Wii-mote Head Tracking

A Three Dimensional Virtual Reality Display

By

David Fairman

Senior Project

Computer Engineering Department

California Polytechnic State University

San Luis Obispo

2010

Table of Contents

Section *Page*

List of Tables and Figures

I. Acknowledgements

I would like to thank Computer Engineering Professor John Oliver for being my senior project advisor and guiding me throughout the last two quarters. John actively helped me come up with the base idea for my senior project, the particular contribution I could make to an existing design, and made sure I was on track throughout the project period.

II. Abstract

The goal of this project is to create a customizable three dimensional virtual reality display on a system available to any non-technical user. This system will use the infrared camera component of a standard Nintendo Wii-mote to track a users head motions in the left, right, forward, backwards, up and down directions. The virtual display will be a customizable image projected onto a screen or simply shown on a computer or TV monitor. In order to make the two dimensional image appear three dimensional, the image on the display will continually change according to the position of the user's head. As the user moves their head to the left and right, potions of the image displayed will be shifted to the right and left respectively by different amounts depending on their depth location in the image. Likewise, as the user moves their head closer and further from the display, portions of the image will be increased or decreased in size again depending on their depth location. In this way the image is being continually redrawn to match the user's perspective or vantage point. The user will be able to select a number of images of their choosing and place them in the virtual display assigning them each a position in space based on x, y, z coordinates and relative scale. The system will create a continually augmented display according to the motion and location of the users head to create a three dimensional presentation on a two dimensional screen.

1. Introduction

The goal of this project is to take a three dimensional virtual reality space and make it customizable to the user. This virtual reality space will consist of a two dimensional image that is continually redrawn to rearrange the size and location of individual images depending on the users changing perspective of the scene. The user will be able to take a background image and insert a variety of different images at different locations of the foreground to create their own virtual display. Each image placed in the foreground, will have its own x, y, z coordinates in the three dimensional space and a specific size relative to all other objects contained in the space. Each aspect of the images location and even the images themselves will be determined by the user in a closed loop system so that values may be changed without the need to recompile and restart the system. The three dimensional characteristic of the space will be conveyed through use of head tracking with a Wii-mote. The user will wear a specially designed hat or set of glasses with infrared LEDs embedded on the front facing towards the screen. A Wii-mote will be placed under or above the screen facing the user so that the Wii-motes infrared camera can determine location and movements of the LEDs. As the user moves their head and the LEDs change location, the information is sent to the computer via a Bluetooth connection. The computer then parses the data from the Wii-mote to determine the changing orientation of the users head. The system uses the head tracking information to perform an appropriate matrix transformation on the three dimensional virtual space. As the user moves their head to the left, the images are moved to the right by different amounts based on their specified depth

in the space. If the user moves closer or farther away from the screen, the images will be scaled larger or smaller, again by amounts determined by their depth in the space, to simulate a three dimensional environment. A communication block diagram for the overall system is shown below in **Figure 1.**

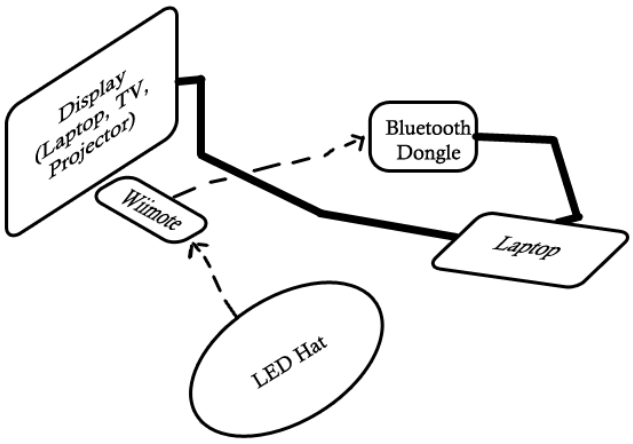

Figure 1 - Overall System Diagram

In this report I will discuss the background of the project and why it was deemed important enough to invest time into, the specific requirements of the end product, my design, construction and testing of the system, and finally what I gained from working on this project and my recommendations for future development of the system.

2. Background

This project is a continuation of an idea previously designed by Johnny Chung Lee. Lee designed the virtual matrix and its interaction with a Wii-mote to create the three dimensional space that this project is loaded into. Originally, his three dimensional space could only be represented with a black background, a coordinate mesh, and number of targets randomly set at different locations. This provided for an interesting display, but did not have much use to the end user. A few three dimensional looking targets can only be appreciated for so long.

However, Lee's project did have a certain "wow factor" that provided the inspiration to take his idea to the next level. A three dimensional display that does not require red and blue shaded glasses or special image alterations has a huge potential to increase user experience whether in gaming, television and movies, or simply just a display of a scene. With Lee's virtual representation of a three dimensional space, I took the opportunity to turn the program into a customizable scene generating system allowing users to create their own three dimensional environment.

Lee's original design also made use of a pair of safety goggles with infrared LEDs embedded into the edges in order for the user's head to be tracked by the Wii-mote. In an attempt to make the system less distracting, I have instead embedded four LEDs into the brim of a baseball cap. Even though the safety glasses were fairly low profile, the frames accentuated

the virtual nature of the system. By using a hat instead, there is nothing in front of the users face or in their field of view, making the virtual reality seem even more realistic.

3. Requirements

The redesign of this virtual reality system had certain requirements that needed to be upheld. First, the system needed to be closed loop, allowing the user to change images and their locations in the space without the need to recompile the program. In the original system, images could only be changed by replacing the root image of the targets and recompiling the program. This did not allow for easy image changes by the user. The system also had to allow for different images to be used as foreground objects. In the original open loop system, changing the root image of the targets would change all images in the foreground and there was no way to have multiple images displayed.

Another requirement of the system was to allow for user determination of image locations, that is, allow the user to change the x, y, z coordinate locations of the individual images as well as their relative size within the environment. Originally, all the targets were set at a specific size relative to each other and the background and their location was determined by a specific depth set for each target at a randomly generated x, y coordinate. The user can now enter all this information into a graphical user interface and immediately see the effects of image location in their three dimensional environment.

Additionally this system required the ability to specify the number of images contained in the three dimensional space. The original target based space had a set number of 10 targets that could be randomly regenerated at different x, y coordinate locations but targets could not be hidden and no additional targets could be created. To make the environment

completely customizable, there had to be a way for the user to determine their own number of images within the three dimensional space.

Finally the system had to be as unobtrusive to the user experience as possible. In order to make the virtual reality feel more realistic, the system needed to be comfortable for the user and easy to use. Distractions needed to be limited in the viewing of the scene and generation or alterations of the environment needed to be a smooth, simplistic process. These requirements are summarized in **Figure 2** below:

Requirement	How is Requirement Met
Closed Loop System	User can change images and locations without needing to recompile
User configurable images	User can change what each image looks like, its x, y coordinate, depth location and size
User configurable amount of images	User can choose anywhere from zero to six images to be displayed in the virtual environment (not including the background)
System must be as unobtrusive as possible	Used LED hat instead of glasses so that user does not feel as if they are part of a system

Figure 2 - System Requirements

4. Design

Although this project was mostly programming intensive, it involved a mixture of both software and hardware design problems. In this section I will discuss the design of the infrared LED hat used to track the users head and the design of my code to make the system a closed loop with customizable features.

4.1 LED Hat Design

The Wii-mote has a 128x96 pixel infrared camera capable of collecting information from up to four separate infrared blobs. When using the Wii gaming system, an LED sensor bar is placed in front of the users display and the Wii-mote calculates its movements based on the changing readings from the LEDs. For head tracking, the setup is used in the opposite direction, with the Wii-mote stationary in front of the display and the LEDs moving with the users head. In order for the Wii-mote to track the movements, we must provide a way to attach at least two infrared LEDs to the user. Lee's original setup used two LEDs wired to a pair of safety glasses, one on each corner of the frame. This provided two reference points for the Wii-mote camera to use to follow the users head movements. For my design, I decided to wire the LEDs into the brim of a hat instead, thinking that simply wearing a hat would feel more natural to the user than having to wear a pair of modified glasses. By using a hat, I was also able to incorporate four LEDs into the headpiece instead of only two as used in the glasses. This should provide better tracking with the Wii-mote as it is closer to the actual workings of the Wii sensor bar which also has four infrared LEDs. Since infrared lights are not visible to the naked human eye, I decided to include a single red colored LED

so that they user is able to tell whether the LEDs are actually receiving power. This LED serves as a test to determine if the switch is on or off, or if the batteries in the hat are dead.

With this design in mind, I created the circuit diagram shown below in **Figure 3**.

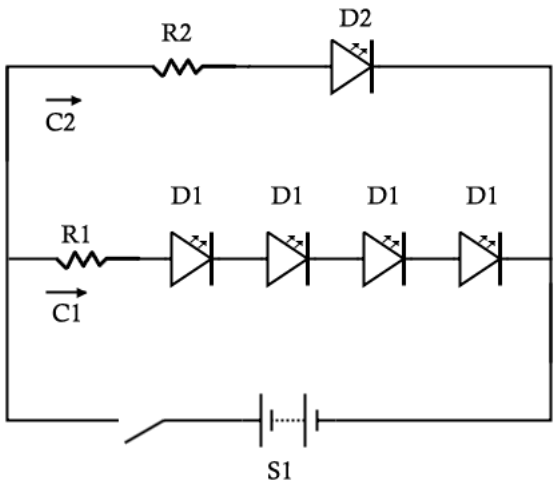

Figure 3 – Circuit Diagram for infrared LED hat

The infrared LEDs were rated at needing 100mA of current and would use 1.28 volts a piece, so in order to have four LEDs I would need at least a six volt power supply. This was obtained by using four AA batteries. The red LED was rated at needing 28 mA of current and would use 2.3 volts. With this information, I was able to calculate the size of the resistors needed, R1 and R2 above, using basic circuit theory. The resistor calculations are shown below in **Figure 4**.

$$4 * 1.28v = 5.12v, \quad S1 > 5.12 \; therefore \; S1 = 6v$$

$$R1 = \frac{6v - 4 * 1.28v}{.1\ A} = 8.8\ ohms$$

$$R2 = \frac{6v - 2.3v}{.028\ A} = 132.14\ ohms$$

Figure 4 - Desired Resistor Value Calculations

Resistors with values as close as possible to the calculated values were chosen: R1 = 10 ohms and R2 = 150 ohms. Using these values for resistors R1 and R2 would yield currents of C1 = 24.6 mA and C2 = 88 mA which is close enough to the desired currents for the system to work. Calculations for these currents are shown below in **Figure 5**.

$$C1 = \frac{6v - 2.3v}{150\ ohms} = 24.6\ mA$$

$$C2 = \frac{6v - 4 * 1.28v}{10\ ohms} = 88\ mA$$

Figure 5 - Current Values from Chosen Resistors

4.2 Code Design

There were two major design decisions that needed to be made with respect to the actual coding of the project: making a graphical user interface and incorporating it into the existing code to close the loop.

