

AD-A262 598



(1)



DTIC
ELECTE
APR 05 1993
S B D

Enhancements to the Animated Tutor for
Air Combat Simulation

THESIS
David Allan Legge
Capt, USAF

AFIT/GOR/FNS/93M-12

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

Reproduced From
Best Available Copy

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

20001006135

AFIT/GOR/ENS/93M-12

Enhancements to the Animated Tutor for
Air Combat Simulation

THESIS
David Allan Legge
Capt, USAF

AFIT/GOR/ENS/93M-12

Approved for public release; distribution unlimited

93 4 02 169

93-07020



14824

THESIS APPROVAL

STUDENT: Capt David A. Legge

CLASS: GOR-93M

THESIS TITLE: Enhancements to the Animated Tutor for Air Combat Simulation

DEFENSE DATE: February 23, 1993

COMMITTEE:	NAME/DEPARTMENT	SIGNATURE
Advisor	Maj Edward W. Negrelli /ENS	<i>Edward W. Negrelli</i>
Reader	Maj Eric F. Christensen/ENG	<i>Eric F. Christensen</i>

DTIC QUALITY INSPECTED 4

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

AFIT/GOR/ENS/93M-12

**Enhancements to the Animated Tutor for
Air Combat Simulation**

THESIS

**Presented to the Faculty of the School of Operational Sciences
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Masters of Science in Operations Research**

**David Allan Legge, B.S.
Capt, USAF**

March, 1993

Approved for public release; distribution unlimited

Acknowledgements

First and foremost I would like to thank my wife Elizabeth. Her patience these last eighteen months has been immeasurable. Not only has she been a great help editing this thesis, but she endured countless hours of "shop talk" with my classmates. For her patience and support I am eternally grateful. Secondly, I would like to thank my classmates for their support and camaraderie. They have made my AFIT experience truly memorable. Lastly, I would like to thank my thesis advisor Major Ed Negrelli and my reader Major Eric Christensen for their assistance. Through them I have learned more about the "real" world of combat modeling than any class could teach.

David Allan Legge

Table of Contents

	Page
Acknowledgements	ii
List of Figures	iv
List of Tables	v
Abstract	vi
I. Introduction	1-1
1.1 Background	1-1
1.2 Problem Statement	1-2
1.3 Thesis Objective	1-2
1.4 Scope and Limitations	1-2
1.5 Definitions	1-3
II. Literature Review	2-1
2.1 Introduction	2-1
2.2 War Games	2-1
2.3 History of Ada	2-2
2.4 Ada Structures	2-3
2.5 Ada in Combat Modeling, Wargaming and Simulations	2-5
2.6 Combat Modeling on Personal Computers	2-5
2.7 Conclusion	2-6
III. Method	3-1
3.1 Introduction	3-1
3.2 Model Design	3-1

	Page
3.3 Graphics Methodology	3-4
3.4 Combat Model Methodology	3-5
3.5 Preprocessor Methodology	3-6
3.6 Documentation Methodology	3-7
3.7 Conclusion	3-7
IV. Program Design	4-1
4.1 Introduction	4-1
4.2 Overall Design	4-1
4.3 Modeling Decisions	4-3
4.3.1 Combat Processes	4-4
4.3.2 Time Advance Mechanism	4-6
4.3.3 Ccoordinate Systems	4-6
4.4 Randomness	4-7
4.5 Model Assumptions	4-8
4.5.1 Weapon Assumptions	4-9
4.5.2 Engagement Assumptions	4-9
4.5.3 Attrition Assumptions	4-10
4.6 Model Flow	4-10
4.7 Graphics Decision ^s and Assumptions	4-12
4.8 Graphics Flow	4-13
V. Program Details	5-1
5.1 Introduction	5-1
5.2 Entities	5-1
5.3 ATACS+ Aircraft	5-1
5.4 Aircraft Parameters	5-5
5.5 Adding Aircraft	5-8

	Page
5.6 Aerodynamics	5-8
5.7 Aircraft Motion	5-11
5.7.1 Maneuver Logic	5-11
5.7.2 Maneuver Descriptions	5-12
5.7.3 Motion Logic	5-18
5.8 Missile Motion	5-18
5.9 Random Number Generation	5-22
5.10 Combat Processes	5-23
5.10.1 GCI Search	5-23
5.10.2 Target Assignment	5-24
5.10.3 Aircraft Search	5-25
5.10.4 Aircraft Engagement	5-27
5.10.5 Other Combat Processes	5-27
VI. Conclusions and Recommendations	6-1
6.1 Introduction	6-1
6.2 Conclusions	6-1
6.2.1 Combat Modeling on the PC	6-1
6.2.2 Meridian Ada	6-2
6.2.3 Ada for Combat Modeling	6-3
6.3 Enhancements	6-5
6.4 Recommendations	6-6
6.5 Final Remarks	6-7
Appendix A. User's Guide to ATACS+	A-1
A.1 Introduction	A-1
A.2 Computer Requirements	A-2
A.3 Installing ATACS+	A-2

	Page
A.4 ATACS+ Components	A-4
A.4.1 The Preprocessor	A-4
A.4.2 The Combat Model	A-4
A.4.3 The Graphics Routines	A-4
A.5 Using the Preprocessor	A-5
A.5.1 Input Methods	A-5
A.5.2 The Main Menu	A-6
A.5.3 The Scenario Editor	A-7
A.5.4 The Aircraft Editor	A-10
A.6 Using the Simulation	A-16
A.6.1 The Run-time Menu.	A-17
A.6.2 2D Icons.	A-20
A.6.3 3D Icons	A-21
A.7 Example ATACS+ Session	A-21
A.8 Troubleshooting	A-27
A.9 Review Questions	A-28
A.10 Command Summary	A-31
Appendix B. ATACS+ SIMTAX Classification	B-1
Appendix C. XSharp in Ada	C-1
C.1 Introduction	C-1
C.2 Background	C-1
C.3 The 320x240 Mode	C-2
C.4 Page Flipping	C-3
C.5 Assembly Language Routines	C-5
C.6 Graphics Primitives	C-7
C.7 Future Enhancements	C-20
C.8 Conclusions	C-20

	Page
Bibliography	BIB-1
Vita	VITA-1

List of Figures

Figure	Page
3.1. Development Flow	3-2
4.1. ATACS+ System Structure	4-3
4.2. Earth Coordinate System	4-7
4.3. Aircraft Coordinate System	4-8
4.4. Combat Model Logic Flow	4-11
4.5. Graphics Logic Flow	4-14
5.1. Velocity-Load Factor Diagram	5-7
5.2. Maneuver Selection Logic	5-13
5.3. Initial Missile Defense Geometry	5-14
5.4. Close-in Missile Defense Geometry	5-14
5.5. Intercept Maneuver Geometry	5-16
5.6. Pursuit Maneuver Geometry	5-17
5.7. Attack Maneuver Geometry	5-17
5.8. Aircraft Motion Logic	5-19
5.9. Missile Motion Logic	5-20
5.10. Missile Closest Approach	5-21
5.11. Random Number Generation	5-22
5.12. Aircraft Search Geometry	5-25
A.1. Preprocessor Main Menu	A-6
A.2. Preprocessor Scenario Editor	A-8
A.3. Preprocessor Name Form	A-10
A.4. Preprocessor Aircraft Editor	A-11
A.5. Preprocessor Aircraft Menu	A-12

Figure	Page
A.6. Preprocessor Mission Menu	A-13
A.7. Preprocessor Rules of Engagement Menu	A-14
A.8. Simulation 2D Screen	A-17
A.9. Simulation 2D Runtime Menu	A-18
A.10. Simulation 3D Runtime Menu	A-19
A.11. Simulation 2D Icons	A-21
A.12. Simulation 3D Icon	A-22
C.1. Display Memory Organization	C-4
C.2. Page Flipping Methodology	C-5

List of Tables

Table	Page
5.1. Aircraft Entity Attributes	5-2
5.2. Aircraft Entity Attributes (cont'd)	5-3
5.3. Missile Entity Attributes	5-4
5.4. Airbase Entity Attributes	5-4
5.5. Aircraft Performance Parameters	5-6
A.1. ATACS+ Files	A-3

Abstract

This study investigates the use of the Ada programming language in combat modeling. It takes an existing combat model, the Animated Tutor for Air Combat Simulation (ATACS), and re-writes it into Ada. The new model also includes enhancements to the original model. ATACS+, as it is now called, is used as a learning tool in combat modeling classes at the Air Force Institute of Technology. ATACS+ consists of a preprocessor and a model, complete with two and three-dimensional graphics and engagement status reports. Conclusion are made about Ada's suitability for combat modeling, some of the features of Ada which can assist combat modelers and the feasibility of combat modeling on personal computers.

Enhancements to the Animated Tutor for Air Combat Simulation

I. Introduction

1.1 Background

"If one grants that modeling is and, for greatest effectiveness, probably ought to be, an intuitive process for the experienced, then the interesting question becomes the pedagogical problem of how to develop this intuition. What can be done for the inexperienced person who wishes to progress as quickly as he can toward a high level of intuitive effectiveness in management science?" (20:B-707)

The Air Force Institute of Technology (AFIT) is the Air Force center for teaching the art of combat modeling at the graduate level. AFIT also teaches combat modeling to a significant number of Army officers. As William T. Morris points out in the above quotation, the problem in teaching modeling is how to develop an intuitive feeling for modeling without actual experience. AFIT solves this problem by exposing students to the basic processes inherent in many combat models. The students are required to investigate combat models which are currently in use, and report on the structures, processes, assumptions, limitations, and algorithms. Unfortunately, most combat models are extremely large and complicated. Also, few are available at AFIT for the student's experimentation. What is needed is a combat model which demonstrates the essential components of combat models but is small enough to allow students to change input parameters and observe their effects. Recognizing this need, Captain Richard Moore, AFIT 1992, developed the Animated Tutor for Air Combat Simulation (ATACS) as a thesis project. It was a first step toward developing an educational aid that could be used in combat modeling classes. But a second problem, now impacting the combat modeling field, was

not addressed; the use of the Ada programming language. Since, April 1987 the Department of Defense (DoD) has mandated that all software development be done in the Ada programming language. This includes combat models. So now in addition to teaching the techniques of combat modeling, AFIT would also like to expose students to Ada-based combat modeling. This thesis solves this problem by rewriting the original ATACS into Ada, and at the same time adding several enhancements.

1.2 Problem Statement

Even though Ada is the mandated Department of Defense computer programming language, all of the military services, and the Air Force in particular, lack analysts who are skilled and experienced in Ada-based combat modeling. Also, there are no Ada-based tools for teaching analysts combat modeling techniques.

1.3 Thesis Objective

The objective of this thesis was to develop a combat model, written in Ada, that could be used as a learning tool at the Air Force Institute of Technology.

1.4 Scope and Limitations

This thesis describes the conceptualization, design and implementation of a simple high-resolution combat model which will be used as a learning tool. This thesis is only a proof-of-concept not a full-scale combat model used for analysis. It does not attempt to recreate any portion of actual air-to-air combat. Instead, it is intended to illustrate some of the major processes common to most high-resolution combat models.

This model builds upon the existing ATACS combat model, adding the following enhancements:

1. *Rewriting of the ATACS model into Ada.* This thesis will re-build the existing ATACS model into an Ada-based learning tool. It will be a proof-of-concept model demonstrating combat modeling techniques.
2. *Addition of three-dimensional aircraft maneuvering.* This thesis will convert the two-dimensional maneuvering aircraft currently in ATACS to three-dimensional maneuvering aircraft having five degrees of freedom motion.
3. *Addition of reactive "Red" forces.* The current non-maneuvering attacking aircraft will be replaced with full motion aircraft.
4. *Create three-dimensional graphics displays.* The current QuickBASIC graphics routines will be replaced by routines written in Ada and Assembly Language.

The completed model is designed to run on an IBM compatible personal computer which has a Video Graphics Array (VGA) graphics card installed. Also, a math co-processor is required. Finally, the program is written to allow further enhancements and expansions. It is intended to be a starting point from which future students can add processes and enhance the existing routines.

1.5 Definitions

Before beginning a review of the current literature on Ada and its use in combat models, several common terms need to be defined. These definitions are provided to clarify subtle differences and to provide a common vocabulary between the author and the reader.

War Game. A simulation, by whatever means, of a military operation involving two or more opposing forces, using rules, data, and procedures designed to depict an actual or assumed real life situation. (16:227)

Ada is a programming language designed in accordance with requirements defined by the United States Department of Defense. ...the language is a modern algorithmic language with the usual control structures, and with the ability to define types and subprograms. It also serves the need for modularity... (8:1-1)

High Resolution Combat Model. A model which includes detailed interaction of individual combatants or weapon systems. Each combatant in a high resolution model has its own vector of state variables which describe its unique situation and its unique perception of the battlefield as the battle progresses. Interactions among combatants are resolved at the one-on-one engagement level - often computing separately the results of each individual shot fired in the battle. The engagement models include terrain and environmental effects as well as the states of firer and target. (13:Sec 1-6)

II. Literature Review

2.1 Introduction

This chapter reviews the pertinent literature used in researching this thesis. It does not cover material already reviewed in Captain Richard Moore's thesis, such as computer aided instruction, computer graphics for instructional purposes, classification of models and a detailed description of combat processes. Instead this review covers the information required to meet the objectives listed in Chapter I.

This chapter covers five major areas. First a brief overview of wargames and the difficulties they create is presented. This is to give the reader an appreciation for the intent of this thesis. The second section presents the history of the Ada programming language. It covers the reasons behind its creation and how it came to be the mandated language of the DoD. The third section reviews the major features of the Ada language. It also describes how these features speed development, increase reliability and enhance large projects. The fourth section discusses Ada's use in combat modeling, wargaming and simulation. It shows Ada's limited use for these purposes. The fifth section examines the possibility of combat modeling on a personal computer. It is only now becoming practical.

2.2 War Games

War games, simulations and combat models are used extensively in the DoD. *The Catalog of Wargaming and Military Simulation Models* has entries on over 500 different war games and simulations (15). A war game normally has one or more humans who participate, or interact, with the model. Conversely, many simulations run without the need of human inputs beyond the creation of the input database (14:42). Both simulations and war games are an abstraction of the real world where a likeness of some portion of reality is created for either an investigation or resource management. Simply put, models are simplifications of reality used to assist in

making decisions (14:3). These models can encompass a vast scenario such as a nuclear exchange or minute details such as the inner working of individual atoms. Military combat models range in scope from a one-on-one engagement to global force interactions (3:18). As the scale of combat increases, the level of engineering detail decreases. A one-on-one model requires a great deal of engineering detail. At this lowest level, the dynamics of individual entities is the modeling goal. Tactics and force interactions begin to play an increasingly important role as the scale of combat increases. This increased complexity and size has caused some models to outgrow their usefulness. They can no longer be maintained and their results have become suspect. Many war games and military simulations are written in FORTRAN or Assembly language, two programming languages which have inferior maintainability, structure and performance than Ada.

2.3 History of Ada

Ada is a programming language designed specifically to be strongly standardized and should result in more reliable and portable software (10:26). It was originally intended for use in embedded systems, computers which are part of a larger system such as missile guidance computers. But the resulting language solves an even wider range of software development problems (29:45). In 1973, the DoD was spending over \$3 billion dollars on software development and maintenance. This amount was projected to grow to over \$30 billion dollars by 1990. Over half of all software developed and maintained by the DoD was for embedded systems. In addition to the escalating costs, lack of a standard language had become a serious problem. Embedded computers employed over 450 different programming languages (29:43).

To halt the rise in costs and to promote a standard language, a joint-service High-Order Language Working Group was formed in January 1975. This group's purpose was three-fold: identify requirements for DoD high-order languages (lan-

languages where the code resembles written English, not computer machine language), evaluate existing languages against the requirements, and recommend a minimal set of languages for DoD use. After establishing the requirements, the working group found that no existing language could meet them. So in April, 1977 the DoD sponsored an international competition to design a new programming language. This new language was named "Ada" upon completion of the competition in May 1979 (30:11). Ada is named in honor of Augusta Ada Byron, Countess of Lovelace, who is reported to have been the first programmer (29:44).

The first Ada language reference manual was published in August 1980. By February 1983 this manual had been accepted as an ANSI (American National Standards Institute) standard and became ANSI/MIL-STD-1815A (30:11-12). Unlike other language standards which are voluntary and define a basic set of language functions, the Ada standard forces programs to be written in a single common language. This language must be the same without regard to the kind of computer or the manufacturer of the compiler (10:27). As of 30 March 1987, DoD Directive 3405.1 requires that all future software development for embedded systems be done in Ada (18:7).

2.4 Ada Structures

The Ada standard provides many of the constructs common to high-order languages. Including such things as if-then-else structures, loops, Boolean (True-False) operators, functions and subprograms. Ada also has constructs which distinguish it from other high-order languages such as C, Pascal, FORTRAN, and BASIC. Some of these constructs are described below.

Strongly typed design. Strong typing means that the range of values a variable can have must be defined. This increases Ada's ability to detect errors at compile time (18:7-8).

Packages. A collection of related functions, variables, subprograms etc. is called a package. Using packages, a large program can be broken down into smaller units. A package normally consists of a package specification and a package body. The specification defines the interface to the package and is visible to the user. On the other hand, the package body is normally hidden from the user and contains the implementing code. This is one of Ada's most important features (18:8).

Data abstraction and information hiding. Ada is very good at compartmentalizing data. Each function, procedure or package can have data or routines which only it has access to. This allows complex code to be presented to the user in a simplified manner. The user does not see the complex portion only the simplified interface (18:9).

Separate compilation. Modularity of a large program can only be achieved if the modules can be compiled separately. Thus a simple change does not require re-compiling of the entire program. Ada takes this concept one step further by allowing package specifications to be compiled before package bodies are written. This ability is extremely important in large projects where multiple programmers are working together (29:9).

Excellent environment for development of reusable code. Because of Ada's modularity and separate compilation abilities, most code can be reused in other programs. The development of libraries of commonly used routines speeds software development (29:9).

Built-in exception handling. When unexpected situations occur, like dividing by zero, an exception is raised in Ada. The programmer is allowed to "catch" these exceptions and resolve the situation rather than having the program terminate (18:9-10).

Run-time error checking. Very few programs allow the checking of data at run-time. This feature allows Ada to handle instances where inputted values are of

the wrong type or computed values are out of range. This greatly improves program reliability (18:10).

Support for parallel processing. Ada has built-in support for parallel processing. Programs can be broken up into individual tasks and each of these tasks can then be executed in parallel on a separate processor. Parallel processing holds the key to vastly improved computational speeds (18:10).

2.5 Ada in Combat Modeling, Wargaming and Simulations

Ada's impact on war gaming and simulation is just beginning to be felt. Of the 537 war games and simulations listed in *The Catalog of Wargames and Military Simulation Models*, only thirteen of them are written, either all or in part, in Ada. Only one is designed to run on an IBM-compatible personal computer (PC). Most of the Ada-based models are written to support Strategic Defense Initiative Office (SDIO) requirements or are produced by the United States Army. The Air Force lacks experience and expertise in using Ada for combat modeling and simulation (15:).

2.6 Combat Modeling on Personal Computers

Until recently combat modeling was done on mainframe and mini-computers. The data storage, program size and computational requirements were beyond the abilities of a PC. All of that has now changed. PC's are fast, cheap, have large memories, expanded graphics and are available to nearly every combat modeling analyst.

When the original IBM-PC was introduced in August 1981 it boasted 16 kilobytes (16,384 characters) of user memory. Ten years later, IBM's top-of-the-line desktop system, the PS/2 model 90, has a standard 8 megabytes of user memory (26:336). Other manufacturers allow memory configurations up to 32 megabytes. Obviously, program size is not the problem it once was.

Speed has also increased. The IBM-PC used an Intel 8088 microprocessor chip and ran at an operating speed of 4.77 MHz. Today Intel has produced a 80486 chip which runs at 100 MHz (26:336). The personal computer now has the computing power to handle many tasks formerly performed on mini-computers and mainframes.

The hallmark of many mini-computer-based combat models has been the graphical displays accompanying them. The personal computer is no match for today's graphics workstations, but with 256 colors and resolutions reaching 1280x1024 pixels (a pixel is the smallest 'dot' which can be drawn on a screen) it is possible to create animated three dimensional displays to accompany PC-based combat models (26:336).

Maybe the most amazing aspect of today's PC is the price. In 1981 an IBM-PC had a suggested price of \$1565. This included a monochrome monitor, no hard disk storage device and the 16 kilobyte memory mentioned earlier. That same \$1565 today would buy an 80386-based computer running at 20MHz with a hard drive and VGA color graphics. Today's high-end PCs rival workstations in speed, and cost considerably less.

But the single factor which makes combat modeling on a personal computer possible is their availability. Gone are the days when organizations have to pay for each minute of computer usage. With the exception of very large programs, most programs can be run right on an analyst's desk. The accessibility of the PC makes it a cheap convenient way to do combat modeling.

2.7 Conclusion

The United States Air Force needs to focus its combat modeling and wargaming efforts on the Ada language, not only because of the DoD directive, but because of the unique features of the language. Ada forces programmers to use efficient, easily modified, modular programming methods. This will pay dividends later when decision-makers ask for more complex analysis. The models will be ready to go,

at minimal cost, in minimal time and with some assurance that the results are defensible.

III. Method

3.1 Introduction

This chapter describes the methodology used to develop the ATACS+ system. The flow chart in Figure 3.1 shows the general sequence of the development process. However, this process was not as straight forward as Figure 3.1 might suggest. At each stage, restructuring of previous work was done as additional requirements or restrictions became apparent. The flow depicted in Figure 3.1 does, however, provide a means to discuss how ATACS+ was developed. As the chart shows, the development process progressed through five stages: model design, development of the animation graphics, development of the combat model, development of the pre-processor, and the on-going task of documentation. Each of these stages is described in the following sections.

3.2 Model Design

The first task undertaken in this thesis was to quantify the real nature of the problem and to determine how the combat model should be designed to solve it. The objective is stated in Chapter 1 as:

... to develop a combat model, written in Ada, that could be used as a learning tool at the Air Force Institute of Technology.

This objective mandates the use of a combat model as the best solution and stipulates that this model should be created using the Ada programming language. This objective provides a solid starting point but since this thesis is also a proof-of-concept in using Ada as a combat modeling language on a personal computer, several other objectives needed to be defined. These objectives and the design considerations stemming from them are:

- ATACS+ must be able to run on an IBM-compatible personal computer in order to allow as many students as possible to use it.

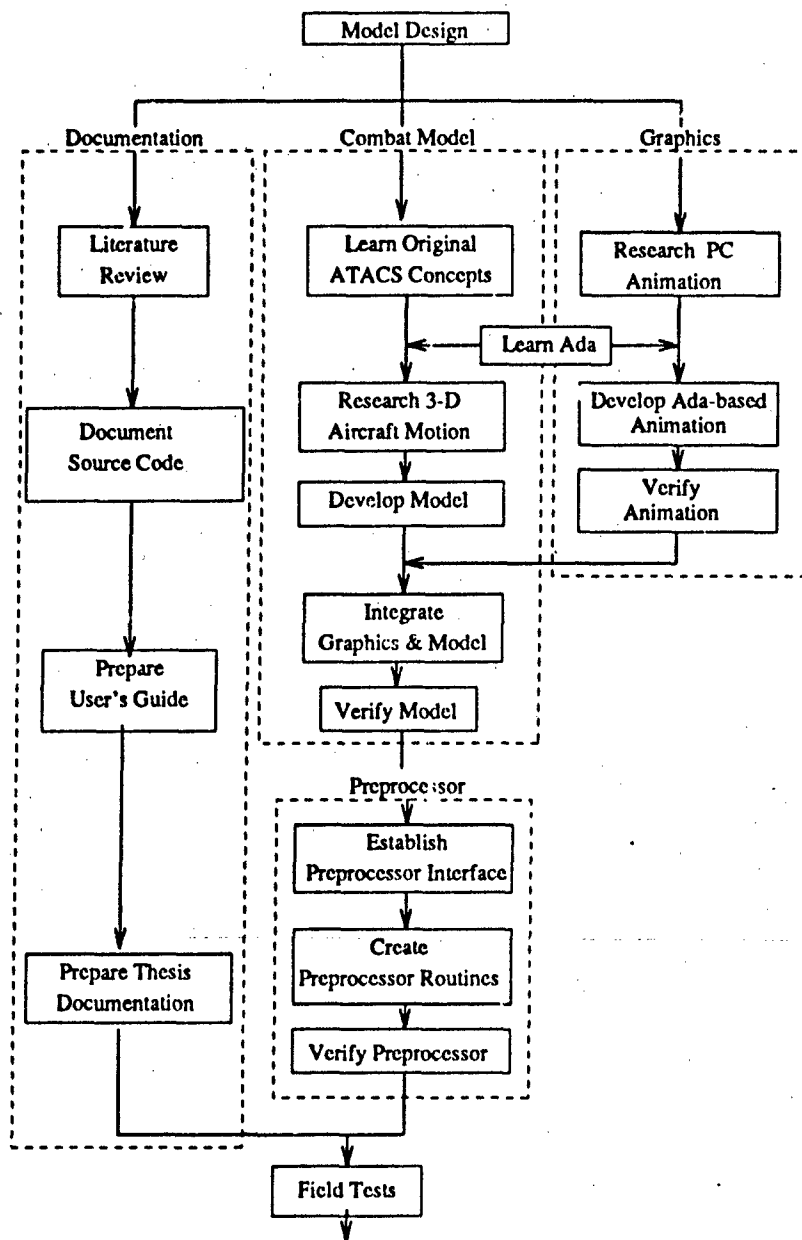


Figure 3.1. Development Flow

- The program should be "user friendly."
 - This program should be simple enough to use that a student with no prior computer experience can use it.
 - This program must also not hinder the learning of an experienced computer user by not being flexible or by requiring excessive user input.
- The combat model should demonstrate many of the combat modeling techniques and processes inherent in many high-resolution combat models.
 - This program should reflect the high-resolution combat modeling course objectives.
- The program should use animated graphics to illustrate the air-to-air engagement.
- The program should allow the user to pause the simulation and check the status of the combatants.
- The simulation should run at least as fast as real-time.
 - Real-time meaning that a simulated engagement lasting ten minutes would take ten actual minutes to execute.
 - The intent of the program is to be a learning tool for students. If the program were to run too slow, the student will lose interest and the program will not fulfill its purpose.
- The simulation and the animation should be one program.
 - The animation should run concurrent to the simulation. If the two were separate, the delay would make the system less enlightening to the student.
- The model, the preprocessor, and the animation should all be written in the Ada programming language.

