12-2004

A Mobile Virtual Reality System

Neena Nambiar
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Neena Nambiar entitled "A Mobile Virtual Reality System." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Daniel B. Koch, Major Professor

We have read this thesis and recommend its acceptance:

Michael J. Roberts, Paul B. Crilly

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Neena Nambiar entitled "A mobile virtual reality system". I have examined the final electronic copy of the thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Dr. Daniel B Koch
Major Professor

We have read this thesis and recommend its acceptance:

Dr. Michael J. Roberts

Dr. Paul B. Crilly

Accepted for the Council:

Anne Mayhew
Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

A Mobile Virtual Reality System

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Neena Nambiar

December 2004

Acknowledgements

I take this opportunity to express my gratitude to my advisor Dr. Daniel B. Koch for letting me work on this project and providing me with invaluable guidance and support throughout my Master's program. I would like to thank Drs. Michael J. Roberts and Paul B. Crilly for serving on my committee.

I am grateful to the SafeSkies organization for the funding and the support during my Master's program.

I also thank Qi Lin for helping me with the WorldToolKit software. I would also like to thank Susan Beck for help with all my questions.

I specially thank my parents Ms. Sujatha and Dr. O.G.B. Nambiar, and my sister Dr. Seena Nambiar for encouraging me to pursue my interests. It is only with their love and belief in me that I have reached this point today.

Most of all I thank God for all the blessings he has showered on me.

Abstract

Virtual Reality is the simulation of real or imagined environments, which can be experienced in three dimensions width, height and length. VR simulations provide interactions with the virtual world using external devices.

Virtual Reality systems have been bulky and not suitable for mobile operations. This thesis aims at creating such a mobile virtual reality system, which is portable and lightweight. This system consists of a lightweight notebook or laptop computer. It also includes a head-mounted display, a wireless mouse, a pinch glove, a tracker and wireless networking. This system is built with off-the-shelf components. This system is described in this thesis. Chapter One gives the introduction. Chapter Two describes history and background. Chapter Three describes the hardware devices. Chapter Four explains the software module and Chapter Five gives the Future Work and Conclusions.

Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Motivation

The Applied Visualization Center (AVC) at the University of Tennessee, Knoxville has been working with the U.S. Transportation Security Administration (TSA) since 2000. The center has developed a system to simulate passenger, cargo and vehicle flow through secure areas in an airport. This helps planners design security checkpoints at the airports.

The personnel involved in organizing security at the airport can first design the checkpoints using this software. The various passenger, cargo and vehicle paths are also laid out. Security devices can be imported into the scene. Each of these devices has properties such as probability of detection, dimensions etc. The probability that any device will set of an alarm can be set by changing the probability of detection property. The simulation is started after these steps are taken. During the simulation, if a passenger, cargo, or vehicle line up is seen at any checkpoint, it can be redesigned so that the time taken for screening is reduced. After the simulation the statistics related to the various paths can be seen. Until now the simulation system can be used only at a fixed location.

The goal of this thesis is to design and develop a virtual reality system, which is mobile. The system is wearable and consists of a lightweight notebook computer, small head-mounted display, pinch glove, hand and head trackers, wireless mouse, GPS receiver, and wireless networking. This system can help a first responder engaged in security operations to access information and assist in security or emergency tasks.

1.2 Problem Statement

The aim of this thesis is to design a system to assist security personnel during real time security operations. The notebook computer has various input devices connected to it. The head-mounted display provides a stereoscopic view of an airport. Graphics images are displayed on a pair of screens on a head-mounted display. A tracking sensor, attached to the user's head and interfaced to the computer, tells the system where the user is

looking. The computer quickly displays a visual image from the vantage point appropriate to the user's position. Thus the user is able to look about the computer-generated airport view similar to the real world.

A GPS receiver is connected to the computer and sends the position of the user to the system. As the user moves these data change. The system analyzes the data, changes the view, and sends the updated view to the head-mounted display. A wireless mouse connected to the system allows the user to select and click on devices as and when required. A pinch glove is interfaced to the system. When there is contact between any two fingertips, data are sent to the system and the system then acts accordingly

These are the devices that are interfaced to the computer. The data sent or received by these devices is analyzed and processed through code written in C, and uses Sense8's WorldToolKit (WTK) environment. WTK is a library of functions, which are written in C, and can be used to build virtual worlds. WTK manages the tasks of rendering, reading input sensors, importing geometries, and a wide range of simulation functions.

1.3 Virtual Reality System Design

1.3.1 Information Flow in VR systems

Figure 1.1 is a representation of the flow of information in a virtual world [1]. The head-tracking device obtains information about the position of the user and the orientation of the user's head. The hand-tracking device detects whether the user touches any object in the representation. The information obtained from the above devices is sent to the system. The system processes the information received. This information is then used to update the rendering of the representation. This representation is then fed back to the user through the head-mounted visual display. Haptic devices, which produce a force feedback, may also be used in the system. A haptic device allows the user to feel the force from certain objects in the scene. Feedback from this device may be in the form of pressure or weight. Feedback from a haptic device is mainly in the form of *tactile displays*, which deal with feeling surface textures or sensing the temperature.

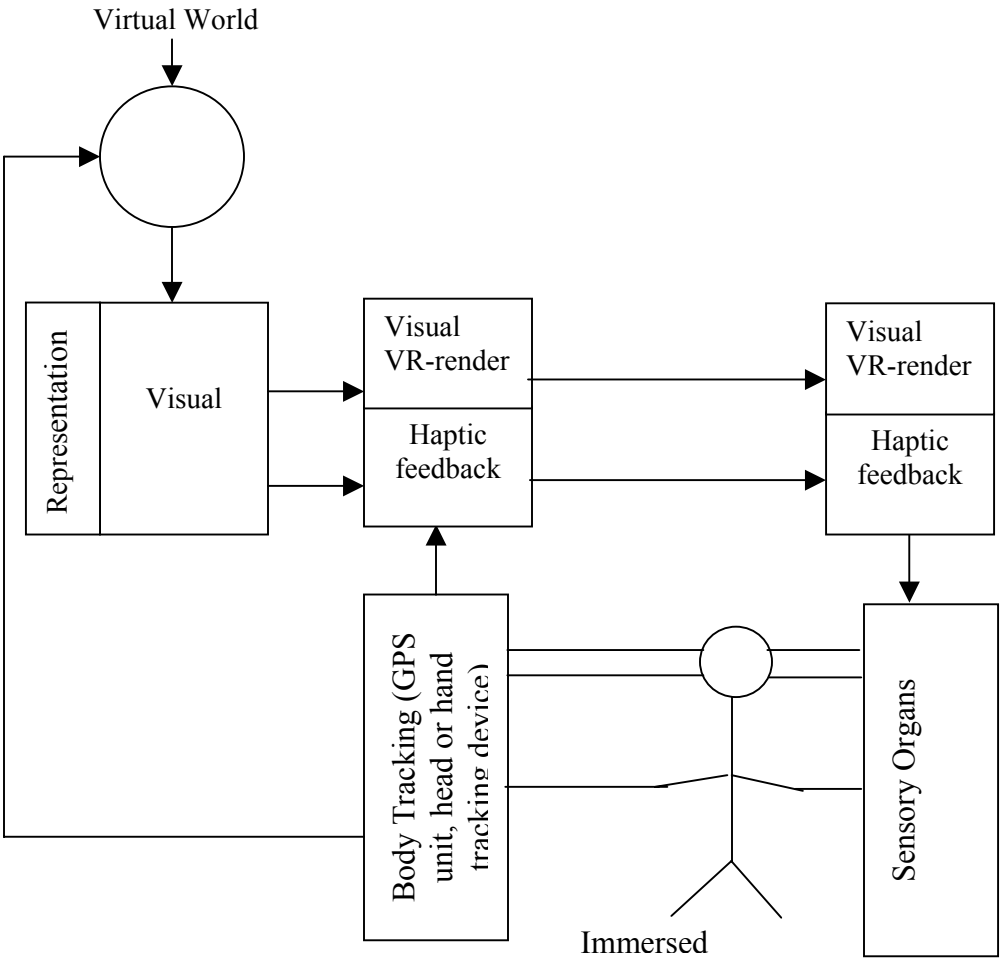

Figure 1.1. Flow of information in a VR system. [1]

The wireless mouse may be used to click on various objects in the scene representation. This act of selection of the object is also fed to the system, which may perform certain actions. The pinch glove may also be used for updating the scene. For the particular application of security, data from sensors in the real world are also fed back to the system and in turn fed to the user so the user can take action depending on the situation.

1.3.2 Components of VR Experience

The Figure 1.2 depicts a virtual reality experience diagram as a Venn diagram. The user interface, which forms an important part of VR experience, consists of hardware interface to the user and the software components, which enable hardware to communicate to the computer. The hardware interface in turn consists of input and output devices. The input devices are mainly the tracking devices, which detect the movement of the head or the hand and change the scene accordingly. The hardware interface may also consist of

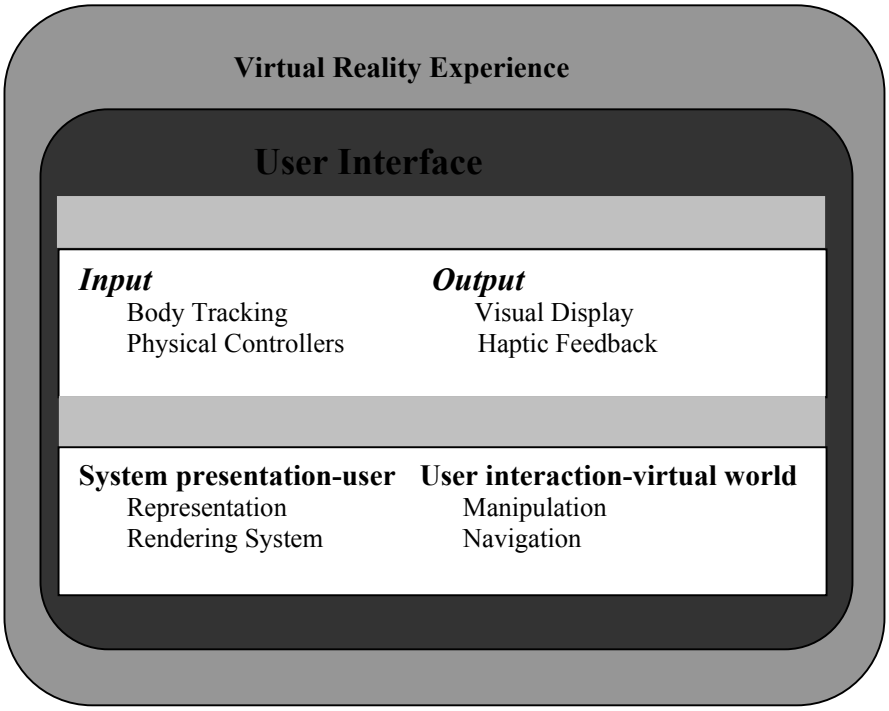

Figure 1.2. Components of a virtual reality experience. [1]

devices like the pinch glove or data glove, which convey contact information, or finger configuration information. The output consists of a visual display. The monitor and the head-mounted display are examples of a visual display. The haptic feedback includes the feedback from any haptic devices, which may be connected to the system.

The software components include the system presentation to the user and the user interaction with the virtual world. The system presentation to the user includes the representation and the rendering system. The representation of objects in a VR environment is done through models created in a modeling environment. The software manages rendering. For purposes of this thesis, WorldToolKit is used to take care of the rendering. The task of rendering is to transform representations of the virtual world into appropriate signals for the display system [1]. In Virtual Reality, rendering must be done in real time that is, at a rate that the human brain perceives as continuous [1]. The speed of the system depends mainly on the rendering capability. The user interaction with the virtual world consists of navigation and manipulation. The mouse and the pinch glove help with the manipulation of the objects in the virtual world. The mouse can be used to move around and navigate in the scene. The pinch glove can be used to perform functions, which can also be performed using the keyboard.

1.4 Implementation

The mobile virtual reality system has been implemented with off-the-shelf components. The head-mounted display, the pinch glove and the tracker are all available as VR gaming components. Initially the software was developed so that it can load the Tyson-McGhee airport model in the window. Also the mouse functions were developed, to allow the user to zoom into or out of the screen using the mouse. The head-mounted display was then interfaced to the system. A suitable driver for the graphics card then enabled stereo mode for the display.

The pinch glove was then interfaced to the computer. Software code was implemented to drive the pinch glove, send commands to it and interpret the received data. Functions of the pinch glove were then programmed. The wireless mouse was then attached to the system. At the end, the tracker was interfaced, to the entire system.

Similar to the pinch glove code, code to drive the tracker and code for the interpretation of the data received from the tracker had to be developed. This system has been integrated and successfully implemented.

1.5 Chapter Overview

Chapter 2 gives a brief introduction about virtual reality and augmented reality. Also the history and development of VR is presented. Future trends of virtual reality are also explained. Chapter 3 describes the various devices interfaced to the computer. It mainly describes the head-mounted display, the wireless mouse, the pinch glove and the tracker. An integrated system with a description of the IEEE 802.11 wireless standard is also described at the conclusion of the chapter. The software components used for these elements of virtual reality will be described in Chapter 4. Chapter 4 describes the modeling environment, and the issues related with it. In addition the WorldToolKit programming environment is explained, relating it to the various devices interfaced. This thesis will attempt to merge all these components to provide a virtual reality experience, which may be used for various purposes but mainly for security applications. The results, conclusions and future work will be presented finally in Chapter 5.

Chapter 2
Background

2.1 Virtual Reality Systems

2.1.1 Definition

Jaron Lanier, founder of VPL Research (1989) initially coined the term 'Virtual Reality' (VR). Virtual reality is a medium composed of an interactive computer simulation that senses the participant's position and actions and replaces or augments the feedback to one or more senses, giving the feeling of being mentally immersed or present in the simulation (virtual world) [1]. Virtual reality is a medium that allows a user to have a simulated experience approaching that of physical reality. There are four key elements of a virtual reality experience:

1. Virtual World: A representation of objects within a simulation is a virtual world.
2. Immersion: Immersion is the sensation of being in an environment. Physical immersion is the property of a VR system that replaces or augments the stimulus to the participant's senses. [1]
3. Sensory Feedback: A VR system should have the capability to provide feedback to one or more of the five senses. Most VR systems have the ability to give the user visual feedback. Some systems also have the facility to track the head of the participant and at least one hand or an object held by the hand in addition to the visual feedback.
4. Interactivity: For virtual reality to seem authentic it should respond to user actions, that is, to be interactive [1].

2.1.2 Terminology

Collaborative environment is an extension of VR and refers to multiple users interacting with the same virtual space or simulation. Users can perceive others within the simulation

allowing for mutual interaction. The users' representations are referred to as their avatars. [1]

Augmented reality is a virtual reality application designed to combine representation with perception of the real physical world [1].

Telepresence is the ability to directly interact with a physically real, remote environment from the first-person point of view [1]. With the help of remote sensing devices the participant is able to see, hear, and interact with the remote location from the first-person point of view. Transducers such as video cameras and microphones may be used as remote location sensors. Telepresence is closer to augmented reality than virtual reality since the participant is immersed in the real world rather than a computer-generated environment.

2.1.3 Devices Used

Various devices may be used to provide the effect of virtual reality. One such device is an HMD or a head-mounted display, which provides a stereoscopic view of the simulation scene. In the past, the design was such that HMDs often showed monoscopic pictures, and the user's head movements were tracked. In modern HMDs, stereo pictures are standard, because the technique has become cheaper. Inside 3-D HMDs, two computer screens direct displays from slightly different angles to each eye. The result is not only 3-D vision, but also the impression that the user has actually been transported to another world. A head-tracking device attached to the participant‘s head tells the computer system where the participant is looking [1].

Other such devices are voice recognition systems that allow one to interact with the world using voice commands. Tactile feedback may be provided by the use of a glove. This may allow one to feel or pick and move objects in the virtual world. Another form of a VR system may be a surgeon interacting with a virtual patient by manipulating medical instruments connected to a computer. The physician's hands are tracked and the computer provides information to the devices that provide haptic feedback (pressure and resistance) to the doctor's hands simulating the feeling of instruments [1].

2.1.4 Applications

VR has been used in many diverse areas for many different purposes. In various industries, the application of VR helps in reducing the cost and testing of ideas. It also helps in solving problems that cannot be tackled in the physical world, those that cannot be studied safely, and those that cannot be experimented with due to many different constraints. Applications for VR can be broadly divided into four different areas, mainly engineering, entertainment, science, and training.

In engineering, CAD and CAM tools have been used for designing 2D schematics of ASICs and PCBs; 3D CAD systems have been used for designing in various areas in engineering. Aircraft engines and submarines are designed using CAD systems. Maintenance of these systems can be made easier using virtual reality. Architects use VR to design new buildings. They can explore new techniques and new building materials using VR simulation. In addition they can also perfect lighting and acoustics for these buildings.

In entertainment, gaming machines have always been VR applications. Computer animation is used to model cartoon characters and experiment with their movements. In the television industry VR is used to simulate the sets with different lighting and sound settings.

In the field of science, VR has been used to visualize molecular structures, and to understand their functions related to their structures. VR has also been used to cure phobias, mainly acrophobia which is fear of heights. In the medical field a high-power rendering graphics machine with a high-resolution HMD is also being used for ultrasound echography to view the interior of a human body. This application however requires high rendering speed and low lag. Whenever a tracker is used in the system to detect head and hand movements, there is a delay between the movement and the system responding to this movement. This is called lag.

