3D Desktop Environment which Implements Headtracking via the Wii Controller

Tsung-Jui Hsieh

A dissertation submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree of Bachelor of Science in Engineering.

Cape Town, October 2008

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Bachelor of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author .....

Cape Town

21 October 2008

Abstract

Nintendo has recently developed the Wii products which dramatically enhances gaming experience. It allows the user to control objects on the screen by relating it to the physical motion experienced by the controller.

People found that by using the infrared camera in the Wii remote and a head mounted sensor bar (two IR LEDs), they can track the location of their head. The image in the screen can then be changed by moving it relative to the head's position. This makes the screen appear to be a window to a different dimension.

In this thesis, a 3D environment was created to test the headtracking capabilities of the wiimote. The environment is a 3D version of a normal desktop, with the shortcut icons replaced by picture billboards, rotating spheres object files. It is a Netbeans project and uses java as the base programing language. The environment is rendered by using the JOGL package in java. Through testing by follow peers, it has been concluded that headtracking is good for viewing objects up close. For objects located in the distance, normal viewing techniques is sufficient.

With thanks to my mother for supporting me all the time

Kan-Hsing Hsieh

Acknowledgements

With many thanks to:

Professor Mike Inggs, my supervisor, for the guidance he has provided.

Sabastion Wyngard from CHPC, for helping me with my coding and for letting me use his equipment.

Victor Kirov who helped me figure out the WiiUseJ API.

Everyone in the radar labs who helped me finish my writeup

Contents

List of Figures

Nomenclature

API—A set of functions, procedures, methods or classes that an operating system, library or service provides to support requests made by computer programs

IR—This stands for infra-red

JOGL—An accronum for Java OpenGL. It is a wrapper library that allows OpenGL to be used in the Java programming language. OpenGL (Open Graphics Library) is an API used to produce 2D and 3D computer graphics.

LED—A light emitting diode. This is used as a small source of light.

Netbeans—Fully-featured Java IDE (Integrated Development Environment) written completely in Java.

TourCanvasGL—A modified version of TourGL.

TourGL—A Netbeans project written in java language and uses java's OpenGL package. It is a basic 3D game which this project is based on.

Wiimote—A shorter version of saying the wii controller

Chapter 1

Introduction

1.1 Background

Nintendo has recently developed the Wii products which dramatically enhances gaming experience. It allows the user to control objects on the screen by relating it to the physical motion experienced by the controller.

The Wii package includes the wii console, wii remote (wiimote), wii sensor bar, Nunchuck and accessories such as the wii fit, wii guitar hero etc.

Recently, people found that by using the infrared camera in the Wii remote and a head mounted sensor bar (two IR LEDs), they can track the location of their head. The image in the screen can then be changed by moving it relative to the head's position. This makes the screen appear to be a window to a different dimension.

1.2 Project Objectives

The objectives of this project is to:

1. Design a 3D environment in which to test the wiimote. In this project the 3D environment will be a 3D desktop with most of the capabilities of a normal desktop. It will be a Netbeans project and will use java as its basic programing language. The environment will be rendered using the JOGL package in java.
2. To design interactive components in the 3D desktop.
3. To modify the wiimote signals so that headtracking is possible.
4. To test the designed functions of the wiimote.
5. To write a detailed report on the project and submit it by 21 October 2008.

1.3 Scope and Limitations of Project

This project is limited to the design of a 3D desktop world which implements headtracking via the Wiimote. The time limit for this project is 13 weeks.

The users' requirement does not restrict the design to any cost budget, however as an undergraduate project it is limited by the given R1000.

1.4 Plan of Development

The following is the chronological order of events of the development of the project.

Chapter 1 - Introduction - This chapter is a brief summary of what the project is about. This includes the background, project objectives, scope and limitations and plan of development.

Chapter 2 - TourGL - Here will be an overview of TourGL, the base 3D model which was used to generate the final 3D desktop world.

Chapter 3 - Keyboard Controls - This section documents the keyboard controls for the 3D world and how they were implemented.

Chapter 4 - Mouse Controls - Here the mouse controls which were added into the 3D desktop are discussed.

Chapter 5 - 3D Desktop Properties - The different abilities programmed into the 3D world are explained in this chapter.

Chapter 6 - Collision Detection - Here the implementation of collision detection for objects in the 3D environment is discussed.

Chapter 7 - Headtracking - This chapter explains how the wiimote interface is implemented into the 3D desktop, along with how headtracking is applied.

Chapter 8 - Tests and Results - Here the functions of the 3D desktop and the headtracking capabilities were tested by varius users and the results collected.

Chapter 9 - Conclusion - This chapter states what has been found in this project.

Chapter 10 - Future Work - Possible improvements and modifications to the 3D world are mentioned here. This includes suggestions for headtracking functions.

Chapter 2

TourGL

Tour GL is a sample program written by Andrew Davison. It is a Netbeans project and is based in java, using the JOGL package. The java program generates a 3D world. The visual elements in the world are:

- A green and blue checkerboard floor with a red square in the center, and numbers along its x- and z-axis.
- A skybox of stars.
- An orbiting earth (a textured sphere).
- A billboard showing a tree, which rotates along its y-axis so that it always faces the camera
- Five red R's placed randomly on the floor.

The user can move around the world in the follow directions:

1. forward (up arrow)
2. backwards (down arrow)
3. translate left (control + left arrow)
4. translate right (control + right arrow)
5. go up (page up)
6. go down (page down)
7. turn left (left arrow)
8. turn right (right arrow)

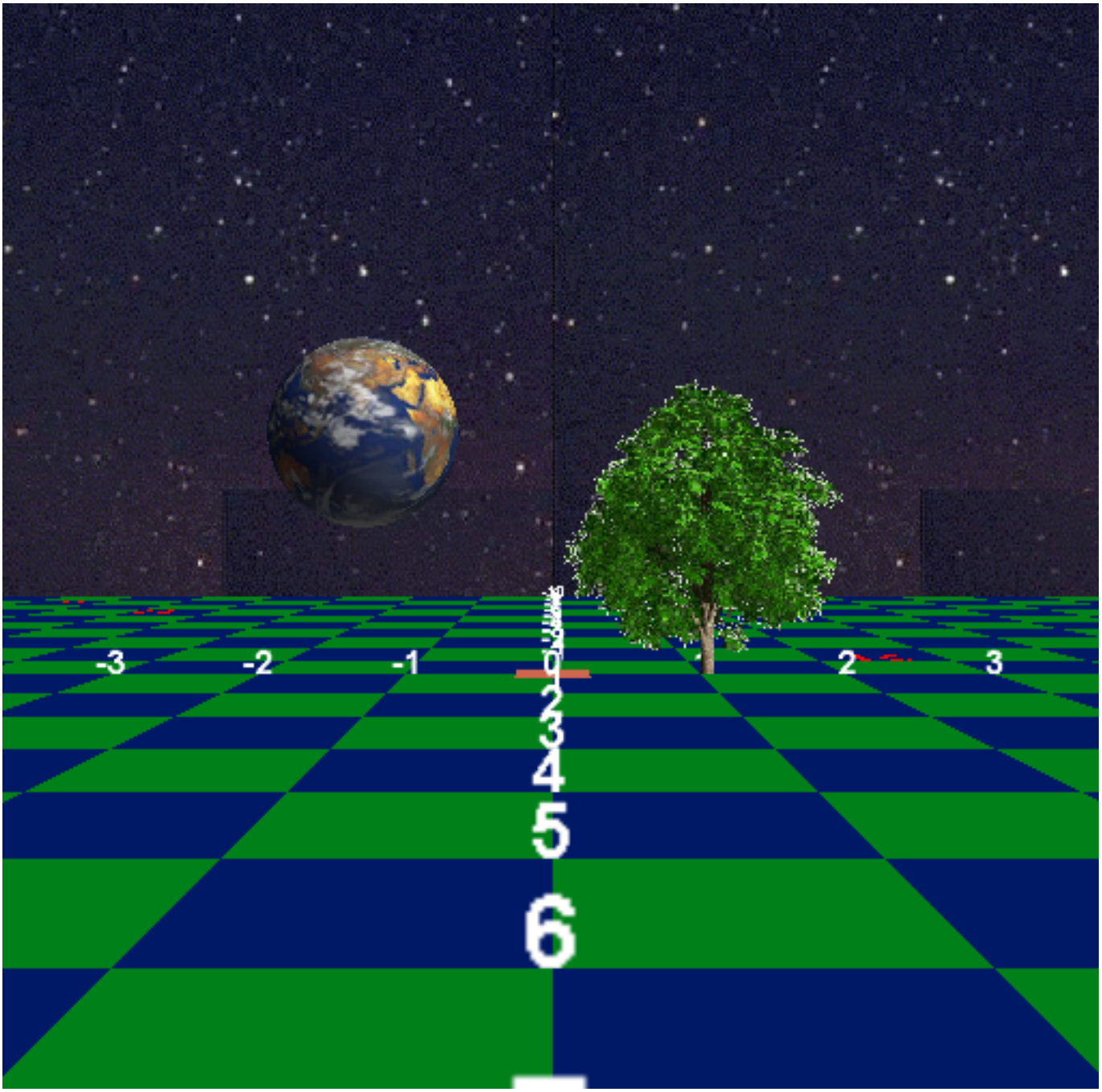

Figure 2.1: The TourGL Application

The player can not walk off the checkerboard or beyond the borders of the skybox. The game is terminated by pressing:

1. ESC
2. Q
3. END
4. control + Q
5. Using the mouse to click the close box.

TourGL is a very basic game – the user must navigate over the red R's lying on the ground to make them disappear. The game ends when all the R's have been deleted, and a score is calculated based on how long it took to delete the R's. The game automatically pauses when the window is iconified or deactivated, and can be resized.

There is another version of TourGL called TourModelsGL. This program has the basic functions of TourGL, but where there are planar images and spheres in TourGL, TourModelsGL has objects. TourModelsGL also has an addFog() method which adds fog to the 3D world.

