THE USE OF OBJECT-ORIENTED TOOLS IN THE DEVELOPMENT OF A PILOT'S VISION SIMULATION PROGRAM TO AID IN THE CONCEPTUAL DESIGN OF AIRCRAFT

by

Kerry S. McClure

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

on partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE

in

Mechanical Engineering

APPROVED:

A. Myklebust, Chairman

J. H. Bøhn

W.H. Mason

November, 1993
Blacksburg, Virginia

THE USE OF OBJECT-ORIENTED TOOLS IN THE DEVELOPMENT OF A PILOT'S VISION SIMULATION PROGRAM TO AID IN THE CONCEPTUAL DESIGN OF AIRCRAFT

by

Kerry S. McClure

Arvid Myklebust, Chairman

Mechanical Engineering

(ABSTRACT)

This thesis discusses the research and development of a program to aid the aircraft designer with determining the pilot's visual acuity. The discussion involves the use of Object-Oriented programming, the use of a graphical user interface based upon the graphics standard PHIGS (**P**rogrammers **H**ierarchical **I**nteractive **G**raphics **S**ystem) and the integration of this Pilot's View Module with an existing aircraft CAD (**C**omputer **A**ided **D**esign) program known as ACSYNT. The result is a program that lends itself to reuse and easy modification and is device independent. The main purpose of the pilot's view module is to provide total vision plots for the pilot in accordance with the military standards as stated in the document 850B. These standards include visibility design goals for several types of aircraft as well as methods for calculating and presenting the vision plots. The integration of the pilots view module with ACSYNT affords the designer the ability to examine the trade-offs associated with a particular cockpit design and the performance of that design within one CAD program.

Abstract

Acknowledgments

I would like to thank Dr. Arvid Myklebust, Dr. Sankar Jayaram and all of the other people associated with the ACSYNT institute for their help and support as well as the offering of their knowledge to help in the completion of this project.

Specifically, I would like to thank Dr. Sankar Jayaram for being such an incredible teacher and sparking my interest in CAD and for guiding me into the ACSYNT program.

I would like to thank Dr. Arvid Myklebust for accepting me in the program and for his guidance as well. I would also like to thank the ACSYNT institute and its members for the financial support that allowed me to continue my education and obtain my Masters Degree.

I would like to thank all of the students I had a chance to work with. When I entered the program, I was welcomed as if I was part of a family. They all put up with my numerous questions and problems and continually offered me help if needed.

Lastly, I would like to thank my Grandmother and Grandfather Hale as well as my Mother and Father. The times haven't always been easy, but through it all, they constantly supported and encouraged me to be the best I could be. I would not be where I am now if it wasn't for them: Thank you!

I would like to dedicate this to my late Uncle, Dave Friday. He was always there for me and was a "father figure" when I needed him. He was one of the first people that sparked my interest in computers and also taught me that I could do anything I wanted, ...if I tried.

I will miss you my friend...

Table of Contents

List of Figures

1.0 Introduction

From the earliest of times, people have held the occupational title of "Engineer." They might not have had this title on their business cards, but when we look back at what they accomplished, they definitely deserve the title of engineer. What's most impressive is that over the years, these engineers had to use primitive tools to visualize their designs, from the earliest caveman who painted a vision of a wheel on the cave wall with paint made from the juice of wild berries to the Egyptian pyramid builders who used stone tablets to record pictures of their visions. With the advancement of technology, these tools of visualization have improved slowly over the years and in the early 1950's when computers became the tools of choice, the first, albeit primitive, computer aided design programs became available. These CAD programs have revolutionized the way engineers approach the problems they face.

Most engineers would say that the engineering design process can be broken down into three steps: conceptual design, preliminary design, and final design. Before the advent of CAD software, most of the design work would be done by hand, on paper. When a change in the design occurred, the entire drawing would need to be changed, consuming valuable time and money. The CAD approach to design has removed this step from the design process and has blended the other steps together so that they are not as distinct as they once

were. With the CAD systems available today, the entire process from conceptual to final design can be controlled by the computer. The initial sketches can be created on the computer and then refined over many design iterations until a final configuration is chosen. The final design can be rendered to reveal any surface flaws and the design information can also be exported to other CAD tools such as finite element analysis programs for further development. The computer can then be instructed to print out complete design blueprints or it can be used to program computer numerically controlled (CNC) milling machines to create the necessary parts. Because all of the pertinent data for a design is stored in a computer, many workers can have access to it at the same time, the time to manage the paperwork is reduced and the final design can be visualized in a photo-realistic rendering without ever actually physically constructing or manufacturung it.

1.1 Objectives

The main goal of this project was to create a method, algorithm and program that would utilize the existing ACSYNT aircraft data to determine the best location for the Design Eye according to the regulations set fourth in the military standards [MILI84]. The program was to be created utilizing a new object-oriented graphical user interface, based on the ISO graphical standard PHIGS, developed by Woyak [Woya92] [Woya93]. This object-oriented method

of coding allows the code to be reused and easily modified in the future. The program should also be integrated into the existing ACSYNT code so that the designer can quickly check the vision of the pilot interactively during the design process. The program should output the pilot's vision data in a format specified in MIL-STD-850B. This standard specifies that the visibility of the pilot is to be presented in three different formats. The first format is a rectangular plot representing the projection of a sphere. The second format is an equal area plot representing the Aitoff's equal area projection of a sphere. The third format is a single-point perspective projection representing the vision as the aircraft approaches for a landing. From the data presented in these three formats, the designer will determine if the aircraft meets the vision requirements.

All of these features should be incorporated in a format that is easy for the designer to use, that is computationally fast, and that is reasonably accurate.

1.2 Thesis Organization

The remainder of this thesis is sub-divided into the following sections:

- Literature Survey
- Object-Oriented Tools
- PHIGS
- PHIGS / Object-Oriented GUI
- B-Spline Surface Intersections
- Pilot's View Module
- Results
- Conclusions

Following the conclusions, there is the bibliography, appendices and the vita.

2.0 Literature Survey

2.1 Background

The CAD industry is a relatively new one, and although it was started in the aircraft industry, there are only a handful of examples of programs specifically for the conceptual development of aircraft. Of these, only a few are actually described in literature.

2.2 ACSYNT

Most of the work for this project was developed for the program known as ACSYNT. ACSYNT, which stands for **A**ir**C**raft **SYNT**hesis, was first developed at NASA-Ames Research Center in the early 1970's. ACSYNT is a complete, conceptual design program for aircraft. Aircraft performance in the areas of trajectory, geometry, aerodynamics, propulsion, stability, weights, economics and takeoff can be analyzed. Each area can be analyzed independently because ACSYNT is modular in its design. This modular design, in which each different type of analysis is handled by a structured group of routines, was chosen for its adaptability to optimization techniques.

In 1987, the CAD lab at Virginia Tech began work on creating a user-friendly, graphical interface to ACSYNT to allow the user to interactively, parametrically modify and see the changes to the aircraft structure as well as to view the output data in graphical format. This interface [Wamp88a],[Wamp88b] is based on the ISO graphical standard known as PHIGS. With the development of the graphical interface, ACSYNT has become a widely used and very powerful design tool in the aerospace engineering community. The work on the graphical interface for ACSYNT was continued by Jones [Jone91] and Marcaly [Marc91] who implemented B-spline geometry and by Gloudemans [Glou89] [Glou90] who investigated filleting and intersection features. Additional work by Jayaram [Jaya91] [Jaya92b] added dimensional geometric parameter extraction from B-spline models. Other functionality and object-oriented modules have been added by Kelly [Kell93], Rivera [Rive93], Schrock [Schr91] [Schr92], Steude [Steu93] and Uhorchak [Uhor93].

2.3 Commercial Programs

Outside the Virginia Tech CAD laboratory, several other companies have developed their own software for the development of aircraft. Larimer and Provost [Lari91] discuss the development of the Visibility Modeling Tool (VMT) that provides the designer with a CAD tool for evaluating the trade-offs

associated with different cockpit decisions and designs. This VMT differs from ACSYNT and the goals of this project in that it would fall into the preliminary or final design process and not the conceptual area. The VMT is capable of determining if the displays inside the cockpit can be read, what fonts are easily read, and it can compute retinal maps. Overall, it is a more complex and detailed analysis of the interior of the cockpit and how the pilot will interact with these surroundings. The major drawback with this program, is that it is a stand-alone tool that must be used along with other CAD programs. The cockpit must be designed separately in another package before the data can be imported into the VMT for analysis. The results of the changing eye location can then be seen, but if the designer wishes to change the geometry of the cockpit, a new model must be created and imported into the VMT.

Bolukbasi and Bertone [Boul90] discuss the development of a CAD program known as MACMAN (McDonnell Anthropometric Computerized Man Model). Its primary goal is to assist in the design of helicopter crew stations. Again, this program is a stand-alone program and is not integrated into a full-featured aircraft design program. The cockpit data must be entered by hand before it can be analyzed. This program differs from this project in that it incorporates a human body model. This human model is based on a twelve-segment kinematic model with 29 degrees of freedom. The program does allow the user to check not only the visual cone of the pilot, but also allows for the positioning of the control sticks, pedals, as well as door handles and latches, due

to the fact that the kinematic equations for a twelve-segment human body model are included. The program can also be used to check the egress and ingress of the crew as well as the strength requirements for lifting and placing components inside the aircraft. Again, this program is a much more complex and detailed design tool intended for the preliminary or final design stages and not conceptual design.

3.0 Object-Oriented Tools

One of the newest and most popular conceptual approaches for computer programming is called Object-Oriented Programming (OOP). It is a new way of thinking about data and how to manipulate it based on an entirely new level when compared to conventional programming techniques. The philosophy of OOP is based on the idea of bottom-up programming. The programmer is to emphasize the representation of data and not the algorithms. The idea is to develop a new *class* that specifies a new data form and the associated functions for this data, and then develop an *object* that describes a data structure constructed according to that *class* plan. A *class* is a generic structure containing data and functions while an *object* is a specific instance or variable of type *class*.

The most important features of OOP are [Prat91]:

- Data abstraction
- Encapsulation and data hiding
- Polymorphism
- Inheritance

To use these important features, the programmer develops classes to incorporate these together. Data abstraction is the step where the programmer determines exactly how the important information or data will be presented to the user. To present the information to the user, the programmer will create a class that can contain data types as well as functions that can operate on the data. In a

class, the data and functions can be classified as either private or public. Private members can only be accessed by member functions of that class. Public members, on the other hand, can be accessed by the rest of the program. Typically, the data associated with a certain class are declared as private and the functions that operate on this data are declared as public. The ability to hide certain data from the rest of the program is known as data hiding. Putting data and the associated functions that operate on the data together is known as encapsulation. When defining the functions, OOP allows the programmer to define several functions with the same name as long as they operate on different types of data in their argument lists. It is then up to the computer to determine what function to use depending on what data is operated on. This is called function overloading or polymorphism. Once a class has been developed, new classes can be derived from them retaining the data and functions associated with the old class and incorporating new data and functions as well. This practice is known as inheritance and is the major reason that OOP is known as a reusable coding method.

4.0 PHIGS

To create a graphical program that can be used on many different platforms, a graphical standard or portable interface tool kit must be used. There are several graphical standards that are commonly used in industry today. Motif is one such standard that runs in conjunction with X-windows. Motif is a very popular graphical interface standard that can be used on many different machines but it does have several drawbacks. It is a two-dimensional standard which can be restrictive in engineering work and it inherits the same functionality as X-windows. Motif does integrate easily with an object-oriented language such as C++, due to the fact that is somewhat object-oriented itself.

PHIGS is another earlier standard for computer graphics programming work. It is more applicable to engineering work because it is a true three-dimensional tool kit and it supports a much larger set of logical input devices than Motif. Some programmers use Motif for the graphical interface and PHIGS for actually displaying the model. This can be done but can also cause many problems due to the differences between the standards. As Woyak [Woya92] points out, it is better to build an interface solely using PHIGS because it provides the user with a system that possesses multi-platform existence without forgoing flexibility.

PHIGS is a high-level library (i.e., that it hides the device dependent details from the user) that contains over 400 functions. These functions allow the

programmer to develop a graphics image based on intuitive graphic primitives such as lines, points, fill-areas without dealing with the actual pixels and device dependent commands. The power of PHIGS is also apparent in that, because it is a true 3-D environment, different views of an object require only that different PHIGS views be set up. PHIGS handles the rest of the transformation calculations. PHIGS is also powerful in that the image that is sent to the screen is usually stored in a virtual central structure store. This storage allows for easy modification or editing of one part of the structure without recalculating the rest of it; this type of storage is known as a display list system. Display lists are appropriate for CAD applications that frequently edit only portions of a structure while simultaneously viewing it in different ways, but it is not appropriate for applications that require animation [Gask92].

PHIGS is appropriate for a wide variety of applications. One of the most important features of PHIGS is device independence because it makes applications written with PHIGS portable. There are penalties and drawbacks associated with this device independence. It can not be optimized to take advantage of all systems because the language must work on so many different platforms. This is where the device dependent implementations of PHIGS like graPHIGS for the IBM systems and PHIGS+ become important. These implementations incorporate features that take advantage of the systems upon which they run.

5.0 PHIGS / Object-Oriented GUI

An interface built on PHIGS was used as the base for this project. This interface was developed by Woyak [Woya92] [Woya93]. It is a complete set of tools based on the PHIGS graphics standard and it is also programmed in an OOP style. This interface framework was chosen for this program because of these two important reasons. Since it uses PHIGS, it is compatible with the existing code for ACSYNT, which makes data transfer much easier, and it retains the device independence of ACSYNT. It will be easier for the code to be upgraded and reused in the future because it is written in C++ to take advantage of the OOP style. A complete description and user's guide for the Graphical User Interface (GUI) toolkit can be found in [Woya92].

This toolkit is very similar to other popular interfaces such as Motif or Microsoft Windows. It based on a set or library of tools that can be assembled quickly and easily to create an attractive and functional interface. The tools consist of menus and windows and the items that can be associated with these menus and windows. These items include check boxes, labels, numbers, pushbuttons, radio buttons, sliders and frames. Due to the OOP style of programming, it is very easy for the user of this interface to assemble these items to form the complete interface. Once they are assembled, the interface manager class is called upon to do just that, manage the windows, menus and the events triggered by the user, and produce the proper responses.

Although this interface is relatively new and unproven in a commercial production environment, it has been used by several other graduate students from Virginia Tech in their graduate work, with much success. This includes the development of a new mission input and planning module for use with ACSYNT [Rive93], the development of a new class library for the creation of engineering graphs for use with or without ACSYNT [Uhor93], the developement of a rule-based fuselage design module for ACSYNT [Kell93], the developement of a graphical interface for an engine design code known as NNEP [Stue93] and the development of an interior design module for ACSYNT that would allow the designer to configure the seats within the cabin [Hasa93].

6.0 B-spline Surface Intersections

6.1 Background

To determine exactly what the pilot can see from inside the cockpit of an aircraft, some measure of visibility must be established. Some details of the aircraft structure are unimportant in the aircraft analysis due to the fact that ACSYNT, at the present stage of development, is a tool for the conceptual design step of the engineering process. These unimportant details include the interior of the cockpit as well as the canopy framing and aircraft windows. They are not included as part of the ACSYNT geometry at this time. Therefore, the limiting factor as to what the pilot can see is the blockage caused by the fuselage and other components such as the wings and canards. If computational time was not a determining factor in the design of the program, a ray tracing method could be used to completely determine the pilot's vision. Ray tracing is a very accurate method and would take into account all of the components if implemented properly, but it would also be very computationally time consuming. If the ACSYNT geometry was a solids-based system with boolean operations available, then the cockpit could be subtracted from the fuselage and the viewing system could be set up with the design eye in the required location as shown in Figure 1. The geometry could then be shaded and the designer would be presented with a representation of what the pilot could see.

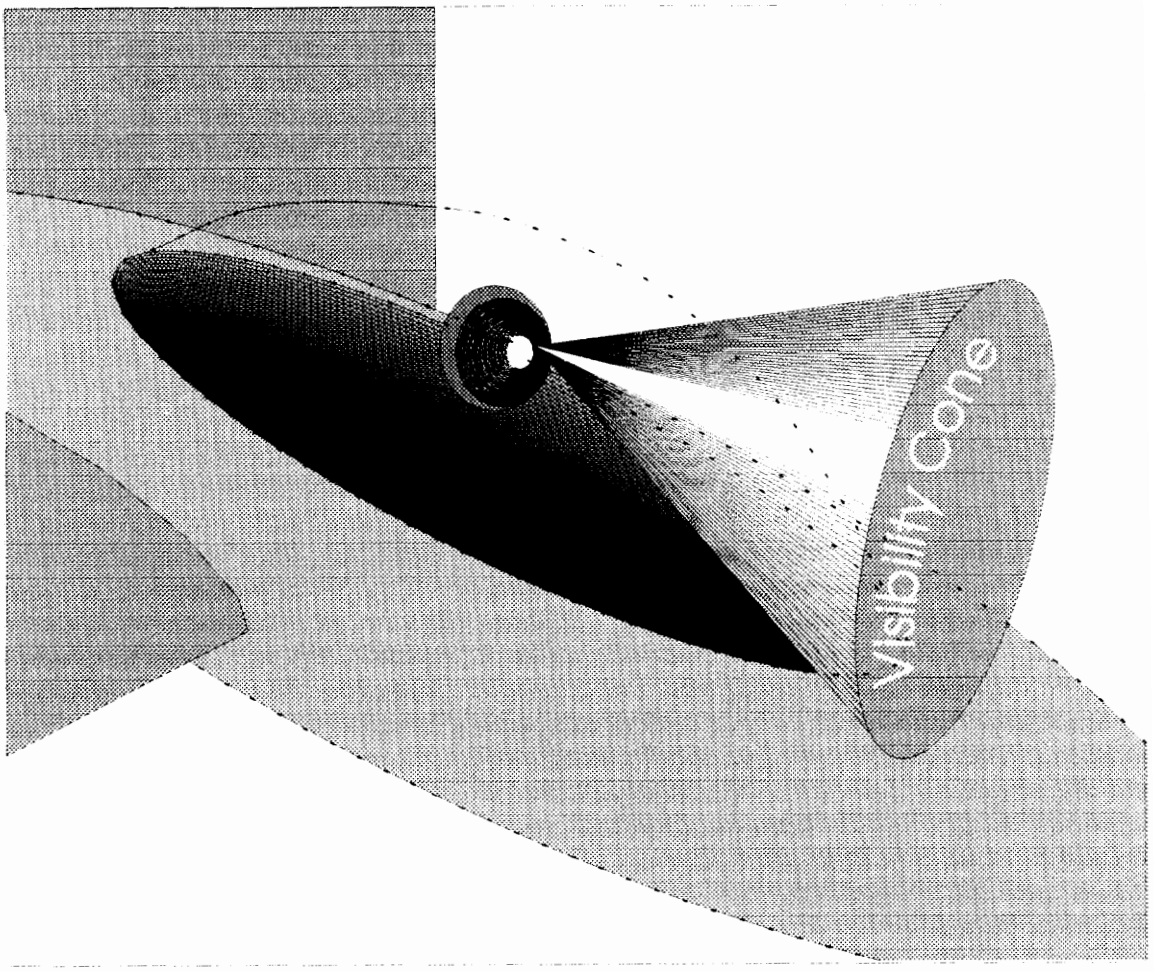


Figure 1 - Boolean model

Another way of representing the pilot's vision would be possible if transparency was available. Neither PHIGS or PHIGS+ (an extension to PHIGS) supports transparency. However some extensions of PHIGS do support shading although it is dependent on the hardware used. If transparency was available, the viewing system could again be set up with the design eye in the proper location and the model could be shaded with transparency set on for the canopy. This would provide the designers with a view as if they themselves were sitting in the cockpit. A similar result can be obtained using only shading. The canopy component can be left as a wire frame component and the remaining model shaded. If the PHIGS view system is set up with the view reference point at the design eye location, the view up vector in the vertical direction and the view plane normal in the direction of the pilot's sight, the designer will be presented with a good representation of what the pilot can see. The viewing direction could be changed by changing the view plane normal to represent the pilot's head rotating. This method would provide the designer with a sensation of actually being in the cockpit, however, it would be very tough to translate this visibility information into the total vision plots as specified in the standards. This translation problem is due to the fact that although the image on the screen appears to be two-dimensional, the data in the computer is actually three dimensional, and as stated earlier, to extract the important values from all of the data would be computationally slow.

Because these methods were not practical or possible, and due to the limitations imposed by computational time and the existing structure of ACSYNT,

it was decided that in order to determine how the fuselage impairs the vision of the pilot, the intersection of the fuselage with the canopy must be computed. This intersection will be the limiting factor as long as the design eye location is relatively low in the cockpit. Figure 2 illustrates this concept. Whenever the design eye location is low in the cockpit, the fuselage is the only component that blocks the vision, however as the design eye location is raised, the pilot's vision will increase subject to other components, in addition to the fuselage, obstructing the vision of the pilot. These other components include the wings, canards and nose of the plane. To reduce computational time, only the most important of these components can be checked for their obstruction. Hence, the method used in this project was to find the intersection of the canopy and fuselage and use this as a base for the vision computations. The ACSYNT geometry is represented mathematical and visually as a set of B-spline surfaces, one for each component. To find the intersection line created by several components, a B-spline intersection routine must be used.

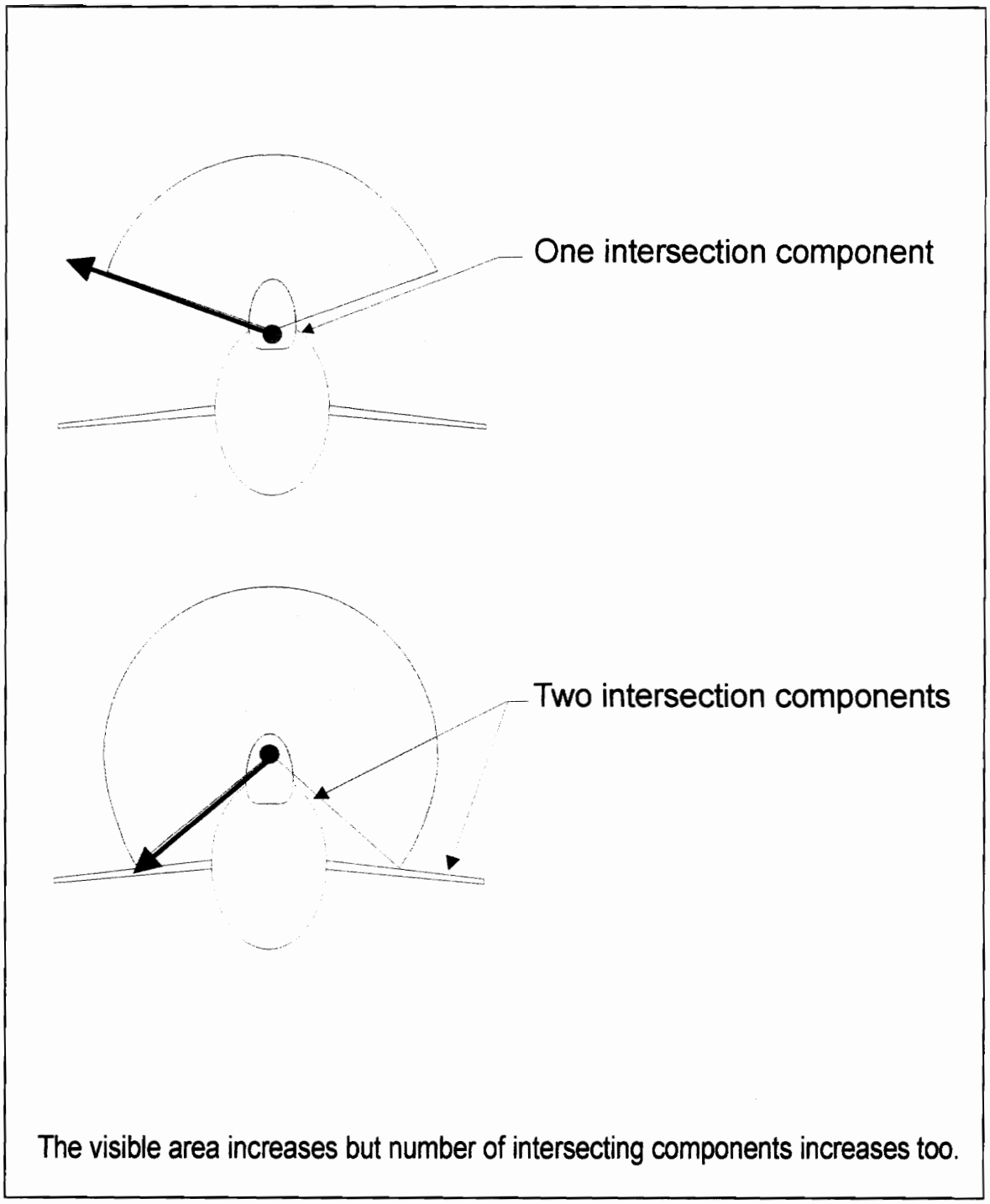


Figure 2 - Design eye location

6.2 Intersection Routine

For computing the intersection between two B-spline surfaces, there are several proven methods. These include hunting algorithms developed by Chen and Ozsoy [Chen86] and "Divide and Conquer" methods developed by Peng [Peng84] and Lasser [Lass86]. It is this "Divide and Conquer" method that was researched and implemented into ACSYNT by Jones [Jone91]. He combined the algorithms of both Peng and Lasser to form a more robust intersection method for non-uniform bi-cubic B-Spline surfaces.

The subdivision algorithm can be divided into these eight steps as shown by [Jone91].

