/VPI	
MODULE NAME: PIMAIN	MODULE: AMAIN.2.2.3.3.3
DESIGNED BY: STEVE WAMPLER	NOTE: PROCESSES A MAIN MENU ITEM PICK
DATE: 4/5/88	

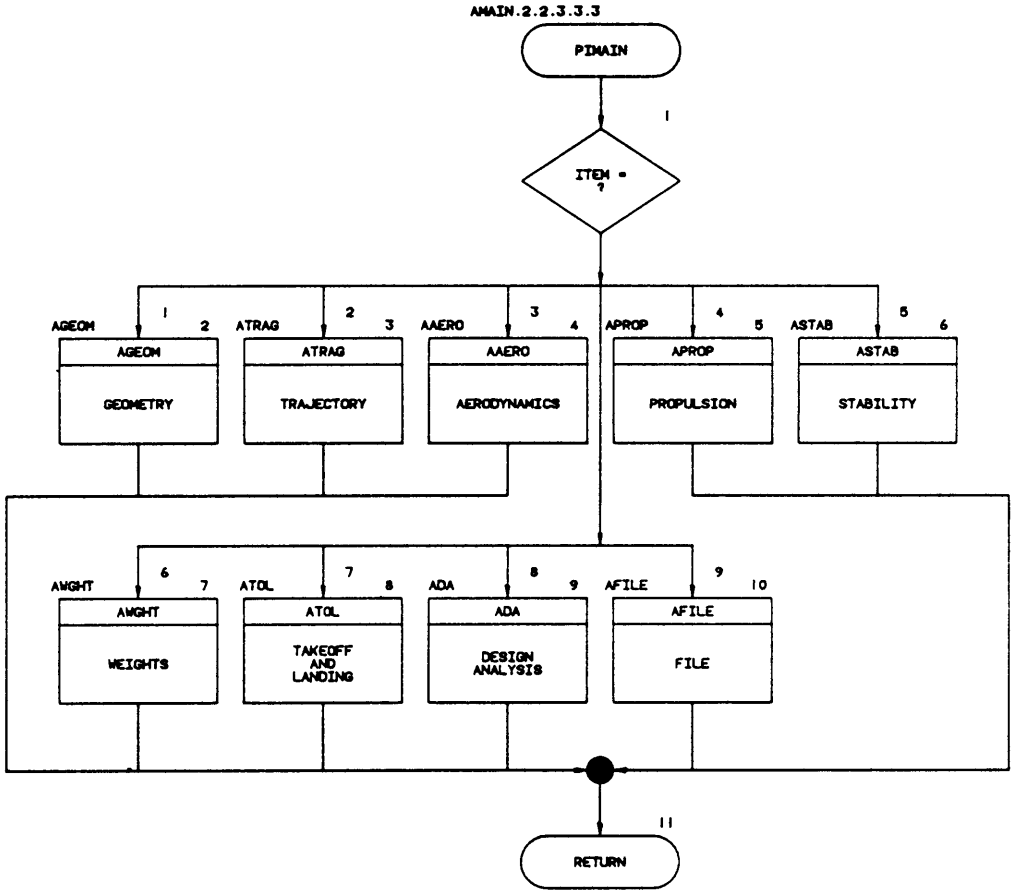


Figure 26. Example process menu item flowchart.

VT	PROGRAM SPECIFICATION - ACSYNT/VPI
MODULE NAME: CLMAIN	MODULE: AMAIN.3
DESIGNED BY: STEVE WAMPLER	NOTE: CLOSES ACSYNT
DATE: 4/5/88	

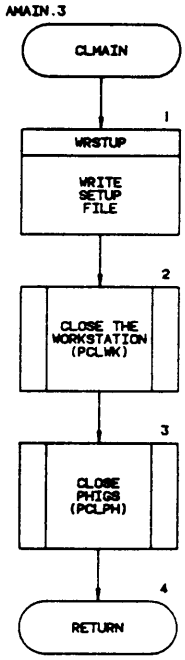


Figure 27. Example close function flowchart.

Geometry Database Routines:
store and retrieve aircraft geometry data

Geometry View Database Routines:
store and retrieve aircraft geometry view data

Shading Database Routines:
store and retrieve shading parameter data

Geometry Routines:
generate and manipulate a PHIGS image of the aircraft geometry

Shading and Hidden Surface Routines:
produce shading and hidden surface images

Menu Routines:
create and control menus

List Manipulating Routines:
process lists of integer numbers

Vector and Matrix Routines:
perform vector and matrix type calculations

Miscellaneous Utility Routines:
general purpose functions such as message generators, input handlers, etc.

The following pages contain functional descriptions of ACSYNT/VPI's most commonly used utility routines excluding the geometry routines which will be discussed in detail in "Data-flow and Geometric Modeling" on page 103. A complete list of utility routines including a short functional description can be found in "Appendix K. List of ACSYNT/VPI Utility Routines" on page 251.

Database Utility Routines

Database routines help to protect data from being destroyed by controlling how and if data is stored and retrieved. For example, if a module attempts to store data in ACSYNT/VPI's view database, the module must supply the following information:

- view identification number
- database item number
- number of items to be stored
- the data to be stored

If any of the first three items are invalid, the database routine will issue an error message, ignore the input data, and set an error indicator.

An ACSYNT/VPI database consists of a set of FORTRAN named common blocks which may contain integer, real, and character data. Each common block is shared by only two database routines, one for getting the data from the common block and the other for putting data into the common block. "Appendix L. ACSYNT/VPI Database Definitions" on page 254 contains a list of all of ACSYNT/VPI's common blocks and the name of the corresponding "get" and "put" utility routines.

Menu Utility Routines

The ACSYNT/VPI menu utility routines were designed to provide functions for manipulating menus. The most commonly used menu utilities are discussed below:

CRMNRT (CREATE MENU ROOT)

CRMNRT is a general purpose routine used to create PHIGS root structures for all of the menu types discussed in "Menu Based Design" on page 67. CRMNRT inserts the following PHIGS elements into the menu root structure:

- view identifier
- menu type pick identifier having one of the following values:

  1 = command type menu
  2 = standard type menu
  3 = geometry type menu
  5 = view type menu

- execute sub-structure

The view, pick, and sub-structure identifiers are all input parameters to the CRMNRT routine.

OPMNU (OPEN MENU)

The OPMNU utility controls creation of the necessary PHIGS structures to display a specified command type menu. OPMNU calls CRMNRT to create a root structure containing the menu view identifier, pick identifier, and execute sub-structure elements. The root structure is then posted in the specified workstation after which the the menu's sub-structure is generated. The sub-structure contains the menu's title and pickable menu items.

CLMNU (CLOSE MENU)

CLMNU removes the current command type menu from the ACSYNT/VPI display screen by unposting the menu's root structure from the workstation. CLMNU also empties the menu's root and sub-structure so that the structures can be reused.

NEWMNU (NEW MENU)

NEWMNU, like OPMNU, is used to display a command type menu but also "pushes" the current menu onto an imaginary "menu stack". "Pushing" the current menu removes the menu from the screen so that a new sub-menu can be displayed. For example, when the user selects a menu item such as "GEOMETRY" from ACSYNT/VPI's main menu, the main menu is "pushed" from the screen onto the "menu stack" and replaced by the geometry sub-menu. To "push" the current menu from the display, NEWMNU calls the routine "PSHMNU" which unposts the current menu root structure and increments the current menu root and sub-structure identifiers by one. The new structure identifiers are then used by OPMNU to create a new menu root structure and sub-

structure for the specified menu. Menus are "popped" from the "menu stack" using the OLDMNU utility discussed below.

OLDMNU (OLD MENU)

OLDMNU performs the opposite function of NEWMNU by closing the current command menu and "popping" the previous menu back onto the screen. For example, when the "RETURN" item is selected from a sub-menu, OLDMNU calls "CLMNU" to remove the sub-menu from the screen and then calls "POPMNU" to redisplay the previous command menu. The combined use of NEWMNU and OLDMNU gives the user the ability to move up and down the ACSYNT/VPI menu structure.

GTMNIN (GET MENU INPUT)

GTMNIN is called from every ACSYNT/VPI menu module that requires user input. The routine works under the assumption that all PHIGS input devices are operating in event mode. Event mode input is discussed in detail in references 14 and 22.

GTMNIN performs the following functions:

1. updates the workstation to reflect any changes to the display since the last input event
2. puts the choice device in event mode which causes the choice button lights to turn on indicating that ACSYNT/VPI is ready for input
3. waits for event mode input from any device in event mode
4. puts the choice device in request mode to turn off the choice button lights indicating that ACSYNT/VPI is processing the input

In item 3. above, GTMNIN waits for the user to trigger an input event by, for example, picking a menu item or turning an input dial. GTMNIN then returns the class and device identification

number of the input device which caused the event. Note that GTMNIN does not return the actual input data but only information concerning the input device which generated an input event.

GTMNTI (GET MENU TYPE AND ITEM)

GTMNTI is called from a menu module only if the input device class returned by GTMNIN is "pick" which indicates that a menu item has been selected. GTMNTI's function is to determine the menu type and item number of a menu. For example, if the user selects the wing of the aircraft, GTMNTI will return a menu type of three and an item number of two. Menu type three indicates the aircraft geometry was selected and item two is the item number for the wings.

To determine the menu type and item, GTMNTI first calls the PHIGS routine PGTHP to get the current "pick path" data. A PHIGS pick path consists of a list of integers grouped in sets of three numbers. The first number in a set is a structure identifier, the second is a pick identifier, and the third is an element number [14]. For an ACSYNT/VPI menu pick, the pick identifiers in the first set identifies the type of menu picked. The pick identifiers in the last pick set identifies the item selected by the user. GTMNTI ignores the remaining pick path data.

PRSTMN (PROCESS STANDARD MENU INPUT)

When the user selects a standard menu item, the current menu module will call PRSTMN to process the item selected. PRSTMN's function is to call another menu module based on the menu item selected.

Shading and Hidden Surface Utility Routines

The following paragraphs describe the four main utility routines used to produce hidden surface and shaded images of ACSYNT/VPI aircraft models. All of the shading and hidden surface routines were developed by J. Sankar and were based in information presented in reference 15. A more detailed discussion of the shading and hidden surface methods used by ACSYNT/VPI can be found in "Data-flow and Geometric Modeling" on page 103.

SHDCMP (SHADE COMPONENTS)

The SHDCMP routine is called from within ACSYNT/VPI when the aircraft geometry is to be shaded. SHDCMP takes as input the number of aircraft components to be shaded and a list of component numbers. SHDCMP then calls subroutines which generate a PHIGS structure containing the shaded aircraft. The routine which called SHDCMP is responsible for posting the shading structure to be displayed on the workstation.

HSFCMP (COMPONENT HIDDEN SURFACE GENERATOR)

HSFCMP is identical to SHDCMP but only performs the hidden surface elimination portion of the shading process. Calling HSFCMP results in a PHIGS structure containing the hidden surface image of the current ACSYNT/VPI aircraft model.

DOSHAD (SHADING DRIVER)

DOSHAD performs all of the tasks necessary to shade an ACSYNT/VPI aircraft. The tasks include the following:

- inquiring the shading and wireframe structure identifiers from the view database
- unposting the current wireframe structure from the workstation
- building an list of aircraft components to be shaded.
- calling SHDCMP to create a shading structure
- posting the shading structure on the workstation for display

DOHSRF (HIDDEN SURFACE DRIVER)

DOHSRF performs the same tasks as DOSHAD but calls HSFCMP to create a hidden surface elimination structure instead of SHDCMP.

List Manipulating Utility Routines

The five list manipulating routines described below were developed to simplify the use of integer lists. Integer lists are used in ACSYNT/VPI to keep track of aircraft components which have been selected for shading, hidden surface elimination, or invisibility (no show). In the following discussion, an integer list as implemented in FORTRAN is a one-dimensional integer array of length n.

SCHLST (SEARCH LIST)

SCHLST searches for a specified integer value in a list of integers. If the integer item is found in the list, the location of the last occurrence of the item is returned to the calling program. A list location is any number between 1 and n, where n is the length of the specified list. If the integer is not found, a location of 0 is returned to the calling program.

ADDLST (ADD ITEM TO LIST)

ADDLST adds a specified integer item to the end of an integer list and increments the length of the list by one.

RMVLST (REMOVE ITEM FROM LIST)

RMVLST removes an integer item from a list at a specified location by shifting all of the items above the location down by one. The length of the list is then decremented by one. The effect is that the integer item at the specified location is squeezed out of the list.

CHKLST (CHECK LIST)

The CHKLST utility calls SCHLST to determine if a specified integer item is in the list. If the item is in the list, CHKLST calls RMVLST to remove the item from the list. If the item is not in the list, CHKLST calls ADDLST to add the item to the list.

UNQLST (UNIQUE LIST)

UNQLST adds a specified integer item to a list if the item is not already in the list. CHKLST is used to determine if the item is in the list and if not, ADDLST is called to add the item to the list. Note that UNQLST differs from CHKLST in that if the item is found within the list, the item will not be deleted.

Vector and Matrix Utility Routines

VNORM and MATIN are currently the only two vector and matrix routines in the ACSYNT/VPI utility library. Both routines are discussed below.

VNORM (NORMALIZE VECTOR)

VNORM normalizes a three dimensional vector using the following equation:

$$\hat{\tilde{v}} = \frac{\tilde{v}}{|\tilde{v}|} = \frac{\tilde{v}}{\sqrt{v_x^2 + v_y^2 + v_z^2}} \tag{7.3}$$

where:

$\tilde{v}$ is the vector to be normalized and

v_x is the x component of vector $\tilde{v}$

v_y is the y component of vector $\tilde{v}$

v_z is the z component of vector $\tilde{v}$

$\hat{\tilde{v}}$ is the normalized vector.

MATIN (MATRIX INVERSE)

MATIN inverts an nxn matrix using Gauss-Jordan reduction, where n is the order of the matrix. MATIN was adopted from reference 25.

Miscellaneous Utility Routines

The most commonly used miscellaneous utility routines are defined in the following pages.

MSG (MESSAGE)

The MSG routines controls the display of messages issued on the ACSYNT/VPI screen. Input to MSG is the workstation identifier and a text string containing the message to be displayed. MSG manages a character array which permits five message lines to be displayed at a time. Each message line is generated by one call to the MSG routine which results in the previous message lines scrolling up and the specified line being inserted as the current message. For example, the first time MSG is called the message area will contain the following:

```
FIRST CALL TO MSG RESULTED IN THIS MESSAGE LINE
```

Calling MSG again will result in:

```
FIRST CALL TO MSG RESULTED IN THIS MESSAGE LINE
SECOND CALL TO MSG RESULTED IN THIS MESSAGE LINE
```

Note that the first message has been scrolled up and the second message is now the current message. The current message is highlighted in white on the ACSYNT/VPI screen. Calling MSG four more times will result in the first message being pushed out of the message area as follows:

```
SECOND CALL TO MSG RESULTED IN THIS MESSAGE LINE
THIRD CALL TO MSG RESULTED IN THIS MESSAGE LINE
FOURTH CALL TO MSG RESULTED IN THIS MESSAGE LINE
FIFTH CALL TO MSG RESULTED IN THIS MESSAGE LINE
SIXTH CALL TO MSG RESULTED IN THIS MESSAGE LINE
```

Another form of the MSG utility is described below.

INFMSG (INFORMATION MESSAGE)

INFMSG is used to generate a screen message which immediately appears on the workstation screen. INFMSG calls the MSG utility to write the message but then updates the workstation before passing control back to the calling program. INFMSG is used to inform the user that ACSYNT/VPI is performing calculations that may take a long time, such as when a shaded image is being generated. In such cases, if the MSG routine were used, the message would not appear on the screen until after the calculations were complete and the GTMNIN routine was called to update the workstation.

CAPS (CAPITALIZE STRING)

The CAPS routine takes as input a character string and outputs the same string with all of the original lower case letters capitalized. For example, if the character string "%ABCdef$123" were input to CAPS, the string would be returned as "%ABCDEF$123". Note that nonalphabetic characters are unaffected.

CK4FIL (CHECK FOR FILE)

CK4FIL checks for the existence of a specified data file. Checking for the existence of data files is necessary to prevent ACSYNT/VPI from trying to read a nonexistent input file. CK4FIL uses the FORTRAN function "INQUIRE" to determine if a specified file exists.

NOTE: The syntax and function for the INQUIRE statement may differ between systems. Therefore, CK4FIL may need to be changed when ACSYNT/VPI is ported to non-IBM systems.

CONERR (CONSOLE ERROR MESSAGE)

CONERR is used to write an error message to the FORTRAN console unit which typically represents a text terminal separate from the actual graphics screen. The following is an example of the console error message produced by CONERR:

```
ERROR (GTRCMP) --> DATABASE ITEM NUMBER OUT OF ALLOWABLE BOUNDS.
```

CONERR requires two input parameters, the name of the subroutine issuing the error and the error message text. In the above example, the subroutine "GTRCMP", which issued the error, is inclosed in parenthesis and the error message text follows the right facing arrow ("-->").

CPYVCT (COPY VECTOR)

CPYVCT copies one vector of information to another vector. A vector in FORTRAN is just a real or integer one-dimensional array of any length. CPYVCT is used in many of the database utility routines to copy data to and from a database array.

AGTVL (ACSYNT/VPI GET VALUATOR)

AGTVL waits until the user stops turning a valuator input valuator dial and then returns the dial's current input value. AGTVL calls the following PHIGS routines within a conditional loop:

PGTVL gets a valuator's input value

PWAIT waits for event mode input for only 0.2 seconds

If a valuator event occurs within the 0.2 second limit, AGTVL loops back to PGTVL to get the new valuator value. The process is repeated until no valuator event occurs within the 0.2 second time limit at which point AGTVL returns control and the last valuator value to the calling program.

AGTVL is currently called by the ACSYNT/VPI utility PGMVL (process geometry valuator) to get valuator input for three-dimensional viewing transformations. AGTVL prevents ACSYNT/VPI from performing unnecessary viewing calculations by passing back only the last valuator input value.

NOTE: For the AGTVL routine to work properly on an IBM CMS userid, the userid's "realtimer" must be turned on using the "DIRM OPT" command.

GTINLN (GET FILE INPUT LINE)

GTINLN reads one line of a specified input file into a character variable supplied by the calling program. If the input line starts with an asterisk the line is considered to be a comment and the next file line is automatically read. Once a line is found which does not start with an an asterisk, GTINLN returns the input line to the calling program.

CKVLVL (CHECK VALUATOR VALUE)

CKVLVL checks to see if a valuator's current value is either 0.0 or 1.0. If the value is 0.0 or 1.0, the valuator is reinitialized to a value of 0.5 using the AINVL utility. If the value is not 0.0 or 1.0, the valuator's current value is stored in the named valuator common block "/VALUAT/ VALUES". "/VALUAT/ VALUES" contains all of the current normalized valuator values.

OPNUM (OPEN NUMBER INPUT)

OPNUM is one of a set of routines designed to implement the number input method described in "Interactive Input Methods" on page 70. A menu module requiring numeric input will call OPNUM before calling the module's menu driver subroutine (Mxxxx). OPNUM was originally

written to perform the functions performed by the INNMIN utility described below, but now simply issues a prompt message to inform the user to enter numeric input.

INNMIN (INITIALIZE NUMBER INPUT)

Each menu module which requires numeric input calls INNMIN from the module's menu input loop routine (INxxxx). INNMIN's function is to initialize the string device for numeric input. The input parameters to INNMIN includes the number of input values, the number of decimal places to be displayed for each value, and a list of values. Each value is inserted into a character string variable using a formated internal write. The character string is then used to initialize the screen echo for the string input device. The format used to write the value into the string depends upon the number of decimal places specified for the value. For example, if the five values 4.5437, 2.3, 5.0, 7.123 and 1.0 are passed to INNMIN with the corresponding number of displayable decimal places as 0, 2, 2, 2, and 0, INNMIN will initialize the string device to display the following:

```
_      4,     2.30,     5.00,     7.12,        1
```

Note that the first and last numbers were truncated and displayed as integers.

PNMVL (PROCESS NUMERIC INPUT VALUATOR)

PNMVL processes numeric input generated by a valuator device as described in "Interactive Input Methods" on page 70. PNMVL is called from a menu module's process routine (Pxxxx) and performs the following functions:

- obtains the valuator's current normalized value using the PHIGS function "GET VALUATOR" (PGTVL)
- converts the normalized valuator value to a non-normalized value using Equation 7.1 on page 73
- checks the current valuator value using the CKVLVL utility.

PNMVL is complemented by the PNMST routine described below.

PNMST (PROCESS NUMERIC INPUT STRING)

Whenever an ACSYNT/VPI user enters a numeric value by pressing the enter key, PNMST is called to process the input string. PNMST's responsibility is to get the input string using the PHIGS function "GET STRING" and to determine if the string is valid.

If the string is valid, PNMST performs the following:

- removes the input values from the string using a FORTRAN internal read
- limits each value to the maximum and minimum ranges specified by the calling program
- returns the limited values to the calling program

If the input string is invalid, PNMST ignores the input line and issues an appropriate error message. To determine an input strings validity, the subroutine PRSERR (PARSE FOR ERROR) is called which performs the following tests on the input line:

- checks to see if each character in the line is one of the following:

  0123456789, + - and BLANK

- determines if all decimal points and minus signs are used properly

If the input line fails either test, PRSERR returns and error indicator informing PNMST to ignore the input line.

CLNUM (CLOSE NUMERIC INPUT)

CLNUM's function is to remove the numeric input string from the ACSYNT/VPI input area. The string is removed by initializing the string device with a blank initial string. String initialization is performed by the AINST (ACSYNT/VPI INITIALIZE STRING) utility routine.

Data-flow and Geometric Modeling

Overview

Reference 27 defines geometric modeling as the "collection of methods used to define the shape and other geometric characteristics of an object using analytic geometry, vector calculus, topology, set theory, and computational methods." Geometric modeling is used within ACSYNT/VPI to mathematically describe the geometry of an ACSYNT aircraft model. The purpose of modeling the geometry is to produce a displayable image of the aircraft and to provide ACSYNT with information required to perform drag calculations.

ACSYNT/VPI is designed to create and maintain the following five geometric models of an aircraft's geometry:

- ACSYNT Component Model
- ACSYNT/VPI Component Model
- ACSYNT/VPI Component Cross Section Model
- ACSYNT/VPI Component Surface Model
- ACSYNT/VPI PHIGS Display Model

Each model is represented by data stored in memory linked together by geometric modeling equations as shown in Figure 28 on page 105.

The ACSYNT/VPI modeling process was developed with the help of J. Sankar and starts with the ACSYNT definition of an aircraft's geometry and ends with a PHIGS representation which can be displayed on a graphics workstation. The relationship between ACSYNT/VPI's geometric models is shown in Figure 29 on page 106[1]. Each of the ACSYNT/VPI geometric models are discussed in detail in the following pages.

ACSYNT Aircraft Component Model

ACSYNT models an aircraft as a set of individual components each defined by a set of parameters supplied by the user or, in some cases, calculated by ACSYNT. Figure 30 on page 107 through Figure 32 on page 109 illustrate ACSYNT's definition of a fuselage, store, and wing.

Currently, ACSYNT's geometric modeling capabilities are limited to the following aircraft components [1]:

- fuselage defined as a nose, cylinder, and after-body
- symmetrical trapezoidal wing
- variable sweep, asymmetric, trapezoidal wings
- forward and trailing edge strakes (wing extensions)
- horizontal tail
- vertical tail
- canard
- wing mounted engine pods defined as cylinders
- fuselage mounted engine pods defined as cylinders
- miscellaneous internal compartments (bays)

---

[1] The F16 shown in Figure 29 on page 106 was drawn on CADAM and is not an ACSYNT/VPI model.

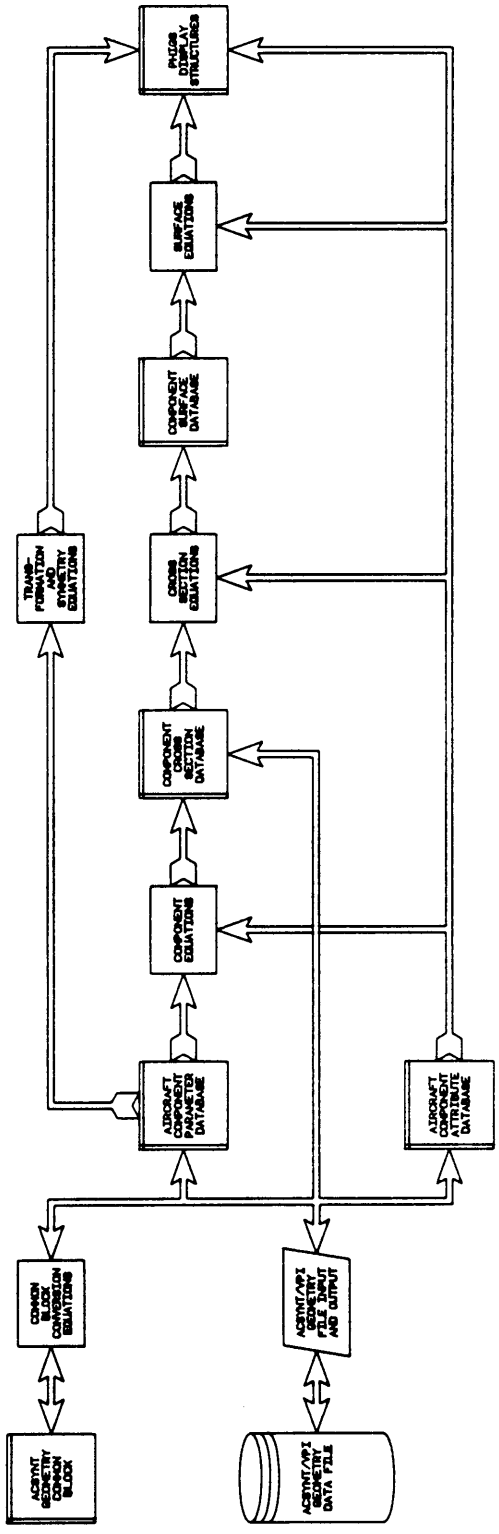


Figure 28. ACSYNT/VP1 data-flow diagram.

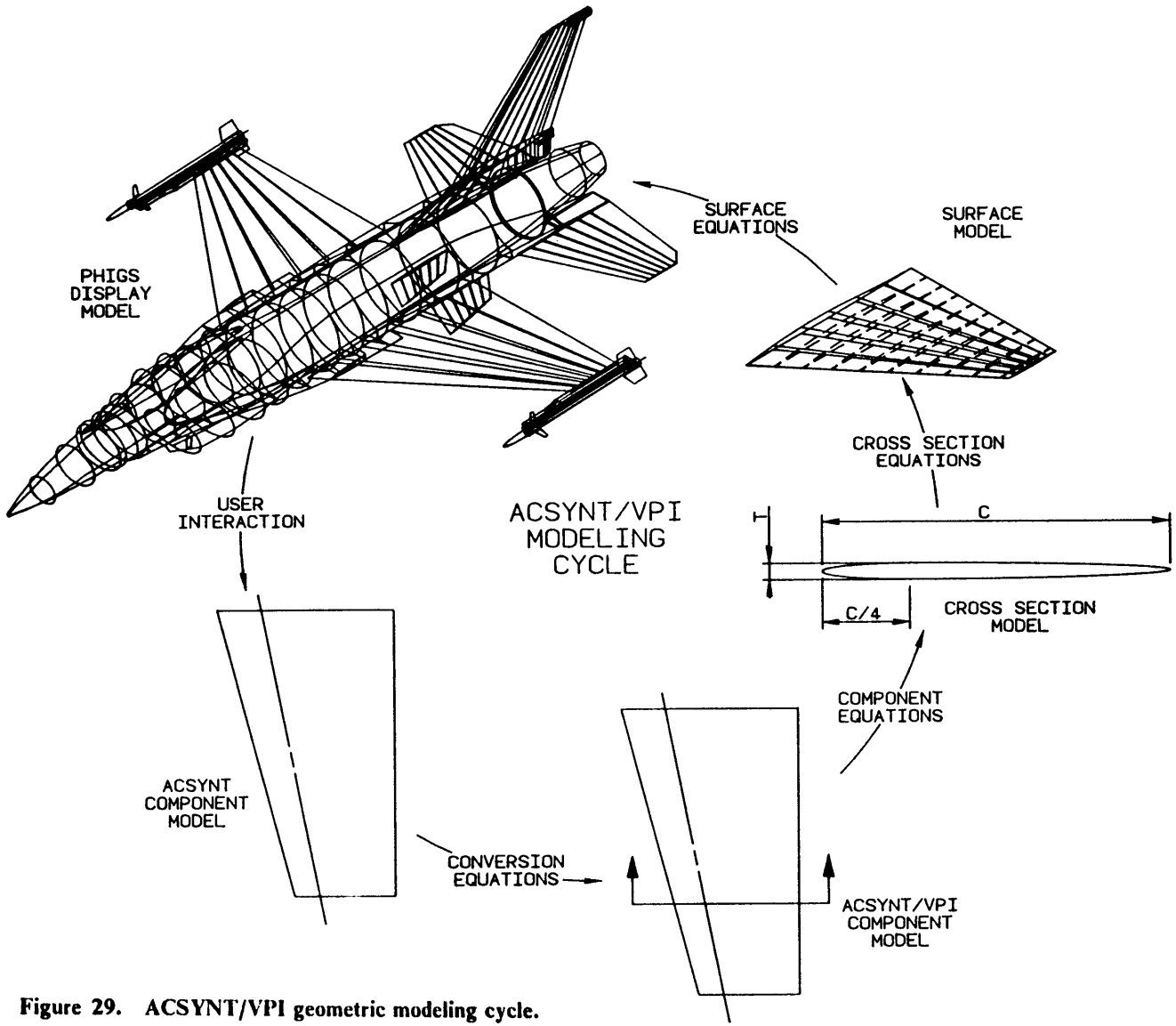


Figure 29. ACSYNT/VPI geometric modeling cycle.

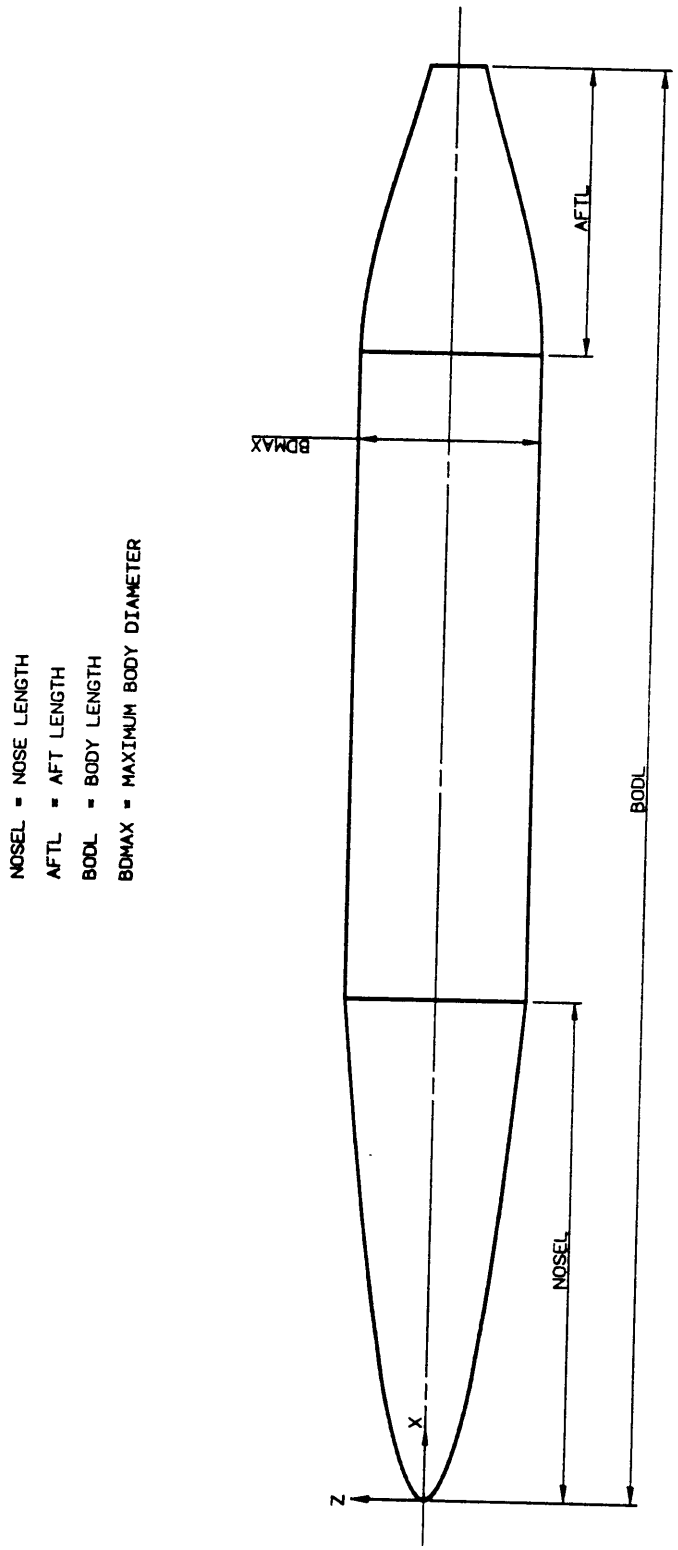


Figure 30. ACSYNT fuselage definition.

R = POD OR STORE RADIUS

L = POD OR STORE LENGTH

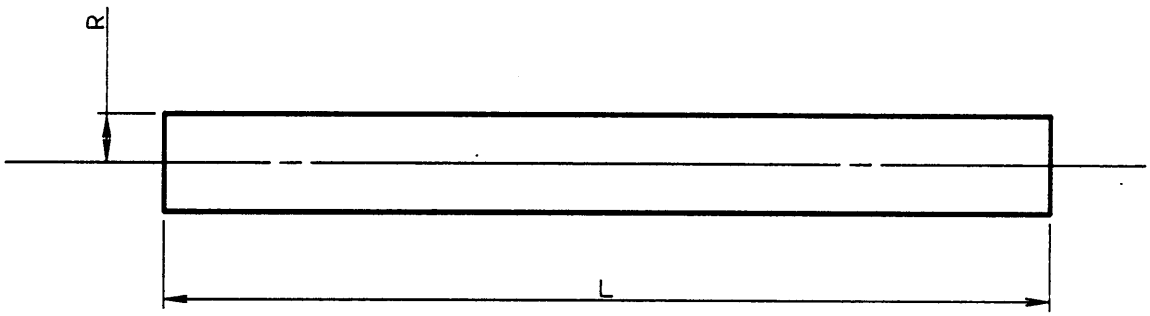


Figure 31. ACSYNT engine pod definition.

AR = ASPECT RATIO = RC/TC

AREA = TOTAL WING AREA = RC•L-((RC-TC)•L)/2.0

DIHED = WING DIHEDRAL ANGLE (MEASURED UP FROM HORIZONTAL)

SWEEP = SWEEP ANGLE

TAPER = TAPER RATIO = TT/RT

TCROOT = THICKNESS TO CHORD RATIO AT ROOT = RT/RC

TCTIP = THICKNESS TO CHORD RATIO AT TIP = TT/TC

LFLAPC = FRACTION OF CHORD COVERED BY LEADING EDGE FLAPS = LF/TC

TFLAPC = FRACTION OF CHORD COVERED BY TRAILING EDGE FLAPS = TF/TC

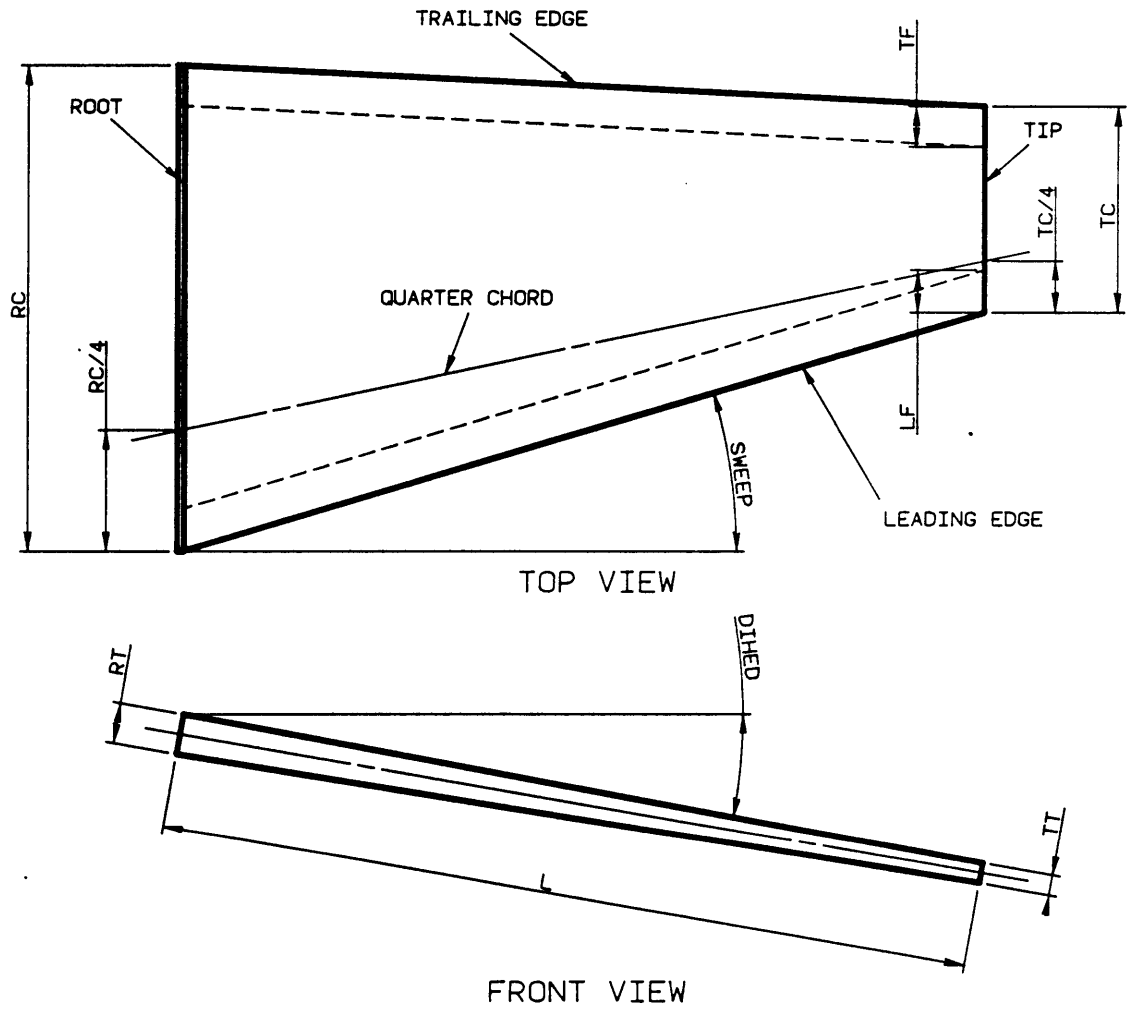


Figure 32. ACSYNT airfoil definition.

Note that ACSYNT does not define items such as canopies, inlets, nozzles, and stores and therefore does not taken them into consideration in drag calculations. Future versions of ACSYNT/VPI will include enhanced drag calculations to take advantage of ACSYNT/VPI's component definition described in the next section.

ACSYNT/VPI Aircraft Component Model

Like the ACSYNT component model, the ACSYNT/VPI component model defines each aircraft component parametrically. A component's parameters, in conjunction with a set of analytical equations, dictate the shape of a component. Figure 33 on page 111 through Figure 35 on page 113 illustrates ACSYNT/VPI's component definitions which can be generalized to generate the following components:

- fuselage including:
  - nose section
  - transition section
  - inlets[2]
  - mid-section
  - aft-section
  - nozzles
- canopy
- airfoils including:
  - wings
  - horizontal tail
  - vertical tail
  - canards
  - strakes
  - pylons
- stores and engine pods including:
  - fuel tanks
  - bombs
  - missiles
  - miscellaneous stores
  - wing mounted engines

---

[2] Note that not all inlets to be modeled are depicted in Figure 33 on page 111.

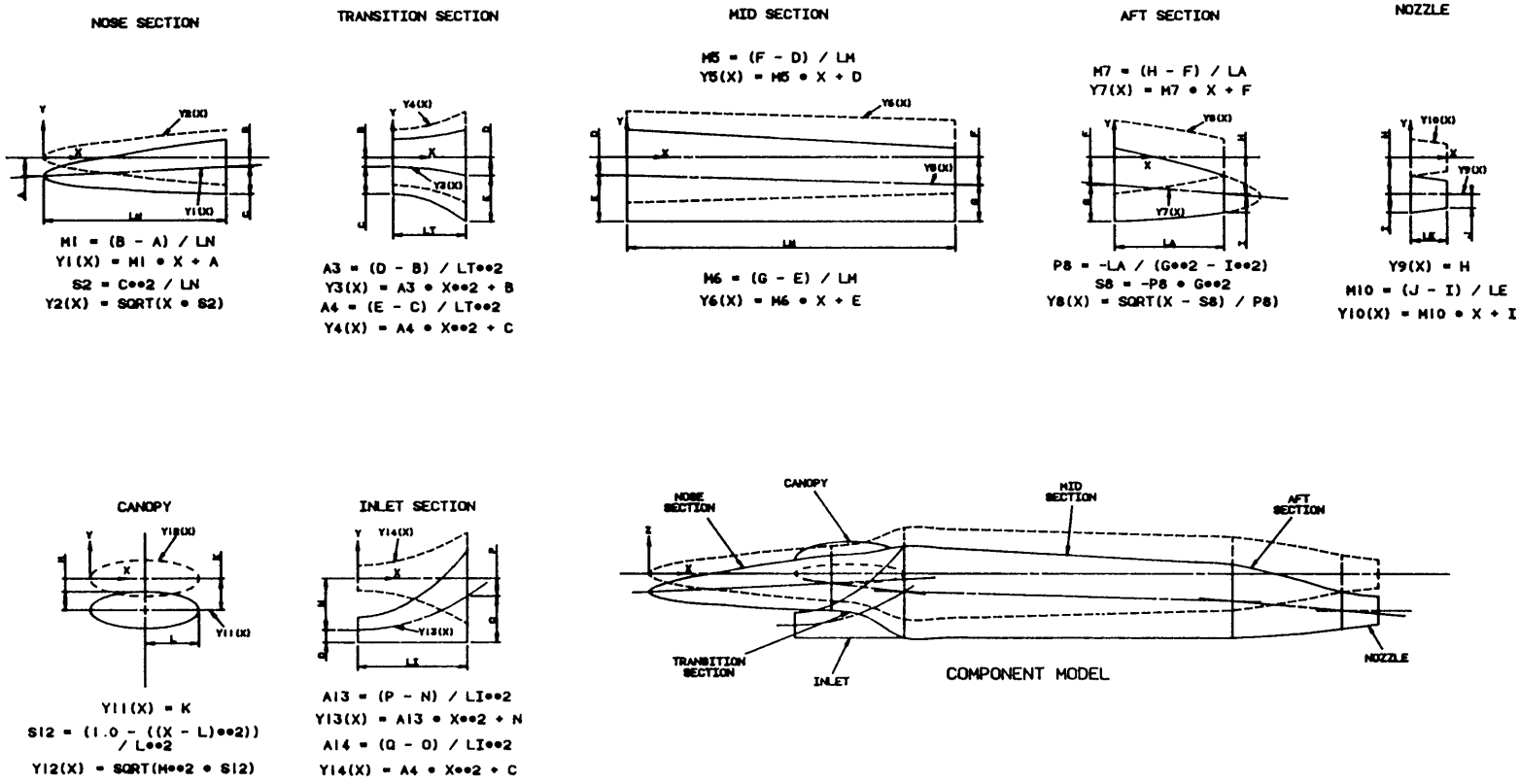


Figure 33. ACSYNT/VPI fuselage definition.

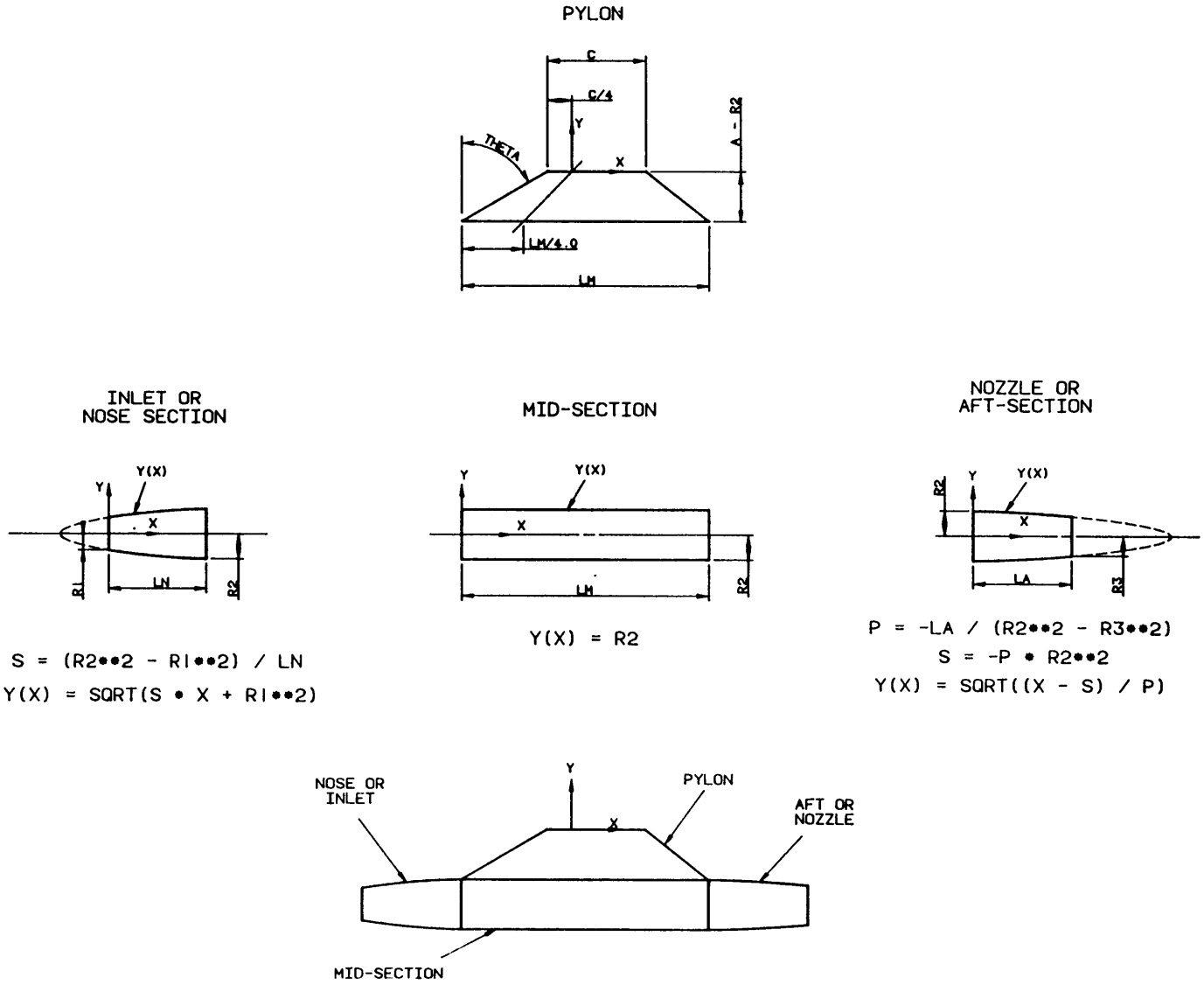


Figure 34. ACSYNT/VPI store and engine pod definition.

AR	= ASPECT RATIO	TCROOT	= THICHNESS TO CHORD RATIO AT THE ROOT
RC	= ROOT CHORD	TCTIP	= THICHNESS TO CHORD RATIO AT THE TIP
TC	= TIP CHORD = RC/AR	RT	= ROOT THICKNESS = RC • TCROOT
SWEEP	= QUARTER CHORD SWEEP ANGLE	TT	= TIP THICKNESS = TC • TCTIP
BDMAX	= MAXIMUM BODY DIAMETER	C(X)	= RC - (RC - TC) • X / L
SPAN	= TOTAL WING SPAN	T(X)	= RT - (RT - TT) • X / L
L	= WING LENGTH = (SPAN-BDMAX)/2.0	Y(X)	= -X • TAN(SWEEP)
		Z(X)	= 0.0

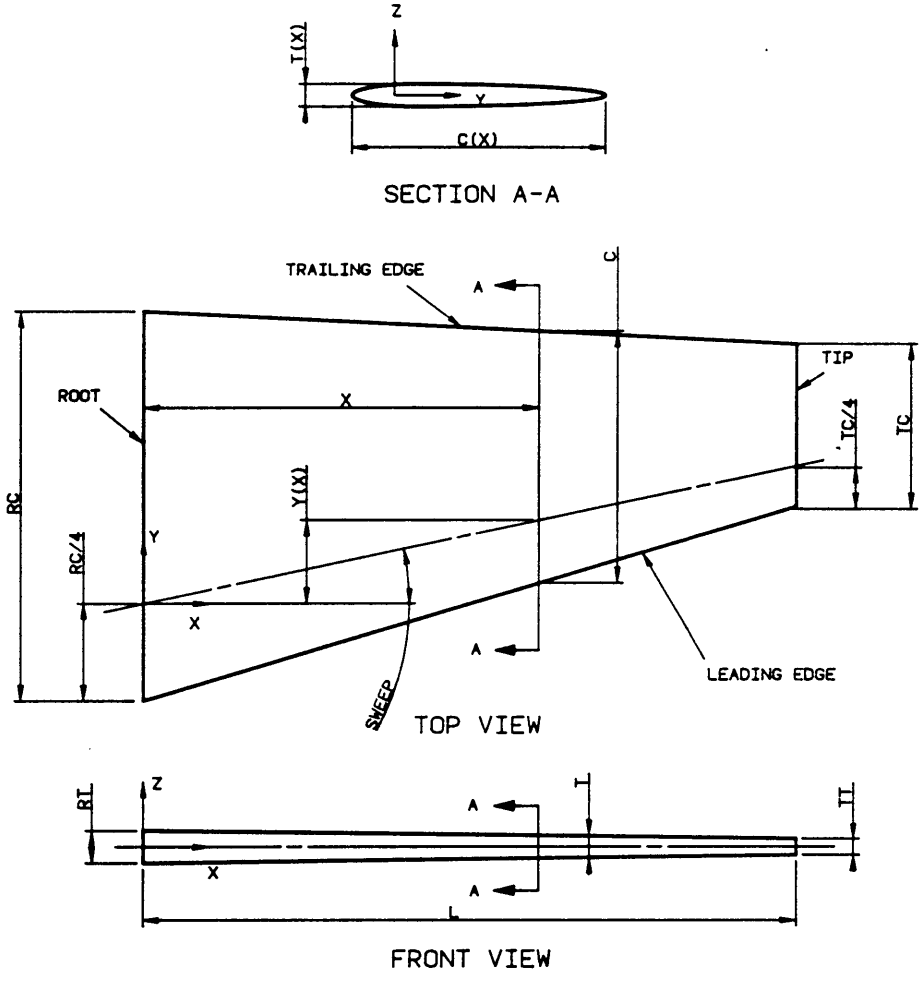


Figure 35. ACSYNT/VPI airfoil definition.

- fuselage mounted engines

The ACSYNT/VPI component model is more flexible than the current ACSYNT model, and in some cases the ACSYNT model has no corresponding component definition. For example, unlike ACSYNT, ACSYNT/VPI can model engine inlets, canopies, stores, and pylons. Also, the ACSYNT/VPI component model defines attributes for each component which aid in the creation of the cross-section, surface, and PHIGS models.

The data for ACSYNT/VPI's component model is stored in the "ACSYNT/VPI geometry component database." The database contains the following information:

- number of defined aircraft components
- aircraft component attributes for each component:
  - component label
  - component existence flag
  - component type identifier
  - number of cross-sections
  - number of points per cross-section
  - number of component parameters
  - number of isoparametric surface lines in the u direction
  - number of isoparametric surface lines in the w direction
  - local symmetry flag
  - global symmetry flag
  - wireframe color index
  - wireframe line type
  - component attribute structure identifier
  - component local symmetry structure identifier
  - component structure identifier
- aircraft component parameters for each component:
  - component rotation about global x axis
  - component rotation about global y axis
  - component rotation about global z axis
  - component translation along x axis
  - component translation along y axis
  - component translation along z axis
  - component dimensions

The first release version of ACSYNT/VPI will include a full implementation of the ACSYNT/VPI component model presented above. But, for the developmental version of ACSYNT/VPI, component dimensions were taken directly from the ACSYNT component model and used to generate the cross-section model discussed in the following section.