Training through simulation can provide a lot more benefits than other methods, since it reduces the cost and helps to monitor the progress of the trainee. Flight

simulators have been used for decades to help pilots acquire flying skills and for transition from one plane to another. In the field of medicine, VR is used to assist surgeons to develop surgical skills without having to harm animals or human beings. In the military, VR technology is used to guide soldiers to use more sophisticated weapons and improve upon their team working skills. [2]

2.2 History And Milestones

Virtual reality systems entered into the public domain after a considerable amount of research and development in industrial, military and academic laboratories. The birth of VR depended a great deal upon the existence of computers, which in turn depended upon the development of television technology. None of these would exist without electricity. The important milestones that led to the advent of current virtual reality technology are listed below:

1916: U.S. Patent for the first head-mounted display (head-based periscope) is awarded to Albert B.Pratt.

1929: Edward Link develops a simple mechanical flight simulator to train a pilot at a stationary location. [1]

1956: *Sensorama* was developed by Morton Heilig. Using this machine a single person could perceive a prerecorded experience, via sights, sounds, smell, vibration, and wind.

1960: Morton Heilig develops and receives a U.S. patent for Stereoscopic-Television Apparatus for individual use. This device resembles the head-mounted display of 1990s.

1965: Ivan Sutherland explains the concept of a display in which the user can interact with objects in some world that does not need to follow the laws of physical reality. The description included kinesthetic (haptic) as well as visual stimuli. [1]

1977: The *Sayre Glove* is developed at the Electronic Visualization Lab at the University of Illinois Chicago. This glove uses light-conductive tubes to transmit varying amounts of light to indicate the amount of finger bending. The computer interprets this information and estimates the hand configuration.

1989: On June 6th, VPL announces a complete Virtual Reality system introducing the phrase *virtual reality*. [1]

1992: Projection VR is introduced as an alternative to the head-mounted display at the SIGGRAPH'92 computer graphics conference in Chicago. CAVE (Cave Automatic Virtual Environment) was demonstrated at this conference. The Electronic Visualization Lab, at the University of Illinois, Chicago developed this technology. CAVE allows up to 10 people to share the visuals, with one person at a time having the optimal view. [1]

1997: Virtual Technologies, Inc. introduces their *Cybergrasp* hand-based force feedback device. This display allows the VR system to restrict the ability of the wearer to close individual fingers, enhancing the sense of touching and grasping in a virtual world. [1]

2000: The first six-sided CAVE in North America is installed at Iowa State University. [1]

Virtual reality can be used to model and explore familiar environments associated with kitchens, planes, offices, studios, ships, submarines, cars and hospitals. It can also be used to explore unfamiliar environments found in molecules, atoms, galaxies, viruses, bacteria and crystals. VR is also employed to understand abstract problems found in mathematics, economics, simulation, artificial life and networks.

2.3 Augmented Reality

In virtual reality the computer generates the environment. The user is immersed into this computer-generated environment. Opposed to this is augmented reality where the environment is the natural world. Into this environment, graphics, sounds, and haptics are added in real time. In addition to adding graphics to the real time environment, the user's head and eye movement are tracked so that the graphics displayed fits the perspective. Augmented reality emerged with Ivan Sutherland's experiments in the 1960s, in which he used a see-through HMD to present 3D graphics.

2.3.1 Head-mounted displays

Two types of head-mounted displays can be used for augmented reality purposes optical see-through and video see-through head-mounted displays.

The video see-through HMD consists of a pair of goggles, which blocks out the surrounding environment. Video cameras are attached to the outside of the goggles. The images from these cameras are transmitted to the inside of the display in real-time, and graphics are superimposed on these images. A drawback of the video see-through display however is a lag in the display. In addition, video see-through displays also have a parallax error, caused by cameras mounted away from the true eye location. The MR Systems Lab developed a relatively lightweight VGA-resolution video see-through display, with a 51-degree horizontal field of view, in which the imaging system and display system optical axes are aligned for each eye. [3]

Several companies are developing displays that embed optics within conventional eyeglasses. Eyeglasses produced by MicroOptical have two right-angle prisms embedded in regular prescription eyeglass lens. It reflects the image of a small color display, mounted facing forward on an eyeglass temple piece [3]. See-through displays have a number of drawbacks such as insufficient brightness, resolution and field of view. In addition, size, cost, and weight are still problems. The Figures 2.1 and 2.2 show the workings of the video see-through and optical see-through displays. Figure 2.3 gives an illustration of computer-generated graphics overlaid on a real world.

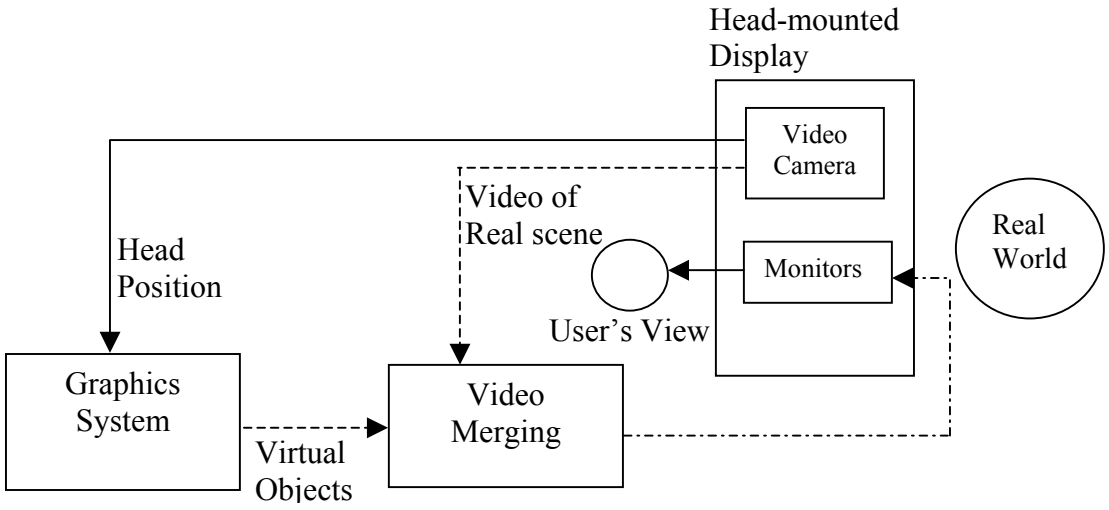

Figure 2.1. Video see-through augmented reality display. [4]

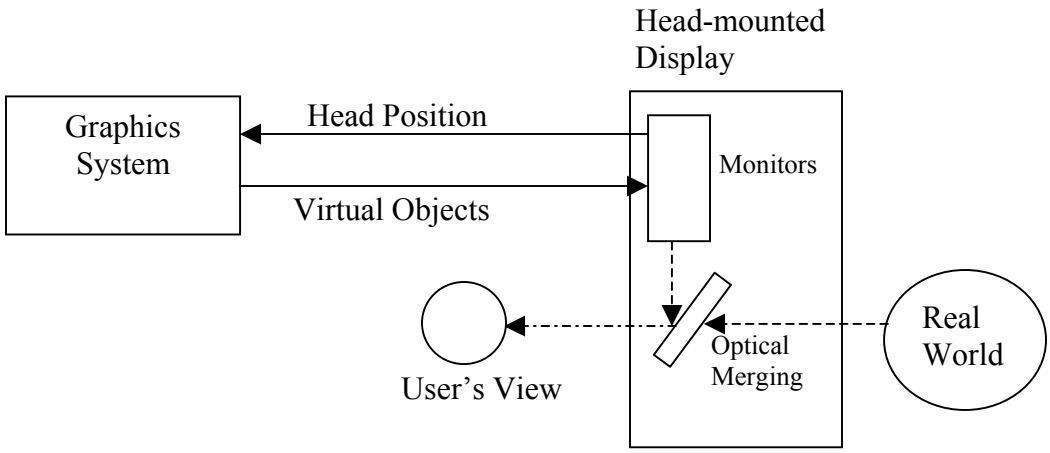

Figure 2.2. Optical see-through augmented reality display. [4]

Figure 2.3. Computer generated graphics are overlaid on the real world.

2.3.2 Tracking and Orientation

Tracking the user's position and viewing orientation is very important for augmented reality displays. Position tracking is required in an augmented reality system in order that the system renders the correct view of the world depending on the user's position and orientation. For an outdoor system, a GPS receiver attached to the system, which gets the longitude and latitude of a location, can obtain the position of the user. But, a GPS receiver on its own can only obtain the data to a precision of meters. Differential GPS can be used for such systems to obtain an accuracy of centimeters. Differential GPS uses data from the three satellites and an antenna, which is stationary. This stationary antenna corrects the data received by the receiver from the satellites. For indoor applications, standard GPS receivers cannot be used since they require a direct view of the sky and have very low signal strength. In this case assisted GPS can be used. An assisted GPS consists of a wireless handset with a partial GPS receiver. It makes use of an MSC (mobile switching center) and base stations to obtain data from the satellites. The main advantage of A-GPS is it can receive and demodulate signals that are orders of magnitude weaker than those required by standard GPS receivers. [5]

A number of devices are being used for tracking the user's viewing orientation. One of the devices currently used is an inertial/magnetic tracker attached to the headband above the AR glasses. A transmitter and receiver used in combination detect the orientation of the user's head. However such devices have a significant delay. Other tracking devices employ a laser, ultrasonic, optical or an RF link to track the user's orientation. Magnetic trackers have the disadvantage that any other metals in the vicinity may cause errors. Laser or optical tracking devices do not have this drawback.

2.3.3 Issues with Augmented Reality

AR systems require extensive calibration to produce accurate registration. Measurements may include camera parameters, field of view, sensor offsets, object locations, distortions and so forth [3]. System delays, which are the main source of registration errors, are caused by many different delays during the processing. These delays include the delay

due to the time taken for the image to reach the host, the delay in processing the image, and the delay while the graphics hardware renders the image. In addition, the delay caused when the image changes between frames also causes the user to see an outdated image. Predicting motion is one of the ways that have been proposed to reduce latency.

2.3.4 Applications

Boeing's IAR wire bundle assembly project was the first project to make use of a see-through optical display and wearable computers. This project was aimed at eliminating the need of guidance boards for wiring cables during airplane manufacturing. The military has been devising uses for augmented reality for decades.

Augmented reality in military is used to provide troops vital information about their surroundings, and highlight movement of troops to give soldiers the ability to move to where the enemy cannot see them. In addition the terrain can be mapped in advance, so that the soldiers can be warned beforehand about hidden mines, etc.

Another important application is in the medical field. Procedures such as biopsies and laproscopic surgeries are examples where AR could be used as an ideal solution. The surgeon would wear the HMD to see where the biopsy needle has to be guided.

Other uses could be for instant information for tourists and students. Industrial applications for AR prove to be useful to technicians. Markers, which are virtual components, are introduced into the real world, which helps the technicians to repair or construct industrial appliances etc. With the help of AR, on-site maintenance crews send system parameters to remote experts and provide mobile system with diagnostics information and guidance [7].

2.4 Future Trends

2.4.1 Head-Mounted Displays

Human beings perceive the world around them when light is focused onto the retina at the back of the eyeball. The retina converts light into signals that enter into brain via the optic nerve [8]. In the virtual reality world head-mounted displays (HMD) may be used to immerse the user into the virtual world. These use up a lot of power. If images can be

directly painted onto the retina, this would be the most power efficient way to display the virtual world scene. In addition, this low power device would be the best suited for a mobile system.

Such a system has been implemented, using tiny semiconductor lasers or special LEDs, one for each of the 3 primary colors red, green, and blue and project their light onto the retina. If slightly different images are projected onto each eye, a 3D scene can be rendered. This device uses a scanned-beam, hence is safe to use, since it does not allow the laser beam to focus on one point on the retina for a long time. In conventional displays jagged edges may be observed, since the pixels are fixed on the screen hardware. Scanned-beam avoids this problem, since there are no hard pixels; the continuously scanning beam creates a much smoother image. MEMS (micro electromechanical systems) are the most recent developments, which allow displays, which are higher in performance, lower in cost and smaller.

2.4.2 Haptic Devices

Haptics broadly refers to touch interactions. Haptic rendering allows the user to feel virtual objects in a simulated environment. Haptic rendering transmits information by exerting forces on the human hand through a haptic device. The most significant feature of haptic rendering is that, unlike visual systems where the data flow in only one direction, in haptic rendering the data flow in two directions to and from the user. An avatar is the virtual representation of the haptic interface through which the user physically interacts with the virtual environment.

There are three kinds of algorithms involved in a haptic rendering system. Collision detection algorithms which detect collisions between the avatar and any object in the virtual environment; force response algorithms which return the ideal interaction force between the two; and the control algorithms which return a force to the user. The force and torque vectors computed by the force response algorithms are fed to the control algorithms. The algorithms' return values are the actual force and torque vectors that will be sent to the haptic device. Haptically enabled products let designers sculpt digital clay figures, museumgoers can feel previously inaccessible artifacts and doctors can train for

simple procedures without endangering patients. The field of haptics is still in a state of infancy much remains to be explored in this field. [9]

Chapter 3
System Design

3.1 System Requirements

Image rendering is the most important element of any virtual reality system. Two factors are vital to attain a high performance rendering. These two factors include the frame rate and the memory associated with the graphics card. A computer with a high frame rate is ideal for the system since it provides a high degree of interactivity. The CPU performs intensive 3D calculations when 3D lifelike images are to be rendered on the screen. The graphics controller processes the textures and bitmap files and stores the pixels into memory. More graphics card memory hence enables a larger amount of space for storage of textures, which may be associated with 3D graphics. In addition, the high graphics card memory requirement is also a must for higher screen resolutions.

3.2 Head-Mounted Display

The user's perception of the environment is a key component of a virtual reality system. A head-mounted display provides the user a view of the virtual world, and isolates the user from the real world. This gives the user a sense of being immersed in the virtual world. The three main categories of visual displays include stationary displays; head based displays, and hand-based displays. Human binocular vision enables one to measure depth using eye convergence and stereoscopic vision [2].

Eye convergence is the measure of the angle between the eyes and the optical axis when fixating upon some point [2].

Stereopsis and *stereoscopic vision* are two terms used frequently in relation with 3D vision. Stereopsis describes the process of obtaining two distinct views of an object when viewed with two eyes, and stereoscopic vision describes the three-dimensional sensation of depth associated when seeing with two eyes [2].

3.2.1 Specifications of Head-Mounted Displays

The resolution of the I-glasses head-mounted display used in this thesis is 1.44 million pixels per display (800 x 600 RGB color sequential system). The input may be VGA (640 x 480) at 60-100Hz. or SVGA (800 x 600) at 60-100Hz. The refresh rate of the HMD is double the input refresh rate up to 60 Hz. Above 60 Hz the display refresh rate equals the input refresh rate. The field of view is 26.5 degrees diagonal. Immersion in the virtual environment can be improved by increasing the field of view. But, a compromise has to be made between the field of view and the weight and size of the optics of the head-mounted display. In addition, the larger the field of view, the lower is the acuity or sharpness of the images displayed. Contrast specification of the HMD is 25:1. The contrast is the difference between the brightest and the darkest pixels and is expressed as a ratio. The eye relief value of the HMD is 25mm. Eye relief is the distance between the eye and the first lens the eye sees. [10]

3.2.2 I-Glasses Operation

The computer monitor should be set to a resolution of 800 x 600. In addition it should be set for a minimum of 16-bit colors. The refresh rate of the monitor should be set to 60 Hz for the 2D mode of operation. The monitor needs to be disconnected in order to connect the I-glasses cable for a single VGA port computer. In case of PCs with two VGA ports, the second port can be connected to the I-glasses, enabling the user to see both the monitor image and the image in the head-mounted display.

The I-glasses SVGA operates in two modes 2D mode and 3D mode. The On-Screen display menu can be used to select one of the two modes. 2D mode requires a refresh rate of 60Hz. 3D mode has two sub-modes. 3D mode should be operated at a minimum refresh rate of 85Hz. 100Hz refresh rate or even higher is however preferable for this mode of operation. nVIDIA stereo drivers support this mode of operation only for the I-glasses (refresh rate of greater than 75Hz not supported for the Dell monitor).

3.2.3 Modes of Operation

Mode 3D1 or 3D2 follows DDC line protocol. DDC or Dynamic Depth Cueing is a process developed by the company Dynamic Digital Depth. Using algorithms, which tease apart each 2D image frame, objects are tracked, and isolated. This process uses clues to determine separate objects, using techniques including image occlusion, object outlines, and determines each object's position within the original 3D space. nVIDIA stereo drivers support DDC line protocol. For this project the graphics card used is a GeForce2 GTS from nVIDIA. The nVIDIA 56.64 stereo driver is used to support 3D graphics mode.

The nVIDIA stereo driver outputs a 3D signal, the I-glasses automatically detects this signal and stays locked on the signal. In the presence of this signal the headset functions correctly in either 3D1 or 3D2 modes.

Full resolution frame sequential inputs are required by both 3D1 and 3D2 modes. In frame sequential mode the operation of the display is such that the image is sent to the left eye first and then the right eye in alternating frames. This mode is used in the 3D1 mode of operation. In the 3D2 mode of operation the image is first sent to the right eye then to the left eye. Hence a high refresh rate of 120Hz is required, so that each eye receives the image at 60Hz. This is needed so that the human eye does not see image flicker on the screen. The graphics card in combination with the driver software outputs a DDC line protocol signal. This signal detected, has the highest priority and the headset works with the 3D signal received.

