1996

Researching the Growing Technology of Virtual Reality

David Weinandt
College of Saint Benedict/Saint John's University

The growing technology of virtual reality

A THESIS
The Honors Program
College of St. Benedict/St.John's University

In Fulfillment
of the Requirements for Departmental Distinction
and the Degree Bachelor of Arts
in the Department of Computer Science

by
David A. Weinandt
May, 1996

John Miller, Advisor

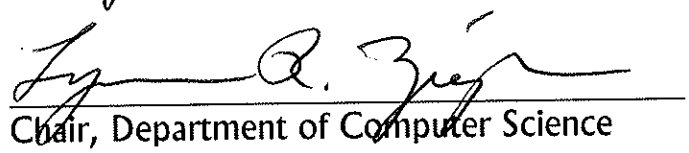
PROJECT TITLE: The growing technology of virtual reality

Approved by:

Dr. John Miller

Assistant Professor of Computer Science

Dr. Lynn Ziegler



Chair, Department of Computer Science

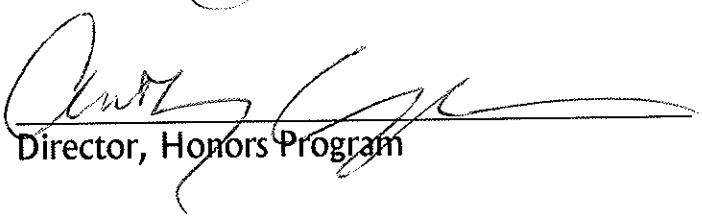
Dr. J. Andrew Holey

Assistant Professor of Computer Science

Margaret Cook

Director, Honors Thesis Program

Anthony Cunningham



Director, Honors Program

1.0 Introduction

Visualize, if you can, walking across the surface of the Moon, feeling the weightlessness, being able to take that "giant step for mankind." As you bend over and touch the dusty surface of the moon, you can feel the soft powder slipping through your fingers. You continue to explore the surface and you come across a crater, you are able to see the enormous size and depth. What a rush this would be for most of us. It is fair to say this seems to be a far-fetched dream for most people, but perhaps not completely out reach. With the amazing advances in today's technology we now have virtual reality. Virtual reality has the ability to give us the sensation of walking on the moon as if we were actually there. It seems to be the gateway to worlds beyond our wildest dreams.

Virtual reality is a rapidly expanding and developing technology that has a "virtually" boundless potential. It has the ability to create the illusion of being immersed in an entirely new world, a world that is completely a computational simulation. "Virtual reality is about discovery, about doing things that couldn't be done before, expressing ideas that couldn't be expressed before." (Pimentel & Teixeira, 7) This is how Eric Gullichsen, one of the first people to create the world's first VR museum exhibits, describes virtual reality. The capabilities of this monumental technology are phenomenal and it is not so far into the future that it

will be in homes all across the world.

Virtual reality is becoming a more commonly known term these days, but that does not mean it is well understood. In this document, I will present a brief history of virtual reality, an explanation of the virtual environment, an overview of the possible user interface tools, an analysis of the interaction between the user and the computer (focusing primarily on the structure of the code), and some applications of virtual reality in our society.

2.0 What is virtual reality?

According to Steve Aukstakalnis and David Blatner, "Virtual reality is a way for humans to visualize, manipulate, and interact with computers and extremely complex data." (Aukstakalnis & Blatner, 7) In a nutshell, virtual reality is the newest way for humans to interface with a computer. We are able to visualize information and interact with this information through the computer. Interaction is established through various interface tools, such as an eye display, headphone, or gloves. Of course the more complex the tool, the more complicated the interface. (Aukstakalnis & Blatner, 7-8)

2.1 Define and Describe

From the beginning computers have been inherently difficult to use. First of

all, they are very literal. They do exactly what they are told to do, nothing more, nothing less. Also, they communicate through a very obscure language which consists of numerous text codes and syntax. This is the reason why the graphical user interface (GUI) was developed. For example, Apple Macintosh and Microsoft Windows are GUI systems. These systems removed the user from having to deal with the text codes and syntax as much. Instead users began using a pen or a mouse to interact with their computer. Thus the term "user friendly" was born. (Aukstakalnis & Blatner 8-9)

With the virtual reality system, users are no longer visualizing data externally, but rather the user is immersed in three-dimensional imagery of the data, in order to perceive it internally. The computer is programmed to create a three-dimensional object or a series of three-dimensional objects in memory. This means that a user, such as an architect, who is submerged in an environment, could interact with this environment, such as a house or room and all of the objects in it, in order to observe the different possible arrangements. He would be able to see fairly accurately how the different combinations would appear without having to build the room first. (Aukstakalnis & Blatner, 9-10)