ACSYNT/VPI Component Cross Section Model

OVERVIEW

As illustrated in Figure 36 on page 116, the component cross-section model defines the shape of each aircraft component as a set of parametric cross-sections. The cross-section model is used within ACSYNT/VPI to generate the surface model of the aircraft as described in "ACSYNT/VPI Component Surface Model" on page 121.

Figure 37 on page 117 shows the cross-section types defined at the time of this writing which include elliptical, rectangular with rounded corners, and airfoil. Note that circular cross-sections are a special case of the elliptical cross-section. Also, rectangular cross-sections can be generated from the rectangular with rounded corners cross-section by setting the corner radii to zero.

Each type of cross-section is generated by one or more parametric equations which yield points and tangents for specified values of a parametric variable in the range 0.0 to 1.0. The point and tangent equations for the elliptical and airfoil cross-sections are given below.

ELLIPTICAL CROSS SECTION EQUATIONS

The point and tangent equations presented below are contained in the ACSYNT/VPI routines XSELLP and XSELLT respectively.

$$x = x_0 \qquad x' = 0.0 \tag{7.4}$$

$$y = y_0 + a\sin(2.0\pi u) \qquad y' = a\cos(2.0\pi u) \tag{7.5}$$

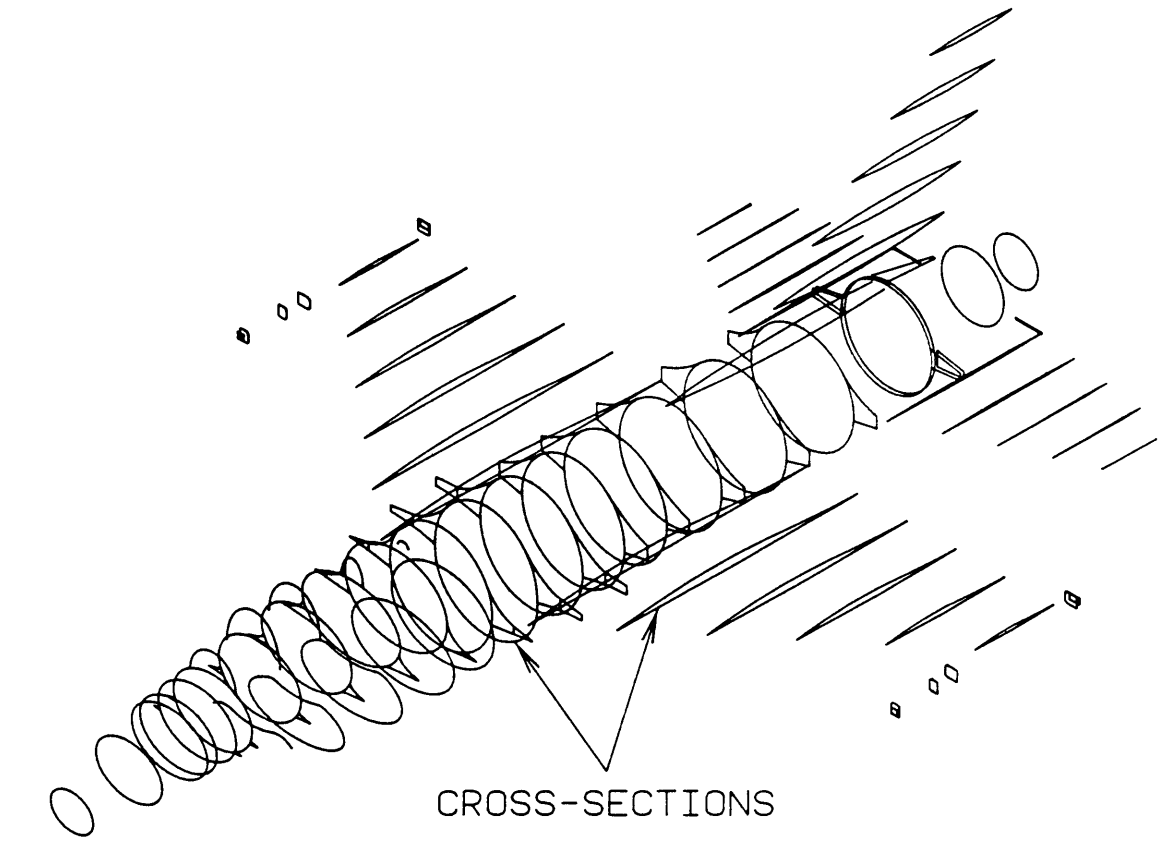
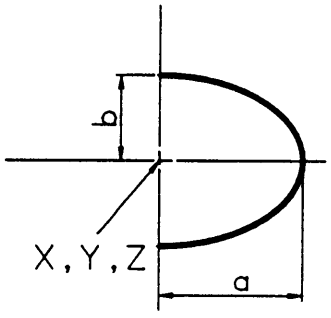
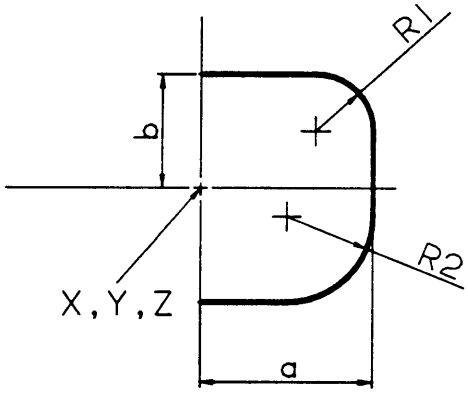


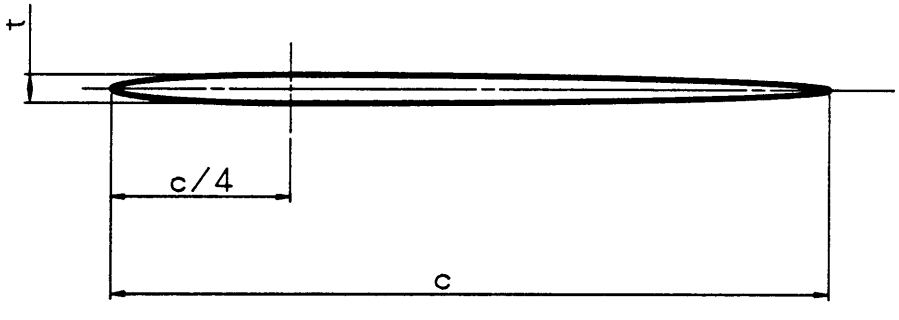
Figure 36. Definition of aircraft components by cross-sections.



ELLIPTICAL



RECTANGULAR WITH ROUNDED CORNERS



AIRFOIL

Figure 37. Cross section type definitions.

$$z = z_0 + b\cos(2.0\pi u) \qquad z' = -b\sin(2.0\pi u) \tag{7.6}$$

where:

x is the calculated x location of a point on the ellipse

y is the calculated y location of a point on the ellipse

z is the calculated z location of a point on the ellipse

x' is the calculated value of the ellipse tangent vector in the x direction

y' is the calculated value of the ellipse tangent vector in the y direction

z' is the calculated value of the ellipse tangent vector in the z direction

x_0 = the x location of the center of the ellipse

y_0 = the y location of the center of the ellipse

z_0 = the z location of the center of the ellipse

a is the major radius of the ellipse

b is the minor radius of the ellipse

u is the parametric variable in the range 0.0 to 1.0

AIRFOIL CROSS SECTION EQUATIONS

The ACSYNT/VPI airfoil cross-section was developed to simplify the generation of airfoils by approximating them as two ellipses; one representing the leading edge of the airfoil and the other the trailing edge. As shown in Figure 37 on page 117, the lead edge of an airfoil cross-section is defined as one quarter of the total cross-section chord. Therefore, by approximating an airfoil cross-section as two ellipses, only the thickness and chord of the cross-section need to be specified. In the future, other airfoil geometries will be added.

The parametric equations presented below are contained in the ACSYNT/VPI subroutines XSAF1P and XSAF1T which calculate points and tangents on an airfoil cross-sections respectively.

$$x = x_0 \qquad x' = 0.0 \tag{7.7}$$

$$y = y_0 + \frac{c}{4}\sin(2.0\pi u) \quad y' = \frac{c}{4}\cos(2.0\pi u) \quad when \quad 0 \le u < \frac{1}{2} \tag{7.8}$$

$$y = y_0 + \frac{3c}{4}\sin(2.0\pi u) \quad y' = \frac{3c}{4}\cos(2.0\pi u) \quad when \quad \frac{1}{2} \le u \le 1 \tag{7.9}$$

$$z = z_0 + b\cos(2.0\pi u) \quad z' = -b\sin(2.0\pi u) \tag{7.10}$$

where:

x is the calculated x location of a point on the airfoil

y is the calculated y location of a point on the airfoil

z is the calculated z location of a point on the airfoil

x' is the calculated value of the airfoil tangent vector in the x direction

y' is the calculated value of the airfoil tangent vector in the y direction

z' is the calculated value of the airfoil tangent vector in the z direction

x_0 is the x location of the cross-section origin

y_0 is the y location of the cross-section origin

z_0 is the z location of the cross-section origin

c is the chord length of the airfoil cross-section

t is the maximum airfoil thickness

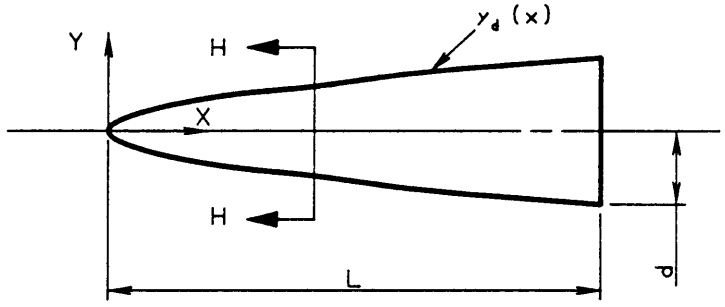
u is the parametric variable in the range 0.0 to 1.0

CALCULATION AND STORAGE OF CROSS SECTION DATA

Cross section parameters are stored in ACSYNT/VPI's component cross-section database which is generated from the parametric equations defining the shape of each aircraft component. For example, as shown in Figure 38, the center location and dimension of a cross-section within the nose of an ACSYNT/VPI fuselage is defined by the following parametric equations:

$$x_c(x) = x \quad y_c(x) = \frac{b-a}{l}x + a \quad z_c(x) = 0.0 \tag{7.11}$$

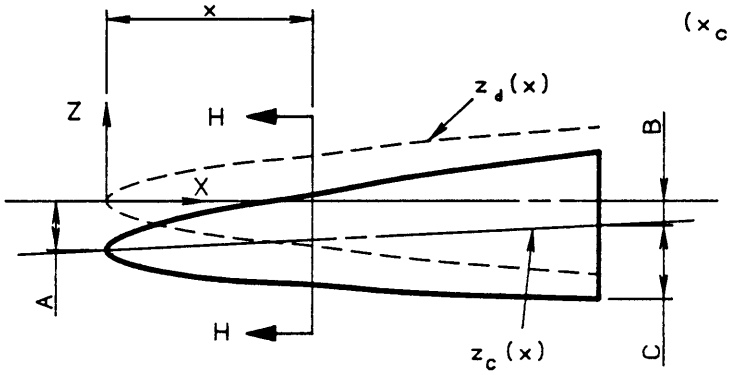
$$y_d(x) = \sqrt{x\frac{c^2}{l}} \quad z_d(x) = \sqrt{x\frac{d^2}{l}} \tag{7.12}$$

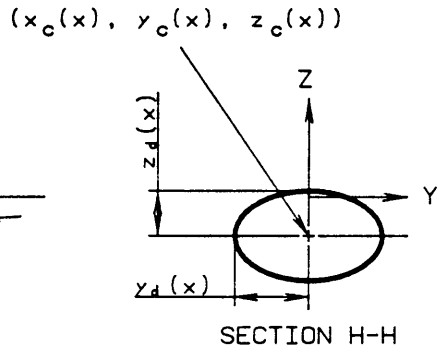


$$s = c^2/l$$

$$y_d(x) = SQRT(x * s)$$

$$y_c(x) = 0.0$$





$$m = (b-a) / l$$

$$z_c(x) = mx + a$$

$$s = c^2/l$$

$$z_d(x) = SQRT(x * s)$$

$$x_c(x) = x$$

Figure 38. Example cross-section definition using nose component equations.

where:

- a, b, c, d and l are the component parameters defined in Figure 38
- $x_c(x)$ is the x location of cross-section center
- $y_c(x)$ is the y location of cross-section center
- $z_c(x)$ is the z location of cross-section center
- $y_d(x)$ is the y dimension of a nose cross-section at position x
- $z_d(x)$ is the z dimension of a nose cross-section at position x
- x is the position of the cross-section measured along the nose
- l is the length of the nose

By incrementing the value of x along the nose, the location and cross-section parameters can be calculated for a given value of the nose parameters a, b, c, d and l. The same is true for all of the components shown in Figure 33 on page 111 through Figure 35 on page 113.

ACSYNT/VPI Component Surface Model

ACSYNT/VPI generates a component surface model by "lofting" parametric surface patches over the cross-sections defining a component. The concept of surface lofting is illustrated in Figure 39.

The surface model of an ACSYNT/VPI component is actually the composite of many surface "patches". A surface patch is defined by reference 27 as a curve-bounded collection of points whose coordinates are given by continuous, two-parameter, single valued mathematical functions of the form:

$$x = x(u,w) \quad y = y(u,w) \quad z = z(u,w) \tag{7.13}$$

where the parameters u and w are constrained to the intervals $[0,1]$.

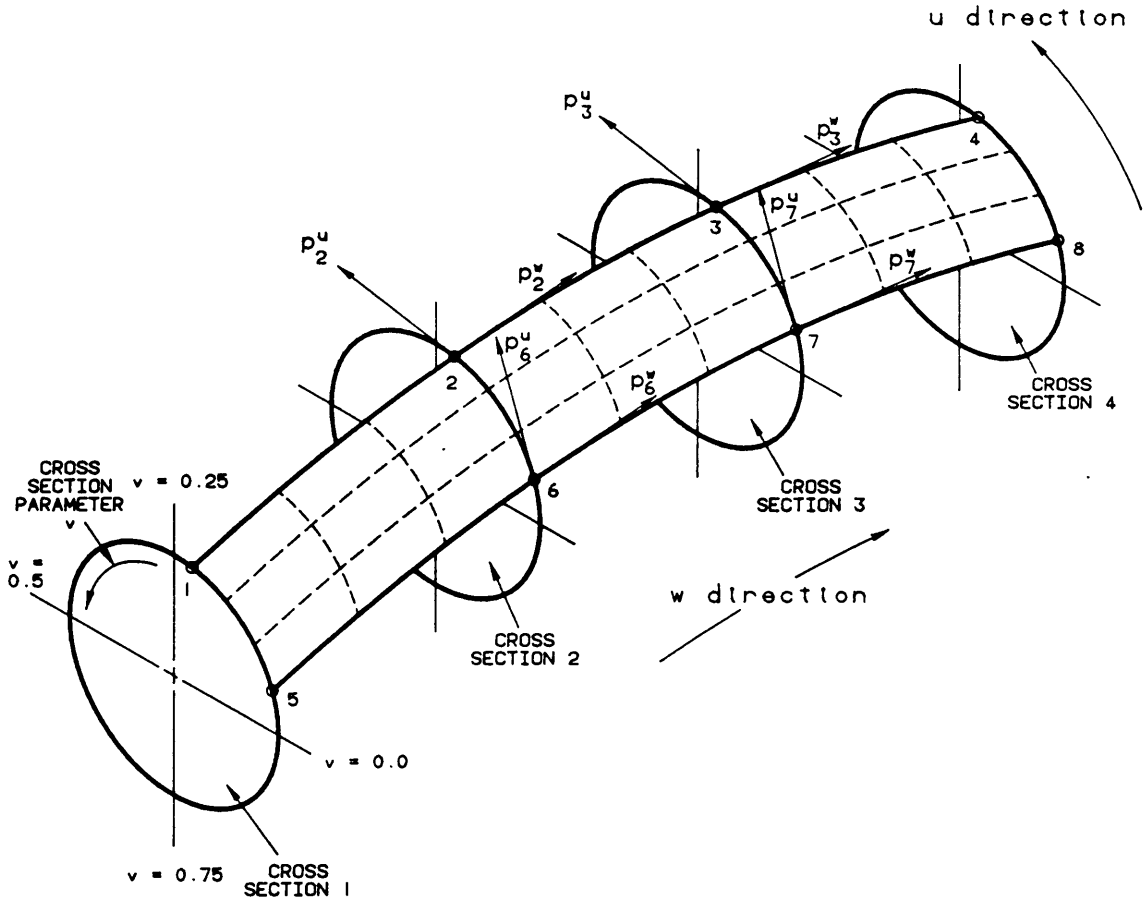


Figure 39. Surface points and tangent vectors for lofted surfaces.

The Hermite bicubic surface patches used in the ACSYNT/VPI code are defined by four corner points, tangent vectors, and twist vectors as depicted in Figure 40 on page 124. All corner points, tangent vectors, and twist vectors are generated directly from the cross-section model and not from a boundry curve representation. Each corner point tangent defines the slope of the surface in either the u or w direction at each corner. And, as shown in the figure, twist vectors define the slope of tangent auxiliary curves formed by the tangent arrowheads along each patch boundry [14].

A point $\mathbf{p}(u,w)$ on a bicubic patch can be calculated using the vector equation presented below. The equation actually represents three scalar equations; one each for the x, y, and z coordinates of the point $\mathbf{p}(u,w)$.

$$\begin{aligned}\mathbf{p}(u,w) &= [F_1(u)\ F_2(u)\ F_3(u)\ F_4(u)]\begin{bmatrix} \mathbf{p}_{00} & \mathbf{p}_{01} & \mathbf{p}^{w}_{00} & \mathbf{p}^{w}_{01} \\ \mathbf{p}_{10} & \mathbf{p}_{11} & \mathbf{p}^{w}_{10} & \mathbf{p}^{w}_{11} \\ \mathbf{p}^{u}_{00} & \mathbf{p}^{u}_{01} & \mathbf{p}^{uw}_{00} & \mathbf{p}^{uw}_{01} \\ \mathbf{p}^{u}_{10} & \mathbf{p}^{u}_{11} & \mathbf{p}^{uw}_{10} & \mathbf{p}^{uw}_{11} \end{bmatrix}\begin{bmatrix} F_1(w) \\ F_2(w) \\ F_3(w) \\ F_4(w) \end{bmatrix} \\ &= \mathbf{F}(u)[\mathbf{B}]\mathbf{F}(w)^T \end{aligned} \tag{7.14}$$

where:

$\mathbf{p}(u,w)$ is the 1X3 position vector of a point on the bicubic patch evaluated at u and w

u, w are the parametric variables ranging from 0.0 to 1.0

$F_i(u)$, where $i = 1,2,3,4$, is the i^{th} one of the following four blending functions evaluated at u:

$$\begin{aligned} F_1(u) &= 2u^3 - 3u^2 + 1 \\ F_2(u) &= -2u^3 + 3u^2 \\ F_3(u) &= u^3 - 2u^2 + u \\ F_4(u) &= u^3 - u^2 \end{aligned} \tag{7.15}$$

NOTE: For a given u, the same values for $F_i(u)$ are used to calculate the x, y, and z coordinates of $\mathbf{p}(u,w)$.

$F_i(w)$, where $i = 1,2,3,4$, is the i^{th} one of the blending function shown above evaluated at w

$\mathbf{p}_{ij}$, where $i,j = 0$ or 1, are the 1X3 position vectors of the four patch corner points

$\mathbf{p}^{u}_{ij}$, where $i,j = 0$ or 1, are the 1X3 tangent vectors at each of the four corner points in the u direction

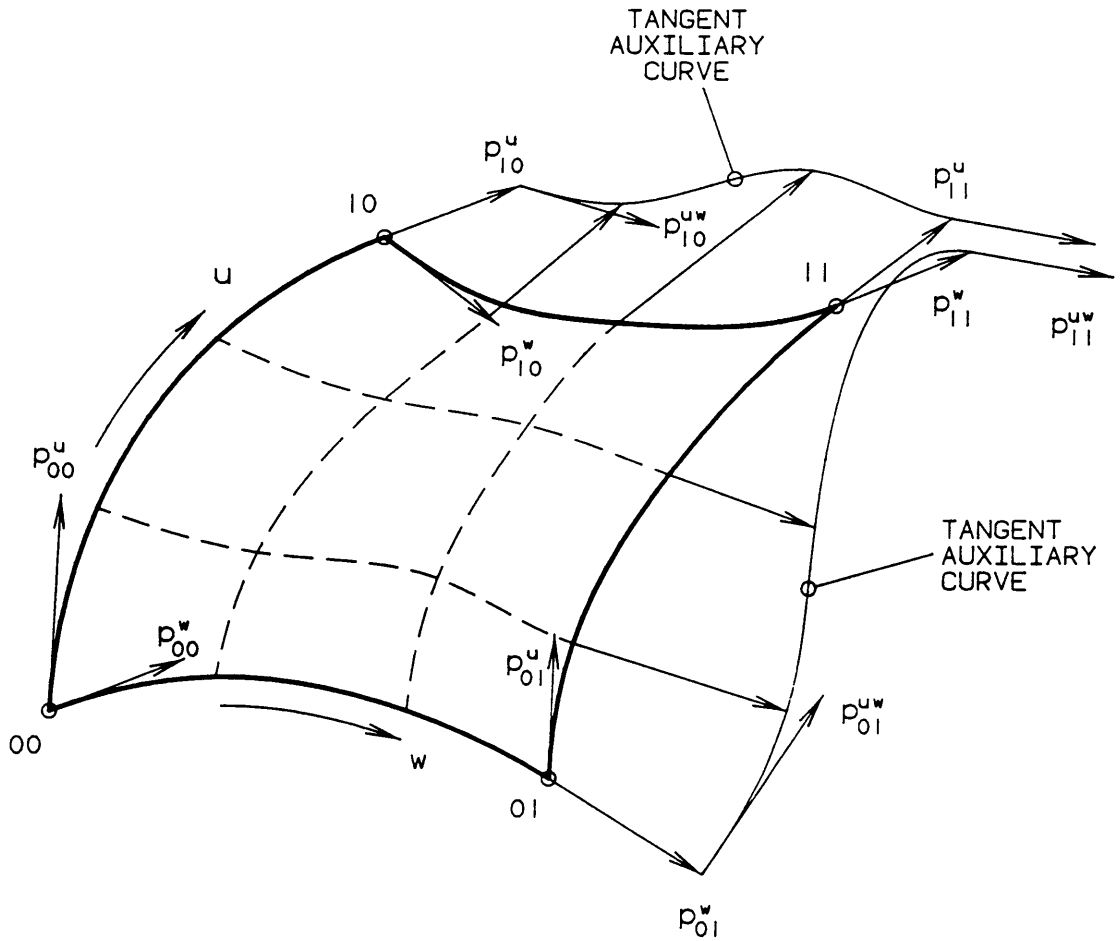


Figure 40. Geometric definition of a bicubic patch.

$\mathbf{p}_{ij}^{w}$, where $i,j = 0$ or 1, are the 1X3 tangent vectors at each of the four corner points in the w direction

$\mathbf{p}_{ij}^{uw}$, where $i,j = 0$ or 1, are the 1X3 twist vectors at each of the four corner points

In the above equation, the 4X4X3 geometric coefficient matrix [**B**] contains all of the point and tangent information necessary to fully define a surface patch. Below is a discussion of how the elements of [**B**] are calculated within ACSYNT/VPI.

In ACSYNT/VPI, the corner points of a bicubic patch are calculated using the parametric cross section equations discussed in "ACSYNT/VPI Component Cross Section Model" on page 115. For example, Figure 39 on page 122 showns three surfaces lofted between four elliptical cross-sections. Points 1 through 8 are calculated with the point form of Equation 7.4 on page 115 through Equation 7.6 on page 115. Likewise, the tangent vectors in the u direction (around the cross-section) are calculated using the tangent form of Equation 7.4 on page 115 through Equation 7.6 on page 115.

To calculate the tangent vectors in the w direction, the following finite difference approximations are used:

$$\hat{\mathbf{p}}_i^{w} = \frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{|\mathbf{p}_{i+1} - \mathbf{p}_i|} \quad \textbf{forward difference} \tag{7.16}$$

$$\hat{\mathbf{p}}_i^{w} = \frac{\mathbf{p}_{i+1} - \mathbf{p}_{i-1}}{|\mathbf{p}_{i+1} - \mathbf{p}_{i-1}|} \quad \textbf{central difference} \tag{7.17}$$

$$\hat{\mathbf{p}}_i^{w} = \frac{\mathbf{p}_i - \mathbf{p}_{i-1}}{|\mathbf{p}_i - \mathbf{p}_{i-1}|} \quad \textbf{backward difference} \tag{7.18}$$

where:

$\hat{\mathbf{p}}_i^{w}$ is the normalized tangent at point i in the w direction

$\mathbf{p}_{i-1}$ is the position vector of the $(i-1)^{st}$ point

$\mathbf{p}_i$ is the position vector of the i^{th} point

$\mathbf{p}_{i+1}$ is the position vector of the $(i+1)^{st}$ point

$|\mathbf{p}_j - \mathbf{p}_k|$ is the magnitude of the difference between points j and k

Forward and backward differences are used to calculate w tangents at the first and last component cross-sections respectively. All other w tangents are calculated using central differences.

For the sake of simplicity, ACSYNT/VPI assumes all surface twist vectors to be zero which results in a special form of the bicubic patch called an "F-patch" or Ferguson patch. F-patchs are easy to construct and modify and are well suited to conceptual design purposes.

Due to the large amount of data required to define a surface model, care was taken in the design of ACSYNT/VPI's surface database to ensure that duplication of data was minimal. Duplication is a concern because the corner points and tangents of adjacent surface patches are identical and only need to be stored once for each aircraft component. Therefore, the ACSYNT/VPI surface database does not store surface data in the form of the geometric coefficient matrix [B]. Instead, surface data is stored in a multi-dimensional array as a mesh of points referenced by component, cross-section, node, and derivative number as illustrated in Figure 41. The ACSYNT/VPI utility routine GTBMAT was written to generate coefficient matrices from the surface database.

PHIGS Display Model

The ACSYNT/VPI PHIGS display model was designed to render the surface geometry of an aircraft as wireframe and shaded images. Rendering is performed through the creation of PHIGS structures displayed on the ACSYNT/VPI workstation. The following two sections present the design of the PHIGS display model under the headings "Wireframe Image Generation"and "Shaded Image Generation".

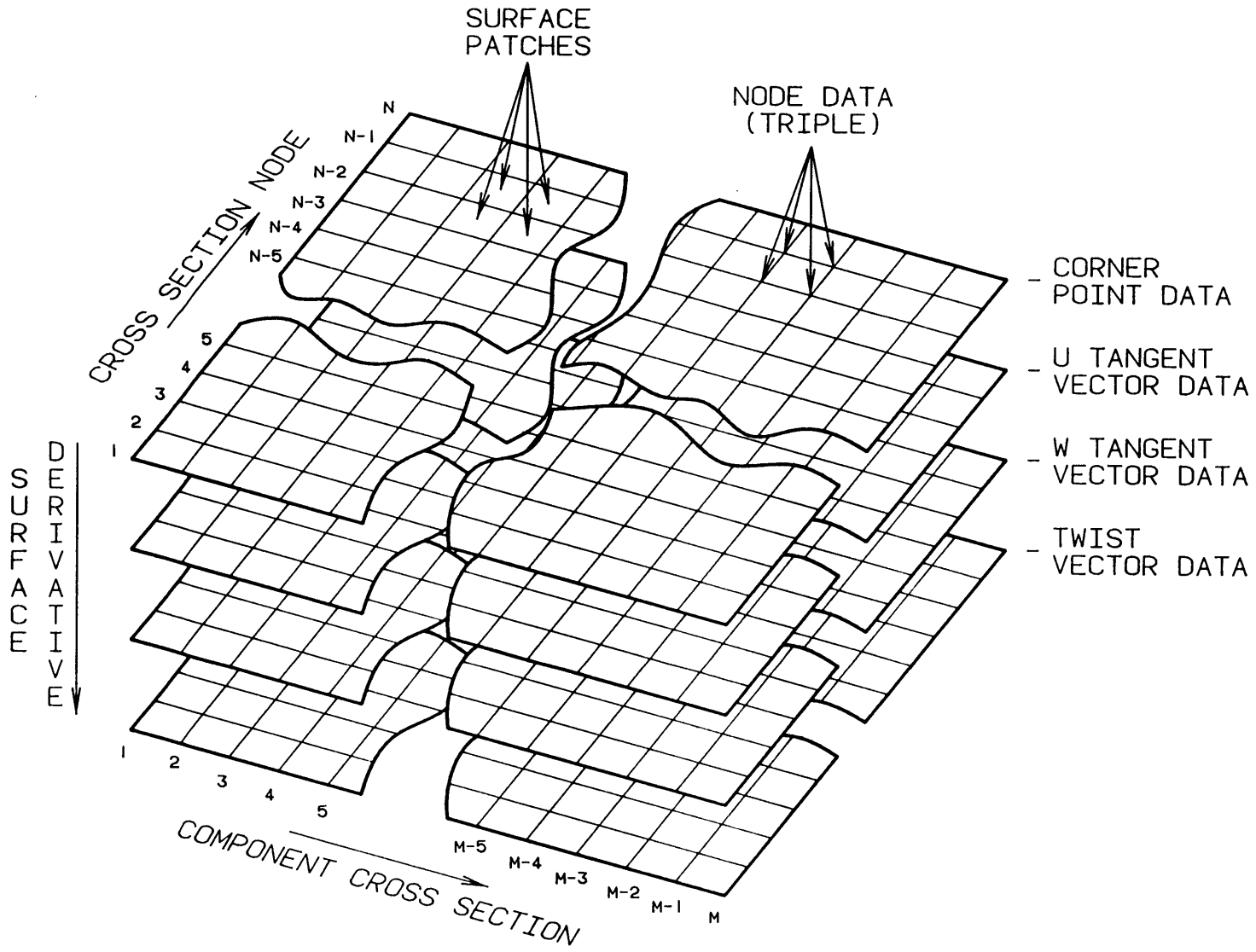


Figure 41. ACSYNT/VPI component surface database.

WIREFRAME IMAGE GENERATION

Overview

An ACSYNT/VPI aircraft wireframe image is displayed as a set of "polyline primitives" *approximating* curves on the surfaces of the aircraft. Note that the polyline coordinates are calculated using the ACSYNT/VPI surface model of the aircraft; no curves exist within ACSYNT/VPI. Each polyline is stored at the bottom of the hierarchical PHIGS structure network as illustrated in Figure 42 on page 129. In addition to the polyline primitives, the structure network contains component attributes such as colors, line types, and transformation matrices which effect the appearance and location of the polylines when displayed on a PHIGS workstation.

Figure 43 illustrates the hierarchical structure network for a single aircraft component. The network was designed to take advantage of components with both global and local symmetry. For example, the external wing-mounted fuel tank shown in Figure 44 on page 131 is locally symmetrical about a vertical plane passing through the tank's length and globally symmetrical about the vertical plane passing through the aircraft's fuselage. By using symmetry, the number of surface calculations required to display the tank is reduced by 3/4.

Below is a discussion of each type of structure in the hierarchy shown in Figure 43.

Root Structure

Through PHIGS, ACSYNT/VPI is capable of displaying multiple views of the same aircraft. For each view, the ACSYNT/VPI routine CRRTS creates a root structure which contains the identifier of the viewport in which the aircraft will be displayed. Following the view identifier, the root executes the attribute structure for each aircraft component.

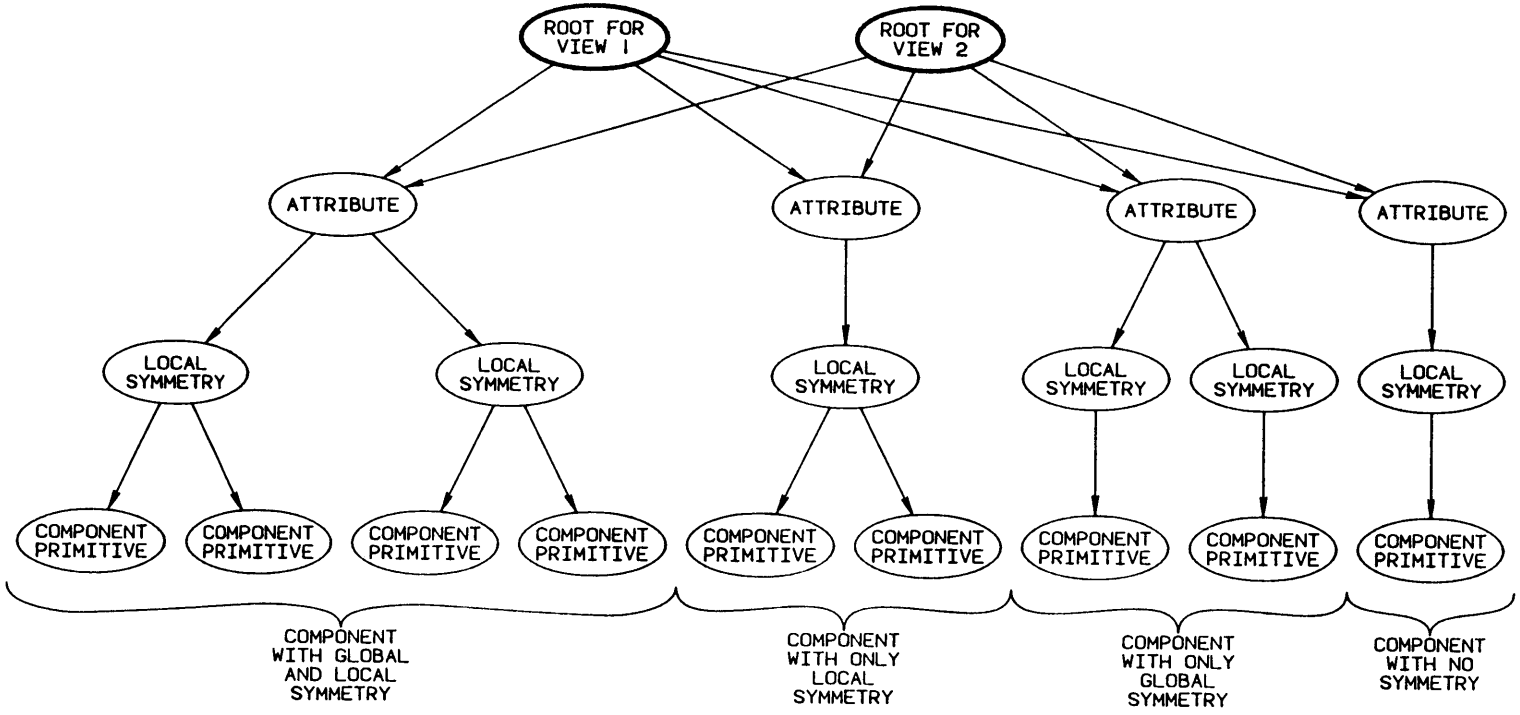


Figure 42. ACSYNT/VPI geometry structure network.

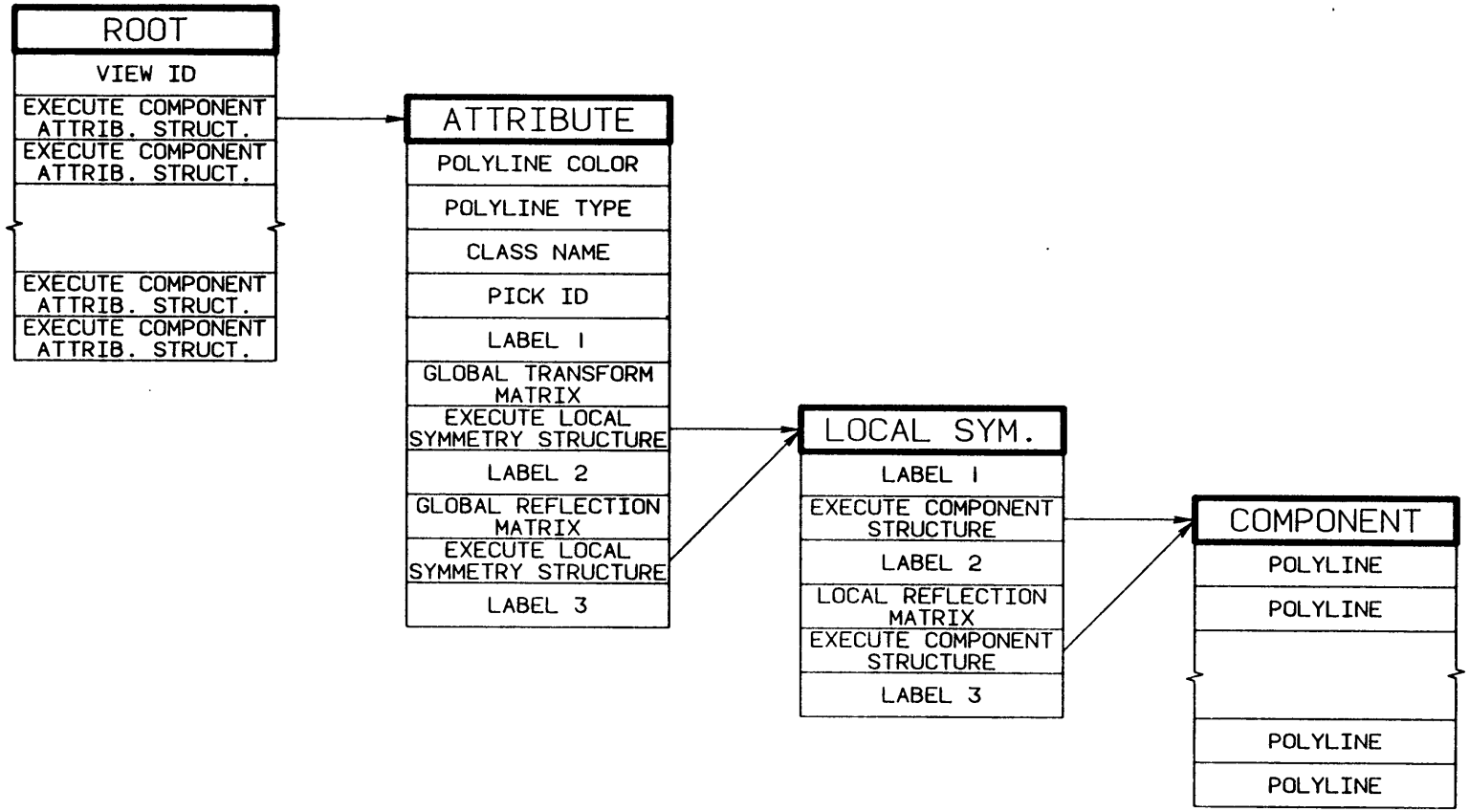


Figure 43. Structure network for a single aircraft component.

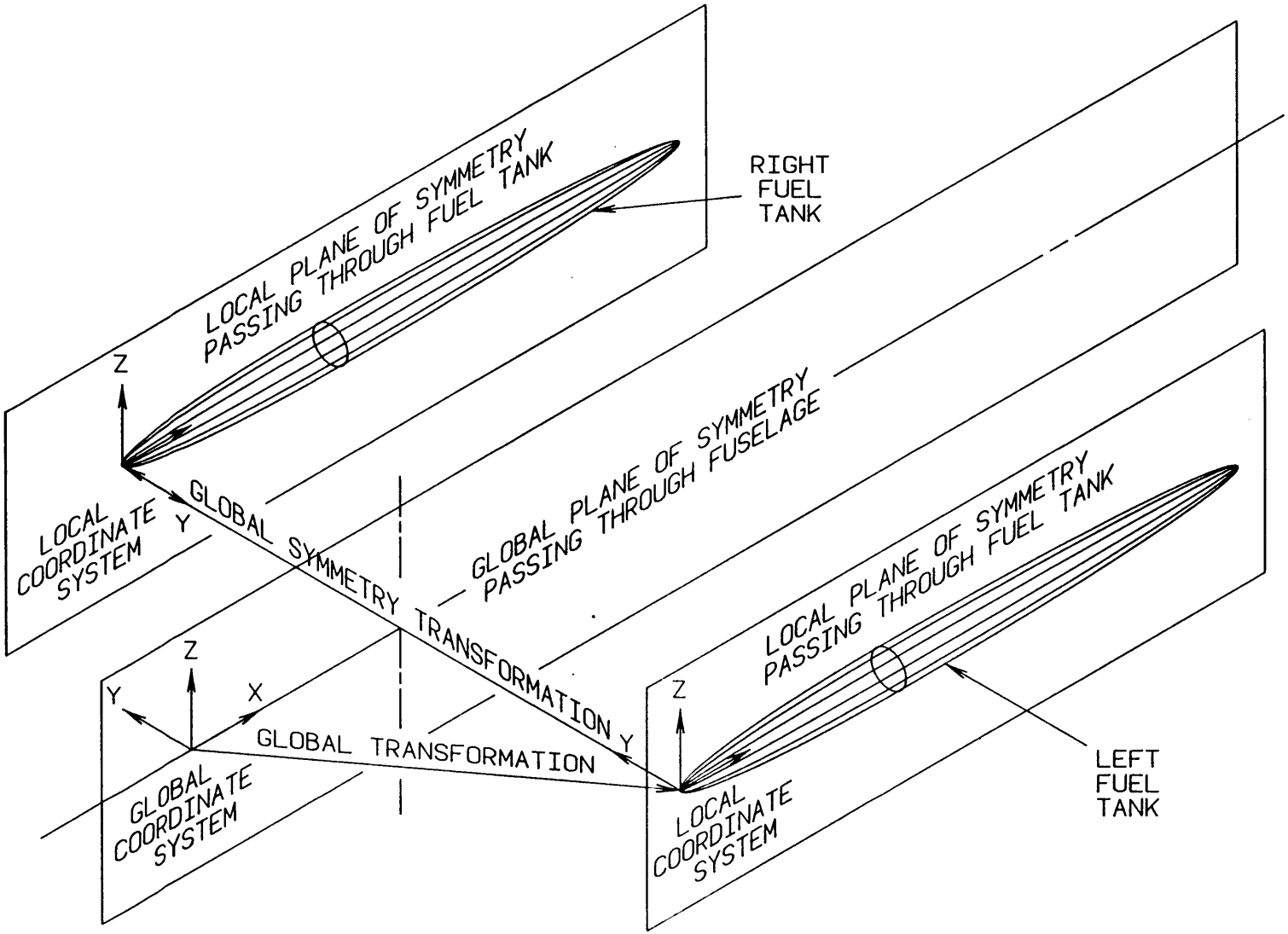


Figure 44. Component symmetry example.

Component Attribute Structures

For each component, the ACSYNT/VPI routine CRATTR creates a component attribute structure containing the following attributes:

polyline color index
controls the color of the component when displayed

polyline type
controls the line type of the component's wireframe image

component pick identifier
identifies the component when picked by the user

component class name
allows the component to be made pickable or invisible

The attributes listed above are all inherited by the local symmetry and component primitive structures.

The attribute structure also contains a 4X4 global transformation matrix of the following form:

$$[\mathbf{M}] = \begin{bmatrix} k_{11} & k_{12} & k_{13} & t_x \\ k_{21} & k_{22} & k_{23} & t_y \\ k_{31} & k_{32} & k_{33} & t_z \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{7.19}$$

where:

$[\mathbf{M}]$ = component transformation matrix

$k_{11} = cos(\alpha) \times \cos(\beta)$

$k_{12} = cos(\alpha) \times \sin(\beta) \times \sin(\gamma) - \sin(\alpha) \times \cos(\gamma)$

$k_{13} = cos(\alpha) \times \sin(\beta) \times \cos(\gamma) + \sin(\alpha) \times \sin(\gamma)$

$k_{11} = sin(\alpha) \times \cos(\beta)$

$k_{12} = sin(\alpha) \times \sin(\beta) \times \sin(\gamma) + \cos(\alpha) \times \cos(\gamma)$

$k_{12} = sin(\alpha) \times \sin(\beta) \times \cos(\gamma) - \cos(\alpha) \times \sin(\gamma)$

$k_{31} = -\sin(\beta)$

$k_{32} = cos(\beta) \times \sin(\gamma)$

$k_{33} = cos(\beta) \times \cos(\gamma)$

α = component rotation about the x-axis

β = component rotation about the y-axis

γ = rotation about the z-axis

t_x = translation along the x-axis

t_y = translation along the y-axis

t_z = translation along the z-axis

The global matrix translates and rotates a component relative to the global origin of the aircraft. For example, the cross-section and surface models for a wing-mounted fuel tank are both generated with respect to a local coordinate system located at the nose of the tank. In order to position the tank correctly with respect to the rest of the aircraft, the tank must be transformed as shown in Figure 44 on page 131.

In addition to the global transformation matrix, the attribute structure may also contain a 4X4 global symmetry matrix. A symmetry matrix will be used if the component is symmetric about one of the principal planes. A principal plane symmetry matrix takes one of the following forms:

$$
\begin{aligned}
[S] &= \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} && \textbf{for symmetry about the } x = 0 \textbf{ plane} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} && \textbf{for symmetry about the } y = 0 \textbf{ plane} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} && \textbf{for symmetry about the } z = 0 \textbf{ plane}
\end{aligned}
\tag{7.20}
$$

Example Figure 44 showns the global symmetry of a fuel tank about the $y = 0$ principal plane. Note that the tank is transformed before being reflected.

Following both the global transformation and symmetry matrices, the attribute structure contains elements which execute the component local symmetry structure discussed in the following section.

Local Symmetry Structure

When components are symmetrical about their own local coordinate system, ACSYNT/VPI generates only half of the component's surface model. To display both halves, the local symmetry structure executes the component primitive structure once without a symmetry matrix and once with a symmetry matrix, resulting in the display of the entire component. Local symmetry matrices have the same form as Equation 7.20 on page 133. For a component with both local and global symmetry, the component primitive structure will be executed four times producing the results shown in Figure 44 on page 131.

Component Primitive Structure

A component primitive structure is created by the routine CRCOMP and contains polylines whose end points are calculated on each component surface patch using the following procedure:

1. set the surface parameter u equal to zero for the $(i,j)^{th}$ component surface patch, where i is the surface counter in the cross-section direction and j is the surface counter around the cross-sections
2. set the surface parameter w equal to zero
3. calculate and store the surface point $\mathbf{p}(u,w)$ using Equation 7.15 on page 123

   NOTE: The surface matrix [**B**] for the $(i,j)^{th}$ bicubic surface patch is generated from the surface database by the ACSYNT/VPI routine GTBMAT.
4. increment w by $1/(N_s - 1)$ where N_s is large enough ($\cong$10) to produce a smooth looking curve
5. if w is less than or equal to 1 then go back to step 3. on page 134 otherwise proceed
6. draw a polyline using the PHIGS PPL3 function
7. increment u by $1/(N_u - 1)$ where N_u is the number of surface lines in the u direction specified in the component database
8. if u is less than or equal to 1 then go back to step 2. on page 134 otherwise proceed
9. perform steps 1. on page 134 through 8. on page 134 with u and w interchanged.

The above procedure generates a polyline grid on each surface patch as shown in Figure 45.

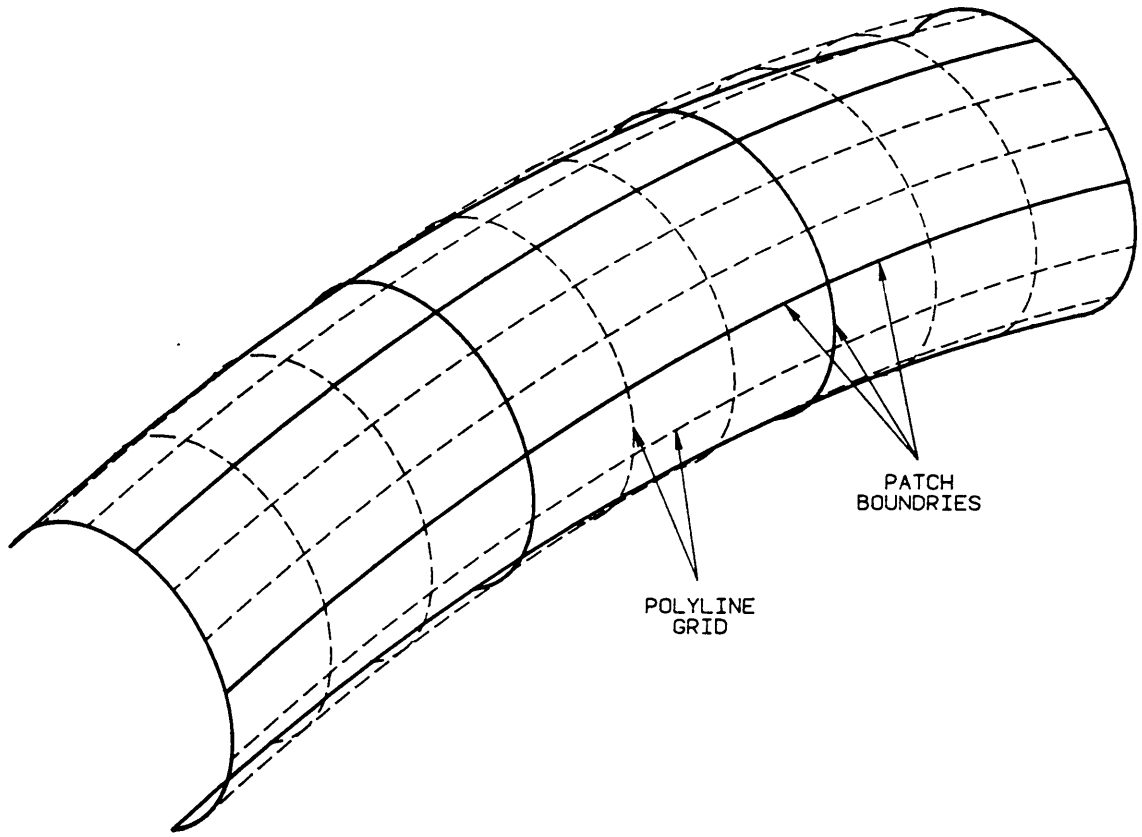


Figure 45. Polyline grid drawn on a surface patch.

When displayed, the polylines will inherit the color, line type, and transformations passed down from the attribute and local symmetry structures.

SHADED IMAGE GENERATION

Overview

ACSYNT/VPI is designed to perform shading with device independent PHIGS graphics calls. The ACSYNT/VPI shading model produces constant shaded images with specular highlights using ambient and multiple movable colored light sources. Sun model shading can also be performed in which each light source is assumed to emit parallel rays that do not decrease in intensity with distance.

Shaded image generation differs from wireframe generation in the following respects:

- polygons are generated on the component surfaces instead of polylines
- only one PHIGS structure is created during the shading process
- component symmetry and transformations are applied directly to the polygon vertices
- no transformation matrices appear in the shading structure.

Shading calculations are performed by the ACSYNT/VPI routine SHDCMP which produces a PHIGS structure containing polygons and polygon attributes for a specified set of aircraft components. As a subset of shading, ACSYNT/VPI can also perform component hidden surface elimination through the routine HIDCMP. HIDCMP is similar to SHDCMP but does not perform polygon color calculations.

The specific functions performed by SHDCMP are listed below:

- generates polygons from the surface model of the aircraft
- removes all polygons facing away from the observer

- sorts the polygons in order of decreasing distance from the observer's eye
- calculates the color of each polygon
- creates a PHIGS display structure containing the shaded polygons

Each of the SHDCMP functions listed above are described in the following sections.

Polygon Generation

Four sided polygons are generated on the surface of each aircraft component in a process called "tiling". Component tiling is the responsibility of the ACSYNT/VPI routine TILEIT.: The number of polygons generated per surface patch is $(N_u - 1) \times (N_w - 1)$ where N_u and N_w are the specified number of surface isoparametric curves in the u and w directions. N_u and N_w are determined by querying the component database for each component to be tiled.

TILEIT calculates each polygon vertex $\mathbf{p}(u,w)$ using Equation 7.15 on page 123 for bicubic patches. The vertex data for each polygon is stored in memory in a right-hand sense so that the polygon's normal will face away from the component as shown in Figure 46. TILEIT also transforms the polygon data for each component to the component's global location and orientation. And, in the case where a component is globally or locally symmetric, the component's polygons are duplicated by reflecting each polygon about the component's global and local planes of symmetry.

Polygon generation results in a polygon model of the aircraft geometry stored in an ACSYNT/VPI array which can be processed for back-face removal, sorting, color calculation, and display as described below.

