

AD-A198 484

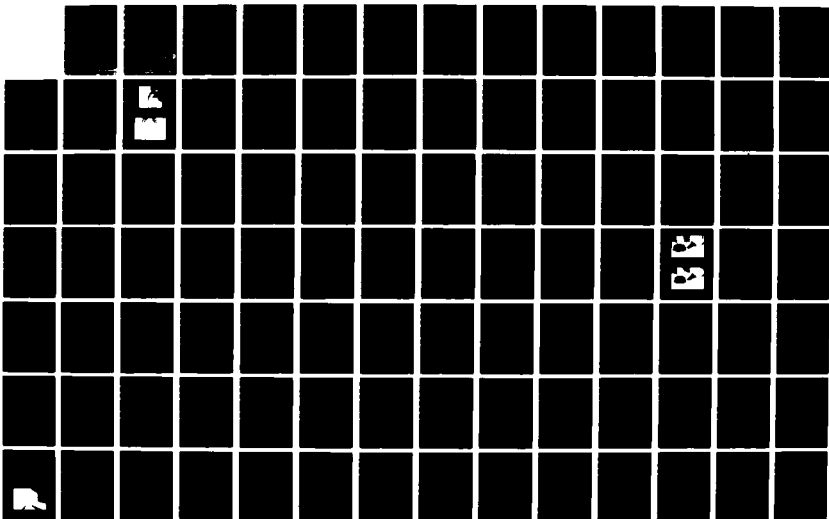
GEOMETRIC MODELING OF FLIGHT INFORMATION FOR GRAPHICAL
COCKPIT DISPLAY(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
M A KANKO DEC 87 AFIT/GCE/ENG/87D-6

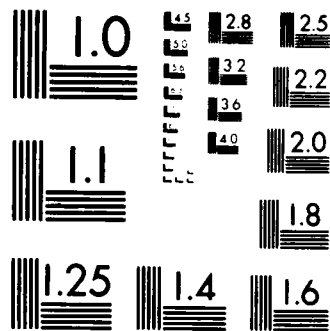
1/2

UNCLASSIFIED

F/G 1/4

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A190 484



DTIC FILE COPY

GEOMETRIC MODELING OF FLIGHT
INFORMATION FOR GRAPHICAL
COCKPIT DISPLAY

THESIS

Mark A. Kanko, B.S.
Captain, USAF

AFIT/GCE/ENG/87D-6

"Original contains color
plates: All DTIC reproduction
ions will be in black and
white"

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

DTIC
ELECTE
MAR 28 1988
S D
E

Wright-Patterson Air Force Base, Ohio

This document has been approved
for public release and sale in
distribution to subscribers

88 3 24 0411

AFIT/GCE/ENG/87D-6

GEOMETRIC MODELING OF FLIGHT
INFORMATION FOR GRAPHICAL
COCKPIT DISPLAY

THESIS

Mark A. Kanko, B.S.
Captain, USAF

AFIT/GCE/ENG/87D-6

"Original contains color
plates: All DTIC reproductions
will be in black and
white"

DTIC
ELECTE
MAR 28 1988
S E D

Approved for public release; distribution unlimited

AFIT/GCE/ENG/87D-6

GEOMETRIC MODELING OF FLIGHT INFORMATION
FOR GRAPHICAL COCKPIT DISPLAY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering



Mark A. Kanko, B.S.
Captain, USAF

December 1987

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution unlimited

Dedication

In loving memory of my father, Elvin Kanko,
whose lessons in patience, persistence, and
attention to detail I hope to reflect in
this work.

Acknowledgements

I would like to thank my advisor, Major Phil Amburn, for his enthusiasm, fresh ideas, and continuous encouragement during this effort. His words of correction were never once critical, but always constructive.

Next I'd like express my thanks to the people at Armstrong Aerospace Medical Research Laboratory who provided the Silicon Graphics IRIS 3130 computer in support of graphics research at AFIT.

I owe my deepest thanks to my wife, Annette, for unfailing support throughout our time at AFIT and especially during my thesis effort. Together with our children, she provided a sense of joy that always encouraged.

Finally, I am grateful to and for the source of all knowledge:

For the Lord gives wisdom; From His
mouth come knowledge and understanding.

Proverbs 2:6

Mark A. Kanko

Table of Contents

	Page
Dedication	ii
Acknowledgements	iii
List of Figures	vii
Abstract	viii
 I. Introduction	 1
Background	1
Problem Statement	3
Scope	3
Assumptions	5
General Approach	6
Presentation	7
 II. Literature Review	 8
Overview	8
Flight Simulator/Cockpit Threat Displays	8
Procedure Modeling Techniques/Issues	9
Geometric Database Concepts and Issues	11
Conclusion	15
 III. System Requirements	 16
General User Requirements	16
Specific Requirements	18
User Interface	18
Hardware	19
Software	19
 IV. System Theory and Design	 20
Overview	20
General Design Considerations	21
Modeling vs. Display	21
Normalization	22
Data Representation Efficiency	22
Model Type	23
Coordinate System Conventions	24
Modeling Process Overview	24
Concept of the Model	26
Implemented Abstraction Levels	28

Data Representation of the Model	29
Data Representation 1A (DR 1A)	30
DR 1A Composition	31
Data Representation 1B (DR 1B)	31
DR 1B Composition	32
Object Generation with Procedural Models	32
Data Representation	36
DR 2A Composition	36
DR 2B Composition	37
Smooth Surfaces, Light, and Shading	37
Simulating Light	38
Gouraud Shading	40
Summary	42
V. System Implementation	43
Overview	43
Modeling Environment	43
Hardware, Software, and Firmware	43
Working Directory and File Conventions	46
Data Representations Implementation	47
Model: A Profile and Object Modeling Tool	48
Function	48
User Interface	48
Program Operation	49
Layout: A Tactical Situation Modeling Tool	53
Function	54
Modeling Session Preparation	54
Commandline Interface	55
Graphical User Interface	56
Menu Functions	57
Main Menu	57
Ops Menu	57
Program Operation	63
See: A Model Viewing Tool	65
Function	65
User Interface	65
Menu Functions	67
(L)oad model	67
(M)odify settings	67
(D)isplay	68
(Q)uit	68
Program Operation	68
Summary	70
VI. Conclusions and Recommendations	72
Results	72
Conclusions	73
Recommendations	73
Software Enhancements	73
Further Research	74

Appendix A: Additional Software Tools	76
Appendix B: Model Building Session Example	79
Bibliography	93
Vita	96

List of Figures

Figure	Page
1 Primary Flight & Tactical Data	4
2 Vertical Situation Display	4
3 Example Threat Locations	19
4 System Structure	25
5 Procedural Model Examples	34
6 Flat Shaded Image	41
7 Gouraud Shaded Image	41
8 "Model" Main Menu	49
9 Profile Input Example	50
10 Surface of Revolution Example	51
11 Profile Data Format: File ./prof/example.prof . .	52
12 Polygon Data Format: File ./poly/example.poly . .	52
13 Menu/Function Hierarchy	58
14 "See" Main Menu	66
15 Example Model Image	72

Abstract

The purpose of this investigation was to design and implement a graphics-based environment capable of modeling tactical situation arenas as viewed from the cockpit. The modeled arena or region was composed of mountains, hostile threat envelopes, and a projected flightpath through the region. The resulting displays were to be used in the Microprocessor-Based Application of Graphics Interactive Communication (MAGIC) Cockpit owned by the Crew Systems Development Branch within the U.S. Air Force Flight Dynamics Lab at Wright-Patterson Air Force Base. This cockpit is used to prototype new graphical display formats that might be used in future aircraft.

The individual three-dimensional objects used to represent threats and mountains in the model were generated by geometric procedural models. A strongly-parameterized procedural model would generate a three-dimensional surface of revolution composed of polygons from a two-dimensional profile input by the user. Once defined, each object could then be instantiated into the model representing the complete tactical situation. Positioning of objects in the model was accomplished via a mouse input device.

The implemented data representation allowed the model to be easily modifiable. Additionally, the model could be stored in a machine-independent form to assure portability.

An overall goal of this investigation was to allow the cockpit display researcher to create an entirely new tactical situation display model in less than one hour.

The applications comprising the modeling environment were written in the C programming language and were hosted on a Silicon Graphics IRIS 3130 graphics workstation.

GEOMETRIC MODELING OF FLIGHT INFORMATION FOR GRAPHICAL COCKPIT DISPLAY

I. Introduction

Background

As aircraft flight and weapon systems get more complex, the pilot's job gets more difficult as well, especially when in a hostile environment. For this reason, efficient methods of data presentation must be developed so the pilot can more easily comprehend, but not be overwhelmed by the information he must process to successfully complete all phases of a mission.

Much of the systems and mission status information needed by the pilot to perform a mission is currently presented in alphanumeric formats. A more efficient presentation method is pictorial formats.

In 1980, the U.S. Air Force Flight Dynamics Laboratory (AFWAL/FI) at Wright-Patterson AFB began investigating methods to compress much of the data needed by the pilot into graphical formats suitable for cockpit display. The research was initially based on computer graphics portrayal ideas which were first implemented in the Navy F-18 aircraft [12:3].

A major tool currently used by the Crew Systems Development Branch (AFWAL/FIGR) to explore new display techniques is the Microprocessor-Based Application of

Graphics and Interactive Communication (MAGIC) Cockpit. This cockpit is used to prototype new graphical display formats that might be used in future cockpit designs. Display formats containing flight and navigation information, system advisory and status information, and tactical situation information are generated and displayed to pilots in a dynamic mission simulation. Afterwards, the pilots are asked to assess the effectiveness of the individual displays.

A key factor in a tactical situation display is the way a hostile ground threat is displayed. Research under government contract by McDonnell Douglas Corporation [10] and Boeing Military Aircraft Company [16] has dealt with how these threats might look. Research has not, however, dealt with how models of these threats might be easily created for display generation in a fast-prototyping environment like the MAGIC Cockpit.

Prior to this design effort, the graphic primitives needed to generate a tactical situation model containing terrain, flight path, and threat information were programmed into the software that generated displays in the MAGIC Cockpit. The model could not be easily change. Further, as the number of objects in the model increased, the amount of time consumed to build the model and the chance of errors being introduced both increased. A software application that would allow a non-programmer to generate a tactical

situation model of varying resolution or complexity in less than one hour was needed.

Problem Statement

The purpose of this research was to design and implement a computer graphics-based software application that would allow the user to interactively create and modify a three-dimensional tactical situation model for display in a dynamic simulator environment. Further, the application had to be capable of generating models that looked more like the 1981 McDonnell Douglas artist conceptions (Figure 1) than the 1984 Boeing results (Figure 2). Additionally, the resulting software procedures had to facilitate creation, modification, and storage of the model representation in a machine-independent form to assure portability. The overall goal of the application was to allow a display researcher to create an entirely new tactical situation display model in less than one hour.

Scope

This research effort dealt with the creation of tactical situation display models required by the Crew Systems Development Branch within the Flight Dynamics Lab. As such, the software application was targeted for a Silicon Graphics IRIS 3130 Workstation owned by that organization.

Development of this application required investigation into three areas:

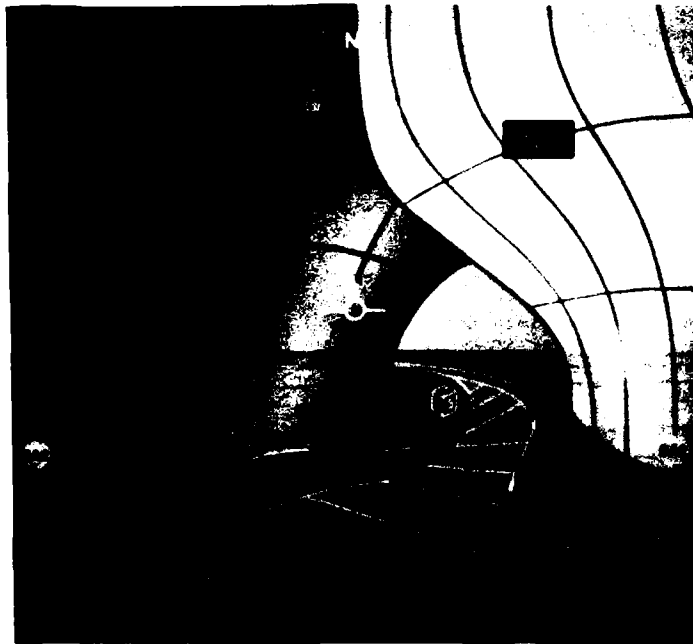


Figure 1. Primary Flight & Tactical Data [10:51]



Figure 2. Vertical Situation Display [16:15]

- 1) Methods proposed and/or used previously to generate threat representations in other flight simulator or cockpit mockup environments.
- 2) The use of geometric procedural models to generate three-dimensional threat representations.
- 3) The selection of a geometric database representation for the tactical situation model.

Assumptions

The MAGIC Cockpit is used to evaluate displays including system advisory and status, flight and navigation, and tactical situation displays. The focus of this thesis research concentrated only on the last of the three. As such, the user interface to application was optimized for that purpose.

The user of this system was viewed as a cockpit display researcher interested in interactively manipulating the model of some tactical environment. Manipulating the model consisted of the creation, dynamic positioning, deletion, and viewing of graphical objects that represented terrain (mountains), ground threats and the projected flightpath that would be followed by the aircraft. A flightpath displayed in this manner is sometimes referred to as a "flightpath in the sky."

The target system that was to host this application used the C programming language. Hence, all software development and implementation was done in C.

General Approach

After the three subject areas covered in the Scope were adequately researched, software development began using a software prototyping design approach. When initial software development began, an IRIS graphics workstation was not yet available for prototyping. Hence initial development, designated "Version 1.0", was accomplished on a Sun Microsystems 360/3 graphics workstation located at the Air Force Institute of Technology (AFIT). Although this was not the target system for the final implementation, it was a good environment to test the basic capabilities that would exist in the final system. Capabilities present in Version 1.0 included basic terrain creation, threat creation, and threat manipulation. All calls to graphics library routines were isolated to a single subroutine during this development so that transfer to the target system would be easier.

After three months of development work, an IRIS workstation was delivered to AFIT by Armstrong Aerospace Medical Research Lab (AAMRL) in support of AFIT computer graphics research. All C source code was then transferred to the 3130. This constituted the beginning of Version 2.0 Development.

Software development was considered complete when the following capabilities were demonstrated:

- 1) Three-dimensional surfaces of revolution representing threats could be generated from two-dimensional profiles; the number of sectors per revolution could be specified by the user.

- 2) Interactive graphical manipulation of mountains, ground threats, and flight path in a two-dimensional horizontal situation display format.
- 3) Flat and smooth (Gouraud) shaded images generated from the same database and displayed in a vertical situation display format.
- 4) Semi-transparent display of threats.