It would seem that virtual reality technology is the next step in "user friendly" interfaces. As Aukstakalnis and Blatner, authors of "The Silicon Mirage,"

assess the use of virtual reality as a GUI, "The shift of focus from working with alphanumeric data to working with a three-dimensional representation of that data can alter drastically both the manner in which we work with computers and our productivity and enjoyment when working with them." (Aukstakalnis & Blatner, 9) The increase in productivity and enjoyment is also supported by Chris Byrne in his technical report R-93-6 "Virtual Reality and Education." He states, "The most exciting part of the process...was to see all of the students experiencing VR...everyone said that they wanted to come back and try VR again. They all talked about how much fun it was. Clearly, VR was a motivating force for all the kids." (Byrne, 7) Virtual reality is so easy and powerful because it relies on our natural ability to observe and understand. It also aids our retention by stimulating both hemispheres of the brain. (Aukstakalnis & Blatner, 9-10) (Byrne, 1-7)

There are three basic stages of virtual reality: passive, exploratory, and interactive. The passive stage refers to such things as TV, radio, movies, books, etc. In none of these do we have control over the environment, we are merely forced through the environment perhaps hearing, seeing or feeling what is transpiring. In the exploratory stage, we are capable of maneuvering around the environment via flying, walking, crawling, etc., but we are only observers. Finally, in the interactive stage (on which we will be focussing) we are able to experience

the environment by interacting with the objects, perhaps changing the environment, and exploring the environment.

3.0 Immersion into virtual reality

Immersion consists of surrounding ourselves with various objects/stimuli in a way that appears the same as the real world. When we look left, right, up, or down we see objects or differences that we would see in reality. Also, we experience the sensation of moving closer and farther away from objects in the simulation.

3.1 Depth Perception

In order to accomplish this, three main components are needed. First of all, there needs to be a three-dimensional model in the computer and a concept of moving within that environment. Next there must exist various ways for the user to experience the virtual environment. Finally, specialized equipment is needed to enter this "world," and the computer uses this special equipment to update the internal models and display the data according to the input from the special equipment.

The special equipment is oriented to three main senses: seeing, hearing, and feeling. The computer generated modeling is based upon visual imagery. Since our eyes see objects three-dimensionally, the computer must generate three-dimensional

imagery in the virtual environment to fool our eyes. Most of these environments also generated common cues such as shading and perspective. The virtual environment must also grant the ability for patterns to be identified. (Aukstakalnis & Blatner ,25-28)

There has not been as much research on hearing, but it is becoming more important. Computer generated sounds or prerecorded sounds are added to scenes in order to create a more believable atmosphere. For example, if the environment were a wilderness scene, we would expect to hear birds chirping or leaves rustling. Hearing is also important when a sound is made behind the user, that user is able to acknowledge the location of the sound. Another possibility is when the user is coming upon and passing the location of the sounds that they become louder and softer as the user moves closer and farther away from the sounds. Obviously, hearing will help the user understand the world he or she is immersed in. (Aukstakalnis & Blatner, 28)

Finally, feeling allows the user to touch objects in the virtual environment and be able to sense whether objects are hard or soft. Also, feeling can give the user the sensation of flying or walking or however the user could travel in the environment. Feeling is still pretty limited, but it is an important step in developing a believable world. (Aukstakalnis & Blatner, 28-29)

3.2 Position and Orientation

In our computer generated world, it is necessary that the computer knows the position and orientation of the user (especially the hands and head). Which direction the head is pointed and if something is being touched is important, so that through our interaction, the computer can execute a reaction. By knowing where the head and hands are at all times, the computer determines what images to display and keeps the virtual environment stable. The orientation and position of the head and hands are determined by the "six degrees of freedom," the (x,y,z) coordinates and the three rotational functions (roll, pitch, or yaw). (Aukstakalnis & Blatner, 30)

Currently there are five main position-sensing techniques in use. They are mechanical, ultrasonic, magnetic, optical, and image extraction. Ivan Sutherland, "the father of the head-mounted displays," is known for developing the mechanical and ultrasonic methods. The mechanical method uses an armature that has one end connected to the helmet and the other end connected to the encoding device on the ceiling. Changes in the head position cause movements in the arm joints. These movements cause information to be relayed to the computer and the computer then updates the information to the screen as necessary. Although this method gives the most accurate position information, it is also the most limiting

because of its design. (Aukstakalnis & Blatner, 31)

The ultrasonic technique uses inaudible clicking produced by the helmet, and four microphones placed on the ceiling which read the information produced by the helmet. Since sound travels at a constant speed, the clicking sounds reach the microphones at different times depending on the position of the user. The time differential among the four microphones is then computed and the approximate position and orientation are given. This is the concept Nintendo used with its Power glove. This method is very flexible, but it is not precise. Also, obstructions could disrupt the traveling of the clicking sounds, and distance also becomes a factor with this method. (Aukstakalnis & Blatner, 31)

The third method, known as the Polhemus Magnetic Positioning System, uses a magnetic current to estimate the approximate position and orientation. This method was developed by Polhemus Navigation Science of Colchester, Vermont. The concept behind this idea is to send an electrical current through a coiled wire, which will in turn cause a magnetic field along a singular axis. When another coil of wire is exposed to the magnetic field, an electric current is generated in that new coil in proportion to the magnetic field's strength. Therefore, the closer the second coil is the axis, the stronger the field. (Aukstakalnis & Blatner, 32)

