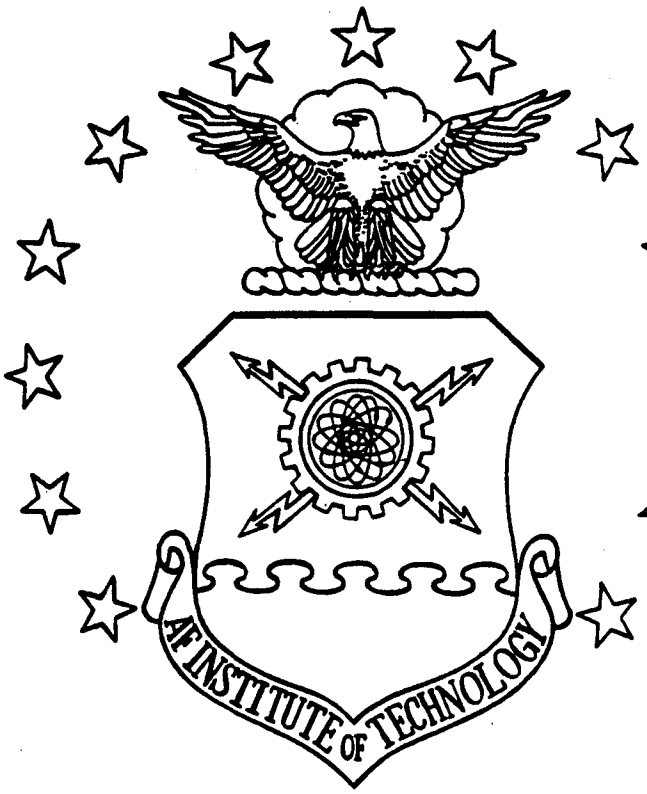


SPACE MODELER: AN EXPANDED, DISTRIBUTED, VIRTUAL ENVIRONMENT FOR SPACE VISUALIZATION

THESIS

John C Vanderburgh
Captain, USAF

AFIT/GCS/ENG/94D-23

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

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-23

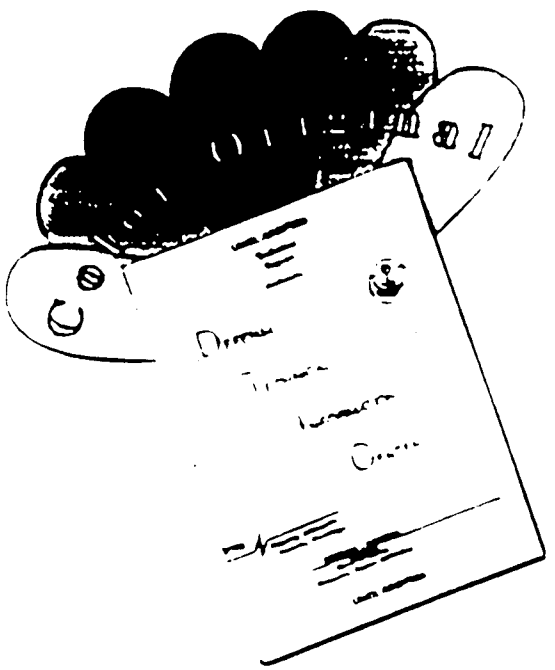
SPACE MODELER: AN EXPANDED, DISTRIBUTED, VIRTUAL ENVIRONMENT FOR SPACE VISUALIZATION

THESIS

John C Vanderburgh
Captain, USAF

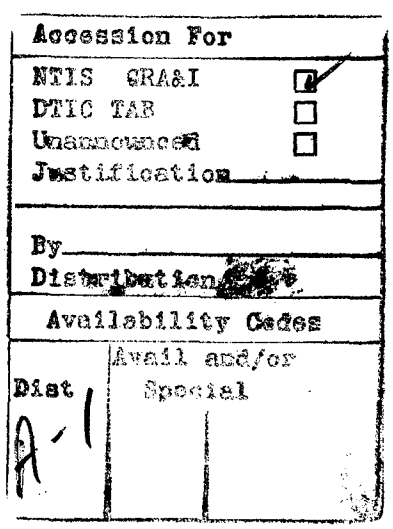
AFIT/GCS/ENG/94D-23

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.



AFIT/GCS/ENG/94D-23

SPACE MODELER: AN EXPANDED, DISTRIBUTED, VIRTUAL ENVIRONMENT FOR SPACE VISUALIZATION

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

John C Vanderburgh, B.S.

Captain, USAF

December, 1994

Acknowledgements

As always, this thesis effort would not have been possible without the assistance of others. First, I thank God for the opportunity to experience AFIT, no matter how painful it was. Second, thanks go to Jim Kestermann for his lessons on ethics, JJ Rohrer for all his views on life, and Milt Diaz for all the afternoon running sessions. Our group discussions and brainstorming sessions were invaluable sources for motivation (as well as laughs). I especially thank Dr. Paul Chodas, from JPL, who provided sample code and numerous explanations for how to model planetary orbits. Finally, I thank my advisor, Lt Col Martin Stytz, for his guidance and expertise, which ultimately kept me on the right track.

John C Vanderburgh

Table of Contents

List of Figures

List of Tables

Abstract

The *Space Modeler* is the first truly immersive virtual environment that models the solar system, models satellites in near-Earth orbit, and can operate in a Distributed Interactive Simulation (DIS) environment. It increases the capabilities of the 1993 *Satellite Modeler* by expanding the physical limits of the environment and by implementing a new three-dimensional user interface. Satellite orbits are modeled using NASA two-line element sets. The positions of the Sun, Moon and planets are computed using an algorithm based on planetary orbital element sets using a linear polynomial fit. For higher precision, the planetary orbits can be computed using an algorithm based on VSOP87 planetary theory. The user interface is a three-dimensional array of panels, with buttons for controlling all aspects of the environment. The interface is easily accessible to an immersed user, and provides interactive movement controls for five degrees of freedom, attachment to objects in the environment, setting the rate at which the simulation runs, and toggling the display of all visual features. The *Space Modeler* can model the solar system and render up to 50 satellites in near-Earth orbit using magnetic head tracking with a 20-30 Hz frame update rate.

SPACE MODELER: AN EXPANDED, DISTRIBUTED, VIRTUAL ENVIRONMENT FOR SPACE VISUALIZATION

I. Introduction

1.1 Background

In the past few years, virtual environment applications have become increasingly popular. Virtual environments are interactive, computer generated surroundings that give the user the illusion of being in another place, either real or imaginary. The technology used in these environments bridges the gap between humans and machines and allows humans to interact more effectively with computers. These environments potentially offer the most realistic simulations by allowing the most natural human-computer interaction. According to one expert, these environments provide a new and valuable communication medium for human-computer interaction. They have broad applications in education, teleoperation, scientific visualization, procedural training and entertainment (10). However, to be worthwhile, these environments must not only display accurate, realistic models within an acceptable frame rate, but must respond reasonably to natural human movements and auditory commands.

AFIT's Virtual Environments, 3D Medical Imaging and Computer Graphics Lab has been developing and implementing virtual environment technology for the past four years. The *Virtual Cockpit*, *Synthetic BattleBridge* and *Satellite Modeler* applications all provide immersive, computer-generated, distributed environments which demonstrate the latest in off-the-shelf hardware and software (31, 47, 26). However, because of the complex interaction and processing of the human sensory system, virtual environments such as these have been limited in their ability to offer a truly natural user interface for immersed applications. The user interfaces for these applications still require substantial training to master, whereas the ideal virtual environment would allow any user to function with little or no training.

1.2 Problem Statement

This thesis effort was aimed at improving the user interface as well as the functionality of the *Satellite Modeler*. The *Satellite Modeler* from 1993 was a direct manipulation virtual environment for

training space systems analysts. It modeled and rendered satellites in accurate near-Earth orbit and could be incorporated into distributed simulations that use the Distributed Interactive Simulation (DIS) protocol (27). By expanding its capabilities and ease of use, this tool can easily be extended in the near future to an operational, remote procedural trainer for ground-based satellite controllers or even a real time, satellite maneuver observation post for analysts.

Although the *Satellite Modeler* provided a very useful and appealing environment for viewing satellite behavior in near-Earth orbit, it had some limitations as to what it modeled. Its user interface was not conducive to immersive operation and its environment was limited to a small subset of outer space, containing only the Earth and the Moon.

The purpose of this thesis effort, which resulted in the design and development of the *Space Modeler*, was twofold. First, it was to address the problems associated with making the user interface accessible to a person who is completely immersed in a virtual environment. Second, it was to explore and implement efficient means for expanding the physical limits of the *Satellite Modeler* environment to include the solar system.

1.3 Assumptions

I made several assumptions concerning all of the enhancements made to the *Satellite Modeler*. First, I did not want to alter any of the orbital mechanics that were implemented in the original software. Last year's thesis effort emphasized the development of accurate and realistic orbital propagation for all satellites as well as the Earth (26). I assumed that all software for orbital propagations produced valid results, including all two-line orbital element sets for individual satellite types. Second, I assumed that the positioning of all the stars in the "celestial sphere" was accurate relative to the Earth. Third, I assumed that real-time propagation of all entities in the space environment was valid and that all information contained in the network broadcast packets was accurate. Finally, I assumed that all software developed for this effort would meet the following guidelines:

- All software would make use of the *ObjectSim* application development framework as well as the *Performer* development libraries
- All software would execute on the Silicon Graphics Onyx machine, under the *IRIX* operating system

- All software would be written in AT&T C++

1.4 Approach

This thesis required three major steps. First, I had to design and implement a suitable immersive user interface to a virtual environment. Second, this interface had to be incorporated into the 1993 *Satellite Modeler* along with small enhancements and corrections. Finally, I had to design a method for modeling the rest of the solar system.

Designing a new three-dimensional, general-purpose user interface for virtual environments presented an interesting challenge. Capt Jim Kestermann, Lt JJ Rohrer and I worked together on this interface. We had to consider the advantages and disadvantages of current virtual environment interfaces as well as research general human factors guidelines for human-computer interaction. We also had to investigate the utility of various user input devices, such as the instrumented glove and the three dimensional mouse. The results of these efforts are discussed in (24).

1.5 Thesis Presentation

This thesis is divided into six chapters and additional appendices. Chapter II presents background information on virtual environment technology as well as fundamentals of orbital mechanics and distributed simulations. Chapter III presents the overall requirements of this effort, Chapter IV discusses the design and implementation of the software and Chapter V summarizes the results. Finally, Chapter VI summarizes the thesis effort and presents recommendations for future work.

II. Background

This chapter provides a foundation for understanding the concepts involved in developing a distributed space visualization environment. It presents a brief explanation and history of virtual environment technology, gives an introduction to orbital mechanics, and describes distributed interactive simulations.

2.1 Virtual Environment Technology

Virtual environments are computer generated surroundings that humans can enter. This paradigm assumes that the surroundings contain information, such as other objects, that the user is motivated to explore, interact with, and learn about. Such environments use computer displays enhanced by special cues, such as colors, sounds and touch, to convince users that they are interacting in a synthetic world. When coupled with an appropriate user interface, these environments are useful because they serve as natural communications media for data analysis and scientific visualization (10, 28, 14).

All virtual environments require certain hardware and software components. Stephen Ellis states that virtual environments require the following three hardware components:

- Sensors to detect the user's body movements
- Effectors to stimulate the operator's senses
- Special-purpose hardware that links the sensors and effectors to produce sensory experiences resembling those in a physical environment (10:18)

Figure 1 illustrates these components. The user interacts with the environment via hand, head or other body movements. These movements are communicated to the hardware linkage via sensors such as instrumented gloves or magnetic head tracking devices. The linkage, which is usually a high-performance computer, interprets the sensor input and provides feedback to the user via an appropriate effector, such as a computer CRT display or head-mounted display (HMD). Of course, the use of these hardware components requires software.

The software architecture must address three issues:

- The shape and kinematics of the actors and objects

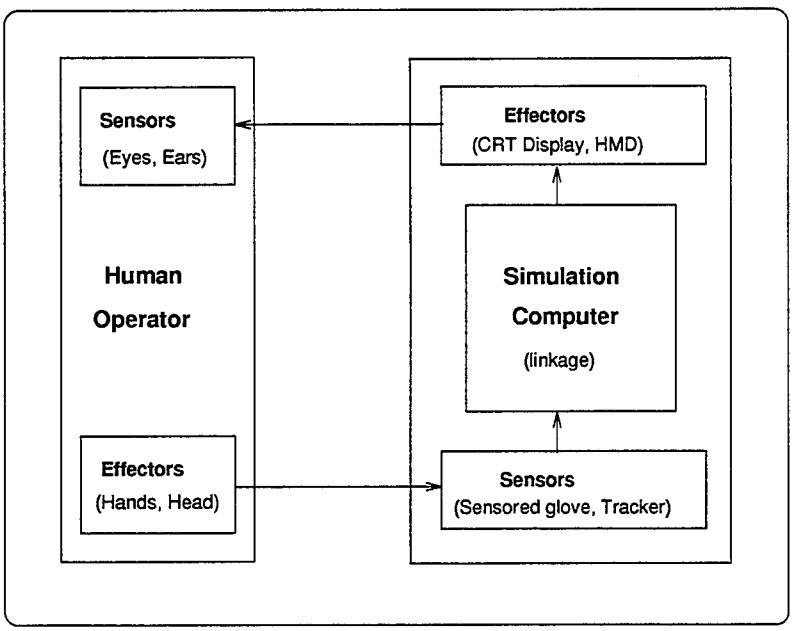


Figure 1. The Hardware Components of a Virtual Environment

- The physical interaction between objects and environment according to rules like Newtonian laws of motion
- The extent and character of the enveloping environment (10:18)

Additionally, the software architecture must be flexible enough to accomodate changes in its use of network protocols, input devices and rendering platforms. Fortunately, many software development architectures exist that consider these issues, such as the *Performer* library from Silicon Graphics as well as AFIT's *ObjectSim* framework (35, 43). Used in conjunction with high-quality computer graphics hardware, such architectures provide excellent development environments for virtual environment simulations. With this background in mind, let us examine the history of this technology.

2.1.1 Virtual Environment History. Ever since Ivan Sutherland proposed the *Ultimate Display* in 1965, researchers have been striving toward building the ideal virtual environment. Sutherland envisioned an environment in which computer generated objects would behave exactly as they do in reality (6). Such environments, where computer generated chairs can be sat upon, computer generated food tastes and looks real and computer generated weapons impart real damage, obviously do not

exist today, yet researchers are finding more and more feasible versions of them as current technology advances.

Virtual environment technology has its roots in systems which were based on vehicle simulation and teleoperation. As early as the 1920s, researchers were building aircraft simulators. In the 1960s teleoperation systems were using the first head-mounted displays (HMD) for controlling remote, closed-circuit television cameras. In 1968, Sutherland presented his own stereoscopic HMD along with a head tracking device, which was successfully used in computer-aided design (6). In the 1970s, other special hardware sensors, such as instrumented gloves, were first used in virtual environments.

In the 1970s researchers at MIT successfully used a commercially available tracking sensor to communicate hand position to a computer. This sensor, called the *Polhemus*, sensed position and orientation of an object and allowed the user to point to and move graphical objects in a virtual environment. Combining hand gestures with speech-recognition, this system was the first to provide a coherent, natural human-computer interface in a virtual environment (3:263). Today, many virtual environment applications still use versions of the *Polhemus* tracking system.

In 1977, the Air Force began work on a virtual aircraft simulator called the "Visually-Coupled Airborne Systems Simulator" (VCASS). VCASS was designed to present a panoramic display of cockpit information in three dimensional virtual space in such a way as to provide the pilot an improved situational awareness. Implemented in the early 1980s, VCASS used a hand controller to allow the pilot to activate switches and make other adjustments in the virtual cockpit (16). In addition to these early developments, several space visualization environments have been built.

A majority of the virtual environments for space visualization were built because three-dimensional, graphical representation of satellite orbital motion provides an important visual aid to understanding complex orbital maneuvers. Such environments are primarily designed to simulate a first person presence in space, but have also been used to train ground-based operators and simulate mission planning and launch activities (42). Ocampo's work in 1990 modeled the space environment in two and three dimensions, using an Earth-Centered Inertial (ECI) coordinate system, and was used to visualize how objects in geocentric orbit react to changes in orbital parameters (37). Similarly, Kobayashi and his collaborators developed a system for the Japanese space program for viewing orbiting satellites in real-time. Their system was used for satellite orbital analysis and allowed several user-defined con-

figurations (25). The National Aeronautics and Space Administration (NASA) has sponsored several virtual environment systems, including the Virtual Visual Environment Display (VIVED), the Virtual Interactive Environment Workstation (VIEW), the Computer Graphics Pilot Project (CGPP) and the Virtual Planetary Exploration (VPE) project (33, 20, 22). Specific mission simulators include the work done by Eyles for visualizing space station operations (12), the satellite modeling system built for the Air Force by Kaman Sciences Corporation, and the Global Positioning Satellite system by Sandia National Laboratories.

Head/helmet-mounted display (HMD) and haptic devices have also been used extensively by NASA. These devices immerse the user into the simulation by engaging the primary senses of sight and touch. The NASA Space Station Freedom project has promoted research in this area as part of the effort to train astronauts to function and perform activities in the natural conditions and constraints of space without the associated risks. By applying virtual environment technologies, astronauts, mission specialists, and station operators may be trained in "space simulators", such as VIEW, much like pilots are trained in flight simulators. This capability provides a virtual experience that emulates reality and may allow individuals to be more effective and productive once they begin to operate in the real environment (13).

While each of these systems presents information in a graphical format, they are all primarily stand-alone workstation-based simulators that portray only a selected portion of our solar system, which is usually Earth-centered. The *Satellite Modeler* was the first virtual environment simulation to portray satellites in accurate, near-Earth orbits in an immersive, distributed setting. This system offers an immersive or CRT-based environment and models satellite motion via the NORAD SGP4 orbital model and associated orbital elements and can broadcast satellite position and attitude in real-time via the Distributed Interactive Simulation (DIS) network protocol (26, 27).

One of the most important features of any virtual environment is its ability to accurately model the motion of all virtual objects. Modeling satellites and planetary bodies requires an understanding of the fundamentals of orbital mechanics.

2.2 *Introduction to Orbital Mechanics*

> *Which is more useful, the Sun or the Moon?...The Moon is the more useful, since it gives us light during the night, when it is dark, whereas the Sun shines only in the daytime, when it is light anyway.* -fictitious philosopher

There are several methods for mathematically describing the behavior of objects in space, but all are based on fundamental astrodynamic principles. In 1609, Johann Kepler published his famous laws of planetary motion. This work, which was based on the observations of Tycho Brahe, represented the first accurate description of planetary motion (1). Kepler's laws can be summarized as follows:

1. The orbit of each planet is an ellipse, with the sun at one focus
2. The line joining the planet to the sun sweeps out equal areas in equal times
3. The square of the period of a planet is proportional to the cube of its mean distance from the sun

In the late 1600's, Isaac Newton developed his laws of motion, which attempted to explain why Kepler's laws were valid. Newton's laws of motion were defined as:

1. Every body continues in its state of rest or of uniform motion in a straight line unless compelled to change that state by external forces
2. The rate of change of momentum is proportional to the force impressed and is in the same direction as that force
3. To every action there is always an opposing and equal reaction

In addition, Newton also formulated his famous law of gravity, which states that any two bodies attract one another with a force proportional to the product of their masses and inversely proportional to the square of the distance between them (1). Based on these laws, a general equation of motion for planetary bodies can be derived.

When depicting artificial satellites in orbit around the Earth, or the planets of our solar system orbiting the Sun, the application of Kepler's and Newton's laws can be limited to the classical *Two-Body Problem.* This problem makes two simplifying assumptions related to Newton's laws:

1. The bodies are spherically symmetric, allowing us to assume that their masses are concentrated at their centers
2. There are no external nor internal forces acting on the system other than their mutual gravitational forces

Based on these assumptions, as well as the use of an appropriate inertial reference coordinate system, one can proceed to mathematically describe the relative motions of such bodies. (See (1) for a detailed discussion of the derivation of the equation of motion.)

The motion of a satellite in orbit about a planet can be described using six orbital elements as integration coefficients in differential equations. As Figure 2 illustrates, the "classical" orbital elements are:

- The semi-major axis, which is one-half the length of the orbit (a)
- The eccentricity or shape of the orbit (e)
- The inclination of the orbit, which is the angle between the equatorial plane and the orbital plane (i)
- The ascending node, which specifies the planetary longitude at which the satellite's orbit crosses the equatorial plane (Ω)
- The argument of perigee, which specifies the angle between the ascending node and the point of perigee (ω)
- The time of perigee passage, which is the time at which the satellite passes closest to the Earth (T)

These six constants provide a basic orbital model that ignores the effects of perturbations upon spacecraft behavior (7, 39). However, more accurate models are available, such as the one developed at the North American Aerospace Defense (NORAD) Space Surveillance Center (SSC). The NORAD model replaces the time of perigee passage element with mean anomaly, which is the average difference between predicted and actual orbits. These "standardized" element sets are processed through one of five mathematical models to calculate values for spacecraft position and velocity. The models make

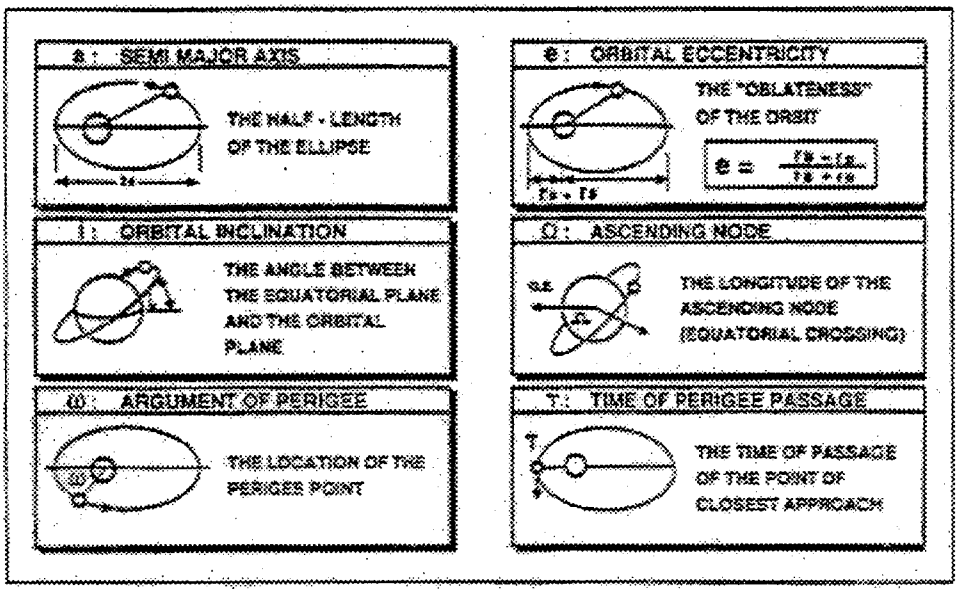


Figure 2. Orbital Elements
(30)

predictions for satellite position and velocity depending on whether they are in near-Earth (period of less than 225 minutes) or deep-space (period greater than or equal to 225 minutes) orbits. The *Satellite Modeler* uses NORAD's SGP4 model for satellite propagation. For a further description of the fundamentals of near-Earth orbits see (7, 36, 1).

