Natalie Hong (45309740)

COSC300 Graphics Report

Semester 1 - 2020

Contents

List of Images

1 Introduction

In this document, you will find my findings and attempts at creating a completed game. The game I had chosen to make was that of the supplied mega racer code.

To create the graphics required for this project, I used Python 3.5 and OpenGL to create objects and renderables within the computer graphics scene. I also used GitHub to allow for version control between each implemented feature that I added.

The features that were made were:

- 1.1 - Scaling the Terrain
- 1.2 - Setting up the Camera
- 1.3 - Orientating and Placing the Racer Model
- 1.4 - Texturing the Terrain
- 1.5 - Lighting from the Sun

In the following documentation. You will find screenshots and findings around each feature as well as how I went about approaching each feature within the project.

1.1 - Scaling the Terrain

Scaling terrain consisted of scaling the terrain according to the path image supplied.

```
# copy pixel 4 channels
    imagePixel = self.imageData[offset:offset+4];
    # Normalize the red channel from [0,255] to [0.0, 1.0]
    red = float(imagePixel[0]) / 255.0;

    xyPos = vec2(i, j) * self.xyScale + xyOffset;
# TODO 1.1: set the height
zPos = 0
```

Initially, the code supplied consisted of zPos = 0 and I was to change it. I thought long and hard about to change and looked at the hints supplied. In it, I was told to use self.heightScale and to somehow utilize it in order to get a consistent z position for each terrain created. I had a thought that multiplying the red value calculated would allow me to get a a consistent z position. This was because red represented a value from 0 to 255 and multiplying it with the height scale would allow for different Z positions to be generated.

```
# TODO 1.1: set the height
zPos = self.heightScale * red
```

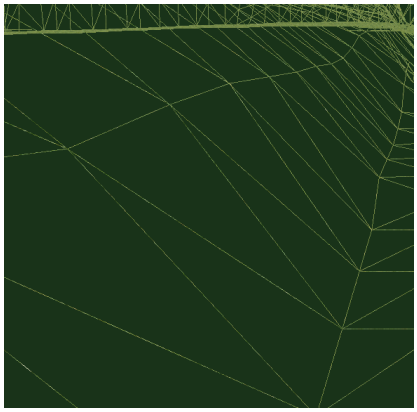

Figure 1: Scaled Terrain

This seemed to work as I was now able to see bumps and curves within the terrain which meant this feature was completed.

1.2 - Setting up the Camera

The next feature I was to implement was that of setting up the camera so that it viewed the racer at all times and was scaled to the supplied camera offset. As well as this, I wanted the camera to be able to rotate according to the heading of the racer. After much experimentation I was able to come up with the following code:

```
x = g_racer.position[0]
y = g_racer.position[1]
z = g_racer.position[2]

....
g_viewTarget = [x, y, z]
```