With this concept in mind, place three magnetically charged coils

perpendicular to one another, forming the (x,y,z) coordinates system. Use another set of perpendicular coils to form an axis, but do not magnetically charge them. When you move the second set off coils through the magnetic field, it produces a distinct electric charge dependent on its position and orientation. These values can be mathematically computed thereby giving the position (x,y,z) and the orientation (roll, pitch, or yaw). This method is used in loud, cramped areas such as a cockpit of a jet. Such environments are too cramped for the mechanical method to work and too loud for the ultrasonic method to work. (Aukstakalnis & Blatner, 32-33)

The problems with this method are its latency and accuracy. The latency problem is the lag time between the physical movements, and the reception and processing time. The accuracy problem occurs because other large, highly conductive metal objects in the area pick up the electrical current. Also the current is usually base on AC and it fluctuates, which causes eddy currents in other objects. This problem can be solved by either using DC technology instead of AC, or by mapping out secondary magnetic fields and compensate. (Aukstakalnis & Blatner, 33)

The optical technique was originally based on placing a LED on the user and using a mounted video camera and image-processing software. In 1984, Gary

Bishop and Henry Fuchs of the University of North Carolina developed a system opposite the original optical model. The cameras were placed on the user's head and the lights were fixed. Their prototype for this technique consisted of a 10 by 12 foot room composed of two foot square tiles each containing thirty-two infrared LED's that flash off and on in a computer generated pattern. Four small cameras were mounted on the user's head. On the back of each camera is positioned an image sensor called "a lateral-effect photo diode" which reported the location of the ceiling mounted LED. When the LED flashes on, and is within view of at least one of the cameras, the sensor reports the coordinates of that LED to the host system. In order to calculate the position and orientation, four of the sensors must report the photo coordinates of at least three LED's. The limitations of this system are the weight of the head set and the boundaries of the LED's. Carrying a ten-pound head set can get tiring pretty quickly. Also, if a user leans against a wall or moves beyond the LED's, the computer will not be able to accurately compute the position and orientation. However, this method is very sensitive to movement less than .08 inches and angular changes of .2 degrees. (Aukstakalnis & Blatner, 34-36)

The last of the five techniques is the image extraction technique. This technique contains the most intensive calculations, but it is also the most powerful

and easiest to use from a user's point of view. Behind the scenes, one or more video cameras are pointed at the participants. The cameras capture the video image's description of where the user is and what they are doing. Typically this kind of computer visual processing is extremely difficult, but the simplicity it produces for the user is very promising. (Aukstakalnis & Blatner, 36)

4.0 Coding Structure for Virtual Reality

At this point I would like to start getting into the essentials of the programming that goes into a virtual reality application. To do this, I will be focussing on code which executes a tracking device. This generic code is able to track head and/or glove movements. Due to the amount of virtual reality code that would cover the entire spectrum of possibilities that I have explained in the first two sections of my thesis, my focus will be covering the code of this tracking device and how it interacts with the hardware.

4.1 Transformations

The environment, in which the user is immersed, is a 3-dimensional environment rendered the same way any other 3-dimensional environment would be. In other words, there is nothing specifically special about the environment itself, but for reference, I will present a brief summary of the environment. As I

stated earlier, the environment is created 3-dimensionally, meaning that any point within the environment is composed of an x, y, and z coordinates. Depending on the system on which the environment is projected, the window will default the zero coordinates either in the upper or lower left side of the coordinates. The UNIX machines in our labs default the zero coordinates in the upper left corner of the screen, with the positive x-value extending right and the positive y-value extending down the window. It is important to realize that while the world environment is 3-D, the window is discussed as being 2-D. I will cover this point in more detail later in this paper.

The programmer mathematically defines the objects within the environment. For reference, I have included code (written in OpenGL) which generates a doughnut in appendix D. Clearly, rendering a simple doughnut is a lengthy piece of code and as might be expected, most environments are constructed of conglomerations of primitives. Therefore the simpler the environment, the less work the computer has to do. This becomes significant when it comes to moving around an environment. Along with creating primitives, the programmer must also establish a lighting source or a collection of lighting sources, because it helps to establish a more realistic 3-D environment. Once lighting and shading is added to a 3-D environment, a more realistic environment becomes possible because it shows

shadows, illumination, texture, material properties, etc, that make up environments we interact with everyday. This realism is achieved by using different types of lighting, whether it be directional light (such as a spotlight), or a more diffuse light source (such as the sun). Every type of light will achieve a different effect on the objects in the environment, whether the entire face of the object is lit up or merely a small section is lit with the rest shaded. The direction from which the light is being emitted is important to how and where shadows should appear. Another major concern is to make sure the lighting shows the material properties, such as a course or smooth surface. All of these things point to creating a realistic, 3-D environment in which the user is able to use the lighting as well as size to determine depth perception. (Foley & Van Dam, 430-433)