While the NORAD element sets and the NORAD satellite propagation model can be used for satellites in near-Earth orbit, the equations do not apply to planetary, comet, or asteroid motion. To calculate the position of a body in its orbit around the Sun, or even the position of a moon around a planet, one may use numerical integration, calculate the sums of periodic terms based on orbital elements, or use orbital elements to approximate a purely elliptical orbit (36).

Numerical integration provides the most accurate method of determining planetary positions. According to Dr. Paul Chodas, the Jet Propulsion Laboratory in California uses this method for precise solar system modeling. They use a file of coefficients of Chebyshev polynomial fits over 32-day time spans. These coefficients are generated by numerically integrating data from all major solar system bodies while fitting hundreds of thousands of observations from the Mariner, Viking, Voyager, and

other spacecraft. As Dr. Chodas points out, however, "this method is far too cumbersome for general use (5)."

In 1982, the Frenchman P. Bretagnon published a method of computing planetary positions based on series of periodic terms. His theory, which was labeled VSOP82, allowed one to obtain various orbital parameters, such as the longitudes of the perihelion and ascending node, for a given instant. These parameters could then be used to compute an accurate, true position for a planet. The VSOP82 method considered orbital perturbations but was not very convenient to use, since one had to arbitrarily truncate the several series when no full accuracy was required (36).

In 1987, Bretagnon revised his original theory and published VSOP87. This newer method gave periodic terms for calculating the planets' positions in heliocentric coordinates. These coordinates were defined as the ecliptical longitude L, the ecliptical latitude B, and the radius vector R (distance from the planet to the Sun). To compute each coordinate, sums of periodic terms were calculated and used as polynomial coefficients, where the polynomial variable was time. VSOP87 was fairly accurate, but again was cumbersome to use, especially if high precision was not required (36).

The simplest approach for computing planetary positions is to approximate the orbit as a true ellipse, making no allowance for orbital perturbations. Meeus offers two methods for computing planetary positions. The first is based on the VSOP87 theory and the second is based on the use of orbital elements sets.

An elliptical planetary orbit can be described in geocentric coordinates using the VSOP87 theory. Using the VSOP87 method, the heliocentric coordinates of the planet, denoted by L, B and R, and those of the Earth, denoted L_0, B_0 and R_0, are computed for a specific time. Then, as shown in (36:209), the geocentric rectangular coordinates can be found using:

$$x = R\cos B\cos L - R_0\cos B_0\cos L_0 \quad (1)$$

$$y = R\cos B\sin L - R_0\cos B_0\sin L_0 \quad (2)$$

$$z = R\sin B - R_0\sin B_0 \quad (3)$$

This method is still cumbersome, since computing the heliocentric coordinates requires summing several periodic terms that must be extracted from published tables. Using orbital element sets to describe elliptical motion is more straight forward.

By using planetary orbital elements along with the geocentric coordinates of the Sun, one can easily obtain the geocentric rectangular coordinates of a planet. The orbital elements are:

- The semi-major axis of the orbit (a)
- The eccentricity of the orbit (e)
- The inclination of the plane of the ecliptic (i)
- The longitude of the ascending node (Ω)
- The longitude of the perihelion (ω)
- The mean longitude of the planet (L)

Figure 3 illustrates these elements. The arc from X'' to G represents the ecliptic (the Earth's orbital plane), as seen from the Sun. Point G represents the vernal equinox, which has longitude 0°. Point N is the ascending node of the planet's orbit and point P is the planet's perihelion (orbit position closest to the Sun). At any given instant, point X represents the mean location of the planet and X' represents the planet's true location. The semi-major axis of the planet's orbit along with the orbit's eccentricity, a and e, are standard parameters of an ellipse. The planet's inclination, i, measures the difference between the planet's orbital plane and the ecliptic. The longitude of the ascending node, Ω, is defined as arc 1. The argument of perihelion, which is ω, is the length of arc 2. The mean longitude of the planet, L, is the sum of arcs 1, 2, and 3. Note that L and ω are measured from the vernal equinox along the ecliptic to the orbit's ascending node and then from this node along the orbit.

The planetary orbital elements are computed from polynomials of the form

$$a_0 + a_1 T + a_2 T^2 + a_3 T^3 \tag{4}$$

where T is the time measured in Julian centuries from January 1.5, 2000, which is referred to as the J2000 epoch (36:197). Note that the value of T is negative for dates *before* that epoch. The coefficients

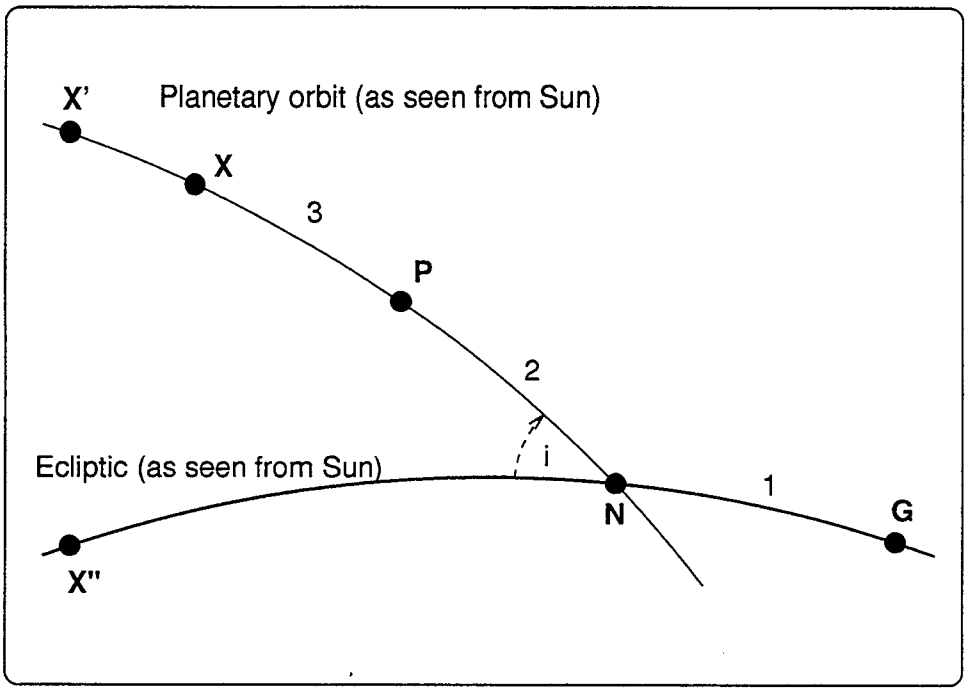


Figure 3. Planetary Orbital Elements
(36:198)

a_0, a_1, a_2, and a_3, which are based on planetary theory, are derived from linear or cubic polynomial fits (36:200–204). The coefficients for a given body may have different values, depending on whether the orbital elements should reference a *mean equinox of the date* or a *standard equinox*, such as the J2000 epoch. Several published sources provide the orbital elements referred to the J2000 epoch, so computing them is not necessary. The orbital elements are used in the following equations:

$$F = \cos\Omega \tag{5}$$

$$G = \sin\Omega\cos\varepsilon \tag{6}$$

$$H = \sin\Omega\sin\varepsilon \tag{7}$$

$$P = -\sin\Omega\cos i \tag{8}$$

$$Q = \cos\Omega\cos i\cos\varepsilon - \sin i\sin\varepsilon \tag{9}$$

$$R = \cos\Omega\cos i\sin\varepsilon + \sin i\cos\varepsilon \tag{10}$$

These values are then used to compute intermediate angular values:

$$\tan A = ra\sin(A+\omega+v) \quad (11)$$

$$\tan B = rb\sin(B+\omega+v) \quad (12)$$

$$\tan C = rc\sin(C+\omega+v) \quad (13)$$

$$a = \sqrt{F^2+P^2} \quad (14)$$

$$b = \sqrt{G^2+Q^2} \quad (15)$$

$$c = \sqrt{H^2+R^2} \quad (16)$$

Before computing the final coordinates, the true anomaly and radius vector must be computed. As Meeus describes, anomaly is the angular distance from the planet's perihelion to its current position. *True* anomaly is the angle to the planet's actual position and *mean* anomaly indicates the position of the planet if it moved around the Sun at a constant angular velocity in a circular orbit. Figure 4 distinguishes the two. Point P represents the planet's true perihelion and P' is the circular (or mean) perihelion. For any given instant, point K represents the actual position compared with point K', which is the corresponding position in a circular orbit. Mean anomaly is easily found, since it is $0°$ at perihelion and increases uniformly by time by exactly $360°$ after one complete revolution around the Sun (36:182). In Figure 4, the mean anomaly is depicted as angle M, while the true anomaly is angle v (which is angle PSK). Unfortunately, planets do not move at constant angular velocity in an elliptical orbit, so the true anomaly must be derived from the mean anomaly, using an approach described in (36).

For modeling elliptical orbits, true anomaly can be computed by solving Kepler's equation. Kepler's equation is

$$E = M + e\sin E \quad (17)$$

The variable M represents the mean anomaly and E is the *eccentric* anomaly, which is an angular distance based on the center of the ellipse rather than a focus of the ellipse. Figure 5 illustrates the difference between true and eccentric anomaly. True anomaly is angle PSK and eccentric anomaly

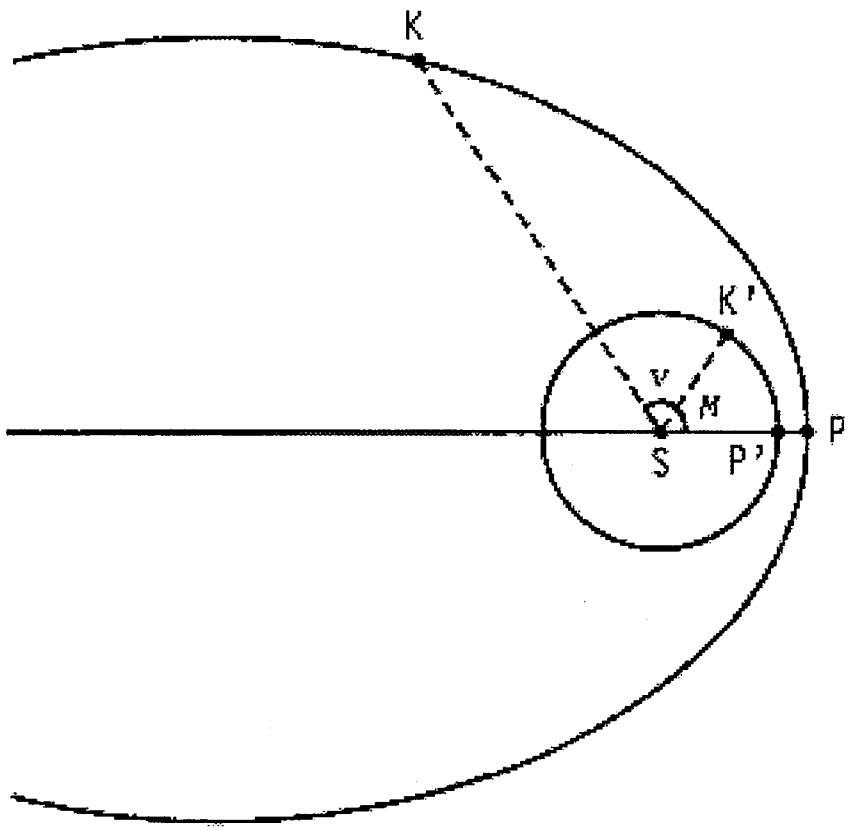


Figure 4. Anomalies of an Elliptical Orbit
(36:183)

is angle PCQ, where CQ represents a distance equal to the aphelion, or point of the elliptical orbit farthest from the Sun.

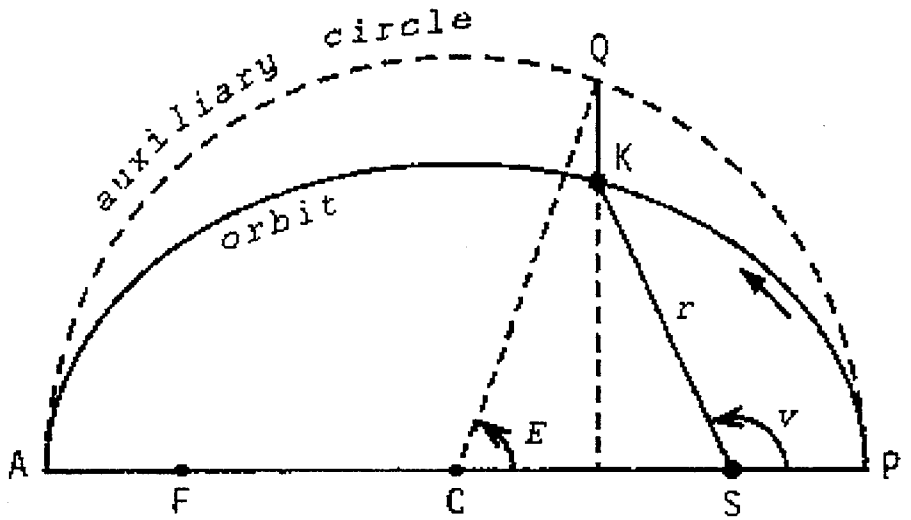


Figure 5. True vs Eccentric Anomaly
(36:181)

Since Kepler's equation is transcendental, it cannot be solved directly. Fortunately, a straightforward, iterative approach can be used to approximate E. One may approximate E with the mean anomaly on the right hand side of the equation and then iteratively derive a new value for E (36:184).

Once a value of E is found, the true anomaly, v, and the radius vector, r, can be found from the equations

$$\tan(v/2) = \sqrt{(1+e)/(1-e)}\tan(E/2) \quad (18)$$

$$r = a(1 - e\cos E) \quad (19)$$

(20)

Finally, the heliocentric rectangular equatorial coordinates are computed via:

$$x = ra\sin(A + \omega + v) \quad (21)$$

$$y = rb\sin(B+\omega+v) \quad (22)$$

$$z = rc\sin(C+\omega+v) \quad (23)$$

These coordinates can be transformed to geocentric via a simple coordinate translation matrix, based on the Sun's current geocentric coordinates. Now that an explanation of the applicable orbital mechanics theory has been presented, we will look at the design of a distributed virtual environment.

2.3 Distributed Simulations

The *Satellite Modeler* and *Space Modeler* rely upon distributed simulation technology to broadcast planetary position and velocity as well as satellite position and velocity data, via a computer network, to other actors in a distributed simulation. Distributed Interactive Simulation (DIS) technology uses heterogeneous hosts and a common synthetic environment definition to insert a wide variety of both human and computer controlled actors into a single, shared synthetic environment. This approach uses several networked virtual environment stations (using long-haul and/or local connections) to form a single environment wherein each node has its own local model of the environment and there are no clients or servers (46). Presently, the hosts are connected using T1 data links and use a common simulation and network protocol to communicate. The DIS protocol is described further in (2, 21, 32, 29) as well as IEEE standard 1278-1993.

Several virtual environment applications developed at AFIT rely on the DIS protocol for distributed interaction. The Virtual Cockpit immerses a user within an accurately modeled F-15E aircraft and allows the user to engage remote airborne and ground-based targets as well as accept damage from remotely launched weapons (11, 18, 9). The Synthetic BattleBridge offers an immersive situational awareness tool for a battlefield commander, displaying all remote DIS players as well as visual cues for battle intensity (45, 24, 40). The Red Flag Remote Debriefing Tool uses the DIS protocol for providing a graphical display of stored Air Force Red Flag missions (17). Finally, the DIS Virtual Cassette Recorder allows a user to store, retrieve and rebroadcast DIS data (15).

2.4 Summary

Developing a distributed space visualization system requires knowledge of some fundamental concepts. This chapter presented these concepts by describing virtual environment technology, providing an introduction to orbital mechanics, and explaining distributed interactive simulation technology. The next chapter presents the general requirements of this thesis effort.

III. General Requirements

This chapter presents the overall requirements for the new *Space Modeler*. The general goal of this year's work was to improve the capabilities of the *Satellite Modeler* as a virtual environment. The improvements consisted of enlarging the virtual environment boundaries, modeling new entities, improving the user interface and improving the Distributed Interactive Simulation (DIS) capabilities.

3.1 The Space Environment

The *Satellite Modeler* environment consists of the Earth, its moon, some stars, and artificial satellites in near-Earth orbit. It is an effective and realistic environment for Earth-centered observation of satellites. However, in order to observe planetary orbits, appreciate interplanetary distances, and simulate deep space operations, it is necessary to model the rest of the solar system. Extending the virtual environment would naturally make the *Satellite Modeler* more realistic and useful.

The specific requirements for the *Space Modeler* were:

- Make the environment totally immersive, including the user interface
- Double the maximum number of satellites that can be modeled
- Model the solar system, including the Sun and the other planets
- Accurately model all planetary motion
- Improve the lunar orbit
- Broadcast the position of the Sun and Moon via the DIS network to provide a means of synchronizing DIS applications

The portrayal of comets, other planets' moons, and other celestial bodies was not required in the scope of this effort since orbital data for these other bodies were not readily available.

3.2 Modeling New Entities

In order to portray a realistic environment, the *Space Modeler* had to use valid mathematical algorithms for the movement of all new entities. Unlike the *Satellite Modeler*, which only had to

incorporate mathematical algorithms for the propagation of the satellites, the *Space Modeler* needed astronomical algorithms for propagating all the planets in their correct orbits around the Sun.

3.3 The User Interface

The original *Satellite Modeler* incorporated a user interface that allowed interactive configuration of the simulation and also provided means for immersing the user via an HMD and head tracking device (26). This interface was designed as a two-dimensional interface, with menus and buttons displayed on the computer screen. For example, the user toggled the display of stars by clicking the mouse on a two-dimensional "button" overlaid on the computer screen. Once immersed (via the HMD), the user could not interact with the simulation, other than modifying the view direction according to head movement. For a truly immersive environment it is imperative that the user interface be easily accessible. This year's effort stressed designing such an interface.

Our goal for the immersive interface was to place the user within an information and control pod that contains all the information and control capabilities of the CRT-based interface from the *Satellite Modeler* while at the same time making their use more convenient and rapid by virtue of their position around the user. The approach we decided upon was to place the devices around the user on sets of 3D panels, with each panel devoted to specific categories of tasks.

Instead of overlaying a two-dimensional array of buttons on the computer screen, we decided that a general purpose, three-dimensional virtual environment interface would be more accessible to an immersed user. Such an interface would consist of panels, or collections, of buttons placed *within* the environment, as opposed to overlaid on the computer screen. The buttons must not obstruct the view, but must be within "reach" of the user. This new interface paradigm would resemble a "user pod" which would travel throughout the environment with the user's viewpoint. By improving the user interface, the *Satellite Modeler* environment would become a truly immersive application.

3.4 Distributed Interactive Simulation Capabilities

The *Satellite Modeler* is a DIS-compatible application because it can broadcast satellite position and orientation information (26). Other DIS-compatible applications, such as AFIT's *Synthetic BattleBridge*, can read this information and display geometry representing the relative locations of those satellites (40). However, one of the limitations of most DIS-compatible environments, which are Earth-based, is that they do not portray the Sun or Moon. Portraying the Sun and Moon and generating shading effects based on those portrayals would make any DIS environment more realistic. Thus, one of the requirements for the *Space Modeler* was to broadcast the position of the Sun and Moon relative to the Earth. This capability would allow the *Space Modeler* to synchronize environmental lighting conditions, thus enhancing truly distributed land, sea and space based simulations.

3.5 Summary

This chapter presented the general goals and requirements of this year's efforts to improve the *Satellite Modeler*. Extending the boundaries of the space environment, accurately modeling the rest of the solar system, broadcasting the positions of the Sun and Moon via the DIS protocol, and redesigning the user interface would result in a more realistic and immersive virtual environment. The next chapter discusses specifically how these requirements were fulfilled as well as implementation details of the design.

IV. Design and Implementation

In this chapter, I discuss how I designed software to meet the requirements and goals mentioned in the previous chapter. First, I present the overall software architecture. Then I address several design issues related to modeling the solar system. Finally, I describe the new user interface.

4.1 The Software Architecture

To integrate the expanded space environment and the new user interface, several new object classes were designed. These new classes, along with classes from the original *Satellite Modeler*, were organized into one overall simulation, which is depicted in Figure 6. The `Space_Simulation` class encapsulates the entire simulation and is derived from the *ObjectSim* `Simulation` class. (See (43) for a complete discussion of the *ObjectSim* framework.) My simulation class is similar to that of the *Satellite Modeler*, in that it comprises a terrain, a view, the starnode, satellites, and the Earth's shadow. However, my class also includes view modifiers, the planets, the Sun and a new user interface.

