DESIGN AND IMPLEMENTATION OF TOOLS TO INCREASE USER CONTROL AND KNOWLEDGE ELICITATION IN A VIRTUAL BATTLESPACE

THESIS

Jim J. Rohrer, Lt, USAF
AFIT/GCS/ENG-94D-20

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-20

DESIGN AND IMPLEMENTATION OF TOOLS TO INCREASE USER CONTROL AND KNOWLEDGE ELICITATION IN A VIRTUAL BATTLESPACE

THESIS

Jim J. Rohrer, Lt, USAF
AFIT/GCS/ENG-94D-20

AFIT/GCS/ENG/94-20

Design and Implementation of Tools to Increase User Control and Knowledge Elicitation in a Virtual Battlespace

THESIS

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

Jim J Rohrer, B.S.
Lieutenant, USAF

November, 1994

Acknowledgments

Thanks to my advisor LtCol Marty Stytz for his guidance and confidence. My appreciation extends to my ARPA sponsors whose support helped make the whole project possible. I am also very thankful to my friends in the graphics lab, who made this fun, I would especially like to thank Jim, John, and Milt.

Table of Contents

List of Tables

List of Figures

Abstract

AFIT's Synthetic BattleBridge is an immersive command observatory for viewing large-area activity within a virtual environment. Three basic areas for improvement are addressed: 1) an improved, immersive, user interface, 2) direct control of atmospheric effects, and 3) improved knowledge elicitation by means of remote viewers, a new space scope and an enhanced RADAR scope. These requirements are analyzed and implemented. Some functionality was lost but the results show a general improvement of environmental control and knowledge elicitation.

1. Introduction

1.1 Introduction

My understanding of how great generals win commenced with realizing how not-so-great generals don't win. .. I stood in a valley of the Taebaek Mountains of eastern Korea and watched American artillery pulverize Hill 983 about 1,000 yards in front of me. ...It all worked out as programmed - the superior UN fire-power at last wrested the peaks from the Communists - but the cost was staggering. UN causalities, the vast bulk of them American, totaled 6,400, while Communist losses may have reached 40,000. Yet the UN command gained nothing. Its strategic position in Korea was not affected one iota, and there were almost no tactical gains: behind Heartbreak loomed another ridgeline equally pitted with bunkers. ... The only thing achieved ... was that American command finally realized the futility of frontal attacks against prepared positions. This was no great intellectual awakening... (pg. 19,20)

- Bevin Alexander, How Great Generals Win

All warfare is based on deception.

- Sun Tzu The Art of War

Over the years, a large number of lives, many of them American, have been lost because learned lessons were forgotten, or the fog of war caused good commanders to make ill-fated decisions. Even though military lessons are taught, sometimes they can only be assimilated by first hand experience. The Army Simnet and CCTT projects are formed on this premise. This is the perfect realm for large scale exercises such as Red Flag and REFORGER, where a commander can get a chance to wage a battle, learn some lessons, and not pay for any mistakes. REFORGER had the advantage of training over the terrain of central Europe, where conflict was most likely to occur (Gerhard). Although extremely valuable, the high cost associated with such exercises either severely curtails their frequency or ends them altogether.

New command specific tools available to the modern commander help him make better command decisions. My research is based on the proposition that tools that build upon the experiences of the past will further increase the commander's ability to make informed and timely decisions. During the Civil War, a scout on horseback rode ahead of the army and then reported back to the commander with an assessment of the situation. Today's commander can get near-real-time satellite photographs of the enemy's position and supplement that information with a J-STARS radar image of the area. The actual sorting of the information and conceptualizing the results can be a time-consuming process. The United States had six months to prepare for the attack on Iraq and sift through thousands of intelligence reports and satellite photographs before firing a single bullet. This extended grace period is unlikely to be offered again but the problem of having too much information for commanders to disseminate to troops and assimilate for himself will only get worse. A single commander is unlikely to absorb all of the available information in a timely manner, nor can he get all of the relevant information to the people who need it. On tomorrow's battlefield we will not have the man power to properly sort all intelligence nor the time to do it before the actual fighting. The next generation of command tools must address the issue of information overload while maintaining a cost conscious outlook.

Recent advancements in computer technology have the potential to supplement traditional full scale training exercises and provide some additional commander's tools, without the associated cost or extensive expenditure of material. The development of high speed computer graphics workstations provide a means of visually creating a battle with significant realism at manageable cost. Also, the advancement of wide area computer networks offer the potential for a large number of people to participate in a single, real-time computer established battle. Hundreds of commanders, in-training, can watch the battle while some of them practice making command decisions. The fight can be recorded and analyzed so that other's can learn from the same battle. The same computer technology used to recreate the battle can also be used to assist the commander. Computer operated command tools can integrate the known information about the battle and help the commander fully conceptualize the essence of the battle with neither the aid of a large staff nor the need for significant preparatory time.

These issues have become contemporary topics, especially since the conclusion of Desert Storm. Related research is being sponsored by the Advanced Research Projects Agency (ARPA). The Air Force Institute of Technology has been contributing to this research by developing the Synthetic BattleBridge (SBB) which uses the latest off-the-shelf hardware and software to provide an immersive observatory for large spaces in the virtual environment with a fundamental emphasis on commander rehearsal and command tool development.

1.2 PROBLEM STATEMENT

This thesis effort was directed at improving the situation awareness, monitoring capability, environmental effects, and human-computer interface of the *Synthetic BattleBridge*. The research and work done on the *SBB* over the previous two years has laid down a solid foundation on which to build, however more work remained to be done. (Wilson, Hadix) An improved user interface and an increased set of knowledge elicitation tools will allow the battlefield commander to capture information about the battle and give him a higher degree of conceptualization of actual battle progress. Explicit focus will also be placed on tools for the commander to manipulate the visual atmospheric conditions of the battlefield, and to provide direct view reconnaissance to the commander using remote observation techniques.

The previous version of the SBB provided a useful interface for viewing and moving within the battlefield environment, however the old interface does not readily translate to an interface which emphasizes complete human computer interaction using virtual reality technology. The old interface assumes a constant resolution display device which inherently conflicts with changing virtual reality technology.

This thesis effort is divided into three components. The first component addresses the problem of translating the human-computer interface into a paradigm that is conducive to work within an immersive environment. The second component is to design and implement tools for the commander which expand his ability to view and control atmospheric phenomena from within the virtual environment. The last component concentrates on the design and implementation of tools which provide the commander a means to improve his battlefield situational awareness through the use of remote view reconnaissance.

1.3 APPROACH

This thesis effort was based upon the assumption that all research would be conducted in AFIT's Virtual Environments, 3D Medical Imaging and Computer Graphics Lab. All software would meet the following guidelines:

- All software would make use of the *ObjectSim* development framework as well as the Silicon Graphics *Performer* real-time development libraries.
- All software would be executed on Silicon Graphics Onyx Reality Engine^2 computers running Irix 5.2
- All software would be written in AT&T C++

The thesis work is broken into three primary steps. The first step was the design and implementation of a general user interface that could be used within an immersive environment. Second, the functionality of the old SBB had to be incorporated into the new virtual environment. Lastly, the new tool set had to be designed and implemented to work with the rest of the SBB.

The design of a new three dimensional interface was a large effort conducted by Capt. Jim Kestermann, Capt. John Vanderburgh, and myself. We had to examine the current virtual reality interfaces and consider their advantages and disadvantages and compare their design to our software and hardware requirements. The results of these efforts are discussed in (Kestermann).

CONCLUSION

This thesis will address the issue of expanded commander knowledge elicitation and the expansion of tools for atmospheric manipulation. The thesis proceeds as follows: following this introduction is a discussion on the background relevant to the SBB, followed by requirements, design considerations, and their implementation. The results with illustrations and final recommendations for future work will conclude this thesis.

2. *Background*

2.1 *Introduction*

The *SBB* is an immersive observatory for large spaces in the virtual environment, much as the Virtual Windtunnel is an observatory for aerodynamics effects around an airframe (Bryson,92, Levit 92). An emphasis is placed upon active environments with autonomous vehicles operating within the commander's sphere of interest. The vehicles can be anything from aircraft flown by pilots in simulators to computer generated enemies that are meant to represent fictitious forces. Key issues as to the effectiveness of the *SBB* are its ability to represent the battlefield in a real-time manner, sustaining an adequate graphical frame rate, and substantial environmental interaction with a wide variety of object types, location, and sizes. The visual quality of the environment must be representative of the actual battlefield and must also be customizable to meet changing exigencies so that the battlefield commander can fully control and comprehend the virtual battlespace.

2.2 *Distributed Simulation*

The long-haul network connections allow air defense, attack helicopter, and close air support flight simulators to interact in the same environment (Gerhard). Simulations are no longer confined to a single machine. There can be many people in a machine working together either in close physical proximity or extremely far apart, all without affecting the appearance to the simulation. The Army has, at Fort Knox, over 60 M-1 Abrams tank simulators networked together to allow the crews to fight in the same environment (Gerhard). Apache helicopter simulators at Fort Rucker are also connected to the network, enabling this combined training. The combination of these elements, along with Air Force and Naval resources to make a complete battle, is the work of the Advanced Research Project Agency's (ARPA) WAR BREAKER project. As a way of evaluating future doctrine, new weapons systems and training methods, ARPA is creating this simulation environment.

All simulation traffic is passed on the Defense Simulation Internet. This network conveys important vehicle information from one person to the rest of the people participating in the battle. DIS is an IEEE standard in which there is no central computer arbitrating among the various players (Sheasby). Each participant is responsible for maintaining the state of the world. This means that participants must tell others when they have been hit, their location, orientation, and vehicle type.. AFIT is supporting DIS and WAR BREAKER by creating the Synthetic BattleBridge as a tool for commanders, as well as the Virtual Cockpit, the Satellite Modeler, and the Red Flag Remote Debriefing tool (Diaz, Vanderburgh, Fortner). All four of these projects run on Silicon Graphics machines and the first three have a strong emphasis virtual reality technology.

To date, most of the research done on distributed virtual environments has concentrated on war fighting training for individual or small group. The *SBB* project builds upon previous work in distributed simulation for small groups or individuals to provide a training platform for the needs unique to battlefield commanders. Commanders are currently trained to understand and act upon the situations in a battlespace that are outside virtual environments (Block). The SBB is being developed as an supplement to future training that includes the recreations of battles and the immersion of the commander into that environment.

2.3 Virtual Environments

Anecdotal evidence from previous ARPA projects suggest that virtual environment training effectively prepares individuals for the actual battlefield (Block). This is supported by studies of pilots and air traffic controllers that indicate training in a realistic environment translates to operational gains quickly and inexpensively. Current theory submits that the degree of training benefit is related to the extent of realism in the simulation.

Virtual environments involve the immersion of the user into the world created by the computer. In the past, immersion primarily focused on the creation of the computer generated world that supplemented physical props, such as a mock-up of an airplane cockpit. Recent advances in head-mounted displays are allowing users to completely immerse themselves into the computer generated world, so that no matter where they look they will only see more computer generated images. Ideally the display should be light weight, have a wide field of view, and have very high resolution (Chung, Walser). The computer should track the user's head movement and display a view appropriate to where the user is looking. Such environments offer special challenges for computer-human interface design. The immersion of the user requires a paradigm shift in how input can be given to the application primarily because the user can no longer see his hands.

The update rates of these computer generated worlds greatly affects the interactivity of the application. At approximately 1 frame per second the program is unusable. Six frames per second represents the lower bounds of usability. 20 frames per second marks the point of diminishing returns for program interaction (Airey). The *SBB* is developed upon the *Performer* framework because of its ability to take advantage of its specialization in multi-processor, real-time, virtual reality applications. *Performer* has special support for fixed frame rates and offers run-time profiling of the application (Rohlf).

2.4 Synthetic Battlebridge

The *Synthetic BattleBridge* is in it's third year of research at the Air Force Institute of Technology. Others have also built virtual environments that were meant to either orient commanders in a large scale environment or orient them and allow them to interact with their environment (Stytz). The work completed at the Naval Post Graduate School complements the AFIT work for implementing large-scale virtual environments on commercial workstations (Cooke).

The previous version of the *SBB* by Capt. Kirk Wilson focused on navigation and information displays in a large, distributed simulation environment (Wilson). An underlying assumption of his work was that the user would be sitting in front of a computer screen with easy access to the mouse, keyboard, and high resolution monitor. However, when using immersive virtual reality, many of the features in Wilson's SBB might not be practical because of his non-immersive assumptions for the interface. Also, the state of the art in HMD display resolution and clarity demand a reconsideration of the size graphical interface.

The *SBB*'s need to function as a platform for commander and staff training demands that the interface impart a sense of the spatial orientation, motion, and distribution of objects across the environment. The raw, unanalyzed information is provided using a combination of icons and text. The positions of all vehicles or actors in the environment are computed in real-time and displayed using a three-dimensional rendering of the battlespace. Locators, which are large opaque bubbles, are placed around objects to help the user identify vehicle location and orientation when beyond visual range. Aircraft trails show the flight path of the vehicle over the past several seconds, allowing quick flight path acquisition and velocity estimations. Trails are also applied to other vehicles, such as tanks and missiles, and the models themselves are represented with accurate 3-D geometric representation. All of these features are encapsulated in the previous *SBB*.

The user interface of the *SBB* plays a considerable roll in the final effectiveness of any new functionality. The overall layout and design the interface is covered indepth by Capt. Jim Kestermann, but an overview is provided for reference. The paradigm of the *SBB* interface centers around the concept of the information pod. This pod is the abstract place in the virtual world where the user is sitting. In this pod are all of the controls and displays that are available to the user, such as controls to move around the battle space or a display of the current vehicle locations as seen on a radar scope. These displays and controls are placed on flat panels located around the pod. Logically coherent features appear on the same panel. These features are controlled by pushing buttons on the panel. The buttons are pushed by moving a mouse around which in turn move a cursor on the panel, a push of the mouse button activates a button on the panel. The user is assumed to be wearing a head mounted display with head tracking equipment. This gives him a complete 360° degree field of view without sacrificing any ease of use.

2.5 Conclusion

The development of the *Synthetic Battlebridge* builds upon the work of others. This chapter explored some the development in distributed simulations. The fundamentals of virtual environments were examined and the recent work on the *SBB* was discussed. The next chapter will discuss the requirements for this thesis effort.

3. Requirements

3.1 Introduction

This chapter presents the overall requirement for the *Synthetic Battlebridge*. The requirements described here determine the overall thesis effort. This chapter will proceed as follows: a discussion about the general requirements of the *SBB* which is then followed with the specific requirements. This will then be followed by a chapter conclusion.

3.2 General Requirements

The *Synthetic Battlebridge* is of a virtual observatory into a battle recreated by the computer. This thesis effort is to make enhancements to the *SBB* that facilitate greater knowledge elicitation and control of the virtual environment. This will be accomplished with enhancements to the user interface, additional situation awareness, and improvements that make the environment more visually realistic and manipulateable.

3.3 Specific Requirements

The specific requirements for the *Synthetic Battlebridge* come from ARPA's Lt Col Dave Neyland.

- Make the viewing environment totally immersive
- Create a user interface that can work within this immersive environment
- Incorporate previous *SBB* tools
- Make the environment more visually realistic
- Create clouds and fog
- Incorporate the sun

  - Incorporate the moon
  - Add nighttime
  - Add stars
- Add tools to increase situational awareness
  - Improve the monitoring capability for air and ground actors
  - Add a space resource evaluation tool
  - Create a remote camera
  - Create a video missile

3.4 Conclusion

This chapter presented the requirements for this *SBB* thesis effort. Enhancements are to be made to the user interface. Basic weather representation is required as well as tools to increase the commander's situational awareness. The next chapter describes the design and implementation of these requirements.

4. Design and Implementation

4.1 Introduction

A commander utilizing a virtual reality environment wants to maximize the conceptualization of available intelligence with the least amount of effort. This chapter addresses the design and implementation of enhancements made to the *SBB* The remainder of this chapter will proceed as follows: the software that the SBB is built upon is discussed, which is then followed by an overview of the *SBB* architecture. This is followed by a discussion of how the new tools are designed and how both new and old tools are implemented in the SBB. These characteristics are broken into the classification of weather manipulation, control of movement, and tools to enhance situational awareness. A description of the required hardware, and a conclusion will complete this chapter.

4.2 Foundation Software

4.2.1 ObjectSim / Performer

The *SBB* is built upon the *ObjectSim* framework, which in turn is built upon the *Performer* development environment. *Performer* breaks every application into three processes, the *App* process, the *Cull* process, and the *Draw* process (Performer). The *App* process is responsible for all non-graphical computations. *Performer* repeatedly runs once through the application loop, culls the geometric scene, and then draws the image to the screen. Figure 1 shows the pipelining of the *App*, *Cull*, and *Draw* processes. At time 1 the *App* process will be processing information pertaining to the first frame. At time 2 the *App* process will be working on information pertaining to the second frame, and the *Cull* process will be culling the scene for the first frame. At time 3 the *App* is working of the third frame, the *Cull* is culling the second frame, and the *Draw* process is drawing the first frame to the screen. If the program is being run on a multi-processor machine then each process will run on its own processor.

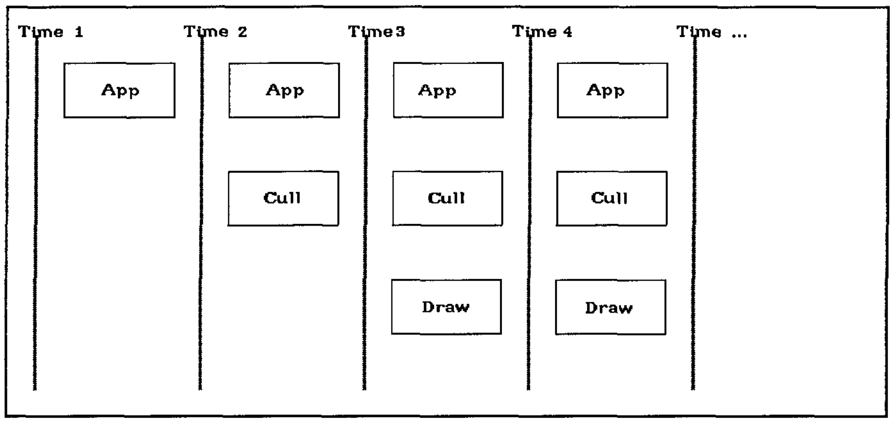


Figure 1. The Sequencing of Performer

The application process is driven by *ObjectSim*, which calls the routine **init** at the beginning of every program. After the initialization **propagate**, **pre-draw**, **draw**, and **post-draw** code is executed by its respective process. Figure 2 shows the process domain of *ObjectSim*, where the **Initialize** and **Propagate** routines are called from the *App* process and the **Pre-Draw**, **Draw**, and **Post-Draw** modules are called from the *Draw* process. The **propagate** code contains all normal functionality and will always be executed by the *App* process, this is were the majority of the application resides. The **pre-draw**, **draw**, and **post-draw** portions of the code are reserved for routines that need to communicate with the draw process by specifying display information such as line drawings color or text manipulation..