This project will use TourGL and TourModelsGL as the base code to make a new 3D desktop environment where the user can interact with the various object in the 3D world. In order to navigate in the world, the keyboard and mouse will be used along with the wiimote which provides headtracking.

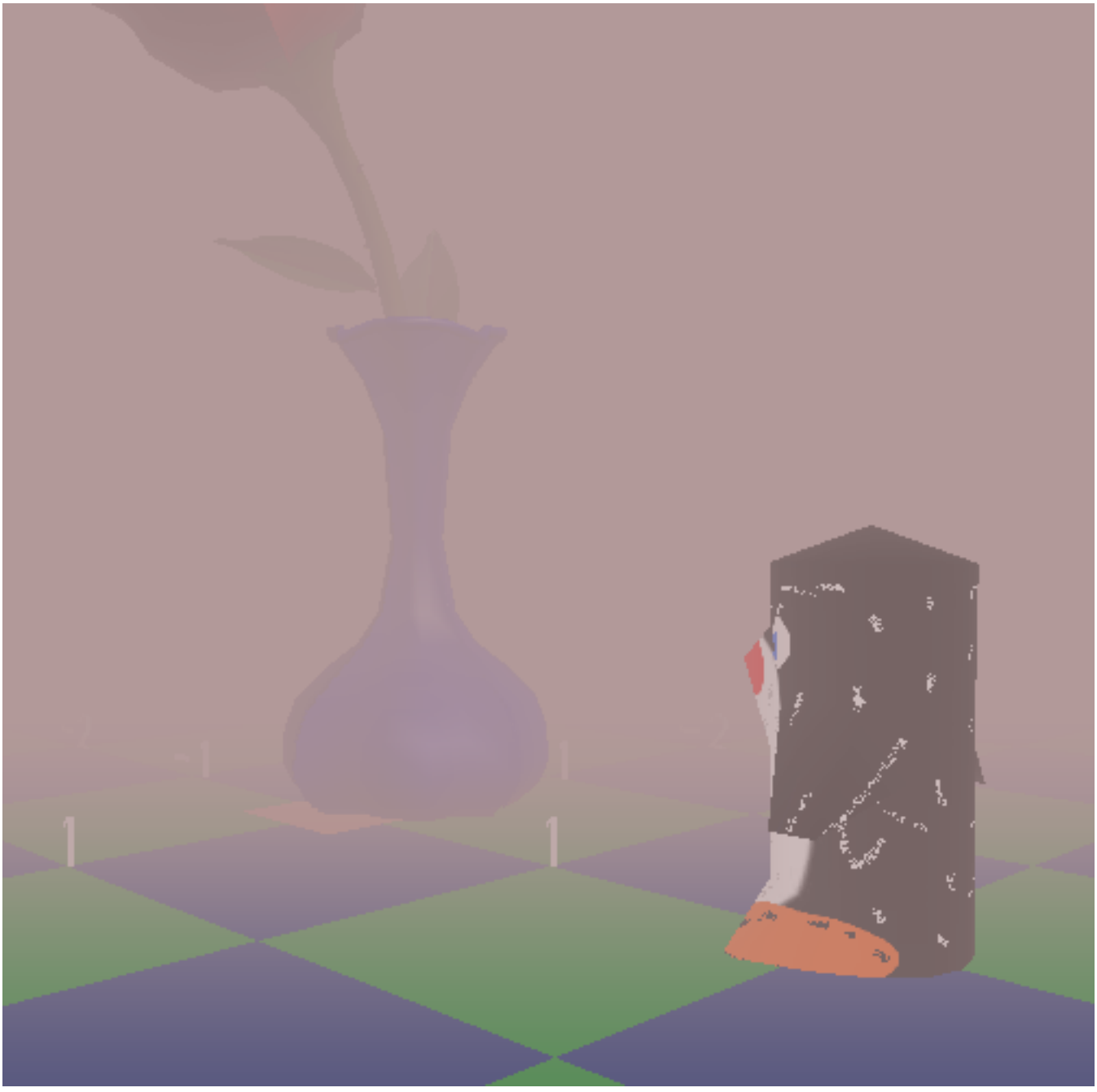

Figure 2.2: The TourModelsGL Application

Chapter 3

Keyboard Controls

3.1 Basic TourGL Motion

In the TourGL java applet, motion in the 3D world is generated by pressing the up arrow, down arrow, left arrow, right arrow, page up and page down. These motions are sufficient for most applications, however:

- The user can only move in one direction at a time. If he/she presses 2 motion keys simultaneously, only one motion key will be registered.
- The motion is jagged, more like skipping than walking.
- If the user holds a motion key down, he/she will take one step forward, lag a bit, then continue walking normally

The following prototypes will attempt to remove these defects.

3.1.1 Prototype 1: Motion in two directions

Here the program tries to make the applet read in two motion keys and process them simultaneously. To do this, the following example code is added:

```
if (keyCode == KeyEvent.VK_UP) { // move forward
    ...
    If (keyCode == KeyEvent.VK_LEFT) { // turn left
        ...
    }
}
```

This did not work since Windows only reads in one keyboard hit at a time. Even if two keyboard keys are pressed simultaneously, only one key will be received by the program.

3.1.2 Prototype 2: Using e.isControlDown()

In TouGL, there is a section in its source code which accepts two keyboard presses simultaneously. This is where the program quits if Ctrl and C are pressed simultaneously. It is a typical function of Windows where if the user presses Ctrl, Alt or Shift along with another keyboard key, both keys will be registered. By using this property of Windows, the program can be modified to accept two key presses simultaneously. However this makes the motion controls very hard to use and irrational. In order to go diagonally forward and left, the user would be required to press the up arrow and Alt instead of up arrow and left arrow.

3.1.3 Prototype 3: Smooth walking

Here the program tries to make the walking motion smoother by removing the lag between the initial step and the following steps. To do this, the code within the walking statement is changed i.e.:

If (keyCode == KeyEvent.VK_UP) { // go forward

(Changed the code here)

}

This did not work because due to how Microsoft Windows functions. In Windows if the user presses a key continuously, he/she will get one initial character followed by a pause, after which the character will be printed continuously. This means changing the code within the walking statement will not remove the lag between steps.

3.1.4 Prototype 4: Using Booleans

This is the code which was used in the 3D desktop application. Here the program initialises a Boolean value for each motion key as false. Then for each key the program checkes if it was pressed. If a specific motion key is pressed, the Boolean value for the key will be changed to true. When the key is released, the Boolean value will be changed to false.

Elsewhere in the code a method was made for each movement i.e. forward(), back(), left(), right(), etc.... Then in the processKey() method the code was adjusted so that if the Boolean value of the motion key pressed is true, then the corresponding process for that key will occur. This way the program was able to make the motion in the 3D world smooth and continuous. It got rid of the irritating lag Windows has when the user press a key continuously. The user can also press two keys simultaneously and the program will register them.

Now that the java applet can process two or more keys simultaneously, conditions on how these keys react will have to be modified. Figure 3.1 represents what I have done. An example would be:

If the up arrow is pressed and the W key is pressed, then the user will move forward and strafe left at the same time.

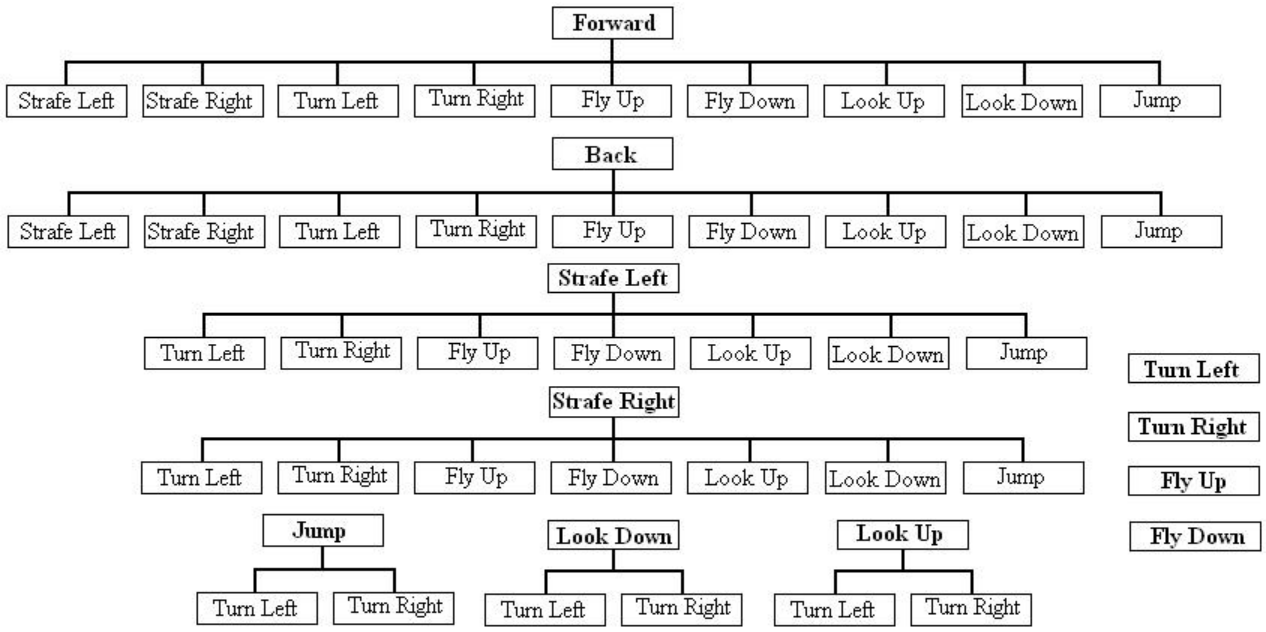

Figure 3.1: Keyboard Movement Controls

3.2 Jumping

Here the program initialises a new Boolean value as false and used it to check whether the space key was pressed. If the space key is pressed, the Boolean value for jump is true. Note that the Boolean value is not changed to false when the space key is released. If the Boolean value for jump is true, then a method called jump() will be called and executed in the processKey() method.