As I mentioned earlier, we have the world environment in which we place objects in a 3-D space and to view it we must project it into a 2-D viewing tool. To make this work, we need to do what is called transformation. Transformations are operations that take a point or a set of points and give it a new point or set of points. These operations occur in either of two ways; they either scale, rotate, or translate a set of points within the world, or take a set of points and transform them into a new coordinate system. (Holloway, Transformations 1)

The first types of transformation take place within the world coordinate

system. The scaling transformation takes an object that has its size defined by a specific matrix and multiplies this current matrix by another matrix which "...stretches, shrinks, or reflects an object along the axes." The rotation transformation, similar to the previous transformation, takes the current matrix and multiplies it by a matrix that will rotate the object about a designated ray from the origin. Translation multiplies the object's matrix by another matrix so that the object is moved to another location. (Davis...etc., 80-82)

The second type of transformation goes hand in hand with translations from the first type. One way of doing this is to construct one object in its respective coordinate system and another object of a different coordinate system and transform the latter system into the initial coordinate system. For example, suppose a programmer has constructed a scene with grass and the sky, and he or she wants to have the sun to rise and set. To do this, the sun needs to be constructed in a unique coordinate system and translated onto the original environment, so that it is capable of rotating through the original environment without rotating the entire original environment. (Halloway, Transformations 1) Another way of using this type of transformation is to project some world space into a 2-D space. In this scenario the world coordinates are mapped into the screen coordinates, so that the user has a way to experience the virtual world through a restricted viewport. A

good way to explain the idea of the viewport in a 3-D world is given, by Foley, van Damm...etc. in their book "Introduction to Computer Graphics." They suggest that we consider the viewport a camera, because we use cameras to "...create a 2-D image of a 3-D object..." We use this camera as our only way to explore the 3-D environment from a variety of orientations and magnifications. (Foley...etc., 177 &193-195)

Suppose our transformation demands us to do more than one operation on the various objects coordinate sets. In such a situation we need to compose the transformations to execute each operation. All transformations for scales, rotates, and translates commute with themselves, except translations with either rotations or scales. These are the possible commuting equations: (S=scale, R=rotate, and T=translate)

S1*S2 = S2*S1 R1*R2= R2*R1

T1*T2 = T2*T1 R*S= S*R.

but rotates and scales are not commutable with translates:

R*T<> T*R T*S<> S*T.

The "general transformation matrix" computes the translate, scale, and rotate operation in one shot. As shown above, we know translations are not commutable with rotates or scales, which causes the translations to always be first in order and

the scales and rotates can be in either order as shown below.

$p' = TRSp$ Or $p' = TSRp$

(Holloway, Transformations 6-7)

As you can see above, there are six transformations (seven if the tracker keeps track of the hand) that occur to have the objects' coordinate system appear in the L/R screen coordinate system. The first is a fixed transformation from each eye's coordinate system to the screen coordinate system. This is necessary for swapping "...from the eyes right-handed coordinate system to the left-handed coordinate system." The next one occurs from the head to the eyes, and in this case there is a different translation for the distance from each eye to the head sensor, and the rotation of the head sensor as measured in the eye system. In this case the head from tracker translation to reverse is actually what gets measured. Thus in the end the tracker first measures the location of the head within the tracker's coordinates system and then sends the inverse for the orientation and positions. The inverse is plugged into the hierarchy so that the correct computations are used. If the tracker includes tracking for a hand, it would come in at this position of the hierarchy as tracker from hand. (Since the hand is not directly involved with the transformation to the eyes, it is actually a branch off the tracker). Here, this branch of the tracker gets transformed at the same time as the

actual tracker is transformed.

Either the application code or the user code controls the coordinate system of the last three transformations. The room to tracker transformation is dependent on which of the three possible origins and two possible orientations the trackerlib puts itself. The first possibility is that the orientation and the origin for trackerlib is placed in the room's origin. Another possibility is to use the tracker's origin and the room's orientation, which boils down to a simple translation. The final possibility is similar to the second possibility in that the tracker's coordinate system is used, however, the room's coordinate system is rotated to the tracker's coordinate system. The world to room transformation is dependent on the user code and whether the user is scaling, tilting, or flying through the virtual world. Our final transformation is the world from object which may be constant for fixed objects or variable for movable objects. (Holloway, Transformations 11-12)

4.2 Basic Format

According to Rich Holloway, there exist three basic parts to a "typical VR application." The first part is to initialize the graphics and the tracker. Next, create an interactive loop that continues to loop until interaction is complete. Within this loop, the application should be able to handle getting tracker data, updating viewing and other transformations (to denote a change in position of the head and/or

hand), handling information from a mouse or data glove, executing "any animation tasks that don't depend on tracker data," and display new frames as needed. Finally, the application must shut down all initialized graphics and the tracker. This format is generic enough to work with any tracking device and it is integrated in the example code found in Appendix A.