- The program should be in the DoD mandated language.
- The Operational Sciences Department has little or no experience in Ada-based combat modeling.
- The model should be accompanied by a preprocessor.
 - A preprocessor will allow students easy access to the model inputs.
 - A preprocessor will let the student change inputs and quickly see the effects.

These objectives require that some basic assumptions be made in order to meet the objectives and still remain within the scope of the project. The assumptions about the objectives are:

- This is a model intended for educational purposes, not as an analytical tool.
- This project must be completed during a five-month period.
- Because of the relatively slow speed of the personal computer, the air-to-air engagement must be simplified but must still retain some "look" of a real engagement.

By laying out these design objectives at the outset, the other four stages of development progressed much smoother; especially the graphics routines.

3.3 Graphics Methodology

A major, and certainly the most visible, component of this project was the development of an Ada-based animation. The method for achieving this was broken down into three steps: researching PC animations, developing an Ada-based animation, and verifying the animation.

The first step was to research existing materials on PC-based animations. The research searched for animation techniques and methodologies and, specifically, whether an Ada-based animation exists.

Once the state-of-the-art was investigated, the next task was to use an existing Ada-based animation to create the displays needed for ATACS+ or, if none existed, to create a new animation.

Finally, the working animation would have to be verified to ensure its reliability. This verification would be done by "exercising" the animation - giving it various contrived flight paths to animate. Errors would then be corrected and the process would be repeated until the animation's performance was satisfactory.

3.4 Combat Model Methodology

Since this project is an extension to an existing combat model, a starting basis existed from which to work. The method for adding enhancements to the original ATACS combat model was broken down into five steps: learning the original ATACS concepts, researching the mathematical model of an aircraft maneuvering in 3-D space, combining the two pieces together and developing an improved combat model, then integrating the animation with the model, and finally, verifying the model.

First, the original ATACS model had to be understood. Doing so consisted of studying Captain Richard Moore's thesis (19), reading the ATACS source code, and deciding which parts were reusable and which parts needed to be replaced.

Since the two-dimensional maneuvers of the original ATACS were to be replaced with three-dimensional maneuvering, research was necessary in this area. Some of the objectives would be: to devise a simple system for computing the motion of the aircraft, to determine a way to execute predefined maneuvers, and to determine a strategy for the engagement logic. Once these pieces were assembled, the next step could be taken.

Developing the model involved taking the reusable parts of the original ATACS, translating them into Ada, and combining them with the new three-dimensional equations of motion and engagement logic. This new combat model would then meet the objectives outlined at the start of this chapter.

Once the combat model was built, it had to be integrated with the animation. The two programs had to be combined into one program in order to satisfy the objective of having concurrent simulation and animation. Once the unified program was created and debugged, it had to be verified.

Initial verification of the model was done by running multiple scenarios and then correcting obvious errors in the animation, unrealistic aircraft motion and decision logic. Class members with operational experience were vital at this point. Once the completed combat model was verified the next major stage of the project began.

3.5 Preprocessor Methodology

The method used to create the preprocessor consisted of three stages. First, the preprocessor requirements were established. Second, the preprocessor was written and debugged. Finally, the preprocessor was verified.

The first, and probably the most important, step was determining what pieces of data would be input to the simulation and how much of this data the user would be allowed to manipulate. Keeping the objectives in mind was critical at this point since too much data would intimidate the novice user, and too little control would deter the more experienced user from reaching the program's goals. Once a list of input data had been formulated, the actual programming began.

The actual preprocessor code had to be written in Ada and had to provide a simple way for the user to change the inputs. As a method of input, it was decided that the ubiquitous "Window" style would make the preprocessor easy to use. Menus and input forms would provide the needed user interface. Once these interfaces were created and the preprocessor was assembled, it had to be verified.

Verification was a simple process for the preprocessor. It consisted of answering three questions: "Does the method of input make logical sense?" "Does the preprocessor generate outputs acceptable to the combat model?" and, "Does the

preprocessor have enough safeguards so that a novice user can use it successfully?" Multiple trial runs were used to answer these questions.

3.6 Documentation Methodology

Since this is a thesis effort, the documentation methodology was rather straight forward. While the research was being conducted, the literature review was being prepared. While the code for the graphics, the combat model, and the preprocessor were being written, necessary comments were included. The code was written to conform to the Ada Style Guide (28). Once the complete ATACS+ system was assembled, a user's guide was written for use in field trials. Finally, the thesis documentation was prepared as a record of the entire process. Although Figure 3.1 shows the documentation to be a separate process it was actually integral to, and was performed concurrent with, the other processes.

3.7 Conclusion

The methodology for meeting the objectives set out in Section 1 of this chapter was, for ease of understanding and time management, broken down into four general tasks. Once each of these tasks had been methodically performed, a series of field trials were conducted. These trials consisted of providing copies of the program and the user's guide to combat modeling students and instructors. Their comments led to numerous changes to the programs and some suggested enhancements which are included in Chapter VI.

IV. Program Design

4.1 Introduction

This chapter reviews the design decisions and the assumptions that were made in setting up the ATACS+ model. These decisions and assumptions were made in order to meet the objectives set out in Chapter III. The most critical of these was the need to complete this entire project in a very short period of time. This time restriction drove the need for simplifications in all aspects of the program. This chapter details these simplifications and how the model was set up because of them.

4.2 Overall Design

Since this is a follow on, the basic idea of this thesis was to add to the work already done by Captain Richard Moore (19). Captain Moore's version of ATACS was written entirely in QuickBASIC, and would not meet the requirement that this thesis be done in the Ada programming language. However, the concept of an educational tool and the basic structure and scenario were reusable. Captain Moore's basic idea was implemented in Ada, using the existing algorithms whenever possible.

Not all of the original ATACS was recreated in ATACS+. Only some parts dealing directly with the combat model were kept. The demonstrations of Circular Error Probable (CEP) and random number generation were discarded because they did not enhance the objectives of this thesis and did not serve to demonstrate the combat processes examined in AFIT's combat modeling classes.

Some of the best features of the original ATACS were retained, such as having a preprocessor which lets the user access the entire data base. The original graphics provided a two-dimensional overhead view of the animated air-to-air engagement. This was also retained and enhancements were added.

In order to meet the fourth objective set out in Chapter I, the two-dimensional, view of the original ATACS was re-written in Ada and a three dimensional animation was added. These animation modes are interchangeable. Once the animation was designed, the next design decision concerned connecting the animation and the combat model.

There were two ways of approaching the model / animation interface; separate the combat model and the animation into two executable programs, or combine the two into a single larger executable program. Separating the programs has the advantages of running the animation faster and "rewinding" to watch a specific portion over again. But the user would have to wait a period of time while the simulation executes. Also, the simulation would have to create a data file almost one megabyte in size for each ten minutes of simulated time. This file would have to contain all of the information required by the animation. A file that large could not be stored in system memory (RAM) unless extended memory was used. Many computers do not have extended memory and those that do, use a variety of memory managers. Since one of the design objectives was to make ATACS+ available to as many students as possible, using extended memory was not a viable alternative. A different alternative would be to have the animation read the information from a file a little at a time. But this would slow the animation because of the slow disk access times. Instead, the model and the animation were combined into a single program. This approach resolved the storage problem but took away some of the flexibility of the animation. It also created potential program size problems.

In the original ATACS, the preprocessor, the combat model, and the animation were one program. In order to remain below an executable program size of 640k (640,000 bytes), the largest program size allowed without using extended memory, the preprocessor was broken out into a separate program. The combat model and the animation remained as a unified program whose size is below the 640k limit.

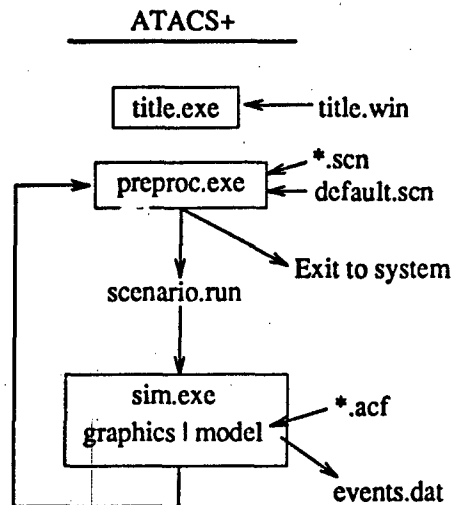


Figure 4.1. ATACS+ System Structure

What finally emerged from the design process, was a system of three programs. Figure 4.1 shows the overall structure of the ATACS+ system. The three executable programs are shown in the boxes. Their order, from top to bottom, is the same in which they are executed. On the right are the names of the files used by the programs. Their contents and how each program uses them is covered in Chapter V and in Appendix A. Figure 4.1 shows that upon exiting the simulation program, `sim.exe`, control is returned to the preprocessor. Only the preprocessor can return control to the operating system.

4.3 Modeling Decisions

The basis for the combat model was the original ATACS program. The overall scenario is that of an airbase, defended by "Blue" aircraft, being attacked by "Red" aircraft. Three entities have been defined: the airbase, the aircraft and the missiles. The number of aircraft and missiles are defined in the scenario file. These are each defined by a record structure which contains the attributes of the entities. These entities are discussed in more detail in Chapter V. The task of the "Blue" forces is

to destroy the attackers as the "Red" aircraft attempt to reach the airbase. Both sides have fully reactive aircraft, unlike the original ATACS where the "Red" forces were non-reactive. In order to enhance the scenario, the "Red" forces can be divided into Strike and Counter-Air missions. Strike aircraft try to avoid engagements while proceeding toward the airbase while the aircraft with the Counter-Air mission engage the "Blue" forces whenever possible. Aircraft, and the missiles they carry, are the only moving entities. No surface-to-air missiles (SAMs), early warning aircraft (AWACS) or electronic warfare were simulated.

4.3.1 Combat Processes. Since the purpose of ATACS+ is to demonstrate combat processes, five of the most important processes were chosen to be modeled. They are: search, target assignment, maneuver, engagement and attrition.

4.3.1.1 Search. In ATACS+, the search process actually combines searching, target acquisition and target identification into one process. The chosen method of searching is the Definite Range Device or "cookie cutter" method (13:5-6). If an opposing entity is within the search area of another aircraft it will be detected, identified and potentially engaged. This same methodology is used by the GCI radar. Any "Red" aircraft detected by the GCI is added to the GCI's target list. This search method was chosen for its simplicity and its transparency to the student user. A stochastic search method would have been confusing to a student watching the animation and trying to understand the search process. A potential target which was obviously within the search area might not be detected. Once the GCI has identified potential threats, it assigns defenders to them.

4.3.1.2 Target Assignment. The target assignment process was designed differently for the "Blue" and the "Red" sides. The "Blue" aircraft have their targets assigned centrally by the GCI. This methodology is a hold-over from the original ATACS. The GCI evaluates the threat of each detected "Red" aircraft

and assigns the closest "Blue" aircraft to it. This pairing continues until the target list is exhausted or all "Blue" aircraft have been assigned. As a simplified means of determining threat, the time to reach the airbase is used. This measure was used for its simplicity and because a more complex algorithm would have been unbalanced with the rest of the combat processes.

4.3.1.3 Maneuver. Since one of the objectives of this thesis was to provide the aircraft entities with the ability to maneuver in all three dimensions, the equations of motion for three dimensional flight had to be developed. These equations give the aircraft five degrees of freedom: X, Y and Z translation, roll and pitch rotations. The sixth degree of freedom, sideslip or rotation around the Z axis, was omitted because sideslip is rarely more than a few degrees and makes little difference in the performance of the aircraft. The missiles are modeled as four degree of freedom bodies. Their roll was eliminated because of the symmetry of missiles. Once the equations of motion were determined, a set of predefined maneuvers was created. The aircraft decision logic chooses one of these maneuvers. Each maneuver is composed of a desired pitch rate, heading change and speed change. This system was derived from the Air-to-Air System Performance Evaluation Model (AASPEM) (23).

4.3.1.4 Engagement. In the real world, rules of engagement and weapon employment strategies are extremely complex. For ATACS+, a simplified way of modeling the engagement process was needed. But the process still needed to demonstrate how engagement is performed. The result is a process where aircraft entities will fire a missile at a target if that target is within the missile's range, if the target is within the aircraft's search area, and if the target is the aircraft's assigned target. All engagements are performed beyond visual range (BVR). It was decided not to model close-in-combat (CIC) because of the complexity of the maneuvers and

the fidelity of the animation would not permit the dynamics of CIC to be clearly seen.

4.3.2 Time Advance Mechanism. When creating a combat model, a time advance mechanism must be chosen. Hartman points out that there are four different methods for advancing the simulation clock (13:2-6). They are:

1. Fixed Time Step
2. Event Scheduling
3. Process Oriented
4. Synchronized to Real Time

The event scheduling and the process oriented approaches require more complex algorithms than the fixed time step method. The synchronized method would be too slow to meet the objectives of this thesis. The method selected for ATACS+ is the same as was used in the original ATACS; the fixed time step. This method is the simplest to implement and it lends itself very well to the air model. At each time-step, the state of every entity is updated. Normally this is the fixed time step's major draw back, but in simulating aircraft flight, the aircraft's state is constantly changing. This makes the fixed time step method the best choice.

4.3.3 Coordinate Systems. The combat model has two coordinate systems. The first is anchored to the earth at an arbitrary origin. The earth is modeled as a flat surface extending to infinity in all directions. The flat earth representation was chosen because of the relatively short distances over which the air-to-air engagements take place. Also, the GCI was modeled so simply that over-the-horizon capabilities were not realistically needed. The earth coordinate system is oriented as shown in Figure 4.2. The earth's positive Y axis points North and the positive X axis points to the East. The positive Z axis, representing altitude, projects upward from the earth's surface making this a right-hand coordinate system. Heading angles are

measured from the positive Y axis in a clockwise direction. Pitch angle is measured from the X-Y plane to the velocity vector of the aircraft. This angle is positive when the aircraft is climbing.

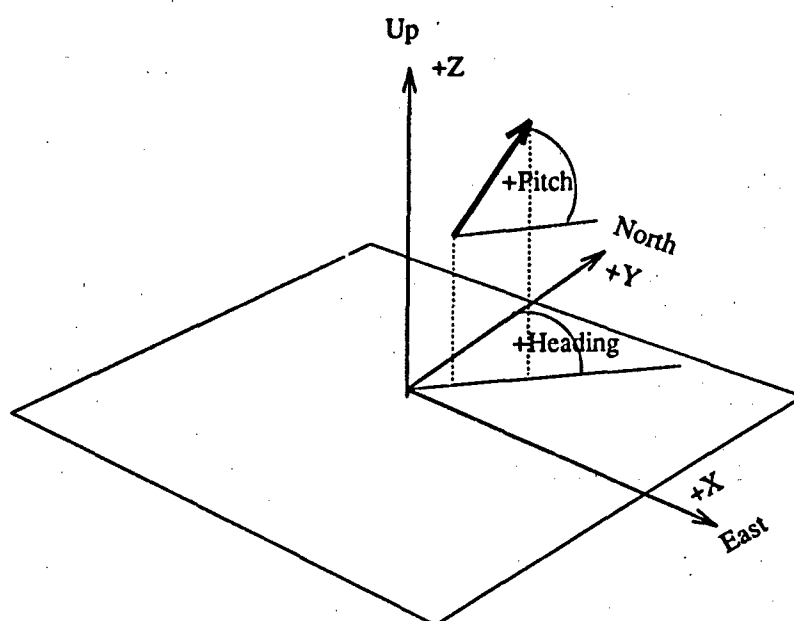


Figure 4.2. Earth Coordinate System

The second coordinate system defined in this model is attached to the aircraft or the missile entity. This coordinate system is shown in Figure 4.3. The aircraft or missile's angle of attack (AOA) is the angle between the entity's velocity vector and its center line. This angle is positive when the entity's nose is above the velocity vector. This angle is used only to compute the aerodynamic performance of the entity. The other angle shown in Figure 4.3 is the aircraft's bank angle. This angle is positive when the aircraft is banked to the right.

4.4 Randomness

One of the major design decisions involved whether to use stochastic processes in the model. Since probabilities are commonly used in many combat processes it

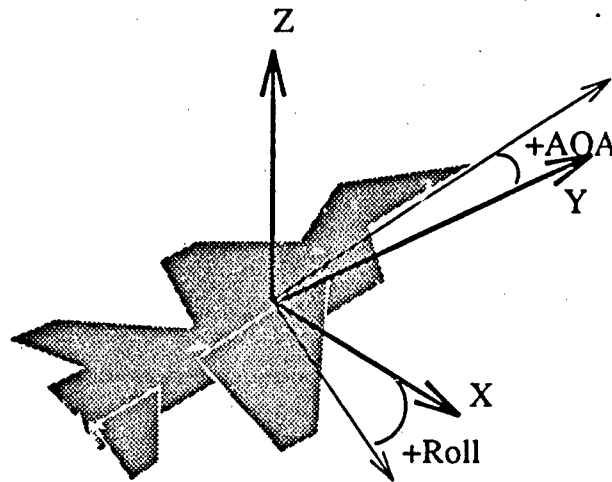


Figure 4.3. Aircraft Coordinate System

was important to include some element of randomness in ATACS+. As was discussed in the previous section on the search process, randomness can make watching the simulation confusing. A student user might not be able to understand why a process did not perform as expected. Since the student's primary source of information is the animation, too many random processes would detract from the learning objectives. On the other hand, it was also important to have at least one random process so the student user could see how changing the probability's critical value affects the engagement outcome. It was decided that randomness would be injected into the simulation in the missile's probability of kill (P_k). For simplicity, the P_k was assumed to be distributed Uniform(0,1).

4.5 Model Assumptions

Once the basic design had been decided upon, several simplifying assumptions were made. Most of these were trade-offs between program complexity and program objectives. Assumptions were made about the weapons, the engagement tactics, and the method of attrition.

4.5.1 Weapon Assumptions. The major assumption concerning weapons was that aircraft will carry a single kind of missile. In the real world, aircraft may carry two or more kinds of missiles, each possibly having variants in seeker type, range etc. Since ATACS+ is a simple model, there was no need to model the missiles to a high degree. ATACS+ missiles are modeled as semi-active radar homing missiles. These missiles will track the target as long as the target remains in the launching aircraft's radar search area. If the launching aircraft is destroyed before the missile reaches its target, the missile is destroyed. All missiles were assumed to fly at a constant speed of 4500 feet per second. Actual air-to-air missiles do not travel at constant speeds but since the time-step is a relatively large one second, and since the missile is being viewed on an animation with a slow update rate, this assumption was reasonable. Another assumption was that all missiles have the ability to withstand 100Gs in a turn. Missiles will also always track their target successfully. The only way a missile can fail to destroy its target is if the random draw is greater than the missile's P_k or if the missile cannot, for performance reasons, get within its lethal radius which is currently set at 100 feet.

4.5.2 Engagement Assumptions. In reality, engagement tactics are very complex and modeling them fully was well beyond the scope of this thesis. Several simplifying assumptions were made to make the simulation possible and still effectively demonstrate the engagement process. First, aircraft on both sides only engage a single target at a time. This was done so that the user can watch the simulation as a series of one-versus-one duels. For the same reason, aircraft do not operate in a mutually supportive way. Wingmen do not cover for flight leaders and aircraft will not be attacked by more than one aircraft. In the case of the "Blue" aircraft, who are defending the airbase, their targets are assigned by the GCI and they will only attack the assigned aircraft. No targets of opportunity are engaged. This was assumed so that the combat process of target assignment could be clearly shown. The last engagement assumption ties in with the previous discussion on weapons;

an aircraft which has launched a missile will not break the engagement if a missile is shot at it. This was a simplifying assumption. Decision logic to add break-off capabilities could be added as a later enhancement.

4.5.3 Attrition Assumptions. Since ATACS+ is a high resolution combat model, attrition is performed on an individual aircraft basis. The only assumption was that if a missile passes within lethal range of its target and a random number drawn from a Uniform(0,1) distribution is less than the missile's P_k , the aircraft will be destroyed. No partial kills are allowed. Aircraft can not be damaged and continue to fight. This assumption was made so that the attrition process could be shown more clearly with the animation. If damaged aircraft were allowed to exist, the student using ATACS+ might be left wondering why a missile which apparently hit its target did not destroy it. "Was it a miss due to the random draw or is the aircraft damaged?". Leaving questions like this unanswered detracts from the intent of the model.

4.6 Model Flow

Once the design decisions and assumptions had been made, the basic logic flow of the model was laid out. Figure 4.4 shows this model structure. The model is structured in three parts: an initialization phase, the simulation loop, and the graphics shutdown. The first four blocks in Figure 4.4 show the initialization of the model. The scenario file created by the preprocessor is read, the aircraft data is loaded into an array, the size of the missile array is set and the graphics are initialized. These four steps are performed before the simulation loop is entered and so are only performed once.

The simulation loop begins with the resetting of the aircraft's target assignment attributes. This is done to prepare for the next two steps, GCI Search, and Target Assignment. Once the GCI has assigned the "Blue" aircraft to the detected attackers,

Simulation Flow

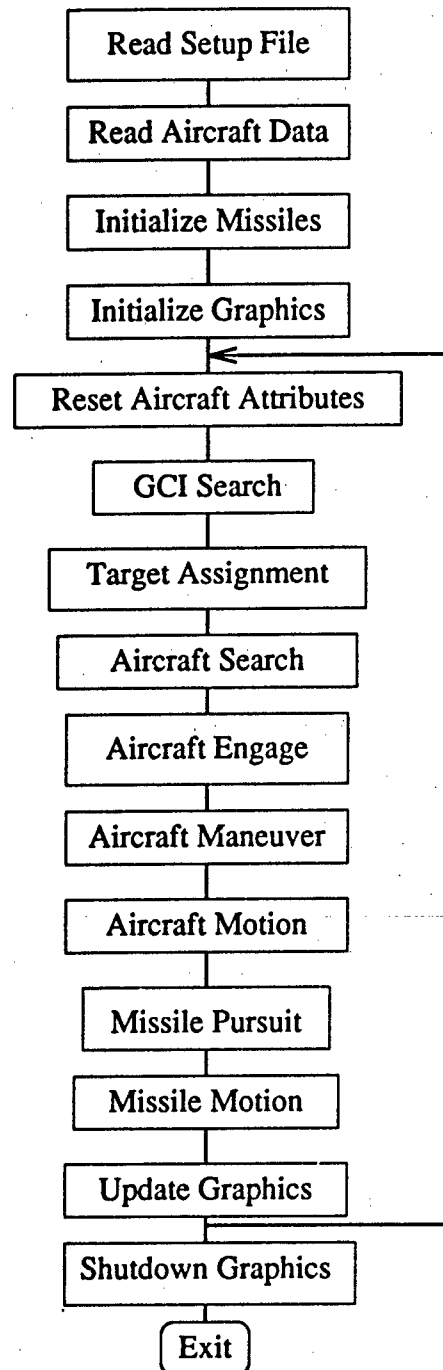


Figure 4.4. Combat Model Logic Flow

four aircraft processes are run. First, all aircraft search for targets, if any are found the aircraft engage. Once the engagement process is complete, the aircraft select a new maneuver and the motion process moves the aircraft to their new positions. After the aircraft processes the missile pursuit motion is calculated. Each of these processes and the algorithms used to perform them are discussed in Chapter V.