Polygon Back-face Removal

The purpose of back-face removal is to reduce the number of polygons which must be processed for shading and hidden surface elimination. By eliminating back-face polygons, the time required

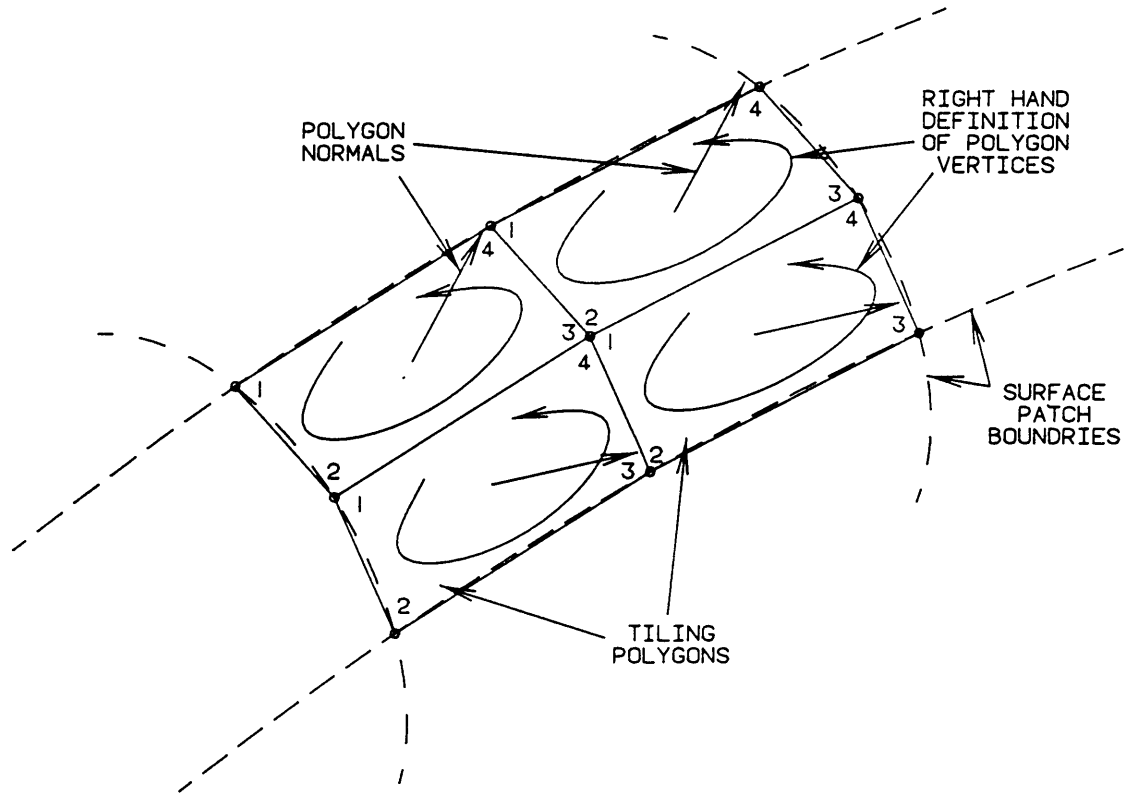


Figure 46. Tiling of a component surface patch showing right hand generation of vertex points.

to perform polygon sorting, coloring, and display is reduced, on the average, by one-half [15].: Conceptually, a polygon has two faces; a front face and a back face. ACSYNT/VPI defines the front face of a polygon by the right-hand ordering of the polygon's vertices as shown in Figure 46. All polygons with vertices that appear to an observer to be ordered in a right-hand sense are defined as front facing and visible to the observer. All other polygons are considered to be back facing and represent the back side of an object. Back facing polygons are considered invisible to an observer.

Mathematically, the two sides of a polygon define two vectors which, when crossed, define a vector normal to the polygon. As shown in Figure 47, the front and back faces of a right handed polygon can also be defined by the direction of the polygon's normal.
ACSYNT/VPI uses the following dot product to determine if a polygon is front or back facing:

$$s_i = \mathbf{n}_i \cdot (\mathbf{e} - \mathbf{c}_i) \tag{7.21}$$

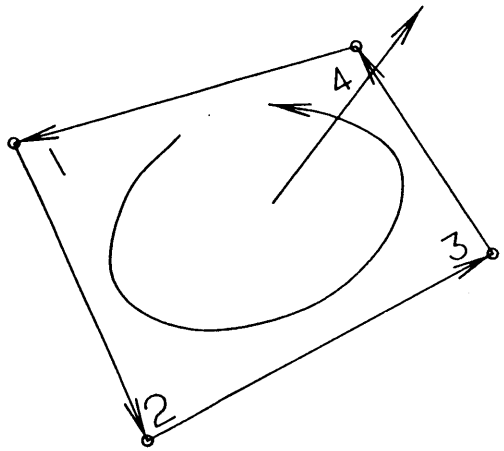
where:

s_i is the scalar value of the dot product
$\mathbf{n}_i$ is the normal vector of the i^{th} polygon
$\mathbf{e}$ is the vector defining the position of the observer's eye
$\mathbf{c}_i$ is the center location of the i^{th} polygon

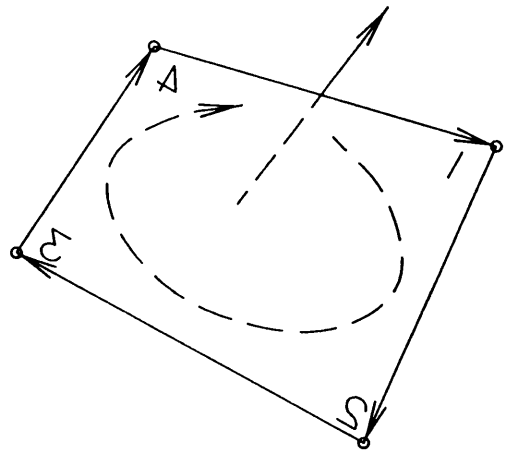
If the scalar s_i is negative, then the i^{th} polygon is facing away from the observer and can therefore be removed. Back-face removal is performed by the ACSYNT/VPI routine BAKFAC.

Polygon Sorting

Back-face removal will eliminate many of the polygons not visible to an observer but does not completely solve the problem of hidden surface elimination. For example, one object might obscure the view of another as shown in Figure 48 on page 142.



FRONT FACE OF POLYGON



BACK FACE OF POLYGON

Figure 47. Definition of polygon front and back faces by vector cross product.

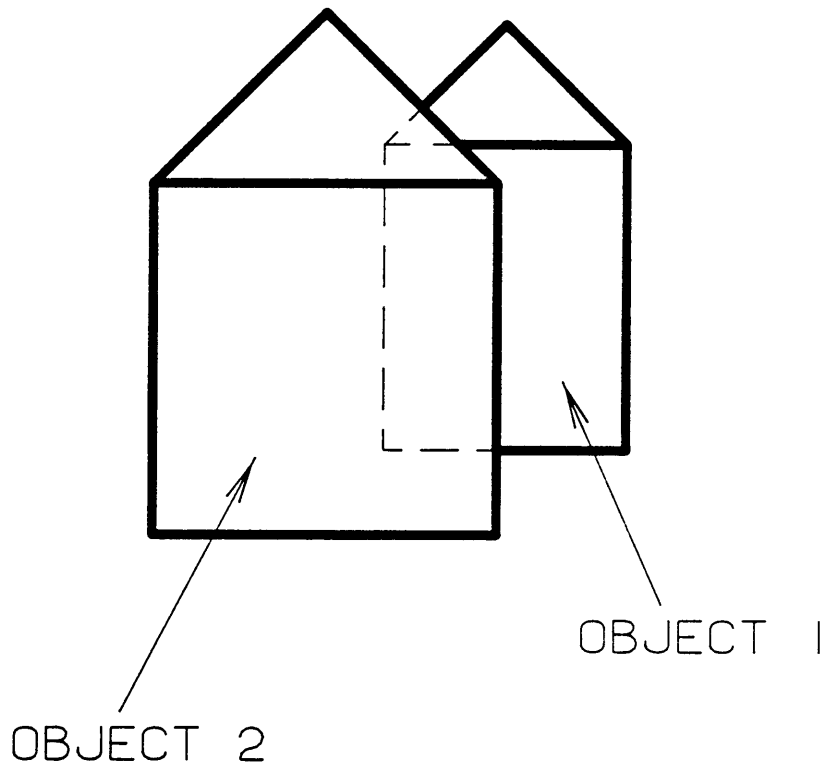


Figure 48. Example of one object obscuring the view of another.

The method used by ACSYNT/VPI to properly render hidden surface images is called the "painter's algorithm". The painter's algorithm consists of sorting all of the front facing polygons according to their distance from the observer's eye so that polygons closest to the observer are drawn last [15].

Polygon sorting is performed by the ACSYNT/VPI routine ZSORT which uses a simple bubble sorting algorithm to order the polygons. The output from ZSORT is a sorted list of polygons which can be either shaded or immediately displayed for hidden surface elimination.

Polygon Color Calculation

A shaded aircraft model is displayed as a set of color filled polygons. A polygon's color is a function of component color, light color, number of lights, surface characteristics, and ambient light. The ACSYNT/VPI routine SHDSET is responsible for calculating the light intensity of each polygon using the following equation which is based on the amount of reflected light reaching the eye of the observer [15]:

$$i = \mathbf{min}(1.0, \frac{br + (1.0 - b)(r(\mathbf{n} \cdot \mathbf{d}) + (1.0 - r)(\mathbf{n} \cdot \mathbf{h})^a)}{(1.0 - d)l}) \tag{7.22}$$

where:

i is the polygon's intensity

$\mathbf{n}$ is the polygon normal vector

$\mathbf{d}$ is the light source direction vector

$\mathbf{h}$ is the vector halfway between the eye and the light

$(\mathbf{n} \cdot \mathbf{d})$ is the cosine of the angle of incidence

$(\mathbf{n} \cdot \mathbf{h})$ is the cosine of the angle between the polygon normal vector and the vector halfway between the eye and the light

b is the ambient light fraction

r is the reflectivity of the polygon

a is the shininess of the polygon

l is the brightness of the light source

Because the IBM 5080 graphics workstation has a limited number of displayable colors, the calculated color intensity of each polygon must be approximated on the workstation. To map the calculated color intensity to the approximated color, each polygon is assigned a color index number by the ACSYNT/VPI routine COLIND. The polygon color index is used as an attribute of the polygon as described in the following section.

Polygon Display

The last step in the generation of a shaded or hidden surface image is the creation of a PHIGS structure containing all of the sorted front facing polygons produced by the previous steps. Polygons are inserted into the structure in the order in which they were sorted and are preceded by the following polygon attributes:

interior style:
specifies whether or not the polygon will be filled with color or be black

interior color index:
specifies the color index of the interior of the polygon calculated during the polygon color calculation step

edge flag:
turns on the edge lines drawn around a polygon when the polygon is used for hidden surface elimination

edge color index:
specifies the color of the polygon edge lines when the polygon is used for hidden surface elimination

When the structure is displayed on a PHIGS workstation a shaded image, hidden surface image, or combination will result. Example shaded and hidden surface images are presented in "Results and Recommendations" on page 145.

8.0 RESULTS AND RECOMMENDATIONS

Overview

This chapter contains a discussion of the functions ACSYNT/VPI can currently perform in light of the required functions specified in "Software Requirements" on page 43. Example photographs taken from the ACSYNT/VPI screen are provided to illustrate ACSYNT/VPI's current capabilities. Also, this chapter recommends how and when to implement some of the requirements not currently met by ACSYNT/VPI.

Functional Completeness

With ACSYNT/VPI, the user can perform the following functions:

- read a geometry common block file containing ACSYNT's definition of the aircraft geometry
- read an ACSYNT/VPI data file containing view and parametric aircraft component definitions
- display the three dimensional aircraft geometry as a surface wireframe image
- display the aircraft geometry using hidden surface elimination
- display the aircraft geometry as a constant shaded image including the following:
  - multiple movable color light sources
  - surface reflectivity
  - surface shininess (specularity)
  - surface color
  - sun model effects
  - ambient light
- display the aircraft in multiple windows
- interactively change viewing parameters including:

  - rotations
  - translations
  - scaling
  - window clipping
- interactively change aircraft component attributes such as:
  - component colors
  - visibility
  - surface approximation values for wireframe and shaded image generation
- interactively change the red, green, and blue color values for the first sixteen available colors

To support the functions listed above, code was also implemented to display interactive menus, acquire numeric input, and manage database memory as described in "Software Design" on page 62. The following paragraphs highlight ACSYNT/VPI's functions complete with example figures.

The main menu screen of ACSYNT/VPI is shown in Figure 49 on page 147 which is the first screen displayed by ACSYNT/VPI. The figure shows ACSYNT/VPI's screen layout and main menu. From the main menu, the user can select the "DESIGN" and "FILE" menu items to read in ACSYNT geometry common block data and ACSYNT/VPI geometry data respectively. Once geometry data has been read, the user can enter the geometry submenu by selecting the "GEOMETRY" menu item.

When the user selects the "GEOMETRY" menu item, ACSYNT/VPI generates a wireframe image of the current aircraft model. Figure 50 on page 148 shows the geometry menu and the wireframe model. Note that aircraft components such as the wing and fuselage, are displayed in different colors. From the geometry menu, the user can choose to modify the geometry, change how the image is displayed, produce a shaded or hidden surface image, and modify the view of the aircraft.

With the development version of ACSYNT/VPI, the user can modify the color and the number of surface approximation curves for each aircraft component. Both modifications affect the appearance of the aircraft's wireframe and shaded images. Figure 51 on page 149 shows the effect of changing both component color and the number of surface approximation curves.

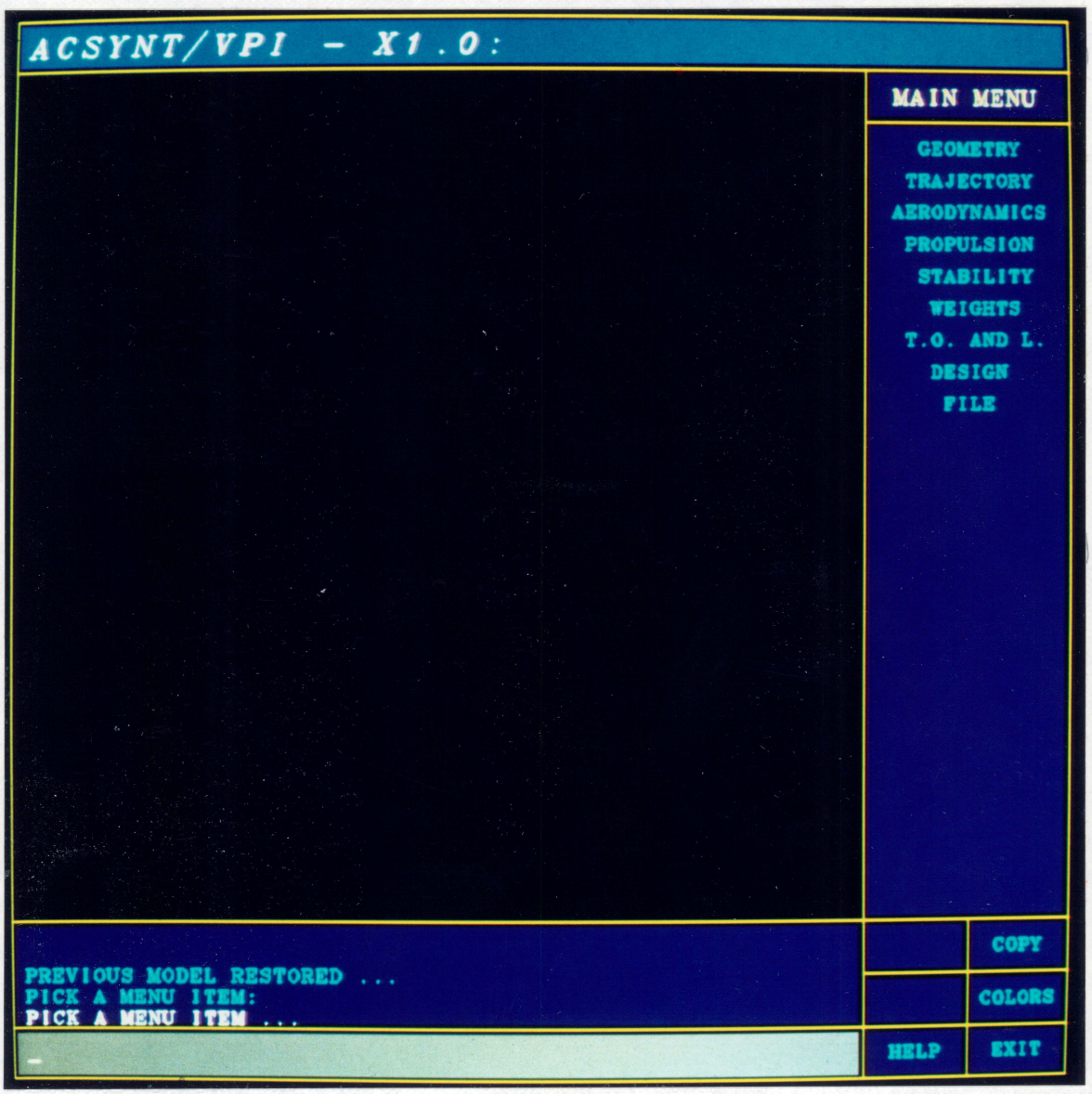


Figure 49. ACSYNT/VPI Main Menu Screen

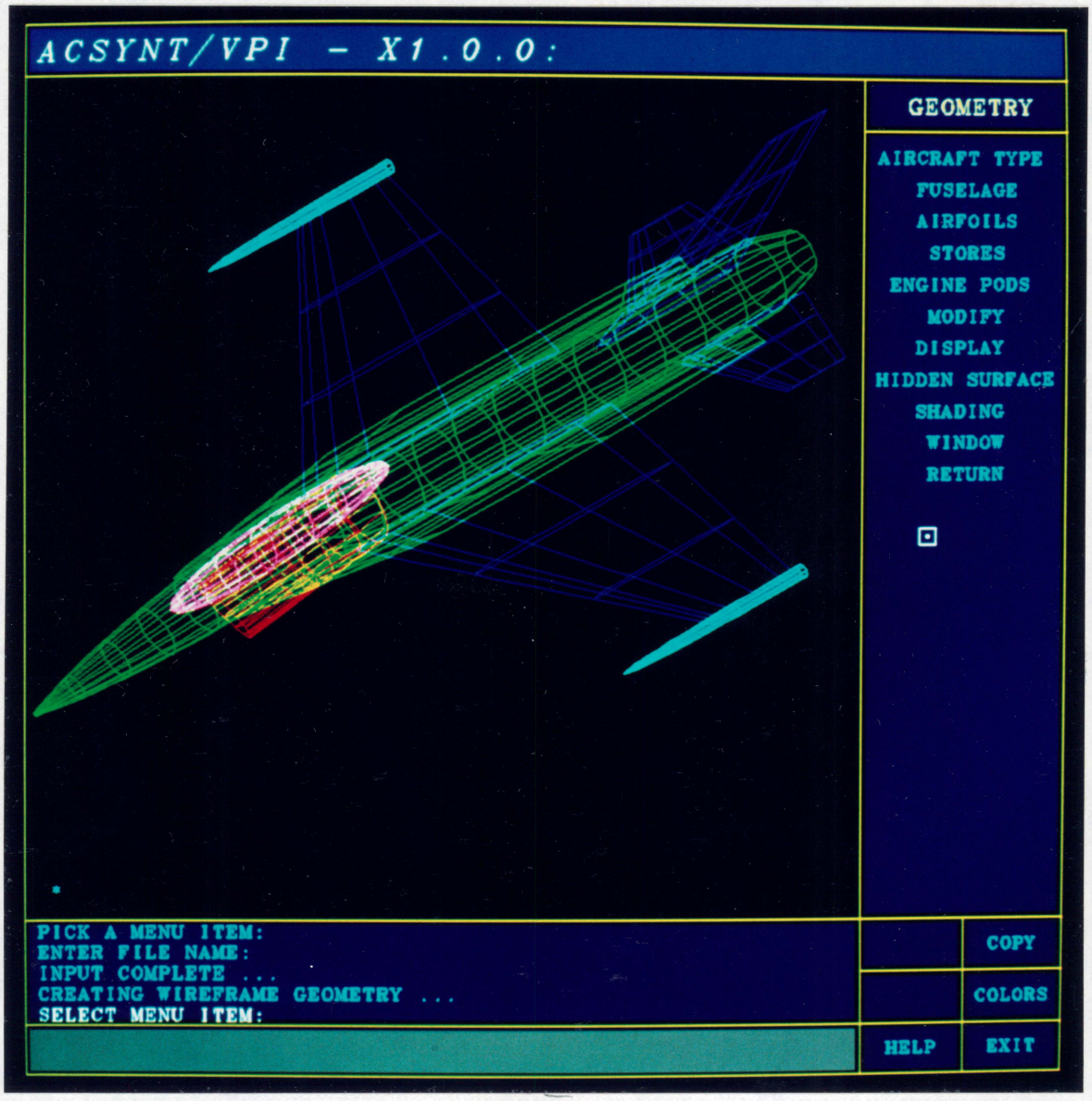


Figure 50. ACSYNT/VPI geometry menu showing F16 wireframe geometry.

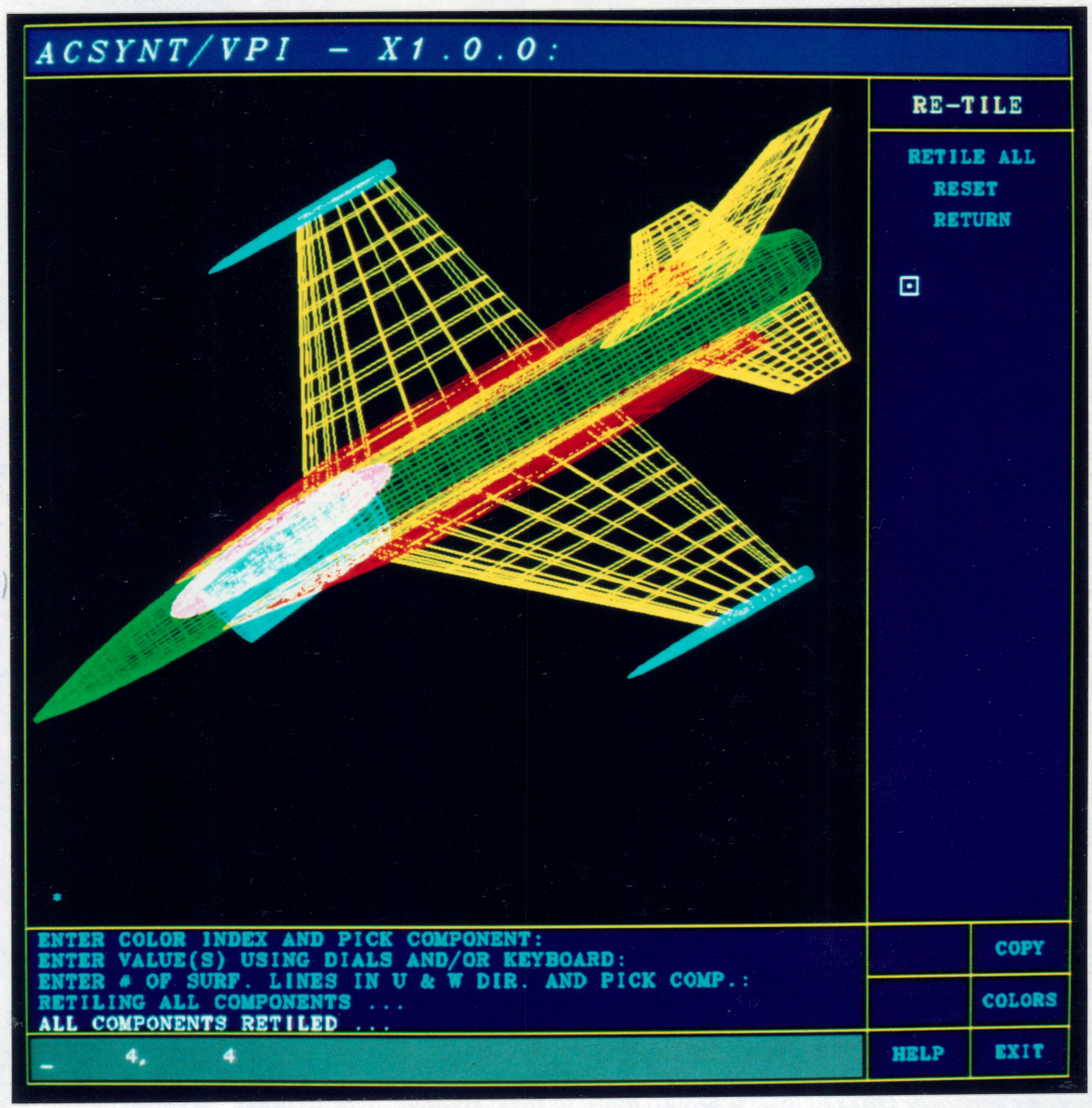


Figure 51. Modified component colors and number of approximation curves.

As stated in the list at the beginning of this section, ACSYNT/VPI is capable of producing shaded and hidden surface images of an ACSYNT aircraft model as illustrated in Figure 52 on page 151. Figure 52 contains an image of an F16 produced using a combination of shading and hidden surface elimination. The hidden surface elimination was applied only to the aircraft canopy. The image shown was produced using three point light sources, shining on a gray aircraft. All of the lights were white and appear as asterisks in the figure. The blue background in Figure 52 also illustrates the ability provided by ACSYNT/VPI to change screen colors.

NOTE: The jagged edges at the intersection of the aircraft components are due to the polygon approximation used during shading. In the future, the addition of fillets and the use of PHIGS+ will improve the appearance of shaded surface intersections.

A front view of the same F16 is shown in Figure 53 on page 152. The aircraft's yellow tint was produced by changing the color of the three light sources to yellow.

ACSYNT/VPI can also perform sun model shading, meaning that the light from each light source is assumed not to decrease in intensity with distance and all light rays are parallel. Sun model shading in combination with a yellow light source shining on a red aircraft, produces the effect shown in Figure 54 on page 153.

Figure 55 on page 154 illustrates ACSYNT/VPI's ability to display multiple views of the same aircraft model. The bottom two windows show the front and side views of the aircraft's red wireframe image and the top windows contain shaded images. An example of selective component shading is shown in the upper left window where only the wing and tail surfaces are shown.

Figure 55 also contains an example of ACSYNT/VPI's method of acquiring numeric input. The menu shown in the figure allows the user to change the fractional amount of red, green, and blue in any one of the first sixteen colors available on the graphics workstation. The color number and

Figure 52. Shaded image of gray aircraft using white light sources.

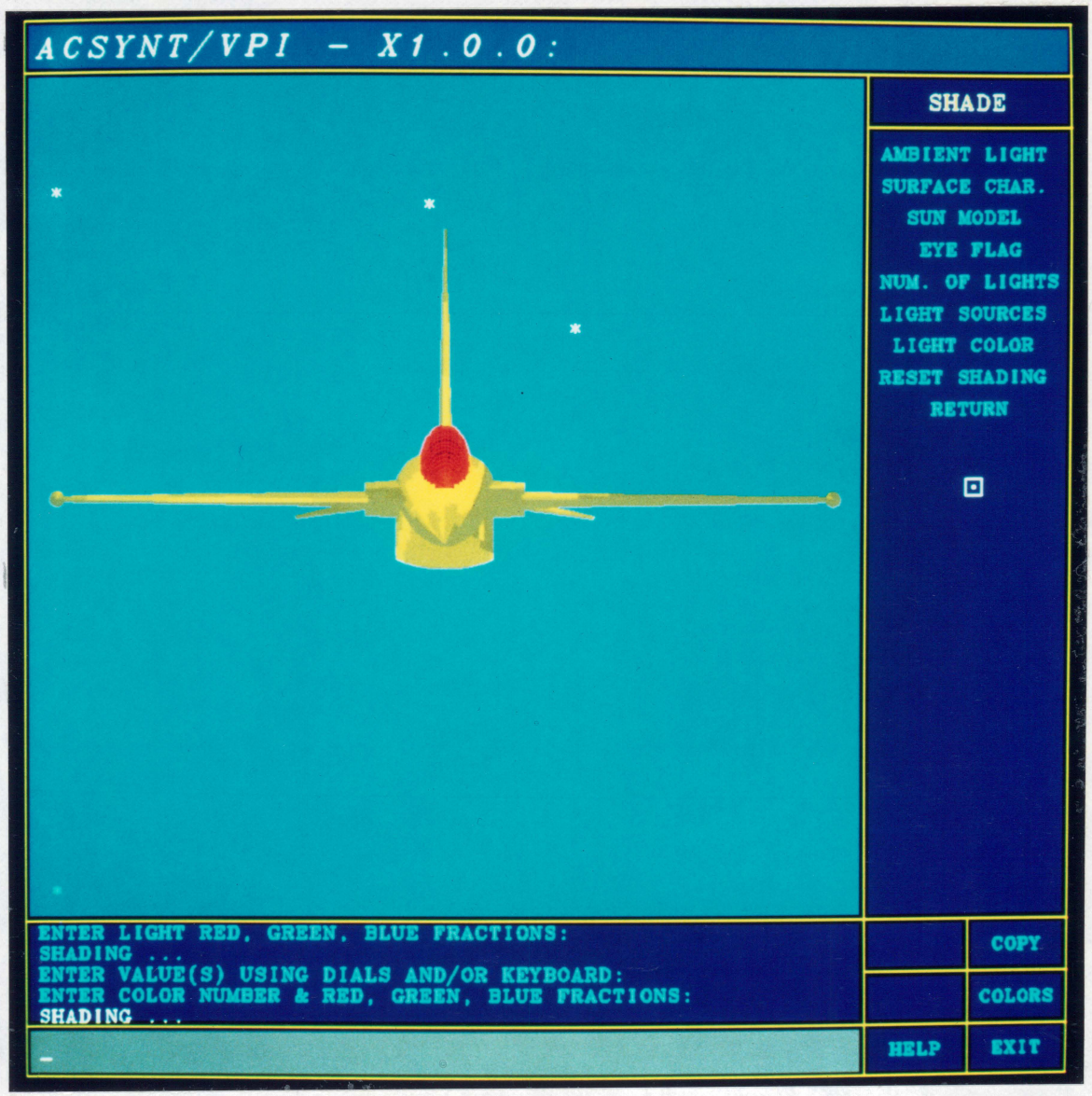


Figure 53. Front view of gray aircraft using yellow light sources.

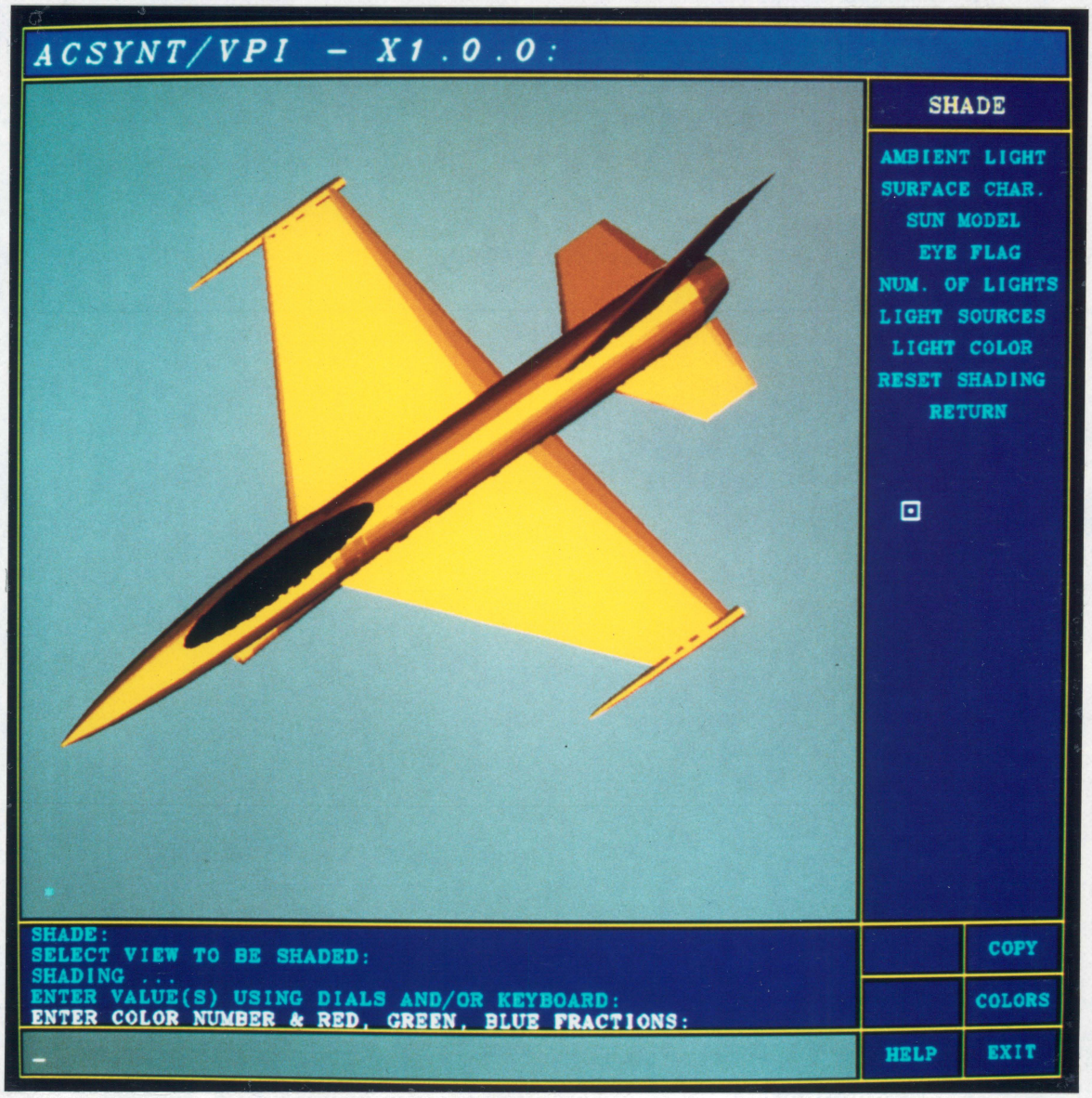


Figure 54. Yellow sun model shading on a red aircraft.

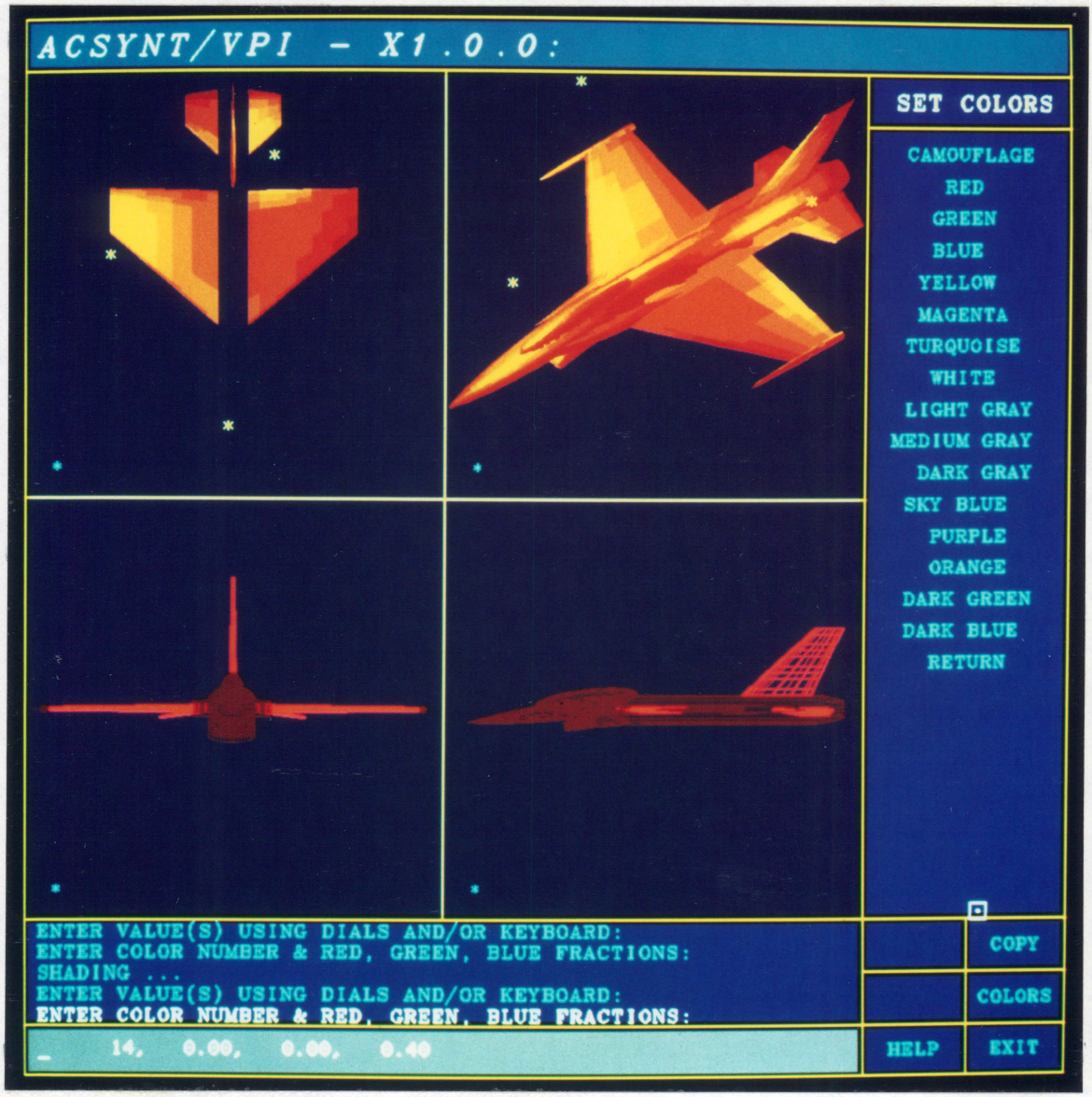


Figure 55. Multi-window display, selective shading, and color input.

the red, green, and blue color values are shown in the gray input area at the bottom of the screen. The user may either type over the values shown or adjust the numbers using valuator input dials as discussed in "Software Design" on page 62.

Figure 56 on page 156 and Figure 57 on page 157 illustrate ACSYNT/VPIs ability to model parametric aircraft components. The first figure shows the result of changing the wing sweep angle on an F16 and the use of symmetry to produce a V-tail. The second figure shows the effect of increasing the root chord on the wing of an F16.

Functional Recommendations

The software produced to date on the ACSYNT/VPI project provides the tools necessary to expand ACSYNT/VPI's functional capabilities. For example, software has been developed to generate and store a surface model representing an aircraft's geometry. In the future, the surface model can be used to calculate surface areas, cross sectional areas, volumes, and finite volume mesh points for drag calculations. The following list contains recommendations for future development of ACSYNT/VPI from a functional point of view:

- complete the ACSYNT/VPI implementation of parametric aircraft components
- develop interactive software to allow the user to create and modify parametric components
- begin work on the ACSYNT/VPI trajectory input module
- install ACSYNT/VPI on a standalone workstation
- begin the process of rewritting ACSYNT (not ACSYNT/VPI) from the top down
- merge redesigned version of ACSYNT with ACSYNT/VPI
- add a database manager to ACSYNT/VPI
- eliminate need for ACSYNT common blocks by passing data through the database manager
- incorporate window management software into ACSYNT/VPI such as the industry standard "x-windows" to improve the flexibility of ACSYNT/VPI's user interface

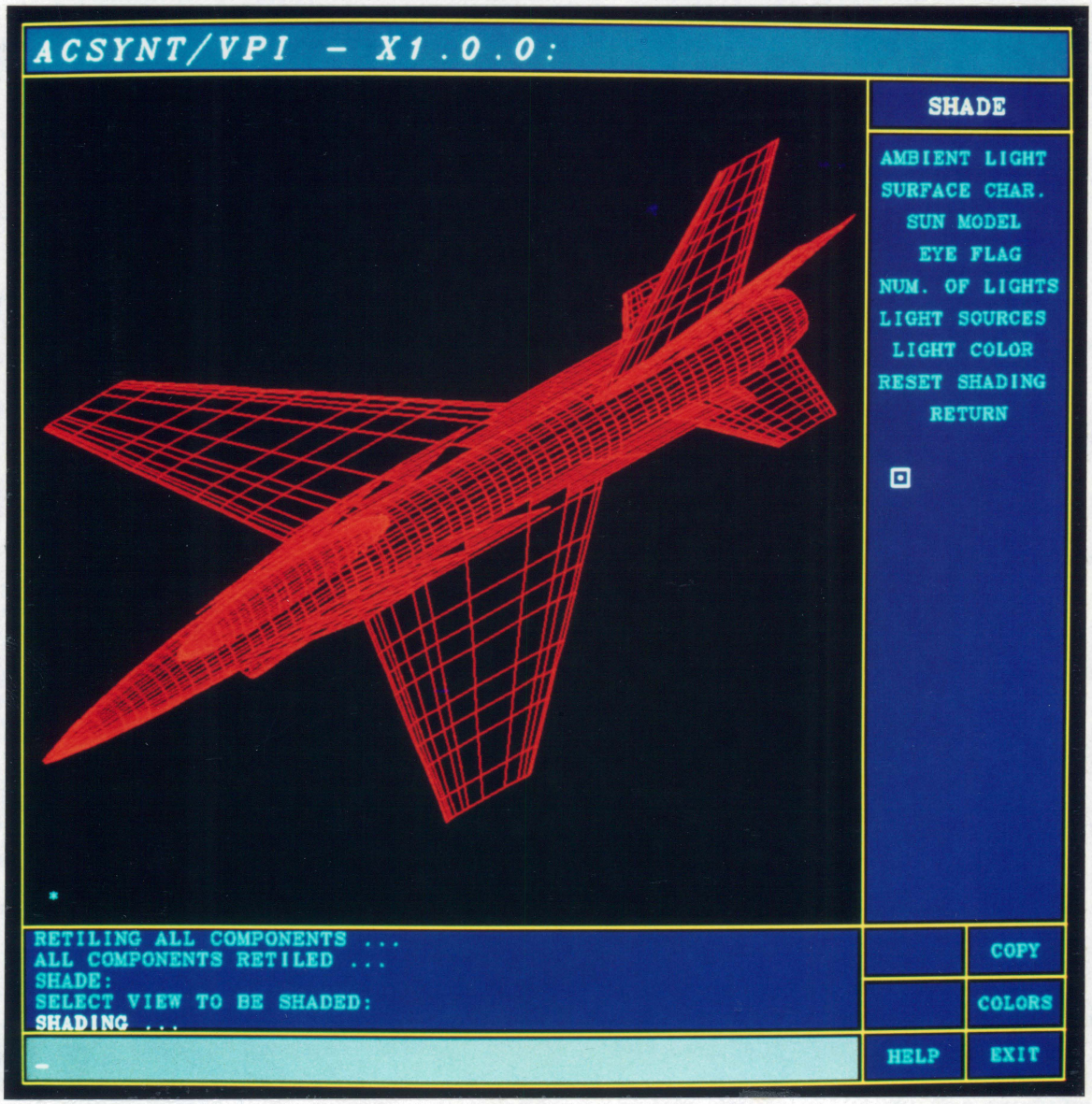


Figure 56. Hidden surface image of aircraft with swept forward wings and V-tail.

Figure 57. Shaded image of aircraft showing variability of wing component.

The most important long term recommendation of those listed above is that the original ACSYNT code be redesigned and used only as a functional example of what ACSYNT/VPI should be capable of performing. The low amount of documentation and bottom up development of ACSYNT will make maintaining and enhancing the ACSYNT/VPI code difficult. Unfortunately, rewriting the ACSYNT portion of ACSYNT/VPI will take time. Therefore, ACSYNT should be merged with ACSYNT/VPI but should be slowly replaced by well designed and documented code. The transition from original code to the redesigned code should include the addition of database management software as described below.

The addition of a database management software will help unify ACSYNT and ACSYNT/VPI into one common code with a common database. The inclusion of database software into ACSYNT/VPI would eliminate the need for the large common blocks and specialized database routines which are now part of ACSYNT and ACSYNT/VPI respectively. The database management software would not necessarily have to be from a commercial vendor but could be one written at Virginia Tech for use in computer aided design projects other than ACSYNT/VPI.

Although much work still remains in order for ACSYNT/VPI to be considered ready for production, a vast majority of the upper level design and development has been completed. Continuing the development of ACSYNT/VPI in a disciplined top-down fashion will lead to a powerful conceptual aircraft design system.

9.0 REFERENCES

1. **ACSYNT User's Manual**, NASA Ames Research Center, Moffett Field, California, 1987.

2. **The American Heritage Dictionary**, Houghton Mifflin Company, Boston, 1982.

3. Brody, H., "CAD Meets CAM", **High Technology**, pp. 12-18, May 1987.

4. Davis, S. J., H. Rosenstein, K. A. Stanzione, and J. S. Wisniewski, **User's Manual for HESCOMP, The Helicopter Sizing and Performance Computer Program**, Technical Report, distributed by Defense Technical Information Center, Defense Logistics Agency, 1979.

5. Schoen, A. H. and J. S. Wisniewski, **User's Manual for VASCOMP, The V/STOL Aircraft Sizing and Performance Computer Program**, Boeing Vertol Company, Report D8-0375, Volumn VI, Rev. 2, September 1973.

6. Myklebust, A., L. Lu, S. War, "Integration of a Helicopter Sizing Code with a Computer-Aided Design System," **Journal of the American Helicopter Society**, Volume 32, Number 4, October 1987.

7. Raymer, Daniel P., "CDS Grows New Muscles," **Astronautics and Aeronautics**, pp. 22-31, June 1982.

8. Myklebust, A., J. R. Mahan, S. G. Wampler, J. Sankar, and M. M. Grieshaber, **A CAD System for Automated Conceptual Aircraft Design**, Volume 1, Design, Algorithms, and User's Guide for ACSYNT/VPI (Version X1.0.0), VPI CAD/CAM Laboratory, forthcoming June 1988.

9. Myklebust, A., J. R. Mahan, S. G. Wampler, J. Sankar, and M. M. Grieshaber, **A CAD System for Automated Conceptual Aircraft Design**, Volume 2, Program Specification Flowcharts (Version X1.0.0), VPI CAD/CAM Laboratory, forthcoming June 1988.

10. Tausworthe, Robert C., **Standardized Development of Computer Software - Part I Methods**, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

11. Tausworthe, Robert C., **Standardized Development of Computer Software - Part II Standards**, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

12. Turner, Ray, **Software Engineering Methodology**, Reston Publishing Company, Inc. A Prentice-Hall Company, Reston, Virginia, 1984.

13. **Draft Proposed American National Standard Programmer's Hierarchical Interactive Graphics System (PHIGS)**, ANSI standard number X3.144-198x, American National Standards Institute, Oct. 23, 1986.

14. **Understanding graPHIGS**, document number SC33-8102-01, International Business Machines Corporation, September 1986.

15. Harrington, S., **Computer Graphics - A Programming Approach**, McGraw-Hill, Inc., New York, 1983.

16. **American National Standard Programming Language FORTRAN**, (also known as FORTRAN 77) ANSI standard number X3.9-1978, American National Standards Institute, 1978.

17. **Programming Languages - FORTRAN**, standard number 1539-1980, International Organization for Standardization (ISO), 1980.

18. **MicroVMS FORTRAN Programmer's Manual**, order number AA-Z212A-TE, Digital Equipment Corporation, September 1984.

19. **VS FORTRAN Version 2 General Information**, document number GC26-4219-1, International Business Machines Corporation, February 1986.

20. **IBM 5080 Graphics System: Principles of Operation**, IBM document number GA23-0134-0, International Business Machines Corporation, March 1984.

21. Liardet, M. H., "WORM: A Data Structuring System for FORTRAN", University of Edinburgh, Edinburgh UK, pp. 230-241.

22. Brown, Maxine D., Michael Heck, Editor, **Understanding PHIGS**, Megatek Corporation, 1985.

23. **graPHIGS Application Programming Notebook**, IBM document number GG24-3187-0, International Business Machines Corporation, 1987.

24. Mink, C., "Speaking Without Words", **The DEC Professional**, Vol. 6, No. 7, July 1987, pp. 32-42.

25. James, M. L, G. M. Smith, and J. C. Wolford, **Applied Numerical Methods for Digital Computation**, Second Edition, Harper & Row, New York, 1977.

26. **VS FORTRAN Version 2 Programming Guide**, document number SC26-4222-0, International Business Machines Corporation, February 1986.

27. Mortenson, M. E., **Geometric Modeling**, John Wiley & Sons, New York, 1985.

APPENDIX A. GLOSSARY

ACSYNT: an AirCraft SYNThesis program written at the NASA Ames Research Center, Moffett Field, California.

ACSYNT/VPI: the Virginia Tech device independent, interactive version of ACSYNT.

ANSI: the American National Standards Institute

attributes: see primitive attributes

bicubic patch: a surface patch bounded by cubic polynomial curves

C: a computer language

CADAM: "Computer Augmented Design And Manufacturing" (a CAD system developed by CADAM Incorporated and marketed by IBM).

color pallet: the available colors on a graphics device

common block: a data storage location in a FORTRAN program accessible from all portions of the program

COPES: "COntrol Program for Engineering Synthesis" used to control ACSYNT's synthesis of an aircraft model.

database: a storage area within a computer program used to organize data.

display list memory: graphics device memory used to store graphics primitive and attribute information

event mode input: one of three input mode defined by the PHIGS standard which allows input to a graphics program from multiple input devices at any time during the program's execution.

flowchart: a graphical representation for the definition, analysis, or soution of a problem in which symbols are used to represent operations [11].

FORTRAN 77: an ANSI standard for the FORTRAN (FORmula TRANslation) programming language

graPHIGS: IBM's implementation of the PHIGS graphics standard.

hierarchical: a relationship between PHIGS structures resulting in the ability of structures to invoke other structures.

hidden surface removal: the process of generating an image of an object by displaying only those surfaces not obscured by other surfaces

IBM 4341: a 16 megabyte mainframe produced and sold by IBM

interactive program: any computer program which prompts, waits for, and acts on user input

ISO: the International Standards Organization

isoparametric curve: a curve generated by holding one parameter of a surface constant while varying the other parameter

kilobyte: one thousand (kilo) bytes of computer memory

megabyte: one million (mega) bytes of computer memory.

menu module: a set of routines which display and control the use of an ACSYNT/VPI menu.

MicroVAX II: a micro-computer system developed and sold by the Digital Equipment Corporation

module: identifiable subportions of a program composed of instructions in a form acceptable to a computer prepared to achieve a certain result [11].

PASCAL: a programming language

patch: a curve bounded collection of three dimensional points whose coordinates are given by three mathematical functions of two parameters.

PHIGS: the proposed ANSI 3D graphics standard which stands for "Programmer's Hierarchical Interactive Graphics System".

PHIGS+: a set of proposed extensions to the PHIGS graphics standard which includes surface and curve primitives and shading.

pick device: a graphics standard concept for an input device used to select (pick) an object on a graphics screen (sometimes implemented as a light pen or cursor tablet).

polygon primitive: a graphics element which generates a closed polygon given the polygons vertex coordinates.

polyline primitive: a graphics element which generates a set of connected lines given the line end point coordinates.

primitives: a graphics standard concept for graphical elements such as lines, polygons, markers, and text.

primitive attributes: a graphics standard concept for the appearance of a graphics primitive such as color, size, line type, etc.

procedure: a set of programming statements forming an algorithm (see "module", "routine", and "subroutine") [11].

rendering: the process of generating a hidden surface or shaded image of an object.

routine: a program or program module that may have some general or frequent use [11].

shading: the process of producing a realistic image of an object by illuminated it with one or more light sources.

Software Development Library (SDL): central storage location for project software and documentation

string device: a graphics standard concept for an input device which returns a string of characters such as a type-writer style keyboard.

structure: a grouping of graphical elements and related information required to generate an image as defined in the PHIGS standard.

structure network: a set of hierarchically linked PHIGS structures

subroutine: a routine that can be invoked from another routine or subroutine (see routine) [11].

tier chart: a tree-graph representation of a program or menu-structure showing the hierarchical relationship between routines or menus [11].

tiling: the process of generating polygons on a surface patch

UNIX: a computer operating system developed by Bell Laboratories

valuator device: a graphics standard concept for an input device which provides a single scalar value such as dials and thumbwheels.

workstation: an abstract graphics device defined by the PHIGS graphics standard which forms a logical interface through which a computer program controls a physical graphics device.