The head-mounted display works with an input voltage of 7-18V DC and can be used with a battery pack. For a mobile operation system, this feature is desirable. However due to the LCD requirements, a large amount of power is required. Recent developments in display systems, allow a low-power display operation, for mobile virtual reality systems. The typical connections of I-glasses SVGA to a PC are as shown in Figure 3.1.

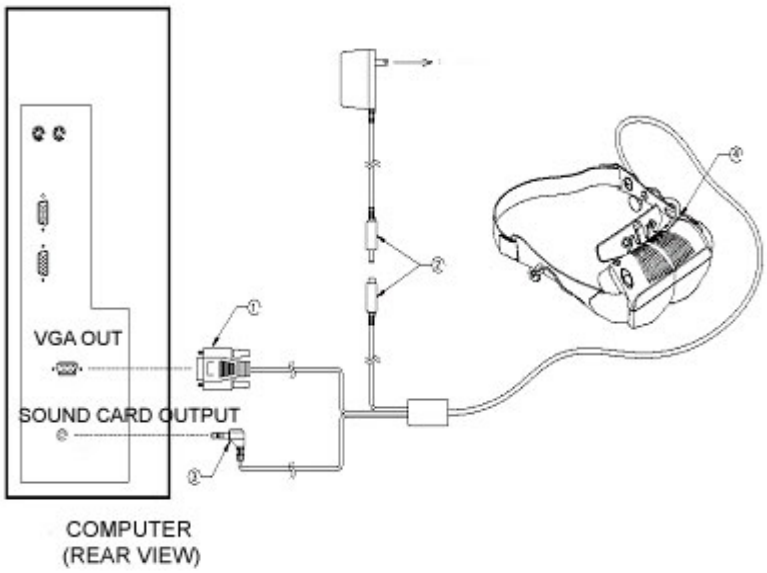

Figure 3.1. Typical PC connections of I-glasses SVGA. [10]

3.3 Wireless Mouse

An Iogear Phaser-wireless RF mouse is used to navigate around in the virtual environment. The left mouse button can be used to zoom into and zoom out of a scene. The right mouse button can be used for panning. A wireless mouse is chosen to help in creating a mobile wireless system and to reduce the number of wires.

A base unit is connected to the CPU of the computer through the USB port. The mouse sends signals to the base unit through an RF link. The mouse can be operated without having to point at the base and can work from even 50 feet away from the computer. The signal the mouse sends to the base has a frequency of 27.045 MHz. Multiple units can operate in the same room and no interference is caused since each unit has its own ID. The function diagram of the wireless mouse is as shown in Figure 3.2.

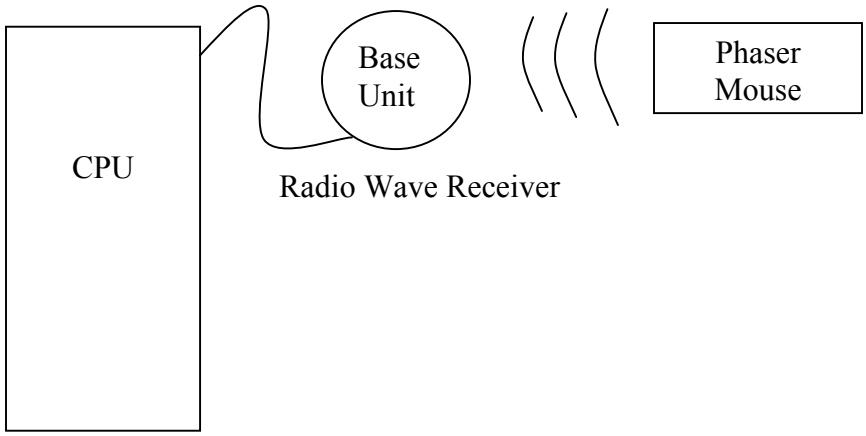

Figure 3.2. Function diagram - Wireless mouse.

3.4 Pinch Glove

The FakeSpace Pinch Glove system provides a low cost and easy method of recognizing natural gestures. The hand-gestures system interface allows the users of the virtual environment to use hand interaction to work within the virtual environment. The pinch glove is comprised of a base unit and a pair of gloves. Fingertip pads are provided to sense the contact between the thumb and any of the four fingers [12]. The base unit provides power for the system. It translates the fingertip contact and communicates the same with the host computer through the RS232 output [12]. The Figure 3.3 describes the front panel of the pinch glove base unit. The pinch glove system is shown in Figure 3.4.

3.4.1 Software Interface

The pinch glove system communicates with a host computer over an RS232 serial link [12]. There are two kinds of data that are sent to the host computer. These are the contact information and the status information. When contact is made between the fingertips, data are sent to the host computer. The data packets are constructed as follows:

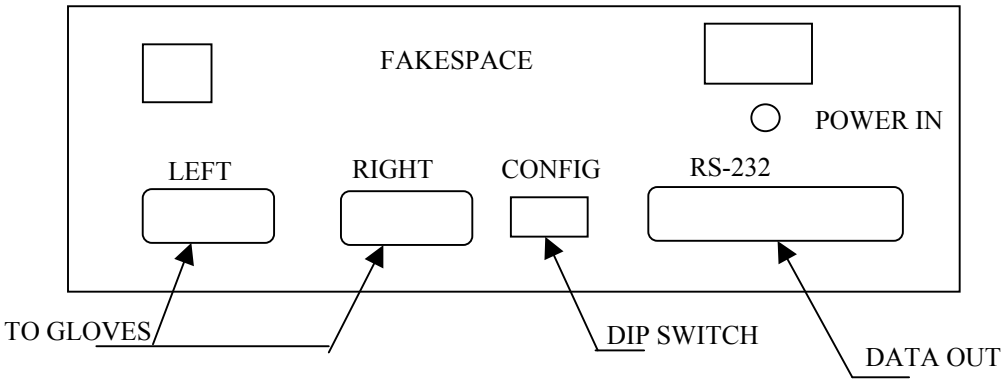

Figure 3.3. Base unit diagram.[12]

Figure 3.4. Base unit and pinch gloves.

STARTBYTE [VARIABLE NUMBER OF DATA BYTES] ENDBYTE [12]

The *STARTBYTE* depends on the type of record sent. The value of the *STARTBYTE* and the type of record being sent is as shown in Table 3.1. There is only one type of *ENDBYTE*, which is 0x8f, which denotes end of message. *DATA* bytes are always 7 bits. The upper bit of control bytes is a 1 and on the data bytes it is 0. Packets with touch data are sent following contact between two fingers.

3.4.2 PC connection

The pinch glove system is connected to a PC through an RS232 cable. The pinch system uses only pins 2,3 and 7 of the RS232 connector. The DIP-switches on the base unit are to be set depending on the baud rate for communication to the host. There are 8 different positions for the DIP-switches. Hence, 8 different baud rates can be selected using different positions of the DIP-switches. There are 8 DIP-switches on the Pinch System.

The switch SW1 is used for a long-touch threshold if ON. This switch is set to ON so that only contacts longer than 120 ticks are considered to be a touch where 1 tick is 833us. Switches 2,3,4 and 5 are set to "off" since they are reserved. Switches 6,7 and 8 are used to select the baud rate. Baud rates can be selected between 1200 and 153600 depending upon the position of these switches. For this thesis a baud rate of 9600 is used. The positions of switches 6, 7 and 8 are OFF, OFF and ON (0,0,1) respectively. [12]

Table 3.1. *STARTBYTE* and record type

STARTBYTE (in HEX)	***Record Type***
0x80	Hand data only
0x81	Hand data and time stamp data
0x82	Text or configuration info

3.4.3 Applications

In an immersive virtual reality environment, the user is wearing the head-mounted display. The head-mounted display will show where the user is in the virtual world. The user cannot observe the keyboard wearing the head-mounted display, since the user cannot see the physical world when immersed in the virtual world. Keyboard entry is necessary for various tasks, for example changing the viewpoint of the scene, or to check the frame rate of the application. In order to use the keyboard, the head-mounted display must be removed. In addition, when the system is wearable, the keyboard cannot be used since the keyboard requires a flat surface to operate. An alternative to the keyboard hence would be to use the pinch glove. Contact between any two fingers emulates the press of a key on the keyboard. Different combinations of fingers in contact can be used to call the different functions of the keyboard. [13]

3.5 Tracker

A tracker is used to track head and hand movements in a virtual reality system. The different tracking methods currently employed are electromagnetic, mechanical, optical, videometric, ultrasonic, inertial and neural. If a tracker is attached to the HMD then with the movement of the head, the viewpoint from which the scene is rendered is changed and it appears as if the user is looking around the virtual environment. A hand tracker can be also used to detect hand movements and zoom in or zoom out depending on the position of the hand. For this thesis the Ascension Flock of Birds tracker is used which is shown in Figure 3.5. This includes a stationary transmitter radiating electromagnetic signals that are intercepted by the sensors. The position and orientation of a sensor is calculated and sent back to the host.

3.5.1 Hardware Details and Configuration

The Flock of Birds (FOB) consists of a transmitter, which may be a standard or extended range transmitter. The flock may be configured to track up to 30 receivers (birds). The position and the orientation of the birds can be measured, and hence it is a six degree-of-freedom device. The FOB determines position and orientation by transmitting a pulsed

Figure 3.5. Flock of Birds tracker.

magnetic field that is simultaneously measured by all sensors in the flock [14]. Using the magnetic field characteristics, each sensor independently computes its position and orientation and makes this information available to the host computer. The extended range transmitter and controller (ERC & ERT) are used if operating ranges of more than 8 feet are required.

The FOB is configured as shown in Figure 3.6. The master bird is connected to the host computer through the RS232C interface. The remaining two birds are connected through the Fast Bird Bus (FBB) in a daisy-chained fashion. The master bird is connected to the transmitter. Each receiver is connected to the bird. Each bird has its own power supply. The master bird controls and coordinates the operation of the slave birds.

An advantage of this configuration is that it requires less of the host hardware but the disadvantage is that it limits the number of measurements that per second that the host can read from each bird. There are two modes of operation for the Flock of Birds standalone mode and FOB mode. In standalone mode there is one transmitter and one sensor and has the bird address set to zero via the DIP-switches. In FOB mode more than one bird is connected. The configuration that is used is the FOB mode since it connects three birds through the FBB.

3.5.1.1 Addressing Modes

The flock of birds can be configured in three different addressing configurations: Normal Addressing Mode, Expanded Addressing Mode, and Super-Expanded Addressing Mode. Normal Addressing mode is used when up to 14 bird units are present in the flock.

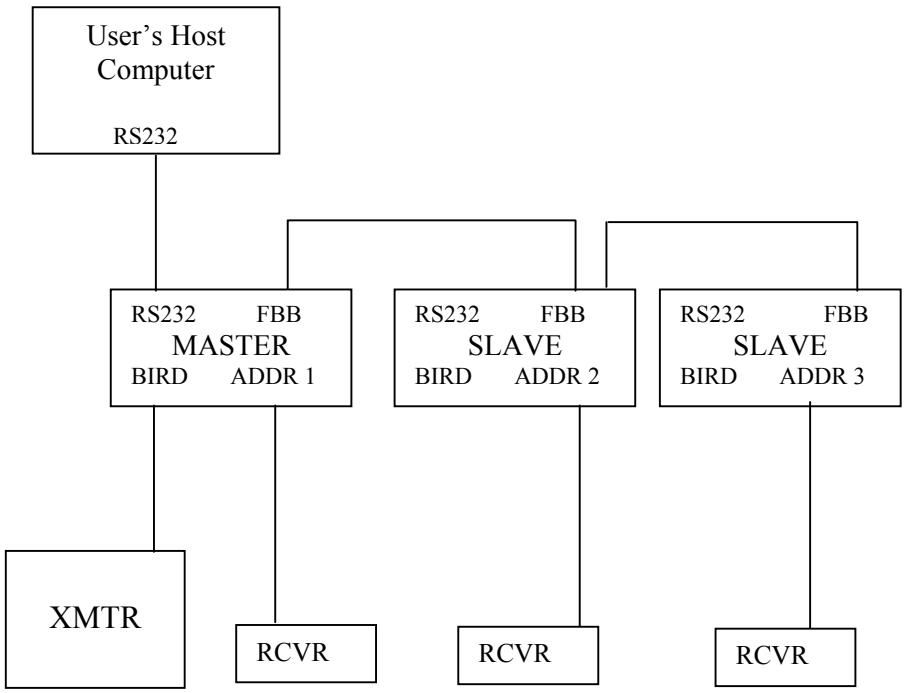

Figure 3.6. FOB configuration with single RS232 interface to host computer. [14]

Expanded addressing mode is used when more than 14 but fewer than 30 birds are present in the flock. Super-Expanded mode is used when more than 30 birds are in the flock. Since only three birds are used the normal addressing mode is used. Depending on the addressing mode used, the DIP-switches are set appropriately.

3.5.1.2 DIP-switch Configuration

Using the DIP-switch located on the back panel of the bird the baud rate, addressing mode and address of the bird are set. DIP-switch 8 is used to switch between normal operation of the flock and test mode. DIP-switches 4 to 7 are used to set the address of the bird in normal addressing mode. Bird 1 has an address of 0001, bird 2 has an address of 0010 and bird 3 has an address of 0011. There can be no zero address.

DIP-switches 1 to 3 are used to select the baud rate at which the flock operates in normal addressing mode. The baud rates from 2400 to 115,200 can be selected using the DIP-switches. The baud rate of 9600 has been selected for purposes of this thesis. The settings are as shown in Table 3.2 where 0 indicates OFF and 1 indicates ON. The Figure 3.7 shows the back panel of the bird and Figure 3.8 shows the front panel.

Table 3.2. DIP-switch configurations.

DIP-switch	Settings	Function
8	0	Normal Operation
4,5,6,7	Bird 1 – 0001 Bird 2 – 0010 Bird 3 – 0011	Bird Address
1,2,3	011	Baud Rate 9600

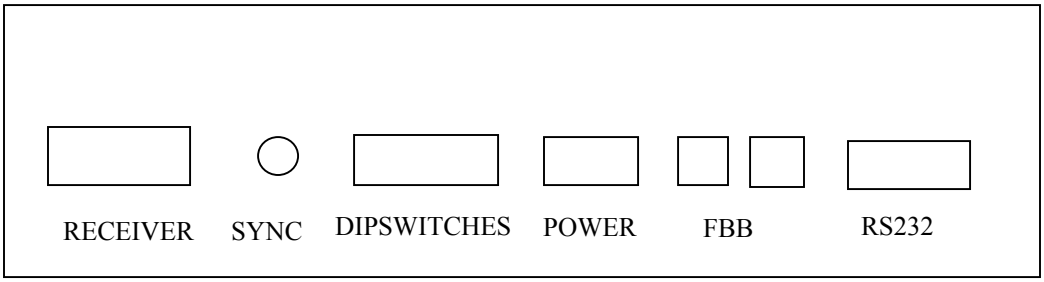

Figure 3.7. Flock of Birds back panel.

Figure 3.8. Flock of Birds front panel.

3.5.1.3 Cables and Connections

For the configuration as shown in Figure 3.6 FBB cables are used to interconnect the birds in a daisy-chained fashion. These are plugged into 8-pin modular connectors in the back panel of the bird. The sensor cable is connected to the 15-pin connector again on the rear panel of the bird. The transmitter cable is attached to the 9-pin connector on the front panel of the bird. Each of the birds requires a power supply, which is connected to the rear panel of each bird.

The master bird communicates with the host computer through the RS232C interface. This cable has to be prepared as per specifications as shown in Table 3.3. These signals are high when the bird is not in reset condition. Changing the front panel switch from FLY to STANDBY resets the bird [14].

3.5.2 Tracker Features and Parameters

When the FLY/STDBY switch on the bird is changed from STDBY mode to FLY mode the bird is reset and all power-up functions are performed. For the normal addressing mode the LED on the front panel blinks quickly five times and then stays off if the mode

Table 3.3. RS232C signal descriptions.

Pin	RS232 Signal	Direction
1	Carrier Detect	Bird to host
2	Receive Detect	Bird to host
3	Transmit Data	Host to Bird
4	Data Terminal Ready	Host to Bird
5	Signal Round	Bird to host
6	Data Set Ready	Bird to host
7	Request to Send	Host to Bird
8	Clear to Send	Bird to host
9	Ring Indicator	No Connect

of operation is FOB and if the bird is functioning normally.

There are five types of data that the bird can return to the host depending on the data type the host requests. These seven types are *ANGLES, MATRIX, POSITION, QUATERNION, POSITION / ANGLES, POSITION / MATRIX,* or *POSITION / QUATERNION*. The *ANGLES* data contains the orientation angles of the sensor with respect to the transmitter. The orientation angles are defined as the rotations about the Z, Y and X axes of the sensor [13]. The values of Z are in the range of +/-180 degrees that of Y are in the range +/-90 degrees and that of X are in the range of +/-180 degrees. The *MATRIX* mode outputs the 9 elements of the rotation matrix that define the orientation of the sensor's X, Y and Z axes with respect to the transmitter's X, Y and Z [14]. *POSITION* mode outputs the X, Y and Z positional coordinates of the sensor with respect to the transmitter. *QUATERNION* mode outputs four quaternion parameters that describe the orientation of the sensor with respect to the transmitter. These quaternions are extracted from the rotation matrix. *POSITION / ANGLES* mode returns both the position and angles output in a single record. *POSITION / MATRIX* mode returns position and matrix in a single record. *POSITION / QUATERNION* mode returns both position and

quaternion in the same record.