After each of the entities has been updated the graphics display is then updated. The logic used to do this is described below. Once the graphics have been updated, the simulation returns to resetting the aircraft attributes. This loop continues until one of the following three stopping criteria is met:

1. Simulation time reaches the specified run length.
2. A "Red" aircraft with a "Strike" mission reaches the airbase.
3. The user selects the *Quit* option from the run-time menu

If any of these three criteria are met, the loop is exited and the graphics are shutdown. This returns the graphics card to the eighty column by twenty-five row text mode used by the preprocessor.

4.7 Graphics Decisions and Assumptions

The biggest decision to be made about the graphical half of the ATACS+ simulation was choosing an Ada-based animation. As discussed in Chapter III, an Ada-based animation was desired because this entire project was a proof-of-concept of Ada-based combat modeling. This decision created considerable difficulties. The only graphics routines written in Ada that were available at AFIT, were the Ada Graphics Library and the Ada Graphics Utility Library, both provided by Meridian Software Systems Inc. Unfortunately, neither was suitable for the high-speed animation required for ATACS+. What was ultimately used was a translation of a C-based animation system developed by Michael Abrash (1). These animation routines create a medium resolution, high-speed animation capable of depicting three dimensional

objects from arbitrary viewpoints. More information on this animation system is contained in Appendix C. These routines were integrated into the ATACS+ model and now provide both two dimensional (2D) and three dimensional (3D) views of the air-to-air engagements.

The 2D display is a re-creation of the display used in the original ATACS retaining the original's zoom and pan features. A decision was made not to use perspective in the 2D mode. Therefore, aircraft moving straight up or down in altitude looks no different than an aircraft traveling at a constant altitude. This decision was made because the 2D display is meant to show the relative positions of the aircraft not their actual maneuvers. To see the dynamic engagement, the user should use the 3D display mode.

Three assumptions were made concerning the 3D display. First, all of the aircraft look the same except for their color. A generic aircraft shape was used. Secondly, the aircraft are not scaled to actual size. If they were, the aircraft would be a single pixel in size. For the same reason the third assumption was made; altitude is scaled by an order of magnitude. With altitudes ranging up to 40-50,000 feet and X and Y distances being as large as a few hundred miles, it made sense to scale the vertical dimension to enhance the aircraft's motion in the vertical plane.

4.8 Graphics Flow

In Figure 4.4 the last step in the simulation loop was shown as updating the graphics. This step is actually a series of steps. These are shown in Figure 4.5.

The first step, whether the display is in the 2D or 3D mode, is to establish the viewpoint. This viewpoint is used to create the transformations required to project the aircraft and missile's images onto the screen. Next the users' inputs are checked. The user may change the viewpoint, the display mode or the level of detail in the display. Once the user's inputs have been processed, either the 2D or the 3D path is followed.

Graphics Flow

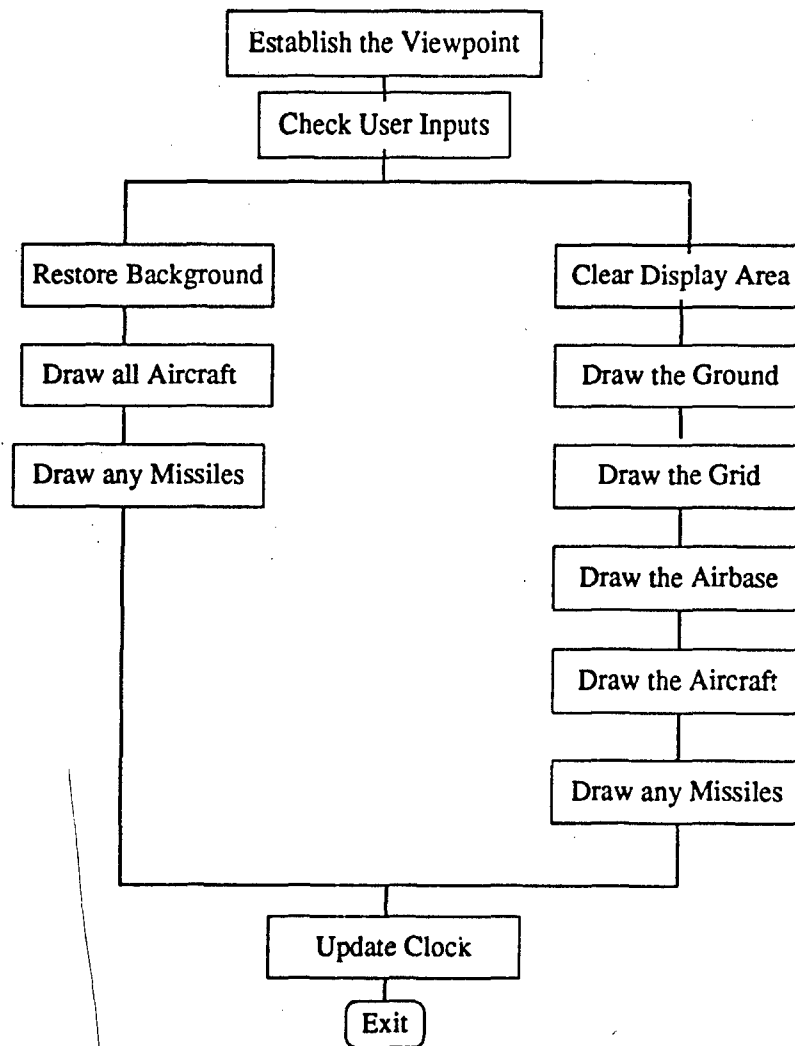


Figure 4.5. Graphics Logic Flow

The three steps performed when in the 2D mode restore the fixed part of the background (the green background, the airbase, and the radar coverage circle), draws each aircraft and its search sector and any missiles which may be in flight. If the 3D mode is begin used, six steps are taken. The first fills the display area with the blue sky color. Then the brown ground is draw and the yellow grid lines are added to the ground. Next, the airbase is drawn, all aircraft are drawn and finally any in-flight missiles are drawn.

The graphics update always concludes with an update of the clock display. The details of some of these steps are covered in the next chapter.

V. Program Details

5.1 Introduction

This chapter provides details to accompany the concepts discussed in Chapter IV. It details the combat processes and also discusses how ATACS+ handles aircraft aerodynamics and motion. A similar section is provided for the missiles and a separate section covers ATACS+'s method of generating random numbers. To begin, the three kinds of entities are described.

5.2 Entities

In the ATACS+ model three kinds of entities exist: aircraft, missiles and the airbase. There is only one airbase entity but the user can define up to fourteen aircraft and a virtually unlimited number of missiles. The number of missiles is only limited by the amount of available computer memory. These entities are composed of a set of attributes which are contained in a composite data type defined in the *Atacs.Types* specification. Each entity's attributes are listed in Tables 5.1, 5.2, 5.3, and 5.4. It is these attributes which change and are updated as the simulation runs. No other objects help define entities. The state of the attributes is what differentiates one entity from another, especially the aircraft. Each aircraft and missile has a composite data type, called *Objectstate*, within its attributes. This *Objectstate* consists of the six pieces of information needed by the graphics routines to place an entity in 3D space. The X, Y, and Z locations and the roll, pitch, and heading angles were grouped together to make passing parameters between graphics routines easier.

5.3 ATACS+ Aircraft

Six kinds of aircraft are currently defined in ATACS+. These particular aircraft were chosen because they fall into three categories: interceptors, strike aircraft, and

Table 5.1. Aircraft Entity Attributes

Attribute	Description
Objectstate.X	East-West location in feet (+East)
Objectstate.Y	North-South location in feet (+North)
Objectstate.Alt	Altitude, height above the ground (feet)
Objectstate.Roll	Bank angle in radians (+right)
Objectstate.Pit	Pitch angle in radians (+nose up)
Objectstate.Hdg	Heading in radians (+clockwise from North)
Speed	Aircraft speed in feet per second
Visual.Range	Maximum range, in miles, at which visual identification of another aircraft can be made
Radar.Range	Maximum range, in miles, of aircraft's radar.
Radar.Sweep.Angle	In radians, measured $\pm$ from aircraft centerline.
Acquire.Status	Either None, Radar or Visual
Alert.Status	Either Detected or Undetected
Assign.Status	Either Assigned, Unassigned or KIA.
Mission	Either Ab.Def, Strike or Counter-Air.
Roe	Rules of Engagement, Fire.First or Fire.Second.
Objective	3D point that the aircraft attempts to reach.
Objective.Index	Index of aircraft to intercept
Defensive.Level	The threat level of the aircraft
Target.Index	Index of the aircraft to engage
Targeted.By.Index	Index of aircraft engaging this aircraft
Ang.To.Target	Angle off the nose to the target
Dist.To.Target	Distance to the target
Depress.Angle	The angle below the nose to the target
Num.Of.Msls	The current number of missiles on board.
Msl.In.Flt	True if aircraft currently has a missile in flight.
Missile.Limits.Speed	Aircraft's missiles speed, in feet per second.
Missile.Limits.Max.Flt.Time	The number of seconds of flight time.
Missile.Limits.Maxgs	Maximum turning acceleration, expressed in Gs.
Missile.Limits.Pk	The probability of kill given a hit.
Maneuver.Name	Either CAP, Pursuit, Intercept, Missile.Def, Attack or Retreat
Aircraft.Kind	Either F15, F16, F111, Mig25, Mig29, Su24.
Time.Of.Manu	Time the last maneuver was begun
Maneuver.Hdgrate	The desired rate of change of the aircraft heading, in radians per second.
Maneuver.Pitrade	The desired rate of change of the aircraft pitch in radians per second

Table 5.2. Aircraft Entity Attributes (cont'd)

Attribute	Description
Maneuver.Vdot	The desired rate of change of the aircraft speed in feet per second per second.
Throttle	Enumerated Corner, Cruise, Military, or Afterburner
Corner_Vel	The airspeed, in feet per second, at which best instantaneous turn rate is created.
Maxgs	The maximum allowable load factor
Wingarea	The area, in square feet of the aircraft's wing
Wgt	The aircraft's weight in pounds
Max_Roll_Rate	Max allowable roll rate in radians per second.
Afterburner_Thrust	The pounds of thrust produced by a single engine in maximum afterburner.
Mil_Pwr_Thrust	The pounds of thrust produced by a single engine in full military power.
Num.Of_Eng	The number of engines on the aircraft
Drag_Multiplier	Limits aircraft performance. Between 0.5 and 1.5.
Color	The designator for the side on which the aircraft fights, either 1 (Blue) or 12 (Red).
Call_Sign	A unique name for this aircraft.

Table 5.3. Missile Entity Attributes

Attribute	Description
Objectstate.X	East-West location in feet (+East)
Objectstate.Y	North-South location in feet (+North)
Objectstate.Alt	Altitude, height above the ground (feet)
Objectstate.Roll	The missile's bank angle in radians (+right)
Objectstate.Pit	The missile's pitch angle in radians (+nose up)
Objectstate.Hdg	Heading in radians (+clockwise from North)
Maneuver.Hdgrate	The desired rate of change of the missile heading, in radians per second.
Maneuver.Pitrate	The desired rate of change of the missile pitch in radians per second
Maneuver.Vdot	The desired rate of change of the missile speed, current set to zero
Status	Either Ready, Launched, Detonated, or Missed
Target.Index	The index of the aircraft to target
Lnch.Ac.Index	The index of the launching aircraft
Time.Of.Launch	The time launch was initiated
Limits.Speed	The speed, in feet per second, of the missile.
Limits.Max.Flt.Time	The number of seconds of flight time.
Limits.Maxgs	Maximum turning acceleration, expressed in Gs.

Table 5.4. Airbase Entity Attributes

Attribute	Description
X	The East-West location of the airbase in miles
Y	The North-South location of the airbase in miles
Radar.Range	The range of the GCI's radar.
Num.Of.Threats	The number of detected enemy aircraft

multi-role aircraft. Interceptors specialize in air-to-air combat and are sometimes referred to as "dog-fighters." Strike aircraft carry bombs and attack ground targets. Multi-role aircraft are capable of performing either the interceptor or the strike role. The six aircraft in ATACS+ are: the F-15A interceptor built in the United States (US) by McDonnell Douglas, the F-16C multi-role fighter built in the US by General Dynamics, the F-111F strike aircraft built in the US by General Dynamics, the Mig-25 interceptor built in the Soviet Union by the Mikoyan/Gurevich Design Bureau, the Mig-29 multi-role fighter built in the Soviet Union by the Mikoyan/Gurevich Design Bureau, and the Su-24 strike aircraft built in the Soviet Union by the Sukhoi Design Bureau. Of the many aerodynamic parameters which might distinguish these six aircraft from one another, only a limited number were used in ATACS+.

5.4 Aircraft Parameters

Nine parameters define each aircraft. For the six current ATACS+ aircraft, Table 5.4 lists the values of these nine parameters.

The first parameter listed is corner velocity. This is the airspeed, here measured in feet per second, where the aircraft generates maximum lift at the maximum allowable load factor. The velocity-load factor (V-n) diagram in Figure 5.1, shows how the corner velocity is derived (25:389). The curved line marked "lift limit" shows the amount of Gs (or load factor) an aircraft can generate at a given speed. The airspeed at which this line meets the structural limit line is called the corner speed or corner velocity. Corner speed is important because at this speed the aircraft attains maximum instantaneous turn performance (25:398). The boundaries of the V-n diagram vary with aircraft weight, configuration, throttle setting and altitude. For ATACS+, an average altitude of 10,000 feet, military power, and the weight shown in Table 5.4 were used. For most aircraft, the corner speed is approximately 0.8 Mach or 80 percent of the speed of sound. Since no values are readily available for

Table 5.5. Aircraft Performance Parameters

	F-15	F-16	F-111	Mig-25	Mig-29	Su-24
Corner Velocity (ft/sec)	861 ^c	808 ^d	861 ^c	900 [*]	800 [*]	861 [*]
Maximum Gs	9 ^a	9 ^a	5 [*]	5 ^a	9 [*]	5.0 [*]
Wing Area (ft ²)	608 ^a	300 ^a	525 ^b	612 ^a	379 ^a	452 ^a
Weight (lbs)	37903 ^a	23765 ^a	90000 [*]	64365 ^a	30700 ^a	68376 ^a
Max Roll Rate (rad/sec)	2 [*]	3 [*]	1 [*]	2 [*]	3 [*]	1 [*]
Afterburner Thrust (lbs)	23770 ^a	23830 ^b	25100 ^b	24700 ^a	18300 ^a	24700 ^a
Military Thrust (lbs)	14670 ^b	14670 ^b	15000 [*]	16775 ^b	11240 ^a	17635
Number of Engines	2 ^a	1 ^a	2 ^a	2 ^a	2 ^a	2 ^a
Drag Multiplier	1.0 [*]	1.0 [*]	1.2 [*]	1.0 [*]	1.0 [*]	1.4 [*]

Sources:

a = Jane's All the Worlds Aircraft 1991-1992 (17)

b = Modern Air Combat (11)

c = T.O. 1F-15A-1 (5:B9-35)

d = T.O. 1F-16C-1-1 (6:A1-19)

e = T.O. 1F-111F-1-1 (7:A11-9)

* = estimated

the Soviet aircraft, the same values as their U.S. counterparts are used as estimates. In ATACS+, corner speed is needed for the missile defense maneuver.

The next parameter in Table 5.4 is maximum Gs or maximum load factor. Gs are a measure of acceleration and are expressed in multiples of the acceleration due to gravity. For example, straight and level, unaccelerated flight would be at 1G. The aircraft's maximum load limit is established either because of structural limits on the aircraft or physiological limits on the pilot. In the case of the F-111 and the Su-24, the maximum allowable load factor is set to 5.0 because the aircraft would be carrying external weapons for the strike mission in ATACS+. The extra load reduces the maximum allowable load factor. The maximum load factor is later used to limit the turn performance of the aircraft.

The wing area of each aircraft is the planform area measured in square feet. It includes the area in the fuselage between the wings (4:81). The aircraft weight was drawn from various references (see Table 5.4) and were corrected to a weight

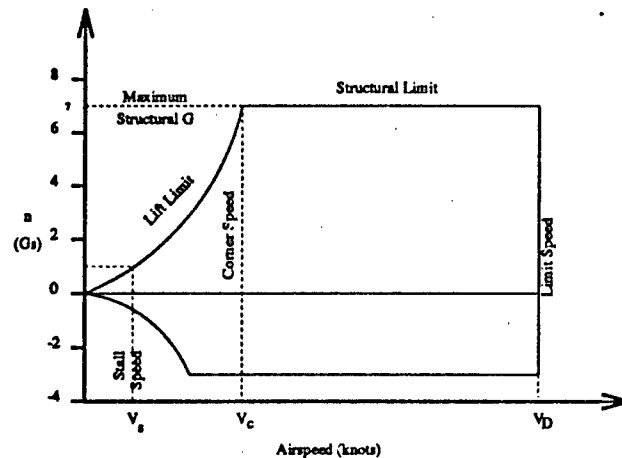


Figure 5.1. Velocity-Load Factor Diagram

reflecting 50 percent internal fuel. These values are approximations and do not change as the air-to-air engagement progresses. The weight is not reduced as fuel is burned or as ordnance is expended.

The maximum roll rate is measured in radians per second. All of these values were estimated since no actual values were available. The values used were drawn from the experience of this thesis' author. His experience with the T-38 aircraft, reputed to be the fastest rolling aircraft in the Air Force inventory, has shown that one revolution per second is a reasonable upper limit, although half this value is a more realistic value in actual usage. The F-16 and the Mig-29, being the smallest of the six aircraft were given a value of 3.0 radians per second. The F-15 and the Mig-25, being larger aircraft, were assigned the value of 2.0, and the strike aircraft were given a slow 1.0 radians per second roll rate. ATACS+ uses these values to limit the aircraft's change in roll angle in each time interval.

The afterburner and military thrust values were taken directly from reference material (see Table 5.4 for references). These values represent the pounds of thrust produced when a single engine is in full afterburner or set to full military power (usually 100 percent rpm).

The number of engines on each aircraft is self-explanatory. These numbers were taken from the references as noted in Table 5.4.

The drag multiplier is not an actual aerodynamic parameter, but rather it was created specifically for ATACS+. Because of the way drag is computed in this program, an adjustment factor was needed for aircraft carrying external stores. External weapons create a sizable amount of drag and limit the aircraft's performance. To account for this effect, the drag multiplier is used.

5.5 Adding Aircraft

In order to add another type of aircraft to ATACS+, the nine parameters listed in Table 5.4 need to be defined for that particular aircraft type. These nine values are then stored in an appropriately named data file with a .ACF extension. For example, the F-15 data is stored in the file F15.ACF. Then the designation of the aircraft should be added to the enumerated values in the *Atacs.Types* and *Preprocessor.Types* specifications. The preprocessor automatically includes the new aircraft name in the aircraft selection menu after the program has been re-compiled. The simulation program also needs to be re-compiled, but once this is accomplished, the new aircraft will be available for use in ATACS+. The data stored in the .ACF files is used to compute the aerodynamic properties of each aircraft. These properties are derived from a generic aircraft. This derivation process is described in the next section.

5.6 Aerodynamics

In order for the aircraft to fly in a realistic manner, certain aerodynamic relationships have to be known about each aircraft. These relationships could be very complex, involving an extensive number of parameters. However, there are some basic aerodynamic relationship which all aircraft share without considering such things as maneuvering flaps, vectored thrust or high drag devices. What was required for

ATACS+ was a set of simple relationships which captures the trends seen in actual aircraft. These relationships are:

$$\text{Angle of Attack(AOA)} = f(\text{Mach Number, Lift Coefficient})$$

$$\text{Drag Coefficient}(C_d) = f(\text{Mach Number, Altitude, Lift Coefficient})$$

$$\text{Maximum Lift Coefficient}(C_{l_{max}}) = f(\text{Mach Number})$$

$$\text{Military Thrust} = f(\text{Static Military Thrust, Mach Number, Altitude, Number of Engines})$$

$$\text{Afterburner Thrust} = f(\text{Static Afterburner Thrust, Mach Number, Altitude, Number of Engines})$$

Collecting and distilling each of these relationships for each aircraft was well beyond the scope of this thesis. Also, for the purpose of ATACS+, it was not required. Instead, general relationships were established for each parameter. All ATACS+ aircraft use the same relationships with the exception of C_d , where the drag multiplier is used to adjust the computed value of C_d . The thrust equations take the static sea-level rating of the aircraft's engines and adjust it for changes in Mach Number and Altitude. These relationships may not be exact for any of the aircraft, but the trends are based on a real aircraft, the T-38 Talon. Data was available for the complete aerodynamics of the T-38 from Captain Shawn Smellie of the Aerodynamics and Flight Manuals Branch, San Antonio Air Logistics Center, Kelly AFB, Texas. The information was converted from data tables into polynomials using least squares regression. Each of the resulting equations has a coefficient of determination, R^2 , of at least 0.85. These equations were then added to the routines in the Ac.Pkg package. Using the aerodynamics of an actual aircraft keeps the ATACS+ aircraft from reaching unrealistic speeds, performing unrealistic turns or climbing to unrealistic heights. All ATACS+ aircraft use the same set of relationships. These relationships are summarized below:

$$\text{Afterburner_Thrust} = \text{Static_Afterburner_Thrust} * (1.0 + 0.342 * \text{Mach} - 2.227 * 10^{-5} * \text{Altitude}) \quad (5.1)$$

$$* \text{Number_Of_Engines} \quad (5.2)$$

$$\text{Military_Thrust} = \text{Static_Military_Thrust} * (1.0 + 0.073 * \text{Mach} - 2.026 * 10^{-5} * \text{Altitude}) * \text{Number_Of_Engines} \quad (5.3)$$

$$C_{l_{\max}} = \begin{cases} 0.996 + 0.421 * \text{Mach} - 2.538 * \text{Mach}^2 \\ + 2.315 * \text{Mach}^3 & \text{if Mach} < 1.1 \\ 6.233 - 6.579 * \text{Mach} + 1.92 * \text{Mach}^2 & \text{otherwise} \end{cases} \quad (5.4)$$

$$C_{d_1} = \begin{cases} 0.017 - 0.009 * \text{Mach} + 6.861 * 10^{-8} * \text{Altitude} \\ + 0.009 * \text{Mach}^2 + 1.108 * 10^{-12} * \text{Altitude}^2 \\ - 7.842 * 10^{-8} * \text{Mach} * \text{Altitude} & \text{if Mach} < 1.0 \\ - 0.24 + 0.085 * \text{Mach} + 1.039 * 10^{-7} * \text{Altitude} \\ - 0.026 * \text{Mach}^2 - 3.237 * 10^{-8} * \text{Mach} * \text{Altitude} & \text{otherwise} \end{cases} \quad (5.5)$$

$$C_{d_2} = -0.143 * \text{Mach} + 0.231 * C_l + 0.105 * \text{Mach}^2 +$$

$$0.057 * Mach * C_l \quad (5.6)$$

$$C_d = C_{d_1} + C_{d_2} \quad (5.7)$$

5.7 Aircraft Motion

The process of creating motion in the ATACS+ aircraft consists of three steps. First, decision logic is used to choose a maneuver. Second, a maneuver routine corresponding to the chosen maneuver is executed. This step determines the three parameters which define a maneuver: desired pitch rate, desired turn rate, and desired change in speed. The final step is to take this desired maneuver and move the aircraft accordingly. This movement is limited by the performance characteristics of the aircraft. Each of these steps is discussed in the following sections.