I had originally planned on using NetBeans to create the GUI for this project. Johnny Lee's code was written in C# which I did not have much experience in and I have heard that NetBeans is very useful for writing java based GUIs. However, I came to realize it would be much harder to integrate the GUI with the rest of the program if it was written in a different language. After some research I found that Microsoft Visual Studio has a helpful drag and drop GUI creating feature and can be used to create interfaces in C#. After reading through a few tutorials I was able to make the graphical user interface shown below in **Figure 6.**

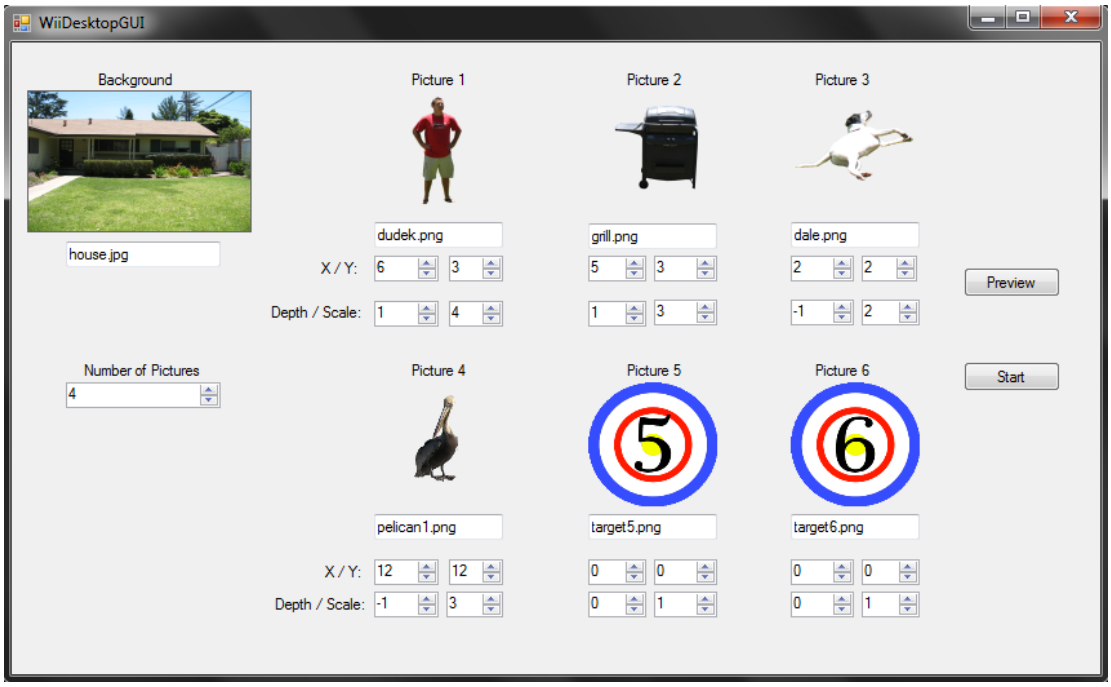

Figure 6 - GUI for Virtual Display

As seen in the figure above, there are a number of text and numerical fields that must be filled in by the user. The background image can be determined by typing in the name of your background image in the text field below the background image previewer. To the right, each individual image has five user configurable fields. The image is determined by the test field directly underneath the image previewer for the respective picture. Below that, the user must enter an x, y coordinate for the images location. These coordinate go from (0, 0) in the bottom left corner to (15, 15) in the top right corner of the virtual space. Below this the user can enter information about the depth location and scale of the image. The depth location is reasonably infinite, where negative values will place the image in the background and positive values will place the image in the foreground. The more negative the number,

the farther away the image will be placed and the more positive the number the closer the image will appear. The scale numeric field controls the size of the image in the system. This value is reasonably infinite in the positive direction only. A value of zero will set the image to a minimum size while the image can be scaled larger by increasing this value. Finally on the far left below the background image previewer is a numeric counter for the number of images. The user can set this number from zero to six to determine how many images will be displayed in their scene. Images included start with picture 1 and are added in order until the determined number of images is met. Finally, the preview button on the far right loads images into the previewers for each image and the background so that the user may verify that the chosen image is correct. If a filename is incorrectly specified, the GUI will notify the user of which image it could not find so that the problem is easily correctly. The start button below the preview button will start the virtual environment in the target and grid stage. User setting can then be loaded from this stage by pressing the ‘r’ (refresh) button on the keyboard.

The next step was to incorporate the GUI into the system and close the loop so that users could change values and images without the need to recompile the executable. Originally, the bulk of the code was located in one file WiiDesktopVR.cs. This file contained all the code for initializing and running the program, which then made calls to the DirectX and Direct3d libraries to load the visuals. In order to close the loop, I had to move a good portion of the initialization code into the GUI file I created, WiiDesktopGUI.cs. The variable initialization would now be done in WiiDesktopGUI.cs and WiiDesktopVR.cs would handle all of the communication with the DirectX/3D libraries. The system would now involve two way

communication between the GUI and WiiDesktopVR.cs and one way communication with the DirectX libraries as shown in the communication flow chart below (**Figure 7).**

Figure 7 - Communication Flow between Major Code Files

5. Construction

The next step was now to actually construct my designs and put together a working project. This section will cover the actual construction of the LED hat based on my designs and how my code was implemented into the system.

5.1 The Making of the LED Hat

Conveniently, all the parts necessary for making the LED hat are available at a local electronics store, such as Radio Shack (with the exception of the baseball cap). A complete parts list for the hat is shown below in **Figure 8**.

- One baseball cap
- Four AA batteries
- One battery wiring pack
- Four high output 5mm infrared LEDs
- One 5mm red LED
- Five LED holders
- One PC board toggle switch
- A pack of 150 ohm resistors
- A pack of 10 ohm resistors
- A roll of connecting wire
- Electrical solder
- Electrical tape

Figure 8 - Parts List for LED Hat

To put the hat together, I first connected all of the components on a small bread board to test that the circuit worked as designed (the actual testing process is described in the following section). Once the circuit was put together and a signal was received from each of the five LEDs, I soldered together the connections with the connecting electrical wire. To embed the LEDs into the hat, I first cut most of the fabric from the brim, specifically all the fabric on the front edge where the infrared LEDs would be placed and a section on the right side for the red LED and toggle switch. I then cut four slits in the front brim of the hat and two holes in the side. The brim now looked as shown below in **Figure 9**.

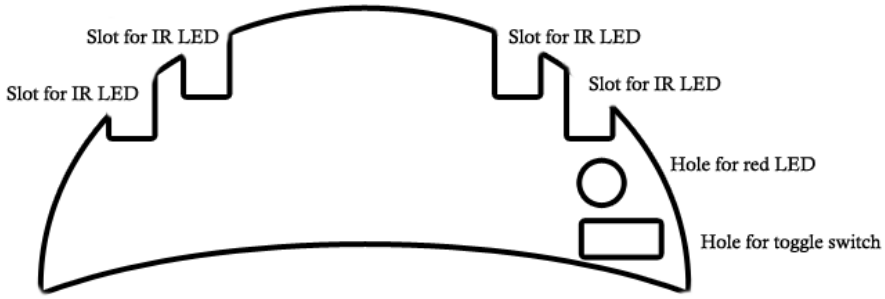

Figure 9 - LED Hole Placement in Hat Brim

Each LED was snapped into its plastic LED holder and each holder was then hot glued into its appropriate spot in the brim. The switch was inserted into its specified hole and secured with a nut and washer. The battery pack was secured to the side of the hat by wrapping wire around the pack and through the mesh of the hat, just above the right ear. Finally, all the connecting wires were taped down to the bottom of the brim so that there were no loose ends that could get caught up during use (I recommend using either a black hat or electrical tape the same color as your brim to make the alterations at least somewhat inconspicuous).

Put the AA batteries into the pack and you should have a working infrared transmitting headpiece. A picture of the final product is shown below in **Figure 10:**

Figure 10 - Final IR LED Hat

5.2 Implementing the Code

With a working LED baseball cap, I was able to start putting together all the pieces of designed code. The visual appearance of the GUI was created using Visual Studios drag and drop features. This made sizing and locating each individual element much easier than having to physically write the code. Once the pieces were in place, code was added to handle each element within the interface: variables needed to be mapped to values entered in the textboxes and number counters (image file names, x, y, z locations of each image in the three dimensional space, relative scale within the environment, number of images to be used, buttons to preview images and start the scene).

Once the GUI was created and coded, it was woven into the rest of the project by adding variables to the WiiDesktopVR.cs code and using these new variables to collect information from the GUI. Inside the WiiDesktopVR.cs code, new methods needed to be added and

many existing methods needed to be changed in order change the environment from a simple target / grid scene into the customizable environment provided by the GUI.

One of the biggest changes made to the system was the relocation of the graphic initialization code shown below in **Figure 11**:

```
using (WiiDesktopVR frm = new WiiDesktopVR(this))
{
    if (!frm.InitializeGraphics()) // Initialize
Direct3D
    {
        MessageBox.Show("Could not initialize Direct3D.
This tutorial will exit.");
        return;
    }
    frm.Show();
    // While the form is still valid, render and process
messages
    while (frm.Created)
    {
        Application.DoEvents();
        if (!frm.doWiimote)
            frm.Render();
    }
    Cursor.Show();
}
```

Figure 11 - Graphics Initialization Code moved from WiiDesktopVR.cs to WiiDesktopGUI.cs

Originally this snippet of code was located in the main() function of the WiiDesktopVR.cs. It was only run through once to initialize the graphics and start up the environment. By moving this to the GUI, it allowed for re-initialization of the graphics in a closed loop environment, that is, it allows for the user to run the environment, then change the images and their locations and re-run the environment without having to restart or recompile the executable. Instead of only initializing and loading the graphics on program start up, the user is now able to reinitialize and reload by simply pressing the “start” button on the GUI.

Another major modification to the code required turning many single value variables into arrays instead, especially variables involved in loading and manipulation of the images. This provided the means to load multiple images into the environment and store their location and scale values as specified by the user.

6. Testing

There were many different facets to the testing process of this system including testing of the designed infrared LED circuit, the LED hat's functioning within the system, Johnny Lee's original code, and my own final version of the code.

6.1 Testing the LED Circuit

One of the benefits of using infrared sensors is that they are invisible to the naked human eye. This allows for us to send signals from the LED hat to the Wii-mote without any distracting lights visible to the user. However, the invisibility of infrared light makes it hard to test if your sensors are working. The addition of a red LED in the hat allows for the user to determine if power is being supplied to the hat while in use. However, this did not help in testing my original circuit design as I was unsure whether or not the infrared LEDs were properly receiving power even when the red LED was on. After some research, I found a few helpful ways to test that my circuit was working. First, I found that the camera used in most standard cell phone cameras is capable of sensing infrared light. By looking at the circuit through my iPhone camera, I was able to see that each of the infrared LEDs were emitting light as expected, displayed as blue blobs of light on my phone's display.

I also did some researching on the internet and found a helpful Wii-mote data parsing application called GlovePIE. GlovePIE parses Wii-mote data received by the computer via Bluetooth and is used by many developers to write scripts for controlling emulated computer games with a Wii-mote. Since the parsing data for this program was already

worked out by Johnny Lee, I did not need to use GlovePIE for this purpose; however it did serve as excellent feedback for testing my LED sensor hat. As I moved the hat around in front of the Wii-mote, I was able to see how the actions were interpreted by GlovePIE to determine that my LED sensors were working as designed.

