System to Test Sensitivity to Head Tracking

Error in Virtual Environments

by

Nathan R. Shnidman

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 28, 1996

Copyright 1996 Nathan R. Shnidman. All rights reserved.

The author hereby grants M.I.T. permission to reproduce
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.
System to Test Sensitivity to Head Tracking Error in Virtual Environments

by

Nathan R. Shnidman

Submitted to the
Department of Electrical Engineering and Computer Science

May 28, 1996

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The design and implementation of a system to test human sensitivity to head tracking error in virtual environments is discussed. Head tracking error corresponds to inaccuracies in the head tracking methods used. A superior head tracking device with minimal intrinsic error is used as the control for these experiments. Error is then induced in software to simulate common types of head tracking error. A psychophysics model is used to create tests to determine the threshold point of human sensitivity to each type of error. As part of this model an adaptive testing method is used in conjunction with a two-alternative forced-choice test. Experiments are also conducted in situations both with and without cognitive loading. A graphical user interface (GUI) has been created to handle all interaction during the threshold experiments.

Thesis Supervisor: Nathaniel Durlach
Title: Senior Research Scientist, Research Laboratory of Electronics
Acknowledgments

I would like to thank the following (in no particular order):
My parents and family
  Eric Foxlin
Dr. Nathaniel Durlach
  Anne Hunter
  Jeffery Wong
  Matt Antone
  Scott Smith
Prof. Erich Ippen
Christopher Richardson
Barbara Shin-Cunningham
  Nora Luongo
  M.I.T.
Dive Laboratories
Intel Architecture Labs
# TABLE OF CONTENTS

1. INTRODUCTION ....................................................................................................................... 6
   1.1 USES OF HEAD TRACKING AND VIRTUAL ENVIRONMENTS ........................................... 7
   1.2 REASONS FOR DETERMINING THRESHOLD ................................................................. 8
   1.3 Threshold Testing System.............................................................................................. 9

2. EXPERIMENT BACKGROUND .................................................................................................. 10
   2.1 ERROR TYPES ............................................................................................................... 10
      2.1.1 Jitter ......................................................................................................................... 11
      2.1.2 Drift ......................................................................................................................... 11
      2.1.3 Non-Linear Mappings and Cross Axis Coupling ...................................................... 12
   2.2 COGNITIVE LOADING .................................................................................................. 12
      2.2.1 Static Test ............................................................................................................... 12
      2.2.2 Dynamic Test ......................................................................................................... 13
   2.3 TRIAL TYPE AND TESTING METHODOLOGY ......................................................... 13
      2.3.1 Two-Alternative Forced-Choice Test ...................................................................... 14
      2.3.2 Adaptive Method .................................................................................................. 15

3. EXPERIMENT PROCEDURES ................................................................................................. 16
   3.1 TESTING PROCEDURE .................................................................................................. 16
   3.2 SOFTWARE INTERFACE ............................................................................................... 19

4. SYSTEM IMPLEMENTATION .................................................................................................. 22
   4.1 HARDWARE ..................................................................................................................... 22
      4.1.1 Head Tracking Systems ......................................................................................... 22
         4.1.1.1 Polhemus Fastrak ............................................................................................ 23
         4.1.1.2 Angularis ......................................................................................................... 24
      4.1.2 Head Mounted Displays ....................................................................................... 25
         4.1.2.1 Virtual iO i-glasses! .......................................................................................... 26
         4.1.2.2 M.I.T. Research Display .................................................................................. 26
      4.1.3 Processor ................................................................................................................ 27
      4.1.4 3D Graphics Accelerator ....................................................................................... 27
   4.2 SOFTWARE ........................................................................................................................ 28
      4.2.1 Operating System ................................................................................................... 28
      4.2.2 Programming Language ......................................................................................... 28
      4.2.3 Virtual Environment Libraries ............................................................................... 30
         4.2.3.1 OF .................................................................................................................... 30
         4.2.3.2 Amber ............................................................................................................... 30
         4.2.3.3 Others ................................................................................................................ 30

5. SOFTWARE IMPLEMENTATION ............................................................................................. 32
   5.1 STRUCTURE AND FUNCTION OF THE SOFTWARE ..................................................... 32
      5.1.1 Document Section .................................................................................................. 32
      5.1.2 View Section ......................................................................................................... 33
      5.1.3 Device Section ....................................................................................................... 34

6. FUTURE ISSUES ....................................................................................................................... 36
   6.1 NON-LINEAR MAPPINGS AND CROSS-AXIS COUPLING ERROR ......................... 36
   6.2 IMPROVED 3D ACCELERATORS .................................................................................... 37
   6.3 OpenGL ............................................................................................................................ 37
   6.4 DIRECTDRAW AND DIRECT3D ................................................................................... 38
1. Introduction

Virtual reality is a rapidly growing field with a large range of possible applications. Virtual reality interfaces have been suggested for uses including: entertainment, education and training, medicine, visualization, networking, telepresence, and business applications [1]. Use of virtual reality in these cases has the potential to dramatically improve the computer-user interface.

Vital to many of these types of virtual reality interfaces are head tracking systems (HTS). An HTS consists of the tracking sensor itself plus whatever data formatting and control circuitry are necessary. This usually refers to whatever tracking equipment and software is external to the system computer.