The shape of the magnetic field transmitted by the bird is symmetrical about each of the axes of the transmitter. This leads to an ambiguity in the values of X, Y and Z position. Hence, the volume around the transmitter is divided into 6 hemispheres. These are forward, aft (rear), upper, lower, left and right. Any of the hemispheres can be selected depending on the application requirements. The X value is always positive when a hemisphere of operation is selected. The X, Y and Z directions are shown in Figure 3.9.

The measurement rate of the bird can be varied from 20 to 144 Hz. The measurement rate default is set to 103 Hz. In the case of a CPU time overflow error, which occurs when the host sends multiple commands in a measurement cycle, it is useful to decrease the measurement rate of the bird. However decreasing the measurement rate has a disadvantage that it may cause errors due to time lag. In addition the noise may increase or reduce depending on the frequency chosen. If a measurement rate equal to the power line frequency is selected the noise increases.

There are two modes in which the data can be retrieved from the bird. These modes are *point* mode and *stream* mode. In *point* mode the bird returns one data record

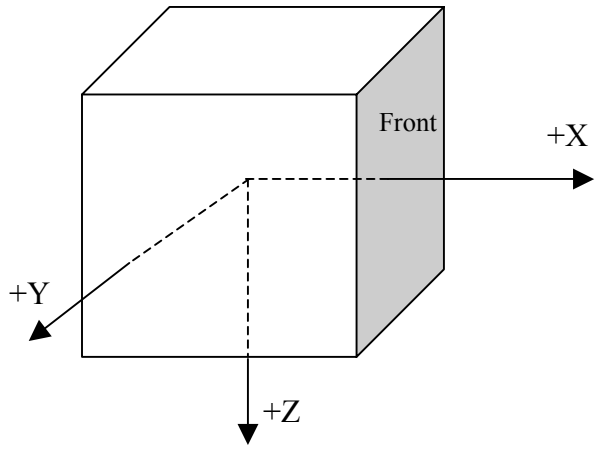

Figure 3.9. Transmitter reference frame.

when it receives a command from the host. In *stream* mode the bird sends data records continuously to the host when it receives the command. It stops sending data only when it receives the point command or any other command. For a group of birds to send data in this mode the group mode must be enabled before the data can be retrieved. If the host computer cannot keep up with the data rate streaming in the report rate should be changed so that the rate at which the data is sent in is reduced.

The bird keeps track of errors, which occur during operation. These errors are indicated by the LED on the front panel of the master bird. When an error occurs the LED blinks 10 short blinks followed by N number of long blinks. The N long blinks indicate the error code. The error code ranges from 1 to 35 each having different error descriptions. Cycling the power switch or reading the system status or issuing an auto-configuration command can reset the error code.

3.6 Mobile System

All the components described in the Sections 3.1 through 3.5 can be integrated together to build a mobile wireless system. The computer used may be a lightweight notebook computer with a high performance graphics card and a wireless card, which connects to the host through a wireless LAN that uses IEEE 802.11 protocol and transmits and receives data as required through this link.

3.6.1 IEEE 802.11 Standard

IEEE 802.11 is a standard developed for wireless LANs that cover an office building or a group of adjacent buildings. The initial standard provided for data rates of 1 and 2 Mbps. This was too slow and proved to be a disadvantage for general requirements. Hence, the IEEE approved the 802.11b standard, which supports higher data transmission rates of up to 11 Mbps. Wireless LANs are able to achieve high performance with 802.11b, which is also known as WiFi.

Figure 3.10 demonstrates the 802.11 standard and the ISO model. 802.11 focuses on the lower two layers of the OSI reference model, the physical and the data link layers. The standard covers three physical layer implementations: direct-sequence (DS) spread spectrum, frequency hopping (FH) spread spectrum, and infrared (IR) [15]. A single MAC layer supports all three physical layer implementations [15]. Two modes are defined in the 802.11 standard. They are *infrastructure* mode and *ad hoc* mode. In infrastructure mode the wireless network consists of at least one access point connected to the wired network infrastructure and a set of wireless end stations [15]. In *ad hoc* mode the 802.11 stations simply communicate directly with one another without using an access point or a connection to the wired network.

In the physical layer the DS spread spectrum has a more robust modulation and covers a larger area than FH spread spectrum modulation. Both FH and DS have their own respective advantages and disadvantages. The data link layer consists of two sub-layers. They are logical link control (LLC) and the media access control (MAC). 802.11 use the same LLC and 48-bit addressing as 802.x standards making it easy to bridge between wired and wireless networks. The MAC layer however is unique to 802.11 standard. The MAC layer supports two fundamentally different schemes, the *distributed coordination function* (DCF) and the *point coordination function* (PCF). In DCF all users have an equal chance of transmitting data, whereas the access point controls transmission based on polling in PCF. DCF uses CSMA/CA or Carrier Sense Multiple Access with Collision Avoidance protocol. In this protocol, packet acknowledgement to confirm the arrival of the packet is used, to avoid loss of data due to collisions. The 802.11 standard uses the 2.4 GHz band. Even though 802.11 supports three different modes of transmission in the physical layer, versions of the standard focus on a single method of transmission. For example 802.11b only allows DSSS whereas 802.11a uses OFDM.

A laptop equipped with a wireless network interface card which uses 802.11b standard may be used for the mobile wireless system described. 802.11b is used since it provides a larger range than 802.11a or 802.11g standard.

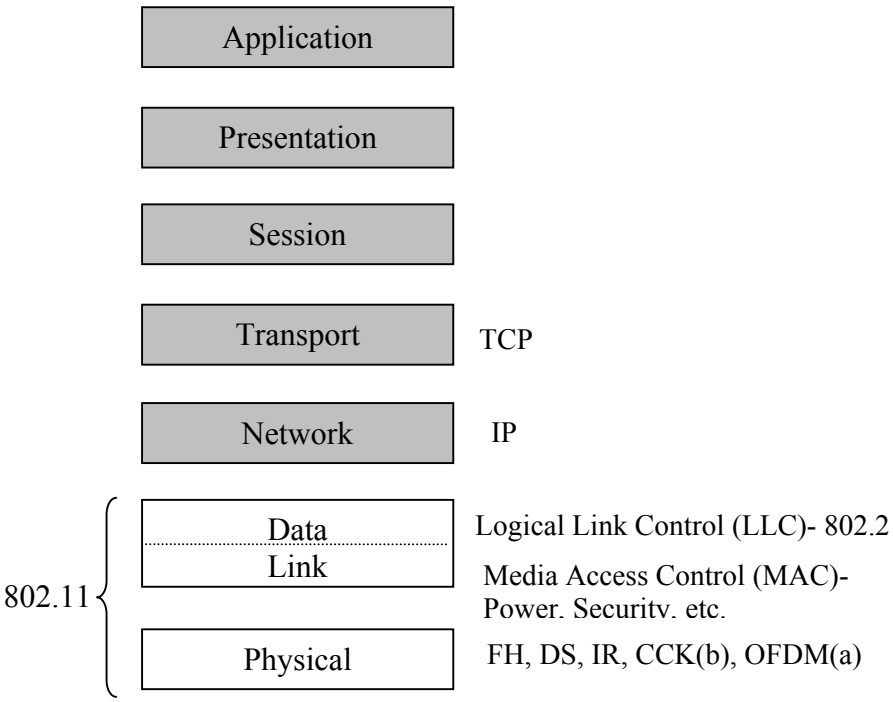

Figure 3.10. The 802.11 standard and the ISO model. [15]

3.6.2 System Requirements

The head-mounted display is connected to the SVGA port of the PC. The HMD can be powered with a battery pack. The pinch glove and the tracker are connected to the two serial ports available and can be used to walk through the virtual environment. However, laptop computers or notebook computers ordinarily are equipped with a single serial port. In addition the pinch glove and the tracker both are wired and require large amount of power. A wireless data glove can be used in place of the pinch glove.

The 5DT Data Glove can be used. A data glove measures and detects the finger flexure. 5DT data glove is wireless and uses fiber optic sensors to detect the finger configuration. The glove has five sensors one for each finger. The glove is connected to the transmitter. The transmitter communicates with the receiver through a radio frequency link. The receiver is connected to the RS232 serial port. The power required by the data glove is a maximum of 150mA at 9V DC and can be supplied by a battery pack. The sensor resolution is 8-bit for 5DT Data glove. The 5DT data glove is shown in Figure

3.11. The Data Glove is light and compact and suitable for a mobile operation. [16]

The disadvantage of the tracker explained in Section 3.5 is that it uses a large amount of power and cannot be powered using batteries hence is unsuitable for a mobile system. In addition, it uses the serial port of the PC, which may be unavailable since the glove uses the serial port. Also, this tracker is heavy and not easily portable. An alternative to this tracker would be InterTrax tracker developed by InterSense. InterTrax connects to the USB port of the PC. The tracker is powered through the USB (5 Volts). This tracker is lightweight and since it takes power from the USB port no additional battery pack is required.

The wireless mouse connected to the PS/2 port can be used without a mouse pad and without having to point at the screen to zoom in or out of the scene. Figure 3.12 illustrates such a system.

This system can be further developed into an augmented reality system where the user sees the real world in real time instead of the computer generated view of the

Figure 3.11. 5DT Wireless Data Glove.

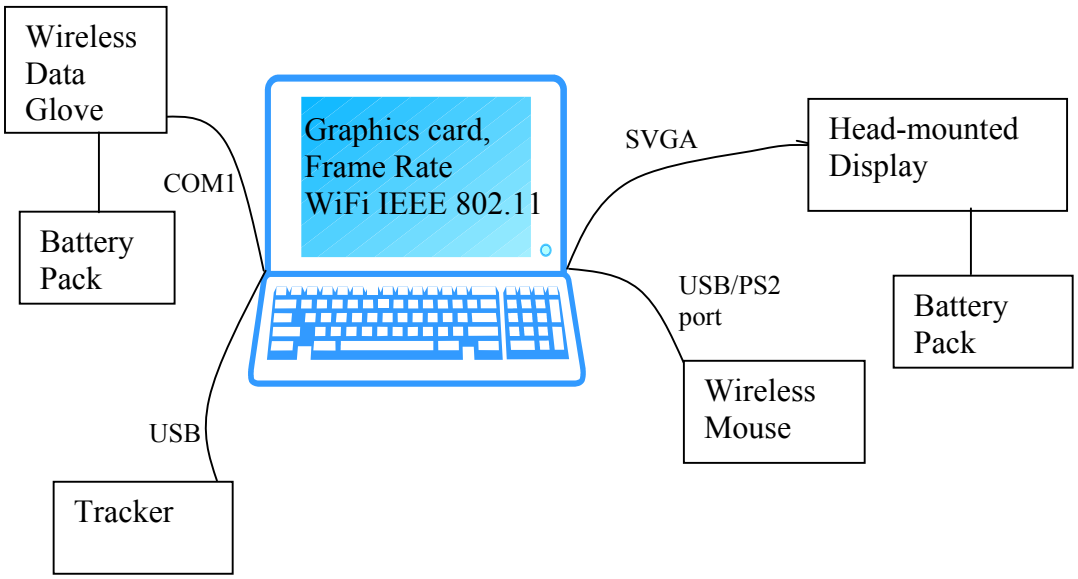

Figure 3.12. Mobile VR system.

surrounding environment and the data about the scene is displayed in real time along with the scene.

Chapter 4
Software Module

4.1 Introduction

VR environments require user-controlled navigation in virtual space and also other functions provided by the hardware devices explained in Chapter 3. Software is hence required to provide these functions and enable the hardware to communicate effectively with the computer. There are two main components to the software module the 3D models, which create a realistic environment and the software code, which enables the user to navigate around in the virtual environment and perform different functions as per the requirement. Both of these components used in this thesis are explained in this chapter.

4.2 3D Modeling

In the real world, building any object requires detailed planning. A lot of rules need to be followed to build the object. In addition, supporting material may be needed to hold the various parts of the real world object together such as nuts, bolts, hinges, glue, screws, etc. In a virtual world, there are no rigid rules that need to be followed while putting together the object. Nevertheless, to create a virtual object the object's dimensions still need to be known [2]. From the basic geometry the individual elements can be described separately, and assembled to form the required object [2]. 3D models are the primary requirement of visualization or simulation. The user can get a better picture of the scenario and conditions when life-like models are used. 3D Studio MAX version 4.2 is used for creating models for the purpose of this thesis.

4.2.1 Basics

Actual dimensions of the model to be constructed are needed for initial size of the blocks that are to be used for constructing it. A number of factors associated with the rendering speed are also taken into consideration when 3D models are built. Polygon count is defined as the number of faces in the scene. The more the number of faces in the scene

the more is the time required for rendering the scene. Polygon count of the model should be maintained low so that the rendering speed is not reduced. In order to keep the polygon count low, box modeling should be used.

The standard primitives such as box, sphere or cylinder can be used to build complex models. Where curves are required to the model, the blocks can be broken into segments. Each standard primitive may be composed of polygons, edges, vertices or faces. Geometry segmentation allows it to be modified on any of these levels.

4.2.2 Material

Materials create greater realism in a scene [17]. The human visual system has the ability to process natural textures, and man-made textures. Hence, through textures much can be communicated to the user. A material describes how an object reflects or transmits light in a scene [17]. Figure 4.1 shows an example of a scene using a brick material. Material properties works hand-in-hand with light properties; shading or rendering combine the

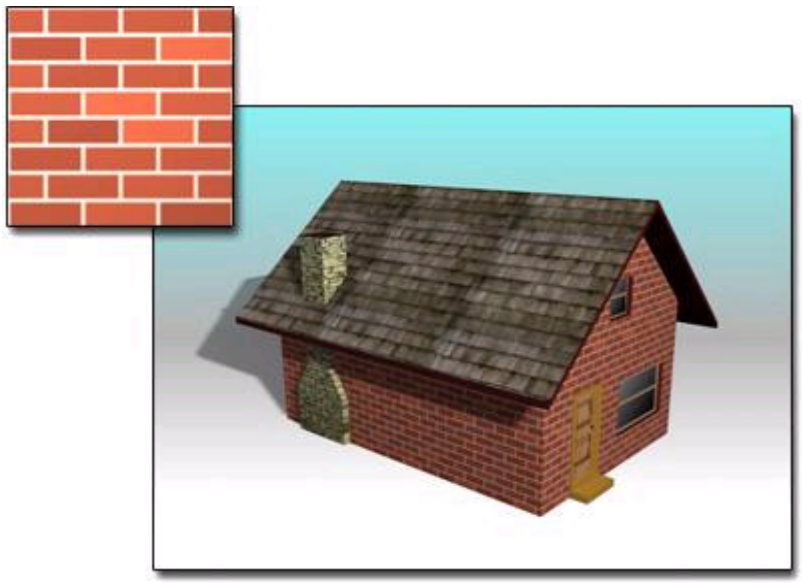

Figure 4.1. Example of usage of material.

two, simulating how the object would look in a real world setting [17]. Each object in the scene has to be assigned a material. Assigning appropriate material to the object is very important in visualization. There are different types of materials, which can be used for different purposes. The basic types are standard, raytrace and matte or shadow. The raytrace material has the ability to create a photo realistic effect but is time-consuming and requires large amounts of memory. This results in a low rendering speed. This is due to the fact that with raytrace, the reflections and refractions of the object need to be calculated. Figure 4.2 shows an example of a scene using raytrace materials.

Materials have different parameters, which can be changed to give a more realistic effect. The diffuse parameter is the color of the object in good or direct lighting. Ambient is the color of the object in shadow [17]. Mostly the diffuse and ambient parameters are locked so that the object appears the same in any light condition. Specular refers to the color of a shiny highlight. Specular highlights appear where the viewing angle is equal to the angle of incidence [17]. The specular level can be changed which changes the intensity of the specular highlight. Glossiness affects the size of the highlight, as the glossiness increases the size of the specular highlight gets smaller. The

Figure 4.2. Example of raytrace materials used in a scene.

effect of the specular parameter can be softened by using the soften parameter which blurs the outline of a specular highlight. The opacity of the object can be reduced with the opacity parameter of the material. Rendering a transparent object can be more time-consuming than ordinary opaque objects, since the amount of light passing through the object and the amount of light reflected or refracted off the surface needs to be calculated by the rendering system. Turning off either reflection or refraction can improve the rendering speed to a large extent. This is because the rays start bouncing inside the object when both parameters are on, and this causes the calculations during rendering to increase greatly.

4.2.3 Lights

Lights are objects, which simulate the real lights. There are two methods by which lights can be added to the scene. The first method is adding lights in the modeling environment when the virtual world is created. The second method would be through the code. In 3D Studio MAX if no lights are added to the scene, default lighting is used. The default lighting for 3D Studio MAX consists of two lights: a *key* light positioned in front and to the left of the scene, and a *fill* light, positioned behind and to the right of the scene. Both these default lights behave as *omni* light. An *omni* light casts light in all directions.

There are other types of lights, which may also be used. They are target spot, free spot, target direct, free direct and omni. A target spot directs the light and the camera on the object, which is chosen as the target. The free spot is directed but there need not be any target object. The free spot can be rotated in any direction. The target direct radiates as parallel rays of light in a particular direction. Target direct has a target object. Free direct is similar to the target direct except that it does not have a target object. Free direct can be used to simulate the sun. Both the target direct and free direct have a beam in the shape of a cylinder unlike the target spot and free spot which have a beam in the shape of a cone. An omni light radiates light in all directions. Figure 4.3 shows the difference between a spot light and a direct light.