Once the Graphics are initialized, the next step is to open the communication to the tracker(s) that is determined by the tracker name list and the index, in the example located in appendix A the list of trackers is represented by the pointer T_ENV_TRACKER, and the index integer, 0. The pointer points to a specific tracker name in the list and the index number specifies which tracker in the list should be opened. Both parameters are needed to ensure the safe usage of more than one tracker. For example if a person wanted to run two Polyhemus head trackers, T_ENV_TRACKER would point to Polyhemus and the index would point to the respective index number. (Holloway, t_open)

The command t_enable opens the head and hand units of the tracker specified by the parameter envTrackerindex and the respective unit determined by the 0. Even though the call t_open has opened the tracker(s), none of the units

have been enabled, this command is called for every unit which is to be used. Once all of the tracker units have been enabled, the coordinate system for the respective tracker must be established. (Holloway, t_enable)

A call to t_frame and its parameters sets the appropriate coordinate system, whether it be T_TRACKER_FRAME, T_ROOM_FRAME, or T_TRACKER_COORDS_ROOM_ORIENT. The parameters consist of pointers denoting which tracker and which one of the possible coordinate system's origins and orientations. The value T_TRACKER_FRAME simply uses the tracker's native coordinate system. The value T_ROOM_FRAME sets the origin to the southwest corner (lower left), with the x-axis extending east, the y-axis extending north, and the z-axis extending up. There are two advantages to this method, first the programmer can model a virtual room which coincides with a real room. The other is that the user code is independent of where the tracker is in the room, thus allowing the tracker's origin can move without the need for recompiling. The final valid value, T_TRACKER_COORDS_ROOM_ORIENT (default value), uses the tracker's origin and the room's orientation. This value is called in order to make all

trackers have the same orientation, while the tracker's origin is world centered. Generally speaking, a frame is simply a coordinate system at a particular point in the real world. The trackers send position and orientation information relative to some frame. (Holloway, t_frame)

The next command listed in the pseudo code is t_data_format, which sets the report data type for specified tracker. This call determines which of the three possible data types will be put into the report list. The report list is returned by the t_read call for reporting position and orientation information form this tracker. The three possible data formats are: T_XYZ_QUAT, T_PPHIGS_MATRIX, and T_XFORM_MATRIX. T_PPHIGS_MATRIX puts the position and orientation data in a 3x4 matrix used by PPHIGS (the Pixel-Planes graphics library). The translation offsets are located in [0] [3], [1] [3] and [2] [3], and the rotation matrix is located in the upper left 3x3 submatrix. Obviously any head mounted display application that are running under PPHIGS will default to this data format. The T_XFORM_MATRIX data format creates a 4x4 matrix in which the rotation submatrix is still in the upper left, but the translation matrix is at [3] [0], [3] [1]

and [3] [2]. Finally, the T_XYZ_QUAT format creates a 4x3 matrix, where the translation is represented by three vectors and a quaternion. (Holloway, t_data_format)

Within this loop, I will be making some assumptions in order to give a simpler, clearer idea of what goes on here. Due to this I am going to momentarily skip it and discuss how the code goes about closing down the units, the trackers, and the graphics. I will then go into detail about the interactive loop.

Once the program exits the interactive loop, it then proceeds to disable the units that have been enabled prior to the loop, with a call to t_disable. Just as the units were enabled, the same is done to disable them, by disabling the units of the specified tracker, which is determined by the parameters envTrackerindex and 0. This command can be called at anytime to disable a specific unit (that is already enabled), so long as it is called before t_close. (Note: closing down excess units increases the update rate.)(Holloway, t_disable)

After all the units have been disabled, the code progresses to close down all of the open trackers and close the graphics. Again these trackers are specified by

the pointer envTrackerIndex, which points into the tracker index at the corresponding tracker. Obviously, this stops any possible communication with the trackers, without a call to t_open. All trackers need to be closed before exiting, in order to reduce the possibility of lingering pointers or any kind of chaotic error. This leads us to the last step in which the code makes a call to close the graphics. (Holloway, t_close)

4.3 Interactive Loop

Within this interactive loop, I will be making a couple of assumptions for simplicity in my explanation. First of all I will assume, that the object hierarchy will be the graphics package PPHIGS. The second assumption is that the vlib, a virtual worlds library, is implemented in this code. PPHIGS is used to model objects in the virtual world using primitives. Vlib is used to encapsulate the common operations within the virtual world applications, this is to simplify these operations for programmers.