5.7.1 Maneuver Logic. Once during each time step every aircraft executes the `Maneuver_Pkg.Aircraft_Maneuver` procedure. This procedure decides which of the six defined maneuvers the aircraft will execute during the next time interval. This decision logic is diagramed in Figure 5.2. The larger font signifies a conclusion in the decision logic. The top portion of Figure 5.2 shows the determination whether a missile has been launched at the current aircraft. This determination is made only if the aircraft does not have a missile in flight. If the aircraft is in danger of being hit by a missile and does not have a missile in flight, then the decision is made to perform the Missile Defense maneuver. If this is not the case, then the decision logic proceeds according to the aircraft's mission. Aircraft performing the airbase defense mission will retreat if they are out of missiles, or will choose between the Combat Air Patrol (CAP), Intercept, or Pursuit missions depending on its assignment and acquisition status. Strike aircraft can perform only the Attack mission. Aircraft flying the Counter-Air (Cntr_Air) mission are similar to the airbase defense aircraft; they will retreat when out of missiles or choose between Attack or Pursuit when

missiles remain. Although this decision logic is very simplistic, it is effective enough for the intended purpose of ATACS+.

5.7.2 Maneuver Descriptions. In each of the following maneuvers, the desired turn and pitch rate are set equal to the angle through which the aircraft wishes to turn. No consideration of aircraft performance is made at this time and it is assumed that the aircraft would like to move through this angle in a single time period. These rates will later be limited by the aircraft's performance.

5.7.2.1 Missile Defense. The first maneuver choice deals with defending against a missile attack. In reality, evading a missile requires tactics far beyond the scope of this project. Still, it was important that aircraft react to being fired upon. Major Randy Nelson suggested a simple abstraction of real world tactics which suits ATACS+'s purpose (21). His suggested defense consists of the two phases shown in Figures 5.3 and 5.4. The initial maneuver, Figure 5.3, has the aircraft diving towards the ground at a sixty degree angle until the aircraft is less than one thousand feet above the ground. At the same time it turns to put the missile "on the beam." This results in a continuous turn by the aircraft which makes the missile constantly decrease its turn radius as it nears its target. This tactic continues until the missile is seven seconds from impact. At this time, the tactic is switched to the one shown in Figure 5.4. This maneuver turns the aircraft into the missile. The aircraft turns, trying to reduce the angle to the missile while at the same time, trying to raise or lower the aircraft's nose to the same angle at which the missile is diving. This sudden hard turn is an attempt to create as much separation as possible as the missile passes the aircraft. The key to this tactic is the fact that the missile's speed is several times greater than the aircraft's speed so will have a much greater radius of turn. The whole tactic behind the missile defense maneuver is to use the aircraft's superior turning ability to evade the missile.

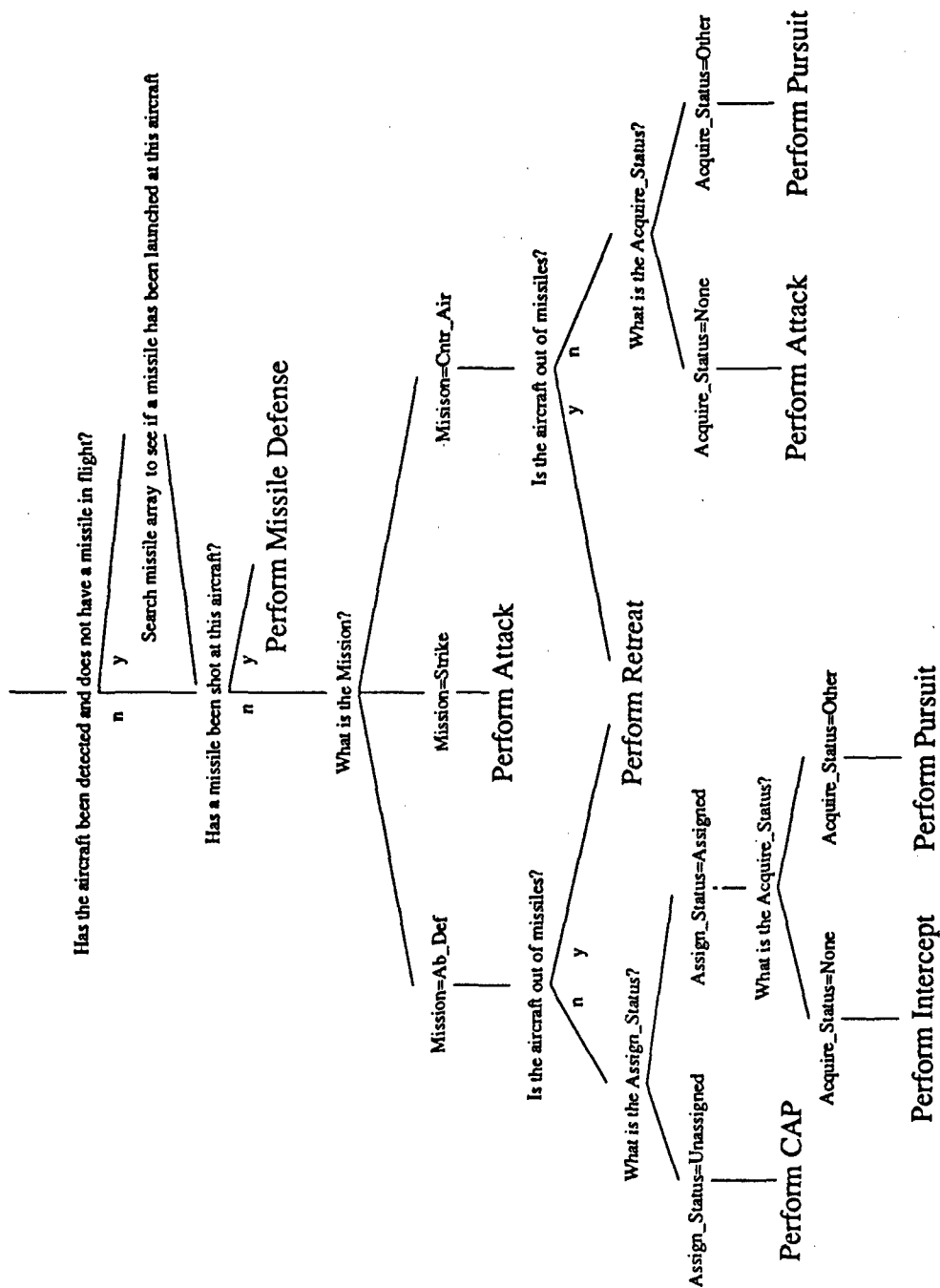


Figure 5.2. Maneuver Selection Logic

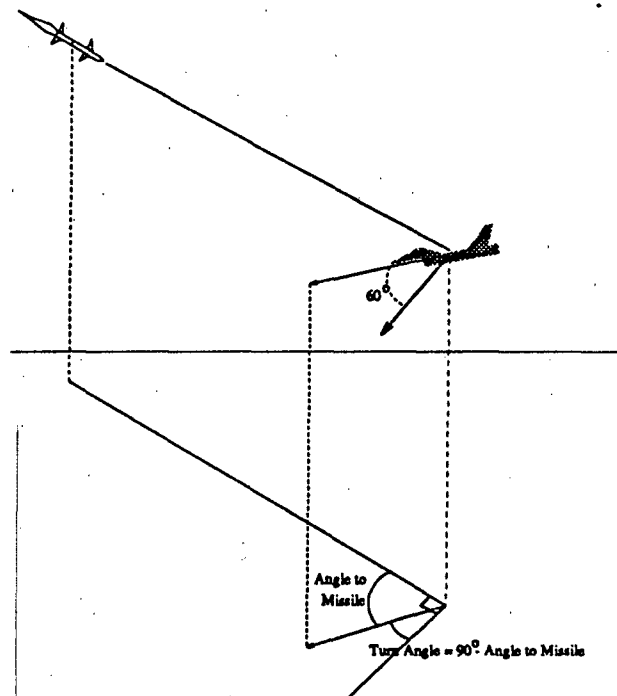


Figure 5.3. Initial Missile Defense Geometry

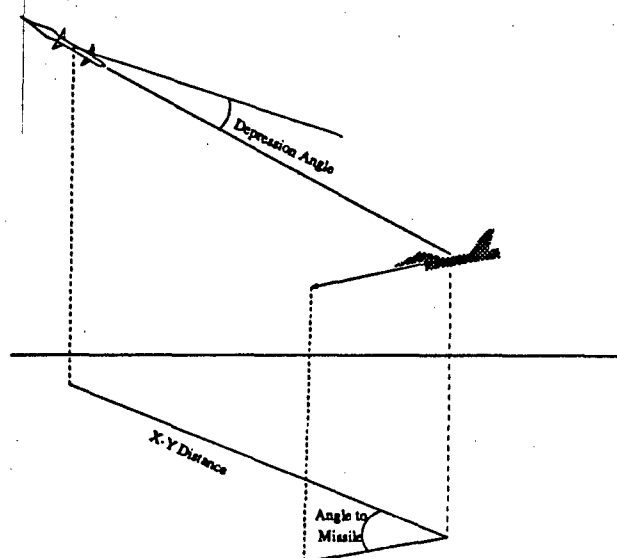


Figure 5.4. Close-in Missile Defense Geometry

5.7.2.2 *Intercept.* The geometry for the intercept maneuver is shown in Figure 5.5. The basic objective of the intercept maneuver is for the intercepting aircraft to arrive at the same point in space as the target aircraft at a given time. The computation of the lead angle is the critical component of the intercept. The lead angle is derived as follows:

Using the Law of Sines:

$$\frac{d_2}{\text{Lead_Angle}} = \frac{d_1}{\text{Angle_Off_Nose}}$$

where:

d_1 = the distance from the interceptor to the intercept point

d_2 = the distance from the target to the intercept point

Line Of Sight = the line connecting the centers of the two aircraft

Angle Off Nose = the angle the Line Of Sight makes with the target's flightpath

Lead Angle = the angle between the Line of Sight and the intercept vector

Turn Angle = the angle the aircraft needs to turn to align its flightpath with the intercept vector

so:

$$\text{Lead_Angle} = \text{Angle_Off_Nose} * \frac{d_2}{d_1} = \text{Angle_Off_Nose} * \frac{v_2 * t_2}{v_1 * t_1}$$

but at intercept $t_1 = t_2$, resulting in the final equation:

$$\text{Lead_Angle} = \text{Angle_Off_Nose} * \frac{v_2}{v_1} \quad (5.8)$$

The intercepting aircraft turns until its flight path is aligned with the intercept vector. At the same time, the interceptor adjusts its pitch angle so that it arrives at

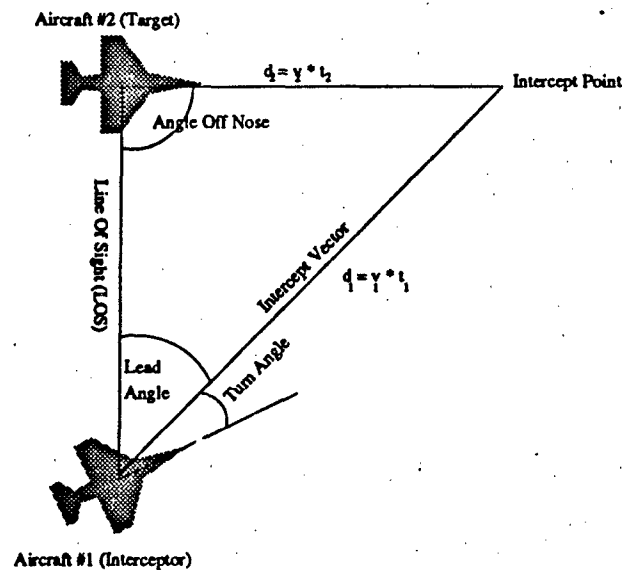


Figure 5.5. Intercept Maneuver Geometry

the target's altitude at the intercept point. This maneuver is a simplification of the intercept tactics discussed in *Fighter Combat* by Robert L. Shaw (25).

5.7.2.3 Pursuit. The pursuit maneuver's geometry is shown in Figure 5.6. The pursuit maneuver is very simple. The pursuing aircraft rotates its heading and pitch angles until its flight path is aligned with the LOS. This is referred to as "pure pursuit" (25:36).

5.7.2.4 Attack. The attack maneuver shown in Figure 5.7 is very similar to the pursuit maneuver. The aircraft attempts to turn its heading so that the angle to the target (the airbase) is reduced to zero. The angle to the target is the difference between the LOS vector's heading and the aircraft's heading. The desired pitch rate is set to the negative of the current aircraft pitch angle. This keeps the aircraft at a constant altitude during the attack maneuver.

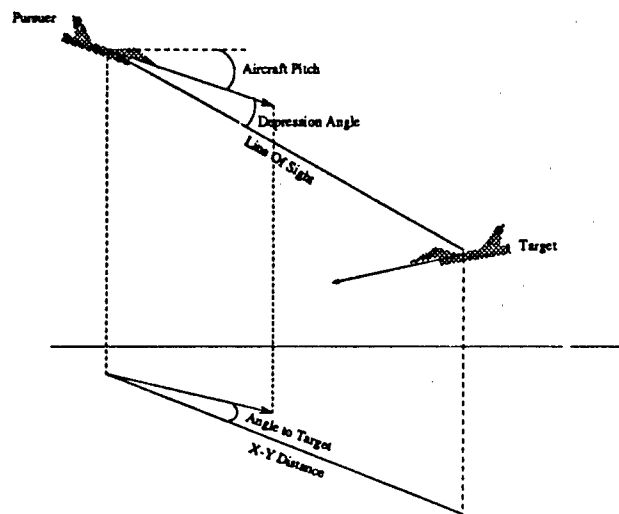


Figure 5.6. Pursuit Maneuver Geometry

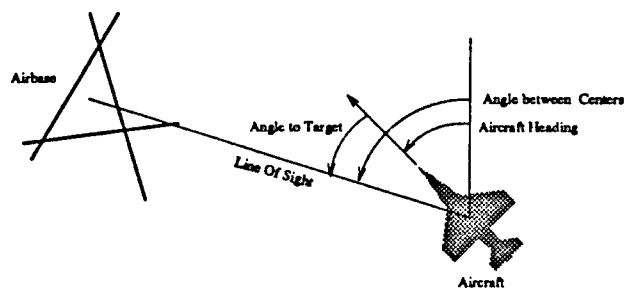


Figure 5.7. Attack Maneuver Geometry

5.7.3 Motion Logic. Once the maneuver has been decided, the aircraft entity performs the *Aircraft_Motion* procedure found in the *Ac_Pkg* package. This procedure uses the aerodynamics of the particular kind of aircraft to limit the maneuver. The actions in this procedure are shown in Figure 5.8. This procedure's methodology is to take the turn and pitch rates defined in the maneuver, and combine them into a desired load factor. This load factor is then limited by aerodynamic and structural G-limits. This restricted load factor is then broken into the vertical and lateral components used to compute new turn and pitch rates. These rates and the computed engine thrust are then used to compute incremental pitch, roll, and heading angles which are added to the current angles. This methodology was taken from the AASPEM combat model (23:1-6).

This entire process, maneuver decision, maneuver execution, and aircraft motion is repeated for each aircraft during each simulation loop. A similar process is used for simulating the missiles.

5.8 Missile Motion

The missiles are aerodynamic vehicles with properties similar to the ATACS+ aircraft. Only one maneuver is used: the pursuit. The missile will always try to point its nose at the target. The logic of the *Missile_Motion* procedure of the *Msl_Pkg* package is shown in Figure 5.9. The first thing this procedure does is check to see if the missile has been in flight longer than the maximum flight time. If this is true, the missile is destroyed and is considered a miss. If the missile still has flight time remaining then, just like the aircraft, the desired turn and pitch rates are converted to a desired load factor. This load factor is limited by the maximum allowable Gs. Then the restricted load factor is converted into vertical and lateral accelerations. These are used to compute actual turn and pitch rates. The next part of the *Missile_Motion* procedure is very different than the *Aircraft_Motion* procedure discussed earlier.

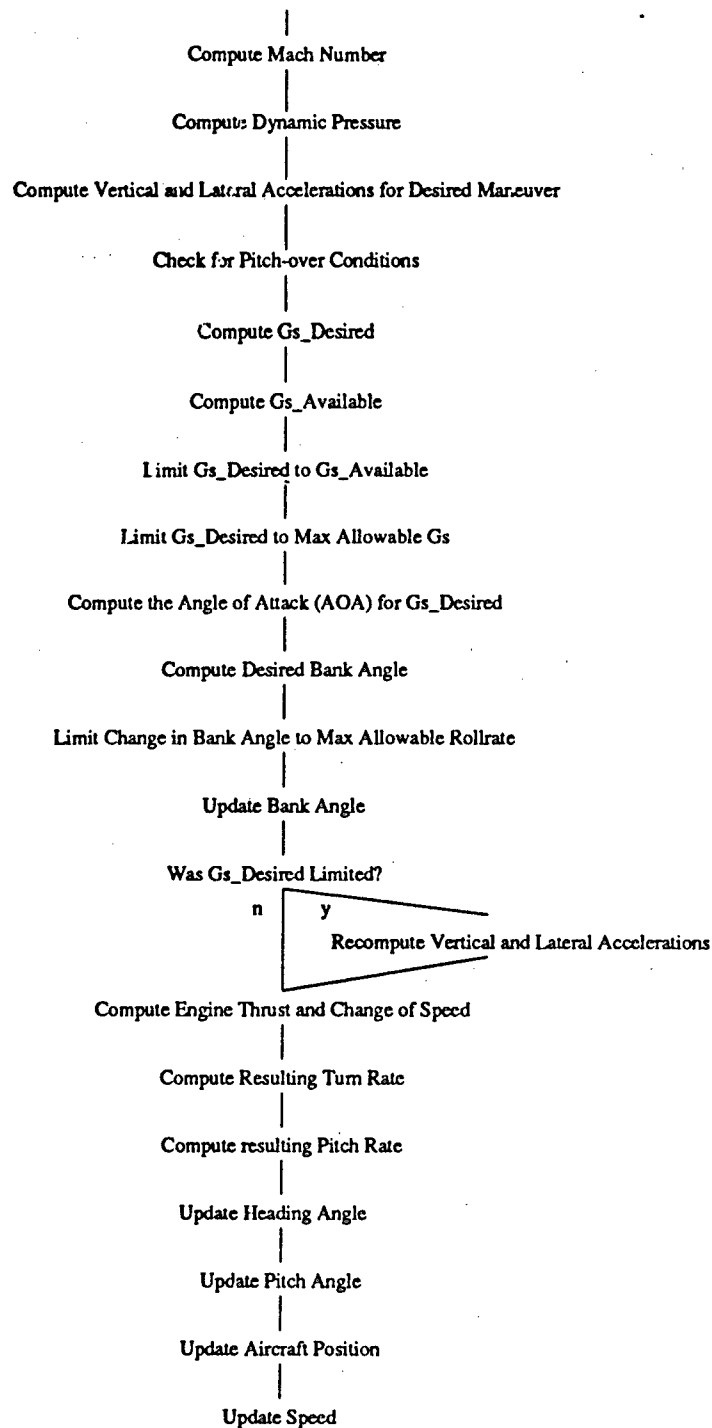


Figure 5.8. Aircraft Motion Logic

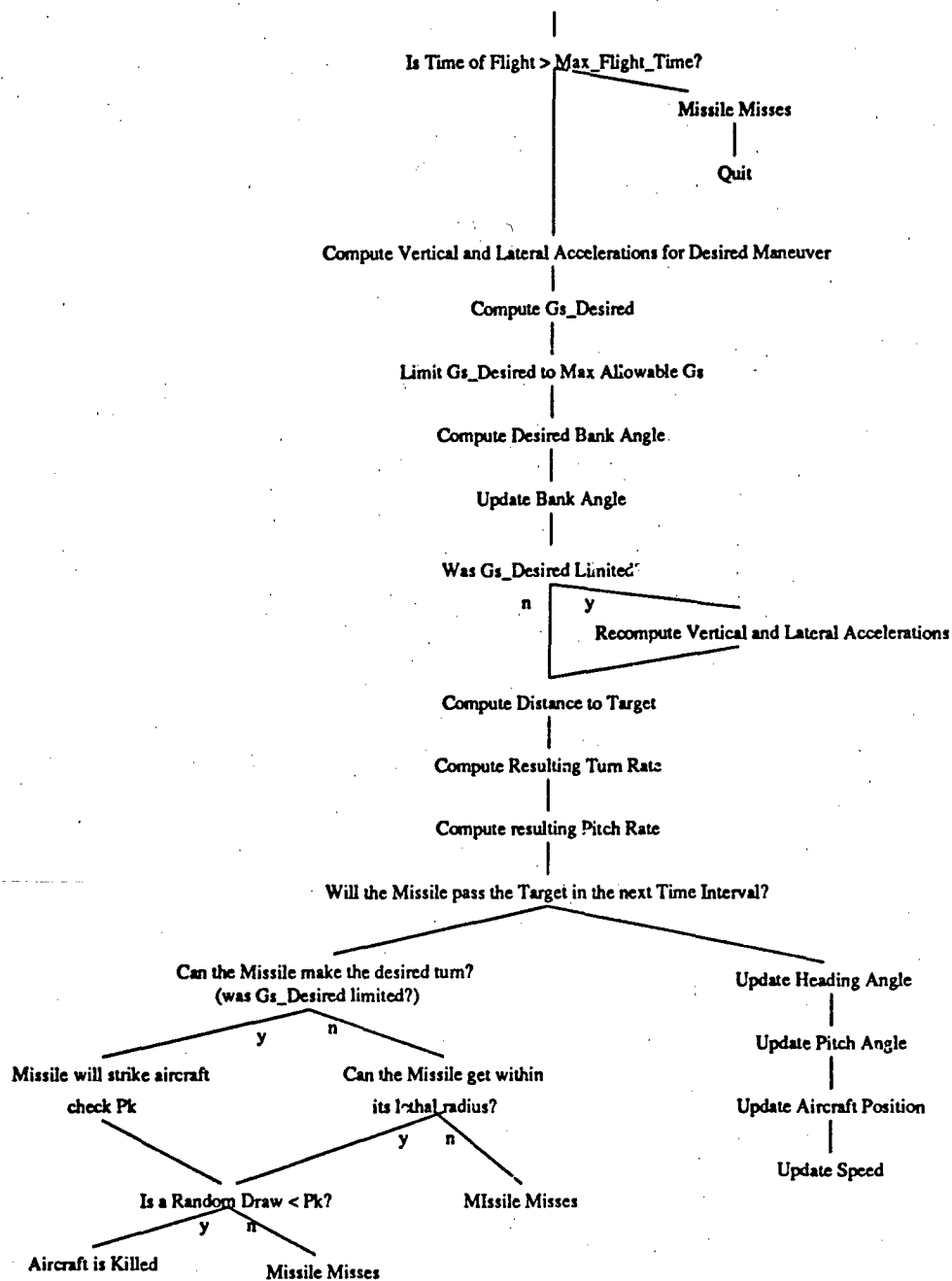


Figure 5.9. Missile Motion Logic

If the missile will pass the target during the upcoming time interval, then the situation depicted in Figure 5.10 will take place. One of two things can happen: either the missile will be able to turn the desired amount in which case the missile will impact the target, or because of the G-limitation, the missile will not be able to turn into the target. In this case, the distance of closest approach is computed. If this distance is less than the lethal radius of the missile, the missile is treated as if it hit the target. If not, the missile is destroyed and is counted as a miss. If the missile either strikes the target or passes within its lethal radius, a random number is drawn from a Uniform(0,1) distribution. If this random number is less than the missile's Probability of Kill (P_k) then the aircraft is destroyed, otherwise the missile is counted as a miss.

The other case is when the missile has not yet reached its target. Then the heading and the pitch angles are updated and the missile's location is moved. This entire process repeats once per simulation loop until either the missile misses or the aircraft is destroyed.

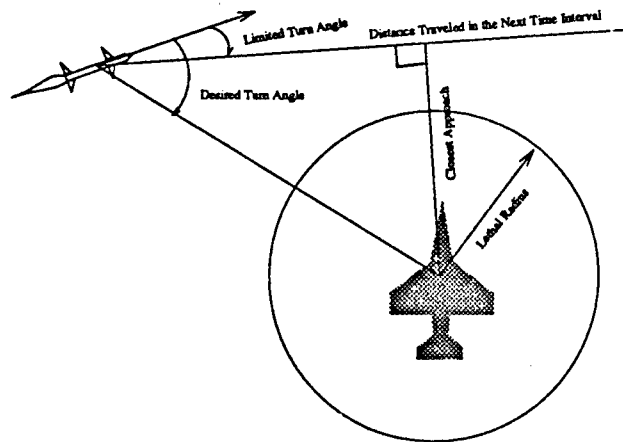


Figure 5.10. Missile Closest Approach

5.9 Random Number Generation

The random draw used with the missile's P_k is the only random feature in ATACS+. Since Meridian's version of Ada does not supply a random number generator, one had to be written. ATACS+ uses an algorithm presented by William H. Press, et. al. in *Numerical Recipes in Pascal* (24). This random number generator was translated from Pascal into Ada and instantiated as a generic function. This function uses a single linear congruential generator and shuffling (24:220). This process is diagrammed in Figure 5.11.