1. Build bounding boxes for each separate surface (component) by determining the minimum and maximum X, Y, Z Cartesian coordinates for the surfaces.
2. Compare the two bounding boxes for a possible intersection. If check is successful, go to step three, otherwise test a different set of bounding boxes.
3. Any two components that pass the component bounding box test are compared on a patch by patch basis. A bounding box check is performed for each patch on one component against every patch on the other component.

4. The larger of the two patches which pass the bounding box check is split into four sub-patches. The sub-patches inherit the controversy (possibility of intersection) from their parent patch.
5. New bounding box checks are performed on each of the sub-patches against their parent's adversary.
6. Steps four and five are repeated until both sub-patches can be approximated by a plane. At this time intersection between any two sub-patches can be determined by calculating the intersection between two planes.
7. Steps three, four, five and six are repeated until all possible intersection checks have been exhausted between the components.
8. The intersection data is sorted to form one of more continuous curves of intersection.

Jones coded this intersection algorithm and it was initially included in a separate B-spline module. This module was then integrated into the existing ACSYNT code so that the geometry displayed on the screen was the B-spline form and not the original Hermite form. An example of an intersection curve can be seen in Figure 3. After studying the existing routines that compute the intersection curves, it was decided that completly new intersection routines were not necessary for this project. The existing routines were designed to accept input

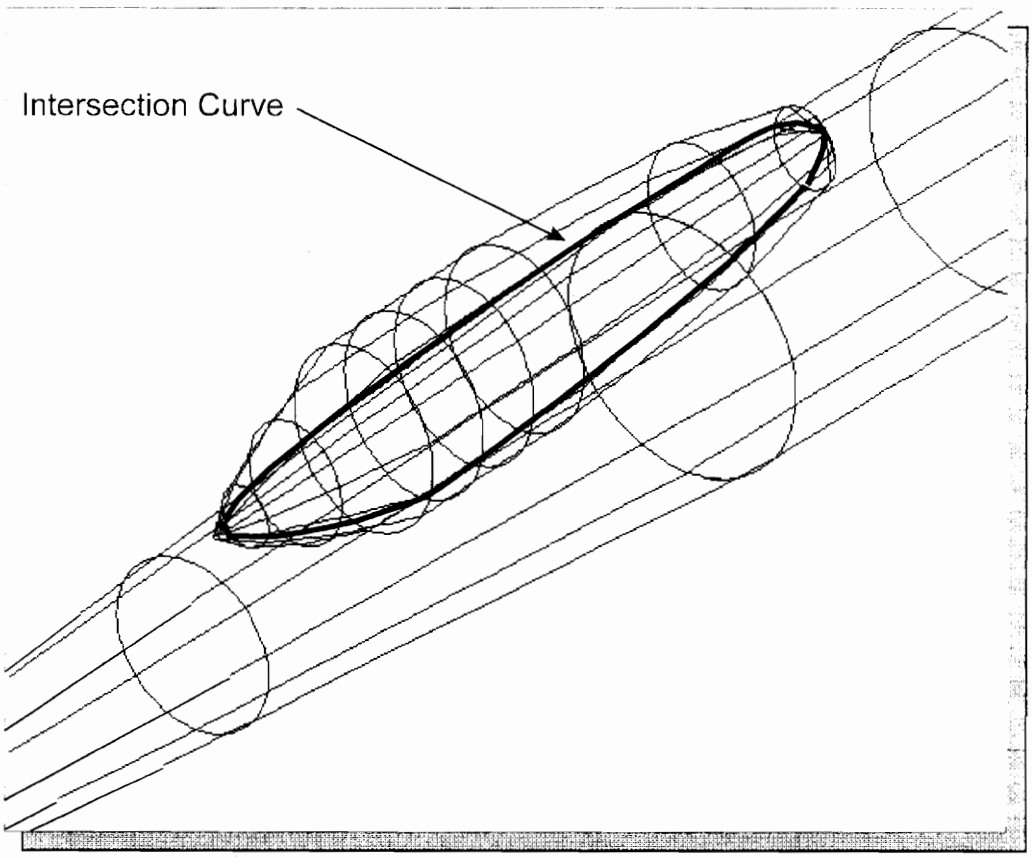


Figure 3 - Sample intersection curve

from the user, through the mouse, for exactly which components to intersect. They were also designed to only intersect two components at a time. For the new application, the routines had to be modified slightly so that they could handle more than two components as well as run independently of ACSYNT. The modified routines would be called from within the pilots view module and the intersection curve data and messages would be sent back to the pilots view module, not the B-spline module.

7.0 Pilot's View Module

7.1 Interface Construction

As stated earlier, one of the goals of this project was to use a set of tools developed by Woyak [Woya92] based on OOP in order to allow the code to be reused and modified more easily in the future. The first step in developing the Pilot's View Module was learning how to use this set of tools to develop a graphical user interface. This interface would be used to display the information to the designer in a pleasing manner. The interface should also retain some of the "look and feel" of the existing ACSYNT code. Because the computer display screen is limited in it's size, the interface must be designed so that the most important information utilizes the most available space. For this interface, the vision plots and the aircraft geometry have the highest priority so most of the screen is devoted to displaying these images. Therefore, it was decided that the most important functions and images should be displayed immediately upon entering the interface and that the other functions and plots should be reached from pop-up menus and windows.

The interface was programmed and the screen that is first encountered by the designer can be seen in Figure 4. On the right side of the screen are the push-buttons and radio-buttons that control the module. On the bottom of the screen, there are number boxes that display the location as well as allow for

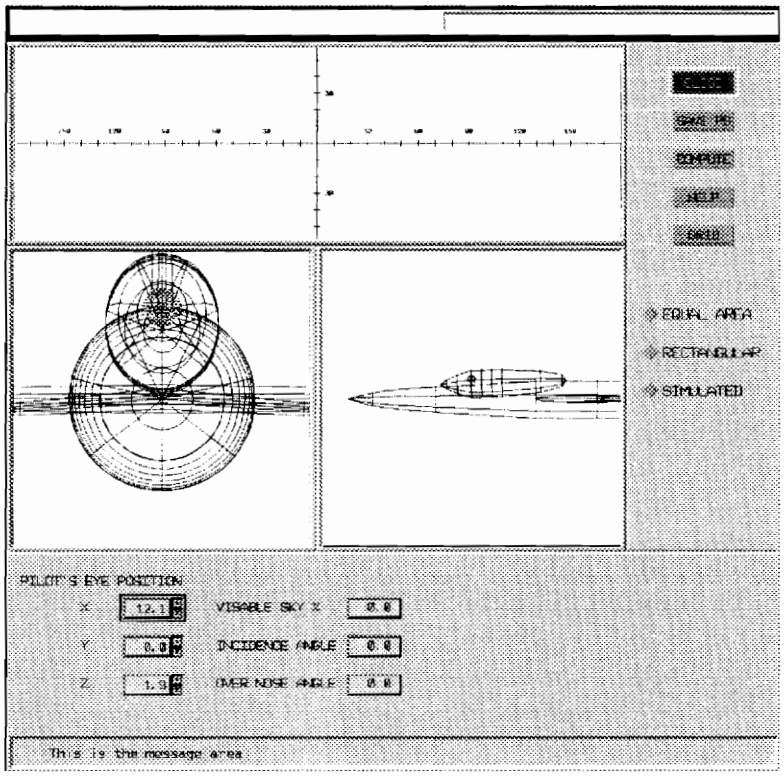


Figure 4 - Pilot's view interface

movement of the design eye location. To the right of these number boxes are several other number boxes that display other important parameters. The major portion of the screen is devoted to the display of the aircraft and the visibility plot. The two views of the aircraft show the design eye, represented by a red sphere, as well as the canopy-fuselage intersection line when computed. The visibility plot above displays the 2-D representation of exactly what the pilot can or cannot see. The graph can be displayed in rectangular or Aitoff's equal area representation. The gray fill area represents what the pilot cannot see and the black area represents what the pilot can potentially see. A grid can also be toggled on or off by the designer to help the reading of the plot.

7.2 Intersection Routine Implementation

After the GUI was completed, the routines that would actually compute the intersection data and plot the results needed to be completed. After examining the existing B-spline intersection routine coded by Jones [Jone91], it was determined that with a little modification, these existing routines would be able to handle more than one intersection component at a time. It was only a matter of calling these routines at the correct time with the correct data and then receiving the intersection data in the new code. A flow chart of exactly how the new code interacts with the old code is shown in Figure 5. The intersection routines lie in a

loop that cycles through the chosen components, checking for an intersection with the canopy. If an intersection is found, the routines calculate the x,y,z values of the line of intersection. This continues until the list of components is complete. The intersection data, in the form of the x,y and z coordinates of the line, is then transferred to the transformation routines.

7.3 Data Transformation

Once the intersection data in the form of x,y, and z coordinates is returned to the Pilot's View Module, it needs to be transformed into 2-D data for plotting. The plotting requires that the x,y and z coordinates be first transformed into R, ϕ and λ coordinates, and then projected into 2-D x and y plotting coordinates as shown in Figure 6. The first set of transformations can be calculated as follows:

$$R = \sqrt{(hx - x)^2 + (hy - y)^2 + (hz - z)^2}$$

$$\phi = \arccos((x - hx)/\sqrt{(hx - x)^2 + (hy - y)^2})$$

$$\lambda = \arcsin((z - hz)/R)$$

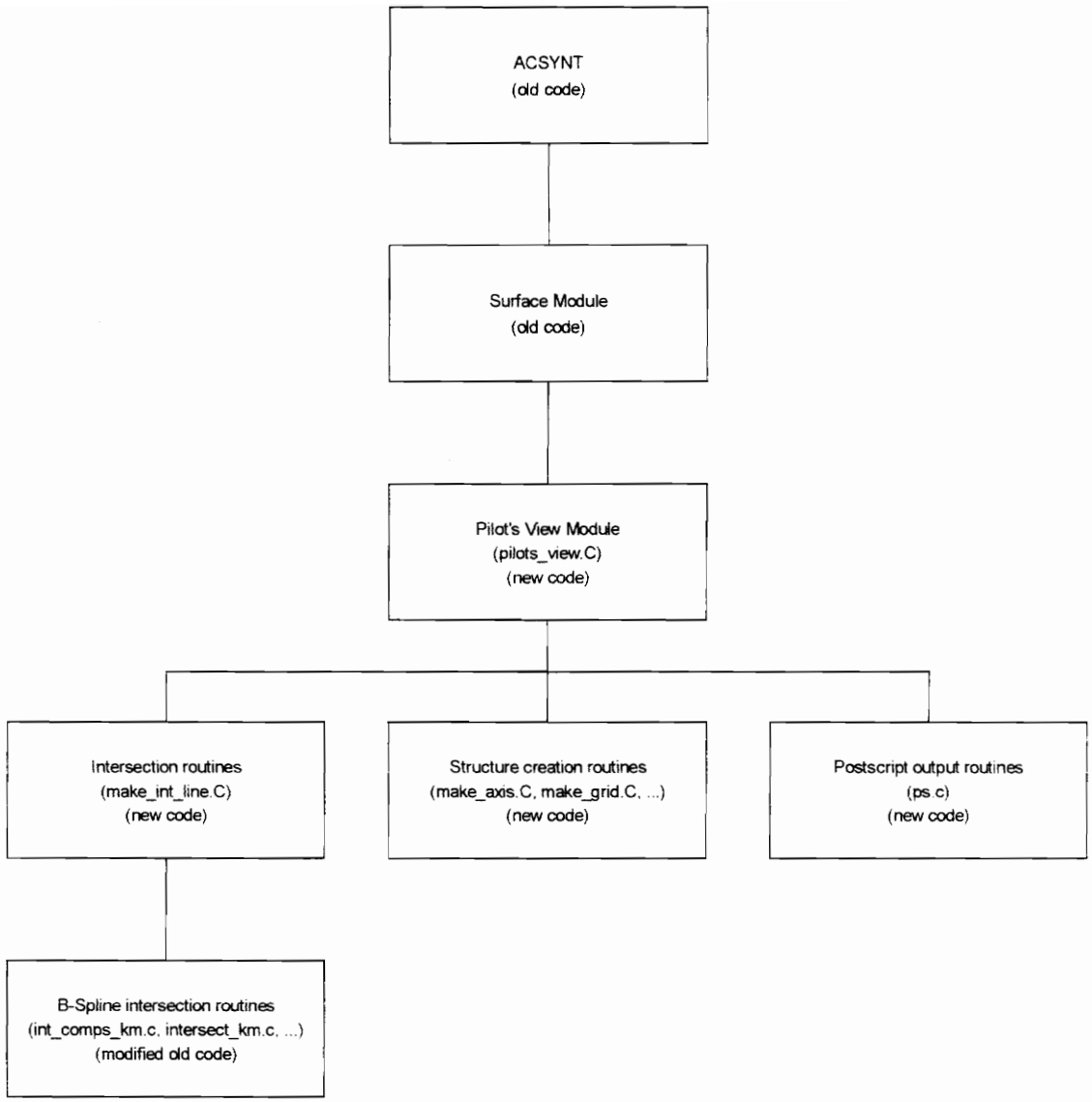


Figure 5 - Pilot's View/ACSYNT integration chart

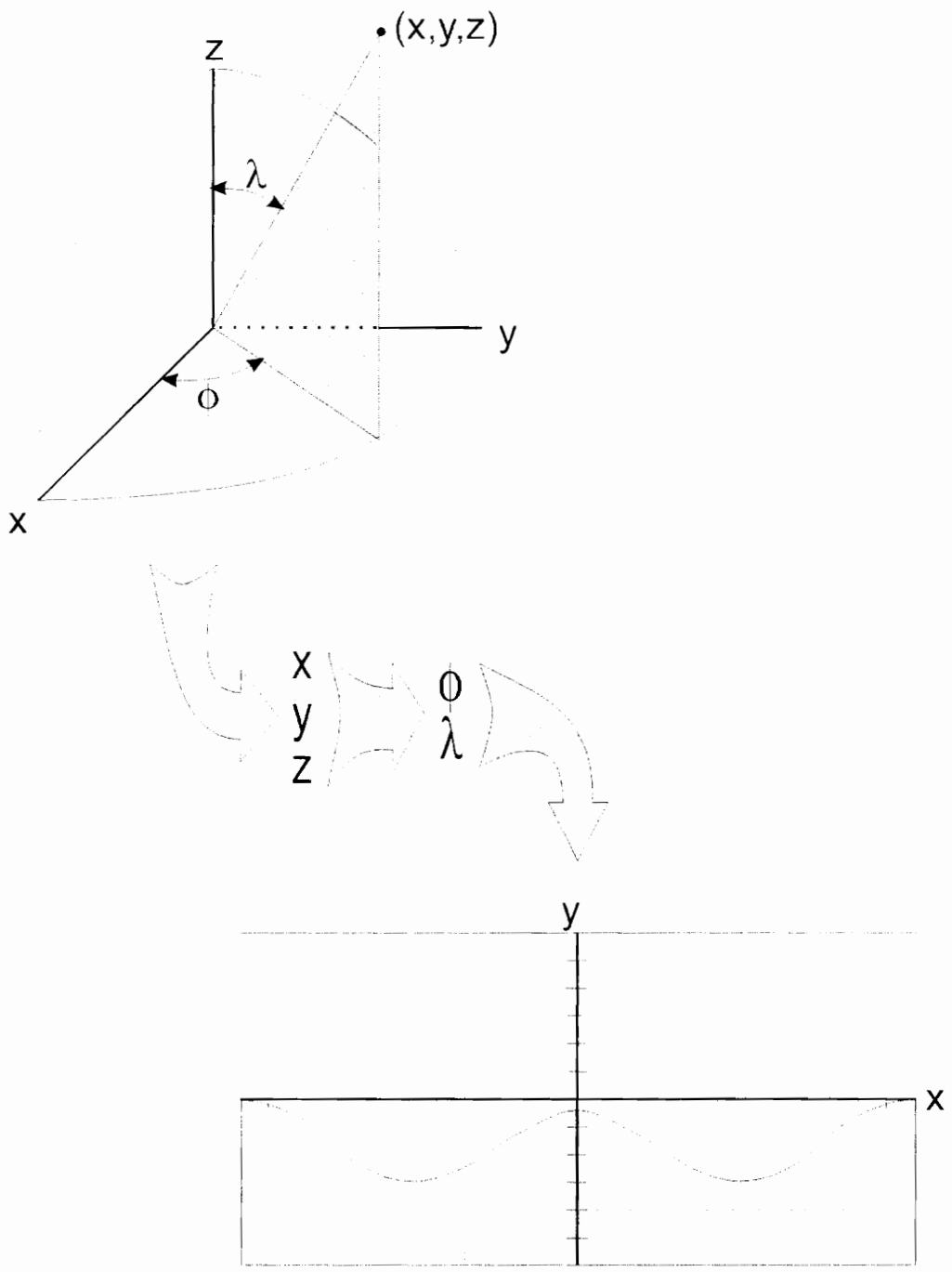


Figure 6 - Coordinate transformations

In these equations, the values hx, hy and hz are respectively, the x, y and z values of the design eye location, and the x, y and z values are the points of the intersection line. The value R is simply the distance from the design eye location to the intersection line.

The rectangular projection represents a sphere by a rectangle with the width equal to three times the height. For this projection, the ϕ and λ coordinates from the above equation were simply transformed to x and y plotting coordinates based on the window projection coordinates by the equations:

$$x = .5 - [(\pi - \phi) \times \frac{180}{\pi} \times 0.002777]$$

$$y = .5 + (\lambda \times \frac{180}{\pi} \times 0.002777)$$

Where x and y are the normalized coordinates of the window on which the graph is drawn and 0.002777 is simply a scale factor dependent on the size of the graph window.

The Aitoff's equal area projection represents a sphere as an ellipse with the semi-major axis twice the length of the semi-minor axis, and with its parallels represented as curved lines. This projection required a slightly different transformation from the R, ϕ and λ coordinates to the x and y plotting coordinates. The coordinates were transformed by the following equations from [Pear84]:

$$x = 2 \times R \times S\sqrt{2(1 - \cos\phi \cos\lambda/2)}\ \sin[\tan^{-1}(\frac{\sin\lambda/2}{\tan\phi})]$$

$$y = R \times S\sqrt{2(1 - \cos\phi\cos\lambda/2)}\ \cos[\tan^{-1}(\frac{\sin\lambda/2}{\tan\phi})]$$

These equations, where R is the radius of the sphere and S is a scale factor, produce a projection as shown in Figure 7. The resulting x and y values were then scaled in a similar fashion as the rectangular coordinates in order to produce coordinates that are based on the windows projection coordinates.

To create the total vision plots, the push button labeled **COMPUTE** must be selected. A pop-up menu appears requesting a list of components that will be checked for intersection with the canopy. When the selection from the list is complete, the intersection routines are called as explained and the total vision plot is created. The initial plot is displayed in rectangular form, but when the equal area button is selected, the Aitoff's projection is displayed.

The push button labeled **GRID** can be selected to provide a grid to aid the designer in reading and interpreting the total vision plots. The grid is available for both the rectangular and equal area plots. The grid spacing is based on ten (10) degree intervals. When the grid is toggled on, and the type of plot is changed, the grid automatically switches to the proper type.

If the radio-button labeled **SIMULATED** is selected, a new window appears containing a simulated landing approach vision plot. The purpose of this plot is to simulate the view of the pilot on approach to land. The horizon line is represented by the horizontal green line. A runway, sized according to the military

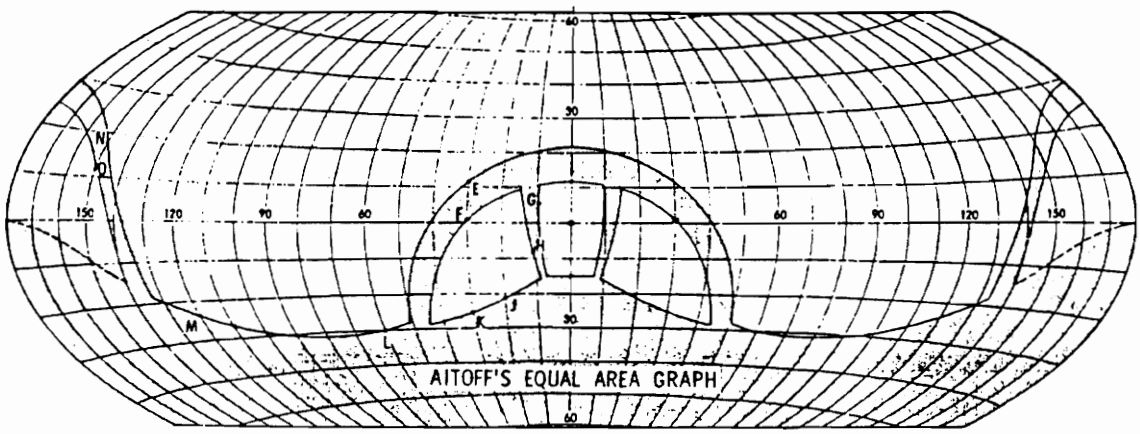


Figure 7 - Sample Equal Area Projection

standards, is also represented as a green trapezoid. The other lines on the screen are the wire-frame lines of the aircraft and canopy as well as the intersection line represented in red. This view is created by setting up a PHIGS view to emulate as closely as possible, a human eye. As required in the standards, several different glide path angle approaches can be simulated by the pilots view module.

The push button labeled **HELP** is included on the menu system but is not functional at this time. When the ACSYNT code is re-written, it is anticipated that an on-line help system similar to that in the Windows environment might be included, therefore a push button was included to make the future implementation easier.

7.4 PostScript Output

The ability of the designer to develop 2-D plots on the screen is very nice, but when a design eye position is chosen, there must be some provision for the designer to send the plots to a printer. This ability exists in the graphing module of ACSYNT for the aerodynamic, economic and other graphs. The most common and transportable format for printing is the PostScript language for describing objects to a printer. It is possible to use the lowest level PostScript commands and develop a file for output to a printer. This is how the existing printing options

in ACSYNT work. However, a new PostScript API (Application Programming Interface) was developed by Schrock [Schr93]. The users guide for this API can be found in the appendix A-3. This PostScript API is a set of C or FORTRAN callable subroutines which allow a PostScript file to be written effortlessly across applications. It was decided that the use of this API rather than the existing output routines in ACSYNT, would produce a program that was more reusable and modifiable in the future as well as reduce the time necessary to create the PostScript output module of the program.

The PostScript API does not take a "snapshot" of the screen and send it to a file, nor does it traverse the PHIGS structures and develop the file. Instead it is necessary to open a PostScript file and then write or draw the structures to this file just as they are drawn on the screen. Once all of the lines, fill areas and text are drawn to the file, the file is closed and is then ready to be printed. A example of the postscript output is shown in Figure 8. The example output represents the results of the calculations for a F16 model with the design eye located as indicated. Both the rectangular and equal area plots are created along with other key parameters.

The push button labeled **SAVE PS** creates this PostScript file. When selected, a pop-up menu appears requesting a filename for the output file. After the filename is entered, (maximum length of eight (8) characters), the file is opened, the graphs are written, and then the file is closed. To print the file, the user can use the appropriate commands from a UNIX window.

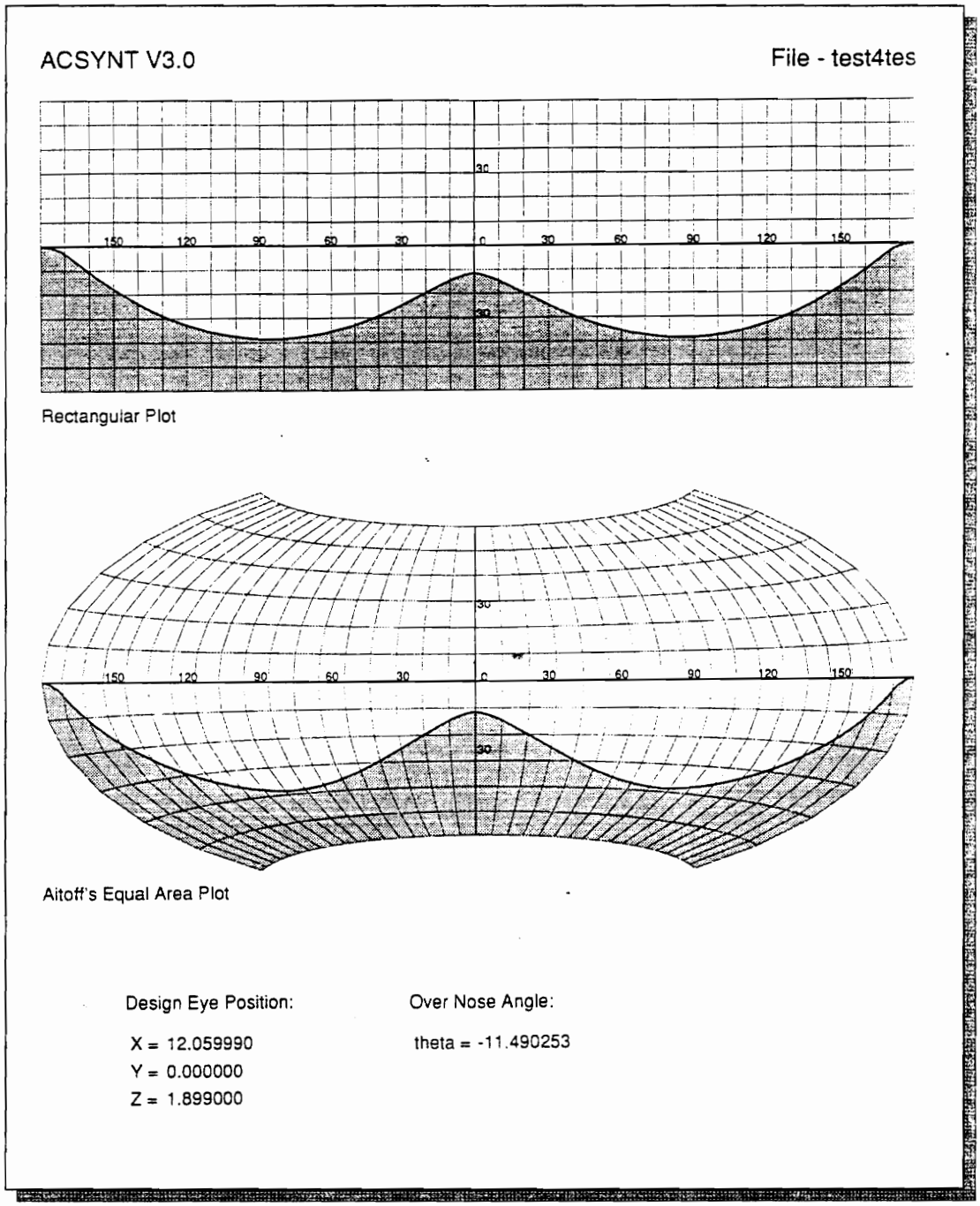


Figure 8 - Sample PostScript Output

7.5 Code Organization

The code for the pilot's view module can be separated into three distinct groups. The first set of routines fall into the GUI construction group. The second set of routines fall into the B-spline surface intersection group. The last set of routines fall into the data transformation group. The purpose of the GUI routines is to create the graphical interface that the designer will use to control the pilot's view module. These routines set up the windows, menu buttons and all of the structures to be displayed. These routines are centered around the interface tool kit developed by Woyak [Woya92] and are coded in C++. The intersection routines, which were originally part of the surface module, find the intersection of the canopy with the fuselage components and then send this data back to the transformation routines. These routines were originally coded in C by Jones [Jone91] but have been modified to work independently of the existing ACSYNT GUI and to interface with the new pilot's view GUI. The transformation routines take this data and transform it into coordinates that can be plotted in the interface windows as either a rectangular or equal area plot. These routines are written C++ and interface with the GUI routines to display the data. A complete list of the functional descriptions of these routines can be found in Appendix D.