Proceeding into the main interactive loop (Appendix B), a number of things may occur. In the pseudo code, the first step in this loop is a call to t_read, which

reads the report(s) from the tracker specified in the parameter. The reportList pointer should point to enough storage for all the enabled trackers' reports. A report is a data structure which contains the position and orientations information for each specific unit, this data is later returned in the form of a report list. (Holloway, t_read)

The next call is to v_update_head_xform and/or v_update_hand_xform depending on the tracking system being used. Each of these commands updates the transformation (tracker from head and tracker from hand respectively) using the most current tracker report. The information from the tracker is put into v_users and then v_replace_xform reflects this update in the PPHIGS display list. (Holloway, v_update_head_xform and v_update_hand_xform)

The interaction phase of the loop remains rather vague because there are many options for interaction, such as flying, grabbing, etc. Some of these calls have quick reference calls held within vlib, however, if some desired action is not in vlib, it is up to the programmer to write the necessary code. In a situation, such as a call to v_grab, the objects world transformation is given to the user's hand space. It is

the application, which decides whether the object is within reach or not. In order to release the object, the v_release command must be called or the following v_grab calls will be unpredictable. (Holloway, v_grab)

After the interaction is activated the v_update _displays is called, this allows the images seen to be updated for each user. Each user has a display index specified in the display index list, and the v_update_displays call uses this as a parameter to return valid displays. It is important that after an interaction within the world occurs, that what each user sees is updated, ackowledging the specified interaction. (Holloway, v_update_displays)

Once the interactive loop has been exited and the tracker has been closed, the display is closed with a call to v_close. In relation to PPHIGS, v_close returns the value from pg_close_pphigs( ). In any case, anything that had been enabled or opened needs to be disabled or closed. (Holloway, v_close)

5.0 How Are Users and Objects Viewed

Objects are graphical structures consisting of primitives, colors, transformations, etc. A user is any object in the virtual world from which a

viewpoint is calculated and displayed. This user has its own hand, eyes, head, ect. so that the users can be in a different virtual world than other users without causing problems. Both the users and the objects contain lists of transformations, pointers, and any other information necessary for each individual user or object as allocated in the virtual world (see Appendix C). So we can see that attributes for users and objects are not stored in a vast database, but rather are generated as needed using pointers and transformations for the necessary information. (Holloway, Vlib 4-12)

6.0 Conclusion

Within this document I have explained what virtual reality is, what it means to be immersed in a virtual reality, what the basic coding structure is, and how users and objects are viewed in the virtual environment. Through research of the tracking devices, I have uncovered some of the complications that virtual reality programmers face. Such complications are efficient tracking in collaboration with more user freedom. Another complication is updating screen displays for real time imaging when multiple users and units are enabled. A less important complication

at this point is registering full body tracking, which will consist of multiple units being enabled. This of course, would increase in complication with the addition of multiple user in one session.

Another advantage to researching virtual reality is that it is certainly the GUI of the future, and we all have seen some application of it. When a pregnant woman goes to the doctor to get an ultrasound, her child is being tracked and display using virtual reality technologies. Children are being introduced to it through experimental education programs, and through video game companies such as Nintendo, who introduced the power glove to the market a few years ago. Even the military uses virtual reality for its pilots, to recreate the terrain ahead of them so that they can make the appropriate turns. Virtual reality has even been used in court to recreate accident scenes, reproducing the conditions, terrain, and any other necessary information, in order to determine the validity of a testimony.

As I stated in my introduction, the potential for virtual reality seems boundless. The future of virtual reality holds numerous possibilities, such as making everyday applications created three-dimensionally. Surfing the network could be

done using a head mounted display and with any possible combination of tracking units. I also expect, since humans are visually oriented creatures, that virtual reality will be commonly used in schools and universities throughout the world. I also believe that it will find its way into numerous medical applications, such as therapy for the physically handicapped, various types of surgery, and possibly even with the mentally handicapped.

Appendix A

This code was written by Rich Holloway of the University of North Carolina, and sent to me in a tar file via e-mail. This code represents the basic function of a simple virtual reality program. It presents the basic calls, but leaves the code generic enough that a programmer unfamiliar with virtual reality coding can easily understand the basic format and be able to change the code appropriately. The basic function of this pseudo code is to make the appropriate open and enabling calls, which will lead into the interactive loop. Within the interactive loop, all updating of displays, interaction, and animation is done. Once the interactive loop is exited, the program proceeds to close and open units and trackers.