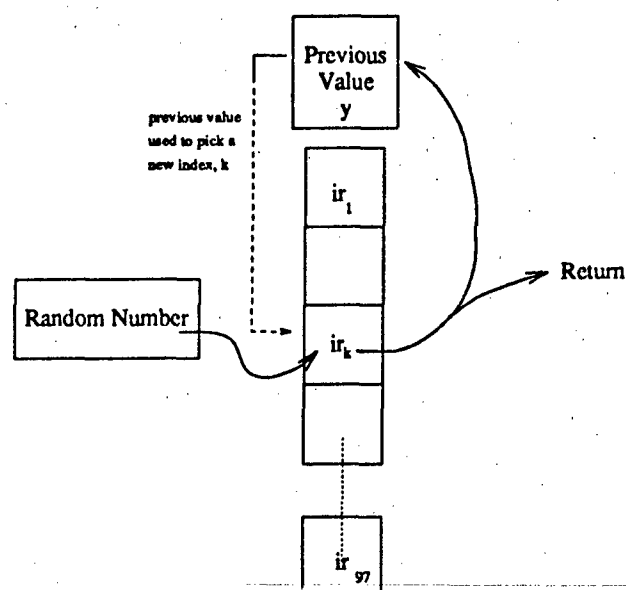


Figure 5.11. Random Number Generation

The first call to the random number generator initializes each of the ninety-seven elements of the ir array. These elements are initialized using the formulas:

$$Seed = (150889 - Seed) \text{ Mod } 714025$$

$$ir(j) = (1366 * Seed + 150889) \text{ Mod } 714025 \quad j = 1 \dots 97$$

The seed value is supplied by the user before the simulation begins. Once the ninety-seven elements are initialized, one more random number is created and is stored in the variable y shown in Figure 5.11. This variable is used to pick a random number from the array each time the function is called.

When a call is made, the value stored in y is converted to an array index using the equation:

$$k = \text{Integer}(1 + \frac{97 * y}{714025})$$

The $ir(k)$ element of the array becomes the new value of y and is the returned value of the function after is is divided by 714025. A new random number is generated to replace the $ir(k)$ element.

This function has an effectively infinite period. Its main limitation is that it returns one of only 714,025 possible values. The values are equally spaced in the interval [0,1) (24:220). This is certainly not a problem in ATACS+ since only a limited number of calls will be made to this function.

5.10 Combat Processes

The general philosophy and assumptions for the combat processes were laid out in Chapter IV. The following sections will add more detail and depth

5.10.1 GCI Search. As mentioned in Chapter IV, the GCI's search radar uses the "cookie-cutter" method. The GCI.Search procedure loops through the aircraft array, computing the distance from the airbase to each of the "Red" attackers, or bogeys. If this distance is less than the GCI radar's range then the bogey is added to the target list. After searching the entire aircraft array, the target list, which is

actually an array, is sorted using a bubble sort. The aircraft posing the greatest threat to the airbase is placed at the top of the threat list. Threat is analogous to the time it will take the bogey to reach the airbase at its current speed. The search procedure is summarized by the following algorithm:

```
for I in Aircraft_Array loop
  if Aircraft_Array(I) is Red and not already killed then
    if Bogey_Range ≤ GCI_Radar_Range then
      Threat = Bogey_Range / Speed
      Add I and Threat to Threat_List
    end if
  end if
  bubble sort by threat, smallest value first
end loop
```

5.10.2 Target Assignment. The concept behind the target assignment process was described in Chapter IV so this section will be used to describe how those concepts were transferred into Ada code. The Target_Assignment procedure begins by looping through the aircraft array looking for "Blue" aircraft which do not have an Assign_Status of "Kia." The procedure then takes the number of living "Blue" aircraft, and the number of "Red" aircraft on the threat list, selects the smallest of the two, and assigns this as the number of Red-Blue pairings.

The procedure then loops through the threat list until the number of pairings is reached. During each loop the "Red" aircraft from the threat list is compared to the available "Blue" aircraft. The closest "Blue" aircraft, not already assigned to another "Red" aircraft, is paired with the "Red" aircraft from the threat list. "Blue" aircraft which are selected have their assignment status updated and the

"Red" aircraft is assigned as its objective. The procedure is outlined in the following algorithm:

```
for I in Aircraft_Array loop
    count available Blue aircraft
end loop
Num_Of_Pairings = min(Num_Available_Blue, Num_Of_Threats)
for I in 1..Num_Of_Pairings loop
    search through the available Blue aircraft for the closest to Threat(I)
    assign Threat(I) as the objective of closest Blue aircraft
end loop
```

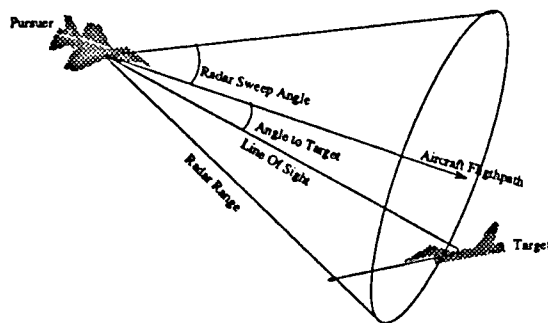


Figure 5.12. Aircraft Search Geometry

5.10.3 Aircraft Search. The Aircraft.Search procedure is almost identical to the GCI.Search procedure except that the aircraft uses a three-dimensional "cookie-cutter." The aircraft's search area is a conical shape extending from the nose of the aircraft as shown in Figure 5.12. The Aircraft.Search procedure loops through the aircraft array looking for aircraft from the opposing side. Each time it finds an opposing aircraft it computes the distance to it. If the distance to the potential target is less than the aircraft's radar range then the angle between the line

of sight (LOS) and the searching aircraft's heading if found by using the dot product of the flight path vector and the LOS vector. If this angle is less than the aircraft's radar sweep angle then the opposing aircraft has its Alert_Status set to "Detected." If the searching aircraft is "Red" it will determine which of the detected aircraft is closest and will target it. "Blue" aircraft will only target the aircraft assigned to it by the GCI. This logic is outlined in the following algorithm:

for I in Aircraft_Array loop

 if Aircraft(I) is not the same color as searching aircraft then

 compute the distance between them

 if distance $\leq$ radar_range then

 compute angle to target = $\cos^{-1} \frac{Hdg \cdot LOS}{||Hdg|| ||LOS||}$

 if angle to target $\leq$ radar_sweep_angle then

 Aircraft_Array(I) is detected

 if searching aircraft is Blue then

 if Aircraft(I) is assign target then

 set Acquire_Status = Radar

 else

 ignore this aircraft

 end if

 else - must be a Red aircraft

 if this is the closest target

 make Aircraft(I) the target

 set Acquire_Status = Radar

 else

 bypass this target

 end if

 end if

 else

```

        out of search area, ignore
    end if
else
    out of range, ignore
end if
else
    Aircraft(I) is the same color, ignore
end if
end loop

```

5.10.4 Aircraft Engagement. The Aircraft_Engage procedure found in the Ac_Pkg package is quite simple. Only four criteria are used when deciding whether to engage or not. They are:

1. Does the aircraft have any missiles left?
2. Does the aircraft not have another missile already in flight?
3. Is the target within the missile's range?
4. Do the rules of engagement allow the aircraft to fire at this time?

If all of these questions can be answered affirmatively, then the next available missile in the missile array is launched with the aircraft's current Objectstate and missile limitations. This method had to be used because Meridian's Ada compiler does not allow unconstrained arrays to be attached to record structures. Otherwise each aircraft would have been given its own array of weapons instead of drawing from a central array of missiles.

5.10.5 Other Combat Processes. The remaining combat processes, aircraft maneuver, aircraft motion, missile pursuit and missile motion were each covered in earlier parts of this chapter.

This chapter has detailed the most important procedures and functions of the combat model portion of ATACS+. The graphics routines were purposely not discusses because an Appendix has been prepared which covers the implimentation of the animation graphics.

VI. Conclusions and Recommendations

6.1 Introduction

This final chapter brings together the experiences drawn from this thesis. Several conclusions are made about combat modeling on the PC, the Ada language and Meridian Software Systems' version of Ada. Other conclusions are made about Ada's use as a combat modeling language, and several future enhancements are discussed. Finally, recommendations are made.

6.2 Conclusions

This section reviews the conclusions drawn from the experience of this thesis. Since this thesis was done as a proof of concept, these conclusions are limited to the area of combat modeling in Ada. Specifically, conclusions concerning combat modeling on a personal computer and using Meridian Software Systems' implementation of Ada on a personal computer are covered. Then general conclusions about combat modeling in Ada are given.

6.2.1 Combat Modeling on the PC. One of the stipulations laid out at the beginning of this thesis effort was that ATACS+ should be created on a personal computer. In retrospect, this was not the best choice. Although today's PCs are computationally faster than ever before and they possess significant graphics capability, the PC is still no match for the performance of the workstation. AFIT has invested heavily in workstations so that now, students use workstations more frequently than they use the school's personal computers. Combat modeling requires significant computing resources to create a model which captures even a limited amount of the reality of combat. The workstation is a better platform for doing this. Some of the problems encountered during this thesis effort would have been avoided if a workstation had been used. Ada is much more prevalent in the workstation

environment so some of the problems experienced with the PC version of Ada might have been avoided if a workstation had been used instead of a PC.

6.2.2 Meridian Ada. After using Meridian's version of Ada for approximately six months of intensive programming, the conclusion is that Meridian Ada is an "immature" implementation. While programing the graphics portion of the model, considerable time was spent overcoming poor documentation. The critical technique used in the graphics was the Ada to assembly language interface. This technique's documentation is almost non-existent. Meridian provides no examples and does not list which assemblers are compatible and which are not. Also, when using assembly language routines, certain assembler-generated object files are not compatible with the Meridian Linker; therefore another linker must be used. Which linker to use is not covered in the documentation either. After several hours of experimentation and numerous telephone conversation with Meridian's Technical Support staff, an acceptable interface was established. However, this could only be done for the 16-bit, 80286 mode. This means that the superior speed and computational power of the 80386 processor was not used. This makes the finished program much slower than it should be.

Another problem supporting the conclusion that Meridian Ada is an "immature" implementation was found while writing the preprocessor. Two major errors in the compiler were identified. Meridian's Technical Support staff was aware of one of the errors but not the other. The first error causes Meridian's User Interface library to use up the computer's heap space (part of the computer's memory). This causes the computer to "lock-up", requiring a re-boot. The second error was an incompatibility with the personal computer's VGA card. Only a single model computer was found to have the compatibility. Unfortunately, this model is the standard 386 computer purchased by the Air Force. These problems caused the rewrite of over 2000 lines of code and the addition of over 3000 extra lines.

Due to Ada's limited use outside of the military, there exists very little third party software or literature. A more widely used language such as C or C++ has a variety of reference material from which to draw. When creating the graphics and the preprocessor, no reference material could be found on the appropriate subjects. Instead, books on C-based graphics were used and the example code was translated to Ada. This translation does not always result in code which takes advantage of the unique qualities of Ada.

6.2.3 Ada for Combat Modeling. One of the main objectives of this thesis was to provide the Department of Operational Sciences with experience in Ada-based combat modeling. Achieving this objective requires some personal observations about this thesis experience. First, Ada's package structure creates much more organized models than seen in traditional FORTRAN-based models. While taking the Combat Modeling course sequence at AFIT, students are asked to investigate existing models, most of these were created before the advent of Ada and are written in other languages, predominantly FORTRAN. These models may contain hundreds of subroutines, nearly all of which are buried in the code. Ada allows like-routines to be grouped together into packages that can then be reused in other programs without having to tear apart the existing model. The reusability of Ada code is one of its greatest strengths. By grouping routines into logical packages, the finished model is much more organized and should be easier to maintain.

From a programmer's point of view, Ada makes for better modeling. This writer has written software in BASIC, FORTRAN, C and now Ada, and is of the opinion that models written in Ada are more robust than models written in FORTRAN and are easier to debug than models written in C. Ada's strong typing feature, (not being able to mix data types in mathematical expressions) reduces mathematical errors. In FORTRAN, mixing Real and Integer data types in a mathematical expression is allowed. But the results can be uncertain when operations such as division are involved. Ada prevents this from happening by requiring the programmer

to explicitly type all variables and by not allowing mixed-type math. Compared to programming in C, Ada is much more syntactically rigid. Ada's use of descriptive variable names and the absence of cryptic operators found in C programs makes Ada much easier to debug and to analyze.

Another Ada feature which makes for superior modeling is the composite data structure. There is no equivalent structure in FORTRAN. By using composite data structures, entities can be created which have their attributes attached. Then the entity can be passed from one procedure to another instead of having to pass each individual attribute. From a programmer's point of view, this makes for a much more organized program.

One further comment concerns the Ada package specification. By using package specifications with "stubs" for bodies ("stubs" are programs that do not do anything), the entire program structure can be laid-out ahead of time. This is even more important when more than one programmer is working on a project. This method creates a much more organized finished product than the old FORTRAN method of "growing" a program one piece at a time. Also, when making changes to a package body, only the body needs to be re-compiled. This feature becomes very important when programs become large and complex. Instead of re-compiling the entire program every time a small change is made, only smaller package bodies need to be.

Package bodies also allow "data hiding." Data hiding is a programming concept where certain routines and data elements exist in a package body but do not appear in the package specification. This "hides" these routines and data from anyone who does not have the package body source code. When working on a large project, involving many programmers, this is very beneficial. Other programmers have access to the package specification and so they can see the interfaces to certain procedures but the details of those procedures are hidden from them.

Ada also reduces system integration time because programs written in one implementation of Ada can be transferred to another computer, re-compiled, and it should run. This is because the Ada language has been standardized by a military standard (MIL-STD) (8). This ensures that all versions of Ada are compatible. Companies which create Ada compilers may add features but all must meet a common baseline in order for the compiler to be certified by the Ada Joint Program Office.

6.3 Enhancements

Although all of the objectives of this thesis were met, many more features could be added to ATACS+. The current beyond visual range (BVR) interception logic could be enhanced to better reflect real world tactics. As an extension to the BVR logic and maneuvers, close-in combat could be added. This would require a more detailed maneuver selection methodology and added engagement logic. Another extension could be the addition of few-on-few logic. The current program only fights multiple one-versus-one engagements. Since Command and Control is an important part of most combat models, a more extensive model of it should be added.

Two major areas of simplification which should be re-evaluated in any future version are the weapons and weapons employment. More types of weapons could be added and aircraft could be allowed to carry a mixture of weapons including guns for close-in combat. Currently, the missile employment envelope and the search area are the same; these should be decoupled. If this is done, much more detail will have to be added to the weapons employment logic.

Finally, more randomness could be added to the model. Although for this thesis, randomness was kept to a minimum to maintain visibility to the combat processes, an expanded model might incorporate more stochastic processes. The search process might be broken in separate search, acquisition and identification processes, each having some element of probability. These enhancements would be

more achievable if the model were ported to a Sun Workstation. This is one of the major recommendations covered in the next section.

6.4 Recommendations

After having studied Ada, written the ATACS+ model, preprocessor and graphics, and having taken the combat modeling courses at AFIT, three recommendations are: The ATACS+ system should be set up on the Sun Workstations, learning Ada should be a requirement for all AFIT students, and a closer working relationship should be established between the Operational Sciences and the Computer Science Departments at AFIT.

The original intent of the ATACS system was to make a simple combat model available to as many students as possible. At the time the original ATACS was developed, personal computers (PCs) were more prevalent at AFIT than were workstations. Today that situation is reversed. Most students use the Sun workstations instead of the school's PCs. Putting the ATACS system on the Sun workstations would make it easier for students to use it. As an added benefit, the Sun workstations have a much more powerful processor which would greatly improve the speed of the model. Also, the software development tools available on the Sun workstations are far superior to the PC tools used in this thesis. Better graphics could be built using the Sun's superior graphics capabilities and memory restrictions would not be as critical as they are on the PC. Overall, the Sun workstation would provide a better platform for the ATACS system because it would eliminate most of the problems encountered in this thesis.

One of the purposes of this thesis was to gain experience in using Ada for combat modeling. Unfortunately, Ada seems to have a dubious reputation in many parts of the military community. The experience of this thesis showed that reputation to be largely unfounded. Ada is a computer language rich with features which make for better programs and programmers. The way to push the acceptance of Ada is

to have more people exposed to it. Therefore, all students in the AFIT School of Engineering should be required to take an Ada programming class. By exposing as many students as possible to the DoD standard computer language, Ada will come to be accepted. AFIT is the obvious place for this to happen.

Finally, there needs to be a closer relationship between the Operational Sciences and the Computer Science (CS) Departments. A follow-on to this thesis could be done as a joint Operations Research (OR)-CS project. The CS student could be responsible for establishing the basic ATACS code on the Sun workstation and the OR student could enhance the model. The two students would gain from their interactions. The OR student would gain a better understanding of how the theoretical algorithms studied in the OR curriculum are used in a larger computer program and the CS student would gain the experience of working with a "user" instead of making a program just for the experience. The need for a closer relationship is not just a problem inherent to AFIT, as Carl M. Harris of George Mason University wrote in *Operations Research*:

I believe that the world of the future holds great promise and challenge for OR, and much of its success will depend on our ability to use computers and information creatively, and, simultaneously, to convince the public at large that we can solve important real problems effectively. But we need to understand better the challenges we currently face before we can move comfortably into the future.(12:1031)

6.5 Final Remarks

Over the course of the last six months, this thesis has taught the writer a great deal about both Ada programming and combat modeling. As future students progress through the Combat Modeling sequence of courses, ATACS+ should help illustrate the basic processes used in most high-resolution combat models. But this thesis is only a step in what should be an on-going effort. Hopefully ATACS+ will continue to improve and will become an even better learning tool.

Appendix A. User's Guide to ATACS+

A.1 Introduction

This user's guide is a stand-alone instruction manual to users of the Animated Tutor for Air Combat Simulation (ATACS+). It provides the first time user with an understanding of both the functionality of the program and the deeper concepts behind ATACS+.

ATACS+ is the second generation of an air-to-air combat model which demonstrates the combat processes inherent in most high-resolution combat models. It is not intended to be an analytical tool, but rather an instructional aid. Because of limitations in the personal computer and the intent of the program, the simulated air-to-air engagements are strictly hypothetical and are not intended to be indicative of actual tactical doctrine. Instead, the user should change the input parameters to the simulation and attempt to identify the effects these changes have on the outcome of the engagement. By investigating the sensitivity of various parameters, the user gains an increased understanding of the inner workings of combat models.

ATACS+ provides the user with the ability to create scenarios consisting of multiple aircraft in few-on-few engagements. The underlying mission is the defense of an airbase which is under attack by enemy aircraft. The user selects the type and number of aircraft for each of the two sides. There are six types of aircraft and a combined total of fourteen can be used. Also, the user sets the starting conditions, weapons load and defines the rules of engagement for each aircraft.

This user's guide is divided into nine sections. The first section presents the computer requirements needed to run ATACS+. The second presents installation instructions. The third section describes the components of ATACS+ and how they interrelate. In the fourth section, a more detailed description of the preprocessor is given. In the fifth section, the simulation is covered in detail. The sixth section provides a step-by-step walk-through of a typical ATACS+ session. Section seven

provides some troubleshooting hints and section eight poses several questions which a student user may wish to investigate while using ATACS+. Finally section nine summarizes the commands used to operate the simulation.

A.2 Computer Requirements

In order to run ATACS+ it must be loaded on a 80286 (286) compatible machine (80286, 80386, 80486). The computer must have a Video Graphics Array (VGA) graphics board installed and must have a math co-processor. A mouse is preferable but is not essential. The ATACS+ system has been separated into two parts, so that it does not require extended memory. These parts are discussed in a later section.

A.3 Installing ATACS+

Installation of the ATACS+ software is simply a matter of copying the files from the distribution floppy disk to a directory on a computer's hard disk drive. A simple installation batch file has been included to make this a simple operation. Begin by placing the floppy disk containing the ATACS+ software into the appropriate floppy disk drive. Move control to the drive containing the disk by typing:

a: (return) or **b:** (return)

whichever is appropriate. To begin the installation type:

install (return)

The batch file creates a new directory called C:\ATACS. If this directory already exists, files in the directory could be overwritten. The installation program pauses before copying the files so that the batch file can be interrupted by pressing

(Ctrl) **C**. Do this if you are not sure about your file status. Once the installation is finished switch back to the C drive by typing:

Table A.1. ATACS+ Files

preproc.exe	the preprocessor program
title.exe	the title screen program
sim.exe	the combat model program
default.exe	a utility program to rebuild the default scenario file
buildacf.exe	a utility program to rebuild all six aircraft files
title.win	the title screen image file
install.bat	the installation batch file
f15.acf	the F-15 aircraft parameter file
f16.acf	the F-16 aircraft parameter file
f111.acf	the F-111 aircraft parameter file
mig25.acf	the Mig-25 aircraft parameter file
mig29.acf	the Mig-29 aircraft parameter file
su24.acf	the Su-24 aircraft parameter file
default.scn	the default scenario file
atacs.bat	the batch file which runs ATACS+
scenario.run	the scenario file which connects the preprocessor and the simulation
sc7.fnt	the font file used by the sim.exe program

C: **<return>**.

Change to the newly created ATACS+ directory by typing:

cd\atacs **<return>**.

The directory should now contain the seventeen files listed in Table A.1: A directory listing of these files can be seen by typing:

dir**<return>**

Once you have confirmed that ATACS+ has been successfully installed, you are ready to use it. Continue reading this User's Guide and try the example ATACS+ session described in section six.

A.4 ATACS+ Components

ATACS+ is made up of three major components: a preprocessor, a combat model, and a graphics system. Each component was developed separately and then interlinked to form the ATACS+ system.

A.4.1 The Preprocessor. The ATACS+ preprocessor is a stand-alone program developed to give the user the ability to easily change the simulation program's input parameters. It consists of a series of menus and data input forms which prompt the user for the required data needed to run the combat simulation. A detailed description of these menus and input forms is given below. When the RUN option is selected in the preprocessor, a file, SCENARIO.RUN is created. After the preprocessor is closed, this file is read by the simulation.

A.4.2 The Combat Model. One half of the simulation program is the combat model. This includes the routines which read the SCENARIO.RUN file and compute the motion of each aircraft and missile. Other routines provide decision logic and output reports. The simulation is a time-step algorithm in which each aircraft and missile is an entity defined by a set of attributes. For aircraft, these attributes include the current position, speed, attitude, weapons load, current maneuver, acquisition parameters, aircraft performance factors and engagement rules. The modeling of the aircraft and missiles is covered in detail in the body of this thesis. The graphics routines use the results computed at each step of the simulation to project either a two-dimensional or three-dimensional representation of the engagement.

A.4.3 The Graphics Routines. The graphics routines unveil the inner workings of the combat model and allow the student user to see how a change in input parameters affects the air-to-air engagement. The graphics contain an option to switch from a two-dimensional overhead view to a three-dimensional view anchored

to a fixed point or anchored to one of the combatant aircraft. The graphics routines also provide a status screen which pauses the simulation and allows the user to check the status of the aircraft.

A.5 Using the Preprocessor

A.5.1 Input Methods. The preprocessor has three methods of retrieving inputs from the user: menus, function keys and data input fields. Each is placed on an input form which is a boxed rectangular area on the computer screen. From a menu, a selection can be made by using the ↑ and ↓ keys to move the hi-light box to a desired selection. Pressing **Return** activates that selection. Additionally, on some menus, a selection can be made by pressing a "Hot Key". These are the keys corresponding to the first letter of the option name and are shown in red on the menu. Only the Main Menu has active "Hot Keys".

The second input method uses function keys. These are the keys labeled F1 through F10 (F1 through F12 on some keyboards). If a function key has an active function, its label and function will be listed on the input form. Not all function keys have the same purpose at every stage of the preprocessor.

The third and most common input method is the input data field. These fields show the current value and accept new values typed from the keyboard. To activate an input field, simply move the cursor to it using the ↑ and ↓ keys. Once a field is activated, the current value is shown in reverse video (black lettering on a white background). A new value can be typed over the old value at this time. The Space bar and the Backspace key are active to erase the old value or to correct entries. Once a new value has been entered, it is locked in by exiting the input field. This can be done by either moving to a new field or by pressing **Return**. The width of the field limits the range of values that can be entered. Improper values are avoided by not letting the user exit a field until a proper value is entered. Although a blank

field is considered a proper value, care should be taken to ensure that a value is eventually input. Errors in the simulation will result otherwise.