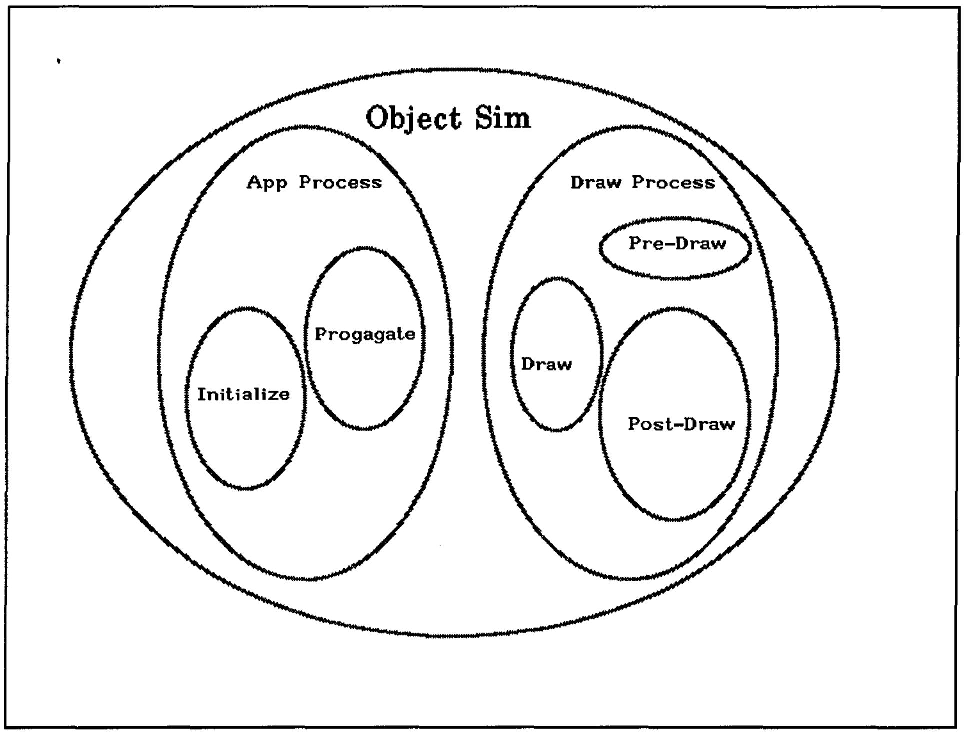


Figure 2: The Process Domains of Object Sim Calls

4.2.2 SBB Architecture

The previous *SBB* assumes, throughout the code, the presence of a non-immersive interface. Because of this, the integration of the enhanced functionality and immersive user interface of the new *SBB* into the previous *SBB* architecture proved impractical. The *SBB* was therefore re-written, old *SBB* functionality was duplicated where ever possible. The previous *SBB* **Vehicle_Manager** class was reused in the new *SBB*. Several new objects where created to facilitate the new user interface and the new enhancements. The class structure as it relates to *ObjectSim* can be seen in Figure 3. This is the complete object hierarchy for the *SBB*. Figure 4 shows **View Player**, and **Vehicle_Manager** as being fixed in their hierarchy. **View**, and **Player** are integral *ObjectSim* components and are therefore expected to be found in designated locations within the hierarchy. The **Vehicle_Manager** must remain under **SBB_Simulation** in order for the module to be effectively reused. The design of the remaining structure is addressed in the subsequent sections.

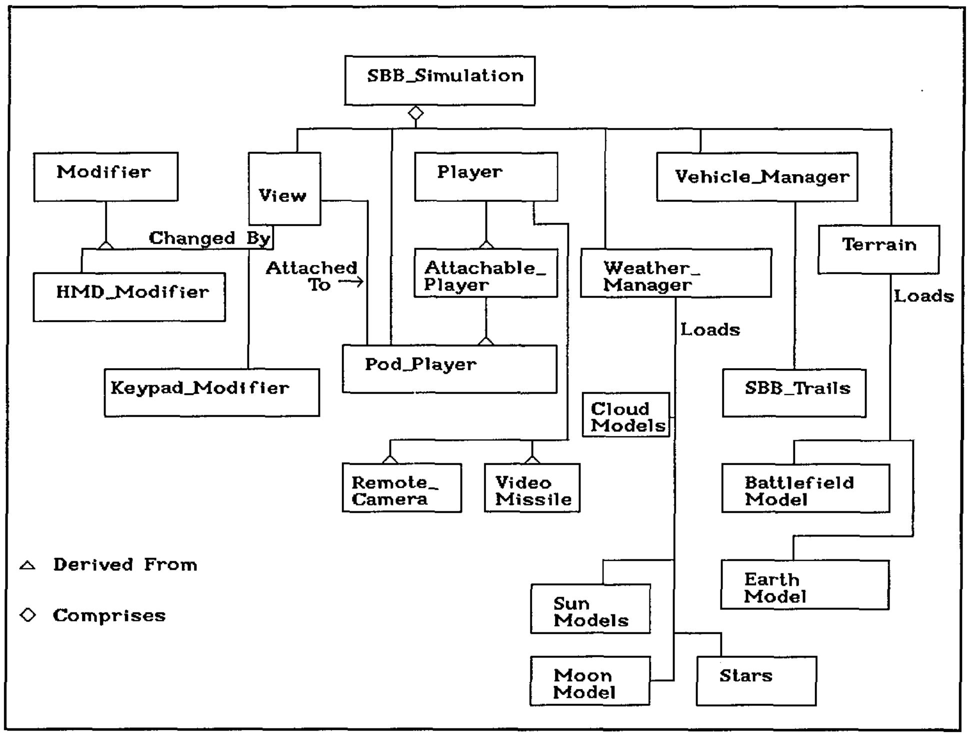


Figure 3: Object Diagram for Synthetic BattleBridge

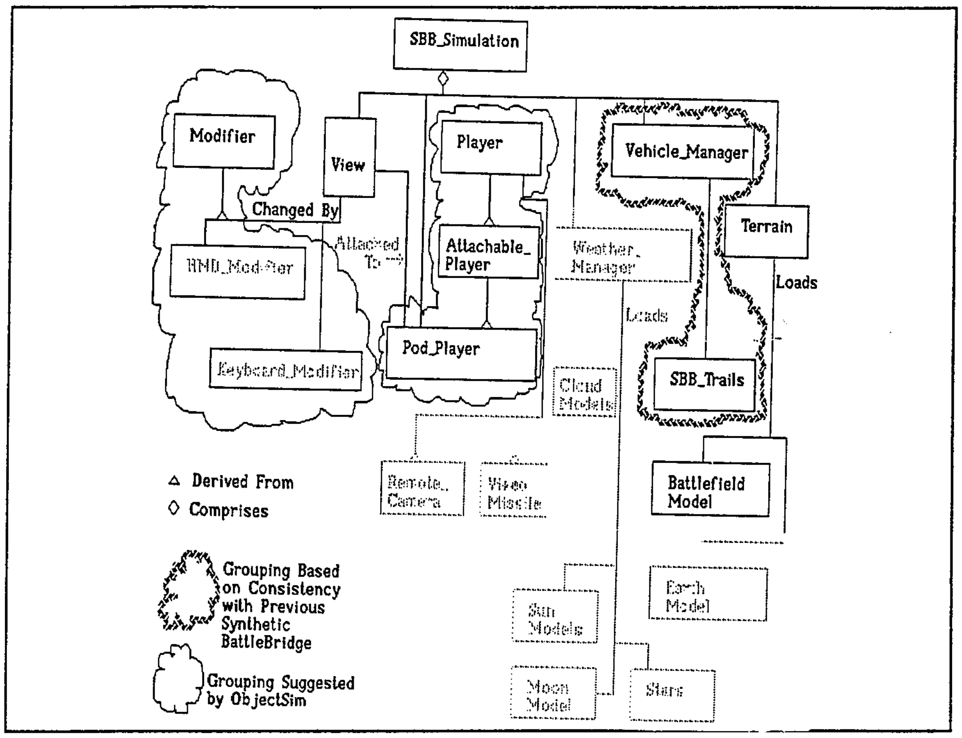


Figure 4: Unmoveable Structures

When a module, say a fog generator, is designed for the *SBB,* the placement of the module within object hierarchy creates a situation which requires design tradeoffs to be made. If the interface for fog is created using an object oriented methodology it is natural to have that interface own a fog object and then invoke fog methods to control environmental fog. In order for this embedded fog to control the environment it must have access to the environment control variables. However, if the current interface is based upon mouse movement, and then a new voice control interface is created, the new voice interface would also have a fog object. Since there is only one environmental control variable that can adjust fog, two different fog instantiations can not be possible. In order to avoid conflicts between interface methods, the interface and the functionality are separated into two distinct objects, one addresses functionality and the other addresses interface. Only one instance of the functionality portion of the tools is ever created, but several versions of the tool's interface can be created. For example, one interface for mouse controlled fog and another interface for voice controlled fog. This method also facilitates placing the functional portions of the code within a portion of the overall hierarchy that is the most appropriate. This allows the functionality portion of the radar scope to lie within the **Vehicle_Manager** and the interface to lie within a player. Figure 5 illustrate the parallel nature of this approach. The left side of the figure shows the methods associated with the **Vehicle_Manager**, the right side of the figure shows the methods associated with the weather module. The lines between methods indicate lines of communications.

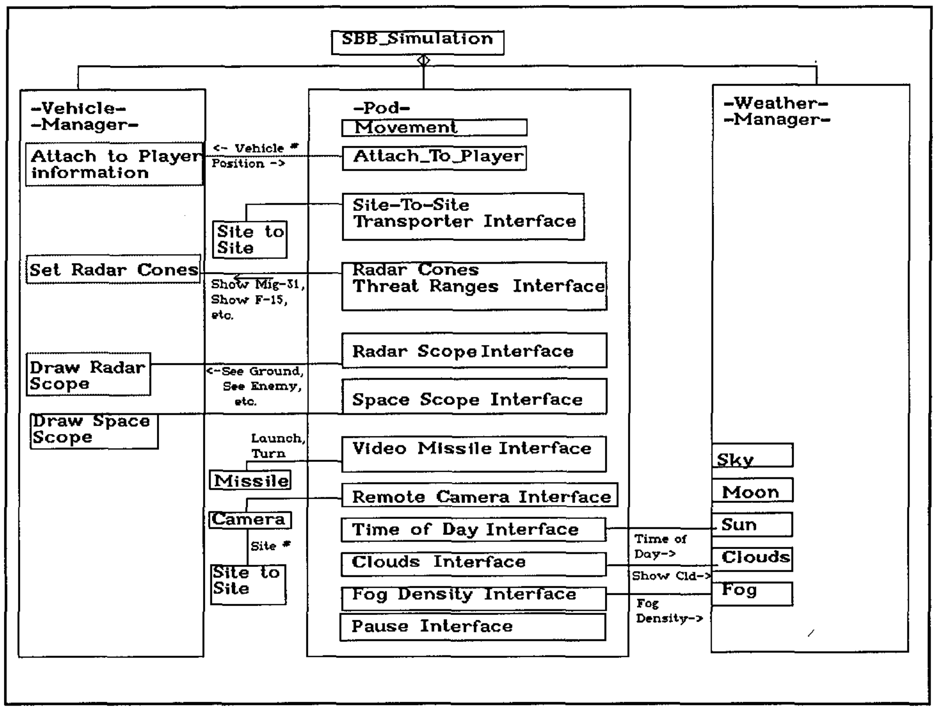


Figure 5: Module Communication

Objects communicate with each other by either invoking class methods or changing feature hooks. Feature hooks are global variables used in one section of code for the purpose of communicating with objects not in the same hierarchy. The entire SBB program follows an Object Oriented Design methodology, but deviations from this methodology arise from the use of global variables that serve as hooks for functionality that cross components. An advantage to this technique is that objects low in the hierarchy can communicate directly with objects elsewhere in the tree. This need manifested itself when the ability to manipulate the weather was added to the *SBB*. The user interface had already been designed and implemented when the **Weather_Manager** was added. The **Weather_Manager** needs several pointers found in the highest level of the hierarchy, and the user interface only needed access to a couple of state variables, such as fog density or time of day to control the weather. There were two basic choices, the first being to put the weather object underneath the user interface, the second being to put the weather object immediately underneath the simulation driver and use a couple of global variables to allow communication with the interface (See Figure 7). Since the user interface had become several levels deep, the primary disadvantage with the former method was the need to reprogram much of the interface to pass the needed pointers down through **Pod_Player** and the Panels and the Sub-Panels and the User Controls, this would make the user interface specific to the *SBB* instead of a generic 3-D interface. Global variables solve the problem with a minimum of extra work and they don't sacrifice the portability of the user interface.

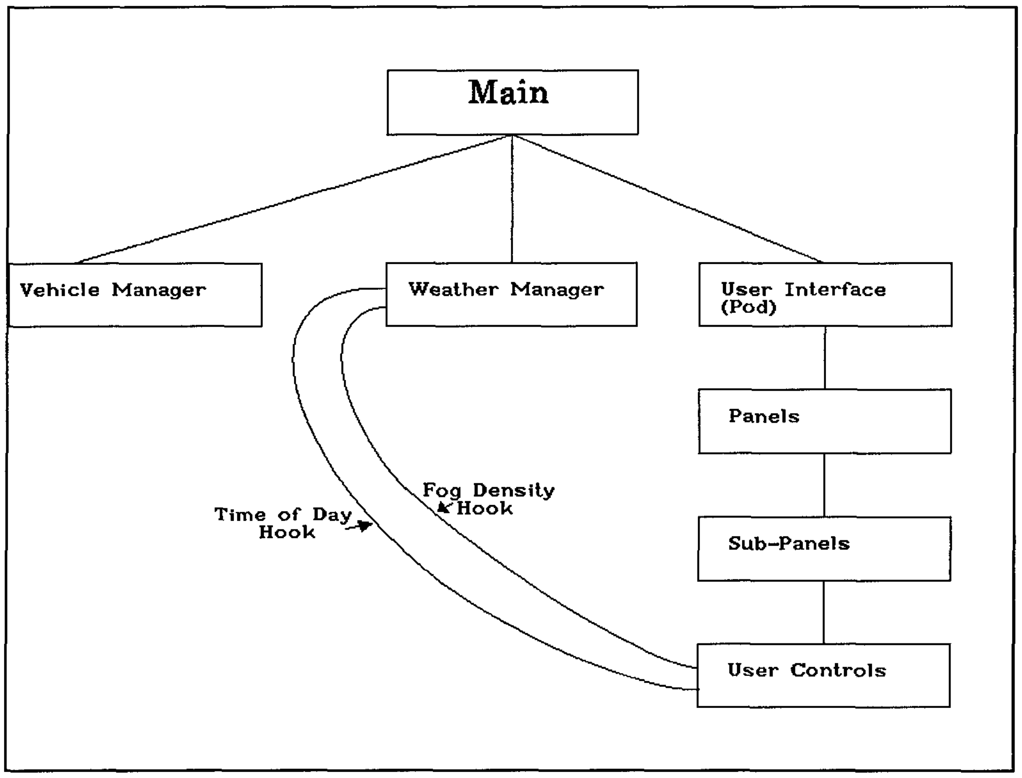


Figure 6: SBB Structure with Example Hooks

Every feature interface within the *SBB* either controls functional hooks or embeds the functionality in the interface portion of code. Figure 5 shows which features are controlled by hooks and which and embedded in the interface.

4.2.2.1 The Terrain

All *ObjectSim* applications must have one terrain. The origin of the terrain defines the origin for the entire simulation. The **Terrain** class transforms the coordinates of all vehicles retrieved from the network and translates them into local coordinates. The *ObjectSim* programming format requires that the **Terrain** class be placed underneath the simulation. Since a model of the Earth is required in the *SBB,* the model is loaded by the **Terrain** object. The alternative was to create a new class that controls the Earth model, however this would not have taken advantage of the **Terrain**'s ability to load models and place them in the world.

4.2.2.2 The View

All *ObjectSim* applications must have at least one instance of a **View** object which determines where the user is looking into the scene. Every view must be attached to an **Attachable_Player**, which determines where the user is in the world. The **Attachable_Player** is analogous to the user's body and the view is analogous to the user's head, where ever the **Attachable_Player** moves to, the **View** follows. The direction in which the user is looking is determined by a **Modifier** object. Since the *SBB* is designed to be an immersive application the **Modifier** should track head movement, this is done by the **HMD_Modifier** class, which is a subclass of the **Modifier** class.

4.3 SBB Architecture

The specific structure unique to the SBB is broken into three components over five levels (see Figure 7). The **SBB_Simulation** forms the top level of this hierarchy. Underneath the **SBB_Simulation** are the remaining three components, the **Vehicle_Manager** class, also known as the **sbb_net_manager**, the **Weather_Manager** class, and the **Pod_Player** class, which functions as the user interface.

4.3.1.1 The Pod Player

The **Pod_Player**, or Pod, is the manifestation of the immersive user interface. The **Pod_Player** is an attachable player and is the only class that the **View** can communicate with. The **Pod_Player** encapsulates all the *SBB* controls as seen in Figure 3.