In many systems an HTS is used with a head mounted display (HMD). The HTS monitors the user’s head position and orientation. This information is then transmitted to the system computer, which makes appropriate adjustments to the virtual environment, and then this updated virtual environment is displayed through the HMD (see Figure 1.1). Thus if the user’s head moves up and to the left, the scene viewed within the HMD will move in a corresponding manner. The HTS closes the feedback loop at the point between the user and the computer generated virtual environment which is displayed in the HMD.
1.1 Uses of head tracking and virtual environments

There are many instances in which an HTS coupled with an HMD can be used to improve task performance. For instance, a semitransparent HMD can be used to overlay synthetic imagery on objects in the environment [2]. In medical procedures this could be used for medical imaging. "The virtual image of the internal organs and the real body of the patient will be merged" [3]. An accurate HTS is then of extreme importance so that the location of the images are properly aligned within the subject's view. When the user's head turns, the generated images must adapt so that a virtual image of an organ does not move with respect to its position on the body.

Another example is in entertainment systems. HTS's and HMD's are often employed in gaming systems to allow the user the illusion of presence within the game.
itself. Exact positioning is not vital in many game situations, so positioning error is not as important. An accurate HTS is still vital, however, as errors other than positioning errors can have an undesirable effect on the system. If, for example, the virtual environment in which the game was set began to rotate slowly, over time the effect of this error could be very disconcerting or distracting.

1.2 Reasons for determining threshold

Multiple types of error degrade the accuracy of the HTS (see section 2.1). These errors cause the user’s illusion of presence within the virtual environment to begin to break down. In many virtual reality applications error within tracking systems is undesirable if not unacceptable.

While it may be possible to remove a large amount of error in an HTS, such an undertaking may be overkill. The reduction of error may incur larger expenses and system performance may be adversely affected by error reducing mechanisms or algorithms. As such, it is advantageous to determine the minimum amount of error reduction necessary and not waste resources attempting to improve the system beyond that point.

The effectiveness of a virtual reality is dependent on the subject’s perception. The absolute amount of error is not as important as the perceived amount [4]. Thus determining the point below which humans can not perceive various types of error, called the perceptual threshold point, is vital to building an optimal system. This threshold point then determines the minimum amount of error reduction needed.
1.3 Threshold Testing System

The system to be built is outlined in Figure 1.1. The system consists of a computer, an HMD, an HTS, and the user. The user’s movement is tracked by the HTS. This tracking information is then sent to the computer which generates a virtual scene from it. The computer also runs the threshold tests and injects error into the scene for testing purposes. This scene is then displayed on the HMD. The user can also issue commands to the computer. These commands range from performance information to commands to be issued to the HTS by the computer (see section 3.2). In addition, the user inputs testing information directly to the computer.

This system to determine a subject’s threshold for various types of error is outlined in the following chapters. In Chapter 2 the types of error tested and possible testing conditions are discussed. In addition, the testing methodology employed is addressed. Chapter 3 details the procedures for administering the threshold test. The various pieces of the system are identified and discussed in Chapter 4. The implementation of the testing software is outlined in Chapter 5. Chapter 6 deals with future issues and possible improvements to the system.

While the system has been completed, subject testing with the system has not begun. The results of tests with the system will be analyzed and discussed in a future publication.
2. Experiment Background

In order to create the tests used in this system three major testing components had to be determined. The first of these was which types of error were to be tested. The second was what method of testing would be used. The third was in what manner would each test be utilized in determining the subject’s error threshold for each type of error.

2.1 Error Types

The types of induced error which are to be tested are: Jitter, Drift, Non-Linear Mappings, and Cross Axis Coupling. These errors fall into two distinct categories: temporal and spatial error.

Temporal error is error which changes or accumulates over time, regardless of the movement of the subject. Temporal error is caused by inaccuracies in the sensing technology. Jitter and Drift are both temporal types of error.

Spatial error is error which can only be detected when the subject changes spatial location or orientation. It is distortion caused by error in the mapping between the subject’s actual head position and the reported position of the HTS. Non-Linear Mappings and Cross Axis Coupling are kinds of spatial error. The current version of this software implements only the temporal errors, drift and jitter. Tests for spatial types of error will be added at a later date (see section 6.1).

It is necessary to create distinct testing situations for temporal versus spatial error. This is due to the fact that spatial error cannot be detected without subject movement and such movement renders small amounts of temporal error imperceptible.
2.1.1 Jitter

Jitter is movement in the output display caused by noise in the sensor. Jitter appears in the display as a high frequency shaking of the scene. In order to properly model jitter an empirical sampling of the jitter error in an HTS was taken. The Polhemus sensor was used for this purpose. The sensor was placed 24 inches from the source and boresighted (the current orientation set as default, see section 3.2) and the output of the stationary sensor was then recorded for approximately 90 seconds. Periodic averages of the output revealed a mean which did not vary over time. Thus drift error was deemed negligible in the data collected and that the error observed was due solely to jitter.

The Fourier transform of the collected data was then taken to determine the frequency response of data. It was determined that the frequency response of the noise was white (see Appendix A). A histogram of the data revealed the distribution of the jitter to be Gaussian with a mean of zero (see Appendix A).