Some input data fields are attached to menus which give a list of the available input options. The menu is usually accessed by pressing **F1**. Once a selection has been made from the menu, the new value appears in the input data field. As an alternative to using the menu, an acceptable value can be typed directly into the input field.

A.5.2 The Main Menu. The preprocessor begins by displaying the Main Menu depicted in Figure A.1. The five menu options and their effects are discussed below:

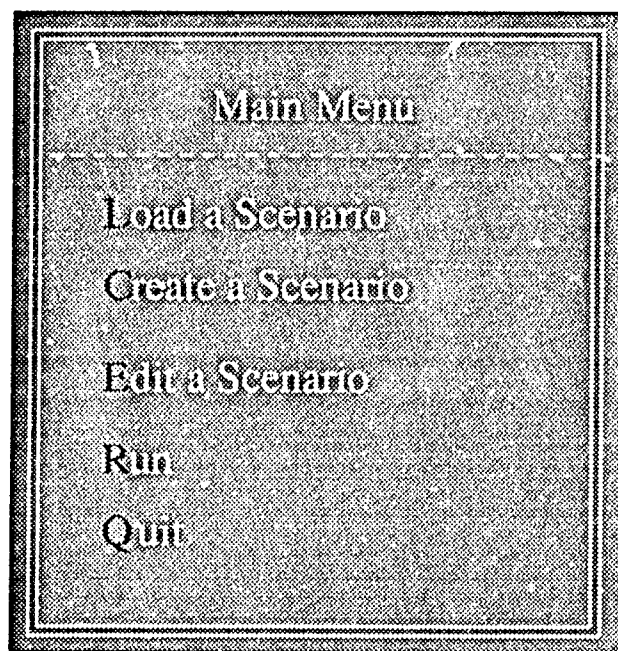


Figure A.1. Preprocessor Main Menu

- *Load a Scenario.* This option brings up a list of available scenarios. This list appears in a scrolling window in the upper left part of the screen. If more

than eight scenarios are available, scroll arrows will appear at bottom of the window to designate the available scroll direction. A scenario file is selected by using ↑ or ↓ to place the hi-light box over a scenario file name. **Return** activates the selection and makes the program load that name as the active file. After making a selection, the file list is removed from the screen and control is returned to the Main Menu.

- *Create a Scenario.* This scenario allows the user to create an entirely new scenario file. When this menu option is selected, a default scenario file is loaded as the active file and the main menu is replaced by the Scenario Editor. The Scenario Editor is described in the next section.
- *Edit a Scenario.* This option is similar to *Create a Scenario* except that a file loaded using *Load a Scenario* is sent to the Scenario Editor instead of the default file. This option is only valid if a scenario has been loaded, otherwise, an error message will appear reminding the user to load a file before editing.
- *Run.* This selection stops the preprocessor and begins the execution of the simulation. When selected, another form requires the input of the random number seed, the length of the simulation run, (in seconds), and a YES or NO flag for the creation of an event listing. Once these values have been entered, **Escape** initiates the creation of the SCENARIO.RUN file and begins the simulation.
- *Quit.* The *Quit* selection shuts down the ATACS+ system by exiting the preprocessor without executing the simulation. Control is returned to the computer's operating system.

A.5.3 The Scenario Editor. If the *Create a Scenario* or the *Edit a Scenario* option is chosen from the Main Menu, the Scenario Editor is displayed on the screen. The Scenario Editor is show in Figure A.2. It is an input form consisting of three parts. The top part of the form contains data input fields for the airbase location and

Ground Controlled Intercept (GCI) radar range. In the middle of the form appears a list of the aircraft which are currently available in the scenario. The bottom part shows the active function keys and their assigned functions. Each of these sections, their purposes, and their functions are described below.

DEFAULT.SCN

Airbase Location

East-West (integer miles) → 0

North-South (integer miles) → 0

Airbase Radar (GCI) Range (integer miles) → 150

Available Aircraft

Call Sign	Type	Mission	Call Sign	Type	Mission
Blue	F15	AB_DEF			
Red1	SU24	STRIKE			
Red2	MIG29	CNTR_AIR			

Function Keys

F2	F3	F4	F5	F6
Add an Aircraft	Edit an Aircraft	Delete an Aircraft	Save & Return	Return

Use the Function Keys

Figure A.2. Preprocessor Scenario Editor

A.5.3.1 Airbase Data. The simulation requires three pieces of information to define the airbase; its East-West location, its North-South location, and the range of the GCI radar at the airbase. These are discussed in detail below.

- **Location.** Two data input fields are provided to give the location of the airbase. The units are in miles and may range from -999 to +999. They have a default

value of zero. These positions are relative to an arbitrary origin established by the simulation. Normally the location can be left at (0,0). This makes it easier to place the aircraft. The altitude of the airbase and the entire world is assumed to be zero throughout the simulation.

- *GCI Radar Range.* The detection area around the airbase is defined by the range of the airbase's Ground Control Intercept (GCI) radar. The input data field will accept positive values up to 999 miles and has a default value of 150 miles. The size of the radar coverage determines the reaction time of the airbase defenders. The shorter the radar range, the shorter the reaction time. Defending aircraft will not engage aircraft outside the radar coverage.

A.5.3.2 Available Aircraft. The center part of the form lists the aircraft which have been added to the scenario and will be active when the simulation is run. The list shows the Call Sign, the type of aircraft and the mission assigned to it. The user can create and list up to fourteen aircraft. This listing is also used to designate a particular aircraft when editing or deleting aircraft. Both of these actions are described in the next section.

A.5.3.3 Function Keys. Five function keys are active in the scenario editor. They are:

- *F6 Return.* This function key quits the Scenario Editor and returns control to the Main Menu. Any changes made to the scenario since entering the Scenario Editor are NOT saved.
- *F5 Save & Return.* This function key saves the changes to the scenario and then returns control to the Main Menu. Pressing this key clears the screen and a form requesting the name of the scenario appears. This form is depicted in Figure A.3. The input field will accept names up to eight characters in

length. There is no need to add an extension to the scenario name because the preprocessor automatically adds the extension .SCN.

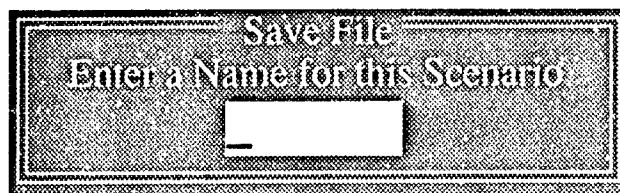


Figure A.3. Preprocessor Name Form

- *Delete an Aircraft.* When this function key is pressed the hi-light box jumps to the Call Sign of the first available aircraft. The hi-light box acts just like it does in a menu. Move the hi-light box with the arrow keys to the aircraft to be deleted. The hi-light box will jump only between the Call Signs of the aircraft. Once the hi-light bar is positioned over the desired aircraft, pressing **Return** deletes the aircraft from the list. The hi-light box then moves back to the first airbase location field.
- *Edit an Aircraft.* This function acts in much the same way as the delete function. Place the hi-light box over the Call Sign of the aircraft to edit and press **Return**. The Scenario Editor is replaced by the Aircraft Editor which is described in detail in the next section. After exiting the Aircraft Editor, the hi-light box appears at the first airbase location field.
- *Add an Aircraft.* This function loads a default aircraft into the Aircraft Editor. After exiting the Aircraft Editor the list of available aircraft is automatically updated with the additional aircraft. As with the other functions, control is returned to the first airbase location field.

A.5.4 The Aircraft Editor. Whenever an aircraft is to be edited or deleted the Aircraft Editor replaces the Scenario Editor. The Aircraft Editor, as seen in Figure A.4, consists of fourteen data input fields arranged into five groupings.

Aircraft Editor

Aircraft Type - Hit F1 for choices

Call Sign (six characters max)

Mission - Hit F1 for choices

Rules of Engagement - Hit F1 for choices

Initial Position:

East-West (integer miles)

North-South (integer miles)

Altitude (integer feet)

Initial Speed (knots)

Search Parameters:

Visual Acquisition range (miles)

Radar Acquisition range (miles)

Radar Sweep angle (+/-) degrees

Engagement Parameters:

Number of Missiles (integer)

Missile range (miles)

Missile Pk (integer 0 -> 100)

Use Escape key to exit

Figure A.4. Preprocessor Aircraft Editor

The top-most group defines the aircraft. The next group defines the initial location of the aircraft. After those fields, the initial speed of the aircraft is displayed. The next group defines the search parameters for the aircraft, and finally, the aircraft's engagement parameters are grouped. Each input field is described below.

- **Aircraft Type.** This input data field accepts a character string for the type of aircraft. Optionally, by pressing **F1**, a menu of available aircraft types is displayed. See Figure A.5. When an aircraft is selected from the menu it is automatically placed in the data field.

At present, there are six types of aircraft available:

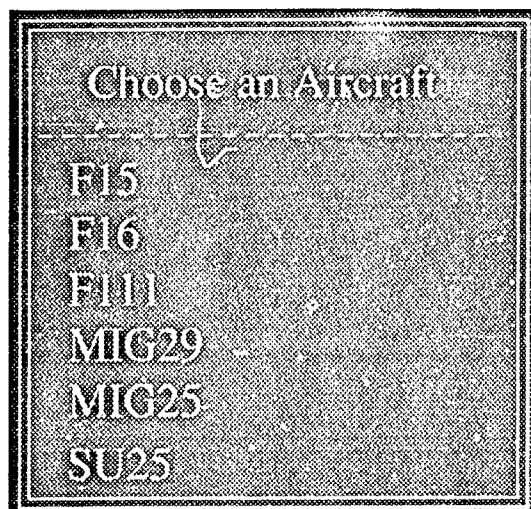


Figure A.5. Preprocessor Aircraft Menu

- *F15*. A U.S.-built air superiority fighter. Used strictly in the airbase defense (Ab.Def) or counter-air (Cntr.Air) missions.
- *F16*. A U.S.-built multi-role fighter capable of performing the same missions as the F15 as well as the Strike mission.
- *F111*. A U.S.-built long-range strike and interdiction aircraft. Normally used for Strike missions.
- *MIG29*. A Soviet-built multi-role fighter. It can perform any of the three ATACS+ defined missions.
- *MIG25*. A Soviet-built interceptor. In ATACS+ it is normally used for Ab.Def or Cntr.Air missions.
- *SU24*. A Soviet-built aircraft very similar to the F111. Normally used in the Strike role.

Although these aircraft were selected for their ability to perform specific missions, they can be assigned to any of the three ATACS+ missions.

- **Call Sign** This input field accepts up to six characters as the Call Sign for the aircraft. The Call Sign is used as a way to reference the aircraft in printouts and status reports. The standard convention is to use the names "Blue1, Blue2, ..., Red1, Red2" etc. As aircraft are added and when the scenario is read in the aircraft are arranged alphabetically by their Call Signs.
- **Mission** This input field works just like the *Aircraft Type* field. By pressing **F1** the menu displayed in Figure A.6 appears. Currently, three defined missions exist. They are:



Figure A.6. Preprocessor Mission Menu

- **Ab.Def.** Aircraft tasked with the airbase defense mission will circle at their initial altitude until assigned a target by the GCI system. Their only task is to destroy attacking aircraft.
- **Strike.** The aircraft assigned the Strike mission attempt to reach the airbase and destroy it. They have no offensive air-to-air capabilities. They fly a straight line toward the airbase unless they are attacked. If they can evade a missile attack, they resume their flight toward the airbase.

- *Cntr_Air*. Aircraft assigned the counter-air mission initially act like strike aircraft - they fly directly toward the airbase. Unlike the strike aircraft, when a counter-air aircraft detects a defender it will engage it. Once the engagement has been decided and if no other defenders are detected, it resumes flying toward the airbase. Once all of its missiles are expended, the aircraft turns and heads away from the airbase.
- *Rules of Engagement*. At the present time, only two Rules of Engagement (ROE) are defined. They can be accessed by pressing **F1**, which shows the menu in Figure A.7.

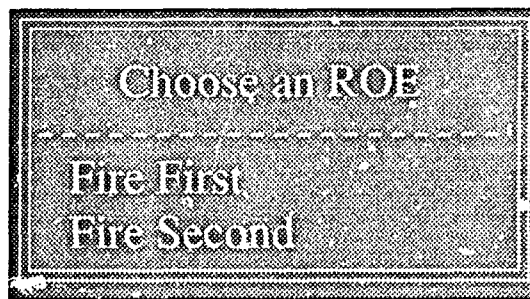


Figure A.7. Preprocessor Rules of Engagement Menu

The two options are:

- *Fire_First*. An aircraft with this ROE will fire on another aircraft at the first opportunity.
- *Fire_Second*. An aircraft with this ROE will not fire a missile until another aircraft fires. That aircraft does not have to be the aircraft with which this aircraft is currently engaged. Any aircraft firing a missile at any other aircraft is sufficient. (Note: do not give every aircraft the Fire_Second ROE or no one will ever fire!)
- *Initial Position*. The aircraft must be given a starting point in three-dimensional space. The following three fields hold this information:

- *East-West.* The East-West location of the aircraft is in units of miles measured from the simulation's origin. If the airbase is placed at the origin, it becomes easier to envision the aircraft's location. On the graphics display, East is to the right.
- *North-South.* The North-South location is also measured in miles with North displayed toward the top of the graphics display.
- *Altitude.* The aircraft's altitude is measured in feet, as the geometric height above the flat earth. This must be a positive number less than 100,000.
- *Initial Speed.* The user must provide each aircraft with an initial speed. The units are knots, or nautical miles per hour. Remember that the aircraft have simulated flight performance, initializing an aircraft with too low an airspeed will make it stall and begin falling. Reasonable initial values would be 250-400 knots for loitering defenders, and 400-700 knots for attacking aircraft. There is no defined upper limit to the initial value but aircraft will not be able to maintain an initial value which is set too high due to the increase in drag at high speeds.
- *Search Parameters.* There are three parameters which define the search area for an aircraft. They are:
 - *Visual Acquisition Range.* This is the range, in miles, at which this aircraft can visually acquire another aircraft.
 - *Radar Acquisition Range.* This is the range, in miles, at which the aircraft's radar can detect another aircraft.
 - *Radar Sweep Angle.* The aircraft's radar sweep angle defines the detection zone. The input value is an integer representing the number of degrees the radar sweeps left and right of the aircraft center line.

- *Engagement Parameters.* Three parameters define the aircraft's ability to engage other aircraft. Since all engagements are missile engagements these parameters define the missile performance. They are:

- *Number of Missiles* This is an integer value which is limited by the size of the field to values between zero and nine. Aircraft assigned the Strike mission should be assigned no missiles since they do not engage other aircraft.
- *Missile range* This is a nominal value used to compute the missile's flight time. Its units are in miles. By changing the missile range, significant differences in the outcomes of engagements can be observed.
- *Missile Pk* The probability of kill for the missile is entered as an integer between zero and one hundred. This value is later divided by one hundred and compared to a random draw from a Uniform(0,1) distribution. If the random draw is less than Pk, the target is destroyed. For example, entering 50 represents a Pk of 0.50 or a 50% chance of destroying a target.

A.6 Using the Simulation

Once the *Run* option is chosen in the preprocessor, it closes and the simulation program begins. The user is presented with the screen shown in Figure A.8. The screen is presented as a raised panel with three cut-outs. The cut-out in the lower left displays the clock. Running time is presented in the form minutes:seconds. This is the simulated time, and does not directly correspond to real time.

Along the bottom of the screen is an elongated cut-out in which messages about events will appear. Figure A.8 shows the example "Blue1 : Fox One!".

The majority of the screen is occupied by the display window. This area displays either a two-dimensional or three-dimensional view of the engagement.

Control of the display is handled through the mouse, the function keys and a pop-up menu. The mouse is used to move the view and to select menu items. The function keys and the pop-up menu are redundant controls – one can be used in lieu of the other.

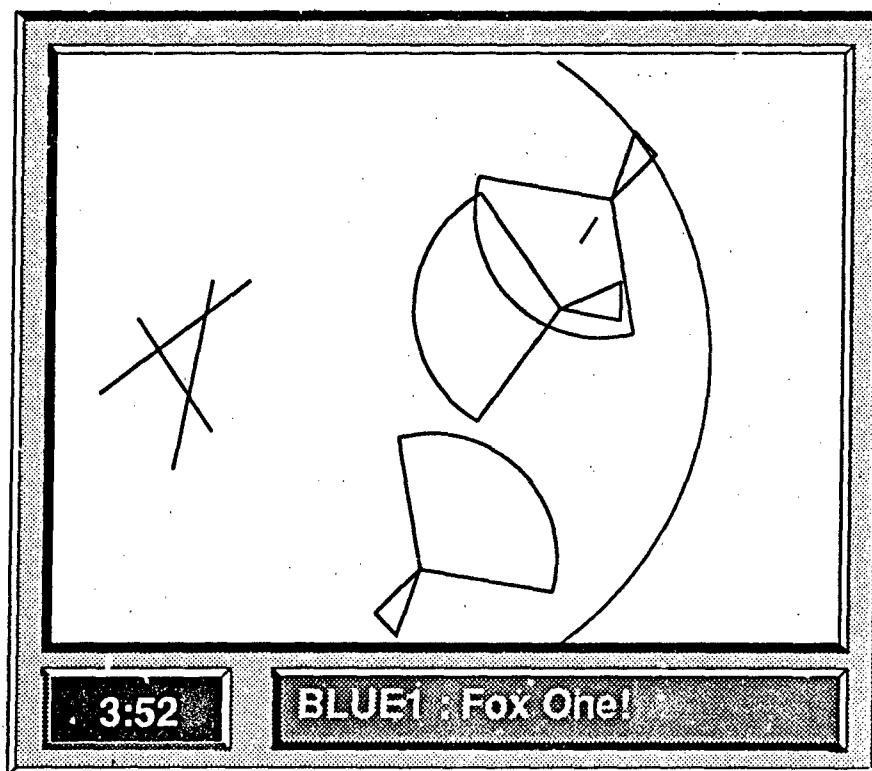


Figure A.8. Simulation 2D Screen

A.6.1 The Run-time Menu. When the simulation is running, pressing **Escape** or the right mouse button pauses the simulation and displays one of the pop-up menus shown in Figures A.9 and A.10. There are eight options on the run-time menus. Figure A.9 shows the menu as it appears in the 2D mode. Figure A.10 shows the 3D mode version. The menu options are:

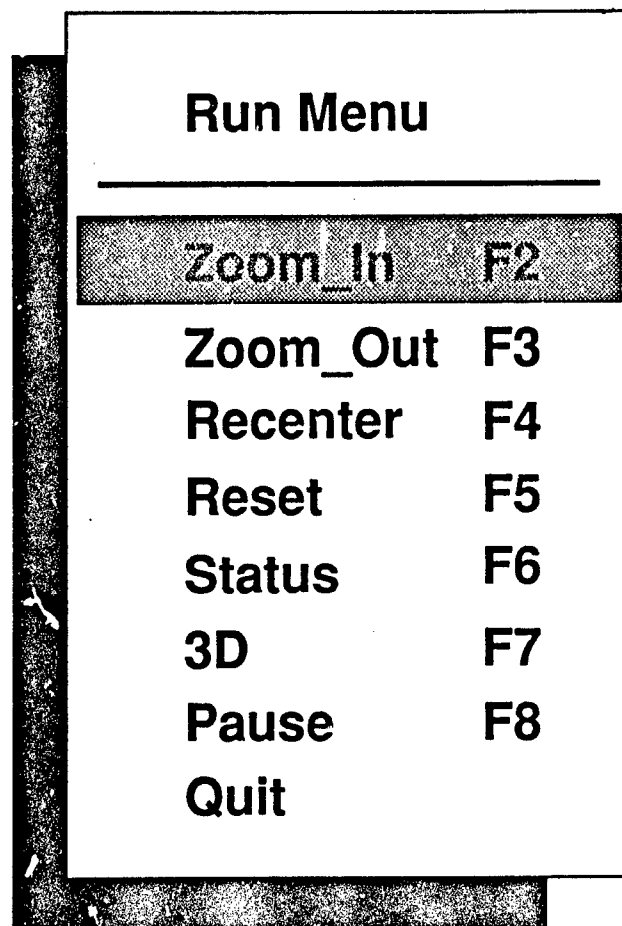


Figure A.9. Simulation 2D Runtime Menu

- *Zoom_In*. This option lets the user zoom in on the engagement. In the 2D mode the user places the cursor over the point where the zoom will be centered. In 3D mode the zoom takes place along the current viewing direction – no cursor is involved. Zoom-In increases the view magnification by a factor of two each time it is used. There is no limit to the number of times this may be used.
- *Zoom_Out*. This option reverses the effect of the last *Zoom_In*. The view magnification is cut in half. The direction of view is not altered.

Run Menu	
Zoom_In	F2
Zoom_Out	F3
Move Viewpoint	F4
Reset	F5
Status	F6
2D	F7
Pause	F8
Quit	

Figure A.10. Simulation 3D Runtime Menu

- *Reset*. This option returns the 2D screen to its original zoom level and resets the center of the screen. In 3D this option only resets the zoom level.
- *Recenter*. This option is only available in the 2D mode. When selected the user is asked to move the cursor to the new center point. The user should move the cursor, then press the left mouse button or **Return**. This point becomes the new center of the display area.
- *Move Viewpoint*. This option replaces the *Recenter* option in the 3D mode. When this option is selected a list of possible viewpoints is presented. A choice is made the same way as on any menu, with the hi-light box. The top choice is *Fixed* which places the viewpoint at a fixed location 20,000 feet above the ground. When the *Fixed* option is selected, the viewing area switches to the

2D display and the user is asked to place the cursor over the desired viewpoint. Once the cursor is positioned and **Return** or the left mouse button is pressed, the display returns to the 3D mode and the new viewpoint is established. The other options list the Call Signs of each aircraft. By selecting one of these options, the view can be anchored to a flying aircraft. The view will turn, pitch and roll with the aircraft.

- *Status*. When this option is selected, the simulation pauses and the display area is replaced by a status screen. This screen lists the aircraft, by Call Sign, and shows their current location, altitude, speed, assignment status, acquisition status and the number of missiles they are carrying. Pressing any key returns the display area to the previous mode. This option is available in both the 2D and the 3D modes.
- *2D/3D*. This option toggles the display between the 2D and the 3D mode. When returning to a previously displayed mode, the center, zoom and viewpoint are the same as when the mode was exited.
- *Pause*. This option halts the simulation temporarily. All graphical functions, such as *Change Viewpoint*, *Zoom* and *Status* continue to operate normally but the simulation does not progress. To resume, just select *Pause* a second time.
- *Quit*. This option lets the user halt the simulation and return to the preprocessor. A confirmation menu will pop up to confirm this selection.

A.6.2 2D Icons. In the 2D mode, aircraft are represented by the icon depicted on the left in Figure A.11. The icon is composed of a triangular aircraft and a sector representing the radar search area. When many aircraft are displayed on the screen the overlapping radar sectors can make the screen very cluttered. By pressing **F1**, the radar sectors can be turned off and only the aircraft icons are drawn. Pressing **F1** again restores the radar sectors. The radar sectors are drawn in the same color as the aircraft when the aircraft is searching. When a radar lock

is established the sector outline becomes yellow. A visual lock results in a white outline.

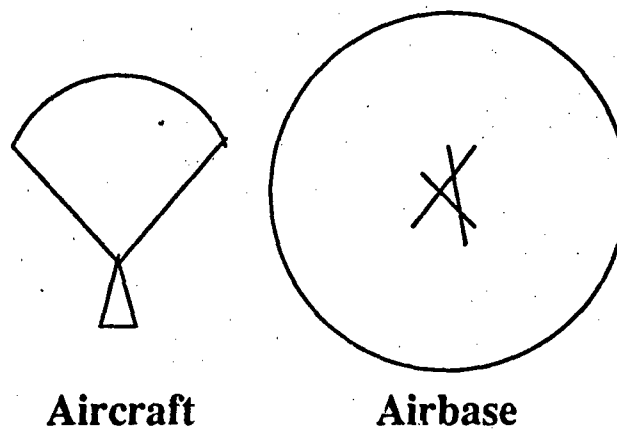


Figure A.11. Simulation 2D Icons

The airbase and the extent of the GCI coverage are represented by the icon on the right in Figure A.11.

A.6.3 3D Icons. In the 3D mode the aircraft have a different icon than the 2D version. This three dimensional representation is depicted in Figure A.12. The icon is the same for all aircraft regardless of the type of aircraft chosen. This icon will translate and rotate in three-dimensional space as the simulation progresses. The ground is represented by the brown portion of the screen and is overlaid by a grid.