Presentation

This thesis is made up of six chapters and two appendices. Chapter II presents a survey of the literature in each of the three areas mentioned in the Scope. Chapter III covers the user requirements that drove the design of the application during the development phases. System theory and design are discussed in Chapter IV, while Chapter V covers the system implementation. Finally, conclusions and recommendations are considered in Chapter VI. Appendix A describes a number of additional software tools developed during this effort. Appendix B takes the reader through all the steps required to construct a tactical situation model.

II. Literature Review

Overview

There were three separate areas of interest that had to be researched before any design activities could begin. These areas were: flight simulator/cockpit graphical displays used for threat representation, procedure modeling techniques, and geometric database concepts and issues. Each of these areas is reviewed in the following sections.

Flight Simulator/Cockpit Threat Displays

Beginning in 1980, the Naval Air Development Center and the Air Force's Armament and Flight Dynamics Laboratories began to pursue a program utilizing color computer graphics concepts for a number of cockpit displays including tactical situation displays [12:3]. Under a 13-month government contract beginning in May 1980, McDonnell Douglas Corporation developed artists' conceptions of various cockpit displays. This study intentionally neglected the then-current technology and budget constraints associated with the implementation of any of the displays. Further, the contractor was instructed to rely primarily on pictorial representation, using alphanumerics only when absolutely necessary [12:4]. The final report [10] depicted eighteen formats including those for primary flight displays and tactical situation displays.

In September of 1981, Boeing Military Airplane Company (BMAC) undertook a government contract to implement (in a

simulator environment) the displays developed by the McDonnell Douglas contract effort. Their study evaluated pilot acceptance of the pictorial formats using three different types of hardware displays. The final report [16] contained photos of the displays actually used as well as questionnaire responses provided by pilots when asked to assess the usefulness/desirability of the pictorial displays. The questionnaire responses pertinent to this thesis effort involve the displays used for threat warning: Pilots felt the displayed threat envelopes were too boxy or harsh and therefore were not optimized depictions of how they perceived the threats. Further, the envelopes were opaque so no graphical information of what might exist beyond the closest threat was available. It is possible that another, more lethal threat envelope or the target itself was being masked by the closer threat envelope. This was not a desirable display implementation.

Capt. Tom Wailes developed a number of threat envelope formats while pursuing a graphics-related AFIT thesis [15] that dealt with hidden line removal methods. These envelopes resembled search light beams and were similar to those produced by the Boeing study.

Procedure Modeling Techniques/Issues

Both Carlson [3] and Yan [18] have reported that tremendous progress has been made in hardware since computer-generated imagery was first used in flight

simulation more than a decade ago. Yet Carlson noted that there is still a problem in the acquisition, description, and generation of data of arbitrary shape and of sufficient complexity to be used in these computer-generated images [3:6].

In his dissertation, Carlson presented the mathematical foundations, issues, and techniques for generating geometric data. His goal was to facilitate intuitive creation of objects possessing "...significant geometric and topological complexity" [3:93]. One of the techniques covered was the procedural model earlier discussed by Martin Newell in 1975.

Newell defined a procedural model as being a model which represents its subject as a procedure with which other procedures can interact [11:28]. The interaction can include passed parameters that further specify the type or degree of the interaction. The model can be classified as "strongly parameterized" if the form of the represented object can vary widely depending on the parameters [11:90].

As an example, consider a strongly parameterized procedural model that can instantiate a building. The parameters passed to the procedure might include the x-y-z position, number of floors, number of windows per floor, and reflectivity of windows. Creating a building might then take the form of the following procedure call:

```
building( 5000, 10000, 0, 10, 12, 0.3 );
```

which instantiates a ten-story building with twelve,

slightly reflective windows per floor centered 5000 feet to the east, 10000 feet to the north, and 0 feet above some reference location. Creating a city could now consist of a short procedure that calls the procedure "building" a number of times with different parameters each time. The point here is that once the procedural model is coded, the user need not consider the actual graphical primitives (e.g., lines, polygons, points, etc.) involved in creating the object. The overall result is a flexible modeling tool that performs a very powerful function with minimal input. Newell concluded that procedure models facilitate the processing of scenes of far greater complexity than has proved practicable using data base modeling techniques [11:92].

While discussing data description languages, Carlson captured this concept when he stated the following:

Three dimensional data descriptions are typically obtained by extending, in some sense, a two dimensional data description. The responsibility of the data description language then is to provide capabilities for 1) constructing these two dimensional models, and 2) for extending them to three dimensional models [3:29-31].

This is precisely how procedural models were used in this research project.

Geometric Database Concepts and Issues

A secondary area of concern to this thesis effort was the transportability of any model generated. Tactical

situation models were to be generated on a single system for display on a number of distinct hardware systems. The data representation for the model was to be the vehicle that would facilitate the transfer between systems so its selection warranted some serious thought.

Most of the work in the literature that deals with geometric data base generation approaches it from a pragmatic point of view: A large number of edges and polygon faces must be displayed to get the best display possible yet the hardware can usually only support a relatively small number of them at a reasonable update rate (at least 30 hertz). A compromise between rate of update and number of displayed polygons must be made. The question then is how should a compromise on the number of displayed edges and polygons be reached? A number of methods are discussed in the literature including cultural feature instantiation [5], "building block" generation [13], and hierarchical tree data structure evaluation by display hardware [4]. Some of the underlying concepts of all these methods are covered in a paper by Widder and Stephens [17] that discusses the component parts of a database used to convert between database types on different simulators. These optimization considerations did not play a large role in this research as the quality of a static display was more important than the effective update rate. This will be discussed more fully in Chapter III.

On a more theoretical level, Carlson outlined some data generation issues [3:23-37] to consider when dealing with the creation and manipulation of models. These included data origin, data domain, data consistency, data representation, and data definition language. The main ones considered for this thesis effort were the latter three.

Data consistency is the quality that ensures the data is stored in a form that can be rendered accurately [3:25].

The representation of data (i.e., the model) must be considered for most any application. Carlson [3:25] outlined four questions that must be answered when a representation is being considered:

- 1) What primitives comprise the model?
- 2) What is the descriptive complexity of the model? A low descriptive complexity implies that a small number of parameters or descriptive elements are necessary to describe the object within the specified representation.
- 3) What is the functional complexity of the model? A high functional complexity indicates that operations can be performed on the model using the specified representation fairly efficiently.
- 4) How difficult is it to obtain the data to describe the model using the specified representation form?

Most graphics systems allow polygons (i.e., faces) to be handled as primitives. Carlson considered surfaces made up of polygons to have relatively moderate descriptive complexity [3:27]. Yan expected that future CGI (computer-generated imagery) systems would employ planar polygons as their major database modeling primitives, with

quadric surfaces as optional primitives [18:50]. Townsend [14] confirmed Yan's prediction by describing the way her company currently uses polygon-based modeling tools for real-time terrain simulators. These tools' capacities are actually specified by the number of polygons that can be simultaneously displayed.

The data definition language used in an application is one abstraction level above the two previous issues. Carlson calls the data definition language the aggregate of programs designed to aid the modeler from perception (of a real object) or conception (of a conceptual object) to the realization of a specific model [3:29]. Interestingly enough, both real objects (mountains) and conceptual objects (threat envelopes and flightpath) were modeled in this application. Carlson continued on data definition languages:

Included in the language are all the necessary primitives and primitive operations required to construct a topological model, as well as those evaluation interactive routines that allow the specification of object attributes so that an object model can be completed. It also includes, at least peripherally, the user interface...

The capabilities for object data generation include those for initially describing and entering the data and those for modifying existing data to create new objects [3:29].

The procedural models used to generate threat envelopes and mountains were the major components of data description language for this application. To a lesser degree, the

two-dimensional profiles used to generate the surfaces of revolution were also a part of the data description language.

Conclusion

It has been shown that although flight simulator hardware systems have increased in capability over the last decade, the methods to create the geometric models to be displayed are still lacking.

The use of procedural models to generate geometric surfaces was a very appropriate way to closely model the (relatively complex) artists' conceptions first proposed by McDonnell Douglas Corporation in 1981.

Significant research in data base creation and composition has been done. The work done by Carlson was particularly applicable to this thesis effort. The use of polygons to represent the tactical model was a decision supported by the literature.

Many of the questions posed in this chapter will be addressed in Chapter IV of this document.

III. System Requirements

General User Requirements

The overall requirement of the user was to have a modeling tool or environment that would allow him to create tactical situation displays that looked similar to the artist conception formats produced by the 1982 McDonnell Douglas study [10] (see Figure 1). This overall requirement was satisfied by providing three main capabilities to the user. The development of these capabilities would comprise most of this thesis effort.

The first capability was to generate the three-dimensional descriptions of objects used to model the threat envelopes, mountains, and flightpath (specified by waypoints). The specific characteristics of the software tools needed to create the objects were identified by considering what the final output should look like. When comparing Figures 1 and 2, the two characteristics that make the artist conceptions more pleasing to the eye are (1) the smooth, contoured nature of the threat envelopes, and (2) the semi-transparency of the envelopes. These characteristics are a mixture of modeling and display issues that must be separated to correctly implement the modeling system. The distinction between the two will be further explained in the next chapter.

The second capability was to reliably generate these tactical situation models interactively in a relatively

short amount of time (defined as one hour). Also, the resulting model was to be easily modifiable. The reason for these requirements was due to the nature of work done by the user.

Most research done in the MAGIC Cockpit is concerned with the psychological implications of information presentation to the pilot. Different methods for displaying information are compared to determine the effectiveness of each. Prior to this thesis effort, the major time constraint when generating a new experiment was the time to actually generate a new display. Typical development time for a new display format was approximately one to four weeks from inception to actual results in MAGIC. Essentially, the user wanted to spend less time generating (or modifying) the display so that the research environment of the MAGIC Cockpit could be more effectively utilized.

The third capability was that of database compatibility. The database representing the model must be transportable between differing computer systems. Most display formats would be developed by the user on the Silicon Graphics 3130 IRIS workstation and then moved over to a faster computer for real-time display. One of the potential real-time systems was a General Electric Compuscene. Data compatibility between systems was critical.

Specific Requirements

User Interface. A fundamental requirement was the ability to easily modify any part of the displayed model. The user should be able to easily change the level of detail dedicated to the representation of the terrain, the threat objects, or the flightpath.

The user requested that a mouse input device be used to position the threats and flightpath in the model. The screen format used when laying out a tactical region should resemble that shown in Figure 3. This particular format is typically called a horizontal situation display.

When laying out such a tactical situation, the user wanted to interactively specify the dimensions of the tactical region being created in nautical miles. For example, the user may start a design session knowing that the region being modeled extends 320 nautical miles to the east and 220 nautical miles to the north of some reference point.

When creating a flightpath, the first waypoint was designated by "I" (for "initial"). All subsequent waypoints were sequentially numbered starting with "1".

Also concerning the flightpath: the user wanted to interactively choose between two different conventions for specifying a waypoint position. The position of each waypoint was to be specified either (1) in absolute east and north displacements from the origin (the southwest corner of the terrain region) or (2) in terms of range and heading

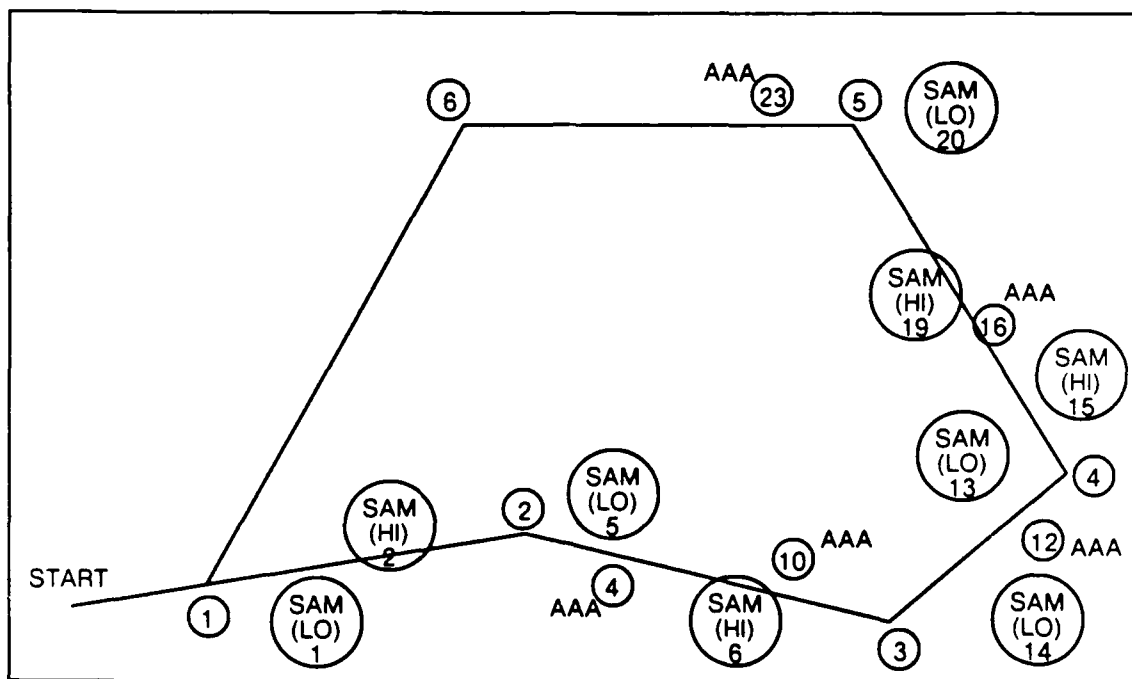


Figure 3. Example Threat Locations

from the previous waypoint. For the case of Waypoint I, the range and heading were relative to the origin.

Hardware. All software tools developed to model the tactical situations were to run on a Silicon Graphics IRIS 3130 workstation.

Software. The single requirement for actual software implementation was that coding be modular with as many comments as practical. The user had future plans to interface the viewing portion of this software with a flight dynamics computer model to simulate real-time flight.

IV. System Theory and Design

Overview

This chapter covers the theory and rationale of design decisions made during the development of the modeling environment that would satisfy these capabilities.

General design considerations are discussed first. Then data representation issues are covered since the data representations dictate the interfaces between all modeling tools and the model itself. Generation of modeled objects via procedural models is covered next. Then a discussion regarding apparent smoothness of modeled objects facilitated by light source modeling and Gouraud shading follows. Finally, a summary will end the chapter.

It should be noted that data representation, modeling, and display issues are all closely related to each other. The order of presentation for these areas poses the same problem as the proverbial chicken and the egg. Where does one concept end and another begin? How should the modeling environment represent the essence of the modeled concepts? Further, where does the division between the data representation (of the model) and the model itself actually lie? Can the data file be considered "the model" or is the model strictly the image on the screen? Can an intermediate symbolic form be considered a completely specified model? The answers to these types of questions will allow us to separate data representation, modeling, and display issues.

The reader is urged to cover all sections of this chapter before questioning his comprehension of any one area.

Finally, expressions like "the system", "the application", "the application environment", "the modeling system", and "the modeling environment" all mean fundamentally the same thing. They refer to a collection of software tools implemented on a Unix-based Silicon Graphics IRIS 3130 workstation that allows the user to fulfill the requirements outlined in the previous chapter. (Unix is a trademark of AT&T Bell Laboratories.)