Finally, the Wii-mote library provided online that is necessary for parsing the Wii-mote data provides a graphical testing executable. Running this executable with the Wii-mote connected provides the user with information about how many LEDs are being picked up by the Wii-motes camera and all the data interpolated from these sensors. The most helpful part of this testing executable is a visual representation of what the Wii-mote camera is capturing. As I moved my hat in front of the Wii-mote, I was able to tell it was working as four different colored dots appeared on the display each representing one of the infrared LEDs in the brim.

6.2 Testing of the LED Hat within the System

After it was determined that the LEDs in the hat were functioning properly, I still needed to test that the hat would work well within the entire system. Johnny Lee has shown that two infrared LEDs on the corners of safety goggles were sufficient, but my design used four LEDs in a slightly different layout. This was tested by simply running the program and comparing how it performed when wearing the LED hat or when holding the Wii sensor bar near my forehead. After some use, it seems as though the program runs a little choppier with the hat than the sensor bar. I think this must be a result of two factors: the LEDs in the hat may not be sending quite as strong of a signal as those in the Wii sensor bar (even though I used high

output infrared LEDs) and the LEDs may not be pointing straight out of the brim of the hat, slightly obscuring the signals being sent to the Wii-mote.

6.3 Testing Johnny Lee's Base Code

Before I could start making modifications and additions to the base code, I needed to test that I could start the system and properly interact with it. This proved to be harder than expected as it entailed much more than simply downloading an executable and starting the system. First I needed to connect to the Wii-mote. The computer I was using did not have a Bluetooth receiver, but I was able to get a Bluetooth dongle that would receive the data from the Wii-mote and transfer it to the system via a USB port. Once the data was received by the computer, it needed to be parsed for the virtual desktop system. To do this I needed to purchase BlueSoleil, a Bluetooth device managing program that was used by Johnny Lee to connect the Wii-mote to his system. I also needed to download the proper Wii-mote Library (version 1.7) that was used by Johnny Lee's system. There are several different versions of this library, not all of which are compatible with the system. Finally I needed to recompile the program into an executable that I could run on my own system. This was also more difficult than expected as the original program was written on a 32-bit machine and I needed to run it on a 64-bit machine. To get the program to work on my machine, I had to download several different versions and change the build properties to compile for an x86 platform before I found a version that would work. Once I finally had the system up and running it was visually verifiable that it was working by simply playing around with the Wii-mote and sensor bar and watching how it affected the display.

6.4 Testing while Developing

The majority of the testing of my code modifications was done incrementally as I developed the program. I would write a segment of code then verify that it produced the desired result before making any other major changes. For example, after writing the GUI code, I first made sure the system worked as a closed loop before attempting to change variable values through the GUI's input blocks. I also wrote and tested code for changing individual images separate from the code written for changing specific image locations and scale. I tested these code alterations separately to make sure they worked as expected, updated them one at a time until they were working properly, then combined the two together and tested the system again. Testing the system as it was developed kept me from being overwhelmed by problems that could result from many different portions of the code and struggling to find where the system was failing.

6.5 Testing of the Final Program

Since the code was tested as it was being developed, the final testing of the system was not extremely difficult. Checking the final system involved many run-throughs of the environment to make sure everything was working without crashing the program. I created a few different backgrounds and images to be placed in the foreground in Photoshop. I then ran through the program changing the images and their locations and size within the environment to see if I could cause an error that I had not yet taken into account. The only major errors I found in the final testing process involved incorrectly labeling image filenames which would cause a "File not found exception" or "null pointer exception" and crash the program. These types of errors were easily fixed with a few simple try / catch blocks that

would halt the program at a safe state and send the user back to the GUI instead of crashing the system.

7. Conclusion

Overall I enjoyed working with the Wii-mote head tracking virtual display as my senior project. By the end of the project I had definitely learned more about system integration and the communication flow of DirectX/3D. Most of the project was based on system integration, from trying to connect all the pieces given to me at the beginning (Wii-mote to Bluetooth, Bluetooth to computer, computer into virtual display) to making modifications to the current system. Incorporating a GUI and making the system a closed loop involved breaking down the code to determine how everything connected with one another, and then rearranging it all in a way that would change the inner workings of the system without disrupting the communication flow. Making these changes to the system layout also forced me to understand how the DirectX/3D visual effects were initialized and connected to each, making this project a lesson in graphics as well (something I have not had much experience working with).

I think that this head tracking virtual display has a potential to be used in many interesting projects in the future. Using the idea behind this project as a base opens doors for three dimensional video games, flight simulators, television, interactive environments, and so much more. I think a good next step for this project would be to implement the ability to move around in the environment, whether it be forward and backward, side to side, or rotating left or right. Moving forward may be able to be implemented by determining how far away the user is from the screen and if they are closer than a specified distance the scene could change to appear as if the user is walking forward. Likewise, if the user is

calculated to be farther than a specified distance the scene could simulate the user walking backwards. Shifting left and right or rotating may be a harder design to implement. It might be possible to use a timer on a switch to oscillate the infrared LEDs on and off at different frequencies. If the system was able to differentiate between these frequencies it, they could be used to signal the program to alter the images accordingly to simulate the desired motion. Adding a dimension of movement into this system would open doors to many other possibilities. With the rate at which technology is changing today, digital experiences are becoming more and more realistic and this new take on three dimensional representations is just the beginning.

8. Bibliography

Johnny Lee's sites (and source code):

http://johnnylee.net/projects/wii/

http://procrastineering.blogspot.com/

Major DirectX and Direct3D references:

http://www.directxtutorial.com/

http://www.gametutorials.com/gtstore/c-5-directx.aspx

http://www.xmission.com/~legalize/book/download/03-Direct3D%20Devices.pdf

http://www.microsoft.com/downloads/details.aspx?familyid=86cf7fa2-e953-475c-abde-f016e4f7b61a&displaylang=en

C# references:

http://msdn.microsoft.com/en-us/vcsharp/default.aspx

Wii mote libraries:

http://blogs.msdn.com/b/coding4fun/archive/2007/03/14/1879033.aspx

http://www.wiimoteproject.com/

GlovePIE:

http://www.simplehelp.net/2009/01/14/how-to-connect-a-wii-remote-to-your-pc/

9. Appendices

9.1 Project Proposal

Senior Project Proposal: Wii-mote Infrared Head Tracking

David Fairman

Advisor: John Oliver

Winter 2010

Abstract

For my senior project I propose to make an infrared head tracking system using multiple infrared LEDs as transmitters and a Nintendo Wii-mote as an infrared receiver. By tracking head movements, the system can be used to display two dimensional images in a three dimensional manner on a standard TV screen or computer monitor by adjusting the image appropriately to the IR transmitters motion.

Theory

A standard Nintendo Wii works by using an infrared transmitter bar to send out four IR signals. The Wii-mote, which contains a 1024x768 infrared camera with 100 Hz built-in blob tracking hardware, then receives the signals simultaneously and calculates its relative

position to control the video game animation. For this project, the roles will be reversed. Two infrared LEDS will be placed on a pair of goggles to be worn on the users head. The Wii-mote will be placed in a stationary position next to a display monitor. As the user moves their head, the Wii-mote will be able to detect the change in position of the IR signals transmitted by the LEDs. In this way, the system will be able to track the location and orientation of the users head and adjust an image on the display monitor appropriately. By moving different aspects of the image to the right as the user moves their head left and vice versa, as well as zooming in or out on an image as the user moves closer or further from the Wii-mote, a three dimensional looking image is able to be created.

Objectives

This idea has already been put into practice by current Microsoft employee Johnny Chung. I plan to implement and expand upon his idea to take it to a level past its current application. By following his design I will first get the transmitter glasses working with the Wii-mote receiver. There is helpful code online to then get a basic three dimensional image application in place. In a demonstration video, Chung also shows how the system may be able to be used as a virtual window by having a photographic image react to head movements to display different parts of the picture. I would like to integrate this with my own panoramic photo design experience to create a new method of looking at panoramic photos. This could be especially useful in navigating around a 360 degree panorama to create a virtual environment for the user.

Budget

This project should be a relatively low cost project as it does not require buying any expensive resources. I will need to buy a Nintendo Wii-mote which should cost somewhere on the order of $35. I will also need to buy a few infrared LEDs and a pair of goggles to attach them to which should cost no more than $20 together. I should be able to use my own computer screen or TV for the display. The rest of the project should only entail a C# SDK which can be downloaded for free from the internet.

9.2 Parts List and Cost

PART	PRICE
One baseball cap	7.99
Four AA batteries	12.99
One battery wiring pack	1.99
Four high output 5mm infrared LEDs	1.75
One 5mm red LED	1.49
Five LED holders	1.49
One PC board toggle switch	4.95
A pack of 150 ohm resistors	0.25
A pack of 10 ohm resistors	0.25
A roll of connecting wire	4.99
Electrical solder	6.99
Electrical tape	3.99
Blue Soleil Subscription	27.99
TOTAL	**77.11**