A.7 Example ATACS+ Session

This section provides a step-by-step example of a typical ATACS+ session. By following it, the user can understand how the areas previously discussed fit together and operate.

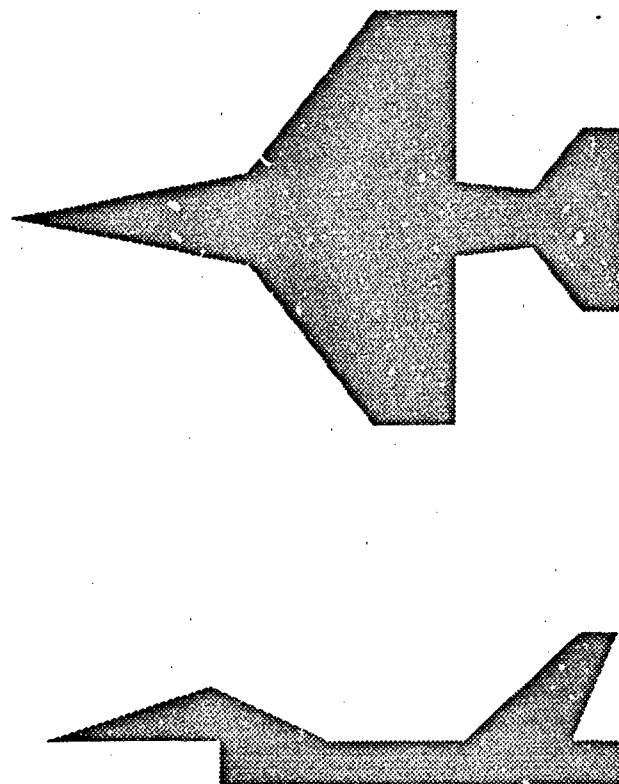


Figure A.12. Simulation 3D Icon

- *Step 1. Set the Directory.* Before beginning, make sure the computer is set to the ATACS directory. If the current directory is not the ATACS directory, change it by typing:

```
cd \atacs <return>
```

- *Step 2. Start ATACS+.* Start the ATACS+ system by typing "atacs" at the DOS prompt.

```
C:\ATACS> atacs <return>
```

After a few seconds the title screen should appear. This screen will remain for several seconds unless a key is pressed in which case it immediately disappears and is replaced by the preprocessor main menu.

- *Step 3. Create a Scenario.* Select the *Create a Scenario* option from the Main Menu. Use the ↑ and ↓ to place the hi-light box over the proper choice and press the **Return**.
- *Step 4. Set Airbase Data.* The Scenario Editor (Figure A.2) has replaced the Main Menu. Now move the hi-light box down to the third input field, the GCI radar range. Change the default value of 150 to 140 by typing over the old value. When done, press **Return**.
- *Step 5. Add an Aircraft.* Now add an aircraft to the scenario. Press **F2**. The Aircraft Editor (shown in Figure A.4) will replace the Scenario Editor. The hi-light box is positioned over the Aircraft Type field. Press **F1** to see the options. Once the menu of aircraft choices appears, move the hi-light box down to the **F16** option and press **Return**. The Aircraft Editor reappears with the Aircraft Type field now reading F16. Move the cursor down to the Call Sign field and type:

BLUE2

The input field defaults to all upper case letters, so the case when typing does not matter. Continue moving down the form, filling in each field with the following data.

Mission = AB_DEF

Rules of Engagement = FIRE_SECOND

East-West = 40

North-South = -40

Initial Speed = 300

Altitude = 10000

Visual Acquisition range = 4

Radar Acquisition range = 40

Radar Sweep angle = 60

Number of Missiles = 2

Missile Range = 20

Missile Pk = 80

When finished entering all of the data, press **Escape**. The Aircraft Editor disappears and the Scenario Editor reappears with the new aircraft added to the list of available aircraft.

- *Step 6. Edit an Aircraft.* Now press **F3**. The hi-light box jumps to the Call Sign of "BLUE1". Move the hi-light box down to "RED2" and press **Return**. The Aircraft Editor appears with "RED2s" data. Change the Initial Speed to 550 knots. Press **Escape** when done.
- *Step 7. Save & Return.* Press **F5** to save this scenario and to run the simulation. The scenario file name form seen in Figure A.3 will appear. Enter the following:

DEMO (return)

Control will return to the Main Menu.

- *Step 8. Run.* It is now time to run the simulation for this new scenario. Press **R** which is the "Hot Key" for the *Run* option. The Run Information form will appear.
- *Step 9. Run Information.* Fill in a random number seed in the first input field. Any integer value up to eight digits in length is acceptable. Press **Return** and fill in the "Run Length" field. Use 600 seconds. Ignore the event printout, leave it set to "N". Then press **Escape** and the screen will clear itself. The user can reproduce this run exactly by using the same scenario and the same random number seed.
- *Step 10. Viewing the 2D display.* The simulation display shown in Figure A.8 should now be visible. The four aircraft should be moving on the display. Note the radar sectors, the GCI zone and the maneuvers each aircraft is making. The

blue aircraft is circling awaiting instructions from the GCI. The red aircraft will all turn towards the airbase and begin flying toward it.

- *Step 11. Radar Sectors off.* On a display with many aircraft the radar search sectors can overlap and confuse the display. By pressing **F1** the radar sectors can be toggled on and off.
- *Step 12. Zoom In.* Press **Escape**. The run-time menu shown in Figure A.9 will appear. The hi-light box is already on the *Zoom.In* option so just press **Return**. A box prompts the user to select a zoom in point appears. Move the cursor with either the mouse or with the cursor keys. The menu and the prompt disappear with the first cursor movement. Place the cursor over the three red aircraft. Press either **Return** or the left mouse button. The display will change showing the magnified view of the aircraft.
- *Step 13. Switching to 3D.* Bring up the run-time menu again by pressing **Escape** or the right mouse button. Select the "3D" option. The display will switch to the three-dimensional mode.
- *Step 14. Viewing the 3D display.* Move the cursor to the right slightly and the display will pan to the right along with the cursor movement. Note that the cursor is not visible but the action is still the same. The heading readout in the lower left corner displays the viewing direction. Keep rotating the view to the right until the red aircraft become visible; this should be at a heading of approximately ninety degrees.
- *Step 15. Zoom In.* The zoom feature acts slightly differently in the 3D mode. Either bring up the run-time menu using **Escape** or the right mouse button or use **F2**. The function key is a short cut. The display can be zoomed to see the attacking aircraft close up. Return the zoom to normal using the *Zoom.Out* or *Reset* menu options.

- *Step 16. Move Viewpoint.* Select the "Move Viewpoint" option from the run-time menu. A list of available viewpoints will appear. Select the *BLUE1* option. The display will change to show the view from the "BLUE1" aircraft. The user may want to watch while the engagement plays out.
- *Step 17. Checking the Status.* By selecting the *Status* option from the run-time menu or by pressing **F5**, the status screen replaces the 3D display. The status screen displays the current location, speed, mission, weapon status, acquisition and assignment status of each aircraft. Pressing any key resumes the simulation and the 3D display.
- *Step 18. Quit.* The simulation will automatically halt when the run length time set in Step 9 is reached. Alternately, the simulation can be halted by selecting the *Quit* option from the run-time menu. There is no function key equivalent to this option. When selecting *Quit* the simulation stop, the screen is cleared and the preprocessor is restarted. When the "Red" strike aircraft reach the airbase or when the run length is reached, the program freezes the screen and waits for a key to be pressed.
- *Step 19. Quit ATACS+.* Once the Main Menu reappears, select the *Quit* option either with the hi-light box or by pressing the **Q** hot key. The screen clears and control is returned to the computer operating system.

These nineteen steps demonstrate the essential functions of ATACS+. The user will want to experiment with the other functions, trying other views and scenarios. To aid in the learning from ATACS+, a list of review questions is provided at the back of this User's Guide. These questions should help draw the user towards an understanding of the combat processes modeled in ATACS+.

A.8 Troubleshooting

As with any software project, occasionally problems will arise in the execution of the program. Attempts have been made to catch errors before they "crash" the program. Since ATACS+ is written in the Ada programming language, much use has been made of the exception handler. These exception handlers allow the program to catch errors and take actions based on the type of error that has occurred. Still, not all errors can be corrected internally so the program may occasionally halt at an unexpected place. When a catchable or correctable error occurs, an error message appears on the screen. But when a catastrophic error occurs in either the preprocessor or the simulation an error message is sent to an error file. These error files have the names, PREPROC.ERR and SIM ERR. These error messages will say what the error was and, in some cases, where the error occurred.

The simulation program is most likely to "crash" because of a missing or corrupted files or a numeric error. Missing or corrupted files are detected by the simulation program and a warning message is sent to the screen. Two types of files are read by the simulation. The first is the SCENARIO.RUN file created by the preprocessor. This file is created each time the preprocessor is run and should not become corrupted. The other type of file is the aircraft configuration files. Each kind of aircraft has an associated file with a .ACF extension. If the simulation displays an error message that one of the .ACF files has been corrupted, or is missing, exit the ATACS+ system and run the utility program BUILDACF.EXE. This program rebuilds all six of the .ACF files.

Numeric errors in the simulation program usually are catastrophic and will halt the program. The best way to correct errors is to change the scenario slightly and re-run the simulation. If the problem persists, try to run a scenario which is known to work. If the problem disappears, there was something wrong with the scenario, if not, then the program has a more serious error and the assistance of an experienced Ada programmer should be sought.

The preprocessor has few numeric processes so is less prone to catastrophic errors. However, the preprocessor must read from and write to files. If a scenario file has been corrupted an error message will appear on the screen. The only scenario file that must not be corrupted is the DEFAULT.SCN file. If this file gets deleted or somehow corrupted, it will not be possible to create new scenarios. If this happens, exit the ATACS+ system and run the utility program DEFAULT.EXE. This program rewrites the file, restoring the original values.

There is one additional problem which may be experienced when using ATACS+. The Ada compiler used to build ATACS+ has an incompatibility with some VGA cards. This incompatibility causes the preprocessor to be unable to draw anything except the Main Menu. If this happens there is, unfortunately, only one solution; move to another computer with a different graphics card. Meridian Software Systems is unable to explain the error or to offer a solution. At present, this error has only occurred on a Unisys 3256DX computer.

Every attempt has been made to create a trouble free learning tool. Where errors may occur attempts have been made to provide corrective utilities and meaningful warning messages. The user will find ATACS+ to be a useful learning tool. By putting the following questions to use with the simulation, the student will maximize that learning.

A.9 Review Questions

- **MODEL CLASSIFICATION**

Was the purpose of the combat demonstration—education and training or analysis ?

- **CLASSIFICATION BY QUALITIES**

What was the model's domain; was land in the domain?

What was the model's span?

What was the model's environment?

Was darkness modeled, and if not, would it make a difference in the outcome?

Was weather modeled, and if not, would it make a difference in the outcome?

What was the model's force composition?

What was the model's scope of conflict?

What was the model's mission area?

What was the model's level of detail of processes and entities?

- CLASSIFICATION BY CONSTRUCTION

Was your involvement required during the combat demonstration?

What time advanced mechanism (fixed step or event step) was used in the combat demonstration?

If fixed step, what was the size of the step, and was there more than one size?

If event step, which events advanced the simulation time?

What time advance mechanism is best for this type of simulation and why?

Is the combat demonstration a deterministic or stochastic model?

Is the combat demonstration a Monte-Carlo simulation?

How many sides were represented in the demonstration?

Was treatment of the sides symmetric or asymmetric?

If asymmetric, was only one or both sides reactive?

• AIRCRAFT

Was altitude a factor in the simulation of the aircraft performance?

Was the aircraft velocity a factor in the simulation?

What aircraft performance measures were used to simulate the aircraft's flight and maneuverability?

What sensors did the aircraft possess?

Were these sensors active?

What aircraft resources were represented?

Were these resources consumed?

What was the outcome if these resources were expended?

- **MISSILES**

What missile performance measures were used in the simulation?

Did missiles share the same time mechanism as the aircraft?

If so, did the missile share the same time step?

Did missiles actively track their targets?

How was the missile effectiveness (killing ability) modeled?

Could a missile miss a target?

- **SEARCH**

What methods were used to simulate the process of search?

Was the process of search explicit or implied?

Which simulated sensors, either explicitly or implicitly simulated, were actively searching for intruders?

Were sensors modeled as entities or attributes of a higher entity?

- **DETECTION**

What method was used to simulate the process of detection?

Was the method deterministic or stochastic?

Could the method be classified as Monte-Carlo?

- **TARGET ASSIGNMENT**

Was the target assignment random or was there an underlying rule?

Would the simulation permit the same target to be assigned to more than one shooter?

How many simultaneous targets could be assigned to one shooter?

- **COMMUNICATION, COMMAND, AND CONTROL**

Was C3 present?

If so, who was in command; what did they command; and was C3 actively used to control entity activities?

Could shooters coordinate or concentrate fire (more than one shooter shooting at a target)?

- **ADVANCE**

Did the entity advance mechanism for aircraft and missiles use a fixed rate throughout the simulation or was the rate altered by changes in the entity's status?

- **TARGET DESTRUCTION**

To destroy a target, was the missile required to impact the target (Hint: think of the time advance and entity advance mechanism)?

A.10 Command Summary

Menu Option	Function Key	2D	3D	Description
	F1	✓		Search Sector Toggle
Zoom In	F2	✓	✓	Zooms In to a point(2D) or along view line
Zoom Out	F3	✓	✓	Zooms Out from the current view point
Recenter	F4	✓		Lets user move the center of the display
Move Viewpoint	F4		✓	Changes the anchor point in 3D
Reset	F5	✓	✓	Sets the display to original condition
Status	F6	✓	✓	Displays the Status Screen
3D	F7	✓		Switches the display to 3D
2D	F7		✓	Switches the display to 2D
Pause	F8	✓	✓	Temporarily halts the simulation
Quit		✓	✓	Halts the simulation

Appendix B. ATACS+ SIMTAX Classification

TITLE: Animated Tutor for Air Combat Simulation

MODEL TYPE: Education and Training

PROPONENT: Air Force Institute of Technology Department of Operational Sciences

PURPOSE: To provide a demonstration of basic fundamentals of air-to-air combat modeling to support student learning of modeling combat

DESCRIPTION:

- Domain: Air
- Span: Local
- Force Composition: Blue and Red
- Scope of Conflict: Air-to-Air Conventional
- Mission Area: Airbase Defense
- Level of Detail of Processes and Entities: Processes are of sufficient detail to demonstrate the phenomena. Blue force entities maneuver and attack Red air targets. Red force entities maneuver and attack either Blue air targets or a predefined ground target.

CONSTRUCTION:

- Human participation: Animated examples require student participation. Scenario creation may require student participation. No student participation required once the combat demonstration begin.
- Treatment of Randomness: Monte-Carlo
- Sidedness: Two sided

LIMITATIONS: Purpose of the model is to provide demonstration only. Some activities and processes are treated simplistically.

INPUT: Optional

OUTPUT: Event report and graphical animation

HARDWARE AND SOFTWARE:

- Computer: IBM AT compatible
- Peripherals: VGA card and color monitor, mouse optional
- Language: Meridian Ada v4.1.1
- Documentation: User's Manual

SECURITY CLASSIFICATION: Unclassified

GENERAL DATA:

- Date Implemented: March 1993
- Data Base: Provided
- CPU time per Cycle: Based on the default scenario file, actual demonstration of air-to-air combat requires 1.5 minutes using a personal computer with a 80386SX processor, 80387 co-processor and operating at 16 MHz.
- Data Output Analysis: N/A
- Frequency of Use: Instructor Dependent
- Users: AFIT

Appendix C. XSharp in Ada

C.1 Introduction

This appendix has been added to this thesis to detail the animated graphics system used in ATACS+. This system is known as XSharp and it produces medium resolution, high-speed animations. Although not spelled out as an objective in this thesis, considerable effort was expended to create these routines because a suitable substitute was not available. This appendix only describes the origins of XSharp and how it was implemented in ATACS+. It does not try to provide the reader with in-depth knowledge of the theory of three-dimensional graphics.

C.2 Background

The XSharp animation routines were created in an on-going series of articles written by Michael Abrash in *Dr. Dobbs' Journal* (1). Over the course of several months beginning in July 1991, he developed a series of graphics routines which were specifically designed to be used for three dimensional animation. The name of this animation project was dubbed XSharp by Michael Abrash because the graphics mode it uses is totally undocumented by IBM, thus the X. He calls it XSharp in the spirit of another on-going project in the same publication called "Dflat" and as he says "who wants a flat animation package?"

The XSharp routines, as presented in the articles, were written in a mixture of assembly language and the C programming language. To use them with the ATACS+ model, the C routines were translated into Ada. The assembly code was only altered so that it would interface with Meridian's version of Ada. These routines became the basis for all of the animated graphics used in ATACS+. The remainder of this appendix details the concepts behind XSharp and some of the graphics procedures used in ATACS+.

C.3 The 320x240 Mode

XSharp uses a video mode (a mode is a particular setting of the screen resolution, color choices, and video memory) which has a resolution of 320 pixels across the screen by 240 pixels from top to bottom. A pixel is the smallest "dot" which can be drawn on the screen. This mode also allows 256 different colors to be used at any one time. As mentioned earlier, this video mode is undocumented by IBM but can be programmed on any VGA graphics card (1:July,133). Of course the obvious question is "Why bother using an undocumented video mode?" The answer lies in five areas where the 320x240 mode is superior to other possible VGA graphics modes.

First, the 320x240 resolution has a 1:1 aspect ratio, resulting in equal horizontal and vertical pixel spacing. This creates square pixels. Square pixels are important because they avoid complicated programming problems (1:July,133). For example, an earlier attempt at graphics for ATACS+ used Meridian's Ada Graphics Library. These library routines used graphics modes which did not have square pixels. In fact, the pixels were taller than they were wide. This caused circles drawn with a library routine to look like ellipses having a major vertical axis. This effect can be overcome by drawing circles as ellipses corrected for the aspect ratio. Unfortunately, Meridian's graphics libraries do not have such a routine. Using square pixels, circles look like circles without the need of complicated corrections.

Second, the video memory can be arranged so that page flipping can be used. Page flipping is an animation technique which is described in the next section. The VGA's documented 256 color mode does not allow page flipping, nor does the high resolution, 640x480, mode (1:July,133).

Third, the 320x240 mode can make use of the VGA's hardware to draw. This can result in a four-fold increase in drawing speed over the 320x200 256 color mode (1:July,133).

Fourth, the 320x240 mode uses one byte to represent a single pixel. Only the 320x200 mode has this feature. This is important because this cuts in half the number of memory access needed to draw a pixel, compared to other VGA modes (1:July,133).

Finally, there is plenty of offscreen memory. The 320x240 mode configures memory in such a way that extra images can be stored, eliminating the need to recreate them when they are needed (1:July,133).

Not all of the features of the 320x240 mode make it a superior choice over other VGA modes. In Michael Abrash's opinion, this mode is more difficult to program (1:July,133). Part of this difficulty lies in the fact that this mode is not widely used and is not even recognized in IBM's documentation.

C.4 Page Flipping

The foundation of high-speed simulation is in being able to create a new image to replace an old image without making the screen flicker during the update. Flicker is very disturbing to the viewer because it distorts the image during the update time. One method for performing flicker-free animation is to use a page flipping technique which is also known as double-buffering.

Page flipping is made possible by dividing the display memory up into two or more pages. Display memory is that memory which is dedicated to holding the image seen on the computer's monitor. On most PCs with VGA graphics cards the display memory size is either 256k (256,000 bytes) or 512k (512,000 bytes). A page is a segment of display memory large enough to hold a single complete display image. If sufficient memory is available, then multiple pages can be defined and the display can be flipped from one page to another. This flip occurs at the screen refresh rate, usually 50-70Hz. This flip happens faster than the human eye can perceive it, so the image appears to change without a flicker. In XSharp, there are three complete display pages. Figure C.1 shows how these pages are arranged in display memory.

The first two pages are used for page flipping. Each of these pages contains 76,800

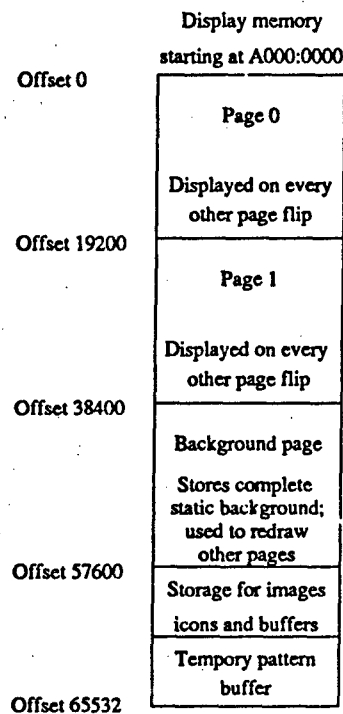


Figure C.1. Display Memory Organization

pixels which take up 19,200 addresses in display memory (1:August,168). The next full page is used as a background page. This page maintains a copy of the static background used in the animation. This background page is used to restore the other two pages before drawing begins. The remainder of display memory can be used to store icons, images, or temporary patterns. In some instances it is easier to draw something once, store it, then copy it to the display page when it is needed rather than drawing it a second time.

The basic methodology of page flipping is shown in Figure C.2. As display page 0 is being shown on the monitor's screen, a new image is drawn on the unseen page. When the unseen page is complete, the display pages are "flipped." The new image is shown on the screen while the formally displayed page, page 0, gets updated.

This page flip does not require the display page to be moved in memory, rather, the

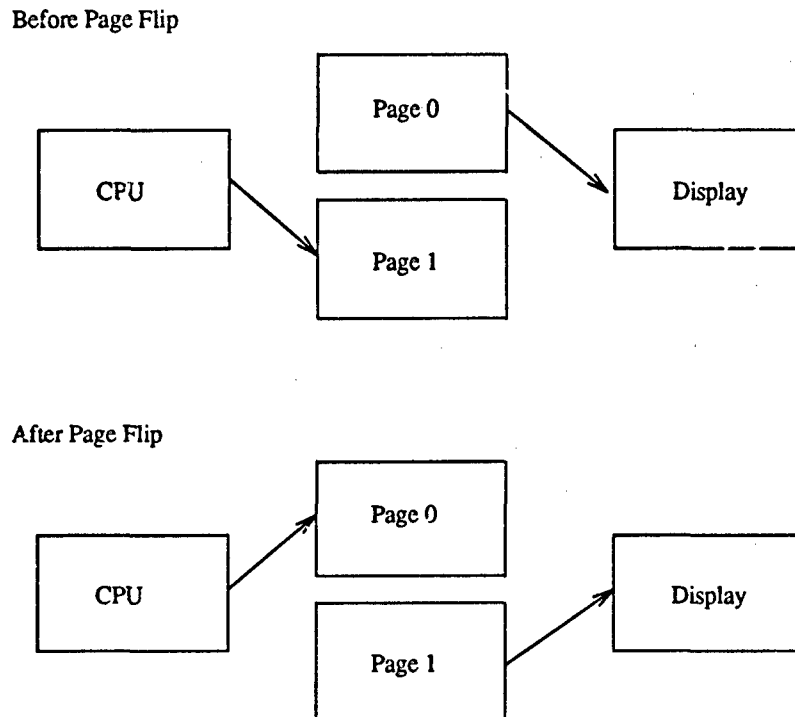


Figure C.2. Page Flipping Methodology

starting point for the next screen update is moved to the memory location of the first pixel of the new display page. This is an extremely fast method of performing animation, but animations using page flipping are slowed by the time needed to draw the new image to the unseen page. The drawing routines must perform the necessary drawing actions as fast as possible.

C.5 Assembly Language Routines

The graphics primitives which form the foundation of XSharp are written in assembly language. This one small excursion away from the “all Ada” philosophy of this thesis was necessary for one reason: speed. Michael Abrash’s opinion is that all critical code for graphics primitives should be written in assembly language (2:21).

XSharp uses assembly language routines to draw a single pixel, draw filled rectangles, draw convex polygons, draw lines, flip display pages and to copy images from one page to another. These routines are the building blocks upon which more complex graphics primitives, such as circles, are built.