In order to make the jump() method the program initialises two variables called jumpv (the jumping velocity) as 0.2 and acc (the gravitational force) as 0.01. In the jump() method I had jumpv decremented by acc and then changed the y-axis camera position of the viewer by jumpv for every iteration. When the y-axis camera position becomes less or equal to 1, the y-axis camera position is set to 1, jumpv is reset to 0.2 and the Boolean value for jump is set to false. This will created a jumping effect in the 3D world.

3.3 Using Items

Normal computer have icons on their desktop which can be activated. They usually function as shortcuts to programs or other places in the hard drives. In this program the icons are replaced with image billboards and objects. When the user is within a boundary of 1.5 radius from the object surface, he/she must be able to activate the object by pressing the E keyboard key.

To implement this function in the 3D world, the program initialises a new boolean value useKey as false for key pressesand a integer variable useValue as 0. When the user is close to the icon he/she

wishes to use, the user presses the E keyboard key. The program registers that the key is pressed by setting useKey to true. It then check whether the user's location is close to an usable icon. If he/she is within range, the useValue is changed to the integer value of the icon and activates the the icon associated with that value. Once the icon is activated, the useValue will be reset to 0. This prevents the icon from activating multiple times.

3.4 Final keyboard controls

Some keys have been modified from the basic TourGL applet and others where added as indicated in the previous sections. Below is a list of all the keyboard controls for the 3D desktop.

- Forward: W
- Back: S
- Move Left: A
- Move Right: D
- Look Up: Up Arrow
- Look Down: Down Arrow
- Look Left: Left Arrow
- Look Right: Right Arrow
- Go Up: Page Up
- Go Down: Page Down
- Jump: Space
- Use Item: E
- Close Applet: Esc, End, Q, Ctrl + C

Chapter 4

Mouse Controls

In java, mouse events tells the user whether the mouse is being used to interact with an object. These events occur when the user presses or releases one of the mouse buttons. Java also allows the user to implement code to track the mouse cursor's motion on the screen, as well as mouse wheel events.

4.1 Viewing with your mouse

In TourGL, looking around in the 3D world is done by adjusting the view with the programmed keyboard keys. This is not always favourable since the accuracy is low and controls hard to use. One way of improving the ease with which the environment is viewed is to use mouse motion tracking. This is often done in first person role playing games.

4.1.1 Mouse Location

To use the mouse to change the viewpoint, track the location of the mouse on the screen. This can be done by using java's MouseMotionListener. If the mouse location exceeds a certain boundary, turn the camera in that direction. Below are 3 different ways to set the boundaries.

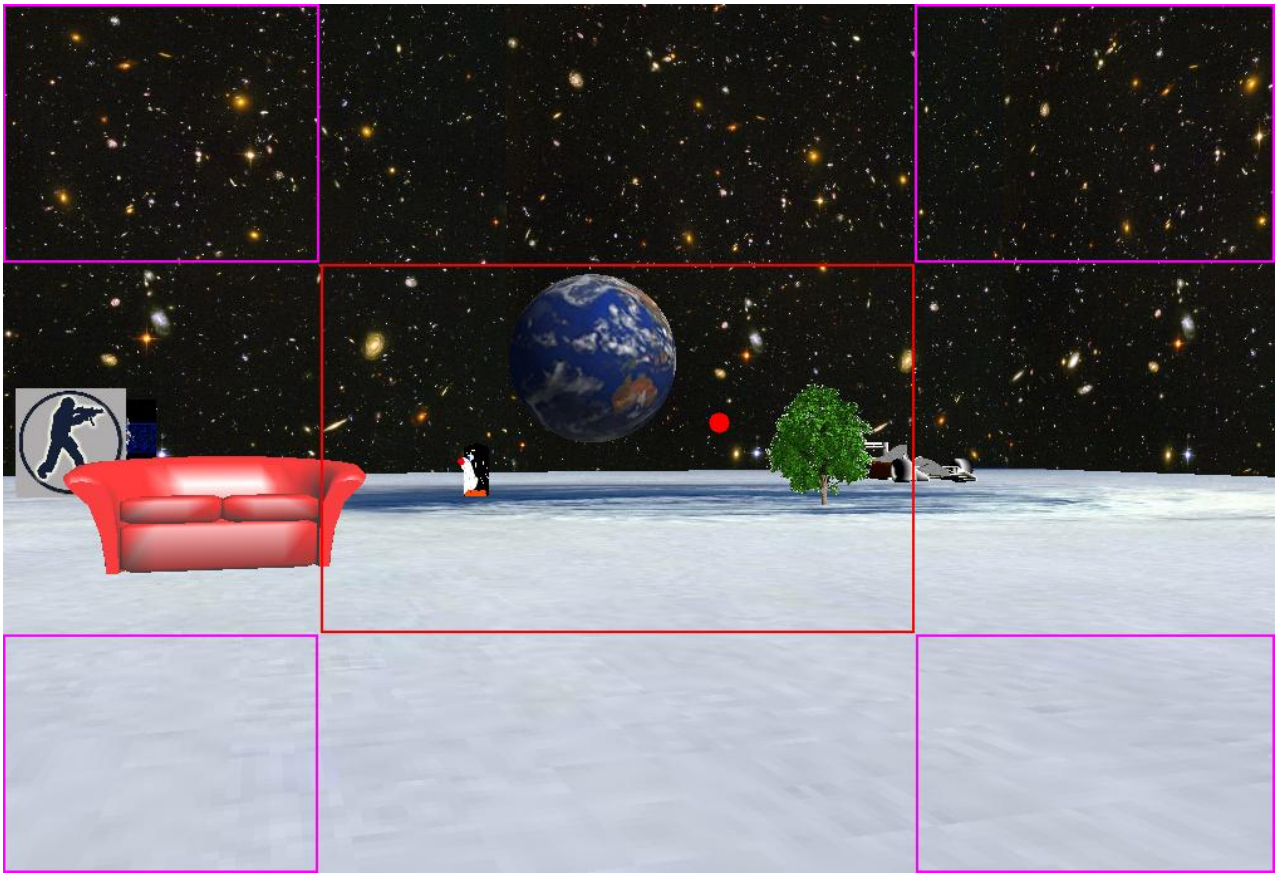

Figure 4.1: Loose Middle Boundary

The red box in figure 4.1 indicates the area where the mouse cursor can move freely. Outside of this boundary, the image will rotate in the direction of the mouse cursor's location. In the actual program the boundary is not visible. The red dot in the image represents the current mouse cursor location. Here the boundary is located roughly at the center of the picture. When the mouse crosses the boundary, the viewpoint will turn in that direction. This gives the user easier control over the view then via the keyboard but is still undesireable because:

1. The user is not granted full use of the image. When the mouse cursor can only move freely within the boundary or else the image will rotate.

2. If the user tries to minimise or exit the program using the mouse, the image will rotate because the mouse cursor is outside the boundary.

3. To stop the image rotating, the mouse cursor has to be moved back within the boundary.

4. In order to rotate the viewpoint both vertically and diagonally, the mouse cursor location has to be inside the pink boxes in the figure4.1.

Figure 4.2: Boundary at Edge

In figure 4.2, the boundary is located close to the edge of the image. This allows the mouse cursor more space to move around. However the above 4 disadvantages were not neutralised, only minimised. Also this method creates a new problem. If the mouse cursor moves too fast, going past the boundary to outside of the program frame without the program detecting the mouse event, the image will not rotate.