9.3 Schedule

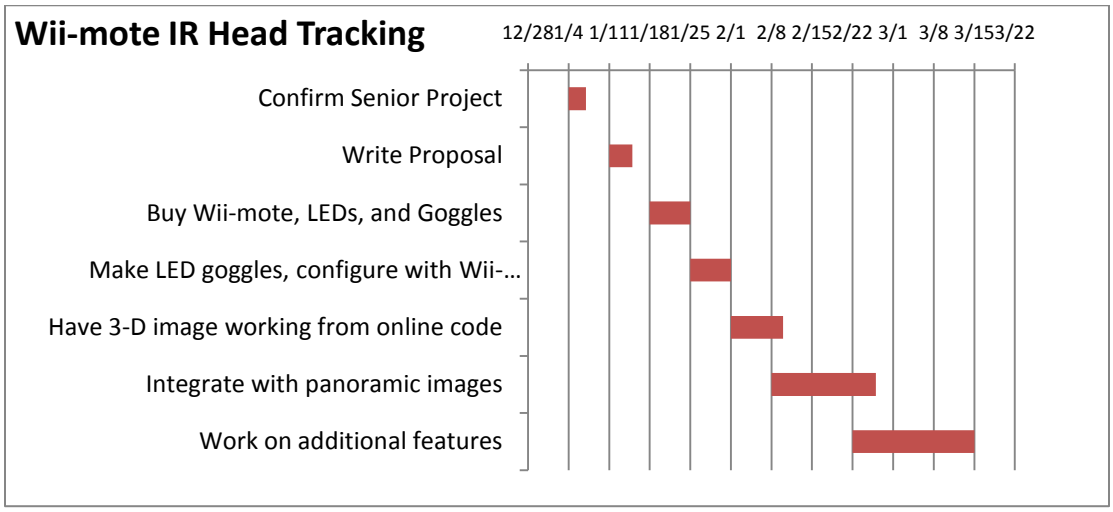

Figure 12 - Gantt Chart for Winter Quarter

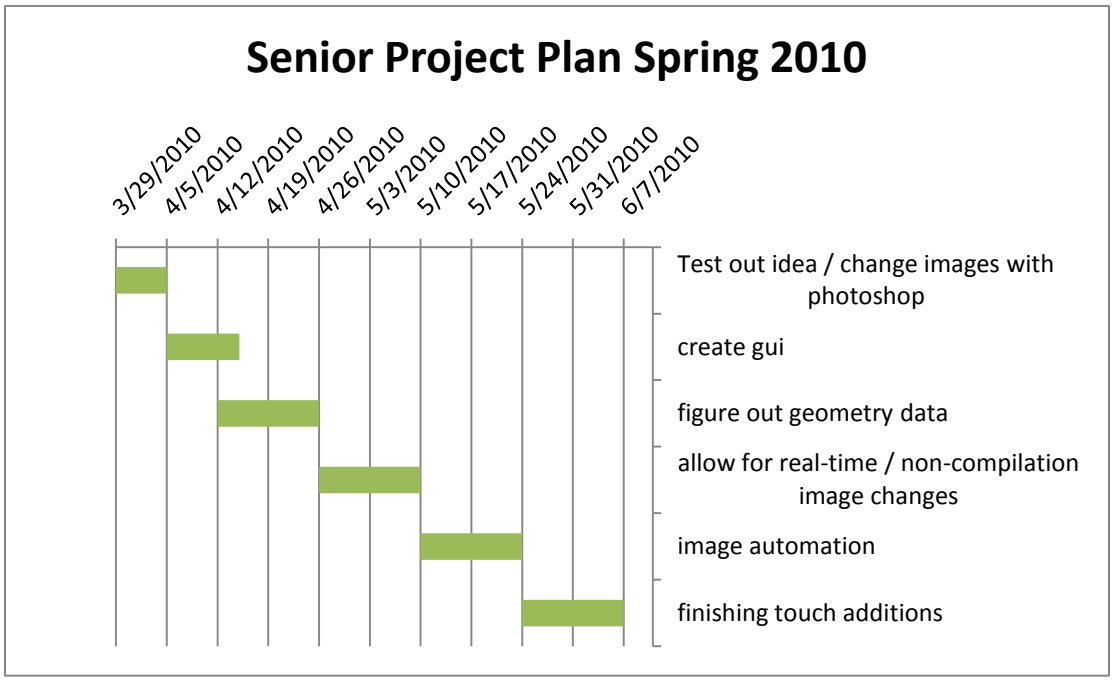

Figure 13 - Gant Chart for Spring Quarter

9.4 Instructions for Use

1. Connect Wii-mote to computer via Bluetooth, if Wii-mote is not connected the system will not be able to initialize and the GUI will not load
2. Put on the LED hat and toggle the switch to “On”. The red LED light next to the switch will turn on, indicating the hat is activated.
3. Place any images to be used in foreground or background in folder /images
4. Run the executable WiiDesktopVR.exe located in /bin/Release
5. Specify the image to be used for the background
6. Specify up to six images to be used in the foreground, indicating their desired X,Y coordinates, depth, and scale within the three dimensional space
7. Specify the number of foreground images to be used in the environment
8. Press the “Preview” button to make sure images were able to be found properly
9. Press the “Start” button to begin the program, the original grid and target scene will load first
10. Press ‘R’ to load your customized three dimensional scene
11. Press ‘H’ for a list of help options. A list of options and actions will be loaded in yellow text over the scene. If the text color is not visible well on the screen, press ‘J’ to change the text to Blue.
12. To make any changes to the scene, press “ESC” and you will return back to the GUI
13. Make any desired changes and press “Start” then ‘R’ to reload your environment
14. To terminate the program, press “ESC” to get back to the GUI then exit out of the system with the X box in the top right corner of the GUI window.

9.5 Development on 64-bit Machine

Most of the software development I contributed to this project was done on my home laptop, which happens to be a 64-bit windows machine. However, the original source code was developed for a 32-bit windows machine. In order to properly edit, compile, and execute the code I had to go through several necessary steps to manage the code on a 64-bit platform. These steps are as follows:

1. Download and install Microsoft Visual Studio 2008
   - Microsoft Visual C# Express can be found online for free and may also work, but I found it to be buggy with the 64-bit architecture
2. Download and install Microsoft DirectX SDK
   - It is important that the most recent version of DirectX is installed for a 64-bit machine
3. Download a copy of the Wiimote Library, which can be found at wiimotelib.codeplex.com, among other places

Once these three major components are downloaded and installed, you will need to change your project setting to force the executables to compile in 32-bit mode. To do this:

1. Open the project solution WiiDesktopVR.sln in Visual Studios / Visual Basic
2. Open the settings windows through Project -> WiiDesktopVR Properties
3. Under both the Build tab and the Debug tab change the Configuration to “Active (Release)” and the Platform to “Active (Any CPU)”
4. Close and reopen the solution file and you should now be able to develop, compile and run the code on your 64-bit machine

9.6 Calculating Head Position

Figure 14 below shows the calculations used to determine the location of the users head based on distance and orientation in reference to the Wii-mote. These calculations were drawn up by Johnny Lee for his virtual reality display and are represented in code in the following section.

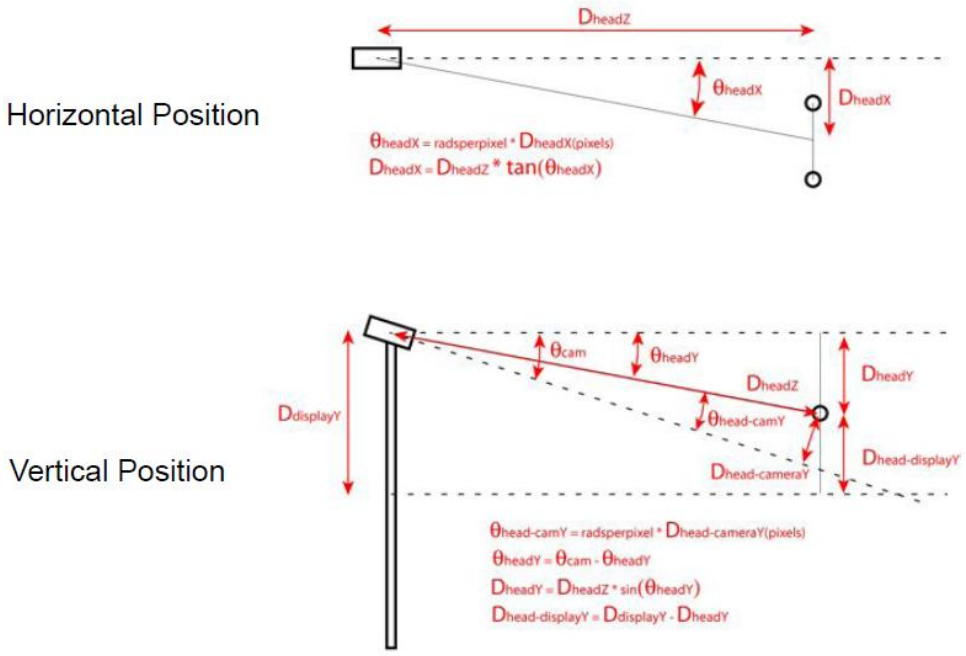

Figure 14 - Calculations to Determine Head Location

9.7 Modified Source Code

There are really only two interesting files that were used to control the system, WiiDesktopVR.cs and WiiDesktopGUI.cs shown below. The rest of the code contained in the project is merely properties files, assembly information, graphics initialization code, and extensive Wii-mote and DirectX libraries.

WiiDesktopVR.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
using Direct3D=Microsoft.DirectX.Direct3D;
using Microsoft.Samples.DirectX.UtilityToolkit;
using WiimoteLib;
using System.Threading;
using System.IO;//for reading config file


namespace WiiDesktopVR
{
    class Point2D
    {
        public float x = 0.0f;
        public float y = 0.0f;
        public void set(float x, float y)
        {
            this.x = x;
            this.y = y;
        }
    }

      public class WiiDesktopVR : Form
      {
        struct Vertex
        {
            float x, y, z;
            float tu, tv;

            public Vertex(float _x, float _y, float _z, float _tu,
float _tv)
            {
                x = _x; y = _y; z = _z;
                tu = _tu; tv = _tv;
            }
```

```
            public static readonly VertexFormats FVF_Flags =
VertexFormats.Position | VertexFormats.Texture1;
        };

        Vertex[] targetVertices =
            {
                  new Vertex(-1.0f, 1.0f,.0f,  0.0f,0.0f ),
                  new Vertex( 1.0f, 1.0f,.0f,  1.0f,0.0f ),
                  new Vertex(-1.0f,-1.0f,.0f,  0.0f,1.0f ),
                  new Vertex( 1.0f,-1.0f,.0f,  1.0f,1.0f ),
        };

        // Our global variables for this project
            Device device = null; // Our rendering device

        int numGridlines = 10;
        float boxdepth = 8;
        float fogdepth = 5;

        //Create arrays to allow for multiple images in the
foreground
        Texture[] textures = new Texture[] { null, null, null, null,
null, null };
        String[] textureFilenames = new String[]
{"images/target.png", "images/target.png",
            "images/target.png", "images/target.png",
"images/target.png", "images/target.png"};

        bool showBackground = false;
        Texture backgroundtexture = null;
        String backgroundFilename = "images/stad_2.png";
        int backgroundStepCount = 10;

        float dotDistanceInMM = 8.5f * 25.4f;//width of the wii
sensor bar
        float screenHeightinMM = 20 * 25.4f;
        float radiansPerPixel = (float)(Math.PI / 4) / 1024.0f; //45
degree field of view with a 1024x768 camera
        float movementScaling = 1.0f;

        int gridColor = 0xCCCCCC;
        int lineColor = 0xFFFFFF;
        int lineDepth = -200;

        VertexBuffer gridBuffer = null;
        VertexBuffer backgroundBuffer = null;
        VertexBuffer targetBuffer = null;

        VertexBuffer lineBuffer = null;
        Random random = new Random();

        int numTargets = 6;
        int numInFront = 3;
```

```
        //create array for different scale of each foreground image
        float[] targetScale = new float[] { .065f, .065f, .065f,
.065f, .065f, .065f };

        Vector3[] targetPositions;
        Vector3[] targetSizes;

        //create array for different x, y coordinates for each image
        Vector3[] locations;

        bool showTargets = true;
        bool showLines = true;

        bool isRendering = false;

        Point2D[] wiimotePointsNormalized = new Point2D[4];
        int[] wiimotePointIDMap = new int[4];

            PresentParameters presentParams = new
PresentParameters();
        private Sprite textSprite = null; // Sprite for batching
text calls
        private Microsoft.DirectX.Direct3D.Font statsFont = null; //
Font for drawing text

        bool isReady = false;
        bool doFullscreen = false;

        //add dimensions of your display
        int m_dwWidth = 1366;
        int m_dwHeight = 768;
        float screenAspect =0;

        float cameraVerticaleAngle = 0; //begins assuming the camera
is point straight forward
        float relativeVerticalAngle = 0; //current head position
view angle
        bool cameraIsAboveScreen = false;//has no affect until
zeroing and then is set automatically.