user.c

```
#include <stdio.h>
#include "tracker.h"
int
main()
{
        int                 i;
        t_index          envTrackerIndex;
        t_report_type reportList [2];
        t_boolean     done = T_FALSE;

/* initialize graphics*/
initGraphics();
/* open tracker; like a unix file */
If ((envTrackerIndex = t_open(T_ENV_TRACKER, 0))
        ==T_NULL_TRACKER)
```

```
            {
            fprintf(stderr, "Open of env tracker failed.\n");
            return(T_ERROR);
            }
      /*open units for head and hand- if you only have on sensor/receiver, kill the
second
      * call to t_enable() */
      if (t_enable (envTrackerIndex, 0) ==T_ERROR)
            {
            fprintf(stderr, "Open of unit failed.\n");
            return(T_ERROR);
            }
      /* set the coordinate system to the tracker's native coord sys for debugging
use
      */
      t_frame (envTrackerIndex, T_TRACKER_FRAME);
      /* set data format to be 4x4 row transformation matrix */
      t_data_format (envTrackerIndex, T_XFORM_MATRIX);

      /* main interactive loop */
      While (! done)
            {
            /*gives object and eye position/orientation in room space */
            t_read(envTrackerIndex, reportList);
            /* update the viewing transformation*/
            updated (reportList);
            /* handle mouse events, user actions, etc */
            interaction(&done);
            /* do any animation here that's independent of tracker data*/
            animate();
            /* draw new frame*/
            drawFrame();
            }
      /* disable units for head and hand- if you only have one sensor/receiver,
       * kill the second call to t_disable() */
      if (t_disable (envTrackerIndex, 0) == T_ERROR)
            {
            printf ("error closing tracker.\n");
```

```
        return (T_ERROR);
        }
/* shut down tracker */

if (t_closeGraphics);

}
```

(Rich Holloway, ex. Program)

APPENDIX B

This code was written by Rich Holloway with some assistance by Erik Erikson from the University of North Carolina. This code emphasizes on what occurs within the interactive loop of a virtual reality program. This code goes a little more into detail of what happens within the loop, such as the updating of transfomations, updating screen displays, and interaction.

```
#include <v.h>

main ( )

{
        t_report_type       reportList[2];
        t_index             trackerIndex;
        v_index             displayIndex;

/* initialize graphics and trakcer */
init(&displayIndex, &trackerIndex);

/*main interactive loop*/
while (V_true)
        {
        /* read tracker (trackerlib call) to get info on user's head and hand */
        t_read (trackerIndex, reportlist);

        /* update head & hand from tracker xform */
        v_update_head_xform (0, &reportList[0]);
        v_update_hand_xform (0, &reportList[0]);

        /*routine for handling mouse button input */
        interaction( );
```

```
/* draw new image for each eye */
v_update_displays (&displayIndex);
}

/* close tracker */
t_close( );
/* close display, etc */
v_close( );
}       /* main*/
```

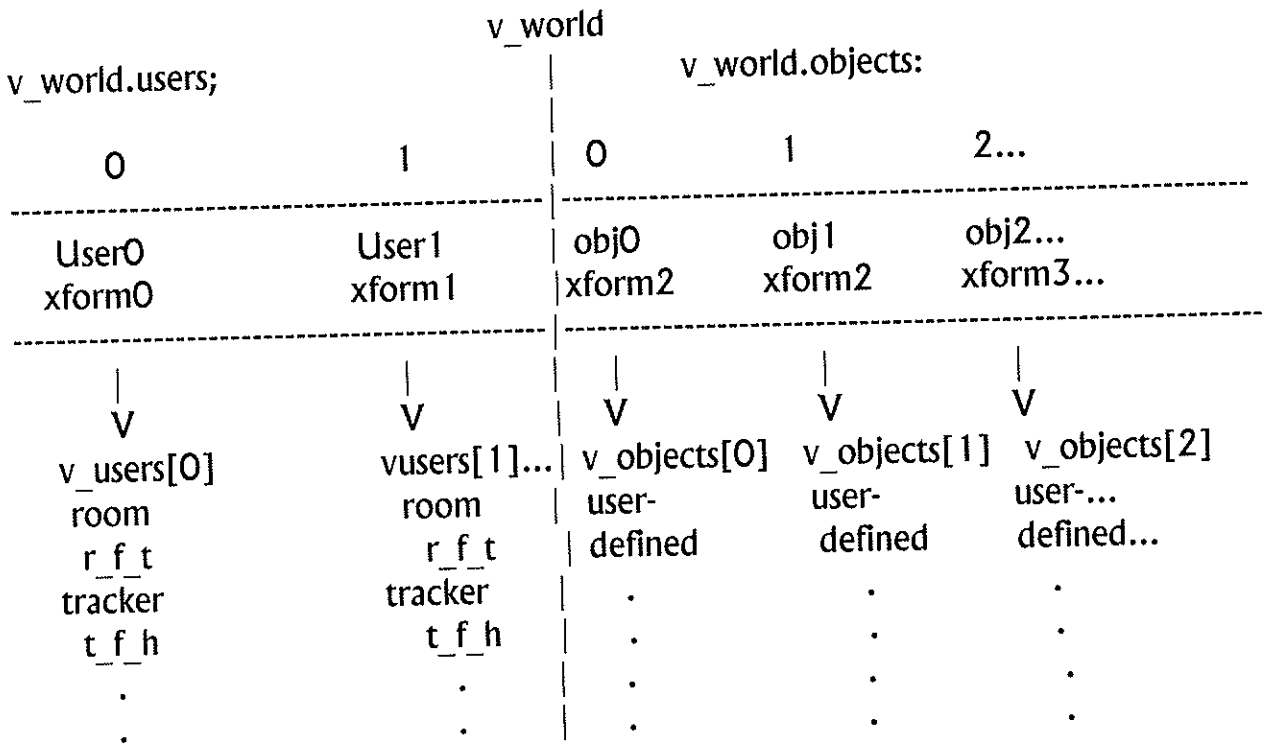
(Holloway, Vlib 2-3)

APPENDIX C

The diagram is a graphical representation created by Rich Holloway of the University of North Carolina. I obtained this representation at the web site:

ftp://ftp.cs.unc.edu/pub/projects/hmd/man/man3/

in the Vlib text file. This is a visual aid to help understand how the computer views users and objects within the virtual world. Basically, a user is an object as well, except the user contains a viewpoint which other objects are not capable of doing.