2.1.2 Drift

Drift is error resulting from the accumulation of small changes in position over time. The longer the time, the greater the accumulated drift. Drift appears as a rotation of the scene around an axis or axes. It is imperative that the human threshold for drift be determined in a static subject experiment. In an dynamic experiment slow drift is almost imperceptible.
2.1.3 Non-Linear Mappings and Cross Axis Coupling

Cross Axis Coupling is erroneous change along some axis due to real motion in another. An example of Cross Axis Coupling would be if real motion in the yaw direction has any effect on the pitch or roll measurements or display.

Non-Linear Mappings refer to situations in which movement in the virtual environment space does not correspond linearly to movement in the real world.

2.2 Cognitive Loading

It has been suggested that the amount of cognitive loading which the subject experiences could have an affect on that subject’s error threshold. That when one is actively engaged in a task one’s likelihood of perceiving error is lessened, as one is not concentrating on cues which would make the error apparent [5].

2.2.1 Static Test

The purpose of static testing is to determine a subject’s error threshold level for an error type when no head movement is present. This version of the static test has no cognitive loading task. It is possible a future static test will have a cognitive loading task to test the effect of cognitive loading on the threshold level when no head movement is present. At a future date a cognitive loading task may be added to the static test.

The static test requires no head movement or task performance on the part of the subject. The test involves the subject simply viewing a scene and reporting whether or not error was perceived in each trial. Static tests will be performed for each error type. These will determine the most demanding requirements on HTS performance, probably stricter than necessary for systems to be deployed in real applications.
2.2.2 Dynamic Test

Dynamic testing is used to find a subject's error threshold level with head movement present. This version of the dynamic test has a cognitive loading task in addition to the head movement. In the future, a version of the dynamic test may be added which does not have a cognitive loading task present.

A simple task is introduced to create a cognitive load for the subject. In this case, the task introduced is to try to track a moving object. A cursor is displayed which represents the current viewing direction of the subject. The subject is then asked to follow the moving object with the cursor. The position of the cursor and object are both monitored and compared during the trial. If the position of the cursor and the object differ in any degree of freedom by more than some value, \( \varepsilon \), the subject is reported to have failed the cognitive loading task for that trial.

2.3 Trial Type and Testing Methodology

The testing type used, a two-alternative forced-choice test, as well as the method of varying the error, the adaptive method, are both well documented in psychophysics texts [6,7]. Psychophysics is the field of research concerned with human perception of physical phenomena. Testing has been done utilizing both the test type and error adjustment method in auditory psychophysics. Both methods apply readily to any type of sensory testing, however. Psychophysics literature also provides information on how to analyze the data collected from this testing software.
2.3.1 Two-Alternative Forced-Choice Test

The two-alternative forced-choice test is simply a test in which, after each observation period, the subject is asked to choose between two possible responses. A test, or block, is made up of multiple trials. A trial consists of one exposure to a scene, which may or may not have error induced, for a fixed period of time, and a subject response, which consists of whether error was present or if the trial was error-free. The run will not continue until a choice is made by the subject.

The subject receives feedback at the end of each trial as to whether his/her response was correct. This feedback helps the subject quickly reach his/her threshold point for that block. In the case of the dynamic test, the feedback also includes whether or not the subject has successfully tracked an object. This feedback helps to make sure that the subject is actively engaged in trying to follow the object for each trial. As another incentive to engage the subject in the task, the subject's score is totaled for each block [6]. This score consists of a single point, per trial, for correctly identifying whether or not error was present, and a second point, per trial, for successfully tracking the moving cube. Equal reward is given for each task so that the subject does not favor the more heavily rewarded task.

Another testing procedure which helps subjects quickly achieve the threshold level is previewing. Before the beginning of each block the subject is shown two observation periods. One period has a noticeable amount of error, and the other has no error at all. These previews help the subject to know what to expect and help preclude errors early in the block [6].
2.3.2 Adaptive Method

An adaptive method is a procedure which determines how each trial is modified from the previous trial. In an adaptive method, this procedure is dependent on the subject response for the previous trial [7]. The simple up-down or staircase method was the adaptive method chosen for this testing program. This method works by the experimenter choosing an initial amount of error and a $\Delta$error, or step size. For each correct subject response the amount of induced error is lowered by $\Delta$error, and for each incorrect response it is raised by $2*\Delta$error. Eventually, the threshold point of the subject’s ability to detect the induced error will be reached and the subject will begin to alternate between correct and incorrect responses. The subject’s threshold level can then be determined by averaging the values of the last six to eight of these reversals [7].

The staircase method is very well suited to finding a subject’s threshold, but it does have a disadvantage. There is a difficulty with too large or small step sizes. If the step size is too large the data will not be focused enough and the accuracy of the result will not be sufficient. If it is too small the number of steps which will be needed to reach the threshold value will be prohibitive [7].
3. Experiment Procedures

The testing procedure specifies how the experimenter and the subject should interact with the test suite, what interaction is expected, and where the results of tests will be stored. The software interface denotes the mapping of the keys which have functionality in this program. Key strokes communicate with either the HTS or the graphics generation software. Examples of some testing procedure steps and the results of some key commands can be seen in Appendix B.

