NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MOUNTING HUMAN ENTITIES TO CONTROL AND INTERACT WITH NETWORKED SHIP ENTITIES IN A VIRTUAL ENVIRONMENT

by

Bryan Christopher Stewart

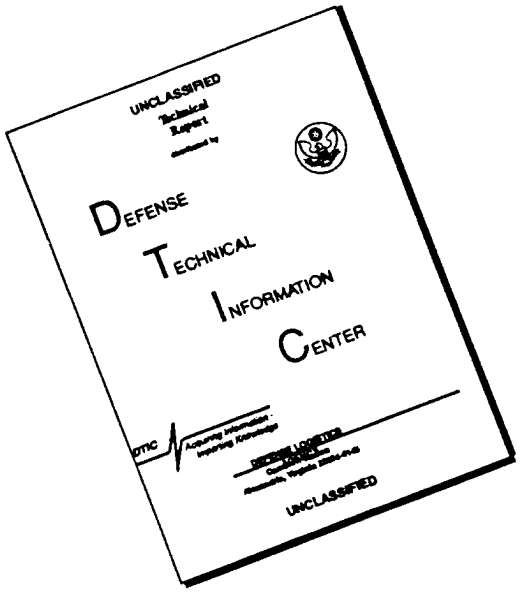
March 1996

Thesis Advisor: Michael J. Zyda
Thesis Co-Advisor: John S. Falby

Approved for public release; distribution is unlimited.

19960520 035

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1996	Master's Thesis

4. TITLE AND SUBTITLE
Mounting Human Entities to Control and Interact With Networked Ship Entities in a Virtual Environment

5. FUNDING NUMBERS

6. AUTHOR(S)
Stewart, Bryan Christopher

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis research addresses the problem of mounting human entities to other non-human entities in the virtual environment. Previous human entities were exercised as individual entities in the virtual environment. Yet there are many applications (i.e. shipboard damage control, amphibious landings, helicopter vertical assaults) where human entities need to mount other vehicles within the virtual environment.

The approach taken was to re-engineer the Naval Postgraduate School's Shiphandling Training Simulator (SHIPSIM) and Damage Control Virtual Environment Trainer (DC VET) onto a common virtual environment system (NPSNET). Using a modified potentially visible set algorithm, a ship hydrodynamics model, and a simple data PDU network packet, NPSNET human entities were given the capability to mount ship vehicles. Additionally, a control panel and voice recognition were added to allow the human entities to control and maneuver the ship vehicles in the virtual environment.

As a result of this thesis, NPSNET human entities can mount ship vehicles, move about the ship, and interact with the ship's internal objects (i.e doors, valves, etc.) all while the ship moves within the virtual environment. This technology opens a new paradigm for simulation designers, where users of virtual environment systems can participate as human entities and interact (i.e. mount, control, and maneuver) with other inanimate vehicles as we do in the real world.

14. SUBJECT TERMS	15. NUMBER OF PAGES
NPSNET, DIS, real-time, 3D, visual simulation, network, distributed, Performer, interactive, virtual world, PVS, Potentially Visible Sets, terrain database, Voice Recognition, mounting humans entities	92
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited.

MOUNTING HUMAN ENTITIES TO CONTROL AND INTERACT WITH NETWORKED SHIP ENTITIES IN A VIRTUAL ENVIRONMENT

Bryan C. Stewart
Lieutenant, United States Navy
B.S.C.S., Stanford University, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

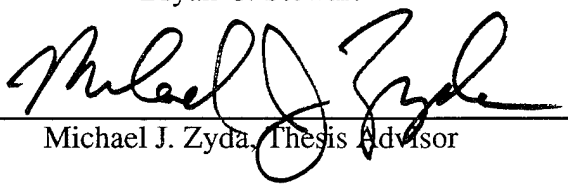
from the

NAVAL POSTGRADUATE SCHOOL
March 1996

Author :

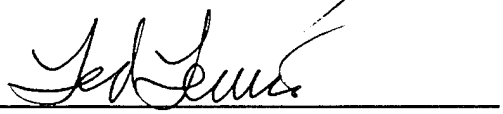
Bryan C. Stewart

Approved by:



Michael J. Zyda, Thesis Advisor

John S. Falby, Thesis Co-Advisor



Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

This thesis research addresses the problem of mounting human entities to other non-human entities in the virtual environment. Previous human entities were exercised as individual entities in the virtual environment. Yet there are many applications (i.e. shipboard damage control, amphibious landings, helicopter vertical assaults) where human entities need to mount other vehicles within the virtual environment.

The approach taken was to re-engineer the Naval Postgraduate School's Shiphandling Training Simulator (SHIPSIM) and Damage Control Virtual Environment Trainer (DC VET) onto a common virtual environment system (NPSNET). Using a modified potentially visible set algorithm, a ship hydrodynamics model, and a simple data PDU network packet, NPSNET human entities were given the capability to mount ship vehicles. Additionally, a control panel and voice recognition were added to allow the human entities to control and maneuver the ship vehicles in the virtual environment.

As a result of this thesis, NPSNET human entities can mount ship vehicles, move about the ship, and interact with the ship's internal objects (i.e doors, valves, etc.) all while the ship moves within the virtual environment. This technology opens a new paradigm for simulation designers, where users of virtual environment systems can participate as human entities and interact (i.e. mount, control, and maneuver) with other inanimate vehicles as we do in the real world.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. BACKGROUND

1. Shiphandling Training Simulator

The first part of the software product developed from this thesis is a continuation of the Shiphandling Training Simulator or SHIPSIM, see Figure 1. SHIPSIM is an interactive networked real-time virtual environment for maneuvering a ship in various shiphandling evolutions, such as piloting in restricted waters, mooring to a buoy, and underway replenishment. It was designed to train junior officers to conn U.S naval ships and reduce at sea collisions [NOBL95].

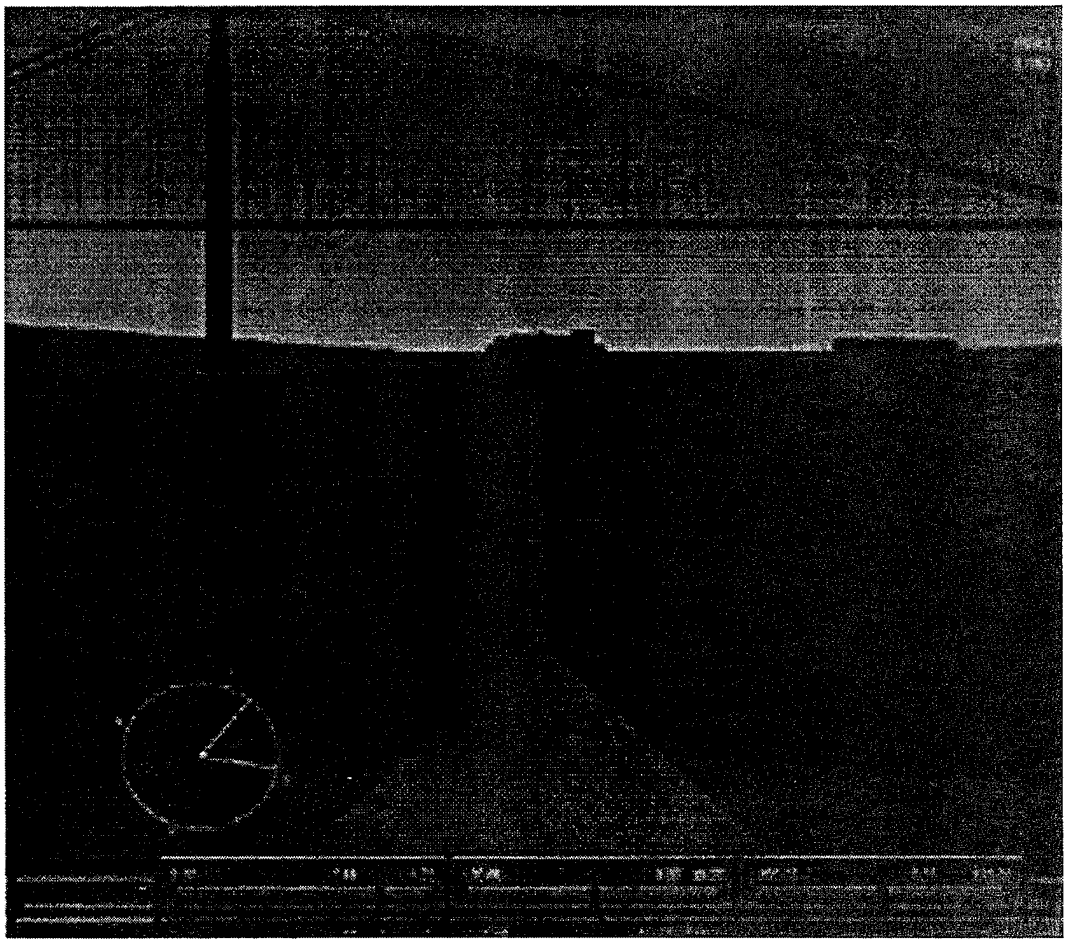


Figure 1: Sample Scene in SHIPSIM

Utilizing Silicon Graphics Reality Engine Workstations, SHIPSIM gives the user the capability to maneuver through the virtual environment using a ship control panel. The virtual environment contains buoys and channel markings to guide the user to different locations in the world. With the use of Distributed Interactive Simulation, multiple people can control their own vessel and participate in fleet maneuvering exercises over an Internet connection [LOCK94]. The limit to the number of ships in the environment is a function of the capabilities and limitations of the user's workstation.

This thesis uses the SHIPSIM virtual environment to represent the exterior world in which the ship vehicles maneuver. Additional ship, submarine, or air vehicles can also participate in the environment to produce cross platform training opportunities.

2. Damage Control Virtual Environment Trainer

The second part of the software product developed from this thesis is a continuation of the Damage Control Virtual Environment Trainer or DC VET, see Figure 2. DC VET is an interactive networked real-time virtual environment for moving a human-entity through an entire ship. It was designed to train navy personnel to fight shipboard casualties using standard navy doctrine, and indoctrinate themselves to the objects found inside of most navy ships [OBYR95].

Utilizing Silicon Graphics Reality Engine Workstations, DC VET enables the user to move about a ship model in a realistic fashion. By visiting certain locations within the ship model, the user can associate objects in the model with real world objects found aboard navy ships. Shipboard casualties, such as fires and steam leaks, can be started to give the user training in damage control procedures. Multiple people can interact within the ship model to perform team training evolutions, an important part of Damage Control Training.

This thesis utilizes the DC VET virtual environment to represent the interior world which the user manipulates to control the ship. With the addition of other human entities into the ship model, multiple people can control the same ship.

Most virtual worlds only give the user the opportunity to interact with one virtual environment; either an exterior world, as with a flight simulator, or an interior world, as with a building walkthough. This thesis seeks to allow an entity to interact with more than one virtual environment simultaneously.

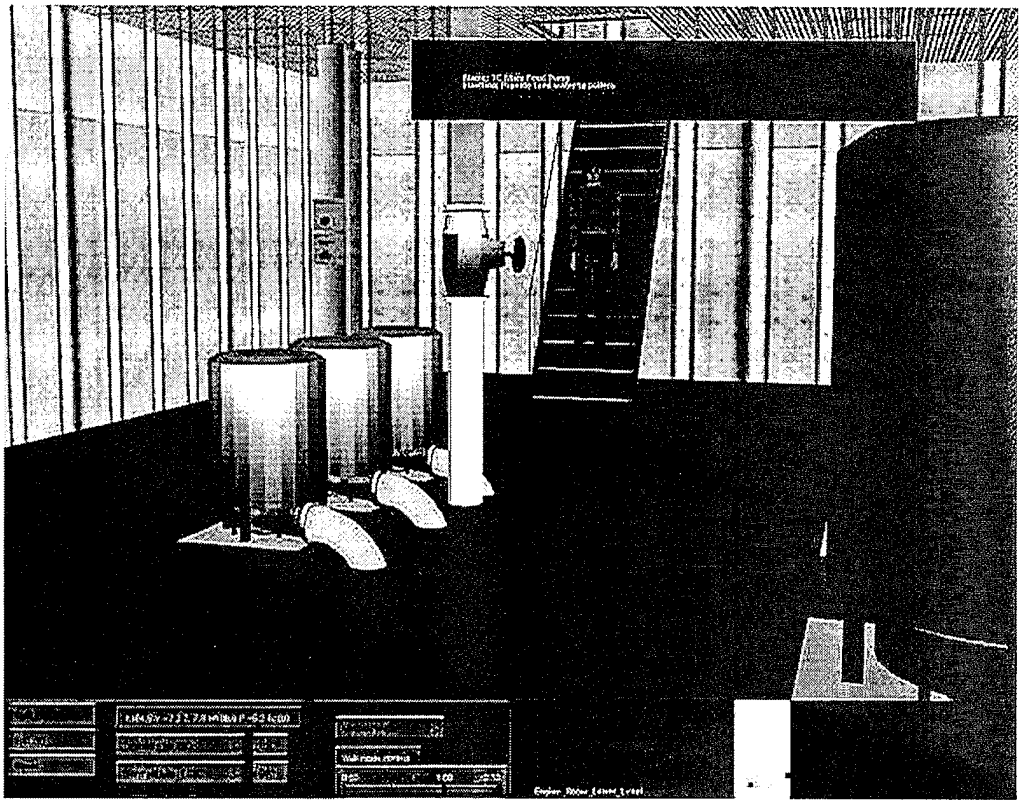


Figure 2: Sample Scene in DC VET

B. MOTIVATION

1. Navy Training

"Naval training of fleet sailors has traditionally started with classroom instruction, followed by simulators and hands on training once aboard ship. Classroom instruction can have many styles ranging from classroom training of a group, to training with a video presentation. Often after a level of instruction is completed, there is mock-up training in a controlled environment."[OBYR95] Often the controlled environment is either too expensive to operate and maintain, or is too unrealistic to be productive. Virtual

environments reduce maintenance and operation costs by being run on self contained workstations, and increase productivity by maintaining realism throughout the evolution.

2. Limited Underway Time

As the defense budget gets smaller, the number of available operational training opportunities for sailors decreases. As a result, Commanding Officers have to take full advantage of all possible training opportunities for their crews. Yet, given the wide variety of exercises and evolutions that a ship must go through for its crew members, it becomes increasingly difficult to ensure that each crew member has satisfactorily fulfilled all the requirements of a given exercise. This can lead to disastrous results when a crew member is qualified on paper, but does not have the "hands-on" skills necessary to perform his duties. The less often ships go to sea, the more sailors will be unqualified to perform their duties at sea.

Virtual environment trainers enable a ship's crew to continue to perform underway evolutions while their ship remains pierside. Rather than reading or talking about an underway evolution, the crew can actually perform the evolution, and qualifications can be verified or maintained through repeated virtual training runs. Today many companies are looking towards virtual environments to train their employees to perform life threatening and time critical evolutions.

3. A Question of Safety

The concern for safe operations at sea has been an issue of paramount importance to the United States Navy for over twenty-five years. Numerous maritime mishaps (collisions, groundings, etc.) have resulted in lost revenue due to costly cleanups and repairs, and loss of life. Review and analysis of mishap data provided by the Naval Safety Center revealed that 95 percent of these maritime mishaps could have been avoided had the personnel involved taken more timely measures to prevent or reduce the impact of the collision[MSI94][USN94].