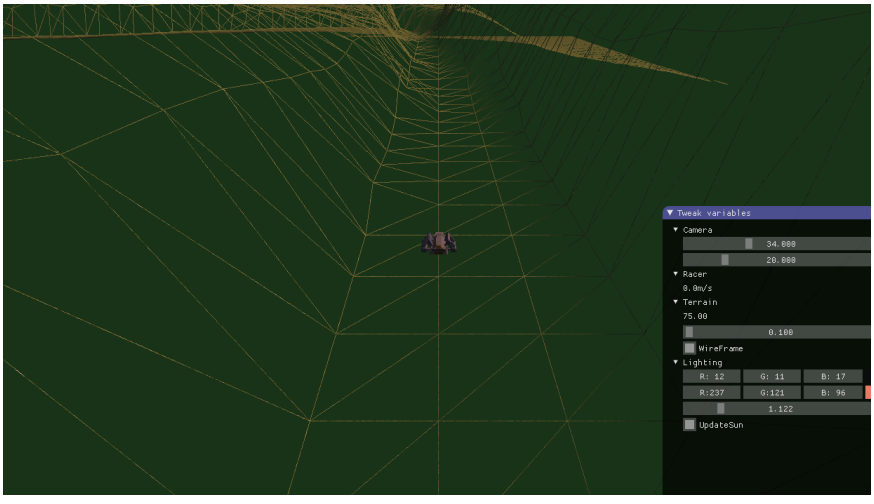

Figure 2: Looking at the Racer

Since I wanted my camera to look at the racer, I decided to make it so that the camera was always looking at where the racer was positioned. However, I did not want the camera to be at the exact same position as the racer as that would make it feel weird. To make sure the camera was positioned away from the racer, I used the assigned follow cam offsets. This code was very similar to Lab 3 of the course. The result can be seen in the above figure, please note that this screenshot was taken later into development of the game.

```
    for i, positionPoint in enumerate(g_racer.position):
        g_viewPosition[i] = positionPoint + g_followCamOffset * -g_racer.heading[i]

    g_viewPosition[2] += g_followCamLookOffset
```

In the code above, I was able to create a viewing position for the camera to be positioned at. Position points were each of the racer's coordinates and were then added to the follow cam offset multiplied by the negative of the racer's heading coordinates. Since -1 for the heading coordinates corresponded to the right rather than the left, I made sure to make the values reverse using the negative version of the heading value.

Since z-axis had an added follow camera offset as it was specified in the task sheet, I made sure that the z axis needed to have the additional multiplier of the follow cam offset to it.

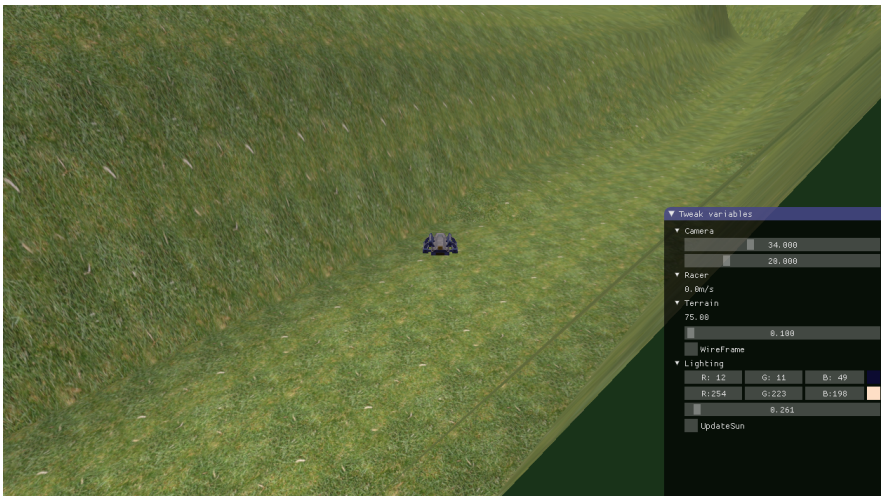

Figure 3: Camera Rotating with the Racer

The result of rotating the camera can be seen in the above figure, please note that this screenshot was taken later into development of the game.

1.3 - Orientating and Placing the Racer Model

This task for the racer required me to rotate and orientate the model. This model loading was very similar to the of Lab 1. To place the model at the racer's location I did the following:

```
        top = [0,0,1] # Define top in the Z-axis
        modelInWorld = lu.make_mat4_from_zAxis(self.position, self.heading, top)
        # Draw model in the specified world coordinates
        renderingSystem.drawObjModel(self.model, modelInWorld, view)
```

First, I defined the top of the world in that of the z-axis. This was necessary as the task sheet recommended to do it and allowed me to use more functions. Using this top value, I made a

4x4 matrix using the racer's position, where it was heading and the top value defined earlier. Once I got this 4x4 matrix which represented coordinates to draw the model in, I then added the model to the coordinates and used the passed in view to place the Racer in the scene.

```
self.model = ObjModel(objModelName)
```

The last thing I did was load the model inside the racer's load function so the racer model could initialise on load.

Figure 4: The Loaded Model

Once loaded, I was able to rotate the model with the camera to allow for the racer model to always be facing forwards no matter which way the camera was adjusted to.

1.4 - Texturing the Terrain

1.5 - Lighting from the Sun

- used shading solutions/what I learnt in lab 4 - noticed that the passed in values were: vec3 materialColour, vec3 viewSpacePosition, vec3 viewSpaceNormal, vec3 viewSpaceLightPos, vec3 lightColour

did something similar with global variable of globalAmbientLight