3.1 Testing Procedure

The following are the procedures for using the test suite on a subject:

- Select the Run Test option from the Test pull-down menu.
- A dialog box opens in which testing parameters are specified.
- The subject’s name and the block number are entered.
- Select the type of error to be tested.
- A check box sets the block to use dynamic or static testing.
- The base amount of error and Δerror are set by the experimenter.
- Trials last for a number of seconds set by the experimenter.
- The number of trials per block is set by the experimenter.
- The experimenter selects which degrees of freedom will have induced error.
- At this time the experimenter can select the Scene Options button to change the test scene configuration (see below).
- The subject is told to follow the specified cube with his/her head in each trial (In static testing, the cube will be red and will not change position. In dynamic testing, a
moving green cube appears in addition to the red one and the subject is told to follow the green cube.

- A white cursor cube appears when the block begins. In the dynamic test this cursor follows the head movement of the subject and denotes the subject's current head position. In the static test the cursor does not move.
- Two previews are given: one with induced error equal to base amount of error, and one with no error.
- The trials then begin.
- Whether or not error is coupled into each trial is determined randomly with a probability of either happening being 0.5.
- Each trial requires feedback in a dialog box after the trial. The subject selects one of two buttons. One button states that error was present, the other that it was not.
- The program responds with whether or not the answer was correct.
- For each correct answer, regardless of whether or not error was displayed in the trial, the amount of error is lowered by \( \Delta \text{error} \).
- For each incorrect answer, regardless of whether or not error was displayed in the trial, the amount of error is raised by \( 2 \times \Delta \text{error} \).
- The subject is informed as to whether or not their answer was correct.
- In the dynamic trials, the person is told to follow the moving green cube with their head.
  - The green cube moves about the room and the subject will attempt to keep his/her head centered on the moving cube. Head position is denoted by a white
cursor cube in order to provide the subject with a visual representation of
his/her head position.

- Movement is only allowed in orientation, not position.
- The program determines if at any time the orientation of the subject’s head
  relative to the object is greater than some $\epsilon$.
- If the orientation of the subject’s head was at any time determined to be off by
  more than $\epsilon$ in any degree of freedom from the object, a dialog box opens at
  the end of the trial stating that the person has failed that test.
- Otherwise, a dialog box opens at the end of the trial saying that the person has
  succeeded at that test.

- The results and commented parameters of each trial is saved to a text file.
- The title of the file is the subject’s name, the block number, and the type of error being
  tested.
- The next trial then begins.

After each block is finished:

- A dialog box will open stating the score of the subject.
- The score consists of the number of correct response plus, in a dynamic test, the
  number of trials during which the subject successfully tracked the moving cube.

Scene options:

- Selecting the Scene Options button in the Test Setup dialog box will open a Scene
  Setup dialog box.
• The experimenter can then select the wall shading type (flat shaded, wire frame, or no walls).

• The experimenter can also specify the room dimensions in inches. When the dialog box first appears the default dimensions are already present in their respective edit boxes.

General testing procedures:

• The results of each block will be averaged over the last 6-8 reversals.

• Each subject will be run on the same block multiple times (>3).

• The subject will be allowed a break between each block to help avoid fatigue and boredom.

3.2 Software Interface

The HTS communicates data to the testing program through a precise data format designed for the Polhemus Fastrak. The HTS sends a packet containing 47 bytes of data (see Figure 3.1) of which the first three bytes are the packet start, then six values made up of ASCII characters representing a sign (S), numbers (X), and a space (_). The packet ends with the ASCII control characters for return (\r) and newline (\n). [8] Communications with the HTS’s are via the RS232 serial port, COM 1, running at 38400 baud.
Figure 3.1: Polhemus Data Packet Format

In addition to the testing interface there are numerous HTS and display commands that can be issued directly to the program using key strokes. A list of commands and their results can be seen in Table 3-1.

<table>
<thead>
<tr>
<th>Key Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Boresight the Tracker</td>
</tr>
<tr>
<td>C</td>
<td>Switch to Continuous Output Mode</td>
</tr>
<tr>
<td>c</td>
<td>Switch to Non-Continuous Output Mode</td>
</tr>
<tr>
<td>D</td>
<td>Display Current Tracker Orientation</td>
</tr>
<tr>
<td>P</td>
<td>View the Performance Information of the Current Display</td>
</tr>
<tr>
<td>S</td>
<td>Display the Current View Frustum Information</td>
</tr>
<tr>
<td>^Y</td>
<td>Reset the Tracker State</td>
</tr>
</tbody>
</table>

Table 3-1: Program Key Commands

The Boresight command allows one to designate any orientation of the sensor as the zero orientation. That is, all yaw, pitch, and roll orientation information will be set to
zero for the current orientation and all future orientation measurements will be with respect to that zero orientation.

Continuous and Non-Continuous modes determine whether or not the HTS is in a “polled” mode. The “polled,” or Non-Continuous, mode sends only a single set of position and orientation data for each request for data. In Continuous mode, the HTS sends a new set of data as soon as the previous set has been sent, regardless of whether a data request command has been sent. The testing software defaults to a Non-Continuous mode.

The Display Orientation command opens a dialog box which contains position and orientation information. This command is convenient for diagnostics and precise positioning of the sensor.

The Performance command opens a dialog box showing various system performance parameters. The parameters shown are: number of polygons, frames per second, and polygons per second.

The command to display the current view Frustum Info. opens a dialog box which contains the current view angle as well as the positions of the near and far clipping planes.