Appendix D

This code was written by Mark J. Kilgard and is located in Dan Challou's faculty file of St. John's University. The file location is

~dchallou/graphics/glut/primitives.c

This OpenGL code generates a doughnut. It is used to impress the mathematical complexity of the merely generating a primitive in a three-dimensional world.

```
static void
doughnut(GLdouble r, GLdouble R, GLint nsides, GLint rings, GLenum
type)
{
        int i, j;
        GLdouble theta, phi, theta1, phi1;
        GLdouble p0[03], p1[3}, p2[3], p3[3];
        GLdouble n0[03], n1[3}, n2[3], n3[3];

        for (i = 0; i < rings; i++) {
          theta = (GLdouble ) i * 2.0 * M_PI / rings;
          theta = (GLdouble ) (i + 1) *2.0 * M_PI / rings;
          for (j = 0; j< nsides; j++) {
            phi = (GLdouble ) j * 2.0 * M_PI / nsides;
            phi = (GLdouble ) (j + 1) *2.0 * M_PI / nsides;

            p0[0] = cos(theta) * (R + r * cos(phi));
            p0[1] = -sin(theta) * (R + r * cos(phi));
            p0[2] = r * sin(phi);

            p1[0] = cos(theta1) * (R + r * cos(phi));
            p1[1] = -sin(theta1) * (R + r * cos(phi));
            p1[2] = r * sin(phi);

            p2[0] = cos(theta) * (R + r * cos(phi1));
```

```
            p2[1] = -sin(theta) * (R + r * cos(phi1));
            p2[2] = r * sin(phi1);

            n0[0] = cos(theta) * (R + r * cos(phi));
            n0[1] = -sin(theta) * (R + r * cos(phi));
            n0[2] = sin(phi);

            n1[0] = cos(theta1) * (R + r * cos(phi));
            n1[1] = -sin(theta1) * (R + r * cos(phi));
            n1[2] = sin(phi);

            n2[0] = cos(theta) * (R + r * cos(phi1));
            n2[1] = -sin(theta) * (R + r * cos(phi1));
            n2[2] = sin(phi1);

            glBegin(type);
            glNormal3dv(n3);
            glVertex3dv(p3);
            glNormal3dv(n2);
            glVertex3dv(p2);
            glNormal3dv(n1);
            glVertex3dv(p1);
            glNormal3dv(n0);
            glVertex3dv(p0);
            glEnd();
          }
        }
}
```

Bibliography

L. Casey Larijani. "The Virtual Reality Primer" New York:McGraw-Hill, Inc. 1994.

Howard Rheingold. "Virtual Reality" New York: Summit Books, 1991.

Alan Wexelblat ed. "Virtual Reality: Applications and Explorations" Boston: Academic Press Professional, 1993.

Steve Aukstakalnis and David Blatner. "Silicoln Mirage: the Art and Science of Virtual Reality" Berkley: Peachpit Press, Inc., 1992.

Ken Pimentel and Kevin Teixeira. "Virtual Reality: Through the new looking glass" New York: Intel/Windcrest/McGraw-Hill Inc., 1993.

Steven R. Holtzman. "Digital Mantras: The Languages of Abstract and Virtual Worlds" Cambridge: The MIT Press, 1994.

Benjamin Woolley. "Virtual Worlds: A Journey in Hype and Hyperreality" Cambridge: Blackwell, 1992.

Michael Heim. "The Metaphysics of Virtual Reality" New York: Oxford University Press, 1993.

Rich Holloway and Erik Erikson. "Trackerlib" Chapel Hill: University of North Carolina 1993
ftp://ftp.cs.unc.edu/pub/projects/hmd/man/man3/

Rich Holloway, Russ Taylor, and Mark Mine. "Virtual reality code for Tracking" Chapel Hill: University of North Carolina

James Foley, Andries van Dam, Steven K. Feiner, John F. Hughs Richard L. Phillips. "Introduction to Computer Graphics" New York: Addison-Wesley Publishing Company, Inc. 1994.

Jackie Neider, Tom Davis, and Mason Woo. "OpenGL: Programming Guide" New York: Addison-Wesley Publishing Company, Inc. 1993.

Mark J Kilgard. "Primitives.c" 1994 ~dchallou/GRAPHICS/glut/ (located on St. John's University UNIX network.