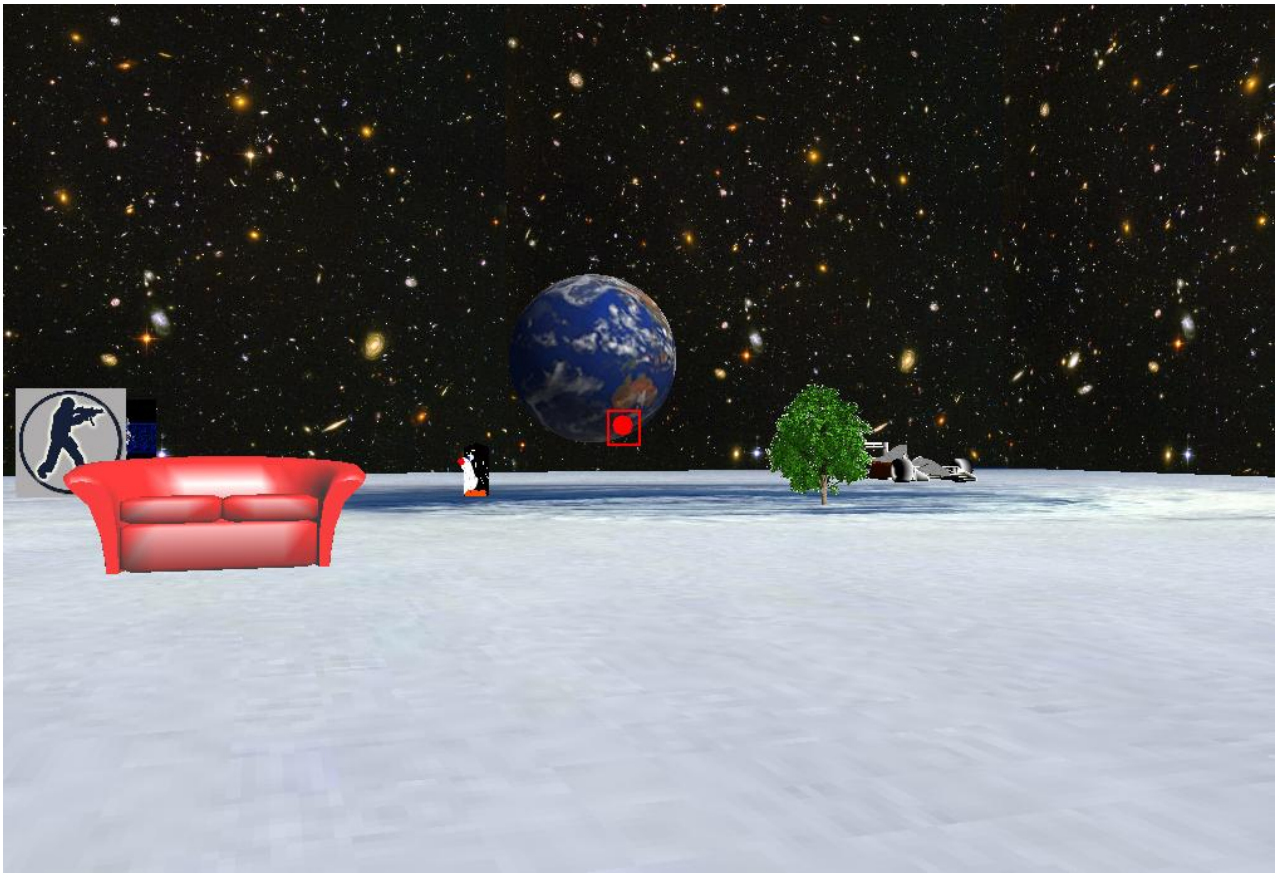

Figure 4.3: Boundary at Exact Centre

In figure 4.3, the boundary is set at the exact centre of the screen. This way any medium to large movement by the mouse will rotate the image, but it has many defects. The user cannot move the mouse around alot since any motion outside the boundary will cause the image to rotate. Once outside the boundary, the user must return the mouse cursor to within the small box in order to stop the rotating viewpoint. Since the red box in the above image is not visible when running the program, stopping the rotation will be fairly difficult.

By using the java.awt.Robot.* package, the applet is able to automatically change the location of the mouse cursor. If the mouse cursor is set to the centre of the viewpoint in every itteration of the program, the above boundary method becomes theoretically viable. However after testing it was found that the image jerks when rotating. This is because in the first iteration of the program after the mouse moved, the mouse cursor location is outside the boundary and rotates the image. Then in the next iteration, the mouse cursor is within the boundary again and the image stops rotating. This repeats itself and the image to jerk when rotating.

4.1.2 Mouse Displacement

Another method of using the mouse cursor location to change the viewpoint is to make the image rotate in the direction of the displacement of the mouse.

First set the mouse cursor location to the exact center of the viewpoint by using java's Robot function. Then get the next mouse cursor location. Subtract the initial mouse position from the final one. That would be the displacement of your mouse cursor. Use the displacement to turn the camera either vertical, horizontally or both. The magnitude of the displacement is the rotating speed.

This method proved to work the best and is used in the 3D Desktop application. It has many advantages from the previous version, namely:

1. The user's camera position follows your mouse cursor exactly and does not need to return to its origional position.
2. The mouse cursor will never go beyond the application's frame.
3. The camera motion can be finely controlled while still providing fast rotation speed when needed.
4. It is intuitively easy to use. Most first person shoots work in a similar fashion.

The only downside of using this method is that the mouse cursor will be entirely devoted to controling the camera position.

The mouse cursor image can be removed by using the following code:

```
Image cursorImage = Toolkit.getDefaultToolkit().getImage("xparent.gif");
Cursor blankCursor = Toolkit.getDefaultToolkit().createCustomCursor(cursorImage, new Point(0, 0), "");
setCursor(blankCursor);
```

4.2 Using the Mouse Button

This is similar to using the E keyboard key. First use java's MouseListener to make the program react to mouse clicks. If the mouse button has been clicked, check the users current location. If the user is within 1.5 units away from the surface of an usable icon, activate the icon.

Chapter 5

Desktop Graphics

In all normal desktops there are icons which allow the user to open programs or jump to a specific folder in the hard drive. In this 3D world, the icons in the normal desktops are replaced with picture billboards, rotating spheres and 3D images with a .obj extention. These shortcuts in the 3D desktop are activated by either pressing the E keyboard key or clicking the primary button on the mouse. The shortcuts will only activate if the users current position is within 1.5 units radius from the surface of the icon.

5.1 Picture Billboards

The picture billboards in the 3D desktop are used as links to executable files and directories in the hard drive. They are simple planar images with either .jpg or .bmp or other similar image extentions.

In order to make these billbenter the path hereoards, first load the image required want as a texture graphic. After that created a drawImage() method. In the method, set the location of the image by using gl.glTranslatef(). After this set the size of the image by setting it's verticies. The verticies are arranged as shown in figure 5.1. Using these verticies and the textured graphic, the image can then be drawn by using the drawScreen() method.

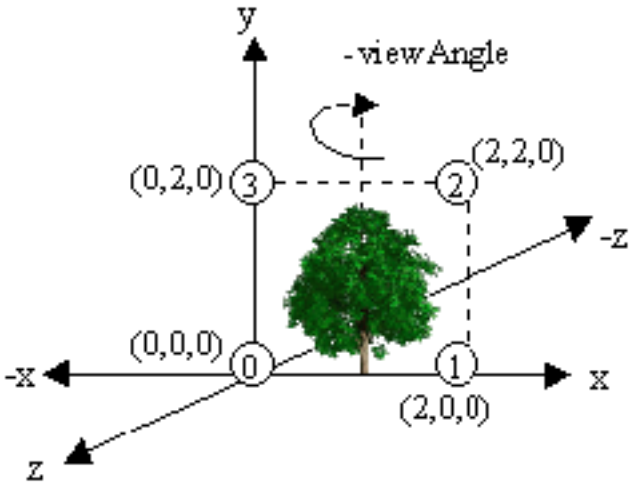

Figure 5.1: The Tree Billboard

In order to create the billboard effect, use gl.glRotatef() and the current camera angle. First set the image to be facing the user when directly in front of him/her, then rotate the image in the opposite direction to where the user is turning.

To link the image billboard to a program or directory, use the following code:

```
Runtime variable = Runtime.getRuntime();
try{
    variable.exec("C:\\Garmin\\MapSource.exe");
} catch(Exception e){
    e.printStackTrace();
}
```

Simply change the path name in order to change the program or directory opened.

5.2 Spheres

To draw a sphere in JOGL, use the code that is given in TourGL's drawSphere() method. In order to make a new sphere, copy the drawSphere() method and change the image texture to the one required. To change how the sphere's surface responds to light, use the gl.glMaterialfv() and gl.glMateriali() methods. After that rotate and translate the sphere by using gl.glTranslatef() and gl.glRotatef(). To set the sphere spinning, increment the angle of rotation in gl.glRotatef() every time the program updates itself.

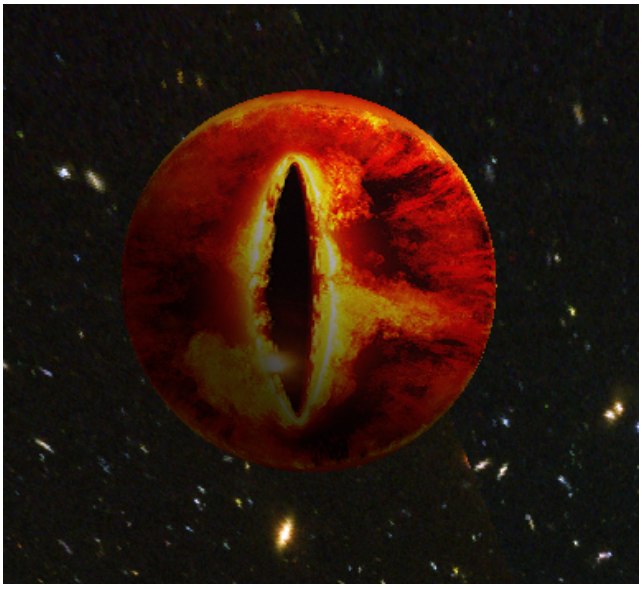

Figure 5.2: Eye from Lord of the Rings

5.3 Objects

The objects in this 3D desktop came from TourModelsGL, an adapted version of TourGL. The objects are randomly placed in the 3D environment and are used later in the headtracking section.

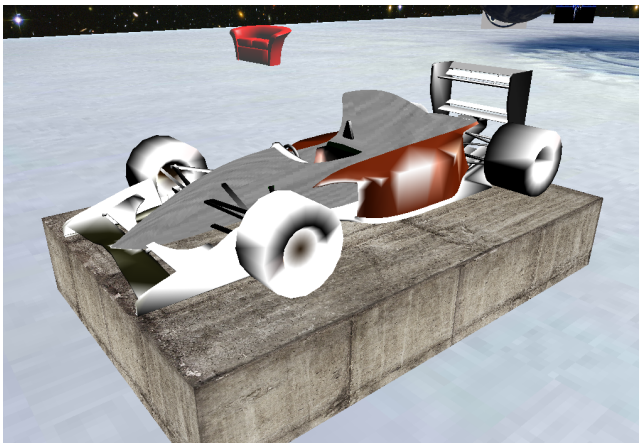

Figure 5.3: Car Object

Chapter 6

Collision Detection

6.1 Skydome and Floor

In TourGL, collision detection for the floor and the skydome was implemented. However the program did not account for the focal length of the camera. The camera can view 1 unit ahead of the user's current location. This means then when the user reaches the edges of the 3D desktop, he/she will be able to 'see' through the walls. This applies to the ceilings and floor as well.

In order to remove this defect, set the boundaries of the moveable area to be 1 unit before the actual edges of the 3D desktop. This way when the user hits a boundary, he/she will not be able to move past it and can not 'see' past the wall.

6.2 Rectangles

In the 3D desktop there are many icons located on the floor. When the user walks into them, nothing happens. In order to correct this, a rectangular or circular collision detection boundary can be erected. Here we will talk about rectangular collision detection.

Most of the icons in the 3D desktop will be located somewhere in the x, z-plane. They will most likely be away from the edges of the 3D environment as well. Therefore in order to do collision detection, an area around the object must be set in the x, z-plane where the user can not enter. This is shown in figure 6.1.