        //bool badConfigFile = false;

        CrosshairCursor mouseCursor;
        int lastFrameTick = 0;
        int frameCount;
        float frameRate = 0;

        Matrix worldTransform = Matrix.Identity;

        bool showGrid = true;
        bool showHelp = false;
        bool showMouseCursor = false;

        int lastKey = 0;
```

```
        bool mouseDown = false;

        //create alternate text color for help menu
        System.Drawing.Color helpColor =
System.Drawing.Color.Yellow;
        WiiDesktopGUI GUI;

        //wiimote stuff
        public bool doWiimote = true;
        bool doWiimote2 = false;
        Wiimote remote;
        //Wiimote remote2;
        CrosshairCursor wiiCursor1;
        CrosshairCursor wiiCursor2;
        CrosshairCursor wiiCursor3;
        CrosshairCursor wiiCursor4;

        bool doWiiCursors = true;
        //int leftCursor = 1; //needed for rotation stabilization
when two points appear

        //headposition
        float headX = 0;
        float headY = 0;
        float headDist = 2;

            public WiiDesktopVR(WiiDesktopGUI GUI)
            {
            //connect GUI to system
            this.GUI = GUI;

            // Set the initial size of our form
            this.ClientSize = new System.Drawing.Size(m_dwWidth,
m_dwHeight);

            loadConfigurationData();

            if (screenAspect == 0)//only override if it's emtpy
                screenAspect = m_dwWidth / (float)m_dwHeight;
            this.Text = "Wiimote Desktop VR";

            //add event handlers
            this.MouseDown += new
System.Windows.Forms.MouseEventHandler(this.OnMouseDown);
            this.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.OnMouseUp);
            this.MouseMove += new
System.Windows.Forms.MouseEventHandler(this.OnMouseMove);
            this.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.OnKeyPress);
            this.FormClosing += new
FormClosingEventHandler(this.OnFormClosing);

            for (int i = 0; i < 4; i++)
```

```
            {
                wiimotePointsNormalized[i] = new Point2D();
                wiimotePointIDMap[i] = i;
            }
        }

        public void loadConfigurationData()
        {
            // create reader & open file
            try
            {
                TextReader tr = new StreamReader("config.dat");
                char[] seps = { ':' };
                String line;
                String[] values;

                line = tr.ReadLine();
                values = line.Split(seps);
                screenHeightinMM = float.Parse(values[1]);

                line = tr.ReadLine();
                values = line.Split(seps);
                dotDistanceInMM = float.Parse(values[1]);

                line = tr.ReadLine();
                values = line.Split(seps);
                screenAspect = float.Parse(values[1]);

                line = tr.ReadLine();
                values = line.Split(seps);
                cameraIsAboveScreen = bool.Parse(values[1]);

                line = tr.ReadLine();
                values = line.Split(seps);
                doWiimote2 = bool.Parse(values[1]);

                // close the stream
                tr.Close();
            }
            catch (System.NullReferenceException)
            {

            }
            catch (System.FormatException)
            {
                //bad config, ignore
                throw new Exception("Config file is mal-
formatted.");

            }
            catch (System.IO.FileNotFoundException)
            {
                //no prexsting config, create one with the deafult
values
```

```
                TextWriter tw = new StreamWriter("config.dat");

                // write a line of text to the file
                tw.WriteLine("screenHieght(mm):" +
screenHeightinMM);
                tw.WriteLine("sensorBarWidth(mm):" +
dotDistanceInMM);
                tw.WriteLine("screenAspect(width/height):" +
screenAspect);
                tw.WriteLine("cameraIsAboveScreen(true/false):" +
cameraIsAboveScreen);
                tw.WriteLine("cnnectToSecondWiimote(true/false):" +
doWiimote2);

                // close the stream
                tw.Close();

                return;
            }
        }


        void OnFormClosing(object sender, FormClosingEventArgs e)
        {
            isReady = false;//set the flag to stop the rendering
call driven by incoming wiimote reports
            Cursor.Show();
        }

        private void OnKeyPress(object sender,
System.Windows.Forms.KeyEventArgs e)
        {
            lastKey = (int)e.KeyCode;
            if ((int)(byte)e.KeyCode == (int)Keys.Escape)
            {
                isReady = false;
                Cursor.Show();//set the flag to stop the rendering
call driven by incoming wiimote reports
                this.Dispose(); // Esc was pressed
                return;
            }
            if ((int)(byte)e.KeyCode == (int)Keys.Space)
            {
                //zeros the head position and computes the camera
tilt
                double angle = Math.Acos(.5 / headDist)-Math.PI /
2;//angle of head to screen
                if (!cameraIsAboveScreen)
                    angle  = -angle;
                cameraVerticaleAngle = (float)((angle-
relativeVerticalAngle));//absolute camera angle
            }
            if ((int)(byte)e.KeyCode == 'C')
```

```
            {
                cameraIsAboveScreen = !cameraIsAboveScreen;
            }
            if ((int)(byte)e.KeyCode == 'B')
                showBackground = !showBackground;
            if ((int)(byte)e.KeyCode == 'G')
                showGrid = !showGrid;
            if ((int)(byte)e.KeyCode == 'R')
            {
                //load customized environment with variables from
GUI
                reload();
            }
            if ((int)(byte)e.KeyCode == 'H')
            {
                //show yellow help menu (good for darker
backgrounds)
                showHelp = !showHelp;
                helpColor = System.Drawing.Color.Yellow;
            }
            if ((int)(byte)e.KeyCode == 'J')
            {
                //show blue help menu (good for light backgrounds)
                showHelp = !showHelp;
                helpColor = System.Drawing.Color.Blue;
            }
            if ((int)(byte)e.KeyCode == 'T')
                showTargets = !showTargets;
            if ((int)(byte)e.KeyCode == 'L')
                showLines = !showLines;
            if ((int)(byte)e.KeyCode == 'M')
                showMouseCursor = !showMouseCursor;
        }

        private void reload()
        {
            //initialize variables from GUI
            numTargets = GUI.numTargets;

            for (int i = 0; i < numTargets; i++)
            {
                textureFilenames[i] = GUI.pics[i];
            }

            locations = GUI.locations;
            backgroundFilename = GUI.bgFile;
            targetScale = GUI.targetScale;

            //reload system with specified variables
            InitTargetsLocated();
            LoadTexture();
            showBackground = !showBackground;
            showGrid = !showGrid;
            showLines = !showLines;
```

```
        }

        private void OnMouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
        {
            switch (e.Button)
            {
                case MouseButtons.Left:
                    mouseDown = true;
                    break;
                case MouseButtons.Right:
                    break;
                case MouseButtons.Middle:
                    break;
                default:
                    break;
            }

        }

        private void OnMouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
        {
            mouseCursor.setDown(screenAspect * (e.X /
(float)m_dwWidth - .5f) + .0f, .5f - e.Y / (float)m_dwHeight);

            if (mouseDown)//is dragging
            {
                //rotation? - consider possibility of adding a
rotational feature to code
            }
        }


        private void OnMouseUp(object sender,
System.Windows.Forms.MouseEventArgs e)
        {
            switch (e.Button)
            {
                case MouseButtons.Left:
                    mouseDown = false;
                    break;
                case MouseButtons.Right:
                    break;
                case MouseButtons.Middle:
                    break;
                default:
                    break;
            }
        }


        private void RenderText(System.Drawing.Color helpColor)
        {
```

```
            TextHelper txtHelper = new TextHelper(statsFont,
textSprite, 15);
            txtHelper.Begin();
            txtHelper.SetInsertionPoint(5, 5);

            // Output statistics
            txtHelper.SetForegroundColor(helpColor);
            txtHelper.DrawTextLine("Stats---------------");

            frameCount++;
            if (frameCount == 100)
            {
                frameRate = 100 * 1000.0f / (Environment.TickCount -
lastFrameTick);
                lastFrameTick = Environment.TickCount;
                frameCount = 0;
            }

            txtHelper.DrawTextLine("Avg Framerate: " + frameRate);
            if (remote != null)
            {

                txtHelper.DrawTextLine("Wii IR dots:" +
remote.WiimoteState.IRState.IRSensors[0].Found + " "
                                                        +
remote.WiimoteState.IRState.IRSensors[1].Found + " "
                                                    +
remote.WiimoteState.IRState.IRSensors[2].Found + " "
                                                     +
remote.WiimoteState.IRState.IRSensors[3].Found);
            }
            txtHelper.DrawTextLine("Last Key Pressed: " + lastKey);
            txtHelper.DrawTextLine("Mouse X-Y: " + mouseCursor.X +
", " +mouseCursor.Y);
            txtHelper.DrawTextLine("Est Head X-Y (mm): " + headX *
screenHeightinMM + ", " + headY * screenHeightinMM);
            txtHelper.DrawTextLine("Est Head Dist (mm): " +
headDist*screenHeightinMM);
            txtHelper.DrawTextLine("Camera Vert Angle (rad): " +
cameraVerticaleAngle);
            if(cameraIsAboveScreen)
                txtHelper.DrawTextLine("Camera Position: Above
Screen");
            else
                txtHelper.DrawTextLine("Camera Position: Below
Screen");
            txtHelper.DrawTextLine("Screen Height (mm) : " +
screenHeightinMM);
            txtHelper.DrawTextLine("IR Dot Width (mm) : " +
dotDistanceInMM);
            txtHelper.DrawTextLine("");

            txtHelper.DrawTextLine("Controls -----");
```

```
            txtHelper.DrawTextLine("Space - calibrate camera
angle/center view");
            txtHelper.DrawTextLine("R - Reposition the targets");
            txtHelper.DrawTextLine("C - Toggle camera position
above/below screen");
            txtHelper.DrawTextLine("esc - Quit");
            txtHelper.DrawTextLine("");

            txtHelper.DrawTextLine("Show--------");
            txtHelper.DrawTextLine("T - Targets");
            txtHelper.DrawTextLine("L - Lines");
            txtHelper.DrawTextLine("B - Background");
            txtHelper.DrawTextLine("M - Mouse Cursor");
            txtHelper.DrawTextLine("G - Grid");
            txtHelper.DrawTextLine("H - Help Text (Yellow)");
            txtHelper.DrawTextLine("J - Help Text (Blue)");
            txtHelper.DrawTextLine("");

            txtHelper.End();
        }

            public bool InitializeGraphics()
            {
                  try
                  {
                this.FormBorderStyle = FormBorderStyle.None;//this
is the bug that kept thing crashing C# on vista

                AdapterInformation ai = Manager.Adapters.Default;
                Caps caps = Manager.GetDeviceCaps(ai.Adapter,
DeviceType.Hardware);

                Cursor.Hide();
                presentParams.Windowed=!doFullscreen;
                presentParams.SwapEffect = SwapEffect.Discard; //
Discard the frames
                        presentParams.EnableAutoDepthStencil = true;
// Turn on a Depth stencil
                        presentParams.AutoDepthStencilFormat =
DepthFormat.D16; // And the stencil format

                presentParams.BackBufferWidth = m_dwWidth;
                  //screen width
                presentParams.BackBufferHeight = m_dwHeight;
                        //screen height
                presentParams.BackBufferFormat = Format.R5G6B5;
                        //color depth
                presentParams.MultiSample = MultiSampleType.None;
                  //anti-aliasing
                presentParams.PresentationInterval  =
PresentInterval.Immediate; //don't wait... draw right away
```