Traditionally, young junior officers have been assigned to frigates and destroyers, which have historically suffered the greatest number of maritime mishaps. Additionally, when considering the role of these particular ships (i.e. ASW) and their high maneuverability, there is a need for the these officers to develop their shiphandling and situational awareness to an even higher level of proficiency than those assigned to other types of ships (i.e. CVN, AOE, PHM). A similar trend has been noticed when comparing large and small amphibious warfare ships. The frequency of collisions in the small ship classes (i.e LSTs, LSDs, LPDs) was found to be 52.3% greater than their larger counterparts (i.e. LHAs, LHDs, LPHs)[USN94].

The mishap data also revealed the types of evolutions in which U.S. Naval vessel collisions were more likely to occur. Surprisingly, the majority of all reported collisions occurred when vessels were involved in basic mooring and anchoring evolutions. It is during these particular evolutions that the skills of a shiphandler, or the lack thereof, are most apparent (i.e. getting underway from a pier, returning to port and mooring to a pier, etc.). To successfully perform each of these tasks, the shiphandler must be competent in his/ her skills. They must exercise more than just a general working knowledge of basic shiphandling by thoroughly understanding the direct impact that each force plays on the handling capabilities of his/her individual vessel. Failure to do so often results in mishaps, which ultimately result in costly damage to equipment and personnel injuries[NOBL95].

Virtual environment trainers can effectively reduce mishaps at sea by giving inexperienced shiphandlers a chance to develope their skills in a safe and controlled environment. The valuable lessons learned in the classroom and in the trainer enhance the at sea experiences gained by the shiphandler in the future.

C. OBJECTIVE

The objective of this thesis is to combine the Damage Control Virtual Environment Trainer (DCVET) and the Shiphandling Training Simulator (SHIPSIM) to produce a single real-time networked interactive virtual environment for shipboard and shiphandling

training. This environment encompasses features of both trainers, and provides a more suitable platform with which to integrate more advanced shipboard and shiphandling features. Additionally, the trainer enables multiple users to coordinate themselves within the same environment to control a single ship, a necessity for team training evolutions.

Many aspects of a true "virtual environment" are explored to give the user the best possible immersive feeling, given the state of current virtual environment technology. The trainer is developed on top of the already established NPSNET (Naval Postgraduate School Network Vehicle Simulator) currently being supported at the Naval Postgraduate School. NPSNET provides many features that were replicated within the SHIPSIM and DCVET trainers (i.e. HMD capabilities, DIS network protocols, immersive sounds, etc.), but does not provide other features such as voice recognition.

The final objective of this thesis is to provide human entities within the virtual environment the capability to embark onboard, or debark from, ships or buildings that themselves are virtual environments. Traditionally, human entities have been limited to a single virtual environment, either an external terrain such as within NPSNET, or an internal structure such as DC VET. Such a capability allows developers to design structures that can be later walked through and manipulated by human entities. Additionally, allowing these structures to move as entities themselves within the exterior virtual environment is a more realistic representation of how we design and utilize structures for transportation. It is hoped that this type of trainer will replace the traditional virtual environment vehicle simulators in the future.

D. SUMMARY OF CHAPTERS

The remainder of the thesis is broken down as follows:

- Chapter II presents an analysis of the advantages and disadvantages of the SHIPSIM, DC VET, and NPSNET simulators.
- Chapter III presents a system overview of the NPSNET ship project.

- Chapter IV presents a description of the ship control panel and the voice recognition system.
- Chapter V presents a detailed description of the potentially visible sets (PVS) algorithm used by NPSNET ships.
- Chapter VI presents a detailed discussion of the methods used to mount human entities to ship entities in a virtual environment.
- Chapter VII provides a final discussion of the results of this thesis and describes follow-on work to be accomplished.

II. PREVIOUS RESEARCH

This chapter briefly examines the research efforts made in the development of virtual environments at the Naval Postgraduate School. Further, it focuses on the distributed computing issues involved with building NPSNET, SHIPSIM and DC VET. Although similar in design, each of these simulators provides different aspects to immerse the user into the virtual environment.

A. NPSNET

NPSNET is a low-cost, student-written, networked vehicle battle simulator that runs on commercial off-the-shelf SGI (Silicon Graphics IRIS) workstations. It has been designed to allow multiple players to drive war-fighting vehicles (tanks, helicopters, infantry) on a simulated 3D battlefield for training and tactics assessment. Players can participate over local LAN networks or multicast over the Internet. Although originally designed as an Army simulation tool, NPSNET has the capabilities and structure to handle Naval units as well.

NPSNET uses the network standard DIS (Distributed Interactive Simulation) protocol for network communications, and thus can be run with any other DIS standard protocol simulator. DIS is a network standard that was developed from SIMNET (Simulation Network), a distributed interactive simulations standard developed by DARPA in 1985 [LOCK94]. Both protocols utilize many types of Protocol Data Units (PDU) to transmit information between simulators. Each PDU is used for a specific purpose, and encompasses all aspects of a simulation. A listing of DIS PDU's is given in Table 1.

PDU (official)	Purpose
Entity State	Update vehicle state, i.e location, appearance
Fire	Report Firing of Weapon
Detonation	Report impact or detonation of munition
Collision	Report collision of entity with another object
Service Request	Request transfer of supplies
Resupply Offer	Offer supplies to another entity
Resupply Received	Acknowledge receipt of some or all offered supplies
Resupply Cancel	Cancel resupply service
Repair Complete	Report completion of repair
Repair Response	Acknowledge completion of repair
Start/Resume	Instruct entity to participate in simulation
Stop/Freeze	Instruct entity to stop participation in simulation
Acknowledge	Acknowledge receipt of Start, Stop, Create, or Remove PDU
Action Request	Request specific action from an entity
Action Response	Acknowledge receipt of Action Request PDU
Data Query	Request data from an entity
Set Data	Set or change parameters in an entity
Data	Return data requested by Data Query or Set Data PDUs
Event Report	Communicate occurance of significant event
Message	Send miscellaneous messages
Create Entity	Establish identity of new entity
Remove Entity	Remove entity from simulation
Electromagnetic Em	Communicate elect emissions other than radio
Designator	Communicate designation emissions
Transmitter	Announce start/stop of radio transmission
Signal	Transmit voice or data
Receiver	Communicate state of radio receiver

Table 1: DIS PDU's

The DIS PDU's allow the developing events of a simulation to be shared among different host sites. However, the PDU's do not describe the initial state of the simulation. This must simply be agreed upon by simulation participants beforehand. The most important part of the initial state is the terrain with which the units act upon. The terrain is a database of elevation measurements for the grid points of some area in the real world. NPSNET contains several of these including FT. Hunter-Liggett, FT. Benning, and San Francisco Bay. Additionally, host sites must agree upon the types of vehicles that may participate in the simulation and how they will be identified.

Although the DIS standard is rich with simulaton PDU's, NPSNET uses only a few of the DIS PDU's: Enitity State, Fire, and Detonation. After a simulator has been brought up, it broadcasts an initial Entity State PDU to everyone on the net. This initial state contains the initial starting position and orientation of the simulator's vehicle. The simulator then begins its simulation loop, consisting of the following:

- updating the environment
- updating the scene
- updating the view point
- rendering the scene
- reading changes to control devices
- writing PDU's for the simulator's vehicle state changes
- reading the network for PDU's

At the beginning of the loop, the simulator environment is updated. The system checks to see if the window size or display modes such as anti-aliasing, texturing, and wireframe have changed.

The scene database is updated next. The scene database is a hierarchical tree that contains all items to be rendered. The terrain, static objects, and dynamic entities are all branches of the scene database. The larger the tree, the more time it takes to render the scene. Updates to the scene, such as movement of dynamic entities or collisions with static

objects, may result in moving branches of the tree from one location to another or changing values within a branch.

Next the user's view point is updated. The view point describes how the user views the scene database. Changes to the view point follow the movements of the user's vehicle throughout the scene.

Once the view point has been set, the updated scene is rendered to the screen. The scene database is stripped of objects that are out of the user's view. The objects to be displayed are placed in a display list and eventually drawn on the screen.

Following scene rendering, the control device (keyboard, flight sticks, space ball, etc.) is read and changes are made to the user's vehicle accordingly. An Entity State PDU is created and sent across the network to inform everyone in the simulation of the user's new position.

Finally, all the other vehicles in the simulation are updated. These changes are read from the network. PDU packets are filtered to make sure that they are both Entity State PDU's and that they are part of this simulation. As PDU packets are read, a vehicle list is checked to see if the PDU entity exists in the list. If it exists, the vehicle is updated based on the information within the PDU, otherwise a new vehicle is created and added to the list from the information in the PDU [HART94].

The simulation loop continues indefinitely until the user quits the application, or the simulator crashes. When the user quits, an exit PDU is sent over the network to inform all others to remove the user's vehicle from their vehicle lists. When the simulator crashes, the other simulators remove the user's vehicle from their vehicle lists after a specified period of time.

A significant problem that occurs when 50 or more players participate in an NPSNET simulation is network saturation. If all the players sent their entity state PDU's every frame of the simulation loop, the network would saturate to a point where no network traffic (email, ftp's, etc.) can occur. To solve this problem, NPSNET puts restrictions on the number of PDU's that a simulator can send. Each simulator only sends out a PDU

packet when the driven vehicle drastically changes its state (i.e. stops moving, begins moving, makes a big speed or course change) or once every 5 seconds. The PDU packet that is sent once every 5 seconds is known as the heartbeat PDU, and it informs every other simulator that this simulator's driven vehicle is still alive in the simulation, regardless of whether it is moving or not.

With the number of PDU packets on the network reduced, how does each simulator maintain accurate positioning information about each vehicle in the simulation? Each simulator dead-reckons all the vehicles in its vehicle list, including itself, to maintain accurate positions. Prior to starting the simulation, all the simulator's must agree upon the method of dead-reckoning for each vehicle. If the user's actual vehicle position deviates by a fixed amount from its calculated DR position, then a PDU packet is sent across the network. This solution solves the network saturation problem for simulations that contain on the order of 250 vehicles. Research is still in progress to find solutions to handle the problem for up to 100,000 vehicles.

As a virtual environment, NPSNET contains a rich library of sounds. The sounds are played in response to events in the simulation, such as bombs exploding, or vehicles crashing into objects. These sounds are played by a sound server process on an SGI workstation. During the load up of the simulator, the sound server is either started on the simulator's machine, or on a machine with a dedicated sound system. The NPSNET sounds are relative to one vehicle; thus each simulator sends its sounds to its own dedicated sound processor. When a vehicle performs an event that generates a sound, the event is processed by all simulators in the simulation, and if the event is close enough to be heard by a simulator, then a sound is played to represent that event. Therefore, simulators that do not have sound capabilities can still produce events that generate sounds that other simulators can play.

The interface between NPSNET and the sound system is accomplished through an IRIS Indigo Workstation and the EMAX II Sound Synthesizer. A *C++* program on the Indigo Workstation analyzes NPSNET user actions via message packets over the network.

If a certain user action has a sound associated with it, a series of MIDI commands are sent to the EMAX II. The EMAX II deciphers the MIDI commands and generates the appropriate sound. This sound signal is then routed to a Carver power amplifier for output to two Infinity speakers which generate the appropriate aural cues.

Subwoofers dramatically add to the realism of the aural cues. During NPSNET demonstrations, numerous participants commented that the low frequencies generated by the subwoofers dramatically increased their immersion into the virtual environment of NPSNET. Additionally, the MIDI pitch bend command is coincident with the host machine's vehicle speed. As a result, when the vehicle's speed increases or decreases, the vehicle's pitch correspondingly increases or decreases, thus increasing the overall realism of the vehicle's sound. The NPSNET sound system provides a greater level of immersion for players in the NPSNET virtual environment [STOR95].

So far, we have seen NPSNET's network computing capabilities. But the simulator's most important capability is its multiprocessing capability. As a 3D real-time simulator, NPSNET must have a way to draw its dynamically changing scenes at an acceptable frame rate. NPSNET is created from a graphics API (Application Programming Interface) known as Performer. Performer provides a multiprocessing capability that NPSNET uses to perform its tasks.

NPSNET is run as three separate processes: the application, cull, and draw processes. The application process is the driver of the simulator. It initially performs the tasks necessary to start the simulation, including allocating shared memory, initializing global variables, loading of the terrain and entity models, starting the cull and draw processes, and opening a network channel for communication. The application process then performs the simulation loop as described above. The cull process receives the updated scene database and the updated view point from the application process each frame. It then traverses the scene database tree creating a list of those objects that can be seen from the user's view point. It finally sends the display list to the draw process. The draw process

receives the display list, and draws each polygon into a buffer. The buffer is then sent down to the graphics hardware for displaying [HART94].

Because of the time necessary to cull and draw a scene, the cull process is always one frame behind the application process and the draw process is always two frames behind. Consequently, the scene the user is viewing is slightly behind the user's inputs to the simulator and is dependent on the display frame rate. This delay is greatly reduced when NPSNET is run on a multiprocessor machine like the SGI Onyx Reality Engine 2. With four CPU's, the machine runs the application, cull, and draw processes on their own processors, greatly increasing the frame rate with a corresponding reduction in the time latency between input and display.

NPSNET can be run on small single CPU machines, but modifications to the simulator have to made to achieve real time performance. Textured polygons require significant CPU cycles to be rendered, thus on the smaller machines texturing is normally turned off. Anti-aliasing (or smoothing) of polygons also burdens the draw process and can be turned off as well. Finally, if all else fails, wireframe mode can be turned on to prevent the polygons from being filled in with color. These modifications greatly reduce the realism and immersive qualities of the simulation, but do help the single CPU machines perform in real time.

NPSNET is a successful distributed, networked application. Research continues to seek out ways to improve the system's virtual environment capabilities (voice recognition, human entity trackers), and provide its users with the best possible simulation experience.

B. SHIPSIM

SHIPSIM is an interactive networked real-time virtual environment for maneuvering a ship in various shiphandling evolutions, such as Piloting in Restricted Waters, Mooring to a Buoy, and Underway Replenishment. It was designed to provide realistic shiphandling training scenarios utilizing a single, high-speed graphics workstation as a host, preferably a Silicon Graphics Inc. Reality Engine series model equipped with a

monitor, keyboard and mouse pointing device. The purpose of hosting the portable simulator on a single workstation is to allow its placement either aboard a deploying vessel or in the immediate vicinity of one that was in port. This close proximity provides easy access to shipboard personnel desiring shiphandling practice without the need for numerous support and technical personnel normally associated with running the training scenarios in a full-scale simulator.

Limited to a single monitor, the training exercise is displayed in a split screen configuration with the display of the ship and its surrounding scene (tactical viewing area) occupying the upper three quarters of the display and the ship's controls (provided by a simple graphical user interface) occupying the lower quarter (see Figure 1). For more enhanced training, the OOD shiphandling training simulator also has the capability to operate in a distributed network configuration, thus providing multiple ship, multiple user interaction and scenarios, wherein different simulators on the network can act as different ship entities.

When designing the simulator, the task of modeling the movement of the ship was not the only problem. The ability to place the conning officer onto a virtual bridge with only a single monitor available and allowing freedom of movement among specific conning stations or viewing locations also needed to be considered. In addition to moving about the bridge, a form of head movement needed to be implemented so the conning officer could observe a desired viewing angle off the bow or raise and lower the view with respect to the horizon. The approach to solving these problems was to attach the conning officer's viewing position to one of three possible locations on the ship model -- the pilot house, the port bridge wing or the starboard bridge wing. Movement between these positions is accomplished through inputs from the control panel. In essence, the conning officer is immersed into the scene by "riding" the ship model as it moves through the terrain database. When viewing forward, in line with the bow or off the beam, the conning officer senses forward motion as the ship moves forward.