4.1.1 The Terrain. In *ObjectSim*, all simulations must have a terrain, which consists of some type of geometry file. The terrain file's local origin establishes the coordinate system origin for the entire simulation (43). The *Satellite Modeler* uses a model of the Earth for its terrain, so the Earth's center is the origin for the entire "world". Because I wanted to limit the modifications to the original *Satellite Modeler* design, the *Space Modeler* uses the same `Space_Terrain` class as the *Satellite Modeler*. As Figure 6 shows, this class, which is derived from the abstract `Terrain` class, loads and establishes the Earth model as the simulation's terrain. Thus, everything in the *Space Modeler* environment is rendered relative to the center of the Earth. As mentioned in (43), this class is also responsible for establishing the lighting for the simulation.

The *Satellite Modeler* defines an infinite light source for shading all of the entities in its environment. This light source is defined in the direction of the Sun, relative to the center of the Earth, so shading of the Earth, Moon and satellites is fairly realistic. Again, to limit the number of changes to the original design, the *Space Modeler* uses the same infinite light source. This provides realistic shading in the vicinity of the Earth, but unfortunately does not provide realistic shading for the planets,

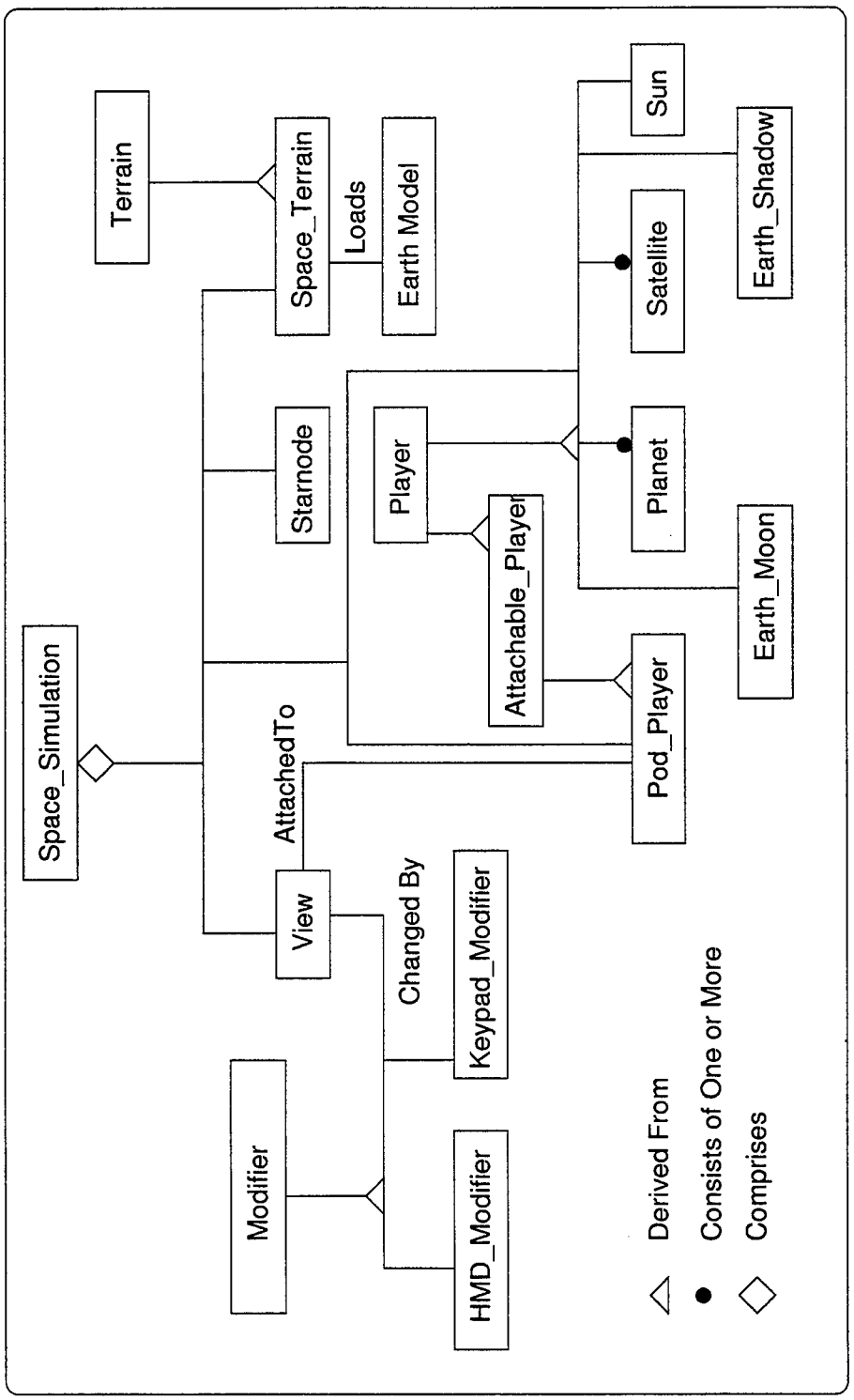


Figure 6. Object Diagram for Solar System Modeler

since an infinite light source would shade the same side of all planets, regardless of their position relative to the Sun. To provide adequate shading for the entire solar system, the light source would have to be modeled as a point light source. Unfortunately, as the documentation for the graphics library indicates, point light sources do not provide adequate lighting over great distances, so a single point light source at the Sun's location would produce incorrect shading effects for the outer planets (34). As a compromise, each planet is textured with a shaded texture map and rendered without any shading at all. Thus, each planet's shading effects come strictly from the texture map. (See (34) and (35) for complete discussions of lighting calculations.)

Unfortunately, as (34) states, local light sources are very expensive, in terms of CPU time, to render in a scene. As a compromise, each planet is textured with a shaded texture map and rendered without any shading at all. Thus, each planet's shading effects come strictly from the texture map. (See (34) and (35) for complete discussions of lighting calculations.)

In addition, the distances at which planets are rendered relative to the Sun prohibits the proper lighting calculations for a local light source.

4.1.2 The View. Every *ObjectSim* application must have at least one instantiation of a `View`, which must be attached to one entity, or `Attachable_Player`. The `View` object establishes the characteristics of the graphics window and establishes the location of the viewpoint based on the entity to which it's attached. As seen in Figure 6, the *Space Modeler* uses one instantiation of the `View` class and attaches it to the `Pod_Player` object. Each `View` instantiation may also have a `Modifier`, which controls the translation and orientation of the viewpoint (43). Because it's an immersive application, the *Space Modeler* is designed to be displayed on an HMD, using a head tracking device to control the position and orientation of the view. However, in order to facilitate monitoring of the simulation, the *Space Modeler* can also be displayed on the conventional computer screen, using keyboard entry for controlling the viewpoint. Thus, two instantiations of the `Modifier` class are needed, namely the `HMD_Modifier` and the `Keypad_Modifier`.

4.1.2.1 The HMD_Modifier. In order to use an HMD to control the viewpoint characteristics, we had to design an interface to the Polhemus 3SPACE® Fastrak™ magnetic tracking system. Because *ObjectSim* provided the `Modifier` class as a superclass for modifying viewpoint characteristics, we derived the `HMD_Modifier` class from that. This class encapsulates the low-level serial input/output (I/O) communication between the Polhemus device and the host computer, and allows the application to transparently access the Polhemus output. It accomplishes this via its constructor and three public methods. (See (44) for details on C++ object class methods and constructors.)

This class's constructor establishes which Polhemus station the modifier will use. The Polhemus tracking system has four available stations, which allow four independent trackers to be used simultaneously (38). An instantiation of this class must be associated with its own station and only one such instantiation may be used in a single application. An application may use more than one Polhemus station, but only one Polhemus station can be used to modify the view. Once instantiated, this modifier must be properly initialized.

The `init()` method allows the application to initialize various characteristics of the serial I/O. This method reads a data file, called `"fastrak.dat"` and establishes the port number and port speed based on its contents. Once initialized successfully, this modifier must be calibrated, in order to establish a reference attitude for computing `hpr` and `xyz` changes.

The calibration method, `Calibrate()` records the most recent `hpr` and `xyz` data and offsets all subsequent data by the stored values. This allows the modifier to compensate for different initial positions of the tracking source, since one user's initial head position may be different from another. After calibration, the modifier must retrieve new data for every rendered frame.

The `poll()` method forces the modifier to retrieve the latest `hpr` and `xyz` data from the serial I/O port. After adjusting for the calibration offsets, the data are transformed into the *Performer* coordinate system and stored for application use. Implementing this capability as a derivation of the `Modifier` class allowed straightforward integration of the Polhemus tracker as a view modifier.

4.1.2.2 The Keypad_Modifier. While the `HMD_Modifier` controls the position and orientation of the view via the Polhemus system, the `Keypad_Modifier` controls the view characteristics

via the keypad on the keyboard. This modifier, which *ObjectSim* provides, is convenient for controlling the view when the simulation is run without the HMD. The arrows on the keypad of the keyboard modify the heading and pitch of the view, while the arrow keys next to the keypad translate the view up, down, left, or right. The + and *Enter* keys on the keypad move the view forward and backward. In addition, the user can toggle between the two modifiers via the <F2> and <F3> keys.

4.1.3 The Pod_Player. The `Pod_Player` represents the immersive interface. As Figure 6 shows, it is derived from the `Attachable_Player` class from *ObjectSim* and serves as the attachment point for the view. This object class encapsulates all the controls available to the user for interaction and modification of the virtual environment. It is the only class in the simulation to which the view attaches, and thus is the only one derived from the `Attachable_Player` class. (The implementation of this object class is discussed in detail in a following section.)

4.2 The Starnode

The stars are rendered via a collection of the `pfLight` data structure in *Performer*. Approximately 5000 star locations are defined in a data file and are used to establish the right ascension and declination locations on a "celestial sphere." A point light source is rendered at each specified location. This design was ported from the original *Satellite Modeler*; see (26) for further details on its implementation.

4.3 Miscellaneous Players

The remaining object classes in the simulation represent the other entities in the virtual space environment. The `Earth_Moon` and `Earth_Shadow` classes encapsulate the rendering of the Moon and the Earth's umbra, respectively, and are taken directly from the *Satellite Modeler* architecture. The `Satellite` class represents the man-made satellites that are propagated in near-Earth orbit. This class encapsulates the necessary orbital mechanics for accurate satellite propagation, and is also taken directly from the *Satellite Modeler*. (See (26) for details on these classes.) In order to render the rest of the solar system, I designed the `Planet` and `Sun` object classes. As seen in Figure 6, all of these classes

are derived from the `Player` class in *ObjectSim* , since they have associated geometry for rendering. The next few sections present details about the implementation of the solar system and user interface classes for the *Space Modeler*.

4.4 The Solar System

In order to make the *Satellite Modeler* space environment more realistic and more useful, I was tasked to extend its boundaries to include the solar system. The sheer size of the solar system required redesigning the placement of objects and modifying the viewing parameters. Based on an initial analysis, the design requirements were:

- Model all nine planets, the Earth's moon, and the Sun
- Accurately model all planet orbits
- Improve the portrayal of the lunar orbit
- Allow viewpoint movement to any point in the solar system
- Maintain solar and lunar positions in ECI coordinates for DIS broadcasting
- Scale all distances and sizes to maintain an accurate model of the real solar system

To meet these requirements, I had to establish an appropriate world coordinate system, decide on an appropriate modeling scale, and incorporate accurate mathematical equations for propagating the objects in the solar system.

4.4.1 The Coordinate System. I used the Earth Centered Inertial coordinate system (ECI) for this virtual environment. This rectangular system's origin is at the center of the Earth, with the x-axis intersecting the equator at the vernal equinox, the z-axis intersecting the North Pole, and y-axis perpendicular to the x-axis in the equatorial plane (26). (See Figure 7.) The ECI coordinate system was selected for several reasons. First, the original *Satellite Modeler* used ECI coordinates, so I didn't need to convert any algorithms for propagating satellites. Second, because I was going to have to broadcast solar and lunar positions in ECI coordinates, I didn't want to convert to ECI every frame from some

other coordinate system. Third, I wanted the Earth to remain the "center of attention" for this world, considering that all the satellites are modeled in near-Earth orbit. The Earth serves as a good "default starting position." Finally, no matter what coordinate system I used, it would be transparent to the user: the planets would still appear to orbit the Sun.

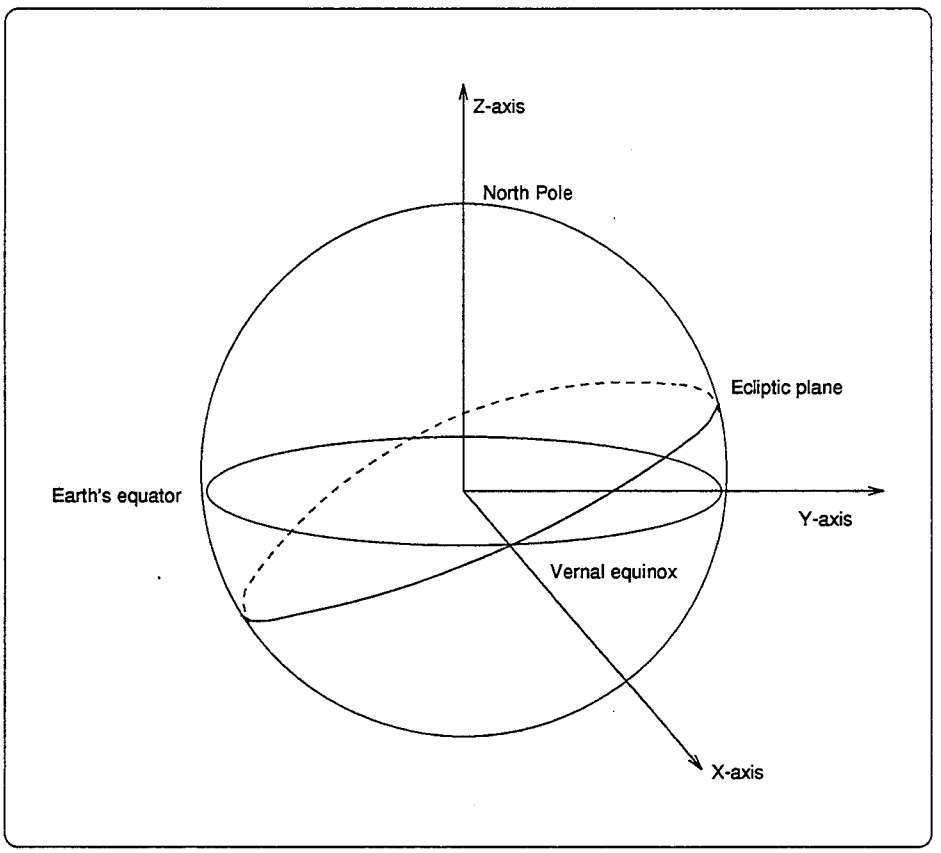


Figure 7. Earth-Centered Inertial Coordinate System

The *ObjectSim* framework makes it easy for an application to use the ECI coordinate system. As previously mentioned, the terrain file's local origin establishes the coordinate system origin for the entire simulation (43). Thus, through the *ObjectSim* framework, I could indirectly establish the ECI coordinate system as the world coordinate system by using the Earth model as the terrain.

4.4.2 Scale. The sheer size of our solar system required the selection of a suitable scaling factor. Our solar system is approximately 7 billion miles in diameter (23). Using a 1:1 scale for this world would prevent the hardware from handling z-buffer depth processing correctly. According to our

hardware documentation, the maximum z-buffer storage is 32 bits, which corresponds to a maximum depth value of just over 2 billion units. A 1:1 scale would also require a large viewing frustum, which would substantially increase the CPU time for rendering each frame (34). Thus, some scaling was necessary.

The original *Satellite Modeler* used a scale of 1:10,000 and used meters for units: one unit in the virtual world was one meter, which corresponded to 10 km in the real world (26). Using this scale would allow the use of all the satellite propagation code from the old system, thus requiring fewer changes. Fortunately, this scale was adequate for modeling the rest of the solar system since the largest units that would be handled by the application would now never exceed 700,000,000. I realized that I could also limit the viewing frustum to just one half of the solar system, since objects would be too small to be seen across the entire solar system. However, additional visual cues were necessary to indicate the positions of objects that were clipped from the viewing frustum. Finally, I had to maintain the "space background" features: the stars and Sun. The Sun and stars would always need to be visible, assuming an appropriate view direction. Based on this analysis, we established these additional requirements:

- Never clip the stars or the Sun from the viewing frustum
- Display visual cues for all planets' positions

Assuming that the rendering and propagation of the Earth and its satellites would not change, the priority was establishing the "bounds" of the new space environment by modifying the viewing frustum and establishing the positions of the stars.

4.4.3 The Viewing Frustum. Because I wanted to prevent the Sun and stars from getting clipped from view, their location and size dictated the dimensions of the viewing frustum. When the Sun is modeled relative to its actual diameter, it is visible at great distances. In my scaled space environment, I determined that the Sun decreased to somewhat of a point light source at a distance of about 35,000,000 units. Thus, I set my new viewing frustum to that length. Although the documentation for *GL Programming* recommended placing the far clipping plane of the frustum as close to the viewpoint as possible, I didn't want the Sun or larger planets constantly clipped from view as the viewpoint was moved (34).

4.4.4 The Stars. Extending the available space much beyond the Earth required redesigning the rendering of the stars. The old system rendered the stars as point light sources fixed to a "celestial sphere", which was centered at the Earth (26). If you moved far enough away from the Earth, you could see this sphere, as shown in Figure 8. To prevent this, I decided to center the starfield at the viewpoint. This would allow me to render the stars at a reasonable, fixed distance, yet prevent the user from moving beyond the boundaries of the celestial sphere. The stars would never be clipped from view, yet they would appear essentially at infinity. As long as the radius of the new celestial sphere was close to the length of the viewing frustum, no objects in the environment would appear in front of the stars. Naturally, this design does not render the stars in their "true" (heliocentric) position. However, according to John Danby in his book *Fundamentals of Celestial Mechanics*, the difference between the heliocentric positions of celestial bodies and their relative positions in an Earth-centered system "are practically neglible for all but the nearest stars" (7). Centering the stars at the viewpoint introduces more star placement error than Earth-centered stars, but this error would only be significant at the outer portions of the solar system and would not be noticeable.

To center the starfield at the viewpoint, the initial "celestial coordinates" of each star are stored when the starfield is initialized. From (26), the position on the celestial sphere is determined by

$$x = \rho \cos\alpha \sin\delta \tag{24}$$

$$y = \rho \sin\alpha \sin\delta \tag{25}$$

$$z = \rho \cos\delta \tag{26}$$

where

$$\rho = \textit{the distance of each star from the Earth's center} \tag{27}$$

$$\alpha = \textit{the star's right ascension} \tag{28}$$

$$\delta = \textit{the star's declination} \tag{29}$$

Each star is positioned 75% of the distance to the far clipping plane by the equation

$$\rho = .75 \cdot FAR\_CLIPPING\_PLANE \tag{30}$$

This guarantees that they are never clipped, while still keeping them in the "background." Every time the "user pod" is updated, the "celestial coordinates" are added to the pod's new `xyz` position and the stars are repositioned. Note that the positions of the stars are independent of the `hpr` of the viewpoint. As the view changes heading, pitch or roll, the visible stars change accordingly. This design allows the user to place the viewpoint anywhere in the environment and still see the stars as a reference point.

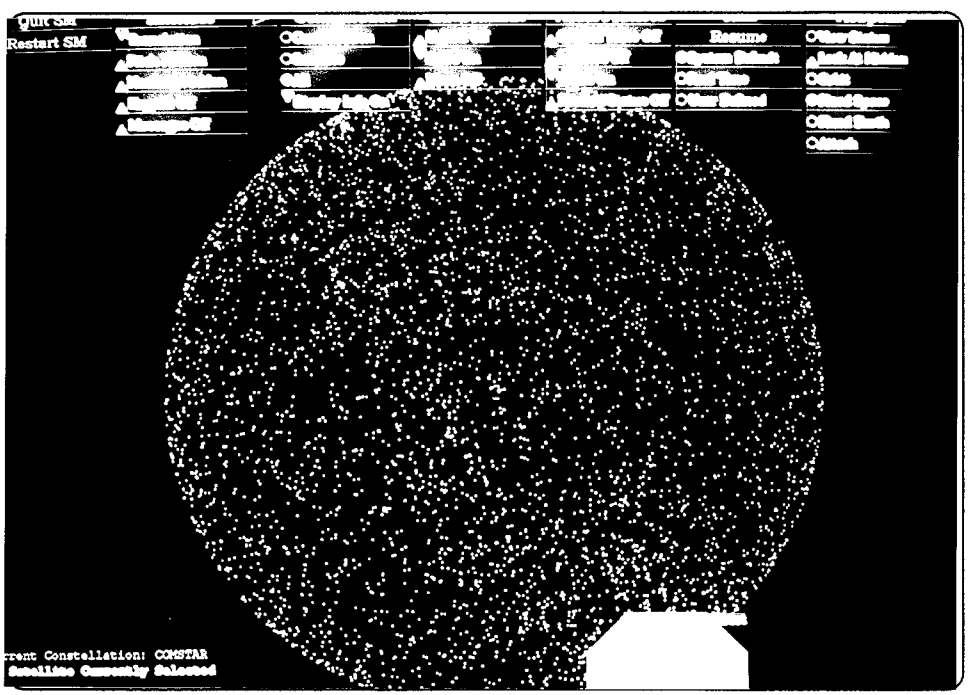


Figure 8. Celestial Sphere in the Old SM

4.4.5 The Sun. Adding a model of the Sun to this environment was fairly simple. The *Satellite Modeler* provided an algorithm for computing the solar position in ECI coordinates based on the current Julian time. The algorithm specified a direction for an infinite light source for generating realistic shading of the Earth and its satellites (26). To add a physical model, I derived a `Sun_Type` object class from the `Player` class in *ObjectSim*.

The `Sun_Type` object class inherits all the attributes for rendering geometry from its parent class. Using a simple 3D model of a yellow sphere, this class positions the model at its correct location and propagates it every frame. Because all objects in the environment are rendered in ECI coordinates, the Sun propagates "around" the Earth, while the Earth remains at the origin. As long as an accurate algorithm is used, the resulting motion of the Sun relative to a static Earth is the same as if the Earth were orbiting the Sun.