```
                device = new Device(0, DeviceType.Hardware, this,
CreateFlags.SoftwareVertexProcessing, presentParams); //Create a
device
                device.DeviceReset += new
System.EventHandler(this.OnResetDevice);
                        this.OnCreateDevice(device, null);
                        this.OnResetDevice(device, null);
                        return true;
                    }
                    catch (DirectXException)
                    {
                        // Catch any errors and return a failure
                        return false;
                    }

            }


        public void InitTargets()
        {
            if(targetPositions==null)
                targetPositions = new Vector3[numTargets];
            if (targetSizes == null)
                targetSizes = new Vector3[numTargets];
            float depthStep = (boxdepth / 2.0f) / numTargets;
            float startDepth = numInFront*depthStep;

             //ORIGINAL - Random target locations
            for (int i = 0; i < numTargets; i++)
            {
                targetPositions[i] = new Vector3(    .7f *
screenAspect * (random.Next(1000) / 1000.0f - .5f),
                                                     .7f *
(random.Next(1000) / 1000.0f - .5f),
                                                     startDepth-
i*depthStep);
                if (i < numInFront)//pull in the ones out in front
of the display closer the center so they stay in frame
                {
                    targetPositions[i].X *= .5f;
                    targetPositions[i].Y *= .5f;
                }
                targetSizes[i] = new Vector3(targetScale[0],
targetScale[0], targetScale[0]);
            }
        }

        //locate images at locations determined by GUI
        public void InitTargetsLocated()
        {
            if (targetPositions == null)
                targetPositions = new Vector3[numTargets];
            if (targetSizes == null)
                targetSizes = new Vector3[numTargets];
```

```
            float depthStep = (boxdepth / 2.0f) / numTargets;
            float startDepth = numInFront * depthStep;

            for (int i = 0; i < numTargets; i++)
            {
                targetPositions[i] = new Vector3(.7f * screenAspect
* (.083f*(15-locations[i].X) - .5f),
                                                 .7f *
(.083f*locations[i].Y - .5f),

0.3f*(locations[i].Z));

                targetSizes[i] = new Vector3(targetScale[i],
targetScale[i], targetScale[i]);
            }
        }

        public void CreateGridGeometry(Device dev)
        {
            int step = m_dwWidth / numGridlines;
            gridBuffer = new
VertexBuffer(typeof(CustomVertex.PositionColored), 4 * (numGridlines
+ 2), dev, 0, CustomVertex.PositionColored.Format, Pool.Default);

            CustomVertex.PositionColored[] verts2;
            verts2 =
(CustomVertex.PositionColored[])gridBuffer.Lock(0, 0); // Lock the
buffer (which will return our structs)
            int vertIndex = 0;
            for (int i = 0; i <= numGridlines * 2; i += 2)
            {
                verts2[vertIndex].Position = new Vector3((i * step /
2.0f) / m_dwWidth, 0.0f, 0.0f);
                verts2[vertIndex].Color = gridColor;
                vertIndex++;
                verts2[vertIndex].Position = new Vector3((i * step /
2.0f) / m_dwWidth, 1.0f, 0.0f);
                verts2[vertIndex].Color = gridColor;
                vertIndex++;
            }
            for (int i = 0; i <= numGridlines * 2; i += 2)
            {
                verts2[vertIndex].Position = new Vector3(0.0f, (i *
step / 2.0f) / m_dwWidth, 0.0f);
                verts2[vertIndex].Color = gridColor;
                vertIndex++;
                verts2[vertIndex].Position = new Vector3(1.0f, (i *
step / 2.0f) / m_dwWidth, 0.0f);
                verts2[vertIndex].Color = gridColor;
                vertIndex++;
            }
            gridBuffer.Unlock();
        }
```

```
        //load in images to be used in the environment
        private void LoadTexture()
        {
            int j = 0;
            try
            {
                System.IO.Directory.SetCurrentDirectory(
                   System.Windows.Forms.Application.StartupPath +
@"\..\..\");

                for (j = 0; j < numTargets; j++)
                {
                    textures[j] = TextureLoader.FromFile(device,
textureFilenames[j]);
                }
            }
            catch
            {
                MessageBox.Show("Cannot find picture "+(j+1));
                isReady = false;
                Cursor.Show();//set the flag to stop the rendering
call driven by incoming wiimote reports
                this.Dispose();
                return;
            }
            device.SamplerState[0].MinFilter = TextureFilter.Linear;
            device.SamplerState[0].MagFilter = TextureFilter.Linear;
            LoadBackground();
        }

        //load background image to be used in environment
        private void LoadBackground()
        {
            try
            {
                System.IO.Directory.SetCurrentDirectory(
                System.Windows.Forms.Application.StartupPath +
@"\..\..\");

                backgroundtexture = TextureLoader.FromFile(device,
backgroundFilename);

            }
            catch
            {
                MessageBox.Show("Cannot find background file");
                isReady = false;
                Cursor.Show();//set the flag to stop the rendering
call driven by incoming wiimote reports
                this.Dispose();
                return;
            }

            device.SamplerState[0].MinFilter = TextureFilter.Linear;
```

```
            device.SamplerState[0].MagFilter = TextureFilter.Linear;
        }

        public void CreateTargetGeometry(Device dev)
        {
            targetBuffer = new VertexBuffer(typeof(Vertex),
                                            targetVertices.Length,
dev,
                                            Usage.Dynamic |
Usage.WriteOnly,
                                            Vertex.FVF_Flags,
                                            Pool.Default);

            GraphicsStream gStream = targetBuffer.Lock(0, 0,
LockFlags.None);

            // Now, copy the vertex data into the vertex buffer
            gStream.Write(targetVertices);
            targetBuffer.Unlock();


            lineBuffer = new
VertexBuffer(typeof(CustomVertex.PositionColored),
                                2,
                                 dev,
                                 Usage.Dynamic | Usage.WriteOnly,

CustomVertex.PositionColored.Format,
                                 Pool.Default);

            CustomVertex.PositionColored[] verts;
            verts =
(CustomVertex.PositionColored[])lineBuffer.Lock(0, 0); // Lock the
buffer (which will return our structs)
            verts[0].Position = new Vector3(0.0f, 0.0f, 0.0f);
            verts[0].Color = lineColor;

            verts[1].Position = new Vector3(0.0f, 0.0f, lineDepth);
            verts[1].Color = lineColor;

            lineBuffer.Unlock();

        }

        public void CreateBackgroundGeometry(Device dev)
        {

            backgroundBuffer = new
VertexBuffer(typeof(CustomVertex.PositionTextured), 2 *
(backgroundStepCount+1), dev, 0,
CustomVertex.PositionTextured.Format, Pool.Default);

            CustomVertex.PositionTextured[] verts;
```

```
            verts =
(CustomVertex.PositionTextured[])backgroundBuffer.Lock(0, 0); //
Lock the buffer (which will return our structs)
            float angleStep = (float)(Math.PI /
backgroundStepCount);
            for (int i = 0; i <= backgroundStepCount; i++)
            {
                verts[2 * i].Position = new
Vector3((float)(Math.Cos(angleStep * i)), -1, -
(float)(Math.Sin(angleStep * i)));
                verts[2 * i].Tu = i / (float)backgroundStepCount;
                verts[2 * i].Tv = 1;

                verts[2 * i + 1].Position = new
Vector3((float)(Math.Cos(angleStep * i)), 1, -
(float)(Math.Sin(angleStep * i)));
                verts[2 * i + 1].Tu = i /
(float)backgroundStepCount;
                verts[2 * i + 1].Tv = 0;

            }
            backgroundBuffer.Unlock();
        }

        public void OnCreateVertexBuffer(object sender, EventArgs e)
        {
            VertexBuffer vb = (VertexBuffer)sender;
            // Create a vertex buffer (100 customervertex)
            CustomVertex.PositionNormalTextured[] verts =
(CustomVertex.PositionNormalTextured[])vb.Lock(0, 0); // Lock the
buffer (which will return our structs)
            for (int i = 0; i < 50; i++)
            {
                // Fill up our structs
                float theta = (float)(2 * Math.PI * i) / 49;
                verts[2 * i].Position = new
Vector3((float)Math.Sin(theta), -1, (float)Math.Cos(theta));
                verts[2 * i].Normal = new
Vector3((float)Math.Sin(theta), 0, (float)Math.Cos(theta));
                verts[2 * i].Tu = ((float)i) / (50 - 1);
                verts[2 * i].Tv = 1.0f;
                verts[2 * i + 1].Position = new
Vector3((float)Math.Sin(theta), 1, (float)Math.Cos(theta));
                verts[2 * i + 1].Normal = new
Vector3((float)Math.Sin(theta), 0, (float)Math.Cos(theta));
                verts[2 * i + 1].Tu = ((float)i) / (50 - 1);
                verts[2 * i + 1].Tv = 0.0f;
            }
            // Unlock (and copy) the data
            vb.Unlock();
        }

        public void OnCreateDevice(object sender, EventArgs e)
            {
```

```
                Device dev = (Device)sender;
            textSprite = new Sprite(dev);
            statsFont =
ResourceCache.GetGlobalInstance().CreateFont(dev, 15, 0,
FontWeight.Bold, 1, false, CharacterSet.Default,Precision.Default,
FontQuality.Default, PitchAndFamily.FamilyDoNotCare |
PitchAndFamily.DefaultPitch, "Arial");

            //init cursors
            mouseCursor = new CrosshairCursor(dev, 0x00ff00, .04f);
            wiiCursor1 = new CrosshairCursor(dev, 0x00ff00, .04f);
            wiiCursor2 = new CrosshairCursor(dev, 0x0000ff, .04f);
            wiiCursor3 = new CrosshairCursor(dev, 0xff0000, .04f);
            wiiCursor4 = new CrosshairCursor(dev, 0xffff00, .04f);

            CreateGridGeometry(dev);
            CreateBackgroundGeometry(dev);
            CreateTargetGeometry(dev);

            InitTargets();
            LoadTexture();

            if (doWiimote)
            {
                try
                {
                    remote = new Wiimote();
                    remote.Connect();
                    remote.SetReportType(InputReport.IRAccel, true);
                    remote.SetLEDs(true, false, false, false);
                    remote.WiimoteChanged +=new
EventHandler<WiimoteChangedEventArgs>(wm_OnWiimoteChanged);
                }
                catch (Exception x)
                {
                    MessageBox.Show("Cannot find a wii remote: " +
x.Message);
                    doWiimote = false;
                }

            }

            }

        void wm_OnWiimoteChanged(object sender,
WiimoteChangedEventArgs args)
        {
            ParseWiimoteData();
            if (isReady)
                Render();//wiimote triggered wiimote thread
        }

        public void ParseWiimoteData()
        {
```

```
            if (remote.WiimoteState == null)
                return;