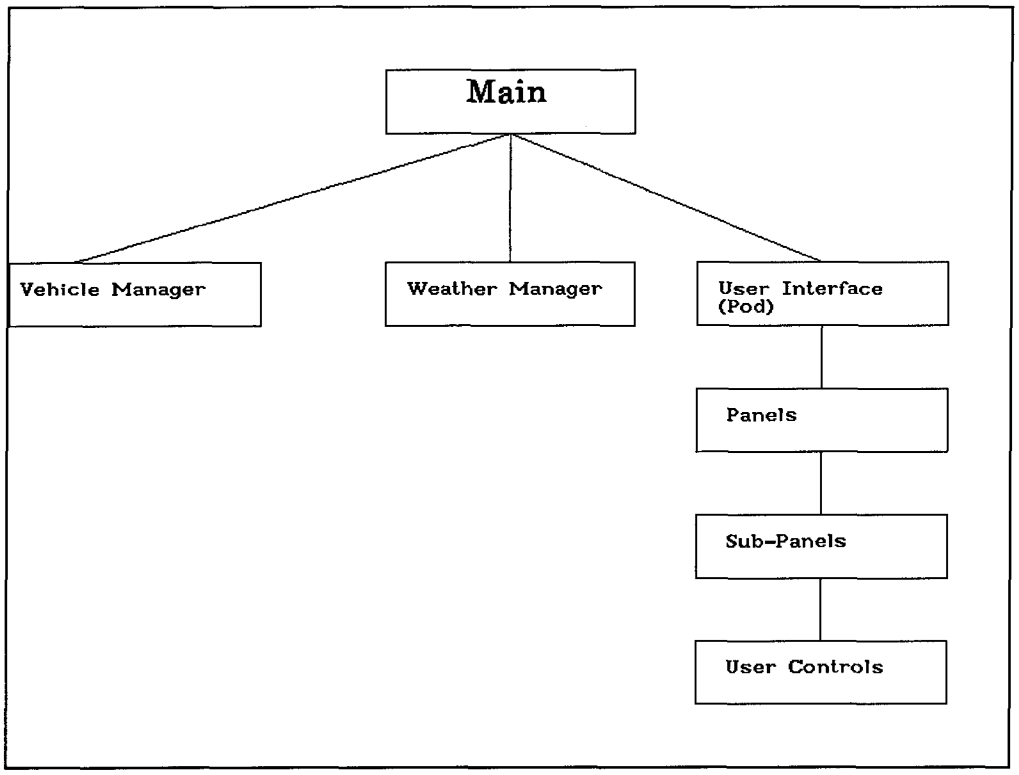


Figure 7: Overall SBB Structure

4.4 Visual Realism

The visual realism of a simulation effects the extent in which a user can immerse himself and concentrate on his objectives. Several aspects of the previous *SBB* environment where unrealistic enough to be noticeable. Improving battlefield effects such as fire, dust, and explosions would increase the realism of the simulation. The creation of such particle system based effects are realistic but computationally expensive, alternative, non-particle, methods provide notional views of the effect at a lesser cost. The Naval Post-Graduate school, among others, have created preliminary version of these effects, consequently this avenue was postponed for the *SBB*. The improvement of vehicle representation and part articulation would also improve the realism of the environment. This area was also rejected because of its lack of relevance to a large scale battle, for example, a rough model of an F-16 communicates its aircraft type just as effectively as a highly detailed model. Atmospheric representation would improve the visual realism of the environment. Weather always has an impact on how a mission is planned and executed. The development of real-time 3-D weather generation is an relatively unexplored topic. These factors lead to the creation of several sample atmospheric effects which can serve as a baseline for future research in real-time weather creation within an immersive environment.

4.4.1 Background, Weather

The virtual reality commander wants control of the environment's weather representation. During a military engagement the weather conditions always affect how the battle must be fought. A low level air strike on a clear day would be approached with a completely different attitude than the same air strike at night in the fog. A commander must be able to visualize the actual and possible weather conditions of a battle so that he can better judge his force capabilities and inclement weather limitations. ARPA directed research is creating a computationally expensive weather generation ability along with a DIS extension that will broadcast weather to the using applications. The recreation of the weather information should be done in a manner that is not computationally intensive and should concentrate and user tailorability of the environment.

4.4.2 Design and Implementation

The **Weather_Manager** can logically be thought of as consisting of seven distinct components (see Table 1), the Sun, the Moon, the stars, the sky, clouds, and fog. Each component represents a particular atmospheric or astronomical effect. The overall weather object must be initialized before any weather effects can be seen in the simulation. The initialization is broken up into two parts, one gives the weather module all of its needed visualization information, the other takes that information and initializes all of the individual weather components.

After the weather object is initialized then the individual weather components must be constantly updated. This is accomplished by having a single update method that calls all of the individual update components. Since some of the components deal exclusively with how an image is displayed, they must be called in the draw thread, so the update portion of the weather object is broken in half, one side dedicated to the application thread and the other side dedicated to the draw thread. The algorithm for each thread can be seen in Table 2. The *APP* module first updates the time of day, it next checks for major position changes such as jumping into space, and then updates the sky color. The *APP* thread ends with updating the position of the moon and the displaying of the clouds. The *Draw* thread updates the fog, the sun position, and finally the displaying of the stars.

Table 1: Weather Components

COMPONENT	*DESCRIPTION*
SUN	Moves Sun and light source
MOON	Moves moon
STARS	Displays stars at correct location
SKY	Adjusts color of the sky according to time of day and viewing direction
CLOUD	Draws clouds out in the scene
FOG	Draws fog out in the scene

Table 2. Algorithms for Weather App thread and Weather Draw thread

App	*Draw*
• Update time of day	• Update the fog
• Check to see if pod had a major position change (such as jumping into space)	• Update the sun position
• Update the sky color	• Update the stars
• Update the moon position	
• Update cloud appearance	

4.4.3 Sun

4.4.3.1 Background

The position of the sun is very important in many navigation and tactical situations. An individual can use the sun to gain rough orientation cues, or an Air Force navigator can use it to pinpoint his position over a battlefield without having to rely upon electronic instruments. Tactically, a battlefield commander usually wants the sun at his back, whether he is on the ground or in the air. The visual flare makes it difficult to see objects that are visually in line with the sun. Knowing the sun position will affect many tactical decisions made in small scale and large scale engagements. Many heat seeking missiles are confused when flying along a vector coinciding with the sun and pilots will consciously use this fact when making flight decisions. These consideration lead to the inclusion of the sun in the SBB.

The sun can be represented in more than one manner. A simple circle straight up in the sky suffices for some purposes, others require perfect visual simulation, including the camera flare seen when looking at bright objects. Precise location is required when trying to use the sun's blinding effect in a real-time engagement, but not all of these considerations are relevant to the large scale battlefield commander. The commander may want to know the sun's general location so that the enemy is looking into the sun when the forces are sent to attack, or equally important, so that the commander doesn't order his troops to attack along a vector that has them looking directly at the sun. Camera flare caused by the sun is usually only relevant when generating photorealistic environments and doesn't contribute to the decisions concerning large scale vehicle movements. The tactical effects of sun heat and sun blinding effects are only relevant to individuals who rely on those artifacts to make decisions, such as a pilot engaged in combat. These effects must be accurately recreated for the individual combatant but are not relevant to the commander.

The sun tracks across the sky depending upon the time of day and the viewers latitude. The *SBB* sun is always at its highest point at 1200 hours. The sun rises 0600 hours and set at 1800 hours, these numbers are only relevant to the *SBB* sun and do not correspond to any standard clock. The current sun clock can be changed dynamically while in the simulation and the illumination of the environment changes according to sun position. This allows a commander to either view the battle under full illumination by setting the sun clock to 1200 or he can view the battle under darker conditions by setting the sun clock to later or earlier in the day.

4.4.3.2 Design and Implementation

The time of day and the sun's relative height above the horizon is used to determine the position of the sun in the sky. The time of day tells us how high the sun is in the East-West plane, the Sun then rides along the arc that is formed by the angle of the Sun above the southern horizon which is constant for a particular location on Earth. Figure 8 and Figure 9 illustrate how to the Sun's position in space relates to its position above the horizon. For the purposes of the *SBB* the Sun is assumed to be at its highest point along the arc when the clock is set to 1200 noon.

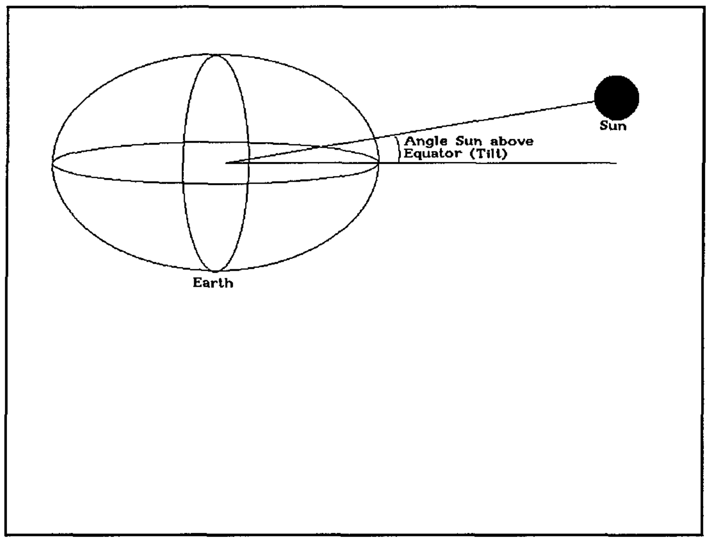


Figure 8: The Sun's Relation to the Equator

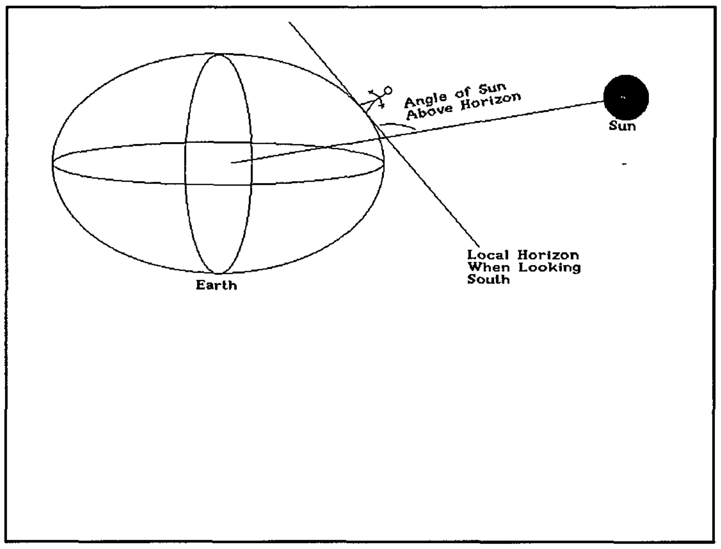


Figure 9: The Sun's Angle Above the Horizon

The color of the Sun is changed by switching out different models that represent the sun in its different phases. When the sun in high in the sky, a yellow sphere is loaded in for the sun, but when the sun is very low on the horizon a dark orange sphere is displayed. It is important to point out that when a different model is used to represent the Sun, or when the Sun is moved across the sky, the scene lighting does not actually change, the lighting is done separately to coincide with where the Sun in drawn.

The position of the light source used to render the scene is adjusted by the **pfLightPos** command. The position of the Sun is passed to this command. The Color of the light source also needs to reflect the position of the Sun. Calls to **pfLightColor** and **pfLightAmbient** are made to make the light source darker and more orange at dusk and dawn, and finally very dark when the Sun is completely below the horizon. Table 3 shows the calls needed to create and update the Sun.

Table 3: Sun Calls

Calls	*Roots*	*Passed Variables*	*Files Needed*
Initialize Sun	Sun Root	pfLight	Bright Sun.flt, Sun at Dawn/Dusk.flt
Update Sun		Time of Day	

4.4.4 Sky Coloring

4.4.4.1 Background

The position of the sun obviously effects the coloring of the sky. The primary importance of coloring the sky is it's qualitative association with sun positioning, if the sky always maintained the same coloration regardless of sun position then any observer would be distracted by the dichotomy between what they are seeing in the sky and how they know sky should really be colored. The sky must be red when looking in line with the setting sun and a darker blue when looking away from the sun. Daytime colorization has a graduation of blues that arc across the day sky. The SBB incorporates these features to create a visually realistic sky that corresponds to sun position and viewer orientation

4.4.4.2 Design and Implementation

The color of the sky is rendered by Performer and is broken into three areas, the horizon, transition zone, and the upper sky. The color of each of these three parts is programmable, so color of the sky can be programmed to reflect the colors seen at various times of the day. Also, since the color of the sky is very orange and bright when looking in the direction of the setting sun, and very blue and dark when looking away from the setting sun, color of the sky must be adjusted to take viewing direction into consideration.

The method of coloring the sky is broken into three phases. During all phases the coloring algorithms picks the color of the horizon and the color in the upper sky and colors the transition zone based on the average of the first two colors. The first phase is when the Sun is high in the sky, during this phase the color of the sky is a washed out blue when looking straight up and a purer blue when looking at the horizon. The colors were matched manually to the real sky. The second phase occurs when the Sun transitions from day to night. During this time the horizon near the Sun goes from blue to red, and then to black, the rest of the sky goes black, the program examines where the user is looking to determine how to color the horizon, red or dark blue. The third phase is when the Sun is below the horizon and the sky goes from dark blue to black , this is also shown in Table 4.

An additional phase of coloring must be added for when the user is in outer space. Since there is no atmosphere in space, the sky should be black no matter what time of day it is. Whether or not the user is in space is checked by a variable that reflects major position changes. This variable is updated every frame by the main weather code and is available to all weather components. Table 5 shows the calls needed to create and update the coloring of the sky.

Table 4: Sky Coloring Phases

Phase	*Horizon Color*	*High Sky Color*	*Where User is Looking*
I-Sun is high in the sky	Blue	Washed out blue	Doesn't Matter
II-Sun is near the horizon	Red	Dark Blue	User is looking at Sun
II-Sun is near the horizon	Dark Blue	Darker Blue	User is looking away from Sun
III-Night	Black	Black	Doesn't Matter
IV-Space	Black	Black	Doesn't Matter

Table 5: Sky Coloring Calls

Calls	*Hooks*	*Passed Variables*
Initialize Sky		pfEarthSky,
Update Sky	Time of Day, Where the User is Looking	Major Position Change

4.4.5 Stars

4.4.5.1 Background

The appearance of stars in the night sky is useful for navigation and provides a visually convincing effect. An Air Force navigator can use celestial positioning to find his location without relying upon electronic devices, a individual on the ground can look up at the night sky and if he sees nothing, he will interpret that as an indication of a cloud layer, since clouds are not readily seen at night. An overcast sky means that the chances of seeing military activity at distant locations is relatively slim. Pilots are less likely to see lights on the ground, and people on the ground are less likely to here the planes above. The commander of a battle space does not need the accurate star representation required by navigators since he can use the computer for his navigation, he will instead use the presence of stars at night as a visual cue for a lack of cloud cover. When planning a night mission, overall visibility is an important issue, the commander needs to know the likelihood of visually acquiring the enemy and the chance of being visually acquired by the enemy, the stars are a natural way of supplementing a user's inherent system of viewing the night environment.

4.4.5.2 Design and Implementation

The stars in the sky are created by using **pfLighPoints** which are each represented as a single pixel on the screen. The star positions are read in from the file *starfield.txt* and then transformed into the *Performer* coordinate system. The actual code to read in the stars and convert the coordinate developed by (Koonse). The intensity of the stars is determined by the time of day. When the Sun enters the transition phase the stars slowly start lighting up, this is done by changing the color of all of the light points. The stars fade at dawn by changing the color value from an opaque representation to a transparent representation. To ensure fast execution, the light points are only changed when the time of day has been changed.

A final consideration for the stars is whether or not the user is within the atmosphere or out in space. When in space all of the above considerations do not apply, the stars should always be rendered using their night time values. This is determined by checking the major position change variable, which indicates whether or not the user has just gone into space or has just come back from space. Table 6 shows the calls needed to instantiate and update the stars in the *SBB*.

Table 6: Star Calls

Calls	Roots	Passed Variables	
Initialize	Star_Root	Max Stars	starfield.txt
Update Stars	Time of Day	Major Position Change	

4.4.6 Moon

4.4.6.1 Background

The presence of the moon is also another important environmental factor. This is especially true during night operations. It is desirable to know how much illumination a night moon might provide. A moon on a clear night will make visual acquisition of targets easier than when the moon is darkened or not visible. This can effect a battle group's vulnerability to visual detection. Similar to the presence of stars, the presence of the moon immediately provides even the casual observer a method of determining evening visibility ranges. When down on the ground, if an individual looks up and sees a clear bright moon then he knows that everybody else will have that same ability to see through the atmosphere. When, on the other hand, an individual looks up and sees the moon through an overcast sky and is only able to make out the moon's fuzzy edges, he knows that nobody can see very far. If a military movement is going to come in on a position a commander can intuitively tell that they will not be able to visually acquire the target until relatively close, under the clouds.

4.4.6.2 Design and Implementation

The SBB employs a moon the tracks across the sky in the same manner as the sun. The height of the moon above the equator and the moon's position relative to the sun are both programmed into the weather software and are controlled in the same manner that the Sun is controlled. Table 7 indicates the calls needed to create and update the moon.

Table 7: Moon Calls

Calls	*Roots*	*Files Needed*
Initialize Moon	Moon Root	Moon.flt
Update Moon	Time of Day Moon Angle Above Horizon	

4.4.7 Clouds

4.4.7.1 Background

The commander wants the ability to see how different cloud covers will effect the events of the battle. The commander can use different cloud configurations to visualize battlefield visibility under different weather conditions. Being able to visualize actual cloud layout provides an intuitive level of realism and capability estimations. The SBB employs a notional set of clouds that can be turned on and off, and the density coverage modified.

4.4.7.2 Design and Implementation

The clouds in the SBB are represented as polygonal models created to roughly represent a typical cloud. A small chunk of nimbus cloud, made of several small spheres, is modeled and all other cloud types are created by this basic building block. In order to make larger clouds from this small piece the programmer must decide how he wants to build the cloud, such as stacking the pieces together to form a column, or spreading them out to form an overcast, and then instantiate each piece giving it its location, orientation, and scale. All calls to **New_Cloud** put the new cloud piece under the single cloud root, this keeps the main program from having to add any of the new pieces into its rendering tree. Figure 10 shows a single cloud building block on the left side of the picture, the larger cloud on the right is creating by making multiply copies of the building block. The cloud models are created with two sided polygons so that when the user enters a cloud the inside of the cloud model obscures the rest of the scene.

Two models of the cloud building block are used, one for the day and one for the night. The night model is a darkened version of the day model. Since the cloud building block is pure white it appears too bright at night, detracting from realism of the scene, a darkened cloud model at night solves this problem. When the cloud object is updated it checks the time of day to determine which model should be used. Also, when updated a cloud hook corresponding to a cloud category is checked to see whether the clouds should be turned on or off, currently this supports nimbus type clouds, other categories could be programmed in by creating a pattern of instantiation to match the desired size and shape of the new category.

An example of how to make an overcast sky of nimbus cloud is presented in the cloud gaggle routine. The cloud gaggle goes through a loop and places a new cloud piece at position over the terrain so that the whole area is roughly covered. Each piece is also given a different orientation and scaling factor. Table 8 shows the calls needed to create clouds. **Initialize Cloud** is first called to set up the clouds, then **Update Cloud** is called every frame, this ensures the proper cloud model is user and that clouds are removed when necessary. A call to **New Cloud** puts a single cloud building block out in the scene with the given position, orientation, and scale. A call to **Make Overcast** calls **New Cloud** several times, placing the building blocks over the scene until a layer of clouds is created.

