COSC343 – OpenGL Report

Patrick Baxter | ID: 1702748

The task for the assignment is to create a render engine using OpenGL that included several features. The majority of the code was provided with the ability to load basic models and materials from *obj* and *mtl* files, as well as methods to allow for multiple rendering modes, setting shader colours, among others. There were a number of changes to be made to allow for, complete model rendering, material loading, shaders (lighting models, and loading mtl values), transparency, and finally adding special effect shaders. I will go over each file that was changed, and what each of them do to provide the final render.

renderApp.cpp is the starting point of our application, and it provides the initializing, loading of models, and also provides control for multiple rendering modes and navigating. It first initializes the window which objects will be rendered to, and loads the OpenGL extensions that will be used. I enabled GL_BLEND which allows us to use transparency later on. We then create our scene using the provided *Scene* class and add the objects using the Assimp library. At this point, I also create an array for all *Group* class objects for later use when changing rendering modes. The camera is then set with controls that allow for movement around the space and the render loop starts. This is where the control inputs occur, including changing rendering modes (press 1: Black & White, press 2: Toon Shader, press 3: normal), and rendering the scene's objects to the camera.

The scene calls each *Group* object's render method which binds shaders on each of the meshes within a for loop, before rendering the objects to the camera. When these *Group* objects are first initialized, they need to setup the values from the provided *mtl* values to the shader. For each of the meshes for an object, it sets the diffuse, ambient, and specular colours, as well as the opacity. It will then set the texture if there is one provided. These values will be passed into the *mtlShader* files.

The *mtlShader.vert* takes our objects vertices and provides the position of them in the clip space, the position in world space, and in camera space. The *mtlShader.frag* handles the colouring and lighting for each point. We take the *rgba*(red, green ,blue ,alpha) values that we got from the group object for our texture colour values. We also calculate some variables for Phong shading and Blinn-Phong reflection model which give us the cosTheta for diffuse, and cosAlpha for the Blinn-phong calculation in our components. Depending on the render mode we add some different effects to our colours. In render mode 1(black & white) we use the relative luminance calculation to get a greyscale, and render mode 2 floors the diffuse's cosTheta at a number of levels times a scaling factor to create cell shading/toon shading. Finally, we can set our *color* variable's *rgb* values with our three components and alpha with the opacity giving us the final result rendered to the camera.

During the testing/building phase it was important that I was constantly running the program to get an understanding of what was happening. Each change usually had a visible difference with exception of coding errors (segmentation faults, etc.), like when I was able to load all meshes provided, I was able to see the entirety of the objects. This was particularly handy when changing the fragment file which I did not need to recompile the program for. I found the provided *obj* files to be adequate for testing all the features, as the *world* object had transparency, and I was able to see the difference in light values from the two objects. The one flaw I would change would be the Toon shading, as I believe changing the specular component would make a more convincing cell shading effect but I was unable to find a working method.