8.0 Results

The Pilot's View Module was tested against several models to make sure that the intersection routines were robust and to make sure that the resulting total vision plots were correct and accurate.

The first test case was a simple cylinder-sphere intersection. This model can be seen in Figure 9. The intersection curve produced within the existing B-spline module of ACSYNT, can be seen in Figure 10 and can also be easily computed by hand. The total vision plots for this model are shown in Figures 11 and 12. It is evident from these figures and from the actual numerical results that the total vision plots are correct for the given design eye location.

A second and more complicated test case was based on the F-16 fighter model shown in Figure 13. This model was loaded into ACSYNT and the intersection curve was calculated as shown in Figure 14. The total vision plots for the respective design eye location are shown in Figures 15 and 16. Although this model is more complicated to check by hand, a few checks of the numerical results will substantiate the accuracy of the plots.

These two cases demonstrate the accuracy of the pilots view module to compute a total vision plot based on the intersection of the canopy with the fuselage, however it must be re-stated that in order for the module to work at this time, an accurate canopy component must be included in the ACSYNT geometry.

At the present time, several of the most important components have been incorporated into the total vision plots. As shown earlier, the intersection line between the canopy and fuselage components is a good approximation. However, other components will reduce the visibility of the pilot depending upon the location of the design eye. So far, the wings and the horizontal tail surfaces have been included in the visibility calculations. Due to the way that the geometric parameters of these components are stored (i.e., as B-Spline surfaces) the component extremes in terms of x,y and z values are difficult to compute. For calculation purposes, these components are approximated as flat planes so as to reduce computational time. Figures 15 and 16 show how the wingtips reduce the visibility of the pilot for this particular F16 model and design eye location. The wingtips are represented as hatched fill areas. As the design eye location is moved rearward, the wingtip fill areas would move towards the 90° position on the total vision plots.

As mentioned, the pilots view module has the functionality to produce a simulated view to represent the landing approach plots as specified in the military standards. The landing approach plot is a one-point perspective projection of the forward visibility of the pilot. To represent this in the pilots view module, a PHIGS view was set up with the view reference point at the location of the design eye. The view up vector was directed in the positive Z direction, up for the pilot, and the view normal vector was positioned in the direction of sight. The view plane was set at 10 inches from the design eye location as per the standards. With the

airplane model represented as a wireframe, a plot similar to that in Figure 17 is created. This plot shows the runway, horizon line and intersection line as well as the canopy and nose of the plane. If the runway is visible above the nose, then this would be an acceptable position for the design eye, assuming the total vision plots also meet the requirements. If the model is shaded, a plot as shown in Figure 18 is created. This plot is much easier to interpret; the nose is the shaded component and the canopy is the wireframe component. The ability for the designer to interactively rotate the pilots head and change the simulated view is not a requirement under the military standards, however it would be a very beneficial tool for the designer and is currently being coded.

This last example of the shaded model with the PHIGS view set up at the design eye location could be extended to represent the view of a weapons guidance system. A PHIGS view with the reference point set up at the design eye location or weapons eye location could be set up and the model shaded. This would present the designer with the "weapons eye view" from which the best possible location for the weapons could be determined.

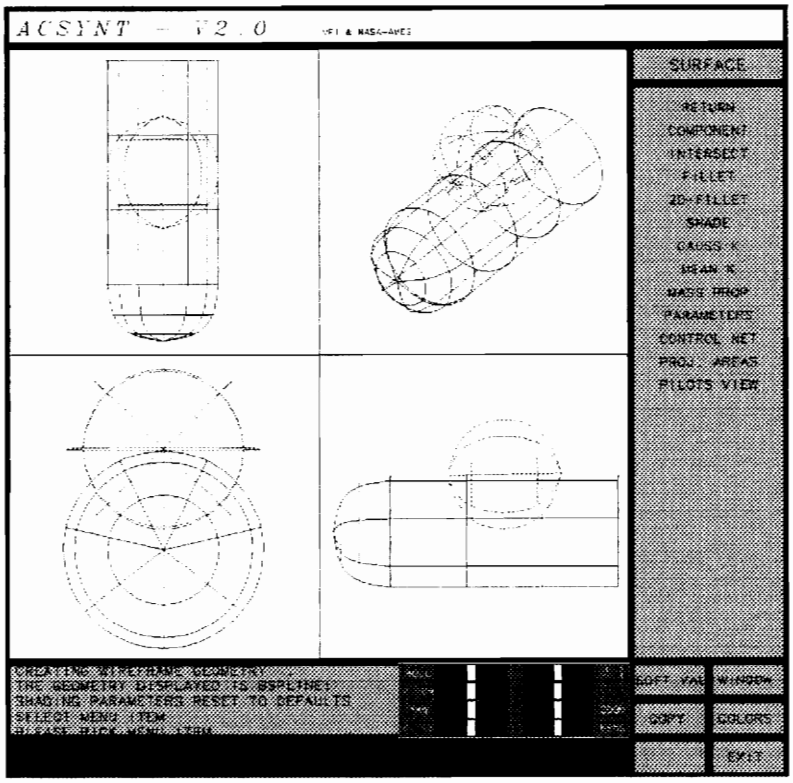


Figure 9 - Cylinder-Sphere model

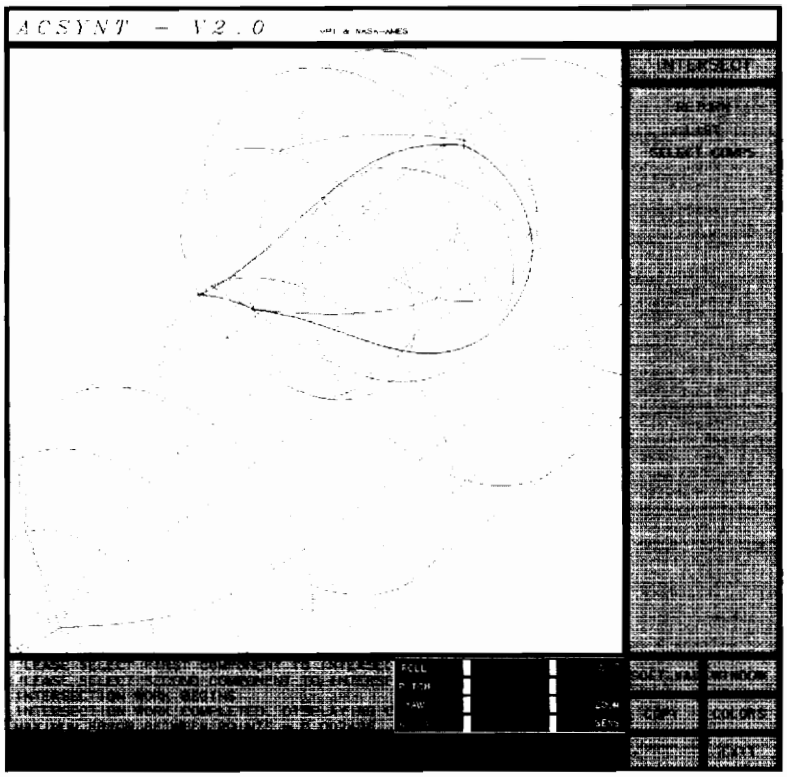


Figure 10 - Cylinder-Sphere intersection curve

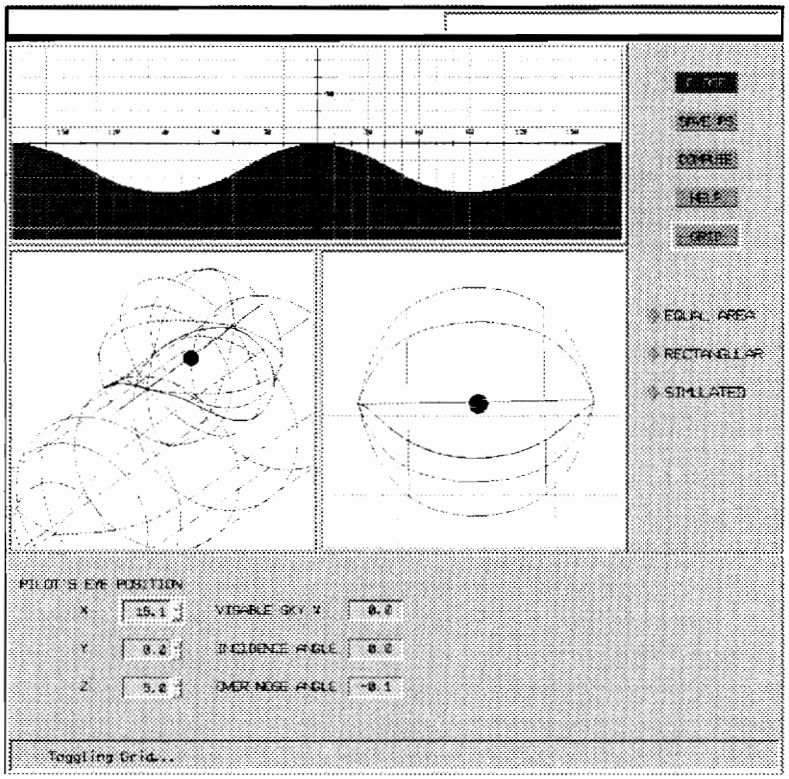


Figure 11 - Cylinder-Sphere rectangular vision plot

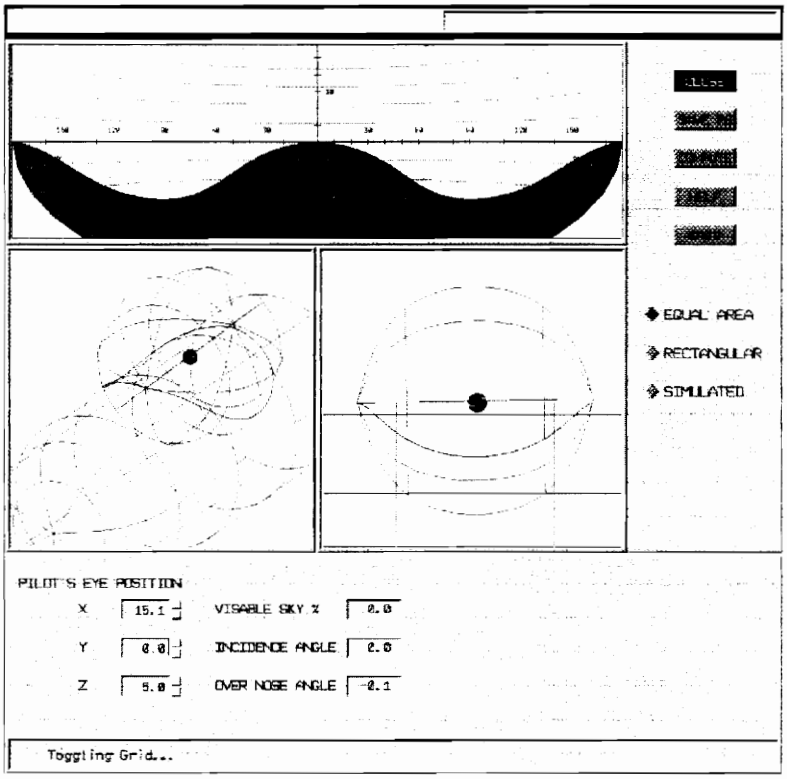


Figure 12 - Cylinder-Sphere equal area vision plot

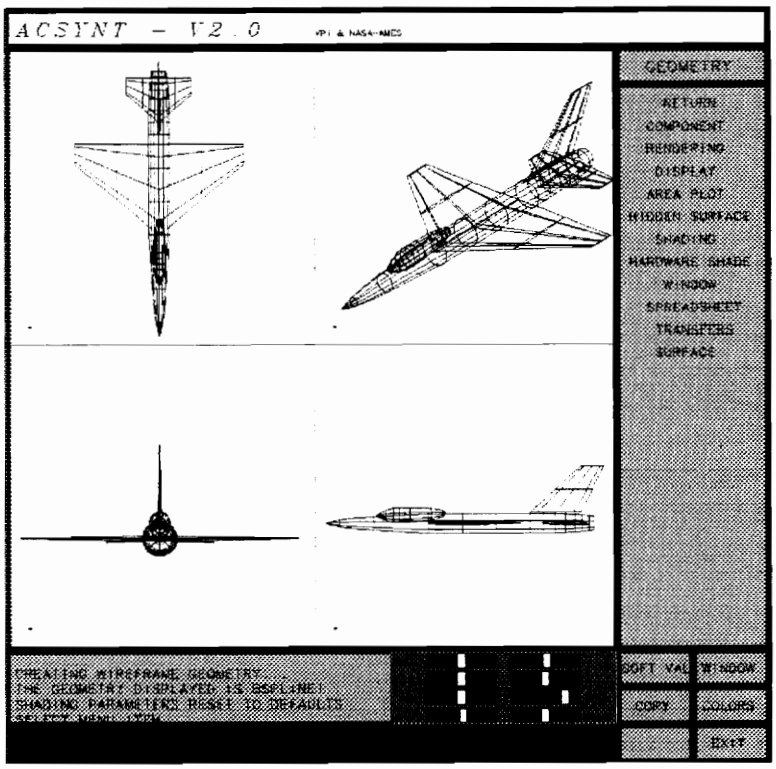


Figure 13 - F16 model

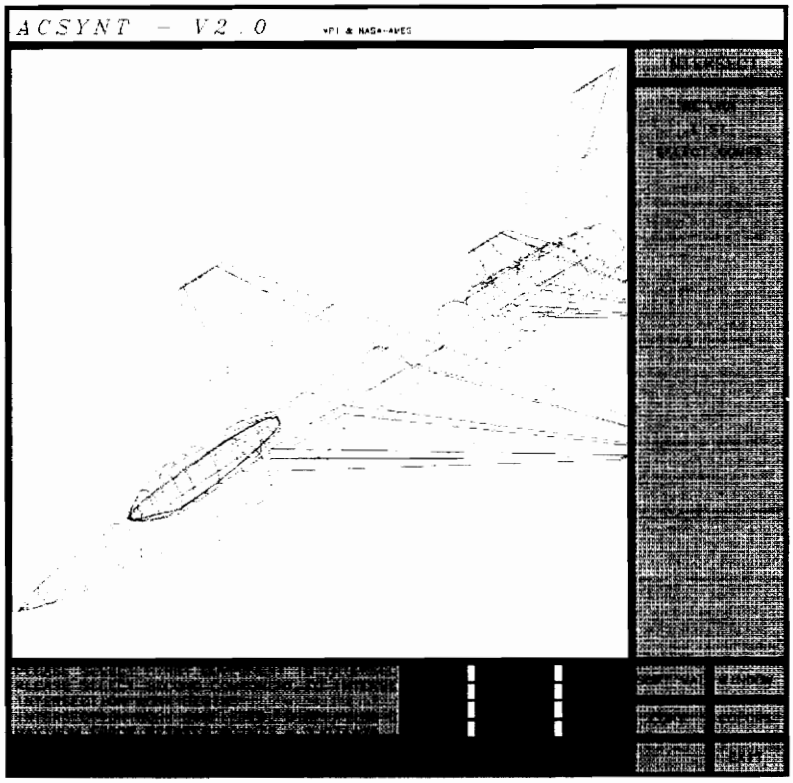


Figure 14 - F16 intersection curve

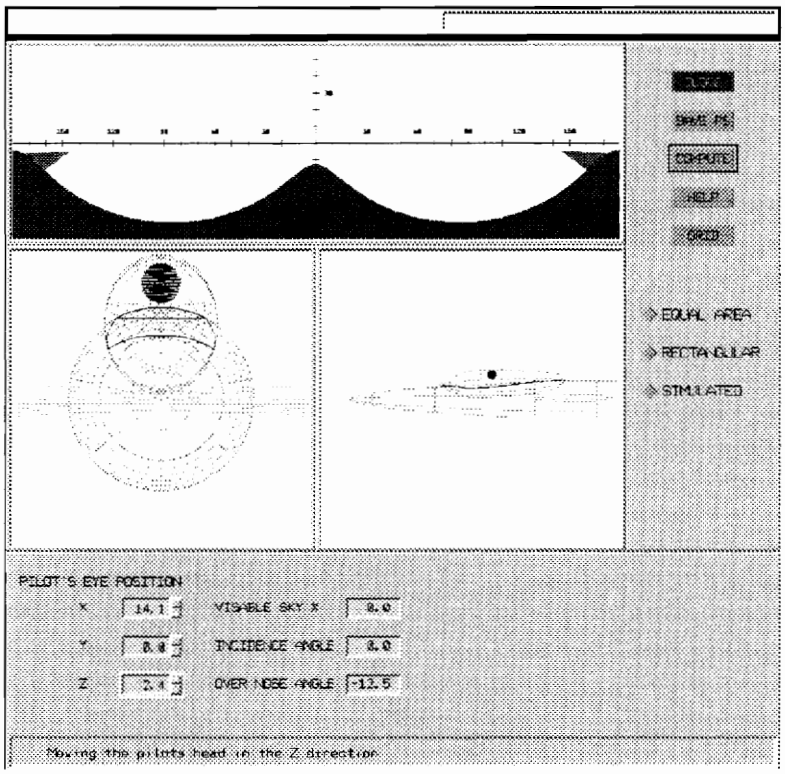


Figure 15 - F16 rectangular vision plot

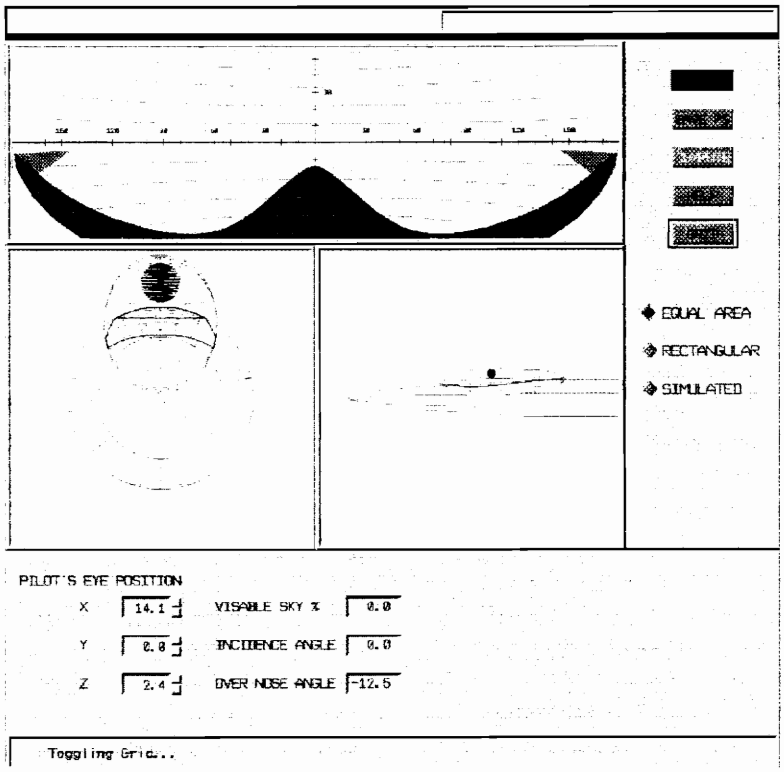


Figure 16 - F16 equal area vision plot

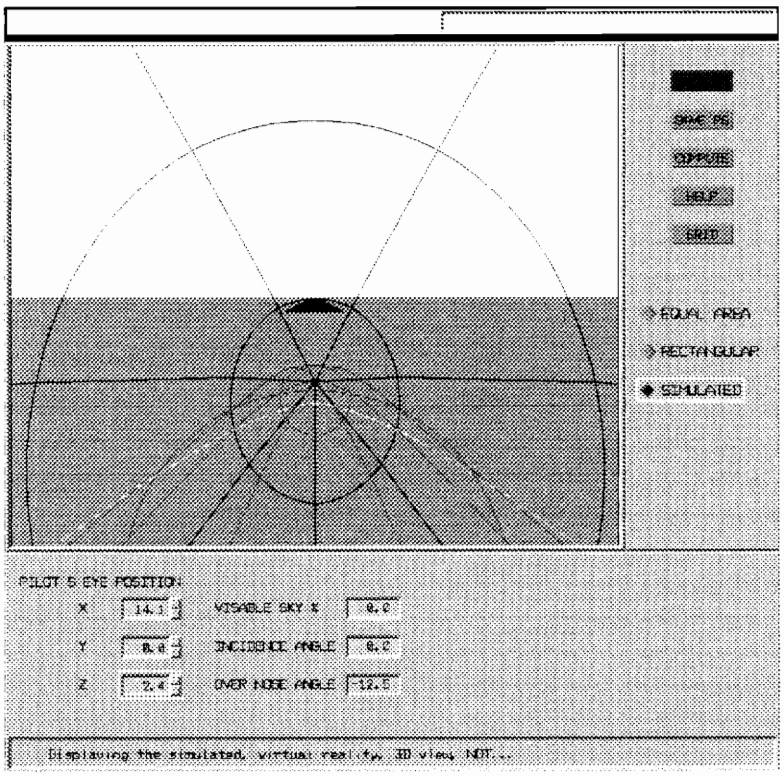


Figure 17 - F16 simulated landing approach view

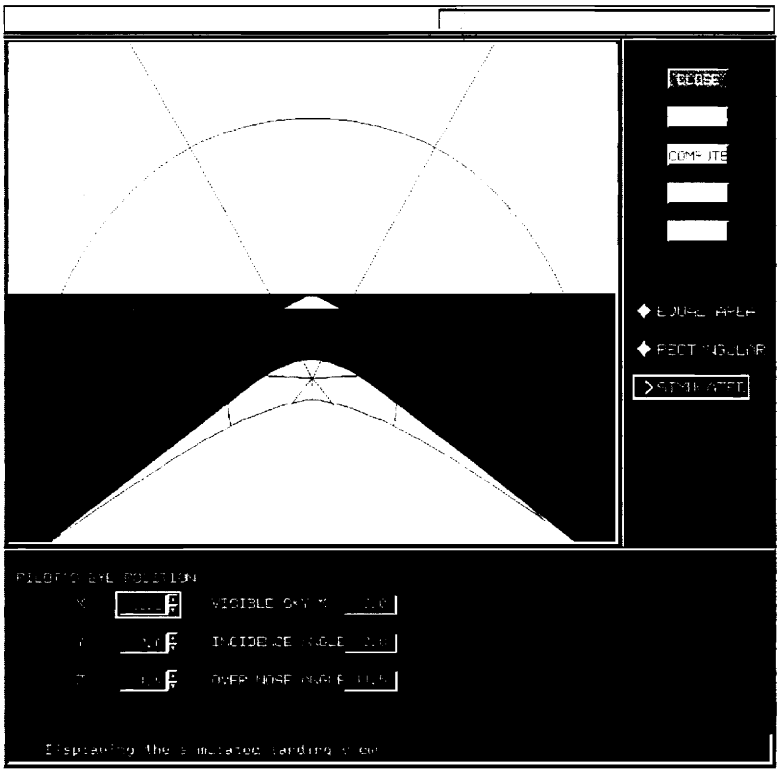


Figure 18 - F16 Shaded simulated landing approach view

9.0 Conclusions

Although the Pilot's View Module can be improved, the question remains "Does this module meet its goals ?" The first goal was to provide the aircraft designer with a program that would automate to process of producing the total vision plots as specified in the military standards. This goal is met by the Pilot's View Module. The second goal was to provide this automation with a considerable time savings over the manual approach. Again, this goal is met by the module. The other goals were that the pilots view module would interface with the existing ACSYNT code and utilize the geometry structures created within ACSYNT for the calculations, but that it would be coded in a way that would allow for easier, future improvement and modification. These goals were met as well. The pilots view module utilizes the existing ACSYNT geometry structures and intersection routines for the calculations but is coded in an OOP method using C++ and an OOP graphical user interface based upon the ISO graphical standard, PHIGS.