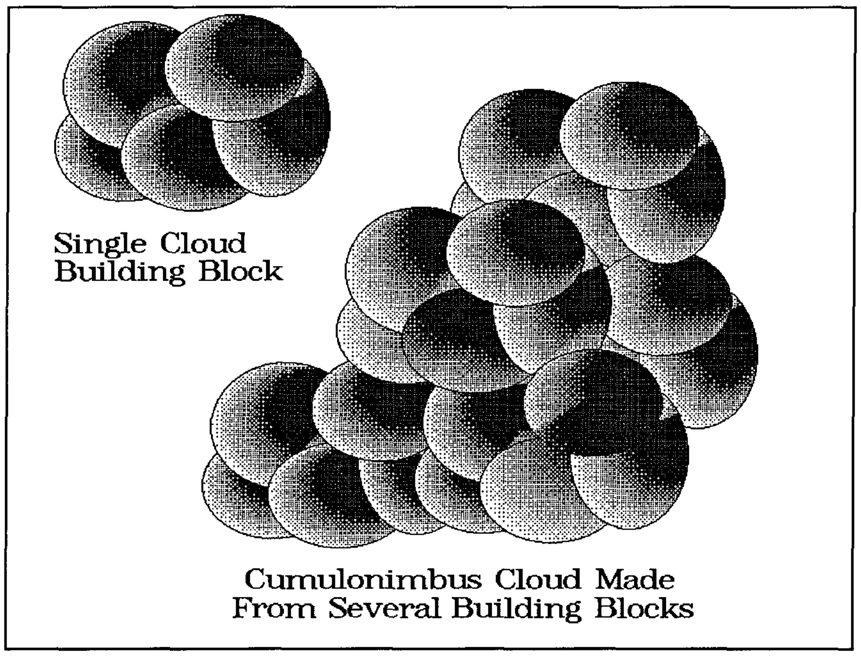

Single Cloud Building Block
Cumulonimbus Cloud Made From Several Building Blocks

Figure 10: Cloud Building Block and Construction

Table 8: Cloud Calls

Calls	*Roots*	*Hooks*	*Passed Variables*	*Files Needed*
Initialize Clouds	Cloud Root			Day Cloud.flt, Night Cloud.flt
New Cloud			Position, Orientation, Scale	
Update Cloud		Show Nimbus Clouds		
Make Overcast				

4.4.8 Fog

4.4.8.1 Background

Fog is the specialized case when a cloud is on the ground. Controlling fog density provides benefits very similar to those attributed to clouds. Since fog can be created in a slightly different manner, taking advantage of the hardware generated fog and fog-like effects, we can compare the visual impact of fog on the commander and contrast that with the impact of clouds. Insights into the strengths and weaknesses of both rendering methods should provide a good starting point for the next generation of atmospheric phenomena. The SBB allows the user to turn fog on and off and then manually adjust the density of the fog.

4.4.8.2 Design and Implementation

The fog object uses the **pfFog** variable to controls fog color and intensity. The intensity is controlled by the global hook **Fog_Density** and the color is always to same as the ambient color of the Sun. The calls needed to create and update the fog are represented in Table 9

Table 9: Fog Calls

Calls	*Hooks*	*Passed Variables*
Initialize Fog		pfFog
Update Fog	Fog Density	

4.5 Movement

When a commander is viewing a battle field , the ability to view the battle from more than one perspective is extremely valuable. In order for the commander to have a complete conceptual understanding of the battle he must understand the big picture and he must also understand the small picture. He will want to view the whole area from a God's eye view of the world, and he will also want to view the battle from close in. This allows the commander to know what is going on the front line as well as getting the overall picture.

4.5.1 Absolute Positioning

The most obvious movement requirement is the need to position the Pod at any particular desired location within the virtual environment. A Commander needs to be able to direct his position so that he may achieve the desired view of the battle. The Pod to provides the user with the means to position himself anywhere within the virtual environment. The movement in the new *SBB* is done in the same manner as the previous *SBB*. Graphical buttons are presented to the user that represent forward/ backward, up/ down, left/ right, and heading/ pitch as described in (Wilson).

4.5.2 Site to Site Movement

4.5.2.1 Background

Once a commander has gone to a particular location, he will likely want to move on to somewhere else, but if he has been to a place once, the chances are high that he will want to return. Instead of manually repositioning himself to the previous location, a means of instantly going to any location of interest would save effort and time. This capability first implemented in (Wilson) but has been implemented in a different manner in the current *SBB*.

4.5.2.2 Design and Implementation

This functionality was re-programmed in order to work with the new user interfaces. The transporter is now implemented by calling an object that stores a list of pre-defined locations, these locations are referred to by their order in the list, so the first location in the list would be site one. The user tells the object which location he wishes to go to and the object returns a variable with the coordinates of that location. The coordinates of the new location are then copied into the variable that represents the Pod's location, moving the Pod to that location.

When the Site to Site object is initialized it creates an array of coordinates and then reads a file of predefined positions and stores them in the array. When the method **Skip_To_Site (int site number)** is called, the input is checked for range errors, then the coordinates stored in that position of the array are copied to the **Current Site Location** variable, and the site number is copied to the **Current Site Number** variable. All variations of **Skip_To_Site** are based on this one method, so the code for **Next_Site** simply call **Skip_To_Site** and passes the current site number incremented by one. Table 10 shows the needed calls to created the **Site_To_Site** capability and the procedures available to extract site information.

Table 10: Site-to-Site Calls

Call	*Passed Variables*	*Returned Variables*	*Files Needed*
Initialize			site_to_site.dat
Get Number of Sites		Haw many sites there are.	
Current Site Number		What stored location is currently selected	
Current Site Location		The coordinates of the current site	
Skip to a Site	New Site Number		
Next Site			
Go Back One Site			
Jump By X Sites	How far forward		
New Site (1)	The coordinates to be stored in the current site		
New Site (2)	The coordinates to be stored, The number of the site in which the coordinates are to be stored		

4.5.3 Vehicle to Vehicle Movement

Since the virtual battle space is often heavily populated with various vehicles, a commander would like to be able to position himself where the vehicles are. He may wish to see an F-15 view of the battle field or get a tank's perspective of the environment. The SBB provides a means of cycling through all vehicles that are represented in the virtual environment. This capability is discussed in further detail in (Wilson).

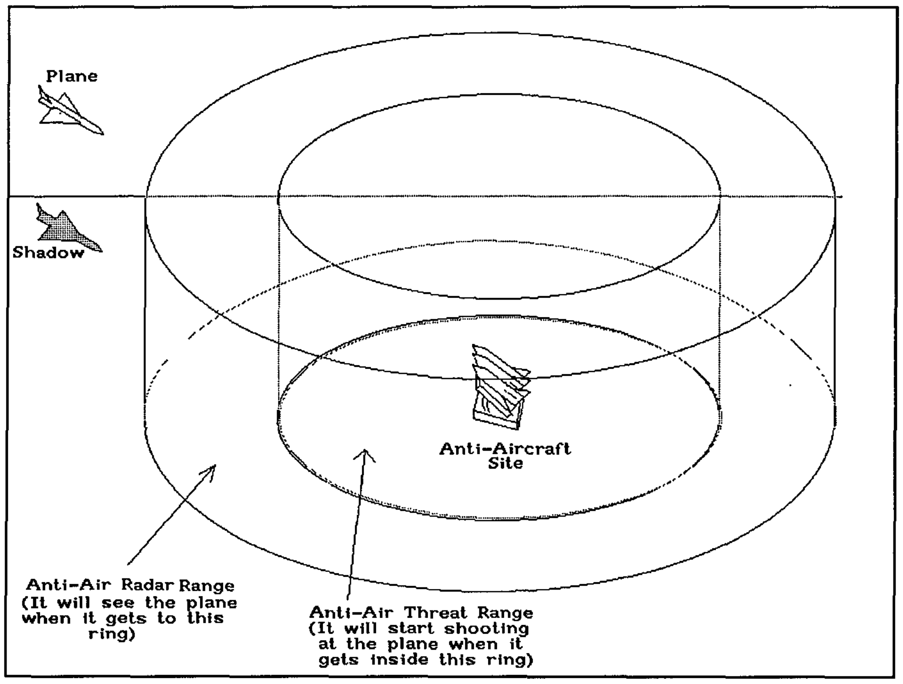
4.6 Situational Awareness

Situational awareness, for the purposes of this thesis, refers to the commander's need to keep track of several aspects of a battle at the same time. Although it will be important to closely watch a particular hill, or keep a close eye on a nearby road, the commander can't afford to loose track of what is going on elsewhere. The commander needs the ability to keep track of the second location without actually taking the time and effort of going there to watch it. Functionality regarding situational awareness makes it easier for a commander to keep track of several things while not diverting his attention from his main point of interest.

4.6.1 Threat Detection

4.6.1.1 Background

The simplistic view of a vehicle's capability corresponds directly to the vehicle's radar emission. When a Mig-31 isn't picking up an F-15 on his radar then the F-15 know that the Mig-31 does not currently pose a threat. Representation of battlefield radar coverage simultaneously illustrates which planes are emitting radar and the extent of coverage provided by that radar. The SBB represents friendly and enemy capabilities by physically drawing the radar cone out into the environment, providing a visual representation of active radar coverage. Radar range is sometimes different than threat range. Anti-aircraft installations can detect enemy aircraft before they can realistically engage that same plane. The range within which a vehicle can successfully engage another vehicle is the threat range, Figure 11 illustrates how the ranges might be displayed.



Plane
Shadow
Anti-Aircraft Site
Anti-Air Radar Range (It will see the plane when it gets to this ring)
Anti-Air Threat Range (It will start shooting at the plane when it gets inside this ring)

Figure 11: Radar Ranges and Threat Ranges

4.6.1.2 *Design and Implementation*

Threat ranges and radar emission visualization are programmed in identical manners. Models of the effects are created in MultiGen, one for threat ranges and another for the radar emissions. These files associate a vehicle with a model that represents the corresponding effort. This is done in the vehicle manager code and utilizes the model manager code developed by (Wilson). When a vehicle's geometric model is loaded from the disk the threat association file and the radar association file are examined and the corresponding geometric files for those effects are loaded and permanently associated with the vehicle. The vehicle manager then puts these effects in *Performer* tree, (see "Figure 12) with **pfSwitches** above each branch so they can be toggled on and off. Object methods are provided to turn the effects on and off. These methods go through each player and assert the associated switch to represent the current state wanted by the user. Table 11 lists the calls available to control the representation of threat ranges and radar emission. Threat and radar visualization is implemented for the Mig-31, F-15, and SA-6 which can serve a basis on which the effectiveness of this method can be evaluated. Threat and radar visualization can then be implemented for all other vehicles.

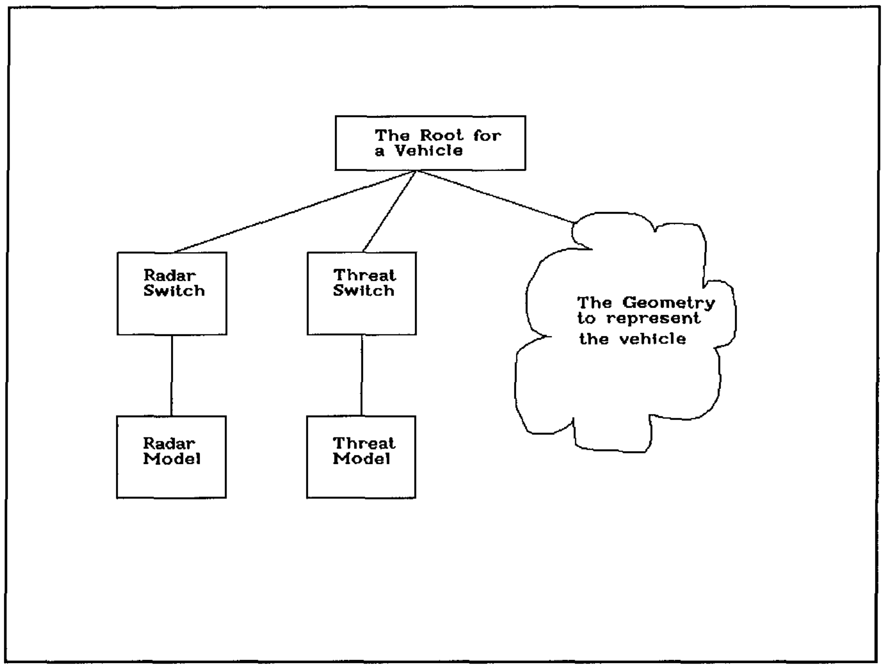


Figure 12: Performer Tree for Threats and Radar

Table 11: Threat and Radar Calls

Calls	*Passed Variables*
See F-15 Radar	On or Off
See F-15 Range info	On or Off
See Mig-31 Radar	On or Off

See Mig-31 Range info	On or Off
See SA-6 Radar	On or Off
See SA-6 Range info	On or Off

4.6.2 Remote Cameras

4.6.2.1 Background

When a commander has a particular hot spot in mind, he can benefit from being able to keep a constant eye on that point. Leaving his current location and going to the second spot forces him to loose track of the first place. A commander riding along the wing of a B-52 might want to know if the airport is safe and not under attack. Instead of constantly jumping to the airport and then back to the B-52, the ability to watch both places at the same time would save effort and time and improve situational awareness. There are several ways to keep track of the activity of a location, one method used successfully in the past is the use of a Sentinel (Soltz). A Sentinel is an entity that keeps track of activity at an area and reports back to the commander. used fuzzy logic to develop a computer program to perform this task. Another method of remote observation is to have another person, possibly using another *SBB*, observe that place of interest and physically or electronically relay the desired information to the first person. A view onto the location of interest would allow the commander to directly watch the sight without diverting attention from his primary point of interest, this would also allow him to gain a first hand understanding of the location and gain other insights not available with the alternative methods. Besides freeing up the other person, he could assess a site's terrain or other visual characteristics that are inappropriate to a Sentinel. Figure 13 shows three camera situated in the terrain, the boxes at the bottom of the figures represent what would be see at those camera positions..

Another issue to address is how this view in into different spots of the world is to be presented to the commander. The view can be presented as a display on the panel of the information pod, with the commander having to look at the screen to see the information, a drawback to this method is that it requires conscious effort by the user to monitor the virtual screen, if he is looking away from the screen then he must constantly divert his attention and physically move his head so that he is again able to see the display. A second alternative would be to have the screen move around with the commander, so if the commander looks behind him then the screen will be in front of his nose, and not at his back. This provides the same effect as tying a display device right onto the user's head, so that no matter where you look, that display goes along with you. In order for the screen to avoid obscuring the commander view out into the environment of primary interest, it should not be too large, yet still big enough to recognize terrain and vehicles.

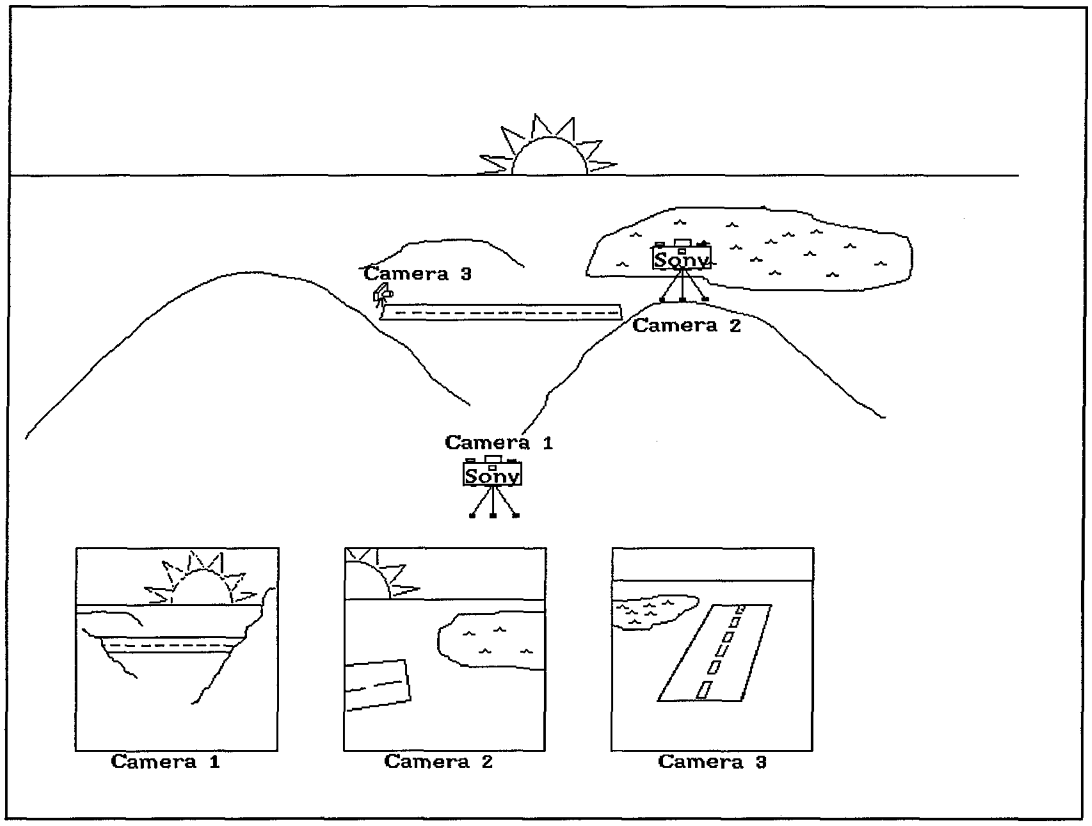


Figure 13: Remote Camera

4.6.2.2 Design and Implementation

The drop camera can provide the user with a view to any other location, this is done using the same object as used by the **Site_To_Site** code but instead of bring the whole pod to a designated location, the camera goes to that location. The **Camera** objects creates an instance of **Site_To_Site** and reuses that functionality. The actual view generated by the remote camera is created by using a **pfView** and by specifying the window location and dimensions to correspond to the position of the viewport. The coordinates of the camera positions are copied to **pfView**, so now this window on the screen is showing what would be seen at the camera site.