To set up the collision detection area, the boundary must be set 1 unit away from the sides of the actual object. This is to prevent the user from looking through the object. In figure 6.1 are areas marked 1-4. These represent the different ways the user can collide with the object. One direction was omitted, namely from above.

In order to prevent the user from crossing boundary 1, the user must not be able to cross the line (a, d) which extends along the blue line. However this prevents the user from reaching locations such

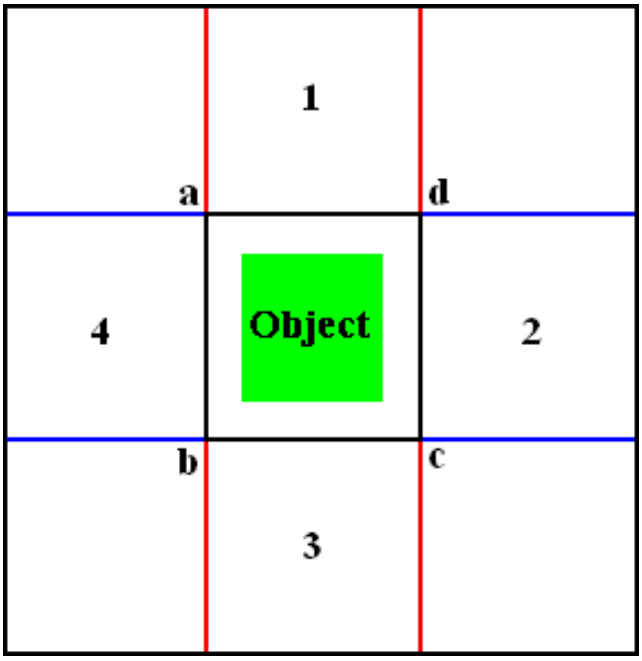

Figure 6.1: Rectangular Collision A

as 2, 3 and 4. To avoid this we must limit the boundary to just between the verticies a and d. Then conditions for collision are:

- If the user below the line (a, d)
- If the user is to the left of line (a, b)
- If the users is also to the right of line (d, c)

The users has collided with the object and must therefore step back to the boundary.

After implementing the above code for boundary detection to areas 1 and 3, a problem was found. If the user crossed the boundary (a, d) or (b, c), the program does not know which boundary is being crossed as they are both triggered. Hence the program executes the first correct condition it comes accross. To fix this, another condition must be made, namely:

- If the user is above the line between the midpoints of verticies (a, b) and (d, c)

Then the user has collided with boundary 1. This works when only boundary conditions 1 and 3 are applied. If boundary detection for areas 2 and 4 are also implemented, another problem arises. When the user crosses either boundary 2 or 4, the boundary conditions for 1 and 3 becomes true, depending on where the user crosses the boundary. The program then sends the user to either area 1 or 3 instead of stopping the user at the boundary line. In order to fix this, the boundary conditions must be ajusted yet again. The modified conditions are show in figure 6.2.

The conditions for boundaries 1 is then:

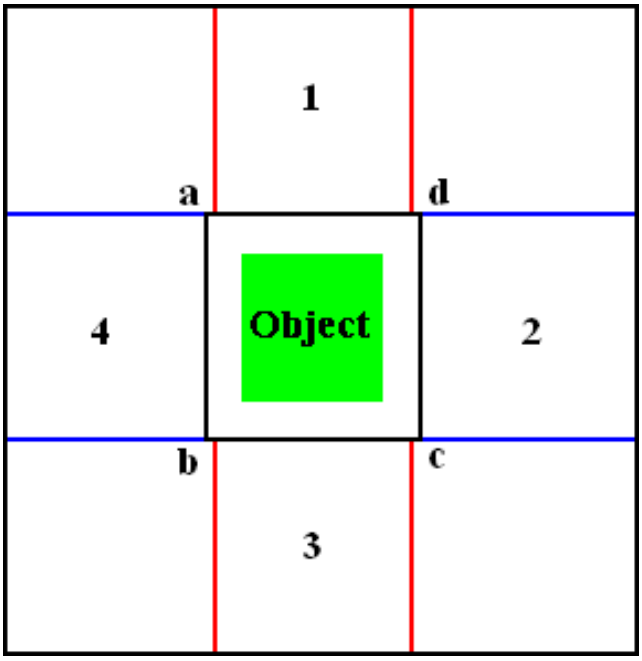

Figure 6.2: Rectangular Collision B

- If the user is below the line (a, d)
- If the user is more than 0.5 units to the left of line(a, b)
- If the user is also more than 0.5 units to the right of line (d, c)
- If the user is above the midpoint of verticies (a, b) and (d, c)

The user has collided with boundary 1. This applies for boundary 3 as well. Boundary conditions 2 and 4 will remain the same. Note that the boundary conditions 1 and 3 must come before 2 and 4. This way if the user collides with boundary 2, only the conditions for boundary 2 will be true.

To finish off the rectangular collision detection, the final boundary above the object must be implemented. To do this we must add another condition to boundaries 1-4, namely:

- If the user is below the maximum height of the object plus 1 units.

Then the conditions for collision will be true. Otherwise if the user is above the maximum height of the object plus 1.5 units and is within the rectangle (a, b, c, d), the user has collided with the top of the object and will remain on top of it. This then completes the rectangular collision detection.

6.3 Circle and Spheres

Some objects in the 3D desktop may be either circular or shperical in nature. For these objects, a circular boundary is required. An example of this is show in figure 6.3.

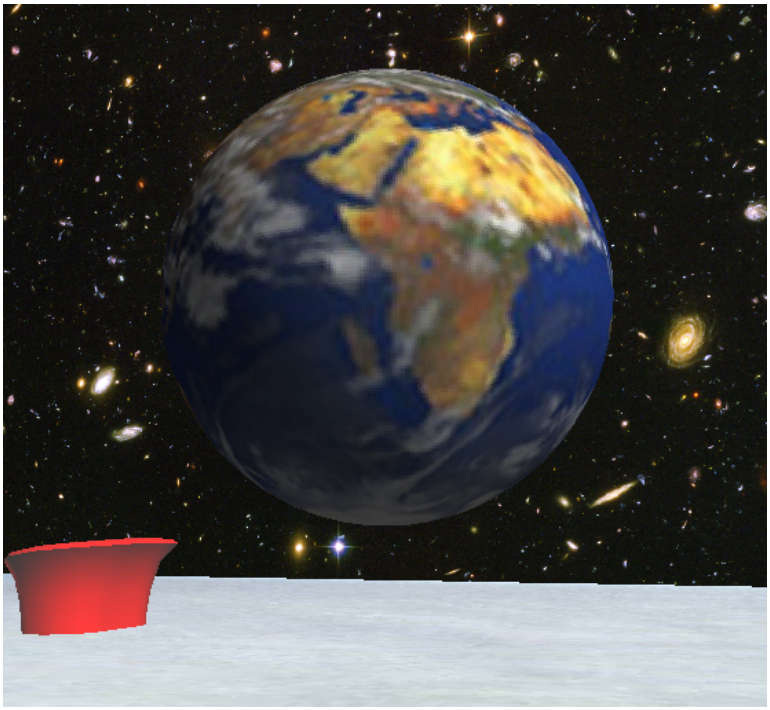

Figure 6.3: The Earth

First we have to set up the boundary conditions. The boundary must be at least 1 units radius from the circumference of the circle to prevent the user from looking into the object. This can be done via simple circle geometry calculations using the objects central position and the users current location. Once the user is at the boundary's circumference, he/she must then be kept at the circumference. There is 3 ways of doing this, all with some defects.

6.3.1 Stepping Back

In this method the user takes a step in the opposite direction of motion if he/she encounters the circular boundary. For example if the user is currently moving forward into the object when he/she encounters the boundary, the user immediately take a step back. This keeps the user right at the boundary. However this method can cause the user to be stuck in one position.

6.3.2 Recall

First create variables to save the previous location of the user. Then if the user collides with the circular boundary, the user's position will revert to its previous location. This method does not really work. It only causes a slight delay when the user first encounters the boundary. Once past the boundary, the user can walk right through the image until he/she encounter the opposite boundary. If we save the last two steps the user takes and when the user encounters the boundary, use the second

last position instead, we will get an image which bounces the user backwards every time he/she collides with the boundary.

6.3.3 Diagonal Motion

This is the method which was use in the 3D desktop application. Here the circle is separated into four quadrants. Depending on where the user collides with the object, the user will br forced to move diagonally around the object until he is no longer colliding with it. An image deplicting this method is show in figure 6.4.

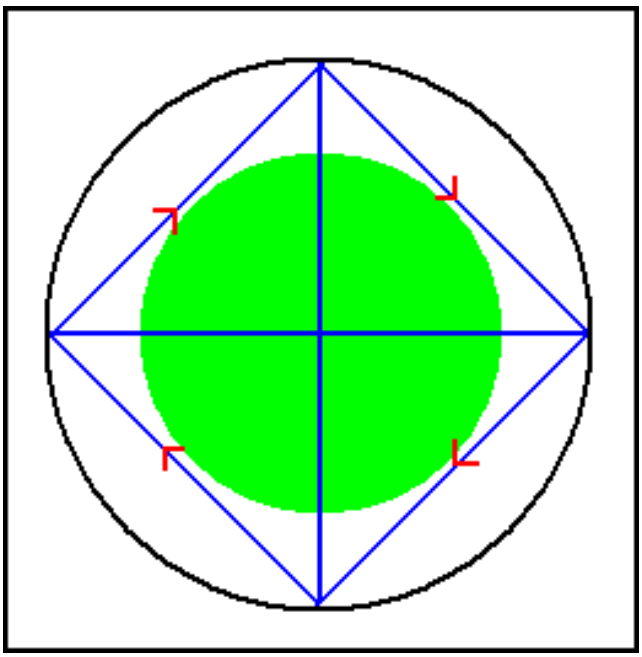

Figure 6.4: Diagonal Motion for Circular/Spherical Collision

This method does not actually prevent the user from moving inside the boundary. It only 'turns' the user away from the object.

