1. Year and Semester : 2013 SRING

2. Course Number : CS-492

3. Course Title : Kinect Programming

4. Work Number : LK-03

5. Work Name : Kinect Laboratory Report

6. Work Version : Version 1

7. Long Date : Sunday, 10, Frebruary, 2013

8. Author(s) Name(s) : Jake Waffle

For the first thirty pages of chapter 3, the book just introduces some basic ideas that I assume will be needed for the projects in this chapter (Borenstein pp109-139.) The ideas include setting up a 3d drawing space (with and without OpenGL,) navigating within the 3d space (rotations, translations,) displaying a point cloud of what the Kinect sees in the 3d space and determining if an object is inside of an imaginary box (for a sort of button activation through the Kinect.) I would have had explanations for all of the parts, but I didn't save the document I typed them onto… So since I've already analyzed and understand them, I'm just going to move onto the projects.

Project 8

This project is all about creating a sort of interface through the Kinect that allows one to activate sounds when touching the "hotspots" within the Kinect's visible space.

1. The first bit of code within the book for this project tells us how to setup the audio in Processing, as well as how to stop the audio when the program is told to close (Borenstein p141).

2. It's good to know that Processing works with .wav's and .mp3's (the book had a .wav file for its example, but I found an .mp3 to test the code with.) And I received my music file from a music site dedicated to independent game developers (Karabegovic).
3. Then I copied the next code snippet from the book (Borenstein p143).
4. It isn't quite the end result of what I thought we'd be making for this project, but it also isn't the last code snippet for this project.
5. This code basically displays the point cloud and checks to see if any points are within the box that the code superimposed into the 3d space. And if any points are within the box, the music I loaded into the code plays until it is done playing.
6. There is also a way to rotate the point cloud based off of the mouse's x-position within the window (along the x-axis.) And the up and down arrows are rigged to zoom in and out on the point cloud.
7. Another big part of the code is that it checks to see if ANY points are within the defined box in our 3d space. And when a point FIRST comes into contact with the box, which will trigger the loaded sound file to play.
8. And of course the point cloud along with our box is drawn to the window like in the introduction code snippets to this chapter.
9. Then I copied the next code snippet from the book (Borenstein pp147-151).
10. This bit of code pretty much just grouped up the box's dimensions and the logic for whether the button was pressed or not into a class called Hotpoint. From there, the global variables, setup() and draw() areas were all altered to declare, initialize and use a couple instances of the Hotpoint class.

11. Within the Hotpoint class the way that the box is drawn is a bit different than before though. The draw() method on page 151 uses pushMatrix() and popMatrix() to make sure that its translations are independent to that of the other boxes. This is part of how OpenGL works (in immediate mode anyways.) Basically pushMatrix() resets the position that you're drawing at to (0,0,0), but since the matrices are on a stack the previous matrix still exists. Then popMatrix() will throw away the matrix we just pushed onto the stack and will then make the previous matrix the one that is used for drawing.
12. The transparency of the Hotpoints changes as you enter it like in one of the introductory code examples also. And the Hotpoint class also gives its box a random color within its constructor.
13. And I had troubles finding any music that was under a license. So I just reused the track I had already found (Karabegovic.)

For modifications (before adding in the Hotpoint class,) I first made it so that the point cloud drew fewer points (while not checking all points for collision) and no points (because the collision detection for the points was rather unresponsive.) I altered the step of the for loop so that some points would be completely omitted in the collision detection and drawing and I also commented out the point() function so that all the points are checked for collision, but no points are drawn (Borenstein p148).

The other modification (also before adding in the Hotpoint class) that I've spent some time on is with making the sound file stop playing when there is no collision with the box anymore. I tried several things, but to no avail. I did find out eventually that it was my logic that was the problem (although I have no clue on how to fix it, because it seems like it should work.) The part that I added was the "if(wasJustInBox && !isInBox){}" in the following code. The

console never prints the string in that if statement, but I know that there is a point where points were just in the box but not in the box anymore. If it did work like I thought it would, then the music would end and rewind when the user isn't colliding with the box anymore.

```
boolean isInBox = (depthPointsInBox > 0);

if (isInBox && !wasJustInBox && !player.isPlaying())
{
  player.play();
}

if (wasJustInBox && !isInBox)
{
  println("STOP PLAYINASidbiasdubfiuabsdf!");
  player.cue(player.length());
}

if (!player.isPlaying())
{
  player.rewind();
  player.pause();
}

wasJustInBox = isInBox;
```