4.6.3 Video Missile

4.6.3.1 Background

A variation of the remote camera gives the commander the ability to go search out locations of interest. Finding locations of interest can be accomplished in at least three ways. The first way is to use a human, using another *SBB,* to move around the battlefield until he finds a new location of interest and then relay that information to the commander, who would then go to that location. One draw back to this method is its reliance upon using another person. A second method could use computer Sentinels to not only watch locations of interest, but to also seek out new locations. Since the Sentinel is not currently integrated into the *SBB* this approach was postponed. A third method launches a video missile ahead of the user and relies upon the user's analysis of the image. A major advantage to this approach is that it allows the commander to explore an area of possible interest without leaving his current location. The implementation of the missile can also reuse code the remote camera, making it quick to implement. The video missile can serve as a reference to the effectiveness of a wandering Sentinel or wandering *SBB.* The *SBB* provides the user the means to launch a virtual missile from his present location. This missile has a camera attached to its nose so that the user can see through the missile in the same manner as the remote camera. This gives the user a means of reconnaissance without leaving his present location. Figure 14 shows a missile flying away from the POD and going straight until it reaches the airfield, at which point the commander order the missile to circle the field. The box at the bottom of the figure shows the view from the missile, which is also seen by the commander.

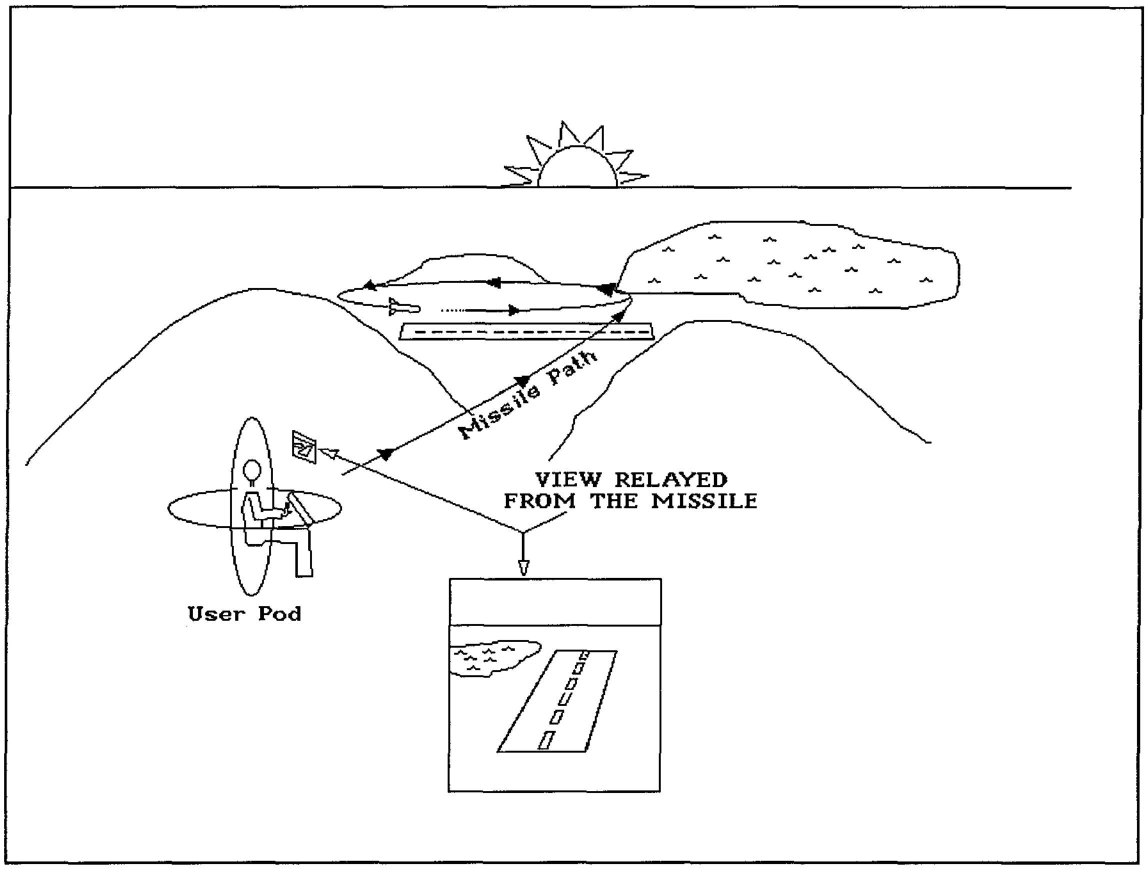


Figure 14: Video Missile

4.6.3.2 Design and Implementation

The video missile serves a similar purpose as the remote camera, the ability to watch something far away. The missile, like the drop camera, acts like a HUD and can only be seen as a window on the overlay plane. This is because the view from the missile is rendered in the same fashion as the view associated with the remote camera. The illusion of a missile is accomplished by giving a *Performer* view a starting position the same as the Pod. If the user chooses a straight launch pattern then the position of the view simply moves forward in a straight line each frame. The flight path can also be circular, which allows the missile to loiter around a spot of interest for an indefinite amount of time. The *ObjectSim* function **Move_Along_Heading** is used to achieve these effects, for a straight line, the heading doesn't change, for a circle, the heading is incremented every frame.

4.6.4 Radar

4.6.4.1 Background

A commander may wish a non-first-person view of the battle space. This can allow him to view the battle in a more abstract manner than direct observation. The concept behind using a radar screen in a virtual environment is covered in more detail in (Wilson). On the SBB radar scope the aircraft are represented by small icons, as are ground vehicles and missiles (see Figure 15 and Figure 16). All of the icons are color coded to represent either friend, foe, or neutral.

The commander examining a battlefield via the radar scope needs the ability to adjust the range of inspection. Another need of the commander is the ability to filter out unwanted radar information. The SBB radar scope provides an interface to dynamically adjust the magnification or range on the scope, also the commander can filter some of the information that can be displayed. The radar can filter what force type is displayed, such as only viewing enemy forces on the scope. The radar can also filter force domain which allows the user to categorically view either air or ground vehicles. This filtering and range manipulation adds enhancement to the radar.

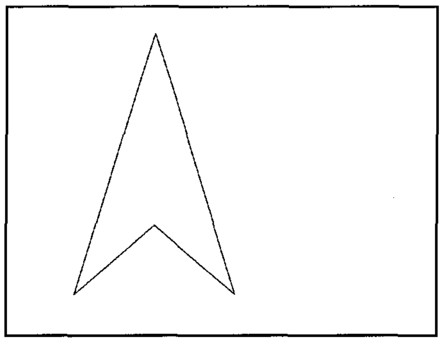


Figure 15: Plane Blip

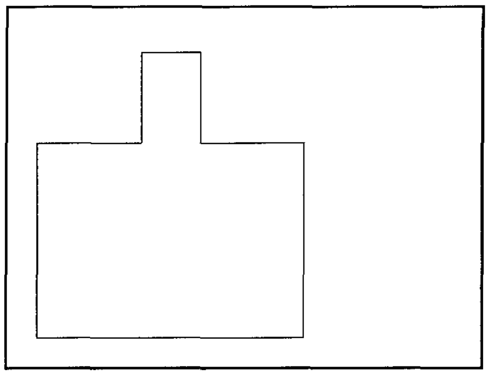


Figure 16. Ground Blip

4.6.4.2 Design and Implementation

The radar scope represents the position of other vehicles relative to the user. Since the scope deals extensively with vehicle location and identification the method for drawing the scope is put within the vehicle manager hierarchy. The actually drawing of the grid is accomplished using GL calls and can be thought of as a three step process. The first two steps are executed by the **Vehicle_Manager** and the third step is accomplished by the user interface section of the code.

The first step represents the lowest level of abstraction, the drawing of a single radar blip. This procedure is given information about a vehicle's parameters that indicate what type of vehicles should be displayed and how they should be drawn. Vehicle characteristics, such as friendly status and domain, are examined and compared to the passed parameters. If it is determined that the calling code wants this type of vehicle displayed then a icon of the vehicle is drawn in GL. If the vehicle is a ground vehicle then an icon for a tank is drawn, otherwise an icon for an airplane is drawn. The icons can either be drawn as solids or as wireframe drawings. The characteristics of the vehicle are carried within the **SBB_Net_Player** structure are discussed in (Wilson). Table 12 shows the call to draw the radar blip and the parameters that need to be passed.

Table 12: Radar Blip Calls

Calls	*Passed Variables*
Draw_Radar_Blip	Vehicle, Show Friendly Flag, Show Enemy Flag, Show Air Flag, Show Ground Flag, Fill Polygons Flag

The next step of the radar encompasses the drawing of the whole radar scope. This code examines every vehicle in the battle and compares their position to the user's position. Then using the range and the screen size it determines whether the vehicle is on the screen or off the screen. The screen is assumed to be square and the ranges are based on the center of the aircraft. If the vehicle is on the screen then a call to **Draw Radar Blip** is made and all visibility flags passed to the **Draw_Scope** code are passed to **Draw_Radar_Blip**. This effectively draws a radar scope.

The last step puts the scope where it is supposed to be. The above code is not aware whether it is drawing itself somewhere on the screen or out in the environment, this allows the user interface designer to specify exactly where the scope will be drawn without changing the underlying code. This portion of the code must push the matrix stack so that the drawing commences where it wants the scope. There are two different ways of viewing the radar scope, one way is to see it on the lower panel in the Pod, the other way is to draw the radar so that it acts as a HUD, always on the overlay plane. When the radar scope is on the lower panel, the viewing stack is pushed and then translated to the panel. A black background is drawn with a GL box command and then the **Draw Radar Scope** command is called. The flagged parameters are determined by user input and then passed to the **Draw Radar Scope** method. When the method returns to the main, the viewing stack is popped to return the viewing matrix to its original state. Table 13 shows the call to Draw Radar Scope and the parameters the need to be passed to it. Table 14 shows the complete algorithm for drawing a radar scope.

Table 13: Draw Scope Calls

Calls	*Passed Variables*
Draw Radar Scope	Range, Screen Size, Show Friendly Flag, Show Enemy Flag, Show Air Flag, Show Ground Flag, Fill Polygons Flag

Table 14: Algorithm for Drawing Radar

```
Get the users viewing parameters (See friendly, See ground, Range, etc.)
    Push the viewing matrix
    Translate to where the scope is to drawn
    Draw a black background
    Call Draw Radar Scope
        For each vehicle in the battle
            Calculate the vehicles position relative to the user
            If the vehicle is within range of the radar
                Push the viewing matrix
                Translate to where that vehicle is relative to the user
                Call Draw Radar Blip
                    If the Vehicle should be drawn (According to
                    the passed parameters, i.e. don't draw a tank if the
                    user doesn't want to see ground vehicles)
                        Check the vehicle's allegiance -
                        (if friendly then draw blue, enemy draw
                         red, natural then draw white)
                        Check vehicle's domain -
                        (if an aircraft then draw a plane,
                         if its a ground vehicle then draw a box,
                        if its a spacecraft then don't draw it at all)
                Pop the viewing matrix (Go back to scope's origin)
    Pop the viewing matrix (Go back to user interface origin)
```

4.6.5 Space Scope

4.6.5.1 Background

On the modern battle field, space based assets are playing ever increasingly important roles. In Desert Storm, the periodic nature of a particular GPS satellite dictated the timing of at navigation sensitive missions (Class Notes). Also, the detection of SCUD launches relied at least in part on space based surveillance. A battlefield commander needs to quickly ascertain available resources, and space based resources should be included. He also wants to know the space based capabilities of his enemy. There are several ways in which this information can be presented to the commander. One way is to simply write out to the screen what he has available to him and what the enemy has available to them. A significant drawback to this method is its inconvenience in the sense that a user cannot quickly conceptualize what is being presented to him, he must read each line and then interpret the information, all of which takes time and is always error prone when in a virtual environment. Another alternative is to realistically render the satellites so that the commander can look out and view what is available to him and ascertain their relative positions over the battlespace. Although the SBB is meant to represent large scale information, the distances associated with satellites is roughly an order of magnitude larger than those associated with a battlefield. This makes it difficult for the commander to conceptualize both frames of reference. In addition to this, there is no intuitive translation between space based motion and battlefield motion. To elaborate, satellites travel in elliptical orbits and their movements are nonlinear, and if an individual thinks about them in a linear manner then they will become confused. Battlefield movement is all linear, for example, people don't worry that the surface of the Earth is curved when driving down the road, they pretend the world is flat and that works fine when not traveling long distances. The extra work associated with presenting satellites in their real positions rules against accurate rendering satellites as a primary method of presenting space information to the battlefield commander.

A third option is to aggregate the available satellite information and present the result to the commander in some format that can be quickly conceptualized by the commander. Laying the satellite resources out on a screen and iconizing the satellite capabilities would allow the commander to quickly glance at the screen and see the pictures of the satellites and immediately determine force type and relative position. The graphical nature of the display facilitates immediate comprehension of available resources and the 2-dimensional display avoids any reference frame transformations by the commander. The previous method of realistically recreating the space environment was implemented for comparison with the space scope.

Like the radar scope, the space scope should also be customizable, allowing the commander to focus only on the information that is relevant to his particular situation. For example, a commander might only be interested in US satellites resources and therefore would wish to turn off any outputs that relate enemy satellite information.

4.6.5.2 Design and Implementation of Space Environment

In order to model space based assets and high altitude observations the inclusion of the space environment is incorporated into the *SBB*. The space environment, which is populated with satellites, provides an environment for commander to visually inspect space based resources such as intelligence and navigation satellites. Vehicles in space generally have velocity and altitudes an order of magnitude higher than battlefield confined vehicles. This distinction between space and Earth vehicles requires extra consideration. The incorporation of the space environment into a simulation that also incorporates a ground based environment can be approached in three ways. The first approach is to lay aside the differences in scale between environments and insert all space based geometry into the normal battlefield environment. This method fails in practice because of the limitations of SGI hardware and lag associated with *ObjectSim* dead-reckoning algorithms. The effect is of the satellite displaying exaggerated jitter to the extent that the view is non-functional. The second approach creates two completely different descriptions of the world, one which includes only the battlefield confined vehicles and another which includes only the space based vehicles, each with their own scale. This is analogous to techniques used to solve interplanetary trajectories where the focus shifts from a planet based scale to a solar system based scale, except that instead the focus shifts from a battlefield scale to a planetary scale. This method does not readily integrate into the *SBB* architecture and was not implemented for this reason. The third approach partitions the scene into two distinct geographical sections, one section being the area near the battlefield, the other section being out in space. The section near the battlespace is stored and manipulated at one scale while the section out in space is stored and manipulated at another scale. This method is in the *SBB* because it achieves a space environment without changing the scaling of the current implementation of the *SBB*. The extent of the battlefield measures approximately 300,000m on a side, the area beyond the 300,000m boundary is compressed 1000 times. Figure 15: Partitioning shows how the Earth is modeled as being around the terrain with the scale on the inside of the planet, near the battlefield, as being scaled 1:1, the scale outside the battlefield is scaled 1,000:1. The algorithm for inserting the vehicles into the environment now follows this algorithm:

```
• For each vehicle
•     If the vehicle is confined to the battlespace then
•             insert the vehicle into the scene, don't scale
•     Else // (The vehicle is in space)
•             Scale the vehicle's coordinates by 1/1000
•             insert the vehicle into the scene
```

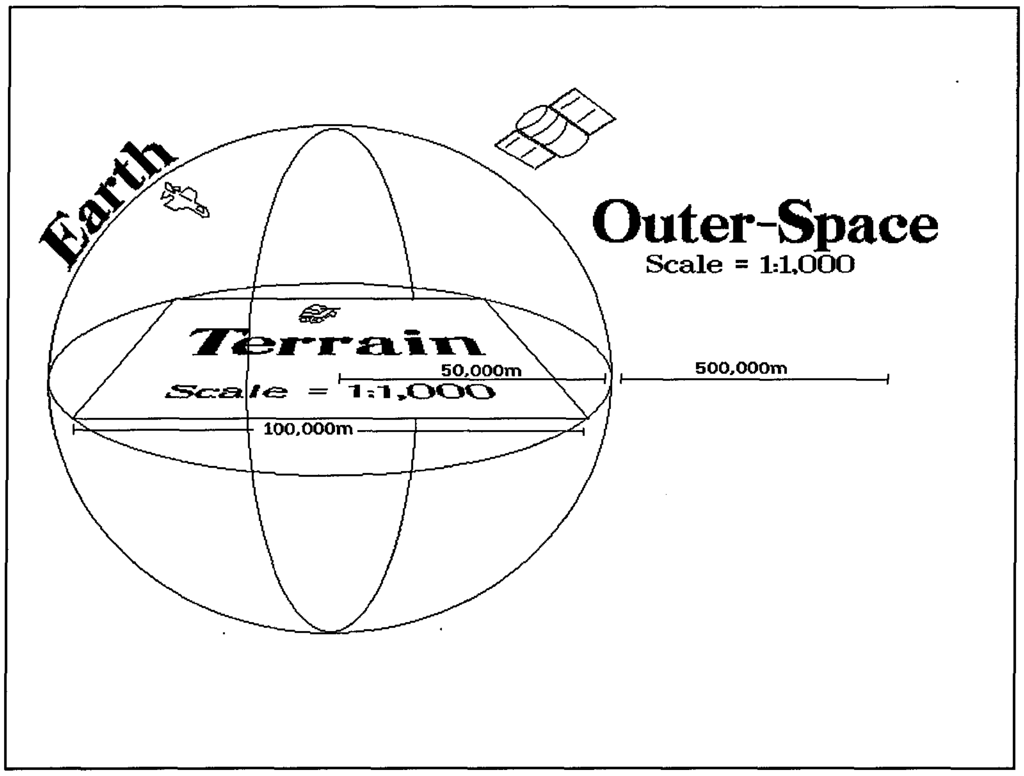


Figure 15: Partitioning of Terrain Between Scales

4.6.5.3 Design and Implementation of Space Scope

The space scope radar barrows heavily from the radar scope. The radar discussion applies perfectly to the space scope except for one aspect of the algorithm. The radar algorithm, (see Table 14) needs to replace the portion where it identifies the vehicles as either ground or air and instead say that it only draws those vehicles that are space objects. A single representation for all satellite types is used, a small icon of a generic satellite is drawn using GL and colored according to force type (see Figure 17).

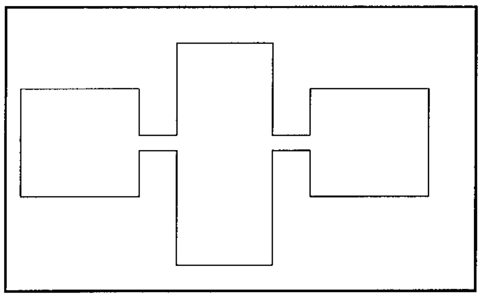


Figure 17: Satellite Blip

4.6.6 Pause

4.6.6.1 Background

The battlefield is an active and rapidly changing environment. The commander may wish to stop all of the action and force the normally dynamic environment into a lifeless static environment that could be explored and examined at leisure. A function that freezes the updating of vehicle position and orientation in the environment would facilitate this need. This would allow the commander to roam the battlefield at his own pace and observe a particular instant in time. One obvious drawback is that even though the commander's view of the battle is stopped, the real fighting is still continuing. Although this is a potential problem for a single commander in a real-time scenario, it could be a useful tools when other *SBB*s are being utilized at the same time or when the battle is only a re-creation of an original conflict that can be paused without consequence.