General Design Considerations

Before delving into the main areas of this chapter, a few issues must be considered and understood.

Modeling vs. Display. In a modeling environment, it is sometimes difficult to separate the attributes that allow the model to be defined from those that allow the model to be displayed. If the distinction between the two is not clearly made, a model prone to incorrect interpretation might result.

These comments are not meant to imply that display issues should not be considered when constructing the model. Organization of the data representation can have significant impact on the performance of the display system. However, the efficiency of the data representation was not of major concern for this application.

Normalization. In order to keep the data representation as generic as possible, both color components and transparency values were expressed in normalized values. The red, green, and blue color components of a polygon were expressed individually as floating point values ranging from 0.0 to 1.0, inclusive. A value of 1.0 for the red component of a polygon will always mean maximum red intensity regardless of the system being used to display the model. This is very similar to the rationale behind the use of normalized device coordinates in many graphic display systems [9:50]. The transparency of a polygon was also expressed as a floating point value ranging from 0.0 for opaque to 1.0 for completely clear.

All surface normals were normalized so that only unit normals would exist in the data representation.

Together, these conventions strengthened the distinction between display and modeling issues in the modeling environment. No thought of how the model will be displayed (display issue) is given when the database is being created (modeling issue). This is only possible because the user did not require a dynamic model that could change in real-time.

Data Representation Efficiency. Since the user required the data representation of the model be transportable, an ASCII file was chosen as the transportable medium. In terms of storage requirements, this was an

inefficient data representation but it was necessary to support the transportability requirement.

Related to transportability is the issue of usability: What good is a data representation if, when ported to another dissimilar system, it cannot be easily used? To assure usability the data representation would have to be as generic as possible. This also leads to a more inefficient (more explicit) representation.

The user was consulted to ensure that these decisions would not compromise his requirements. Although this modeling system would allow the user to create any one display in a relatively short amount of time, he would still be spending a great deal of time tweaking that display to get exactly the image he wanted. So for this type of work, high update rate would not be important to the user. Once the model was correct and transported to the display system, concern for efficiency would be appropriate. Since these efficiency considerations would be system-dependent, they would be very hard to predict for all systems, and were therefore beyond the scope of this thesis effort. Nevertheless, the data representations were chosen so that interface to most any display system would be as easy as possible. Normalization was one method used to facilitate system-independent usability.

Model Type. Since we are representing the modeled objects with polygons, the model used for this application is considered a geometric model. Further, because the

entities being modeled are spatially oriented relative to each other, we will say that the the model has intrinsically geometric data associated with it. This is not to say that non-geometric information such as color, texture, or transparency cannot be included in the model. Indeed, this additional information allows the properties of the modeled entities to be more accurately represented.

Coordinate System Conventions. The x, y, and z axes of the data representation coordinate system will be mapped into east displacement, north displacement, and vertical altitude, respectively, in the modeling environment. Data representation coordinates will all be specified in feet. East and north displacements within the model will be specified in nautical miles while altitude will be specified in feet. This will necessitate some conversion between the data representation and the model.

Modeling Process Overview. Finally, an overall explanation of the intended modeling process will aid the reader's comprehension of the theoretical aspects discussed in the remainder of this chapter.

Figure 4 depicts the relationship between the modeling tools and the data representations. The user starts at Abstraction Level 2 by defining two-dimensional profiles of threat envelopes and mountains. These are stored and can then be revolved in space to define three-dimensional surfaces that are then stored into an object library. These surfaces represent the objects that will be placed

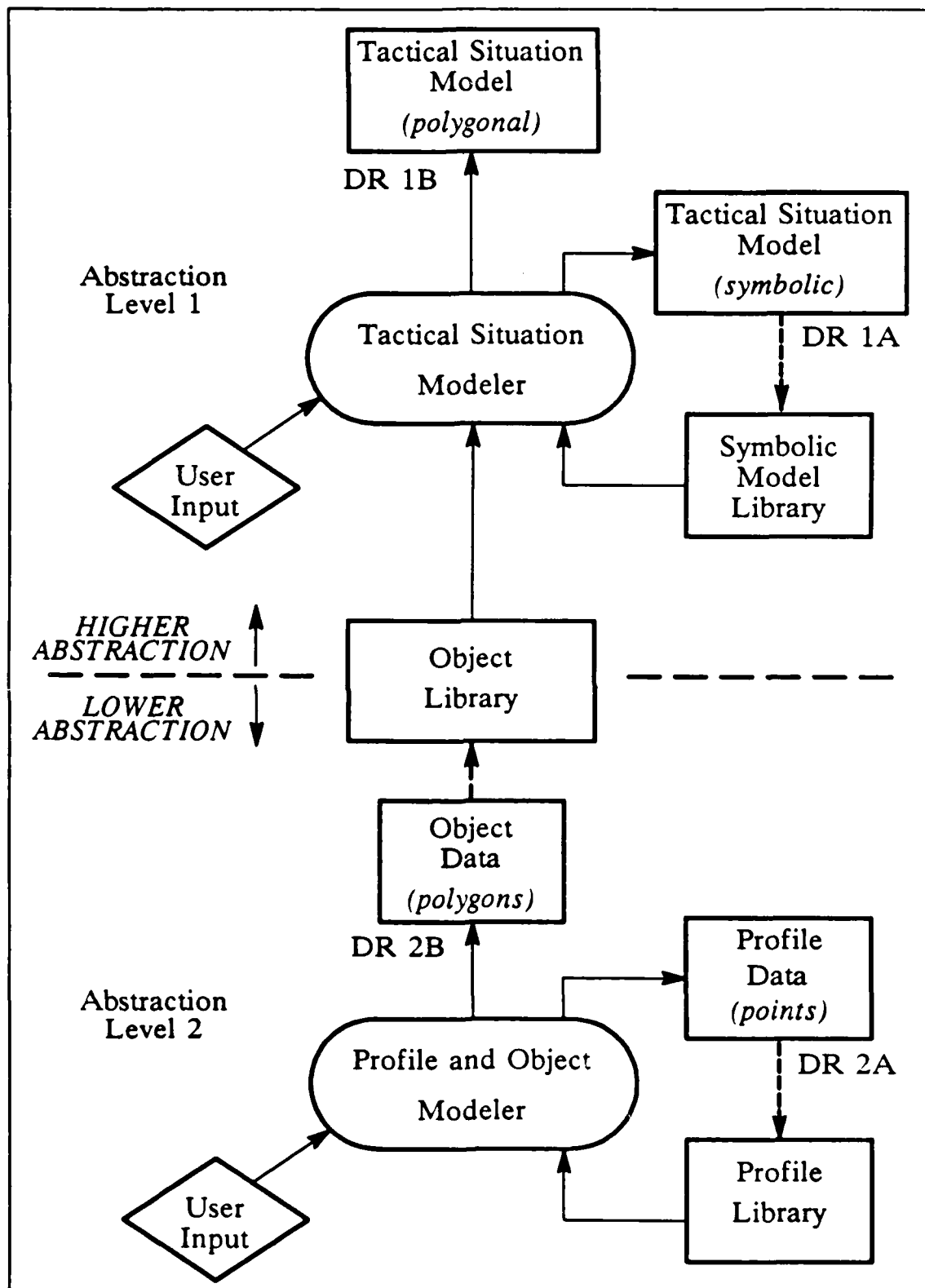


Figure 4. System Structure

(instantiated) into the tactical model. The user now exits the profile and object modeling tool. After this point, the user need not work at Abstraction Level 2 unless a new object must be created.

Next, the tactical situation modeling tool is used at Abstraction Level 1 to place the previously-created objects at desired positions in the tactical region being modeled. Optionally, a previously-created model can be loaded for modification. When done, the user saves a modifiable version of the model in the symbolic library. Lastly, the user may create the ASCII file that can be used as input to a display program on the modeling system or another system.

Concept of the Model

The concept of a model is easy to grasp. Models are used around us every day: weather forecast models, financial models, and population models, to name a few. We understand that "the model" will allow simulation, testing, and prediction of the behavior of the entities modeled for such purposes as understanding, visualization, experimentation, and learning [6:319]. But how does the model facilitate this "understanding"? The model is a kind of superstructure supporting levels of abstraction that promote a better understanding of the problem.

When levels of abstraction are provided to a user, he can ignore the lower, less important details of the modeling task at hand. Although the model may have to deal with

these details at some lower level, the user does not need them. The abstraction supported by the model (hopefully) allows the user to focus his attention on the level of abstraction he is most concerned with. His understanding of the basic problem will not be cluttered by unnecessary details.

As an example, consider the design and construction of a house. The architect wants a modeling environment that allows him to express the house in terms of walls, windows, and doorways, etc. The building contractor is more concerned with the lumber, glass and hinges needed to build the physical representation the architect's plan (model). If the architect is forced to work at the contractor's abstraction level, his assigned task will be much harder to accomplish. (Note that for this example, a single model could provide both levels of abstraction simply by presenting the model differently to each user. It must, however, always be clear to the user what level of abstraction the model is at.)

The level of abstraction provided by the system is more completely understood when the lowest component modifiable by the user is specified as well. In the above example, how low will our house construction model allow us to go? Is a wall the smallest manipulable unit, or does the model allow wall studs to be moved and paneling seam placement to be specified as well? The user must know the level of detail provided at each level of abstraction.

Implemented Abstraction Levels. Two levels of abstraction, depicted in Figure 4, were needed for the tactical situation model.

The lower of the two, Abstraction Level 2, supports creation of objects (mountains and threats) that will be used at the higher level. Each object is specified relative to its own coordinate system. The lowest level of component detail available at this level of abstraction is the endpoints of line segments that make up a profile. Each line segment represents the edge of a polygon that will be generated when the profile is revolved in space. The primary output of this level is a polygon file saved in the Object Library. This library constitutes the interface between Abstraction Levels 1 and 2.

The higher level, Abstraction Level 1, facilitates the placement of objects created at the lower level as well as the placement of waypoints that define the flightpath of the aircraft. Placement locations are specified relative to the two-dimensional origin of the tactical region being modeled. The user will specify the dimensions of this rectangular region before any objects are placed within it. The primary output of this level is a polygon file that can be transferred to any other computer system via network or tape. The user will spend most of his time at this abstraction level since he is mainly concerned with modeling the tactical environment.

Data Representation of the Model

The user specifications outlined the requirements for a modeling system capable of building a modifiable model representing a tactical situation scenario. The result of a modeling session was to be a data representation of the model that could be interpreted and displayed by a number of dissimilar computer systems. The data representation chosen was a critical component in the success of the entire effort.

When considering the underlying data structures for a modeling environment, one must consider the varied representation of data that can be used. Some representations are more economical, others are more explicit and possibly more redundant. When discussing PROLOG databases, Bratko noted that the drawback of the more economical (less explicit) representation is that some information always has to be recomputed when it is required [2:118]. That observation applies equally well to the data representation needed for this geometric modeling system.

Since the user desired both modifiability and portability of the model, two data representations of the model were developed. These will be referred to as data representations 1A and 1B (DR 1A and DR 1B). The number "1" signifies that they are used at Abstraction Level 1 (see Figure 4 and modeling discussion above). Note that two other data representations (DR 2A and DR 2B) are used at

Abstraction Level 2. They are used for object definition and do not represent the model itself.

Data Representation 1A (DR 1A). DR 1A facilitates model modification and is, in fact, the internal representation used by the main modeling tool. This particular representation depicts the model as object identifiers or icons located at particular x-y (east-north) positions in the world coordinates of the model, hence it might be considered a two-dimensional symbolic representation of the model. As such, DR 1B and the graphical representation cannot be directly generated from DR 1A because the description of each individual object is not included in DR 1A. Additional information must be retrieved from object descriptions created at Abstraction Level 2. This relationship between abstraction levels will be discussed further in the object generation section below. It is interesting to note that DR 1A completely specifies the model at Abstraction Level 1 even though it lacks information from the lower abstraction level.

The capability to modify the model is accomplished by storing the state of the modeling tool to a file. Then when modifications are desired, the file is loaded into the modeling tool, and modifications can be made at will. This method requires that other information needed for scaling, display, and DR 1B generation be included when this data representation is saved to a file.

DR 1A Composition. The essential information contained in DR 1A is the east-north positions of all mountains, threats and waypoints. An altitude value for each of the waypoints is also represented. Since this is a symbolic representation of the modeled objects, some type of pointer to the geometric data for each object must also be included so the modeling tool can construct DR 1B.

Data Representation 1B (DR 1B). DR 1B facilitates portability and is (in the form of a data file) the primary output of the modeling system.

In order to assure usability of the model representation between various hardware display systems, the format of the output representation had to be as generic as possible. The actual output took the form of a pure ASCII file to ensure transportability. The file contained a very explicit data representation comprised solely of the three-dimensional polygons that make up all surfaces in the model. This type of representation resulted in a data representation that was quite redundant in some respects. This redundancy was a direct result of each polygon being explicitly specified.

Once output to the data file, each polygon is independently defined, therefore it loses its association with all other polygons (except that all polygons are specified relative to the same origin). If four polygons share a common vertex, the vertex will be specified four times: once for each polygon.

This polygon independence also results in each polygon losing its association with any particular object, so the modeling tool can no longer modify this data representation. This would not be a restriction to the user as the modeling system was to only reside on a single (type of) computer system and model modifications would only be done on that system. The data representation transported to other display systems would not need to be modifiable.

DR 1B Composition. DR 1B consists of a homogeneous collection of polygons. The information needed to specify each polygon follows:

- 1) Number of vertices
- 2) Transparency value of polygon
- 3) Ordered list of vertices; each vertex specified as:
 - 3a) x, y, z position
 - 3b) x, y, and z components of unit normal that is normal to the surface at this vertex
 - 3c) red, green, and blue components of the vertex color

This explicit data representation format possesses data consistency for the following reasons:

- 1) the number of vertices are always specified for each polygon,
- 2) the vertices are always ordered in a counter-clockwise direction when viewed from the "outside" of the surface, and
- 3) the vertices are always coplanar.

Object Generation with Procedural Models

The end goal of a system for generating three-dimensional data is to remove as many constraints for

data description that are placed on the user as is possible, and to replace them with efficient algorithms within the system itself [3:40].

The use of strongly parameterized procedural models to create the objects in Abstraction Level 2 would allow the user to express a complex object with very little input.

The main procedural model used for this application creates a three-dimensional surface of revolution by sweeping a two-dimensional profile about the z axis. The profile is specified by a ordered set of connected line segments in the x-z plane (Figure 5a). This profile is then revolved to form a symmetric surface of revolution about the z axis (Figure 5b). The revolution is not continuous but is broken up into a number of sectors (input by the user) much as a round pie is cut into slices. The resulting surface is composed of polygons bounded by the sector boundaries and the horizontal lines connected the same profile endpoint in adjacent sectors. Note that these polygon boundaries can be viewed as the lines that define a wire mesh of the surface.