The Reset command resets the HTS. This means the HTS is returned to its startup state. The reset command takes about 11 seconds and the system is inoperable for that time.
4. System Implementation

Implementing the system involved mating the HTS’s which we already possessed with a computer system which could accomplish our goals and software which had the versatility we desired, while still retaining usability. The most significant factor which limited our ability to do this was a severely constrained budget. Another important factor, however, was the availability of the hardware and software. Many of the components which were used are newly released.

Newer, better, and cheaper versions of many of the components will be available in the near future. Thus the system has ample room for improvement as new products become available (see section 6). The current system is the integration of the best components available at this time for the lowest cost possible.

4.1 Hardware

The hardware which was employed in this system consisted of four major parts: the head tracking system (HTS), the head mounted display (HMD), the processor, and the 3D graphics accelerator card.

4.1.1 Head Tracking Systems

The Head Tracking Systems (HTS) used varied greatly in the amount of intrinsic error present. Each type of system used a different type of sensing technology. Minimizing the amount of intrinsic error is imperative in order to allow the threshold experiments to be as accurate as possible. Any error injected by the HTS and not by the software is an additional factor which must be taken into account when analyzing the data.
collected from the threshold experiment. This unintentional error can be of a magnitude where the data results could be significantly affected.

4.1.1.1 Polhemus Fastrak

The Polhemus Fastrak is the HTS which was used in the development of the testing software. The Fastrak is one of the most commonly used trackers and its interface is very well supported [2].

The Fastrak uses magnetic fields to sense head movement. A transmitter broadcasts a magnetic field. A sensor then uses this magnetic field as a reference to determine the sensor’s position and orientation. The sensor can be attached to a subject’s head and then reset (called boresighting) such that the orientation and the position are zeroed to the desired starting point. The change in position and orientation of the sensor then reflect the subject’s head movements.

The Fastrak was used to help design the software in this system, but it will not be used for testing purposes. The Fastrak has a significant amount of intrinsic error resulting from the sensing technology it uses. The Fastrak is very accurate when the sensor is within a couple of inches of the transmitter, but, as the sensor is moved farther away, distortions in the magnetic field sensed by the sensor introduce errors.

One of these errors is a rotation in the orientation axes as the sensor moves farther away from the transmitter. This is caused by the bending of the magnetic fields by metal in the environment. The magnetic fields, which are accurate near the transmitter, have a more apparent distortion as one moves away from the transmitter. The sensor still accurately senses the fields, but the fields are increasingly warped as the distance from the
transmitter increases. This type of error results in an effect similar to that of non-linear mapping and cross-axis coupling errors.

Another type of error common to the Fastrak is that caused by the sensing of unintended magnetic fields. This is especially a problem if there are large quantities of metal in the area. The magnetic fields cause eddy currents in the metal which, in turn, create a magnetic field. These superfluous magnetic fields confuse the sensing device and cause error in the sensor output.

While the problem of unintended magnetic fields is often caused by metal close to the sensing area, any type of magnetic interference will cause the Fastrak to malfunction. This becomes extremely apparent when one tries to use the Fastrak in conjunction with an HMD. The magnetic fields created by the tracker circuitry of the Virtual i/O HMD destroy the accuracy of the Fastrak. The inability to use the Fastrak in conjunction with the available HMD make it a non-optimal choice for use in the threshold experiments.

4.1.1.2 Angularis

The Angularis HTS is an inertial head tracker designed and built by Eric Foxlin at M.I.T. The Angularis by itself can only be used to sense orientation information, but it can be used in conjunction with other sensing technologies in order to obtain both orientation and position information.

The Angularis displays negligible amounts of the types of error being tested with this system. The gyros and accelerometers used in the Angularis have very little jitter associated with their output. Drift in the gyros is a potential problem, but by making use of the accelerometers and magnetic compasses, the drift is removed from the sensor
output. The Angularis also does not have the problem of orientation changing with position, as seen in the Fastrak. Thus the Angularis does not have error which could affect the scaling or cross-axis coupling error tests.

Another advantage of the Angularis is its ability to be used with HMD’s. Because the Angularis does not employ magnetic fields as its primary sensing technology the magnetic interference of HMD’s has little effect. The Angularis can be used in conjunction with HMD’s with no visible detriment to sensing ability.

The Angularis has been designed to make use of the same command set as that of the Fastrak. This means that even though the system was designed using the Polhemus Fastrak, the Angularis can be substituted transparently. The lack of error in the Angularis and the fact that it can be used with HMD’s make it the preferred HTS to be used with this system.

4.1.2 Head Mounted Displays

Head Mounted Displays (HMD) are the means by which the virtual environments are shown to the subjects. They usually consist of goggles, equipped with some sort of head attachment, which are relatively light in weight, and which have a small display screen for each eye.

While it is possible, with many types of HMD’s, to send slightly different pictures to each eye in order to create a 3D effect, this system does not have the processing power to do so and still achieve an acceptable frame rate. The same image could be sent to each eye, but this is often confusing. For the experiments with this system the environments will be displayed in only one of the subject’s eyes.
Experiments will be conducted with two different HMD’s. The two HMD’s to be used have very different fields of view (FOV). The FOV may affect the outcome of the threshold experiments, so tests will be run with both HMD’s in an attempt to ascertain whether or not the FOV has a significant effect on the different error thresholds.