And after adding in the Hotpoint class, I made it so that the fill color of the box is a different color than that of its edges. I also made it so that the Hotpoint requires more than just one point to trigger the music. The code changed is as follows and exists within the Hotpoint's constructor.

```
threshold = 20;
fillColor = color(random(255), random(255), random(255));
strokeColor = color(random(255), random(255), random(255));
```

Project 9

This project essentially taught us how to load in 3d models, use lighting in OpenGL and how to use PeasyCam to look around our 3d space.

1. First I downloaded the saito object loader library from the place specified in the book (Borenstein p157). And of course I placed it within our …/sketchbook/libraries directory along with the SimpleOpenNI library.
2. Then I downloaded the kinect model/texture from the place specified in the book and placed the files within the soon to be project folder for the upcoming code (Borenstein p158).
3. Then I copied the first bit of code from the book (Borenstein pp158-159).
4. After that I ran the code and moved my mouse around to look at the model from various angles. I thought that the model was pretty white and hard to see when I looked at it from certain angles (it did start out pretty dark like the picture on page 159,) so I commented out the call to the lights() function to see if it would make it darker. But that just made it so that the model didn't show up at all. Then I realized that the model was actually displaying in white (against the white background) from reading the book (Borenstein p163).
5. Then I added in the code snippet in the book that allows transitions between triangle and line drawing modes for the model (Borenstein p165). And it reminded me of how OpenGL renders lines, quads and triangles with the use of an array of points (you can also switch between the shapes by changing a separate parameter than the points.)
6. Next I copied the code from the following code snippet (Borenstein p166). And this snippet essentially draws the model we were just dealing with, but in the place where the

kinect is supposed to be in relation to the point cloud. And the point cloud is drawn the same as in the previous project. The only difference is the lights() function as well as the kinect model being drawn to its respective position (note that the model had to be rotated around so that it is right-side up.)

7. Then a minor alteration to the last code snippet was made according to the example in the book (Borenstein p171). And this essentially draws the line of sight for the kinect within our 3d space. This essentially is done by drawing a series of rays from the kinect to each point in the point cloud (note that the lines are semi-transparent.)
8. The camera control section is next and it requires that another library is downloaded called PeasyCam. And this can be downloaded from the place specified in the book (Borenstein p173). Afterward, the library must be placed within our .../sketchbook/libraries directory like all the other libraries.
9. I then copied the code from the book for the PeasyCam example (Borenstein pp173-174).
10. The PeasyCam's controls are much nicer to use than when we wrote the code to move the point cloud around ourselves (although we didn't come up with the code.) I really like how there's a way to go back to the starting position by double-clicking the mouse (the image can get a little messed up with these controls too, even though they seem nicer.)
11. And for then the final code snippet was copied from the book (Borenstein pp179-180). This code makes use several libraries as well as the Hotpoint class that we wrote for project 8.
12. The code that was copied essentially is a combined version of our previous code snippet (with the peasy cam controls for the point cloud) and project 8 (except the camera is told to look at the hotpoint instead of play music.) And when the hotpoint is hit, a PeasyCam

method is called “lookAt” is used to make the camera position itself so that it is zoomed in on the selected hotpoint (from there a double-click is required to go back to the beginning camera position.)

First when doing modifications, I was unsure about why the PeasyCam's lookAt() method needed the y- and z-values negated. So I made them not get negated to see what would happen. And when a hotpoint got hit, the camera would move away from the hotpoint on the y- and z-axis (meaning PeasyCam's coordinate system just works differently than OpenGL's.) Beyond that I wasn't really sure what to add besides more hotpoints and I was running out of time. So I just left it at that.

References

-Borenstein, G (2012). Making Things See. Sebastopol,California: O'Reilly Media.

-Adel Karabegovic. (undefined). . In IndieGameMusic. Retrieved February 19, 2013, from http://indiegamemusic.com/viewtrack.php?id=1150.