Although the program is functional and meets the goals set forth, it can be improved. Determination of the pilot's vision is a non-trivial calculation. ACSYNT is a conceptual design tool, and in order to create a program that is helpful, yet quick, many assumptions and compromises had to be made. The existing ACSYNT canopy and fuselage geometry do not have the details such as canopy

framing and instrumentation. Thus, the only determining factor as discussed earlier is the intersection line. As far as other components are concerned, (e.g., engines, wings, canards, etc...) , more simplifications must be made such as representing them as flat planes to reduce computational time. The intersection routines and the methods of implementation are robust but are not capable of handling all possible configurations. An example of a case that would fail include a canopy located on a wing, located on the tip of the nose or located in any other abnormal place.

There are other limitations of the module due to the existing structure, component list and conceptual level of ACSYNT. For most commercial and some military airplanes, such as cargo planes, there is no defined canopy component, but instead, these aircraft have windows. This module will not handle the analysis of such aircraft. The research work of Tjung was in the area of mapping plane curves onto B-spline surfaces [Tjun93]. A future version of the pilots view module could be modified to utilize this technique of plane curve mapping to place windows onto the surface of the aircraft. The engineer could design the windows as 2-D curves then map them on the surface. Once the mapping was complete, the new 3-D coordinates could be sent into the existing transformation routines to provide the proper total vision plots.

One other possible application of this pilots view module would be in the area of the weapons systems on aircraft. The weapon systems on today's aircraft are very similar to the pilots, for they too have "eyes" that need to see. These

"eyes" range from infrared sensors to radar antennas and lasers sights. All of these "eyes" have a respective transmitting or receiving cone that must not be infringed upon by the aircraft structure. The code could be modified so that the design eye location could be located at the tip of the weapon and given the parameters of the weapons sensor cone, a plot similar to the total vision plot could be produced. If shading was incorporated, the geometry could be shaded with the view set up representing the cone of the weapon. This would result in a simulated "weapons view".

To conclude, this research laid down a foundation for the pilots view module and accomplished the basic goals as stated. The future applications, modifications and extensions of this module as well as ACSYNT are numerous. Figures 19 and 20 represent some color photos of the pilot's view module interface.

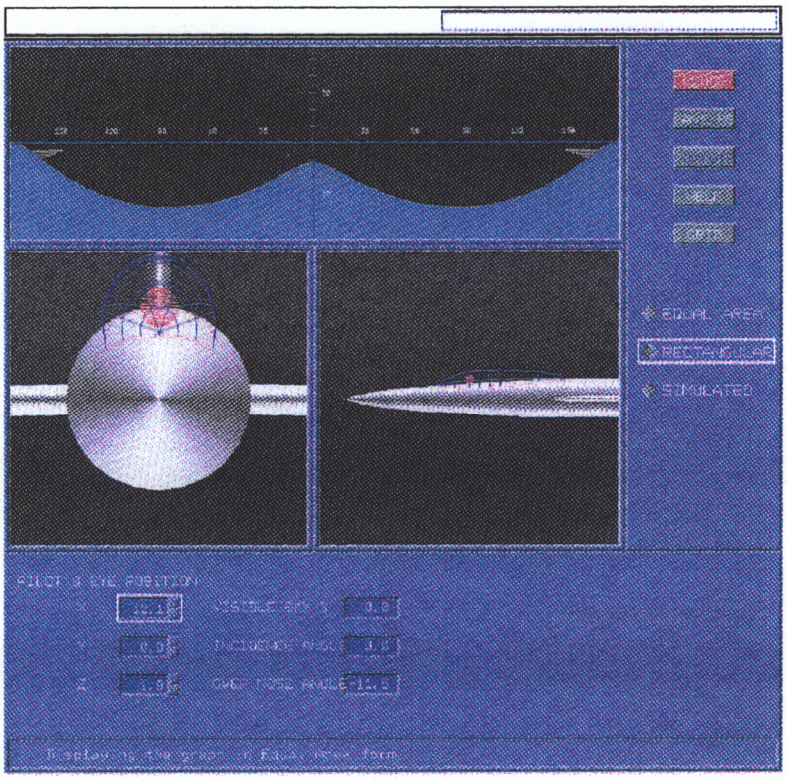


Figure 19 - Photo of pilot's view interface

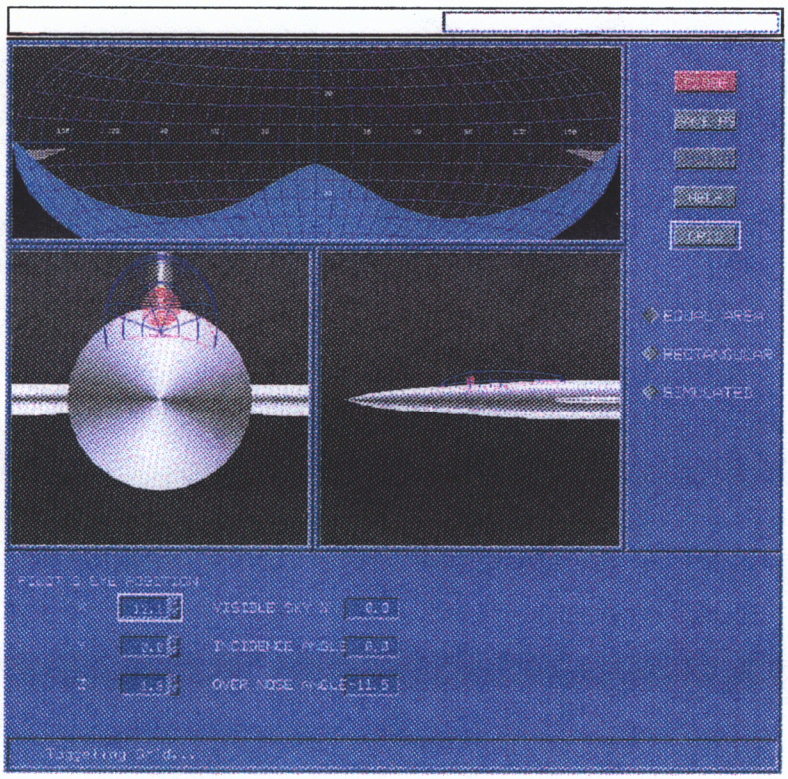


Figure 20 - Photo of pilot's view interface

10.0 References

[Bolu90] Bolukbasi, A.O., Bertone, C.M., "Helicopter Crew Station Design Using a Computerized Human Model", *Annual Forum Proceedings - American Helicopter Society Volume 1.0*, 1990, pp.645-648.

[Brau92] Braune, R.J., Graeber, C., "Human Centered Designs in Commercial Transport Aircraft", *Proceedings of the Human Factors Society 36th Annual Meeting*, 1992, pp.1118-1122.

[Casp90] Casper, P., "Using Hypercard® in Crewstation Design: Beyond Rapid Prototyping", *Annual Forum Proceedings - American Helicopter Society Volume 1.0*, 1990, pp.645-648.

[Chen86] Chen, John J., Tulga, M., "An Intersection Algorithm For C^2 Parametric Surface", *Knowledge Engineering and Computer Modelling in CAD, Proceedings of CAD86*, 1986, pp. 69-77.

[Deet69] Deetz, C., Adams, O., "Elements of Map Projection", Greenwood Press, 1969.

[Gask92] Gaskins, T., "PHIGS Programming Manual", O'Reilly & Associates, Inc., 1992.

[Glou89] Gloudemans, J. R., "Filleting of Aircraft Components Using Non-Uniform B-Spline Surfaces", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1989.

[Glou90] Gloudemans, J. R., Myklebust, A. "Filleting of Aircraft Components Using Non-Uniform B-Spline Surfaces", presented at *The International Federation for Information Processing WG 5.2 Workshop on Geometric Modeling*, Rensselaerville, New York, June 17-21, 1990.

[Hasa93] Hasan, S., Research work towards a Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1993.

[Howa91] Howard, T.L.J., "Evaluating PHIGS for CAD and General Graphics Applications", *Computer-Aided Design*, May 1991, pp.244-251.

[Jaya91] Jayaram, U., "Extracting Dimensional Geometric Parameters from B-Spline Surface Models", Dissertation - Doctor of Philosophy in

Mechanical Engineering, Virginia Polytechnic Institute and State University, 1991.

[Jaya92a] Jayaram, S., Myklebust, A., Gelhausen, P., "ACSYNT - A Standards-Based System for Parametric Computer Aided Conceptual Design of Aircraft", *American Institute of Aeronautics and Astronautics*, AIAA 92-1268, *1992 Aerospace Design Conference*, February 3-6, 1992 / Irvine, CA.

[Jaya92b] Jayaram, U., Myklebust, A., Gelhausen, P., "Extracting Dimensional Geometric Parameters from B-Spline Surface Models of Aircraft", *American Institute of Aeronautics and Astronautics*, AIAA 92-4283, *AIAA Aircraft Design Systems Meeting,* August 24-26, 1992 / Hilton Head, SC.

[Jaya93a] Jayaram, S., and Myklebust, A., "Device-Independent Programming Environments for CAD/CAM Software Creation", Computer Aided Design, vol. 25, no.2, February, 1993.

[Jaya93b] Jayaram, S., and Myklebust, A., "Evaluating PHIGS for CAD Applications - A Case Study" , 1st Annual PHIGS User's Group Conference, March 21-24, 1993, Orlando, Florida.

[Jone91] Jones, R., "Intersection and Filleting of Non-Uniform B-Spline Surfaces", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1991.

[Kell93] Kelly, J., "Rule-based Fuselage and Spline and Cross-Section Methods for Computer Aided Design of Aircraft Components" Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1993.

[Lari91] Larimer, J., Prevost, M., "Human Visual Performance Model for Crewstation Design", *SPIE Vol. 1456 - Large-Screen-Projection, Avionic, and Helmet-mounted Displays*, 1991, pp.196-210.

[Lass86] Lasser, D., "Intersection of Parametric Surfaces in the Berstein-Bezier Representation", *Computer Aided Design* , 18, No. 4, 1986, pp.186-192.

[MILI84] Military Standards 850B - Aircrew Station Vision Requirements For Military Aircraft, 1984.

[Pear84] Pearson, F. II., "Map Projection Methods", Sigma Scientific, Inc., 1984, pp.91-154.

[Peng84] Peng, Q. S., "An Algorithm for Finding the Intersection Lines Between Two B-spline Surfaces", *Computer Aided Design*, 16, No. 4, 1984, pp.191-196.

[Prat91] Prata, S., "C++ Primer Plus - Teach Yourself Object-Oriented Programming", Waite Group Press, 1991.

[Rive93] Rivera, F., "An Object-Oriented Method of Mission Profile Input for Aircraft Design" Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1993.

[Schr91] Schrock, E., "A PHIGS-Based Spreadsheet for Conceptual Design", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1991.

[Schr92] Schrock, E., Jayaram, S., and Myklebust, A., "A PHIGS-Based Spreadsheet for Conceptual Design", presented at and published in the proceedings of the ASEE 5th International Conference on Engineering

Computer Graphics and Descriptive Geometry, Melbourne, Australia, August 17-21, 1992.

[Schr93] Schrock, E., PostScript Application User's Guide, 1993.

[Squi93] Squire, D., "Afterbody Drag Prediction for Conceptual Aircraft Design", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1992.

[Steu93] Steude, A., "Development of an Object-Oriented Graphical User Interface for an Aircraft Engine Cycle Analysis Program", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1993.

[Tjun93] Tjung, Jie Wen, "Projection, Design and Representation of Curves on B-Spline Surfaces", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1993.

[Uhor93] Uhorchak, S., "An Object-Oriented Class Library for the Creation of Engineering Graphs", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1993.

[Wamp88a] Wampler, S., "Development of a CAD System for Automated Conceptual Design of Supersonic Aircraft", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1988.

[Wamp88b] Wampler, S., Myklebust, A. Jayaram, S., and Gelhausen, P., "Improving Aircraft Conceptual Design - A PHIGS Interactive Graphics Interface for ACSYNT", *American Institute of Aeronautics and Astronautics*, AIAA-88-4481, 1988.

[West84] Westfall, C., "Basic Graphics and Cartography", University of Maine at Orono Press, 1984.

[Woya92] Woyak, S., "Motif-Like Object-Oriented Interface Framework Using PHIGS", Thesis - Masters of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1992.

[Woya93] Woyak, S., and Myklebust, A., "Motif-Like Object-Oriented Interface Framework Using PHIGS", 1st Annual PHIGS User's Group Conference, March 21-24, 1993, Orlando, Florida.

Appendix A - Pilot's View User's Guide

Pilots View Module -- User's Guide

Introduction

The *Pilots View Module* of **ACSYNT** was designed to aid the aircraft engineer in the design, placement and performance of different canopies. The main purpose is for the *Pilots View Module* to automate the process of producing the total vision plots as specified in the military standards 850B. These plots can then be of use to determine the position of the design eye to provide adequate external vision from within the aircrew stations as required by the standards.

Requirements

To use the *Pilots View Module* to examine an aircraft's visibility, a canopy component **must** exist. It is the intersection of this canopy component with the aircraft structure that determines the basic visibility envelope. Without a properly constructed canopy, the intersection routine will not work properly and the analysis will fail.

Procedure

To enter the *Pilots View Module*, first load or create the model of the aircraft and then select the GEOMETRY menu. From this menu, pick the SURFACE selection and then the PILOTS VIEW selection. The new *Pilots View Module* interface will appear in a separate window. The interface should appear as in Figure 1. The aircraft should be visible in two views, front and side. The displayed buttons and their function are:

Pushbuttons

- COMPUTE -- Computes the intersection data
- SAVE PS -- Saves the total vision plots as a PostScript file for printing
- HELP -- Brings up help window, Not functional yet
- EXIT -- Return to SURFACE menu
- GRID -- Toggles the grid on or off

Radio Buttons: (Only one is active at a time)

- RECTANGULAR -- Displays the total vision plot in rectangular form
- EQUAL AREA -- Displays the total vision plot in Aitoff's equal area form
- SIMULATED -- Displays a simulated, perspective view with a runway in the distance

Number Boxes:

- X -- Moves the design eye location in the x-direction
- Y -- Moves the design eye location in the y-direction
- Z -- Moves the design eye location in the z-direction

From the buttons on the upper right of the window, select the COMPUTE button. A pop-up menu will appear requesting the components from which the routine will check for intersection with. Choose the components from the list and then select the DONE button. The intersection routines will then compute the intersection of the canopy with the other components, and after a few moments, the total vision plot will appear. The intersection line will also appear as a red line on the two aircraft views.

Once the intersection has been computed, the design eye location, represented by a red sphere in the two aircraft views, can be moved and the total vision plot will be automatically updated. To move the design eye, use the number boxes on the lower left of the screen. Using the mouse, the x, y and z locations can be incremented up and down. As these values are changed, the red sphere should move accordingly and the plot will be updated.

When a suitable total vision plot is obtained, it can be saved to a PostScript file for printing to any PostScript capable printer. Select the SAVE PS push button which will pop up a new menu. This menu requests a file name to save the output under. The name should be 8 characters in length. Type the name and

select the DONE button. The total vision plots, both rectangular and equal-area along with the design eye location and other information will be save as a PS file in the current directory. This file can the be sent to the printer with the usual UNIX qprt or lpr command. **NOTE**: These files are fairly large so they do take several minutes to print on most printers.

The other viewing option is the simulated view. This view tries to represent exactly what the pilot would see looking out the front of the aircraft. When the SIMULATED button is selected, a menu pops up with several runway selections. These are the required runways as specified by the military standards. Once a selection is made, the simulated view will appear. The aircraft will be displayed as a wire-frame model in gray. The intersection line will be the red line. If the intersection line appears above the nose of the aircraft and the nose is below the runway in the distance, then the design eye location is acceptable. To exit from the simulated view, select either the rectangular or equal area choices.

To return to the surface module of ACSYNT, select the EXIT button from the upper right of the screen. The *Pilots View Module* window will be erased and the surface module will be functional.

Appendix B - Military Standard 850B

MIL-STD-850B
3 November 1970
Superseding
MIL-STD-850A
8 June 1967

MILITARY STANDARD

AIRCREW STATION VISION
REQUIREMENTS FOR MILITARY AIRCRAFT

FSC 15GP

DEPARTMENT OF DEFENSE
Washington, D. C. 20301

Aircrew Station Vision Requirements for Military Aircraft

MIL-STD-850B

* 1. This Military Standard is mandatory for use by all Departments and Agencies of the Department of Defense.

* 2. Recommended corrections, additions, or deletions should be addressed to the Aeronautical Systems Division (ENZSA), Wright-Patterson Air Force Base, Ohio 45433.

MIL-STD-850B
Notice 1
23 November 1984

MILITARY STANDARD

AIRCREW STATION VISION REQUIREMENTS FOR MILITARY AIRCRAFT

TO ALL HOLDERS OF MIL-STD-850B:

1. THE FOLLOWING PAGES OF MIL-STD-850B HAVE BEEN REVISED AND SUPERSEDE THE PAGES LISTED:

NEW PAGE	DATE	SUPERSEDED PAGE	DATE
1	3 November 1970	(REPRINTED WITHOUT CHANGE)	
2	23 November 1984	2	3 November 1970
5	23 November 1984	5	3 November 1970
6	23 November 1984	6	3 November 1970
7	23 November 1984	7	3 November 1970
8	3 November 1970	(REPRINTED WITHOUT CHANGE)	
9	3 November 1970	(REPRINTED WITHOUT CHANGE)	
10	23 November 1984	10	3 November 1970

2. RETAIN THIS NOTICE AND INSERT BEFORE TABLE OF CONTENTS

3. Holders of MIL-STD-850B will verify that page changes and additions indicated above have been entered. This notice page will be retained as a check sheet. The issuance, together with appended pages, is a separate publication. Each notice is to be retained by stocking points until the Military Standard is completely revised or canceled.

Custodians:
Army - AV
Navy - AS
Air Force - 11

Preparing activity:
Air Force - 11

Project No. 15GP-0048

International interest: (see 15.1)

CONTENTS

FIGURES

1. PURPOSE AND SCOPE

1.1 Purpose. The purpose of this document is to establish requirements for providing adequate external vision from within the aircrew stations of military aircraft.

1.2 Scope. The requirements contained herein apply to external vision from aircraft procured by the military departments. The general vision requirements are for the various type seating arrangements and limited aircraft missions.

2. REFERENCED DOCUMENTS

2.1 The issues of the following documents in effect on date of invitation for bids or request for proposal form a part of this standard to the extent specified herein.

STANDARDS

Military

MIL-STD-1333	Aircrew Station Geometry for Military Aircraft
MS33573	Dimensions, Clearance, Cockpit, Fixed Wing Aircraft
MS33574	Dimensions, Basic, Cockpit, Stick Controlled, Fixed Wing Aircraft
MS33575	Dimensions, Basic, Cockpit, Helicopter
MS33576	Dimensions, Basic, Cockpit, Wheel Controlled, Fixed Wing Aircraft

(Copies of specifications, standards, drawings, and publications required by suppliers in connection with specific procurement functions should be obtained from the procuring activity or as directed by the contracting officer.)

3. DEFINITIONS

* 3.1 Types of cockpit seating arrangements. The vision criteria set forth in this standard are applicable to the following types of cockpit seating arrangements:

(a) Single pilot

(b) Tandem pilot

(c) Side-by-side pilot.

* 3.2 Types of aircraft. The vision criteria set forth in this standard are applicable to the following types of aircraft:

(a) Fighter/attack

(b) Bomber/transport

(c) ASW/patrol

(d) Helicopter

(e) V/STOL

(f) Trainers.

3.3 Design eye position. The design eye position shall be that eye position as defined by MS33573, MS33574, MS33575, and MS33576 or MIL-STD-1333.

* 3.4 Clear vision area. The clear vision area is defined as that area of transparent material through which vision is unobstructed by structure, edge bonding material, or any material which prohibits clear vision. These areas shall be kept clear of obstructions to vision such as misting, rain, icing and insects.

4. GENERAL REQUIREMENTS

4.1 Vision requirements. The vision requirements set forth in this standard were based on utilization of monocular vision. The reference plane from which the vision angles are specified shall be the pilot's horizontal vision plane (or line) with respect to the specific aircraft longitudinal fuselage reference line. The zero reference in azimuth shall be straight ahead of the design eye position.

4.1.1 Extent of external vision. The maximum practicable external vision shall be provided for the pilot(s). The vision requirements established herein are minimal and are subject to further definition by the procuring activity. Particular consideration shall be given in each case to the external vision provided in conjunction with the specified mission of the aircraft. For landing approach conditions, downward and forward vision shall be provided to insure effective vision of all ground and shipboard landing aids specified by the procuring activity. For carrier aircraft, the landing area, landing aids, and stern of the carrier (drop lights) must be visible from landing pattern altitude in level flight at glide slope intercept. The landing area, including the contiguous areas port and starboard, must be visible on the glide slope. The pilot's vision of these landing aids shall be provided when the aircraft longitudinal fuselage reference line is pitched at the critical approach conditions Vpa (min). Verification for meeting this requirement shall be accomplished as specified in 4.1.2.2.

4.1.2 External vision verification. The vision data derived in accordance with the instructions herein shall be submitted to the procuring activity for review and approval. Vision plots shall be updated on completion of design and at any time a change in the vision envelope is either proposed or completed.

* 4.1.2.1 Total vision envelope plot. The total vision envelope (plus and minus 180° in azimuth and plus and minus 90° in elevation) shall be plotted for each crewmember in the flight compartments where the crewmember's duties require external vision. The plots shall reflect the unobstructed vision as defined in 3.4. The plots shall be presented as specified by the procuring activity on either US Department of Commerce form number 3099 (Aitoff's Equal Area Projection of a Sphere) or on rectilinear plots using the technique described in figures 1 and 2 (reference figures 3 through 7). The scale of the rectilinear plots shall be one-tenth of an inch equal to 1° of azimuth and elevation. The obscuration of external vision caused by accessory equipment, fixed or retractable (such as gunsight framework and air-refueling probes), shall also be shown and identified on the vision plots. The vision plots shall be made using minimum one-fourth scale accurate crew station engineering data; i.e., loft line contours, structural arrangement, and transparency installation.

* 4.1.2.2 Landing approach vision plots. A plot (as illustrated on figures 8, 8A, and 8B) of each pilot station shall be provided. The plot shall be presented on transparent material 8-1/2 by 11 inches in size. Minimum vision evaluations shall include:

* (a) Land-based aircraft - 300- by 11,600-foot runway on a 3° glide slope at distances from runway end of 1 mile, 1/2 mile, and 1/4 mile, as illustrated by figures 9, 10, and 11.

* (b) Carrier-based aircraft - Same as (a) above plus aircraft carrier at distances of 1/2 mile, 1/4 mile, 450 feet, 300 feet, and 150 feet on a 4° glide slope as illustrated by figures 12, 13, 14, 15, and 16. Additional evaluations using varying glide slopes and carrier sizes may be required by the procuring activity. If so, the distances noted above shall be used.

(c) Each vision plot submitted shall include the following information clearly marked on the face:

(1) Specific pilot stations

(2) Fuselage angle with relation to the horizon

(3) Maximum rate of closure in feet per second.

* 4.1.3 Quality of external vision. Radii-of-curvature and angles of incidence of the transparent components of the cockpit shall be consistent with the aerodynamic, structural, and fabricating considerations, which will result in the least possible optical distortions in these parts, and shall minimize reflections of objects both within and without the cockpit from interfering with the pilot's vision. Minimum distortion specifications will be established by the procuring activity. At the intersection of the horizontal vision line and the windshield, the angle of incidence shall not exceed 60°. Every attempt shall

be made to meet the maximum 60° requirement on other areas of the transparency(ies) from the design eye position as closely as possible. The angle of incidence is defined as the angle between the line of sight and the normal (perpendicular) to the surface.

* 4.1.4 The external vision requirements are based on the assumption that the vision areas are symmetrical with respect to the centerline of the aircraft. The minimum vision requirements stated for the pilot's position in the side-by-side piloted aircraft are, therefore, applicable to the copilot's position with the angles of azimuth reversed.

* 4.1.5 The requirements for the extent of external vision for transparent areas between azimuth positions, as specified in sections 5, 6, 7, and 8 shall progressively increase, or decrease, as specified between the azimuthal positions. It is acknowledged that the presence of framing at any of the specified points shall not be considered a nonconformance with the standard requirements.

5. FIGHTER/ATTACK AIRCRAFT

* 5.1 Single pilot/tandem pilot. The vision requirements set forth in this paragraph are applicable to single- and tandem-piloted fighter aircraft (reference figure 3).

* 5.1.1 Forward pilot position. The vision criteria set forth in this paragraph are applicable to the forward pilot station:

(a) The following shall be the minimum angles of unimpaired vision available to the pilot from the design eye position:

(1) At 0° azimuth at least 11° down and 10° up. (Every effort should be made to exceed 11°.)

(2) At 20° azimuth, left and right, 20° down.

(3) At 30° azimuth, left and right, 25° down.

(4) At 90° azimuth, left and right, 40° down.

(5) At 135° azimuth, left and right, 20° down.

(b) The area above the canopy rail from the canopy bow aft, past the pilot's headrest shall be of transparent material.

(c) The windscreen forward of the canopy bow shall be as free of structure as possible consistent with sound engineering practice. No windscreen structure in this area shall exceed 2.0 inches in width when projected onto a plane perpendicular to a line between the structure and the design eye position.