While "underway" in the virtual environment, adjusting the ship's course and speed is easily accomplished through the use of rudder and engine controls located on the control panel immediately below the tactical display area. To add more realism to the scenario, a second person acting as a helmsman could operate the controls in response to voice commands passed by the conning officer. With this arrangement, a more experienced conning officer could give instruction to the less experienced one as the exercise progresses.

To further enhance training, the conning officer is allowed to detach himself/herself from the ship and view the entire exterior of the ship from different external viewpoints while moving through the water. This added feature was developed to provide better visualization feedback to the conning officer as to what the ship looks like during various maneuvering evolutions. Furthermore, the conning officer can fly away from the ship to view the maneuvering evolutions of other ships being reported over the network as well as viewing anticipated turning points. Additionally, controls are provided to adjust both the local time of day and the local visibility by manipulation of lighting and fog levels.

SHIPSIM was implemented using many of the same features as NPSNET. The network interface between ships uses the DIS protocol; the simulator allows multiprocessing of the application, cull, and draw processes; and the simulator performs the same simulation loop steps. However, the two uniquely differ in their simulation scope. SHIPSIM does not have the weapons or sound capabilities that NPSNET does. These effects greatly increase the immersive qualities of NPSNET's virtual environment, and their absence limits the immersive qualities of SHIPSIM.

Another immersive limitation is SHIPSIM's view point control mechanism. The SHIPSIM control panel allows the user to maneuver the ship and control the environmental features of the virtual environment. Although necessary, the control panel limits the amount of user immersion within the virtual environment. The view point controls allow the user to change his view point on the ship, but this movement is not smooth.

Consequently, it is very difficult for the user to hold his view on a particular object in the virtual world. This makes it very difficult to sail close to objects in the virtual environment.

Finally, SHIPSIM utilizes simple textured polygons to represent its waterways. To enhance the sense of immersion into the environment, the waterways should be represented by true hydrodynamic models. Currently, the user only feels that he is "at sea" when the ship is in motion. A hydrodynamic model would give the user a better "at sea" feeling even when the ship is not in motion.

Despite its limitations, SHIPSIM is a successful real-time 3D networked application. Future research is required to improve the system's immersive qualities and interface design, add weapons and sound capabilities, and develope a more realistic hydrodynamic model to make it a better simulation tool.

C. DC VET

DC VET is an interactive networked real-time virtual environment for moving a human entity through an entire ship. It was designed to train navy personnel to fight shipboard casualties using standard navy doctrine, and indoctrinate themselves to the objects found inside most navy ships. Basic shipboard familiarization and damage control skills can be learned from the virtual representations of a ship. The damage control trainer is also designed as a networked environment using the computer network communication protocols, which allows multiple people to train together in the same virtual environment.

The concept of "team" is a very important part of damage control training. With the use of network communications, multiple users participate in team damage control exercises within the virtual ship. Additionally, a user can participate actively, or as a silent observer watching others react in the virtual environment. The latter role is ideal for the training instructor or evaluator.

As an example of the type of training that occurs in DCVET, an instructor remotely starts a casualty aboard the virtual ship and observes the trainees fighting the casualty. Upon instructor initiation, a fuel leak occurs in a fuel oil pipe in the engine room. The

trainees are given twenty seconds to shut off fuel flow at a valve, located on the fuel oil pipe, before a fire ignites. If the fire ignites, the trainees must then work together to access a water nozzle and extinguish the fire. During this time, the scene gets steadily darker as smoke from the fire fills the space. Once the fire is extinguished, a vent control switch is accessed to vent the compartment of residual smoke. Other casualties, such as a steam leak, are then initiated by the instructor to further complicate the training scenario.

DCVET is designed to allow a novice user to acquire ship familiarization by allowing him to move about a virtual ship model in a realistic fashion (see Figure 2). By visiting key points of interest within the ship model (such as Combat Information Center, Damage Control Central, or the engine room), the user can later associate these virtual environment spaces with their equivalent real world counterparts. The entire ship virtual environment is comprised of eight multi-level compartments, including a Combat Information Center compartment, Communicaton Shack, Hull Technician Shop, and other miscellaneous spaces. The user navigates throughout the virtual ship to familiarize himself with the ship's compartment organization. In each of these compartments, the user can "grab" or "point to" various objects in the space and is shown textual information to describe what the object is and how it is used. To further assist the user, an autonomous entity is available to walk around the ship showing the user various parts of the ship. The user can either follow the autonomous entity around the ship or watch the autonomous entity move and perform various exercises, such as fighting a fire. The autonomous agent follows a pre-written script. By modifying the script, alternative movements and exercises are demonstrated by the autonomous agent.

DCVET is also implemented similar to NPSNET. It uses the same DIS protocol, simulation loop, and multiprocessing as NPSNET. It also contains a diverse sound system that plays sound, for example, when the user runs into objects, starts a fire, or follows the autonomous entity. Currently, DCVET's fire and steam casualties are not very robust. They are limited to exactly the same place in the engine room. This limits the number of exercises that may be performed on the virtual ship [WEAV95]. Also a real ship contains many more

than eight compartments. More compartments will give the ship better connectivity and allow the active participants more capacity to engage one another.

Despite its limitations, DCVET is a successful real-time 3D networked application. Future research needs to improve the system's database model by including more than just eight compartments, and add more robustness to the fire and steam casualties.

III. SYSTEM OVERVIEW

A. DESIGN PHILOSOPHY

The NPSNET Shipboard Simulator is designed to allow human entities to mount and interact with a ship entity. A ship can be maneuvered and systems controlled by entities other than itself. This enables users to perform shiphandling and fire fighting exercises within a virtual battlefield.

The NPSNET Shipboard Simulator was designed from SHIPSIM and DC VET, two existing NPS simulators [NOBL95][OBYR95]. SHIPSIM enabled users to perform shiphandling exercises in a virtual environment. Users viewed the environment from the bridge and bridge wings of the virtual ship. SHIPSIM provided excellent shiphandling training capabilities, but did not allow for joint operations with networked land and air vehicles. DC VET enabled users to perform internal shipboard walkthroughs and fire fighting exercises in a virtual environment. Users viewed the environment from the perspective of human entities maneuvering about the ship. DC VET provided excellent shipboard training opportunities, but did not allow the ship to move nor allow the human entities to interact with other networked land and air vehicles.

The NPSNET Shipboard Simulator was designed to take advantage of the specific capabilities of both the SHIPSIM and DCVET simulators. First, it allows ship entities to interact with other networked land, air, and water vehicles in a virtual environment. This provides for excellent joint force training opportunities. Second, it allows human entities to mount ship vehicles and remotely control these vehicles in the virtual environment. This provides for excellent shiphandling and shipboard fire fighting training opportunities.

B. NPSNET SHIP ENTITIES

In order to achieve the above design objectives, the NPSNET ship vehicle class had to be modified to accommodate mounted entities and realistic movement characteristics. The previous NPSNET ship vehicle class only allowed the ship to move over water, similar

to the movement of a land vehicle over flat terrain. When the ship moved forwards or backwards, it moved unrealistically, accelerating and decelerating to the requested speeds instantaneously and stopping instantaneously. When the ship turned, it would rapidly change its heading regardless of the amount of headway on the ship. These movements were satisfactory to demonstrate ship movement in a virtual environment, but were not physically based enough to allow the ship to operate as a realistic ship.

The NPSNET ship class has been modified to allow realistic physically based movements for ship vehicles. When the ship accelerates it gradually increases in speed until it reaches the ordered speed. When the ship decelerates, it does so gradually until it reaches the desired speed. When the ship changes course it does so by gradually changing its heading based upon the amount of speed being used. Additionally the class supports multiple engine operations. Thus the user can split his engines to perform twisting maneuvers. These changes give the ship vehicle more physically based movement characteristics.

C. REMOTE CONTROL PANEL

Since the NPSNET ships move in a more physically based fashion, they need a more realistic way to have commands input to them. NPSNET supports many input devices to control vehicles. Yet, most of these do not necessarily match the dynamics of ship vehicles. For example, the keyboard input device allows the user to accelerate and decelerate the driven vehicle, but for multiple engine ships more keys need to be utilized to control each individual engine.

A better solution was to implement a simple graphical user interface (GUI) control panel that matches the unique needs of ship vehicles (see Figure 8). The GUI control panel allows the user to control the ship vehicle in a physically based manner.

D. VOICE RECOGNITION COMMANDS

One of the major flaws with control panels is that they distract the user from the visual simulation. When the user wishes to change course or engine speed, he needs to

remove his focus from the visual display of the scene, and focus his attention on the control panel in order to properly select his command. This causes the user to temporarily lose his sense of immersion in the virtual environment, and may cause him to miss important information within the virtual environment, such as a weapon firing or a fast moving vessel.

Voice recognition helps eliminate this problem. Giving a voice command to the system allows the user to retain his focus on the visual simulation and still control the driven vehicle. The NPSNET Shipboard Simulator combines voice recognition with the control panel to allow the user to control the ship driven vehicle with his voice. By giving the standard commands of Table 2, the user can control his ship vehicle while remaining immersed in the virtual world . Voice command capability also allows the user to wear a head mounted display (HMD) to view the environment and still control his vehicle.

E. SHIPBOARD CASUALTIES

Additional to the movement characteristics, the ship entities were modeled with interior as well as exterior spaces (see Figure 3). The interior spaces contain ramps, ladders, doors, and bulkheads that the user can see once inside the ship. Many of the objects (such as the doors and valves) can be manipulated by the user to enhance the user's belief of being immersed within the ship.

As in DC VET, several shipboard casualties can be started onboard the ship to allow the user to perform damage control exercises. A steam leak (see Figure 4) can be started near the main feed booster pumps. A fuel oil leak (see Figure 5) can be started in the fuel oil pipe. And a fire (see Figure 6) can start near the fuel oil piping if the fuel oil leak is not stopped. Each of these casualties is modelled within a class that describes their behavior. The casualties could be started anywhere on the ship, but have been implemented to start only in the engine room. These casualties allow the user to dynamically interact with the ship and its systems to perform the damage control exercises as in DC VET.

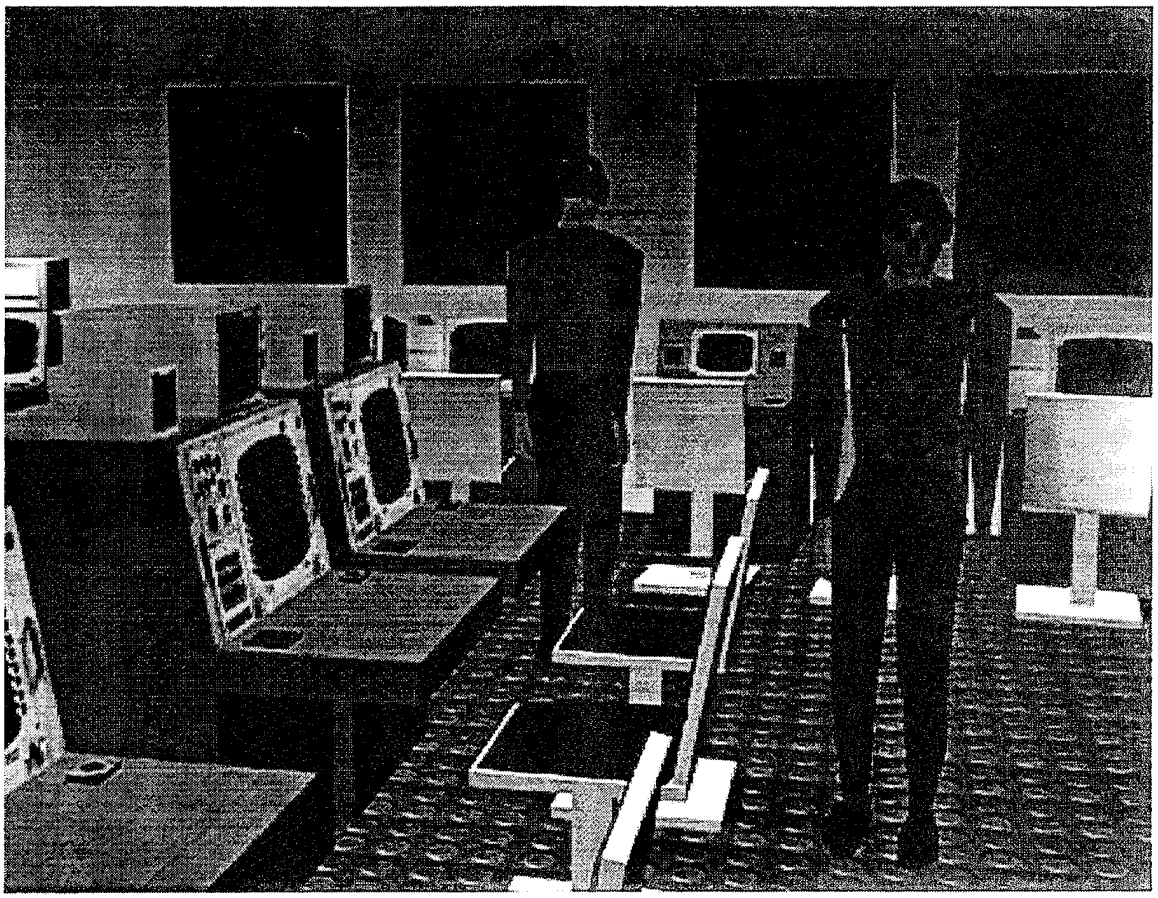


Figure 3: Ship Interior Space (Combat Information Center)

F. NPSNET HUMAN ENTITIES

NPSNET has a rich set of commands for manipulating human entities in the virtual environment. Currently humans can give signals with their hands, kneel down, or lay down in the virtual environment. Yet, because the humans up to this point have been dismounted infantry, they have not had the capability to directly interact with or manipulate other vehicles.

An additional class of human entities was created to deal with the unique aspects of humans working on ships. The NPSNET jack_sailor_veh class utilizes a simple picking mechanism to manipulate objects within the ship. The user needs to select SHIP PICK as the picking mode, and then using either the mouse buttons or buttons on the joysticks (see Appendix A) can manipulate doors, valves, and buttons within the ship.