Adding lights to the scene increases the realism of the scene but again slows down rendering. Each light added to the scene increases the time required for rendering.

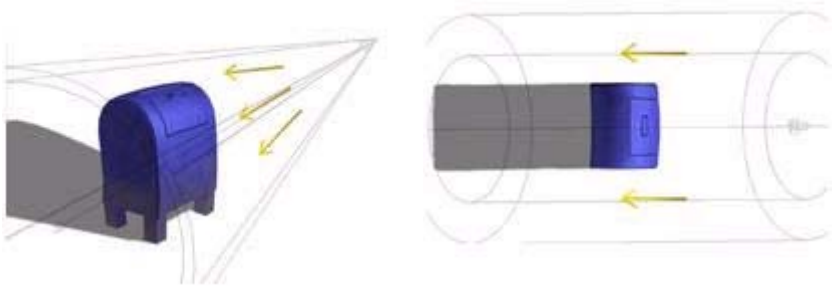

Figure 4.3. Differences between spot light and direct light.

4.3 Tyson Airport Model

For the purposes of this thesis a model of the McGhee Tyson Airport has been used. The original model was in the VRML **.wrl* format. This file was initially converted to the 3D Studio **.3ds* file format using NuGraf Rendering System software. This model was then imported into 3D Studio MAX for further modifications. Textures were added in 3D Studio MAX to create a more realistic virtual environment. Before the model is exported into **.3ds* format a number of factors are to be taken into consideration so that the model can be imported into the software used.

4.3.1 Single Mesh

While building a 3D model different geometries are used and assembled together. When imported into software each of the geometries used will be treated as a different node if they are not converted to a single mesh. In this thesis all the geometries have to be attached to form a single mesh, so that the entire scene is treated as one node. While attaching it is necessary that all geometries have a material assigned. This is required if either of the geometries, the one that is being attached or the one that the geometry is being attached to, does not have a material, it inherits the material of the other. Figure 4.4 shows the model before it is attached. The geometries used to build the model are listed on the right. Figure 4.5 shows the model as a single mesh.

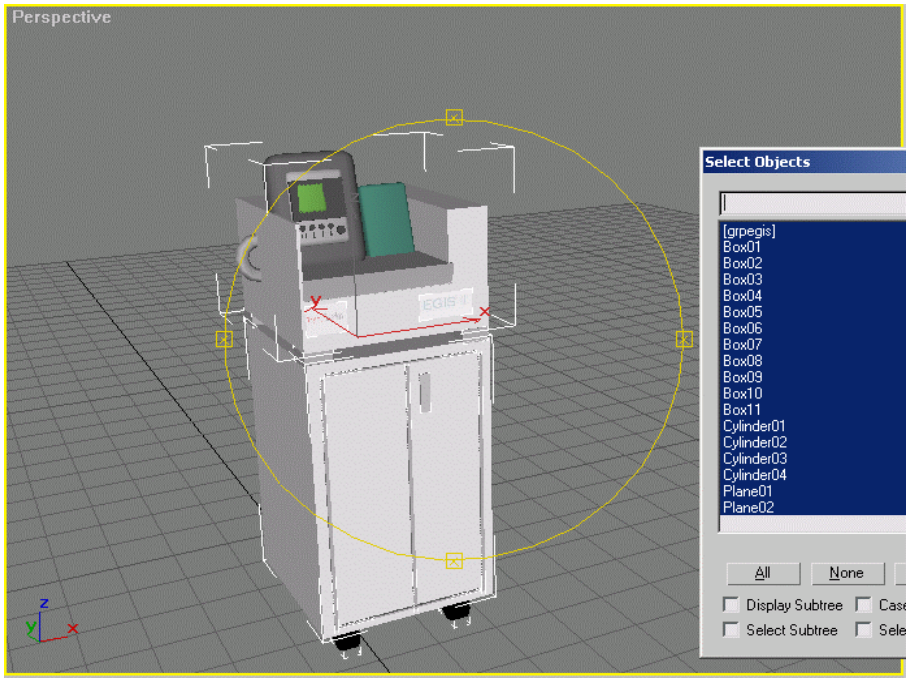

Figure 4.4. Model before attaching.

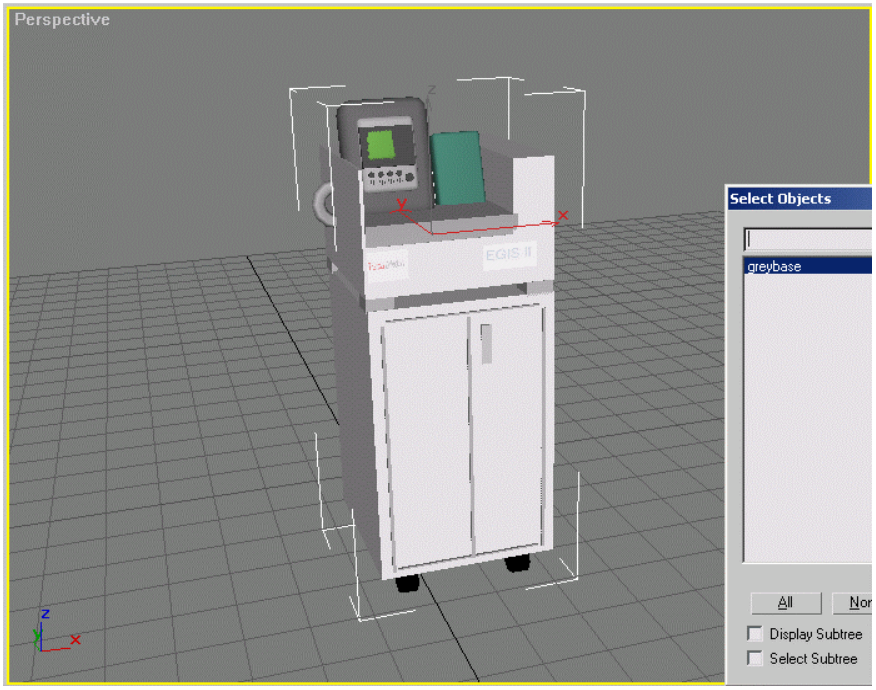

Figure 4.5. Model after attaching.

4.3.2 Midpoint of Model

In a visualization scene, it is very important to obtain the rendered position of any object in the scene. If the object needs to be moved around during a simulation it is important to note the midpoint and apply the net translation of the object to it, so that the object is placed at the right position. In order to get the rendered position, the midpoint of the geometry node is used. Hence the midpoint of the geometry should be at least approximately at the center of the 3D model. If the midpoint is not at the center of the object, the results of a visualization simulation may be erroneous.

Figure 4.6 shows a model with the midpoint outside the model. The coordinate *gizmo*, a tool that shows the X, Y and Z, coordinates in red, green and blue respectively is the midpoint of the object. This midpoint should be shifted so that it is well inside the model and represents the approximate midpoint of the geometry. The *Xform* modifier tool

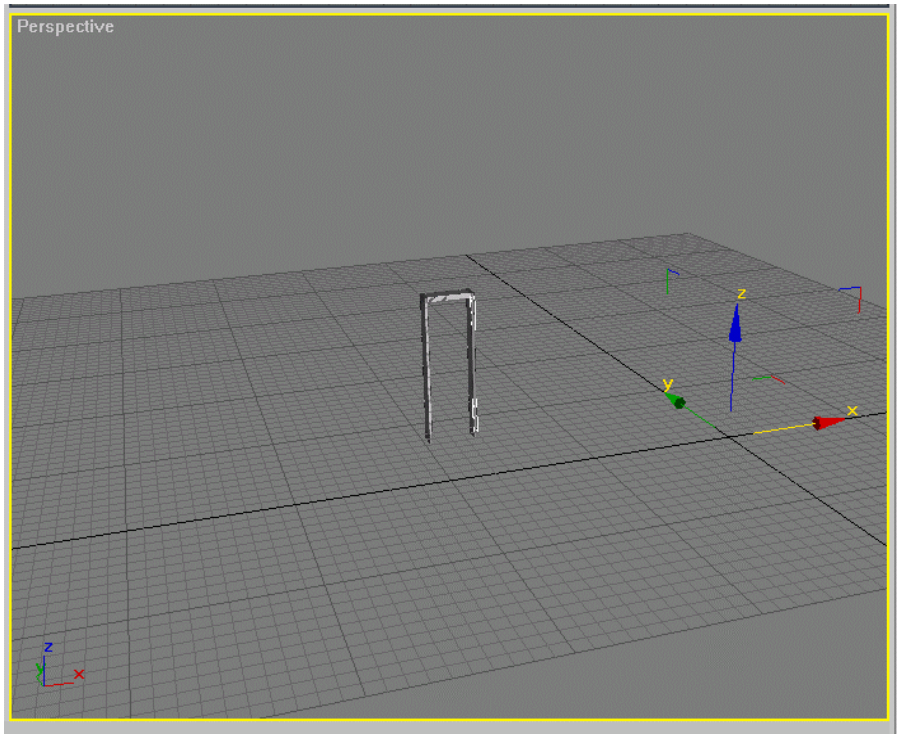

Figure 4.6. Gizmo shows the midpoint of the object.

in 3D Studio MAX can be used for this purpose, to move the midpoint of a model to the center.

The Figure 4.7 presents the model with the midpoint moved to the center. Before the midpoint is moved it must be determined that there are no lights in the scene other than the default lighting, which may be affecting the position of the midpoint.

4.4 WorldToolKit (WTK)

The software, which is used for creating the virtual reality environment in this thesis, is WorldToolKit from Sense8. WTK is a library of functions written in C. Virtual worlds can be built using WTK functions. These functions replace hundreds of lines of C code which call low-level OpenGL graphics functions. WTK manages the reading of input sensors, importing geometries and updating the scene. WTK has a simulation loop at its

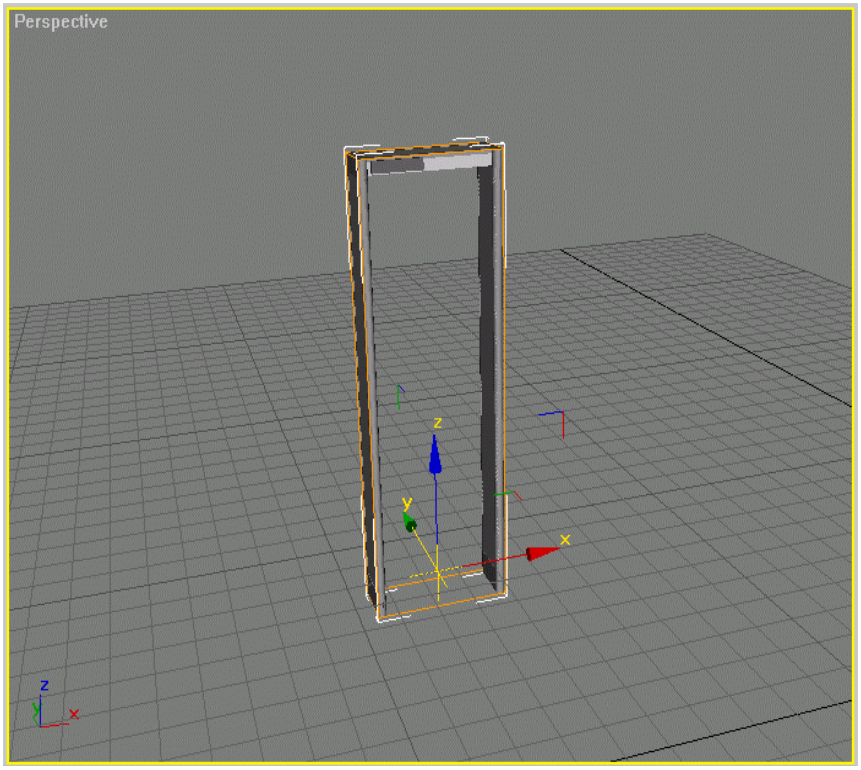

Figure 4.7. Modified midpoint of model.

core, which reads input sensors, updates objects and renders a new view of the scene onto the display. WTK uses a scene graph hierarchy to create virtual worlds. Nodes are assembled in a scene graph and these nodes decide the rendering of a scene. WTK also has an object-oriented structure. The functions used by WTK are grouped into classes such as *universe, windows, sensors, etc.* [18]

4.4.1 Universe

The universe is the container for all WTK objects, which include sensors, geometries, lights, viewpoints etc. In any WTK application the first call must be to create a new universe. This call initializes the graphics devices, configuring it for the output device with which the virtual world is to be viewed [18]. The graphics device may be simply the computer monitor or a head-mounted display. Two parameters are passed to this function. The first parameter configures the display device. For example if Crystal Eyes glasses are used the parameter passed defines the output device as Crystal Eyes glasses. The second parameter passed defines the characteristics of the window that is created using this function. For systems that have hardware support for stereo display the parameter *WTWINDOW_STEREO* can be used to specify the window as stereo. For the head-mounted display used it does not need the window configuration to be specified as stereo since the graphics driver and the HMD 3D mode handle the stereo display.

As a new universe is created before any other function calls in WTK, the universe must be deleted at the very end. This function call frees all objects used during the simulation and cleans up and closes the graphics driver or WTK display. The simulation loop, which is the core of any WTK application, is entered calling *WTuniverse_go()* function. The simulation loop performs a number of tasks. These include, calculating the frame rate, reading sensor inputs, and any user defined function, which is to be called every time the simulation loop is run, and finally the rendering of the universe. Before deleting the universe the simulation loop must be stopped. A universe action function can be defined to include user-defined actions in the simulation loop.

4.4.2 Scene Graphs

A scene is a collection of nodes specifying geometry, lights etc. along with the transform information which places these nodes at particular locations [18]. Scene graph architecture is necessary to help the grouping of objects together and to improve the performance of the rendering stage. Geometries, which are not visible from the current viewpoints are rejected and are not drawn when the scene is rendered.

A node is the basic building block of a scene graph. The topmost node is the root node, which holds all the other nodes. This node has no parents. The nodes below it are called child nodes. Any node, which holds nodes below it are called parent nodes of those nodes below it. Any node, which has a node above it are called child nodes of the node directly above it. Figure 4.8 demonstrates the parent child relationship of nodes.

WTK traverses the scene graph starting at the root node and proceeds down the graph from left to right. WTK draws the scene while traversing the scene graph. Each time a node is encountered the appropriate geometry is drawn or the transform is applied. WTK traverses the scene graph once per frame. Traversing the scene graph is the last action in a simulation loop.

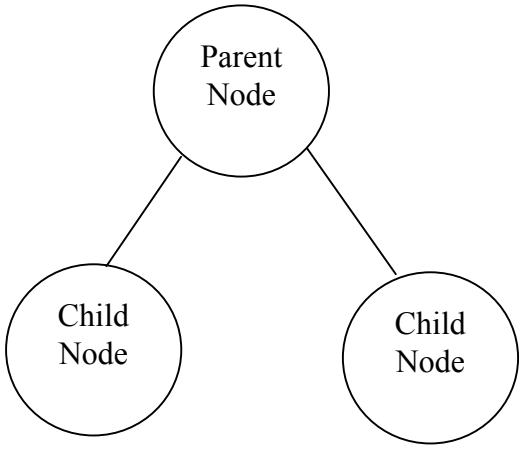

Figure 4.8. Parent child relationship. [18]

4.4.3 Nodes

New nodes can be created under the root node. Nodes, which can be moved, need to be created as movable nodes. Movable nodes are composed of three basic components. They are separator node, transform node, and content node. The separator node causes any changes applied to the transform to affect only the movable node with which it is associated. The movement of the movable node is dictated by the translation transform.

The transform node can control the position and the orientation of the node. Giving new values to the transformation matrix causes the node to move to a new position defined by the transformation matrix. The content node is the geometry node or light node, which needs to be moved. Light nodes can be added in a scene graph. As mentioned in Section 4.2.3 the greater the number of lights the worse the performance. The light nodes affect only the nodes that are below them. The types of light in WTK are ambient, direct, point and spot. The position, orientation, intensity and color of light nodes can be controlled using functions provided.

4.4.4 Viewpoints and Sensors

A WorldToolKit viewpoint defines the position and orientation from which all of the geometries associated with a simulation are rendered and projected to the computer screen [18]. Each WTK window has a viewpoint from which the scene graph is drawn. For each viewpoint the orientation, position and direction can be set to get a good view of the geometry. The code can define different viewpoints and the pinch glove can be programmed to change the viewpoint of a scene. The different parameters that are associated with a viewpoint are *position, orientation, direction, aspect ratio, parallax, convergence, and convergence distance.* Different functions can be used to create new viewpoints, delete viewpoints, copy viewpoints etc. The parallax of the viewpoint is useful for stereo viewing. Parallax is the distance between the right and left eye views in a simulation.

Sensor objects generate position, orientation and other such data that originate in the real world. Sensors include the mouse, pinch glove and the tracker. Every frame WTK receives data from the hardware devices and interprets it into useful data for the

simulation [18]. Inputs from the sensors are used to control the motion of objects in the scene and various other functions for the simulation. The mouse for example, is used to navigate around the environment when the left button is held down. From the raw data transmitted by the mouse, the particular mouse button can be determined by using parameters like *WTMOUSE_LEFTDOWN* and *WTMOUSE_RIGHTDOWN*.