Z-Buffer: graphics device memory which provides the device with the ability to perform hidden surface elimination

APPENDIX B. EXAMPLE ACSYNT INPUT FILE

The following is an ACSYNT input file supplied by the NASA Ames Research Center. The file contains ACSYNT input for a analyzing an F16 fighter. ACSYNT output for this input file is shown in "Appendix C. Example ACSYNT Output File" on page 166.

```
$ DATA BLOCK A
***** GENERAL DYNAMICS F-16A FIGHTER - AIR SUPERIORITY MISSION *****
$ DATA BLOCK B
         1
$ DATA BLOCK V
END
FIGHTER
    5    3    4  570  585    1    0    0    2   -1    7    0

    1    2    3    4    6
    1    2    6
    2    6    4    1
***** GENERAL DYNAMICS F-16A BLOCK 15 GEOMETRY *****
 $WING AR=3.0,AREA=300.0,SWEEP=40.0,TAPER=0.2275,TCROOT=0.04,
  TCTIP=0.04,XWING=0.468,ZROOT=0.00,KSWEEP=0,FDENWG=49.0,SWFACT=0.69,
  WFFRAC=0.326,
 $END
 $HTAIL AR=2.114,AREA=63.7,SWEEP=40.0,TAPER=0.39,TCROOT=0.06,
  TCTIP=0.035,XHTAIL=0.974,ZROOT=0.0,KSWEEP=0,SIZIT=F,SWFACT=1.57,
 $END
 $VTAIL AR=1.294,AREA=54.75,SWEEP=47.5,TAPER=0.437,TCROOT=0.053,
  TCTIP=0.030,XVTAIL=0.843,YROOT=0.0,ZROOT=1.0,KSWEEP=0,SIZIT=F,
  SWFACT=1.067,$END
 $FPOD SOD=-1.0,THETA=-90.0,X=0.20,SYMCOD=1,SWFACT=0.76,
 $END
 $FUS BDMAX=3.8,BODL=47.58,DRADAR=2.6,FRAB=3.0,FRN=4.8,LRADAR=5.9,
  SFFACT=1.466,WFUEL=11876.0,
 $END
 $CREW NCREW=1, $END
 $FUEL DEN=49.0,FRAC=0.68,WFUEL=11876.0, $END
 $FUEL DEN=49.0,FRAC=0.32,WFUEL=11876.0, $END
 $ENGINE N=1, $END
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
 $TRDATA DESLF=9.0,ULTLF=13.5,WFUEL=6972.0,IPSIZE=-2,
  RANGE=2000.0,
  TIMTO1=0.0,TIMTO2=3.9,
  WFEXT=4810.0,WFTRAP=74.0,
  FRFURE=0.0,
  LENVEL=.TRUE.,
  MMPROP=0
 $END
 $DATENV NALT=10, ENVALT=0.0,5000.0,10000.0,15000.0,20000.0,
  25000.0,30000.0,35000.0,40000.0,45000.0,
  NNZ=1,NZ=6.4,9*0.0,
```

```
  NPS=1,PSENV=-800.0,8*0.0,
  COMFUL=0.50, MLIMIT=2.2, IPSENV=1, IDEBUG=3, $END
      10
CLIMB          0.50               0.0   10000.0
    330.0        2        -1                  0                        1.00
CLIMB          0.60              -1.0   30000.0
    315.0        2        -1                  0                        1.00
CLIMB          0.87              -1.0   37600.0
                 2        -1                  0                        1.00
CRUISE        0.873     0.873    -1.0              250.
                 4                            0                        1.00
COMBAT         0.90      0.90  30000.0  30000.0              3.06
                 1         0                  0
ACCEL          0.90      1.60  30000.0  30000.0
                 1         0                  0                        1.00
COMBAT         1.20      1.20  30000.0  30000.0              3.25
                 1         0        1         0
CLIMB         0.873            20000.0  45250.0
                 2        -1                  0                        1.00
CRUISE        0.873     0.873    -1.0              250.
                 4                            0                        1.00
LOITER         0.43      0.43     0.0       0.0              20.0
                 4         0                  0                        1.00
***** GENERAL DYNAMICS F-16A BLOCK 15 AERODYNAMICS *****
 $IPDATA ABOSB=0.15,ALELJ=3,
  FCDO=2.34,FCDW=0.80,FCL=1.0,FCLH=0.85,
  FCDF=0.27,FINTF=1.5,
  FBLNT=0.0225,
  CGM=0.35,
  ESSF=0.40,
  ALIN=0.0,4.0,8.0,10.0,12.0,14.0,16.0,20.0,24.0,28.0,
  ALTV=10*30000.0,
  SMN=0.4,0.6,0.8,0.9,0.95,1.05,1.2,1.4,1.6,2.0,
  SMNSWP=0.0,0.4,0.6,0.8,0.9,1.0,1.2,1.4,1.6,2.0,
  IALF=0,IALP=2,ICOD=1,NALF=10,NMDTL=10,
  CDTNK=1.2,1.2,1.32,1.44,1.92,4.20,4.65,4.65,4.65,4.65,
  SMTANK=0.0,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,2.0,
 $END
****** F100 CYCLE ANALYSIS **************
2
 $LEWIS
  AM=0.,AUAENG=.005,AWAENG=.26,BA=.63,DIA1=3.875,ALTI=0.,POSA=29.92,
  FRBT=1.,FRPN=1.,MACH2=1.1,PRFD=1.,R10A=-1,
  P11P1=3.12,P2P1=25.,RLENG=1.5,SFADP=1.,SFADSP=1.,SFINSP=1.,T7M=3200.,
  TWAB=23294.,TWOAB=12410.,TWTO=1.,YREN=76.,T3=2567.,SCPR=1.28,
  IPRINT=0,KT5=2,KT7=2,NAB=5,ALTD=0.,5*30000.,
  XMACH=0.,.8,1.,1.2,1.4,1.6,
  IPR=-3,XMDES=2.2,XMT=1.0,
   $END
FIGHTER
 GENERAL DYNAMICS F-16A BLOCK 15 WEIGHTS, A/G MISSION
 $OPTS AFMACH=2.2,IDELT=1,KBODY=2,WGTO=29805.0,
 $END
 $FIXW
  WAIRC=267.0,WBODY=3422.0,WCREW=215.0,WE=3452.0,WELT=1143.0,
  WINST=107.0,WLG=988.0,WNA=655.0,WSC=716.0,WVT=330.0,WWING=1936.0,
  WHT=498.0,WEP=586.0,WFS=375.0,WCA=312.0,WGEAR=172.0,WHDP=311.0,
  WARM=598.0,WMISS=338.0,WBOMB=0.0,WBB1=339.0,WAMMUN=287.0,
  WETANK=1250.0,
 $END
```

APPENDIX C. EXAMPLE ACSYNT OUTPUT FILE

The following is an output file created by ACSYNT running on an IBM 4341 mainframe. The file contains output produced by processing the input file shown in "Appendix B. Example ACSYNT Input File" on page 164 using ACSYNT.

```
      CCCCCCC   OOOOOOO   PPPPPPP   EEEEEEE   SSSSSSS
      C         O     O   P     P   E         S
      C         O     O   P     P   E         S
      C         O     O   PPPPPPP   EEEE      SSSSSSS
      C         O     O   P         E               S
      C         O     O   P         E               S
      CCCCCCC   OOOOOOO   P         EEEEEEE   SSSSSSS

              C O N T R O L   P R O G R A M

                        F O R

        E N G I N E E R I N G   S Y N T H E S I S

                      T I T L E

      ***** GENERAL DYNAMICS F-16A FIGHTER - AIR SUPERIORITY MISSION *****
1     CARD IMAGES OF CONTROL DATA

      CARD                      IMAGE
0
        1)  $ DATA BLOCK A
        2)  GENERAL DYNAMICS F-16A FIGHTER - AIR SUPERIORITY MISSION *****
        3)  $ DATA BLOCK B
        4)          1
        5)  $ DATA BLOCK V
        6)  END
1     TITLE:
      ***** GENERAL DYNAMICS F-16A FIGHTER - AIR SUPERIORITY MISSION *****

      CONTROL PARAMETERS;
      CALCULATION CONTROL,                NCALC =    1
      NUMBER OF GLOBAL DESIGN VARIABLES,    NDV =    0
      NUMBER OF SENSITIVITY VARIABLES,      NSV =    0
      NUMBER OF FUNCTIONS IN TWO-SPACE,   N2VAR =    0
      NUMBER OF APPROXIMATING VAR.       NXAPRX =    0
      INPUT INFORMATION PRINT CODE,      IPNPUT =    0
      DEBUG PRINT CODE,                   IPDBG =    0

      CALCULATION CONTROL, NCALC
      VALUE    MEANING
        1      SINGLE ANALYSIS
        2      OPTIMIZATION
        3      SENSITIVITY
```

```
  4     TWO-VARIABLE FUNCTION SPACE
  5     OPTIMUM SENSITIVITY
  6     APPROXIMATE OPTIMIZATION

* * ESTIMATED DATA STORAGE REQUIREMENTS

          REAL                                  INTEGER
INPUT  EXECUTION  AVAILABLE     INPUT  EXECUTION  AVAILABLE
  9         9      5000           1         1      1000

    AAAAAAA   CCCCCCC   SSSSSSS   Y     Y   N     N   TTTTTTT
    A     A   C         S         Y   Y     NN    N      T
    A     A   C         S           Y Y     N N   N      T
    AAAAAAA   C         SSSSSSS     Y       N  N  N      T
    A     A   C               S     Y       N   N N      T
    A     A   C               S     Y       N    NN      T
    A     A   CCCCCCC   SSSSSSS     Y       N     N      T

          N A S A - A M E S  P R O G R A M

                      F O R

          A I R C R A F T  S Y N T H E S I S

                    T I T L E

***** GENERAL DYNAMICS F-16A FIGHTER - AIR SUPERIORITY MISSION *****

AIRCRAFT TYPE - FIGHTER
1  TITLE:
***** GENERAL DYNAMICS F-16A FIGHTER - AIR SUPERIORITY MISSION *****

AIRCRAFT TYPE - FIGHTER

CONTROL PARAMETERS:
READ CONTROL,                          MREAD =    5
EXECUTION CONTROL,                     MEXEC =    3
WRITE CONTROL,                        MWRITE =    4
NUMBER IDENTIFYING CONVERGENCE
  VARIABLE FOR CONVERGED VEHICLE,       IOBJ =  570
NUMBER IDENTIFYING COMPARISON
  VARIABLE FOR CONVERGED VEHICLE,       JOBJ =  585
SUMMARY OUTPUT PRINT CODE,             IPSUM =    1
GLOBAL ERROR PRINT CODE,              KGLOBP =    0
GLOBAL COMMON INITIALIZATION CODE,     INIT =    0
DEBUG PRINT CODE,                      IPDBG =    2
GLOBAL PLOT CONTROL,                   IGPLT =   -1
DATA TRANSFER INFORMATION FILE, IRDDTR   =    7
DATA TRANSFER INFORMATION PRINT, IPDTR   =    0

VEHICLE CONVERGENCE INFORMATION:
CONVERGENCE TOLERANCE, TOL = 0.10000E-03
ESTIM WCALC VS WEXT SLOPE  = 0.75000E+00
BOUNDING WEIGHT, WGMAX     = 0.50000E+06

MODULE IDENTIFICATION NUMBERS:
 NUMBER      MODULE
    1       GEOMETRY
    2       TRAJECTORY
    3       AERODYNAMICS
    4       PROPULSION
    5       STABILITY AND CONTROL
    6       WEIGHTS
    8       GLOBAL I/O
   11       SUMMARY OUTPUT
   14       TAKEOFF AND LANDING
```

```
     MODULES ARE CALLED FOR INPUT IN THE FOLLOWING ORDER:
         1    2    3    4    6

     MODULES ARE CALLED FOR EXECUTION IN THE FOLLOWING ORDER:
         1    2    6

     MODULES ARE CALLED FOR OUTPUT IN THE FOLLOWING ORDER:
         2    6    4    1
     CALLING MODULE NUMBER  1
1GEOMETRY DATA CARDS
  1***** GENERAL DYNAMICS F-16A BLOCK 15 GEOMETRY *****
  2 $WING AR=3.0,AREA=300.0,SWEEP=40.0,TAPER=0.2275,TCROOT=0.04,
  3  TCTIP=0.04,XWING=0.468,ZROOT=0.00,KSWEEP=0,FDENWG=49.0,SWFACT=0.69,
  4  WFFRAC=0.326,
  5 $END
  6 $HTAIL AR=2.114,AREA=63.7,SWEEP=40.0,TAPER=0.39,TCROOT=0.06,
  7  TCTIP=0.035,XHTAIL=0.974,ZROOT=0.0,KSWEEP=0,SIZIT=F,SWFACT=1.57,
  8 $END
  9 $VTAIL AR=1.294,AREA=54.75,SWEEP=47.5,TAPER=0.437,TCROOT=0.053,
 10  TCTIP=0.030,XVTAIL=0.843,YROOT=0.0,ZROOT=1.0,KSWEEP=0,SIZIT=F,
 11  SWFACT=1.067,$END
 12 $FPOD SOD=-1.0,THETA=-90.0,X=0.20,SYMCOD=1,SWFACT=0.76,
 13 $END
 14 $FUS BDMAX=3.8,BODL=47.58,DRADAR=2.6,FRAB=3.0,FRN=4.8,LRADAR=5.9,
 15  SFFACT=1.466,WFUEL=11876.0,
 16 $END
 17 $CREW NCREW=1, $END
 18 $FUEL DEN=49.0,FRAC=0.68,WFUEL=11876.0, $END
 19 $FUEL DEN=49.0,FRAC=0.32,WFUEL=11876.0, $END
 20 $ENGINE N=1, $END
0END OF GEOMETRY DATA CARDS
   20 CARDS READ
 --ICASE,BDMAX,BODL,FRATIO,ACDEN ARE 4      3.80     47.58     12.52      0.00
1NASA/AMES     A C S Y N T     INITIAL OUTPUT
     FUSELAGE DEFINITION
      X        R        AREA
    0.00     0.00      0.00
    0.95     0.34      0.37
    1.90     0.56      1.00
    2.85     0.75      1.76
    3.81     0.91      2.59
    4.76     1.05      3.47
    5.71     1.18      4.35
    6.66     1.29      5.23
    7.61     1.39      6.09
    8.56     1.48      6.91
    9.52     1.56      7.68
   10.47     1.63      8.40
   11.42     1.70      9.05
   12.37     1.75      9.63
   13.32     1.80     10.13
   14.27     1.83     10.55
   15.23     1.86     10.88
   16.18     1.88     11.12
   17.13     1.89     11.28
   18.08     1.90     11.34
   19.03     1.90     11.34
   19.98     1.90     11.34
   20.94     1.90     11.34
   21.89     1.90     11.34
   22.84     1.90     11.34
   23.79     1.90     11.34
   24.74     1.90     11.34
   25.69     1.90     11.34
   26.64     1.90     11.34
   27.60     1.90     11.34
   28.55     1.90     11.34
```

```
  29.50     1.90      11.34
  30.45     1.90      11.34
  31.40     1.90      11.34
  32.35     1.90      11.34
  33.31     1.90      11.34
  34.26     1.90      11.34
  35.21     1.90      11.34
  36.16     1.90      11.34
  37.11     1.91      11.52
  38.06     1.91      11.47
  39.02     1.89      11.17
  39.97     1.84      10.64
  40.92     1.77       9.87
  41.87     1.68       8.89
  42.82     1.57       7.71
  43.77     1.42       6.37
  44.73     1.25       4.89
  45.68     1.03       3.34
  46.63     0.76       1.79
  47.58     0.30       0.28
                    **FUSELAGE**
 MAX. DIAMETER.....      3.800
 FINENESS RATIO.....    12.521
 SURFACE AREA.......   688.777
 VOLUME.............   401.454
1NASA/AMES     A C S Y N T     INITIAL OUTPUT
          DIMENSIONS OF PLANAR SURFACES

                          WING   H.TAIL   V.TAIL   CANARD   UNITS
 OPLAN AREA.....         300.0     63.7     54.8      0.0  (SQ.FT.)
  SURFACE AREA.....      333.0    115.3    116.9      0.0  (SQ.FT.)
  VOLUME.............     95.1     13.3     11.6      0.0  (CU.FT.)
  SPAN...............  30.000   11.604    8.417    0.000  (FT.)
  L.E. SWEEP.........  40.000   40.000   47.500    0.000  (DEG.)
  C/4 SWEEP..........  32.183   32.273   43.226    0.000  (DEG.)
  T.E. SWEEP.........   0.000    0.500   25.909    0.000  (DEG.)
  ASPECT RATIO ......   3.000    2.114    1.294    0.000
  ROOT CHORD.........  16.293    7.898    9.053    0.000  (FT.)
  ROOT THICKNESS.....   7.821    5.687    5.758    0.000  (IN.)
  ROOT T/C ..........   0.040    0.060    0.053    0.000
  TIP CHORD..........   3.707    3.080    3.956    0.000  (FT.)
  TIP THICKNESS......   1.779    1.294    1.424    0.000  (IN.)
  TIP T/C ...........   0.040    0.035    0.030    0.000
  TAPER RATIO .......   0.228    0.390    0.437    0.000
  MEAN AERO CHORD....  11.320    5.842    6.837    0.000  (FT.)
  LE ROOT AT.........  18.194   38.445   31.057    0.000  (FT.)
  C/4 ROOT AT........  22.267   40.419   33.320    0.000  (FT.)
  TE ROOT AT.........  34.487   46.343   40.110    0.000  (FT.)
  LE M.A.C. AT.......  23.167   40.523   35.050    0.000  (FT.)
  C/4 M.A.C. AT......  25.997   41.983   36.759    0.000  (FT.)
  TE M.A.C. AT.......  34.487   46.365   41.887    0.000  (FT.)
  Y M.A.C. AT........   5.927    2.477    3.659    0.000
  LE TIP AT..........  30.781   43.313   40.242    0.000  (FT.)
  C/4 TIP AT.........  31.707   44.083   41.231    0.000  (FT.)
  TE TIP AT..........  34.487   46.394   44.199    0.000  (FT.)
  ELEVATION..........   0.000    0.000    1.900    0.000  (FT.)
  VOLUME COEFF. .....            0.300    0.065    0.000

          E X T E N S I O N S
                                STRAKE   REAR EXTENSION
  CENTROID LOCATION AT.....     0.000    0.000
  AREA.....................     0.000    0.000
  SWEEP ANGLE..............     0.000
0         F U E L   T A N K S
   TANK          VOLUME     WEIGHT    DENSITY
OWING              24.       1153.      49.00
  FUS#1           149.       7292.      49.00
```

```
 FUS#2            70.     3431.     49.00
 TOTAL                   11876.
0MISSION FUEL REQUIRED=    11876.
0AVAILABLE FUEL VOLUME IN WING=       72.

 AIRCRAFT WEIGHT  =      0.000 Lbs.
 AIRCRAFT VOLUME  =    521.435 Cu.Ft.
 AIRCRAFT DENSITY =      0.000 Lbs./Cu.Ft.
ICASE= 4
1                          TRAJECTORY INPUT

 TIMTO1 = 0.0       MENDUR =    1.    NCRUSE = 2    IPLOT = 0
 TIMTO2 = 3.9         QMAX =   700.   IPSIZE =-2    HMINP =     0.
 FRFURE = 0.00       XDESC =  80.0    IPSTO1 = 5    HMAXP = 40000.
  DESLF = 9.00      WKFUEL = 1.000    IPSTO2 = 2    DELHP =  4000.
  ULTLF =13.50      CRMACH = 0.800     IBREG = 0   SMMINP = 0.300
  RANGE =  2000.    WKLAND = 0.570    IENDUR = 0   SMMAXP = 0.900
  WFUEL =  6972.     FLFAC = 0.600    IPRINT = 0    DELMP = 0.100
  WFEXT =  4810.     DECEL = 0.250    KERROR = 2   WCOMBP =     0.70
 WFTRAP =    74.    NLEGCL = 0        NLEGCR = 0   NLEGLO = 0
 FWGMAX = 1.200        TOL = 0.001    MILCOM = 0    NMISS = 1
                                                    NCODE = 0
  MMPROP = 0

                          MISSION 1

        MACH  MACH   ALT      ALT   HORIZ          NO.   VIND   IP   IX  IW
 PHASE  START  END  START      END    DIST  TIME TURNS WKFUEL IPRT  IB  IA

 CLIMB   0.50 0.00      0.  10000.     0.0   0.0   0.0  330.0    2   -1   0
                                                          1.000    0    0   0
 CLIMB   0.60 0.00     -1.  30000.     0.0   0.0   0.0  315.0    2   -1   0
                                                          1.000    0    0   0
 CLIMB   0.87 0.00     -1.  37600.     0.0   0.0   0.0    0.0    2   -1   0
                                                          1.000    0    0   0
 CRUISE  0.87 0.87     -1.      0.   250.0   0.0   0.0    0.0    4    0   0
                                                          1.000    0    0   0
 COMBAT  0.90 0.90  30000.  30000.     0.0   3.1   0.0    0.0    1    0   0
                                                          1.000    0    0   0
 ACCEL   0.90 1.60  30000.  30000.     0.0   0.0   0.0    0.0    1    0   0
                                                          1.000    0    0   0
 COMBAT  1.20 1.20  30000.  30000.     0.0   3.3   0.0    0.0    1    0   1
                                                          1.000    0    0   0
 CLIMB   0.87 0.00  20000.  45250.     0.0   0.0   0.0    0.0    2   -1   0
                                                          1.000    0    0   0
 CRUISE  0.87 0.87     -1.      0.   250.0   0.0   0.0    0.0    4    0   0
                                                          1.000    0    0   0
 LOITER  0.43 0.43      0.      0.     0.0  20.0   0.0    0.0    4    0   0
                                                          1.000    0    0   0

     CALLING MODULE NUMBER  3
1                   AERODYNAMIC INPUT DATA

              ***** GENERAL DYNAMICS F-16A BLOCK 15 AERODYNAMICS *****

     FACTORS                INTERGERS AND LOGICALS                    REALS

 FALEL  =   1.000    ALELJ  =      3  NALF   =     10    ABOSB  =   0.150  SHK    =
25.000
 FBLNT  =   0.023    IALF   =      0  NMDTL  =     10    ALMAX  =  30.000  SM     =
0.000
 FCD    =   1.000    IALP   =      2  ECHOIN =      1    AMC    =  40.000  SMNDR  =
0.900
 FCDF   =   0.270    IAXE   =      0  ECHOUT =      0    BTEF   =   0.000  SWPMAX =
60.000
 FCDL   =   1.000    ICDO   =      0  ELLIPC =      F    CGM    =   0.350  SWPMIN =
0.000
```

```
 FCDLH  =   1.000    ICOD   =       1  ELLIPH =       F    CLLAND =  -1.000  YO     =
0.000
 FCDO   =   2.340    INORM  =       1  ELLIPW =       F    CLTO   =  -1.000  ZCG    =
0.000
 FCDW   =   0.800    INTM   =       0                       CSF    =   0.000
 FCDWB  =   1.000    IPBLNT =       0                       DELFLD =  45.000
 FCL    =   1.000    IPENG  =       0                       DELFTO =  45.000
 FCLH   =   0.850    IPFRIC =       0                       DELLED =  30.000
 FDNOSE =   1.000    IPTEXT =       0                       DELLTO =  30.000
 FENG   =   1.000    IPINTF =       0                       ESSF   =   0.400
 FEXP   =   0.780    IPLIFT =       0                       EXA    =   0.667
 FINTF  =   1.500    IPLOT  =       0                       EXC    =   2.000
 FLBCOR =   1.000    IPMIN  =       0                       IT     =   0.000
 FLD    =   1.000    IPWAVE =       0                       LDLAND =  -1.000
 FLECOR =   1.000    ISMNDR =       0                       LDTO   =  -1.000
 FLNOSE =   1.000    ISUPCR =       0                       MACHN  =   0.750
 FLSCOR =   1.000    ITRAP  =       0                       QHOQI  =   0.000
 FMDR   =   1.000    IVCAM  =       0                       RCLMAX =   1.000
 FSEP   =   1.000    IXCD   =       0                       ROC    =   0.015
 FTRIM  =   1.000    KERROR =       0                       SFWF   =   0.000
```

ARRAYS

```
 ALIN   =   0.000   4.000   8.000  10.000  12.000  14.000  16.000  20.000  24.000  28.0
00
 ALTV   =  30000.  30000.  30000.  30000.  30000.  30000.  30000.  30000.  30000.  3000
0.
 CDBMB  =  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.00
00
 CDEXTR =  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.00
00
 CDONPT =  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.00
00
 CDSTR  =  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.00
00
 CDTNK  =  1.2000  1.2000  1.3200  1.4400  1.9200  4.2000  4.6500  4.6500  4.6500  4.65
00
 CLINPT =   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.0
00
 CLO    =   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.0
00
 CMO    =   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.0
00
 FCDLRA =   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.0
00
 FCLRA  =   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.0
00
 FLDM   =   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.0
00
 FVCAM  =   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.0
00
 ISTRS  =       0       0       0       0       0       0       0       0       0
0
 ITB    =       0       0       0       0       0       0       0       0       0
0
 ITS    =       0       0       0       0       0       0       0       0       0
0
 ITRIM  =       1       1       1       1       1       1       1       1       1
1
 SMEXTR =   0.000   0.200   0.400   0.600   0.800   1.000   1.200   1.400   1.600   1.8
00
 SMN    =   0.400   0.600   0.800   0.900   0.950   1.050   1.200   1.400   1.600   2.0
00
 SMNBMB =   0.000   0.200   0.400   0.600   0.800   1.000   1.200   1.400   1.600   1.8
00
```

```
 SMNCDO =   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.0
00
 SMNSWP =   0.000   0.400   0.600   0.800   0.900   1.000   1.200   1.400   1.600   2.0
00
 SMSTRS =   0.000   0.200   0.400   0.600   0.800   1.000   1.200   1.400   1.600   1.8
00
 SMTANK =   0.000   0.600   0.700   0.800   0.900   1.000   1.100   1.200   1.300   2.0
00

     CALLING MODULE NUMBER  4
1                       PROPULSION INPUT
                         VERSION 04-76

 ****** F100 CYCLE ANALYSIS **************
  AENDIA =     0.000  AENLE  =     0.000  AENWT  =     0.000
  ALTI   =     0.000  AM     =     0.000  ATURB  =     0.328
  AENTW  =     0.000
  AUAENG =     0.005  AWAENG =     0.260  BA     =     0.630
  DELPR  =     0.000  DELT57 =   100.000  DEPWCC =     2.000
  DIA1   =     3.875  EAB1   =     0.750  EB1    =     0.940
  ED1    =     0.750  ETAC1  =     0.800  ETAF1  =     0.820
  ETAT1  =     0.900  HTR    =     0.600  HVF    = 18600.000
  MACH1  =     1.000  MACH2  =     1.100  PCDFAC =     1.000
  POSA   =    29.920  PRFD   =     1.000  PWCC   =   100.000
  P11P1  =     3.120  P2P1   =    25.000  R10A   =    -1.000
  R32    =     0.940  R54    =     0.920  R54N   =     0.880
  R711   =     0.980  R711N  =     0.980  SCPR   =     1.280
  SFADP  =     1.000  SFADSP =     1.000  SFAUXP =     0.000
  SFBEP  =     0.000  SFBPP  =     0.000  SFBTP  =     0.000
  SFDIVP =     0.000  SFINSP =     1.000  SFIP   =     0.000
  SFSFC1 =     1.000  SFSFC2 =     1.000  SFSFC3 =     1.000
  SM1    =     0.500  SODG   =     1.000  TOSA   =   412.000
  TR     =   520.000  TWAB   =    23294.  TWOAB  =    12410.
  TWTO   =     1.000  T3     =     2567.  T5M    =     3400.
  T51    =        0.  T7M    =     3200.  T71    =        0.
  VC1    =     0.980  WCWA1  =     0.018  XMDES  =     2.200
  XMT    =     1.000  YREN   =       76.  FRBT   =     1.000
  FRPN   =     1.000  RDIAM  =     1.150  RLENG  =     1.500

  IPR    =        -3  IPRINT =         0  IPLOT  =         0
  KERROR =         0  KODE   =         2  KT5    =         2
  KT7    =         2  MINPR  =         0  NAB    =         5
  NOZZ   =         1  NPROP  =         6  NSUMM  =        15
                                          IENG   =         2 (TF30)

  THESE VARIABLES ARE USED BY TABLE LOOK UP
  ESF    =     1.000  NDTAIL   = 0  IPDBUG  = 0  IIPRNT = 0

  ALTD   =        0.    30000.    30000.    30000.    30000.    30000.
  XMACH  =     0.000     0.800     1.000     1.200     1.400     1.600
  XMPRI  =     0.000     0.300     0.600     0.900     1.200     4.000
  XPRI   =     1.000     1.000     1.000     1.000     1.000     1.000
  XMPRI1 =     0.000     0.000     0.000     0.000     0.000     0.000     0.000     0.
000     0.000     0.000
  XPRI1  =     0.000     0.000     0.000     0.000     0.000     0.000     0.000     0.0
00     0.000     0.000
  XMPRI2 =     0.000     0.000     0.000     0.000     0.000     0.000     0.000     0.
000     0.000     0.000
  XPRI2  =     0.000     0.000     0.000     0.000     0.000     0.000     0.000     0.0
00     0.000     0.000
     CALLING MODULE NUMBER  6
1                          INITIAL WEIGHTS INPUT DATA
                           ACSYNT MODULE NUMBER 6
 AIRCRAFT TYPE:  FIGHTER
 TITLE:   GENERAL DYNAMICS F-16A BLOCK 15 WEIGHTS
0CONTROL OPTIONS
```

```
     IPRINT =  1   IGRAPH =  1   ITAIL  =  0   KERROR =  0
     IDELT  =  1   IOBLIQ =  0   KWING  =  1   KBODY  =  2
     K1     = 1.00 K2     = 1.00 K3     =  1.0 K4     =  1.0
     K5     = 1.00 K6     = 1.00 AFMACH = 2.20 MAXIT  =  1
     KB     = 0.00 KP1    = 0.00 KP2    = 0.00
     FR     = 1.50 TECHG  = 1.00 WGTO   =    29805.
     STRESS =    30000.          DENS   =  0.056     FLIFTF =0.000
     TECHI  =  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00
   SLOPE(1) =  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00
   SLOPE(9) =  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00
0INITIAL ESTIMATES
  IF CODE = 1, WEIGHT IS FIXED

  QUANTITY       VALUE   CODE          QUANTITY       VALUE   CODE

   WAF    =      8942.    0           WAIRC  =       267.    1
   WAPU   =         0.    0           WAMMUN =       287.    1
   WARM   =       598.    1           WBAG   =         0.    0
   WBB1   =       339.    1           WBODY  =      3422.    1
   WCAND  =         0.    0           WCARGO =         0.    0
   WCREW  =       215.    1           WE     =      3452.    1
   WELT   =      1143.    1           WEP    =       586.    1
   WETANK =      1250.    1           WFEQ   =      4113.    0
   WFS    =       375.    1           WCA    =       312.    1
   WGEAR  =       172.    1           WHDP   =       311.    1
   WHT    =       498.    1           WINST  =       107.    1
   WLG    =       988.    1           WMISS  =       338.    1
   WNA    =       655.    1           WPA    =         0.    0
   WPASS  =         0.    0           WPL    =      3547.    0
   WPS    =      4113.    0           WSC    =       716.    1
   WTSUM  =     29805.    0           WVT    =       330.    1
   WWING  =      1936.    1           WBOMB  =         0.    0
   WENVP  =         0.    0           WPIV   =         0.    0
   WLIFTF =         0.    0           WBB2   =         0.    0

     * * * BEGIN VEHICLE CONVERGENCE

     ESTIMATED GROSS WEIGHT  = 0.29805E+05
     CALLING MODULE NUMBER  1
     CALLING MODULE NUMBER  2
     CALLING MODULE NUMBER  6

     CALCULATED GROSS WEIGHT = 0.30344E+05

     SLOPE OF WCALC VS WEST LINE = 0.75000E+00

     ESTIMATED GROSS WEIGHT  = 0.31960E+05
     CALLING MODULE NUMBER  1
     CALLING MODULE NUMBER  2
     CALLING MODULE NUMBER  6

     CALCULATED GROSS WEIGHT = 0.31100E+05

     SLOPE OF WCALC VS WEST LINE = 0.35080E+00

     ESTIMATED GROSS WEIGHT  = 0.30635E+05
     CALLING MODULE NUMBER  1
     CALLING MODULE NUMBER  2
     CALLING MODULE NUMBER  6

     CALCULATED GROSS WEIGHT = 0.30600E+05

     SLOPE OF WCALC VS WEST LINE = 0.33171E+00

     ESTIMATED GROSS WEIGHT  = 0.30582E+05
     CALLING MODULE NUMBER  1
     CALLING MODULE NUMBER  2
```

```
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30570E+05

SLOPE OF WCALC VS WEST LINE = 0.52708E+00

ESTIMATED GROSS WEIGHT  = 0.30557E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30575E+05

SLOPE OF WCALC VS WEST LINE = 0.30000E+00

ESTIMATED GROSS WEIGHT  = 0.30572E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30585E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05
```

```
SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
CALLING MODULE NUMBER  2
CALLING MODULE NUMBER  6

CALCULATED GROSS WEIGHT = 0.30584E+05

SLOPE OF WCALC VS WEST LINE = 0.66559E+00

ESTIMATED GROSS WEIGHT  = 0.30608E+05
CALLING MODULE NUMBER  1
```

```
     CALLING MODULE NUMBER  2
     CALLING MODULE NUMBER  6

     CALCULATED GROSS WEIGHT = 0.30584E+05

     SLOPE OF WCALC VS WEST LINE = 0.66559E+00

     ESTIMATED GROSS WEIGHT  = 0.30608E+05
     CALLING MODULE NUMBER  1
     CALLING MODULE NUMBER  2
     CALLING MODULE NUMBER  6

     CALCULATED GROSS WEIGHT = 0.30584E+05

     SLOPE OF WCALC VS WEST LINE = 0.66559E+00

     ESTIMATED GROSS WEIGHT  = 0.30608E+05
     CALLING MODULE NUMBER  1
     CALLING MODULE NUMBER  2
     CALLING MODULE NUMBER  6

     CALCULATED GROSS WEIGHT = 0.30584E+05

     * * * END VEHICLE CONVERGENCE

     21 CONVERGENCE ITERATIONS REQUIRED
     CALLING MODULE NUMBER  1
     CALLING MODULE NUMBER  2
 SUMMARY OF NON FATAL ERRORS ENCOUNTERED IN PROP
 P5 IS LESS THAN P6S    1  TIMES
 SUMMARY OF NON FATAL ERRORS ENCOUNTERED IN PROP
 P5 IS LESS THAN P6S    3  TIMES
     CALLING MODULE NUMBER  6
     CALLING MODULE NUMBER  2
1                        TRAJECTORY OUTPUT

                       MISSION  1  (PAYLOAD =   3027. LB)
```

PHASE	M SFC(I) SFC(U)	H THRUST(I) THRUST(U)	CL CD CDINST	ALPHA GAMMA L/D	WFUEL W THR/THA	TIME WA PR	VEL Q X
CLIMB	0.60	10000.	0.2701	3.35	189.4	0.99	649.
	0.95	11403.	0.0362	14.05	30418.7	318.49	369.
CYCLE	0.00	0.	0.0000	7.47	0.00	1.00	6.
CLIMB	0.87	30000.	0.3398	4.12	442.7	3.23	869.
	0.94	7450.	0.0497	4.64	29976.0	185.35	337.
CYCLE	0.00	0.	0.0000	6.83	0.00	1.00	24.
CLIMB	0.89	37600.	0.4496	5.66	382.7	3.93	864.
	0.94	5794.	0.0699	1.22	29593.3	135.74	246.
CYCLE	0.00	0.	0.0000	6.43	0.00	1.00	34.
CRUISE	0.87	38322.	0.4193	5.26	1495.3	22.34	845.
	0.91	4300.	0.0631	0.00	28097.9	118.98	227.
CYCLE	0.00	0.	0.0000	6.65	0.00	1.00	186.
COMBAT	0.90	30000.	0.6763	9.00	1928.0	3.06	895.
	2.57	14695.	0.1355	0.00	26169.9	175.81	357.
CYCLE	0.00	0.	0.0000	4.99	0.00	1.00	27.
ACCEL	1.60	30000.	0.0715	1.34	2596.2	2.53	1592.
	2.52	28302.	0.0801	0.00	23573.7	346.65	1128.
CYCLE	0.00	0.	0.0000	0.89	0.00	0.90	36.
COMBAT	1.20	30000.	0.3780	3.76	2712.4	3.25	1194.

```
        2.44      20499.      0.1074     0.00    19273.3   237.67      635.
CYCLE   0.00          0.      0.0000     3.52       0.00     0.99       38.

CLIMB   0.92      45250.      0.3989     4.89      224.5     2.40      888.
        0.94       4071.      0.0459     4.80    19048.8    96.30      180.
CYCLE   0.00          0.      0.0000     8.70       0.00     1.00       21.

CRUISE  0.87      45893.      0.3892     4.82      859.2    27.47      845.
        0.89       2071.      0.0436     0.00    18189.6    79.50      158.
CYCLE   0.00          0.      0.0000     8.93       0.00     1.00      229.

LOITER  0.43          0.      0.2214     2.83     1382.3    20.00      480.
        2.54       1631.      0.0200     0.00    16807.4   331.90      274.
CYCLE   0.00          0.      0.0000    11.09       0.00     1.00       95.
1                   FUEL SUMMARIES

    MISSION FUEL =  12213.
    RESERVE FUEL =      0.
    TRAPPED FUEL =     74.
  ---------------------------
    TOTAL FUEL   =  12287.

    TAKEOFF FUEL
    WFTO1 =       0.
    WFTO2 =       0.

    FUEL CARRIED :
    EXTERNALLY =   4810.
    INTERNALLY =   7477.

                    ADDITIONAL COMBAT PARAMETERS

              CONDITIONS     PS      NZ     TDOT  RADIUS   ALPHA    CL      CD

M= 0.90       1 G FLIGHT   328.2   1.00    0.00      0.    2.98   0.262  0.0410
H=30000.      SUSTAINED      0.0   2.66    5.07  10110.    9.00   0.676  0.1355
              MAX. INST. -2691.3   4.68    9.40   5455.   30.00   1.158  0.9074
              COMBAT ENERGY = 0.602628E+05

M= 1.20       1 G FLIGHT   240.7   1.00    0.00      0.    0.95   0.124  0.0827
H=30000.      SUSTAINED      0.0   3.11    4.55  15040.    3.76   0.378  0.1074
              MAX. INST. -2902.2   9.00   13.81   4953.   14.19   1.001  0.3894
              COMBAT ENERGY = 0.469386E+05


                    BLOCK TIME   =  1.486   HOURS
                    BLOCK RANGE =   696.0   N. M.
TAKEOFF FIELD LENGTH(TOTAL RUN)  =-17158.   FEET
LANDING FIELD LENGTH(TOTAL RUN)  =   9084.  FEET
LANDING FIELD LENGTH(GROUND RUN) =   4131.  FEET
  WEIGHT FOR LANDING CALCULATION =  23647. POUNDS
LANDING THRUST TO WEIGHT RATIO   =  0.656
                  TAKEOFF WEIGHT =  30608. POUNDS
                  LANDING WEIGHT =  16807. POUNDS
   CALLING MODULE NUMBER  6
1                    WEIGHT STATEMENT - FIGHTERS
                      GENERAL DYNAMICS F-16A BLOCK 15 WEIGHTS
                     ACSYNT MODULE NUMBER 6

            COMPONENT                     POUNDS      KILOGRAMS      PERCENT

          AIRFRAME STRUCTURE               7829.        3551.         25.60
            WING                           1936.         878.          6.33
            FUSELAGE                       3422.        1552.         11.19
            HORIZONTAL TAIL                 498.         226.          1.63
            VERTICAL TAIL                   330.         150.          1.08
            NACELLES                        655.         297.          2.14
```

ALIGHTING GEAR	988.	448.	3.23
PROPULSION	3827.	1736.	12.51
ENGINES (1)	3452.	1566.	11.29
FUEL SYSTEM	375.	170.	1.23
FIXED EQUIPMENT	3614.	1639.	11.82
HYD. + PNEU.	311.	141.	1.02
ELECTRICAL	586.	266.	1.92
AVIONICS	1143.	518.	3.74
INSTRUMENTATION	107.	49.	0.35
DE-ICE/AIR CONDITION	267.	121.	0.87
AUXILIARY GEAR	172.	78.	0.56
FURNISH. + EQPT.	312.	142.	1.02
FLIGHT CONTROLS	716.	325.	2.34
FUEL	12287.	5573.	40.17
PAYLOAD	3027.	1373.	9.90
FLIGHT CREW (1)	215.	98.	0.70
ARMAMENT	598.	271.	1.96
AMMUNITION	287.	130.	0.94
MISSILES	338.	153.	1.11
BOMBS	0.	0.	0.00
EXTERNAL TANKS	1250.	567.	4.09
ADV. WEAPONS 1	339.	154.	1.11
ADV. WEAPONS 2	0.	0.	0.00
CALCULATED WEIGHT	30584.	13873.	100.00
ESTIMATED WEIGHT	30608.	13884.	
PERCENT ERROR	-0.08		

CALLING MODULE NUMBER 4

1 ENGINE SUMMARY

ENGINE DIAMETER = 4.25 FEET
ENGINE LENGTH = 14.80 FEET
ENGINE WEIGHT = 4745.39 POUNDS
BYPASS RATIO = 0.63
NO OF ENGINES = 1.
DRAG REF AREA = 300.00 SQ FEET
PWCC = PERCENT OF ENGINE CORRECTED AIRFLOW
THRUST = ENGINE THRUST (POUNDS PER ENGINE)
SFC = ENGINE SPECIFIC FUEL CONSUMPTION
THRUSTU= THRUST PER ENGINE IN LBS, W/O INSTAL DRAG CORR
SFCU = SFC,1/HR, W/O INSTALLATION DRAG CORR
CDINS = TOT INSTALLATION DRAG COEF PER A/C (SWING REF)

1

MACH	ALT	PWCC	THRUST	THRUSTU	SFC	SFCU	CDINS
0.000	0.	100.	15355.	0.	0.817	0.000	0.0000
		98.	14177.	0.	0.803	0.000	0.0000
		96.	12000.	0.	0.775	0.000	0.0000
		94.	9775.	0.	0.761	0.000	0.0000
		92.	7393.	0.	0.783	0.000	0.0000
		90.	4505.	0.	0.960	0.000	0.0000
0.800	30000.	100.	6065.	0.	0.934	0.000	0.0000
		98.	5137.	0.	0.914	0.000	0.0000

		96.	4229.	0.	0.906	0.000	0.0000
		94.	3325.	0.	0.919	0.000	0.0000
		92.	2414.	0.	0.981	0.000	0.0000
		90.	1455.	0.	1.210	0.000	0.0000
		88.	346.	0.	3.541	0.000	0.0000
1.000	30000.	100.	7597.	0.	0.976	0.000	0.0000
		98.	6453.	0.	0.954	0.000	0.0000
		96.	5340.	0.	0.941	0.000	0.0000
		94.	4250.	0.	0.944	0.000	0.0000
		92.	3162.	0.	0.985	0.000	0.0000
		90.	2057.	0.	1.129	0.000	0.0000
		88.	874.	0.	1.861	0.000	0.0000
1.200	30000.	100.	8693.	0.	0.994	0.000	0.0000
		98.	8368.	0.	0.996	0.000	0.0000
		96.	6963.	0.	0.978	0.000	0.0000
		94.	5597.	0.	0.974	0.000	0.0000
		92.	4257.	0.	0.996	0.000	0.0000
		90.	2917.	0.	1.088	0.000	0.0000
		88.	1546.	0.	1.445	0.000	0.0000
		86.	48.	0.	29.304	0.000	0.0000
1.400	30000.	100.	9265.	0.	1.019	0.000	0.0000
		98.	9279.	0.	1.027	0.000	0.0000
		96.	8988.	0.	1.031	0.000	0.0000
		94.	7271.	0.	1.022	0.000	0.0000
		92.	5602.	0.	1.036	0.000	0.0000
		90.	3962.	0.	1.100	0.000	0.0000
		88.	2313.	0.	1.336	0.000	0.0000
		86.	606.	0.	3.269	0.000	0.0000

AFTERBURNER OR DUCTBURNER (OR BOTH) LIT

1.600	30000.	100.	28302.	0.	2.522	0.000	0.0000
		100.	27450.	0.	2.466	0.000	0.0000
		100.	26594.	0.	2.410	0.000	0.0000
		100.	25732.	0.	2.353	0.000	0.0000
		100.	24866.	0.	2.296	0.000	0.0000
		100.	23993.	0.	2.238	0.000	0.0000
		100.	23113.	0.	2.179	0.000	0.0000
		100.	22226.	0.	2.119	0.000	0.0000
		100.	21331.	0.	2.058	0.000	0.0000
		100.	20426.	0.	1.995	0.000	0.0000
		100.	19512.	0.	1.931	0.000	0.0000
		100.	18587.	0.	1.865	0.000	0.0000
		100.	17650.	0.	1.797	0.000	0.0000
		100.	16700.	0.	1.727	0.000	0.0000
		100.	15737.	0.	1.653	0.000	0.0000

SEA-LEVEL STATIC THRUST = 23294. (MAX)
SEA-LEVEL SFC = 3.042
CALLING MODULE NUMBER 1

1NASA/AMES A C S Y N T FINAL OUTPUT

FUSELAGE DEFINITION			NACELLE DEFINITION			NACELLE LOCATION		
X	R	AREA	X-XNOSE	R	AREA	X	Y	Z
0.00	0.00	0.00	0.00	2.09	13.66	9.52	0.00	-1.90
0.95	0.34	0.37	4.89	2.45	18.78			
1.90	0.56	1.00	17.31	2.45	18.78			
2.85	0.75	1.76	22.20	2.34	17.19			
3.81	0.91	2.59						

```
  4.76    1.05     3.47
  5.71    1.18     4.35
  6.66    1.29     5.23
  7.61    1.39     6.09
  8.56    1.48     6.91
  9.52    1.56     7.68
 10.47    1.63     8.40
 11.42    1.70     9.05
 12.37    1.75     9.63
 13.32    1.80    10.13
 14.27    1.83    10.55
 15.23    1.86    10.88
 16.18    1.88    11.12
 17.13    1.89    11.28
 18.08    1.90    11.34
 19.03    1.90    11.34
 19.98    1.90    11.34
 20.94    1.90    11.34
 21.89    1.90    11.34
 22.84    1.90    11.34
 23.79    1.90    11.34
 24.74    1.90    11.34
 25.69    1.90    11.34
 26.64    1.90    11.34
 27.60    1.90    11.34
 28.55    1.90    11.34
 29.50    1.90    11.34
 30.45    1.90    11.34
 31.40    1.90    11.34
 32.35    1.90    11.34
 33.31    1.90    11.34
 34.26    1.90    11.34
 35.21    1.90    11.34
 36.16    1.90    11.34
 37.11    1.99    12.50
 38.06    2.07    13.48
 39.02    2.13    14.23
 39.97    2.16    14.72
 40.92    2.18    14.91
 41.87    2.17    14.78
 42.82    2.14    14.33
 43.77    2.07    13.51
 44.73    1.98    12.31
 45.68    1.84    10.69
 46.63    1.65     8.55
 47.58    1.28     5.11
                **FUSELAGE**          **PODS**
 MAX. DIAMETER.....     3.800      .....     4.890
 FINENESS RATIO.....   12.521
 SURFACE AREA.......  688.777      .....   334.096 (EACH)
 VOLUME.............  401.454
1NASA/AMES     A C S Y N T     FINAL OUTPUT
          DIMENSIONS OF PLANAR SURFACES

                          WING  H.TAIL  V.TAIL  CANARD  UNITS
OPLAN AREA.........      300.0    63.7    54.8     0.0 (SQ.FT.)
 SURFACE AREA.......     333.0   115.3   116.9     0.0 (SQ.FT.)
 VOLUME.............      95.1    13.3    11.6     0.0 (CU.FT.)
 SPAN...............    30.000  11.604   8.417   0.000 (FT.)
 L.E. SWEEP.........    40.000  40.000  47.500   0.000 (DEG.)
 C/4 SWEEP..........    32.183  32.273  43.226   0.000 (DEG.)
 T.E. SWEEP.........     0.000   0.500  25.909   0.000 (DEG.)
 ASPECT RATIO ......     3.000   2.114   1.294   0.000
 ROOT CHORD.........    16.293   7.898   9.053   0.000 (FT.)
 ROOT THICKNESS.....     7.821   5.687   5.758   0.000 (IN.)
 ROOT T/C ..........     0.040   0.060   0.053   0.000
 TIP CHORD..........     3.707   3.080   3.956   0.000 (FT.)
```

```
 TIP THICKNESS.....    1.779    1.294    1.424    0.000 (IN.)
 TIP T/C ..........    0.040    0.035    0.030    0.000
 TAPER RATIO ......    0.228    0.390    0.437    0.000
 MEAN AERO CHORD....  11.320    5.842    6.837    0.000 (FT.)
 LE ROOT AT........   18.194   38.445   31.057    0.000 (FT.)
 C/4 ROOT AT.......   22.267   40.419   33.320    0.000 (FT.)
 TE ROOT AT........   34.487   46.343   40.110    0.000 (FT.)
 LE M.A.C. AT......   23.167   40.523   35.050    0.000 (FT.)
 C/4 M.A.C. AT.....   25.997   41.983   36.759    0.000 (FT.)
 TE M.A.C. AT......   34.487   46.365   41.887    0.000 (FT.)
 Y M.A.C. AT.......    5.927    2.477    3.659    0.000
 LE TIP AT.........   30.781   43.313   40.242    0.000 (FT.)
 C/4 TIP AT........   31.707   44.083   41.231    0.000 (FT.)
 TE TIP AT.........   34.487   46.394   44.199    0.000 (FT.)
 ELEVATION.........    0.000    0.000    1.900    0.000 (FT.)
 VOLUME COEFF. ....             0.300    0.065    0.000

          E X T E N S I O N S
                              STRAKE   REAR EXTENSION
 CENTROID LOCATION AT.....    0.000    0.000
 AREA.....................    0.000    0.000
 SWEEP ANGLE..............    0.000
0         F U E L   T A N K S
  TANK          VOLUME     WEIGHT    DENSITY
OWING             24.      1153.      49.00
 FUS#1            88.      4300.      49.00
 FUS#2            41.      2024.      49.00
 TOTAL                     7477.
OMISSION FUEL REQUIRED=      7477.
OAVAILABLE FUEL VOLUME IN WING=        72.

 AIRCRAFT WEIGHT  =  30608.109 Lbs.
 AIRCRAFT VOLUME  =    521.435 Cu.Ft.
 AIRCRAFT DENSITY =     58.700 Lbs./Cu.Ft.
ICASE= 4
     CALLING MODULE NUMBER 11
1         SUMMARY --- ACSYNT OUTPUT --- NASA, AMES RESEARCH CENTER

  ENGLISH UNITS -
 ***** GENERAL DYNAMICS F-16A FIGHTER - AIR SUPERIORITY MISSION *****
     DISTANCES IN FEET

     WEIGHTS IN LBS.
    GENERAL            FUSELAGE                          WING    HTAIL   VTAIL
     FORCES IN LBS.

     PRESSURES IN LBS/FT**2
 WG      30584.   LENGTH          47.6   AREA           300.0    63.7    54.8
 W/S       101.9  DIAMETER         3.8   WETTED AREA    333.0   115.3   116.9
 T/W         0.76 VOLUME         401.5   SPAN            30.0    11.6     8.4
 N(Z) ULT 13.5    WETTED AREA    688.8   L.E. SWEEP      40.0    40.0    51.2
 CREW        1.   FINENESS RATIO  12.5   C/4 SWEEP       32.2    32.3    43.2
 PASENGERS   0.                          ASPECT RATIO    3.00    2.11    1.29
                                         TAPER RATIO     0.23    0.39    0.44
    ENGINE              WEIGHTS          T/C ROOT        0.04    0.06    0.05
                                         T/C TIP         0.04    0.04    0.03
 NUMBER      1.                    W     WG  ROOT CHORD    16.3     7.9     9.1
 LENGTH   14.8    STRUCT.    7829. 25.6   TIP CHORD       3.7     3.1     4.0
 DIAM.     4.3    PROPUL.    3827. 12.5   M.A. CHORD      11.3     5.8     6.8
 WEIGHT 4745.4    FIX. EQ.   3614. 11.8   LOC. OF L.E.    18.2    38.4    31.1
 TSLS   23294.    FUEL      12287. 40.2
 SFCSLS    3.04   PAYLOAD    3027.  9.9

                        MISSION SUMMARY
```

PHASE	MACH	ALT	FUEL		TIME		DIST	L/D	THRUST	SFC	Q
TAKEOFF	0.00	0.	0.	3.9	-17157.5						
CLIMB	0.60	10000.	189.	1.0	5.8	7.47	11402.6	0.947	369.3		
CLIMB	0.87	30000.	443.	3.2	24.2	6.83	7450.3	0.944	336.5		
CLIMB	0.89	37600.	383.	3.9	33.6	6.43	5793.9	0.938	245.9		
CRUISE	0.87	38322.	1495.	22.3	186.4	6.65	4300.2	0.914	227.3		
COMBAT	0.90	30000.	1928.	3.1	27.1	4.99	14694.5	2.573	357.0		
ACCEL	1.60	30000.	2596.	2.5	35.8	0.89	28302.4	2.522	1128.4		
COMBAT	1.20	30000.	2712.	3.3	38.3	3.52	20499.3	2.443	634.7		
CLIMB	0.92	45250.	224.	2.4	20.7	8.70	4071.3	0.940	180.0		
CRUISE	0.87	45893.	859.	27.5	229.3	8.93	2071.3	0.890	158.2		
LOITER	0.43	0.	1382.	20.0	94.8	11.09	1630.6	2.543	273.9		
LANDING					9084.1						

BLOCK TIME = 1.486 HR
BLOCK RANGE = 696.0 NM

COMBAT PHASES

MACH	ALT	PS1G	NZS	CLS	CDS	ALS	NZI	PSI	CLI	CDI	ALI	CBE
0.90	30000.	328.	2.7	0.676	0.1355	9.0	4.7	-2691.	1.158	0.9074	30.0	60263.
1.20	30000.	241.	3.1	0.378	0.1074	3.8	9.0	-2902.	1.001	0.3894	14.2	46939.