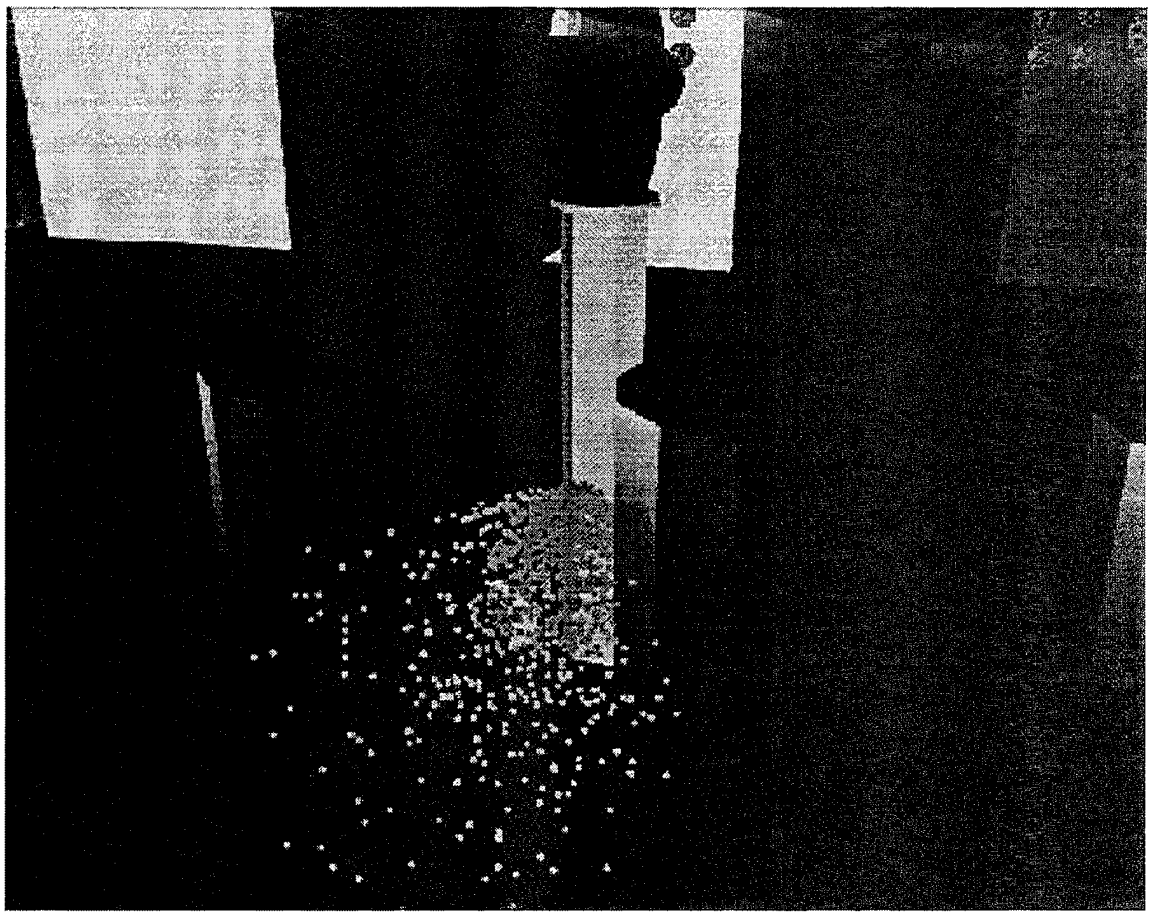


Figure 4: Ship Steam Leak

But before the human entities can manipulate shipboard objects, they need to get within the ship model. As mentioned earlier, human entities up to now have functioned as individual infantry in the virtual environment. Consequently, they have never been placed on anything but the ground or ground equivalents such as building floors or window sills. This thesis research has enhanced the human entity's capabilities by allowing them to mount ship entities. Once mounted, the human entities can freely move about the interior spaces of the ship and manipulate its objects, even while the ship is moving within the virtual environment.

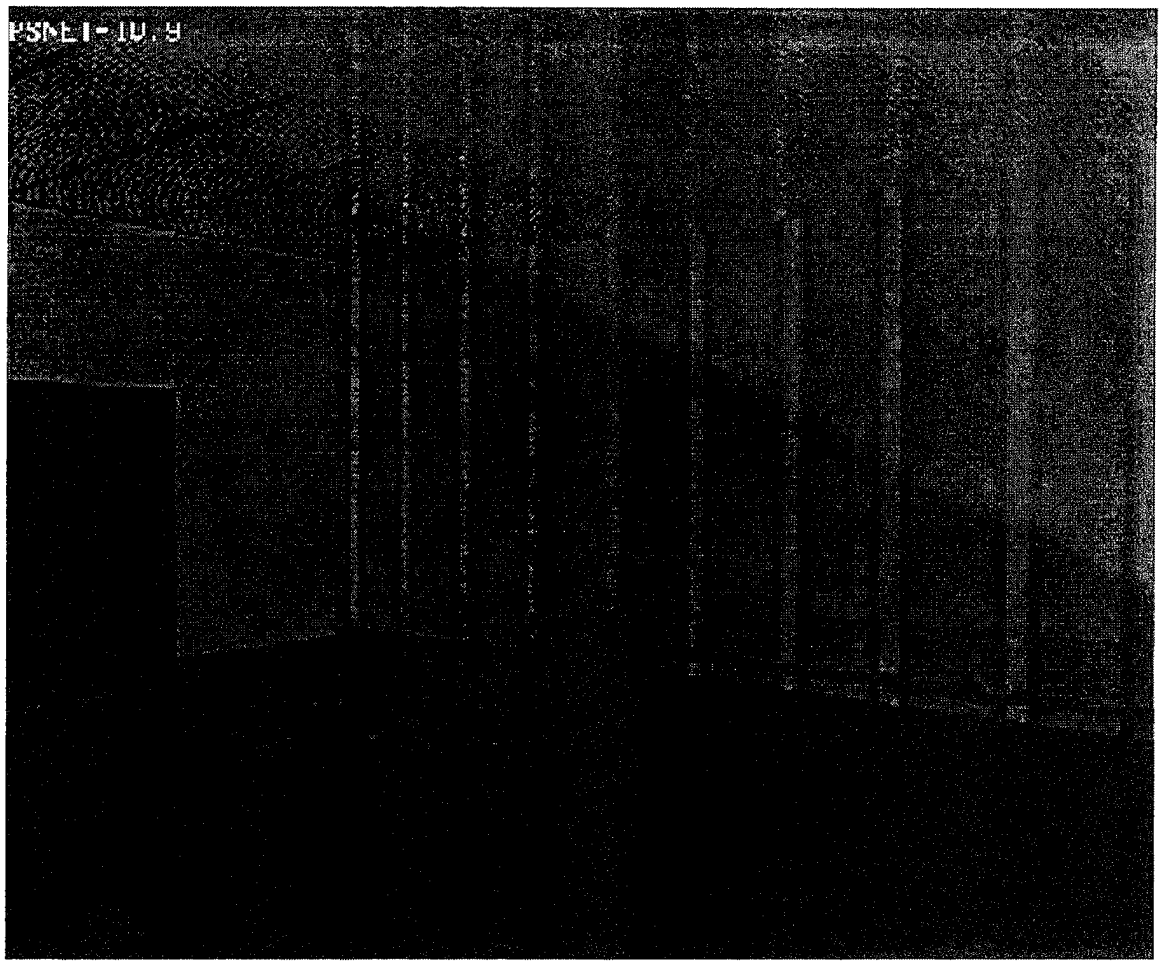


Figure 5: Ship Fuel Oil Leak

G. HIGH RESOLUTION NETWORK

NPSNET uses the DIS standard to perform all of it network activities. The Entity State PDU is the main packet that informs all players of the state of each vehicle in the simulation. Yet, there are many additional bits of information about an entity that need to be communicated that are not in the DIS standard. Hence, NPSNET uses a different network to relay this additional information. The high resolution network (HIRESNET) sends Data PDU packets (see Table 3) between entities on the network. These packets contain the additional information that is not supported by the Entity State PDU packet. If a system participating in the ship's simulation is not listening to the HIRESNET, it will not display and process the detailed information of entities using the HIRESNET.

Figure 6: Ship Engine Room Fire

The ship vehicles use this net to communicate event changes such as starting a fire, or opening a door. The human entities use this net to communicate mounting information. Until the DIS standard or some other protocol can support higher levels of detail for an entity, the HIRESNET will be necessary to handle specific detailed entity information.

IV. CONTROL PANEL AND VOICE RECOGNITION

A. CONTROL PANEL

Since the NPSNET ships move in a more physically based fashion, they need a more realistic way to have commands input to them. NPSNET supports many input devices to control vehicles. Yet, most of these do not necessarily match the dynamics of ship vehicles. For example, the keyboard input device allows the user to accelerate and decelerate the driven vehicle, but for multiple engine ships more keys need to be utilized to control each individual engine.

A better solution is to implement a simple graphical user interface (GUI) control panel that matches the unique needs of ship vehicles. The GUI control panel allows the user to control the ship vehicle in a physically based manner.

1. SHIPSIM'S Control Panel

SHIPSIM created such a GUI as part of its simulation source code (see Figure 7). The GUI was created from Silicon Graphics Performer Utilities and consists mostly of push buttons and sliders. Using the "SHAFT RPM" sliders the user can advance a ship forward or backwards. By toggling the "COMB SHAFT" button the user can split the engines of the ship in order to twist the ship in place or aid the ship in completing a turn. The "RUDDER ANGLE" slider changes the rudder to port or starboard and causes the ship to turn left or right. Additional buttons (such as "QUIT", "PLAYBACK", and "RESET ALL") control the state of the simulation. The environmental buttons ("FOG", "TIME OF DAY") change the environmental characteristics of the simulation. And finally, the view point sliders and buttons change the users view point on or off the ship[NOBL95].

These buttons and sliders give the user adequate control of the ship entities and the simulation, but the GUI as a whole has several flaws with its construction. First of all, it takes up the lower third of the screen. This leaves a smaller observation window through which to view the scene. Also the GUI is fixed in place on the screen, leaving very little

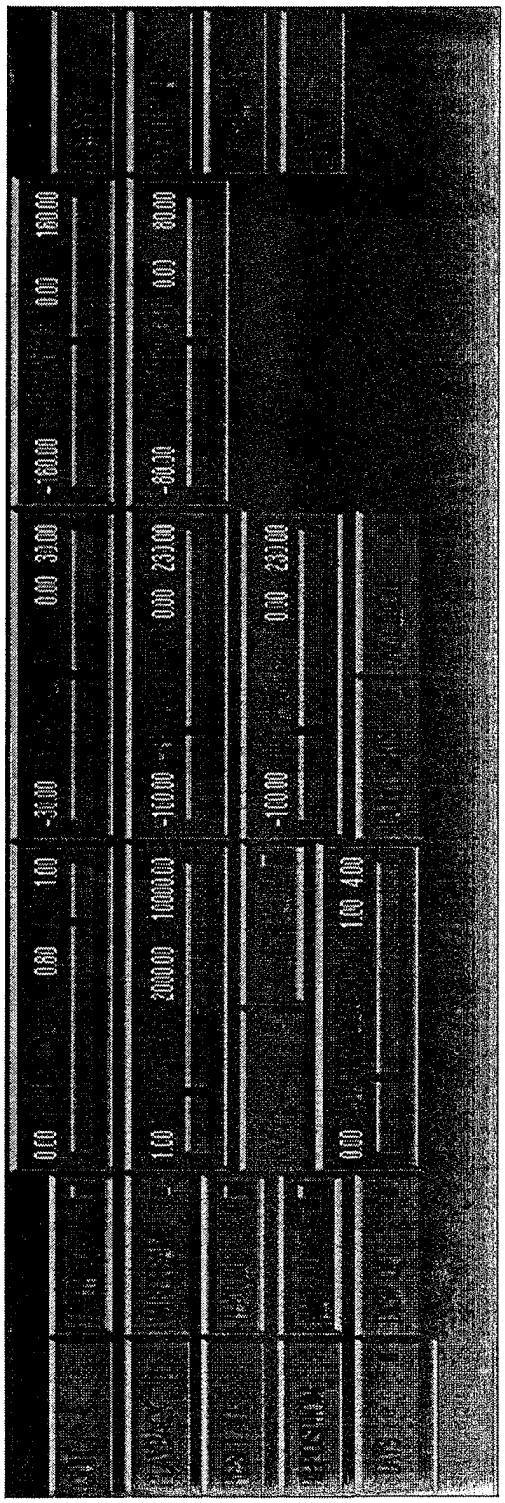


Figure 7: SHIPSIM GUI Panel

user flexibility as to its location on the screen.

Secondly, the sliders make it very difficult to accurately select a desired setting. Each slider is given a range of values which can be selected, and a fixed number of positions that the slider can position itself in [SGIB94]. Consequently, the slider's range is divided into fixed intervals, making it difficult for the user to accurately position the slider into a desired setting. For example, the "VIEW ANGLE" slider has a range between -160 to 160 degrees. This slider determines the users view offset while on the ship. Because the slider only gives fixed interval values, when the user moves the slider to different positions the scene radically jumps between frames. This makes it very difficult to fix the view point onto an object in the scene, which results in the user losing his sense of immersion into the scene.

Lastly, the GUI makes it very difficult to select specific engine speeds. Similar to the "VIEW ANGLE" slider, the "SHAFT RPM" sliders have the same problem when it comes to specifying an exact value for the engine's speed. But additionally, the user must try to dial in the appropriate rpms to obtain the desired speed. This constant trial and error can once again cause the user to fall out of a simulation, or worse yet develope an unrealistic feel for maneuvering by constantly selecting the easy to reach extreme values. All of these flaws make the SHIPSIM GUI less effective in terms of controlling a ship in a virtual environment.

2. NPSNET'S Ship Control Panel

To correct these flaws, a new control panel was created for NPSNET (see Figure 8) . The control panel was created using Developer Magic's RapidApp application builder [SGIC94]. This GUI development environment contains many more widgets (such as dial knobs, input text boxes, and radio buttons) with which to build a GUI panel. Consequently the panel was built to look and feel more like a ship's helm console. Radio buttons are used to delineate engine bells associated with standard speeds. Also the rpms can be exactly set using the rpm selection buttons. Finally, a dial knob, which has more precision than a slider,

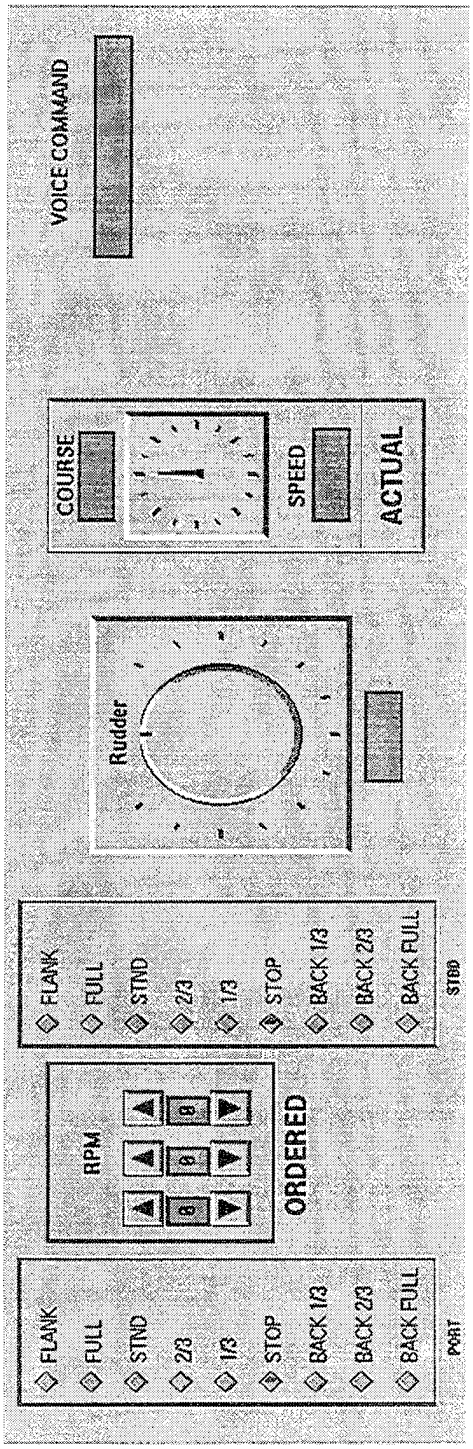


Figure 8: NPSNET's Ship Control Panel

is used as the rudder controller. All of the ship control functionality of SHIPSIM's GUI panel are contained within this panel, but the new widgets are much more intuitive to use for controlling a ship.

Because NPSNET was an already existing vehicle simulator, many of the environmental, and vehicle view point controls were already present. Consequently, the new ship control panel did not need a mechanism to change the user's view point, control environmental factors, or control the simulation display. The new control panel runs as a separate program from NPSNET. Thus it can be placed anywhere on the screen, resized, or even hidden during the simulation. This gives the user much more flexibility in placing the panel on the screen, and also the panel need only take up as much of the screen as the user desires.