The number of segments specified in the profile directly controls the resolution of the surface contour, while the number of sectors specified controls the radial resolution around the symmetric surface. By specifying only the profile and sector count, the user can construct a relatively complex object in a short amount of time. This indicates the procedural model is strongly parameterized. Each profile can be saved ("Profile Data" in Figure 4) and used to generate a number of different objects, each having

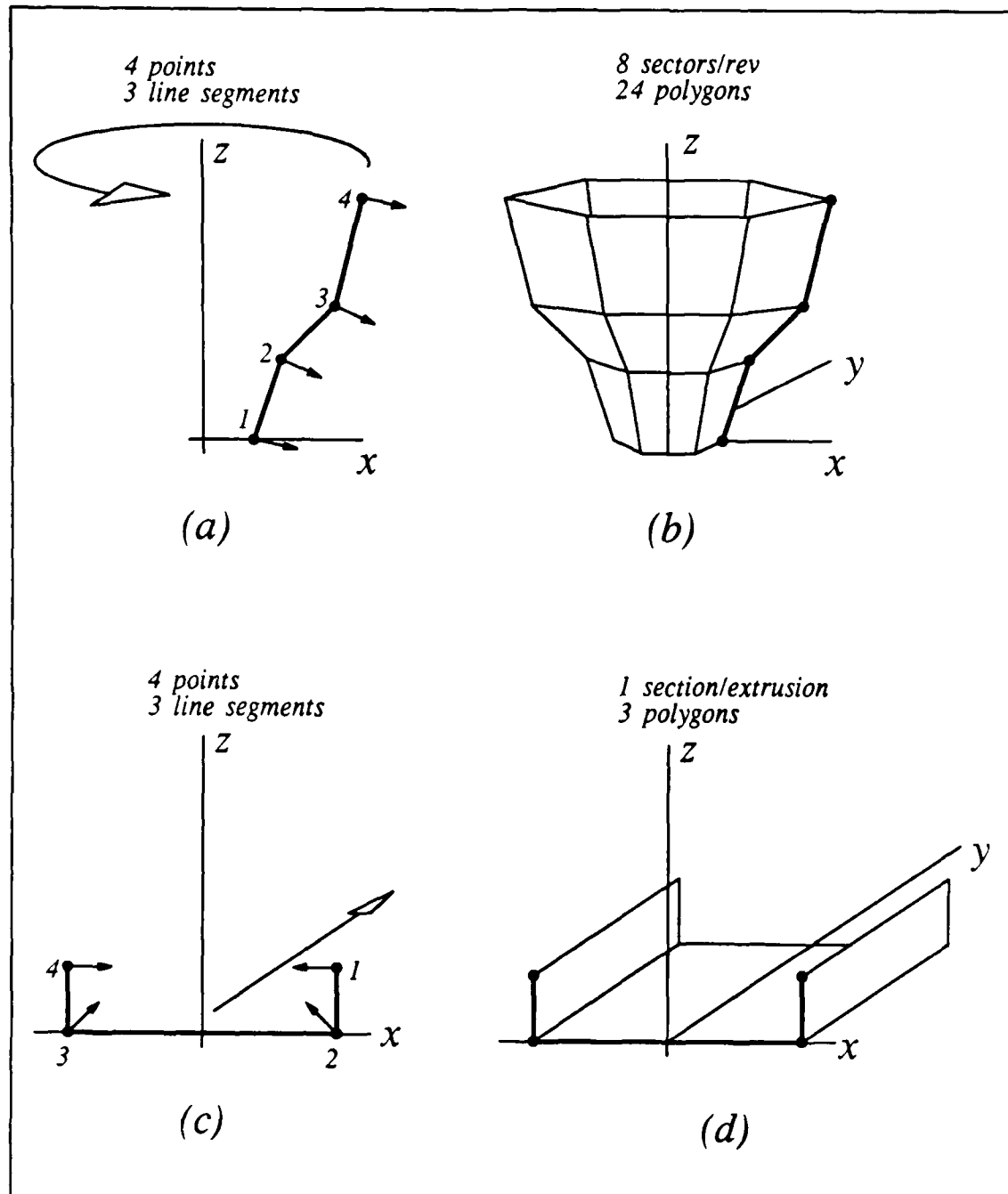


Figure 5. Procedural Model Examples. A profile (a) used to generate a surface of revolution (b). A profile (c) used to generate a surface of extrusion (d).

a different number of sectors in the revolution. This allows the user to run experiments that determine the relationship between display system performance versus the number of polygons displayed per object.

In order to simplify object generation, the normalized, two-dimensional contour normals at each line segment endpoint are calculated from the profile and stored as profile data. The three-dimensional surface normals are then generated by revolving the contour normals with the profile. The direction of a normalized contour normal is along the line that bisects the outside angle between two adjacent line segments at their common endpoint. If the particular endpoint is the first or the last in the profile, the unit normal direction is defined to be perpendicular to the segment owning the endpoint. One exception to this is when the first or last endpoint lies on the z axis. This indicates that the object will be closed on the bottom or top, respectively. In this case, the tangent plane is parallel to the x-y plane which means the normal points straight down (-z) if it's the first endpoint or straight up (+z) if it's the last endpoint. The outside of a profile is considered to lie to the right of the profile when "walking" in the x-z plane from the first endpoint to the last endpoint (see Figure 5a).

The normalized transparency value of the object that will be generated from the profile is also included as profile data. It is assumed that one transparency value

will be allowed for each object, i.e., variable transparency across the surface of an object will not be allowed.

To conclude the discussion on procedural models, note that Figure 5 also depicts a profile (5c) and its resulting surface of extrusion (5d). A similar procedural model will be used to create the flightpath in the sky by moving a profile from waypoint to waypoint.

Data Representation. (See Figure 4.) Two data representations must be used at Abstraction Level 2 to support object generation via procedural models. The first, Data Representation 2A (DR 2A), is used to represent the two-dimensional profile while the second, Data Representation 2B (DR 2B), is used to represent the objects generated. Note that the DR 2B format is identical to that of DR 1B. The difference is that DR 2B is assumed to be specified in object-centered coordinates while DR 1B is specified in the world coordinate system of the entire tactical region. If the tactical situation model were to be considered an object itself, it would, in fact, be specified in its own object-centered coordinate system. However, this seems to violate the abstraction levels defined for the modeling system, so further consideration is not warranted.

DR 2A Composition. DR 2A consists of a transparency value followed by an ordered set of points representing the endpoints of the line segments that make up the profile. The information needed to specify each profile follows:

- 1) Transparency of the object resulting from this profile
- 2) Ordered list of segment endpoints; each endpoint specified as:
 - 2a) x-z position values
 - 2b) x and z components of unit normal that is normal to the line tangent to the profile at this point
 - 2c) red, green, and blue components of the endpoint color

DR 2B Composition. DR 2B consists of a homogeneous collection of polygons. The information needed to specify each polygon follows:

- 1) Number of vertices
- 2) Transparency value of polygon
- 3) Ordered list of vertices; each vertex specified as:
 - 3a) x, y, z position
 - 3b) x, y, and z components of unit normal that is normal to the surface at this vertex
 - 3c) red, green, and blue components of the vertex color

Smooth Surfaces, Light, and Shading

The desire to make threat envelopes "more pleasing to the eye" really means we want them to appear to interact with the environment more as if they were real objects. So if there were truly some type of envelope surrounding the high-lethality region of an anti-aircraft threat, how would it appear when viewed in the tactical region during the day? It would have a smooth surface and would appear brighter on its sunlit side while its shade side would be appreciably darker. Although smooth, the surface would probably have

some texture as well; in this case, a semi-transparent one. Also, any surfaces hidden from the viewer would not be displayed, and we would expect the object to be in perspective depending on the viewer position.

It was assumed that both hidden surface removal and perspective viewing would be supported on any of the display systems used, so the geometric information provided by the data representation itself would suffice. The transparency was considered a display issue (versus modeling) that would be taken up on the display system as well. So all that remained was the lighting and smoothness issues.

Simulating Light. When we view an object, we see the intensity of reflected light from the surface(s) of the object [9:277]. The reflected light comes from various light sources around the object. The most important, and certainly the most natural light source in the tactical environment would be the sun. Since the distance to the sun is so much greater than the dimensions of the model, the sun is considered a point source. This greatly simplifies the calculations needed to simulate the effect of the sun in the tactical environment.

A second source of light in the modeled environment is the ambient light that exists because of multiple reflections of light from nearby objects such as the ground or mountains. This ambient component is strictly an additive quantity as ambient light produces a uniform illumination of the surface at any viewing position from

which the surface is visible [9:278]. Typical values range between 0.1 and 0.3. Allowing the user to vary this value will provide extra control of the modeling environment.

Calculating the reflective component at a point on the surface due to the light source involves calculation of the dot product between the unit normal vector of the surface at that point, and the unit vector that points from the surface point to the light source. This value will range from 0.0 to 1.0. The user will be allowed to modify the reflective value by specifying the position of the light source in the world coordinates of the tactical model.

Note that the point on the surface at which the unit normal vector is specified could be either at the center of each polygon or at each vertex of each polygon. The direction of the normal for the former would simply be perpendicular to the polygon face at that point. For the latter case, the direction of the normal would be the average of the normals to the faces of all polygons that share the vertex. The latter was chosen because it was the more explicit case and also because it allows Gouraud shading to be used.

The composite calculation for determining the color intensity at a point on the surface (i.e., at the vertex of a polygon) is:

$$I_{rgb} = C_{rgb}(\text{ambient} + \text{reflective})$$

where C_{rgb} is the actual normalized red, green, or blue color component of the point on the surface

I_{rgb} is the resulting normalized rgb color intensity

Care must be taken so the value (ambient+reflective) does not exceed 1.0. Also note that the intensity calculation must be done once for each of the red, green, and blue components of a surface point [9:276-282].

Gouraud Shading. If a surface composed of polygons is viewed with only the above lighting model, what does the object look like? The surface looks faceted. This is a characteristic of a constant or flat shaded model. As more polygons are added to the object, the apparent smoothness increases as intensity discontinuities become less noticeable. Adding polygons to the object is not practical, however, because most display systems' performance goes down as more polygons are added. So a faceted appearance must be tolerated in favor of display performance unless another method to display the model can be utilized.

Gouraud shading is an intensity interpolation scheme, developed by Gouraud [7], that removes intensity discontinuities between adjacent planes of a surface representation by linearly varying the intensity over each plane (polygon) so that intensity values match at the plane boundaries [9:289], [11:13-17]. Figure 6 shows an object in



Figure 6. Flat Shaded Image



Figure 7. Gouraud Shaded Image

a tactical environment model that is flat shaded while Figure 7 depicts the same data with Gouraud shading.

Some graphics systems that support Gouraud shading in software or firmware require the color intensity value of each vertex of a polygon be calculated by the user; others do not. The flat shading intensity formula above can be used to calculate these values, if necessary.

Summary

The major design considerations of this thesis effort have been discussed. Many of the issues were tightly interrelated and required much thought. It could be argued that data representation implementation has been outlined in this, the theory and design chapter. Presentation at this point was warranted, however, as the data representation was so fundamental to the rest of the design and the ensuing implementation.

V. System Implementation

Overview

This chapter outlines the major software tools developed during this thesis effort to satisfy the user's requirements. Three main applications were developed to satisfy these requirements. The profile and object modeling tool was called "Model"; the tactical situation modeling tool was called "Layout"; and the model viewing program supporting Gouraud shading was called "See". A number of utility programs were also developed; they are discussed in Appendix A. All programs were written in C.

Modeling Environment

Hardware, Software, and Firmware. All applications were developed for the Silicon Graphics IRIS 3130 graphics workstation. An extensive graphics library residing on the system provides access to any graphics routine via subroutine calls. The 3130 comes standard with 32 bitplanes that support a number of different display modes. The two used for this project were double-buffer mode for program Layout and single buffer mode with z-buffering for program See.

Double buffering [6:84] facilitates smooth movement of graphics objects without any perceived flicker. This was necessary in program Layout so objects could be dragged into position and so control bars to specify position and altitude could be utilized.

Z-buffering [6:560-561] is an image-space approach to eliminate hidden surfaces. When a pixel is rendered on the screen, its distance from the view point is stored in a 16-bit register. The next time that same pixel is about rendered, the 16-bit register is checked first: if the new pixel's distance from the viewer is less than the pixel that is already there, the new pixel is drawn to the screen and the new distance is written to the register; otherwise nothing is done. Since a 16-bit register is required for every pixel position (1024 x 767 for the Silicon Graphics) on the screen, a large portion of memory (16 bitplanes) is used when z-buffering is invoked. Additionally, all world coordinate values can be smaller than -32767 or larger than +32767. Thus, the world coordinate system must be normalized to this range if any values are expected to violate these bounds, as was the case for program See.

The IRIS provides a powerful window manager called "mex" which allows multiple windows with pull-down menus to be easily controlled from an application program. On the positive side, the user interface provided by mex is excellent. One of the sub-goals when implementing the program Layout was to restrict most user inputs to the mouse device. This sub-goal was met, but only because the window manager took care of so many tasks.

On the negative side, the special subroutine calls that must be included in an application program to use mex (along with all the other graphics routine calls) make the code

non-portable to other systems. Further, when invoked, mex requires two dedicated bitplanes for its own use when in single buffer mode; four when in double-buffer mode. This can be a limitation, depending on the number of bitplanes needed for color. It was this restriction that prevented mex from being used in program See.

Another feature provided by the Silicon Graphics was polygon backface removal [9:261-262]. This technique checks to see if the viewer is on the "inside" or "outside" of the plane of a polygon based on the normal to the plane. The direction of the normal vector is calculated from the order in which the polygon vertices are specified. If the viewer is on the "inside" of a polygon, it will not be rendered. This technique works the best for solid volumes modeled with polygons because there will always be an "outside" face toward the viewer, regardless of his position. All surfaces of solid disappear (i.e., are not rendered) if the viewer's position happens to lie within the solid being modeled.

Gouraud shading was also supported by the IRIS. The application using the technique has to set up the color table appropriately to be used for shading polygons. Then, when a polygon was to be rendered, the application would pass an index into the color table that was to be associated with each vertex of the polygon. Shading interpolation was then performed automatically by the system. A minor firmware error (confirmed by vendor) was discovered when a Gouraud shaded polygon was (hardware) clipped by the

viewport boundary. No solution was proposed by the vendor.

Unfortunately, no lighting model was supported by the 3130. The application, therefore, was responsible for modeling any light sources needed for the application image.

A nice feature that the 3130 did offer was textured pattern fill for polygons. This would effectively allow an alternating pattern of background and foreground color to be written to a polygon. The background color portions of the polygon appear to pass the color of whatever is behind the polygon. A simple semi-transparency viewing tool was implemented to see what the results looked like.

Before leaving the 3130 discussion, it should be noted that Silicon Graphics chose to implement a non-standard version of C on the IRIS. What most C implementations call "double", the IRIS calls "float". Further, any trigonometry or math functions that return a "float" value must be called with a prefix as `l_cos`, `l_tan`, and `l_sqrt`. This caused a bit of confusion when the Version 1.0 programs were ported from the Sun Microsystems computer.