            Point2D firstPoint = new Point2D();
            Point2D secondPoint = new Point2D();
            int numvisible = 0;

            if (remote.WiimoteState.IRState.IRSensors[0].Found)
            {
                wiimotePointsNormalized[0].x = 1.0f-
remote.WiimoteState.IRState.IRSensors[0].RawPosition.X / 768.0f;
                wiimotePointsNormalized[0].y =
remote.WiimoteState.IRState.IRSensors[0].RawPosition.Y / 768.0f;
                wiiCursor1.isDown = true;
                firstPoint.x =
remote.WiimoteState.IRState.IRSensors[0].RawPosition.X;
                firstPoint.y =
remote.WiimoteState.IRState.IRSensors[0].RawPosition.Y;
                numvisible = 1;
            }
            else
            {//not visible
                wiiCursor1.isDown = false;
            }
            if (remote.WiimoteState.IRState.IRSensors[1].Found)
            {
                wiimotePointsNormalized[1].x = 1.0f -
remote.WiimoteState.IRState.IRSensors[1].RawPosition.X / 768.0f;
                wiimotePointsNormalized[1].y =
remote.WiimoteState.IRState.IRSensors[1].RawPosition.Y / 768.0f;
                wiiCursor2.isDown = true;
                if (numvisible == 0)
                {
                    firstPoint.x =
remote.WiimoteState.IRState.IRSensors[1].RawPosition.X;
                    firstPoint.y =
remote.WiimoteState.IRState.IRSensors[1].RawPosition.Y;
                    numvisible = 1;
                }
                else
                {
                    secondPoint.x =
remote.WiimoteState.IRState.IRSensors[1].RawPosition.X;
                    secondPoint.y =
remote.WiimoteState.IRState.IRSensors[1].RawPosition.Y;
                    numvisible = 2;
                }
            }
            else
            {//not visible
                wiiCursor2.isDown = false;
            }
            if (remote.WiimoteState.IRState.IRSensors[2].Found)
            {
```

```
                wiimotePointsNormalized[2].x = 1.0f -
remote.WiimoteState.IRState.IRSensors[2].RawPosition.X / 768.0f;
                wiimotePointsNormalized[2].y =
remote.WiimoteState.IRState.IRSensors[2].RawPosition.Y / 768.0f;
                wiiCursor3.isDown = true;
                if (numvisible == 0)
                {
                    firstPoint.x =
remote.WiimoteState.IRState.IRSensors[2].RawPosition.X;
                    firstPoint.y =
remote.WiimoteState.IRState.IRSensors[2].RawPosition.Y;
                    numvisible = 1;
                }
                else if(numvisible==1)
                {
                    secondPoint.x =
remote.WiimoteState.IRState.IRSensors[2].RawPosition.X;
                    secondPoint.y =
remote.WiimoteState.IRState.IRSensors[2].RawPosition.Y;
                    numvisible = 2;
                }
            }
            else
            {//not visible
                wiiCursor3.isDown = false;
            }
            if (remote.WiimoteState.IRState.IRSensors[3].Found)
            {
                wiimotePointsNormalized[3].x = 1.0f -
remote.WiimoteState.IRState.IRSensors[3].RawPosition.X / 768.0f;
                wiimotePointsNormalized[3].y =
remote.WiimoteState.IRState.IRSensors[3].RawPosition.Y / 768.0f;
                wiiCursor4.isDown = true;
                if(numvisible==1)
                {
                    secondPoint.x =
remote.WiimoteState.IRState.IRSensors[3].RawPosition.X;
                    secondPoint.y =
remote.WiimoteState.IRState.IRSensors[3].RawPosition.Y;
                    numvisible = 2;
                }
            }
            else
            {//not visible
                wiiCursor4.isDown = false;
            }

            if (numvisible == 2)
            {


                float dx = firstPoint.x - secondPoint.x;
                float dy = firstPoint.y - secondPoint.y;
```

```
                float pointDist = (float)Math.Sqrt(dx * dx + dy *
dy);

                float angle = radiansPerPixel * pointDist / 2;
                //in units of screen hieght since the box is a unit
cube and box hieght is 1
                headDist = movementScaling *
(float)((dotDistanceInMM / 2) / Math.Tan(angle)) / screenHeightinMM;


                float avgX = (firstPoint.x + secondPoint.x) / 2.0f;
                float avgY = (firstPoint.y + secondPoint.y) / 2.0f;


                //should  calaculate based on distance

                headX = (float)(movementScaling *
Math.Sin(radiansPerPixel * (avgX - 512)) * headDist);

                relativeVerticalAngle = (avgY - 384) *
radiansPerPixel;//relative angle to camera axis

                if(cameraIsAboveScreen)
                    headY = .5f+(float)(movementScaling *
Math.Sin(relativeVerticalAngle + cameraVerticaleAngle)  *headDist);
                else
                    headY = -.5f + (float)(movementScaling *
Math.Sin(relativeVerticalAngle + cameraVerticaleAngle) * headDist);
            }



            //position the graphical cursors at the 3rd and 4th ir
points if they exist
            if (wiiCursor1.isDown)
                wiiCursor1.setDown(wiimotePointsNormalized[0].x,
wiimotePointsNormalized[0].y);
            if (wiiCursor2.isDown)
                wiiCursor2.setDown(wiimotePointsNormalized[1].x,
wiimotePointsNormalized[1].y);
            if (wiiCursor3.isDown)
                wiiCursor3.setDown(wiimotePointsNormalized[2].x,
wiimotePointsNormalized[2].y);
            if (wiiCursor4.isDown)
                wiiCursor4.setDown(wiimotePointsNormalized[3].x,
wiimotePointsNormalized[3].y);


            wiiCursor1.wasDown = wiiCursor1.isDown;
            wiiCursor2.wasDown = wiiCursor2.isDown;
            wiiCursor3.wasDown = wiiCursor3.isDown;
            wiiCursor4.wasDown = wiiCursor4.isDown;

        }
```

```
            public void OnResetDevice(object sender, EventArgs e)
            {
                  Device dev = (Device)sender;
                  // Turn off culling, so we see the front and back
of the triangle
                  dev.RenderState.CullMode = Cull.None;
                  // Turn off D3D lighting
                  dev.RenderState.Lighting = false;
                  // Turn on the ZBuffer
                  dev.RenderState.ZBufferEnable = true;
        }
            private void SetupMatrices()
            {

            device.Transform.World = Matrix.Identity;

                  // Set up our view matrix. A view matrix can be
defined given an eye point,
                  // a point to lookat, and a direction for which
way is up. Here, we set the
                  // eye five units back along the z-axis and up
three units, look at the
                  // origin, and define "up" to be in the y-
direction.
            //device.Transform.View = Matrix.LookAtLH(new
Vector3(mouseCursor.X, mouseCursor.Y, -5.0f), new Vector3(0.0f,
0.0f, 0.0f), new Vector3(0.0f, 1.0f, 0.0f));
            device.Transform.View = Matrix.LookAtLH(new
Vector3(headX, headY, headDist), new Vector3(headX, headY, 0), new
Vector3(0.0f, 1.0f, 0.0f));

                  // For the projection matrix, we set up a
perspective transform (which
                  // transforms geometry from 3D view space to 2D
viewport space, with
                  // a perspective divide making objects smaller in
the distance). To build
                  // a perpsective transform, we need the field of
view (1/4 pi is common),
                  // the aspect ratio, and the near and far clipping
planes (which define at
                  // what distances geometry should be no longer be
rendered).

            //compute the near plane so that the camera stays fixed
to -.5f*screenAspect, .5f*screenAspect, -.5f,.5f
            //compting a closer plane rather than simply specifying
xmin,xmax,ymin,ymax allows things to float in front of the display
            float nearPlane = .05f;
            device.Transform.Projection =
Matrix.PerspectiveOffCenterLH(    nearPlane*(-.5f * screenAspect +
headX)/headDist,
```

```
nearPlane*(.5f * screenAspect + headX)/headDist,

nearPlane*(-.5f - headY)/headDist,

nearPlane*(.5f - headY)/headDist,

nearPlane, 100);

        }

            public void Render()
            {
            if (isRendering)
                return;
            isRendering = true;

            if (device == null)
                return;
                  //Clear the backbuffer to a blue color
                  device.Clear(ClearFlags.Target |
ClearFlags.ZBuffer, System.Drawing.Color.Black, 1.0f, 0);
                  //Begin the scene
                  device.BeginScene();
                  // Setup the world, view, and projection matrices

            SetupMatrices();

            device.RenderState.FogColor = Color.Black;
            device.RenderState.FogStart = headDist;
            device.RenderState.FogEnd   = headDist+fogdepth;
            device.RenderState.FogVertexMode = FogMode.Linear;
            device.RenderState.FogEnable = true;

            if (doWiimote)
            {
                if (doWiiCursors)
                {
                    //draw the cursors
                    /*
                    if (wiiCursor1.isDown)
                        wiiCursor1.Render(device);
                    if (wiiCursor2.isDown)
                        wiiCursor2.Render(device);
                    if (wiiCursor3.isDown)
                        wiiCursor3.Render(device);
                    if (wiiCursor4.isDown)
                        wiiCursor4.Render(device);
                */
                 }
                device.Transform.World = worldTransform;
            }

            if (showGrid)
```

```
            {
                device.TextureState[0].ColorOperation =
TextureOperation.Disable;

                device.RenderState.AlphaBlendEnable = false;
                device.RenderState.AlphaTestEnable = false;

                //back
                device.Transform.World = Matrix.Translation(new
Vector3(-.5f, -.5f, -1*boxdepth/2));
                device.Transform.World *= Matrix.Scaling(new
Vector3(screenAspect, 1, 1));
                device.SetStreamSource(0, gridBuffer, 0);
                device.VertexFormat =
CustomVertex.PositionColored.Format;
                device.DrawPrimitives(PrimitiveType.LineList, 0, 2 *
(numGridlines + 2));

                //left and right
                device.Transform.World = Matrix.Translation(new
Vector3(-.5f, -.5f, 0));
                device.Transform.World *= Matrix.Scaling(new
Vector3(1 * boxdepth / 2, 1, 1));
                device.Transform.World *=
Matrix.RotationY((float)(Math.PI / 2));
                device.Transform.World *= Matrix.Translation(new
Vector3(0.5f * screenAspect, 0, -.5f*boxdepth/2));
                device.DrawPrimitives(PrimitiveType.LineList, 0, 2 *
(numGridlines + 2));
                device.Transform.World *= Matrix.Translation(new
Vector3(-1.0f * screenAspect, 0, 0));
                device.DrawPrimitives(PrimitiveType.LineList, 0, 2 *
(numGridlines + 2));


                //floor and ceiling
                device.Transform.World = Matrix.Translation(new
Vector3(-.5f, -.5f, 0));
                device.Transform.World *= Matrix.Scaling(new
Vector3(screenAspect, 1 * boxdepth / 2, 1));
                device.Transform.World *=
Matrix.RotationX((float)(Math.PI / 2));
                device.Transform.World *= Matrix.Translation(new
Vector3(0, 0.5f, -.5f*boxdepth/2));
                device.DrawPrimitives(PrimitiveType.LineList, 0, 2 *
(numGridlines + 2));
                device.Transform.World *= Matrix.Translation(new
Vector3(0, -1.0f, 0));
                device.DrawPrimitives(PrimitiveType.LineList, 0, 2 *
(numGridlines + 2));
            }

            if (showTargets)//draw targets
            {
```

```
                device.SetTexture(0, textures[0]);

                //Render States
                device.RenderState.AlphaBlendEnable = true;
                device.RenderState.AlphaFunction = Compare.Greater;
                device.RenderState.AlphaTestEnable = true;
                device.RenderState.DestinationBlend =
Blend.InvSourceAlpha;
                device.RenderState.SourceBlend = Blend.SourceAlpha;
                device.RenderState.DiffuseMaterialSource =
ColorSource.Material;