4.1.2.1 Virtual iO i-glasses!

The Virtual iO i-glasses! are a commercially available HMD. The i-glasses! come with their own VGA-to-NTSC converter. The view on the i-glasses! have a relatively small FOV of 30 degrees. While the small FOV decreases the immersive feel of the i-glasses!, it does create a sharper image.

4.1.2.2 M.I.T. Research Display

Another available HMD, built by Eric Foxlin in 1991 for research purposes, has a much larger FOV of 100 degrees. This means that while one seems to be able to see more of the scene, one is really seeing just a more spread out and less sharp view of the same image as with the i-glasses!. This display does not have a VGA-to-NTSC converter, but it is possible to make use of Virtual iO i-glasses!’ converter. A 16 pin 2mm connector allows one to easily tap into the converter’s NTSC image (see figure 4.1).
4.1.3 Processor

The processor used in this system was an Intel 100 Megahertz Pentium processor. It was set in a motherboard using the Intel Triton chipset. The system was also equipped with 16 Megabytes of EDO RAM. While this was nominally enough computing power, the system would have run much better with more RAM and a faster processor (see section 6.5).

4.1.4 3D Graphics Accelerator

The graphics accelerator used was the Matrox MGA Millennium PCI with 2 Megabytes of DRAM. The Millennium claims to accelerate 3D Gouraud shading, texture mapping, double buffering, and Z-buffering.

The Matrox Millennium supports both 3DR and OpenGL, so its acceleration features are utilized in the current system and can be relied upon for future upgrades.
The Matrox Millennium did speed up the system, but not as much as expected. In the near future multiple other 3D accelerators are planned to debut from other companies. Some of these might have better performance (see section 6.2).

4.2 Software

Multiple types of software complimented the hardware in creating the testing system. The operating system, programming language, and virtual environment libraries were all necessary and integral parts of making the system work.

4.2.1 Operating System

The operating system used was Microsoft Windows 95. Windows 95 was chosen because it is a 32 bit platform and eminently suited for development with Microsoft Visual C++ 4.0. In addition, the crash protection of Windows 95 made development and testing of programs much easier and quicker. Also the availability, low cost, and ease of use made Windows 95 a good choice. It should be noted, however, that the system would run as well, if not better, on Microsoft Windows NT.

4.2.2 Programming Language

The programming language was determined mainly by the virtual environment libraries. All of the libraries under consideration used C or C++ and all made use of Microsoft Visual C++ (MSVC) and the Microsoft Foundation Classes (MFC). This made Microsoft Visual C++ the obvious choice of development environment. The WIN32 API of MSVC and MFC also made window creation, user interface, and I/O interface possible.
4.2.3 Virtual Environment Libraries

Numerous virtual environment libraries were considered. The main criteria which determined the final library used were cost and ease of use. 3DR was the original choice based on these criteria, but was later replaced by Amber when it became available.

4.2.3.1 3DR

The first version of the testing software was created using the 3DR virtual environment library from Intel Architecture Labs. 3DR provides a fast, low level interface to 3D modeling and is optimized for both the Intel Pentium and Pentium Pro processors.

The 3DR version was fast. With the Z-Buffering turned off (see section 6.6) the throughput reached 30 frames per second. Turning off the Z-Buffering would not be an acceptable solution for a general purpose 3D environment, but did not change the visual output of the system in this case.

3DR has two major problems which precipitated the switch to Amber. The first of these problems is a very intricate and unwieldy interface. The commands for 3DR are very low level. This lack of abstraction allows the user great control over the system output, but also means the user must specify a large number of parameters. 3DR requires a significant amount of time and effort to create simple objects, and more complex or involved scenes or display actions take an unacceptable amount of time to implement.

The second major problem with 3DR is its lack of interaction with the display window. 3DR creates a window in which to display its output and then never communicates with the window again. Changes in the size of the window do not affect the 3DR viewport. Menu and dialog support do not exist within the 3DR program
structure. While support for these features could be created within a 3DR program, doing so would require a significant effort and time allotment.

For these reasons the 3DR version of the testing program was discontinued and the project was moved to Dive Lab’s Amber virtual environment library.

4.2.3.2 Amber

Dive Lab’s Amber version 1.3.1 is the library which was eventually chosen with which to write the testing software. This version of Amber is actually a user interface which makes use of 3DR to implement a very large range of functionality. Future versions of Amber will make use of the OpenGL libraries. Windows 95 will support OpenGL by that time.

Amber provides simple commands for functionality ranging from creating a cube to imposing physical properties upon objects in a scene. The usability and power of Amber provided an extremely convenient environment in which to create the testing software.

Amber also has the advantage of being in C++ as opposed to C. This allowed use of the very convenient object oriented window setup and maintenance functionality of MSVC. It also allowed for convenient abstraction barriers, such using a special class to handle all direct interaction with the sensor.

4.2.3.3 Others

Multiple other virtual environment libraries were considered for use. Among these were Sense8’s WorldToolkit, Microsoft Reality Lab, and Criterion RenderWare. One common factor, cost, eliminated the majority of the possible choices. A large number of
the available virtual environment libraries were prohibitively priced and therefore had to be excluded from consideration.