(d) The blind spot caused by the structure between the top of the windscreen and the canopy shall not exceed 35° horizontally nor 7° vertically. Reference shaded area, figure 3.

(e) Aft vision from 135° to 180° azimuth shall be provided, as specified by the procuring activity.

5.1.2 Aft pilot position. The vision criteria set forth in this paragraph are applicable to the aft pilot station in tandem-piloted aircraft.

(a) At 0° azimuth, a minimum of 5° vision down shall be provided, measured from the aft design eye position with the forward seat in the neutral seat adjustment. The front seat headrest shall be designed to a minimum width, contoured about the upper corners and the forward seat structure; ejection rails and actuator shall be so designed and positioned as to obstruct a minimum of forward vision.

(b) The extent and quality of the aft pilot's external vision shall be as specified in 5.1.1. Various lateral eye positions in a plane through the aft pilot's design eye position may be employed to meet the above requirements. Sufficient canopy width shall be provided to permit eye movement in the frontal plane to obtain the specified external vision.

(c) The vision required by the above criteria may not be sufficient in all respects for all operational uses of an aircraft. Therefore, particular consideration must be given in each case to the vision provided in conjunction with the mission of the aircraft. In aircraft utilized for fighter-bomber missions, reconnaissance missions, search, and night intruder missions, the maximum amount of vision that can be incorporated is desirable.

5.2 Side-by-side pilot fighter/attack aircraft

* 5.2.1 The vision requirements set forth in this paragraph are applicable to side-by-side piloted fighter/attack-type aircraft (reference figure 4).

(a) The following shall be the minimum angles of unimpaired vision available to the pilot and copilot measured from their respective design eye positions:

(1) At 0° azimuth, provide at least 13° down and 12° up.

(2) From 0° through 70° left azimuth for the pilot and 0° through 70° right azimuth for the copilot, vision shall increase from 13° down to 40° down and from 12° up to 40° up. The canopy bow vision obstruction within this area shall be within the limits as specified in 5.1.1 (c).

(3) At all points between 70° and 110° left azimuth for the pilot and between 70° and 110° right azimuth for the copilot, provide 40° down and 90° up.

(4) At 135° azimuth, provide 20° down.

(b) In the cockpit area inboard of the pilot and copilot centerlines, maximum vision with a minimum of obstruction shall be provided for the pilot and copilot.

(c) Sufficient vision for the pilot(s) to see the engines with his head next to the window shall be provided where practicable.

(d) At 90° left azimuth for the pilot and 90° right azimuth for the copilot, at least 70° downward from a point not more than 14 inches outboard of the pilots' respective design eye positions shall be provided.

(e) In aircraft utilizing inflight refueling, sufficient vision must be provided for the pilot to see the tanker, refueling signal lights, boom, and probe, when in position for refueling.

6. BOMBER/TRANSPORT AIRCRAFT

6.1 Side-by-side pilot aircraft. The vision criteria stated in this paragraph are applicable to side-by-side piloted aircraft (reference figure 5).

(a) The following shall be the minimum angles of unimpaired vision available to the pilot and copilot left and right respectively, measured from the respective design eye positions:

(1) At 0° through 30° left azimuth, provide 17° down and 20° up.

* (2) The vertical angles of clear vision in the area between 30° and 70° left azimuth shall increase from 17° down to 35° down and from 20° up to 40° up.

(3) At all points between 70° and 110° left azimuth, provide 35° down and 40° up.

(b) To the right of the pilot's centerline the angular vision downward and upward may decrease slightly from 17° and 20° specified in (a) (1) above due to the increase in distance from the design eye position to the windshield.

(c) The transparent area shall extend to 135° left azimuth at the eye level but the vertical angles of vision may decrease below those values at 110°. Sufficient vision for the pilot to see the engines with his head next to the window shall be provided where practicable. If sufficient vision is not provided, provisions must be made for another crewmember to view the engines.

(d) At least 70° downward vision from a point not more than 14 inches outboard of the design eye positions shall be provided. Sufficient headroom shall be provided to permit this eye position.

(e) The entire transparent area between 20° right and 30° left shall be free of windscreen structure or posts. No windscreen post or structure shall exceed 2.5 inches in width when projected onto a plane perpendicular to a line between the post or structure and the pilot's design eye position.

(f) Vision throughout the entire overboard area aft to the seat reference point is desirable, though often not practicable due to structure, overhead console, etc. However, transparent material should be utilized to the maximum extent practicable throughout this area.

7. ASW/PATROL AIRCRAFT

7.1 The vision criteria stated in section 6 for bomber/transport aircraft are the minimum applicable to ASW/patrol aircraft. Maximum vision downward consistent with structural integrity shall be provided for pilot and copilot between 30° and 135°, measured from the design eye position. When lookout stations are specified by the procuring activity, vision available at these stations shall be 75° fore and aft or horizontally and 85° downward measured from zero reference.

8. HELICOPTERS

8.1 Side-by-side pilot. The vision requirements set forth in this paragraph are applicable to side-by-side piloted helicopters and are given relative to the longitudinal fuselage reference line (reference figure 6).

(a) Controls, consoles, and instrument panels shall be so located as not to restrict vision, with particular emphasis on adequate over-the-nose visibility. Insure that mounting and reinforcing frames or strips which divide transparent areas and form obstructions to vision are not more than 2 inches wide when projected onto a plane perpendicular to a line between the structure and the pilot's eyes at the design eye position. Distribute such obstruction to avoid critical vision areas.

(b) The following minimum angles of unimpaired vision, designated with respect to the right side pilot, shall be available to the pilot from the design eye position as shown on MS33575. When a co-pilot is designated on the left side, equivalent vision angles shall be provided for that side.

(1) At 0° azimuth, 25° down and 20° up shall be available.

(2) From 10° left azimuth to 10° right azimuth, downward vision shall increase from 20° to 30°.

(3) In the area between 10° and 135° right azimuth, 50° of downward vision shall be available.

(4) From 0° to 80° right azimuth, upward vision shall increase from 20° to 40°.

(5) From 80° right azimuth to 100° right azimuth, upward vision shall be 40°.

(6) From 100° right azimuth to 135° right azimuth, upward vision requirement may decrease gradually from 40° to 20°.

(7) From 10° left azimuth to 100° left azimuth, 20° up and 20° down shall be available.

(c) There shall be no vertical obstruction between 20° right and 20° left of the longitudinal axis relative to the design eye position.

(d) There shall be no horizontal obstructions in the area extending 15° above the horizon from 135° right to 40° left and decreasing to a point 10° above the horizon at 100° left. No horizontal obstructions shall be in the area 15° below the horizon between 135° right and 100° left. If necessary, horizontal obstructions in this area shall be limited to one above the horizon and one below the horizon and shall be restricted to 4 inches in width.

* 8.2 Single pilot/tandem pilot. The vision requirements set forth in this paragraph are applicable to single- and tandem-piloted helicopters with pilot in either forward or aft position, and are given relative to the longitudinal fuselage reference line (reference figure 7). The forward cockpit position, if occupied by other than the primary pilot (i.e., gunner or observer), also shall comply with these requirements.

(a) The following shall be the minimum angles of unimpaired vision available to the pilot from the design eye position:

(1) At 0° azimuth, at least 25° down and 70° up.

(2) [illegible]° azimuth, left and right, 25° down and 70° up.

(3) [illegible] left and right, 30° down and 70° up.

(4) [illegible]nd right, 50° down and 70° up.

(5) [illegible]°, 34° down and 70° up.

(b) Visibility in elevation (up) above that specified in 8.2(a) of at least 90° shall be provided. Interruptions of vision by horizontal structure of not more than 2 inches of width along or above the elevation boundary of 8.2(a) and vertical structural members of not more than 2 inches in width located in accordance with 8.1(d) are permitted.

9. V/STOL AIRCRAFT

9.1 The minimum vision requirements for V/STOL aircraft shall be those of the helicopter (side-by-side or tandem, as appropriate, for the configuration chosen) and of the fixed-wing-type aircraft which designates the primary mission. The precise requirements shall be specified by the procuring activity and shall consider the aerodynamic characteristics, engineering and power designs, and primary and secondary missions.

10. TRAINER AIRCRAFT

10.1 The extent and quality of external vision in trainer aircraft should be at least that of the aircraft which they are to represent. It is to be noted that the flight proficiency and the nature of the training flight performed will require a considerable increase in external vision. Accordingly, every attempt should be made to provide a higher degree of external vision in these aircraft than in their parent type to insure adequate safety of flight.

11. OTHER CREW STATIONS

11.1 In single-piloted aircraft employing tandem or side-by-side seating arrangements where the second crew station is occupied by a radar operator or bombardier/navigator, the vision requirements in the second crew station shall be as specified by the procuring activity. Insofar as practicable, the design of this crew station will follow the intent of this standard.

12. ADDITIONAL REQUIREMENTS

* 12.1 In those aircraft where the external vision need exceeds the minimum requirements specified herein, the maximum external vision that can be incorporated without sacrificing engineering integrity is desirable. When excessive solar heating through the transparent portions of the canopy roof is anticipated, means to reduce or eliminate this problem, such as by incorporation of flight crew operated extendable/retractable shields, may be employed at the discretion of the procuring activity. Methods which affect the transmissive characteristics of the clear transparencies shall be subject to the approval of the procuring activity.

13. RECONNAISSANCE AIRCRAFT

* 13.1 Those aircraft assigned to reconnaissance and research and night intruder missions require maximum external vision for tactical utilization of the aircraft as well as for carrier and field landing, taxiing, takeoff, approach and landing under all weather conditions.

14. AIRCRAFT UTILIZING INFLIGHT REFUELING

14.1 In aircraft utilizing inflight refueling, sufficient vision shall be provided for the pilot to see the tanker, refueling signal lights, boom, and probe when in position for refueling.

15. NOTES

* 15.1 Certain provisions of this document are the subject of international standardization agreements (NATO Standardization Agreement (STANAG) 3622, 3639 and ASCC Air Standards 10/53 and 10/55). When change notice, revision, or cancellation of this document is proposed which will affect or violate the international agreement concerned, the preparing activity shall take appropriate reconciliation action through international standardization channels, including departmental standardization offices, if required.

15.2 Data requirements. The selected data requirements in support of this standard will be reflected in a Contractor Data Requirements List (DD Form 1423) attached to the request for proposal, invitation for bid, or the contract as appropriate.

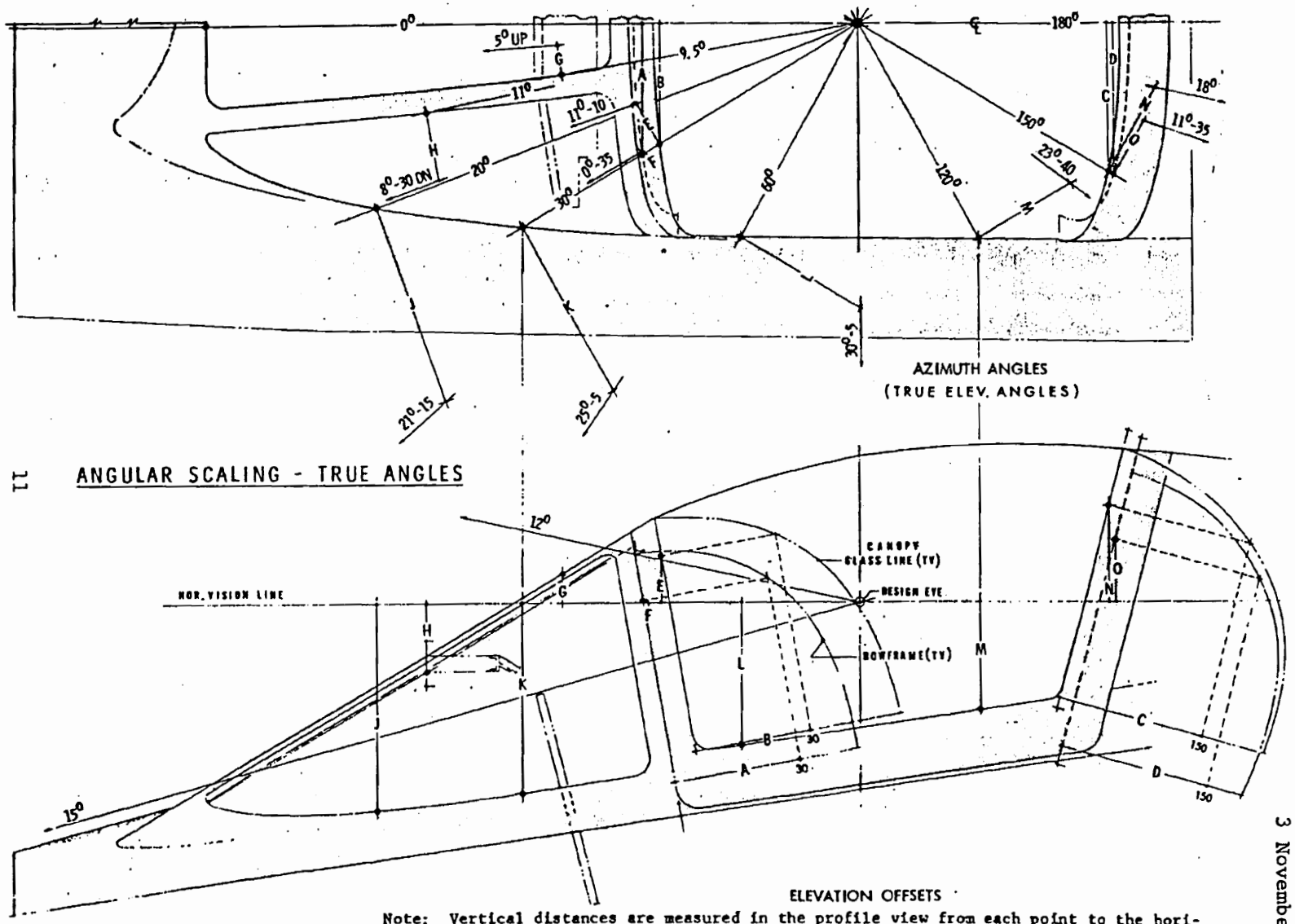
15.3 The margins of this standard are marked with an asterisk to indicate where changes from the previous issue were made. This was done as a convenience only and the Government assumes no liability whatsoever for any inaccuracies in these notations. Bidders and contractors are cautioned to evaluate the requirements of this document based on the entire content irrespective of the marginal notations and relationship to the last previous issue.

Custodians:
Army - AV
Navy - AS
Air Force - 11

Preparing activity:
Air Force - 11

Project No. 15GP-0048

* International interest: (see 15.1)



Note: Vertical distances are measured in the profile view from each point to the horizontal vision line and layed out perpendicular to the azimuth line in the plan view. The included angle measured in the plan view is the true angle of depression or elevation.