1 PROGRAM CALLS TO ANALIZ

ICALC	CALLS
1	1
2	1
3	1

APPENDIX D. GRAPHIGS INFORMATION

The PHIGS graphics standard was used throughout the ACSYNT/VPI project to insure code portability. The standard specifies the function, name, and parameter list for each PHIGS subroutine; all of which were followed carefully during the design and coding of ACSYNT/VPI.

In order for IBM's non-standard PHIGS implementation to be used during development of ACSYNT/VPI, a layer of conversion routines were written which follow the PHIGS standard but actually call IBM "graPHIGS" routines to perform the PHIGS functions. When the ACSYNT/VPI code is ported to a non-IBM environment, the conversion and graPHIGS routines will be replaced by standard PHIGS routines for that environment.

The following pages of this appendix contain a list of the IBM graPHIGS enhancements and omissions to the proposed PHIGS standard. The information was taken from reference 14. Each enhancement or omission is followed by the name of the corresponding graPHIGS subroutine.

D.1 Functional Enhancements

Functional enhancements include any IBM graPHIGS function which is not supported in the PHIGS standard. The graPHIGS enhancements are listed below:

- Output Primitives
  1. disjoint polyline (GPDPL2, GPDPL3)
  2. annotation text (GPAN2, GPAN3)
  3. pixel (GPPXL2, GPPXL3)
  4. circle (GPCR2)

5. circular arc (GPCRA2)
6. ellipse (GPEL2, GPEL3)
7. elliptical arc (GPELA2, GPELA3)

- Polyline End Types (GPPLET)
- Viewing
  1. View Port Shielding (GPVCH)
  2. Viewport Borders (GPVCH)
  3. Viewport Priority (GPVP)
- Posting Root Structures
  1. Associate Structure with Workstation (GPARW, GPDARW, GPDRW)
  2. Associate Structure with View (GPARV, GPDARV, GPDRV)
- Second Level Error Handling
- Configuration Variability (see Understanding graPHIGS p. 11-5)
- Structure Editing and Manipulation
  1. Set Element Pointer at Pick Identifier (GPEPPK)
  2. Delete Structure Network (GPDLNT)
- Programmable Input Device Triggers
  1. Set Input Device Trigger (GPIT)
  2. Inquire Number of Secondary Triggers (GPQNST)
  3. Inquire Actual Input Trigger Capabilities (GPQAIT)
  4. Inquire Input Device Trigger State (GPQITS)
  5. Inquire Default Input Device Triggers (GPQDIT)
- Asynchronous Update Workstation (GPUPWA)

D.2 Omissions

Omissions include any function specified in the PHIGS standard which has not been implemented in graPHIGS. graPHIGS omissions include:

- Cell Array Primitive

- Pattern Reference Point
- Pattern Plane Vector
- Empty Interior Style
- Fill Area Primitive is graPHIGS Polygon Primitive
- Archival Functions
- Change Structure Identifier
- Change Structure References
- Change Structure Identifier and References

D.3 Differences

Differences between IBM's implementation of PHIGS and the PHIGS standard other than omissions and enhancements are listed below:

- The default deferral state for graPHIGS is WAIT not ASAP as in the PHIGS standard.
- By default, graPHIGS defines all bundle table entries.
- The default view in graPHIGS is from -1.0 to 1.0 in both the U and V directions. The PHIGS standard uses 0.0 to 1.0.

D.4 Language Binding

The PHIGS proposal calls for a specific FORTRAN binding which details the PHIGS subroutine names and parameters for each PHIGS function. graPHIGS uses a "generic language binding" which provides a single interface between many programming languages. This may make installation on of graPHIGS applications on non-IBM systems difficult. Again, ACSYNT/VPI was coded using the PHIGS standard to promote code portability.

For more information concerning graPHIGS, please refer to the following IBM documents:

- Introducing graPHIGS, SC33-8100
- Installing GDDM/graPHIGS, SC33-8101
- Understanding graPHIGS, SC33-8102
- Writing Applications with graPHIGS, SC33-8103
- Programmer's Reference for graPHIGS, SC33-8104
- Messages and Error Codes for graPHIGS, SC33-8105
- Programmer's Pocket Reference for graPHIGS, SC33-8107
- Problem Diagnosis for GDDM/graPHIGS, SC33-8108

APPENDIX E. FORTRAN 77 EXTENSION USED IN ACSYNT/VPI CODE

For the development of the ACSYNT/VPI code, only one FORTRAN extension was used to improve ACSYNT/VPI's ability to read numeric input files and input lines type at the workstation keyboard. The extension involves the use of a "list-directed internal read" which converts a character string into real or integer numbers. The character string is treated as if it were an input line from an external file. The following is an example program containing a list-directed internal read:

```
      PROGRAM XMPL

      CHARACTER*22 STRING
      INTEGER NUMBER

C     DEFINE EXAMPLE DATA STRING WITH INTEGER NUMBER & COMMENT
      DATA STRING /'  1  COMMENT STRING'/

C     READ NUMBER FROM EXAMPLE STRING USING LIST-DIRECTED INTERNAL READ
      READ(STRING, *) NUMBER

C     WRITE NUMBER TO TERMINAL
      WRITE(*,*)'NUMBER = ',NUMBER

      STOP
      END
```

Compiling the above program using IBM's VS FORTRAN version 2 (77 option) compiler produced the following warning message:

```
ISN    5:       READ(STRING, *) NUMBER
(W) INTERNAL LIST-DIRECTED INPUT/OUTPUT IS A NON-STANDARD USAGE.
```

When executed, the program reads the value "1" from the string "STRING" into the integer variable "NUMBER" then prints the following:

```
NUMBER =           1
```

The example program listed above could be standardized by replacing the list-directed internal read with a *formated* internal read and a format statement as shown below:

```
C     READ NUMBER FROM INPUT STRING FORMATED
      READ(STRING, 1000) NUMBER

 1000 FORMAT(I3)
```

For the ACSYNT/VPI code, formated internal reads were not desirable do to the need for free format input files containing random comment lines. But, if the "list-directed internal read" extension is not supported on the ACSYNT/VPI target system, the ACSYNT/VPI code and data files can be modified to use standard formated internal reads as shown in the example above.

The extension described above was used in the following ACSYNT/VPI routines:

- in file "GMVWDB NASA":
  - RDNVWS - reads the number of defined geometry views from an ACSYNT/VPI input file
  - RDVWNM - reads a geometry view identification number from an ACSYNT/VPI input file
  - RDCVWS - reads geometry view character data from an ACSYNT/VPI input file
  - RDIVWS - reads geometry view integer data from an ACSYNT/VPI input file
  - RDRVWS - reads geometry view real data from an ACSYNT/VPI input file
- in file "GEOMDB NASA":
  - RDNCMP - reads the number of defined aircraft components from an ACSYNT/VPI input file
  - RDCPNM - reads aircraft component number from an ACSYNT/VPI input file
  - RDCCMP - reads aircraft component character data from an ACSYNT/VPI input file
  - RDICMP - reads aircraft component integer data from an ACSYNT/VPI input file
  - RDRCMP - reads aircraft component real data from an ACSYNT/VPI input file
- in file "UTILTY NASA":
  - PARSE - parses a keyboard (string) input line for numeric values

In addition, the routine "GTINLN" in file "UTILTY NASA" may need to be modified if any of the routines listed above are changed. "GTINLN" is responsible for reading an input line from a file and putting the line into a character variable.

APPENDIX F. EXAMPLE ACSYNT/VPI MENU MODULE

This appendix contains an example ACSYNT/VPI menu module for version X1.0 of ACSYNT/VPI.

```
C=====================================================================
C                        M O D U L E   A S H A D
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL ASHAD(WKID)
C=====================================================================
C INPUT PARAMETERS:
C     WKID   - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     NONE
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     GEOMETRY SHADING MODULE
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     OPSHAD - OPENS SHADING
C     MSHAD  - SHADING MENU DRIVER
C     CLSHAD - CLOSES SHADING
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  11/30/87 (12:24:46)
C=====================================================================
C DEVELOPMENT SITE:
```

```
C     THE MECHANICAL ENGINEERING DEPARTMENT
C     VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C     BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE ASHAD(WKID)

      INTEGER WKID

C     OPEN SHADING
      CALL OPSHAD(WKID)

C     SHADE MENU
      CALL MSHAD(WKID)

C     CLOSE SHADING
      CALL CLSHAD(WKID)

      RETURN
      END

C=====================================================================
C                      M O D U L E   O P S H A D
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.1
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL OPSHAD(WKID)
C=====================================================================
C INPUT PARAMETERS:
C     WKID - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     NONE
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     PREPARES FOR SHADING
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     MSG - WRITES A SCREEN MESSAGE
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  01/05/88 (12:17:54)
C=====================================================================
C DEVELOPMENT SITE:
C     THE MECHANICAL ENGINEERING DEPARTMENT
C     VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
```

```
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE OPSHAD(WKID)

      INTEGER WKID

C     WRITE MESSAGE
      CALL MSG(WKID, 'SHADE:')

      RETURN
      END

C=====================================================================
C                        M O D U L E  M S H A D
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.2
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL MSHAD(WKID)
C=====================================================================
C INPUT PARAMETERS:
C     WKID   - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     TITLE  - CHARACTER, MENU TITLE
C     NITEMS - INTEGER, NUMBER OF MENU ITEMS
C     ITEMS(NITEMS) - CHARACTER, MENU ITEMS
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     SHADING MENU DRIVER
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     NEWMNU - DISPLAYS A NEW MENU
C     EXSHAD - EXECUTES SHADING MENU
C     OLDMNU - DISPLAYS PREVIOUS MENU
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  11/30/87 (16:40:06)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE MSHAD(WKID)
```

```
      INTEGER WKID

      CHARACTER*12 TITLE
      PARAMETER(TITLE = '   SHADE    ')

      INTEGER NITEMS
      PARAMETER(NITEMS = 9)
      CHARACTER*16 ITEMS(NITEMS)

C     DEFINE MENU ITEMS
      DATA ITEMS / ' AMBIENT LIGHT  ',
     >             ' SURFACE CHAR.  ',
     >             '   SUN MODEL    ',
     >             '    EYE FLAG    ',
     >             ' NUM. OF LIGHTS ',
     >             ' LIGHT SOURCES  ',
     >             '  LIGHT COLOR   ',
     >             ' RESET SHADING  ',
     >             '     RETURN     ' /

C     DRAW NEW MENU
      CALL NEWMNU(WKID, TITLE, NITEMS, ITEMS)

C     EXECUTE MENU
      CALL EXSHAD(WKID)

C     DISPLAY OLD MENU
      CALL OLDMNU(WKID)

      RETURN
      END

C=======================================================================
C                      M O D U L E   E X S H A D
C=======================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.2.2
C=======================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=======================================================================
C CALLING SEQUENCE:
C     CALL EXSHAD(WKID)
C=======================================================================
C INPUT PARAMETERS:
C     WKID   - INTEGER, WORKSTATION IDENTIFIER
C=======================================================================
C OUTPUT PARAMETERS:
C     NONE
C=======================================================================
C COMMON INPUTS:
C     NONE
C=======================================================================
C COMMON OUTPUTS:
C     NONE
C=======================================================================
C LOCAL VARIABLES:
C     PREV - LOGICAL, PREVIOUS MENU FLAG
C=======================================================================
C FUNCTIONAL DESCRIPTION:
C     EXECUTES SHADING MENU
C=======================================================================
C EXAMPLE(S):
C     N/A
C=======================================================================
```

```
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     MSG    - WRITES A SCREEN MESSAGE
C     INSHAD - GETS AND PROCESSES SHADING MENU INPUT
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  09/12/87 (19:18:18)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE EXSHAD(WKID)

      INTEGER WKID

      LOGICAL PREV

C     ASSUME PREVIOUS MENU FLAG IS FALSE
      PREV = .FALSE.

C     WRITE PROMPT MESSAGE
      CALL MSG(WKID, 'SELECT VIEW TO BE SHADED:')

C     GET AND PROCESS MENU INPUT
 100  CALL INSHAD(WKID, PREV)

      IF (.NOT. PREV) THEN
        GOTO 100
      ENDIF

      RETURN
      END

C=====================================================================
C                      M O D U L E  I N S H A D
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.2.2.3
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL INSHAD(WKID, PREV)
C=====================================================================
C INPUT PARAMETERS:
C     WKID   - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     PREV   - LOGICAL, PREVIOUS MENU FLAG
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     ICL  - INTEGER, INPUT DEVICE CLASS RETURNED BY GTMNIN
C     IDNR - INTEGER, INPUT DEVICE NUMBER RETURNED BY GTMNIN
C=====================================================================
```

```
C FUNCTIONAL DESCRIPTION:
C     GETS AND PROCESSES SHADING MENU INPUT
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     GTMNIN - GETS MENU INPUT
C     PSHAD  - PROCESSES A SHADING MENU PICK
C     PSHVL  - PROCESSES SHADING VALUATOR INPUT
C     MSG    - WRITES A SCREEN MESSAGE
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  09/17/87 (13:26:41)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE INSHAD(WKID, PREV)

      INTEGER WKID
      LOGICAL PREV

      INTEGER ICL, IDNR, PPICK, PVALUA
      PARAMETER(PVALUA = 3, PPICK = 5)

C     GET MENU INPUT
      CALL GTMNIN(WKID, ICL, IDNR)

      IF (ICL .EQ. PPICK) THEN
C       PROCESS MENU ITEM
        CALL PSHAD(WKID, PREV)
      ELSE IF (ICL .EQ. PVALUA) THEN
C       PROCESS SHADING VALUATOR
        CALL PSHVL(WKID, IDNR)
      ELSE
C       WRITE ERROR MESSAGE
        CALL MSG(WKID, 'PICK A MENU ITEM OR VIEW ...')
      ENDIF

      RETURN
      END

C=====================================================================
C                       M O D U L E   P S H A D
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.2.2.3.3
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL PSHAD(WKID, PREV)
C=====================================================================
C INPUT PARAMETERS:
C     WKID   - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     PREV   - LOGICAL, PREVIOUS MENU FLAG
```

```
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     MTYPE  - INTEGER, MENU TYPE IDENTIFIER RETURNED BY GTMNTI
C     ITEM   - INTEGER, MENU ITEM IDENTIFIER RETURNED BY GTMNTI
C     CURMNU - CHARACTER, CURRENT MENU IDENTIFIER
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     PROCESSES A SHADING MENU PICK
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     GTMNTI - GETS THE MENU TYPE AND ITEM NUMBER OF THE PICK MENU
C     PISHAD - PROCESSES A PICKED SHADING MENU ITEM
C     PRSTMN - PROCESSES A STANDARD MENU PICK
C     PRSHPK - PROCESS A SHADING VIEW PICK
C     STSHPK - STORE SHADING GEOMETRY PICK
C     MSG    - WRITES A SCREEN MESSAGE
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  09/17/87 (13:40:55)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE PSHAD(WKID, PREV)

      INTEGER WKID
      LOGICAL PREV

      INTEGER MTYPE, ITEM, MREG, MSTND, MVIEW
      PARAMETER(MREG = 1, MSTND = 2, MGEOM = 3, MVIEW = 5)
      CHARACTER*4 CURMNU
      PARAMETER(CURMNU = 'SHAD')

C     GET MENU TYPE AND ITEM
      CALL GTMNTI(MTYPE, ITEM)

      IF (MTYPE .EQ. MREG) THEN
C       PROCESS MENU ITEM
        CALL PISHAD(WKID, ITEM, PREV)
      ELSE IF (MTYPE .EQ. MSTND) THEN
C       PROCESS STANDARD MENU ITEM
        CALL PRSTMN(WKID, CURMNU, ITEM, PREV)
      ELSE IF (MTYPE .EQ. MGEOM) THEN
C       STORE SHADING GEOMETRY PICK
        CALL STSHPK(WKID, ITEM)
      ELSE IF (MTYPE .EQ. MVIEW) THEN
C       PROCESS SHADING VIEW PICK
        CALL PRSHPK(WKID, ITEM)
      ELSE
C       WRITE ERROR MESSAGE
        CALL MSG(WKID, 'PICK A MENU ITEM ...')
      ENDIF
```

```
      RETURN
      END

C=====================================================================
C                    M O D U L E  P I S H A D
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.2.2.3.3.3
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL PISHAD(WKID, ITEM, PREV)
C=====================================================================
C INPUT PARAMETERS:
C     WKID   - INTEGER, WORKSTATION IDENTIFIER
C     ITEM   - INTEGER, PICK MENU ITEM NUMBER
C=====================================================================
C OUTPUT PARAMETERS:
C     PREV - LOGICAL, PREVIOUS MENU FLAG
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     NONE
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     PROCESSES A SHADING MENU ITEM PICKED BY THE USER
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     AAMBL - SET SHADING AMBIENT LIGHT FRACTION
C     ASRFC - SET SHADING SURFACE CHARACTERISTICS
C     ASUNF - SET SHADING SUN FLAG
C     AEYEF - SET SHADING EYE FLAG
C     ANLTS - SET NUMBER OF LIGHT SOURCES
C     ALTSR - SET SHADING LIGHT SOURCE LOCATION AND BRIGHTNESS
C     ALTCR - SET SHADING LIGHT SOURCE COLOR
C     ASHRS - RESET SHADING DATABASE
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  11/30/87 (09:19:01)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE PISHAD(WKID, ITEM, PREV)

      INTEGER WKID, ITEM
      LOGICAL PREV
```

```
      IF (ITEM .EQ. 1) THEN
C       SET AMBIENT LIGHT
        CALL AAMBL(WKID)
      ELSE IF (ITEM .EQ. 2) THEN
C       SET SURFACE CHARACTERISTICS
        CALL ASRFC(WKID)
      ELSE IF (ITEM .EQ. 3) THEN
C       SET SUN FLAG
        CALL ASUNF(WKID)
      ELSE IF (ITEM .EQ. 4) THEN
C       SET EYE FLAG
        CALL AEYEF(WKID)
      ELSE IF (ITEM .EQ. 5) THEN
C       SET NUMBER OF LIGHT SOURCES
        CALL ANLTS(WKID)
      ELSE IF (ITEM .EQ. 6) THEN
C       SET LIGHT SOURCE
        CALL ALTSR(WKID)
      ELSE IF (ITEM .EQ. 7) THEN
C       SET LIGHT COLOR
        CALL ALTCR(WKID)
      ELSE IF (ITEM .EQ. 8) THEN
C       RESET SHADING PARAMETERS TO DEFAULT
        CALL ASHRS(WKID)
      ELSE IF (ITEM .EQ. 9) THEN
C       RETURN TO PREVIOUS MENU
        PREV = .TRUE.
      ENDIF

      RETURN
      END

C=======================================================================
C                    M O D U L E   S T S H P K
C=======================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.2.2.3.3.5
C=======================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=======================================================================
C CALLING SEQUENCE:
C     CALL STSHPK(WKID, COMP)
C=======================================================================
C INPUT PARAMETERS:
C     WKID - INTEGER, WORKSTATION IDENTIFIER
C     COMP - INTEGER, PICKED AIRCRAFT COMPONENT NUMBER
C=======================================================================
C OUTPUT PARAMETERS:
C     NONE
C=======================================================================
C COMMON INPUTS:
C     NONE
C=======================================================================
C COMMON OUTPUTS:
C     /GMSHAD/ SHDFLG - INTEGER, FLAG INDICATING WHETHER OR NOT TO
C                       PERFORM ALL SHADING CALCULATIONS FROM SCRATCH
C                       (=0) OR TO USE THE DATA STORED IN MEMORY (=1)
C=======================================================================
C LOCAL VARIABLES:
C     NONE
C=======================================================================
C FUNCTIONAL DESCRIPTION:
C     STORES A PICKED AIRCRAFT COMPONENT NUMBER AND RESETS THE SHADING
C     FLAG
C=======================================================================
```

```
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     STGMPK - STORES A PICKED AIRCRAFT COMPONENT NUMBER
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  03/08/88 (08:54:18)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE STSHPK(WKID, COMP)

      INTEGER WKID, COMP

      COMMON /GMSHAD/ SHDFLG, NSHDVW, SHDVW
      INTEGER SHDFLG, NSHDVW, SHDVW(8)

C     STORE GEOMETRY PICK
      CALL STGMPK(WKID, COMP)

C     RESET SHADING FLAG
      SHDFLG = 0

      RETURN
      END

C=====================================================================
C                       M O D U L E  P S H V L
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.2.2.3.4
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL PSHVL(WKID, IDNR)
C=====================================================================
C INPUT PARAMETERS:
C     WKID - INTEGER, WORKSTATION IDENTIFIER
C     IDNR - INTEGER, VALUATOR DEVICE NUMBER
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C COMMON INPUTS:
C     /GMSHAD/ SHDFLG - INTEGER, SHADING FLAG INDICATING WHETHER
C                       OR NOT A SHADED IMAGE IS BEING DISPLAYED
C              NSHDVW - INTEGER, NUMBER OF GEOMETRY VIEWS CONTAINING
C                       A SHADED IMAGE
C              SHDVW( ) - INTEGER, LIST OF GEOMETRY VIEWS CONTAINING
C                        A SHADED IMAGE
C     /GMVIEW/ CURVW  - INTEGER, CURRENT GEOMETRY VIEW
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
```

```
C     START - INTEGER, STARTING INDEX FOR LIST SEARCH ROUTINE
C     LOC   - INTEGER, LOCATION OF AN ITEM IN A LIST RETURNED BY
C             SCHLST
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     PROCESSES SHADING VALUATOR INPUT
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     PGMVL  - PROCESSES GEOMETRY VALUATOR INPUT
C     SCHLST - SEARCHES AN INTEGER LIST FOR A SPECIFIED ITEM
C     DOSHAD - PERFORMS SHADING
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  02/15/88 (21:51:59)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE PSHVL(WKID, IDNR)

      INTEGER WKID, IDNR

      COMMON /GMSHAD/ SHDFLG, NSHDVW, SHDVW
      INTEGER SHDFLG, NSHDVW, SHDVW(8)
      COMMON /GMVIEW/ CURVW
      INTEGER CURVW

      INTEGER START, LOC
      PARAMETER(START = 1)

C     PROCESS GEOMETRY VALUATOR
      CALL PGMVL(WKID, IDNR)

C     CHECK VIEW LIST TO SEE IF CURRENT VIEW HAS BEEN SHADED
      CALL SCHLST(CURVW, START, NSHDVW, SHDVW, LOC)

      IF (LOC .NE. 0) THEN
C       RESHADE CURRENT VIEW
        CALL DOSHAD(WKID, CURVW, SHDFLG)
      ENDIF

      RETURN
      END

C=====================================================================
C                    M O D U L E  C L S H A D
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.3
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL CLSHAD(WKID)
C=====================================================================
C INPUT PARAMETERS:
```

```
C     WKID - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C COMMON INPUTS:
C     NONE
C=====================================================================
C COMMON OUTPUTS:
C     /SHCMPS/ NCMPS  - INTEGER, NUMBER OF COMPONENTS TO BE SHADED
C     /GMSHAD/ SHDFLG - INTEGER, SHADING FLAG INDICATING THAT GEOMETRY
C                       HAS BEEN SHADED
C              NSHDVW - INTEGER, NUMBER OF SHADED VIEWS
C              SHDVW( ) - INTEGER, LIST OF SHADED VIEW IDENTIFIERS
C=====================================================================
C LOCAL VARIABLES:
C     NONE
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     CLOSES SHADING
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     DROPSH - DROPS SHADED IMAGES FROM THE DISPLAY AND REPOSTS
C              WIREFRAME STRUCTURES
C     PSHLFT - PHIGS SET HIGHLIGHTING FILTER
C=====================================================================
C CODED BY:  STEVE WAMPLER
C     DATE:  01/05/88 (12:10:18)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE CLSHAD(WKID)

      INTEGER WKID

      COMMON /SHCMPS/ NCMPS, COMPS
      INTEGER NCMPS, COMPS
      COMMON /GMSHAD/ SHDFLG, NSHDVW, SHDVW
      INTEGER SHDFLG, NSHDVW, SHDVW(8)

      INTEGER ZERO, OFF
      PARAMETER(ZERO = 0, OFF = 0)

C     DROP SHADED IMAGES
      CALL DROPSH(WKID)

C     RESET SHADING FLAG
      SHDFLG = OFF

C     RESET NUMBER OF SHADED VIEWS
      NSHDVW = ZERO

C     RESET THE NUMBER OF PICKED COMPONENTS
      NCMPS = ZERO

C     RESET HIGHLIGHTING FILTER
      CALL PSHLFT(WKID, ZERO, ZERO, ZERO, ZERO)
```

```
      RETURN
      END

C=====================================================================
C                      M O D U L E   D R O P S H
C=====================================================================
C  PROJECT:  NASA/AMES
C  VERSION:  X1.0.0
C   PART #:  VT-A01B
C MODULE #:  ASHAD.3.1
C=====================================================================
C COMPILER:  FORTVS2 (FORTRAN 77)
C=====================================================================
C CALLING SEQUENCE:
C     CALL DROPSH(WKID)
C=====================================================================
C INPUT PARAMETERS:
C     WKID - INTEGER, WORKSTATION IDENTIFIER
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C COMMON INPUTS:
C     /GMSHAD/ SHDFLG - INTEGER, SHADING FLAG INDICATING WHETHER OR
C                       NOT A SHADED IMAGE IS BEING DISPLAYED
C              NSHDVW - INTEGER, NUMBER OF GEOMETRY VIEWS CONTAINING
C                       SHADED IMAGES
C              SHDVW( ) - INTEGER, LIST OF VIEWS CONTAINING SHADED
C                        IMAGES
C=====================================================================
C COMMON OUTPUTS:
C     NONE
C=====================================================================
C LOCAL VARIABLES:
C     SHITM  - INTEGER, VIEW DATABASE ITEM NUMBER FOR SHADING ROOT
C              STRUCTURE IDENTIFIER
C     NITEMS - INTEGER, NUMBER OF DATABASE ITEMS TO BE RETRIEVED
C     SHDRT  - INTEGER, SHADING ROOT STRUCTURE
C     ERRIND - INTEGER, ERROR INDICATOR RETURNED BY DATABASE ROUTINES
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     DROPS SHADED IMAGE FROM THE DISPLAY
C=====================================================================
C EXAMPLE(S):
C     N/A
C=====================================================================
C METHODS/ALGORITHMS:
C     N/A
C=====================================================================
C MODULES CALLED:
C     GTIVWS - GETS INTEGER VIEW DATA FROM VIEW DATABASE
C     PUPORT - PHIGS UNPOST ROOT STRUCTURE
C     PEMST  - PHIGS EMPTY STRUCTURE
C     STVWRT - SETS GEOMETRY VIEW ROOT STRUCTURE
C=====================================================================
C CODED BY:
C     DATE:  02/16/88 (13:02:39)
C=====================================================================
C DEVELOPMENT SITE:
C    THE MECHANICAL ENGINEERING DEPARTMENT
C    VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
C    BLACKSBURG, VA  24061
C=====================================================================

      SUBROUTINE DROPSH(WKID)
```

```
      INTEGER WKID

      COMMON /GMSHAD/ SHDFLG, NSHDVW, SHDVW
      INTEGER SHDFLG, NSHDVW, SHDVW(8)

      INTEGER SHITM, NITEMS, SHDRT, ERRIND
      PARAMETER(SHITM = 4, NITEMS = 1)

      DO 100 I = 1, NSHDVW

C       GET SHADING ROOT STRUCTURE IDENTIFIER FROM VIEW DATABASE
        CALL GTIVWS(SHITM, NITEMS, SHDVW(I), SHDRT, ERRIND)

C       UNPOST SHADING STRUCTURE
        CALL PUPORT(WKID, SHDRT)

C       EMPTY SHADING STRUCTURE
        CALL PEMST(SHDRT)

C       SET GEOMETRY VIEW ROOT
        CALL STVWRT(WKID, SHDVW(I))

 100  CONTINUE

      RETURN
      END
```

APPENDIX G. EXAMPLE ACSYNT/VPI GEOMETRY DATA FILE

This appendix contains an ACSYNT/VPI data file which specifies the geometry view and aircraft component data for an ACSYNT/VPI aircraft model. The file is read by the ACSYNT/VPI subroutine RDGMFL and is formated as follows:

Column	*Description*
1-5	database item number
6-13	database data (real, integer, or character data)
14-80	data description string (ignored by ACSYNT/VPI)

Data from an ACSYNT/VPI geometry data file is used to load the ACSYNT/VPI geometry view database and aircraft component database which contain the following information:

- number of defined geometry views
- current geometry view identifier
- view attribute data for each view:
  - view active flag
  - geometry root structure identifier
  - view projection type
  - geometry shading root structure identifier
  - geometry hidden surface root structure identifier
  - view frame root structure identifier
  - view frame sub-structure identifier
- view parameter data for each view:
  - x, y, z view rotation angles
  - x, y, z view translation values
  - view scale factor
  - viewport limits
  - window limits
  - coordinates of the projection reference point
  - view plane distance
  - front and back clipping plane distances
- number of defined aircraft components

- aircraft component attributes for each component:
  - component label
  - component existence flag
  - component type identifier
  - number of cross sections
  - number of points per cross section
  - number of component parameters
  - number of isoparametric surface lines in the u direction
  - number of isoparametric surface lines in the w direction
  - local symmetry flag
  - global symmetry flag
  - wireframe color index
  - wireframe line type
  - component attribute PHIGS structure identifier
  - component local symmetry PHIGS structure identifier
  - component PHIGS structure identifier
- aircraft component parameters for each component:
  - component rotation about global x axis
  - component rotation about global y axis
  - component rotation about global z axis
  - component translation along x axis
  - component translation along y axis
  - component translation along z axis

NOTE: All lines starting with an "*" in the following file are treated as comments by ACSYNT/VPI.

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
*****************************************************************
* GEOMETRY VIEW DATA
*****************************************************************
     4        NUMBER OF DEFINED VIEWS
     2        CURRENT VIEW IDENTIFIER
*****************************************************************
     2        GEOMETRY VIEW NUMBER
     1        NUMBER OF SPECIFIED CHARACTER VIEW PARAMETERS
1   '*'
     7        NUMBER OF SPECIFIED INTEGER VIEW PARAMETERS
1    1        VIEW ACTIVE FLAG
2    1002     GEOMETRY ROOT STRUCTURE IDENTIFIER
3    0        VIEW PROJECTION TYPE (0=PARALLEL, 1=PERSPECTIVE)
4    7020     GEOMETRY SHADING ROOT STRUCTURE IDENTIFIER
5    7020     GEOMETRY HIDDEN SURFACE ROOT STRUCTURE IDENTIFIER
6    7021     VIEW FRAME ROOT STRUCTURE IDENTIFIER
7    7022     VIEW FRAME SUB-STRUCTURE IDENTITIER
     23       NUMBER OF SPECIFIED REAL VIEW PARAMETERS
1    330.0    X AXIS ROTATION (DEGREES)
2    30.0     Y AXIS ROTATION (DEGREES)
3    30.0     Z AXIS ROTATION (DEGREES)
4    -19.0    X AXIS VIEW TRANSLATION
5    -9.0     Y AXIS VIEW TRANSLATION
```

```
6     0.0      Z AXIS VIEW TRANSLATION
7     1.00     VIEW SCALE FACTOR
8     0.0      VIEWPORT LIMIT (X MINIMUM)
9     0.8      VIEWPORT LIMIT (X MAXIMUM)
10    0.15     VIEWPORT LIMIT (Y MINIMUM)
11    0.95     VIEWPORT LIMIT (Y MAXIMUM)
12    0.00     VIEWPORT LIMIT (Z MINIMUM)
13    1.00     VIEWPORT LIMIT (Z MAXIMUM)
14    -25.0    VIEW WINDOW LIMIT X MINIMUM
15    25.0     VIEW WINDOW LIMIT X MAXIMUM
16    -25.0    VIEW WINDOW LIMIT Y MINIMUM
17    25.0     VIEW WINDOW LIMIT Y MAXIMUM
18    0.0      X COORDINATE OF PROJECTION REFERENCE POINT
19    0.0      Y COORDINATE OF PROJECTION REFERENCE POINT
20    100.0    Z COORDINATE OF PROJECTION REFERENCE POINT
21    0.0      VIEW PLANE DISTANCE
22    70.0     FRONT CLIPPING PLANE DISTANCE
23    -100.0   BACK CLIPPING PLANE DISTANCE
*****************************************************************
      3        GEOMETRY VIEW NUMBER
      1        NUMBER OF SPECIFIED CHARACTER VIEW PARAMETERS
1     '*'
      7        NUMBER OF SPECIFIED INTEGER VIEW PARAMETERS
1     0        VIEW ACTIVE FLAG
2     1003     GEOMETRY ROOT STRUCTURE IDENTIFIER
3     0        VIEW PROJECTION TYPE (0=PARALLEL, 1=PERSPECTIVE)
4     7030     GEOMETRY SHADING ROOT STRUCTURE IDENTIFIER
5     7030     GEOMETRY HIDDEN SURFACE ROOT STRUCTURE IDENTIFIER
6     7031     VIEW FRAME ROOT STRUCTURE IDENTIFIER
7     7032     VIEW FRAME SUB-STRUCTURE IDENTITIER
      23       NUMBER OF SPECIFIED REAL VIEW PARAMETERS
1     0.0      X AXIS ROTATION (DEGREES)
2     0.0      Y AXIS ROTATION (DEGREES)
3     90.0     Z AXIS ROTATION (DEGREES)
4     0.0      X AXIS VIEW TRANSLATION
5     -20.0    Y AXIS VIEW TRANSLATION
6     0.0      Z AXIS VIEW TRANSLATION
7     0.65     VIEW SCALE FACTOR
8     0.0      VIEWPORT LIMIT (X MINIMUM)
9     0.4      VIEWPORT LIMIT (X MAXIMUM)
10    0.55     VIEWPORT LIMIT (Y MINIMUM)
11    0.95     VIEWPORT LIMIT (Y MAXIMUM)
12    0.00     VIEWPORT LIMIT (Z MINIMUM)
13    1.00     VIEWPORT LIMIT (Z MAXIMUM)
14    -25.0    VIEW WINDOW LIMIT X MINIMUM
15    25.0     VIEW WINDOW LIMIT X MAXIMUM
16    -25.0    VIEW WINDOW LIMIT Y MINIMUM
17    25.0     VIEW WINDOW LIMIT Y MAXIMUM
18    0.0      X COORDINATE OF PROJECTION REFERENCE POINT
19    0.0      Y COORDINATE OF PROJECTION REFERENCE POINT
20    100.0    Z COORDINATE OF PROJECTION REFERENCE POINT
21    0.0      VIEW PLANE DISTANCE
22    70.0     FRONT CLIPPING PLANE DISTANCE
23    -100.0   BACK CLIPPING PLANE DISTANCE
*****************************************************************
      4        GEOMETRY VIEW NUMBER
      1        NUMBER OF SPECIFIED CHARACTER VIEW PARAMETERS
1     '*'
      7        NUMBER OF SPECIFIED INTEGER VIEW PARAMETERS
1     0        VIEW ACTIVE FLAG
2     1004     GEOMETRY ROOT STRUCTURE IDENTIFIER
3     0        VIEW PROJECTION TYPE (0=PARALLEL, 1=PERSPECTIVE)
4     7040     GEOMETRY SHADING ROOT STRUCTURE IDENTIFIER
5     7040     GEOMETRY HIDDEN SURFACE ROOT STRUCTURE IDENTIFIER
6     7041     VIEW FRAME ROOT STRUCTURE IDENTIFIER
7     7042     VIEW FRAME SUB-STRUCTURE IDENTITIER
```

```
     23       NUMBER OF SPECIFIED REAL VIEW PARAMETERS
1    -90.0    X AXIS ROTATION (DEGREES)
2    90.0     Y AXIS ROTATION (DEGREES)
3    0.0      Z AXIS ROTATION (DEGREES)
4    0.0      X AXIS VIEW TRANSLATION
5    0.0      Y AXIS VIEW TRANSLATION
6    0.0      Z AXIS VIEW TRANSLATION
7    0.95     VIEW SCALE FACTOR
8    0.0      VIEWPORT LIMIT (X MINIMUM)
9    0.4      VIEWPORT LIMIT (X MAXIMUM)
10   0.15     VIEWPORT LIMIT (Y MINIMUM)
11   0.55     VIEWPORT LIMIT (Y MAXIMUM)
12   0.00     VIEWPORT LIMIT (Z MINIMUM)
13   1.00     VIEWPORT LIMIT (Z MAXIMUM)
14   -25.0    VIEW WINDOW LIMIT X MINIMUM
15   25.0     VIEW WINDOW LIMIT X MAXIMUM
16   -25.0    VIEW WINDOW LIMIT Y MINIMUM
17   25.0     VIEW WINDOW LIMIT Y MAXIMUM
18   0.0      X COORDINATE OF PROJECTION REFERENCE POINT
19   0.0      Y COORDINATE OF PROJECTION REFERENCE POINT
20   100.0    Z COORDINATE OF PROJECTION REFERENCE POINT
21   0.0      VIEW PLANE DISTANCE
22   70.0     FRONT CLIPPING PLANE DISTANCE
23   -100.0   BACK CLIPPING PLANE DISTANCE
*****************************************************************
     5        GEOMETRY VIEW NUMBER
     1        NUMBER OF SPECIFIED CHARACTER VIEW PARAMETERS
1    '*'
     7        NUMBER OF SPECIFIED INTEGER VIEW PARAMETERS
1    0        VIEW ACTIVE FLAG
2    1005     GEOMETRY ROOT STRUCTURE IDENTIFIER
3    0        VIEW PROJECTION TYPE (0=PARALLEL, 1=PERSPECTIVE)
4    7050     GEOMETRY SHADING ROOT STRUCTURE IDENTIFIER
5    7050     GEOMETRY HIDDEN SURFACE ROOT STRUCTURE IDENTIFIER
6    7051     VIEW FRAME ROOT STRUCTURE IDENTIFIER
7    7052     VIEW FRAME SUB-STRUCTURE IDENTITIER
     23       NUMBER OF SPECIFIED REAL VIEW PARAMETERS
1    270.0    X AXIS ROTATION (DEGREES)
2    0.0      Y AXIS ROTATION (DEGREES)
3    0.0      Z AXIS ROTATION (DEGREES)
4    -20.0    X AXIS VIEW TRANSLATION
5    0.0      Y AXIS VIEW TRANSLATION
6    0.0      Z AXIS VIEW TRANSLATION
7    0.65     VIEW SCALE FACTOR
8    0.4      VIEWPORT LIMIT (X MINIMUM)
9    0.8      VIEWPORT LIMIT (X MAXIMUM)
10   0.15     VIEWPORT LIMIT (Y MINIMUM)
11   0.55     VIEWPORT LIMIT (Y MAXIMUM)
12   0.00     VIEWPORT LIMIT (Z MINIMUM)
13   1.00     VIEWPORT LIMIT (Z MAXIMUM)
14   -25.0    VIEW WINDOW LIMIT X MINIMUM
15   25.0     VIEW WINDOW LIMIT X MAXIMUM
16   -25.0    VIEW WINDOW LIMIT Y MINIMUM
17   25.0     VIEW WINDOW LIMIT Y MAXIMUM
18   0.0      X COORDINATE OF PROJECTION REFERENCE POINT
19   0.0      Y COORDINATE OF PROJECTION REFERENCE POINT
20   100.0    Z COORDINATE OF PROJECTION REFERENCE POINT
21   0.0      VIEW PLANE DISTANCE
22   70.0     FRONT CLIPPING PLANE DISTANCE
23   -100.0   BACK CLIPPING PLANE DISTANCE
*****************************************************************
* AIRCRAFT COMPONENT DATA
*****************************************************************
     8        NUMBER OF AIRCRAFT COMPONENTS
*****************************************************************
     1        COMPONENT NUMBER
```

```
     1       NUMBER OF SPECIFIED CHARACTER PARAMETERS
1   'FUSELAGE'
     20      NUMBER OF SPECIFIED INTEGER PARAMETERS
1    1011    COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2    1013    COMPONENT STRUCTURE IDENTIFIER
3    1       EXISTENCE FLAG
4    1       COMPONENT TYPE (BODY = 1, WING = 2)
5    20      NUMBER OF CROSS SECTIONS
6    8       NUMBER OF POINTS PER CROSS SECTION
7    6       NUMBER OF COMPONENT PARAMETERS
8    2       NUMBER OF SURFACE LINES IN U DIRECTION
9    2       NUMBER OF SURFACE LINES IN W DIRECTION
10   0       <FREE>
11   2       WIREFRAME COLOR INDEX
12   1       WIREFRAME LINE TYPE
13   2       LOCAL SYMMETRY FLAG
14   0       GLOBAL SYMMETRY FLAG
15   1012    COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16   0       <FREE>
17   0       <FREE>
18   0       <FREE>
19   0       <FREE>
20   0       <FREE>
     6       NUMBER OF SPECIFIED REAL PARAMETERS
1    0.0     COMPONENT ROTATION ABOUT GLOBAL X AXIS
2    0.0     COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3    0.0     COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4    0.0     COMPONENT TRANSLATION ALONG X AXIS
5    0.0     COMPONENT TRANSLATION ALONG Y AXIS
6    0.0     COMPONENT TRANSLATION ALONG Z AXIS
************************************************************
     2       COMPONENT NUMBER
     1       NUMBER OF SPECIFIED CHARACTER PARAMETERS
1   'WING'
     20      NUMBER OF SPECIFIED INTEGER PARAMETERS
1    1021    COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2    1023    COMPONENT STRUCTURE IDENTIFIER
3    1       EXISTENCE FLAG
4    2       COMPONENT TYPE (BODY = 1, WING = 2)
5    4       NUMBER OF CROSS SECTIONS
6    8       NUMBER OF POINTS PER CROSS SECTION
7    0       NUMBER OF COMPONENT PARAMETERS
8    2       NUMBER OF SURFACE LINES IN U DIRECTION
9    2       NUMBER OF SURFACE LINES IN W DIRECTION
10   0       <FREE>
11   3       WIREFRAME COLOR INDEX
12   1       WIREFRAME LINE TYPE
13   0       LOCAL SYMMETRY FLAG
14   2       GLOBAL SYMMETRY FLAG
15   1022    COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16   0       <FREE>
17   0       <FREE>
18   0       <FREE>
19   0       <FREE>
20   0       <FREE>
     6       NUMBER OF SPECIFIED REAL PARAMETERS
1    0.0     COMPONENT ROTATION ABOUT GLOBAL X AXIS
2    0.0     COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3    90.0    COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4    22.2672 COMPONENT TRANSLATION ALONG X AXIS
5    1.9     COMPONENT TRANSLATION ALONG Y AXIS
6    0.0     COMPONENT TRANSLATION ALONG Z AXIS
************************************************************
     3       COMPONENT NUMBER
     1       NUMBER OF SPECIFIED CHARACTER PARAMETERS
1   'HORZ_TAIL'
```

```
     20       NUMBER OF SPECIFIED INTEGER PARAMETERS
1    1031     COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2    1033     COMPONENT STRUCTURE IDENTIFIER
3    1        EXISTENCE FLAG
4    2        COMPONENT TYPE (BODY = 1, WING = 2)
5    4        NUMBER OF CROSS SECTIONS
6    8        NUMBER OF POINTS PER CROSS SECTION
7    0        NUMBER OF COMPONENT PARAMETERS
8    2        NUMBER OF SURFACE LINES IN U DIRECTION
9    2        NUMBER OF SURFACE LINES IN W DIRECTION
10   0        <FREE>
11   3        WIREFRAME COLOR INDEX
12   1        WIREFRAME LINE TYPE
13   0        LOCAL SYMMETRY FLAG
14   2        GLOBAL SYMMETRY FLAG
15   1032     COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16   0        <FREE>
17   0        <FREE>
18   0        <FREE>
19   0        <FREE>
20   0        <FREE>
     6        NUMBER OF SPECIFIED REAL PARAMETERS
1    0.0      COMPONENT ROTATION ABOUT GLOBAL X AXIS
2    10.0     COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3    90.0     COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4    40.4192  COMPONENT TRANSLATION ALONG X AXIS
5    1.9      COMPONENT TRANSLATION ALONG Y AXIS
6    0.0      COMPONENT TRANSLATION ALONG Z AXIS
*****************************************************************
     4        COMPONENT NUMBER
     1        NUMBER OF SPECIFIED CHARACTER PARAMETERS
1   'VERT_TAIL'
     20       NUMBER OF SPECIFIED INTEGER PARAMETERS
1    1041     COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2    1043     COMPONENT STRUCTURE IDENTIFIER
3    1        EXISTENCE FLAG
4    2        COMPONENT TYPE (BODY = 1, WING = 2)
5    4        NUMBER OF CROSS SECTIONS
6    8        NUMBER OF POINTS PER CROSS SECTION
7    0        NUMBER OF COMPONENT PARAMETERS
8    2        NUMBER OF SURFACE LINES IN U DIRECTION
9    2        NUMBER OF SURFACE LINES IN W DIRECTION
10   0        <FREE>
11   3        WIREFRAME COLOR INDEX
12   1        WIREFRAME LINE TYPE
13   0        LOCAL SYMMETRY FLAG
14   0        GLOBAL SYMMETRY FLAG
15   1042     COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16   0        <FREE>
17   0        <FREE>
18   0        <FREE>
19   0        <FREE>
20   0        <FREE>
     6        NUMBER OF SPECIFIED REAL PARAMETERS
1    0.0      COMPONENT ROTATION ABOUT GLOBAL X AXIS
2    270.0    COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3    90.0     COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4    36.82    COMPONENT TRANSLATION ALONG X AXIS
5    0.0      COMPONENT TRANSLATION ALONG Y AXIS
6    1.9      COMPONENT TRANSLATION ALONG Z AXIS
*****************************************************************
     5        COMPONENT NUMBER
     1        NUMBER OF SPECIFIED CHARACTER PARAMETERS
1   'CANOPY'
     20       NUMBER OF SPECIFIED INTEGER PARAMETERS
1    1051     COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2    1053     COMPONENT STRUCTURE IDENTIFIER
```

```
3    1        EXISTENCE FLAG
4    1        COMPONENT TYPE (BODY = 1, WING = 2)
5    10       NUMBER OF CROSS SECTIONS
6    8        NUMBER OF POINTS PER CROSS SECTION
7    0        NUMBER OF COMPONENT PARAMETERS
8    2        NUMBER OF SURFACE LINES IN U DIRECTION
9    2        NUMBER OF SURFACE LINES IN W DIRECTION
10   0        <FREE>
11   5        WIREFRAME COLOR INDEX
12   1        WIREFRAME LINE TYPE
13   2        LOCAL SYMMETRY FLAG
14   0        GLOBAL SYMMETRY FLAG
15   1052     COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16   0        <FREE>
17   0        <FREE>
18   0        <FREE>
19   0        <FREE>
20   0        <FREE>
     6        NUMBER OF SPECIFIED REAL PARAMETERS
1    0.0      COMPONENT ROTATION ABOUT GLOBAL X AXIS
2    -7.0     COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3    00.0     COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4    14.87    COMPONENT TRANSLATION ALONG X AXIS
5    0.0      COMPONENT TRANSLATION ALONG Y AXIS
6    1.27     COMPONENT TRANSLATION ALONG Z AXIS
*****************************************************************
     6        COMPONENT NUMBER
     1        NUMBER OF SPECIFIED CHARACTER PARAMETERS
1    'MISSILES'
     20       NUMBER OF SPECIFIED INTEGER PARAMETERS
1    1061     COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2    1063     COMPONENT STRUCTURE IDENTIFIER
3    1        EXISTENCE FLAG
4    1        COMPONENT TYPE (BODY = 1, WING = 2)
5    5        NUMBER OF CROSS SECTIONS
6    5        NUMBER OF POINTS PER CROSS SECTION
7    0        NUMBER OF COMPONENT PARAMETERS
8    2        NUMBER OF SURFACE LINES IN U DIRECTION
9    2        NUMBER OF SURFACE LINES IN W DIRECTION
10   0        <FREE>
11   6        WIREFRAME COLOR INDEX
12   1        WIREFRAME LINE TYPE
13   2        LOCAL SYMMETRY FLAG
14   2        GLOBAL SYMMETRY FLAG
15   1062     COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16   0        <FREE>
17   0        <FREE>
18   0        <FREE>
19   0        <FREE>
20   0        <FREE>
     6        NUMBER OF SPECIFIED REAL PARAMETERS
1    0.0      COMPONENT ROTATION ABOUT GLOBAL X AXIS
2    00.0     COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3    00.0     COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4    34.585   COMPONENT TRANSLATION ALONG X AXIS
5    15.0     COMPONENT TRANSLATION ALONG Y AXIS
6    0.00     COMPONENT TRANSLATION ALONG Z AXIS
*****************************************************************
     7        COMPONENT NUMBER
     1        NUMBER OF SPECIFIED CHARACTER PARAMETERS
1    'INLET'
     20       NUMBER OF SPECIFIED INTEGER PARAMETERS
1    1071     COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2    1073     COMPONENT STRUCTURE IDENTIFIER
3    1        EXISTENCE FLAG
4    1        COMPONENT TYPE (BODY = 1, WING = 2)
5    6        NUMBER OF CROSS SECTIONS
```

```
6     8         NUMBER OF POINTS PER CROSS SECTION
7     0         NUMBER OF COMPONENT PARAMETERS
8     2         NUMBER OF SURFACE LINES IN U DIRECTION
9     2         NUMBER OF SURFACE LINES IN W DIRECTION
10    0         <FREE>
11    1         WIREFRAME COLOR INDEX
12    1         WIREFRAME LINE TYPE
13    2         LOCAL SYMMETRY FLAG
14    0         GLOBAL SYMMETRY FLAG
15    1072      COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16    0         <FREE>
17    0         <FREE>
18    0         <FREE>
19    0         <FREE>
20    0         <FREE>
      6         NUMBER OF SPECIFIED REAL PARAMETERS
1     0.0       COMPONENT ROTATION ABOUT GLOBAL X AXIS
2     00.0      COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3     00.0      COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4     20.22     COMPONENT TRANSLATION ALONG X AXIS
5     00.0      COMPONENT TRANSLATION ALONG Y AXIS
6     0.00      COMPONENT TRANSLATION ALONG Z AXIS
************************************************************
      8         COMPONENT NUMBER
      1         NUMBER OF SPECIFIED CHARACTER PARAMETERS
1    'STRAKE'
      20        NUMBER OF SPECIFIED INTEGER PARAMETERS
1     1081      COMPONENT ATTRIBUTE STRUCTURE IDENTIFIER
2     1083      COMPONENT STRUCTURE IDENTIFIER
3     1         EXISTENCE FLAG
4     2         COMPONENT TYPE (BODY = 1, WING = 2)
5     4         NUMBER OF CROSS SECTIONS
6     8         NUMBER OF POINTS PER CROSS SECTION
7     0         NUMBER OF COMPONENT PARAMETERS
8     2         NUMBER OF SURFACE LINES IN U DIRECTION
9     2         NUMBER OF SURFACE LINES IN W DIRECTION
10    0         <FREE>
11    2         WIREFRAME COLOR INDEX
12    1         WIREFRAME LINE TYPE
13    0         LOCAL SYMMETRY FLAG
14    2         GLOBAL SYMMETRY FLAG
15    1082      COMPONENT LOCAL SYMMETRY STRUCTURE IDENTIFIER
16    0         <FREE>
17    0         <FREE>
18    0         <FREE>
19    0         <FREE>
20    0         <FREE>
      6         NUMBER OF SPECIFIED REAL PARAMETERS
1     0.0       COMPONENT ROTATION ABOUT GLOBAL X AXIS
2     00.0      COMPONENT ROTATION ABOUT GLOBAL Y AXIS
3     90.0      COMPONENT ROTATION ABOUT GLOBAL Z AXIS
4     18.19     COMPONENT TRANSLATION ALONG X AXIS
5     1.73      COMPONENT TRANSLATION ALONG Y AXIS
6     0.00      COMPONENT TRANSLATION ALONG Z AXIS
```

APPENDIX H. EXAMPLE ACSYNT GEOMETRY COMMON BLOCK DATA FILE

This appendix contains an example ACSYNT data file as described in "Data-flow and Geometric Modeling" on page 103. An ACSYNT common block data file is generated every time ACSYNT is executed and contains data copied from the two ACSYNT common blocks "/GEOMCM/" and "/GEOSAV/" found in ACSYNT "GEOMETRY" module. The file serves as the link between ACSYNT and ACSYNT/VPI and is used to load two ACSYNT/VPI common blocks which are identical to "/GEOMCM/" and "/GEOSAV/". Once ACSYNT is merged with ACSYNT/VPI, the common block file will not be required.

A geometry common block file is created by the subroutine GMOUT called from ACSYNT's "GEOMETRY" module. GMOUT simply writes the contents of ACSYNT's two geometry common blocks to a disk file in the format given below:

Column	*Description*
1-16	common block data (real or integer)
17	blank space
14-80	data description string (ignored by ACSYNT/VPI)

The common block data used by ACSYNT/VPI includes the following:

- for the fuselage:
  - fuselage length
  - maximum body diameter
  - afterbody length to diameter ratio (fineness ratio)
- for the wings, tail surfaces, and canards
  - sweep angle
  - root chord
  - wing aspect ratio
  - span
  - tip to chord ratio at the root

- tip to chord ratio at the tip

In the X1.0.0 version of ACSYNT/VPI, the ACSYNT geometry common block data is read by the subroutine "RDACIN" found in module "ADA". RDACIN was generated from GMOUT by changing all of GMOUT's "WRITE" statements to "READ" statements.

NOTE: In the following file, "????????" next to an item indicates that the item is undefined but was included for sake of completeness.