Working Directory and File Conventions. The main working directory, called the home directory and designated `"/"`, contained the executables for the main modeling tools. Two directories were required to exist under this home directory; a third was optional.

The first, "prof", contained all profile data (Profile Library, Figure 4) created and used by program Model. All data files that existed in this directory had to be named

with a ".prof" extension. Program Model would only operate on files with this extension.

The second directory, "poly", contained all polygon files generated by program Model (Object Library, Figure 4). All data files that existed in this directory had to be named with a ".poly" extension. Programs Model and Layout would only operate on files with this extension.

The third directory, "lo", contained all symbolic data generated and used by program Layout (Symbolic Model Library, Figure 4). All data files that existed in this directory should have been named with a ".lo" extension. This convention was not expected nor enforced by program Layout, but its use was encouraged so the home directory could be kept as clean as possible.

Other special files were required in the home directory, but they will be discussed below with their respective applications.

Data Representations Implementation

The implemented data representations closely followed the design put forward in the previous chapter. Inclusion of comment lines in DR 1B and DR 2B was implemented. Any line at the head of the file with a leading "//" was considered a comment. Any number of these comment lines could be added to any of these files as long as they were at the head of the file. Once a line without "//" in columns 1 and 2 is encountered, all input procedures expect only

numeric data from that point on.

Additional information was added to DR 1A to facilitate complete reconstruction of a symbolic model representation.

Model: A Profile and Object Modeling Tool

Function. This application was developed to construct two-dimensional profiles as well as three-dimensional surfaces.

The program allows the user to see a directory of the existing profiles as well as a directory of the existing objects (surfaces). A new profile can be defined or a previously-created profile can be loaded and listed to see its values. It cannot be modified. If a new profile is created, the user can optionally save it to a file. Once a new profile has been defined or an old one has been loaded, a surface of revolution can be generated. The profile may also be extruded in the +y direction, but this is of limited use since the program, Layout, cannot rotate objects in the tactical region. The user may exit the main menu at any time.

User Interface. Model is a text-based program. It is invoked by typing "model" [RETURN] at the system prompt. The main (and only) menu is shown in Figure 8. Figure 9 shows an example of defining a profile while Figure 10 depicts the same profile being revolved into an eight-sided pyramid of sorts. Note that the profile is defined starting at the max-x/min-z position toward the min-x/max-z position.

```
----- MAIN MENU -----  
  
0> EXIT  
1> Profile Directory  
2> Load Profile  
3> Store Profile  
4> List Profile  
5> Construct Profile  
6> Revolve Profile  
7> Extrude Profile  
8> Surface Directory  
  
Enter menu choice -->
```

Figure 8. "Model" Main Menu

Following this convention allows us to assume any position to the right of the profile is on the "outside" of it while any position to the left is on the "inside".

The interface to the extrusion portion is virtually identical to the revolution portion except that sectors of the revolution are replaced with sections of the extrusion.

Program Operation. After a profile is input by the user, the program checks to see if more than one point had been entered. If not, an error message is issued and control is returned to the main menu. If two or more points have been defined, the program determines the "normals" to this profile at the points defined. This is done by bisecting the outside angle with a unit vector. By

```

Enter menu choice --> 5

Will all points be the same color? (y/n) --> y

Enter red green blue
      components for all points --> 1.0 0.0 0.0

Enter transparency value for surface -----> 1.0

Max points = 100.

Point #1: Enter X and Z or RETURN to quit --> 2500 0
Point #2: Enter X and Z or RETURN to quit --> 1700 500
Point #3: Enter X and Z or RETURN to quit --> 0 1000
Point #4: Enter X and Z or RETURN to quit -->

      Confirm quit? (y/n) --> y

Save profile to file? (y/n) --> y

Enter profile name (without . extention) --> example

Enter comment up to 65 chars.
Comment --> This is an example profile

```

Figure 9. Profile Input Example

definition, the normals to the first and last points are at right angles to the first and last line segments, respectively. The exception to this is when $x=0$ for the first and/or the last point. In this case, the unit normal points straight down ($-z$) if the first point's x coordinate is equal to zero, and/or straight up ($+z$) if the last point's x coordinate is equal to zero. The reason being

```
Enter menu choice --> 6
3 points in profile: 2 polygons per sector.
Sectors per revolution? -----> 8
16 polygons will be generated.

Enter name to save under (without . extention) --> example
Enter comment up to 65 chars.
Comment --> Generated from example.prof

Processing...Complete
```

Figure 10. Surface of Revolution Example

that if either of these two points' x coordinate is equal to zero, then that end of the surface is closed and the tangent plane at that point lies parallel to the $z=0$ plane.

The data file that resulted from the profile definition above is shown in Figure 11. Figure 12 shows the first two polygons of the object that resulted from revolving the profile from above. As indicated by the figure captions, the profile definition resulted in a file "example.prof" being stored in directory "./prof", while the revolution procedure resulted in a file "example.poly" being stored in directory "./poly". Both files did not require the same filename prefix, "example"; each could have been named differently.

```
//This is an example profile
0.000000
2500.000000 0.000000 0.529999 0.847998 1.000000 0.000000 0.000000
1700.000000 500.000000 0.409883 0.912138 1.000000 0.000000 0.000000
0.000000 1000.000000 0.282166 0.959366 1.000000 0.000000 0.000000
```

Figure 11. Profile Data Format: File ./prof/example.prof

```
//Generated from example.prof
4 0.000000
2500.000000 0.000000 0.000000 0.529999 0.000000 0.847998 1.000000 0.000000 0.000000
1767.766914 1767.766992 0.000000 0.374766 0.847998 1.000000 0.000000 0.000000
1202.081502 1202.081554 500.000000 0.289831 0.289831 0.912138 1.000000 0.000000
1700.000000 0.000000 500.000000 0.409883 0.000000 0.912138 1.000000 0.000000
4 0.000000
1700.000000 0.000000 500.000000 0.409883 0.000000 0.912138 1.000000 0.000000
1202.081502 1202.081554 500.000000 0.289831 0.289831 0.912138 1.000000 0.000000
0.000000 0.000000 1000.000000 0.199522 0.199522 0.959366 1.000000 0.000000
0.000000 0.000000 1000.000000 0.282166 0.000000 0.959366 1.000000 0.000000
```

Figure 12. Polygon Data Format: File ./poly/example.poly

The profile "example" could be used repeatedly to generate objects with different numbers of sectors per revolution.

The application that uses the surfaces or objects created by Model will now be discussed.

Layout: A Tactical Situation Modeling Tool

The need for a high-quality user interface made this the most complex application developed during this effort. Double-buffering was used to facilitate smooth movement of objects. The window manager, mex, was used to implement pull-down menus, multiple windows, and graphical input via the mouse device. These features facilitated a very intuitive user interface. Before proceeding, a few definitions are in order.

For the remainder of the Layout discussion, "pointing" to or at a graphical object refers to pressing down the left mouse button after the cursor has been positioned over the object. "Dragging" a graphical object refers to pointing to the object and then moving the cursor while the left button is being held down. "Selecting" an object means two different things, depending on the object: "selecting" a menu or menu entry consists of pressing down the right mouse button, moving the mouse until the cursor is positioned over the desired menu entry, and then releasing the button. Conversely, "selecting" an object icon consists simply of

pointing to the icon which, in turn, makes it the "current object."

Function. This application allows the user to easily create tactical situation models. Icons representing mountains, threats, and waypoints are positioned on a two-dimensional terrain grid whose size is user-specified. The tool then creates a polygonal data representation (DR 1B) of the model by instantiating DR 2B objects (mountains and threats) in three-space at the locations represented by the two-dimensional icon positions. Additionally, a pre-defined flightpath channel profile is moved from waypoint to waypoint to "sweep out" a polygonal representation of the flightpath in three-space. A symbolic representation (DR 1A) is also created to facilitate later modification, if necessary.

Modeling Session Preparation. Before a modeling session can begin, the user must indicate which objects from the DR 2B Object Library will be used during the session. This is accomplished by listing those objects in two files: "./mountains.in" for mountain objects and "./threats.in" for threat objects. The format for these files is:

```
objectname_1.poly
objectname_2.poly
objectname_3.poly
...
objectname_n.poly
```

where each line is the name of a file (in directory "./poly") which contains the DR 2B data describing an

object. If some filename "objectname_n.poly" does not exist, a warning message will be issued when "layout setup" is executed.

Commandline Interface. Layout is invoked by entering one of the following commands at the system prompt:

```
layout [-t] setup
layout [-t] eastdist northdist
layout [-t] filename.lo
```

The first command verifies the existence of the DR 2B object files listed in files "./mountains.in" and "./threats.in". Then, a working file needed to map DR 2B object files to DR 1A icon positions is created. The text strings needed to create the Object Type menu entries are also generated and stored for later use by the graphical portion of the program. This command need only be used when files "./mountains.in" and/or "./threats.in" have been modified.

The second command form allows the user to specify the dimensions of the tactical region to be modeled. A terrain grid of the specified dimension is then displayed and the user can begin a modeling session. When done, the user can store the model in both DR 1A and DR 1B formats.

The final command form allows a previously-created model, in DR 1A format, to be loaded and modified. When done, the modified model can be saved under the original DR 1A filename or under a new one. The user may also store the model in DR 1B format.

In all three cases above, "-t" is an optional flag that enables program trace mode for debugging purposes. This option was not deleted in the final product because the user intended to modify the code.

Graphical User Interface. Once invoked, Layout is controlled via the mouse. A status window is provided to keep the user apprised of current editing modes, object types, positions, and altitudes. All object icons can be directly positioned on the terrain grid with the mouse-driven cursor. Alternately, objects can be positioned via graphical control bars or "sliders" for exact placement. The value controlled by a slider is changed by dragging the control knob on the slider. Waypoint altitudes can only be specified with a slider.

Anytime a mountain, threat, or waypoint icon is selected, the icon in the terrain grid, along with its type and position in the status window will flash to indicate that it is selected. An object icon is selected either by instantiating it when in Add mode, or by pointing to it when in Delete or Edit mode. When in Delete or Edit mode, the cursor must be within one grid square distance from the desired object icon for it to become selected.

Mex is used extensively to present the user with a multi-tiered menu system. Figure 13 gives an overview of the menu/function hierarchy. A rounded box represents an intermediate menu level, while a rectangular box represents the lowest menu reachable.

Menu Functions. The basic operations performed at each level of the menu structure will be described along with the related user interface issues. Create Database and EXIT menu selections will be discussed under "Program Operation".

Main Menu. Allows user to enter Terrain Operations, Flightpath Operations, or Threat Operations. Whenever one of these three options is selected, the status window will become visible and the operations mode selected will be displayed at the top of the status window. Sliders also become visible for positioning objects icons and for specifying waypoint altitudes.

If Create Database is selected, the polygon file (DR 1B) will be generated and saved to file "database.out" in the current directory. A message will be displayed while this operation is being performed. Time to create database can range from 10 to 60 seconds or more depending on how many polygons were placed into the model. When the database has been created, control again returns to the main menu.

If Exit is selected, the graphics image will disappear, but the user will be prompted for a filename to store the symbolic model under. The user should specify a filename of the form "lo/filename.lo" so that the symbolic model goes into the ./lo directory. If the user does not specify a name, the symbolic information will be saved in file "session.lo" in the current directory.

Ops Menus. The menus used for Terrain, Threat, and Flightpath operations will be discussed together as they