To compute the Sun's ECI position, I used the algorithm described in (36), which is similar to the method of deriving planetary positions from orbital elements described in Chapter II. This algorithm generates an accuracy of 0.01 degree and assumes a purely elliptical motion of the Earth. First, the current time is converted to Julian centuries from 1 January 2000. Based on that value, the geometric mean longitude and anomaly of the Sun are computed. Using the mean anomaly and the Sun's equation of center, the true longitude and anomaly can be found. Then, the current eccentricity of the Earth's orbit and the true anomaly are used to compute the distance from the Earth to the Sun. Finally, this distance R, the true longitude TL, and the eccentricity E are used to compute the ECI coordinates from the following equations:

$$Sun\_Position\_X = R \cdot \cos TL \tag{31}$$

$$Sun\_Position\_Y = R \cdot (\sin TL \cdot \cos E) \tag{32}$$

$$Sun\_Position\_Z = R \cdot (\sin TL \cdot \sin E) \tag{33}$$

To prevent the Sun's being clipped, especially when the viewpoint is near the outer planets, I never render the actual model beyond the viewing frustum. Each time the Sun's position is updated, I determine the distance from the pod (which is where the view is attached) to the new solar position. If this distance exceeds the viewing frustum then I force the Sun's model to be rendered just inside the far clipping plane. Thus, the entire propagation algorithm for the Sun can be described by:

```
1. Calculate_Solar_ECI_Position (Julian_Time);
2. Distance = distance from Pod to new Solar position;
3. if (Distance > Far_Clipping_Plane) then
     Distance = .97 * Far_Clipping_Plane;
```

```
4. Sun_Position = Distance;
```

Broadcasting the Sun's ECI position is trivial, once computed for the current time. From the DIS perspective, the Sun is treated as another "satellite" entity. Thus, it is initialized once as a DIS entity and then broadcast during its propagation, provided the simulation is running in "real-time" mode.

4.4.6 The Planets. As mentioned in Chapter II, there are several methods for calculating a planet's position, depending on the required precision. For a graphical depiction of the solar system, the computation complexity associated with the numerical integration method far outweighs the utility of its high precision results, especially since the available graphics hardware uses single precision (32-bit) values for rendering, as opposed to the double precision (64-bit) values used in the numerical integration method. On the other hand, using orbital element sets, which is the least precise method, does not require as much CPU time but would not provide the precision required for rendering accurate eclipses, planetary oppositions or planetary conjunctions.

For the *Space Modeler*, I used an orbital set algorithm and the method of summing periodic terms. As the default method for computing planetary positions, I used orbital element sets with a linear polynomial fit to approximate elliptical orbits. Even though this method is only accurate to within tens of thousands of kilometers, it is far less computationally expensive and produces acceptable visual results, considering that most of the planets are thousands of kilometers in diameter. For just observing relative positions and distances of the planets, this method is acceptable. When the user is "attached" to a planet, the method of summing periodic terms is used for computing that planet's position. Depending on the number of terms used in the series, this method can produce results accurate to within two degrees of heliocentric longitude (36:208). This design demonstrates that either method can be used without a severe degradation in the frame update rate. The orbital element set algorithm was tailored for use in the *Space Modeler*, but the summation of terms method was taken directly from (41).

Using software from the Jet Propulsion Laboratory as a basis, I implemented the orbital element set algorithm as follows. First, the current time is converted to Julian centuries from the epoch J2000. Based on the converted time, the non-angular true elements (semi-major axis and eccentricity)

are computed from the mean orbital values. Then, the angular elements of inclination, longitude of ascending node, longitude of perihelion and mean longitude are computed. Then, Kepler's equation is iteratively solved to find the eccentric anomaly. Finally, these values are transformed to ECI by subtracting the Earth's heliocentric position after which they are scaled appropriately. This algorithm is summarized by:

```
1. Compute_Heliocentric_Coordinates (Julian_Time);
2. Earth_Heliocentric_Position = - Solar_ECI_Position;
3. Planet_ECI_Pos = Planet_Heliocentric_Pos - Earth_Heliocentric_Pos;
4. Scale (Planet_ECI_Pos);
```

Because all the planets have similar attributes, one object class was designed to represent them. Each planet is instantiated with the name of its model file as well as the name of its orbital element set. The file containing the orbital element set is loaded into attributes representing the Keplerian elements and then used by the propagation method to update the orbit every frame.

Because of the relatively large distances between planets, it was important to implement a visual cue for the direction of the various planets whenever they are outside the viewing frustum. As each planet is propagated in its orbit, if its distance from the viewpoint exceeds the limits of the viewing frustum, a text label is rendered just inside the viewing frustum along the vector from the pod to the planet's location. The display of the text labels can be enabled or disabled via the user interface.

4.4.7 The Earth's Moon. Because accurate portrayal of the Moon's orbit was not critical in the *Satellite Modeler*, the Moon model was propagated in a circular orbit, parallel with the Earth's `x-y` plane, instead of its actual inclined, elliptical orbit (26). To improve this, I mathematically described the actual lunar orbit and propagated the model according to the algorithm in (36).

In (36), Jean Meeus provides a convenient algorithm for computing the position of the Moon for a given time. The algorithm requires the current time in Julian centuries from the J2000 epoch. Using the converted time (T) along with published lunar orbital elements, the Moon's mean longitude (L'), mean elongation (D), mean anomaly (M') and argument of latitude (F) are computed using the

following formulae:

$$L' = 218.31645 + 481267.88134T - 0.001326T^2 + T^3/53884 - T^4/6519400 \quad (34)$$

$$D = 297.85020 + 445267.11151T - 0.001630T^2 + T^3/54586 - T^4/1130650 \quad (35)$$

$$M' = 134.96341 + 477198.86763T + 0.008997T^2 + T^3/69699 - T^4/1471200 \quad (36)$$

$$F = 93.272099 + 483202.01752T - 0.003402T^2 - T^3/35260 + T^4/8633100 \quad (37)$$

Then, the sums of the periodic terms for the Moon's longitude (Σl), distance from Earth (Σr) and latitude (Σb) must be computed to allow the final calculation of the positional coordinates. There are hundreds of periodic terms in the Moon's longitude, latitude and distance from the Earth which must be considered to accurately compute a position. However, by limiting the computation to the three most significant terms, one can produce a position vector accurate to within 10 degrees in the longitude and 4 degrees in the latitude (36:307). For higher accuracy, more periodic terms may be considered. The final coordinates are computed via the following:

$$\lambda = L' + \Sigma l/1000000 \quad (38)$$

$$\beta = \Sigma b/1000000 \quad (39)$$

$$\Delta = 385000.56 + \Sigma r/1000 \quad (40)$$

These angular coordinates can then be transformed into geocentric rectangular coordinates. (All of these computations were implemented using commercially available software (41).)

4.5 The User Interface

The new user interface consists of three-dimensional panels that surround the view. These panels contain buttons which allow the user to modify attributes of the simulation, such as moving the viewpoint, toggling the display of visual features, and attaching to different entities. It assumes the presence of a head-mounted display or large-screen surround display as well as input devices such as a DataGlove®, 3D mouse, 2D mouse, spaceball, or BioMuse® controller for interacting with the

buttons. Because the panels "follow" the viewpoint as it is moved, it resembles a "space pod" in which the user is placed for interacting with the space environment. (See (24) for a complete discussion of the new user interface.)

To maintain the capabilities of the original *Satellite Modeler*, several user interface features had to be available in the new interface. These were:

- Setting the simulation time
- Moving view left, right, up, down, forward and backward
- Changing heading and pitch of view
- Toggling visual features, such as stars
- Attaching to satellites or moon
- Modifying satellite orientation
- Displaying satellite information

To provide these capabilities, the `Pod_Player` (or pod) for the *Space Modeler* consists of three panels. As Kestermann states, each *panel* comprises *sub-panels*, which contain the actual buttons for different pod capabilities (24). As Figure 9 illustrates, the *Space Modeler* pod contains a left, center and right panel, each derived from the `Panel_Type` object class. In addition, it contains a `Mouse` object for encapsulating the use of the computer mouse as an input device for controlling the panel cursor. The *pfSeg* attribute corresponds to a data type in the *Performer* library and represents the "pointing" device used for button selection, which in this case is the mouse cursor. The *pfChannel* attribute, which is also a *Performer* type, is used for displaying rendering statistics, such as the frame update rate. As Figure 11 shows, the left panel contains sub-panel classes for hiding the buttons on that panel (`Hide_Sub_Panel`), attaching to the different objects in the environment (`Attach_Sub_Panel`) and pausing the simulation (`Pause_Sub_Panel`). Each sub-panel instantiation, which is derived from the `Sub_Panel_Type` object class, contains the actual buttons for implementing the various capabilities. Each button can be instantiated with its own size, decoration, and arm/disarm colors. In addition, each of the sub-panels uses a configuration text file for defining the locations of the actual

buttons and button labels. As Figure 10 shows, the center panel comprises six sub-panels, in addition to the `Hide_Sub_Panel` and `Pause_Sub_Panel`. The `Rotation_Type`, `Translation_Type`, and `Zoom_Type` all encapsulate the movement controls. The `Panic_Type` sub-panel contains one button for returning the pod to its initial default position. The `Position_Sub_Panel` contains toggle buttons for displaying the pod's current position and orientation. Finally, Figure 12 depicts the organization of the right panel. The `Terrain_Features_Buttons` sub-panel contains the buttons for the visual features. The `Sat_Features_Sub_Panel` contains controls for satellite trails, locators and ground trace cones and the `Site_To_Site_Pod_Inteface_Type` sub-panel encapsulates the "transport" capability.

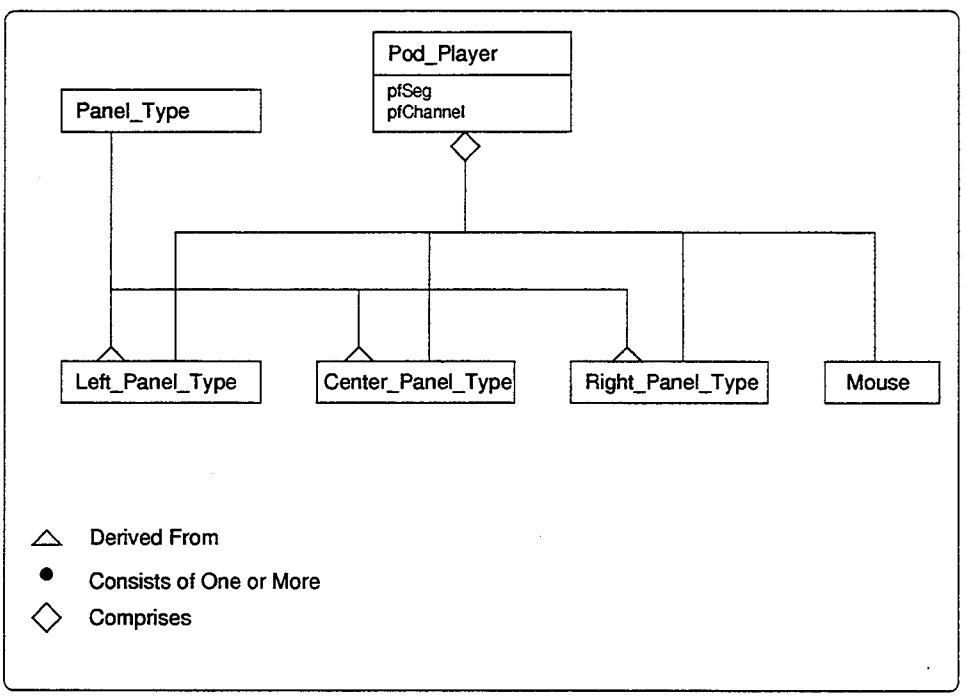


Figure 9. The `Pod_Player` Class

All the panels are fixed in relation to the user, so that by turning his head the user can bring the desired panel into view. The viewer enables a panel by looking at it, but can remove it from the field of view by clicking on a hide button for the panel. To avoid confusing the user, only one panel is "active" at any time. When active, the panel's border is highlighted and its buttons are visible. When not active, a panel is surrounded by a thin border, with a label describing its features. Figure 13 shows an inactive panel, and Figure 15 shows the same panel when it is active.

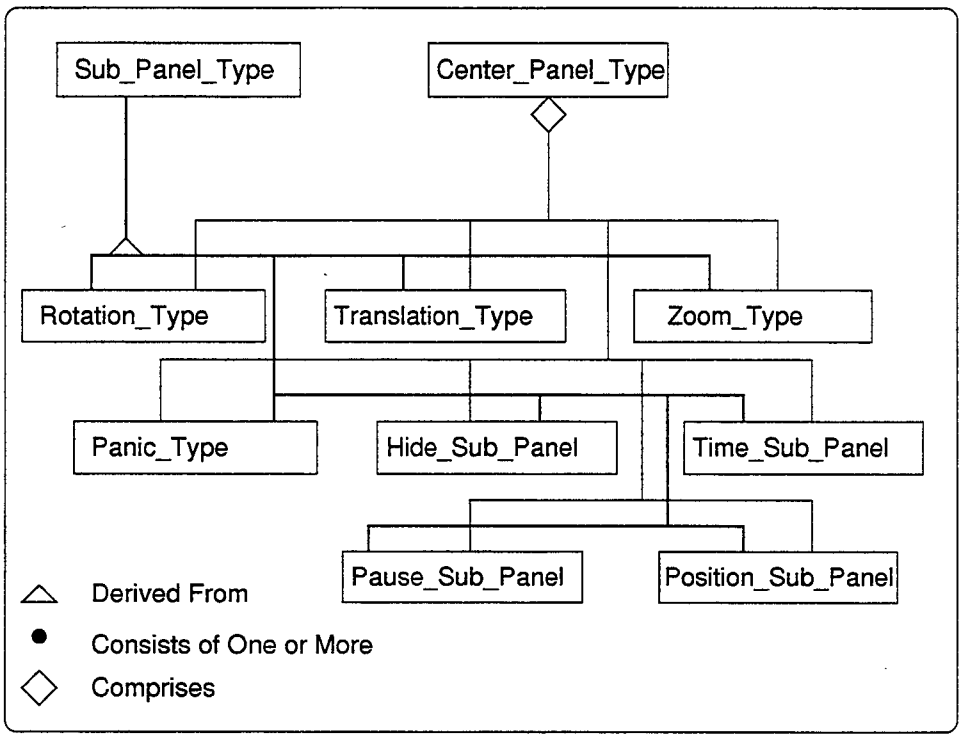


Figure 10. The `Center_Panel_Type` Class

4.5.0.1 The Center Panel. The center panel for the *Space Modeler* is tilted at an angle and located forward and slightly lower than the viewpoint, similar to an automobile's dashboard. This prevents the panel from obstructing the user's view while still allowing easy access. As Figure 14 shows, it contains the following controls:

- Heading and pitch changes
- Left and right movement
- Up and down movement
- Forward and backward movement
- Pod position display
- Simulation time changes
- Panic positioning

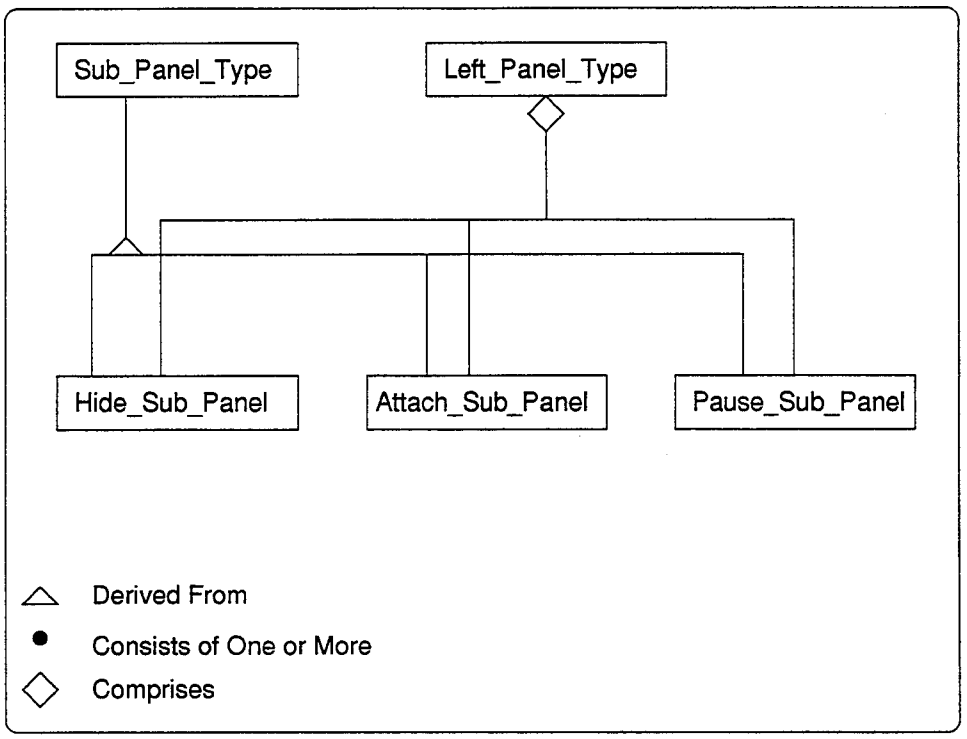


Figure 11. The `Left_Panel_Type` Class

In order to allow the user to move the view to any point in the environment, the pod provides movement controls for five degrees of freedom. There are three buttons associated with each direction, representing slow, medium and fast speeds. As long as each button is pressed, movement continues in the specified direction and speed.

Because it is extremely easy to get disoriented in a space environment, it was imperative that the interface provide positional feedback to the user. Based on work by Darken and Sibert, which demonstrated the importance of navigational feedback in a virtual environment (8), the user can toggle the display of the pod's rectangular coordinates via a button on the center panel. When activated, this information is displayed on the top portion of the scene, and follows the view similar to a Heads-Up Display. The information consists of an `xyz` location, using kilometers from the center of the Earth as units, and an `hpr` display, corresponding to the current orientation of the pod. All movement of the pod is relative to its current attitude. For example, left and right movement is relative to the current pitch and heading. However, the pod does not allow direct manipulation of its roll, in order to minimize

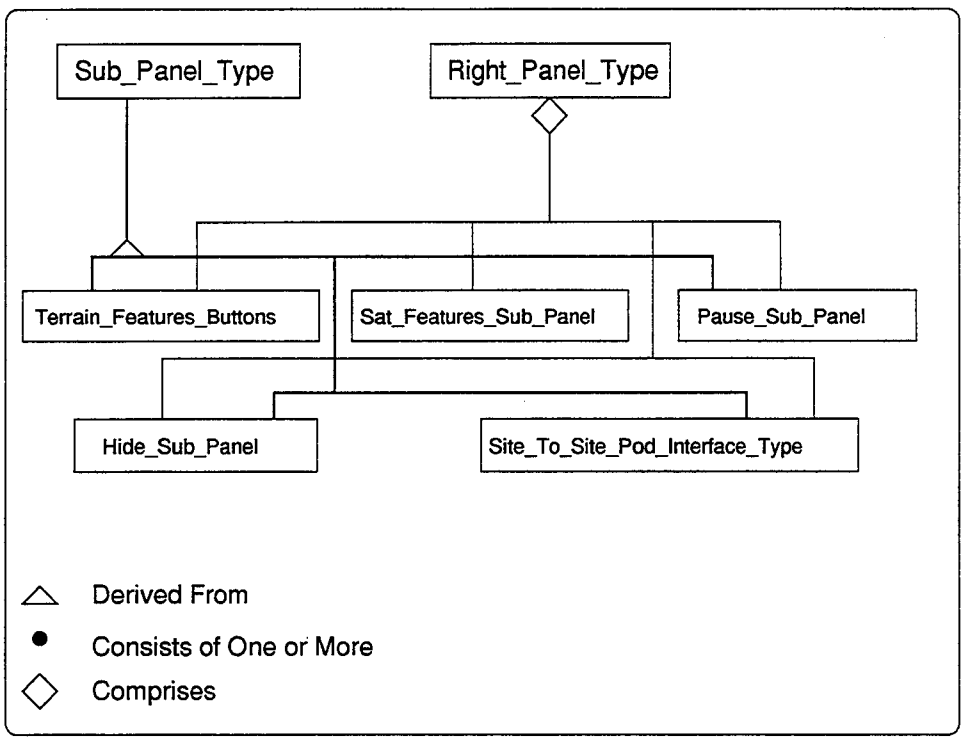


Figure 12. The `Right_Panel_Type` Class

Figure 13. An Inactive Panel

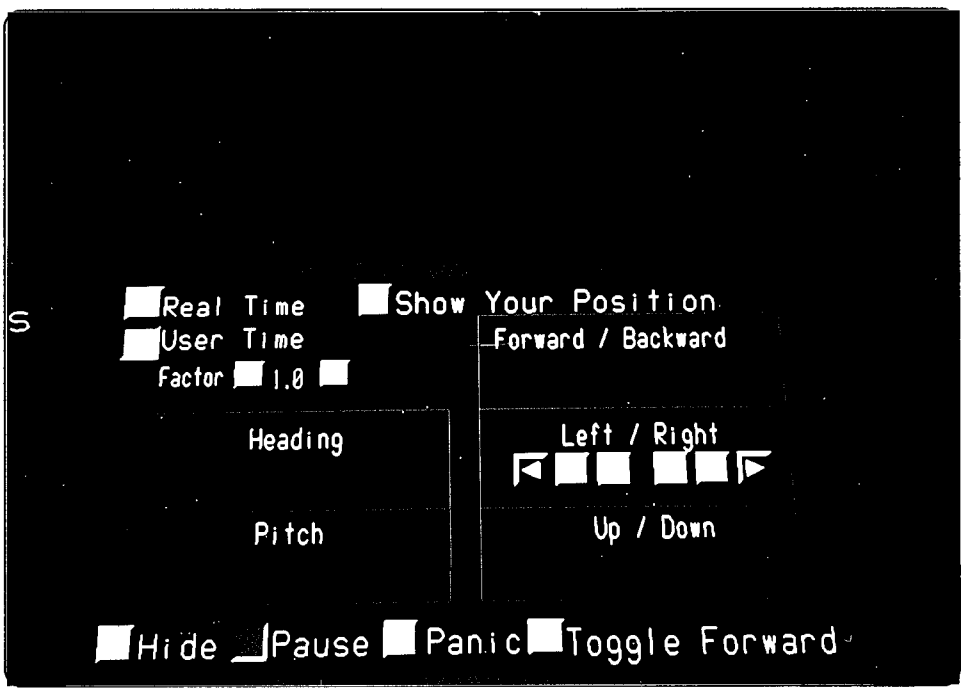


Figure 14. The Front Panel

the number of controls needed for pod movement and decrease the complexity of computing relative movement.