FIGURE 1. METHOD FOR DETERMINING DATA FOR TOTAL VISION PLOTS

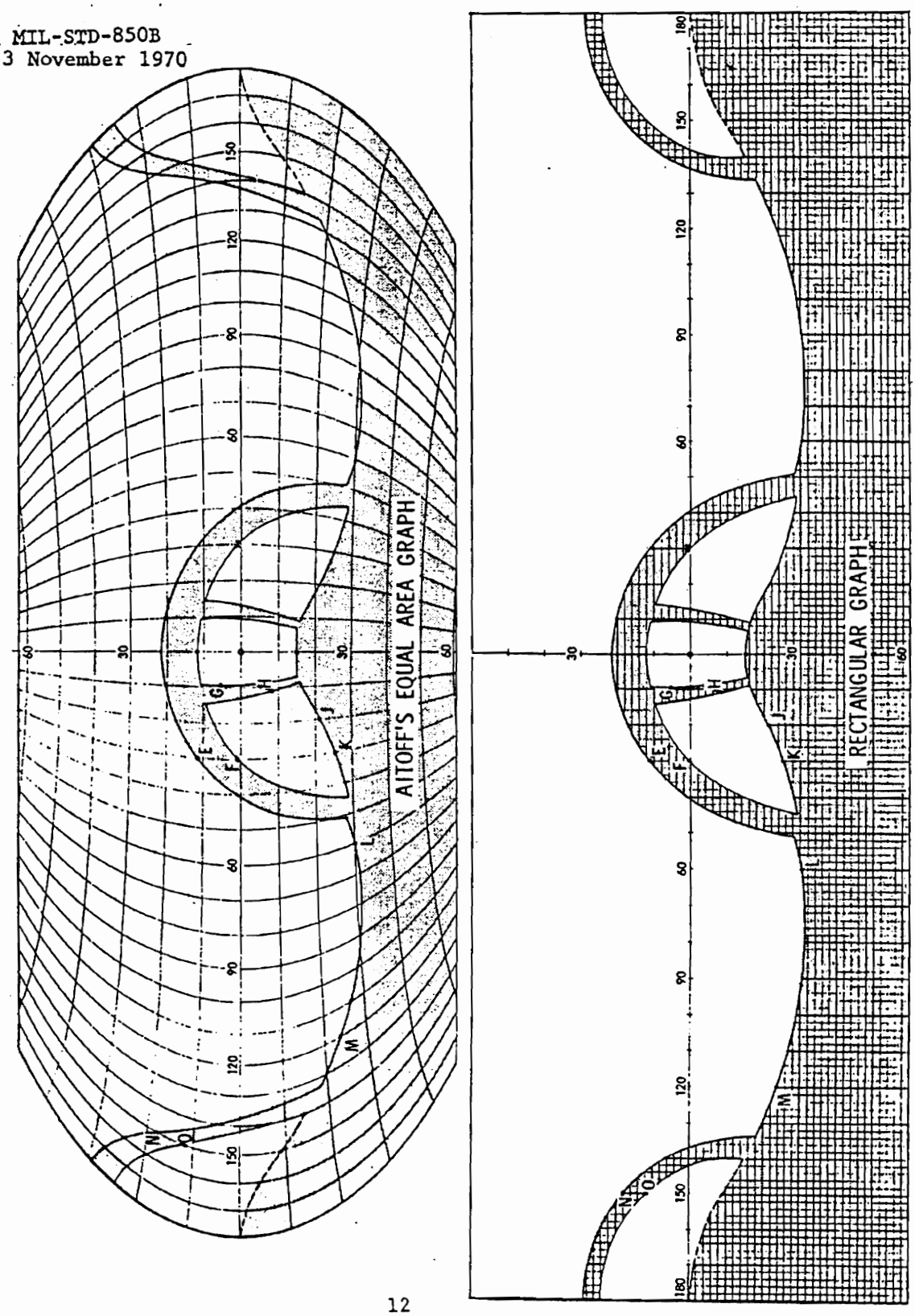


FIGURE 2. SAMPLE OF TOTAL VISION PLOTS

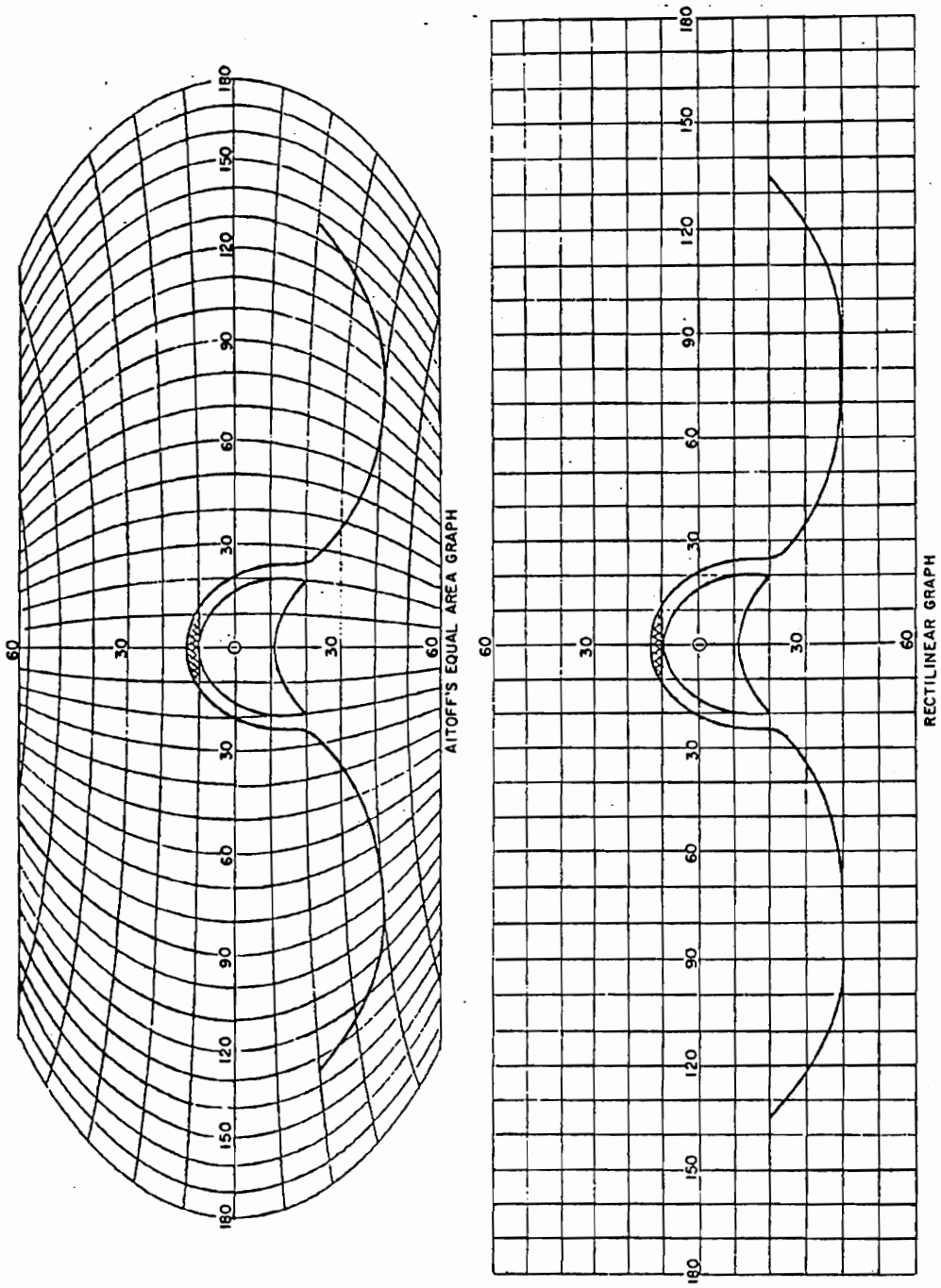


FIGURE 3. SINGLE AND TANDEM PILOT (FIGHTER) VISION PLOT

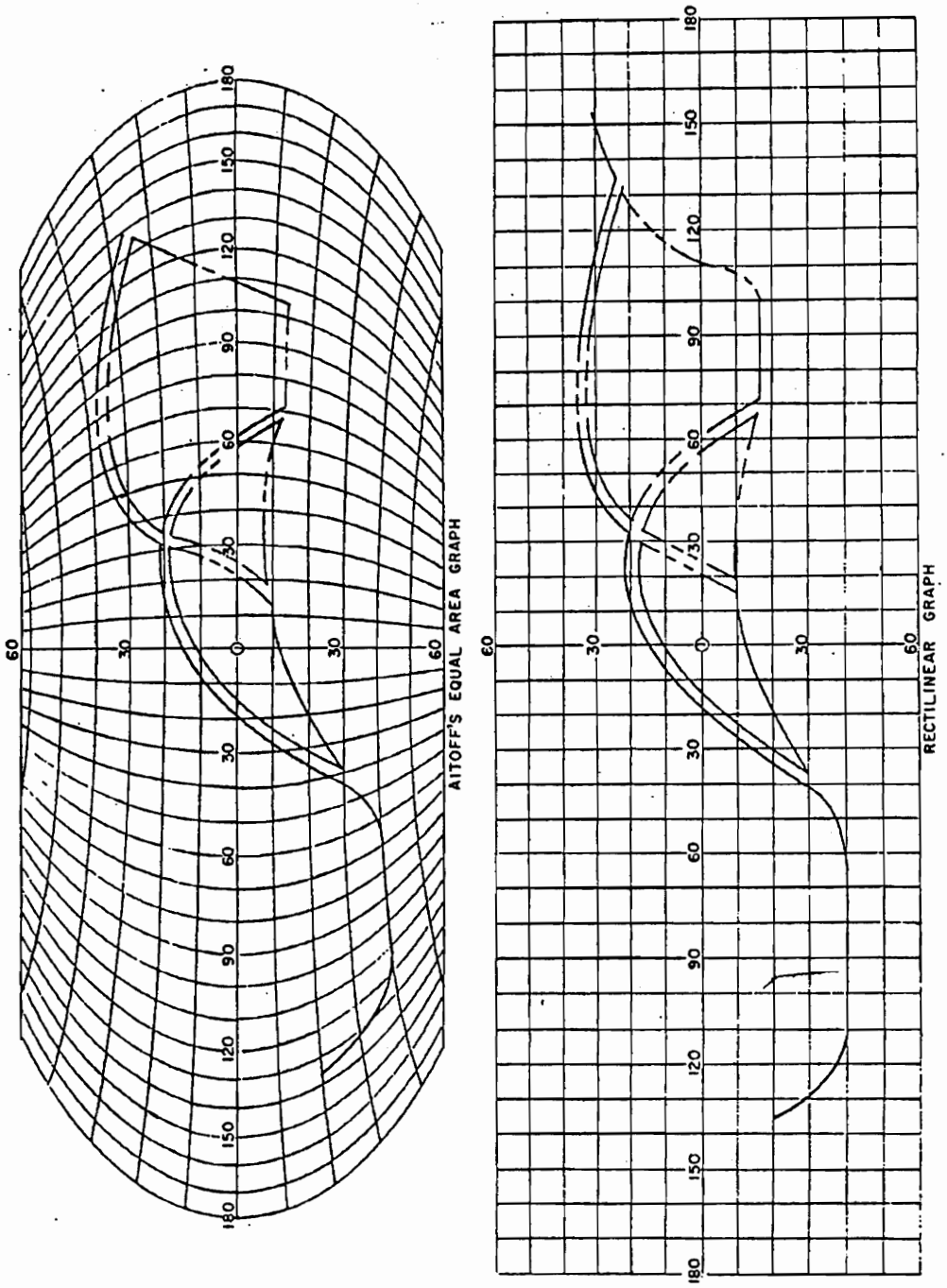


FIGURE 4. SIDE-BY-SIDE PILOT (FIGHTER) VISION PLOT

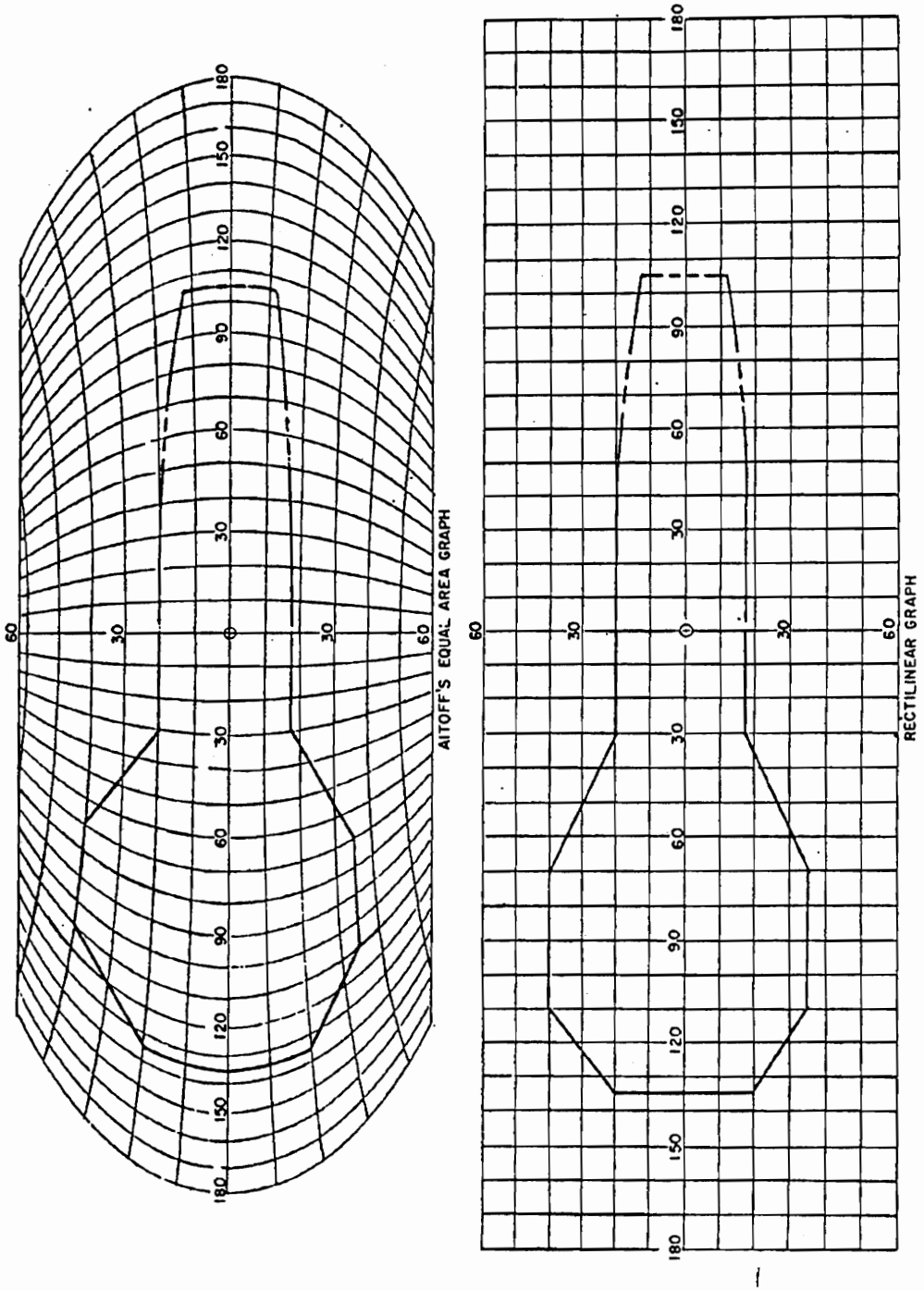


FIGURE 5. SIDE-BY-SIDE PILOT (BOMBER/TRANSPORT) VISION PLOT

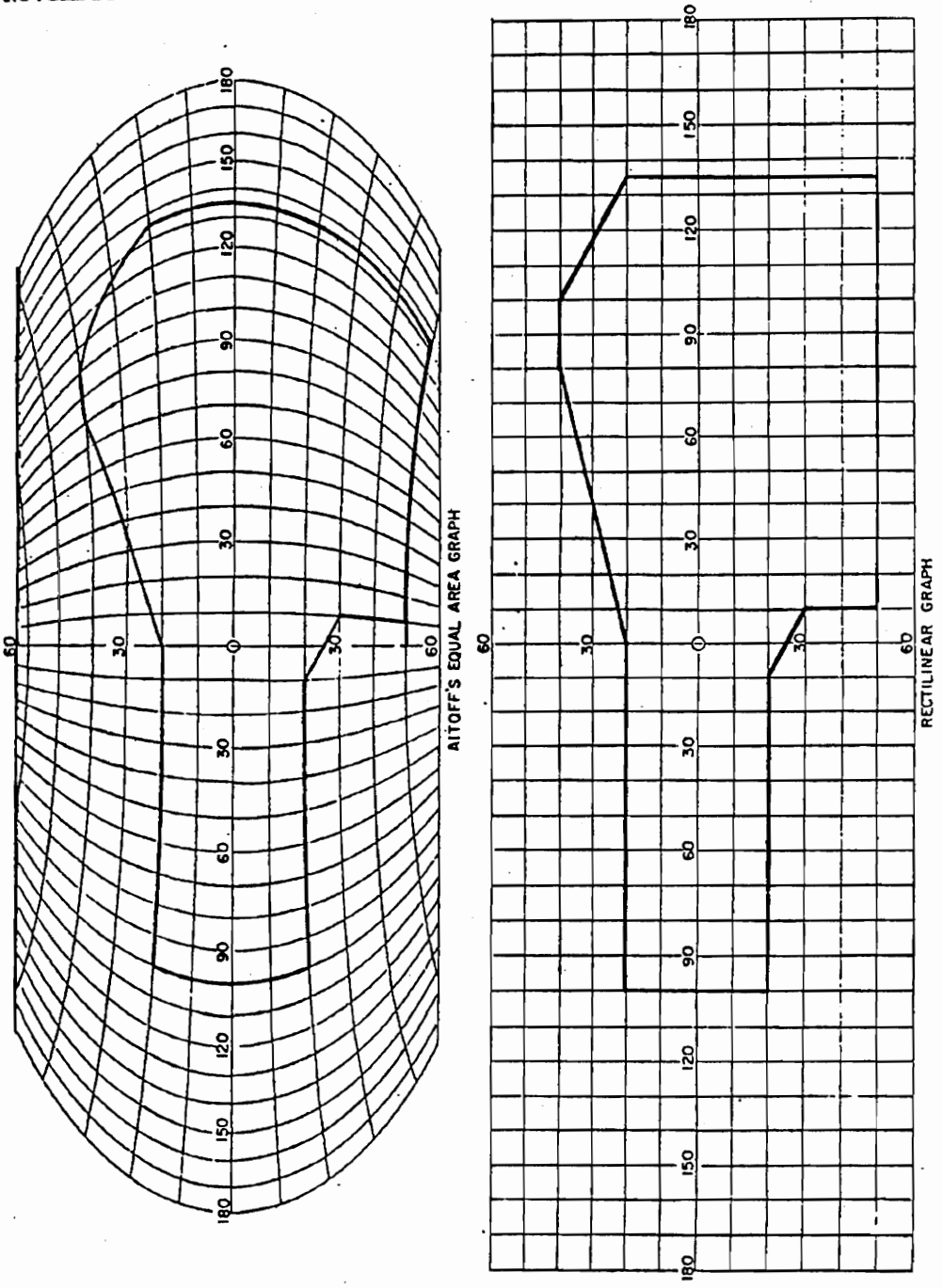


FIGURE 6. SIDE-BY-SIDE PILOT (HELICOPTER) VISION PLOT

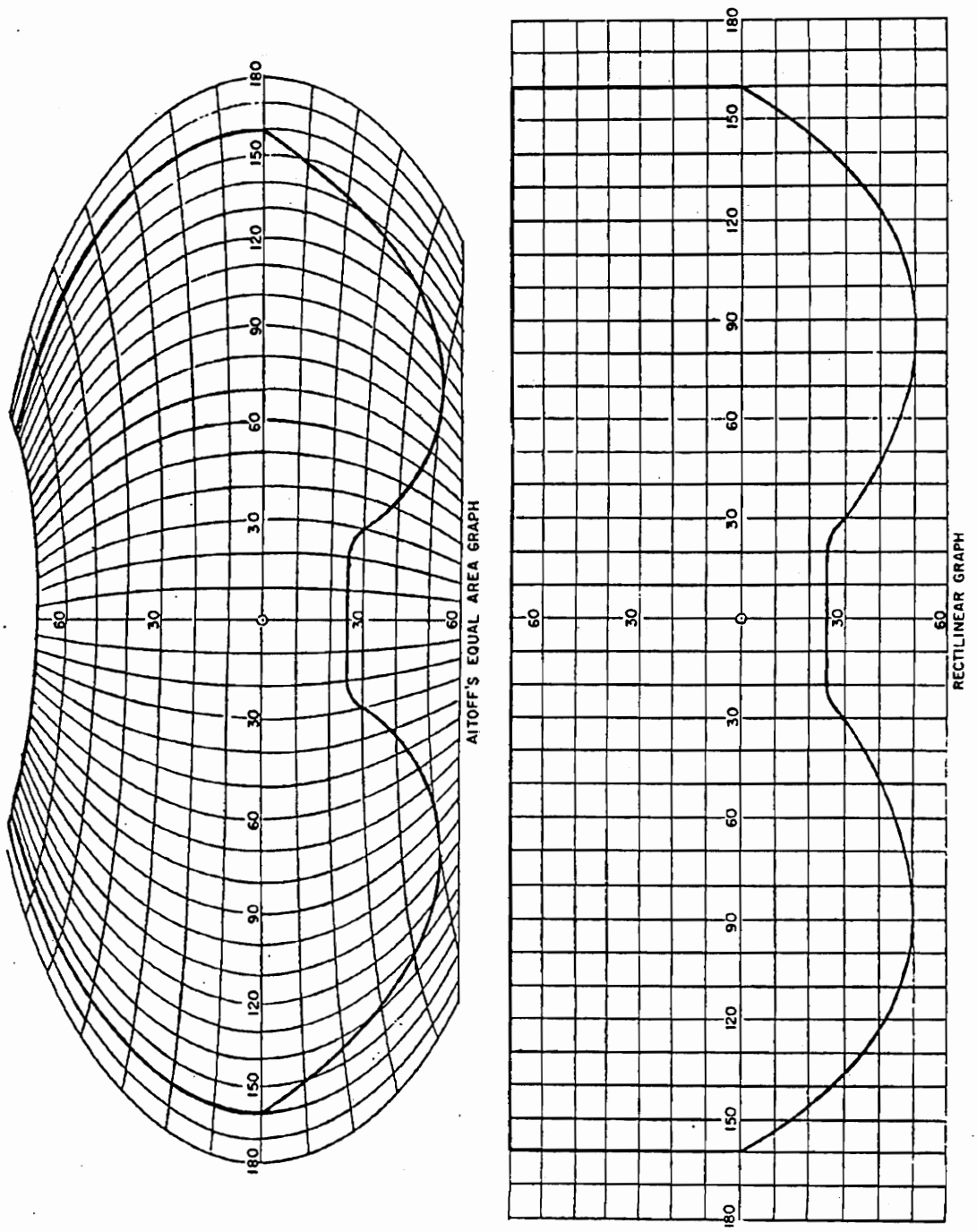


FIGURE 7. SINGLE PILOT/TANDEM PILOT (HELICOPTER) VISION PLOT

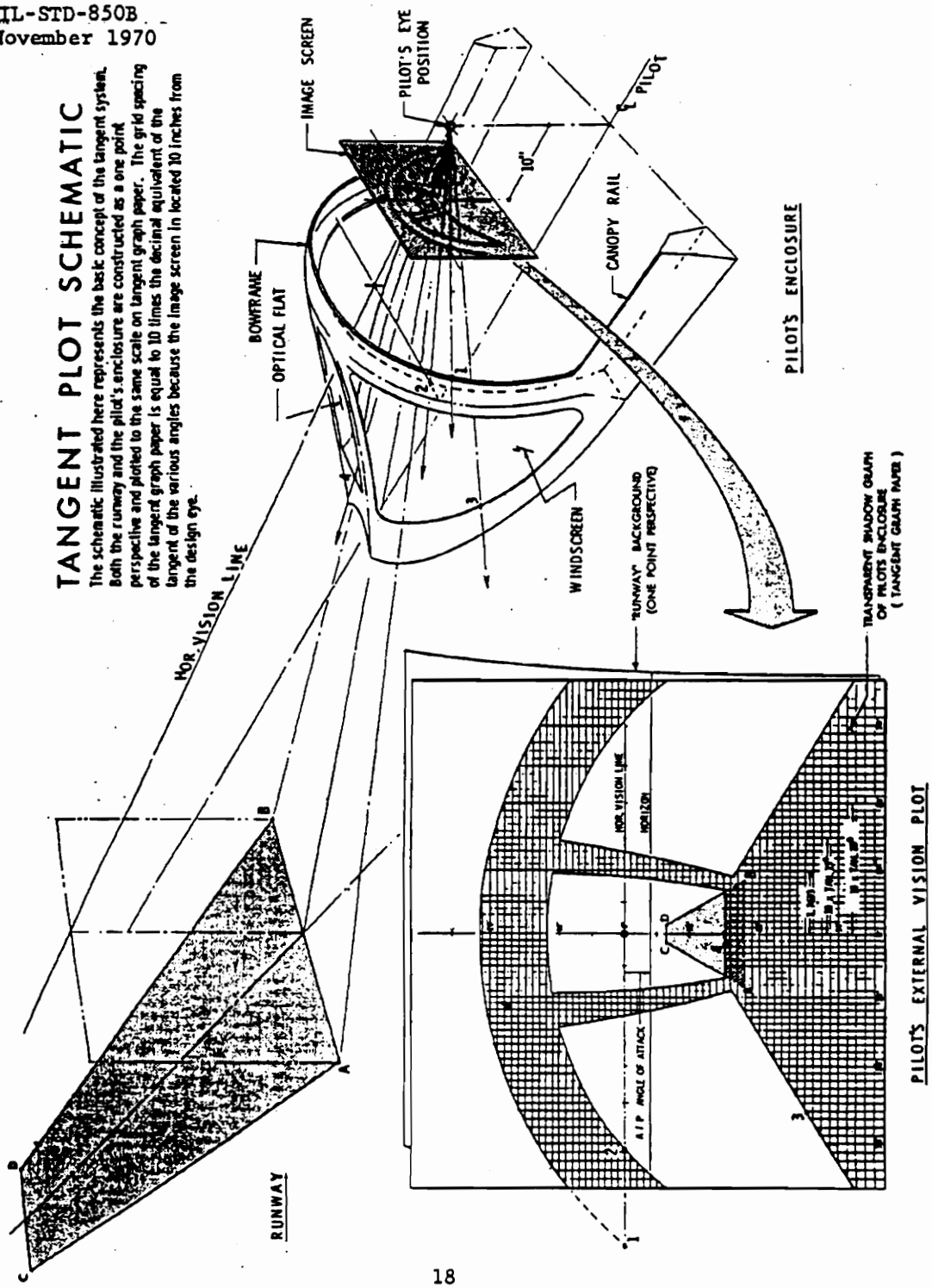


FIGURE 8. SCHEMATIC OF LANDING APPROACH VISION PLOTS

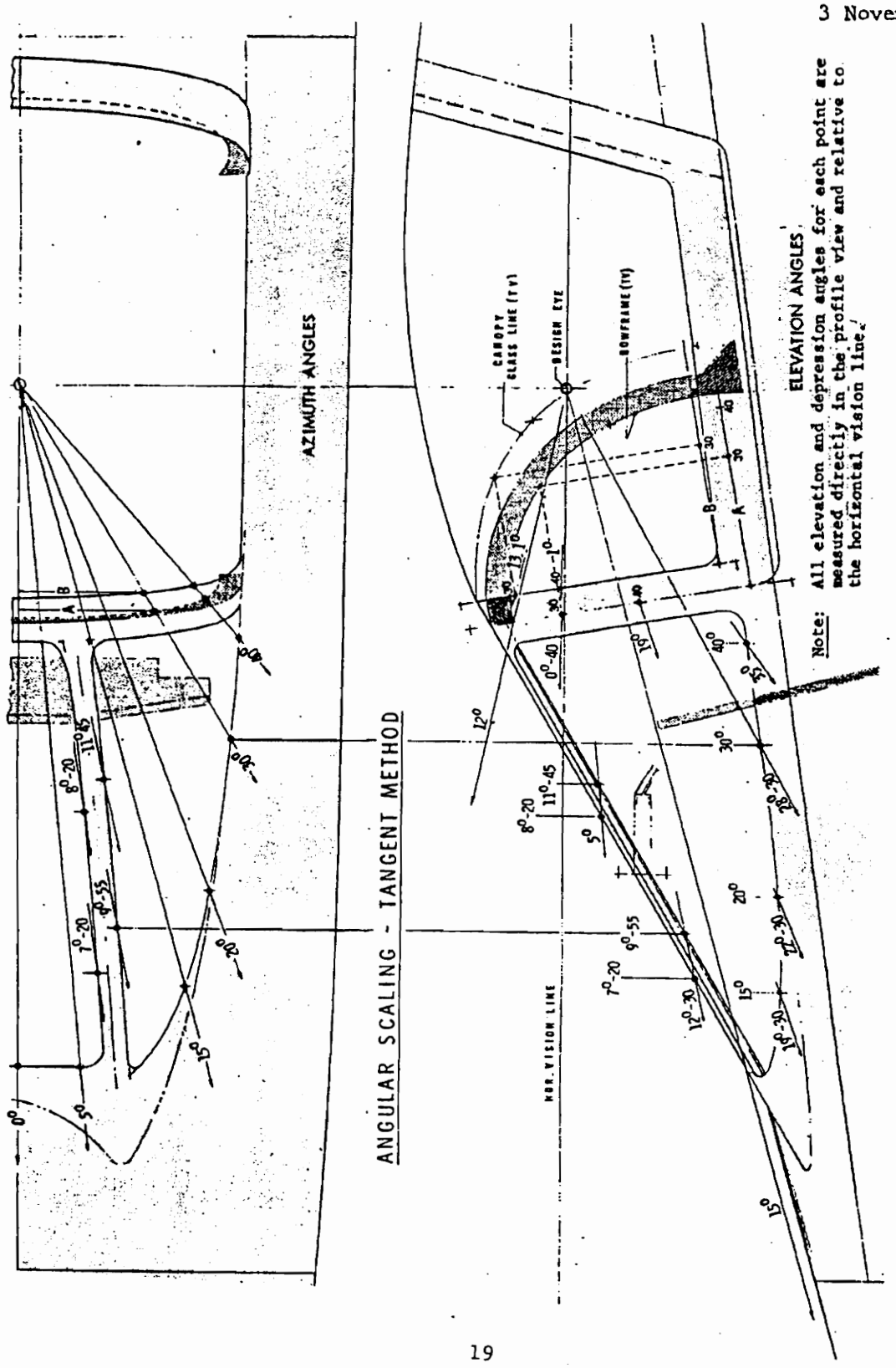


FIGURE 8A. METHOD FOR DETERMINING DATA FOR LANDING APPROACH VISION PLOTS

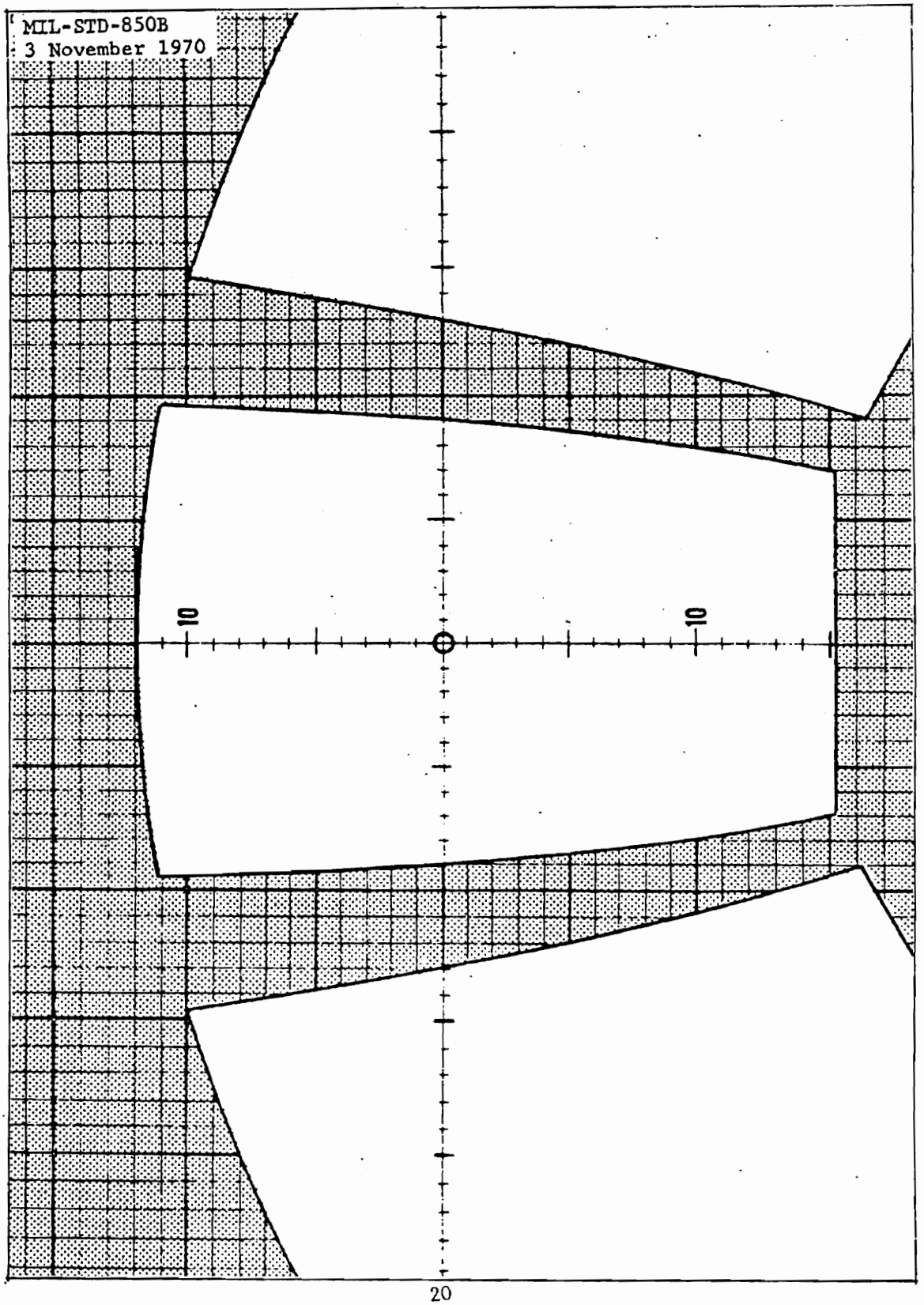


FIGURE 8B. SAMPLE LANDING APPROACH VISION PLOT