```
   0.8527435 RPOD   (1) - R(4), POD RADIUS
   3.0000000 AR     (1) - R(4), ASPECT RATIO
   0.0400000 TCROOT(1) - R(4), ROOT T/C
   0.0400000 TCTIP  (1) - R(4), TIP T/C
   0.2275000 TR     (1) - R(4), TAPER RATIO
   0.0000000 XPOD   (1) - R(4), POD FACE X LOCATION
  32.1831970 SWEEPW(1) - R(4), C/4 SWEEP ANGLE
  11.3201714 CBAR   (1) - R(4), MEAN AERO CHORD
  16.2932739 CROOT  (1) - R(4), ROOT CHORD
  30.0000000 SPAN   (1) - R(4), PLANAR SURFACE SPAN
  18.1941071 ROOTLE(1) - R(4), LEAD EDGE ROOT LOC
  25.9963379 XIQCB  (1) - R(4), MAC C/4 LOCATION
 333.0041500 SWWET  (1) - R(4), WETTED SURFACE AREA
 300.0000000 SWING  (1) - R(4), WING AREA
   0.0000000 ZW     (1) - R(4), ROOT CHORD ELEVATION
   1.0000000 RPOD   (2) - R(4), POD RADIUS
   2.1140003 AR     (2) - R(4), ASPECT RATIO
   0.0600000 TCROOT(2) - R(4), ROOT T/C
   0.0350000 TCTIP  (2) - R(4), TIP T/C
   0.3900000 TR     (2) - R(4), TAPER RATIO
   0.2203055 XPOD   (2) - R(4), POD FACE X LOCATION
  32.2727356 SWEEPW(2) - R(4), C/4 SWEEP ANGLE
   5.8416986 CBAR   (2) - R(4), MEAN AERO CHORD
   7.8982859 CROOT  (2) - R(4), ROOT CHORD
  11.6043873 SPAN   (2) - R(4), PLANAR SURFACE SPAN
  38.4446106 ROOTLE(2) - R(4), LEAD EDGE ROOT LOC
  41.9828796 XIQCB  (2) - R(4), MAC C/4 LOCATION
 115.3180390 SWWET  (2) - R(4), WETTED SURFACE AREA
  63.6999969 SWING  (2) - R(4), WING AREA
   0.0000000 ZW     (2) - R(4), ROOT CHORD ELEVATION
   1.0000000 RPOD   (3) - R(4), POD RADIUS
   2.4000000 AR     (3) - R(4), ASPECT RATIO
   0.0530000 TCROOT(3) - R(4), ROOT T/C
   0.0300000 TCTIP  (3) - R(4), TIP T/C
   0.4370000 TR     (3) - R(4), TAPER RATIO
   0.7796944 XPOD   (3) - R(4), POD FACE X LOCATION
  43.2261810 SWEEPW(3) - R(4), C/4 SWEEP ANGLE
   6.8374844 CBAR   (3) - R(4), MEAN AERO CHORD
   9.0531225 CROOT  (3) - R(4), ROOT CHORD
   8.4170351 SPAN   (3) - R(4), PLANAR SURFACE SPAN
  34.5567932 ROOTLE(3) - R(4), LEAD EDGE ROOT LOC
  36.7581787 XIQCB  (3) - R(4), MAC C/4 LOCATION
 116.9370420 SWWET  (3) - R(4), WETTED SURFACE AREA
  54.7500000 SWING  (3) - R(4), WING AREA
   1.8999996 ZW     (3) - R(4), ROOT CHORD ELEVATION
   0.9565217 RPOD   (4) - R(4), POD RADIUS
```

```
     0.0000000 AR     (4) - R(4), ASPECT RATIO
     0.0000000 TCROOT(4) - R(4), ROOT T/C
     0.0000000 TCTIP (4) - R(4), TIP T/C
     0.0000000 TR     (4) - R(4), TAPER RATIO
     1.0000000 XPOD   (4) - R(4), POD FACE X LOCATION
     0.0000000 SWEEPW(4) - R(4), C/4 SWEEP ANGLE
     0.0000000 CBAR   (4) - R(4), MEAN AERO CHORD
     0.0000000 CROOT  (4) - R(4), ROOT CHORD
     0.0000000 SPAN   (4) - R(4), PLANAR SURFACE SPAN
     0.0000000 ROOTLE(4) - R(4), LEAD EDGE ROOT LOC
     0.0000000 XIQCB  (4) - R(4), MAC C/4 LOCATION
     0.0000000 SWWET  (4) - R(4), WETTED SURFACE AREA
     0.0000000 SWING  (4) - R(4), WING AREA
     0.0000000 ZW     (4) - R(4), ROOT CHORD ELEVATION
     3.0000000 FRAB    - R, FINENESS RATIO OF AFTBDY
     4.7999964 FRN     - R, FINENESS RATIO OF NOSE
    12.5210524 FRATIO  - R, TOTAL FINENESS RATIO
     1.0000000 GSTCWG  - R, ???????
     0.1200000 LFLAPC  - R, CHORD FRAC COVRD BY LE FLAPS
     0.2750000 TFLAPC  - R, CHORD FRAC COVRD BY TE FLAPS
     0.1541840 FUFRAC  - R, WING FUEL FRACTION
     1.0000000 CREW    - R, NUMBER OF CREW MEMBERS
     1.0000000 EN      - R, NUMBER OF ENGINE PODS?
     0.0000000 ENWING  - R, NUMBER OF WING PODS?
     0.0000000 PASS    - R, NUMBER OF PASSENGERS
     0.0000000 COACH   - R, NUMBER OF COACH PASSENGERS
     1.0000000 VTNO    - R, NUMBER OF VERT. TAILS
    47.5800018 BODL    - R, FUSELAGE BODY LENGTH
    22.1985931 PODLNG  - R, POD LENGTH
     2.4452362 PODRAD  - R, POD RADIUS
    18.7561798 XWBJWG  - R, ???????
     3.8000002 BDMAX   - R, MAXIMUM BODY DIAMETER
     0.0000000 XPAY    - R, LENGTH OF CARGO PAYLOAD
     0.0000000 YPAY    - R, WIDTH OF CARGO PAYLOAD
     0.0000000 ZPAY    - R, HEIGHT OF CARGO PAYLOAD
     4.2525864 DIA1    - R, FUSELAGE ENGINE DIAMETER
    14.7990627 XLENG   - R, FUSELAGE ENGINE LENGTH
  1000.0000000 DX1     - R, (ACTUAL BDY LEN)-(REQD LEN) TO ENCLOSE PAYLOAD
  1000.0000000 DX2     - R, (ACTUAL BDY LEN)-(REQD LEN) TO PAYLOAD/FUEL OVERLAP
    17.9400024 DX3     - R, DELTAX DUE TO REQD FINENESS RATIO
     0.0000000 DX4     - R, ???????
     0.3000002 DD1     - R, (ACTUAL BDY DIA)-(REQD DIA) TO ENCLOSE RADAR & CREW
    -1.0904722 DD2     - R, (ACTUAL BDY DIA)-(REQD DIA) TO ENCLOSE PAYLOAD
     2.8000002 DD3     - R, (ACTUAL BDY DIA)-(REQD DIA) TO ENCLOSE ENGINES
     0.0000000 DD4     - R, ???????
   688.7773440 SWETB   - R, FUSELAGE BODY WETTED AREA
   334.0961910 SWETP   - R, POD WETTED AREA
   401.4541020 VOLB    - R, FUSELAGE BODY VOLUME
  -129.0590060 DELVOL  - R, (ACTL FUEL VOL)-(REQD VOL IF ALL FUEL IN WING)
     0.0000000 WFUELV  - R, ???????
  7476.6757800 WFUEL   - R, TOTAL FUEL WEIGHT
     0.0000000 SEXT    - R, STRAKE AREA
     0.0000000 XEXT    - R, STRAKE CENTROID LOCATION
     0.0000000 SLEXT   - R, STRAKE SWEEP ANGLE
     0.0000000 SAFT    - R, AFT STRAKE AREA
     0.0000000 XAFT    - R, AFT STRAKE CENTROID LOCATION
 30608.1094000 WGTO    - R, AIRCRAFT WEIGHT (LBS)
    58.6997681 ACDEN   - R, AIRCRAFT DENSITY (LBS/FT**3)
             1 NPODS   - I, NUMBER OF PODS
             0 ICAN    - I, CANARD INDICATOR
             0 NCAN    - I, CANARD LOCATION
             0 IFLEX   - I, STRAKE BLUNTNESS FLAG
     9.5159998 XLEPOD( 1) - R(10), POD X LOCATION
     0.0000000 XLEPOD( 2) - R(10), POD X LOCATION
     0.0000000 XLEPOD( 3) - R(10), POD X LOCATION
     0.0000000 XLEPOD( 4) - R(10), POD X LOCATION
     0.0000000 XLEPOD( 5) - R(10), POD X LOCATION
```

```
 0.0000000 XLEPOD( 6) - R(10), POD X LOCATION
 0.0000000 XLEPOD( 7) - R(10), POD X LOCATION
 0.0000000 XLEPOD( 8) - R(10), POD X LOCATION
 0.0000000 XLEPOD( 9) - R(10), POD X LOCATION
 0.0000000 XLEPOD(10) - R(10), POD X LOCATION
 0.0000000 YROOT (1) - R(4), Y LOC OF ROOT CHORD OF VERTICAL TAIL
 0.0000000 ZROOT (1) - R(4), ELEV OF ROOT CHORD ABOVE FUSELAGE PLANE
22.2674255 XIW   (1) - R(4), LOCATION OF ROOT C/4 FOR PLANAR SURFACES
         0 SIZIT (1) - L(4), LOGICAL--0,INIT. EST --1,RECOMP EVERY PASS
 0.4680000 XWLOC (1) - R(4), LOCATION OF ROOT CHORD'TRAILING EDGE
95.0893707 WVOL  (1) - R(4), VOLUME ENCLOSED BY SURFACE
 0.0000000 YROOT (2) - R(4), Y LOC OF ROOT CHORD OF VERTICAL TAIL
 0.0000000 ZROOT (2) - R(4), ELEV OF ROOT CHORD ABOVE FUSELAGE PLANE
40.4191895 XIW   (2) - R(4), LOCATION OF ROOT C/4 FOR PLANAR SURFACES
         0 SIZIT (2) - L(4), LOGICAL--0,INIT. EST --1,RECOMP EVERY PASS
 0.9740000 XWLOC (2) - R(4), LOCATION OF ROOT CHORD'TRAILING EDGE
13.2680073 WVOL  (2) - R(4), VOLUME ENCLOSED BY SURFACE
 0.0000000 YROOT (3) - R(4), Y LOC OF ROOT CHORD OF VERTICAL TAIL
 1.0000000 ZROOT (3) - R(4), ELEV OF ROOT CHORD ABOVE FUSELAGE PLANE
33.3200836 XIW   (3) - R(4), LOCATION OF ROOT C/4 FOR PLANAR SURFACES
         0 SIZIT (3) - L(4), LOGICAL--0,INIT. EST --1,RECOMP EVERY PASS
 0.8430000 XWLOC (3) - R(4), LOCATION OF ROOT CHORD'TRAILING EDGE
11.6237297 WVOL  (3) - R(4), VOLUME ENCLOSED BY SURFACE
 0.0000000 YROOT (4) - R(4), Y LOC OF ROOT CHORD OF VERTICAL TAIL
 0.0000000 ZROOT (4) - R(4), ELEV OF ROOT CHORD ABOVE FUSELAGE PLANE
 0.0000000 XIW   (4) - R(4), LOCATION OF ROOT C/4 FOR PLANAR SURFACES
         0 SIZIT (4) - L(4), LOGICAL--0,INIT. EST --1,RECOMP EVERY PASS
 0.0000000 XWLOC (4) - R(4), LOCATION OF ROOT CHORD'TRAILING EDGE
 0.0000000 WVOL  (4) - R(4), VOLUME ENCLOSED BY SURFACE
 1.0000000 CVHT   - R, VOLUME COEFF. OF HORIZ. TAIL
 0.1000000 CVVT   - R, VOLUME COEFF. OF VERT. TAIL
 1.0000000 CVCAN  - R, VOLUME COEFF. OF CANARD
 5.8999996 LRADAR - R, LENGTH OF RADAR
 2.6000004 DRADAR - R, DIAM OF RADAR DISH LOCATED IN NOSE
 3.5000000 XCREW  - R, LENGTH OF CREW COMPARTMENT
 3.5000000 DCRWRQ - R, WIDTH OF CREW COMPARTMENT
 4.8000002 FNDES  - R, BODY SHAPE PARAMETER
49.0000000 FDENWG - R, DENSITY OF FUEL STORED IN WINGS
 0.0000000 HTFRAC - R, HORIZ TAIL SIZE FACTOR--IF 0 SPECIFIED BY CVHT
 0.0000000 VTFRAC - R, (VERT. TAIL AREA)/(WING AREA)
 0.0000000 CFRAC  - R, (CANARD AREA)/(WING AREA)
 0.3260000 WFFRAC - R, FRAC OF AVAIL WING FUEL VOL TO BE FILLED WITH FUEL
 4.0000000 SETABF - R, SEATS ABREAST IN FIRST CLASS
 6.0000000 SETABC - R, SEATS ABREAST IN COACH
38.0000000 SEATPF - R, SEAT PITCH IN FIRST CLASS
32.0000000 SEATPC - R, SEAT PITCH IN COACH
18.0000000 SEATWC - R, SEAT WIDTH IN COACH
18.0000000 AISLEF - R, AISLE WIDTH IN FIRST CLASS
18.0000000 AISLEC - R, AISLE WIDTH IN COACH
 1.4659996 SFFACT - R, SURFACE AREA COMPLEXITY FACTOR
 1.0000000 WNGFAC - R, WING SIZING FACTOR
 1.0000000 DENGBY - R, DIAMETER OF ENGINE BAY
 0.0000000 YAW    - R, YAW ANGLE OF AN OBLIQUE WING AIRCRAFT
 0.0000000 CLIND  - R, ??????????????
 0.0000000 H1     - R, ??????????????
 0.0000000 ENGMUL - R, ??????????????
 0.0000000 ATHETA - R, ??????????????
 0.0000000 FMODE  - R, ??????????????
 0.0000000 H2MAX  - R, ??????????????
 0.0400000 TCRARI - R, INTERMEDIATE ROOT T/C
 0.0400000 TCTARI - R, INTERMEDIATE TIP T/C
 0.0000000 WALL   - R, FUSELAGE WALL THICKNESS
         5 NTHETA - I, # OF MERIDIAN LINES ON HALF OF FUSELAGE
 0.0000000 XELEC (1) - R(2), LENGTH OF ELECTR.BAY
 0.0000000 VELEC (1) - R(2), VOLUME OF ELECTR.BAY
 0.6800000 FFRAC (1) - R(2), FRAC OF FUEL IN FUSELAGE FUEL BAY
49.0000000 FDEN  (1) - R(2), FUEL DENSITY IN EACH FUSELAGE FUEL BAY
```

```
 0.0000000 XELEC (2) - R(2), LENGTH OF ELECTR.BAY
 0.0000000 VELEC (2) - R(2), VOLUME OF ELECTR.BAY
 0.3200000 FFRAC (2) - R(2), FRAC OF FUEL IN FUSELAGE FUEL BAY
49.0000000 FDEN  (2) - R(2), FUEL DENSITY IN EACH FUSELAGE FUEL BAY
 0.2000000 XREF  ( 1) - R(10), POD X REFERENCE
-1.0000000 YREF  ( 1) - R(10), POD Y REFERENCE
-1.5707951 ZREF  ( 1) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 1) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 2) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 2) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 2) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 2) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 3) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 3) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 3) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 3) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 4) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 4) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 4) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 4) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 5) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 5) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 5) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 5) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 6) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 6) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 6) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 6) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 7) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 7) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 7) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 7) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 8) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 8) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 8) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 8) - R(10), POD TYPE ARRAY
 0.0000000 XREF  ( 9) - R(10), POD X REFERENCE
 0.0000000 YREF  ( 9) - R(10), POD Y REFERENCE
 0.0000000 ZREF  ( 9) - R(10), POD Z REFERENCE
 0.0000000 PODCOD( 9) - R(10), POD TYPE ARRAY
 0.0000000 XREF  (10) - R(10), POD X REFERENCE
 0.0000000 YREF  (10) - R(10), POD Y REFERENCE
 0.0000000 ZREF  (10) - R(10), POD Z REFERENCE
 0.0000000 PODCOD(10) - R(10), POD TYPE ARRAY
 0.0000000 XLEG  (1) - R(3), ?????
 0.0000000 XOBJ  (1) - R(3), ?????
 0.0000000 XLEG  (2) - R(3), ?????
 0.0000000 XOBJ  (2) - R(3), ?????
 0.0000000 XLEG  (3) - R(3), ?????
 0.0000000 XOBJ  (3) - R(3), ?????
         0 TAND   - L, LOGICAL--TANDEM SEATING
         2 OUTCOD - I, OUTPUT CONTROL PARAM-- OUTPUT LEVEL GENERATED
         0 OBLIQE - L, LOGICAL--OBLIQUE WING?
         0 KERROR - I, ERROR PRINT CODE--1 PRINT --0 SUPRESS PRINT
         4 ICASE  - I, TELLS WHICH ARE DESIGN VARIABLES
         0 NAISLE - I, ??????????????
         6 BDYTYP(1) - I(8), TYPE OF FUSELAGE BAY
         2 BDYTYP(2) - I(8), TYPE OF FUSELAGE BAY
         4 BDYTYP(3) - I(8), TYPE OF FUSELAGE BAY
         4 BDYTYP(4) - I(8), TYPE OF FUSELAGE BAY
         1 BDYTYP(5) - I(8), TYPE OF FUSELAGE BAY
         0 BDYTYP(6) - I(8), TYPE OF FUSELAGE BAY
         0 BDYTYP(7) - I(8), TYPE OF FUSELAGE BAY
         0 BDYTYP(8) - I(8), TYPE OF FUSELAGE BAY
 0.6900000 SFACT (1) - R(5), SCALE FACTOR
 1.5699997 SFACT (2) - R(5), SCALE FACTOR
 1.0670004 SFACT (3) - R(5), SCALE FACTOR
```