A mouse sensor can be attached to a viewpoint. This can be done using an object called the *motionlink*. A *motionlink* connects a source of position and orientation information with a target that moves in correspondence with that changing set of information.

4.4.5 Mouse

WTK supports a mouse sensor. Functions are provided in the WTK library to drive the mouse and update the scene depending on the data that is received from the mouse. The mouse is initialized using the *WTmouse_new*() function. The mouse sensor is then associated with each viewpoint defined for the scene. These functions hence allow the user to fly around in the scene and zoom in to various viewpoints. A wireless mouse is used which makes it easier for the user to navigate around in the environment with a mobile system.

4.4.6 Pinch Glove

The pinch glove driver is provided along with the WTK package. A header file needs to be included in the program to be able to use the plugin driver. *WTXpinchglove_new()* may be used to open a new sensor object. The baud rate and port are passed as parameters to this function. Each frame the number of touches for that frame and the list of fingers touching for each touch event is updated. The pinch glove sensor returns the raw data in a structure, which contains the number of touches and the list of fingers in an array, which have been used for the touch event. Depending on which fingers have been used the various functions can be called to implement keyboard key hits.

4.4.7 Flock of Birds Tracker

The driver for the Flock of Birds tracker is delivered as a plugin with WTK. In order to use the drivers provided by WTK a header file needs to be included. WTK initially searches for a driver, which is provided with the tracker. This driver is in the form of a **.dll* (dynamic link library) file. If this driver is not found, WTK uses the internal driver. To initialize and activate the plugin, the hardware must be initialized using the function *WTXascension_initialize()* after creating the universe.

The reference frame for Ascension Flock of Birds is normally positive X in the forward direction, positive Y to the right and positive Z in the downward direction. WTK modifies this reference frame. The reference frame reported by Ascension sensors is modified by this driver to align with WTK's coordinate system where Z is forward, X is the to the right, and Y is down [18]. The different reference frames are shown in Figure 4.9. Absolute and relative records are used to obtain data from the sensors. *Absolute records* reported, give the position, direction and orientation of the receiver with respect to the transmitter. *Relative records* give the offset of the receiver in position, orientation and direction from the previous frame. Both records can be obtained by calling WTK

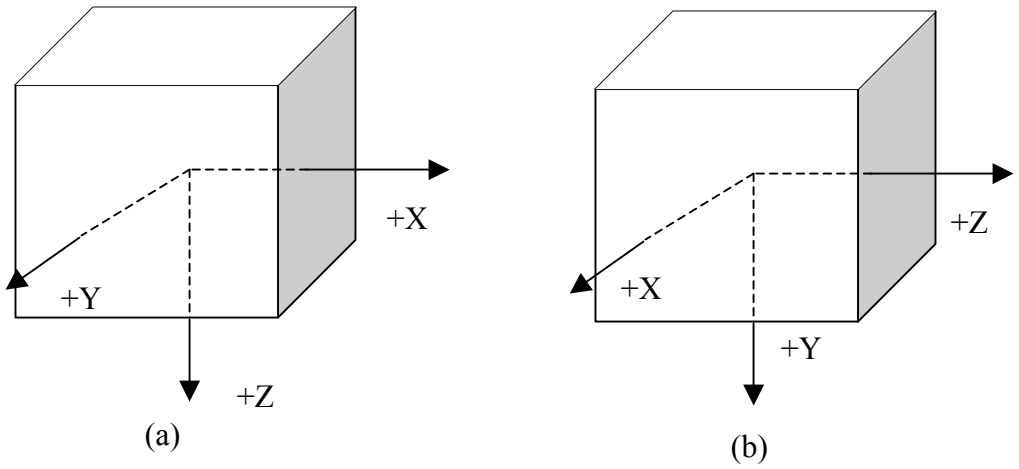

Figure 4.9. Reference frame (a) Defined by tracker (b) Defined by WTK.

functions, which issue commands to the tracker to get the position, orientation and the direction.

Since the tracker uses the RS232 port to communicate, the sensor object must be opened by calling the function *WTXascension_newRS232()*. The parameters passed to this function are the port, the baud rate, the mode of operation and unit. The port specifies to which port the tracker is connected (serial port 1 or 2). In this project the tracker is connected to COM2. The baud rate is defined by the DIP-switch setting. The mode of operation may be standalone; that is if only one bird (sensor) is connected or flock mode if more than one bird is connected. The unit specifies the number of birds in the flock. The absolute record that the bird communicates to the software is updated every frame when a record is available from the device. The update rate of the flock can also be adjusted depending on the requirements. The hemisphere of operation of the flock of birds can also be changed depending on the requirements.

The functions that have been used for purposes of this thesis are those provided directly by the Ascension tracker. In order to use the functions directly provided by the manufacturer, it is necessary to link a library file *bird.lib*. In addition to this, a header file *Bird.h* must be included in the main program, and *bird.dll* must be present in the same directory as the software. In order to wake up the bird the function used is *birdRS232WakeUp()*. The parameters passed to this function, and the description of each parameter is as shown in Table 4.1. In case where the bird cannot be detected by the software or an incorrect parameter is passed to the function the software exits.

Functions like *birdGetSystemConfig()* and *birdGetDeviceConfig()* can be used to obtain the system configuration and the device configuration of the Flock of Birds. These functions return the configuration of the bird in a structure. The configuration settings can be changed by changing the parameters in the structure. The structure can be then returned by using the respective *set* functions. To obtain a stream of data the function *birdStartFrameStream()* may be used. This function initiates frame capture. Before streaming in the data, it has to be assured that the flock is ready to send the data to the software. This can be done by the function *birdGetMostRecentFrame()*. The bird returns the position and orientation information in 16-bit values.

Table 4.1. Parameters and functions.

Parameter	Data Type	Function
GroupId	Integer	Enables the user to connect multiple flocks and address each flock separately.
Standalone	Boolean	True if only a single bird is connected. In this case False since three birds are connected
NumDevices	Integer	If the flock is not in standalone it defines the number of birds in the flock. In this case three.
Comport	WORD array	Comport to which the bird is connected. COM2=2
BaudRate	DWORD	Defines the baud rate as per DIP switch setting
ReadTimeOut	DWORD	Maximum time in msecs the application takes when trying to receive a character.
WriteTimeOut	DWORD	Maximum time the application will take when trying to transmit a character.

Table 4.2 shows the format of the frame returned by the function. Table 4.3 shows the contents of the *birdreading* structure. This structure further contains structures for position, angle, matrix, quaternion and buttons. Only position data have been used. This structure contains X, Y and Z readings.

The X, Y and Z readings are scaled to represent floating point values. The variable *pos* is the position value returned by the bird and *ang* is the angle value returned by the bird. Equations 4.1 and 4.2 demonstrate how these can be converted into floating point values. *birdStopFrameStream()* is used to stop the streaming of data. In order to close the connection *birdShutDown()* must be used. This function stops all data transmission, turns off the transmitter and disables all further communication with the group.

The *Scale* factor in Equation 4.1 is a value returned in the device configuration structure. It represents the full-scale measurement in inches.

Table 4.2. *BIRDFRAME* structure.

Field	Type	Description
dwTime	DWORD	Time Stamp in milliseconds
Reading	BIRDREADING	Reading from each bird

Table 4.3. *BIRDREADING* structure.

Field	Type	Description
Position	BIRDPOSITION	Position of receiver
Angles	BIRDANGLES	Orientation of receiver
Matrix	BIRDMATRIX	Orientation as matrix
Quaternions	BIRDQUATERNION	Orientation as quaternion
wButtons	WORD	Button State

$$Position = pos \times \frac{Scale}{32767} \tag{4.1}$$

$$Angle = ang \times \frac{180}{32767} \tag{4.2}$$

4.5 Operations and Navigation

The program can be executed from the command line. The parameter passed to the program is the name of the model, which is to be loaded. For this thesis the McGhee Tyson airport model is used. The entire airport is seen initially. Figure 4.10 shows the initial view. The left mouse button can be used to zoom in or out of the scene. When the mouse cursor is in the top half of the scene the scene zooms in. If the mouse cursor is in the lower half of the scene, the view zooms out. Similarly if the mouse cursor is in the left half of the screen the scene turns left and vice versa if the mouse cursor is in the right half of the scene. The right mouse button can be used for panning in the scene.

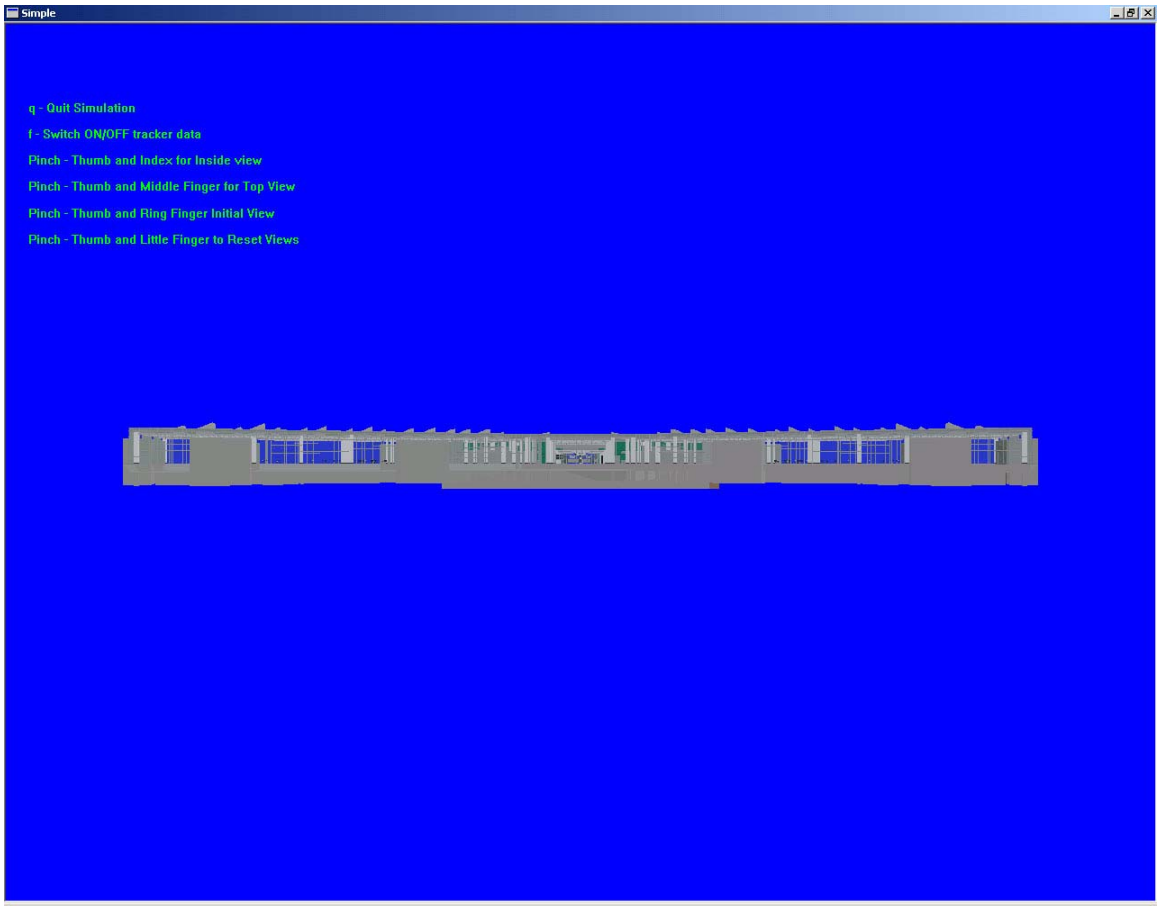

Figure 4.10. Initial view of McGhee Tyson airport.

It has to be assured that both the pinch glove system and the Flock of Birds must have power supplied to them when the program is started up. On the left hand side of the screen a list of functions can be seen which can help the user to use the keyboard commands and the pinch glove features to navigate around the screen. The F-key on the keyboard can be hit to switch on or off the position data from the tracker. If this feature is switched on the positions of the three sensors can be seen. The display shows the X, Y and Z co-ordinates on the screen.

The pinch glove is used to change the viewpoints in the scene. When contact is made between the thumb and the index finger, the viewpoint is changed to the inside of the airport. Figure 4.11 depicts this viewpoint. When contact is made between the middle finger and the thumb, the top view of the airport can be seen. This is shown in Figure 4.12. When contact is made between the ring finger and the thumb, the viewpoint is the initial viewpoint from outside the airport. This is as shown in Figure 4.10. In each of these views the mouse can be used to zoom in or out of the scene thus changing the viewpoint coordinates. Thus, if one loses the scene during navigation then making contact between the little finger and the thumb resets all the viewpoints and brings the user back to the first viewpoint, that of the inside of the airport. The left glove or the right glove can be used for these operations. When the Q-key on the keyboard is hit the simulation stops and the window closes. Also the Flock of Birds is shut down. Figure 4.13 shows the function listing screenshot.

For the head-mounted display, since the HMD itself along with the graphics driver provide 3D stereovision, no adjustments need to be made as far as parallax and convergence are concerned. As stated in the Chapter 3 either the 3D1 or 3D2 mode can be used for stereoscopic vision.

Figure 4.11. Inside view of airport model.

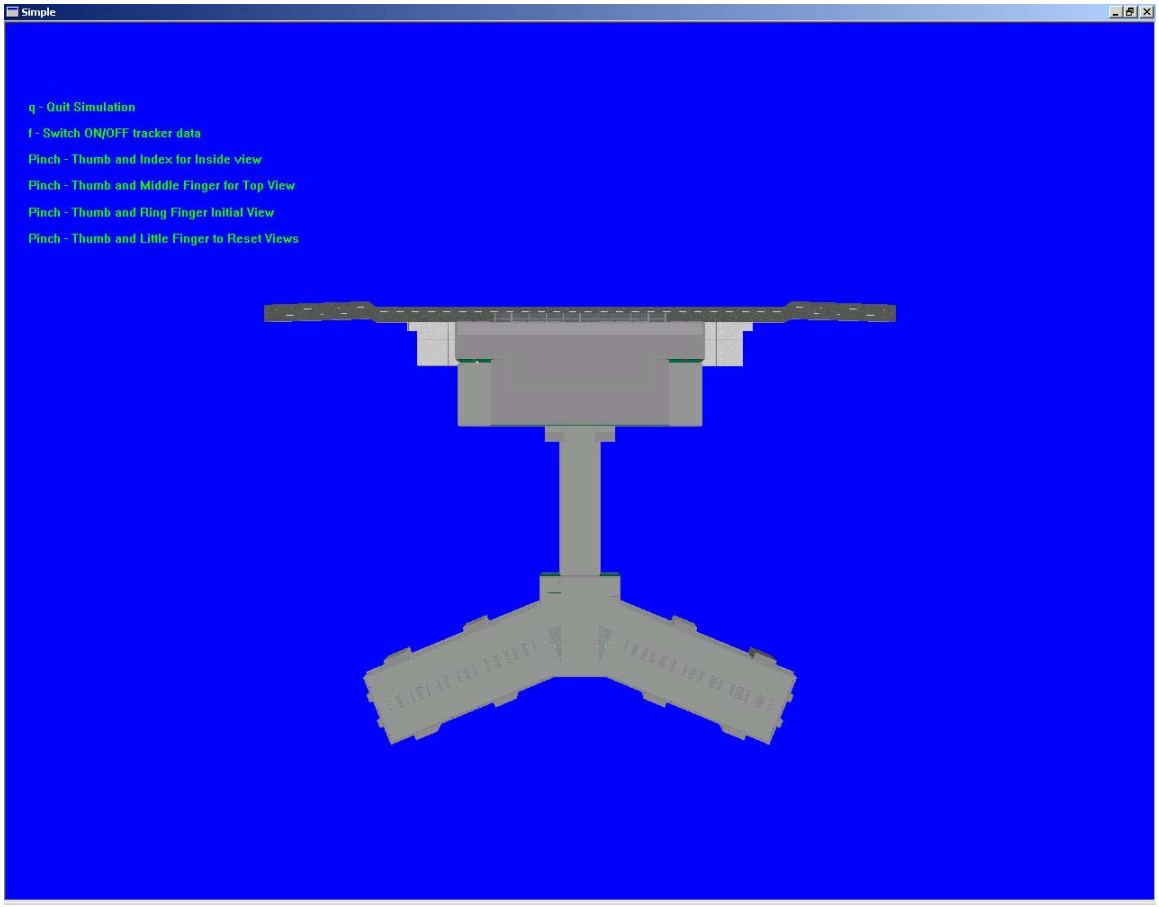

Figure 4.12. Top view of airport.