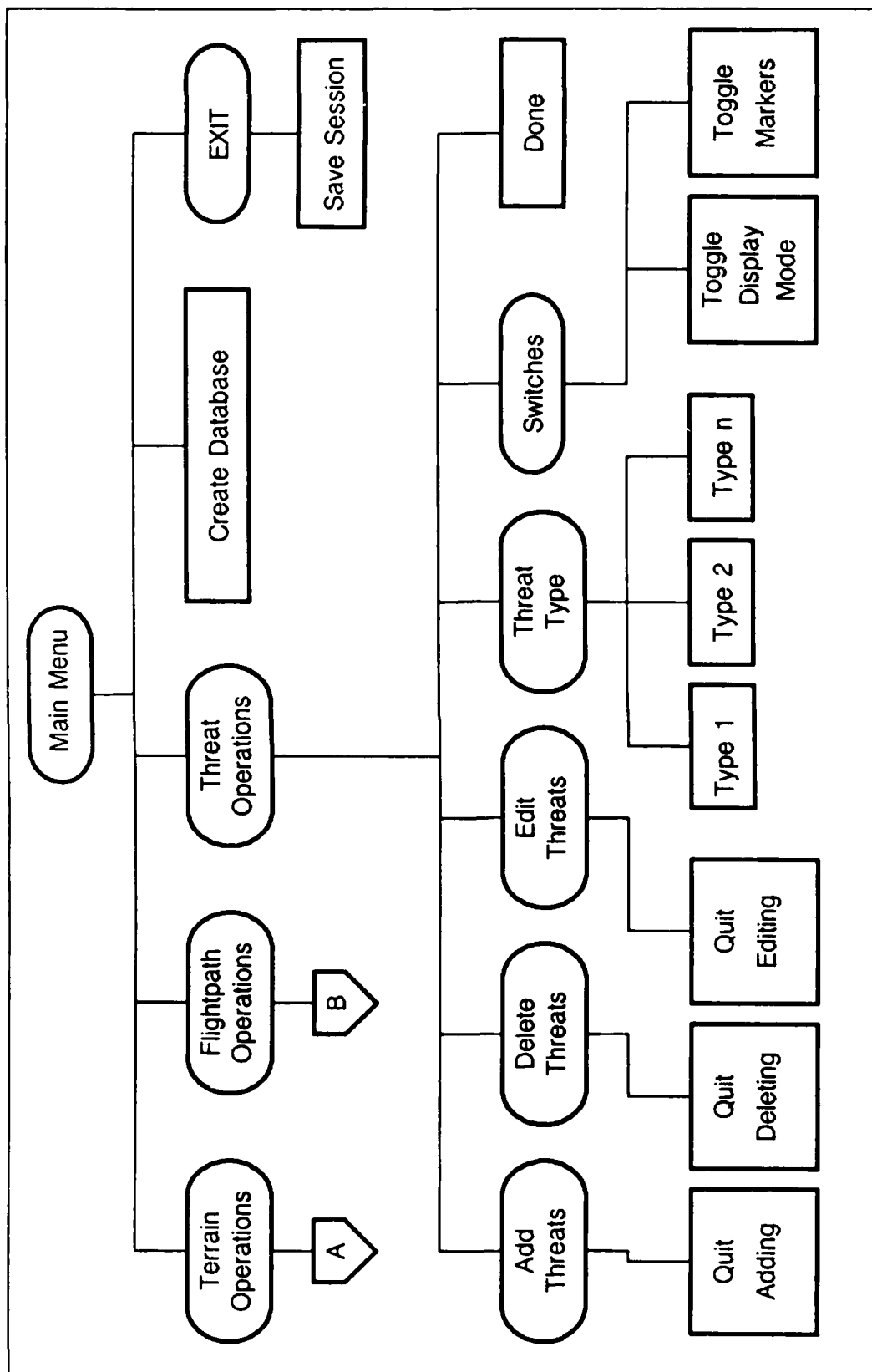


Figure 13. Menu/Function Hierarchy

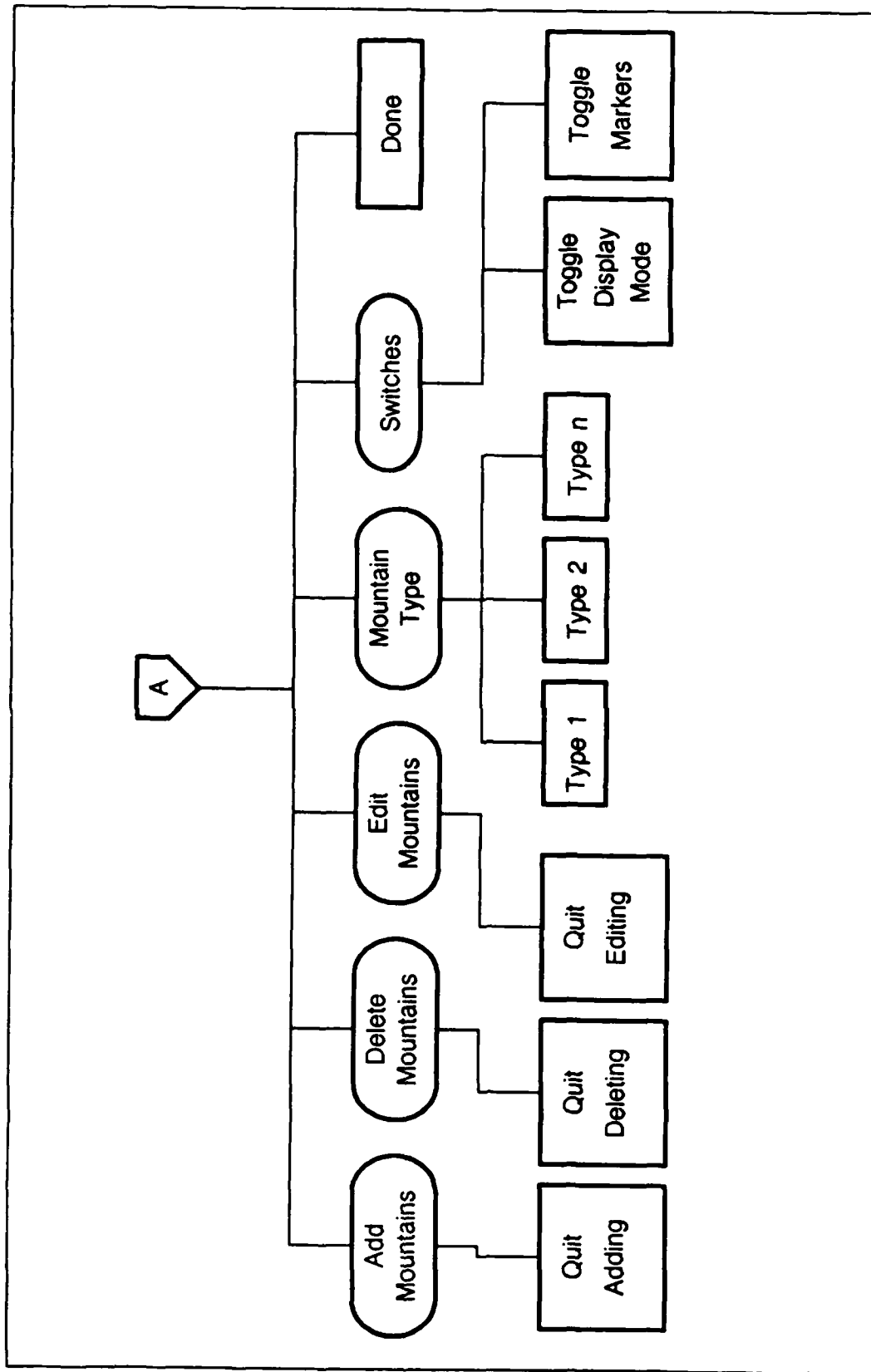


Figure 13. (continued)

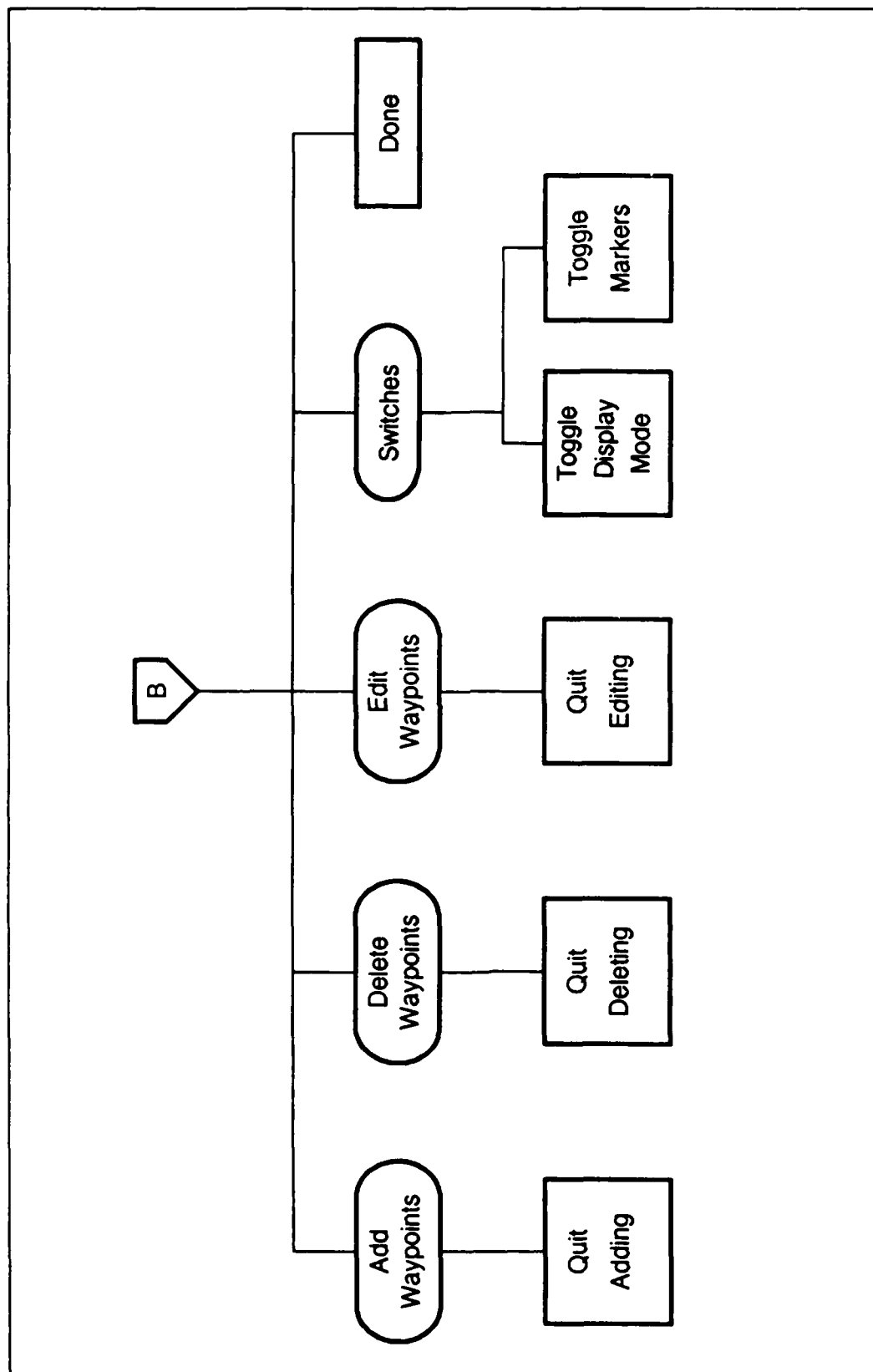


Figure 13. (continued)

all perform the same basic functions using the same interface. For the sake of this short discussion, mountains, threats, and waypoints will collectively be called "objects". Exceptions will be noted.

"Add objects" allows the user to place new icons that represent instances of the object currently displayed in the "Current object" box of the status window. To place an icon, move the cursor to the desired position on the green terrain grid and push the left mouse button; an icon will appear. While the button is held down, the icon may be dragged around the grid. When the button is released, the icon will no longer track the cursor. Since the object icon is still selected (i.e., is still the current object), however, it can still be moved by dragging the appropriate slider knob. The selected object icon will move to the new position as the slider knob is moved. Release the left button when desired position is reached. Whether dragging the object icon or moving it with the slider knob, the flashing position boxes in the status window are constantly updated so the exact position is always known. Continue adding new object icons by pressing the left mouse button. When done, select "Yes" from the "Quit Adding?" menu. Control will return to the specific Operations menu.

"Delete objects" allows the user to selectively delete object icons belonging to the current operations mode by pointing to the specific icon and selecting "Yes" when prompted "Delete?". Any existing icons with icon numbers

greater than the deleted icon will be renumbered. When done deleting, select "Yes" from the "Quit Deleting?" menu.

"Edit objects" allows the user to move any existing object icons. The icon may be dragged with the cursor or moved with the sliders. When done editing, select "Yes" from the "Quit Editing?" menu.

"Object Type" (Terrain and Threat Ops only) allows the user to change the current type of object that can be added. To select a new object, press the right mouse button and move the cursor to the "objects Type" menu entry. Before releasing the button, move the cursor off the right or left side of "objects Type" to expose a submenu of object types. Select the desired object type; its name will appear in the "Current object" box of the Status window.

"Switches" allows the user to change the display mode used in the status window as well as to change whether the icon numbers or markers should be displayed on the terrain grid. This menu operates like the "object Type" menu above. To toggle either of these settings, simply press the right mouse button and move the cursor to the "Switches" menu entry. Before releasing the button, move the cursor off the right or left side of "Switches" to expose the "Toggle Display Mode" and "Toggle Markers" submenu. Now select the setting to toggle; the change will immediately take effect. Note that the values shown in the position boxes of the Status window are relative to the southwest corner of the terrain grid when the X-Y display mode is selected. When

the Heading-Range mode is selected, heading and range are specified relative to the object preceding the selected object. For the case of "Current object" equal to I or 1, heading and range are specified relative to the southwest corner of the terrain grid. The Heading-Range display mode really only makes sense when in Flightpath Operations.

Discussing the program-user interface gave an accurate understanding of overall program operation. The actions of the program after the user has constructed the model are outlined next.

Program Operation. While the user is constructing a tactical situation model, Layout maintains a symbolic representation of the model. Conceptually, this representation takes the form of three separate lists, one for each object icon type. The lists for mountains and threats each contain east displacement, north displacement, and object type information for each of the mountain or threat icons placed in the model. The list for waypoints contains east displacement, north displacement, and altitude of each waypoint icon.

When the user selects "Create Database" from the main menu, the model is saved in DR 1B format. To accomplish this, the following actions are taken:

- 1) File "database.out" is opened for output in the current directory.
- 2) A single polygon the size of the modeled terrain region is written to the output file.

- 3) For each mountain icon in the mountain list: a copy of the DR 2B object file corresponding to the object type specified for the icon in the list is read and translated by the east-north displacements also specified in the list. The resulting data is appended (i.e., instantiated) to the output file.
- 4) Step 3 is repeated for all threat icons in the threat list.
- 5) The flightpath is generated by moving a flightpath channel profile from waypoint to waypoint, starting from the first one. The resulting surface swept out by the profile is represented by polygons that are appended to the output file.
- 6) The output file is closed.

All dimensioned data in the DR 1B output file is specified in feet.

Finally, when the user exits the program, the model is saved in DR 1A format either to file "session.1a" or to a file named by the user. This file contains:

- 1) Mapping information that relates mountain and threat object types to specific data files in the ./poly directory.
- 2) Data needed to construct the actual submenus under "Mountain Type" and "Threat Type".
- 3) The number of mountains in the model along with their location and type.
- 4) The number of threats in the model along with their location and type.
- 5) The number of waypoints in the model along with their location and altitude.

This file can be later loaded by Layout, modifications can be made, and new DR 1A and DR 1B files can be generated.

Implementation discussion will now turn to the display application developed on the 3130 for viewing a user-defined model in DR 1B format.

See: A Model Viewing Tool

Function. This application was developed to view a tactical situation model using mesh (wireframe), flat, or Gouraud shading.

The program allows the user to specify viewing position, center of interest, light position, type of shading, and amount of ambient light used when viewing a model. These values can be stored in file "./.defaults" in the home directory, if desired. Z-buffering was used to provide hidden surface removal.

The coordinate system for See maps the X coordinate into east displacement, the Y coordinate into north displacement, and the Z coordinate into absolute altitude.

User Interface. See is a text-based program similar to Model. It will display any polygon file (DR 2B or DR 1B) existing in directory "./poly". The program is invoked by typing "see" [RETURN] at the system prompt. Figure 14 shows the main status and menu display after a model called "env" was loaded.

The text above the center dashed line lists some attributes of the model currently loaded. The figure indicates the current model is made up of 242 polygons and that the model extends 50,000 feet (8.3 nautical miles at

MAIN MENU

Model Name: env

Comment: 50,000 by 50,000. from valley.prof.

of Polys= 242

Xmin=	+0.0	Xmax=	+50000.0
Ymin=	+0.0	Ymax=	+50000.0
Zmin=	+0.0	Zmax=	+19100.0

Viewer Position	Center of Interest	Light Position
X= +18000.0	X= +17000.0	X= +100000.0
Y= +0.0	Y= +40000.0	Y= -100000.0
Z= +50000.0	Z= +20000.0	Z= +50000.0

ambient = 0.15
shading = SMOOTH

(L)oad model, (M)odify settings, (D)isplay model, or (Q)uit?

Enter choice -->

Figure 14. "See" Main Menu

6000 feet per nautical mile) to the east and north. The highest point in the model is at an altitude of 19,100 feet. A comment entered by the user when the model was created is also shown. Note that the origin is located at the southwest corner of the model, as it was in Layout.

The data below the center dashed line show the values of the user-modifiable display settings. Currently, the viewer position is located three nautical miles to the east of the origin at an altitude of 5000 feet. The viewer is "looking" at a point 40,000 feet to the north, 1000 feet (18,000 - 17,000) to the west, and 3000 feet (5000 - 2000) below his current position. The light source is located 100,000 feet to the east and 100,000 to the south of the origin at an altitude of 50,000 feet. Smooth (Gouraud) shading is being used and the ambient light level is 0.15.

Menu Functions. The operations performed for each menu selection will now be described.

(L)oad model. This menu choice allows the user to load an existing polygon file from directory "./poly". While the model is loading, periods will be printed to the screen at the rate of one per 50 polygons loaded. This confirms the loading process for the user.