4.6.6.2 Design and Implementation

The pausing of the battlefield is controlled by a global **Pause** hook. When this hook is set to 1 the **Vehicle_Manager** stops update the network and stops updating vehicle positions. All other *SBB* code is updated.

4.7 Performance

4.7.1 Level of Detail/ Transparency

Whenever semi-transparent materials are rendered on the Silicon Graphics computer, the frame rate suffers considerably, especially when the transparencies occupy a large percentage of the screen. In order to minimize this problem a general rule is adopted that avoids the use of semi-transparent materials unless they have a significant impact upon the display of information. Currently, the biggest user of semi-transparent material is the locator. The locator had been semi-transparent so that when a vehicle got close to the commander he could still see the vehicle inside.

The locators are now opaque and implemented as levels of detail of the original aircraft. When the vehicle manager creates the performer tree for the vehicles, it puts a level of detail node above each vehicle model (see *Figure 18*). The other child of this node is the locator associated with the vehicle (Wilson). The range is set so the vehicle will no longer be rendered when the craft is too distant to see. This switching of the levels of detail is then automatically executed by *Performer*.

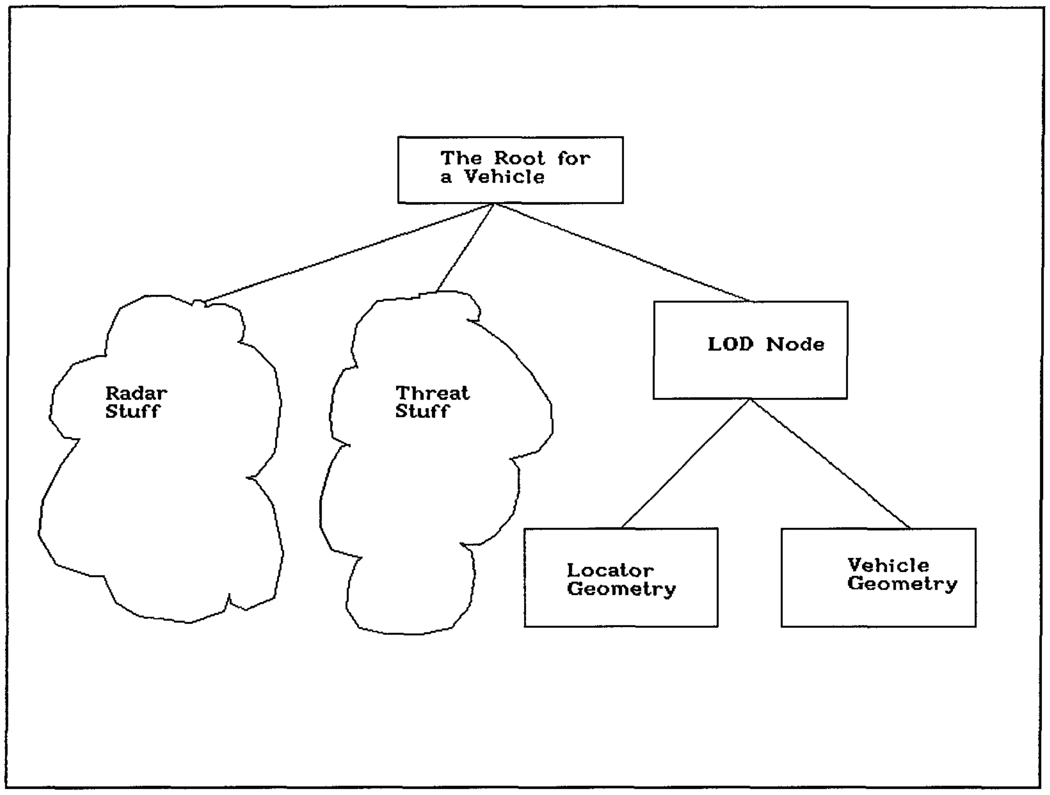


Figure 18: Level of Detail Tree

4.8 Hardware

4.8.1 Computer

Silicon Graphics Inc. (SGI) computers were used to test and develop the *SBB*. An SGI Reality Engine2 Onyx with four 100Mhz MIPS processors, Z-buffered graphics, 4MB texture RAM, and 128 MB of processor RAM served as the test platform for the *SBB*. The *SBB* has also been run on slower machines, a two processor Reality Engine2 Onyx is the lowest acceptable configuration.

4.8.2 Headtracking

Since the SBB is now designed for an immersive environment, the application must have some method of tracking where the user is looking. When the application knows where the user is looking it can then render the new view into the environment.

The Polhemus headtracking device is used for the purpose, it provides head position information over the RS232 line. A routine was written to parse the input and convert it into a form usable by *ObjectSim*. The packets sent to the serial port are retrieved and broken into components representing position and orientation. A coordinate transformation converts the user's head position from room coordinates to Pod coordinates. This is discussed in greater detail in (Vanderburgh).

4.8.3 Display Devices

Three different display devices were used during the development of the SBB. The first device was an extremely low resolution, about 200 x 200, head mounted display. The second device was the PT-O1 head mounted device. This device is a relatively low cost, about a thousand dollars, device that is also comfortable to wear. The PT-O1 has a narrow field of view and a resolution comparable to a poor quality TV set. The SBB interface is geared primarily to a head mounted display that is comparable to the PT-O1. At this resolution, the text is large enough to be legible. The highest quality head mounted display used was the N-Vision. This provided resolution comparable to the computer CRT, the $50,000 unit is somewhat heavy to wear and takes about five minutes to install. Display devices used for the *SBB* are discussed in greater detail in (Kestermann).

4.9 Conclusion

This chapter presents some of the most important features of the Synthetic BattleBridge. Some meaningful components of the SBB such as movement, situation awareness, and weather have been discussed as they relate to the battle field commander and how they were implemented in the *SBB*. The old *SBB* was reprogrammed to work with an immersive user interface. The next chapter presents the results of the enhancements and the thesis concludes with recommendations for future work.

5. Results

5.1 Introduction

AFIT's Synthetic BattleBridge is a functional prototype that can provide help to a DIS battlefield commander. An unintrusive system has been implemented that improves the commander's ability to elicit knowledge of the battle being fought over the network. This chapter follows by addressing each significant new SBB feature with an accompanying figure to help illustrate its final version as it appears to the user. A discussion about new performance enhancements will follow the photographs and a conclusion is at the end of the chapter.

Table 15. Synthetic Battlebridge Capabilities - Past and Present

Capability	*previous SBB*	*current SBB*
Performance	3-30 fps	5-30 fps
Viewing Devices		
CRT	X	X
Boom	X	
Head Mounted Displays		X
Radar	X	X
Variable Range	2 ranges	infinite
View only Friend/ Foe		X
View only Air/ Ground		X
Click on icon - attach to player	X	
Space Scope		X
Site -to-Site transport	X	X
Store sites in a file		X
Threat Ranges & Radar	92' version by Hadix	X
Attach to Vehicle	X	X
Detach from Vehicle		X
Move around vehicles	Heading change gives different view of the vehicle	Movement is relative to the vehicle
Display Number of Vehicles	X	X
Display Vehicle Speed & Direction	X	X
Weather		X
Clouds		Nimbus
Fog		X
Sun		Manual movement
Moon		Manual movement
Stars		X
Sky coloring		X
Remote Camera		X
Head Tracking		X
Video Missile		X
Head Tracking		X
Fuzzy Logic Sentinel	X	
Plan View (Overhead view)	X	Must maneuver into position
Pause		X
HUD	Everything	Remote Camera,

		Video Missile, Radar, Pod Position

5.2 Hardware

5.2.1 Computer

Silicon Graphics Inc. (SGI) computers were used to test and develop the *SBB*. An SGI Reality Engine2 Onyx with four 100Mhz MIPS processors, Z-buffered graphics, 4MB texture RAM, and 128 MB of processor RAM served as the test platform for the *SBB*. The *SBB* has also been run on slower machines, a two processor Reality Engine2 Onyx is the lowest acceptable configuration.

5.2.2 Headtracking

Since the SBB is now designed for an immersive environment, the application must have some method of tracking where the user is looking. When the application knows where the user is looking it can then render the new view into the environment. The Polhemus headtracking device is used for the purpose, it provides head position information over the RS232 line. A routine was written to parse the input and convert it into a form usable by *ObjectSim*. The packets sent to the serial port are retrieved and broken into components representing position and orientation. A coordinate transformation converts the user's head position from room coordinates to Pod coordinates. This is discussed in greater detail in (Vanderburgh).

5.2.3 Display Devices

Three different display devices were used during the development of the SBB. The first device was an extremely low resolution, about 200 x 200, head mounted display. The second device was the PT-O1 head mounted device. This device is a relatively low cost, about a thousand dollars, device that is also comfortable to wear. The PT-O1 has a narrow field of view and a resolution comparable to a poor quality TV set. The SBB interface is geared primarily to a head mounted display that is comparable to the PT-O1. At this resolution, the text is large enough to be legible. The highest quality head mounted display used was the N-Vision. This provided resolution comparable to the computer CRT, the $50,000 unit is somewhat heavy to wear and takes about five minutes to install. Display devices used for the *SBB* are discussed in greater detail in (Kestermann).

5.3 Weather

5.3.1 Sun

Figure 19 shows the sun near the horizon. The picture shows how the yellow model for the Sun has been switched out for a darker version.

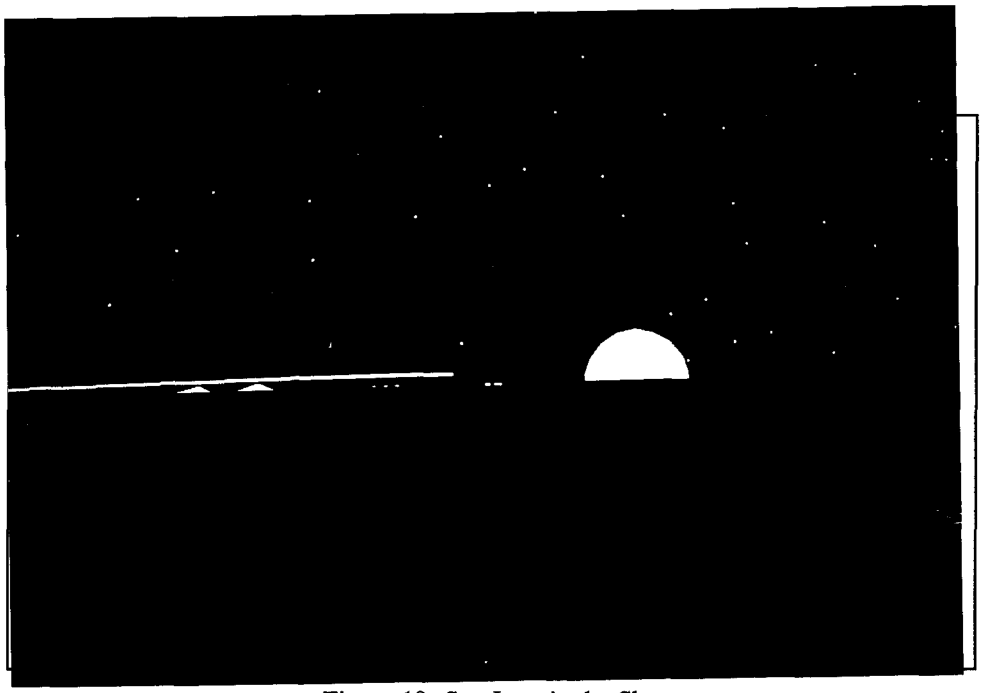


Figure 19: Sun Low in the Sky

5.3.2 Moon

The moon's orbit is represented by a notional path across the sky, similar to the Sun's orbit. The sphere of the moon is in a position relative to the Sun so that a person on the battle field only sees the ambient lit sphere, creating the impression that the moon is a circle instead of a sphere.

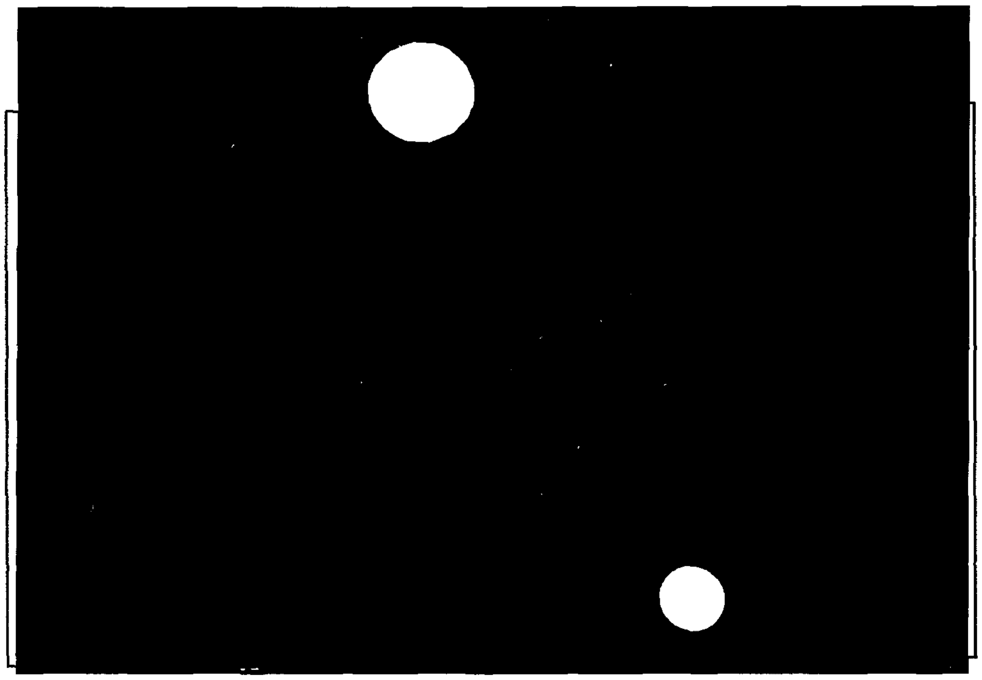


Figure 20: Moon

5.3.3 *Sky Coloring*

The coloring of the sky corresponds to the sun position and view direction. The actual colors provide a rough approximation of the real sky but the match is not exact. Figure 19 shows the graduation of blues going from the horizon on up to the top of the screen, also note the bright color of the sun which will contrast with the Figure 21 where the view is looking away from the Sun.

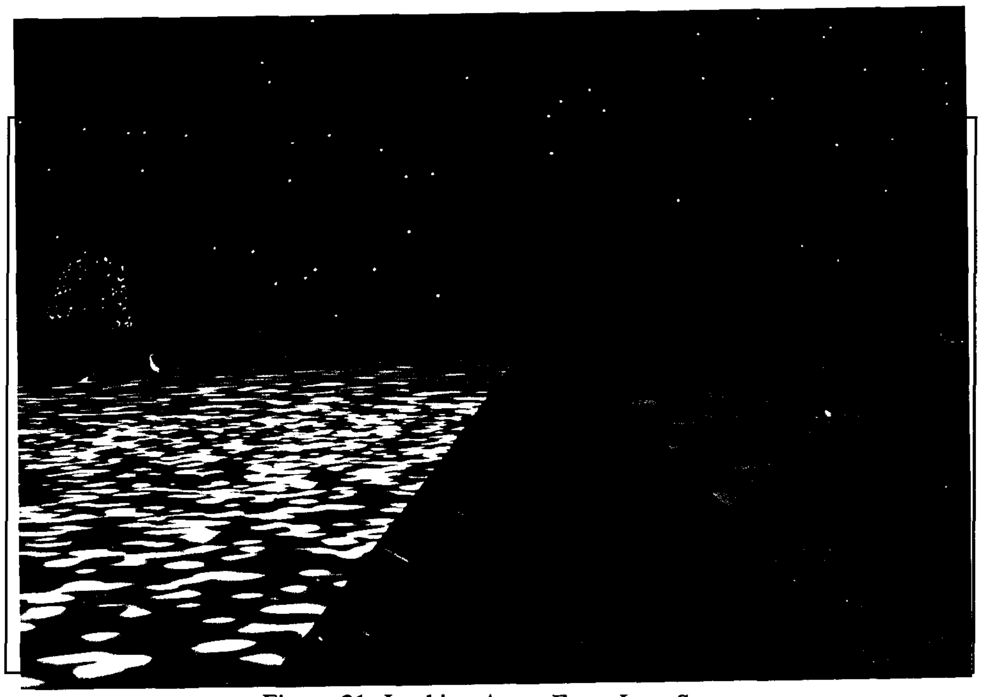


Figure 21: Looking Away From Low Sun

5.3.4 Stars

The stars in the SBB slowly come out at night and provide a realistic simulation of the night sky. The star positions are accurate relative to each other. Figure 22 shows the stars at night.

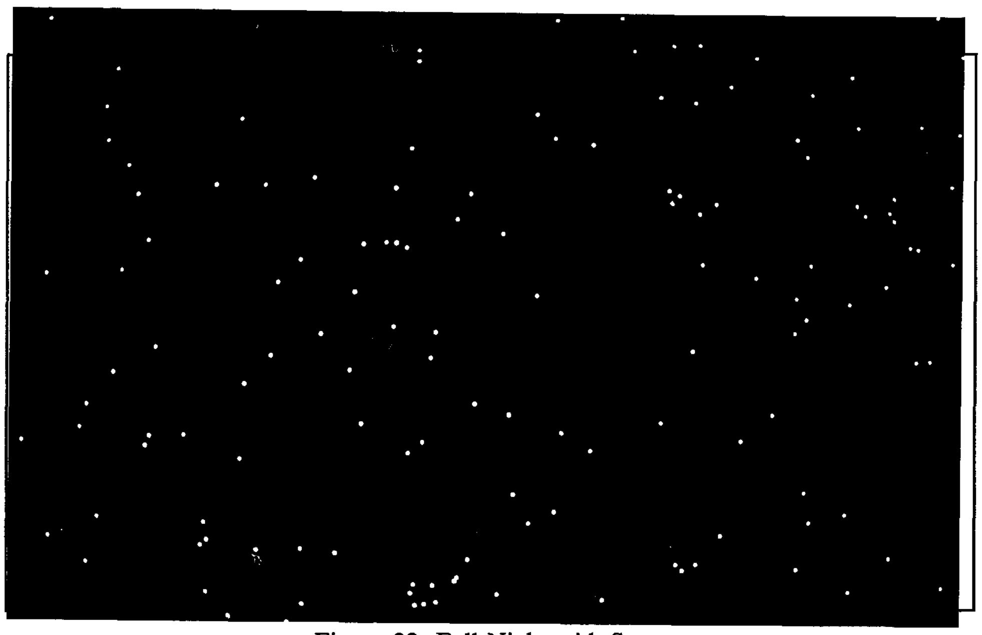


Figure 22: Full Night with Stars

5.3.5 Clouds

The clouds in the SBB represent what can be done in a real time environment. The clouds created can represent nimbus type clouds and are convincing during the day and night. The figure of night time clouds shows how the clouds effectively obscure the stars. The polygon count for an individual cloud element is in the hundreds and the count for an entire overcast uses several thousand polygons, noticeably slowing down the frame rate. One possible alternative is to render the clouds using only square building blocks, or even simple squares that pivot around a point and always face the viewing. The construction of a complex clouds would become more difficult but the new method might allow the denser clouds without compromising performance. Polygon count will become even more important when complex cloud structures are created.

