CITS3003 – 2016 Semester 1
Graphics and Animation

Project Report – Part 1

Caleb Fetzer – 213849876
Group # 18

Part 1 Overview

For the first part of this project, my partner and I divided up the tasks to even out the workload, and did help each other and assist if required for some small aspects. All of the required functionality and additional functionality works as described in the spec and by us. For my group contribution to the main tasks, I implemented parts A, B, E and I. For my individual implementation, I implemented object deletion and for our group additional implementation (e.g. Part K), we added in the ability to save and load scenes.
Some of the tasks required did require additional research and were not intuitive at first. The lecture slides, labs, text-book and external internet sources were used in order to successfully implement everything.

Part A – Camera Rotation

To implement the camera rotation, we needed to pass the provided *camRotUpAndOverDeg* and *camRotSidewaysDeg* values to the *RotateX()* and *RotateY()* functions respectively, in the *display()* function. These were then multiplied by the view matrix to achieve the desired effect:

```
view = Translate(0.0, 0.0, -viewDist) * RotateX(camRotUpAndOverDeg) *
                                        RotateY(camRotSidewaysDeg);
```

Part B – Object XYZ Rotation

Like Part A, the rotation needed to be added except in to the drawMesh() function for the model view.

```
mat4 rotateXYZ = RotateX(sceneObj.angles[0]) * RotateY(sceneObj.angles[1]) *
                                               RotateZ(sceneObj.angles[2]);
mat4 model = Translate(sceneObj.loc) * rotateXYZ * Scale(sceneObj.scale);
```

This allowed the objects to be rotated on the x, y and z axis. In order to match the sample solution, we had to also modify the appropriate toolCallback for adjusting the angles, from:

```
setToolCallbacks(adjustAngleYX, mat2(400, 0, 0, -400),
                 adjustAngleZTexscale, mat2(400, 0, 0, 15) );
```

to:

```
setToolCallbacks(adjustAngleYX, mat2(400, 0, 0, 400),
                 adjustAngleZTexscale, mat2(-400, 0, 0, 15) );
```

Part E – Reshape Function

In this part I referenced our 5th Lab[1] that had to do with our reshape function. In order to keep the objects in view, and to not be clipped out of the window, I needed to multiply the height and width by the nearDist variable, and to also check if the width of the window was less than the height of the window. I used the height-width ratio to scale it accordingly.

```
if (width < height) {
    projection = Frustum(-nearDist, nearDist,
        -nearDist*(float)height/(float)width,
        nearDist*(float)height/(float)width,
        nearDist, 100.0);
  } else {
    projection = Frustum(-
nearDist*(float)width/(float)height,
        nearDist*(float)width/(float)height,
        -nearDist, nearDist,
        nearDist, 100.0);
  }
```

Part I – Additional Light

To add the second light to the scene, it was easy to use the existing code for the first light and add a new object to represent it. Directional lights, however, should simulate the effect of being a very far away light[2]; therefore I needed to set its homogenous coordinate, *w*, to 0.

```
SceneObject lightObj2 = sceneObjs[2];
lightObj2.loc.w = 0.0;
vec4 lightPosition2 = view * lightObj2.loc;
```

The lighting calculations from our fragment shader were then duplicated, in order to use the second light, and so we simply created new variables to accommodate, such as *ambient2* and *specular2.*

Part J – Individual Implementation

For my individual implantation I implemented object deletion. The first thing to be done was to check the scene for the amount of objects, and if there were no objects except for the original 3, then to return the function immediately and go no further.

```
if(nObjects == 3) return;
```

This was to prevent accidentally deleting the original light sources or ground.
To extend on my partners functionality of switching and changing between objects, the following line was written.

sceneObjs[currObject = sceneObjs[--nObjects];

Previously it was simply:

nObjects--;

Which only removed the last object added, however the revised code sets the current object to the last object, as well as decrementing the object.
I then set the tool object to the current object, and to equal -1, therefore the toolObj was to be modified and also the current object.

toolObj = currObject = -1;

After adding the tool call backs, the function was successfully working.

Part K – Group Implementation

For our group added functionality, we implemented saving and loading the scene. This was done by saving the view distance, camera rotations, the number of objects and scene objects to a file. We then created the appropriate tool call backs to enable saving and loading the scene from the menu. All the files would be saved in the directory of which the scene-start was ran from, and also the scene loading was loaded from the same directory.

Reflection

I found this part to be a little challenging, and I certainly had to look up a few extra resources apart from the lecture slides and recommended textbook. Thankfully some of the information was able to be adapted from the labs, but I did find the project to test our ability to think out of the box a little bit and go into the deep end. Some parts, in which I didn't have much of an idea on, did require trial and error to achieve the desired result. If I was stuck, I was glad I had partner so that we could bounce ideas off of each other in order to help one another.

Bibilography

1. CITS3003, UWA: http://teaching.csse.uwa.edu.au/units/CITS3003/labsheet.php?fname=lab5

2. Tom Dalling, Directional Lights, http://www.tomdalling.com/blog/modern-opengl/08-even-more-lighting-directional-lights-spotlights-multiple-lights/. [Accessed 16th May 2016]