Chapter 7

Headtracking

7.1 Choosing the API

This is a Netbeans program is written in java language. This means that in order to implement head tracking via the wiimote, an API which connects the wiimote to the computer which works in java is required.

There are currently four differnet java API's for the wiimote, namely:

1. WiiUseJ http://code.google.com/p/wiiusej/
2. WiiremoteJ http://www.wiili.org/index.php/WiiremoteJ
3. motej http://motej.sourceforge.net/index.html
4. Wiimote simple http://code.google.com/p/wiimote-simple/

This project will be using the WiiUseJ API because:

- Its website is the most regularly updated one.
- It has may sample code which users can browse through.
- It has a wiiusej.jar file with which users can test their wiimote connectivity and functions. This is crucial because it shows the users that this API does work since the file works.
- It has the API's source code which helps immensely in troubleshooting.

7.2 Using WiiUseJ

First download the WiiUseJ source files (wiiuseJ 0.12b src.zip) and API (wiiusej 0.12b.zip) from http://code.google.com/p/wiiusej/. They are located in the downloads section. Also download the javadoc for the API (WiiuseJ 0.12a Javadoc.zip) as a reference. Once downloaded, follow the directions in the readme.txt file in wiiusej 0.12b.zip.

Make a new library in Netbeans by going to the tools, library. This will open up Netbeans' Library Manager window. Inside the window at the bottom left is an icon called New Library. Click on that icon, name the library in the new window and select 'class libraries' for the library type. Now a new library has been created. To set up the WiiUseJ library, use the wiiusej.jar file inside the wiiusej 0.12b.zip. For the sources, unzip WiiuseJ 0.12b src.zip and use the folder as the source. Click ok and the new WiiUseJ library is created and ready to be used.

Now that the library has been made, simply add the library to your Netbeans project. Now you can import all the wiimote events and implement the WiimoteListener.

In the 3D desktop program, only the wiimote's IREvent was actively used. The IREvent is used to track infra-red lights. The wiimote can track up to 4 different points in space, but it only has 45 degrees viewing range.

7.3 Headtracking

In order to do headtracking, the wiimote's infra-red camera must be used. This camera only tracks infra-red lights. It can follow up to 4 different points in space, but only has a viewing angle of 45 degrees. It works well within a range of 5 meters.

Using the WiiUseJ library, we can program the 3D desktop to use the wiimote and track infra-red lights. By placing some infra-red LED's on a person's head, we can then track the motion of his/her head and thereby change the image shown on the screen. In figure 7.1, the electronic assemply of the IR LED's.

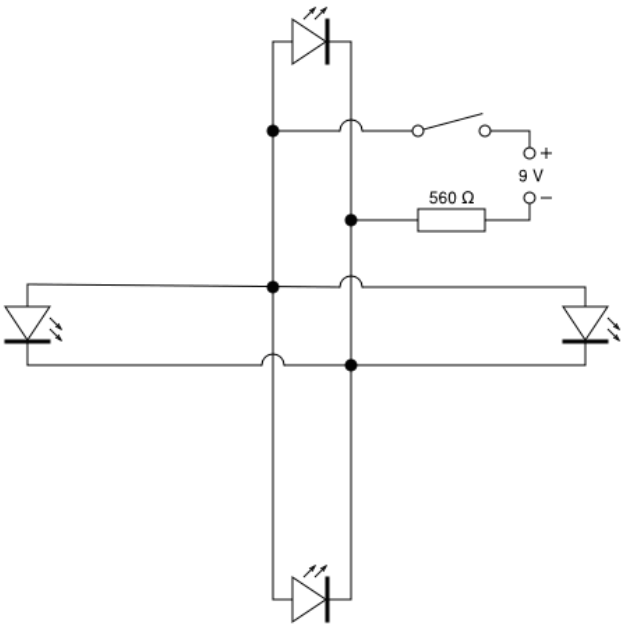

Figure 7.1: Circuit diagram for IR tracking LED's

This circuit uses a 9V battery as a power source, a standard switch, and four IR LEDs connected in parallel. To limit current to the LEDs, a single 560 Ohm resistor is placed between the battery and the four LEDs. The number of LEDs and the length of their cable leads can be varied according to the user's needs.

There are many ways for the user to wear the LED's on his/her head. They can use either a hat or some large glasses and place the LED's on them. For this project, the hat was chosen as a base for the LED's.

The LED's can be placed on the hat in many different ways. One way is to arrange the LED's as shown in figure 7.2.

Figure 7.2: Hat with 3 LED's

Here 3 LED's were used, two on the sides of the cap and one in the center. This should allow the user to program an application where translation and rotation of the head can be tracked via triangulation. The user can also get the distance of the head from the camera by calculating the distance between the LED's. However when tested this was not the case. When the user turns his head to the right, the cap blocks the light emitted from the left LED and vice versa. This means the user is only allowed a very limited range for head movements.

Another way to arrange the LED's is to use just 2 LED's. The placement of the LED's are shown in figure 7.3.

Figure 7.3: Hat with 2 LED's

With two LED's located at the tip of the cap, there is less chance of the cap obstructing the light emitted from the LED's. However with only two LED's, the user will not be able to differentiate between the distance his/her head from the camera and the rotation of the head via calculations, since they both use the distance between the LED lights.

In the 3D desktop program, the hat with 2 LEDs at the tip was used. The position of the user's head would be the midpoint of the two LEDs. The distance of the user's head from the camera was set to be constant. The rotation of the users head is included along with the translation of the his/her head. For example, if the user moves his head to the left, the camera will move left and turn right. When tested, a user's head motion limitation was found. This is because while the IR camera in the wiimote has a viewing angle of 45 degrees, the IR LEDs have an emission angle of 18 degrees. This means that even if the user's head is inside the wiimote camera's viewing angle, it will not pick up the LED's IR light if the users head is turned too much. This effect is show in figure 7.4 and figure 7.5.