4.5.0.2 The Left Panel. The left panel is angled 45° to the viewer to facilitate access to its buttons. As Figure 15 shows, it contains the following controls:

- Detach from object
- Attach to Moon
- Attach to satellite
- Attach to planet

The user may attach to only one object at a time, so only one of the selections may be chosen at a time. When one attachment button is selected, all others are automatically deselected.

As Figure 16 shows, when the user is attached to a satellite, the following controls are displayed on the panel:

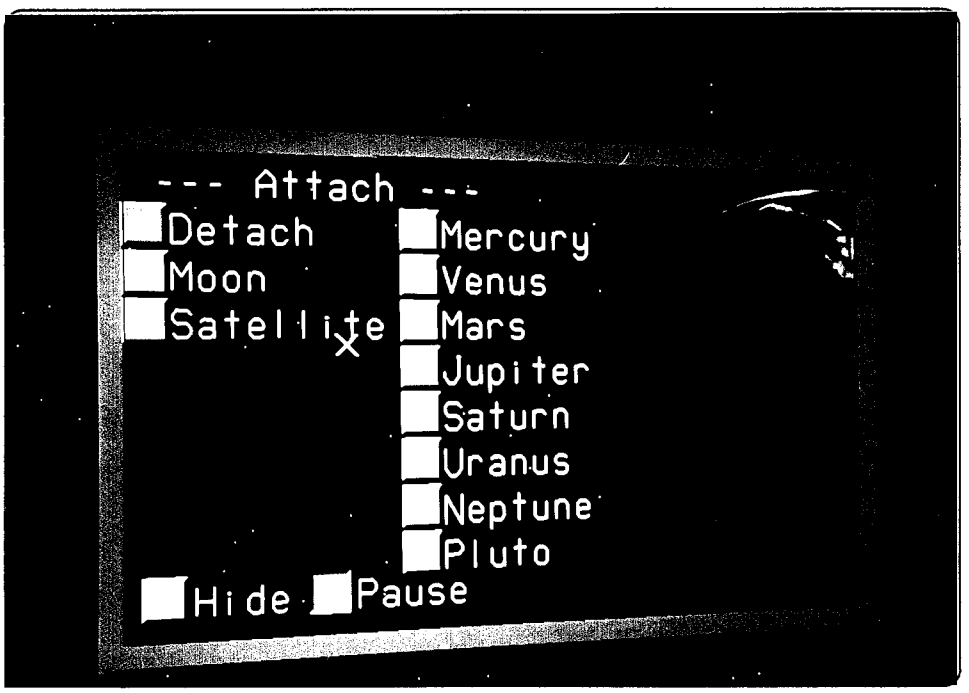


Figure 15. The Left Panel

- Change satellite
- Change constellation
- Change heading, pitch and roll of satellite

All of the satellites are organized into constellations. The simulation can have one or more constellations and a single constellation can contain one or more instances of a single type of satellite (26). Thus, there is an "increase" and "decrease" button for the satellite number as well as the constellation number. When attached to a satellite, the user can modify the heading, pitch or roll of the satellite to simulate attitude corrections. In addition, the panel displays the name of the satellite, the name of its orbit, the satellite's position relative to the Earth's center, and its altitude above the Earth.

To prevent disorientation, the pod is attached such that it is always facing the Earth. Because each satellite's heading, pitch and roll are updated to keep it in a nadir-pointed attitude, the pod must be explicitly positioned every frame, relative to the satellite's orientation. To properly position the pod "behind" the satellite, the following algorithm is used:

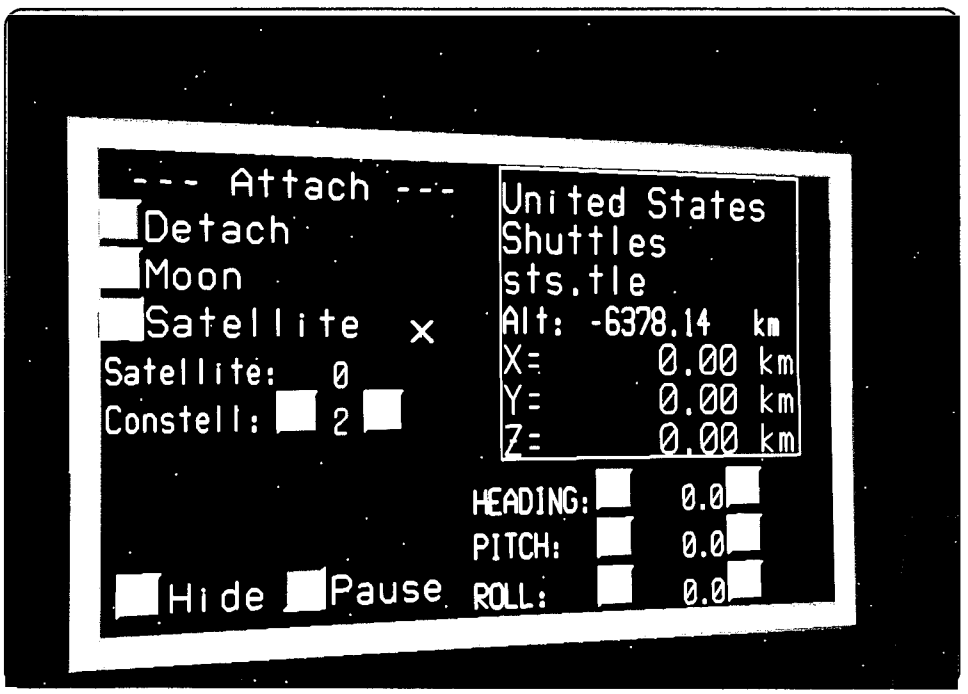


Figure 16. The Left Panel - Attached to a Satellite

```
1. Normalize the satellite's position vector;
2. Linearly scale result by some arbitrary offset so pod
    isn't attached to the middle of the satellite model;
3. Add that result to the original satellite position vector
    to get the "attach" point;
4. Orient the pod toward the earth;
5. Add in any user-input pod movements;
```

Figure 17 illustrates the derivation of the attach point, relative to the satellite's current position. Vector $\overline{P}$ represents the satellite's current position, vector $\overline{d}$ represents the offset distance based on the normalized $\overline{P}$ and the arbitrary offset n, and the point of attachment is the sum $\overline{P} + \overline{d}$.

Attaching to the Moon is accomplished in the same manner, except a larger offset is used since the Moon is so much larger than a satellite. However, a different approach is used for attaching to planets.

When the user attaches to a planet, the view is oriented toward the Sun, since, for most of the planets, the Earth is clipped from view. To attach to a planet, the following algorithm is used:

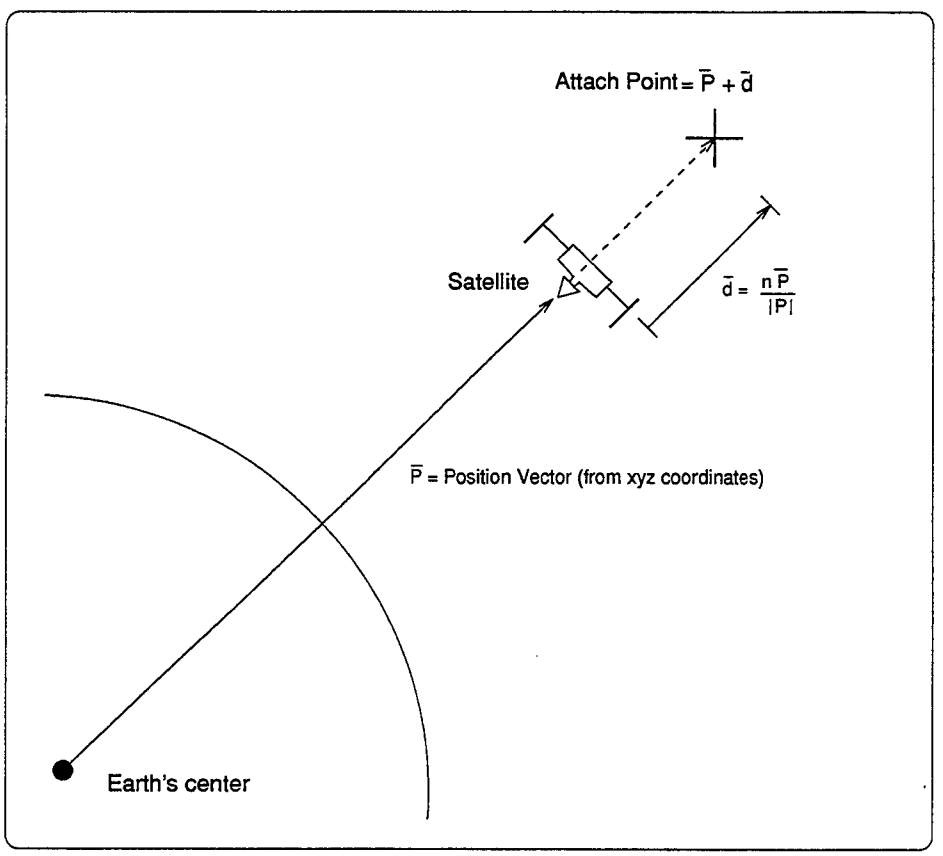


Figure 17. Attaching the Pod to a Satellite

```
1. Determine planet's position in heliocentric coordinates;
2. Normalize that vector;
3. Linearly scale result;
4. Add that result to the original heliocentric position vector;
5. Convert back to ECI coordinates;
6. Orient the pod toward the Sun;
7. Add in any user-input pod movements;
```

The simulation maintains the Sun's position in ECI coordinates, so it is trivial to transform ECI coordinates to heliocentric. The Earth's heliocentric position is simply the inverse of the Sun's ECI position. Thus, any ECI position can be converted to heliocentric by adding the Earth's heliocentric position. Figure 18 illustrates this relationship, along with the resulting attach point. Vector $\overline{p}$ represents the planet's heliocentric position, which is computed by $\overline{p} = \overline{p}' - \overline{S}$. Vector $\overline{S}$ is the Sun's ECI position and vector $\overline{p}'$ is the planet's ECI position.

Once in heliocentric coordinates, the attach point is determined in a similar fashion to the satellite attachment algorithm. However, because planetary size varies, a different offset is used for each planet. Transforming the attach point back to ECI coordinates (vector $\overline{A}$) is accomplished by adding it to the Sun's ECI position.

4.5.0.3 The Right Panel. The right panel is also angled 45^o to the viewer. As Figure 19 shows, it contains the following controls:

- Toggles for the display of:
  - Moon
  - Stars
  - Earth's umbra
  - Planet locators
  - Satellite inter-visibility links
  - Satellite ground trace
  - Satellite locators

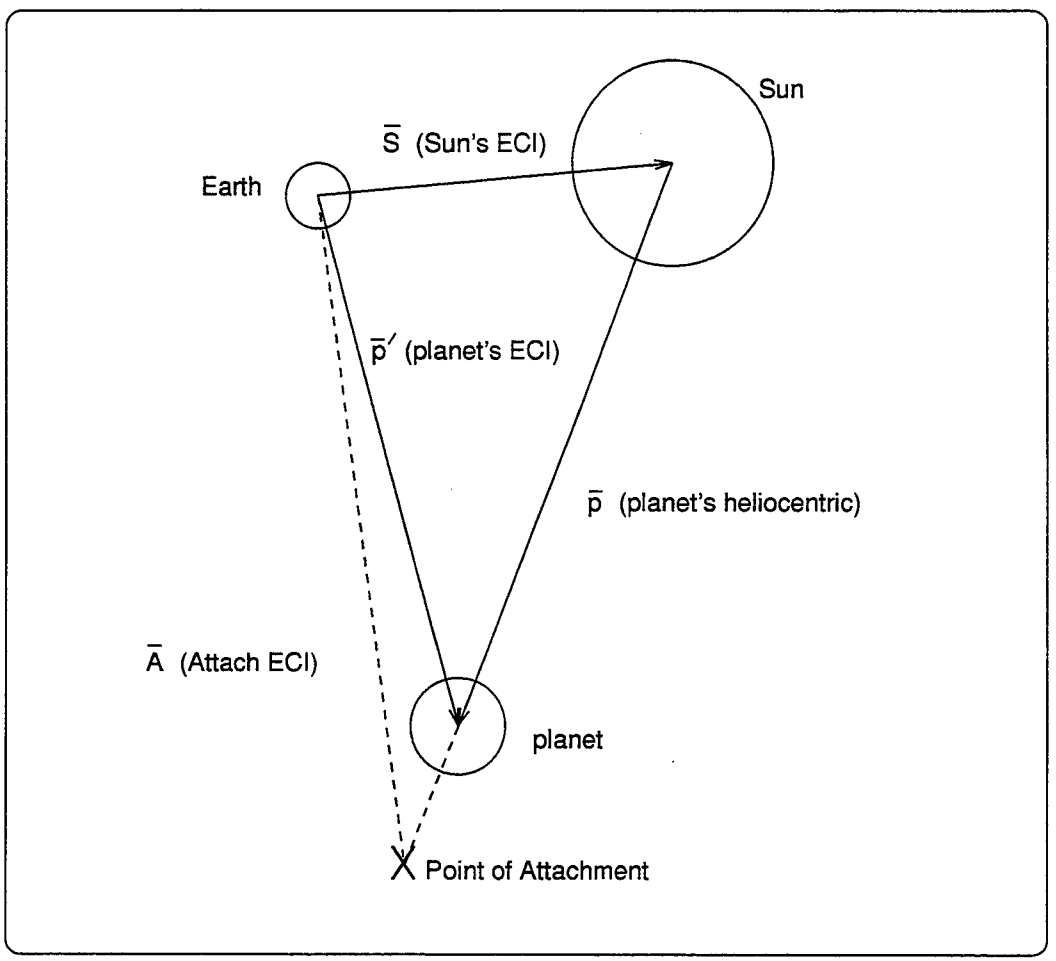


Figure 18. Attaching the Pod to a Planet

- Satellite trails

- Set length of satellite trails
- Reset satellite trails
- Transport the pod to user-defined positions

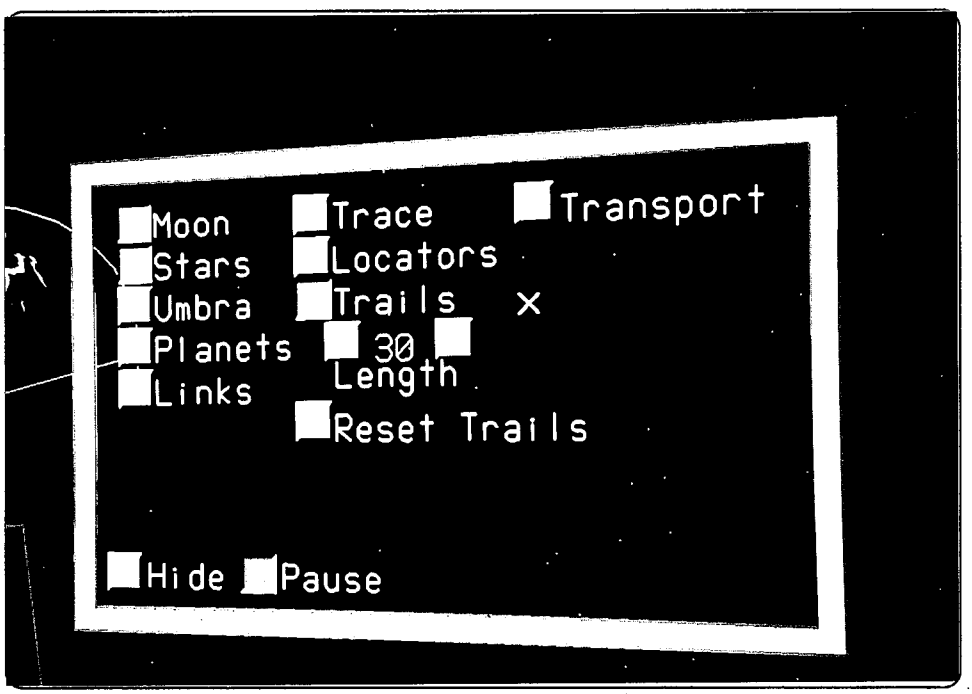


Figure 19. The Right Panel

Toggling the display of various visual features is critical to letting users customize the environment for their needs. As prescibed in (4), items should only be displayed when needed in order to avoid overwhelming the user. This design lets the user decide what is needed. In the *Space Modeler*, there is a button for each visual feature. To display a particular feature, the user presses the appropriate button. The button will stay pressed "in" until the user selects it again. As a visual cue for the user, when a button is pressed "in," it is green and returns to gray when pressed "out."

The toggle buttons determine if specific visual cues are rendered. Most of the toggles set flags in shared memory, which are used to insert or remove their respective objects to or from the *Performer* rendering tree. The `Moon`, `Stars`, `Umbra`, and `Trace` toggles work this way. The `Links` and `Trails`

toggles are used by the draw process to determine whether to calculate and render the cues for the satellite communication links and the satellite trails. The 3D lines representing the satellite links are computed using a standard algorithm for ray intersection with a sphere (19). The trails are rendered by transforming the stored world coordinates of the satellites into viewing coordinates. The `Locators` toggle for the satellites determines whether to render a locator "bubble" for each satellite when it is too far from the viewer to be seen. Similarly, the `Planets` toggle for the planet locators determines whether to display the text label for each planet.

Through the capabilities provided by the three panels, the new 3D user interface provides all necessary environment controls without obstructing the user's view. By not using valuable screen space and by rendering 3D interface panels, this interface is well suited to an immersive environment.

4.6 Conclusion

This chapter presented the software architecture of the *Space Modeler* and discussed the implementation of the various object classes. New object classes were needed for the Sun and the planets as well as for the new user interface. Using the *ObjectSim* framework, we built a modified simulation comprising instantiations of these new object classes. The next chapter presents the overall results and recommendations for future enhancements.

V. Results

In any computer graphics intensive application, the appearance of the rendered scene and the frame update rate are the two most important criteria for judging overall quality. In order to judge the appearance of the rendered scenes, this chapter presents several photographs of different simulation entities and maps those scenes to specific project requirements. Also, a summary of user feedback from a public demonstration of the *Space Modeler* is included. Finally, tabulated frame update rates are provided as a general performance metric.

5.1 Satellite Modeling

Because all the software for satellite propagation was taken from previous years' work, no attempt was made to validate the accuracy of any of the satellite orbits within the *Space Modeler*. In (26), Kunz spent ample time validating the satellite propagation methods from the *Satellite Modeler* from proven results from NORAD's SGP4 model. The software for the *Space Modeler* uses her SGP4 model with no modifications, so no additional validation was necessary.

As a general comparison to the *Satellite Modeler* application, Figure 20 illustrates the default view from the *Space Modeler*. It shows the Earth, with several satellites in orbit, along with the ground trace cones for each satellite. Note that in the *Space Modeler*, satellite trails are color coded to identify the different constellations.

To demonstrate that "tethered" views are still possible in the *Space Modeler*, Figure 21 depicts a view tethered to a GPS satellite. In "tethered" (or attached) mode, all movement of the pod is relative to the satellite to which it is attached. Since the *Space Modeler* uses the same orbital propagation code as the *Satellite Modeler*, the same two-line element sets can be modeled along with the same actual satellite models, namely the GPS, Molniya and DSCS satellites. In addition, the *Space Modeler* also has the capability of loading and rendering a model of NASA's proposed Freedom Space Station. Figure 22 illustrates the Freedom in low-altitude orbit.