Some of the affordable libraries were not optimized and were too slow to be considered. Avril 2.0 is an example of such a library.
5. Software Implementation

Microsoft Visual C++ version 4.0 (MSVC) was used to create the underpinnings of the final version of the software. Microsoft Foundation Classes and the standard Microsoft Windows interface were used to create the test suite and the subject and interviewer interaction with the program. A series of screen shots documenting this interface can be seen in Appendix B.

5.1 Structure and Function of the Software

The structure of the program is based upon the model created by MSVC. This model contains two major components: the Document and the View. In addition to these two sections, a Device section has been added to allow communication and interaction with the head tracking Device as well as running the testing suite.

The file Fastrakl.cpp contains code created by MSVC to setup the interaction between the different code sections. Fastrakl.cpp contains the base class for the program and all of the other sections are classes invoked by this class.

5.1.1 Document Section

The MSVC reference defines the Document as “A document is a unit of data that a user opens... [it] is responsible for storing the data and for loading it and storing it to persistent storage, usually a disk file” [9]. The Document code performs operations necessary to store and keep track of the data for the current file. In the canonical model users make changes to this data via the View section. The Document code then interacts with the View and records the changes made there [9]. The Document implementation
code resides in the *Fastrak1Doc.cpp* file. The testing software does not make use of the Document section. There is no file to be loaded at the beginning of a test and there is no information which the program reads from a data file. All changes to the View are dynamic and only relevant at a particular instant. In addition, the only data file interaction is periodic writing of information to a file specified when a test is run which is handled by the Device section. Amber is used to update the View (see section 4.2.3.2) and interaction with the user is handled explicitly by the Device section of the software. Thus the Document itself is superfluous.

### 5.1.2 View Section

The View section is the code which creates the window itself and handles all of the user interaction with the window and all display changes. Multiple files make up the View section. The two main files of this section are: *MainFrm.cpp* and *Fastrak1View.cpp*. In addition to these files there are various files which create and control the dialog boxes and pull-down menus.

The file *MainFrm.cpp* is responsible for creating the window frame of the main window. This code creates the window and invokes such features as: the window size and position, the pull-down menus, and the toolbar (which has been disabled in this case). This code has an effect only during the creation of the window.

The second part of the View section is the file *Fastrak1View.cpp*. The code in this file handles the contents of the window and all user interaction with the window. The graphical user interface (GUI) interacts with other sections of the program through the code in this file. Calls to create dialog boxes and the routing of the information returned
from these boxes is part of this code. The calls to Amber to create the universe and its original population appear in this View code. The call to the Device section to create the connection to the HTS is part of this code. Key stroke interaction with the display is trapped by the code in this file and the appropriate commands are then dispatched to the Device section (see section 5.1.3).

There are two dialog files which create the Test Setup Dialog Box, and the Scene Setup Dialog Box. These files are TestSetupDlg.cpp and SceneSetupDlg.cpp. TestSetupDlg.cpp opens a dialog which allows the user to specify such variables as: subject’s name, type of test, type of error, block number, number of trials per block, and others (see Section 3.1 or Appendix B for a full list). One of the options in the Test Setup Dialog Box is to open the Scene Setup Dialog Box. This box allows one to set the room dimensions as well as the shading model to be used to draw the walls (see section 3.1 or Appendix B).

5.1.3 Device Section

The Device section of the code is implemented in the file Device.cpp. This file creates the device Class which initiates, monitors, and controls the interaction of the HTS with the program and with Amber. The errors being tested are induced in the device Class. This class receives the positioning information from the HTS and then relays those values to Amber so that Amber can update the display. Most of the types of error can be induced simply by appropriately adjusting the values before they are passed on to Amber. In addition, the device Class also contains the code which runs the test suite and which
records the test results. All commands to the HTS are actually issued to the device Class which then issues them to the HTS.
6. Future Issues

There are numerous ways in which the system can be improved. Implementing spatial error types, such as Non-Linear Mappings and Cross-Axis Coupling, could increase the breadth of the system and allow testing information on both spatial and temporal error types.

Improved technology could improve the speed of the system. If the system were faster, the test's accuracy may improve. Also, more complex and intricate scenes and tasks could also be employed. In addition, if the system were fast enough, threshold tests could also be performed on latency error.

6.1 Non-Linear Mappings and Cross-Axis Coupling Error

The ground work for implementing Non-Linear Mappings and Cross-Axis Coupling errors has already been done. There are links to both error types in the interface already. Rudimentary functionality to induce the errors has been created, but is not accessible in the current version of the software. Activating the errors themselves would take little effort.

The main impediment to implementing these errors is the dynamic, cognitive loading, test. The current dynamic test would not be usable with these error types. The current test does not involve enough head movement, and the spatial errors are not detectable without head movement. In order to allow full testing with these errors a new dynamic test would need to be created and implemented.
6.2 Improved 3D Accelerators

Numerous companies are releasing new 3D graphic accelerators in the near future. Accelerator cards such as the 3DBlaster by Creative Labs, the 3D Xpression by ATI Technologies, and the Stealth 3D by Diamond Multimedia could provide significant speed increases in the system. Unfortunately, none of these cards were available at the time of the system implementation.

Performance comparisons of the different cards were not available, but preliminary benchmarks imply that these cards will provide significant acceleration. Therefore it is entirely possible that many of these soon-to-be-available cards may significantly outperform the Matrox MGA Millennium card and increase the speed of the system.