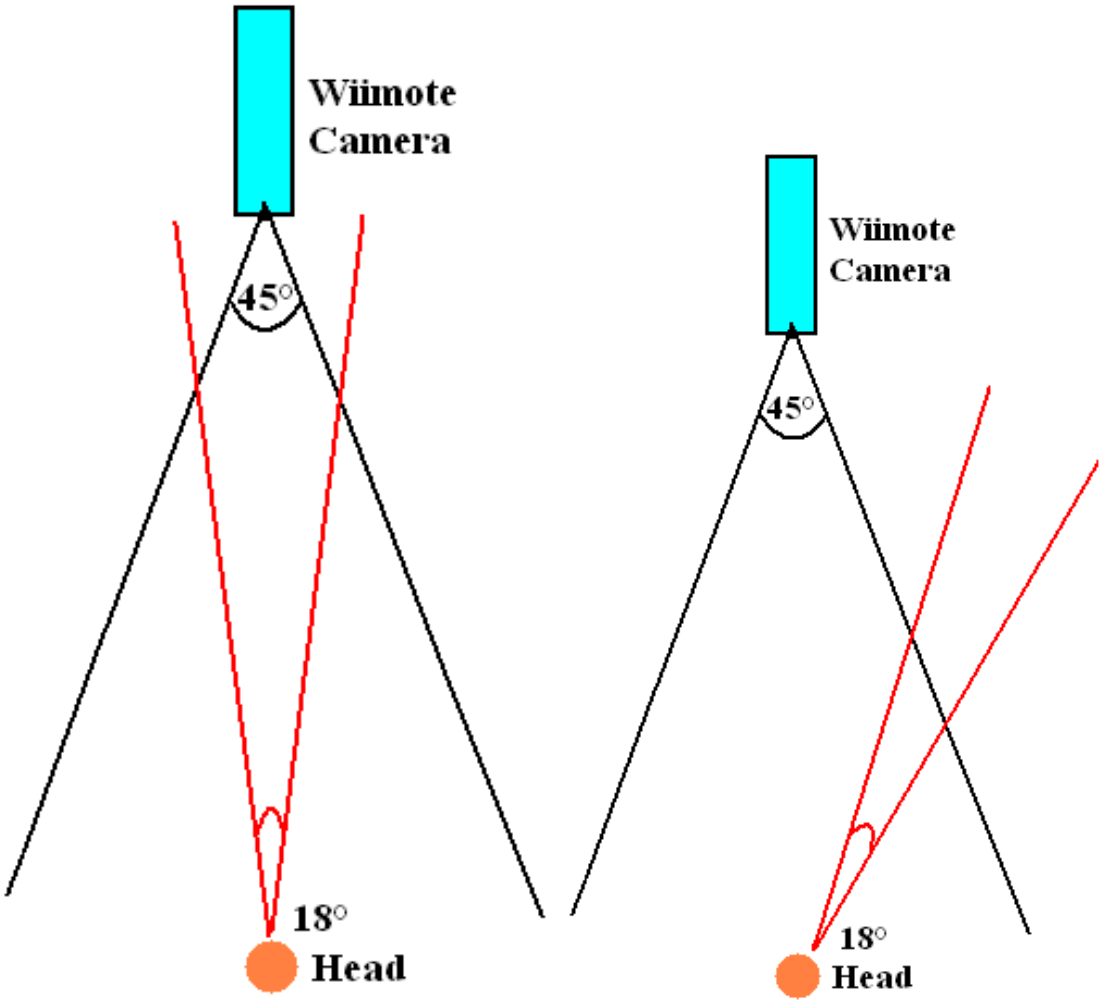

Figure 7.4: Wiimote camera and LED emission angle when the head is at the centre

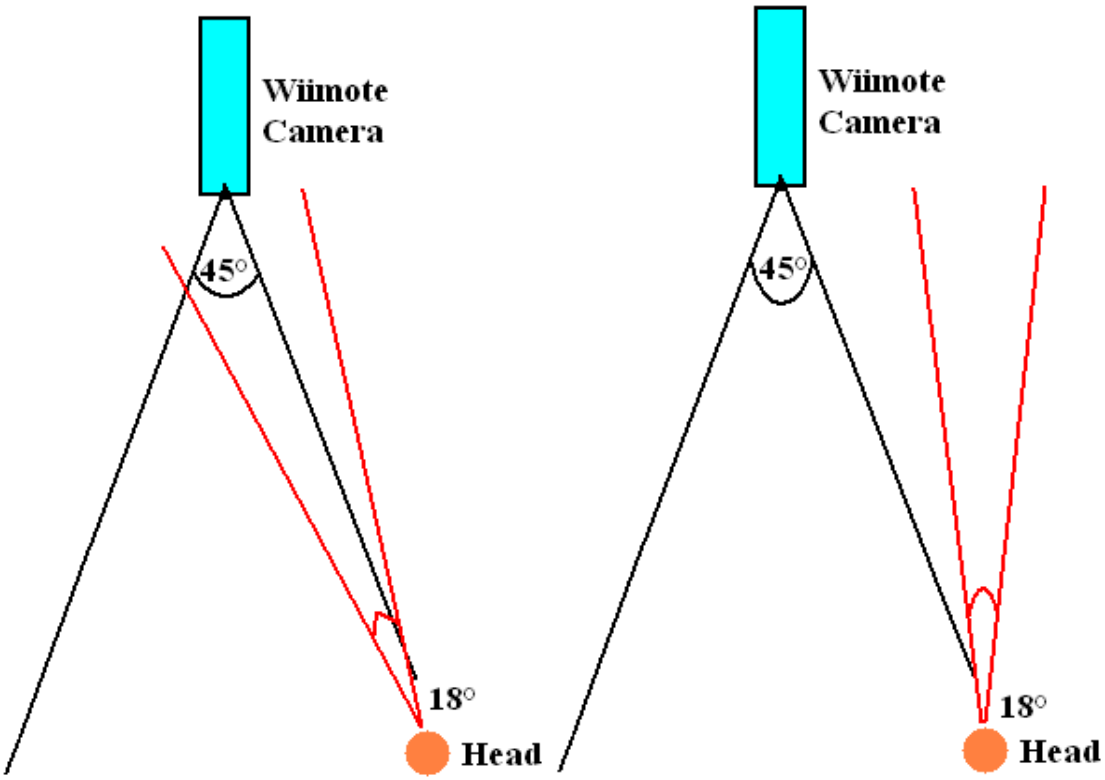

Figure 7.5: Wiimote camera and LED emission angle when the head is at the edge

7.4 Placement of the Wiimote

The placement of the wiimote is crucial to the performance of the 3D desktop program. The wiimote must be placed in front of the user, level with his/her eyes. The best way to do this if the user's screen is a pc monitor, is to place the wiimote on top of the monitor and align it so that it the camera faces the user's head directly. If the user is working off a laptop, stack some books or other objects until the wiimote can be place level with his/her eyes.

There are other ways to place the wiimote. All that is required is for the light from the IR LED's on the user's head to be close to the center of the wiimote camera's focus. To verify the position of the IR lights, open the wiiusej.jar application located inside the wiiusej 0.12b.zip. Note that the wiimote's camera must not be facing sun or any reflective surface as this may cause the camera to pick up alternative sources of IR light.

Chapter 8

Tests and Results

8.1 Normal Keyboard and Mouse Controls

Most people are comfortable with the normal keyboard and mouse controls. However there are some complaints:

1. When jumping, once the user is off the ground, he/she should continue moving in the specified direction regardless of camera rotations.

2. When jumping, if the user continues pressing forward and strafe left, the user will 'float' in the air until one of the buttons are released.

3. If the user jumps on top of an object, he can jump one more time. After that the jump function stops working.

4. When the user is on top of an object and walks off the edge of the object, he remains in midair.

5. The Q keyboard key is too close to the movement keys of the application. It is easy for users to close the program by accident.

8.2 Headtracking

8.2.1 Setting up the wiimote

People are comfortable with connecting the wiimote to the pc. Most can do so with ease after one trial run. The problem is the positioning of the wiimote. Unless the user has a pc monitor, it will be had to find a surface at sufficient height to place the wiimote.

8.2.2 Viewing objects

Most testers feel that using headtracking to view objects in a distance is a redundant function. The mouse and keyboard controls gives the user better control over the camera than headtracking. Instead using headtracking for object that are close to the user is much more preferable.

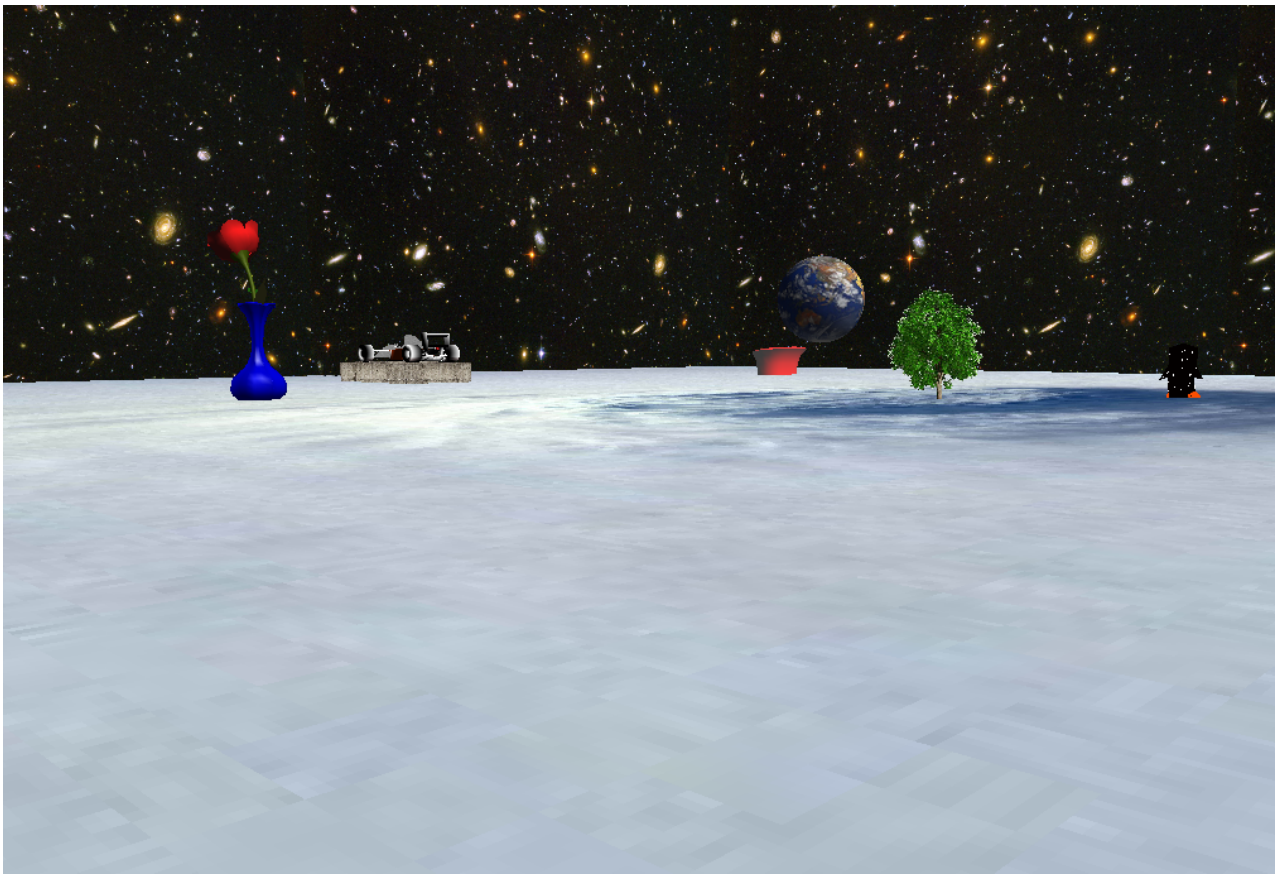

Figure 8.1: Viewing object from a distance

8.2.3 Headtracking Motion

Testers find the required head motions for headtracking counter-intuitive at first but most can adapt rapidly. It is counter-intuitive because the user must always be facing the wiimote camera for head-tracking to work, else the camera will not pick up the IR LED's lights. Most tester either rotate their head in order to turn the camera, or moves their head to the side while look forward and not at the camera.

8.2.4 Hardware

There were some complaints about the hat. Some users feel a pair of glasses is preferable to the hat. The hat makes the user hot when the day is warm. It also makes some people itch and to scratch it,