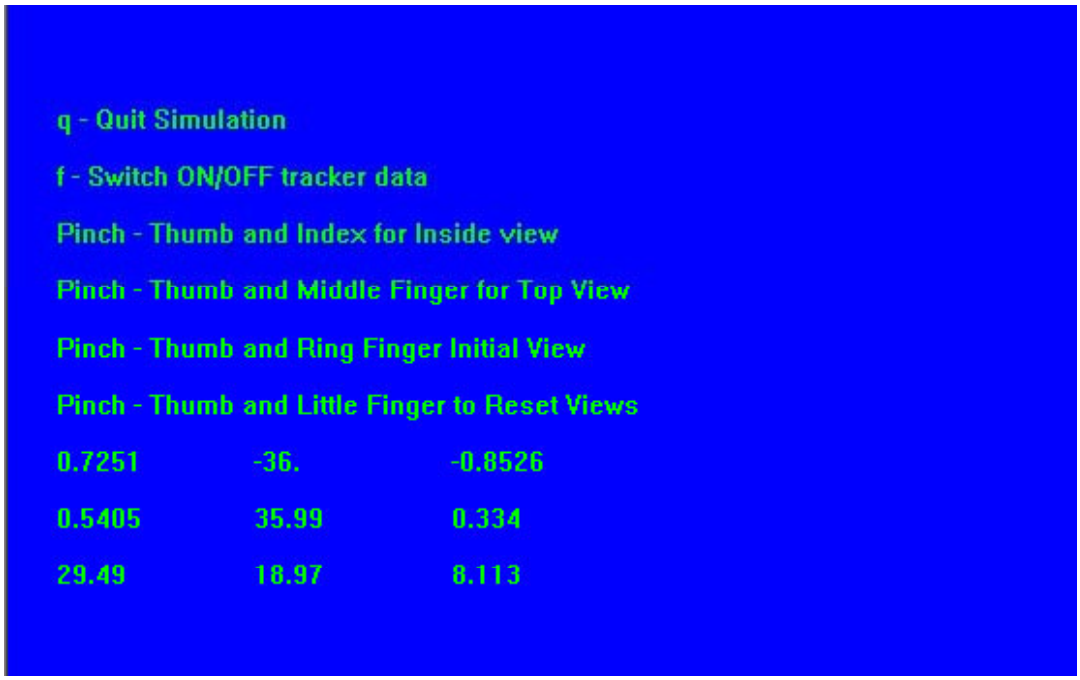

Figure 4.13. Command function listing.

4.6 Issues

All the procedure mentioned in Section 4.3 need to be followed in order to load a model created in 3D Studio MAX into the WTK environment. Attaching the model is important since the model is loaded as a node into the WTK environment. If the geometries are not merged into one mesh each of the geometries may be treated as a different node. In addition it is necessary that the model have its midpoint in the center. WTK uses the midpoint along with the transformation matrix to obtain the location of the object in a scene. If the midpoint is not in the center the positioning of the object may be erroneous.

Both the pinch glove and the tracker must be powered up before the executable is run. The tracker must be reset using the FLY/STANDBY switch located on the front panel so that all registers including the error registers are cleared before the program is run. The model must be loaded at the command line when the program is executed, so that the model can be changed if required.

Chapter 5
Conclusions and Future Work

5.1 Results and Conclusions

VR has become a useful and productive technique that industry uses for data exploration, mission planning, and training. Wearable VR systems have become very useful in fieldwork where the user can perform operations on location. Making such systems wire-free and low-power is necessary.

5.1.1 Summary

Such a system has been implemented as described in the previous chapters. The system includes two important aspects. These are the hardware and the software. The hardware includes the head-mounted display, the wireless mouse, the pinch glove and the tracker. All these devices have been interfaced to the computer. These devices talk to the computer through the software developed. The software effort consisted of using the modeling tool 3D Studio MAX and the actual programming using WorldToolKit.

The head-mounted display is used as a display device. The wireless mouse is used to zoom into and out of the scene. It is also used for panning. The pinch glove is used to change the scene viewpoint. The tracker data is displayed on the scene. With 3D Studio MAX the necessary changes were made to the McGhee Tyson airport model so that it can be loaded into the WorldToolKit environment. Function calls in WorldToolKit are used to enable the tracker, the pinch glove and the wireless mouse to communicate with the computer and interact with the VR environment.

Still a large amount of work needs to be done on this system to make it wearable and portable. Some of the future work that can be done on this system is explained next.

5.2 Future Work

There is still much work to be done to improve the system and make it portable and wire-free.

5.2.1 Pinch Glove

A drawback of the pinch glove used is that it is wired, consumes more power and cannot be operated with a battery pack. An alternative to this would be to use a 5DT data glove, which is explained in Section 3.6.2. This data glove uses less power and can be battery powered. In addition, this data glove also has the feature that the finger configuration can be recognized. This feature may prove useful if the user can pick objects in the virtual reality environment scene.

5.2.2 Tracker

The tracker, similar to the pinch glove, is wired and uses a lot of power. An alternative to this would be to use a wire-free tracker. In addition, the tracker requires a second serial port, which is not available in ordinary laptop or notebook computers. A tracker, which communicates through the USB port of the computer, becomes more suitable for this mobile application.

Also, the data returned by the tracker is currently just displayed in the scene. More work could be done with this data. This data could be interpreted and used to move around in the scene. So, that if the sensor is fastened to the user's head along with the head-mounted display, the angle and position values of the tracker could be used. This would assist the user in looking around the scene. When the sensor is attached to the wrist, hand movements can be used to navigate around in the virtual environment.

5.2.3 Sensors

An improvement to this system would be to add live data from the real world. Sensors such as smoke detectors are located in the real natural environment. These sensors could be made to send data wirelessly to the mobile system. This data could then be then read and interpreted. This would help the user to detect any kind of emergency in the real environment. In addition data from a GPS receiver could be used so that the exact location of the user can be known in the real world.

All of these improvements are really just refinements of the basic proof-of-concepts demonstrated in this thesis.

Bibliography

Bibliography

[1] William R. Sherman, Alan B. Craig, *Understanding Virtual Reality Interface, Application, and Design,* Morgan Kaufmann Publishers.

[2] John Vince, *Virtual Reality Systems,* Addison-Wesley Publishing Company

[3] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, Blair MacIntyre, "Recent Advances in Augmented Reality", *IEEE Computer Graphics and Applications* vol 21, no. 6, November/December 2001, pp 34-44.

[4] Jim Vallino, "*Introduction to Augmented Reality,* Augemented Reality Home Pages http://www.se.rit.edu/~jrv/research/ar/introduction.html" accessed on July 13, 2004

[5] Dr. Frank van Diggelen Global Locate Inc., "Indoor GPS and Implementation", *IEEE Position, Location and Navigation Symposium*, 2002.

[6] HowStuffWorks.com " How augmented reality will work , http://computer.howstuffworks.com/augmented-reality1.htm", Internet website accessed on July 16th 2004.

[7] Lawrence Rosenblum, Michael Macedonia, "Developing Killer Apps for Industrial Augmented Reality", *IEEE Computer Graphics and Applications* vol 24, no. 3, May/June 2004, pp 16-20

[8] John R. Lewis, "In the Eye of the Beholder", *IEEE Spectrum* vol 41, no.5, May 2004, pp.24-28.

[9] Kenneth Salisbury, Francois Conti and Federico Barbagli, "Haptic Rendering: Introductory Concepts", *IEEE Computer Graphics and Applications* vol 24, no. 2, March/April 2004, pp 24-31.

[10] i-O Display Systems (2003), *High Resolution Computer Monitor User's Manual*.

[11] IOGEAR, Inc, "Phaser Wireless Trackball Mouse GME322R User's Manual, 2003, http://www.iogear.com/support/manual/GME322Rmanual.pdf"

[12] Fakespace Inc. (1995), *Fakespace Pinch Glove System Installation Guide and User Handbook.*

[13] Doug A. Bowman, Vinh Q. Ly and Joshua M. Campbell, "Pinch Keyboard: Natural Text Input for Immersive Virtual Environments, Virginia Tech University, http://people.cs.vt.edu/~bowman/papers/pinch_keyboard.pdf."

[14] Ascension Technology Corporation, *The Flock of Birds Position and Orientation Measurement System Installation and Operation Guide*, 910002-A Rev A, January 18,1999.

[15] Borko Furht, Ph.D., Mohammed Ilyas, Ph.D., *Wireless Internet Handbook Technologies, Standards, and Applications,* CRC Press

[16] Fifth Dimensions Technology (5DT), *"5DT Data Glove 5DT Data Glove 5-W Data Glove for the fifth dimension User's Manual,* Version 2.00, February 2000, http://www.5dt.com/downloads/5DTDataGlove5Manual.pdf"

[17] AutoDesk Inc., 3ds max 4 User Reference, 2001

[18] BMA/Sense8 Incorporated, *WorldToolKit Reference Manual Release 10,* Version 2003.

[19] Sense8 Incorporated, *WorldToolKit R10 Sensor Driver Reference Ascension Technology Corporation*, Version 1.3, December 2003.

[20] Martin Goebel, Michitaka Hirose and Lawrence Rosenblum, "Today's VR*", IEEE Computer Graphics and Applications* vol 21, no. 6, November/December 2001.

[21] Theresa-Marie Rhyne and Lloyd Treinish, "Visualization Viewpoints*", IEEE Computer Graphics and Applications* vol. 21, no. 2, March/April 2001.

[22] Ascension Technology, *Windows Driver User Manual,* Version May 2004.

[23] Rashmi Bajaj, Samantha Lalinda Ranaweera, Dharma P. Agarwal, "GPS: Location – Tracking Technology*", Computer* April 2002, IEEE Computer Society.

[24] Goran M. Djuknic and Robert E. Richton Bell Laboratories, Lucent Technologies, , "*Geolocation and Assisted GPS*, http://www.lucent.com/livelink/090094038000e51f_White_paper.pdf"

[25] Upkar Varshney, "The Status and future of 802.11-Based WLANs", *IEEE Computer*, vol. 36, no. 6, June 2003.