6.3 OpenGL

OpenGL is a graphics library created by Silicon Graphics. It is soon to become a standard part of the Windows 95 operating system. This change should not significantly affect the system, however. There is currently an Amber version which works in conjunction with OpenGL instead of 3DR. Switching to OpenGL will simply require a switch to the new version of Amber.

Switching to OpenGL should help improve system performance. OpenGL takes advantage of both the DirectDraw and Direct3D API’s (see section 6.4). These API’s, working in conjunction with OpenGL, should result in a system performance increase.
6.4 DirectDraw and Direct3D

DirectDraw and Direct3D are addendums to Windows 95 which should be available in the near future. Both DirectDraw and Direct3D allow programs more direct access to the graphics hardware. DirectDraw is an API which allows faster display and 2D manipulation. The Direct3D API is supposed to increase the speed of many 3D operations such as shading, dithering, and texture mapping. This direct access should allow significant increases in the display speed. OpenGL supports both DirectDraw and Direct3D, so the system should be able to take advantage of whatever speed increases DirectDraw and Direct3D have to offer.

6.5 Faster Processor

A faster CPU could result in a major increase in system performance. The first version of the system used and Intel 486 DX2 66 processor. When the system was upgraded to a 100 Megahertz Pentium processor the performance of the system increase by more than a factor of 2. A Pentium with a faster clock speed may increase the system performance even more. Use of a Pentium Pro should result in an even larger increase in performance, as that processor is optimized for 32 bit programs and all of the system's software is 32 bit. Upgrading the processor would be a simple and easy way to increase the system performance, but budget constraints prevented upgrading in this iteration of the system.

6.6 Z-Buffering

In the 3DR version of the system, the system performance was increased by at least a factor of two by removing z-buffering. Z-buffering is an operation done to the
scene which makes sure that closer objects block objects which are farther away. Most scenes require z-buffering to make the displayed image comprehensible. The current scenario being used in this system, however, can be created in such a fashion that z-buffering is not necessary. Skipping the z-buffering stage resulted in a large boost in system performance.

Removing the z-buffering stage is only an option if the movement of the objects and subjected are restricted such that objects never change their relative distances from the subject. This strict requirement makes a system without z-buffering of limited use.

Amber does not support an option to turn off the z-buffering stage. Because of this, the current version of the system, which uses Amber, has z-buffering implemented.

6.7 Single Frame Latency

Latency error, or lag, is a common problem in virtual reality systems. Lag results from some part of the system being too slow. The bottleneck could be the display, the sensor, the processor, or a combination of the two. Systems which minimize lag issues have, historically, tended to be expensive and require specialized hardware and software. Because of the extra cost and effort, systems often are not optimized in this fashion and have significant amounts of lag in them.

Lag is such a common problem that finding an acceptable lag threshold would be of significant value. If a level of acceptable lag could be found, then systems would only need to be optimized to that level in order to avoid lag problems. These slower systems would not incur all of the expense necessary to remove lag completely.
Single Frame Latency means that the current HTS information determines the very next scene frame displayed. The system is then running at the fastest possible speed. As soon as an orientation and/or position reading is taken it is used to create the next frame. The amount of time the information takes to be processed by the system and displayed is therefore less than the length of time in between frames.

In order to find a subject’s lag threshold a single frame latency system would need to be achieved. Without single frame latency the system would not have an ‘error free’ reference point. The whole reason for the threshold testing is to find out how much error is allowable such that the system with the error is indistinguishable from an ‘error free’ system. If there is no ‘error free’ system with which to compare then the testing would have no meaning.

The current system is too slow to test for lag. If, in the future, through hardware or software or a combination of the two, the speed of the system can be increased and single frame latency achieved, then lag testing can be performed.

6.8 Improved Display Systems

A limitation of the testing system results from the resolution of the output display. This limitation is imposed by the quantization effect of the pixels in the display. If, in attempting to find the error threshold, the error induced changes the output by less than a pixel width, the displayed output will not change. This is because the display cannot show changes smaller than a pixel width. The only way to compensate for this is to increase the display resolution such that the pixel width is smaller. Unfortunately, current HMD’s have
a very limited resolution. If an HMD with higher resolutions were to become available, the effects of this limitation would be greatly reduced.

6.9 Testing

The system built here has not yet been used in a testing situation. Testing with this system will begin in the near future, however. By the commencement of testing a new version of the Angularis HTS, with even less tracking error than the current system, should be available. This new HTS should allow for more accurate results than the previous HTS’s.
7. Bibliography


Appendix A: Polhemus Noise Data
Yaw Jitter Histogram

Yaw Jitter Fourier Transform

Number of Occurrences

Sensor Output (rad)

Magnitude

Frequency

3 3.05 3.1 3.15

0 50 100 150 200 250 300

0 50 100 150 200 250 300

0 200 400 600 800

0 200 400 600 800
Appendix B: Screen Shots
Figure 7.1: Screen on Start-Up
Figure 7.2: After the D command (see section 3.2)
Figure 7.3: Test Setup Dialog Box
Figure 7.4: Scene Setup Dialog Box
Figure 7.5: A Static Testing Situation
Figure 7.6: A Dynamic Testing Situation