In order to communicate with NPSNET to control a ship vehicle, a network protocol was established. The control panel communicates with NPSNET via a multicast port using a special PDU packet known as an Information PDU (IDU). The IDU packet contains simple header information (see Figure 9) and a defined data structure. The ship

```
#define NPSNET_To_SHIP_Type              (IDUType)107
#define SHIP_Ood_To_NPSNET_Type          (IDUType)108

typedef struct {
   unsigned char        version;
   IDUType              type;
   unsigned short       length;
} IDUHeader;
```

Figure 9: IDU Packet Header

control panel uses two different data structures to communicate with NPSNET (see Figure 10). The first structure is used to relay speed and heading information from the NPSNET ship vehicle to the control panel for display. The second structure is used to pass the desired orders from the control panel to the ship. With this network configuration, the control panel is able to maneuver a ship vehicle in a NPSNET virtual environment.

```
typedef struct {
   IDUHeader    header;
   //the following field are used to communicate data
   //from NPSNET to SHIPCONTROL
  float ship_course;   //actual ,000
  float ship_speed;    //actual ,00
  float ship_rudder_angle;    //actual ,00

  float ship_ordered_rudder_angle; // 000
  float ship_ordered_rpm;          // 000
  float ship_ordered_port_bell;//-3 - 5 (back full - flank)
  float ship_ordered_stbd_bell;//-3 - 5 (back full - flank)

  u_long space_holder;
} NPSNETToShipIDU;

typedef struct {
   IDUHeader    header;
   float ood_rudder_angle;  // ordered
   float ood_rpm;           // 000
   float ood_port_bell;     //-3 - 5 (back full - flank)
   float ood_stbd_bell;     //-3 - 5 (back full - flank)
   u_long space_holder;
} OodToNPSNETIDU;
```

Figure 10: IDU Packet Structures

B. VOICE RECOGNITION

One of the major flaws with control panels is that they distract the user from the visual simulation. When the user wishes to change course or engine speed, he needs to remove his focus from the visual display of the scene, and focus his attention on the control panel in order to properly select his command. This causes the user to temporarily lose his sense of immersion in the virtual environment, and may cause him to miss important information within the virtual environment, such as a weapon firing or a fast moving vessel. Voice recognition helps eliminate this problem. Giving a voice command to a system allows the user to retain his focus on the visual simulation and still control a vehicle. NPSNET's ship control panel can be used with voice recognition to allow the user to control a ship vehicle.

1. Voice Recognition Background

Current research in the field of voice recognition has produced three types of voice recognition systems: dictation, control, and continuous speech. Dictation systems are designed to recognize natural language and translate it into text. IBM's Voice Dictation System is one such system. Control systems are designed to recognize phrases and perform specified functions. SGI's Speech Manager is one such system. Continuous Speech systems are designed to interpret meaning from one's speech and then perform specified tasks. SRI's Nuance is an example of such a system. Of the three, continuous speech recognition is the most difficult to achieve, given the complexities of modern languages. Dictation and control recognition are the easiest to achieve, due to the fact that only a limited vocabulary of single words or short phrases need be recognized. Because ship handling commands are short words and phrases (i.e. "Right, Full Rudder"), a control voice recognition system is sufficient to manage the voice commands for the ship control panel.

In addition to the type of voice recognition system, the voice recognition field has also produced performance parameters for systems. In MIT's study on voice control systems for military applications [PICO95], they recognized that the performance parameters speaker independence, recognition rate, latency, and noise sensitivity were the most significant for a control voice recognition system. Given that SGI's Speech Manager program [SGID95] was the only control voice recognition system available, the above performance parameters within the Speech Manager were examined to determine if another system needed to be purchased or developed for the ship control panel.

A control voice recognition system should be speaker independent and recognize native English of any pitch or intonation, preferably without any pre-trial training sessions. The Speech Manager requires the user to train it on a vocabulary either prior to initial usage or when it does not properly recognize certain words in the vocabulary. Thus if the vocabulary is very distinct and easily discernible, each new user may only need to train the system on a few words instead of the entire vocabulary.

A control voice recognition system should exhibit a high recognition rate, above 80% [PICO95], and not reject very many misinterpreted commands. The Speech Manager has the capability of allowing the user to selectively decide whether it should be very discerning in finding a match, or whether it should accept the first logical match. Thus the user can tune the Speech Manager to accommodate specific application recognition rate requirements.

A control voice recognition system should not have a long (above 2 seconds) latency time to process voice commands. The Speech Manager gives the user the ability to select the maximum amount of time the system should spend when processing inputs, and thus control the latency requirements for a specific application.

Finally, a control voice recognition system should be insensitive to background noise. The Speech Manager documentation recommends that the user use a directional stereo microphone to input commands to the system. This type of microphone helps to eliminate much of the background noise that would interfere with the recognition process. Also, the user can de-select certain frequency ranges from the system's frequency spectrum to help eliminate the background noises. After examining the above specified performance parameters, it was determined that the Speech Manager is an adequate control voice recognition system for the ship control panel.

2. Control Panel Voice Command Interface

The ship control panel was design specifically to operate with the Speech Manager. The Speech manager was designed to act as an alternative method for manipulating SGI's X windows and executing unix commands. Each window can have an associated vocabulary attached to it that the Speech Manager will search through when the user selects that window with the mouse cursor and issues a voice pattern. After recognizing the voice pattern as being within the window's vocabulary, the Speech Manager gives an audio queue, displays a video message (see Figure 11), and either executes a unix command or places characters in the keyboard input stream. The keyboard inputs are equivalent to

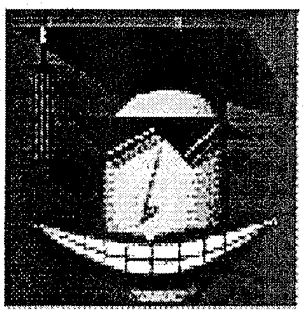


Figure 11: Speech Manager Video Message

having typed those characters from the keyboard, and thus if the application running in the selected window had a command associated with the inputted characters, the command would subsequently be executed.

Given this interface, the control panel has a single line text input box (see Figure 8) that receives character inputs and interprets those characters into commands (see Table 2). These commands are then displayed in the GUI's widgets and sent to the NPSNET ship

Command	Code	Command	Code
RIGHT	r	AHEAD ONE THIRD	m
LEFT	l	AHEAD TWO THIRDS	w
FULL RUDDER	d	AHEAD STANDARD	j
STANDARD RUDDER	c	AHEAD FULL	y
10 DEGREES RUDDER	b	AHEAD FLANK	v
5 DEGREES RUDDER	a	INDICATE	i
HARD RIGHT RUDDER	e	ZERO	0
HARD LEFT RUDDER	f	ONE	1
RUDDER AMIDSHIPS	h	TWO	2
SHIFT YOUR RUDDER	g	THREE	3

Table 2: Voice Command To Control Panel Input Syntax

Command	Code	Command	Code
ALL	o	FOUR	4
PORT	p	FIVE	5
STARBOARD	s	SIX	6
BACK FULL	z	SEVEN	7
BACK TWO THIRDS	x	EIGHT	8
BACK ONE THIRD	n	NINE	9
STOP	u		

Table 2: Voice Command To Control Panel Input Syntax

vehicle. The command syntax describes the grammar for how voice commands are to be stated. It was designed to be as similar to continuous speech commands as possible and still be accurately discernible by the Speech Manager system. For example, a continuous speech turn command would be given as "RIGHT FULL RUDDER." To the Speech Manager this may look very similar to "LEFT FULL RUDDER." So to increase the system's recognition of these commands, they were broken up as first the direction of turning, "LEFT" or "RIGHT," followed by the amount of turn "FULL RUDDER". This scheme proved to make the Speech Manager recognize all of the ship control panel's vocabulary with a very high success rate, 90% or above.

C. SUMMARY

The ship control panel was built to allow mounted human entities to remotely control the ship that they were mounted upon. Previously, all NPSNET vehicles were controlled directly by an input device (such as a keyboard or flight control joystick). The control panel allows one user to directly control a human entity and remotely control a ship entity in the virtual environment on the same workstation. The control panel also recognizes voice commands, allowing the user to control a ship vehicle and remain visually immersed in the virtual environment. However, the voice recognition system does have a

few draw backs. First, Speech Manager is not completely speaker independent. The user should not be required to train the system on a fixed vocabulary. Secondly, Speech Manager only gives the user two ways to interact with the speech program, either by a unix command or a character sequence. A better means of interaction would be to provide the user with a programming interface, or a set of system calls that could be compiled into the source code of the control panel. This would allow the control panel to be completely portable to other systems. With future research, a better control voice recognition system can be found to compensate for the Speech manager's deficiencies.

V. POTENTIALLY VISIBLE SETS

A. BACKGROUND

One of the many problems with implementing large scale real time virtual environments has been how to efficiently display the environment at a frame rate which the human eye perceives as real time. In order for the human eye to perceive real time, the virtual environment must be displayed at least at frame rates of 8 to 10 frames per second[NATI94]. One of the elements that slows down frame rates is the number of polygons that have to be rendered to the screen. The more polygons in the scene the longer it takes to render a frame. One way to reduce the number of polygons that have to be rendered each frame is to cull out those polygons that are not within the view frustrum. This helps get rid of most polygons, but there still may be polygons that are obscured by other polygons closer to the view point, such as polygons that are behind a polygon representing the wall of a room. With culling techniques, the draw process is freed of drawing unnecessary polygons, but that does not necessarily mean the frame rate will increase. The time taken to cull the scene may actually be more expensive then drawing the polygons in the first place. Thus another technique had to be developed.

Airey in his Ph.D. thesis at the University of North Carolina, at Chapel Hill presented a potentially visible set (PVS) algorithm that helps the culling process by trimming the database of non-visible polygons each frame [AIRE90]. The database model is broken down into spatial volumes or "cells". Each cell may have many hundreds or thousands of polygons within it. From within each cell, only a small subset of all cells should be visible at a given time. For example, in a model of a house, only adjacent rooms and hallways are visible to one another (see Figure 12). The cells within the subset are all potentially visible and should be considered for drawing. All the other cells can be ignored and should not be sent to the culling process. With the model broken down into cells, and subsets of potentially visible cells created for each cell, only the visible cells set provided to the culling process need be updated each frame as the view point changes. If the culling

process only processes the potentially visible cell set instead of the entire model, the frame rate should be maintained at a higher rate, providing the PVS algorithm to maintain the visible cells set is not expensive.

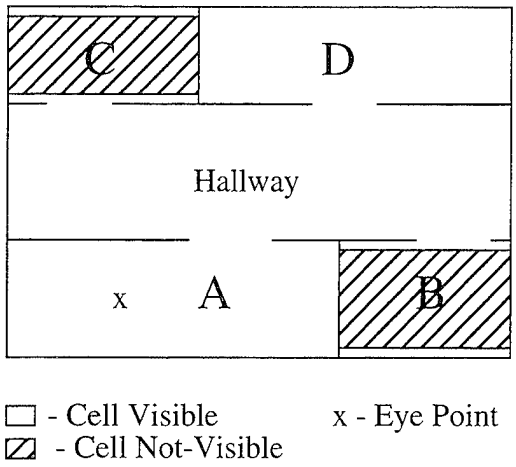


Figure 12: House Model with Visible & Non-Visible Cells

B. DC VET'S IMPLEMENTATION

DC VET uses a PVS algorithm to render the interior of its ship model. The "antares" ship model contains many thousands of polygons that represent the objects within the ship, such as walls, ceilings, floors, decks, chairs, and machinery. Without the PVS algorithm, certain views of the model (i.e. standing on the bridge looking back at the rest of the ship) were rendered at frame rates of 1 to 2 frames per second. With the PVS algorithm, the same views of the model were rendered at frame rates of 20 to 30 frames per second. DC VET's implementation depends upon the manipulation of a simple list of cells and the ship's scene graph.

Initially, the ship model is broken up into discrete cells (such as engine room lower level, bridge, damage control central, etc.) and each cell is numbered. Then each cell is provided with a list of all the potentially visible cells to itself. During each frame the view point is compared to the bounding volume of the current cell. If the viewpoint falls within the current cell's bounding volume, then the algorithm is exited with no changes. If not,

each cell in the model is queried until the viewpoint falls within the bounding volume of a cell. At that point, the potentially visible cells to this new found cell are added to a visible cells list, and all the previously visible cells are removed from the list. Additionally, the nodes in the ship's scene graph corresponding to the visible cells are placed in the frame's scene graph, and those not visible are removed from the frame's scene graph. The algorithm is then exited until the next frame.

DC VET's implementation performs extremely well. It renders a 24,000 polygon ship model at frame rates of 20 to 30 frames per second versus frame rates of 5 to 6 frames per second without the PVS algorithm [MCDO95]. However, the algorithm is put together with a "brute force" mentality. The system must store the bounding volumes and potentially visible cells for each cell in an array. This information is loaded during the system's start-up phase, and must be updated if the ship's database changes. Also, the implementation must modify the scene database when a cell change is required. For a database with a small number of cells, this is not a problem, but for a database with hundreds or thousands of cells, this can be rather expensive. However, despite its clumsiness, the implementation is efficient and does not waste time when a cell change is not required. The largest time loss occurs during a cell change. But since this occurs in one frame, a noticeable pause may only occur if the model had a large number of cells to swap in and out.

C. UNC'S PF_PORTAL IMPLEMENTATION

The University of North Carolina, Chapel Hill, developed a different methodology for implementing the PVS algorithm in their walkthrough project [LUEA95]. They utilized portals to change the view into the database model. Portals are windows and doors through which one can see into and out of the database model. Their idea was to improve the initial breakdown of cells within a database by attaching portal references to each cell. The potentially visible cells are only those cells that are visible through the portals [LUEA95]. For example, in Figure 13 the user is within cell A. If his view direction were towards

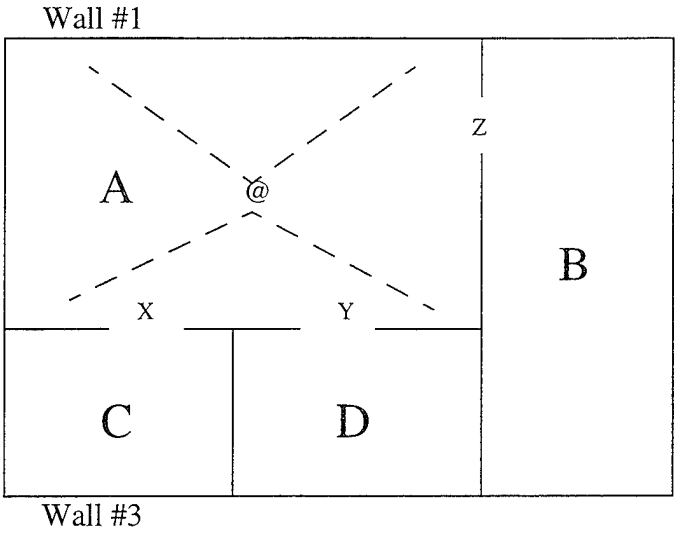


Figure 13: Cells Visible from Portals

wall #1, none of the portals within cell A are visible, so there is no need to render cells B, C and D. If on the other hand, the view direction were towards wall #3, then the cells behind portals X and Y are visible and should be rendered, but the cell behind portal Z is not visible and should not be rendered. DC VET would have rendered all of the cells in this figure because they are all potentially visible from cell A. Thus, UNC's algorithm should save rendering time when compared to DC VET.