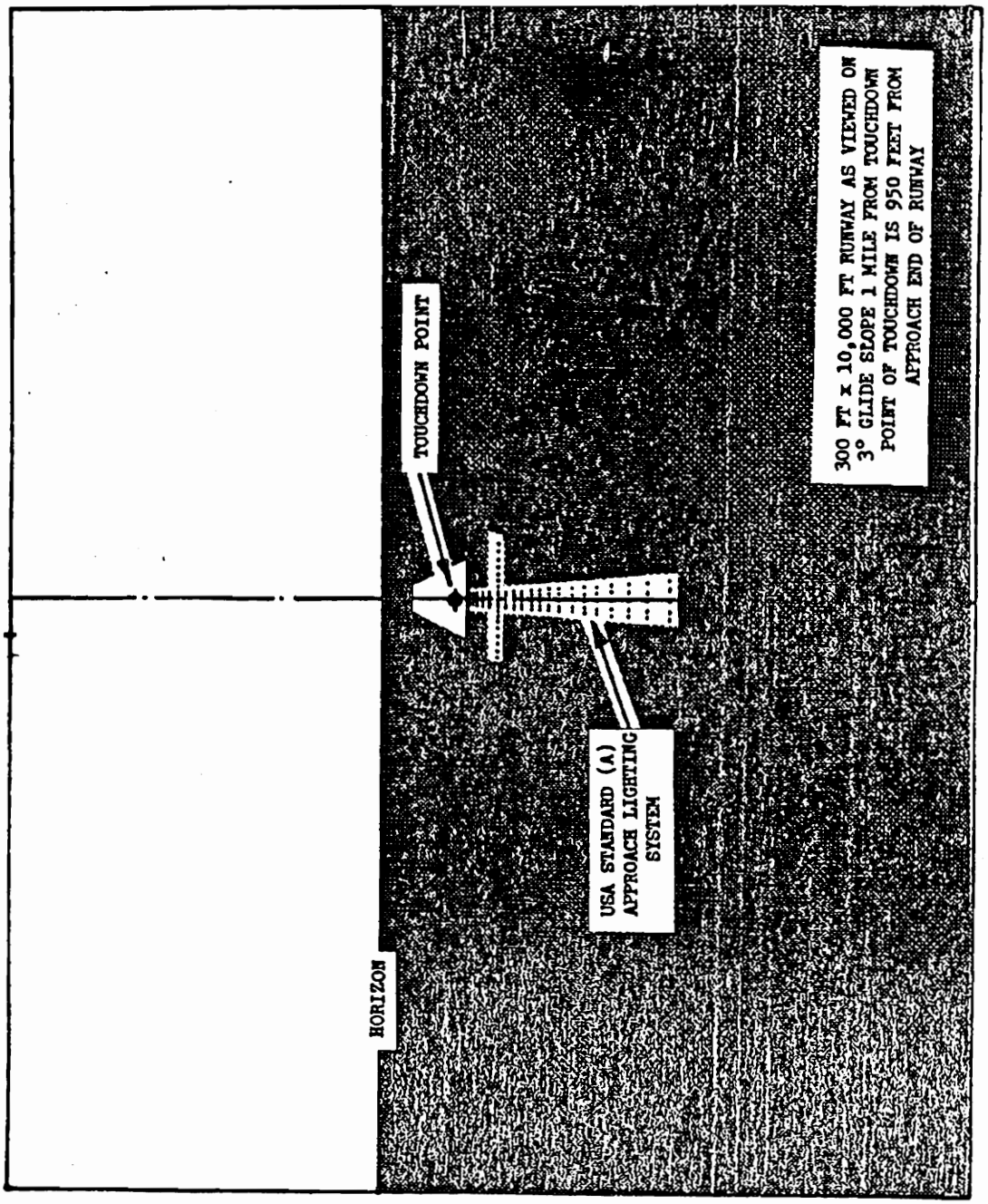


FIGURE 9. FIELD APPROACH AT 1 MILE

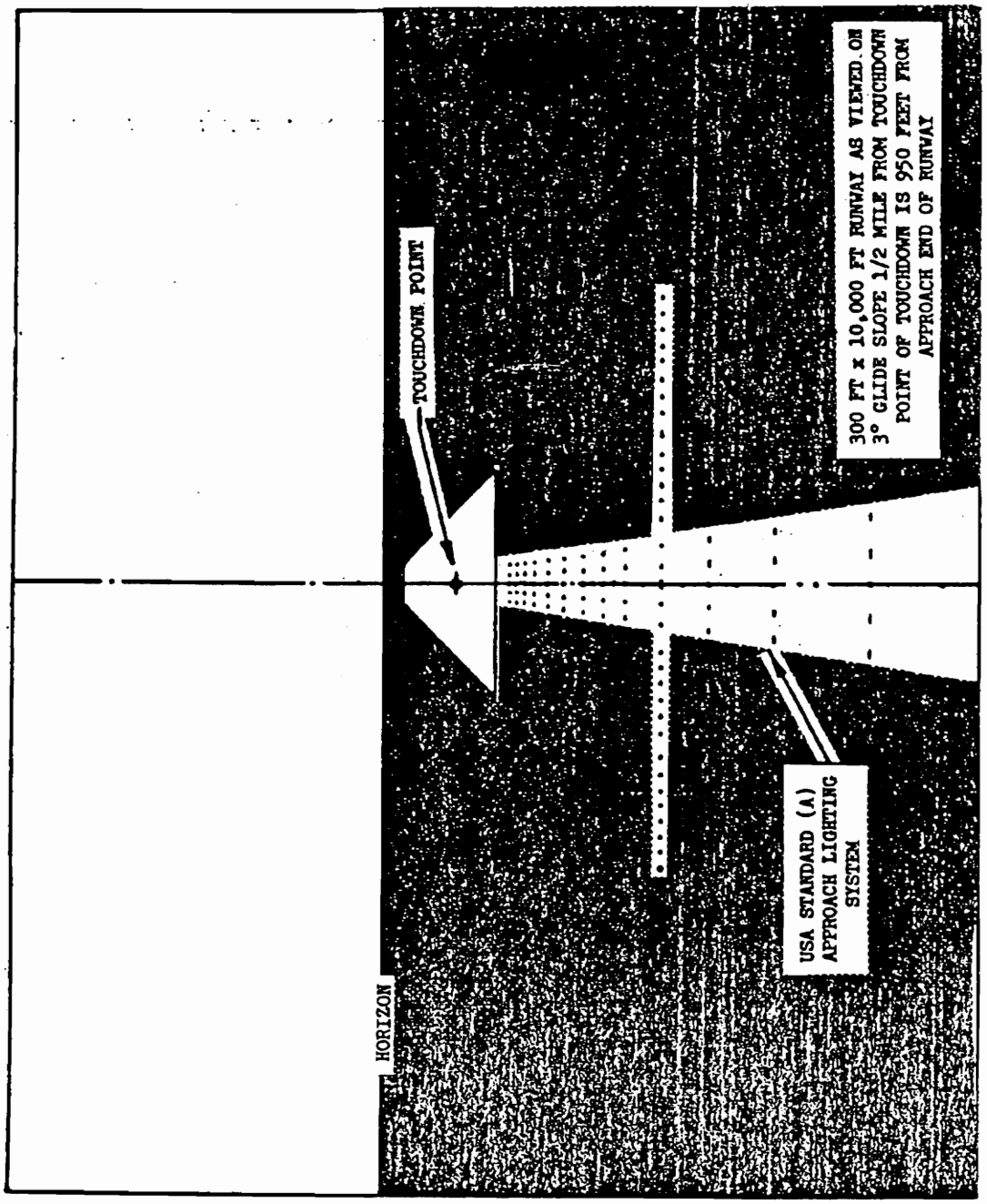


FIGURE 10. FIELD APPROACH AT 1/2 MILE

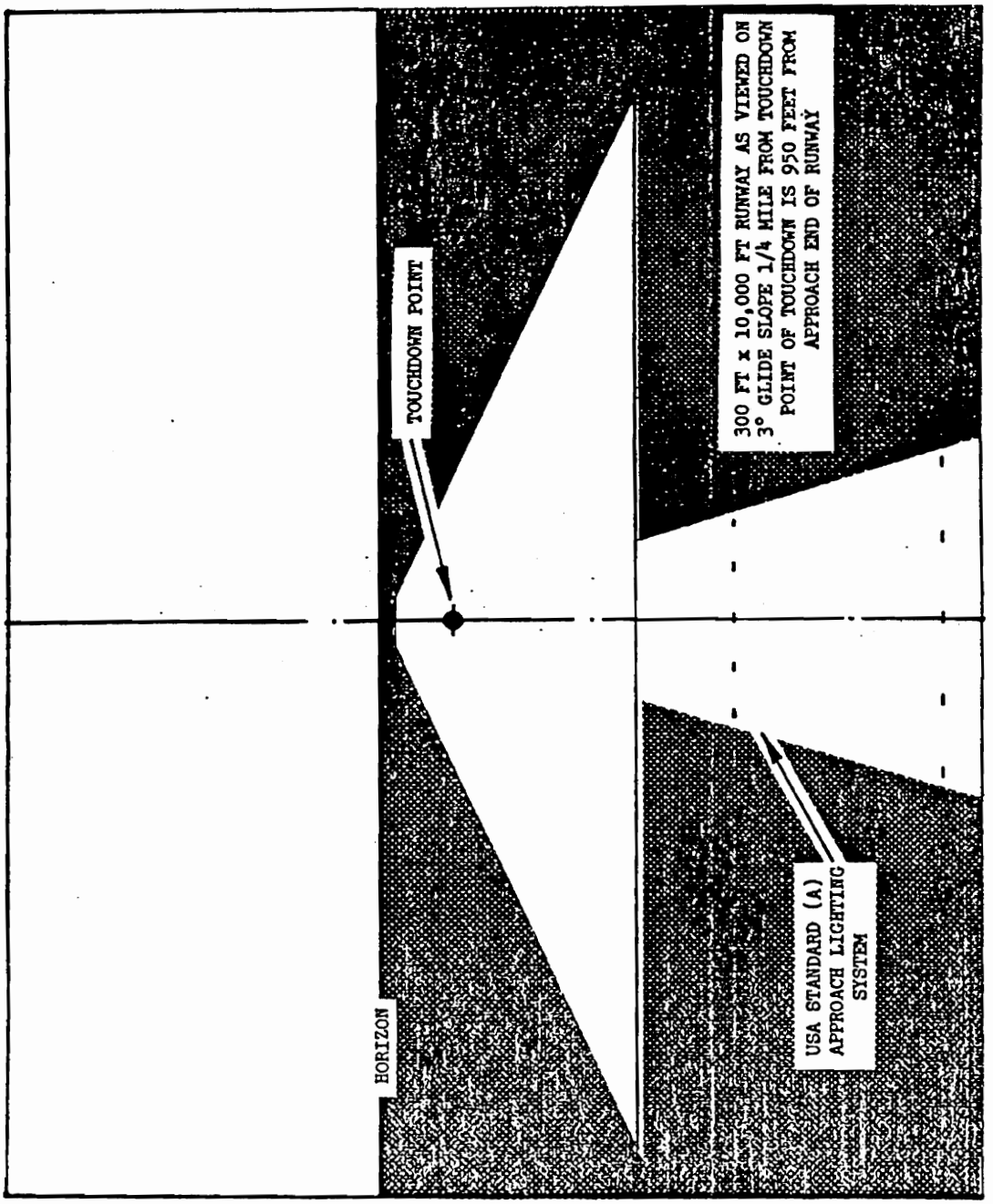


FIGURE 11. FIELD APPROACH AT 1/4 MILE

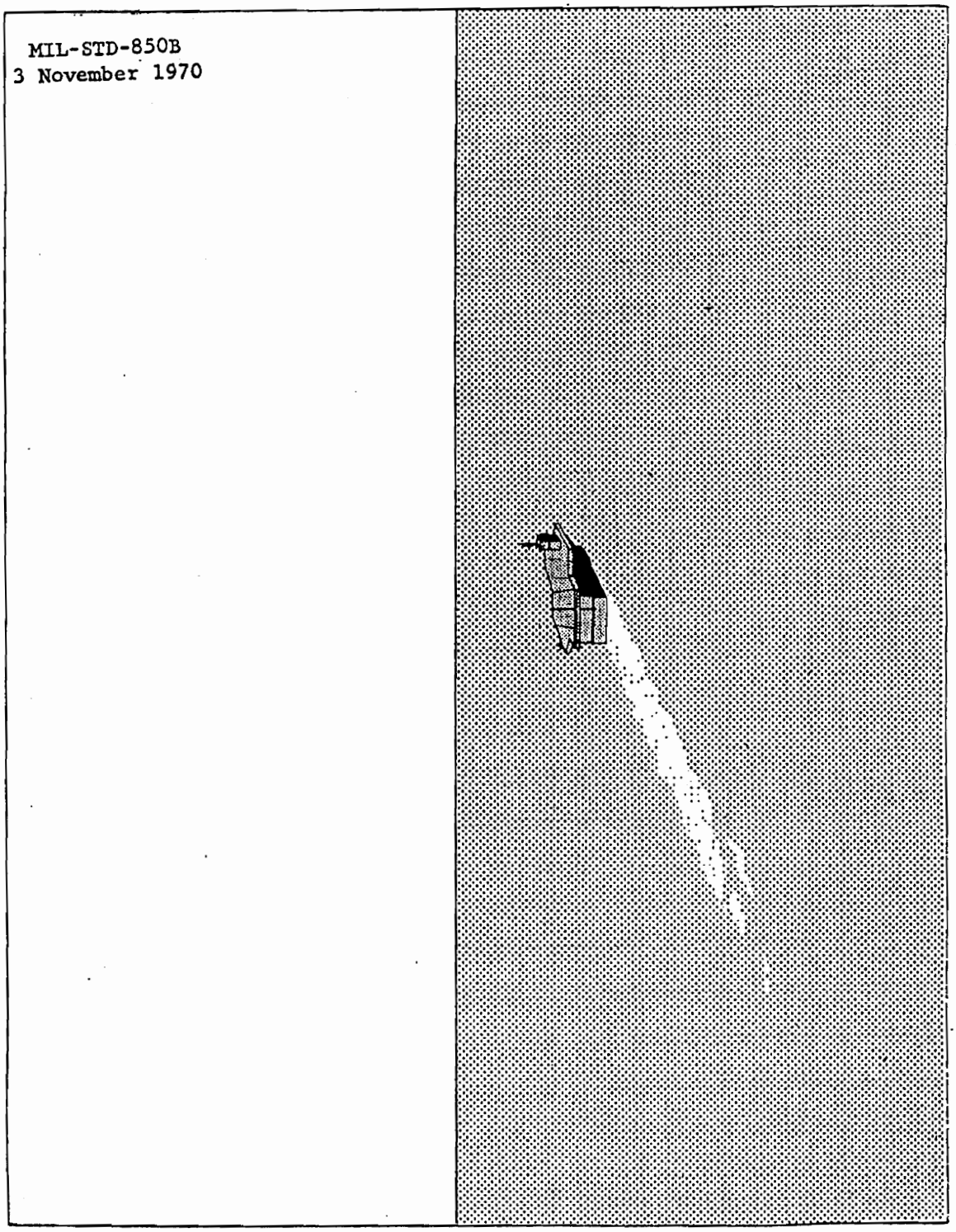


FIGURE 12. APPROACH VIEW - CVA-59 - 4° GLIDE SLOPE - 1/2 MILE ASTERN

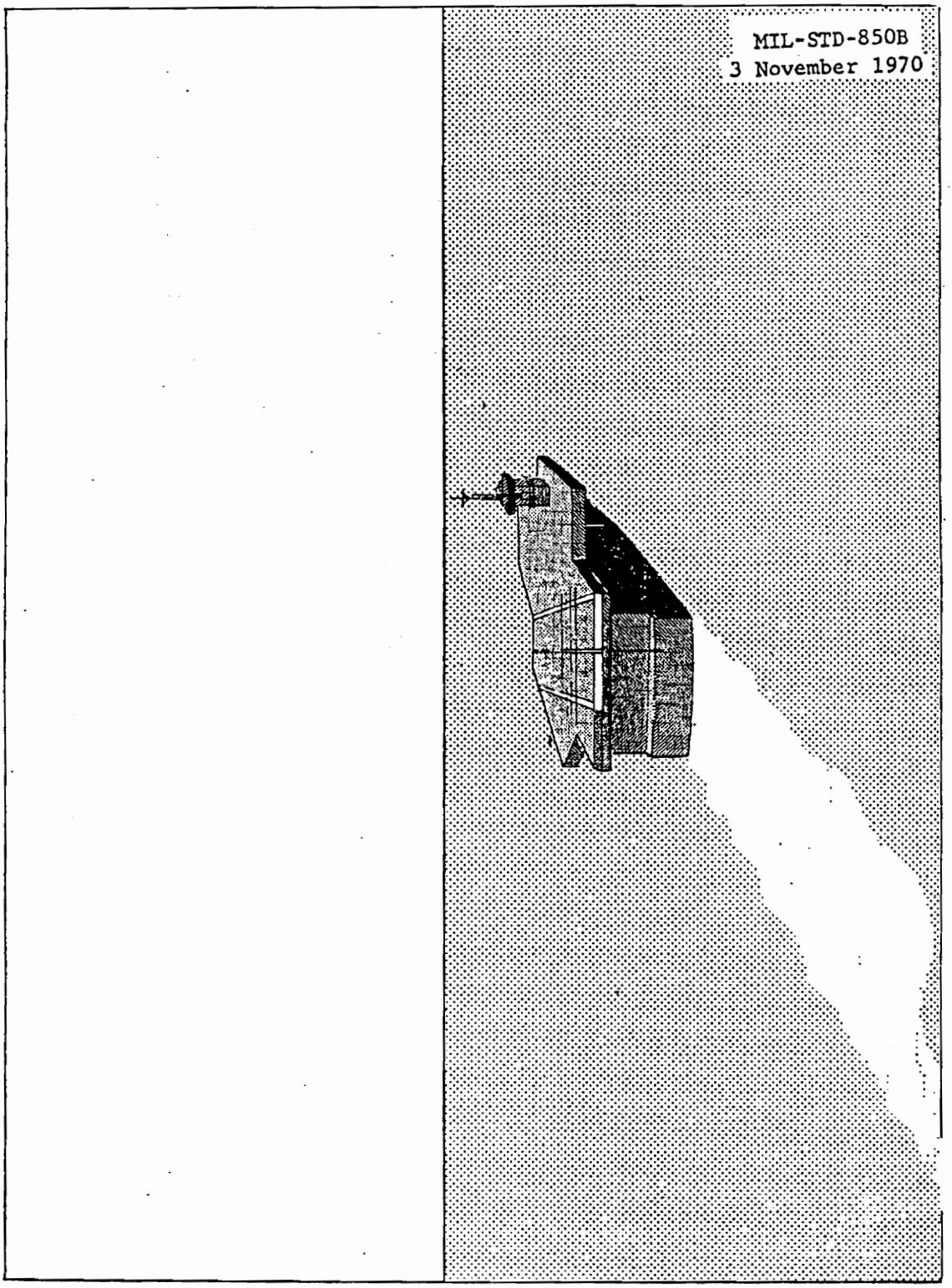


FIGURE 13. APPROACH VIEW - CVA-59 - 4° GLIDE SLOPE - 1/4 MILE ASTERN

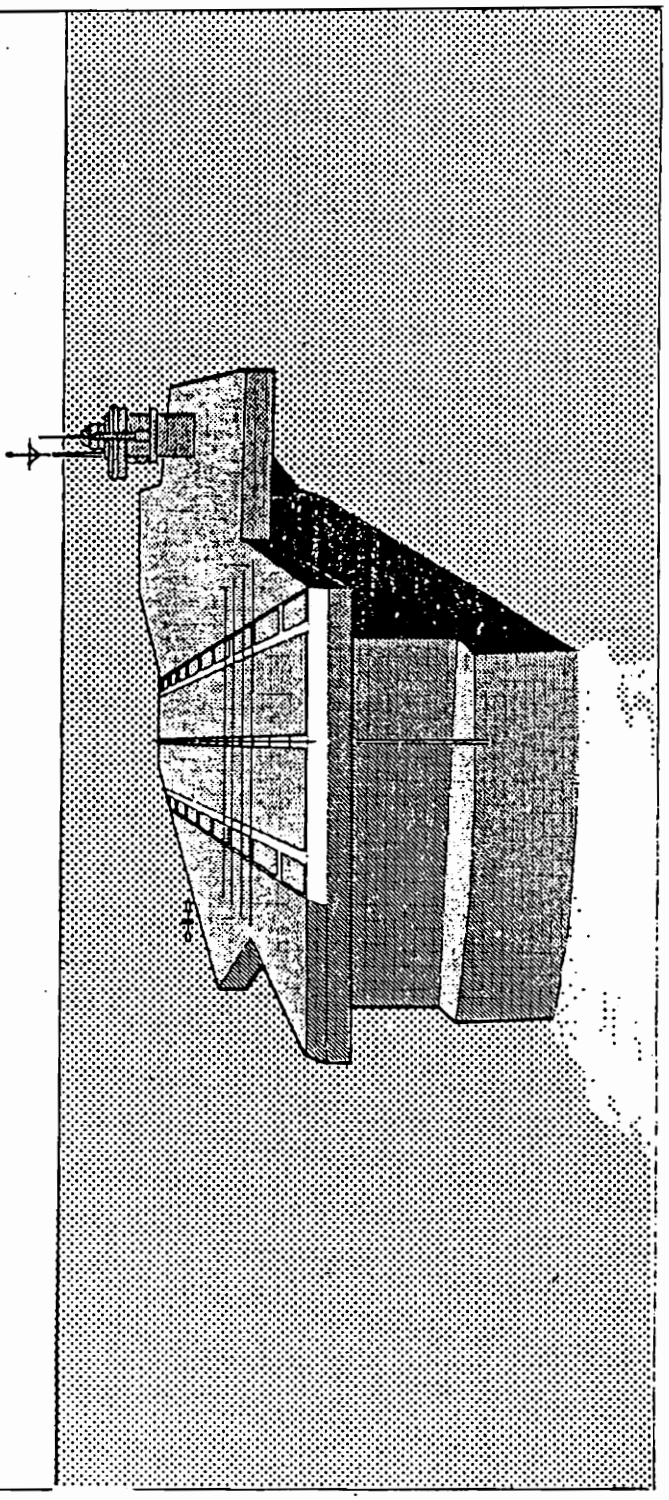


FIGURE 14. APPROACH VIEW - CVA-59 - 4° GLIDE SLOPE - 450 FT ASTERN

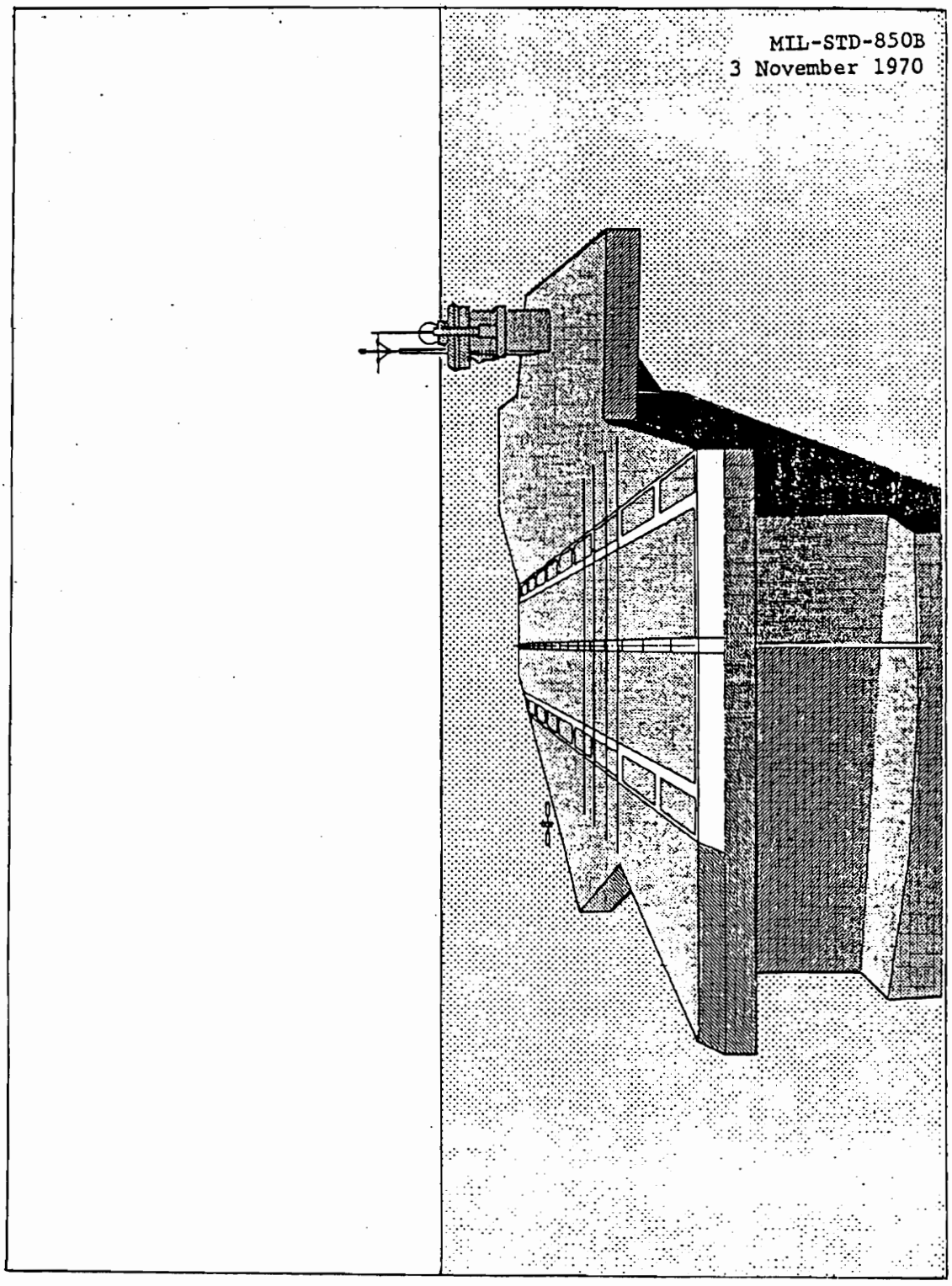


FIGURE 15. APPROACH VIEW - CVA-59 - 4° GLIDE SLOPE - 300 FT ASTERN

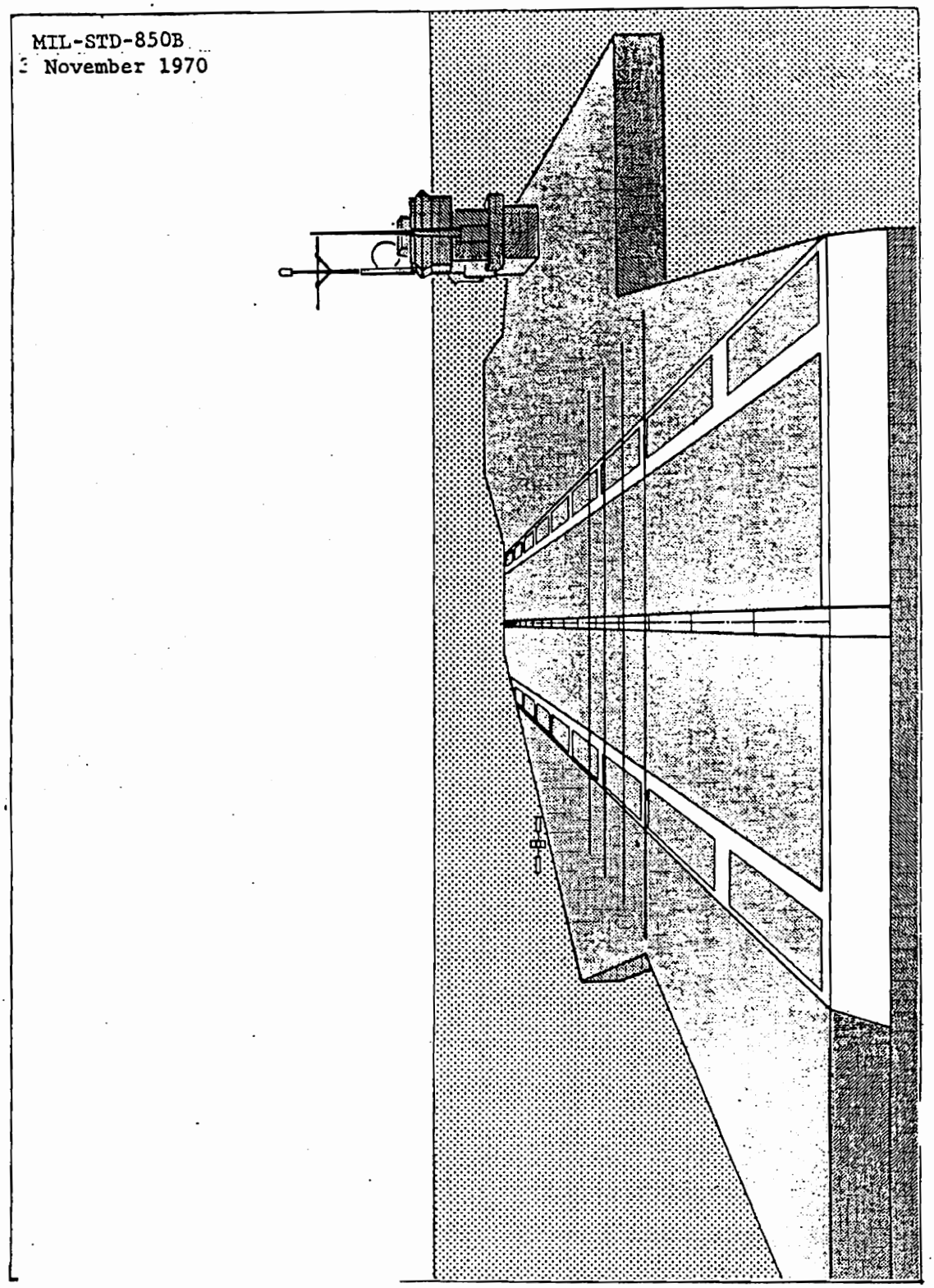


FIGURE 16. APPROACH VIEW - CVA-59 - 4° GLIDE SLOPE - 150 FT ASTERN

Appendix C - PostScript API User's Guide

Note: The following API user's guide was written by Eric Schrock.