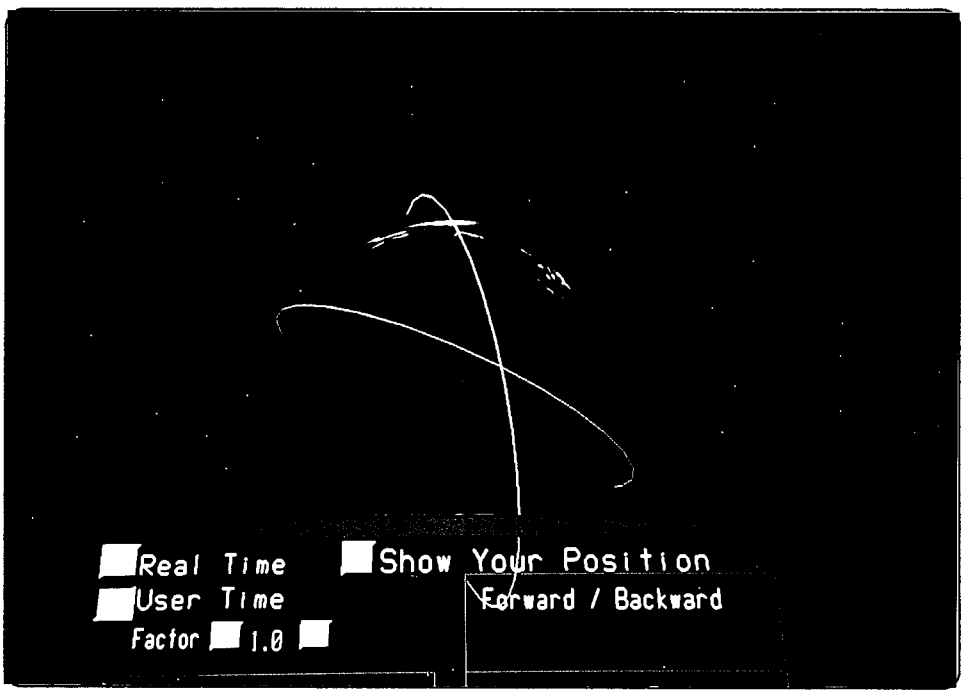


Figure 20. Default View for the *Space Modeler*

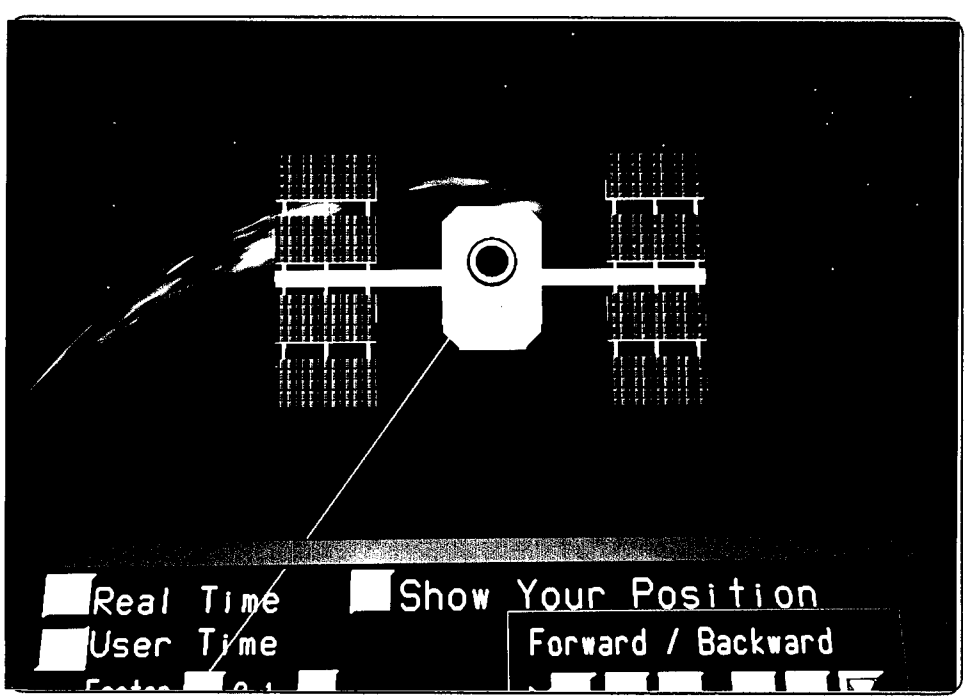


Figure 21. View Attached to a Satellite

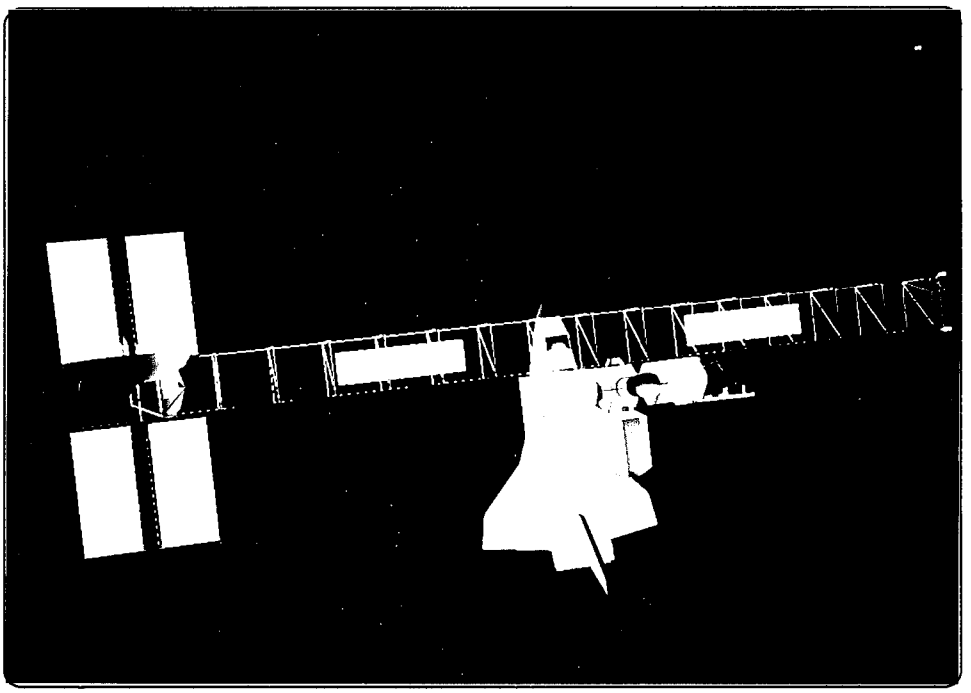


Figure 22. The Freedom Space Station

5.2 The Solar System

One of the major goals of the *Space Modeler* was to model the rest of the solar system. Using the attach/detach capabilities of the new user interface, it was easy to verify that all the modeled planets appeared in their correct position relative to the Sun and the Earth.

In order to validate the rendered positions of the Sun, Moon, and other planets, commercially available software from *Willman-Bell, Inc.* was used. This software, referred to as "the PC software", runs on an IBM-compatible personal computer and comprises several executables which compute complete ephemerides for the Sun, Moon, planets and other celestial bodies (41).

5.2.1 Sun. The Sun was modeled using a 180-polygon sphere, which was scaled appropriately relative to the world coordinate system. In order to prevent the facets of the geometry from showing, the Sun model was rendered without hardware shading. Thus, it appears in the simulation as a uniformly-shaded, light-emitting sphere.

The Sun's position, as rendered by the *Space Modeler*, was validated using the PC software. The initial solar position in ECI (geocentric mean equatorial) coordinates computed by the *Space Modeler* was displayed for three arbitrary dates. The positions were compared to those computed by the PC software and the results are shown in Table 1. The "Source" field entry is labeled "PC" to indicate the PC software and "SM" to indicate the *Space Modeler*. All locations are relative to the Earth's center and all times are Greenwich times. A slight difference in the manners in which the *Space Modeler* and the PC software computed Julian time accounts for the small difference in values.

Date	Time	Source	x (km)	y (km)	z (km)
15 Jan 1987	00:00:00	PC:	60550623.852	-123042999.194	-53349968.763
		SM:	60550623.557	-123042999.315	-53349968.815
02 Oct 1992	00:00:00	PC:	-147855290.999	-21613342.401	-9370953.105
		SM:	-147855291.054	-21613342.115	-9370952.980
06 Nov 1994	00:00:00	PC:	-107846253.049	-93391896.452	-40491640.210
		SM:	-107846253.271	-93391896.243	-40491640.119

Table 1. Comparison of XYZ Positions for the Sun

5.2.2 Moon. The Earth's moon can be viewed from two different perspectives. Figure 23 shows the view when attached to the Earth's moon. When attached to the Moon, the view is always oriented toward the Earth, thus the Earth is in the background. Figure 24 depicts the Earth's moon with the Sun in the background. This figure verifies the relative sizes and distances of the objects rendered. This view is from the vicinity of the Earth's surface and demonstrates that the Moon appears as big as the Sun, which is accurate as demonstrated in total solar eclipses.

To validate the rendered position of the Moon, I compared output from the *moon* program from the PC software to the rendered position in the *Space Modeler*. For each test, I chose an arbitrary date and time, ran the *Space Modeler* simulation, attached to the Moon, paused the simulation and used the displayed date/time and pod position as the reference point. Table 2 summarizes the `xyz` positions as reported by the PC software and those displayed by the *Space Modeler* for the indicated dates and times. The PC software outputs `xyz` positions with three decimal places of precision, while the *Space Modeler* only displays one decimal place of precision. Note that the results for 22 Jul 1994 are approximately

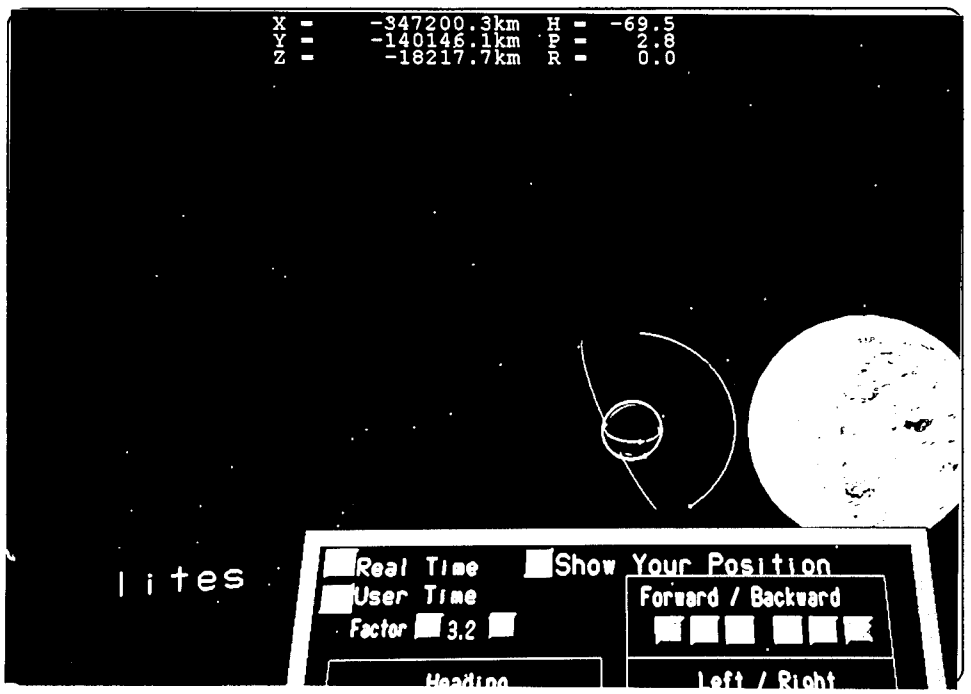


Figure 23. View Attached to the Moon

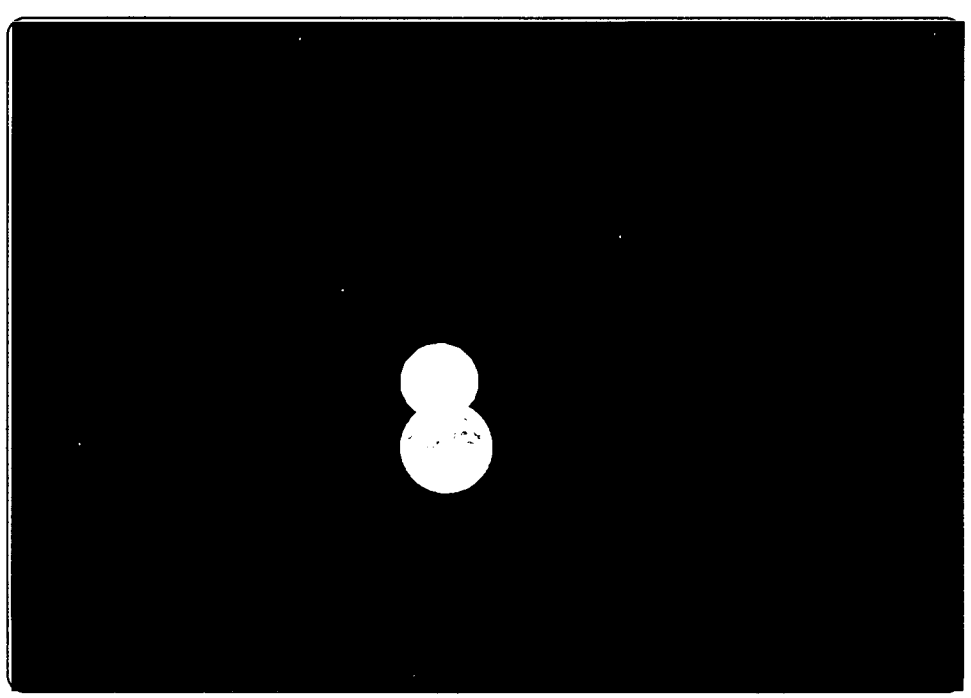


Figure 24. The Moon Against the Sun

the same as those for 25 Jun 1994, since the period of the Moon's orbit is approximately 27 days. These results indicate that the *Space Modeler* accurately models the Earth's moon.

Date	Time	Source	x (km)	y (km)	z (km)
25 Dec 1903	03:16:03.8	PC:	380949.332	-133665.925	12893.682
		SM:	380949.3	-133666.0	12893.7
25 Jun 1994	01:10:43.2	PC:	152256.411	-339131.344	29421.498
		SM:	152256.4	-339131.4	29421.5
22 Jul 1994	10:48:08.2	PC:	153539.671	-340254.553	29340.711
		SM:	153539.6	-340254.6	29340.7

Table 2. Comparison of XYZ Positions for Earth's Moon

5.2.3 Planets. To validate the rendered positions of the planets, I compared results from the three methods described in Chapter II. I used the *Space Modeler* to generate planetary coordinates based on orbital element sets with linear polynomial fits. Then, I used the PC software to generate the coordinates using a cubic polynomial fit as well as the VSOP87 method. Since the VSOP87 method is the most accurate, I used it as the reference for comparing the results from the other two methods. Some modifications were necessary for both the *Space Modeler* and the PC software to make this comparison successful.

The *Space Modeler* required two modifications for this comparison. First, I had to consider the attachment point for the individual planets. The *Space Modeler* uses a data file called `attach_points.dat` to determine the offset at which to attach to the different entities in the environment. I made the offsets for all the planets 0.0 to ensure that the pod position displayed on the HUD would correspond to the planet's position, when the pod was attached. Second, I had to modify the propagation of the planets' orbits. Recall that the *Space Modeler* uses a linear-fit implementation of the orbital element method for computing planetary positions when the pod is not attached, but uses the VSOP87 method when the pod is attached. For this test, I wanted to compare the linear-fit algorithm results with the others, so I changed the *Space Modeler* temporarily. To find the coordinates for the linear-fit algorithm, I ran the simulation in "real-time" mode and attached to the planet of interest. Then, I paused the simulation and read the results from the HUD position display.

The PC software also required two modifications. The `planet` program displays planetary information for a given date, time and location but only displays spherical coordinates. Since the *Space Modeler* uses rectangular coordinates, I had to modify the source file `planet.c` to convert those spherical coordinates to geocentric rectangular. By default, this program uses the VSOP87 method for computing positions. In order to compare the results of using a cubic-fit version of the orbital element set method, I had to add another computation. The software library provided with the PC software contains a function for computing positions based on the cubic-fit algorithm. So I just added the additional code to the `planets.c` source. The modified version of `planets` displays ECI coordinates for the planets based on the VSOP87 method as well as the cubic-fit version of the orbital element set method. (The version of `planets` used for this test is in `~mstytz/sax.planetary.code`.)

Table 3 lists the results of these comparisons. For each planet, the xyz position in kilometers is given in ECI coordinates for each of the three algorithms for an arbitrary date and time. The linear-fit algorithm is referred to as the "Chodas" method, the cubic-fit algorithm is referred to as the "Sax" method, and the method of using summation of periodic terms is referred to as the "VSOP87" method. Because of the way the cubic-fit algorithm was implemented in the PC software, it will not compute a position for Pluto. (See (36) and (41) for more information.) The numbers in parentheses next to each result represent the percentage error of that figure relative to the corresponding value for the VSOP87 method.

As for the visual results of rendering the planets, Figures 25 through 32 depict the rest of the planets, with the Sun in the background. Notice that the relative size of the Sun decreases as the view progresses to the outer planets.

5.3 User Interface

Measuring the overall success of this thesis effort was somewhat subjective since the quality of any virtual environment is directly related to how effectively the user is immersed and how naturally the user can interact with the environment. Fortunately, we had an opportunity to demonstrate the *Space Modeler* to the public at the annual Air Force Association Symposium in Washington, D.C. Over three

Planet	Date	Time	Coord	Chodas	Sax	VSOP87
Mercury	06 Nov 94	00:55:37.9	X:	-134653584.0 (0.05%)	-134653724.5 (0.05%)	-134582462.2
			Y:	-58982592.0 (0.08%)	-58983001.8 (0.08%)	-58938088.4
			Z:	-19317376.0 (0.11%)	-19316935.3 (0.10%)	-19296994.4
Venus	17 Mar 92	14:52:08.7	X:	207323104.0 (0.10%)	207323783.9 (0.10%)	207123102.7
			Y:	-88803136.0 (0.13%)	-88801871.8 (0.13%)	-88916141.7
			Z:	-43552140.0 (0.11%)	-43553405.4 (0.11%)	-43601287.8
Mars	02 Aug 87	03:41:45.6	X:	-290931968.0 (0.18%)	-290887258.0 (0.16%)	-290411670.1
			Y:	245942368.0 (0.24%)	245980219.0 (0.23%)	246542026.2
			Z:	115375728.0 (0.23%)	115393016.3 (0.22%)	115645593.6
Jupiter	09 Nov 89	14:47:22.6	X:	-126635368.0 (2.22%)	-126048916.2 (1.75%)	-123882826.1
			Y:	605871680.0 (0.08%)	605967128.6 (0.09%)	605375522.5
			Z:	259825440.0 (0.07%)	259900426.3 (0.10%)	259635875.4
Saturn	29 Jan 99	06:57:58.3	X:	1246125056.0 (0.06%)	1248282406.0 (0.23%)	1245438566.8
			Y:	617718976.0 (0.61%)	618408698.4 (0.50%)	621516735.0
			Z:	203368352.0 (0.76%)	203458333.8 (0.72%)	204930233.5
Uranus	03 Dec 90	04:42:50.9	X:	430904640.0 (1.26%)	469862703.5 (10.41%)	425555489.3
			Y:	-2752693248.0 (0.03%)	-2746847304.6 (0.24%)	-2753443973.1
			Z:	-1211646464.0 (0.05%)	-1209911488.2 (0.19%)	-1212224926.0
Neptune	19 Sep 93	09:13:38.3	X:	1412984064.0 (0.40%)	1376266558.6 (2.21%)	1407359324.0
			Y:	-3897096704.0 (0.08%)	3918816464.5 (0.48%)	-3900074751.3
			Z:	-1633981952.0 (0.06%)	-1641855489.8 (0.42%)	-1635032288.1
Pluto	20 Nov 94	22:01:30.1	X:	-2370081536.0 (0.06%)	N/A	-2371457770.3
			Y:	-3907410944.0 (0.03%)	N/A	-3906319376.1
			Z:	-542561920.0 (0.12%)	N/A	-543218608.3