UNC implemented their methodology by modifying Performer's cull and draw mechanisms to use their pfPortal structures [LUEB95]. Each cell in the database has a list of all its portals and a reference to each cell behind the portals. Each frame, the current view point is checked to make sure it is within the bounding volume of the current cell, and a similar swap mechanism is performed, as with DC VET, if the view point is not within the current cell. Unlike DC VET, after the view point location is checked, the UNC implementation checks the view frustrum to see if any portals are inside of it. If none are found, then all the potentially visible cells of the current cell are not rendered. On the other hand, if one or more are found, then their appropriate cells are rendered.

UNC tested their methodology on a walkthrough model composed of 367,000 radiositized triangles. They achieved in certain portions of their model, frame rate increases

of up to 10 times that of the entire model unculled [LUEB95]. Yet, they discovered that their approach was extremely dependent on the model and the current view path. Frame rates dropped when looking down long hallways for example, due to the high numbers of portals that were visible. On models that are well segmented with very few portals being visible at any one time, their algorithm performed very well.

D. NPSNET'S IMPLEMENTATION

NPSNET's ship project implements the PVS algorithm to render a very large interior ship model (24,000 polygons). However, unlike the other two projects, NPSNET's ship is just one vehicle in an entire scene containing terrain, static objects and other vehicles. Consequently, NPSNET's implementation requirements are more restrictive than DC VET or UNC's walkthrough project.

One problem with DC VET's implementation is that it requires the database to be dynamically changed by the algorithm. DC VET places cells that are visible in the scene graph, and maintains a pointer array to all the other cells in the model. When necessary, DC VET dynamically removes and adds cells to and from the scene graph. NPSNET vehicles are added to the scene graph when the vehicle is created (either as the driven vehicle, or a networked vehicle). Once created, NPSNET requires that the database nodes of the model not change through out the simulation. Consequently, DC VET's approach to swapping cells in and out of the scene graph is not acceptable for NPSNET. To get around this restriction, NPSNET's implementation modifies the ship database by adding a Performer switch node to the highest database node of each cell in the database (see Figure 14). The switch nodes act as gateways to direct Performer's database traversal mechanism during the cull and draw processes. By turning the switch of visible cells to the "ON" position and turning the switch of occluded cells to the "OFF" position, swapping cells in and out for rendering is achieved.

One of the problems with UNC's pfPortal implementation is that the portals are represented by 2D orthogonal projections that do not match NPSNET's ship model. The

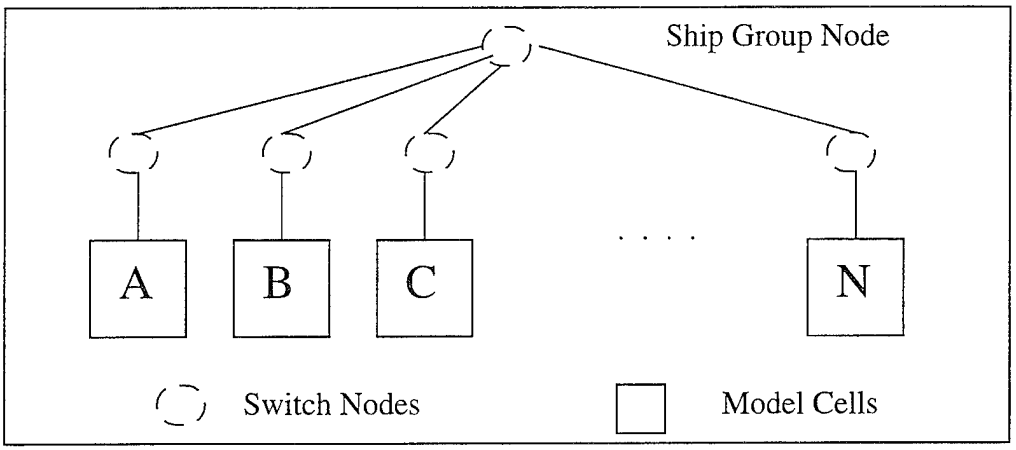


Figure 14: Database Structure with Switch Nodes

portals do not shrink and grow as the 3D NPSNET ship model's shrink and grow when the view point moves closer or farther from the 3D model. The 2D portals remain a fixed size as the view point changes. Consequently, when tested, the portals drifted across the screen and did not properly display the cells behind them at the correct time. There were times when a 3D model door object was visible but the 2D projection portal for the door was not. Additionally, UNC implemented their algorithm to operate on a single processor workstation, and it did not function properly on a multiprocessor workstation. Consequently, UNC's implementation using portals could not be used in NPSNET.

However, UNC's implementation did take advantage of Performer's cull and draw callback mechanism. By attaching the visible cell data to the model nodes, UNC's algorithm is performed primarily by the cull and draw processes leaving the application process free to do other things. NPSNET's implementation uses Performer's data attachment feature like UNC's implementation, but still performs the algorithm in the application process. NPSNET vehicles unfortunately already use the cull and draw callback mechanism for other purposes (i.e. terrain masking, network vehicle identification).

NPSNET's PVS algorithm is implemented similarly to DC VET. Cells are pre-determined and swapped in and out of the scene graph as necessary. But NPSNET performs the swapping sequence with Performer's switch node instead of adding and removing nodes. NPSNET pre-loads the potentially visible cell list data into the appropriate cell

nodes of the model. Then each frame the view point or driven vehicle position is checked against the bounding volume of the current cell. If the point is within the cell the routine is exited. If not, a list of all the cells in the model is searched until a cell is found that bounds the view point. If no cell is found, the exterior of the ship is displayed; otherwise the potentially visible cells are swapped into the renderable model by turning their switches on and turning all others off. By attaching the potentially visible data to the nodes (as in UNC's implementation), an extra array traversal is avoided when looking up which cells are visible to the current cell.

NPSNET's implementation successfully meets the restrictions of its vehicle models and still achieves real time frame rates. While on the inside of the ship model, frame rates are comparable to that of DC VET (20 to 30 frames per second), and while on the exterior of the ship, frame rates are maintained similar to other vehicle models (25 to 30 frames per second).

E. SUMMARY

In order to successfully mount humans to ship vehicles, the rendering of ship models that are mountable must not reduce the frame rate of the system below that of real time imaging. Consequently, a modified DC VET PVS algorithm is implemented in NPSNET for ship vehicles to successfully meet the frame rate criteria. Although UNC's pfPortal methodology could not be implemented at this time, due to its incompatibility with NPSNET's 3D models and its inability to run on multiprocessor workstations, it does have the best potential for maximizing frame rates in the future.

VI. MOUNTING OF HUMAN ENTITIES

A. BACKGROUND

Up until now, all entities in the virtual environment were individual objects that moved through the virtual world terrain model. Depending on the application, the terrain may be as large as an entire country, such as the terrain models used in NPSNET, or as small as a person's house, such as with UNC's walkthrough project. Each application may contain different types of entities, but all the entities are independent of one another.

In a typical DIS simulation, entities communicate with one another via a network over which they pass entity state information about themselves. The state information contains the position, orientation and other information about the entity (see Chapter 2). An entity's position in the virtual environment is referenced to an agreed upon world coordinate in the terrain database. A collision takes place between two entities when their representative polygon geometries intersect one another. When two entities collided with one another, a colllisonPDU packet is sent to all participating players in the simulation informing them of the results of the collision.

Mounting consists of one entity's polygon geometry intersecting the polygon geometry of another entity. Once mounted, the mounting vehicle changes its coordinate reference point to be that of some point within the mounted vehicle's bounding volume instead of the world coordinate reference point. While mounted, the mounting vehicle may freely move within the mounted vehicle's database as it would over the terrain database. When the mounted vehicle moves in the world coordinate system, the mounting vehicle moves as an articulated part attached to the mounted vehicle. Using the current DIS network protocol, mounting as defined above would constitute a collision between the mounting entity and the mounted entity. How then do we solve the problem of mounting entities to one another?

In a non-networked simulation, the mounting problem is easily solved (see Figure 15). The user's vehicle movement matrix is pushed onto the matrix stack after the

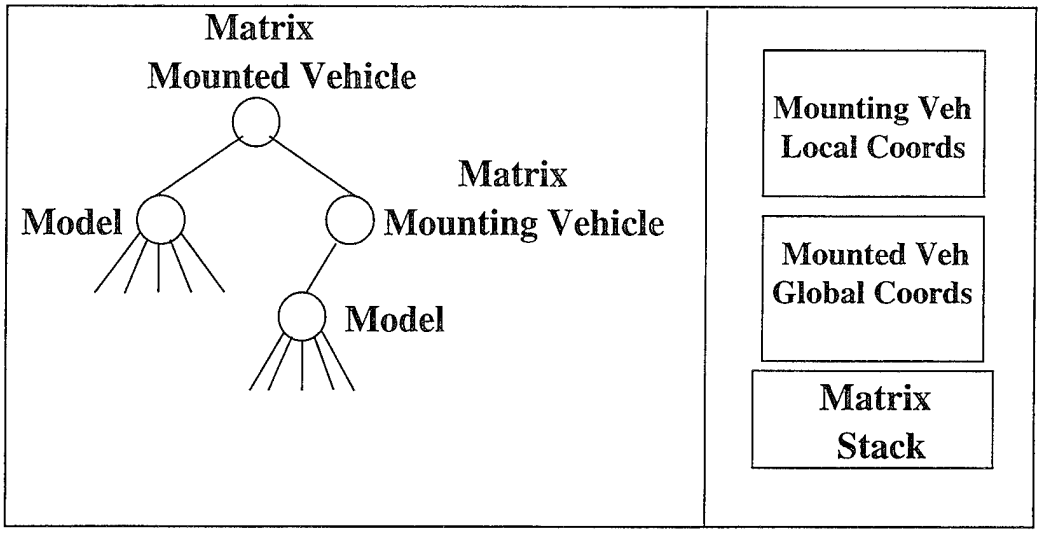


Figure 15: Non-Network Mounting Solution

mounted vehicle's movement matrix, and the user's vehicle matrix is set to a local coordinate within the mounted vehicle's bounding volume. In a networked simulation, however, the mounting problem becomes more difficult. When you change the coordinate points of the mounting vehicle to local coordinates, the networked ghost entities of the mounting vehicle jump to the wrong coordinates in the virtual world (see Figure 16)

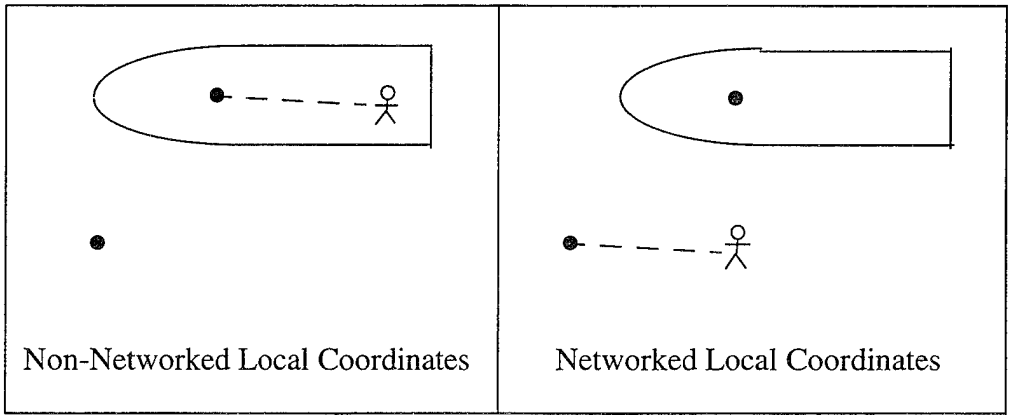


Figure 16: Problem w/ Local Coordinates in Network Solution

because the ghost entities are interpreting the local coordinate values as global coordinates.

The simplest solution to mounting entities across a network is to have your network protocol support both global and local coordinate positions. When a vehicle mounted another vehicle, the local coordinate positions would be sent to the networked

ghost entities and used as in the non-networked case. In the absence of such a protocol, an alternate means of mounting vehicles is to change the velocity vector of the mounting vehicle (see Figure 17). When mounted, the vehicle maintains its local velocity, and then

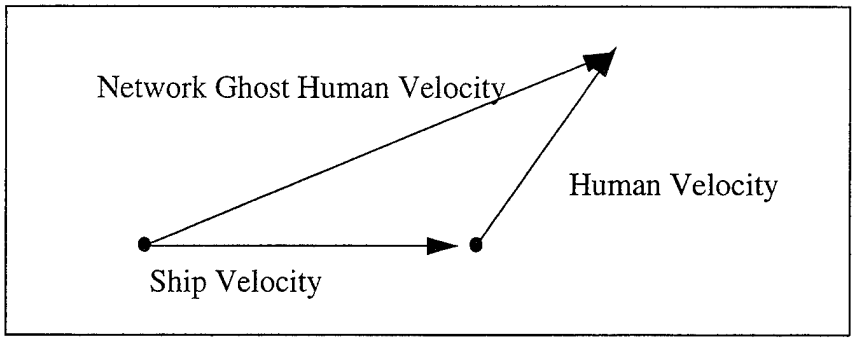


Figure 17: Mounting Using Velocity Vectors

sends the sum of its velocity and the mounted vehicle's velocity to the networked ghost entities. This solution is satisfactory providing the positions of all vehicles are determined by their velocity vector, and providing what the vehicle looks like in the scene (i.e. walking, running, etc.) is not determined by the velocity vector.

B. MOUNTING ALGORITHM

NPSNET cannot utilize any of the above methods of solving the network mounting problem to mount human entities. NPSNET utilizes the DIS network protocol (see Chapter 2) which does not support local coordinate positions. Additionally, NPSNET calculates the position and appearance (i.e. walking, running, etc.) of all its networked vehicles based upon the velocity of the vehicle. Thus the above solutions cannot be utilized to solve the mounting problem. To handle this problem, NPSNET uses a method similar to the non-networked solution to mount its human entities.

When a human entity intersects with a ship vehicle, it mounts the ship by storing the entity state id of the ship and setting a relative position vector. The relative position vector is the difference between the global position vectors of the ship and the human, and represents the human's local reference point to the ship. Each frame while mounted, the human entity first updates its position in the virtual world as if the ship entity were a static

object. Collisions with the ship's objects and decks are handled before determining the intended position of the human entity. After finalizing its intended position, the relative position vector is updated from the new position. After the mounted ship entity has completed its movement for the frame, the human entity's final global coordinate position is updated based on the relative position vector and the ship's new global coordinate position and orientation (see Figure 18). To determine the final position, the relative

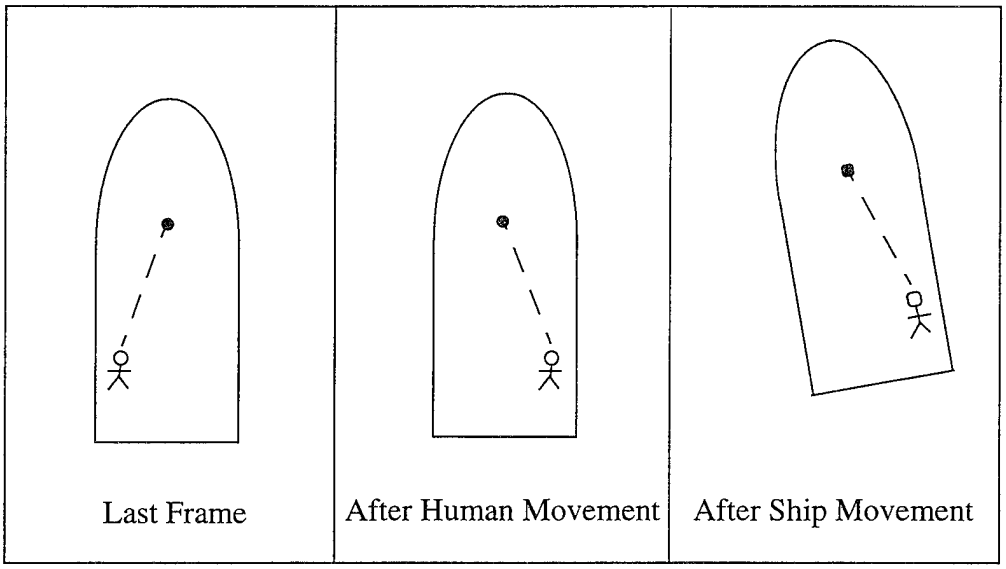


Figure 18: Sequence for Network Mounting

position vector is first normalized, then rotated based upon the change in orientation of the mounted ship vehicle, and finally scaled by the length of the original relative position vector. This new relative position vector is then added to the position of the mounted ship vehicle to obtain the human entity's final position (see Figure 18). With this solution, a human entity can mount a networked ship entity.