Appendix

Appendix

The following pages provide the C code used for this thesis.

```
/*
 * simple.c
 * Copyright (c) 1993-97 Sense8 Corporation
 *
 * This program loads in a model specified on the command line
 * and lets you navigate your viewpoint using a Mouse.
 * Press the 'q' key to quit.
 *
Usage:
        simple <modelname>
 */

/* ------------------------------------------------------------------*/
/*          Neena Nambiar                                          */
/*         Code for Thesis Modified from Simple.c                  */
/* Added functionality for Pinch Glove, Tracker and other functions   */
/*------------------------------------------------------------------*/

/* Include this header with all WTK applications */
#include "wt.h"
#include "wtx_pinchglove.h"
#include "wtkui.h"
#include "wtx_ascension.h"
#include "Bird.h"
#include <stdlib.h>
#define LOCAL_MAX_BIRDS  10
```

```
WTviewpoint *GetViewpointByName(char *ViewpointName);
void ForegroundActionFunction(WTwindow *win,int eye);
void ShowHelp(WTwindow *Window);
void CreateRenderingWindow(void);
WTui *shell = NULL;
WTui *toplevel;
FLAG Showbird =FALSE;

int countdisp=0;

/* variables for tracker data*/
double pos[3];
double ang[3];
double pos_scale;
double pos1[3];
double ang1[3];
double pos_scale1;
double pos2[3];
double ang2[3];
double pos_scale2;
double pos0[3];
double ang0[3];
double pos_scale0;
/*
 * The universe's action function is called each time through the simulation
 * loop. This particular action function looks for and acts upon keyboard
 * key presses. The 'q' key ends the program by calling WTuniverse_stop().
 */
```

```
WTsensor *pinch,*bird;
static void actionfn();
WTq Orientation;
WTp3 Direction, Position,dir,p;
int devcnt;
int i;
BOOL check=FALSE;
BIRDDEVICECONFIG devconfig[LOCAL_MAX_BIRDS + 1];
FLAG StartStream = FALSE;
WTXpinchglove_rawdata *raw;
WTnode *root;

void main(int argc, char *argv[])
{
        WTsensor *sensor;     /* the Mouse */
        WTnode *rootnode;   /* the universe rot node */
        WTviewpoint *Viewpoint = NULL;
        WTq q,Rotation;
        WTp3 Translation;
        void *Target = NULL;
        void *Source = NULL;
        FLAG flockinit=FALSE;
        float update_bird;
        WTmotionlink *MotionLink = NULL;
        int hem_bird;
        char *MotionLink1,*MotionLink2,*MotionLink3;
        WTnode *Child;

        WORD comport[LOCAL_MAX_BIRDS + 1];
        BIRDSYSTEMCONFIG sysconfig;
```

```
BIRDDEVICECONFIG *pDeviceConfig     ;
/* check for "-usage" flag */
toplevel=WTui_init(&argc,argv);

// added to provide pinch glove functionality
WTuniverse_new(WTDISPLAY_NOWINDOW, WTWINDOW_DEFAULT);
WTui_showconsole(TRUE);
root = WTuniverse_getrootnodes();
WTnode_setname(root,"root");

if ( argc>1 ) {
        if( strcmp( argv[1], "-usage" ) == 0 ) {
                WTmessage( "Usage: wtk [<modelname>]\n" );
                WTmessage( "This program loads in a model specified \
                                        on the command line\n" );
                WTmessage( "and lets you navigate your viewpoint \
                                        using a Mouse.\n" );
                WTmessage("Hit any key to exit\n");
                getchar();
                exit( 0 );
        }
}


/* Initialize the universe.
 * (This must be the 1st WorldToolKit call.)
 */
/*Setting path fro the models and images  */
WTinit_setimages( "./data;../data" );
WTinit_setmodels( "./data;../data" );
```

```
/* Initialise the pinch glove            */
        WTXpinchglove_initialize();
        pinch = WTXpinchglove_new(WTSERIAL1,9600);

        /* initialise the flock of birds device*/
/* for direct initialisation                    */

        // wait 10 seconds for light to stop blinking
        Sleep(1000);

        // wake up a flock of birds using the RS232 interface

        // tell driver to look on comport 1 for master bird
        comport[1] = 2;
        for (i = 2; i <= LOCAL_MAX_BIRDS; i++)
        {
                comport[i] = 0;
        }

        if (!birdRS232WakeUp(1,FALSE,3,comport,9600,2000,2000))
        {
                MessageBox(NULL,"Unable to connect with Bird at Com1:",
                "Bird Error",0);
                exit(-1);
        }

        // get the system configuration
        birdGetSystemConfig(1,&sysconfig);
```

```
// check to see how many devices are really present

for (i = 0, devcnt= 0; i < LOCAL_MAX_BIRDS; i++)
{
        if (sysconfig.byFlockStatus[i] & BFS_FBBACCESSIBLE)
                devcnt++;
}

printf("devcnt = %d\n",devcnt);
birdShutDown(1);
if (!birdRS232WakeUp(1,FALSE,devcnt,comport,9600,2000,2000))
{
        MessageBox(NULL,"Unable to connect with Bird at Com1:",
        "Bird Error",0);
        exit(-1);
}
// get a device configuration for each device

for (i = 1; i <= devcnt; i++)
{
        check = birdGetDeviceConfig(1,i,&devconfig[i]);
        printf("birdgetdeviceconfig  = %d\n",check);
}

check = birdStartFrameStream(1);

printf("birdStartFrameStream  = %d\n",check);


/* get a pointer to the universe root node */
```

```
/* load lights from "lights.lit", as the first child */
WTlightnode_load( root, "lights.lit");

/* Now load the object specified on command line */
WTnode_load( root, argv[1], 1.0f );

/* Zoom the viewpoint out until the whole model is visible */

/* Initialize a Mouse sensor */
sensor = WTmouse_new();
if (!sensor) WTerror("Couldn't find mouse\n");
/* Attach the sensor to the viewpoint. */

WTviewpoint_addsensor(WTuniverse_getviewpoint(), sensor);
WTmessage("the file is :+++ %s", argv[1]);

/* Scale sensor sensitivity with the size of the universe. */
WTsensor_setsensitivity(sensor, 0.04f*WTnode_getradius(root));

WTviewpoint_getorientation(WTuniverse_getviewpoint(),q);
WTviewpoint_getposition(WTuniverse_getviewpoint(),p);
WTviewpoint_getdirection(WTuniverse_getviewpoint(),dir);

//New viewpoint to reset when the pinch glove is used with the index and the
thumb
Viewpoint = WTviewpoint_new();

WTviewpoint_setname(Viewpoint,"Viewpoint-1");
Orientation[X] = (float) 0.000000;
```

```
        Orientation[Y] = (float) 0.000000;
        Orientation[Z] = (float) 0.000000;
        Orientation[W] = (float) 1.000000;
        WTviewpoint_setorientation(Viewpoint,Orientation);
        WTviewpoint_setconvdistance(Viewpoint,100.000000);
        Direction[X] = (float) 0.000000;
        Direction[Y] = (float) 0.000000;
        Direction[Z] = (float) 0.000000;
        WTviewpoint_setdirection(Viewpoint,Direction);
        WTviewpoint_setconvergence(Viewpoint,0);
        Position[X] = (float) 0.000000;
        Position[Y] = (float) -5.000000;
//      Position[Y] = (float) -2750.245972;
        Position[Z] = (float) -150.000000;
//      Position[Z] = (float) -5700.140503;
        WTviewpoint_setposition(Viewpoint,Position);
        WTviewpoint_setparallax(Viewpoint,0.000000);
        Viewpoint = NULL;

//New viewpoint to reset when the pinch glove is used with the middle  and the thumb
        Viewpoint = WTviewpoint_new();
        WTviewpoint_setname(Viewpoint,"Viewpoint-2");
        Orientation[X] = (float) 0.707107;
        Orientation[Y] = (float) 0.000000;
        Orientation[Z] = (float) 0.000000;
        Orientation[W] = (float) 0.707107;
        WTviewpoint_setorientation(Viewpoint,Orientation);
        WTviewpoint_setconvdistance(Viewpoint,(float) 0.004000);
        Direction[X] = (float) 0.000000;
        Direction[Y] = (float) 1.000000;
```

```
Direction[Z] = (float) 0.000000;
WTviewpoint_setdirection(Viewpoint,Direction);
WTviewpoint_setconvergence(Viewpoint,0);
Position[X] = (float) 0.000000;
Position[Y] = (float) -300.058228;
Position[Z] = (float) -70.453003;
WTviewpoint_setposition(Viewpoint,Position);
WTviewpoint_setparallax(Viewpoint,0.000000);
Viewpoint = NULL;

/*New viewpoint to reset when the pinch glove is used with the ring finger
  and the thumb*/
Viewpoint = WTviewpoint_new();
WTviewpoint_setname(Viewpoint,"Viewpoint-3");
Orientation[X] = (float) 0.000000;
Orientation[Y] = (float) 0.000000;
Orientation[Z] = (float) 0.000000;
Orientation[W] = (float) 1.000000;
WTviewpoint_setorientation(Viewpoint,Orientation);
WTviewpoint_setconvdistance(Viewpoint,(float) 0.004000);
Direction[X] = (float) 0.000000;
Direction[Y] = (float) 0.000000;
Direction[Z] = (float) 0.000000;
WTviewpoint_setdirection(Viewpoint,Direction);
WTviewpoint_setconvergence(Viewpoint,0);
Position[X] = (float) 0.000000;
Position[Y] = (float) -5.000000;
Position[Z] = (float) -300.000000;
WTviewpoint_setposition(Viewpoint,Position);
WTviewpoint_setparallax(Viewpoint,0.000000);
```

```
        Viewpoint = NULL;
        WTviewpoint_addsensor(GetViewpointByName("Viewpoint-1"), sensor);
        WTviewpoint_addsensor(GetViewpointByName("Viewpoint-2"), sensor);
        WTviewpoint_addsensor(GetViewpointByName("Viewpoint-3"), sensor);

/* Create two lights and load the light node */

        Child = WTmovlightnode_newdirected(NULL);
        WTnode_setname(Child,"Light-1");
        Rotation[X] = (float) 0.55991;
        Rotation[Y] = (float) -0.188428;
        Rotation[Z] = (float) 0.152609;
        Rotation[W] = (float) 0.792279;
        WTnode_setorientation(Child,Rotation);
        Translation[X] = (float) -197.422394;
        Translation[Y] = (float) -200.765427;
        Translation[Z] = (float) 2.869659;
        WTnode_settranslation(Child,Translation);
        WTnode_boundingbox(Child,FALSE);
        WTlightnode_setintensity(Child,(float) 0.15);
        WTnode_enable(Child,TRUE);
        WTlightnode_setambient2(Child,1.0,1.0,1.0);
        WTlightnode_setdiffuse2(Child,1.0,1.0,1.0);
        WTlightnode_setspecular2(Child,1.0,1.0,1.0);
        Child = NULL;

        /* Create two lights and load the light node */
        Child = WTmovlightnode_newdirected(NULL);
        WTnode_setname(Child,"Light-2");
        Rotation[X] = (float) 0.118062;
```

```
        Rotation[Y] = (float) 0.647979;
        Rotation[Z] = (float) -0.102629;
        Rotation[W] = (float) 0.745420;
        WTnode_setorientation(Child,Rotation);
        Translation[X] = (float) 296.677643;
        Translation[Y] = (float) -200.765427;
        Translation[Z] = (float) 0.000000;
        WTnode_settranslation(Child,Translation);
        WTnode_boundingbox(Child,FALSE);
        WTlightnode_setintensity(Child,(float) 0.15);
        WTnode_enable(Child,TRUE);
        WTlightnode_setambient2(Child,1.0,1.0,1.0);
        WTlightnode_setdiffuse2(Child,1.0,1.0,1.0);
        WTlightnode_setspecular2(Child,1.0,1.0,1.0);
        Child = NULL;

        Child = WTuniverse_findnodebyname("Light-1",0);
        WTnode_insertchild(root,Child,0);
        Child = WTuniverse_findnodebyname("Light-2",0);
        WTnode_insertchild(root,Child,1);

        WTmessage("flock is initialised = %d",flockinit);
        shell =
WTuiform_new(toplevel,"Simple",WTUIATT_LEFT,0,WTUIATT_TOP,0,WTUIATT_
WIDTH,600,WTUIATT_HEIGHT,600,NULL);

        CreateRenderingWindow();

        WTui_manage(shell);
        WTwindow_zoomviewpoint(WTuniverse_getwindows());
```

```
        /* open the keyboard */
        WTkeyboard_open();

        /* Prepare the universe for start of the simulation. */
        WTuniverse_ready();

        /* Set the action function so that button presses will be acted on. */
        WTuniverse_setactions(actionfn);

        /* enter the main simulation */
        WTui_go(toplevel, TRUE);


        /* all done; clean everything up. (This must be the last WTK call.) */
//      WTuniverse_delete();
}

/*Creates the rendering window =, sets the viewpoint and
draws text of function list  */
void CreateRenderingWindow(void)
{
        WTviewpoint *Viewpoint;
        WTwindow *Window;
        Window = WTuiwtkwindow_new(shell,WTWINDOW_DEFAULT);
        WTwindow_setname(Window,"Window-1");
        WTwindow_setrootnode(Window,root);
        Viewpoint = GetViewpointByName("Viewpoint-1");
        WTwindow_setviewpoint(Window,Viewpoint);
```

```
        WTwindow_add2Ddrawfn(Window,ForegroundActionFunction);
}

/*
 * Checks for keyboard presses and pinch glove.
 * Exits the simulation loop if both buttons are pressed.
 */
static void actionfn()
{
        short key;
        short rp,rr,rm,ri,rt,lp,lr,lm,li,lt;
        int i;
        FLAG flgset;
        BIRDFRAME frame;
        BIRDREADING *preading;

        /* get key presses, if any */
        key = WTkeyboard_getlastkey();

        if ( key=='q' )
        {
                printf("q is pressed");
                birdShutDown(1);
                WTuniverse_stop();
        }

        if ( key=='?' )
        {
                printf("\nPress 'q' key to quit\n ");
```

```
		printf("To change to Top Viewpoint contact between the middle finger and the thumb");
		printf("To change to Viewpoint inside the airport contact between the index finger and the thumb");
		printf("To reset all Viewpoints contact between the pinkie and the thumb and then between index and thumb");
	}

	if (key == 'f')
	{
		if(Showbird==TRUE)
			Showbird=FALSE;
		else
			Showbird=TRUE;
	}

	if (pinch==NULL)
		WTmessage("Pinch is NULL");
	raw =(WTXpinchglove_rawdata *) WTsensor_getrawdata(pinch);
	if (raw==NULL)
		WTmessage("Raw is null");

	/* Attach the sensor to the viewpoint. */
	raw->ntouch=5;


/*---------- data from tracker   ------------------*/
	StartStream = TRUE;
	if (StartStream == TRUE)
	{
```

```
if( countdisp==100)
{
        countdisp = 0;
        /* check if the bird is ready to send data   */
        if (birdFrameReady(1))
        {
                check = birdGetMostRecentFrame(1,&frame);

                for (i = 1; i <= devcnt; i++)
                {
                        // reading array is indexed by device address (1
relative)
                        preading = &frame.reading[i];
                        pos_scale = devconfig[i].wScaling;

                        // convert position and angle data
                        pos[0] = preading->position.nX * pos_scale /
32767.;
                        pos[1] = preading->position.nY * pos_scale /
32767.;
                        pos[2] = preading->position.nZ * pos_scale /
32767.;
                        ang[0] = preading->angles.nAzimuth * 180. /
32767.;
                        ang[1] = preading->angles.nElevation * 180. /
32767.;
                        ang[2] = preading->angles.nRoll * 180. / 32767.;


                        // print to screen
```

```
                    WTmessage("Bird %d: %+05.1f %+05.1f %+05.1f
|",
                                        i, pos[0], pos[1], pos[2]);
                    WTmessage("   %+06.1f %+06.1f %+06.1f \n",
                                        ang[0], ang[1], ang[2]);
                }
                preading = &frame.reading[2];
                pos_scale1 = devconfig[2].wScaling;

                // convert position and angle data
                pos1[0] = preading->position.nX * pos_scale / 32767.;
                pos1[1] = preading->position.nY * pos_scale / 32767.;
                pos1[2] = preading->position.nZ * pos_scale / 32767.;
                ang1[0] = preading->angles.nAzimuth * 180. / 32767.;
                ang1[1] = preading->angles.nElevation * 180. / 32767.;
                ang1[2] = preading->angles.nRoll * 180. / 32767.;

                preading = &frame.reading[3];
                pos_scale2 = devconfig[3].wScaling;

                // convert position and angle data
                pos2[0] = preading->position.nX * pos_scale / 32767.;
                pos2[1] = preading->position.nY * pos_scale / 32767.;
                pos2[2] = preading->position.nZ * pos_scale / 32767.;
                ang2[0] = preading->angles.nAzimuth * 180. / 32767.;
                ang2[1] = preading->angles.nElevation * 180. / 32767.;
                ang2[2] = preading->angles.nRoll * 180. / 32767.;


                preading = &frame.reading[1];
```

```
                        pos_scale1 = devconfig[1].wScaling;

                        // convert position and angle data
                        pos0[0] = preading->position.nX * pos_scale / 32767.;
                        pos0[1] = preading->position.nY * pos_scale / 32767.;
                        pos0[2] = preading->position.nZ * pos_scale / 32767.;
                        ang0[0] = preading->angles.nAzimuth * 180. / 32767.;
                        ang0[1] = preading->angles.nElevation * 180. / 32767.;
                        ang0[2] = preading->angles.nRoll * 180. / 32767.;
                }
        }
        else
                countdisp = countdisp+1;
}

//WTmessage("Number of touches %d\n",raw->ntouch);

for (i=0;i<raw->ntouch;i++)
{
        rp = raw->touch[i]&WTXPINCHGLOVE_RPINKIE;
        rr = raw->touch[i]&WTXPINCHGLOVE_RRING;
        rm = raw->touch[i]&WTXPINCHGLOVE_RMIDDLE;
        ri = raw->touch[i]&WTXPINCHGLOVE_RINDEX;
        rt = raw->touch[i]&WTXPINCHGLOVE_RTHUMB;
        lp = raw->touch[i]&WTXPINCHGLOVE_LPINKIE;
        lr = raw->touch[i]&WTXPINCHGLOVE_LRING;
        lm = raw->touch[i]&WTXPINCHGLOVE_LMIDDLE;
        li = raw->touch[i]&WTXPINCHGLOVE_LINDEX;
        lt = raw->touch[i]&WTXPINCHGLOVE_LTHUMB;
```

```
		if(((rr) && (rt))||((lr) && (lt)))
		{
			//back to initial view

	WTwindow_setviewpoint(WTuniverse_getwindows(),GetViewpointByName("Vi
ewpoint-3"));
		}
		if(((rm) && (rt))||((rm) && (rt)))
		{
			//Top Viewpoint
			WTmessage("rmiddle and the thumb");

	WTwindow_setviewpoint(WTuniverse_getwindows(),GetViewpointByName("Vi
ewpoint-2"));
		}
		if(((ri) && (rt))||((li) && (lt)))
		{
			// Viewpoint inside the airport
			WTviewpoint_getposition(GetViewpointByName("Viewpoint-1"),
Position);
			WTmessage("Position[X] = %f,Position[Y] = %f,Position[Z]
=%f,",Position[X],Position[Y],Position[Z] );

	WTwindow_setviewpoint(WTuniverse_getwindows(),GetViewpointByName("Vi
ewpoint-1"));
		}
		if (((rp) && (rt)) || ((lp) && (lt)))
		{
			// Reset all viewpoints
			Orientation[X] = (float) 0.000000;
```

```
                    Orientation[Y] = (float) 0.000000;
                    Orientation[Z] = (float) 0.000000;
                    Orientation[W] = (float) 1.000000;
                    WTviewpoint_setorientation(GetViewpointByName("Viewpoint-
1"),Orientation);
                    Direction[X] = (float) 0.000000;
                    Direction[Y] = (float) 0.000000;
                    Direction[Z] = (float) 0.000000;
                    WTviewpoint_setdirection(GetViewpointByName("Viewpoint-
1"),Direction);
                    Position[X] = (float) 0.000000;
                    Position[Y] = (float) -5.000000;
                    Position[Z] = (float) -150.000000;
                    flgset =
WTviewpoint_setposition(GetViewpointByName("Viewpoint-1"),Position);
                    WTviewpoint_getposition(GetViewpointByName("Viewpoint-1"),
Position);

                    Orientation[X] = (float) 0.707107;
                    Orientation[Y] = (float) 0.000000;
                    Orientation[Z] = (float) 0.000000;
                    Orientation[W] = (float) 0.707107;
                    WTviewpoint_setorientation(GetViewpointByName("Viewpoint-
2"),Orientation);
                    Direction[X] = (float) 0.000000;
                    Direction[Y] = (float) 1.000000;
                    Direction[Z] = (float) 0.000000;
                    WTviewpoint_setdirection(GetViewpointByName("Viewpoint-
2"),Direction);
                    Position[X] = (float) 0.000000;
```

```
                        Position[Y] = (float) -300.058228;
                        Position[Z] = (float) -70.453003;
                        WTviewpoint_setposition(GetViewpointByName("Viewpoint-
2"),Position);
                }

                if(rt) WTmessage("rthumb ");
                if(lp) WTmessage("lpinkie ");
                if(lr) WTmessage("lring ");
                if(lm) WTmessage("lmiddle ");
                if(li) WTmessage("lindex ");
                if(lt) WTmessage("lthumb ");
        //      WTmessage("\n");
        }

}

// To get the viewpoint by the name

WTviewpoint *GetViewpointByName(char *ViewpointName)
{
        WTviewpoint *Viewpoint = NULL;
        WTviewpoint *Return = NULL;
        char *Name = NULL;

        Viewpoint = WTuniverse_getviewpoints();
        while ((Viewpoint != NULL)&&(Return == NULL)) {
                Name = (char*) WTviewpoint_getname(Viewpoint);
                if (Name != NULL)
                {
```

```
                        if (strcmp(Name,ViewpointName) == 0)
                        {
                                Return = Viewpoint;
                        }
                        else
                        {
                                Viewpoint = WTviewpoint_next(Viewpoint);
                        }
                }
                else
                {
                        Viewpoint = WTviewpoint_next(Viewpoint);
                }
        }
        return 0;
}

// To write text on the screen
void ForegroundActionFunction(WTwindow *win,int eye)
{
        WTwindow *Window = WTuniverse_getwindows();
        if (Window == NULL)
                printf("Window is NULL");
        WTwindow_set2Dfont(Window,0);
        //  This is the foreground action function for the primary rendering window.  This
is used to
        //  print the help menu as well as any other necessary information to the rendering
window.
        ShowHelp(Window);
}
```

```
//Help menu and the tracker data displayed.
void ShowHelp(WTwindow *Window)
{
//  If the showHelp flag is set to TRUE, a help menu is placed in the top-left of the primary
//  rendering window by this function.
        char *buffer00,*buffer01,*buffer02;
        char *buffer11,*buffer12,*buffer10;
        char *buffer21,*buffer20,*buffer22;
        buffer00 = new char [20];
        buffer01 = new char [20];
        buffer02 = new char [20];
        buffer10 = new char [20];
        buffer11 = new char [20];
        buffer12 = new char [20];
        buffer20 = new char [20];
        buffer21 = new char [20];
        buffer22 = new char [20];
        WTwindow_set2Dcolor(Window,0,255,0);
        WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.90,"q - Quit Simulation");
        WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.87,"f - Switch ON/OFF tracker data");
        WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.84,"Pinch - Thumb and Index for Inside view");
        WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.81,"Pinch - Thumb and Middle Finger for Top View");
        WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.78,"Pinch - Thumb and Ring Finger Initial View");
```

```
        WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.75,"Pinch - Thumb and
Little Finger to Reset Views");

        // Tracker data is displayed depending on the switch (f key)
        if (Showbird==TRUE)
        {
                gcvt( pos0[0], 4, buffer00 );
                gcvt( pos0[1], 4, buffer01 );
                gcvt( pos0[2], 4, buffer02 );
                WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.72,buffer00);
                WTwindow_draw2Dtext(Window,(float) 0.1,(float) 0.72,buffer01);
                WTwindow_draw2Dtext(Window,(float) 0.18,(float) 0.72,buffer02);
                gcvt( pos1[0], 4, buffer10 );
                gcvt( pos1[1], 4, buffer11 );
                gcvt( pos1[2], 4, buffer12 );
                WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.69,buffer10);
                WTwindow_draw2Dtext(Window,(float) 0.1,(float) 0.69,buffer11);
                WTwindow_draw2Dtext(Window,(float) 0.18,(float) 0.69,buffer12);
                gcvt( pos2[0], 4, buffer20 );
                gcvt( pos2[1], 4, buffer21 );
                gcvt( pos2[2], 4, buffer22 );
                WTwindow_draw2Dtext(Window,(float) 0.02,(float) 0.66,buffer20);
                WTwindow_draw2Dtext(Window,(float) 0.1,(float) 0.66,buffer21);
                WTwindow_draw2Dtext(Window,(float) 0.18,(float) 0.66,buffer22);
        }
}
```

Vita

Neena Nambiar

Neena Nambiar was born in Pune, Maharashtra, India. She obtained a degree of Bachelor of Engineering in Electronics and Telecommunications from Pune University, India in August 2001. She obtained a Master of Science degree in Electrical Engineering from the University of Tennessee, Knoxville in December 2004.