Table 3. Comparison of XYZ Positions for the Planets

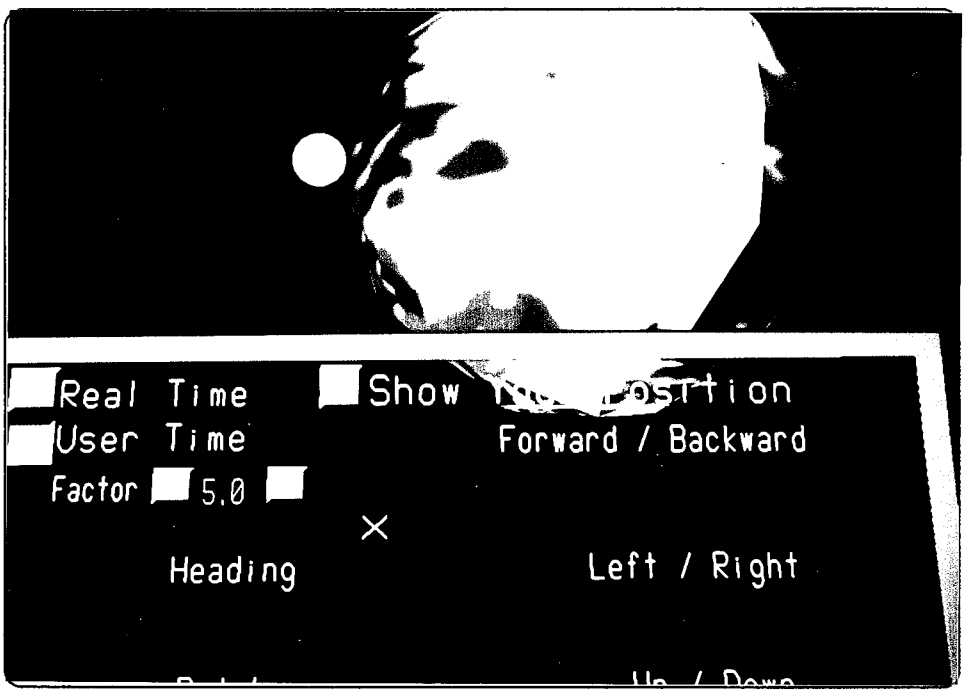


Figure 25. View Attached to Mercury

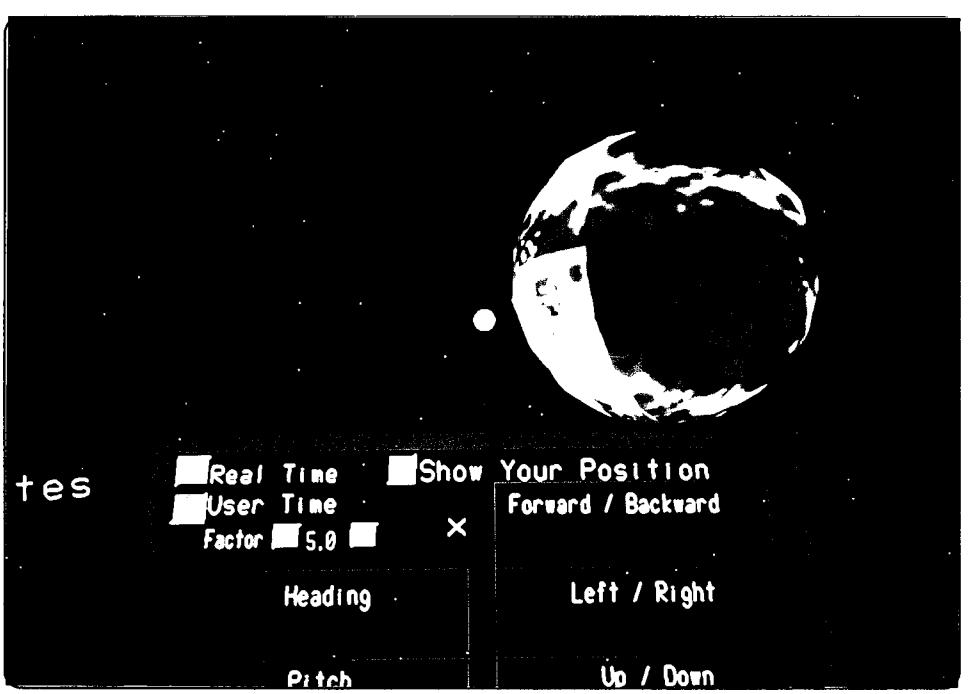


Figure 26. View Attached to Venus

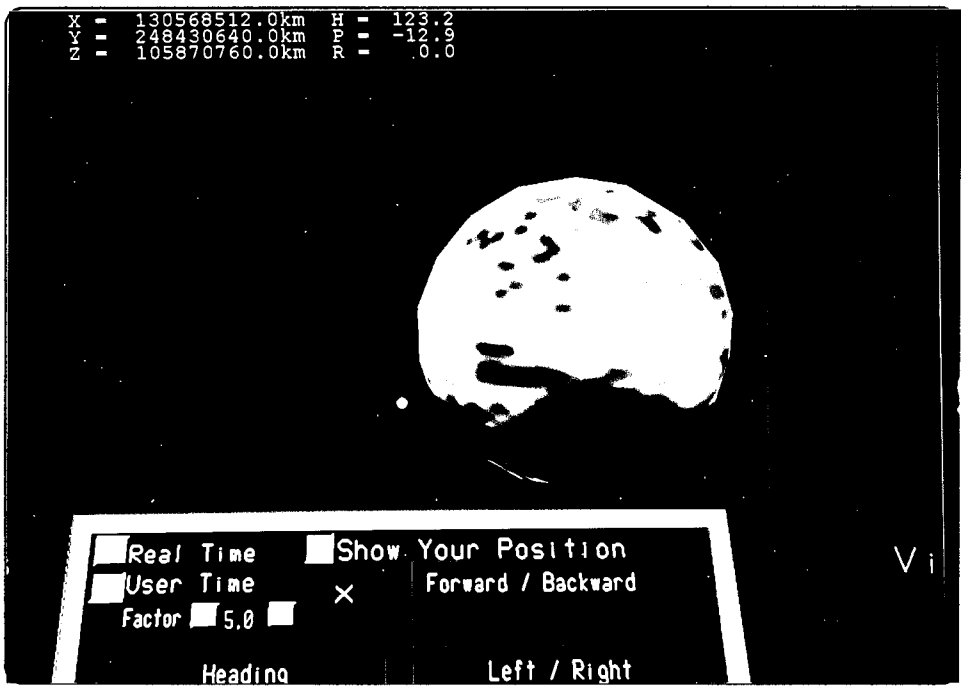


Figure 27. View Attached to Mars

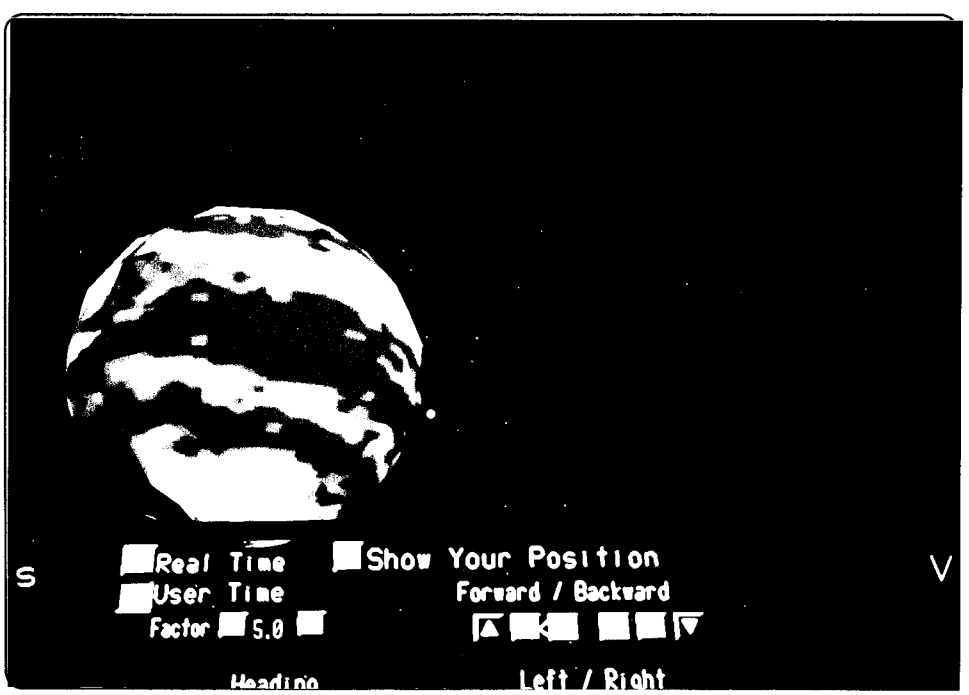


Figure 28. View Attached to Jupiter

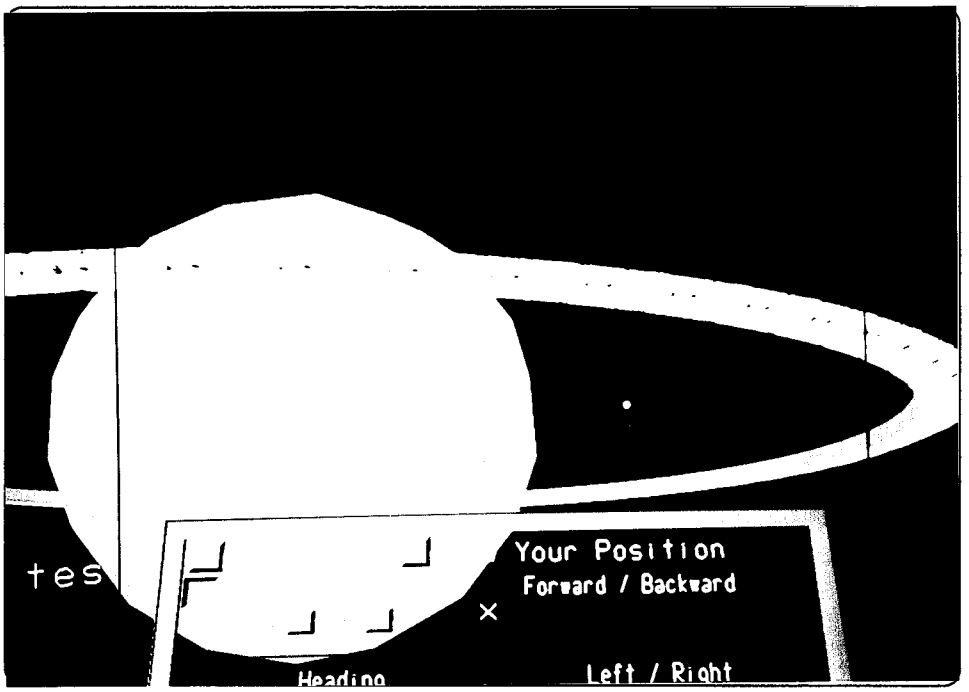


Figure 29. View Attached to Saturn

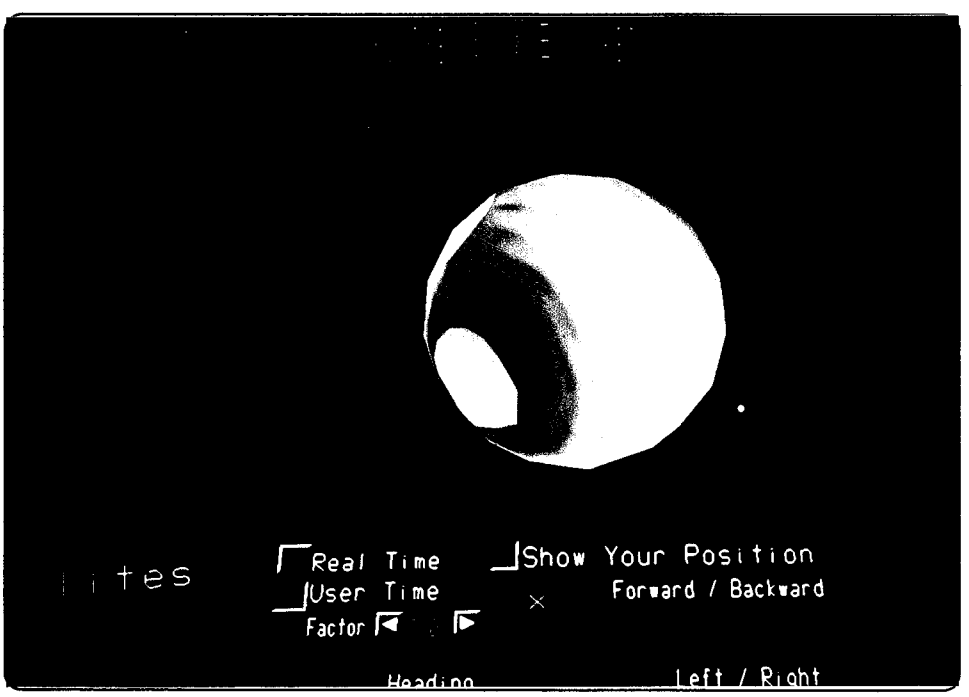


Figure 30. View Attached to Uranus

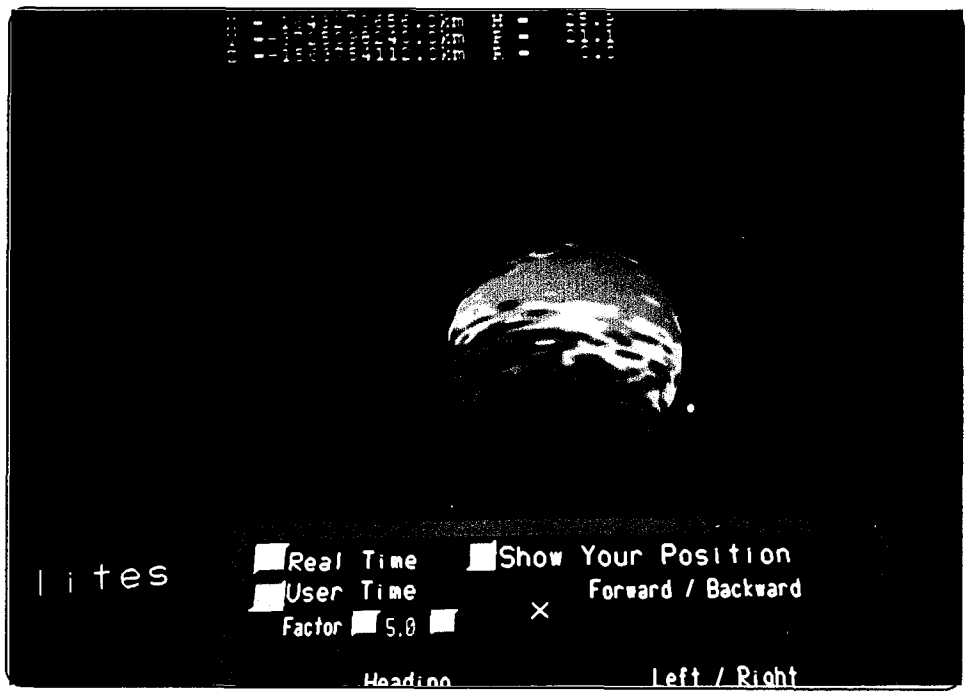


Figure 31. View Attached to Neptune

Figure 32. View Attached to Pluto

days, approximately 70 users entered and interacted with the *Space Modeler*. Most provided valuable feedback.

Several of the users had difficulty coordinating the use of the standard mouse with the HMD. A few users had no experience with a computer mouse and thus naturally had difficulty using the mouse to control the pointer on the user interface panels. Others had trouble keeping the mouse pointer in sight, especially when changing panels. However, after a few minutes of using the interface, most users were able to accomplish at least simple tasks, such as movement.

A few of the users were simulation experts who had substantial knowledge of virtual environment technology. Most of their feedback concerned improving the capabilities of the *Space Modeler*, such as allowing interactive picking via the mouse, modeling dynamic orbits, and computing collision detection. However, they were impressed with the basic simulation and especially appreciated the distributed simulation broadcasting capabilities.

Overall, most of the AFA users were impressed with the immersive interface of the *Space Modeler* and its rendering abilities. They appreciated the environment as a demonstration of current off-the-shelf virtual environment technology and recognized that this technology can easily be tailored to specific application needs.

5.4 Frame Update Rates

To measure the performance of this simulation, I ran four different simulation configurations on two Silicon Graphics Onyx machines. The configurations ranged from modeling a maximum of fifty satellites to a minimum of four satellites, using a broad range of two-line element sets for orbit propagation. The tests were run in an NTSC-size graphics window, as opposed to a full-screen window, in order to simulate the conditions under which the *Space Modeler* is run as an immersive application. (To drive an NTSC-driven HMD, the graphics window must be no larger than 640 pixels wide and 480 pixels high.) These benchmark tests certainly did not represent an exhaustive, precise measurement of the CPU performance of the simulation. They served only as a guideline for expected performance of the *Space Modeler* system.

The *Space Modeler* will execute on any of the Silicon Graphics machines, but will only execute with reasonable response time on the Onyx platforms. Thus, the two machines used for the benchmarks were:

- Machine A : Two-processor Onyx, with RE^2 graphics and one RM using *IRIX 5.2* and *Performer 1.2*
- Machine B : Four-processor Onyx, with RE^2 graphics and one RM using *IRIX 5.2* and *Performer 1.2*

The different simulation configurations represented the range of satellites that may be modeled by the *Space Modeler*. The first benchmark modeled four constellations, comprising a total of fifty satellites. Each satellite had a unique two-line orbital element set. The second bechmark modeled one constellation of twenty-five Global Positioning System (GPS) satellites. The third and fourth benchmarks modeled four constellations comprising a total of four unique satellites. The only difference between these two was that the fourth benchmark used the "molniya" two-line element set for one of the satellites. Because this particular orbit was extremely elliptical, it required more processing time than the other orbits (as measured by these benchmarks). (See Appendix A for complete listings of each benchmark configuration.)

The results of the benchmark tests are summarized in Table 4 and Table 5. For each benchmark, the test was run with and without the *Polhemus* head tracking system, in order to demonstrate that the serial I/O of the *Polhemus* incurred additional CPU time. For each test, an average time spent in the application, cull and draw threads, per frame, is given. Finally, an overall average frame rate is provided. All of these values were produced via the `pfDrawChanStats` method provided by *Performer*.

An ideal minimum frame rate, especially when considering the "lag time" of a head tracking system, is 12.0 to 15.0 HZ. As seen from these tables, the best performance for an immersive simulation is provided by modeling no more than twenty-five satellites on a four-processor Onyx machine. Additionally, if modeling the *Molniya* satellite orbit, the total number of satellites should not exceed five.

Benchmark	Number of sat's	Polhemus tracker	app (ms)	cull (ms)	draw (ms)	Frame Rate (HZ)
1	50	no	133.4	5.9	87.7	7.8
		yes	144.8	1.8	61.8	6.7
2	25	no	71.3	1.9	66.1	12.5
		yes	99.3	2.6	62.0	10.3
3	4	no	49.6	1.9	54.4	12.5
		yes	78.4	2.0	61.5	11.0
4	4	no	93.1	1.5	65.8	9.8
		yes	112.1	2.2	55.8	9.8

Table 4. Performance Specs for Two-Processor Onyx

Benchmark	Number of sat's	Polhemus tracker	app (ms)	cull (ms)	draw (ms)	Frame Rate (HZ)
1	50	no	105.3	2.7	32.0	9.3
		yes	148.7	2.8	33.7	6.7
2	25	no	47.3	2.5	29.7	20.0
		yes	90.2	2.7	30.8	11.0
3	4	no	33.3	2.2	30.1	28.0
		yes	73.3	2.3	33.5	13.0
4	4	no	66.3	2.5	28.5	15.0
		yes	107.3	2.5	29.1	10.0

Table 5. Performance Specs for Four-Processor Onyx

5.5 Conclusion

Important aspects of evaluating a virtual environment application are examining the rendered scenes for realistic images, validating mathematical algorithms used to propagate entities, measuring the frame update rate, and soliciting feedback from potential users. This chapter presented photographs of various scenes from the *Space Modeler*, discussed the validity of its propagation algorithms, summarized user feedback and discussed the frame update rate based on various simulation configurations. The final chapter summarizes this thesis effort and provides recommendations for future work.

VI. Conclusions and Recommendations

6.1 Overview

This thesis presented the design and implementation of the *Space Modeler* virtual environment. This environment, which was based on the *Satellite Modeler* , expanded the physical boundaries of the *Satellite Modeler* by modeling the solar system and included a new three-dimensional user interface based on the paradigm described in (24). Chapter II discussed a history of virtual environment technology and presented the fundamentals of modeling celestial objects. Chapter III provided the general requirements of the *Space Modeler* and Chapter IV explained how those requirements were met by the system design and implementation. Chapter V summarized the results and offered validations of the algorithms that were used to model the celestial bodies. Although the *Space Modeler* is a useful virtual environment for modeling and observing satellite and planetary motion in our solar system, several enhancements to its basic capabilities should be made.

6.2 Recommendations for Future Work

Future work with the *Space Modeler* should concentrate on improving the space environment, improving satellite modeling and improving the user interface. The most important improvement to the overall environment is to change the world coordinate system to a Sun-centered (heliocentric) system. As mentioned, in order to save time and to use implemented code from the *Satellite Modeler*, the *Space Modeler* uses an ECI coordinate system. Unfortunately, this leads to erroneous images. For example, when the pod is near one of the outer planets, and is not attached to any object, the user should be free floating in space. However, the pod is actually attached to the Earth, in the sense that everything is rendered relative to an ECI coordinate system, so when a planet "floats" by, it may actually be the user "floating" by the planet. Because our solar system is Sun-centered, it would make more sense to render the environment that way. This modification would require applying an additional coordinate transform to all the satellite coordinates, the Moon coordinates, and the Earth's rotation but would simplify the propagation of the planets' orbits. The addition of more celestial bodies would also improve the space environment.

Based on the algorithms provided in (36) and the software designed by (41), many more celestial bodies can be accurately modeled. Meeus provides algorithms for describing the motion of comets, asteroids, and even the moons of Jupiter. In addition, there are mathematical models for describing the dynamics of the rings of Saturn. If the *Space Modeler* were to be tailored to outer space operations, these enhancements would add a great deal of realism to the environment, assuming that accurate geometry is used to represent the celestial bodies.

Better geometric models for the planets are needed, since the texture mapped models used by the *Space Modeler* are rather crude approximations of the planets. As mentioned, the one infinite light source that is used by the *Space Modeler* does not provide realistic shading for the planets. Since hardware shading is disabled for the planets, their crude textures are easily seen. With improved geometry and texture maps, local light sources could be implemented that would provide appropriate shading effects.

The rendering of satellites can be enhanced in several ways. Dynamic satellite orbits should be modeled, such as the Hohmann transfer for satellite rendezvous as well as orbit decay due to gravitational forces. The satellites should model sensor systems by displaying DIS entities that are located within their line of sight. This would require that a "receive capability" be added to the simulation. Because the other AFIT virtual environment applications all have DIS "receive" capability, it should be trivial to add that to the *Space Modeler*. Finally, the most useful enhancement to modeling satellite behavior would be to use live telemetry data from currently orbiting satellites to render the models. This would give ground controllers real-time visual feedback of satellite position and orientation in an immersive environment.

Finally, to make the user interface easier to use, research should be directed toward integrating true 3D input devices, such as the BioMuse® controller and the 3D Mouse. Table 6 summarizes all these suggestions.

Functional Area	Suggested Work	Possible Source
Space Environment	Change to heliocentric world coordinate system	
	Model comets and asteroids	see (36:pp 209–224)
	Model Jupiter's moons	see (36:pp 285–299)
	Animate Saturn's rings	see (36:pp 301–306)
	Incorporate better models of the planets	
	Implement correct shading of the planets	see (34)
Satellite Modeling	Model dynamic orbits	
	Model orbit decay	
	Incorporate satellite DIS receive capability	see (40)
	Implement DIS remote satellite ground station	
	Model non-orbiting satellites such as Voyager	
	Incorporate live telemetry data	
User Interface	Incorporate 3D input devices	

Table 6. Future Enhancements

6.3 Conclusions

The *Space Modeler* is the first truly immersive virtual environment that models the solar system, models satellites in near-Earth orbit, and can operate in a Distributed Interactive Simulation (DIS) environment. It was designed and built using off-the-shelf hardware and software and thus serves as an excellent demonstrator of current virtual environment technology. Enhancing the current capabilities of this system will only improve its utility as a space-based virtual environment for observation and interaction.

Appendix A. Benchmark Configurations

The *Space Modeler* uses the file `constellation_setup.dat` to establish how many satellite constellations to model. For benchmarking purposes, four configuration files were made. Each file is organized by constellation and lists the number of constellations, a constellation name, a country code, an RGB triple for defining its color of trails, the number of satellites in the constellation, the name of the geometry file, and the list of two-line element sets that describe the orbits.