Interface Introduction

The PostScript API is a C or FORTRAN-callable set of routines that allow a PostScript file to be written seamlessly across applications. This document describes this API and its use. It is in three parts, an introduction, a programmer's guide, and a source code listing. The listing is for the code as of June 17, 1993.

The source code is currently located in /users2/g553044/postAPI on the HP700 systems. Test cases for FORTRAN and C implementations and their makefiles are also in this directory. As the code is enhanced and modified, updates will be made to this directory.

States:

Before using any of the PostScript API routines, the API must be in an active state. This is accomplished by using the ps_open function. This function must be called before any other ps_* function. It opens the PostScript file, selects the type of strings (FORTRAN or C) that will be passed, and sets device-specific flags. Currently, the device-specific flags are not used in the code, but as differences become apparent in PostScript interpreters, they will be used.

The PostScript file will remain open, and ps_* routines can be called, until the ps_close() function is called. Then the file is closed and the PostScript data structure in the API is emptied. Error messages are reported to stderr during program execution if any rules in the API are violated. The API currently has no method for reporting printer or device errors, so it is best to use an interactive interpreter such as ghostscript (/apps/bin/ghostscript on the HP 700's) to decipher PostScript runtime problems.

Use:

Source code for the API can be linked in a UNIX makefile by supplying its directory path. The functions will link with application code, and the files need not be duplicated. Conventions for passing data between FORTRAN applications and the API are discussed below.

FORTRAN Integer Values:

FORTRAN passes all parameters as pointers. Even though this is not obvious to the average FORTRAN programmer, it's true. Thus, when calling the API from FORTRAN, the %val() function must be used when passing integers. The API expects all integers to be passed as values, and using this function allows matters to proceed normally. As an example, the call:

```
call ps_setdeftrans(%val(300), %val(300))
```

would be used to ensure that the API gets 300 and 300 as the arguments to ps_setdeftrans.

FORTRAN Real Values:

When passing real variable values from FORTRAN, they must be passed by reference (as a pointer). FORTRAN allows this through the use of the %ref() function. Problems with integrating C and FORTRAN on different systems require this to be used. On the function call argument list, use the %ref() function for each real argument. For example:

```
call ps_setgray(%ref(0.25))
```

would be used from FORTRAN to set the grayscale to 25%. Likewise,

```
gscale = 0.25
call ps_setgray(%ref(gscale))
```

uses a variable value to do the same thing.

FORTRAN String Values:

The API is written in C, and C strings are terminated with a null character by default. When FORTRAN is calling API routines, a vertical bar character, or pipe, ("|") should be used to properly terminate the string. The API will look for this character, and ensure that the FORTRAN string is handled properly in C. For example, the call:

```
call ps_setfont(%val(12), 'Helvetical')
```

would set the current font to Helvetica 12.

Using the API from C:

Function call parameters are standardized, because the API was written in C. Neither floats nor integers are passed by reference (as pointers). Strings need no pipe character for termination. The include file ps_c.h must be #included at the top of each C source file calling the API. Placing the line:

```
#include "ps_c.h"
```

will accomplish this.

Sample FORTRAN Application Using The API:

Below is source code for a FORTRAN program that uses the PostScript API to draw a sine curve and assorted text. Notice the passing of parameters by reference and by value. This really makes you want to program in C, doesn't it?

```
C=====================================================================
C                       F M A I N
C=====================================================================
C  PROJECT:  INTERACTIVE CARPET PLOT
C  VERSION:  V1.0.0
C=====================================================================
C COMPILER:  cc (HP 9000/360)
C=====================================================================
C CALLING SEQUENCE:
C     fmain()
C=====================================================================
C INPUT PARAMETERS:
C     NONE
C=====================================================================
C OUTPUT PARAMETERS:
C     NONE
C=====================================================================
C FUNCTIONAL DESCRIPTION:
C     FORTRAN subroutine called from C main program
C=====================================================================
C MODULES CALLED:
C     NONE
C=====================================================================
C CODED BY:  ERIC SCHROCK
C     DATE:  06/16/93
C=====================================================================
C DEVELOPMENT SITE:
C    LOCKHEED AERONAUTICAL SYSTEMS COMPANY
C    ADVANCED DESIGN DEPARTMENT (DEPT 73-06)
C    MARIETTA, GA 30063
C=====================================================================
C234567
```

```
      subroutine fmain()

      character*12 psfile
      real x(37), y(37), pi
      integer i

C     set some variables to test passing of vars and literals
      psfile = 'test.ps'
      ival = 72
      pi = 3.14159265

C     open PS file with FORTRAN style strings
      CALL ps_open(0, 1, psfile)

C     put a comment in the PS file
      CALL ps_comment('Test PostScript API application...|')

C     set a default scale of 72 points per inch that will be applied
C     to primitives (lines/fills/movetos/linetos)
      CALL ps_defscale(%val(ival), 'inch|')

C     set the font to 36 helv
      CALL ps_setfont(%val(36), 'Helvetica')

C     draw 1st message, change gray and draw the 2nd one
      CALL ps_xytext(%ref(1.0), %ref(1.0), 'hello, world|')
      CALL ps_setgray(%ref(0.50))
      CALL ps_xytext(%ref(2.0), %ref(2.0), 'hello again |')

C     make sine wave array if I remember how...
      do 10 i = 1, 37
        x(i) = pi/18*real(i-1)
   10 continue

      do 20 i = 1, 37
        y(i) = 2*sin(x(i))+5.0
   20 continue

C     move to (0.25, 4)
      CALL ps_moveto(%ref(0.25),%ref(4.))

C     set a grayscale and linewidth
      CALL ps_setgray(%ref(0.10))
      CALL ps_setlinewidth(%val(8))

C     draw a fill area with the sine wave
      CALL ps_fill(%val(37), %ref(x), %ref(y))

C     use helvetica 12 font
      CALL ps_setfont(%val(12), 'Helvetica')
```

```
C     move to (3,5)
      CALL ps_moveto(%ref(3.0),%ref(5.0))

C     rotate the text 10 deg from current posn (0 deg)
      CALL ps_rotate(%ref(10.0))

C     draw text
      CALL ps_text('Wowee, a sine wave!!!|')

C     move to (5,5)
      CALL ps_moveto(%ref(5.0),%ref(5.0))

C     rotate 21 more degrees (now 31 deg from horiz.)
      CALL ps_rotate(%ref(21.0))

C     set grayscale and font
      CALL ps_setgray(%ref(0.20))
      CALL ps_setfont(%val(24), 'Times|')
      CALL ps_setgray(%ref(1.00))

C     draw some text
      CALL ps_xytext(%ref(5.0),%ref(5.0),'PostScript is fun!|')

C     rotate back 10 deg (to 21), set grayscale, and draw more text
      CALL ps_rotate(%ref(-10.0))
      CALL ps_setgray(%ref(0.40))
      CALL ps_xytext(%ref(6.0), %ref(2.0),'posers must die!|')

C     advance the page
      CALL ps_showpage()

C     close PS file
      CALL ps_close()

      return
      end
```

Sample C Application Using The API:

Below is source code for a C program that uses the PostScript API to draw a sine curve and assorted text. The output is the same as the FORTRAN program described above.

```
/*  include ps_c.h to substitute float-using PS API calls  */
```

```
#include "ps_c.h"
/*=====================================================================
                        F M A I N
 =====================================================================
   PROJECT:  INTERACTIVE CARPET PLOT
   VERSION:  V1.0.0
 =====================================================================
  COMPILER:  cc (HP 9000/360)
 =====================================================================
  ING SEQUENCE:
      fmain()
 =====================================================================
  INPUT PARAMETERS:
      NONE
 =====================================================================
  OUTPUT PARAMETERS:
      NONE
 =====================================================================
  FUNCTIONAL DESCRIPTION:
      C version of FORTRAN subroutine called from C main program
 =====================================================================
  MODULES ED:
      NONE
 =====================================================================
  CODED BY:  ERIC SCHROCK
      DATE:  06/16/93
 =====================================================================
  DEVELOPMENT SITE:
     LOCKHEED AERONAUTICAL SYSTEMS COMPANY
     ADVANCED DESIGN DEPARTMENT (DEPT 73-06)
     MARIETTA, GA 30063
 =====================================================================
C234567  */

   void fmain()
      {
      char psfile[12];
      float x[37], y[37], pi;
      int i, ival;

/*    set some variables to test passing of vars and literals */
      strcpy(psfile, "test.ps");
      ival = 72;
      pi = 3.14159265;

/*    open PS file with C style strings */
      ps_open(0, 0, psfile);

/*    put a comment in the PS file  */
      ps_comment("Test PostScript API application...");
```

```
/*    set a default scale of 72 points per inch that will be applied
 */
/*    to primitives (lines/fills/movetos/linetos)  */
      ps_defscale(ival, "inch");

/*    set the font to 36 helv  */
      ps_setfont(36, "Helvetica");

/*    draw 1st message, change gray and draw the 2nd one  */
      ps_xytext(1.0, 1.0, "hello, world");
      ps_setgray(0.50);
      ps_xytext(2.0, 2.0, "hello again ");

/*    make sine wave array if I remember how...  */
      for (i=0; i<=36; i++)
        x[i] = pi/18*(i-1)+0.75;

      for (i=0; i<=36; i++)
        y[i] = 2*sin(x[i]-0.75)+5.0;

/*    move to (0.25, 4)  */
      ps_moveto(0.25,4.);

/*    set a grayscale and linewidth  */
      ps_setgray(0.10);
      ps_setlinewidth((8));

/*    draw a fill area with the sine wave  */
      ps_fill(37, x, y);

/*    use helvetica 12 font  */
      ps_setfont(12, "Helvetica");

/*    move to (3,5)  */
      ps_moveto(3.0,5.0);

/*    rotate the text 10 deg from current posn (0 deg)  */
      ps_rotate(10.0);

/*    draw text  */
      ps_text("Wowee, a sine wave!!!");

/*    move to (5,5)  */
      ps_moveto(5.0,5.0);

/*    rotate 21 more degrees (now 31 deg from horiz.)  */
      ps_rotate(21.0);

/*    set grayscale and font  */
      ps_setgray(0.20);
      ps_setfont(24, "Times");
```

```
      ps_setgray(0.33);

/*    draw some text  */
      ps_xytext(5.0,7.0,"PostScript is fun!");

/*    rotate back 10 deg (to 21), set grayscale, and draw more text
 */
      ps_rotate(-10.0);
      ps_setgray(0.40);
      ps_xytext(6.0,2.0,"posers must die!");

/*    advance the page  */
      ps_showpage();

/*    close PS file  */
      ps_close();
      }
```

PostScript Application Programming Interface Programmer's Guide

The Programmer's Guide lists the function calls available in the PostScript API. These calls can be used from FORTRAN or C according to the rules in the Introduction. The following pages show a function call, its purpose, and the argument types. The functions are placed in the following groups:

1. control - opening and closing PostScript API
2. primitives - lines. points, fill areas, etc.
3. attributes - setting grayscales, fill types, rgb colors, etc
4. transformations - rotations, default transformations, "movetos"

Arguments are shown as they would be called from C. Remember that if the API is called from FORTRAN, the integer arguments must be passed by value (%val()) and the real arguments must be passed by reference (%ref()). Float arguments are placed in italics to help FORTRAN programmers remember to pass them by reference. C programmers should remember to #include "ps_c.h" at the top of each source file that uses API calls.

This listing is for the API as of 6/17/93.

Control:

ps_open(ps_type, strings, ps_file_name) Open PostScript API

Arguments:
ps_type -- integer (Apple=0, HP=1, Color PS=2)
strings -- integer (set to 1 if calling the API from FORTRAN, else 0)
ps_file_name -- filename to open that will hold PostScript output

ps_close() Close PostScript API

Arguments:
none

ps_showpage() Print Graphics / Advance Page

Arguments:
none

ps_gsave() Store Current PS Graphics State

Arguments:
none

ps_grestore() Restore Current PS Graphics State

Arguments:

none

ps_comment(text) Place Comment In PS File

Arguments:
text -- pointer to character - text to place in comment

ps_string(mode) Change String Handling Method

Arguments:
mode -- integer, 1 for FORTRAN string handling, 0 for C

Primitives:

ps_beginpath() Start New Graphics Path

Arguments:
none

ps_stroke() Stroke Current Path

Arguments:
none

ps_closepath() Close Current Path

Arguments:
none

ps_line(n, x, y) Draw Line of N points (X,Y)

Arguments:
n -- integer, number of points in this line
x -- real, array of x locations
y -- real, array of y locations
(the real arrays are passed by reference by default - no %ref() is needed from FORTRAN)

ps_cline(n, x, y) Draw Closed Line of N points (X,Y)

Arguments:
n -- integer, number of points in this line
x -- real, array of x locations

y -- real, array of y locations
(the real arrays are passed by reference by default - no %ref() is needed from FORTRAN)
This function draws a line with a closed path - no overhangs or mismatches will occur where the line meets itself.

ps_fill(n, x, y) Draw Fill Area of N points (X,Y)

Arguments:
n -- integer, number of points
x -- real, array of x locations
y -- real, array of y locations
(the real arrays are passed by reference by default - no %ref() is needed from FORTRAN)

ps_rline(n, x, y) Draw Relative Line of N points (X,Y)

Arguments:
n -- integer, number of points in this line
x -- real, array of x locations
y -- real, array of y locations
(the real arrays are passed by reference by default - no %ref() is needed from FORTRAN)

ps_rfill(n, x, y) Draw Relative Fill Area of N points (X,Y)

Arguments:
n -- integer, number of points
x -- real, array of x locations
y -- real, array of y locations
(the real arrays are passed by reference by default - no %ref() is needed from FORTRAN)

ps_text(text) Draw Text at Current Position

Arguments:
text -- pointer to character, text (must be terminated with '|' from FORTRAN)
The text will be drawn using the current placement method (left, right, center - see ps_settextal). This routine demands that the current location be the last thing on the PostScript stack. To prevent errors, it is suggested that the routine ps_xytext() be used instead.

ps_xytext(x, y, text) Draw Text at Current Position

Arguments:
x -- real, x location
y -- real, y location
text pointer to character, text (must be terminated with '|' from FORTRAN)
The text will be drawn using the current placement method (left, right, center - see ps_settextal) .

ps_xytext(x, y, text) Draw Text at Current Position

Arguments:
x -- real, x location
y -- real, y location
text -- pointer to character, text (must be terminated with '|' from FORTRAN)
The text will be drawn using the current placement method (left, right, center - see ps_settextal).

ps_moveto(x, y) Move to a Position

Arguments:
x -- real, x location to move to (can be scaled)
y -- real, y location to move to (can be scaled)

ps_lineto(x, y) Draw A Line

Arguments:
x -- real, x location to draw the line to (can be scaled)
y -- real, y location to draw the line to (can be scaled)

Attributes:

ps_setgray(gray) Set %Grayscale

Arguments:
gray -- real, %grayscale to be applied (warning - 100% gray is white, and 0% gray is black!)

ps_setrgb(r, g, b) Set RGB Color

Arguments:

r, g, b -- real, % of red, green, and blue to be applied

ps_setfont(size, name) Set Current Font

Arguments:
size -- integer, font size in points
name -- pointer to character - name of font ("Times", "Helvetica", etc.)
It is recommended that this function be used to set font size and type simultaneously, instead of a combination of other calls.

ps_setlinewidth(width) Set Line Width

Arguments:
width -- real, width in points (0.24 corresponds to 1/300 inch, the smallest unit on many laser printers

ps_settextal(align) Set Text Alignment

Arguments:
align -- integer, 0 = left, 1 = center, 2 = right
This call sets a flag that justifies all text according to the argument "align". The alignment will continue until the flag is re-set. This is not a feature of PostScript interpreters, but of the API.

ps_setdash(dash_size, dash_space) Set Dashed Linetype

Arguments:
dash_size -- integer, width of dashes in points
dash_space -- integer, space between dashes in points

Transformations:

ps_deftrans(x_org, y_org) Move Origin

Arguments:
x_org -- integer, new origin x location in points
y_org -- integer, new origin y location in points
This function sets a new origin for the PostScript interpreter. The origin starts at the lower left corner of the page, and this function can move it to the middle. For instance, the center of an 8-1/2 by 11 inch page is at (306,396). Calling ps_deftrans(306,396) will allow all further operations to proceed relative to this point.

ps_defscale(pointvalue, name) Set Scale Factor

Arguments:
pointvalue -- integer, amount of scaling in points
name -- pointer to character, name of scale applied
The PostScript API can support one scale factor. It is set using the above call. For instance, if the user wants to use inches as (x,y) locations for drawing text and lines, the call ps_defscale(72, "inch") will cause all further units to be multiplied by 72 (72 points/inch) when lines and text are drawn and located.

ps_rotate(angle) Apply A Rotation

Arguments:
angle -- real, angle in degrees to rotate (positive counterclockwise)
The angles do not accumulate. A call to ps_rotate(20) followed by a call to ps_rotate(30) would place the current angle of rotation at 50 degrees, not 30.

Appendix D - Functional Descriptions

Function:

void pilots_view(MODEL *Model)

Description:

This is the main function for the pilot's view module. The screen layout is set up and the main event loop is entered. This function is called from the surface menu of ACSYNT.

Input Arguments:

MODEL *Model -- pointer to the model data structure

Output Arguments:

none

Function:

void comp_pop_up(Interface_Manager *interface_manager)

Description:

This function is called when the COMPUTE push button is selected. It displays a pop-up menu listing the components to check for intersection.

Input Arguments:

Interface_Manager *interface_manager -- pointer to the window management system

Output Arguments:

none

Function:

void get_data( )

Description:

This function simply calls the existing B-Spline intersection routines within a loop.

Input Arguments:

none

Output Arguments:

none

Function:

void make_three_d_int_line_structure( )

Description:

This function creates a PHIGS structure for the intersection line.

Input Arguments:

none

Output Arguments:

none

Function:

void make_int_line_( )

Description:

This function transforms the intersection data into the equal area and rectangular graph data.

Input Arguments:

none

Output Arguments:

none

Function:

void create_int_line_( )

Description:

This function creates the PHIGS structure for the rectangular graph.

Input Arguments:

none

Output Arguments:

none

Function:

void create_aitoffs_int_line_( )

Description:

This function creates the PHIGS structure for the equal area graph.

Input Arguments:

none

Output Arguments:

none

Function:

void create_axis( )

Description:

This function creates the PHIGS structure for the graph axis.

Input Arguments:

none

Output Arguments:

none

Function:

extern "C" float project_x(float i, float j)

Description:

This function transforms the λ and ϕ coordinates into the x coordinate for plotting on the equal area graph.

Input Arguments:

float i -- λ value

float j -- ϕ value

Output Arguments:

float x -- the value of the x coordinate

Function:

extern "C" float project_y(float i, float j)

Description:

This function transforms the λ and ϕ coordinates into the y coordinate for plotting on the equal area graph.

Input Arguments:

float i -- λ value

float j -- ϕ value

Output Arguments:

float y -- the value of the x coordinate

Function:

void create_aitoffs_grid( )

Description:

This function creates the PHIGS structure for the equal area grid.

Input Arguments:

none

Output Arguments:

none

Function:

void create_bg_fill( )

Description:

This function creates the PHIGS structure for a background fill area.

Input Arguments:

none

Output Arguments:

none

Function:

void int_2_comps_km(Model)

Description:

This function is the main function to calculate the intersection between any two components within the model. Originally coded by Robert Jones and called int_2_comps(Model).

Input Arguments:

Model -- Pointer to the Model data file

Output Arguments:

none

Function:

void draw_int_curve_km(Model)

Description:

This function displays the intersection curve computed in int_2_comps_km. Originally coded by Robert Jones and called draw_int_curve(Model).

Input Arguments:

Model -- Pointer to the Model data file

Output Arguments:

none

Function:

float draw_intersection_km(Model, comp_list, line, curve)

Description:

This function draws the actual intersection line. Originally coded by Robert Jones and called draw_intersection(Model, comp_list, line, curve).

Input Arguments:

MODEL *Model -- Pointer to the Model data file

intersection *comp_list -- component intersection list

line_pts *line -- point list

Output Arguments:

float max_dev -- maximum deviation of intersection line

Function:

void file_pop_up(Interface_Manager *interface_manager)

Description:

This function is called when the SAVE PS push button is selected. It displays a pop-up menu requesting a file name.

Input Arguments:

Interface_Manager *interface_manager

Output Arguments:

none

Function:

void save_ps(char *FILE_NAME)

Description:

This function is called when the SAVE PS push button is selected. It opens, writes to and closes the PostScript file for the pilot's view module

Input Arguments:

char *FILE_NAME -- File name for PostScript file

Output Arguments:

none

Function:

void create_components( )

Description:

This function creates PHIGS structures for the wing and other components to be displayed in the rectangular and equal area plots.

Input Arguments:

none

Output Arguments:

none

Function:

void create_head( )

Description:

This function creates PHIGS structures for the sphere or head that represents the design eye location.

Input Arguments:

none

Output Arguments:

none

Function:

void create_runway( )

Description:

This function creates a PHIGS structure for the runway to be displayed in the simulated landing approach view.

Input Arguments:

none

Output Arguments:

none

Function:

void create_grid( )

Description:

This function creates a PHIGS structure for the rectangular graph grid.

Input Arguments:

none

Output Arguments:

none

Function:

void redraw_graph(int displayed)

Description:

This function will empty the graph structure, create a new set of points based on the new eye position and redraw the graph.

Input Arguments:

int displayed -- type of graph presently displayed

Output Arguments:

none

Function:

SUBROUTINE INTRIOR( )

Description:

This function begins the pilot's view module by calling the "C" function pilots_view().

Input Arguments:

none

Output Arguments:

none

Vita

Kerry Stephen McClure was born November 19, 1969 in Charleston, West Virginia. It is rumored that the first thing he saw was a television set in the delivery room showing the second landing on the moon and this sparked his interest in space. From a very early age, he was destined to one day become an astronaut. Through middle and high school, he showed an interest in pursuing an engineering degree. He graduated in the top ten from high school and decided on enrolling at Virginia Tech in aerospace engineering. While in college, he joined the Virginia Tech Solar Car team so that he could be exposed to the entire design process. After graduating as Cum Laude and Commonwealth Scholar in May 1992 with a degree of Bachelor of Science in Aerospace Engineering, he immediately began graduate School in the Mechanical Engineering Department under Dr. Arvid Myklebust. After graduating, he plans on entering the work force with one goal in the back of his mind, to eventually explore Space.

Kerry S. McClure