Yet, the networked ghost human entities are still a problem. Because they do not receive an entity state packet every frame, they are constantly out of position. For example, if the ship were moving at a speed of 5 knots and the human entity were standing still, the human entity would only send a heart beat packet every few seconds to the networked ghost human entities. Between heart beat packets, the ghost entities would remain in the same global coordinate because their speed is zero, but the human entity is actually moving at 5

knots due to being mounted on the ship. Thus the ghost entities are constantly jumping to their new positions in an unrealistic motion when any new packet is received. These ghost entities would need to receive a new packet at frame rate in order to move with the ship. Using the network in this fashion wastes network band width and with 20 human entities mounted to the ship, the network would quickly become saturated. To solve the mounting problem for the ghost human entities, a method had to be developed to inform the ghost entities that they should be mounted to the ship.

C. HIGH RESOLUTION NETWORK

NPSNET utilizes a different network to send non-DIS compliant information packets to entities in its simulation. This network is known as the high resolution network (HIRESNET). The HIRESNET sends DIS Data PDU packets (see Table 3) through a different multicast port. These packets contain special entity information that is not

Field Size (bits)	HIRESNET Data PDU Fields	
96	PDU Header	the information needed by any receiving node to properly characterize and operate on the data being received
48	Originating Entity ID	ID and force of originating entity
8	Originating Force ID	
48	Receiving Entity ID	ID and force of receiving entity
8	Receiving Force ID	
8	Request ID	Type of data in datum records
8	Num_datum_fixed	Total number of Datum Records
n*32	Fixed_datum	Datum Records
Total PDU size = (224 + 32n) bits where n=number of fixed datum records		

Table 3: HIRESNET PDU Structure

contained in the entity state PDU. NPSNET's human entities utilize the HIRESNET to send specific human arm joint positions across the network. This information is more detailed

Field Size (bits)	Entity State PDU Fields	
96	PDU Header	the information needed by any receiving node to properly characterize and operate on the data being received
48	Entity ID	information about the simulation application and the entity generation data carried in the PDU
8	Force ID	
8	# of Articulation Parameters(n)	
64	Entity Type	
64	Alternative Entity Type	
96	Entity Linear Velocity	information about the current state of the entity. These records provide the position, rate of movement, orientation and parameters needed by dead reckoning algorithms which describe the entity originating the information
192	Entity Location	
96	Entity Orientation	
32	Entity Appearance	
320	Dead Reckoning Parameters	
96	Entity Marking	
32	Capabilities	
n*128	Articulation Parameters	Information about articulated parts which are attached to the platform being represented
Total Entity State PDU size = (1152 + 128n) bits where n=number of articulations		

Table 4: Entity State PDU Structure

then the articulations in the entity state PDU (see Table 4). During an NPSNET simulation, only those simulators capable of reading the HIRESNET will see the extra information.

Those simulators who choose not to read the HIRESNET will continue to get entity state PDU's indicating where the human entities are, but will not see them moving their arms and upper body.

Since DIS does not support local coordinate positioning, the HIRESNET was chosen to relay mounting information to the ghost human entities. Using a special mounting identification tag, the data PDU packet sent by the mounting vehicle places the entity id of the mounted vehicle in the receive entity id field of the packet. When the mounting entity dismounts it sends the data PDU packet with the mounting tag, but does not include an entity id in the received field. Hence, the mounting vehicle can relay to its ghost human entities when it mounts and dismounts a ship vehicle. Once a ghost entity receives the mounting data PDU on the HIRESNET, it mounts the ship by storing the entity state id of the ship and setting a relative position vector. Then each frame, the ghost entity moves in relation to the mounted ship entity as the actual human entity does utilizing its dead reckoning position.

This solution is not ideal because it relies on all the other simulators being able to read the HIRESNET. But, until the DIS standard supports local coordinate positioning, this is an acceptable alternative to get entities to mount other entities in a networked simulation.

D. SHIP BOARD CASUALTIES

Now that human entities can mount ship entities, they should be able to interact with the objects within the ship model. To incorporate this capability, DC VET's shipboard casualties and object manipulations were implemented in NPSNET's ship model. The functionality of the casualties remain the same as in DC VET (i.e. the casualties are the same type and in the same locations), but the method of implementation is different in NPSNET.

First of all, DC VET treats the ship model as the terrain of the simulation. Consequently, its casualties are controlled with references to global variables. The casualty object models, such as the fire fighting nozzle and fuel oil leak, are rigidly placed in the

model, and swapped in and out of the scene as needed [MCDO95][OBYR95]. In NPSNET, the ship is just one of many vehicles in the simulation, and it may not be the only ship in the simulation. Thus, NPSNET can not utilize global references to manage the casualties, and can not swap the casualty models in and out of the scene.

All of NPSNET's ship casualties are objects that are created and manipulated by the ship vehicle. Given that the ship vehicle moves in the virtual world, the casualties are placed in the same local positions relative to the ship as in DC VET (i.e. the fire always burns in the engine room next to the fuel oil valve), but can be moved in the global coordinate system. Also their behavior and positions could be easily modified, allowing the casualties to dynamically move about the ship (see Chapter 7 for this future research topic). Many of the casualties, such as the steam leak and the fuel oil leak, are modeled exactly the same in both simulations. But some of them are modeled in different ways.

Both DC VET and NPSNET use Performer's fire and smoke animations to model their fires. But NPSNET manages its smoke and fires differently than DC VET. DC VET creates one smoke object and one flame object at the beginning of its simulation, and then turns the objects on and off as necessary. NPSNET has a pool of flame and smoke objects that all vehicles can use; thus the fire casualty object must request from the pool both a flame and a smoke object. Once obtained, the fire casualty object continuously updates the position of the smoke and flames and the size of the smoke and flames throughout the life of the fire casualty. When the fire goes out, the flame and smoke objects are returned to the pool for other vehicles to use.

When a fire is started, DC VET fills the engine room with smoke. The smoke is modeled using Performer's fog model. NPSNET cannot use the fog model to represent smoke, because it already utilizes the fog model to represent terrain fog and haze. Performer allows the user to create more than one fog model, but can only animate one model at a time [SGIA94]. Consequently, NPSNET instead uses its light model to represent the smoke. As the smoke fills the engine room, the light model is darkened. When the ship's ventilation is activated, the light model is gradually restored to it pre-fire state.

NPSNET and DC VET also handle the manipulation of the fire nozzle in different manners. DC VET loads two different models of the nozzle, one representing the nozzle turned off, and the other with the nozzle turned on. When the nozzle is turned off, the off_model's matrix location is set to the appropriate location in the ship model, and the on_model's matrix location is set to an infinity location. When the nozzle is turn on, the models swap matrix locations. In NPSNET, the ship's nozzle is loaded as part of the ship model. The models of the on and off nozzle position are managed by a switch node. Thus the position of the nozzle can be retained in a single matrix and only a switch need be toggled to turn the nozzle on and off.

Finally, DC VET and NPSNET manage the networking of these casualties differently. DC VET sends an entity state packet for the ship with the state of each casualty stuffed into the articulations field of the entity state PDU. NPSNET, on the other hand, utilizes the HIRESNET with a special ship tag to pass casualty information across the net between ship entities and their ghost representatives. Although the casualties may be implemented differently in both simulators, their functionality remains the same. The goals of the DC VET project [MCDO95][OBYR95] are still contained in NPSNET's ship model, and thus future research in this area need only be accomplished in NPSNET.

E. SUMMARY

Allowing human entities to successfully mount a ship entity and move around within that ship entity as it moves through the virtual environment is a significant achievement in the virtual reality community. Mounting allows human entities to interact with one another and control inanimate objects within the ship model, which more closely resembles how we humans behave in the real world. Although, NPSNET's implementation of mounted humans is crude, with advances in the DIS standard to support local coordinate values, more efficient means can be utilized to implement mounted humans in the future. With future research efforts, mounting will allow other entities (helicopters, jets, tanks,

small boats, etc.) to perform unique mounting exercises specific to their vehicle types, such as flight operations on the decks of ships or amphibious craft off loadings.

VII. CONCLUSION

A. RESULTS

The goal of this research was to implement a solution to mounting entities to other entities in the virtual environment. To achieve this goal, this thesis explored several issues of networked virtual environments in computer science, and achieved the following results:

- Remote control of ship entities has been successfully implemented. By using a simple control panel, the user has the ability to control a ship entity in the virtual environment from a different workstation then the simulation is running on. This allows the user of the control panel to be a different entity in the virtual environment and still control a ship entity.
- Voice recognition has been utilized to control the ship entity. Using SGI's "Speech Manager" voice recognition program and the ship control panel, the user can give voice commands to the ship and watch it respond to those commands in the virtual environment.
- A hybrid PVS algorithm implementation was utilized to allow the interior of the polygonal ship model to be displayed in real time within NPSNET's networked virtual environment. The implementation improved the one utilized in DCVET by operating simple switches instead of mangling the database.
- An additional network port is being utilized to send data PDU packets that represent shipboard events. The state of events such as fires, door positions and valve positions, are sent to all systems listening to the additional network port. This allows multiple human entities to interact with the ship and one another to perform team damage control exercises.

Human entities have successfully mounted a ship entity and moved around within that ship entity as it moves through the virtual environment. This is the most significant result of this thesis research. Being able to interact with and control inanimate objects in the virtual environment more closely resembles how we humans behave in the real world.

B. RECOMMENDATIONS FOR FUTURE WORK

This thesis is a continuation to the research done in the SHIPSIM and DC VET projects. Although these projects have been enhanced significantly due to their implementation within NPSNET, there are still many unexplored research areas involved with making these and other NPS Ship projects satisfactory for fleet utilization. Below is a summary of future research in this area.

1. Mounting Entities to Other Networked Entities

This research involves the implementation of a network protocol to have NPSNET entities mount one another only if the two agree to the mounting. Currently, the DIS standard does not support mounted vehicles, nor do current NPSNET vehicles agree on the mounting. This will enable vehicles like helicopters and jets to land on ships, or allow humans to embark and disembark from landing craft inside of ships.

2. Aquatic Environmental Effects

This research involves the implementation of physically based currents, eddies, wind and sea state in the virtual environment. All of these environmental factors should effect all ship and submarine vehicles, and operate across a network. Research is currently being conducted to implement environmental effects within the DIS standard and ocean effects should also be included within the standard.

3. Physically Based Weapons Suite Development for Ships

This research involves the design and implementation of shipboard offensive and defensive weapons in the virtual environment. Additionally the control and tracking of these weapons should be performed by human entities interacting with the ship inside its Combat Information Center, not through a separate control panel.

4. Dynamic Casualties

This research involves the implementation of physically based fires and flooding within the ship. The fires should be of all classes (A, B, C & D). These casualties should

be able to spread and move throughout the ship. Additionally, measures to control these casualties should also be implemented.

5. Human Interaction with the Ship

This research involves the implementation of more control measures within the ship that only human entities can manipulate. Things such as engine room throttles, helm steering wheel, and gauge indicators within the ship, would greatly increase the human immersion into the virtual ship. This would eliminate the need for external control panels to maneuver the ship. Only the ship's steering wheel and engine throttle positions would dictate how the ship would move through the virtual environment.

6. Improve Interface and Input Devices

This research involves the implementation of a better means of getting inputs from the user vice a 2D mouse. It is currently difficult for the user to interact with a three dimensional object (i.e fire nozzle) using a 2D mouse device. Perhaps a 3D device, such as a data glove, would allow the user to manipulate objects in a more realistic manner.

7. Increased Data Display

This research involves the improvement to displaying ship data from being just name and function to a whole host of other information (such as system diagrams, the effect manipulating such object has on the ship, etc.). In the case of complex systems, the program should be capable of displaying a diagram of the selected system. Then the user could immediately learn about how each selected object fits into the larger system. Until virtual environments are capable of displaying the more intrinsical details of real world objects, the embedding of information, such as video clips, should be explored. With video clips, detailed illustrations of equipment operation can be readily shown.

8. Test and Evaluation of Training within a Virtual Environment

This research involves the design and implementation of test and evaluation procedures for the exercises performed within the virtual environment. An objective

evaluation, considering human factors and recording empirical data, is required to prove the validity of training within a virtual environment.

9. Small Boat Operations

This research involves the modelling and utilization of small boats in the virtual environment. Small boats allow man overboard drills and amphibious landings to be accomplished in the virtual environment. Small boats have to operate as ship entities, but also be able to mount the ship when they are not in use.

10.Voice Communications over a Network

This research involves the design and implementation of shipboard communication systems in a virtual environment. Users should be able to select a communication device within the virtual environment, speak into a microphone, and have their voice message sent across the network to the station that was selected on the communication device. This message should then be either displayed or heard by other users in the selected station. All audio transactions should be over a network and not a temporary real communication device.

APPENDIX A. USER'S GUIDE

This appendix is the user's guide for operating the NPSNET Ship Project System (NPS SHIP). It explains how to start and operate the ship and human entities in a NPS SHIP demonstration. It does not describe all the features of NPSNET; the reader should consult NPSNET's User's Guide located on the Naval Postgraduate School Computer Science Department's world wide web page (www.cs.nps.navy.mil).

A. STARTING NPS SHIP

NPS SHIP is a simulation run within the NPSNET vehicle simulator. NPSNET is a robust system capable of configuring many input and output devices to control vehicles in a simulation. There are three basic parts to NPS SHIP; the Speech Manager, the control panel, and the NPSNET simulation. Each of these shall be discussed individually.

1. Speech Manager

As discussed in chapter 4, the Speech Manager is the software used to manage voice recognition in the system. The user is encouraged to consult the Speech Manager User's Guide [SGIC94] before starting a Speech Manager session. Prior to starting the Speech Manager, the user ensure a microphone is attached to his workstation. The Speech Manager system performs at its best with a stereo directional microphone. This type of microphone eliminates background noise, and provides the most direct input into the Speech Manager. The user should consult his system administrator for the installation of the Speech Manager Program on his workstation.

To start the Speech Manager, from a unix shell prompt execute the command below. Once started the user should see two Speech Manager windows. The first is the main

```
% speech &
```

window (see fig) where the voice recognition takes place. The second window is a listing of all the commands for the "shipControlWindow" (see fig).