A.1 Benchmark 1

```
4
Global_Pos
91
255 100 100
25
GPS.flt
./tle/GPSBI-09.tle
./tle/GPSBI-10.tle
./tle/GPSBI-11.tle
./tle/GPSBII-01.tle
./tle/GPSBII-02.tle
./tle/GPSBII-03.tle
./tle/GPSBII-04.tle
./tle/GPSBII-05.tle
./tle/GPSBII-06.tle
./tle/GPSBII-07.tle
./tle/GPSBII-08.tle
./tle/GPSBII-09.tle
./tle/GPSBIIA-10.tle
./tle/GPSBIIA-11.tle
./tle/GPSBIIA-12.tle
./tle/GPSBIIA-13.tle
./tle/GPSBIIA-14.tle
./tle/GPSBIIA-15.tle
./tle/GPSBIIA-16.tle
./tle/GPSBIIA-17.tle
./tle/GPSBIIA-18.tle
./tle/GPSBIIA-19.tle
./tle/GPSBIIA-20.tle
./tle/GPSBIIA-21.tle
./tle/GPSBIIA-22.tle

Glonass
92
0 255 50
10
Molniya.flt
./tle/glonass34.tle
./tle/glonass36.tle
./tle/glonass39.tle
```

```
./tle/glonass40.tle
./tle/glonass41.tle
./tle/glonass44.tle
./tle/glonass45.tle
./tle/glonass46.tle
./tle/glonass47.tle
./tle/glonass48.tle

Shuttles
91
100 100 255
3
Freedom.dwb
./tle/sts.tle
./tle/sts_1.tle
./tle/sts_2.tle

Miscellaneous
92
230 230 230
12
DSCSII.flt
./tle/alouette1.tle
./tle/comstar2.tle
./tle/comstar4.tle
./tle/dmsp5d2-5.tle
./tle/gorizont1.tle
./tle/gorizont2.tle
./tle/kristall.tle
./tle/kvant.tle
./tle/kvant2.tle
./tle/mir.tle
./tle/progressM14.tle
./tle/soyuzTM-15.tle

//----------------------------------------
// Format:
//   - Number of constellations
//   - Constellation Name
//   - Country Code (defined by enum entity_country)
//   - Trail color - R G B values
//   - Number of satellites
//   - Model name for satellite
//   - Two-line element data file (*.tle)
//     for each satellite
//----------------------------------------
```

A.2 Benchmark 2

```
1
Global_Pos
91
255 100 100
25
GPS.flt
./tle/GPSBI-09.tle
./tle/GPSBI-10.tle
./tle/GPSBI-11.tle
./tle/GPSBII-01.tle
./tle/GPSBII-02.tle
./tle/GPSBII-03.tle
./tle/GPSBII-04.tle
./tle/GPSBII-05.tle
./tle/GPSBII-06.tle
./tle/GPSBII-07.tle
./tle/GPSBII-08.tle
./tle/GPSBII-09.tle
./tle/GPSBIIA-10.tle
./tle/GPSBIIA-11.tle
./tle/GPSBIIA-12.tle
./tle/GPSBIIA-13.tle
./tle/GPSBIIA-14.tle
./tle/GPSBIIA-15.tle
./tle/GPSBIIA-16.tle
./tle/GPSBIIA-17.tle
./tle/GPSBIIA-18.tle
./tle/GPSBIIA-19.tle
./tle/GPSBIIA-20.tle
./tle/GPSBIIA-21.tle
./tle/GPSBIIA-22.tle

//----------------------------------------
// Format:
//   - Number of constellations
//   - Constellation Name
//   - Country Code (defined by enum entity_country)
//   - Trail color - R G B values
//   - Number of satellites
//   - Model name for satellite
//   - Two-line element data file (*.tle)
//     for each satellite
//----------------------------------------
```

A.3 Benchmark 3

```
4
Global_Pos
91
255 100 100
1
GPS.flt
./tle/GPSBI-09.tle

Glonass
92
0 255 50
1
Molniya.flt
./tle/glonass34.tle

Shuttles
91
100 100 255
1
Freedom.dwb
./tle/sts.tle

Miscellaneous
92
230 230 230
1
DSCSII.flt
./tle/alouette1.tle

//----------------------------------------
// Format:
//   - Number of constellations
//   - Constellation Name
//   - Country Code (defined by enum entity_country)
//   - Trail color - R G B values
//   - Number of satellites
//   - Model name for satellite
//   - Two-line element data file (*.tle)
//     for each satellite
//----------------------------------------
```

A.4 Benchmark 4

```
4
Global_Pos
91
255 100 100
1
GPS.flt
./tle/GPSBI-09.tle

Glonass
92
0 255 50
1
Molniya.flt
./tle/molniya.tle

Shuttles
91
100 100 255
1
Freedom.dwb
./tle/sts.tle

Miscellaneous
92
230 230 230
1
DSCSII.flt
./tle/alouette1.tle

//----------------------------------------
// Format:
//   - Number of constellations
//   - Constellation Name
//   - Country Code (defined by enum entity_country)
//   - Trail color - R G B values
//   - Number of satellites
//   - Model name for satellite
//   - Two-line element data file (*.tle)
//     for each satellite
//----------------------------------------
```

Appendix B. User Manual

This appendix describes how to operate the *Space Modeler* simulation. It will run successfully on any of the SGI platforms that are running *IRIX 5.2* or greater, but will have the best performance on one of the *Onyx* machines. Directions are provided for running the DIS daemons, using an HMD as well as configuring the simulation for different satellite scenarios.

B.1 Running the DIS Daemons

Because the *Space Modeler* is a DIS application, the DIS network daemons, `sgisend` and `sgirecv`, must be running. These daemons allow access to the DIS PDUs that are broadcast across the network. If the DIS daemons are *not* running when you execute the *Space Modeler*, you will see the following message:

```
Could not get semid: No such file or directory
Bus error (core dumped)
```

Several UNIX shell scripts are provided for interfacing with the DIS daemons. These scripts are located in `/rem4/afa94/bin` and are used as follows.

- **startnet**: starts both daemons and executes them in the background
- **stopnet**: stops both daemons
- **port**: indicates which port the daemons are using
- **statnet**: displays status information similar to the following:

```
IPC status from /dev/kmem as of Mon Nov  7 18:22:44 1994
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
Shared Memory:
m      0 0x53637444 --rw-r--r--     root      sys
m    101 0x00bac0ed --rw-rw-rw- jfortner      eng
m   2902 0xccbac0ed --rw-rw-rw- jfortner      eng
m   2903 0x00bac0ec --rw-rw-rw- jfortner      eng
m   2204 0xccbac0ec --rw-rw-rw- jfortner      eng
m   2205 0x000009a4 --rw-rw-rw- jfortner      eng
Semaphores:
s    290 0x00bac0ed --ra-ra-ra- jfortner      eng
s    221 0x00bac0ec --ra-ra-ra- jfortner      eng
jfortner 7290 1 0 Oct 31 30:18 ./sgisendd -b100 -p3000
jfortner 7291 1 1 Oct 31 20:59 ./sgirecvd -b100 -p3000
```

This information indicates how many shared memory segments are currently being used. The daemons require four shared memory id's and two semaphore id's. In addition the userid of the person who started the daemons will be displayed. (See the *IRIX* "man page" on `ipcs` for more information.)

B.2 Running the Space Modeler

To execute the simulation correctly, it must be run from a directory that contains all the necessary data files. The executable filename is `smpod` and the following data files must be available in the directory from which it is run:

- `./tle/*.tle` files - the two-line element files for satellite orbits
- `constellation_setup.dat` - the configuration for the simulation
- `time_parms.dat` - defines the start time for the simulation
- `*.flt` files - model files for planets, satellites, locators, etc. (MultiGen format)
- `*.rgb` files - textures for models
- `*.attr` files - texture attributes
- `*.obj` files - model files for satellites (SGI binary format)
- `*.dwb` files - model files for satellites (Designer's Workbench format)
- `starfield.tx` - definitions for all the stars
- `fastrak.dat` - configuration file for the HMD
- miscellaneous `*.dat` files - configuration files for the 3D panels

The default screen shows a view of the Earth behind part of the front panel. The following keys are functional:

- `Keypad 8`: increase view pitch
- `Keypad 2`: decrease view pitch
- `Keypad 6`: increase view heading
- `Keypad 4`: decrease view heading
- `Keypad 5`: restore original view orientation
- `Keypad +`: move viewpoint forward
- `Keypad Enter`: move viewpoint backward
- `UPARROW`: translate viewpoint up
- `DOWNARROW`: translate viewpoint down
- `LEFTARROW`: translate viewpoint left
- `RIGHTARROW`: translate viewpoint right
- `<F1>`: toggle display of rendering stats
- `<F2>`: switch view modifier to the *Polhemus* tracker
- `<F3>`: switch view modifier to the keyboard
- `<F4>`: decrease field of view (essentially zooms in)
- `<F5>`: increase field of view (essentially zooms out)
- `<F6>`: restore field of view to default
- `PKEY`: toggles pausing of the simulation
- `<ESC>`: exit the program

If the view is controlled by the HMD, the Keypad keys above do not have any effect on the view.

B.3 Using an HMD

The *Space Modeler* has been successfully run with three different HMDs: the *PT-01*, the *nVision* and the *Kaiser VIM*. All of the HMDs require the use of the *Polhemus* head tracking system.

B.3.1 Using the Polhemus. The *Polhemus* is an input device that uses an RS-232 serial connection to the host computer. Several 9-wire cables have been designed with DB9 connectors for interfacing the *Polhemus* to the SGI machines. Connect the *Polhemus* box to one of the SGI serial ports (preferable port 2 or 3) using one of these cables. Set the DIP switches on the *Polhemus* box according to the baud rate and the number of stations you would like to use. (See (38) for DIP switch settings.) Make sure the entries in the `fastrak.dat` file specify the proper port number and baud rate and make sure the last entry in the file is "1". (See the `fastrak.dat` file for more information.) Place the *Polhemus* sensor above and behind the user. We usually use the ladder in the lab to support the sensor. Then, turn on the *Polhemus* box, wait ten seconds to allow it to initialize, and then run the *Space Modeler* . You will be prompted to calibrate the *Polhemus*. Do so by placing the HMD in whatever position and orientation you would like to use as the origin and press the `<ENTER>` key. Use the `<F2>` key to make the *Polhemus* control the view.

B.3.2 Using the PT-01 HMD. The *PT-01* is the easiest and most portable HMD we have. It uses a "beltpack" connector for interfacing the HMD with the video signal from the computer. Both *Onyx* machines have coax ports which drive NTSC signals. Use a coax cable to connect one of these ports to the "beltpack" HMD connector. Simply turn the "beltpack" connector on and you should see an image in the HMD. Use a 640x480 window in the upper left portion of the CRT for displaying on the entire HMD. Attach a *Polhemus* tracker on top of the HMD for tracking head movement. The NTSC signal can be split using a "T" connector and piped through the VCR or TV for recording or monitoring the view.

B.3.3 Using the nVision HMD. The *nVision* HMD has the highest resolution of our HMDs, but is more complicated to use. First, use a "T" connector to pipe the BLUE RGB connector from the *Onyx* to the interface unit for the HMD. The BLUE signal from the computer must be connected to both the left and right GREEN inputs on the interface unit. Then, the computer must be put into a special mode for driving the appropriate video signal. This must be done remotely, since the graphics console will be unusable while the computer is in the special mode. To set up the special mode, log in to the *Onyx* as *root* and execute the `nVision-setup` shell script in the `/usr/gfx/ucode/RE/dg2/vof` directory. This script file sets the computer in a field-sequential, 640x480 mode. As soon as this is done, the login screen should be visible in the HMD. Simply log in, and execute the *Space Modeler* . To monitor the view on a TV screen, run the `make_ntsc_work` script in the `/usr/gfx/patch` directory. To return the computer to its default configuration, execute the `Load60HZ` shell script as *root*.

B.3.4 Using the Kaiser VIM HMD. To use the *VIM* HMD, connect all three RGB connectors from the computer to the interface box. Set the computer to the proper display mode by executing the `Loadlvim` script in the `/usr/gfx/ucode/RE/dg2/vof` directory. Unfortunately, while using this HMD, no NTSC signal can be generated simulataneously for display to a TV monitor. To return the computer to its default configuration, execute the `Load60HZ` shell script. Remember that these scripts can only be executed as the *root* user.

B.4 Configuring the Simulation

The number and types of satellites that are modeled are defined in the file `constellation_setup.dat`. This file defines the number of constellations and, for each constellation, the number of satellites, the model file for the satellites, the country code, the trail color, and the "tle" files for each satellite. Several configuration files have already been defined:

- **constellation_setup.dat.bench1** - represents the first benchmark configuration
- **constellation_setup.dat.bench2** - represents the second benchmark configuration
- **constellation_setup.dat.bench3** - represents the third benchmark configuration
- **constellation_setup.dat.bench4** - represents the fourth benchmark configuration
- **constellation_setup.dat.GPS** - eleven Global Positioning System satellites
- **constellation_setup.dat.twelve** - twelve assorted satellites in multiple constellations

To use any particular configuration, simply copy that file to `constellation_setup.dat` and run the simulation.

Finally, the start time for the simulation is defined in the file `time_parms.dat` file. Only the year, month and day are used to initialize the time.

Bibliography

1. Bate, Roger R., et al. *Fundamentals of Astrodynamics*. 180 Varick St, New York, NY 10014: Dover Publications, Inc., 1971.
2. Blau, B., et al. "The DIS (Distributed Interactive Simulation) Protocols and their Application to Virtual Environments." *Proceedings of the Meckler Virtual Reality '93 Conference*. 1993.
3. Bolt, Richard A. "Put-That-There: Voice and Gesture at the Graphics Interface." *Computer Graphics (SIGGRAPH 80)14*. 262–270. Association for Computing Machinery, July 1980.
4. Brown, Judith R. and Steve Cunningham. *Programming the User Interface - Principles and Examples*. New York, NY: John Wiley & Sons, 1989.
5. Chodas, Paul W. Member of the Technical Staff, Solar System Dynamics Group, Jet Propulsion Laboratory, November 1994. e-mail: paul.chodas@zeus.jpl.nasa.gov, address: MS 301-150, Jet Propulsion Lab, 4800 Oak Grove Dr, Pasadena, CA 91109.
6. Chung, James C., et al. "Exploring Virtual Worlds with Head-Mounted Displays." *Non-Holographic True 3-Dimensional Display Technologies1083*. January 1989.
7. Danby, J.M.A. *Fundamentals of Celestial Mechanics* (2nd Edition). Willmann-Bell, Inc., 1989.
8. Darken, Rudy P. and John L. Sibert. "A Toolset for Navigation in Virtual Environments." *Proceedings of the ACM Symposium on User Interface Software and Technology*. ACM Press, November 1993.
9. Diaz, Milton E. *The Photo Realistic AFIT Virtual Cockpit*. MS thesis, GCS/ENG/94D-02, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1994.
10. Ellis, Stephen R. "What Are Virtual Environments?," *IEEE Computer Graphics & Applications*, 17–22 (January 1994).
11. Erichsen, Matthew N. *Weapon System Sensor Integration for a DIS-compatible Virtual Cockpit*. MS thesis, AFIT/GCS/ENG/93D-07, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1993.
12. Eyles, Don E. *A Computer Graphics System for Visualizing Spacecraft in Orbit*. Technical Report N90-22952, Cambridge, MA: Charles Stark Draper Laboratory (NASA), 1990.
13. Feng, X., et al. "An Intelligent Control and Virtual Display System for Evolutionary Space Station Workstation Design." *Proceedings of the Fifth Annual Workshop on Space Operations Applications and Research (SOAR)*. 582–587. 1991.
14. Figueiredo, Mauro, et al. "Advanced interaction techniques in virtual environments," *Computers & Graphics*, *17*(6):655–661 (1993).
15. Fortner, Jon L. *Distributed Interactive Simulation Virtual Cassette Recorder (DIS VCR): A Datalogger with Variable-Speed Replay*. MS thesis, GE/ENG/94D-10, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1994.
16. Furness III, Thomas A. and Dean F. Kocian. "Putting Humans Into Virtual Space," *Armstrong Aerospace Medical Research Laboratory, Wright-Patterson AFB, OH* (1988).
17. Gardner, Michael T. *A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions*. AFIT/GCS/ENG/93D-09, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1993.

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

19. Glassner, Andrew, editor. *Graphics Gems.* Boston, MA: Academic Press, Inc., 1990.

20. Hagedorn, J., et al. "A Computer Graphics Pilot Project: Spacecraft Mission Support with an Interactive Graphics Workstation." *Proceedings of the SCS Simulators Conference.* 61–64. 1986.

21. Harvey, E.P. and R.L.Schaffer. "The Capability of the Distributed Interactive Simulation Network Standard to Support High Fidelity Aircraft Simulation." *Proceedings of the Thirteenth Interservice/Industry Training Systems Conference.* 127–135. 1991.

22. Hitchner, L. "Virtual Planetary Exploration: A Very Large Virtual Environment." *Workshop on Implementation of Immersive Virtual Environments.* 1992.

23. Huffer, Charles M., et al. *An Introduction to Astronomy.* Holt, Rinehard and Winston, Inc., 1967.

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

25. Kobayashi, H., et al. "Interactive Graphic Display of Satellite Orbital Information." *Proceedings of the Tenth International Symposium on Space Technology and Science.* 901–908. 1973.

26. Kunz, Andrea. *A Virtual Environment for Satellite Modeling and Orbital Analysis in a Distributed Interactive Simulation.* MS thesis, AFIT/GCS/ENG/93D-14, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1993.

27. Kunz, Andrea A. and Martin R. Stytz. "A Virtual Environment for Satellite Modeling and Orbital Analysis in a Distributed Interactive Simulation." *Proceedings of the 1994 Simulation Multiconference.* 55–60. The Society for Computer Simulation, April 1994.

28. Latta, John N. and David J. Oberg. "A Conceptual Virtual Reality Model," *IEEE Computer Graphics & Applications*, 23–29 (January 1994).

29. L.B.McDonald, et al. "Standard Protocol Data Units for Entity Information and Interaction in a Distributed Interactive Simulation." *Proceedings of the Thirteenth Interservice/Industry Training Systems Conference.* 119–126. 1991.

30. Logsdon, Tom. *The Navstar Global Positioning System.* New York: Van Nostrand Reinhold, 1992.

31. McCarty, W. Dean, et al. "A Virtual Cockpit for a Distributed Interactive Simulation," *IEEE Computer Graphics & Applications*, 49–54 (January 1994).

32. McDonald, L.B., et al. "The Standardization of Protocol Data Units for Interoperability of Defense Simulations." *Proceedings of the Twelfth Interservice/Industry Training Systems Conference.* 93–102. 1990.

33. McGreevey, M. *Virtual Reality and Planetary Exploration.* New York, NY: Academic Press, 1993.

34. McLendon, Patricia. *Graphics Library Programming Guide.* Silicon Graphics, Inc., 2011 N.Shoreline Blvd, Mt. View, CA 94039, 1991.

35. McLendon, Patricia. *IRIS Performer Programming Guide.* Silicon Graphics, Inc., 2011 N.Shoreline Blvd, Mt. View, CA 94039, 1992.

36. Meeus, Jean. *Astronomical Algorithms*. Richmond, VA: Willmann-Bell, 1991.

37. Ocampo, Cesar. "Computer Graphics Applied to the Visualization of Two-Body Orbital Mechanics Problems." *Proceedings of the 28th Aerospace Sciences Meeting*. 1990.

38. Polhemus, Inc., PO Box 560, Colchester, VT 05446. *3SPACE FASTRAK User's Manual*, November 1992.

39. Rimrott, F. *Introductory Orbit Dynamics*. Braunschweig, Germany: Frieder Vieweg & Son, 1989.

40. Rohrer, JJ. *Design and Implementation of Tools to Increase User Control and Knowledge Elicitation in a Virtual Battlespace*. MS thesis, GCS/ENG/94D-20, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1994.

41. Sax, Jeffrey, "Astronomical Algorithms Basic Source Toolbox Version 2.1." Distributed by Willman-Bell, Inc., 1994. PC-based software.

42. Smith, R. S., et al. "Dynamic spacecraft simulators in Military Space Ground Systems." *Proceedings of the Symposium on Military Space Communications and Operations*. 119–126. 1983.

43. Snyder, Mark. *ObjectSim Application Developers Manual*. Air Force Institute of Technology, WPAFB, OH, 1993.

44. Stroustrup, Bjarne. *The C++ Programming Language* (Second Edition). New York: Addison-Wesley Publishing Company, 1991.

45. Stytz, Martin R., et al. "Providing Situation Awareness Assistance to Users of Large-Scale, Dynamic, Complex Virtual Environments," *Presence*, 2(4):297–313 (1993).

46. Thorpe, J. "Warfighting with SIMNET - A Report from the Front." *Proceedings of the 10th Interservice/Industry Training Systems Conference*. 263–273. 1988.

47. Wilson, Kirk G. *Synthetic Battle Bridge: Information Visualization and User Interface Design Applications in A Large Virtual Reality Environment*. MS thesis, AFIT/GCS/ENG/93D-26, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1993.

Vita

Captain John C Vanderburgh was born on 9 Nov, 1966 in Dayton, Ohio. He graduated from Wayne High School in 1984. He attended the United States Air Force Academy and graduated with a degree in Computer Science in 1988. His first assignment was in Las Vegas, Nevada where he specialized in the design and development of graphical displays for real time data analysis. He entered the School of Engineering, Air Force Institute of Technology in May, 1993.

Permanent address: 8357 Timberwalk Ct.
Huber Heights, OH 45424

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

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

4. TITLE AND SUBTITLE
SPACE MODELER: AN EXPANDED, DISTRIBUTED, VIRTUAL ENVIRONMENT FOR SPACE VISUALIZATION

5. FUNDING NUMBERS

6. AUTHOR(S)
John C. Vanderburgh

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/94D-23

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

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Abstract

The *Space Modeler* is the first truly immersive virtual environment that models the solar system, models satellites in near-Earth orbit, and can operate in a Distributed Interactive Simulation (DIS) environment. It increases the capabilities of the 1993 *Satellite Modeler* by expanding the physical limits of the environment and by implementing a new three-dimensional user interface. Satellite orbits are modeled using NASA two-line element sets. The positions of the Sun, Moon and planets are computed using an algorithm based on planetary orbital element sets using a linear polynomial fit. For higher precision, the planetary orbits can be computed using an algorithm based on VSOP87 planetary theory. The user interface is a three-dimensional array of panels, with buttons for controlling all aspects of the environment. The interface is easily accessible to an immersed user, and provides interactive movement controls for five degrees of freedom, attachment to objects in the environment, setting the rate at which the simulation runs, and toggling the display of all visual features. The *Space Modeler* can model the solar system and render up to 50 satellites in near-Earth orbit using magnetic head tracking with a 20-30 Hz frame update rate.

14. SUBJECT TERMS
Virtual Environments, Space Visualization, Orbital Mechanics, Distributed Interactive Simulation, DIS, Computer Graphics, Immersive, HMD

15. NUMBER OF PAGES
90

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