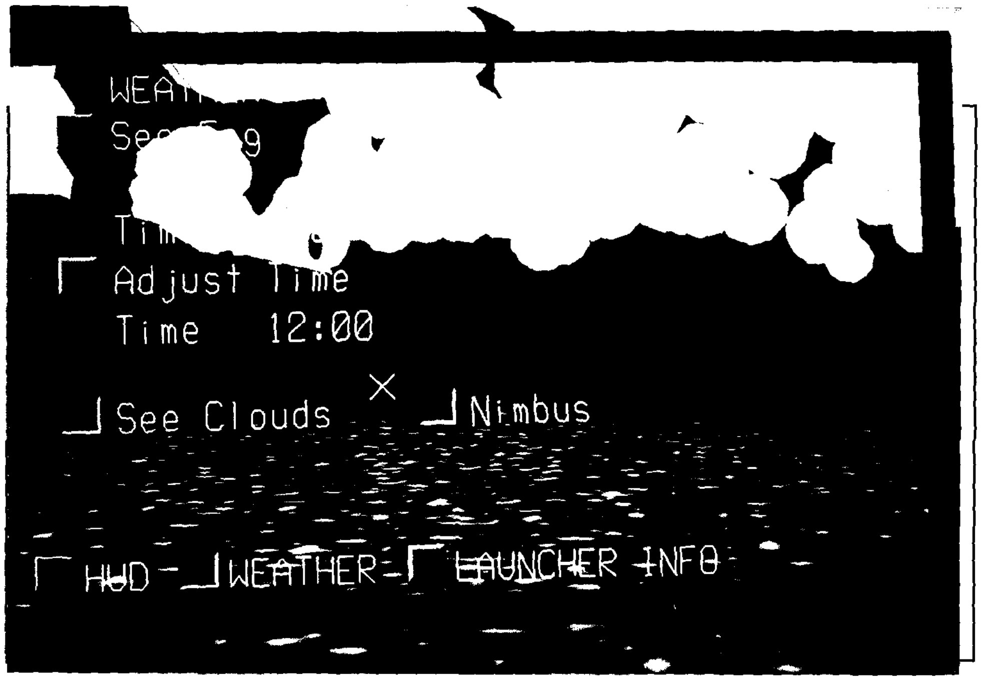


Figure 23: Clouds During the Day (Using Light Cloud Model)

Figure 24: Clouds at Night (Using Dark Cloud Model)

5.3.6 Fog

The SBB fog was created using built-in hardware routines supporting fog effects. The density of the fog is adjusted dynamically by the user. The hardware fog assumes that the viewer is inside the fog bank, which keeps us from rendering any clouds using this technique. Even though clouds and fogs are essentially the same thing in nature, the hardware's ability to assist in fog creation dramatically affects the visual quality of the scene.

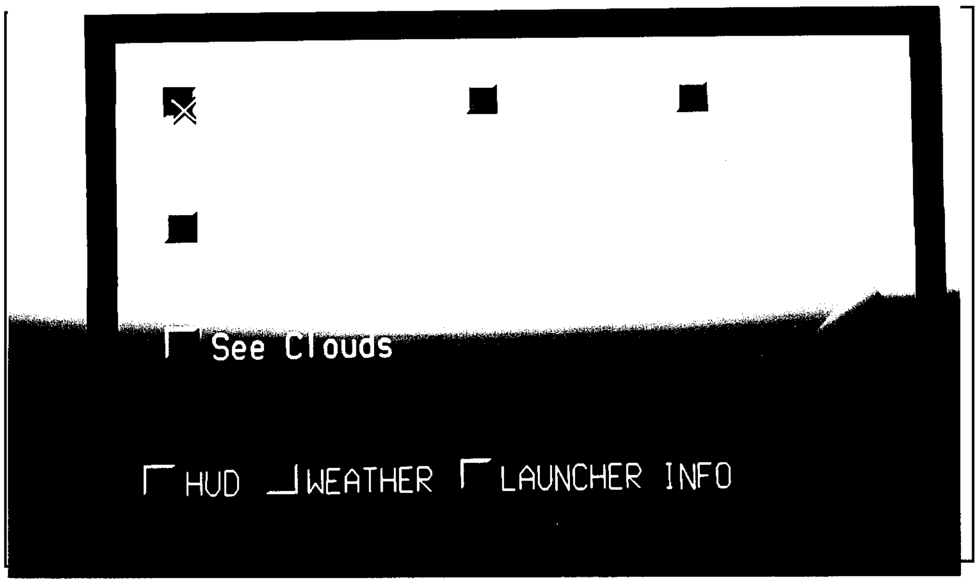


Figure 25: Light Fog

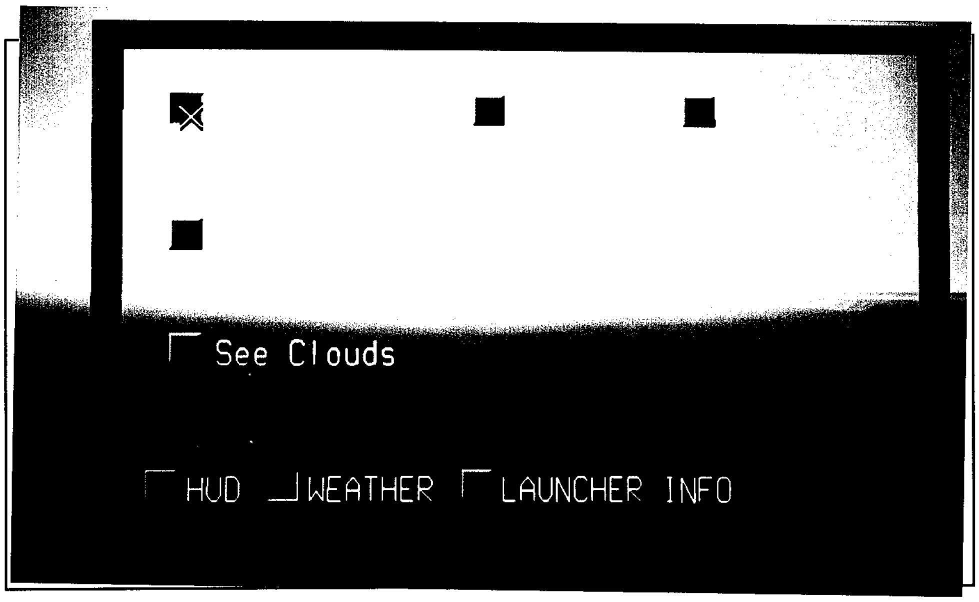


Figure 26: Heavy Fog

5.4 Situational Awareness

5.4.1 Radar

The radar scope is visually very different from the previous implementation. The radar on the lower panel is controlled by buttons next to the scope. The radar that appears in the overlay plane in controlled by buttons on the side panel. The overlay radar always stays in the same spot on the screen, the controls do not follow the view.

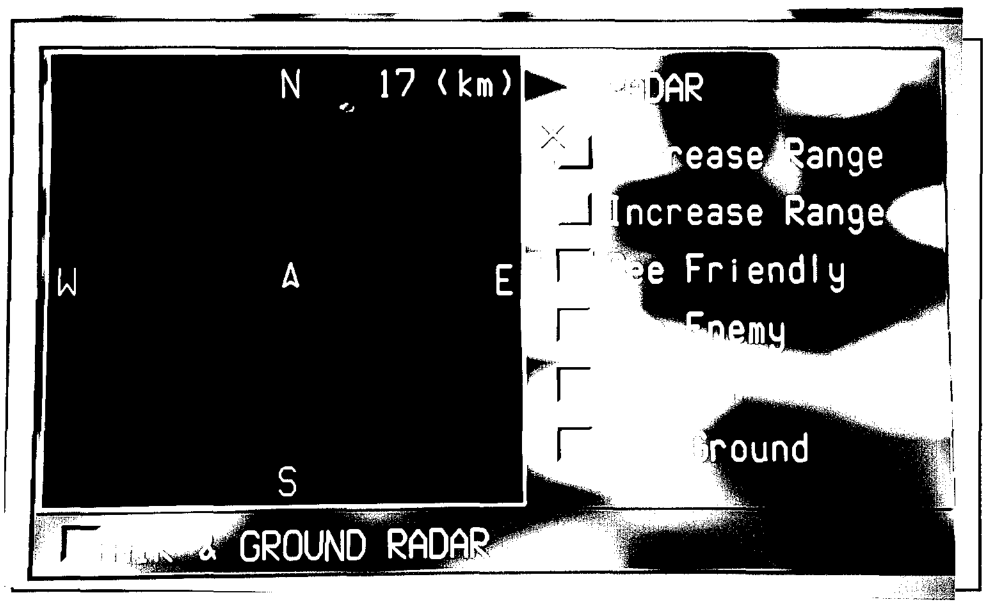


Figure 27: Radar Scope (On Lower Panel)

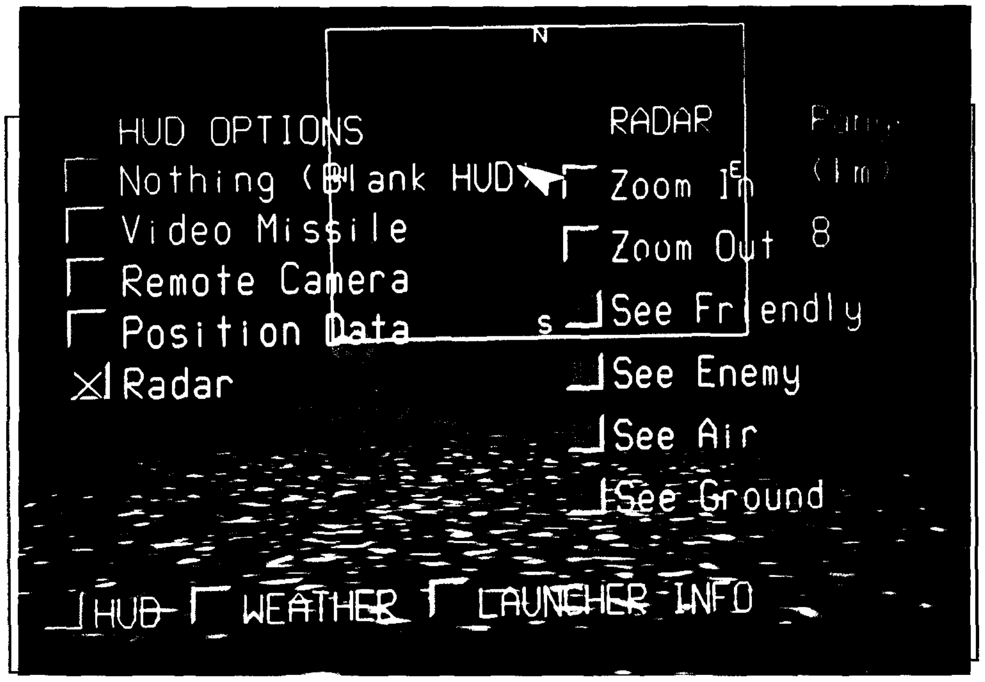


Figure 28: Radar Scope (As a HUD in the Overlay Plane)

5.4.2 Space Scope

The space scope is positioned on a panel slightly above and in front of the user. The controls are positioned next to the scope. The controls to filter satellite types are disabled because these identifiers are not yet part of the DIS standard.

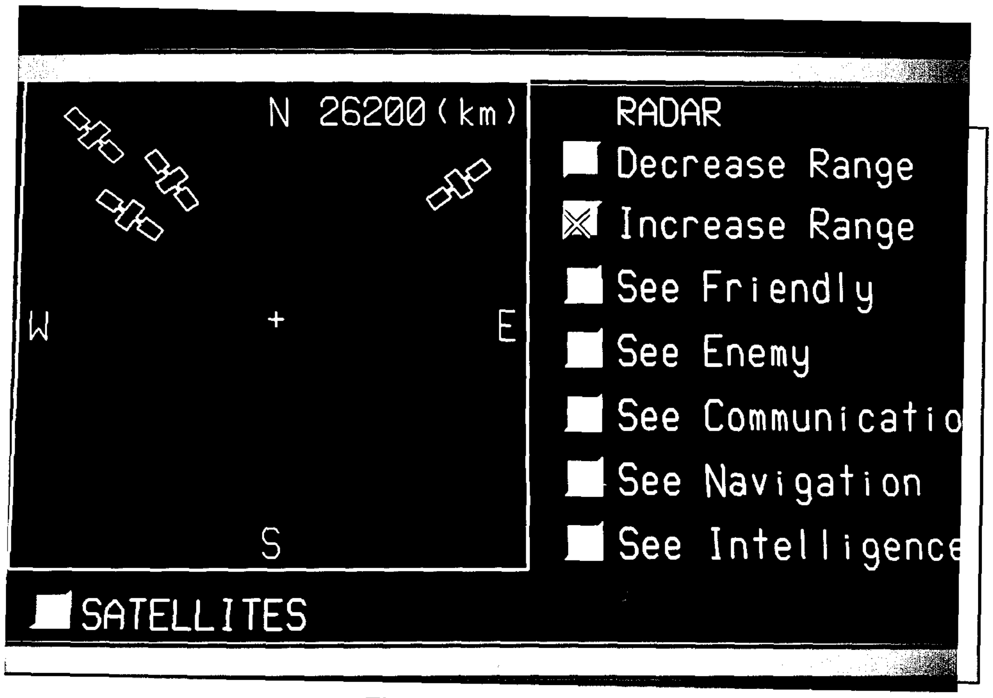


Figure 29: Space Scope

5.4.3 Remote Camera

Figure 30 shows the Pod at the canyon with the remote camera showing the view from the lake.

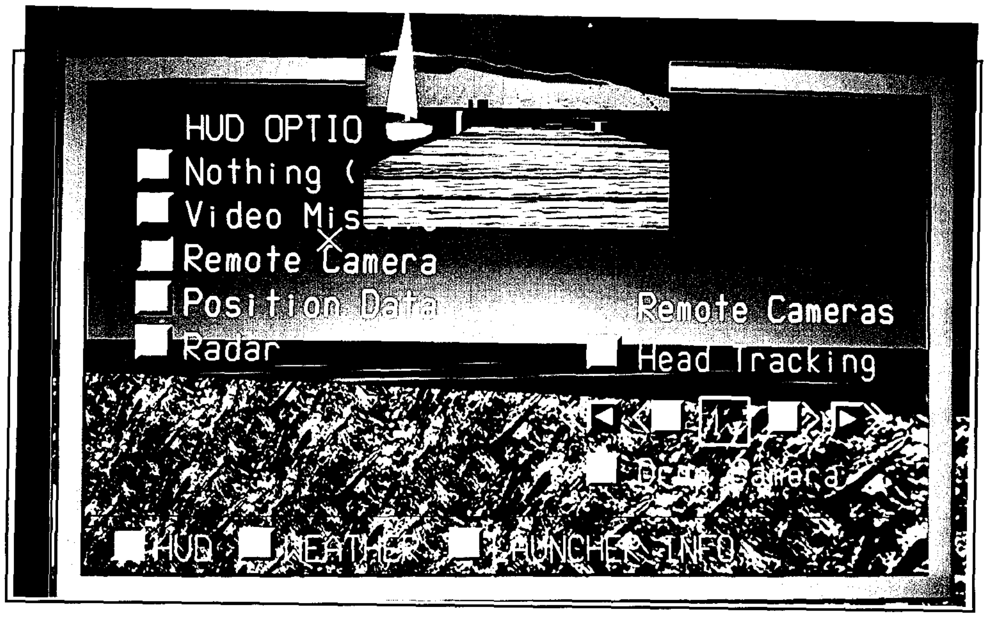


Figure 30: Remote Camera (No Head Tracking)

5.4.4 Video Missile

Figure 31 shows the view from a missile that launched over the lake and is now at the far shore. Tanks can be seen on the horizon.

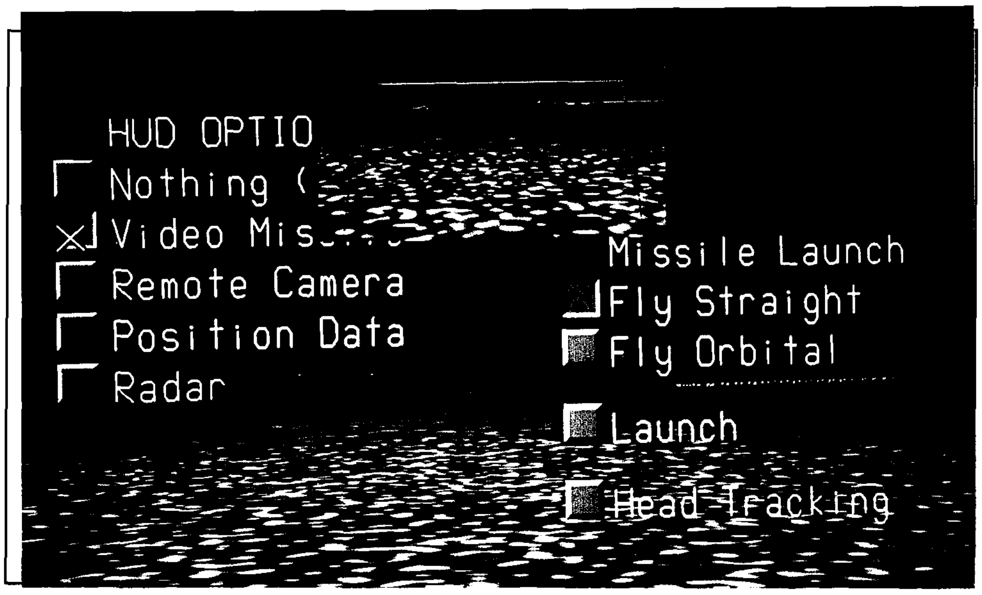


Figure 31: Video Missile

5.4.5 Threat Ranges/ Radar Cones

The threat cones and radar cones are represented as models in the SBB and can be turned on and off. The translucent nature of the cones slows the frame rate and there are some Z-buffer artifacts when they are rendered. The Z-buffer problem can be solved in software with some redesign of the rendering process but at the cost of a performance penalty. The cones go all the way to the ground and into revines and valleys, as opposed to being blocky near the ground. This previous limitation is overcome because I used polygon models to represent the cone. Figure 32 shows a radar cone generated by a Mig-31, which is inside the locator. Figure 33 shows the threat envelope of an SA-6 site.