The Speech Manager program is user sensitive, thus the user is required to periodically train the Speech Manager on the selected vocabulary in the "shipControlWindow." Consult the Speech Manager User's Guide for instructions on how to train the Speech Manager. The user does not necessarily need to re-train the Speech Manager before each use. The user should test out various words and phrases in the vocabulary listing, and if the Speech Manager is recognizing the user, then re-training is not necessary.

2. Control Panel

As also discussed in chapter 4 the control panel is the software that controls the ship in the virtual environment. To start the panel, the user should execute the command sequence below. As of Feb 23, 1996, the control panel and NPSNET are maintained by the

```
Remotely login to another workstation on your network
% rxterm machine

machine% <get into a view in configuration management >

[view]machine% cd /cm/npsnet
[view]machine% bin/demo-voice-shipcontrol
```

Naval Postgraduate School Graphics and Video Laboratory configuration management system, located in the directory /cm/npsnet. The user should use the view *bcstewar* until the NPSNET software is placed within the bin_demo subdirectory of /cm/npsnet. At that time the user will not need to get into a view to execute the control panel and NPSNET, but need only change to the /cm/npsnet directory and run these programs from the *bin_demo* subdirectory versus the *bin* directory.

3. NPSNET

NPSNET is the visual simulation manager of NPS SHIP. The user needs to have access to more than one workstation on a network, before starting NPSNET. Each

workstation will be numbered 0..n in this discussion to help alleviate confusion. The Speech Manager and the control panel should have been started on workstation #0. To start NPSNET, the user needs to open shell windows on workstations #0 and #1, then get into the configuration management view *bcstewar* and change to the directory /cm/npsnet on both workstations. The user should then execute the commands below. Because NPSNET

```
Workstation #0 - start the human entity
[view]machine% bin/npsnetIV -f config.bcstewar.sailor -f config.fcs

Workstation #1 - start the ship entity
[view]machine% bin/npsnetIV -f config.bcstewar.ship -control ship_panel
```

is a robust system, it can be started in many different configurations. Below describes some of the alternative ways to start NPSNET.

```
1) To start a ship on workstation #1 for use without a control panel
[view]machine% bin/npsnetIV -f config.bcstewar.ship

2) To start a human for use without the flight control joysticks
[view]machine% bin/npsnetIV -f config.bcstewar.sailor

3) To start a human with a HMD & flight control sticks
- Setup your HMD hardware and execute the NPSNETcommand below:
[view]machine% bin/npsnetIV -f config.bcstewar.sailor -f config.fcs
    -f config.vim -window vim -hmd_file datafiles/fastrak_vim.dat

4) To start a human using the upperbody tracking system
  - Setup the tracking hardware and execute the NPSNETcommand below:
 [view]machine% bin/npsnetIV -f config.bcstewar.sailor -f config.fcs
     -upperbody datafiles/fastrak_ubs.dat

  - Once the sensors are on your body, follow the commands in the shell window
to calibrate the sensors.
```

B. OPERATING NPS SHIP

NPS SHIP is primarily operated from workstation #0 where the human entity resides. There are many important details the user should be aware of when operating the system.

1. Speech Manager & Control Panel

You can practice giving commands to make sure the Speech Manager is working properly, by placing the cursor in the "shipControlWindow" but not in the pink box. The Speech Manager recognizes your commands when the FACE ICON in the main window is a happy face. If it is a face with a frown face, or a face with tears, then you need to say the command again. You should re-train any missed words, when Speech Manager does not find such words after three tries.

To issue a voice command to NPSNET, the cursor must be in the "shipControlWindow" in the pink box labelled Voice Commands. If the voice command is successful, and you are in the pink box, then the dials and switches should move accordingly. Additionally, the voice command will be acknowledged by the control panel with an audio reply, indicating that the command was given correctly.

2. NPSNET

NPS SHIP operates as an NPSNET vehicle simulation. All NPSNET commands functions as described in the NPSNSET User's Guide. The exceptions occur when operating the "antares96" ship and the "sailor" human entities.

The "antares96" ship moves within the virtual environment using either the control panel, as described above, or the normal NPSNET movement keyboard keys ('a' move forward, 's' stop, 'd' move backwards, 'right arrow" turn right, and 'left arrow' turn left). To start the casualties in the engine room of the "antares96" ship; <CNTL f> starts a fuel oil leak and fire; <CNTL s> starts a steam leak. The casualty commands must be issued from the workstation where the ship resides at.

The "sailor" humans move within the virtual environment also using the standard NPSNET commands. The exceptions occur when transporting about the "antares96" ship, and when manipulating objects in the "antares96" ship. Once mounted on the "antares96" ship, the "sailor" can transport to different locations about the ship using the keyboard commands described in Table A-1. Additionally, the user can manipulate objects in the

Transport Keys & Locations (you must be mounted on the ship to use these)	
F9	Pier on Island
CNTL F9	Cargo Deck/Vehicle Ramp
CNTL F10	Port Bridge Wing
CNTL F11	Engine Room Lower Level
CNTL F12	Combat Information Center

Table A-1: Transport Keys & Locations

ship as follows. First, the "sailor" simulator must be in the SHIP PICK mode. Using the middle mouse button, the user can change the NPSNET picking mode (see NPSNET User's Guide for more information). Second, in order to manipulate an object in the model, (i.e Picking up the nozzle) use the looking mechanism (white arrow keys on key pad, or hats of the fcs) and line up the center cross hairs of the heads up display (HUD) or the center of simulation window over the object to be manipulated. Then press either the Left Button or Right Button as described in Table A-2 and Table A-3.

Joystick Buttons for Ship Picking	
Throttle #7	Left Button
Throttle #3	Nozzle Toggle Switch
Stick Bottom	Right Button

Table A-2: Joystick Buttons for Ship Picking

Mouse Buttons for Ship Picking	
Left	Left Button
Middle	Change Picking Modes
Right	Right Button

Table A-3: Mouse Buttons for Ship Picking

When the user manipulates a fixed object, (such as bulkhead, deck, ceiling, desk, etc.) information about that object is displayed in the unix shell window. When the user manipulates a movable object (such as doors or valves), the movable object moves in the direction of the buttons, either opening or closing. The engine room nozzle is the only exceptional movable object. The nozzle can be move about in the model by the user's human entity. To pick up the nozzle, use the Left Button. To toggle the water on and off, either use the throttle button #3 on the flight control sticks, or look at the nozzle and use the Left Button. To return the nozzle to the stanchion, use the Right Button.

C. DEMONSTRATION SEQUENCE OF NPS SHIP

Once NPS SHIP is operating, the user can follow the demonstration below to see many of its features. The '...' in the voice commands below indicate you should pause and wait for the FACE ICON to be a happy face before saying the next part of the command. For example, the command "ALL AHEAD FULL" should be said as "ALL" and then "AHEAD FULL.'

- From the island facing the ship, walk (using either joysticks or keyboard) on board the ship. Move around aboard the ship. Walk up the vehicle ramps until you are on the main deck outside of the ship. Make your way to the Forward Superstructure. Walk up the ladders that are centerline to the Bridge or transport to the Bridge.
- Get the ship underway.

  Control Panel: select 999 on the RPM indicator

or

Voice: say (indicate ... 9 ... 9 ... 9).

Control Panel: select BACK_FULL on both port and stbd indicators

or

Voice: say (all ... back_full)

• When the ship is clear of the island, turn the ship counter clockwise.

Control Panel: move the RUDDER dial to the left about 20 degrees

or

Voice: say (left ... full_rudder)

• Now move the ship forward away from the island.

Control Panel: move the RUDDER dial to the right about 20 degrees

select AHEAD_FULL on both the port and stbd indicators

or

Voice: say (shift_your_rudder)

say (all ... ahead_full)

• You can maneuver the ship as you please issuing commands. When ready to show the next step of the demo, stop the ship.

Control Panel: select STOP on both the port and stbd indicators

or

Voice: say (all ... stop)

• Start a fuel oil leak and fire.

- Transport to the engine room <CNTL F11>
- Start the fuel oil leak on the workstation #1 <CNTL f>.
- Wait until the fuel oil leak starts a fire.

• Stop the fuel oil leak and fire.

- Move to the stanchion with the red and black buttons on it.
- Using the mid mouse button: put NPSNET in SHIP PICK picking mode.

- Put the picking cross hairs over the red (halon) button.
- Push the Left Button to put the fire out using halon.
- Move back towards the fire, to stop the fuel oil leak.
- Put the picking cross hairs over the fuel oil valve.
- Push the Left Button to close the valve all the way.
- Move back to the stanchion.
- Put the picking cross hairs over the nozzle.
- Push the Left Button to pick up the nozzle.
- Move back to the fire.
- Either, push the nozzle toggle switch (if using joysticks)

or

put the picking cross hairs over the nozzle, and

push the Left Button to turn the water on.

- Put water plume over fire until it goes out.
- Either, push the nozzle toggle switch (if using joysticks)

or

put the picking cross hairs over the nozzle, and

push the Left Button to turn the water off.

- Put the picking cross hairs over the nozzle.
- Push the Right Button to put the nozzle back on the stanchion.
- Move back to the stanchion.
- Put the picking cross hairs over the black (vent) button.
- Push the Left Button to clear the engine room of smoke.

• Start a steam leak.

- Transport to the engine room <CNTL F11>
- Start the steam leak on the workstation #1 <CNTL s>.

• Stop the steam leak.

- Move between the 6 grey boxes (feed pumps).
- Turn to look at the big valve on the white vertical pipe.
- Using the mid mouse button, put NPSNET in "SHIP PICK" picking mode.
- Put the picking cross hairs over the steam valve.
- Push the Left Button to close the valve all the way.
- Put the picking cross hairs over the steam valve.
- Push the Right Button to open the valve all the way.

• Walk through the ship model.

- Transport to the Combat Information Center <CNTL F12>
- Move to in front of the yellow door in the space.
- Put the picking cross hairs over the yellow door.
- Push the Right Button to open the door all the way.
- Move to the passage way, and turn to your left.
- Move in front of the next yellow door to the left of the stair case.
- Put the picking cross hairs over the yellow door.
- Push the Left Button to open the door all the way.
- Move into the radar room, just in front of the yellow table.
- Turn to your left and face the blue cabinets.
- Put the picking cross hairs over one of the blue cabinet doors.
- Push the Left Button to open the door all the way.
- Turn to your left, and move out of the radar room.
- Turn right, move forward and go down the stairs.
- Turn right 180 degrees.
- Move behind the ladder in front of one of the two yellow doors.
- Put the picking cross hairs over a yellow door.
- Push the Right Button to open the door all the way.
- Move into the room, pan the room, then move out of the room.
- Move to the door access opposite the ladder.

- Move through the access, and move towards the opposite stair case.
- Go down the stair case into the engine room.
- Turn left or right, go behind the ladder you just came down.
- Move in front of the next stair case.
- Repeat the last three steps until you are on the lower level of the engine room.
- Transport to the bridge <CNTL F10>.
- Turn around 180 degrees, move into the bridge.
- Look and walk around the bridge.

LIST OF REFERENCES

[AIRE90] Airey, John M., Brooks, Frederick P. Jr., and Rohlf, John H. "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments," *Computer Graphics*, Vol 24, No 2, March 1990.

[CARD86] Card, Stuart K., Undersatnding Key Constraints Governing Human-Computer Interfaces, Xerox Palo Alto Reasearch Center, Palo Alto, CA, 1986.

[FOSS94] Fossen, Thor I., *Guidance and Control of Ocean Vehicles.* New York: John Wiley & Sons Inc, 1994.

[HART94] Hartman, Jed and Creek, Patricia, IRIS Performer Programming Guide, Document Number 007-1680-020, Silicon Graphics, Inc, 1994.

[IST93] Institute for Simulation and Training, *Standard for Information Technology - Protocols for Distributed Interactive Simulaton Applications (Draft)*, Orlando, FL, 1993.

[LENT95] Lentz, F., Shaffer, A., Zyda, M., Pratt, D., Falby, J., *NPSNET: Naval Training Integration*, Naval Postgraduate School, Monterey, CA, 1995.

[LOCK94] Locke, John, "An Introduction to the Internet Networking Environment and SIMNET/DIS", Computer Science Department, Naval Postgraduate School, October 24th,, 1994.

[LUEA95] Luebke, David P., Georges Chris, *Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets*, Department of Computer Science, University of North Carolina, Chapel Hill, NC, 1995.

[LUEB95] Luebke, David P., *The pfPortal Visiblity Library*, Department of Computer Science, University of North Carolina, Chapel Hill, NC, 1995.

[MCDO95] McDowell, Perry and King, Tony, *A Networked Virtual Environment for Shipboard Training,* Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.

[MALO85] Maloney, *Elbert S., Dutton's Navigation and Piloting*, Annapolis, MD, Naval Institute Press, 1985.

[MSI94] Marine Safety International, *U.S. Navy Simulator Shiphandling Training Student Guide*, Middletown, RI, 1993.

[NAS95] National Academy of Sciences, *Virtual Reality - Scientific and Technological Challenges,* National Academy Press, Washington, D.C., 1995.

[NOBL95] Nobles, Joseph and Garrova, James, *Design and Implementation of Real-Time, Deployable Three Dimensional Shiphandling Training Simulator,* Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.

[NRL95] Naval Research Laboratory,"Virtual Environments for Shipboard Damage Control and Firefighting Research", http://nrl.com.damageControl, 1995.

[OBYR95] O'Byrne, James E., *Human Interaction within a Virtual Environment for Shipboard Training,* Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.

[PIOC95] Pioch, N., *Officer of the Deck: Validation and Verification of a Virtual Environment for Training,* The Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[SGIA94] Silicon Graphics Inc. Document Number 007-1680-020, *IRIS Performer Programming Guide*, J Hartman and P. Creek, 1994.

[SGIB94] Silicon Graphics Inc. Document Number 007-1681-020, *IRIS Performer Reference Pages*, S Fischler, J. Helman, M. Jones, J. Rohlf, A. Schaffer and C. Tanner, 1994.

[SGIC94] Silicon Graphics Inc., Document Number 007-2590-001, *Developer Magics's RapidApp User Guide, 1994,*.

[SGID95] Silicon Graphics Inc. Document Number 007-2282-001, *Speech Manager User's Guide*, March 1995.

[SHNE92] Shneiderman, Ben., *Designing the User Interface*, Addison-Wesley Publishing Co., Menlo Park, CA 1992

[STOR95] Storms, Russell L., *NPSNET-3D Sound Server: An Effective Use of the Auditory Channel,* Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.

[USN94] U.S. Naval Regional Contracting Center, Washington Navy Yard, *Statement of Work to Contract N00600-91-R-1558 for services from Marine Safety International*, June 1993.

[WEAV95] Weaver, J., Perrin, B., Zeltzer, D., Robinson, E., *Damage Control Training in a Virtual Environment,* Naval Personnel Research and Development Center, 1995.

[ZELT94] Zeltzer, D., Aviles, W., Gupta, J., Nygren, E., Pfautz, E., Pioch, N., Reid, B., *Virtual Environment Technology for Training: Core Testbed,* The Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA,1995.

[ZYDA93] Zyda, M., Pratt, D., Falby, J., Barham, P., and Kelleher, K., "NPSNET and the Naval Postgraduate School Graphics and Video Laboratory," *Presence*, Vol. 2, No. 3, 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Dr. Michael J. Zyda, Professor 2
   Computer Science Department Code CS/ZK
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. John S. Falby, Lecturer 2
   Computer Science Department Code CS/FJ
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. LCDR John A. Daley 1
   Computer Science Department Code CS/PA
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. Paul Barham 1
   Computer Science Department Code CS/FJ
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. LT Bryan C. Stewart 1
   1025 Rucker Ave
   Gilroy, CA 95020