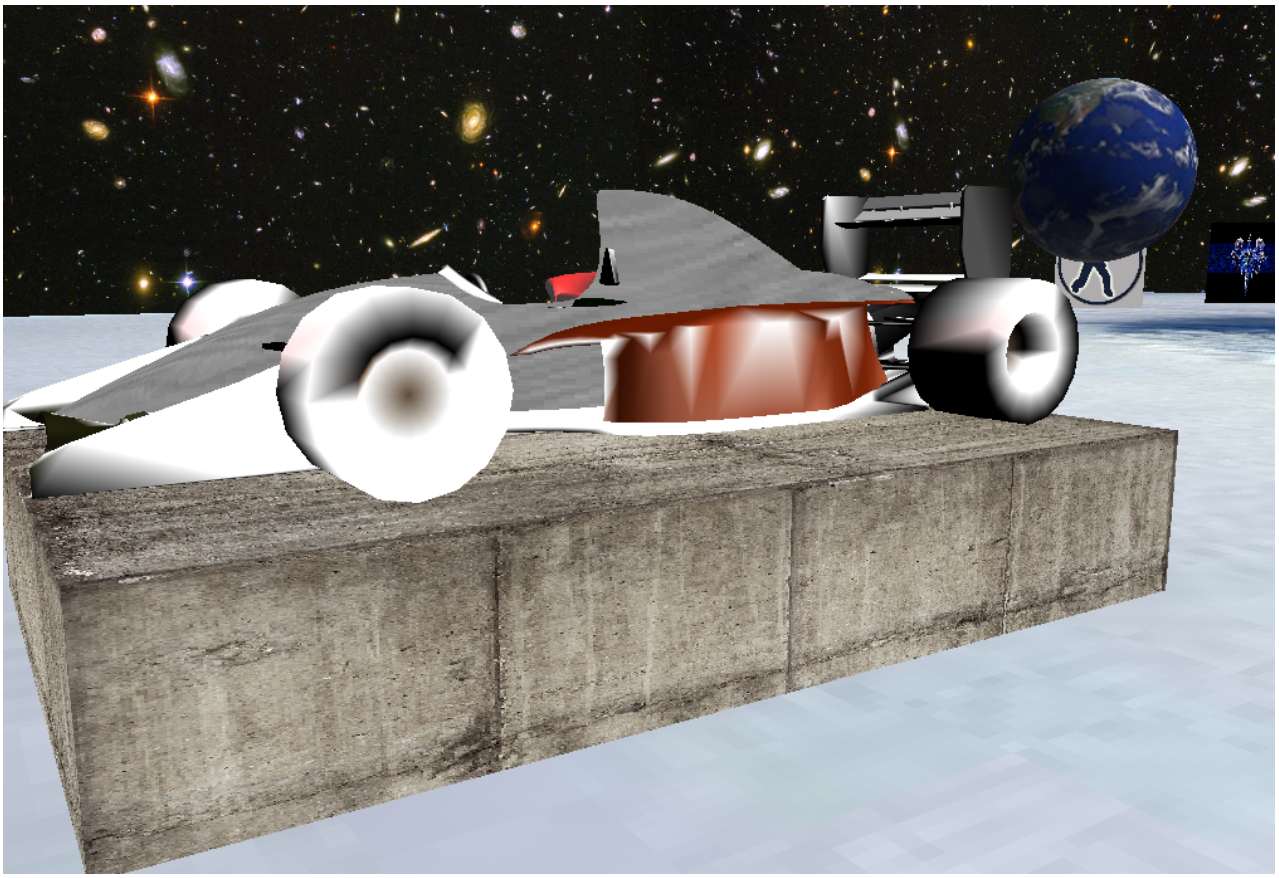

Figure 8.2: Viewing object from close up

the hat comes off.

8.3 3D Desktop

Most testors feel that, although a 3D desktop is a novel idea, it is not really practical when put into use. This is because:

1. It is takes longer for a user to open an application in a 3D desktop. He/she has to first walk to the icon before he/she can activate it.
2. Having a 3D desktop while the microsoft folder systems are planor makes the whole system appear conflicting.
3. Only people with good pc's and RAM to spare can use this system.
4. Placing new icons in the 3D world will take too long.

8.4 Other Issues with the Program

- The image billboards does not turn as they optimally should. When the billboards are along the edge of the screen, they do not face the user. This is shown in figure 8.3.

- The collision detection system works when the user is directly facing the object, but fails if the camera rotates. Sometimes the user can still see inside the object. This is shown in figure 8.4. The see-through part is circled in red.

- When a .exe file is opened, the 3D desktop program becomes paused. This means you can only open one program at a time.
- When a explorer file is opened throught the applet, the applet still tracks the mouse cursor movement and rotates the screen. This gives a disconcerting effect for the user if the explorer file frame is not maximized.

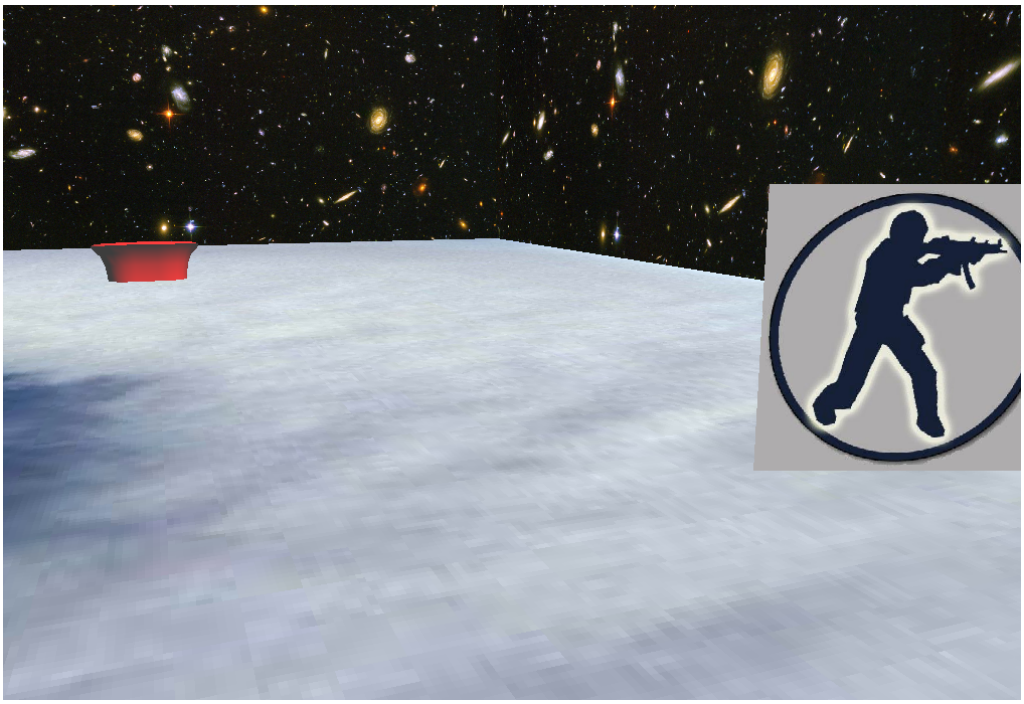

Figure 8.3: Image Billboards

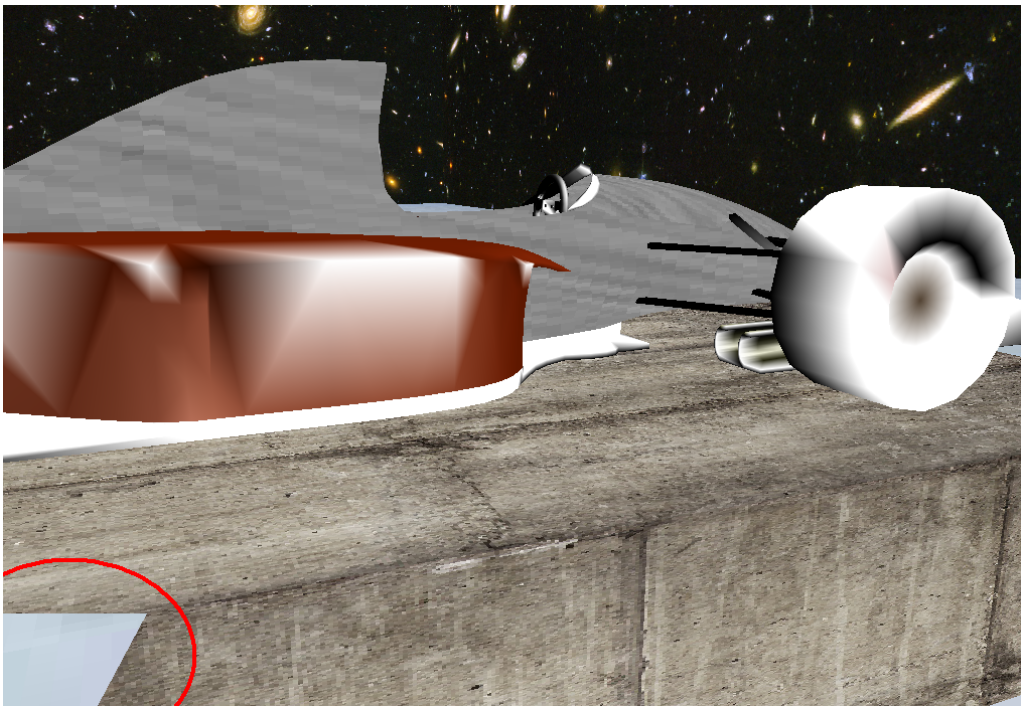

Figure 8.4: Looking through objects

Chapter 9

Conclusions

Through the process of this project, the following has been determined:

- Headtracking via the wiimote can be implemented through the use of the WiiUseJ API.
- The placement of the wiimote relative to the user's head greatly influences the sensitivity to the IR camera in the wiimote. If done correctly the performance of the headtracking in the 3D desktop program will be greatly enhanced.
- Using headtracking to view close objects works well but becomes redundant when viewing thing far away.
- Although 3D desktops provide better graphics, it is still in the developmental phase and so most people still use the normal 2D desktops.

Chapter 10

Future Work

The 3D desktop was found to be more troublesome to use then the normal planor desktop. However the 3D environment does works and can be improved upon. Here are some modifications which can be added to the program.

- The collision detection for circular/spherical objects can be improved.
- The 3D environment can be changed into a game area.
- Use 3 or 4 IR LED's instead of just 2 LED's and try to track the distance of the head from the camera and the rotation of the head.
- Try to increase the emission angle of the IR LED's by either placing 2 LED's at the same spot or using a LED which has a larger emission angle.
- Create some objects which can be used by the user or can interact with the user.
- The skydome can be replaced by a rotating circular skydome to make it more realistic.
- The jumping method can be improved upon.
- When walking off the edge of an object, the user should fall to the ground.

Appendix A

Software Source Code

All of this project's software code will be included in the attached Project CD. A software copy of this document is also included in the CD.

Bibliography

[1] Pro Java 6 3D Game Development
http://fivedots.coe.psu.ac.th/~ad/jg2/index.html

[2] Developer Resources for Java Technology
http://java.sun.com/

[3] Johnny Chung Lee - Projects - Wii
http://www.cs.cmu.edu/~johnny/projects/wii/

[4] WiiUseJ, Java API for Wiimotes
http://code.google.com/p/wiiusej/

[5] Wikipedia
http://en.wikipedia.org/