In creating the graphics for ATACS+ the first challenge was to make the assembly language routines, originally intended to be used with C programs, compatible with Meridian's Ada. A subroutine written in another language can be called from an Ada program using *pragma INTERFACE* (8:13-15). However, which languages this interface works with is left up to the individual Ada implementor. Fortunately, Meridian does provide an interface to assembly language routines. The biggest difference between the original assembly language routines and the versions modified to work with Meridian Ada, deals with how arguments are passed to the routines. When C passes the address of an argument to the assembly routines it only passes the offset of the location. This is because all data elements reside in the same segment of memory. Only the offset in that segment is needed to find a specific piece of data. On the other hand, arguments passed from a Meridian Ada program are passed as both a segment and an offset. So, in order to access the value of the argument, both the segment and the offset had to be loaded into registers (assembly language "variables"). This was a simple fix which then allowed procedures written in Ada to call assembly language routines.

The only remaining hurdle was how to link the assembly routines with the Ada code. Unfortunately, the assembled code is not compatible with Meridian's linker. Meridian's recommended solution is to use their linker to create a re-linkable object code file out of the Ada code, then assemble the assembly language routines using Microsoft Assembler. Finally, link the two together using Microsoft Link. As mentioned in Chapter VI, this method only works for Ada code compiled in the 16-bit mode. The higher-performance 32-bit mode was tried but Meridian was not able to recommend a workable Ada-Assembler-Linker combination.

Once the basic primitives were established, the rest of the XSharp routines could be built. Not all of these procedures were taken from Michael Abrash's work. Some are original code, while others are adaptations of Meridian's Ada Graphics Library routines.

C.6 Graphics Primitives

All of the XSharp graphics routines are contained in two packages; Xsharp and Screen. The Xsharp package contains all of the functions and procedures for the graphics primitives. The Screen package contains the procedures for manipulating the display memory. The specification of each of these packages contains a description of each visible routine. The package specifications are:

```
-----
-- Package: Xsharp
--
-- Purpose: This is the specification for the 320x240 graphics mode
--          graphics primitives. This undocumented VGA mode was
--          presented in a series of articles by Michael Abrash in
--          "Doctor Dobbs Journal" beginning in July 1991. Some of
--          these routines are modified versions of the code presented
--          in those articles. Others are modified versions of Meridian's
--          Ada Graphics Utility Library and Ada Graphics Library routines.
--          Still others are original code. These routines are intended
--          for use with the 320x240 graphics mode and may not work in
--          other VGA modes.
--
-- Author: David Legge
-- Implementation Date: 23 Nov 92
-- Modification Date:
-- Reason for Modification:
--
-----
package Xsharp is

-- define dimensions of the Mode X graphics screen
Screen_Width  : constant Integer := 320;
Screen_Height : constant Integer := 240;
subtype Screenx_Type is Integer range 0 .. Screen_Width - 1;
subtype Screeny_Type is Integer range 0 .. Screen_Height - 1;
```

```

-- define the offset in display memory of each video page
Page0_Start_Offset : constant Integer := 0;
Page1_Start_Offset : constant Integer := Integer(Long_Integer(Screen_Width*
Screen_Height/4));
Bg_Start_Offset    : constant Integer := Integer(Long_Integer(Screen_Width*
Screen_Height*2/4));

```

```

type Page_Offset_Type is array(0 .. 1) of Integer;

```

```

Page_Offset : Page_Offset_Type := (Page0_Start_Offset, Page1_Start_Offset);

```

```

-- define the types for use in 2 dimensional graphics

```

```

type Point is
  record
    X : Integer;
    Y : Integer;
  end record;

```

```

type Long_Point is
  record
    X : Long_Integer;
    Y : Long_Integer;
  end record;

```

```

type Points is array(Natural range <>) of Point;
type Long_Points is array(Natural range <>) of Long_Point;

```

```

type Rectangle is
  record
    Ul : Point;
    Lr : Point;
  end record;

```

```

-- define the types for use in 3 dimensional graphics

```

```

type Point3d is
  record
    X : Float;
    Y : Float;
    Z : Float;
    W : Float;
  end record;

```

```

type Points3d is array(Natural range <>) of Point3d;

type Xform_Matrix_Type is array(1 .. 4, 1 .. 4) of Float;
type Xform_Vector_Type is array(1 .. 4) of Float;

-- graphical objects have six characteristics, these are grouped
-- together into a single record structure

type Objectstate_Type is
  record
    Objx : Float;
    Objy : Float;
    Alt  : Float;
    Roll : Float;
    Pit  : Float;
    Hdg  : Float;
  end record;

-- graphics can be displayed in either 2 dimensions or 3

type Display_Type is (Twod, Threed);
Show_2d_Or_3d : Display_Type;

-- when clipping lines there are three types of clipping
type Clip_Type is (Top_Clip, Side_Clip, Bottom_Clip, Back_Clip, Off_Screen)
;

-- define the 16 basic colors

subtype Colors is Integer range 0 .. 255;

Black      : constant Colors := 0;
Blue       : constant Colors := 1;
Green      : constant Colors := 2;
Cyan       : constant Colors := 3;
Red        : constant Colors := 12;
Magenta    : constant Colors := 5;
Brown      : constant Colors := 6;
White      : constant Colors := 7;
Gray       : constant Colors := 8;
Light_Blue : constant Colors := 9;
Light_Green : constant Colors := 10;
Light_Cyan : constant Colors := 11;
Light_Red  : constant Colors := 4;

```

```
Light_Magenta : constant Colors := 13;
Yellow        : constant Colors := 14;
Intense_White : constant Colors := 15;
```

```
-----
--
-- Function Are_Equal
--
```

```
-- Purpose: This function is the equality tester for Clip_Type
--           objects.
--
```

```
-- Author: David Legge
```

```
-- Implementation Date: 7 Dec 92
```

```
-- Modification Date:
```

```
-- Reason for Modification:
--
```

```
-- Inputs : Arg1, Arg2 : objects of Clip_Type which are to be compared
```

```
-- Outputs : Boolean : true if the arguments are equal
--
```

```
-----
function Are_Equal(Arg1, Arg2 : in Clip_Type) return Boolean;
```

```
-----
--
-- Procedure: Set_Graphics_Mode_X
--
```

```
-- Purpose: This procedure sets the VGA graphics card to the
--           320x240 graphics mode known as mode X.
--
```

```
-- Author: David Legge
```

```
-- Implementation Date: 21 Sep 92
```

```
-- Modification Date:
```

```
-- Reason for Modification:
--
```

```
-- Inputs : None
```

```
-- Outputs : None
--
```

```
-----
procedure Set_Graphics_Mode_X;
pragma Interface(Assembly, Set_Graphics_Mode_X, "SetGraphicsMode");
```

```
-----
--
-- Function: Shutdown_X
--
```

-- Purpose: This function returns the graphics card to the
-- standard 24x80 text mode.
--

-- Author: David Legge
-- Implementation Date: 21 Sep 92
-- Modification Date:
-- Reason for Modification:
--

-- Inputs : None
-- Outputs : None
--

procedure Shutdown_X;

-- Procedure: Write_Pixel_X
--

-- Purpose: This procedure draws a single pixel to the screen
-- at the location and color specified.
--

-- Author: David Legge
-- Implementation Date: 21 Sep 92
-- Modification Date:
-- Reason for Modification:
--

-- Inputs : X,Y : the screen location of the pixel
-- Pagebase : the offset in video memory or the start of the page
-- Color : the color of the pixel to be drawn
-- Outputs : None
--

procedure Write_Pixel_X(X : in Integer;
Y : in Integer;
Pagebase : in Integer;
Color : in Integer);
pragma Interface(Assembly, Write_Pixel_X, "WritePixelX");

-- Function : Fill_Convex_Polygon_X
--

-- Purpose: This function draws a filled convex polygon
--

-- Author: David Legge

```

-- Implementation Date: 21 Sep 92
-- Modification Date:
-- Reason for Modification:
--
-- Inputs : Poly : an array of 2D points
--          Clip : a rectangle boundry to which the polygon is clipped
--          Pagebase : the offset in video memory of the page to draw to
--          Color : the color to fill the polygon with
--          Xoffset : optional translation of the polygon in X direction
--          Yoffset : optional translation of the polygon in Y direction
-- Output : Boolean : true if polygon is successfully drawn
--
-----

```

```

function Fill_Convex_Polygon_X(Poly      : in Points;
                               Clip       : in Rectangle;
                               Pagebase   : in Integer;
                               Color      : in Colors;
                               Xoffset    : in Integer;
                               Yoffset    : in Integer) return Boolean;
-----

```

```

--
-- Procedure : Fill_Rectangle_X
--
-- Purpose: This procedure calls the assembler interface to draw
--          a filled rectangle.
--
-- Author: David Legge
-- Implementation Date: 21 Sep 92
-- Modification Date:
-- Reason for Modification:
--
-- Inputs : Rect : the boundries of the rectangle to be filled
--          Clip : the boundries to clip the rectangle to
--          Pagebase : the offset in video memory of the page to draw to
--          Color : the color to fill the rectangle with
-- Outputs : None
--
-----

```

```

procedure Fill_Rectangle_X(Rect      : in Rectangle;
                           Clip       : in Rectangle;
                           Pagebase   : in Integer;
                           Color      : in Integer);
-----

```

```

--
-- Procedure Line_X
--
-- Purpose: This procedure is the interface to the assembler
--          routine Line.asm. These routines draw a line between
--          two points.
--
-- Author: David Legge
-- Implementation Date: 15 Oct 92
-- Modification Date:
-- Reason for Modification:
--
-- Inputs : P1 : a two-dimensional point
--          P2 : a two-dimensional point
--          Clip : the boundries to clip the line to
--          Pagebase : the offset in video memory that the line is drawn to
--          Color : the color of the line
-- Outputs : None
--

```

```

-----
procedure Line_X(P1      : in Point;
                 P2      : in Point;
                 Clip     : in Rectangle;
                 Pagebase : in Integer;
                 Color    : in Colors);

```

```

-----
--
-- Procedure Polygon_X
--
-- Purpose: This procedure draws an unfilled, closed polygon.
--
-- Author: David Legge
-- Implementation Date: 16 Oct 92
-- Modification Date:
-- Reason for Modification:
--
-- Inputs : Poly : an array of two-dimensional points
--          Clip : the boundries to clip the polygon to
--          Pagebase : the offset in video memory of the page to draw to
--          Color : the color of the polygon
-- Outputs : None
--

```

```

-----
procedure Polygon_X(Poly      : in Points;

```

Clip : in Rectangle;
Pagebase : in Integer;
Color : in Colors);

--
-- Procedure : Arc_X
--

-- Purpose: This procedure draws an arc having a given radius
-- between to angles. The Arc_Step parameter defines
-- the length of the line segments which will be used
-- to draw the arc
--

-- Author: David Legge
-- Implementation Date: 21 Sep 92
-- Modification Date:
-- Reason for Modification:
--

-- Inputs : Cp : the center point of the arc
-- Radius : the distance from the C₁ to the edge of the arc
-- Arc_Step : the length of the line segment making up the Arc
-- AO, A1 : the starting and ending angles
-- Clip : the boundries to clip the Arc to
-- Pagebase : the offset in video memory of the page to draw to
-- Color : the color of the arc
-- Outputs : None
--

procedure Arc_X(Cp : in Xsharp.Point;
Radius : in Natural;
Arc_Step : in Float;
AO, A1 : in Float;
Clip : in Rectangle;
Pagebase : in Integer;
Color : in Colors);

--
-- Procedure : Sector_X
--

-- Purpose: This procedure draws an arc connected by lines to
-- the center. The parameters are the same as for
-- procedure Arc_X
--

-- Author: David Legge

```

-- Implementation Date: 21 Sep 92
-- Modification Date:
-- Reason for Modification:
--
-- Inputs : Cp : the center point of the arc
--          Radius : the distance from the Cp to the edge of the arc
--          Arc_Step : the length of the line segments making the Sector arc
--          A0, A1 : the starting and ending angles
--          Clip : the boundaries to clip the Arc to
--          Pagebase : the offset in video memory of the page to draw to
--          Color : the color of the arc
-- Outputs : None
--

```

```

-----
procedure Sector_X(Cp      : in Xsharp.Point;
                  Radius   : in Natural;
                  Arc_Step : in Float;
                  A0, A1   : in Float;
                  Clip     : in Rectangle;
                  Pagebase : in Integer;
                  Color    : in Colors);

```

```

-----
-- Procedure : Circle_X
--
-- Purpose: This procedure draws an unfilled circle
--
-- Author: David Legge
-- Implementation Date: 21 Sep 92
-- Modification Date:
-- Reason for Modification:
--
-- Inputs : Cp : the center point of the arc
--          Radius : the distance from the Cp to the edge of the arc
--          Arc_Step : the length of the line segments making the circle
--          Clip : the boundaries to clip the Arc to
--          Pagebase : the offset in video memory of the page to draw to
--          Color : the color of the arc
-- Outputs : None
--

```

```

-----
procedure Circle_X(Cp      : in Xsharp.Point;
                  Radius   : in Natural;
                  Arc_Step : in Float;
                  Clip     : in Rectangle;

```

Pagebase : in Integer;
Color : in Colors);

--
-- Procedure: Clip_Line
--

-- Purpose: This procedure clips the end point of a line to the
-- boundary of a clip rectangle. It returns a variable
-- telling which boundary the line was clipped to.
--

-- Author: David Legge
-- Implementation Date: 25 Nov 92
-- Modification Date:
-- Reason for Modification:
--

-- Inputs : P1, P2 : the ends of a 2D line
-- Clip : the boundary to which the line is clipped
-- P1_Status : which boundary P1 was clipped to
-- P2_Status : which boundary P2 was clipped to
-- Outputs : None
--

procedure Clip_Line(P1 : in out Long_Point;
P2 : in out Long_Point;
Clip : in Rectangle;
P1_Status : out Clip_Type;
P2_Status : out Clip_Type);

--
-- Procedure: Clip_3D
--

-- Purpose: This procedure clips that portion of a line which
-- falls behind the viewer. It does not clip to the
-- view volume. The Clip_LineX routine should be used
-- after using this routine and projecting the lines to
-- a viewport.
--

-- Author: David Legge
-- Implementation Date: 15 Dec 92
-- Modification Date:
-- Reason for Modification:
--

-- Inputs : P1,P2 : three dimensional endpoints of a 3D line

```

--          Line_Status : is Back_Clip if line is behind the viewer
-- Outputs : None
--
-----
procedure Clip_3d(P1          : in out Point3d;
                  P2          : in out Point3d;
                  Line_Status : out Clip_Type);

end Xsharp;

with Xsharp;
-----
--
-- Package: Screen_Pkg
--
-- Purpose: This package provides the user with a control
--          over the two pages of video memory. These
--          procedures are adaptations of the Screen package
--          found in the Meridian Ada Graphics Utility Library
--          (AGUL). They were modified to make them compatible
--          with the undocumented VGA mode known as XSharp.
--
-- NOTE: These routines will only work when the graphics
--       environment has been initialized to the XSharp 320x240
--       VGA mode.
--
-- Author: David Legge
-- Implementation Date: 16 Sep 92
-- Modification Date:
-- Reason for Modification:
--
-----
package Screen_Pkg is

-- only two pages of video memory are allowed
  subtype Display_Page is Integer range 0 .. 1;

  -- Current_Page is the page of video memory which is currently
  -- being displayed.
  Current_Page : Display_Page;

  -- Active_Page is the page of video memory which is being written
  -- to in all subsequent output routines.
  Active_Page : Display_Page;

```

--
-- Procedure: Swap_Page
--

-- Purpose: This procedure toggles the Current_Page
--

-- Author: Captain Dave Legge

-- Implementation Date: 16 Sep 92

-- Modification Date:

-- Reason for Modification:
--

-- Inputs : None

-- Outputs : None
--

procedure Swap_Page;

--
-- Procedure: Set_Active_Page
--

-- Purpose: This procedure sets the Active_Page variable.
--

-- Author: Captain Dave Legge

-- Implementation Date: 16 Sep 92

-- Modification Date:

-- Reason for Modification:
--

-- Inputs : Which_Page : either a 1 or a 0

-- Outputs : None
--

procedure Set_Active_Page(Which_Page : in Display_Page);

--
-- Procedure: Set_Display_Page
--

-- Purpose: This procedure sets the Current_Page variable.
--

-- Author: Captain Dave Legge

-- Implementation Date: 16 Sep 92

-- Modification Date:

-- Reason for Modification:
--

-- Inputs : Which_Page : either a 1 or a 0
-- Outputs : None
--

procedure Set_Display_Page(Which_Page : in Display_Page);

--
-- Procedure: Flip_Page
--

-- Purpose: This procedure swaps the Current and Active pages. It
-- is the foundation of the animation technique of page flipping.
--

-- Author: Captain Dave Legge
-- Implementation Date: 16 Sep 92
-- Modification Date:
-- Reason for Modification:
--

-- Inputs : None
-- Outputs : None
--

procedure Flip_Page;

--
-- Procedure: Copy_Screen_To_Screen_X
--

-- Purpose: This procedure copies a rectangle from one place in display
-- memory and places it in another.
--

-- Author: Captain Dave Legge
-- Implementation Date: 16 Sep 92
-- Modification Date:
-- Reason for Modification:
--

-- Inputs : Source_Rect : A rectangular area to be copied
-- Dest_Upper_Left : The point where the upper left corner
-- of the copy of Source_Rect is to go
-- Source_Pagebase : The offset in display memory of the source
-- page
-- Dest_Pagebase : The offset in display memory of the destination
-- page
-- Outputs : None
--

```

-----
procedure Copy_Screen_To_Screen_X(Source_Rect      : in Xsharp.Rectangle;
  Dest_Upper_Left   : in Xsharp.Point;
  Source_Pagebase   : in Integer;
  Dest_Pagebase     : in Integer );

end Screen_Pkg;

```

C.7 Future Enhancements

Because of time constraints, the XSharp routines were not optimized nearly enough. There are several areas in which these routines could be improved. The algorithm for drawing lines could be sped up tremendously. The same is true of the circle procedure. In another series of articles in *Programmer's Journal*, Michael Abrash presents a much faster way of drawing circles (2). These algorithms could be used to enhance XSharp. A major speed improvement could be achieved if the 32-bit mode could interface with the assembly language routines. The matrix manipulations used in the 3D graphics would be much faster if the native 80386 commands could be used instead of settling for the 80286 commands.

C.8 Conclusions

Throughout the course of this thesis the XSharp graphics have proven to be a programming challenge. Although the resolution is not as high as other VGA modes, its speed has made it, at least, workable in the context of ATACS+. With some further refinements, XSharp could find many more uses, not just for this one project.

Bibliography

1. Abrash, Michael. "Graphics Programming", *Dr. Dobbs's Journal*. 16:7 (July thru September 1991, January thru March 1992.)
2. Abrash, Michael. "Faster Circles for the VGA", *Programmer's Journal*. 8:2 (March/April 1990).
3. Battilega, John A. and Judith K. Grange eds. *The Military Applications of Modeling*. Air Force Institute of Technology, Wright Patterson AFB OH, 1984.
4. Bertin, John J. and Michael L. Smith. *Aerodynamics for Engineers*. Englewood Cliffs NJ: Prentice-Hall, Inc., 1979.
5. Department of the Air Force. F-15A Flight Manual. Technical Order 1F-15A-1. Washington: HQ USAF, 1 January 1992.
6. Department of the Air Force. F-16C Flight Manual. Technical Order 1F-16C-1-1. Washington: HQ USAF, 31 July 1989.
7. Department of the Air Force. F-111F Flight Manual. Technical Order 1F-111F-1-1. Washington: HQ USAF, 9 Feb 1990.
8. Department of Defense. *Ada Programming Language*. ANSI/MIL-STD-1815A. Washington: Ada Joint Program Office, 22 January 1983.
9. Department of Defense. *Computer Programming Language Policy*. DoD Directive 3405.1. Washington: GPO, 2 April 1987.
10. Feldman, Michael B., and Elliot B. Koffman. *Ada Problem Solving and Program Design*. Reading, MA: Addison-Wesley Publishing Co., 1992.
11. Gunston, Bill and Mike Spick. *Modern Air Combat*. New York: Crescent Books, 1983.
12. Harris, Carl M. "Computers and Operations Research: A Marriage for Growth", *Operations Research*. 40:6 1031-1039 (November-December 1992).
13. Hartman, James K. "Lecture Notes in High Resolution Combat Modeling." James K. Hartman, 1985.
14. Hughes, Wayne P. Jr et al. *Military Modeling*. The Military Operations Research Society, Inc., 1984.
15. Joint Analysis Directorate, Organization of the Joint Chiefs of Staff, *The Catalog of Wargaming and Military Simulation Models*. JADAM 207-91. Washington: GPO, 1991.

16. The Joint Chiefs of Staff. *Dictionary of Military and Associated Terms*. JCS Pub 1. Washington: GPO, 1 June 1987.
17. Lambert, Mark et. al. eds. *Jane's All the World's Aircraft 1991-1992*. Alexandria VA: Jane's Information Group, 1992.
18. Lopez, Pablo. *Ada Programming Language and Its Application to Weapon Systems Simulation*. Final Report. Naval Ordnance Station, Indian Head MD, 19 December 1990.
19. Moore, Capt Richard S. *A Computer Based Educational Air for the Instruction of Combat Modeling*. MS thesis, AFIT/GOR/ENS/92M-21. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB OH, February 1992.
20. Morris, William T. "On the Art of Modeling", *Management Science*. 11:B-707-B-717 (August 1967).
21. Nelson, Randy. F-4 Pilot, United States Air Force. Personal Interview. 24 November 1992.
22. Open Ada 386. Version 4.1.1, IBM, 512k, disk. Computer software. Meridian Software Systems Inc., Irvine, CA, 1990-1991.
23. Porter, Richard F. and Doug D. Perry. *Programmer's Manual for the Advanced Air-to-Air System Performance Evaluation Model (AASPEM) Eglin Version 3.3*. Contract F33657-86-C-0084. Columbus, OH: Battelle, January 8, 1990.
24. Press, William H. and Brian P. Flannery et. al. *Numerical Recipes in Pascal: The Art of Scientific Computing*. New York: Cambridge University Press, 1989.
25. Shaw, Robert L. *Fighter Combat Tactics and Maneuvering*. Maryland: United States Naval Institute Press, 1985.
26. Sheldon, Kenneth M. "You've Come a Long Way, PC", *Byte*. 16:336 (August 1991).
27. Smellie, Shawn. Aerodynamics and Performance Engineer, United States Air Force. Personal Interview. 16 October 1992.
28. Software Productivity Consortium. *Ada Quality and Style: Guidelines for Professional Programmers*. SPC-91061-N. Herndon VA: Software Productivity Consortium, 1991.
29. Unger, Brian W., Greg A. Lomow, et al. *Simulation Software and Ada*. La Jolla CA: Simulations Councils, Inc., 1984.
30. Yocom, Michael L. *The Utility of Ada for Army Modeling*. Individual study project. U.S. Army War College, Carlisle Barracks PA, 1990.

Vita

Captain David A. Legge was born on 23 July 1963 in Franklin, Massachusetts. Following graduation from Embry-Riddle Aeronautical University in Daytona Beach, Fl in 1985, he received his commission through the university's Air Force Reserve Officer Training Corps. His first assignment was to the San Antonio Air Logistics Center at Kelly AFB, Tx. During his assignment to the Aerodynamics and Flight Manuals Branch he became the Air Force expert on computer-generated reconstruction of aircraft accidents. He also served as an Aircraft Battle Damage Repair Engineer. In August 1991 he was assigned to the Air Force Institute of Technology where he earned the Degree of Master of Science in operations research. In March 1992 he was assigned to the Concepts and Analysis Division, Aeronautical Systems Center, Wright-Patterson AFB at Dayton, OH.

Permanent address: 3545 Knollwood Dr
Beavercreek, OH 45432

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
This report is to be filled out by the person or persons who are responsible for the collection of information. It is estimated to take approximately 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE March 1992	3. REPORT TYPE AND DATES COVERED Master's Thesis		
4. TITLE AND SUBTITLE Enhancements to the Animated Tutor for Air Combat Simulation		5. FUNDING NUMBERS		
6. AUTHOR(S) David A. Legge				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583		8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GOR/ENS/93M-12		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING / MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Unlimited		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) This study investigates the use of the Ada programming language in combat modeling. It takes an existing combat model, the Animated Tutor for Air Combat Simulation (ATACS), and re-writes it into Ada. The new model also includes enhancements to the original model. ATACS+, as it is now called, is used as a learning tool in combat modeling classes at the Air Force Institute of Technology. ATACS+ consists of a preprocessor and a model, complete with two and three-dimensional graphics and engagement status reports. Conclusions are made about Ada's suitability for combat modeling, some of the features of Ada which can assist combat modelers and the feasibility of combat modeling on personal computers.				
14. SUBJECT TERMS Ada, Computer Programming, Simulation, Combat Modeling		15. NUMBER OF PAGES 138		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	