(M)odify settings. This menu choice allows any of the user-modifiable display settings to be changed. After entering "m" [RETURN], a submenu will be displayed allowing the user to select and modify any of the settings. The values of all settings can be saved from this lower submenu.

(D)isplay. After all values have been appropriately set, this menu choice allows the model to be viewed. After this selection is made, the screen will go blank while the image is being written into the display memory. The time needed to display a single image ranges from one to about seven seconds as z-buffering and Gouraud shading are both computationally expensive. Once the model image appears, the user may return to the main menu by pressing "ESC".

(Q)uit. Self-explanatory.

Program Operation. Some of the more complex aspects of the actions resulting from the above menu choices will now be explained.

When first invoked, See loads all viewing settings from file "./.defaults". It then displays the main menu. The logical next step is to load a model.

The application prompts the user for a model name. It then reads the response, appends a ".poly" extension to it, and uses the result as the filename of the model to be loaded from directory "./poly". All polygons are read into an array in memory.

Once loaded, the program scans or "scopes" all polygons to determine the domain of the input data. This information is needed 1) to let the user know the extent of the model so he can select a reasonable viewing position, and 2) to allow the program to scale all input data for z-buffering.

Since the z-buffer is only sixteen bits long, no world coordinate value can be less than -32767 or greater than 32767 for z-buffering to work correctly from all possible viewing positions. Therefore, the maximum data value must be mapped into 32767 and all other values must be scaled by 32767.

See assumes that the largest displacement in the model extends from the origin to the farthest corner of the box defined by the planes $X=0$, $X=X_{max}$, $Y=0$, $Y=Y_{max}$, $Z=0$, and $Z=Z_{max}$. If this displacement is less than 32767, the scaling value is set to 1.0.

After scoping the model for minimum and maximum values, the program again scans the model to determine how many colors are present. The color table [4:132,134] is then divided into this many blocks, with the maximum number of steps per block equal to 256. Color ramps ranging from zero intensity to maximum intensity are then entered into the color table for each color present in the model. There is a restriction: Although 256 distinct shades can exist for each color in the model, See can only support seven different colors (plus black) in the model. This is strictly a limitation of the application.

Before the model can be viewed, the light source-induced color intensity variations of each polygon vertex must be calculated. The color intensity is calculated at each vertex using the method outlined in the previous chapter. The color table index number corresponding to the

calculated intensity for that color is then assigned to the vertex. This calculation is required whenever a new model has been loaded, whenever the light source has been moved, or when the ambient light value has been changed. Further, it must be done regardless of the type of shading being used.

Having done all the necessary preprocessing, the application is now ready to display the model. If any transparent polygons are present in the model, a pattern mask will be used when each polygon is rendered. This will give the appearance of transparency in the same way that a screen door appears to be transparent.

After the user is done viewing the image on the screen, the "ESC" key may be pressed to return to the main menu. This is the only way, short of crashing the system, to return control to the user.

Summary

The three major applications developed during this thesis effort have been described. The first application called "Model" was developed to generate object profiles and, in turn, object surfaces from the profiles. It does not require any graphics capability as it is text-based.

The second application, "Layout", allows the user to create a tactical mission scenario model by instantiating the objects created by "Model" at various locations on a terrain grid. The generated model is output in a system-

independent form that can be transferred to other graphics systems.

The last application, "See", was developed so that models generated by "Layout" could be viewed to verify the designer's intentions.

VI. Conclusions and Recommendations

Results

The overall purpose of this thesis effort was to develop the capability to model and display tactical situation scenarios similar to the artist conceptions developed by McDonnell Douglas Corporation (Figure 1) for the Air Force Flight Dynamic Lab in 1981. An example of the results is shown in Figure 15.

The implementation of this capability took the form of a number of computer applications that can model mountains, threat envelopes, and flightpath channels. By using these programs, the user can interactively create a complex tactical situation model in less than one hour.



Figure 15. Example Model Image

The data representation chosen for the model facilitates transfer between differing computer systems. This allows a tactical situation model to be created and verified on a prototyping system before it is transferred to a faster, simulator-type system.

The user has been able to reduce his model construction time from 1-4 weeks down to less than a day.

Conclusions

The success of a modeling system is directly related to how well the system presents the model abstraction to the user. Levels of abstraction within the model help the user to organize his activities in a more logical manner. The modeling environment developed during this effort employs two levels of abstraction: a lower one for modeling objects (mountains and threat envelopes); a higher one for modeling the entire tactical region made up of terrain, hostile threat regions, and a projected flightpath.

The data representation chosen for a model is closely related to abstraction levels used within the model. Careful consideration must be given to the choice of data representation as the performance of the modeling system in terms of speed and ease of use is directly related.

Recommendations

Software Enhancements. As with any software project, a number of improvements could be made to the applications developed during this effort.

Program "Model" should be modified to include a graphics interface so that profiles can be interactively defined using the mouse input device. This would speed the profile entry process and would provide immediate visual feedback to the user as well.

The modeling tool, "Layout" should display the number of polygons represented by the object icons on the terrain grid. This would give the user an idea of the model complexity or size as he builds.

Program "See" should be modified to support more than seven colors in the model. The color table could be optimized to accomodate this change by removing the color table entries lower than the ambient level specified by the user.

Another area needing improvement for this program involves the time needed to render a Gouraud-shaded, z-buffered image. It is currently much too long.

Further Research. A number of interesting areas of research have arisen from this effort.

The interaction between terrain features and threat envelopes was not modeled in any of the developed applications. Terrain masking is one such interaction that refers to a terrain feature effectively blocking the range of a hostile threat on one or more sides. The result is that the threat envelope is no longer symmetrical as it must conform to the terrain around it. This would be desirable

effect to model as it is important to a pilot flying a tactical mission.

One potential method to model this effect involves the interaction of the two procedural models that build the terrain feature and threat. Procedural geometric model research has been done [1] in the area of communicating procedures that may modify each other as they execute.

Another area that could be followed up is the use of Defense Mapping Agency (DMA) data to model the terrain in place of the mountain objects used in this effort. [4], [16], and [8] all discuss some aspect of using DMA data for terrain modeling

To conclude, the applications developed during this thesis effort will be used by the Air Force researchers involved with the MAGIC Cockpit. AFWAL's Super Cockpit Project is another program that could benefit from further work on these applications. Segments of the Air Force System Command's Forecast II Initiative are also related to the research done during this thesis effort.

Appendix A: Additional Software Tools

This appendix outlines two additional software tools, "translate" and "mix", that were developed during this thesis effort. Additionally, two tools not developed as part of this effort are also discussed. These latter two tools were delivered to the user to aid in model viewing. The purpose of each tool will be briefly described.

Tool #1: Translate

"Translate" is a tool that will translate all polygons in a DR 1B or DR 2B file (see Figure 4) by some X, Y, and/or Z displacement specified by the user.

The program is invoked by typing "translate [RET]" at the system prompt. The user will be prompted for the filename (minus the ".poly" extension) of the object to be translated. Then the user will be prompted for an output filename, again without the extension.

Next, the user will be prompted for the X-Y-Z offset or translation values in the form:

Xvalue Yvalue Zvalue [RET]

Translate simply adds the user-specified X-, Y-, and Z-displacements to all X-Y-Z coordinate values in the input file and then writes the new data to the output file. All comment lines will be stripped and discarded from the input file. The user is prompted for a new comment line for the output file.

Tool #2: Mix

"Mix" is a tool that will combine two DR 1B or DR 2B files. This capability might be used to combine two threat envelopes so that they are concentric about the z-axis.

The program is invoked by typing "mix [RET]" at the system prompt. The user will be prompted for the filenames (minus the ".poly" extension) of both objects that are to be combined. Then the user will be prompted for an output filename, again without the extension.

The program strips off all comment lines from both input files. The user is prompted for a new comment line for the output file.

Tool #3: Treemaker

This tool takes a polygon file as input. It then generates a binary space partitioning tree of polygons so that the model can be rendered from back-to-front with a viewing program (Tool #4, below).

An example of this tool being used can be found in Appendix B.

Tool #4: Bspview

This tool takes a binary space partition tree format prepared by the previous tool as input. It allows the user to dynamically change the viewpoint when viewing a model, effectively providing simulated movement through the model. Control is provided via the mouse input device and allows adjustments in simulated pitch and yaw in addition to

forward and backward movement. Simulated velocity can be adjusted, as can the angular rate for the simulated pitch and yaw. Additionally, absolute position of the viewpoint in world coordinates can be viewed at any time.

An example of this tool being used can be found in Appendix B.

Appendix B: Model Building Session Example

This appendix will outline a complete modeling session using software tools developed during this thesis effort. Most of the text is taken from an actual modeling session. As such, system prompts ("[xxx]davinci ") are shown along with some Unix commands. Further, there where some previously-created profiles and surfaces used in this session.

All user responses will be underlined for clarity.

"[RET]" signifies pressing the RETURN key.

All commands are executed from a directory that will be defined as the home directory. This directory will be specified as "./" following the Unix file system convention. The home directory is assumed to contain certain programs and directories for this discussion:

```
[202]davinci ls [RET]
bspview*      model*      see*
database.out   mountains.in  threats.in
layout*       poly/       treemaker*
lo/           prof/
```

Constructing Profiles and Surfaces

Before any tactical situation models can be constructed, the objects to be placed in the model must first be created. Program "Model" is used to create profiles which are used to created both threat envelopes and mountains.

Program Conventions. The first convention to recognize is that the assumed unit of measure for this tool is feet. This is important because the modeling tool, "layout", converts to nautical miles (6000 ft/nautical mile).

Secondly, the order of specified profile points is important as the tool assumes the "outside" of a profile lies in the x-z plane to the right of the line segments defining the profile when traversed from first to last (see Figure 5). The reasons for this are discussed in the main body of this document.

Operation. After invoking Model from the system prompt, the screen clears and the main menu is displayed.

[203]davinci model [RET]

***** TACTICAL SITUATION MODEL GENERATOR *****

----- MAIN MENU -----

- 0> EXIT
- 1> Profile Directory
- 2> Load Profile
- 3> Store Profile
- 4> List Profile
- 5> Construct Profile
- 6> Revolve Profile
- 7> Extrude Profile
- 8> Surface Directory

Enter menu choice -->

The user can view the names of all existing profiles (if any) by selecting "1". This will effectively do a directory listing of directory ./prof.

Enter menu choice --> 1 [RET]

Existing profile files:

bigaaa.prof	fltmt2.prof	samred.prof
bigsam.prof	mountp.prof	samyellow.prof
flatmt.prof	mountp2.prof	

RETURN to Continue... [RET]

At this point, control is returned to the main menu, which will not be shown below in the interest of space.

If none of the already-existing profiles are suitable for creation of a new object, the user can choose to create a new profile by entering "5" at the main menu prompt. After a profile is defined, the user can optionally save the profile into directory ./prof with an optional comment. Even if the user does not elect to save the profile, it can still be used to generate a surface, as it remains the current profile until another is loaded or defined.

Enter menu choice --> 5 [RET]

Will all points be the same color? (y/n) --> y [RET]

Enter red green blue
components for all points --> 1.0 0.0 0.0 [RET]

Max points = 100.

Point #1: Enter X and Z or RETURN to quit --> 25 0 [RET]

Point #2: Enter X and Z or RETURN to quit --> 15 10 [RET]

Point #3: Enter X and Z or RETURN to quit --> 10 18 [RET]

Point #4: Enter X and Z or RETURN to quit --> 0 22 [RET]

Point #5: Enter X and Z or RETURN to quit --> [RET]

Confirm quit? (y/n) --> y [RET]

Save profile to file? (y/n) --> y [RET]

Enter profile name (without .extension) --> example [RET]

Enter comment up to 65 chars.

Comment --> This is an example of making a profile [RET]

A new file named "example.prof" now exists under directory ./prof. The extension ".poly" was automatically appended to the filename input by the user.

Control will again be returned to the main menu.

Responding "4" to the main menu prompt will allow the user to view the current profile, if desired.

Enter menu choice --> 4 [RET]

Pt#	X	Z	red	green	blue
1	25.00	0.00	1.0000	0.0000	0.0000
2	15.00	10.00	1.0000	0.0000	0.0000
3	10.00	18.00	1.0000	0.0000	0.0000
4	0.00	22.00	1.0000	0.0000	0.0000

RETURN to Continue...[RET]

Now the current profile can be used to generate a surface of revolution by entering "6" at the main menu prompt. Only the number of sectors must be specified. Then the resulting surface will can be saved to directory ./poly under a name specified by the user. The extension ".poly" will automatically be appended to the filename input by the user.

Enter menu choice --> 6 [RET]

4 points in profile: 3 polygons per sector.

Sectors per revolution? ----> 5 [RET]

15 polygons will be generated.

Enter name to save under (without .extention) testsurf [RET]

Enter comment up to 65 chars.

Comment --> From example.prof [RET]

Processing...Complete

The input sequence for a surface of extrusion is identical.

To verify the existence of this new object, "8" is entered at the main menu prompt. This will effectively do a directory listing of directory ./poly.

Enter menu choice --> 8 [RET]

Existing polygon files:

baaa.poly	mnt4ns.poly	mnt8r3.poly	sam8.poly
bsam8.poly	mnt5ns.poly	mnt8r4.poly	samred.poly
mnt10r4.poly	mnt6r3.poly	mount3.poly	samyellow.poly
mnt3ns.poly	mnt6r4.poly	mounts.poly	testsurf.poly

RETURN to Continue...[RET]

The user continues defining profiles and surfaces until he has created all the objects he intends to place into the tactical situation model. When done, the user selects "0" from the main menu.

Enter menu choice --> 0 [RET]

Program ended--normal termination.

The use of descriptive names for the created objects is important as these are the names that will appear in the

modeling tool menus. If the user wishes, he may move down into directory ./poly and add comment lines to any of these object files. The only restrictions are that the comment lines start with "//", that they are placed at the beginning of the file only, and that they are no longer than 65 characters per line. There is no limit to the number of comment lines that may be added.

Model Generation

Now that a suitable collection of objects exists, the user can proceed to construct a model by using the modeling tool "layout".

The first step in the modeling process is to let the modeling tool know which objects (threats and mountains) the user wants to be able to add to the model. This is done via two files that list objects to be used for a modeling session. The files need only be changed when the user wishes to add or delete objects from the lists. Any text editor can be used to modify the files.

The contents of each of files is simply a list of filenames for objects that are contained in directory ./poly. File "threats.in" lists threat objects while file "mountains.in" lists the mountain objects. Both exist in the home directory.

```
[207]davinci cat threats.in [RET]
baaa.poly
samred.poly
samyellow.poly
bsam8.poly
```

```
[208]davinci cat mountains.in [RET]
mnt6r3.poly
mnt6r4.poly
mount3.poly
mnt3ns.poly
```

Obviously, the names of these objects must have some meaning to the user. Above, "mnt6r4" indicated that the particular mountain was 6000 feet high and had 4 sides. Any convention can be used, just so it's consistent. The object "testsurf" that we created earlier could be added to either of these files; most likely to the mountain file, since the profile looks like that of a mountain. It would appear as "testsurf.poly" in file "mountains.in". Order of the list entries is not important, although they will appear in the same order in the modeling tool menu entry.