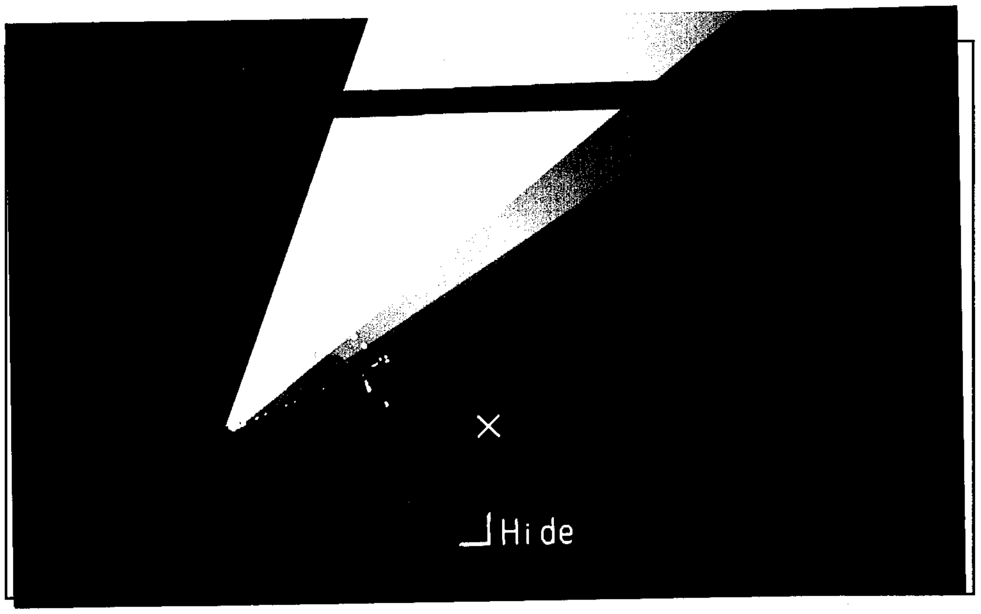


Figure 32: Mig-31 Radar Cone

Figure 33: SA-6 Threat Envelope

5.5 Performance

5.5.1 Pause

The pause function stops the update of vehicles in the environment. When this is activated all Pod functionality remains fully active. When the pause is removed a period of about 45 seconds when the display is inaccurate.

5.5.2 Level of Detail/ Transparency

The locators are no longer semi-transparent, they are now opaque. This doesn't to detract from the effectiveness of the locators. The locators now also act as a levels of detail on all vehicles. This helps the display rate in extreme situations when the polygon count is high. It is difficult to measure the exact effects because there is no easy way to switch the effect on and off, but the theory of Performer says the reduction in polygon count should have a positive impact on the frame rate.

5.6 Recommendations

5.6.1 Weather

Two major categories of improvement can be made to the weather generator. Firstly, the weather should be more accurately/scientifically rendered. Secondly, an extension of current available weather features is needed.

The position of the sun and moon currently makes a notional orbit around the battlefield, it would better to have a sun and moon position server. This server could be a program executing on a physically separate computer that can calculate the precise position of the sun and moon in the sky given a location on the planet and a precise time and date. Such calculation would surely not be unique to just the SBB and a remote server would allow the SBB and all future program to have accurate weather positioning.

The clouds in the SBB could also be improved. The current set of nimbus clouds are a notional representation of what the clouds might look like under one particular circumstance. An accurate real-time and simulated cloud server, similar to the sun/ moon server, would be extremely beneficial. Tough technical questions must first be addressed in order to ensure that all players in a DIS exercise are using the same weather information before this is feasible. An additional issue of how to quickly render more complex cloud formations on a computer must also be addressed. Significant progress has been made in rendering realistic looking clouds and similar weather phenomena in a non-real-time application but there has not been any noticeable research addressing the issue of representing clouds in a real-time, polygon based computer.

5.6.2 Movement

The current method of attaching yourself to a desired vehicle can be improved. If the user knows that he wants to attach and fly along with the F-15 cruising down the canyon he must currently cycle through all players until he finds the right plane. Perhaps a method of specifying airplane type and geographical location could be employed to accelerate vehicle acquisition.

5.6.3 Situation Awareness

Improvement could also come from the full incorporation of the Fuzzy Logic Sentinel into the current *SBB*. Software incompatibilities have slowed integration, if these can be overcome then remote monitoring of sites of interest can be obtained without direct visual observation, the computerized sentinel can watch the interesting positions and report to the commander the activity levels of the area.

Another area of improvement would be the ability for multiple Simulate Battle Bridges to communicate and cooperate with each other. Commanders in one room could communicate with another commander across the nation and have a meaningful tactical dialogue without leaving the virtual environment.

5.7 Conclusion

The Synthetic Battlebridge has been implemented with tools to improve knowledge elicitation and control while in a virtual environment. The weather effects provide a greater sense of immersion and realism in the environment and the ability to control these elements adds to the control available to the commander. The addition tools affecting situational awareness provide the means for the commander to observe more than one position at a time without diverting resources from the primary area of interest. The overall design has been directed toward a virtual environment but the tools can also readily translate to normal CRT based operation. The SBB has the potential to be a powerful training tools as well as an aid to commanders on tomorrow's battlefield.

Appendix A. Users Manual

This user manual is intended to give simple and direct instructions for the first time user. Note that the Daemons and Fastrak must be up and running before the SBB can be executed; however, if you are not using a head mounted display (HMD), then you will not need the Fastrak.

A.1 How to Run the Daemons

The SBB will not run unless the Daemons are up and running on the intended workstation. The Daemons allow the SBB to receive PDUs across the network. There are two ways to execute and terminate the Daemons: by using script files or manual commands.

A.1.1 The Script (Easy) Method for Running the Daemons

To start the Daemons

- go to the directory that the Daemons are in
- type: **startnet**

To stop the Daemons

- go to the directory that the Deamons are in
- type: **stopnet**

To see what port the Daemons are running on

- go to the directory that the Deamons are in
- type: **port**

To see if the Daemons are running

- go to the directory that the Deamons are in
- type: **statnet**
- You should see something like this:

```
IPC status from /dev/kmem as of Wed Sep  7 12:22:22 1994
T      ID     KEY            MODE           OWNER       GROUP
Message Queues:
Shared Memory:
m      0      0x000009a4     --rw-rw-rw-    mdiaz       eng
m      1101   0x00bac0ed     --rw-rw-rw-    rohrer      eng
m      1102   0xccbac0ed     --rw-rw-rw-    jrohrer     eng
m      903    0x00bac0ec     --rw-rw-rw-    ssheasby    eng
m      904    0xccbac0ec     --rw-rw-rw-    ssheasby    eng
Semaphores:
s      110    0x00bac0ed     --ra-ra-ra-    jrohrer     eng
s      91     0x00bac0ec     --ra-ra-ra-    ssheasby    eng
6959   ttyq1  0:00    sgisendd      -p3000
24945  24:22          sgirecvd      -p3000
```

A properly functioning Daemon will have four shared memory ids, two semaphores ids, a sgisendd process running, and a sgirecvd process running.

A.1.2 The Manual (Hard) Method for Running the Daemons

To start the receive Daemon

- go to the directory that the Deamons are in
- type: **sgirecvd -p3000 -b50 &**
- You should see something like this:

```
Starting send Daemon on
Version 2.00
Mode: DIS
Broadcast address: 129.92.101.111
Number of buffers: 50
Network interface: et0
Port: 3000
Buffer size: 552
```

To start the send Daemon

- go to the directory that the Deamons are in
- type: **sgisendd -p3000 -b50 &**
- You should see something like this:

```
Starting send Daemon on
Version 2.00
Mode: DIS
Broadcast address: 129.92.101.111
Number of buffers: 50
Network interface: et0
Port: 3000
Buffer size: 552
```

To kill the receive Daemon

- go to the directory that the Deamons are in
- type: **sgirecvd -p3000 -q**
- You should see something like this:

```
Stopping network Daemon and cleaning up
```

To kill the send Daemon

- go to the directory that the Deamons are in
- type: **sgisendd -p3000 -q**
- You should see something like this:

```
Stopping network Daemon and cleaning up
```

If you don't see this, then the above method did not work, and you must kill the Daemons manually.

To kill the Daemons manually (if necessary)

- type: **ps -elf | grep sgi** (shows the proccess id's)
- You should see something like this:

```
30 S    ssheasby       24943          1  0  28 20  *   300:102
83bc5b00       Aug 29?         16:46           sgisendd -p3000 -b100
30 S    ssheasby       24945          1  0  26 20  *   300:113
80186850       Aug 29 ?        24:18           sgirecvd -p3000 -b100
```

- type: **kill 24943 24945** (XXXXX is the process #)
- type: **ipcs** (This will list the shared memory id's)
- You should see something like this:

```
IPC status from /dev/kmem as of Wed Sep  7 12:01:38 1994
T       ID      KEY            MODE          OWNER       GROUP
Message Queues:
Shared Memory:
m       0       0x000009a4     --rw-rw-rw-   mdiaz       eng
m       1001    0x00bac0ed     --rw-rw-rw-   ssheasby    eng
m       1002    0xccbac0ed     --rw-rw-rw-   ssheasby    eng
m       903     0x00bac0ec     --rw-rw-rw-   ssheasby    eng
m       904     0xccbac0ec     --rw-rw-rw-   ssheasby    eng
Semaphores:
s       100     0x00bac0ed     --ra-ra-ra-   ssheasby    eng
s       91      0x00bac0ec     --ra-ra-ra-   ssheasby    eng
```

- type: **ipcrm -m1001 -m1001 -m903 -m904 -s100 -s91**
- type: **ipcs** (verify that everything is gone, except the root)

A.2.2 Details on the Fastrak Hardware

Ensure you have the following external settings on the POLHEMUS 3SPACE FASTRAK unit:

A.2.2.1 Front panel Settings of Fastrak Unit

For all the 94d applications, you should be using one receiver and one transmitter. Not surprisingly, the transmitter cord plugs into the outlet labeled TRANSMITTER on the front side of the box. The one receiver should be plugged into receiver port ONE (there are a total on four receiver ports available).

IMPORTANT: The select receivers switch on the front of the box should have the following settings:

1 - **OFF**
2 - **ON**
3 - **ON**
4 - **ON**

Yes, this is counter-intuitive and does not make sense. But that is the way it needs to be to work.

A.2.2.2 Rear panel Settings of Fastrak Unit First, make sure you have power applied to the unit. You can confirm this by checking if the little fan in the back is blowing-out air. Also, when disconnecting/connecting the power, do this with the plug at the transformer. It is not recommended to frequently plug/unplug directly into the unit itself. Fortunately, the newer Fastrak models have a on/off switch in the back with a power indicator on the front panel.

Connect your cable from the computer port to the RS-232 outlet on the FASTRAK unit. Note that you need a special cable for this. For our SGI applications, the cord should be customized such that pin numbers 12345678 correspond to 12352678 (not one-to-one). A one-to-one connection will also work, but you will have to re-start the FASTRAK twice before it works properly because the grounding is not proper.

Finally, make sure the I/O SELECT port is properly set. For all our afa94 applications, the setting should be as follows:

Switch	Setting	Comments
1	0	9600 baud
2	0	9600 baud
3	1	
4	1	
5	1	8 bits/character
6	0	no parity
7	0	no parity
8	1	RS232

This should be enough information to get all the 94d applications up and running with the position tracker on the HMD. Finally, realize that the transmitter and receiver work off of electromagnetic fields. That means keeping the transmitter and receiver relatively close together, and away from metal or any monitors.

Refer to the *3SPACE USER'S MANUAL* for more detail.

A.3 How to Run the Gaggle

The gaggle software generates players for the SBB to read off of the network. The gaggle allows for 10 to 500 players to be generated and broadcasted. Note that the gaggle and the SBB cannot be run on the same machine because the Daemons were designed to handle only one application at a time. Thus the Daemons must also be running on the machine the gaggle is running on. Finally, it is not necessary for the gaggle to be running prior to starting the SBB.

To start the gaggle

- Ensure that the send and receive Daemons are running.
- Go to the directory that the gaggle is in
- Decide how many people that you want to be in the gaggle.

This number must be a multiple of 10! The first line in the file*init_gaggle* should reflect the number of players divided by 10.

ex 1. If you want 100 players broadcast by the gaggle then you would put a 10 in the file *init_gaggle.*

ex 2. If you want 30 players broadcast by the gaggle then you would put a 3 on the first and only line of the file *init_gaggle.*

Note:

Max Players = 500	(50 on first line)
Min Players = 10	(1 on the first line)

- type: **gaggle**

To Stop the gaggle

- type: **CTRL C**

A.4 How to Run the SBB

To start the SBB

- Ensure the receive Daemon is running
- If you do not want to use the Fastrak, then the last line of the file *fastrak.dat* should be set to '0'. If you want to use the head tracking abilities (Fastrak) then do the following:
    - Ensure that the first line of *fastrak.dat* is set to the port that the Fastrak is hooked up to.
    - Ensure that the last line of *fastrak.dat* is set to '1'. This tells the SBB to use the Fastrak instead of the keyboard.
    - Ensure the Fastrak_Server is running.
- Go to the SBB directory
- type: **SBB**

  Expect the screen to be black/blank for about a minute before the SBB is up and running.

To run the SBB

- There are a total of five panels to choose from:

  1. Front panel
  2. Left panel
  3. Right panel
  4. Top panel
  5. Bottom panel

- Only one panel is active at a time. The panel the viewer is looking at is the panel currently active. To change the active panels, just look in the direction of the desired panel.

- The keyboard mouse moves the X on the selected panel. Use the X to select a button by positioning the X over the desired button.
- The left mouse button activates the selected button on the panel.
- If the Fastrak/HMD is not used, you can change your view by putting the mouse in the SBB window and then using the keypad.

Look up
<8>

Rotate left <4> <6> Rotate right

<2>
Look down

Move Up
<up arrow>

Move left <left arrow> <right arrow> Move right

<down arrow>
Move down

<+> Move forward

<enter> Move back

To Stop the SBB

- Hit **ESC** to quit. The mouse must be in the SBB window, but it will not be visible.

7. Bibliography

Aeiry, John M. et al. "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments" *Proceedings of the Human Factors in Computing Systems conference,* 1989.

Bevin, Alexander. *How Great Generals Win.* New York: W W Norton & Company, 1993

Block, Elizabeth G. and Martin R. Stytz. "Tools for Commander and Staff Training in Large-Scale, Distributed Virtual Realities: Concepts and Implementation." *Proceeding of the Military, Government and Aerospace Simulation Conference.* 1994

Bryson and Levision, as cited in Block

Chung, J.C. et al. "Exploring Virtual Worlds with Head-Mounted Displays." *Proceedings from the Non_holographic True-3 Dimensional Display Technologies, SPIE.* 1989

Cooke, Joseph M. and Michael Zyda. "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions," *Presence,* 404-420. Fall 92

Diaz, Milton E. *The Photo Realistic AFIT Virtual Cockpit.* MS thesis, GCS/ENG/94D-02, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1994

Fortner, Jon L. *Distributed Interactive Simulation Virtual Cassette Recorder (DIS VCR): A Datalogger with Variable-Speed Replay.* MS thesis, GCS/ENG/94D-10, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1994

Gerhard, Jr., William E. *Weapon System Integration for the AFIT Virtual Cockpit.* MS thesis, GCS/ENG/93D-10, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1993

Kestermann, Jim B. *Immersing the User in a Virtual Environment: The AFIT Information Pod Design and Implementation.* MS thesis, GCS/ENG/94D-13, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1994

Levison and Bryson, as cited in Block

Rohlf, John and James Helman "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics." *Proceeding of the SIGGRAPH 94.* 1994

Sheasby, Steven. *Management of Simnet and DIS Entities in Synthetic Environments.* MS thesis, GCS/ENG/92D-16, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1992

Snyder, Mark. *ObjectSim Application Developers Manual.* Air Force Institute of Technology, Wright-Patterson AFB, OH

Stytz, Martin R. and Elizabeth Block. "Providing Situation Awareness Assistance to Users of Large-Scale, Dynamic, Complex Virtual Environments," *Presence*, 297-313. Fall 93

Sun Tzu. *The Art of War*

Walser, Randal. "Doing It Directly- the Experiential Design of Cyberspaces." *Proceedings from the 1990 Symposium on Interactive 3D Graphics*: 1990

Vanderburgh, John. *Space Modeler: an Expanded, Distributed, Virtual Environment for Space Visualization.* MS thesis, GCS/ENG/94D-23, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1994

8. Vita

Lieutenant J.J. Rohrer was born on 4 Nov, 1970 in Seattle Washington. He graduated from Liberty High School and then attended the US Air Force Academy, graduating in 1993. The Air Force Institute of Technology was his first assignment as a commissioned officer.

Permanent address: 13817 162nd Ave SE
Renton, WA 98059

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
DESIGN AND IMPLEMENTATION OF TOOLS TO INCREASE USER CONTROL AND KNOWLEDGE ELICITATION IN A VIRTUAL BATTLESPACE

5. FUNDING NUMBERS

6. AUTHOR(S)
Jim J. Rohrer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology,
WPAFB OH 4533-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/94D-20

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Lt Col F.T. Case
ARPA/ASTO
Advance Simulation Technology Office
3701 North Fairfax Drive
Arlington VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

AFIT's Synthetic BattleBridge is an immersive command observatory for viewing large-area activity within a virtual environment. Three basic areas for improvement are addressed: 1) an improved, immersive, user interface, 2) direct control of atmospheric effects, and 3) improved knowledge elicitation by means of remote viewers, a new space scope, and an enhanced RADAR scope. These requirements are analyzed and implemented. Some functionality was lost but the results show a general improvement of environmental control and knowledge [illegible] elicitation.

14. SUBJECT TERMS Synthetic BattleBridge, Satellite Modeler, Virtual Reality, Pod, Panel, Commander	15. NUMBER OF PAGES 94
	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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102