```
0.0000000 SFACT (4) - R(5), SCALE FACTOR
0.7600000 SFACT (5) - R(5), SCALE FACTOR
0.0000000 DUMSAV( 1) - R(16), ????????????????
0.0000000 DUMSAV( 2) - R(16), ????????????????
0.0000000 DUMSAV( 3) - R(16), ????????????????
0.0000000 DUMSAV( 4) - R(16), ????????????????
0.0000000 DUMSAV( 5) - R(16), ????????????????
0.0000000 DUMSAV( 6) - R(16), ????????????????
0.0000000 DUMSAV( 7) - R(16), ????????????????
0.0000000 DUMSAV( 8) - R(16), ????????????????
0.0000000 DUMSAV( 9) - R(16), ????????????????
0.0000000 DUMSAV(10) - R(16), ????????????????
0.0000000 DUMSAV(11) - R(16), ????????????????
0.0000000 DUMSAV(12) - R(16), ????????????????
0.0000000 DUMSAV(13) - R(16), ????????????????
0.0000000 DUMSAV(14) - R(16), ????????????????
0.0000000 DUMSAV(15) - R(16), ????????????????
0.0000000 DUMSAV(16) - R(16), ????????????????
```

APPENDIX I. ACSYNT/VPI MENU TIER CHARTS

The menu tier charts contained in this appendix illustrate the hierarchical relationship of the ACSYNT/VPI menus. Some of the menus contain items which are undefined at this writing but are included for design purposes.

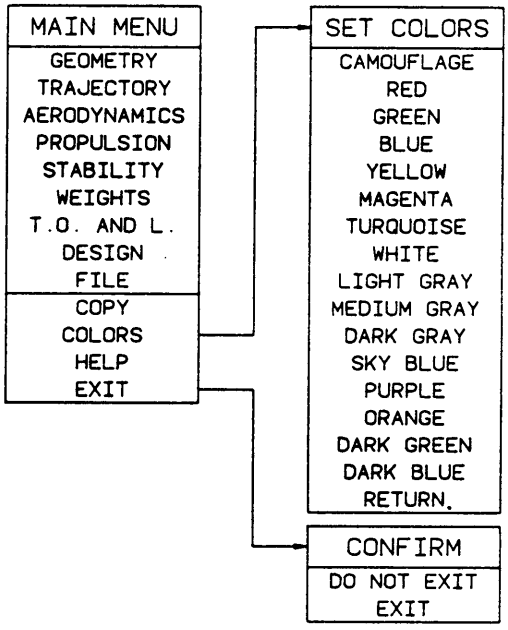


Figure 58. Menu structure tier chart.

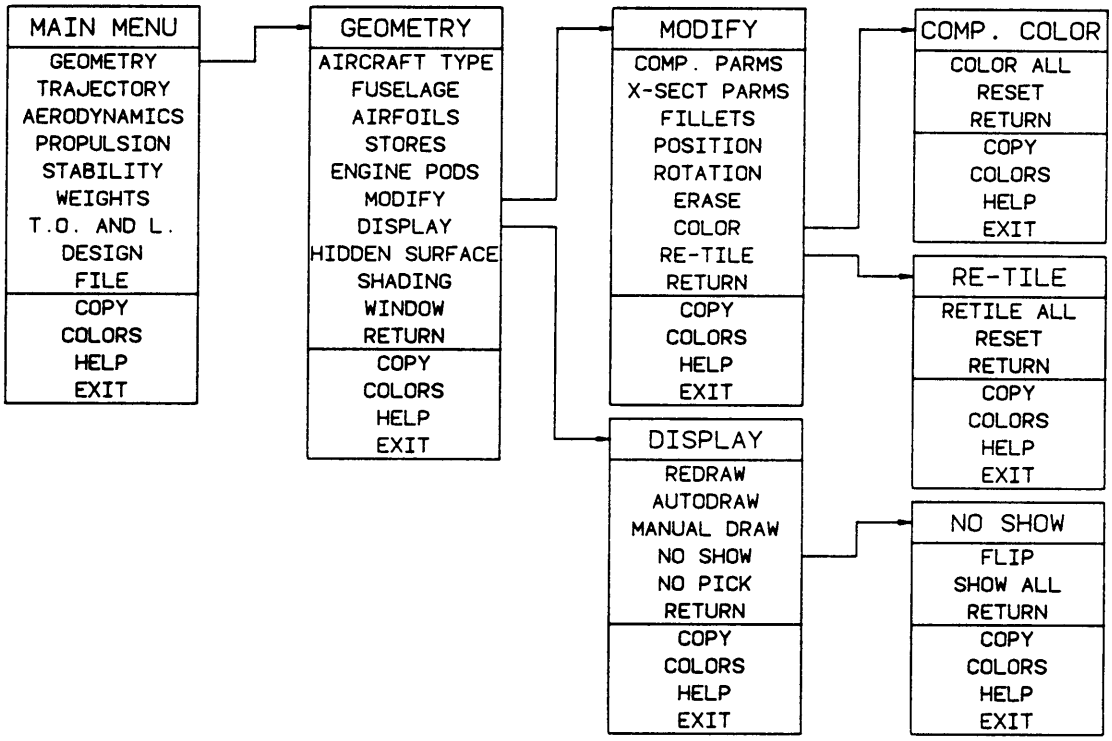


Figure 59. Menu structure tier chart (continued).

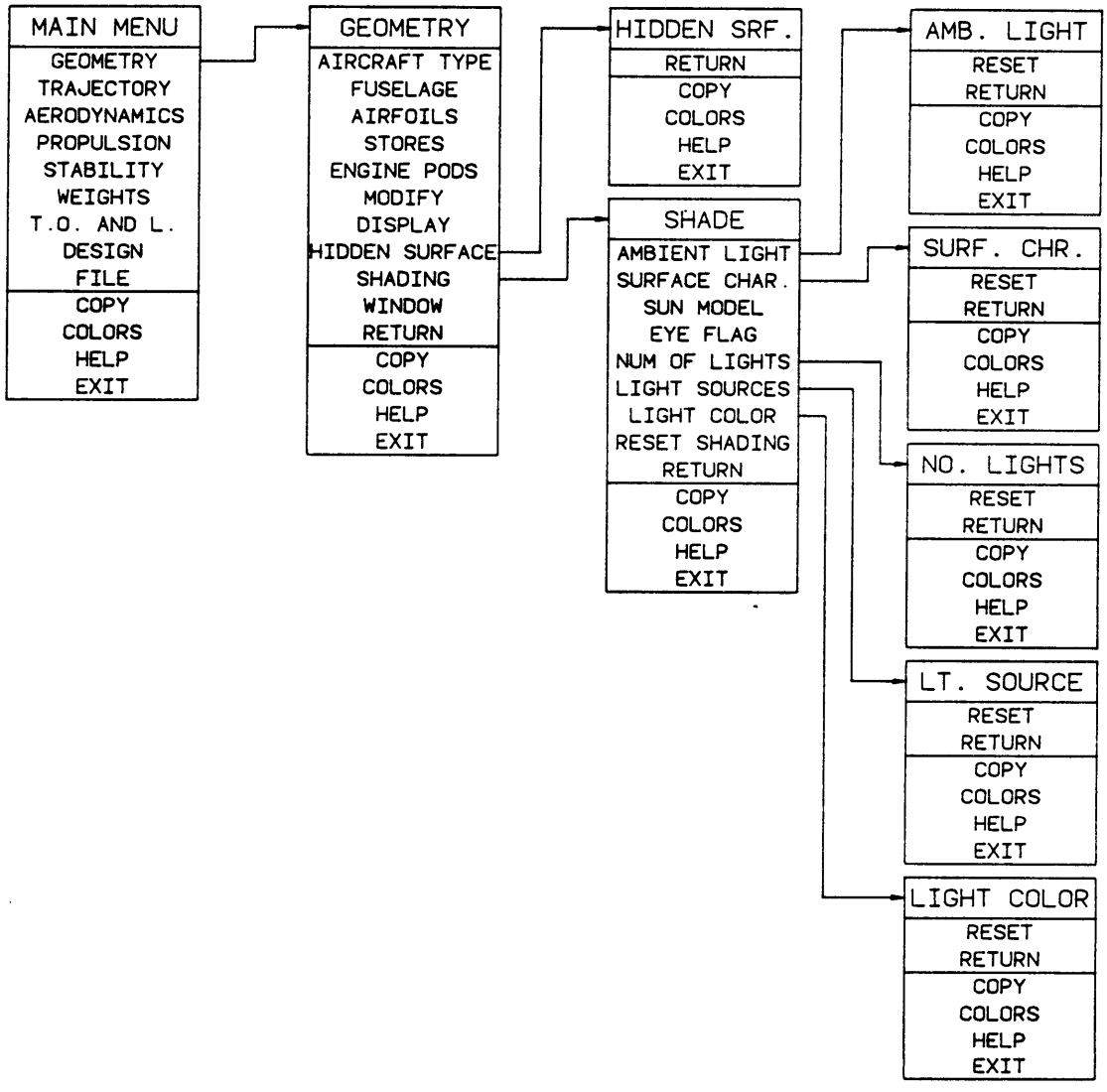


Figure 60. Menu structure tier chart (continued).

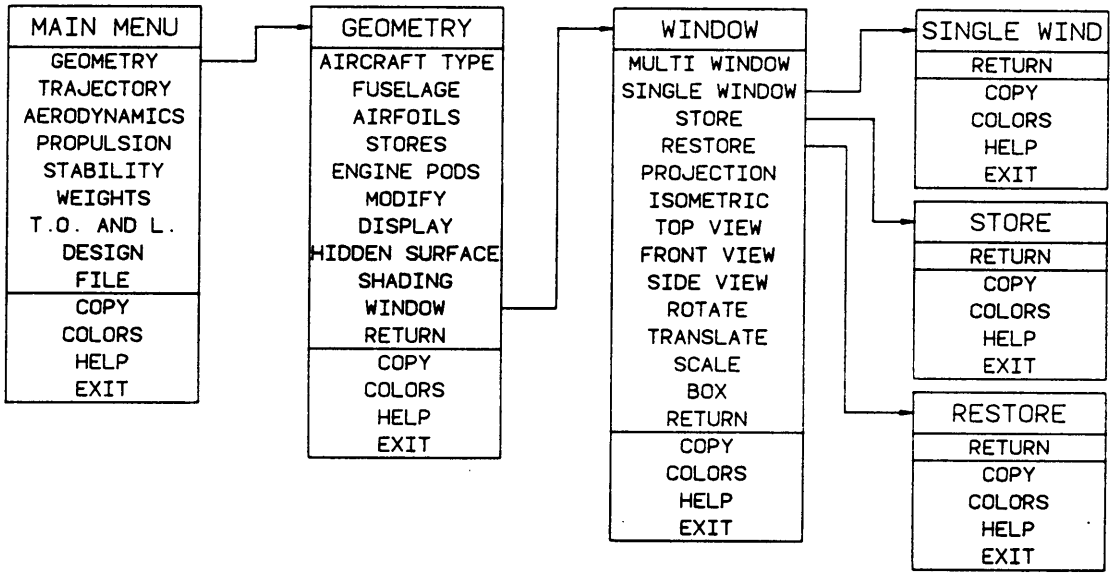


Figure 61. Menu structure tier chart (continued).

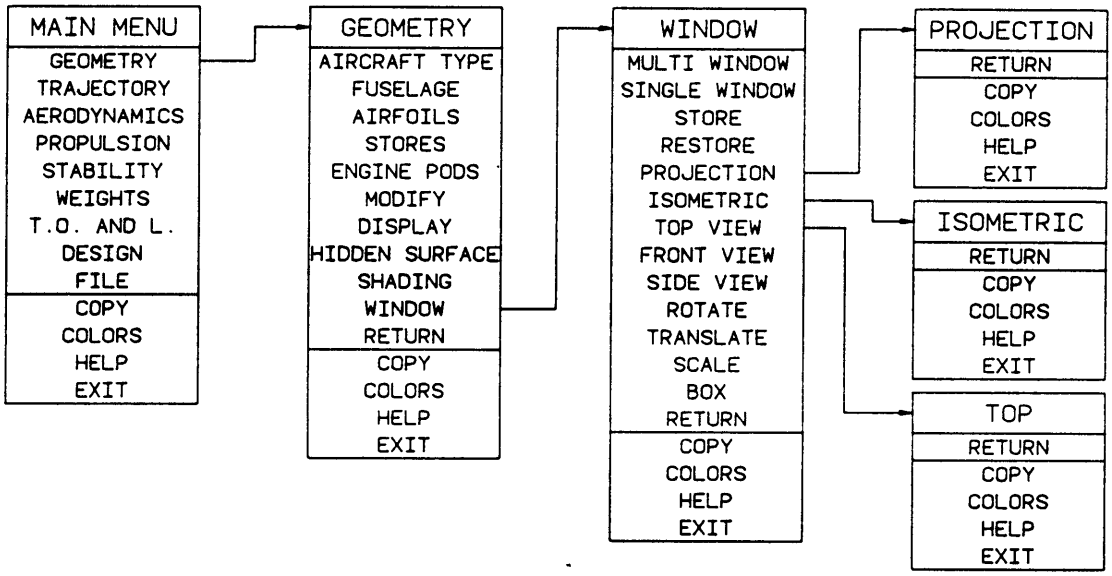


Figure 62. Menu structure tier chart (continued).

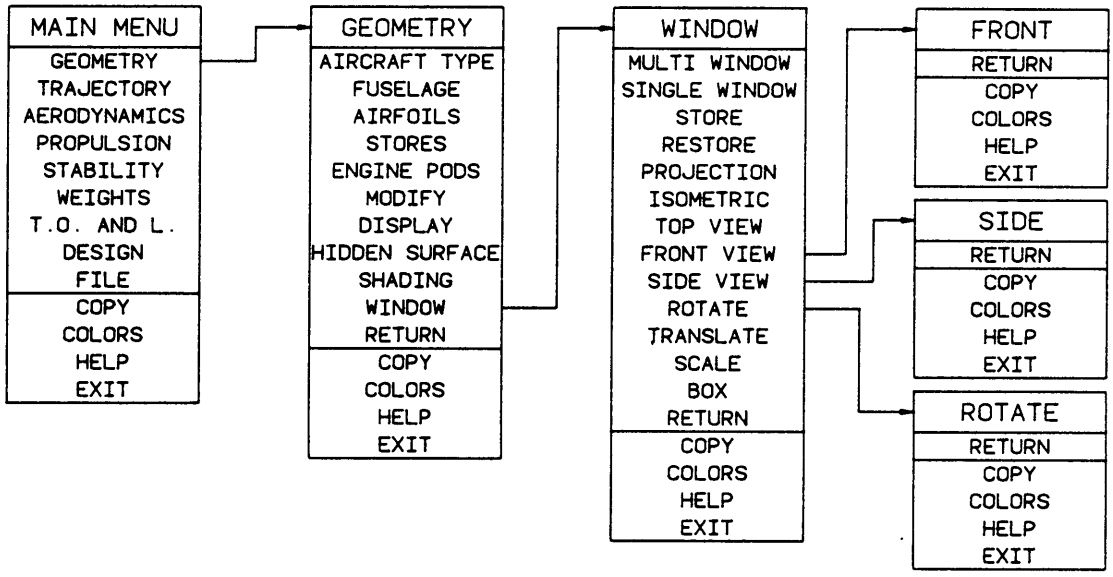


Figure 63. Menu structure tier chart (continued).

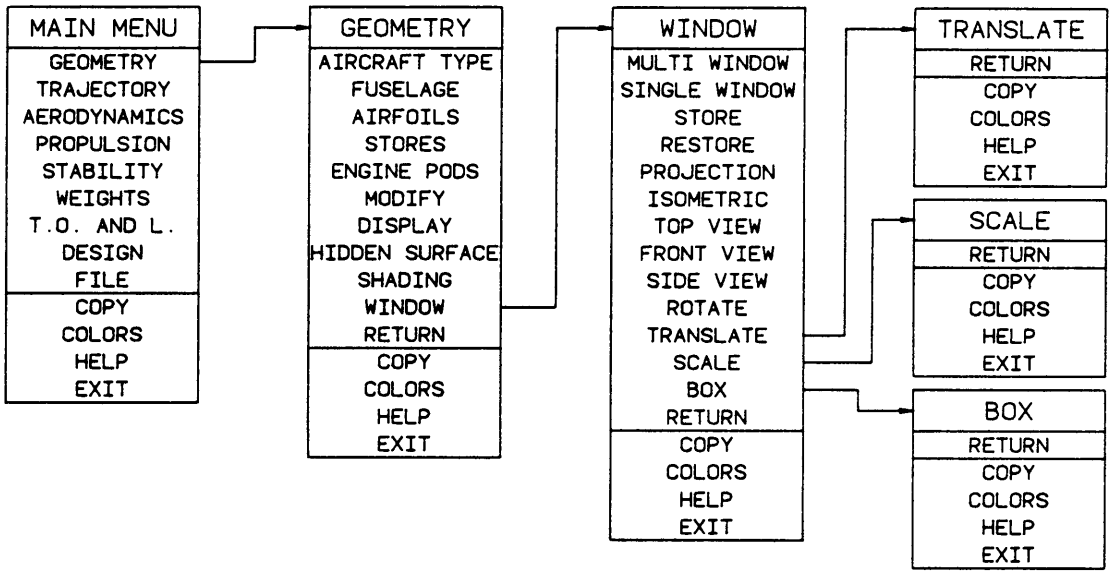


Figure 64. Menu structure tier chart (continued).

APPENDIX J. FUNCTIONAL DESCRIPTIONS FOR ACSYNT/VPI SUBROUTINES

The following pages list all of the ACSYNT/VPI source files and the subroutines which they contain. The name of each subroutine is followed by a short functional description. All ACSYNT/VPI source files are located in the ACSYNT/VPI online software development library and can be checked out as described in "Appendix M. Software Development Library User's Guide" on page 257.

J.1 AAMBL NASA

Subroutine	*Functional Description*
AAMBL	AMBIENT LIGHT MODULE
MAMBL	MENU DRIVER FOR AMBIENT LIGHT FRACTION
EXAMBL	EXECUTES AMBIENT LIGHT MENU
INAMBL	GETS AND PROCESSES AMBIENT LIGHT MENU INPUT
PAMBL	PROCESSES AMBIENT LIGHT MENU INPUT
PIAMBL	PROCESSES AN AMBIENT LIGHT MENU ITEM

J.2 ACFRM NASA

Subroutine	*Functional Description*
ACFRM	ASKS THE USER TO CONFIRM AN OPERATION
EXCFRM	EXECUTES CONFIRMATION MENU
INCFRM	GETS AND PROCESSES CONFIRMATION MENU INPUT
PCFRM	PROCESSES CONFIRMATION MENU INPUT
PICFRM	PROCESSES CONFIRMATION MENU ITEM

J.3 ACMFG NASA

Subroutine	*Functional Description*
ACMFG	SETS RGB COLOR VALUES TO CAMOUFLAGE

J.4 ACPCR NASA

Subroutine	*Functional Description*
ACPCR	ALLOWS USER TO CHANGE A COMPONENTS COLOR
MCPCR	COMPONENT COLOR MENU DRIVER
EXCPCR	EXECUTES COMPONENT COLOR MENU
INCPCR	GETS AND PROCESSES COMPONENT COLOR MENU INPUT
PCPCR	PROCESSES A COMPONENT COLOR MENU ITEM
PICPCR	PROCESSES A COMPONENT COLOR MENU ITEM

J.5 ADISP NASA

Subroutine	*Functional Description*
ADISP	ALLOWS USER TO CHANGE THE GEOMETRY DISPLAY OPTIONS
MDISP	GEOMETRY DISPLAY OPTIONS MENU DRIVER
EXDISP	EXECUTES GEOMETRY DISPLAY OPTIONS MENU
INDISP	GETS AND PROCESSES GEOMETRY DISPLAY OPTIONS MENU INPUT
PDISP	PROCESSES GEOMETRY DISPLAY OPTIONS MENU INPUT
PIDISP	PROCESSES A GEOMETRY DISPLAY OPTIONS MENU ITEM

J.6 AEYEF NASA

Subroutine	*Functional Description*
AEYEF	SWITCHES CURRENT SHADING EYE FLAG VALUE BETWEEN TRUE AND FALSE

J.7 AGEOM NASA

Subroutine	*Functional Description*
AGEOM	AIRCRAFT GEOMETRY MENU MODULE
MGEOM	GEOMETRY MENU DRIVER
EXGEOM	EXECUTES GEOMETRY MENU
INGEOM	GETS AND PROCESSES GEOMETRY MENU INPUT
PGEOM	PROCESSES GEOMETRY MENU INPUT
PIGEOM	PROCESSES A GEOMETRY MENU PICK

J.8 AHSRF NASA

Subroutine	*Functional Description*
AHSRF	HIDDEN SURFACE DISPLAY MODULE
OPHSRF	PREPARES FOR HIDDEN SURFACE ELIMINATION
MHSRF	HIDDEN SURFACE MENU DRIVER
EXHSRF	EXECUTES HIDDEN SURFACE MENU
INHSRF	GETS AND PROCESSES HIDDEN SURFACE MENU INPUT
PHSRF	PROCESSES A HIDDEN SURFACE MENU PICK
PIHSRF	PROCESSES A PICKED HIDDEN SURFACE MENU ITEM
STHSPK	STORES A PICKED AIRCRAFT COMPONENT NUMBER AND RESETS THE HIDDEN SURFACE FLAG
PHSVL	PROCESSES HIDDEN SURFACE VALUATOR INPUT
PRHSPK	PROCESSES A HIDDEN SURFACE VIEW PICK
CLHSRF	CLOSES HIDDEN SURFACE ELIMINATION
DROPHS	DROPS HIDDEN SURFACE IMAGE FROM THE DISPLAY

J.9 ALTCR NASA

Subroutine	*Functional Description*
ALTCR	MODULE FOR CHANGING THE SHADING LIGHT SOURCE COLOR

MLTCR	LIGHT SOURCE COLOR MENU DRIVER
EXLTCR	EXECUTES LIGHT COLOR MENU
INLTCR	GETS AND PROCESSES LIGHT COLOR MENU INPUT
PLTCR	PROCESSES A LIGHT SOURCE COLOR MENU PICK
PILTCR	PROCESSES A LIGHT SOURCE COLOR MENU ITEM

J.10 ALTSR NASA

Subroutine	*Functional Description*
ALTSR	ALLOWS USER TO SPECIFY LIGHT SOURCE LOCATIONS AND BRIGHTNESSES
MLTSR	MENU DRIVER FOR SETTING SHADING LIGHT SOURCES
EXLTSR	EXECUTES LIGHT SOURCE MENU
INLTSR	GETS AND PROCESSES LIGHT SOURCE MENU INPUT
PLTSR	PROCESSES LIGHT SOURCE MENU INPUT
PILTSR	PROCESSES A LIGHT SOURCE MENU ITEM

J.11 AMAIN NASA

Subroutine	*Functional Description*
AMAIN	MAIN MENU MODULE FOR ACSYNT/VPI
OPMAIN	OPENS ACSYNT
GWINFO	ASK USER TO INPUT WORKSTATION TYPE AND CONNECTION ID
INITWK	CALLES ROUTINES TO INITIALIZE THE WORKSTATION
INITCL	INITIALIZES WORKSTATION COLOR TABLE
INITIN	INITIALIZES INPUT DEVICES
INITVL	INITIALIZES VALUATOR DEVICES
INITST	INITIALIZES THE STRING INPUT DEVICE
INITPK	INITIALIZES PICK DEVICE
DRSTAT	DRAWS STATIC STRUCTURES
BCKGRD	DRAWS SCREEN BACKGROUND
BKGNDS	DRAWS BACKGROUND FILL AREAS IN THE OPEN STRUCTURE

BKGDFA	CONTROLS DRAWING THE FILL AREAS FOR EACH BACKGROUND
FRAME	DRAWS SCREEN FRAME
DRFRM	DRAWS SCREEN FRAME
DRFLNS	DRAWS SCREEN FRAME LINES IN THE OPEN STRUCTURE
PRGTTL	DRAWS PROGRAM TITLE
DRPTTL	DRAWS PROGRAM TITLE IN THE OPEN STRUCTURE
STATVW	SETS UP THE VIEW TO DISPLAY THE STATIC STRUCTURES
STDMNU	DRAWS STANDARD MENU
DRSTD	DRAWS STANDARD MENU
DSITMS	DRAWS THE STANDARD MENU ITEMS IN THE OPEN STRUCTURE
MMAIN	EXECUTES MAIN MENU
INMAIN	GETS AND PROCESSES MAIN MENU INPUT
PMAIN	PROCESSES MAIN MENU INPUT
PIMAIN	PROCESSES A MAIN MENU ITEM PICK
CLMAIN	CLOSES ACSYNT

J.12 AMDFY NASA

Subroutine	*Functional Description*
AMDFY	ENTRY POINT FOR GEOMETRY MODIFICATION MODULE
MMDFY	MODIFY GEOMETRY MENU DRIVER
EXMDFY	EXECUTES GEOMETRY MODIFY MENU
INMDFY	GETS AND PROCESSES MODIFY MENU INPUT
PMDFY	PROCESSES MODIFY MENU INPUT
PIMDFY	PROCESSES A MODIFY MENU ITEM PICKED BY THE USER

J.13 ANLTS NASA

Subroutine	*Functional Description*
ANLTS	MODULE USED TO SET THE NUMBER OF LIGHT SOURCES

GTNLTS	GETS NUMBER OF SHADING LIGHT SOURCES AND RETURNS THE NUMBER AS AN INTEGER
MNLTS	MENU DRIVER FOR SETTING THE NUMBER OF LIGHT SOURCES
EXNLTS	EXECUTES NUMBER OF LIGHT SOURCES MENU
INNLTS	GETS AND PROCESSES NUMBER OF LIGHT SOURCES MENU INPUT
PNLTS	PROCESSES NUMBER OF LIGHTS MENU INPUT
PINLTS	PROCESSES A NUMBER OF LIGHTS MENU ITEM

J.14 ARAMB NASA

Subroutine	*Functional Description*
ARAMB	RESET AMBIENT LIGHT FRACTION INPUT VALUES FOR AMBIENT LIGHT INPUT Routine.

J.15 ARETL NASA

Subroutine	*Functional Description*
ARETL	GEOMETRY RETILING MODULE
MRETL	RETILING MENU DRIVER
EXRETL	EXECUTES RETILING MENU
INRETL	GETS AND PROCESSES RETILING MENU INPUT
PRETL	PROCESSES RETILING MENU INPUT
PIRETL	PROCESSES A RETILING MENU ITEM
PTRETL	PROCESSES A RETILING GEOMETRY PICK

J.16 ARSCC NASA

Subroutine	*Functional Description*
ARSCC	RESETS COMPONENT COLOR INPUT VALUES TO DEFAULT VALUES

J.17 ARSLC NASA

Subroutine	*Functional Description*
ARSLC	RESETS LIGHT SOURCE COLOR INPUT VALUES

J.18 ARSLS NASA

Subroutine	*Functional Description*
ARSLS	RESETS SHADING LIGHT SOURCE INPUT VALUES

J.19 ARSNL NASA

Subroutine	*Functional Description*
ARSNL	RESET INPUT FOR NUMBER OF SHADING LIGHT SOURCES

J.20 ARSRT NASA

Subroutine	*Functional Description*
ARSRT	RESETS INPUT VALUES FOR RETILING TO DEFAULT VALUES

J.21 ARSSC NASA

Subroutine	*Functional Description*
ARSSC	RESETS SHADING SURFACE CHARACTERISTICS INPUT VALUES

J.22 ARTAL NASA

Subroutine	*Functional Description*
ARTAL	RETILES ALL AIRCRAFT COMPONENTS

J.23 ASACC NASA

Subroutine	*Functional Description*
ASACC	SETS ALL AIRCRAFT COMPONENT COLORS TO THE SPECIFIED COLOR

J.24 ASCLR NASA

Subroutine	*Functional Description*
ASCLR	ALLOWS USER TO DEFINE RGB COLOR VALUES FOR EACH COLOR INDEX
MSCLR	SET COLORS MENU DRIVER
EXSCLR	EXECUTES SET COLOR MENU
INSCLR	GETS AND PROCESSES SET COLOR MENU INPUT
PSCLR	PROCESSES SET COLOR MENU INPUT
PISCLR	PROCESSES A SET COLOR MENU ITEM
PCLVL	PROCESSES SET COLOR VALUATOR INPUT

J.25 ASHAD NASA

Subroutine	*Functional Description*
ASHAD	GEOMETRY SHADING MODULE
OPSHAD	PREPARES FOR SHADING
MSHAD	SHADING MENU DRIVER
EXSHAD	EXECUTES SHADING MENU
INSHAD	GETS AND PROCESSES SHADING MENU INPUT
PSHAD	PROCESSES A SHADING MENU PICK
PISHAD	PROCESSES A SHADING MENU ITEM PICKED BY THE USER
STSHPK	STORES A PICKED AIRCRAFT COMPONENT NUMBER AND RE-SETS THE SHADING FLAG
PSHVL	PROCESSES SHADING VALUATOR INPUT
CLSHAD	CLOSES SHADING
DROPSH	DROPS SHADED IMAGE FROM THE DISPLAY

J.26 ASHOW NASA

Subroutine	*Functional Description*
ASHOW	ALLOWS USER TO SHOW AND NO SHOW AIRCRAFT COMPONENTS
MSHOW	SHOW/NO SHOW MENU DRIVER
EXSHOW	EXECUTES SHOW MENU
INSHOW	GETS AND PROCESSES SHOW MENU INPUT
PSHOW	PROCESSES SHOW MENU INPUT
PISHOW	PROCESSES A SHOW MENU ITEM
STSWPK	ADDS OR REMOVES A SPECIFIED COMPONENT TO OR FROM THE COMPONENT LIST AND SET THE INVISIBILITY FILTER BASED ON THE RESULTS.

J.27 ASHRS NASA

Subroutine	*Functional Description*
ASHRS	CONTROLS RESETING OF SHADING DATABASE

J.28 ASRFC NASA

Subroutine	*Functional Description*
ASRFC	ALLOWS USER TO SET THE SURFACE CHARACTERISTICS FOR SHADING
MSRFC	MENU DRIVER FOR SURFACE CHARACTERISTICS
EXSRFC	EXECUTES SURFACE CHARACTERISTICS MENU
INSRFC	GETS AND PROCESSES SURFACE CHARACTERISTICS MENU INPUT
PSRFC	PROCESSES SURFACE CHARACTERISTICS MENU INPUT
PISRFC	PROCESSES A SURFACE CHARACTERISTICS MENU ITEM

J.29 ASTCL NASA

Subroutine	*Functional Description*
ASTCL	SETS THE RGB VALUE OF A SPECIFIED COLOR TO A PREDEFINED COLOR

J.30 ASUNF NASA

Subroutine	*Functional Description*
ASUNF	SWITCHES SHADING SUN FLAG ON OR OFF

J.31 ASWAL NASA

Subroutine	*Functional Description*
ASWAL	REMOVES ALL COMPONENTS FROM NO SHOW LIST

J.32 ASWFP NASA

Subroutine	*Functional Description*
ASWFP	FLIPS THE SHOW AND NO SHOW SCREENS

J.33 AWFRT NASA

Subroutine	*Functional Description*
AWFRT	WINDOW FRONT VIEW MENU MODULE
MWFRT	FRONT VIEW MENU DRIVER
EXWFRT	EXECUTES FRONT VIEW MENU
INWFRT	GETS AND PROCESSES FRONT VIEW MENU INPUT
PWFRT	PROCESSES FRONT VIEW MENU INPUT
PIWFRT	PROCESSES A FRONT VIEW MENU ITEM

J.34 AWIND NASA

Subroutine	*Functional Description*
AWIND	GEOMETRY WINDOW MENU MODULE
MWIND	WINDOW MENU DRIVER
EXWIND	EXECUTES WINDOW MENU
INWIND	GETS AND PROCESSES WINDOW MENU INPUT
PWIND	PROCESSES WIND MENU INPUT
PIWIND	PROCESSES A WINDOW MENU ITEM

J.35 AWISO NASA

Subroutine	*Functional Description*
AWISO	ISOMETRIC WINDOW MENU MODULE
MWISO	ISOMETRIC WINDOW MENU DRIVER
EXWISO	EXECUTES ISOMETRIC MENU
INWISO	GETS AND PROCESSES WINDOW ISOMETRIC MENU INPUT
PWISO	PROCESSES ISOMETRIC WINDOW MENU INPUT
PIWISO	PROCESSES A ISOMETRIC MENU ITEM

J.36 AWMLT NASA

Subroutine	*Functional Description*
AWMLT	DISPLAYS MULTIPLE VIEWS OF AIRCRAFT GEOMETRY

J.37 AWPRJ NASA

Subroutine	*Functional Description*
AWPRJ	WINDOW PROJECTION TYPE MENU MODULE
MWPRJ	PROJECTION TYPE MENU DRIVER
EXWPRJ	EXECUTES PROJECTION TYPE MENU
INWPRJ	GETS AND PROCESSES WINDOW PROJECTION TYPE MENU INPUT
PWPRJ	PROCESSES PROJECTION TYPE MENU INPUT

PIWPRJ	PROCESSES A PROJECTION TYPE MENU ITEM
PRPTPK	PROCESSES A PROJECTION TYPE VIEW PICK

J.38 AWRES NASA

Subroutine	*Functional Description*
AWRES	RESET WINDOW MENU MODULE
MWRES	WINDOW RESTORE MENU DRIVER
EXWRES	EXECUTES WINDOW RESTORE MENU
INWRES	GETS AND PROCESSES RESET WINDOW MENU INPUT
PWRES	PROCESSES WINDOW RESTORE MENU INPUT
PIWRES	PROCESSES A WINDOW RESTORE MENU ITEM
PRWRES	PROCESSES A WINDOW RESET VIEW PICK

J.39 AWROT NASA

Subroutine	*Functional Description*
AWROT	WINDOW ROTATION MENU MODULE
MWROT	MENU DRIVER FOR WINDOW ROTATE
EXWROT	EXECUTES WINDOW ROTATE MENU
INWROT	GETS AND PROCESSES WINDOW ROTATE MENU INPUT
PWROT	PROCESSES WINDOW ROTATE MENU INPUT
PIWROT	PROCESSES AN WINDOW ROTATE MENU ITEM
PWRVL	PROCESSES WINDOW ROTATION VALUATOR INPUT
PWRST	PROCESSES WINDOW ROTATION STRING INPUT
PRWROT	PROCESSES A VIEW ROTATION VIEW PICK

J.40 AWSCL NASA

Subroutine	*Functional Description*
AWSCL	WINDOW SCALE MENU MODULE
MWSCL	WINDOW SCALE MENU DRIVER

EXWSCL	EXECUTES WINDOW SCALE MENU
INWSCL	GETS AND PROCESSES WINDOW SCALE MENU INPUT
PWSCL	PROCESSES WINDOW SCALE MENU INPUT
PIWSCL	PROCESSES A WINDOW SCALE MENU ITEM
PWSVL	PROCESSES WINDOW SCALE VALUATOR INPUT
PWSST	PROCESSES WINDOW SCALE STRING INPUT
PRWSCL	PROCESSES A WINDOW SCALE VIEW PICK

J.41 AWSID NASA

Subroutine	*Functional Description*
AWSID	WINDOW SIDE VIEW MENU MODULE
MWSID	SIDE VIEW MENU DRIVER
EXWSID	EXECUTES SIDE VIEW MENU
INWSID	GETS AND PROCESSES SIDE VIEW MENU INPUT
PWSID	PROCESSES SIDE VIEW MENU INPUT
PIWSID	PROCESSES A SIDE VIEW MENU ITEM

J.42 AWSNG NASA

Subroutine	*Functional Description*
AWSNG	SINGLE WINDOW MENU MODULE
MWSNG	SINGLE WINDOW MENU DRIVER
EXWSNG	EXECUTES SINGLE WINDOW MENU
INWSNG	GETS AND PROCESSES SINGLE WINDOW MENU INPUT
PWSNG	PROCESSES SINGLE WINDOW MENU INPUT
PIWSNG	PROCESSES A SINGLE WINDOW MENU ITEM
PRWSNG	PROCESSES SINGLE WINDOW CHOICE INPUT
DAGMVW	DEACTIVATES ALL GEOMETRY VIEWS

J.43 AWSTR NASA

Subroutine	*Functional Description*
AWSTR	STORE WINDOW MENU MODULE
MWSTR	WINDOW STORE MENU DRIVER
EXWSTR	EXECUTES WINDOW STORE MENU
INWSTR	GETS AND PROCESSES STORE WINDOW MENU INPUT
PWSTR	PROCESSES STORE WINDOW MENU INPUT
PIWSTR	PROCESSES A WINDOW STORE MENU ITEM
PRWSTR	PROCESSES WINDOW STORE VIEW PICK

J.44 AWTOP NASA

Subroutine	*Functional Description*
AWTOP	TOP VIEW MENU MODULE
MWTOP	TOP VIEW MENU DRIVER
EXWTOP	EXECUTES TOP VIEW MENU
INWTOP	GETS AND PROCESSES TOP VIEW MENU INPUT
PWTOP	PROCESSES TOP VIEW MENU INPUT
PIWTOP	PROCESSES A TOP VIEW MENU ITEM

J.45 AWTRN NASA

Subroutine	*Functional Description*
AWTRN	WINDOW TRANSLATION MENU MODULE
MWTRN	WTRNOW MENU DRIVER
EXWTRN	EXECUTES WINDOW TRANSLATE MENU
INWTRN	GETS AND PROCESSES WINDOW TRANSLATION MENU INPUT
PWTRN	PROCESSES WINDOW TRANSLATION MENU INPUT
PIWTRN	PROCESSES A WINDOW TRANSLATE MENU ITEM
PWTCH	PROCESSES WINDOW TRANSLATION CHOICE INPUT

J.46 CENTAN NASA

Subroutine	*Functional Description*
CENTAN	THE MODULE CENTAN TAKES A NUMBER OF 3-D POINTS AND CALCULATES THE TANGENTS AT THE POINTS BY USING CENTRAL DIFFERENCE. THE FIRST TANGENT IS CALCULATED USING FORWARD DIFFERENCE AND THE LAST TANGENT IS CALCULATED USING BACKWARD DIFFERENCE.

J.47 CLGEOM NASA

Subroutine	*Functional Description*
CLGEOM	REMOVES AIRCRAFT GEOMETRY DISPLAY FROM SCREEN
RMGEOM	REMOVES AIRCRAFT GEOMETRY FROM DISPLAY
RMGMRT	EMPTIES ALL GEOMETRY ROOT STRUCTURES
EMGMST	EMPTIES ALL GEOMETRY STRUCTURES
RMVFRM	REMOVES GEOMETRY VIEW FRAMES FROM THE DISPLAY

J.48 CMPCLR NASA

Subroutine	*Functional Description*
CMPCLR	CHANGES THE COLOR OF A SPECIFIED COMPONENT

J.49 GEOMDB NASA

Subroutine	*Functional Description*
PTNCMP	STORES CURRENT NUMBER OF COMPONENTS IN DATABASE
GTNCMP	RETURNS NUMBER OF COMPONENTS DEFINED IN COMPONENT DATABASE
PTRCMP	STORES REAL COMPONENT DATA IN COMPONENT DATABASE
GTRCMP	RETURNS REAL DATA FROM COMPONENT DATABASE
PTICMP	STORES INTEGER COMPONENT DATA IN COMPONENT DATABASE
GTICMP	RETURNS INTEGER COMPONENT DATA

PTCCMP	STORES CHARACTER DATA IN COMPONENT DATABASE
GTCCMP	RETURNS CHARACTER DATA FROM COMPONENT DATABASE
PTRXS	STORES REAL X-SECTION DATA
GTRXS	RETURNS REAL X-SECTION DATA
PTIXS	STORES INTEGER X-SECTION DATA
GTIXS	RETURNS INTEGER X-SECTION DATA
PTRSRF	STORES SURFACE NODE DATA
GTRSRF	RETURNS SURFACE NODE DATA
RDCPDT	READ AIRCRAFT GEOMETRY COMPONENT DATA FROM THE SPECIFIED LOGICAL UNIT AND STORES THE DATA IN THE GEOMETRY DATABASE
RDNCMP	READS NUMBER OF AIRCRAFT COMPONENTS FROM THE GEOMETRY FILE
RDCPNM	READS COMPONENT IDENTIFICATION NUMBER FROM THE GEOMETRY FILE
RDCCMP	READS AIRCRAFT COMPONENT CHARACTER DATA FROM INPUT FILE AND STORES THE DATA IN THE COMPONENT DATABASE
RDICMP	READS AND STORES INTEGER AIRCRAFT COMPONENT DATA
RDRCMP	READS AND STORES REAL AIRCRAFT COMPONENT DATA

J.50 GMTRY NASA

Subroutine	*Functional Description*
PGMVL	PROCESS GEOMETRY VALUATOR
DRGEOM	CONVERTS AIRCRAFT COMPONENT X-SECTION DATA TO SURFACE DATA IF THE COMPONENT EXISTS
PRVCPK	PROCESSES A VIEW PICK FOR CHANGING A VIEW'S TRANSFORMATION VALUES
STGMVW	INITIALIZES GEOMETRY VIEWS
STVWCS	SETS GEOMETRY VIEW CHARACTERISTICS
STVWM3	SETS THE VIEW MATRIX FOR A GEOMETRY VIEW
STVMP3	SETS GEOMETRY VIEW MAPPING
CRVWFM	CREATES A GEOMETRY VIEW FRAME STRUCTURE
CRVFSS	CREATES A VIEW FRAME SUB-STRUCTURE

DRVWFM	DRAWS A VIEW FRAME AROUND THE SPECIFIED VIEW PORT
DRVWPK	DRAWS A VIEW PICK ELEMENT
STVWRT	SETS GEOMETRY VIEW ROOT STRUCTURE
PRVWPK	PROCESSES A GEOMETRY VIEW PICK
STVWFM	POSTS OR UNPOSTS A GEOMETRY VIEW FRAME DEPENDING UPON WHETHER THE VIEW IS ACTIVE OR INACTIVE RESPECTIVELY

J.51 GMVWDB NASA

Subroutine	*Functional Description*
PTNVWS	PUTS NUMBER OF DEFINED VIEWS INTO GEOMETRY VIEW DATABASE
GTNVWS	GETS NUMBER OF DEFINED VIEWS FROM GEOMETRY VIEW DATABASE
PTRVWS	PUTS REAL VIEW DATA INTO GEOMETRY VIEW DATABASE
GTRVWS	GETS REAL VIEW DATA FROM GEOMETRY VIEW DATABASE
PTIVWS	PUTS INTEGER VIEW DATA INTO GEOMETRY VIEW DATABASE
GTIVWS	GETS INTEGER VIEW DATA FROM GEOMETRY VIEW DATABASE
PTCVWS	PUTS CHARACTER VIEW DATA INTO GEOMETRY VIEW DATABASE
GTCVWS	GETS CHARACTER VIEW DATA FROM GEOMETRY VIEW DATABASE
RDVWDT	READS GEOMETRY VIEW DATA FROM THE SPECIFIED INPUT UNIT AND PUTS THE DATA INTO THE GEOMETRY VIEW DATABASE
RDNVWS	READS NUMBER OF DEFINED GEOMETRY VIEWS FROM INPUT FILE
RDVWNM	READS PHIGS VIEW NUMBER FROM INPUT FILE
RDCVWS	READS CHARACTER VIEW DATA FROM INPUT FILE AND STORES IN DATABASE
RDIVWS	READS INTEGER VIEW DATA FROM INPUT FILE AND STORES IN DATABASE
RDRVWS	READS REAL VIEW DATA FROM INPUT FILE AND STORES IN DATABASE

J.52 GTGMPK NASA

Subroutine	*Functional Description*
GTGMPK	RETURNS THE LIST OF AIRCRAFT COMPONENTS WHICH HAVE BEEN PICKED BY THE USER. IF NO COMPONENTS HAVE BEEN PICKED THEN ALL OF THE COMPONENT NUMBERS ARE RETURNED.
LDCPLS	LOAD A LIST WITH ALL OF THE AIRCRAFT COMPONENT NUMBERS

J.53 LIST NASA

Subroutine	*Functional Description*
CHKLST	SEARCHES FOR AN ITEM IN A LIST AND ADDS THE ITEM IF THE ITEM IS NOT FOUND IN THE LIST OTHERWISE THE ITEM IS DELETED FROM THE LIST.
SCHLST	RETURNS THE LOCATION OF AN ITEM IN THE SPECIFIED LIST
ADDLST	ADDS AN ITEM TO THE END OF A LIST
RMVLST	REMOVES A SPECIFIED ITEM FROM A LIST
UNQLST	SEARCHS FOR AN ITEM IN A LIST AND ADDS THE ITEM IF THE ITEM IS NOT FOUND

J.54 MENU NASA

Subroutine	*Functional Description*
DRITM	DRAWS A MENU ITEM
OPMNU	OPENS A NEW MENU TO BE DISPLAYED
CRMNU	CREATES A MENU SUB-STRUCTURE
DRMNU	DRAWS THE MENU IN THE OPEN MENU STRUCTURE
DRMTTL	DRAWS THE MENU TITLE
DRMITM	CONTROLS DRAWING OF MENU ITEMS IN THE OPEN STRUCTURE
DRITMS	DRAWS MENU ITEMS IN THE OPEN STRUCTURE
CLMNU	CLOSES THE CURRENT MENU
GTMNIN	GETS MENU INPUT

GTMNTI	GETS MENU TYPE AND ITEM FROM PICK PATH
PRSTMN	PROCESSES A STANDARD MENU ITEM PICK
NEWMNU	DISPLAYS A NEW MENU
PSHMNU	PUSHES THE CURRENT MENU ON THE MENU STACK
OLDMNU	DISPLAYS PREVIOUS MENU
POPMNU	POPS PREVIOUS MENU OFF OF MENU STACK
MNUATB	SETS MENU ATTRIBUTES
CRMNRT	CREATES A MENU ROOT STRUCTURE WHICH EXECUTES A SUB-STRUCTURE CONTAINING THE ACTUAL MENU ITEMS.

J.55 MKTILE NASA

Subroutine	*Functional Description*
MKTILE	THE MODULE MKTILE FILLS THE COMMON BLOCK 'POLDAT' WITH POLYGONAL DATA FOR A COMPONENT. THE ROUTINE DISPTL CAN BE EXECUTED AFTER EXECUTING THIS ROUTINE TO DISPLAY THE TILED IMAGE.
GNTILE	GENERATES SURFACE POLYGONS FOR A SPECIFIED COMPONENT
GNSFTL	GENERATES SURFACE PARAMETERS FOR (I,J)TH SURFACE OF THE SPECIFIED COMPONENT
GNPOLY	GENERATES THE POLYGONS ON A SURFACE GIVEN THE SURFACE MATRIX, AND THE NUMBER OF POLYGONS IN THE U AND W DIRECTIONS.
CRPOLY	GENERATES POLYGONS ON A SURFACE FOR A FIXED VALUE OF THE PARAMETER 'U'.
DRPOLY	GENERATES THE DATA FOR A POLYGON ON A SURFACE FOR A GIVEN LOCATION ON THE SURFACE DETERMINED BY THE VALUES OF THE PARAMETERS U AND W. THE FIRST VERTEX OF THE POLYGON IS AT THE LOCATION (U,W) ON THE SURFACE. THE SECOND VERTEX IS AT THE LOCATION (U,W+DW) ON THE SURFACE. THE THIRD VERTEX IS AT THE LOCATION (U+DU,W+DW) ON THE SURFACE. THE FOURTH VERTEX IS AT THE LOCATION (U+DU,W) ON THE SURFACE.
POLPT1	CALCULATES THE COORDINATES THE FIRST VERTEX OF A POLYGON ON A SURFACE
POLPT2	CALCULATES THE COORDINATES THE SECOND VERTEX OF A POLYGON ON A SURFACE
POLPT4	CALCULATES THE COORDINATES THE FOURTH VERTEX OF A POLYGON ON A SURFACE

J.56 OPGEOM NASA

Subroutine	*Functional Description*
OPGEOM	CREATES DISPLAY STRUCTURES FOR THE AIRCRAFTS GEOMETRY AND SETS UP THE DISPLAY SCREEN TO DISPLAY THE GEOMETRY.
DSGEOM	CREATES PHIGS DISPLAY STRUCTURES FOR AIRCRAFT GEOMETRY
CRRTS	CREATES GEOMETRY ROOT STRUCTURES FOR EACH GEOMETRY VIEW
CRROOT	CREATES GEOMETRY ROOT STRUCTURE
GNROOT	GENERATES GEOMETRY ROOT STRUCTURE ELEMENTS
CRATTR	CREATES ATTRIBUTE STRUCTURES FOR ALL AIRCRAFT COMPONENTS
CKATTR	CREATES A COMPONENTS ATTRIBUTE STRUCTURE IF THE COMPONENT EXISTS
MKATTR	CREATE A COMPONENT'S ATTRIBUTE STRUCTURE
FLATTR	INSERTS ATTRIBUTE ELEMENTS INTO OPEN ATTRIBUTE STRUCTURE FOR A A GIVEN COMPONENT
INATTR	INSERTS PRIMITIVE ATTRIBUTES INTO COMPONENT ATTRIBUTE STRUCTURE
INCLPK	INSERTS CLASS NAME AND PICK IDENTIFIER INTO COMPONENT ATTRIBUTE STRUCTURE
INEXST	INSERTS TRANSFORMATION MATRICES AND STRUCTURE CALLS INTO THE OPEN COMPONENT ATTRIBUTE STRUCTURE
GINSRT	INSERTS TRANSFORMATION MATRIX AND LOCAL SYMMETRY STRUCTURE CALLS INTO THE OPEN COMPONENT ATTRIBUTE STRUCTURE
GTCPXF	CALCULATES A 3D TRANSFORMATION MATRIX FOR A SPECIFIED SET OF X, Y, Z EULER ANGLES AND X, Y, Z TRANSLATIONS
GRFLCT	INSERTS REFLECTION MATRIX AND LOCAL SYMMETRY STRUCTURE CALL IF COMPONENT HAS GLOBAL SYMMETRY.
INRFLT	INSERTS A SYMMETRY MATRIX AND STRUCTURE CALL INTO THE OPEN STRUCTURE
GTSMMT	CALCULATES A SYMMETRY MATRIX GIVEN A FLAG INDICATING WHICH PRINCIPAL PLANE IS THE SYMMETRY PLANE
CRLSYM	CREATES LOCAL SYMMETRY STRUCTURES FOR ALL COMPONENTS

CKLSYM	CREATES LOCAL SYMMETRY STRUCTURE IF SPECIFIED COMPONENT EXISTS
MKLSYM	CREATES A LOCAL SYMMETRY STRUCTURE FOR A SPECIFIED COMPONENT
FLLSYM	FILLS A LOCAL SYMMETRY STRUCTURE
LINSRT	EXECUTES THE COMPONENT STRUCTURE
LRFLCT	INSERTS A LOCAL REFLECTION MATRIX AND EXECUTES THE COMPONENT STRUCTURE FOR THE SPECIFIED COMPONENT
CRCOMP	CREATES AIRCRAFT COMPONENT STRUCTURES
CKCOMP	CREATES A COMPONENT STRUCTURE IF THE COMPONENT EXISTS
MKCOMP	CREATES A COMPONENT STRUCTURE FOR A SPECIFIED COMPONENT
FLCOMP	FILLS A COMPONENT STRUCTURE
GNSURF	GENERATES SURFACES FOR A SPECIFIED COMPONENT
GNSFEL	GENERATES SURFACE ELEMENTS FOR (I,J)TH SURFACE OF THE SPECIFIED COMPONENT
GTBMAT	GETS THE (I,J)TH SURFACE MATRIX FROM THE GEOMETRY DATABASE
CPYSRF	COPIES SURFACE DATA INTO A SURFACE MATRIX FROM THE SURFACE DATABASE
GNMESH	GENERATES A POLYLINE MESH ON THE SPECIFIED SURFACE
DRU	CONTROLS DRAWING OF CONSTANT U CURVES ON (I,J)TH SURFACE
CONSTU	DRAWS CONSTANT U CURVES ON SPECIFIED SURFACE
DRUCRV	DRAWS U CURVE ON SPECIFIED SURFACE
DRCONU	DRAWS CONSTANT U CURVE ON SURFACE PATCH
PSRF	USES THE GEOMETRIC FORM TO COMPUTE THE X,Y,Z COORDINATES OF A POINT ON A BICUBIC PATCH FOR SPECIFIC VALUES OF THE PARAMETRIC VARIABLES U AND W.
BF	CALCULATES THE FOUR COMPONENTS OF THE BLENDING FUNCTIONS AT A SPECIFIED VALUE OF THE PARAMETRIC VARIABLE U.
CALCPT	CALCULATES A SPECIFIED COORDINATE ON A SURFACE
SRFMLT	PERFORMS MATRIX MULTIPLICATION FOR CALCULATING A POINT COORDINATE ON A SURFACE PATCH
DRW	CONTROLS DRAWING OF CONSTANT W CURVES ON THE (I,J)TH SURFACE

CONSTW	DRAWS CONSTANT W CURVES ON SPECIFIED SURFACE
DRWCRV	DRAWS CONSTANT W CURVE ON SPECIFIED SURFACE
DRCONW	DRAWS CONSTANT W CURVE ON THE SPECIFIED SURFACE
INGMVW	INITIALIZES GEOMETRY VIEWS

J.57 RETILE NASA

Subroutine	*Functional Description*
RETILE	RETILES A SPECIFIED COMPONENT

J.58 SHADDB NASA

Subroutine	*Functional Description*
SHDRST	RESETS SHADING DATABASE
STSFLG	SETS SHADING FLAGS
STSLGT	SETS SHADING LIGHT SOURCE
STSVLS	SETS SHADING VALUES
PTSSUN	PUTS SHADING SUN FLAG INTO SHADING DATABASE
GTSSUN	GETS SHADING SUN FLAG FROM SHADING DATABASE
PTSEYE	PUTS SHADING EYE FLAG INTO SHADING DATABASE
GTSEYE	GETS SHADING EYE FLAG FROM THE SHADING DATABASE
PTSCOL	PUTS SHADING RGB COLOR VALUES INTO SHADING DATABASE
STSCOL	STORES ITH RGB COLOR VALUE IN SHADING DATABASE
GTSCOL	GETS SHADING RGB COLOR VALUES FROM SHADING DATABASE
PTSNLT	PUTS NUMBER OF LIGHT SOURCES INTO SHADING DATABASE
GTSNLT	GETS NUMBER OF LIGHT SOURCES FROM SHADING DATABASE
PTSLCR	PUTS LIGHT SOURCE RGB VALUE INTO SHADING DATABASE
STSLCR	STORES LIGHT COLOR IN SHADING DATABASE
GTSLCR	GETS LIGHT SOURCE RGB VALUE FROM SHADING DATABASE
PTSLGT	PUTS LIGHT SOURCE LOCATION AND BRIGHTNESS INTO SHADING DATABASE

GTSLGT	GETS LIGHT SOURCE LOCATION AND BRIGHTNESS FROM SHADING DATABASE
PTSSRF	PUTS SURFACE CHARACTERISTICS INTO SHADING DATABASE
GTSSRF	GETS SURFACE CHARACTERISTICS FROM SHADING DATABASE
PTSAMB	PUTS AMBIENT LIGHT FRACTION INTO SHADING DATABASE
GTSAMB	GETS AMBIENT LIGHT FRACTION FROM SHADING DATABASE

J.59 SHDHDE NASA

Subroutine	*Functional Description*
DOSHAD	DRIVER ROUTINE FOR SHADING THE AIRCRAFT GEOMETRY
DOHSRF	DRIVER ROUTINE FOR PERFORMING HIDDEN SURFACE ELIMINATION ON THE AIRCRAFT GEOMETRY
PRSHPK	PROCESSES SHADING VIEW PICK

J.60 STGMPK NASA

Subroutine	*Functional Description*
STGMPK	ADDS OR REMOVES A SPECIFIED COMPONENT TO OR FROM THE COMPONENT LIST AND SET THE HIGHLIGHTING FILTER BASED ON THE RESULTS.

J.61 TILEIT NASA

Subroutine	*Functional Description*
TILEIT	THE MODULE TILEIT GENERATES A POLYGONAL MODEL. THE DATA FOR THE POLYGONS FOR EACH COMPONENT IS GENERATED AND LOADED INTO A COMMON BLOCK. THE POLYGON VERTICES ARE THEN TRANSFORMED TO THE CORRECT LOCATION TO LOCATE AND ORIENT THE PART. LOCAL AND GLOBAL SYMMETRIES ARE TAKEN CARE OF BY COPYING THE POLYGONS AND MOVING THEM TO THE CORRECT GLOBAL LOCATION.
TRANTL	THE MODULE TRANTILE INQUIRES THE LOCATION AND ORIENTATION OF A COMPONENT, GENERATES THE NECESSARY TRANSFORMATION MATRIX AND APPLIES IT TO ALL THE VERTICES OF THE POLYGONS IN THE COMMON BLOCK 'POLDAT'.

LSYMTL	THE MODULE LSYMTL INQUIRES THE LOCAL SYMMETRY OF A COMPONENT. IF THE COMPONENT HAS LOCAL SYMMETRY, LSYMTL DUPLICATES THE POLYGON DATA IN THE COMMON BLOCK 'POLDAT' AFTER APPLYING THE NECESSARY TRANSFORMATIONS TO ACCOUNT FOR THE SYMMETRY.
GSYMTL	THE MODULE GSYMTL INQUIRES THE GLOBAL SYMMETRY OF A COMPONENT. IF THE COMPONENT HAS GLOBAL SYMMETRY, GSYMTL DUPLICATES THE POLYGON DATA IN THE COMMON BLOCK 'POLDAT' AFTER APPLYING THE NECESSARY TRANSFORMATIONS TO ACCOUNT FOR THE SYMMETRY.
SYMTXF	THE MODULE SYMTXF TAKES A COMPONENT TRANSFORMATION MATRIX AND A LOCAL SYMMETRY MATRIX AND COMPOSES THE FINAL TRANSFORMATION MATRIX TO GENERATE A SYMMETRICAL PART AND MOVE IT TO THE CORRECT LOCATION. THIS ROUTINE ASSUMES THAT THE DATA WHICH WILL BE TRANSFORMED HAS ALREADY BEEN MOVED TO THE CORRECT LOCATION USING THE TRANSFORMATION MATRIX ' XFMAT '.

J.62 UTILTY NASA

Subroutine	*Functional Description*
GTCNIN	GETS INTEGER INPUT FROM THE CONSOLE
MSG	WRITES THE SPECIFIED MESSAGE ON THE SCREEN AFTER SCROLLING UP ALL PREVIOUS MESSAGE LINES.
SCRMSG	SCROLLS THE CURRENT TEXT LINES STORED IN THE ARRAY TEXT UP ONE LINE AND INSERTS THE NEW INPUT STRING INTO POSITION 1.
SFTMSG	SHIFTS THE ITH LINE OF TEXT TO THE I+1 LINE WHERE 1<I<NLINES-1
PLGMSG	PLUGS NEW MESSAGE STRING INTO MESSAGE ARRAY
WRTMSG	WRITES MESSAGE LINES ON SCREEN
DRMSG	INSERTS TEXT ATTRIBUTES AND DRAWS MESSAGE LINES IN THE OPEN STRUCTURE
DRMSTX	DRAW THE MESSAGE TEXT LINES
CAPS	CONVERTS A STRING OF CHARACTERS TO ALL UPPER CASE
CK4FIL	USES FORTRAN INQUIRE FUNCTION TO DETERMINE IF A FILE EXISTS
AQDSP	USED TO INQUIRE ACTUAL DISPLAY SIZE FOR A SPECIFIED WORKSTATION
ASVWM3	COMPUTES AND SETS A VIEW MATRIX GIVEN THE VIEW TRANSFORMATION VALUES

GTXFMT	CALCULATES A 3D TRANSFORMATION MATRIX
GTNAVL	GETS THE NUMBER OF AVAILABLE VALUATOR DEVICES
AINVL	INITIALIZES A SPECIFIED VALUATOR DEVICE
AINST	INITIALIZES STRING INPUT DEVICE
CONERR	WRITES A CONSOLE ERROR MESSAGE
CPYVCT	COPIES DATA FROM ONE ARRAY TO ANOTHER
INFMSG	WRITES A SCREEN MESSAGE AND UPDATES THE WORK-STATION
AINLC	INITIALIZES LOCATOR DEVICE
CKVLVL	CHECKS THE VALUE OF A SPECIFIED VALUATOR
AGTVL	RETURNS LAST VALUATOR VALUE FROM A SERIES OF VALUATOR EVENTS
OPNUM	OPENS NUMBER INPUT USING VALUATORS AND THE STRING DEVICE
CLNUM	CLOSES NUMBER INPUT
PNMVL	PROCESSES NUMBER VALUATOR INPUT
GTNMVL	GETS NUMBER FROM VALUATOR INPUT
PNMST	PROCESSES NUMBER STRING INPUT
PARSE	PARSES INPUT VALUES OUT OF INPUT STRING
PRSERR	PARSES INPUT STRING FOR POSSIBLE INPUT ERRORS
CKCHAR	CHECK THE SPECIFIED CHARACTER AGAINST A LIST OF VALID NUMERIC CHARACTERS
CKDBL	DETERMINES IF THE SPECIFIED CHARACTER IS AN INVALID USE OF EITHER '.' OR '-' IN THE NUMERIC INPUT STRING
LMNMVL	FORCES INPUT VALUES TO BE BETWEEN MAXIMUM AND MINIMUM LIMITS
INNMIN	INITIALIZES NUMBER INPUT
CRNMST	CREATES A STRING OF NUMBERS SEPARATED BY COMMAS
FLNMST	WRITES A SPECIFIED NUMBER INTO A STRING
AGTST	GETS STRING INPUT FROM STRING DEVICE
GTINLN	GETS AN INPUT LINE FROM THE SPECIFIED INPUT LOGICAL UNIT NUMBER IGNORING ALL INPUT LINES STARTING WITH '*'

J.63 VECMAT NASA

Subroutine	*Functional Description*
VNORM	THE MODULE VNORM NORMALIZES A VECTOR AND RETURNS A VECTOR OF UNIT LENGTH. THIS ROUTINE CHANGES THE INPUT VECTOR.

J.64 XFRMTL NASA

Subroutine	*Functional Description*
XFRMTL	THE MODULE XFRMTL APPLIES A TRANSFORMATION MATRIX TO POLYGONS IN THE COMMON BLOCK 'POLDAT'. AFTER APPLYING THE TRANSFORMATION TO THE VERTICES OF THE POLYGONS, THE NEW SET OF POLYGONS ARE EITHER ADDED TO THE POLYGONS ALREADY EXISTING IN THE COMMON BLOCK OR ARE USED TO REPLACE THE EXISTING POLYGONS.
XFPTTL	THE MODULE XFPTTL APPLIES A TRANSFORMATION MATRIX TO THE VERTICES OF A POLYGON IN THE COMMON BLOCK 'POLDAT'. THIS MODULE IS CALLED BY THE MODULE XFRMTL.

J.65 XSSECT NASA

Subroutine	*Functional Description*
XSELLP	THE MODULE XSELLP CALCULATES THE X,Y,Z COORDINATES OF A POINT ON AN ELLIPSE BY USING THE PARAMETRIC EQUATION OF AN ELLIPSE.
XSELLT	THE MODULE XSELLT CALCULATES THE X,Y,Z COMPONENTS OF THE TANGENT AT A POINT ON AN ELLIPSE (IN THE PLANE OF THE ELLIPSE) BY USING THE PARAMETRIC EQUATION OF AN ELLIPSE.
XSAF1P	THE MODULE XSAF1P CALCULATES THE XYZ COORDINATES OF A POINT ON AN AIRFOIL TYPE 1 CROSS SECTION BY USING PARAMETRIC EQUATIONS.
XSAF1T	THE MODULE XSAF1T CALCULATES THE X,Y,Z COMPONENTS OF THE TANGENT AT A POINT ON AN AIRFOIL TYPE 1 CROSS SECTION (ON THE PLANE OF THE CROSS SECTION) BY USING PARAMETRIC EQUATIONS.

APPENDIX K. LIST OF ACSYNT/VPI UTILITY ROUTINES

The following is a list of utility subroutines developed for the ACSYNT/VPI project. Each routine name is preceded by a module identification number and followed by a functional description.

Geometry Database Routines

D1 - PTNCMP - PUTS NUMBER OF COMPONENTS INTO GEOMETRY DATABASE
D2 - GTNCMP - GETS NUMBER OF COMPONENTS FROM GEOMETRY DATABASE
D3 - PTRCMP - PUTS REAL COMPONENT DATA INTO GEOMETRY DATABASE
D4 - GTRCMP - GETS REAL COMPONENT DATA FROM GEOMETRY DATABASE
D5 - PTICMP - PUTS INTEGER COMPONENT DATA INTO GEOMETRY DATABASE
D6 - GTICMP - GETS INTEGER COMPONENT DATA FROM GEOMETRY DATABASE
D7 - PTCCMP - PUTS CHARACTER COMPONENT DATA INTO GEOMETRY DATABASE
D8 - GTCCMP - GETS CHARACTER COMPONENT DATA FROM GEOMETRY DATABASE
D9 - PTRXS - PUTS REAL X-SECTION DATA INTO GEOMETRY DATABASE
D10 - GTRXS - GETS REAL X-SECTION DATA FROM GEOMETRY DATABASE
D11 - PTIXS - PUTS INTEGER X-SECTION DATA INTO GEOMETRY DATABASE
D12 - GTIXS - GETS INTEGER X-SECTION DATA FROM GEOMETRY DATABASE
D13 - PTRSRF - PUTS REAL SURFACE DATA INTO GEOMETRY DATABASE
D14 - GTRSRF - GETS REAL SURFACE DATA FROM GEOMETRY DATABASE
D15 - RDCPDT - READS COMPONENT DATA FROM INPUT FILE AND LOADS DATABASE

Geometry Routines

G1 - OPGEOM - OPENS GEOMETRY DISPLAY
G2 - CLGEOM - CLOSES GEOMETRY DISPLAY
G3 - PGMVL - PROCESSES GEOMETRY VALUATOR INPUT
G4 - DRGEOM - CONVERTS GEOMETRY X-SECTION DATA TO GEOMETRY SURFACE DATA
G5 - DSGEOM - CONVERTS GEOMETRY SURFACE DATA TO PHIGS DATA FOR DISPLAY
G6 - MKATTR - MAKES A COMPONENT ATTRIBUTE PHIGS STRUCTURE
G7 - GTCPXF - CALCULATES A 3D COMPONENT TRANSFORMATION MATRIX
G8 - INRFLT - INSERTS A REFLEC. MATRIX & STRUCTURE CALL INTO A STRUCTURE
G9 - GTSMMT - CALCULATES A 3D SYMMETRY MATRIX
G10 - MKLSTM - MAKES A COMPONENT LOCAL SYMMETRY PHIGS STRUCTURE
G11 - MKCOMP - MAKES A COMPONENT PHIGS STRUCTURE
G12 - GTBMAT - LOADS A SURFACE MATRIX FROM THE GEOMETRY SURFACE DATABASE
G13 - PSRF - CALCULATES THE COORD. OF A POINT ON A BICUBIC SURFACE
G14 - CRROOT - CREATES A PHIGS GEOMETRY ROOT STRUCTURE
G15 - CRATTR - CREATES A COMPONENT ATTRIBUTE PHIGS STRUCTURE
G16 - CRLSYM - CREATES A COMPONENT LOCAL SYMMETRY PHIGS STRUCTURE
G17 - BF - CALCULATES BLENDING FUNCTIONS FOR SURFACE CALCULATIONS
G18 - CALCPT - CALCULATES ONE COORDINATE OF A POINT ON A BICUBIC SURFACE
G19 - GTSTR - CALCULATES THE PHIGS STRUCTURE IDS FOR A COMPONENT
G20 - CRCOMP - CREATES A COMPONENT PHIGS STRUCTURE
G21 - CKCOMP - CREATES A COMPONENT PHIGS STRUCTURE IF THE COMP. EXISTS
G22 - CKATTR - CREATES A COMPONENT ATTRIB. STRUCTURE IF THE COMP. EXISTS
G23 - XSELLP - CALCULATES A POINT ON AN ELLIPTICAL X-SECTION
G24 - XSELLT - CALCULATES A TANGENT TO AN ELLIPTICAL X-SECTION
G25 - XSAF1P - CALCULATES A POINT ON AIRFOIL X-SECTIONS TYPE 1
G26 - XSAF1T - CALCULATES A TANGENT TO AIRFOIL X-SECTION TYPE 1
G27 - STGMPK - STORES THE COMP. NUMBER OF A COMP. & SETS HIGHLIGHT FILTER
G28 - CENTAN - CALCULATES TANGENTS FOR A NO. OF POINTS USING CENTRAL DIFF
G29 - GTGMPK - RETURNS LIST OF PICKED COMPONENTS

G30 - TILEIT - DRAWS POLYGONS AFTER TRANSFORMING COORDINATES
G31 - MKTILE - LOADS POLYGON COMMON BLOCK WITH INITIAL SET OF POLYGONS
G32 - DISPTL - DRAWS POLYGONS FROM POLYGON COMMON BLOCK
G33 - XFRMTL - TRANSFORMS POLYGONS IN POLYGON COMMON BLOCK
G34 - RETILE - RETILES A SPECIFIED COMPONENT
G35 - CMPCLR - CHANGES THE COLOR OF A SPECIFIED COMPONENT
G36 - PRVCPK - PROCESSES A GEOMETRY VIEW PICK
G37 - STGMVW - SETS ALL GEOMETRY VIEWS
G38 - STVWCS - SETS THE VIEW CHARACTERISTICS FOR A SPEC. GEOMETRY VIEW
G39 - STVWM3 - SETS THE VIEW MATRIX FOR A SPECIFIED GEOMETRY VIEW
G40 - STVMP3 - SETS THE VIEW MAPPING FOR A SPECIFIED GEOMETRY VIEW
G41 - STVWRT - SETS THE VIEW ROOT STRUCTURE FOR A SPEC. GEOMETRY VIEW
G42 - PRVWPK - PROCESS GEOMETRY VIEW PICK
G43 - STVWFM - SETS THE VIEW FRAME STRUCTURE FOR A SPEC. GEOMETRY VIEW

Geometry View Database Routines

GV1 - PTNVWS - PUTS NUMBER OF VIEWS INTO GEOMETRY VIEW DATABASE
GV2 - GTNVWS - GETS NUMBER OF VIEWS FROM GEOMETRY VIEW DATABASE
GV3 - PTRVWS - PUTS REAL VIEW DATA INTO GEOMETRY VIEW DATABASE
GV4 - GTRVWS - GETS REAL VIEW DATA FROM GEOMETRY VIEW DATABASE
GV5 - PTIVWS - PUTS INTEGER VIEW DATA INTO GEOMETRY VIEW DATABASE
GV6 - GTIVWS - GETS INTEGER VIEW DATA FROM GEOMETRY VIEW DATABASE
GV7 - PTCVWS - PUTS CHARACTER VIEW DATA INTO GEOMETRY VIEW DATABASE
GV8 - GTCVWS - GETS CHARACTER VIEW DATA FROM GEOMETRY VIEW DATABASE
GV9 - RDVWDT - READS GEOMETRY VIEW DATABASE FROM INPUT FILE

Shading and Hidden Surface Routines

SH1 - SHDCMP - SHADES SPECIFIED AIRCRAFT COMPONENTS
SH2 - HSFCMP - PERFORMS HIDDEN SURFACE ELIMINATION ON AIRCRAFT COMPONENTS
SH3 - DOSHAD - SHADING DRIVER
SH4 - DOHSRF - HIDDEN SURFACE DRIVER
SH5 - PRSHPK - PROCESS A SHADING VIEW PICK

Menu Routines

M1 - ?????? - CURRENTLY UNDEFINED
M2 - DRITM - DRAWS A STANDARD MENU ITEM
M3 - OPMNU - OPENS A MENU
M4 - CLMNU - CLOSES A MENU
M5 - GTMNIN - GETS MENU INPUT
M6 - GTMNTI - GETS MENU TYPE AND MENU ITEM OF A PICKED MENU
M7 - PRSTMN - PROCESSES STANDARD MENU INPUT
M8 - NEWMNU - DISPLAYS A NEW MENU
M9 - OLDMNU - DISPLAYS A PREVIOUS MENU
M10 - MNUATB - SETS MENU TEXT ATTRIBUTES
M11 - CRMNRT - CREATES A MENU ROOT STRUCTURE

Integer List Manipulating Routines

LM1 - CHKLST - ADDS ITEM TO A LIST IF NOT IN LIST OTHERWISE REMOVES ITEM
LM2 - SCHLST - SEARCHES A LIST OF ITEMS FOR A SPECIFIED ITEM
LM3 - ADDLST - ADDS AN ITEM TO THE END OF A SPECIFIED LIST
LM4 - RMVLST - REMOVES A SPECIFIED ITEM FROM A LIST OF ITEMS
LM5 - UNQLST - ADDS AN ITEM TO A LIST IF THE ITEM IS NOT IN THE LIST

Shading Database Routines

S1 - SHDRST - RESETS SHADING DATABASE
S2 - PTSSUN - PUTS SHADING SUN FLAG INTO SHADING DATABASE
S3 - GTSSUN - GETS SHADING SUN FLAG FROM SHADING DATABASE
S4 - PTSEYE - PUTS SHADING EYE FLAG INTO SHADING DATABASE
S5 - GTSEYE - GETS SHADING EYE FLAG FROM SHADING DATABASE
S6 - PTSCOL - PUTS SHADING RGB COLOR VALUES INTO SHADING DATABASE
S7 - GTSCOL - GETS SHADING RGB COLOR VALUES FROM SHADING DATABASE
S8 - PTSNLT - PUTS NUMBER OF SHADING LIGHT SOURCES INTO SHADING DATABASE

S9 - GTSNLT - GETS NUMBER OF SHADING LIGHT SOURCES FROM SHADING DATABASE
S10 - PTSLCR - PUTS LIGHT SOURCE RGB COLOR VALUES INTO SHADING DATABASE
S11 - GTSLCR - GETS LIGHT SOURCE RGB COLOR VALUES FROM SHADING DATABASE
S12 - PTSLGT - PUTS LIGHT LOCATION AND BRIGHTNESS INTO SHADING DATABASE
S13 - GTSLGT - GETS LIGHT LOCATION AND BRIGHTNESS FROM SHADING DATABASE
S14 - PTSSRF - PUTS SURFACE CHARACTERISTICS INTO SHADING DATABASE
S15 - GTSSRF - GETS SURFACE CHARACTERISTICS FROM SHADING DATABASE
S16 - PTSAMB - PUTS AMBIENT LIGHT FRACTION INTO SHADING DATABASE
S17 - GTSAMB - GETS AMBIENT LIGHT FRACTION FROM SHADING DATABASE

Miscellaneous Utility Routines

U1 - GTCNIN - GETS CONSOLE INPUT
U2 - ?????? - CURRENTLY UNDEFINED
U3 - MSG - WRITES A SCREEN MESSAGE
U4 - ?????? - CURRENTLY UNDEFINED
U5 - ?????? - CURRENTLY UNDEFINED
U6 - CAPS - CAPITALIZES A TEXT STRING
U7 - CK4FIL - CHECKS FOR THE EXISTENCE OF A SPECIFIED FILE
U8 - AQDSP - QUERIES ACTUAL DISPLAY SURFACE SIZE
U9 - ASVWM3 - SETS A VIEW MATRIX GIVEN VIEW ROTATION, TRANS. & SCALE
U10 - GTXFMT - CALCULATES A 3D TRANSFORMATION MATRIX
U11 - GTNAVL - GETS NUMBER OF AVAILABLE VALUATOR DEVICES
U12 - AINVL - INITIALIZES A VALUATOR DEVICE
U13 - AINST - INITIALIZES A STRING DEVICE
U14 - CONERR - WRITES A CONSOLE ERROR MESSAGE
U15 - CPYVCT - COPIES A VECTOR OF DATA FROM ONE VECTOR TO ANOTHER
U16 - INFMSG - WRITES AN INFORMATON MESSAGE ON THE SCREEN
U17 - AINLC - INITIALIZES A LOCATOR DEVICE
U18 - CKVLVL - CHECKS VALUATOR LIMITS
U19 - ?????? - CURRENTLY UNDEFINED
U20 - AGTVL - GETS LAST VALUATOR VALUE FROM A SET OF VALUATOR EVENTS
U21 - OPNUM - OPENS NUMERIC INPUT
U22 - CLNUM - CLOSES NUMERIC INPUT
U23 - PNMVL - PROCESSES NUMERIC VALUATOR INPUT
U24 - PNMST - PROCESSES NUMERIC STRING INPUT
U25 - INNMIN - INITIALIZES NUMERIC INPUT DEVICES
U26 - AGTST - GETS STRING INPUT FROM A PHIGS STRING DEVICE
U27 - GTINLN - GETS AN INPUT LINE FROM AN INPUT FILE

Vector and Matrix Routines

VM1 - VNORM - NORMALIZES A 3D VECTOR
VM2 - MATIN - INVERTS A MATRIX

APPENDIX L. ACSYNT/VPI DATABASE DEFINITIONS

This appendix contains the definitions of ACSYNT/VPI's databases including the subroutine used to get and put data to and from the database.

- Geometry Component Database

  NOTE: The geometry component real, integer, and character data can be loaded from an ACSYNT/VPI data file using the RDCPDT utility routine.

  - Component Data:

    Named Common: /CMPDB/ NCMPS
    Variable(s): NCMPS - integer number of aircraft components
    Put Routine: PTNCMP
    Get Routine: GTNCMP

  - Real Component Data:

    Named Common: /RCMPDB/ RDATA
    Variable(s): RDATA - real 2D array containing aircraft component parameters
    Put Routine: PTRCMP
    Get Routine: GTRCMP

  - Integer Component Data:

    Named Common: /ICMPDB/ IDATA
    Variable(s): IDATA - integer 2D array containing aircraft component attributes
    Put Routine: PTICMP
    Get Routine: GTICMP

  - Character Component Data:

    Named Common: /CCMPDB/ CDATA
    Variable(s): CDATA - character 2D array containing aircraft component labels
    Put Routine: PTCCMP
    Get Routine: GTCCMP

  - Real Component Cross Section Data:

    Named Common: /RXSDB/ RDATA
    Variable(s): RDATA - real 4D array containing component x-section data
    Put Routine: PTRXS
    Get Routine: GTRXS

  - Integer Component Cross Section Data:

    Named Common: /IXSDB/ IDATA

Variable(s): IDATA - integer 3D array containing aircraft x-section attributes
Put Routine: PTIXS
Get Routine: GTIXS

- Real Component Surface Data:

  Named Common: /RSRFDB/ RDATA
  Variable(s): RDATA - real 5D array containing aircraft surface data
  Put Routine: PTRSRF
  Get Routine: GTRSRF

- Geometry View Database

  NOTE: The geometry view database can be loaded from an ACSYNT/VPI data file with the RDVWDT utility routine.

  - Geometry View Data:

    Named Common: /VWDB/ NVWS
    Variable(s): NVWS - integer number of defined geometry views
    Put Routine: PTNVWS
    Get Routine: GTNVWS

  - Real Geometry View Data:

    Named Common: /RVWDB/ RDATA
    Variable(s): RDATA - real 2D array containing view parameters
    Put Routine: PTRVWS
    Get Routine: GTRVWS

  - Integer Geometry View Data:

    Named Common: /IVWDB/ IDATA
    Variable(s): IDATA - integer 2D array containing view attributes
    Put Routine: PTIVWS
    Get Routine: GTIVWS

  - Character Geometry View Data:

    Named Common: /CVWDB/ CDATA
    Variable(s): CDATA - character 2D array containing view labels
    Put Routine: PTCVWS
    Get Routine: GTCVWS

- Shading Database:

  NOTE: The entire shading database can be reset to predefined default values using the utility routine SHDRST.

  - Shading Flag Data:

    Named Common: /SHDLFG/ SUNFLG, EYEFLG
    Variable(s): SUNFLG - integer flag indicating that shading should be performed using a sun model
    EYEFLG - integer flag indicating that all light sources are at the eye of the observer
    Put Routine: PTSSUN and PTSEYE
    Get Routine: GTSSUN and GTSEYE

  - Shading Color Data:

Named Common: /SHDCOL/ COLORS
Variable(s): COLORS - real 2d data containing shading RGB color values
Put Routine: PTSCOL
Get Routine: GTSCOL

- Shading Light Source Data:

  Named Common: /SHDLGT/ NL, LOC, BRGHT
  Variable(s): NL - integer number of shading light sources
  LOC - real 2D array of light source locations
  BRGHT - real light source brightnesses
  Put Routine: PTSNLT and PTSLGT
  Get Routine: GTSNLT and GTSLGT

- Shading Light Color Data:

  Named Common: /SHDLCR/ LTGRGB
  Variable(s): LTGRGB - real light color RGB values for all light sources
  Put Routine: PTSLCR
  Get Routine: GTSLCR

- Shading Surface Characteristics:

  Named Common: /SHDSRF/ REFL, SHIN
  Variable(s): REFL - real surface reflectivity
  SHIN - real surface shininess
  Put Routine: PTSSRF
  Get Routine: GTSSRF

- Shading Ambient Light Fraction:

  Named Common: /SHDAMB/ B
  Variable(s): B - real shading ambient light fraction
  Put Routine: PTSAMB
  Get Routine: GTSAMB

APPENDIX M. SOFTWARE DEVELOPMENT LIBRARY USER'S GUIDE

M.1 Overview

This guide explains the operation and use of the ACSYNT/VPI online software development library (SDL). The guide is designed as a reference for the ACSYNT/VPI software development team and contains a functional description of the SDL and information for general users.

M.2 SDL Functional Description

M.2.1 Functional Overview

The ACSYNT/VPI software development library runs continuously as a disconnected service machine on Virginia Tech's VM3[3] computer system under the user identification of "NASASDL". As a disconnected service machine, NASASDL can automatically receive files and execute commands sent from privileged users. The ACSYNT/VPI SDL provides the following services:

[3] VM3 is a VM/CMS system running on an IBM 4341.

- stores master copies of all project documents, source code, executable code, development tools, and data files
- prevents multiple users from making changes to the same file at the same time (change control)
- supplies source file part numbers upon request
- maintains a source file part number log

M.3 Operation

NASASDL is controlled by a set of "REXX" command language (executive) programs and a special interrupt handling program called "NAPPER". NAPPER allows a CMS userid to automatically trap and process files, messages, and commands sent from other userids. NAPPER can also wake-up the userid at any time to perform tasks such as backing up important files and erasing scratch files.

After logging on to the NASASDL userid, NAPPER can be started by entering the following command:

```
NAP <cmd>
```

where:

NAP is the filename of the REXX command file which starts NAPPER

< cmd > is any command to be executed after invoking NAPPER

To put NASASDL in a disconnected state, parameter < cmd > should be "DISC" which will drop the connection between NASASDL and the terminal but still leave NASASDL running on the system.

NAPPER can also be started from any VM3 userid through the "AUTOLOG" command which automatically logs a CMS userid onto the system and runs a specified program. The format of the AUTOLOG command to run NAPPER on NASASDL is as follows:

```
AUTOLOG NASASDL <password> NAP
```

where:

< password > is the password for the NASASDL userid

NOTE: The AUTOLOG command will only work if the NASASDL userid is not already logged onto the system.

The after loading NAPPER into memory and executing the "< cmd >" command, the NAP EXEC calls NAPWAIT to wait for and process one of the following "userid interrupts":

console interrupt:
generated when the enter key is pressed on the console keyboard

message interrupt:
generated when a message is sent from another VM3 user

special message interrupt:
generated when a special message is sent from another VM3 user

reader interrupt:
generated when a file is sent from another VM3 user

wakeup interrupt:
generated at the time of day specified in a WAKEUP PARMS file

Each interrupt received is automatically logged in the file "AUDIT TRAIL" and then processed by one of the following command language programs:

NAPCONS	processes a console interrupt
NAPMSG	processes a message interrupt
NAPVMCF	processes a special message interrupt
NAPREAD	processes a reader interrupt
NAPWAKE	processes a wakeup interrupt

Each of the above programs are discussed in detail below.

M.3.1 NAPCONS

NAPWAIT calls NAPCONS whenever the ENTER key on NASASDL's console keyboard is pressed. For ACSYNT/VPI's software development library, NAPCONS simply issues the command "CP SMSG NASASDL SHUTDOWN" which causes NAPPER to shutdown and return control of NASASDL the user.

NOTE: If you log on to NASASDL while NASASDL is running NAPPER you will receive the message "RECONNECTED" and the "CP READ" indicator will appear at the bottom of screen. If you then enter the command "BEGIN", NASASDL will resume execution of NAPPER. At that point if you press the ENTER key, NAPPER will shut itself down and return control of NASASDL to you.

M.3.2 NAPMSG

NAPMSG processes "TELL" and "CP MSG" type messages sent from other VM3 users. The current NASASDL version of NAPMSG responds to all messages with the following message:

```
NASASDL ACTIVE ...
```

Sending a "TELL" message to NASASDL is a fast means of determining if NAPPER is functioning.

M.3.3 NAPVMCF

All command messages (special messages) sent to NASASDL are processed by NAPVMCF. When NAPVMCF is called to process a command, the privilege level of the userid which sent the command is first compared with the required privilege level for the command. If the user's privilege level is greater than or equal to the command's privilege level, the command will be executed;

otherwise the command is ignored. User and command privilege levels range from 0 (no privilege) to 50 (full privilege) and are contained in the files USERPRIV EXEC and CMDPRIV EXEC on the NASASDL userid. For example, a user with a privilege level of 50 can command NASASDL to erase the file "TEST DATA A" by entering the following:

```
CP SMSG NASASDL ERASE TEST DATA A
```

Commands defined to have a privilege level of 0 can be executed through NASASDL by any VM3 user. A user with a privilege level of 50 can execute any command through NASASDL. By default, all commands have privilege levels of 50 and all users have privilege levels of 0.

M.3.4 NAPREAD

NAPREAD is called by NAPWAIT whenever a new file is sent to NASASDL from a remote userid. NAPREAD's responsiblity is to decide whether or not to receive the file into the software development library. The decision is based on the privilege level of the user and the whether or not the user had previously checked out the file. For example, a file sent from a user with a privilege level of 50 is received regardless of whether or not the file is new or is on loan to another user. A file sent from a user with a privilege level less than 50 is only received if the file was previously checked out with the CHECKOUT command. Files that are not received are put "ON HOLD" by NAPREAD and must be either received or discarded by the individual in charge of NASASDL. Once a file becomes part of the NASASDL library, the file is available for checkout by one person at a time. Details concerning file checkout are contained in "General User Information" on page 263.

M.3.5 NAPWAKE

NAPWAIT calls NAPWAKE when NAPPER generates a wakeup interrupt at the times specified in NASASDL's "WAKEUP PARMS" file. The NASASDL WAKEUP PARMS file specifies when NASASDL is to execute a specified command. For example, the last two lines of the following WAKEUP PARMS file create a new backup copy of the NAPPER log file "AUDIT TRAIL A" every night at midnight:

```
*************************************************************************
*  'WAKEUP PARMS' FILE FOR NASASDL
*
*  Column  1: day/date to wakeup for this event
*         10: time of day/interval to wakeup
*         19: date/time stamp when completed
*         28: user text for this event - any CP/CMS/EXEC command
*
*  1          10        19          28       <-- COLUMNS
*  mm/dd/yy   hh:mm   datestamp   user-text   (done once
*  ==/dd/yy   hh:mm   datestamp   user-text   (once a month
*  ==/==/==   hh:mm   datestamp   user-text   (once a day
*  ==/01/==   hh:mm   datestamp   user-text   (on the 1st
*
*  ALL        hh:mm   datestamp   user-text   (once a day
*  dayofweek  hh:mm   datestamp   user-text   (once a week
*  WEEKEND    hh:mm   datestamp   user-text   (on weekends
*  S-S        hh:mm   datestamp   user-text   (same as above
*  WEEKDAY    hh:mm   datestamp   user-text   (on weekdays
*  M-F        hh:mm   datestamp   user-text   (same as above
*
*  ==/==/==   +05        timestamp  user-text   (every 5 minutes
*  WEEKEND    +10:30     timestamp  user-text   (every 10:30 on weekends
*  WEEKDAY    +20        timestamp  user-text   (every 20:00 on weekdays
*  dayofweek  +5         timestamp  user-text   (every 5 minutes on 'day'
*
*************************************************************************
==/==/== 00:00:00 04/16/88 ERASE AUDIT BACKUP A
==/==/== 00:00:00 04/16/88 RENAME AUDIT TRAIL A AUDIT BACKUP A
```

M.4 General User Information

M.4.1 Overview

General users of ACSYNT/VPI's software development library can perform the following:

- send messages to the NASASDL userid
- send commands to NASASDL
- check out files
- send files to NASASDL
- inquire and log file part numbers

Each of the functions listed above are discussed in detail below.

M.4.2 Sending Messages to NASASDL

To send a message to NASASDL from a remote userid, enter the following CMS command:

```
TELL NASASDL <msg>
```

where:

< msg > message text

Currently, NASASDL responds to all messages by sending back the message "NASASDL ACTIVE ..." which indicates that NASASDL is logged on and running NAPPER.

M.4.3 Sending Commands to NASASDL

To send a command to NASASDL from a remote userid, type the following command:

```
CP SMSG NASASDL <cmd>
```

where:

< cmd > is the command to be executed by NASASDL

NASASDL will only execute commands issued from privileged VM3 userids. Privilege levels are defined in the file USERPRIV EXEC and CMDPRIV EXEC located on the NASASDL userid.

NOTE: NASASDL can be shutdown and logged off of the system from a remote userid with the following command:

```
CP SMSG NASASDL SHUTDOWN LOGOFF
```

M.4.4 Checking Out Files from NASASDL

The CHECKOUT command is used to checkout files from NASASDL. In order to check out a file, a user must be defined in NASASDL's "USERPRIV EXEC" with a privilege level of at least 40. The format of the CHECKOUT command is as follows:

```
CHECKOUT <fn> <ft> <fm>
```

where:

< fn > is the filename of the file to be checked out of the SDL. If left blank the list of files in the SDL will be sent to your userid.

< ft > is the filetype of the file to be checked out of the SDL. If left blank, the list of files in the SDL with a filename of < fn > will be sent to your userid.

< fm > is the filemode of the file to be checked out of the SDL. If left blank a filemode of "A" will be assumed.

If the specified file has already been checked out by another person, you will receive the following messages from NASASDL:

```
FILE <<fn> <ft> <fm>> CHECKED OUT BY <userid> ON <date> AT <time>
FILE <<fn> <ft> <fm>> CAN NOT BE CHECKED OUT
```

where:

< fn > filename of file requested

< ft > filetype of file requested

< fm > filemode of file requested

< userid > userid which checked out the specified file

< date > the checkout date

< time > the checkout time.

NASASDL will also issue the following message to the userid which currently has the checked out file:

```
FILE <<fn> <ft> <fm>> NEEDED BY ANOTHER USER.
```

If the file has not been checked out previously, you will receive the following messages:

```
FILE <<fn> <ft> <fm>> CHECKED OUT ON <date> AT <time>
PUN FILE <spoolid> FROM NASASDL COPY 001 NOHOLD
```

where:

< fn > filename of file requested

< ft > filetype of file requested

< fm > filemode of file requested

< date > the checkout date

< time > the checkout time

< spoolid > file spool identifier

The file checked out of NASASDL will be sent to your userid and can be copied to your "A" disk by using the "RECEIVE" command. Information concerning the RECEIVE command is available through the CMS help facility.

M.4.5 Sending Files to NASASDL

The CMS "SENDFILE" command is used to send files to NASASDL. The syntax of SENDFILE is as follows:

```
SENDFILE <fn> <ft> <fm> NASASDL
```

where:

< fn > filename of file to be sent to NASASDL

< ft > filetype of file to be sent to NASASDL

< fm > filemode of file to be sent to NASASDL

When a file previously checked out of the development library is checked back in, NASASDL will issue the following message:

```
FILE <<fn> <ft> <fm>> CHECKED OUT BY <userid> ON <date_out> AT <time_out>
- CHECKED IN ON <date_in> AT <time_in>
```

where:

< fn > filename of file checked into NASASDL

< ft > filetype of file checked into NASASDL

< fm > filemode of file checked into NASASDL

< userid > your userid

< date_out > date checked out

< time_out > time checked out

< date_in > date checked in

< time_in > time checked in

In most cases, if you send a *new* file to the SDL the file will be placed "on hold". The person in-charge of NASASDL is responsible for actually receiving new files into the library. Once a file in part of the library it is available for checkout by other privileged users.

M.4.6 Inquiring and Logging File Part Numbers

The PARTNUM command is used to query NASASDL for the next available file part number and to log the description of the file. The format of the PARTNUM command is as follows:

```
PARTNUM <cmd>
```

where:

< cmd > is one of the following part number commands:

INCrement	increments the current part number by 1 without logging a part description
DECrement	decrements the current part number by 1 without logging a part description
OFFset +/-n	offsets the current part number by +/- n where n is an integer
SET x	sets the current part number to x where x is any hex number between A000 and FFFF.
LIst	commands NASASDL to send a copy of the part log to your userid
LOg < fn > < text >	Increments the part number by 1 and logs < text > under the new part number and the filename < fn >
QUEry	queries the current part number without logging a part description or incrementing the part number
PREfix < string >	commands the SDL to change the part number prefix to < string >

After issuing the PARTNUM command, you will receive the following message from NASASDL:

```
CURRENT PART NUMBER = <prefix>-<partnum>
```

where:

< prefix > is the part number prefix

< partnum > is the current part number

APPENDIX N. SOFTWARE DEVELOPMENT TOOLS USER'S GUIDE

This appendix describes the use of the software development tools created during the development of ACSYNT/VPI. All of the tools are CMS command language programs which were written to run on Virginia Tech's IBM 4341 mainframe under CMS. Each tool is listed by CMS file identifier and can be obtained from the ACSYNT/VPI on-line software development library.

N.1 COMMENT EXEC and COMMENT XEDIT

COMMENT is used to count the number of comment lines in a FORTRAN source file to provide programmers with a rough estimate of how thoroughly they have documented a program. To execute COMMENT from CMS type the following:

```
COMMENT <fn> <ft> <fm>
```

where:

fn is the source filename of the FORTRAN source file to be processed by COMMENT

ft is the optional source filetype of the file to be processed by COMMENT (default = FORTRAN)

fm is the optional source filemode of the module to be processed by COMMENT (default = "A")

COMMENT can also be executed from the XEDIT command line to count the number of comments in the current XEDIT file. To execute COMMENT from XEDIT, simply type "COMMENT" on the XEDIT command line.

Below is an example of the output produced by COMMENT:

```
LINE AND WORD COUNT INFORMATION FOR <TESTFILE FORTRAN A>:

NUMBER OF SOURCE LINES  = 300
NUMBER OF CODE LINES    = 150
NUMBER OF BLANK LINES   = 50
NUMBER OF COMMENT LINES = 158
NUMBER OF COMMENT WORDS = 527

COMMENT LINE PERCENTAGE =  52.67%
COMMENT WORDS PER LINE  =   5.27
```

N.2 MOD EXEC

The MOD command is used to create a new FORTRAN source file containing a copy of the comment header block stored in the file MODULE TEMPLATE. The syntax for MOD is as follows:

```
MOD <fn>
```

where:

< fn > is the name of the module file to be created.

Before using MOD EXEC, a MODULE TEMPLATE file should be obtained from the software development library.

N.3 GRAB XEDIT

The GRAB command is a modification of the XEDIT command "GET" which allows a portion of a file to be included into the current XEDIT file after the current line. GRAB can only be issued from the XEDIT command line. The format of the GRAB command is as follows:

```
GRAB <fn> <ft> <fm> <start_ln> <end_ln>
```

where:

< fn > is the file name of the file to be included in the current XEDIT file

< ft > is the file type of the file to be included in the current XEDIT file

< fm > is the file mode of the file to be included in the current XEDIT file

start_ln is the starting line number of the block of text to be included in the current XEDIT file

end_ln is the ending line number of the block of text to be included in the current XEDIT file.

The only difference between GRAB and GET is that GET requires the NUMBER of lines to include instead of the range of lines.

N.4 MATH XEDIT and MATH EXEC

The MATH command can be issued from CMS or XEDIT and is used to evaluate simple mathematical expressions. The format of the MATH command is as follows:

```
MATH <expression>
```

where:

< expression > is the any simple mathematical expression containing multiplication, division, addition, and subtraction.

For example, to evaluate the expression "(4*5)/2+.05" type the following:

```
MATH (4*5)/2+.05
```

MATH will respond with the following answer:

```
(4*5)/2+.05 = 10.05
```

N.5 COMP EXEC

The COMP command is used to compile FORTRAN source files independent of the file's CMS filetype. In other words, the FORTRAN source file does not have to have a file type of "FORTRAN". The format of the COMP command is as follows:

```
COMP <fn> <ft> <fm>
```

where:

<fn> is the filename of the FORTRAN file to be compiled

<ft> is the filetype of the FORTRAN file to be compiled

<fm> is the filemode of the FORTRAN file to be compiled.

A file may also be compiled by typing COMP next to name of a FORTRAN file appearing in a CMS filelist.

N.6 FORT XEDIT

The FORT command is similar to the COMP command in that FORT will compile any FORTRAN source file regardless of the file's filetype. Unlike COMP, the FORT command is issued from within XEDIT and performs the following functions:

- turns file packing off (see XEDIT help)
- saves the current XEDIT file
- Invokes the CMS FORTRAN compiler if the filetype of the XEDIT file is "FORTRAN"
- Invokes the COMP command if the filetype of the XEDIT file is not "FORTRAN"
- issues an error message if any warning or error messages were generated during compilation
- turns file packing on.

The format of the FORT command is simply:

```
FORT
```

The FORT command can only be executed from the XEDIT command line.

N.7 BACKUP EXEC

The BACKUP command is used to create a backup of any CMS file. The format of the BACKUP command is as follows:

```
BACKUP <fn> <ft> <fm>
```

where:

< fn > is the filename of the file to be backed up

< ft > is the filetype of the file to be backed up

< fm > is the filemode of the file to be backed up.

BACKUP copies the specified file to the file "< fn > BACKUP < fm >". If a backup file already exists for the specified filename, you will be asked the following:

```
Replace '<fn> <ft> <fm>' ? (y|n)
```

If you enter a "y" then the old backup file will be overwritten.

N.8 TRASH EXEC

The TRASH command provides a safe means of erasing files from a CMS disk and is an alternative to using the CMS ERASE command. TRASH creates a temporary disk which acts as a trash can for unwanted files. The temporary disk and its contents are discarded whenever the userid is logged off the the system. All files "trashed" with the TRASH command are actually just copied to the temporary disk and erased from the specified disk. If a file is trashed by accident, the file can be retrieved by copying it from the temporary disk.

The format of the TRASH command is as follows:

```
TRASH fn ft fm
```

where:

fn is the filename of the CMS file to be trashed and must be specified

ft is the filetype of the CMS file to be trashed and must be specified

fm is the filemode of the CMS file to be trashed and if not specified will be assumed to be "A"

The exec TEMPDISK EXEC is used to create the temporary disk and is a modification of the TDISK EXEC available on Virginia Tech's VM3 system.

N.9 SEARCH EXEC

The SEARCH command is used to search a file for a specified string of characters. The format for SEARCH is as follows:

```
SEARCH fn ft fm string
```

where:

fn is the filename of the file to be searched and is required

ft is the filetype of the file to be searched and is required

fm is the filemode of the file to be searched and is required

string is the string to be searched for in the specified file

If "string" is not found in the specified file then the following message will be generated:

```
STRING <string> NOT FOUND IN FILE <fn ft fm>.
```

If "string" is found, the file will be XEDITed and only the lines containing "string" will be shown.

To view all of the lines in the file type "ALL" on the XEDIT command line.

APPENDIX O. ACSYNT/VPI USER'S GUIDE

O.1 Overview

This guide is designed to document the use of ACSYNT/VPI development version X1.0 by answering the following questions:

- How is the ACSYNT/VPI software accessed?
- What requirements must be meet to run ACSYNT/VPI?
- What functions can ACSYNT/VPI perform and how are they performed?

The information contained in this guide is organized by functional topic such as how to run ACSYNT/VPI, how to display an aircraft's image, how to generate a shaded image, etc. The reader is assumed to be familiar with the logon procedure for Virginia Tech's VM3 system and to have access to a VM3 userid.

O.2 Functional Capabilities of ACSYNT/VPI Version X1.0

With the development version of ACSYNT/VPI, a user can perform the following functions:

- read an ACSYNT geometry common block data file containing ACSYNT's definition of an aircraft's geometry

- read an ACSYNT/VPI geometry data file which contains viewing parameters, aircraft component attributes, and component location and orientation data
- display the aircraft's geometry as a three dimensional wireframe image
- rotate, translate, and scale the three dimensional image using valuator dials or menu input
- display multiple views of the aircraft
- store and retrieve viewing parameters
- switch individual views between parallel and perspective projections
- change the color and surface approximation values for individual aircraft components
- no-show (make invisible) selective aircraft components
- produce a hidden surface image of the aircraft
- produce a shaded image of the aircraft
- mix hidden surface and shading on an aircraft component level
- change the red, green, and blue color fractions of the first 15 displayable workstation colors

All of the above functions are discussed in the "ACSYNT/VPI Functions" section of this user's guide.

O.3 ACSYNT/VPI Run-time Requirements

O.3.1 Requirements Overview

All of the requirements to run ACSYNT/VPI on Virginia Tech's VM3 system can be categorized under the following headings:

- Userid Requirements
- File Requirements
- Hardware Requirements

Each of the above requirements are discussed in detail below:

O.3.2 Userid Requirements

To run ACSYNT/VPI on a Virginia Tech VM3 userid, the userid must meet the following requirements:

- must have access to at least 12K kilobytes of virtual storage (memory)
- must have the userid's real timer turned on
- must have at least 10 loader tables defined

If any one of the above requirements are not met, ACSYNT/VPI will not function properly. A discussion of each userid requirement is presented below.

O.3.2.1 MEMORY REQUIREMENT

To determine the amount of virtual memory which can be accessed by your userid, enter the following command after logging onto the system:

```
QUERY STORAGE
```

The system will respond with:

```
STORAGE = <n>K
```

where:

<n> is the number of kilobytes of accessible virtual memory

If the number <n> is less than 12K then you will need to increase your virtual storage with the following directory maintenance command:

```
DIRM STORAGE 12228K
```

You will then be asked to enter your password to validate the memory change. Once you have entered your password you will receive the following message:

```
COMMAND DIRM STORAGE : SOURCE UPDATED AND CHANGE ONLINE.
```

The actual memory change will not take place until you logoff of the system and then log back on at which point the "QUERY STORAGE" command will respond with the following:

```
STORAGE = 12288K
```

O.3.2.2 REAL TIMER REQUIREMENT

Each VM3 userid has a "virtual timer" which is updated only during virtual CPU run time. An optional "real timer" can be activated which is updated during both virtual CPU run time and wait time. For the ACSYNT/VPI input routines to function properly the real timer must be activated using the following directory maintenance command:

```
DIRM OPTION REALTIMER YES
```

The system will respond by asking you to enter you password after which you will receive the following message:

```
COMMAND DIRM OPTION : SOURCE UPDATED AND CHANGE ONLINE.
```

As with the "DIRM STORAGE" command, the real timer is not actually activated until the next time you log onto the system.

O.3.2.3 LOADER TABLE REQUIREMENT

Loader tables are used to store the names of externally defined routines when a program is loaded into memory for execution. Due to the large number of routines used in ACSYNT/VPI, the default of three loader tables is insufficient and must be increased in order for ACSYNT/VPI to load properly. The following command will increase the number of loader tables to 10:

```
SET LDRTBLS 10
```

The command should be placed in your userid's "PROFILE EXEC" if ACSYNT/VPI will be executed on a regular basis.

O.3.3 File Requirements

To run ACSYNT/VPI the following files should be copied from the VM3 userid ACSDEM using the CMS command "COPYFILE"[4] :

FILE SETUP
contains default ACSYNT/VPI aircraft geometry data

FILE DBLTIAL
contains example ACSYNT/VPI aircraft geometry data for a V-tailed aircraft

FILE ACSDATA
contains example ACSYNT geometry common block data for an F16

ACSVPI EXEC
is the command language program used to run ACSYNT/VPI

Access to the above files can be obtained through the following disk link command:

```
LDISK ACSDEM
```

O.3.4 Hardware Requirement

The development version of ACSYNT/VPI will currently run only on an IBM 5080 workstation. To run ACSYNT/VPI, an IBM 5080 must be logically attached to your userid. On Virginia Tech's VM3 system, an IBM 5080 is attached using the following procedure:

1. logon to VM3 on a text type terminal
2. press the RESET and ENTER keys on the IBM 5080 of your choice
3. wait for the GTM (Graphics Task Monitor) screen to appear
4. enter your VM3 userid on the first input line

---

[4] Help for "COPYFILE" command can be obtained by entering the CMS command "HELP COPYFILE".

5. press the "jump key"[5] twice to move the cursor to the input field containing the word "YES"
6. type the word "NO" and press the space bar
7. press the enter key

After pressing the enter key, the IBM 5080 screen will clear and you will receive the following message at your text terminal:

```
GRAF 200 ATTACHED
```

The message indicates that the IBM 5080 is now logically attached to your userid and you can now run ACSYNT/VPI.

O.4 Running ACSYNT/VPI

If you have met all of the requirements listed in the previous section you can run ACSYNT/VPI by typing the following command:

```
ACSVPI
```

ACSVPI is a command language program which performs all of the actions necessary to run ACSYNT/VPI such as linking to and loading the ACSYNT/VPI software.

After entering the ACSVPI command you will receive the following messages on your text terminal screen:

---

[5] The "jump key" on an IBM 5080 is the key with the right facing arrow pointing to a vertical bar (-->|).

```
                          ******************************
                          * ACSYNT/VPI - VERSION X1.0 *
                          ******************************

WORKSTATION TYPE --> 5080
CONNECTION ID    --> IBM 5080

ACCESSING ACSYNT/VPI AND PHIGS DISKS ...
T (104) R/O
U (105) R/O
V (106) R/O

LOADING ACSYNT/VPI ...

STARTING ACSYNT/VPI ...

EXECUTION BEGINS...
ENTER WORKSTATION TYPE (INTEGER):
?
ENTER WORKSTATION CONNECTION ID (INTEGER):
?
```

The two questions following the "EXECUTION BEGINS..." are asked by ACSYNT/VPI but are automatically answered by the ACSVPI program. After the above messages are printed on your text screen, the ACSYNT/VPI main menu screen should appear on the IBM 5080 screen.

NOTE: Depending upon the current system load, the main menu screen may take up to one minute to appear.

When the menu does appear, the following two messages will be printed in the blue message area at the bottom of the screen:

```
PREVIOUS MODEL RESTORED ...
PICK A MENU ITEM:
```

The "PREVIOUS MODEL RESTORED ..." message indicates that the ACSYNT/VPI model file "FILE SETUP" was read successfully.

O.5 ACSYNT/VPI Functions

O.5.1 Overview

The following pages contain details of how to use the functions currently available in ACSYNT/VPI. Under each function the following format is used to explain the function's use:

Purpose:
explains the purpose of the function.

Menu Path:
shows the nesting of ACSYNT/VPI menu items which lead to the desired function. The example below indicates where the "geometry-color-modification" function can be found:

```
MAIN ( GEOMETRY ( MODIFY ( COLOR ) ) )
```

Description:
describes the function's use including an example of the ACSYNT/VPI message and input areas as shown below:

```
message line 5
message line 4
message line 3
message line 2
MESSAGE LINE 1 (CURRENT MESSAGE)
-------------------------------------
INPUT AREA
```

The message and input areas appear at the bottom of ACSYNT/VPI's screen.

NOTE: The current message line is displayed in white on the ACSYNT/VPI screen but is shown in capital letters throughout this guide. Also, the under line ("_") in the input area represents the keyboard cursor.

Example:
provides an example of the function's used

O.5.2 Reading ACSYNT Geometry Common Block Files

Purpose:

To load the ACSYNT definition of the aircraft's geometry.

Menu Path:

MAIN (DESIGN)

Description:

When you select the "DESIGN" menu item from the main menu you will be ask to enter the name of an ACSYNT geometry common block data file as shown below:

```
previous model restored ...
pick a menu item:
ENTER FILE NAME:
ACSDATA
```

Common block data files are generated each time an aircraft model is analyzed with ACSYNT and serve as the current means of passing data between ACSYNT and ACSYNT/VPI.

You can press the ENTER key to accept the default file of "ACSDATA" or type in a new file name. The file name entered is actually the CMS *filetype*. The actual CMS filename and filemode are by default "FILE" and "A" respectively.

If the name of the file you entered does not exist on your userid, you will receive a "FILE NOT FOUND ..." message and will be asked to reenter the file name. If you wish to terminate the input without entering a file name, press the "ALT" and "CNCL" keys on the 5080 keyboard at the same time. You will then receive the message "INPUT TERMINATED ...". If the file is found, you will receive the message "INPUT COMPLETE ...".

Example:

1. from the ACSYNT/VPI main menu select the "DESIGN" menu item
2. press the "ENTER" key to accept the default geometry file of "ACSDATA" which should be on your userid's "A" disk (ACSYNT/VPI should respond with "INPUT COMPLETE ...".)

O.5.3 Reading ACSYNT/VPI Geometry Files

Purpose:
to load geometry view and aircraft component data into ACSYNT/VPI's view and geometry databases

Menu Path:
MAIN (FILE)

Description:
If you select the "FILE" MAIN menu item, the FILES sub-menu will appear and you will be prompted to enter the name of an ACSYNT/VPI geometry data file as shown below:

```
previous model restored ...
pick a menu item:
enter file name:
input complete ...
ENTER FILE NAME OR SELECT FROM MENU:
GEOM
```

In the development version of ACSYNT/VPI, the only item on the FILES menu is "RETURN" which allows you to abort the input of a file name and exit to the main menu.

When prompted for a file name, you have the option of selecting the default file of "GEOM" or entering the name of another file. If the file you enter does not exist on your userid, you will receive the message "FILE NOT FOUND ..." and will be prompted to reenter a file name. If the file you enter does exist you will receive the message "INPUT COMPLETE ...".

Example:

1. from the ACSYNT/VPI MAIN menu select the "FILE" menu item
2. in place of the default file name of "GEOM" type "DBLTAIL" and press the ENTER key (The DBLTAIL file contains the definition of an aircraft with a "V" shaped vertical tail.)
3. select the "FILES" menu item "RETURN" to exit back to the main menu.

O.5.4 Displaying the Aircraft's Geometry

Purpose:
to display the wireframe image of the aircraft's geometry and to enter the geometry submenu.

Menu Path:
MAIN (GEOMETRY)

Description:

The "GEOMETRY" menu item on the MAIN menu is used to enter the geometry sub-menu from which you can view, modify, and shade the aircraft. When the "GEOMETRY" menu item is selected you will receive the following message:

```
enter file name:
input complete ...
enter file name or select from menu:
input complete ...
CREATING WIREFRAME GEOMETRY ...
_
```

After the wireframe image appears you will be prompted to select a menu item as shown below:

```
input complete ...
enter file name or select from menu:
input complete ...
creating wireframe geometry ...
SELECT A MENU ITEM:
_
```

The geometry will be removed from the screen whenever you exit the GEOMETRY menu.

Example:

1. from the MAIN menu, select the "GEOMETRY" menu item (In a few moments the aircraft's geometry will appear.).

O.5.5 Performing Three Dimensional Viewing Transformations

Purpose:

to rotate, translate, and zoom the three dimensional image of the aircraft.

Menu Path:

any geometry menu not requesting numeric input

Description:

The workstation valuator dials are used to perform viewing transformations and are mapped as follows:

- dial 1 - global x-axis rotation
- dial 2 - global y-axis rotation
- dial 3 - global z-axis rotation
- dial 4 - global x-axis translation
- dial 5 - global y-axis translation
- dial 6 - global z-axis translation
- dial 7 - view scale

If multiple views of the aircraft model are displayed then the viewing transformations are applied to the "current view". The current view is changed by selecting the marker shown in the bottom left corner of any view.

Example:

1. after displaying the aircraft's geometry, rotate any one of the seven valuator dials.

O.5.6 Changing Aircraft Component Colors

Purpose:
to change the color of an aircraft component such as the wing, fuselage, canopy, etc.

Menu Path:
MAIN (GEOMETRY (MODIFY (COLOR)))

Description:
Each aircraft component is displayed as a wireframe image in one of six base colors referenced by an color index. Each color index represents a color available on the IBM 5080. For example, color index 1 is by default red but can be changed as described in "Changing Workstation Colors" on page 303.

The allowable component color index values actually range from -6 to +6. The actual color index used to display the component's wireframe image is the absolute value of the component's color index. The negative color indices are used to designate components which are to be shown only as hidden surface images by the shading algorithm. Components with zero color indices are "camouflaged" and components with positive color indices are shaded normally. For each of the 6 base component colors, 16 shades are derived for use in shading. By mixing positive and negative component colors, the shading function will produce a combined shaded and hidden surface image.

After selecting the "COLOR" menu item from the MODIFY menu, the message and input area should appear as follows:

```
input complete ...
creating wireframe geometry ...
select menu item:
enter value(s) using dials and/or keyboard:
ENTER COLOR INDEX AND PICK COMPONENT:
_     1
```

To change the color of a component, either type in an integer number between -6 and +6 or dial in a value with valuator dial 1. When the desired color index is displayed, you can select an aircraft component to change the component's color or select the "COLOR ALL" menu item will to change all component colors. If you select the "RESET" menu item, the component color index in the input area will be reset to a value of 1.

Example:

1. from the GEOMETRY menu select the "MODIFY" menu item to display the MODIFY submenu
2. select the "COLOR" menu item to display the "COMP. COLOR" submenu
3. type or dial in a component color index
4. select one of the aircraft's components (The color of the component should change.)
5. select the "COLOR ALL" menu item to change the color of all of the components.

O.5.7 Changing the Number of Surface Tiles

Purpose:
to change the number of wireframe curves drawn on each component.

Menu Path:
MAIN (GEOMETRY (MODIFY (RE-TILE)))

Description:
The wireframe image of an aircraft is generated by drawing a grid of isoparametric curves on the surfaces which define each aircraft component. The default number of surface curves is 2 in both the u and w directions resulting in a curve on each surface boundary. The intersection between surface curves form the vertices for polygon tiles used to shade the aircraft. The smoothness and time required to produce a shaded image are both functions of the number of surface tiles on each component.

The "RE-TILE" menu item under the MODIFY menu will request the number of surface curves as shown below:

```
enter value(s) using dials and/or keyboard:
enter color index and pick component:
all component colors set ...
enter value(s) using dials and/or keyboard:
ENTER # OF SURF. CURVES IN U & W DIR. AND PICK COMP.:
_      2,       2
```

Once inside the RE-TILE menu, you can either type or dial in the number of surface curves and then select a component to be retiled. ACSYNT/VPI automatically redraws the component's wireframe image to reflect the change. You may also retile all of the aircraft's components using the "RETILE ALL" menu item.

Example:

1. Enter the RE-TILE menu and dial in "4,4"
2. select the wing
3. once the wing is redrawn, select the "RETILE ALL" menu item to see the effect of retiling all of the components.

O.5.8 No-showing Aircraft Components

Purpose:
to make selected aircraft components invisible.

Menu Path:
MAIN (GEOMETRY (DISPLAY (NO-SHOW)))

Description:
The no-show function allows you to temporarily remove aircraft components from the display. When you enter the NO-SHOW menu, you will be prompted to pick components as shown below:

```
select menu item:
creating wireframe geometry ...
select menu item:
select geometry display option from menu:
PICK COMPONENT TO NO SHOW OR PICK MENU ITEM:
_
```

You can use the "FLIP" menu item to show all no-showed components and no-show all showed components. Selecting "SHOW ALL" will cause all of the components to be shown.

Example:

1. enter the NO-SHOW menu and select the wing, vertical tail, and horizontal tail components (Each component should disappear as you select them.)
2. select the "FLIP" menu item to see only the selected components
3. select "SHOW ALL" to make all of the components visible.

O.5.9 Producing a Shaded Image

Purpose:
to generate a shaded image of the aircraft's geometry.

Menu Path:
MAIN (GEOMETRY (SHADE))

Description:
To shade an aircraft's geometry, enter the shading submenu and select the view marker at the bottom left corner of the view to be shaded. ACSYNT/VPI will respond with the following message:

```
creating wireframe geometry ...
select menu item:
shade:
select view to be shaded:
SHADING ...
_
```

The time required to produce a shaded image and the quality of the image is a function of the density of the surface grid lines (see "Changing the Number of Surface Tiles" on page 287). The higher the number of surface grid lines, the longer the shading will take but the smoother the resulting image. If you have multiple views of the aircraft on the screen, the shaded image will appear only in the view you have selected.

To perform selective component shading, pick only the aircraft components you wish to be shaded. ACSYNT/VPI will highlight each component and shade only the ones you selected. All other components will be removed from the view.

If you change one of the shading parameters such as the location of a light source, you can reshade a view by reselecting the view's marker. Shading will also automatically be reperformed on the most recently shaded view if you turn one of the viewing valuator dials (see "Performing Three Dimensional Viewing Transformations" on page 285).

To drop the shaded image from all views, select the "RETURN" menu item on the SHADING menu. More information concerning shading is provided in the following sections:

- "Change the Shading Ambient Light Fraction" on page 290
- "Changing the Shading Surface Characteristics" on page 290
- "Switching Sun Model Shading On and Off" on page 291
- "Switching the Light Source Eye Flag On and Off" on page 292
- "Controlling the Number of Shading Light Sources" on page 293
- "Specifying the Color of All Shading Light Sources" on page 293
- "Specifying Shading Light Parameters" on page 294
- "Resetting the Shading Parameters to Default Values" on page 296

Example:

1. enter the shading sub-menu and select the view marker of a view to be shaded.
2. once the shaded image appears, rotate valuator dial 1 which will cause the geometry to automatically be shaded in a different orientation.

O.5.10 Change the Shading Ambient Light Fraction

Purpose:
allows the user to specify the fractional amount of ambient light used to produce a shaded image.

Menu Path:
MAIN (GEOMETRY (SHADING (AMBIENT LIGHT)))

Description:
When you select the "AMBIENT LIGHT" menu item the message and input area should appear as follows:

```
select menu item:
shade:
select view to be shaded:
enter value(s) using dials and/or keyboard:
ENTER AMBIENT LIGHT FRACTION:
_    0.30
```

The ambient light fraction controls the amount of light illuminating the aircraft from all directions and can be any real number between 0.0 and 1.0. The ACSYNT/VPI default value of 0.3 will to give good results but you can change the value by dialing or typing in a new number between 0.0 and 1.0.

Example:

1. from the SHADE menu, shade the aircraft model using the default ambient light value of 0.3
2. select the "AMBIENT LIGHT" menu item
3. enter a new ambient light fraction of 0.5
4. exit the AMBIENT LIGHT menu and reshade the aircraft model to see the effect of changing the ambient light fraction.

O.5.11 Changing the Shading Surface Characteristics

Purpose:
to allow the user to control the reflectivity and shininess of the aircraft model.

Menu Path:
MAIN (GEOMETRY (SHADING (SURFACE CHAR.)))

Description:
The reflectivity of the aircraft's surface is defined as the ratio of light reflected to the total incoming light and can vary from 0.0 to 1.0. Shininess specifies how glossy the surface will

appear and can vary from 0.1 to 50.0. The higher the value of shininess, the shinier the surface.

After selecting the "SURFACE CHAR." menu item from the shading menu, the message and input area should appear as follows:

```
select menu item:
shade:
select view to be shaded:
enter value(s) using dials and/or keyboard:
ENTER SURFACE REFLECTIVITY AND SHININESS:
_    0.10,  30.00
```

By default, the reflectivity and shininess used for shading is 0.1 and 30.0 respectively. Both values can be changed by typing or dialing in the desired values.

Example:

1. shade the aircraft using the default reflectivity and shininess
2. change the reflectivity and shininess to "0.20, 20.0" using the "SURFACE CHAR." menu item
3. reshade to see the effect of reflectivity and shininess.

O.5.12 Switching Sun Model Shading On and Off

Purpose:
to allow the user to produce shaded images using a sun model.

Menu Path:
MAIN (GEOMETRY (SHADING (SUN MODEL)))

Description:
The sun model menu item allows you to turn on and off sun model shading. When the sun model is active, the light rays emitted from all of the light sources are assumed to be parallel and do not decrease in intensity with distance. The sun model simulates the effect of the sun shining on the aircraft. If you turn the sun model off, the light rays emitted from each light source will diverge and decrease in intensity with distance.

Sun model shading is by default "active". When you first select the "SUN MODEL" menu item from the SHADE menu, you will receive the following message indicating that the sun model has been turned off:

```
creating wireframe geometry ...
select menu item:
shade:
select view to be shaded:
SUN MODEL NOW INACTIVE.
_
```

If you then select the "SUN MODEL" item again the shading sun model will be activated and you will receive the following message:

```
select menu item:
shade:
select view to be shaded:
sun model now inactive.
SUN MODEL NOW ACTIVE.

_
```

Example:
Please see the example in "Specifying Shading Light Parameters" on page 294.

O.5.13 Switching the Light Source Eye Flag On and Off

Purpose:
to force the light source to be located at the eye of the observer.

Menu Path:
MAIN (GEOMETRY (SHADING (EYE FLAG)))

Description:
When the eye flag is on, all of the light sources are forced to be located at the eye of the observer (your eye) which means that the rotated image of the aircraft will always be fully illuminated. The eye of the observer is automatically defined by the view selected for shading.

The shading eye flag is by default "on" which means that all light sources are assumed to be at the eye of the observer as opposed to fixed in space. When you select the SHADE menu's "EYE FLAG" item for the first time, you will receive the following message:

```
shade:
select view to be shaded:
sun model now inactive.
sun model now active.
LIGHT SOURCE(S) NOW FIXED AT SPECIFIED COORDINATES

_
```

If you select the "EYE FLAG" again you will receive the following message:

```
select view to be shaded:
sun model now inactive.
sun model now active.
light source(s) now fixed at specified coordinates
LIGHT SOURCE(S) NOW FIXED AT EYE OF OBSERVER

_
```

If you turn the eye flag is off, each light source location is defined by a three dimensional point specified by selecting the "LIGHT SOURCES" menu item (see "Specifying Shading

Light Parameters" on page 294). Each light source will appear as asterisks on the ACSYNT/VPI screen and are subject to the same viewing transformations as the aircraft's geometry.

Example:
Please see the example in "Specifying Shading Light Parameters" on page 294.

O.5.14 Controlling the Number of Shading Light Sources

Purpose:
to allow the user to change the number of light sources used for shading.

Menu Path:
MAIN (GEOMETRY (SHADING (NUM. OF LIGHTS)))

Description:
Whenever you specify a new light source with the "LIGHT SOURCES" menu item, the number of light sources used for shading is automatically increased. Therefore, the "NUM. OF LIGHTS" menu item is typically used only to reduce the number of lights.

When you select the "NUM. OF LIGHTS" SHADE menu item, you will receive the following prompt:

```
sun model now active.
light source(s) now fixed at specified coordinates
light source(s) now fixed at eye of observer
enter value(s) using dials and/or keyboard:
ENTER NUMBER OF LIGHT SOURCES:
_       1
```

To change the number of light sources either type or dial in the desired value and then select the "RETURN" menu item.

Example:
Not applicable.

O.5.15 Specifying the Color of All Shading Light Sources

Purpose:
to change the red, green, blue color fractions used for all light sources

Menu Path:
MAIN (GEOMETRY (SHADING (LIGHT COLOR)))

Description:
The default color for all shading light sources is white but can be changed using the

"LIGHT COLOR" menu item. When you select "LIGHT COLOR" you will be asked to enter the fractions of red, green, and blue (RGB) to be used for all light sources as shown below:

```
light source(s) now fixed at eye of observer
enter value(s) using dials and/or keyboard:
enter number of light sources:
enter value(s) using dials and/or keyboard:
ENTER LIGHT RED, GREEN, BLUE FRACTIONS:
_    1.00,   1.00,   1.00
```

NOTE: The actual values may vary.

The following is a list of common colors and their corresponding RGB color values:

WHITE	= (1.0, 1.0, 1.0)
RED	= (1.0, 0.0, 0.0)
GREEN	= (0.0, 1.0, 0.0)
BLUE	= (0.0, 0.0, 1.0)
YELLOW	= (1.0, 1.0, 0.0)
MAGENTA	= (1.0, 0.0, 1.0)
TURQUOISE	= (0.0, 1.0, 1.0)
PURPLE	= (0.5, 0.3, 0.9)
ORANGE	= (0.8, 0.4, 0.0)
GRAY	= (0.5, 0.5, 0.5)

NOTE: The color values entered apply to *all* light sources.

Example:

1. select the "LIGHT COLOR" menu item
2. enter 1.0, 1.0, 0.0 for a yellow light source
3. select "RETURN"
4. select the view marker of the current view to see the effect of having a yellow light source.

O.5.16 Specifying Shading Light Parameters

Purpose:
to control the position and brightness of individual light sources.

Menu Path:
MAIN (GEOMETRY (SHADING (LIGHT SOURCES)))

Description:
The "LIGHT SOURCES" menu item will allow you to specify the location and brightness

of up to 10 light sources. When you select the "LIGHT SOURCES" menu item, you will be prompted as follows:

```
enter number of light sources:
enter value(s) using dials and/or keyboard:
enter light red, green, blue fractions:
enter value(s) using dials and/or keyboard:
ENTER LIGHT NUMBER, LOCATION, AND BRIGHTNESS:
_       1,   0.00,   0.00,   0.00,  10.00
```

NOTE: The actual numbers shown in the input area may vary.

The first value in the input area is the light source number. If the number is increased, ACSYNT/VPI will automatic increase the number of light sources used for shading.

The next three numbers represent the light source location in the global coordinate system. The origin of the global coordinate system is at the nose of the aircraft. The positive x-axis points down the length of the fuselage and the positive y-axis points to the left when looking at the aircraft head-on.

The fifth number in the input area defines the brightness of the light source and can range from 0.0 to 50.0.

As with all numeric input, you can either type or dial in the desired values for the light source number, location, and brightness.

Example:

The following list contains an example of the steps necessary to to produce a shaded image with three colored light sources.

1. select the "SUN MODEL" menu item until you receive the message "SUN MODEL NOW INACTIVE."
2. select the "EYE FLAG" menu item until you receive the message "LIGHT SOURCE(S) NOW FIXED AT SPECIFIED COORDINATES".
3. select the "SURFACE CHAR." menu item and enter the following values:

   ```
   0.13, 26.00
   ```

4. select "RETURN" from the SURF. CHR. menu:
5. select the "LIGHT COLOR" menu item and enter the following values for a yellow light source:

   ```
   1.0, 1.0, 0.0
   ```

6. select "RETURN":
7. select the "LIGHT SOURCES" menu item and enter the values for light source number 1 as given below:

   ```
   1, 26.7, 15.0, 11.7, 21.0
   ```

8. select "RETURN" to store the values:

9. select the "LIGHT SOURCES" menu item and enter the values for light source number 2 as given below:

```
2, 6.12, 0.61, 11.25, 25.4
```

10. select "RETURN" to store the values:

11. select the "LIGHT SOURCES" menu item and enter the values for light source number 3 as given below:

```
3, 38.5, -5.0, 6.56, 15.0
```

12. select "RETURN" to store the values:

13. select the view marker of the view to contain the shaded image

Once the shaded image has been generated, you should see three yellow asterisks which represent the three light sources.

If you would like, you can adjust any of the shading parameters such as the surface characteristics and then reshade the aircraft by reselecting the view marker.

To drop the shaded image, select the "RETURN" menu item from the SHADE menu.

O.5.17 Resetting the Shading Parameters to Default Values

Purpose:
to set the shading parameters back to default values.

Menu Path:
MAIN (GEOMETRY (SHADING (RESET SHADING)))

Description:
When you select the "RESET SHADING" menu item from the SHADING menu, you will receive the following message:

```
enter surface reflectivity and shininess:
shading ...
enter value(s) using dials and/or keyboard:
enter surface reflectivity and shininess:
SHADING PARAMETERS RESET TO DEFAULT VALUES

_
```

The "RESET SHADING" menu item resets all of the shading parameters to the default values listed below:

ambient light fraction	--> 0.3
surface characteristics	--> 0.10, 30.00
sun model	--> active
eye flag	--> on
number of lights	--> 1

light 1 parameters --> 0.0, 0.0, 0.0, 10.0
light color RGB values --> 1.0, 1.0, 1.0

For more information on the above items, see the following sections of this guide:

- "Change the Shading Ambient Light Fraction" on page 290
- "Changing the Shading Surface Characteristics" on page 290
- "Switching Sun Model Shading On and Off" on page 291
- "Switching the Light Source Eye Flag On and Off" on page 292
- "Controlling the Number of Shading Light Sources" on page 293
- "Specifying the Color of All Shading Light Sources" on page 293
- "Specifying Shading Light Parameters" on page 294

O.5.18 Displaying the Geometry in Multiple Windows

Purpose:
to display the aircraft's geometry in four views.

Menu Path:
MAIN (GEOMETRY (WINDOW (MULTI WINDOW)))

Description:
Selecting the "MULTI WINDOW" menu item causes four views of the aircraft's geometry to be displayed. Only one of the views is considered to be the "current view". The current view can be changed by selecting one of the view markers located in the bottom left corner of each view. The current view is the only view that will be transformed if you turn one of the first seven valuator dials (see "Performing Three Dimensional Viewing Transformations" on page 285).

Example:

1. select the "MULTI WINDOW" menu item from the WINDOW menu
2. turn valuator dial 1 (you should see the current view change.)
3. select another view to be the current view
4. turn valuator dial 1 (the new current view should change.)

O.5.19 Displaying the Geometry in a Single View

Purpose:
to change the window layout from multiple views to a single view.

Menu Path:
MAIN (GEOMETRY (WINDOW (SINGLE WINDOW)))

Description:
The "SINGLE WINDOW" menu item will prompt you to select one of the four multiple views to be displayed as a single view as shown below:

```
enter light number, location, and brightness:
enter value(s) using dials and/or keyboard:
enter light red, green, blue fractions:
pick menu item:
SELECT VIEW:
_
```

To select a view, pick the marker found in the view's lower left corner. Once a view has been selected, the other three views will disappear.

Example:

1. use the "MULTI WINDOW" menu item to display multiple views of the aircraft's geometry (see "Displaying the Geometry in Multiple Windows" on page 297).
2. select the "SINGLE WINDOW" menu item and select the view of your choice to switch the display to a single window format.

O.5.20 Storing Window Parameters

Purpose:
to store a window's current transformation parameters for later retrieval.

Menu Path:
MAIN (GEOMETRY (WINDOW (STORE)))

Description:
When you select the "STORE" menu item under the WINDOW menu, the following prompt will be displayed:

```
enter value(s) using dials and/or keyboard:
enter light red, green, blue fractions:
pick menu item:
select view:
SELECT VIEW TO STORE:
_
```

ACSYNT/VPI will store the current viewing transformations for any view you choose. The view can later be reset through the use of the "RESTORE" menu item (see "Restoring Window Parameters"). The view transformation values for a particular view are stored within the STORE menu by selecting the view's view marker.

Example:
Please see the example in "Restoring Window Parameters".

O.5.21 Restoring Window Parameters

Purpose:
to reset the view transformation values for a selected view.

Menu Path:
MAIN (GEOMETRY (WINDOW (RESTORE)))

Description:
With the menu item "RESTORE", you can reset the transformation values of a view saved using "STORE" (see "Storing Window Parameters" on page 298). The "RESTORE" submenu will prompt you to select a view as follows:

```
enter light red, green, blue fractions:
pick menu item:
select view:
select view to store:
SELECT VIEW TO RESTORE:

_
```

After selecting the view of you choice ACSYNT/VPI will restore the view's transformation values.

Example:

1. enter the STORE submenu by selecting the "STORE" menu item from the WINDOW menu
2. transform the current view using valuator dial 1
3. select the current view's marker to save the view's current transformation values
4. exit the STORE menu by selecting the "RETURN" menu item
5. transform the current view using valuator dial 1
6. select the "RESTORE" menu item
7. select the same view marker you selected in step 3 to restore the view's transformation values

O.5.22 Switching Between Parallel and Perspective Projections

Purpose:
allows the user to change a view's projection type between parallel and perspective.

Menu Path:
MAIN (GEOMETRY (WINDOW (PROJECTION)))

Description:
When you select the "PROJECTION" menu item you will be prompted to select a view as follows:

```
pick menu item:
select view:
select view to store:
select view to restore:
SELECT VIEW:
_
```

While within the "PROJECTION" menu item, you can toggle a view between parallel and perspective projections by selecting the view's view marker. Each time you select a view you will receive either the message "PROJECTION TYPE CHANGED TO PARALLEL ..." or "PROJECTION TYPE CHANGED TO PERSPECTIVE ...".

NOTE: Shading should only be performed on parallel views.

Example:

1. enter the PROJECTION submenu by selecting the "PROJECTION" menu item from the WINDOW menu
2. select a view marker to see the effect of both parallel and perspective projections.

O.5.23 Producing Top, Front, Side, and Isometric Views

Purpose:
to change a view's transformation values to display either a top, front, side, or isometric view.

Menu Path:
MAIN (GEOMETRY (WINDOW (ISOMETRIC | TOP VIEW | FRONT VIEW | SIDE VIEW)))

Description:
Once you have selected either the "ISOMETRIC", "TOP VIEW", "FRONT VIEW", or "SIDE VIEW" menu items, you will be asked to select a view as shown below:

```
pick menu item:
select view:
select view to store:
select view to restore:
SELECT VIEW:
_
```

After selecting a view, ACSYNT/VPI will change the view's transformation values to reflect the menu item you selected.

Example:

1. select the "TOP VIEW" menu item from the WINDOW menu
2. pick the current view's view marker to make the view a top view
3. repeat the above steps for the "FRONT VIEW", "SIDE VIEW", and "ISOMETRIC" menu items.

O.5.24 Specifying View Rotation Values

Purpose:
to allow the user to enter the rotation values for *any* specified view

Menu Path:
MAIN (GEOMETRY (WINDOW (ROTATE)))

Description:
When the "ROTATE" menu item is selected from the WINDOW menu you will be prompted to enter view rotation values as shown below:

```
select view:
select view to store:
select view to restore:
select view:
ENTER VIEW ROTATION VALUES:
_  330.00,  30.00,  30.00
```

NOTE: The above values may vary.

To enter the current view's rotation, you can either type or dial in the desired values ranging from -360.0 to + 360.0 degrees. Three values are required; one each for rotation about the x, y, and z axes.

As you change the rotation values ACSYNT/VPI will update the current view to reflect the change. To change which view is the current view, select the view marker of your choice.

Example:

1. select the "ROTATE" menu item from the WINDOW menu
2. turn any one of the first three valuators to see the effect of specifying view rotation values
3. enter the rotation values "0.0, 0.0, 0.0" to produce a side view of the aircraft.

O.5.25 Specifying View Scale Factors

Purpose:

to allow the user to change a view's scale factor.

Menu Path:

MAIN (GEOMETRY (WINDOW (SCALE)))

Description:

When the "SCALE" menu item is selected, the current view scale factor will be displayed in the input area as shown below:

```
select view to store:
select view to restore:
select view:
enter view rotation values:
ENTER VIEW SCALE FACTOR:
_    1.00
```

NOTE: The above value may vary.

To change the scale, either type or dial in the value desired between 0.1 and 5.0.

If multiple views of the aircraft are displayed, you can change the current view by picking a view marker. Changing the current view causes the input area to be updated to reflect the scale of the selected view.

Example:

1. select the "SCALE" menu item from the WINDOW menu
2. turn valuator dial 1 to see the effect of changing a view's scale factor.

O.5.26 Changing Workstation Colors

Purpose:
to allow the user to modify the workstation's color table.

Menu Path:
any menu except the SET COLORS menu.

Description:
When the "COLORS" menu item is selected, the SET COLORS menu will be displayed and you will be prompted as follows:

```
select view:
enter view rotation values:
enter view scale factor:
enter value(s) using dials and/or keyboard:
ENTER COLOR NUMBER & RED, GREEN, BLUE FRACTIONS:
_       1,   1.00,   0.00,   0.00
```

NOTE: The above values may vary.

The first number in the input area represents a workstation color index from 1 to 15. The index can be changed by turning valuator dial 1 or by typing over the given value. The other three numbers represent the red, green, and blue (RGB) color fractions associated with the specified color index. The RGB values range from 0.0 to 1.0 and can by changed using one of the following methods:

- select one of the colors appearing in the SET COLORS menu
- turn valuator dials 2, 3, and 4
- enter new values from the keyboard (press ENTER).

In ACSYNT/VPI, color indices 1 through 15 are used as follows:

Index	*Use*
1	aircraft component color and highlighting color
2	aircraft component color
3	aircraft component color
4	aircraft component color
5	aircraft component color
6	aircraft component color
7	color of current message line
8	unused
9	unused
10	program title area background color
11	color of menu items and message lines 2-4
12	input area background color
13	menu and message area background color
14	geometry display area background color
15	border color

Example:

1. select the "COLORS" menu item at the bottom right corner of the ACSYNT/VPI screen
2. from the SET COLORS menu, enter in the following to change the display area background color to gray:

```
14, 0.5, 0.5, 0.5
```

3. select the menu item "SKY BLUE". to change the geometry area background to light blue
4. turn valuator dial 4 counter-clockwise until the forth input value (blue) becomes 0.0 to produce a background color of light green.

O.5.27 Exiting ACSYNT/VPI

Purpose:
to allow the user to end an ACSYNT/VPI modeling session.

Menu Path:
MAIN (EXIT)

Description:
To end an ACSYNT/VPI modeling session, select the "EXIT" menu item from the main menu. The "EXIT" item is located in the bottom right corner of the ACSYNT/VPI screen. In order to prevent you from accidentally exiting the program, ACSYNT/VPI will ask you to confirm the exit by displaying the CONFIRM menu. If you select the "DO NOT EXIT" menu item, the MAIN menu will reappear and the program will not end. If you select the "EXIT" menu item from the CONFIRM menu, the screen will clear and the program will terminate.

NOTE: You can only exit the program from the MAIN menu. To get back to the main menu from a submenu, select each submenu's "RETURN" menu item until the MAIN menu appears.

Example:

1. from the MAIN menu select the "EXIT" menu item
2. select "DO NOT EXIT" from the CONFIRM menu so that the MAIN menu reappears
3. again select the "EXIT" menu item from the MAIN menu
4. exit the program by selecting the CONFIRM menu item "EXIT"

NOTE: The screen should blank and you should receive the "R;" prompt on your text terminal which indicates that ACSYNT/VPI has ended.

The vita has been removed from the scanned document