Even though "testsurf" would appear as a 25-foot mountain (i.e., a bump) in a model scaled to nautical miles, let's assume we wish to use it as a mountain object. Its name is added to the input list file which will then look like:

```
[209]davinci cat mountains.in [RET]
mnt6r3.poly
mnt6r4.poly
mount3.poly
mnt3ns.poly
testsurf.poly
```

Now the modeling tool must be made aware of the new object the user has added to the list. This is done by using the "setup" specification when invoking the modeling

NO-A190 484

GEOMETRIC MODELING OF FLIGHT INFORMATION FOR GRAPHICAL
COCKPIT DISPLAY(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
M A KANKO DEC 87 AFIT/GCE/ENG/87D-6

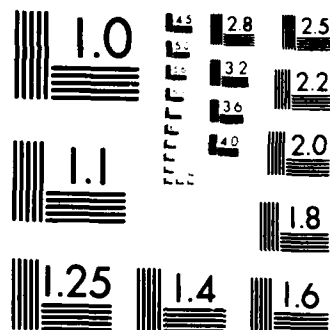
2/2

UNCLASSIFIED

F/G 1/4

ML





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

tool. This option need only be used when the user has added or deleted objects from the list files.

```
[210]davinci layout setup [RET]
```

Setting up layout parameters...

Setup complete.

The modeling tool is now ready for a modeling session. The user may proceed by specifying the size of the region to be modeled as follows:

```
[211]davinci layout 100 150 [RET]
```

This specification means that the region extends 100 miles to the east and 150 north of the origin.

Once this command is issued, the user utilizes the graphical interface as described in the implementation chapter of this document. Assuming the user wishes to create a polygon file as a result of the modeling session, he will select "Create Database" from the main menu. The polygon file will be stored to file "database.out" in the home directory (more on this file shortly). When the user exits the modeling program, it will prompt him for a save name for the symbolic form of the model.

```
Enter filename to save session  
under [session.10]--> 10/example.10 [RET]
```

This will save the symbolic model to directory ./10. The model can now be read into the modeling tool by specifying it at invocation:

[212]davinci layout lo/example.lo [RET]

If no save name is specified, the symbolic model will automatically be saved to file "session.lo" in the home directory. "session.lo" can similarly be used as input to the modeling tool.

Model Viewing

The model may be viewed in two ways. If the user wishes to view a Gouraud shaded image, program "see" must be used. If he instead wishes to dynamically move through the model, "hspview" must be used. These tools will be discussed in this order.

Gouraud Shaded Viewing. Before "see" can be used to view the model, the polygon file must be copied or moved to directory ./poly.

[213]davinci cp database.out poly/testmodel.poly [RET]
[214]davinci see [RET]

The main menu for program "see" is shown in Figure 14. It will not be shown below in the interest of space. Once the main menu is displayed, the user can load a model to be viewed by entering "1" after the menu prompt:

(L)oad model, (M)odify settings, (D)isplay model, or (Q)uit?

Enter choice --> 1 [RET]

Enter model name (without .extention)

or RETURN to quit --> testmodel [RET]

Loading model Complete

Scoping model Complete

Next the user will most likely wish to modify some of the position values. The view position is first changed, followed by the center of interest.

(L)oad model, (M)odify settings, (D)isplay model, or (Q)uit?

Enter choice --> m [RET]

*** MODIFIABLE DEFAULT VARIABLES ***

- 1) Viewpoint Position
- 2) Center of Interest Position
- 3) Light Source Position
- 4) Ambient Light Component
- 5) Type of Shading
- 6) Store Defaults

Enter item # or RETURN to quit --> 1 [RET]

*** MODIFY VIEWPOINT ***

Current viewpoint position: X = 18000.0
Y = 0.0
Z = 5000.0

Enter new viewpoint (X Y Z)

or RETURN to continue -----> 0 30 10 [RET]

*** MODIFIABLE DEFAULT VARIABLES ***

- 1) Viewpoint Position
- 2) Center of Interest Position
- 3) Light Source Position
- 4) Ambient Light Component
- 5) Type of Shading
- 6) Store Defaults

Enter item # or RETURN to quit --> 2 [RET]

*** MODIFY CENTER OF INTEREST ***

Current center of interest position: X = 17000.0
Y = 40000.0
Z = 3000.0

Enter new center of interest (X Y Z)
or RETURN to continue -----> 0 0 5 [RET]

*** MODIFIABLE DEFAULT VARIABLES ***

- 1) Viewpoint Position
- 2) Center of Interest Position
- 3) Light Source Position
- 4) Ambient Light Component
- 5) Type of Shading
- 6) Store Defaults

Enter item # or RETURN to quit --> [RET]

All other settings can be modified likewise. Note that the modifiable settings can be saved by selecting "6" from the modification menu. These settings are saved to file ".defaults" in the home directory and will be in effect the next time "see" is invoked.

Control has now been returned to the main menu and the user is ready to view the model. After "d" is entered, a message will be displayed to remind the user how to exit the

viewing mode when done. The user must press return to view the model.

(L)oad model, (M)odify settings, (D)isplay model, or (Q)uit?

Enter choice --> d [RET]

After viewing scene, press ESC to return to main menu

Press RETURN to view model ...[RET]

After the user is done viewing the model he may exit the program by entering "q" at the main menu prompt.

(L)oad model, (M)odify settings, (D)isplay model, or (Q)uit?

Enter choice --> q [RET]

Hope to 'see' you again soon!

[215]davinci

Dynamic Movement. To accomplish dynamic movement, the polygon file must first be organized into a binary space partitioned tree. This is done with program "treemaker" as follows:

[215]davinci treemaker [RET]

Name of polygon file: database.out [RET]

Enter number or polygons to check [5]: [RET]

Enter seed [0]: [RET]

Building tree...

Seed: 0 Test: 5 In: 466 Out: 886 Ratio: 1.90

Would you like an ASCII dump of the tree? n [RET]

Enter binary tree file name: testmodel.bsp [RET]

All done

[216]davinci

The above program output indicates that the input polygon file had 466 polygons and that the resulting binary space

partitioned tree contains 866 polygons. The increase occurs because some polygons in the original file are split into two or more polygons along bounding plane boundaries.

It would be helpful if a directory called "./bsp" were used to store the resulting bsp files. The binary tree file above would have then be saved as "bsp/testmodel.bsp". This is not necessary, but it tends to keep the home directory cleaner.

The model may now be viewed dynamically by invoking program "bspview". Beginning position and center of interest are specified by the user. Also, number of light sources, their positions, and the amount of ambient light present are specified, as well. Note that the light locations are actually specified as direction vectors from the origin to the light source, hence actual position is not necessary. This is reasonable, since a point light source (such as the sun) is usually considered to exist at an infinite distance, hence all light rays from the source are parallel to each other.

Once all starting parameters are specified, the main help screen is displayed. After pressing any key the user will see the image of the model.

```
[216]davinci bspview [RET]
Enter binary tree file name: testmodel.bsp [RET]
Initial eye position (3fp): 0 0 0 [RET]
Initial center of interest (3fp) : 10 10 0 [RET]
Number of lights: 2 [RET]
    location of light 1: 1 0 .1 [RET]
    location of light 2: -1 0 .25 [RET]
Ambient light: .2 [RET]
```

Hold down the indicated mouse buttons for constant motion:

- R turn right
- M- move forward
- L-- turn left
- MR turn down
- LM- turn up
- L-R move backwards

Strike the indicated keys for state changes:

- ESC stop viewing model and return to this menu
- h help
- s increase speed
- d decrease speed
- w increase angular turn rate
- e decrease angular turn rate
- q quit the program

Press any key board character to continue

Eye position	0.0	0.0	0.0
center position	10.0	10.0	0.0

State changes only take effect when the image is actually being viewed. Pressing the escape key will return the user to the help screen and the viewer's position and center of interest can be noted. This is particularly helpful if the user wishes to view the same model from a specific position with the "see" program.

When the user is finished viewing the model, he presses "q" while viewing the image. This will return control to the operating system after writing a framecount message to the screen. This framecount indicates how many times the framebuffer was swapped to accomplish dynamic movement through the model.

Bibliography

1. Amburn, Phil, Eric Grant, and Turner Whitted. "Managing Geometric Complexity with Enhanced Procedural Models," SIGGRAPH '86 Proceedings, published as Computer Graphics, 20: 189-195 (November 4, 1986).
2. Bratko, Ivan. PROLOG Programming for Artificial Intelligence. Addison-Wesley, Wokingham, England, 1986.
3. Carlson, Wayne E. Techniques for the Generation of Three Dimensional Data for Use in Complex Image Synthesis. PhD dissertation. Ohio State University, Columbus, Ohio, 1982.
4. Clark, Charles L. and Michael R. Pafford. "Geographic Subdivision and Top Level Data Structures: Columbus, Magellan, and Expanding CIG Horizons," (AD-P004 317), The Image III Conference Proceedings Held at Phoenix, Arizona on 30 May - 1 June 1984, (Proceedings Paper), 129-149, Cameron Station, Alexandria, Virginia: Defense Technical Information Center, (AD-A148 636).
5. Costenbader, J. L. "CIG Data Bases in an Instance: Bits and Pieces," (AD-P004 318), The Image III Conference Proceedings Held at Phoenix, Arizona on 30 May - 1 June 1984, (Proceedings Paper), 151-163, Cameron Station, Alexandria, Virginia: Defense Technical Information Center, (AD-A148 636).
6. Foley, James D. and Andries Van Dam. Fundamentals of Interactive Computer Graphics. Addison-Wesley, Reading, Massachusetts, 1982.
7. Gouraud, H. Computer Display of Curved Surfaces. PhD disseration. UTEC-CSc-71-113, NTIS AD-762 018, University of Utah Computer Science Dept., Salt Lake City, Utah, June 1971.
8. Haas, Manfred, Diether Elflein, and Peter M. Gueldenpfennig. "Data Base Generation System for Computer Generated Images and Digital Radar Landmass Simulation Systems," (AD-P000 182), Proceedings of the 4th Interservice/Industry Training Equipment Conference, 16-18 November, 1982, Volume I, (Proceedings Paper), 231-235, Cameron Station, Alexandria, Virginia: Defense Technical Information Center, (AD-A122 155).

9. Hearn, Donald, and M. Pauline Baker. Computer Graphics. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
10. Jauer, R. A., and T. J. Quinn. "Pictorial Formats, Vol. 1: Format Development." Air Force Wright Aeronautical Laboratories Technical Report AFWAL-TR-81-3156. Wright-Patterson Air Force Base, Ohio: February 1982.
11. Newell, Martin E. The Utilization of Procedure Models in Digital Image Synthesis. PhD dissertation. University of Utah, Salt Lake City, Utah, 1975.
12. Reising, John M. and Carol Jean Kopala. "Cockpit Applications of Computer Graphics." Report for Harvard Computer Graphics Week, Harvard University, Graduate School of Design, 1982.
13. Smart, Donald D., Richard D. Teichgraeber and Anthony C. Chirieleison. "The Generation of Three-Dimensional Data Bases Using a Building Block Approach" (AD-P004 319), The Image III Conference Proceedings Held at Phoenix, Arizona on 30 May - 1 June 1984, (Proceedings Paper), 165-176, Cameron Station, Alexandria, Virginia: Defense Technical Information Center, (AD-A148 636).
14. Townsend, Barbara, Manager for Modeling Tools. Personal Interview, Evans & Sutherland Computer Corporation, Salt Lake City, Utah, 20 Aug 1987.
15. Wailes, Capt Tom S. Placing Hidden Surface Removal Within The Core Graphics Standard: An Example. MS Thesis, AFIT/GCS/ENC/83D-9. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, December 1983.
16. Way, T. C., M. E. Hornsby, J. D. Gilmour, R. E. Edwards, and R. E. Hobbs. "Pictorial Format Display Evaluation." Air Force Wright Aeronautical Laboratories Technical Report AFWAL-TR-84-3036. Wright-Patterson AFB, Ohio, May 1984.
17. Widder, Patricia A. and Clarence W. Stephens. "Data Base Generation: Improving the State-of-the-Art," (AD-P003 470), Proceedings of the Interservice/Industry Training Equipment Conference (5th) Held at Washington, D. C. on November 14-16, 1983. Volume 1, (Proceedings Paper), 164-170, Cameron Station, Alexandria, Virginia: Defense Technical Information Center, (AD-A142 774).

18. Yan, Johnson K. "Advances in Computer-Generated Imagery for Flight Simulation," IEEE Computer Graphics & Applications, 9: 37-51 (August 1985).

Vita

Captain Mark A. Kanko was born on January 30, 1960 in Minot, North Dakota. He graduated from Butte Public High School in Butte, North Dakota in 1978 and entered North Dakota State University in Fargo, North Dakota. He graduated in 1982 with the Bachelor of Science in electrical engineering specializing in computer engineering. Captain Kanko served three and a half years at NASA's Johnson Space Center in Houston, Texas before entering the School of Engineering, Air Force Institute of Technology in June 1986. He is married to Annette (Lindaas) Kanko of Mayville, North Dakota and they have three children: Delayna, Alexander, and Emma.

Next military address: ASD/ENASC

WPAFB, OH 45433

Comm 513-255-2262

Permanent address: Box 314

Butte, ND 58723

SECURITY CLASSIFICATION OF THIS PAGE

Form Approved
OMB No. 0704-0188

UNCLASSIFIED

(Continued from front, Block 18)

Three-dimensional Model, Terrain Model, Modeling Environment, Procedural Model, Surface of Revolution

(Continued from front, Block 19)

The purpose of this investigation was to design and implement a graphics-based environment capable of modeling tactical situation arenas as viewed from the cockpit. The modeled arena or region was composed of mountains, hostile threat envelopes, and a projected flightpath through the region. The resulting displays were to be used in the Microprocessor-Based Application of Graphics Interactive Communication (MAGIC) Cockpit owned by the Crew Systems Development Branch within the U.S. Air Force Flight Dynamics Lab at Wright-Patterson Air Force Base. This cockpit is used to prototype new graphical display formats that might be used in future aircraft.

The individual three-dimensional objects used to represent threats and mountains in the model were generated by geometric procedural models. A strongly-parameterized procedural model would generate a three-dimensional surface of revolution composed of polygons from a two-dimensional profile input by the user. Once defined, each object could then be instantiated into the model representing the complete tactical situation. Positioning of objects in the model was accomplished via a mouse input device.

The implemented data representation allowed the model to be easily modifiable. Additionally, the model could be stored in a machine-independent form to assure portability.

An overall goal of this investigation was to allow the cockpit display researcher to create an entirely new tactical situation display model in less than one hour.

The applications comprising the modeling environment were written in the C programming language and were hosted on a Silicon Graphics IRIS 3130 graphics workstation.

END

DATE

FILMED

APRIL

1988

DTIC