                //Color blending ops
                device.TextureState[0].ColorOperation =
TextureOperation.Modulate;
                device.TextureState[0].ColorArgument1 =
TextureArgument.TextureColor;
                device.TextureState[0].ColorArgument2 =
TextureArgument.Diffuse;

                //set the first alpha stage to texture alpha
                device.TextureState[0].AlphaOperation =
TextureOperation.SelectArg1;
                device.TextureState[0].AlphaArgument1 =
TextureArgument.TextureColor;


                device.VertexFormat = Vertex.FVF_Flags;
                device.SetStreamSource(0, targetBuffer, 0);
                for (int i = 0; i < numTargets; i++)
                {
                    //load different images
                    device.SetTexture(0, textures[i]);
                    device.Transform.World =
Matrix.Scaling(targetSizes[i]);
                    device.Transform.World *=
Matrix.Translation(targetPositions[i]);

device.DrawPrimitives(PrimitiveType.TriangleStrip, 0, 2);

                }

            }

            if (showLines)
            {
                device.VertexFormat =
CustomVertex.PositionColored.Format;
                device.SetStreamSource(0, lineBuffer, 0);
                device.TextureState[0].ColorOperation =
TextureOperation.Disable;
                device.RenderState.AlphaBlendEnable = false;
                device.RenderState.AlphaTestEnable = false;
                for (int i = 0; i < numTargets; i++)
```

```
                {
                    device.Transform.World =
Matrix.Scaling(targetSizes[i]);
                    device.Transform.World *=
Matrix.Translation(targetPositions[i]);
                    device.DrawPrimitives(PrimitiveType.LineList, 0,
1);
                }
            }


            if (showBackground)
            {
                device.RenderState.FogEnable = false;
                device.Transform.World = Matrix.Scaling(new
Vector3(3,2,3));
                device.SetTexture(0, backgroundtexture);
                //Render States
                device.RenderState.AlphaBlendEnable = false;
                device.RenderState.AlphaTestEnable = false;
                device.TextureState[0].ColorOperation =
TextureOperation.Modulate;


                device.VertexFormat =
CustomVertex.PositionTextured.Format;

                device.SetStreamSource(0, backgroundBuffer, 0);
                device.DrawPrimitives(PrimitiveType.TriangleStrip,
0, (backgroundStepCount)*2);
            }

            if (showMouseCursor)
            {
                device.TextureState[0].ColorOperation =
TextureOperation.Disable;
                device.RenderState.AlphaBlendEnable = false;
                device.RenderState.AlphaTestEnable = false;

                device.Transform.World = Matrix.Identity;
                mouseCursor.Render(device);
            }

            //display help menu
            if (showHelp)
                RenderText(helpColor);

            //End the scene
                  device.EndScene();

                  // Update the screen
                  device.Present();
            isRendering = false;
            }
```

```
            protected override void
OnPaint(System.Windows.Forms.PaintEventArgs e)
            {
            isReady = true;//rendering triggered by wiimote is
waiting for this.
            }
        protected override void OnResize(System.EventArgs e)
        {
        }

        /// <summary>
            /// The main entry point for the application.
            /// </summary>
            static void Main()
            {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            //system initialization occurs in GUI creation
            Application.Run(new WiiDesktopGUI());
            }
      }
}
```

WiiDesktopGUI.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;


namespace WiiDesktopVR
{
    public partial class WiiDesktopGUI : Form
    {
        public bool bgCheckBox;
        public String bgFile;
        public float[] targetScale;
        public bool start;
        public int numTargets;
        public String[] pics;
        public Vector3[] locations;

        public WiiDesktopGUI()
        {
            //Initialize GUI
            InitializeComponent();

            //initialize GUI variables in case user does not they
will be set
            this.bgCheckBox = false;
            bgFile = "images/stad_2.png";
            start = false;
            numTargets = 6;
            pics = new String[] { "images/target1.png",
"images/target2.png",
            "images/target3.png", "images/target4.png",
"images/target5.png", "images/target6.png" };
            locations = new Vector3[numTargets];
            targetScale = new float[] { .065f, .065f, .065f, .065f,
.065f, .065f };

            backgroundbox.SizeMode =
PictureBoxSizeMode.StretchImage;
            pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
            pictureBox2.SizeMode = PictureBoxSizeMode.StretchImage;
            pictureBox3.SizeMode = PictureBoxSizeMode.StretchImage;
            pictureBox4.SizeMode = PictureBoxSizeMode.StretchImage;
            pictureBox5.SizeMode = PictureBoxSizeMode.StretchImage;
            pictureBox6.SizeMode = PictureBoxSizeMode.StretchImage;
        }
```

```
        //determine background image
        private void backgroundText_TextChanged(object sender,
EventArgs e)
        {
            this.bgFile = "images/" + this.backgroundText.Text;
        }

        //START button
        private void button1_Click(object sender, EventArgs e)
        {
            using (WiiDesktopVR frm = new WiiDesktopVR(this))
            {
                if (!frm.InitializeGraphics()) // Initialize
Direct3D
                {
                    MessageBox.Show("Could not initialize Direct3D.
This tutorial will exit.");
                    return;
                }
                frm.Show();

                // While the form is still valid, render and process
messages
                while (frm.Created)
                {
                    Application.DoEvents();
                    if (!frm.doWiimote)
                        frm.Render();
                }
                Cursor.Show();
            }

        }

        //NUMTARGETS counter
        private void numericUpDown2_ValueChanged(object sender,
EventArgs e)
        {
            this.numTargets = (int)this.numericUpDown2.Value;
        }

        //determine images for foreground
        private void picBox1_TextChanged(object sender, EventArgs e)
        {
            this.pics[0] = "images/" + this.picBox1.Text;
        }

        private void picBox2_TextChanged(object sender, EventArgs e)
        {
            this.pics[1] = "images/" + this.picBox2.Text;
        }

        private void picBox3_TextChanged(object sender, EventArgs e)
```

```
        {
            this.pics[2] = "images/" + this.picBox3.Text;
        }

        private void picBox4_TextChanged(object sender, EventArgs e)
        {
            this.pics[3] = "images/" + this.picBox4.Text;
        }

        private void picBox5_TextChanged(object sender, EventArgs e)
        {
            this.pics[4] = "images/" + this.picBox5.Text;
        }

        private void picBox6_TextChanged(object sender, EventArgs e)
        {
            this.pics[5] = "images/" + this.picBox6.Text;
        }

        //determine x, y coordinates of each image
        private void numericUpDown1_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[0].X = (float)this.numericUpDown1.Value;
        }

        private void numericUpDown3_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[0].Y = (float)this.numericUpDown3.Value;
        }

        private void numericUpDown6_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[1].X = (float)this.numericUpDown6.Value;
        }

        private void numericUpDown7_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[1].Y = (float)this.numericUpDown7.Value;
        }

        private void numericUpDown4_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[2].X = (float)this.numericUpDown4.Value;
        }

        private void numericUpDown5_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[2].Y = (float)this.numericUpDown5.Value;
```

```
        }

        private void numericUpDown8_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[3].X = (float)this.numericUpDown8.Value;
        }

        private void numericUpDown9_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[3].Y = (float)this.numericUpDown9.Value;
        }

        private void numericUpDown10_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[4].X = (float)this.numericUpDown10.Value;
        }

        private void numericUpDown11_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[4].Y = (float)this.numericUpDown11.Value;
        }

        private void numericUpDown12_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[5].X = (float)this.numericUpDown12.Value;
        }

        private void numericUpDown13_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[5].Y = (float)this.numericUpDown13.Value;
        }

        //determine depths for each image
        private void numericUpDown14_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[0].Z = (float)this.numericUpDown14.Value;
        }

        private void numericUpDown15_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[1].Z = (float)this.numericUpDown15.Value;
        }

        private void numericUpDown16_ValueChanged(object sender,
EventArgs e)
        {
```

```
            this.locations[2].Z = (float)this.numericUpDown16.Value;
        }

        private void numericUpDown17_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[3].Z = (float)this.numericUpDown17.Value;
        }

        private void numericUpDown18_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[4].Z = (float)this.numericUpDown18.Value;
        }

        private void numericUpDown19_ValueChanged(object sender,
EventArgs e)
        {
            this.locations[5].Z = (float)this.numericUpDown19.Value;
        }

        //determine scale for each image
        private void numericUpDown20_ValueChanged(object sender,
EventArgs e)
        {
            this.targetScale[0] = 0.065f *
(float)this.numericUpDown20.Value;
        }

        private void numericUpDown21_ValueChanged(object sender,
EventArgs e)
        {
            this.targetScale[1] = 0.065f *
(float)this.numericUpDown21.Value;
        }

        private void numericUpDown22_ValueChanged(object sender,
EventArgs e)
        {
            this.targetScale[2] = 0.065f *
(float)this.numericUpDown22.Value;
        }

        private void numericUpDown23_ValueChanged(object sender,
EventArgs e)
        {
            this.targetScale[3] = 0.065f *
(float)this.numericUpDown23.Value;
        }

        private void numericUpDown24_ValueChanged(object sender,
EventArgs e)
        {
```

```
            this.targetScale[4] = 0.065f *
(float)this.numericUpDown24.Value;
        }

        private void numericUpDown25_ValueChanged(object sender,
EventArgs e)
        {
            this.targetScale[5] = 0.065f *
(float)this.numericUpDown25.Value;
        }

        //preview images in GUI, if image cannot be located, do not
crash the system
        //but let the user know which file is incorrectly determined
        private void button2_Click(object sender, EventArgs e)
        {
            System.IO.Directory.SetCurrentDirectory(
                    System.Windows.Forms.Application.StartupPath +
@"\..\..\");
            try
            {

                this.backgroundbox.Load(this.bgFile);
            }
            catch
            {
                MessageBox.Show("Cannot find background file");
                return;
            }

            try
            {
                this.pictureBox1.Load(this.pics[0]);
            }

            catch
            {
                MessageBox.Show("Cannot find picture 1 file");
                return;
            }

            if (numTargets > 1)
            {
                try
                {
                    this.pictureBox2.Load(this.pics[1]);
                }

                catch
                {
                    MessageBox.Show("Cannot find picture 2 file");
                    return;
                }
                if (numTargets > 2)
```

```
                {
                    try
                    {
                        this.pictureBox3.Load(this.pics[2]);
                    }

                    catch
                    {
                        MessageBox.Show("Cannot find picture 3
file");
                        return;
                    }
                    if (numTargets > 3)
                    {
                        try
                        {
                            this.pictureBox4.Load(this.pics[3]);
                        }

                        catch
                        {
                            MessageBox.Show("Cannot find picture 4
file");
                            return;
                        }

                        if (numTargets > 4)
                        {
                            try
                            {
                                this.pictureBox5.Load(this.pics[4]);
                            }

                            catch
                            {
                                MessageBox.Show("Cannot find picture
5 file");
                                return;
                            }
                            if (numTargets > 5)
                            {
                                try
                                {

this.pictureBox6.Load(this.pics[5]);
                                }

                                catch
                                {
                                    MessageBox.Show("Cannot find
picture 6 file");
                                    return;
                                }
                            }
```

```
                                    }
                                }
                            }
                        }
                    }
                }
            }
```