For our current project, we were tasked with re-creating a 2-dimensional image as a 3D scene. To design and translate this 2D image into my scene, I utilized several primitive objects. The primary shape I began with was to sketch out the general positions of the sword parts with cylinders, as well as using triangles to form the base plane, as well as a box I had used as a placeholder for the torus. The cylinders cover several aspects of this scene, including an additional flat layer of the plane adding the ashen texture. A cylinder is also used for the tang/grip as well as the cross-guard and blade. These cylinders are scaled to form proportionate objects that reflect the 2D image. A cylinder was also used where the sword meets the plane to form a light source of white ‘heat’ at the center of where flames would be, using a lightingCubeShader object. The other primitive shape that was used to great effect is the torus. To make the pommel, coiled part of the blade, and the ribs of the skeleton in the ashen plane. To form the skull, I utilized the fourth primitive shape, a sphere. I used a bone texture for obvious reasons and used the black leather texture that is used in the grip to create the eye sockets. The square plane created with triangles was also rendered using the lightingShader, which allows ambient, diffuse, and specular lighting in our scene, providing us with our full Phong lighting model. There is a red ambient lighting and our spotlight casts a light orange over the red to provide our user with highlights when exploring the scene.

As for maneuvering within the 3D scene, I chose a traditional control scheme for those accustomed to mouse and keyboard movement. W is for forward, S for back, A for left strafing, D for right strafing. Additional verticality movements are Q for upwards motion and E for downwards. Additionally, if a user wishes to view an orthographic projection, the P key toggles this feature on and off.

Moving the mouse cursor affects the camera view and allows our user to fully utilize the keyboard movement options within a 3D space to fully explore the world and lighting offered in the scene. Additionally, the mouse wheel allows the user to zoom in and out so that they may look more closely at details and textures.

There are several custom functions included in the source code for this scene. The first is setCoords(), which handles calculations for defining vertices that are used to build a torus. This function can be re-used/re-purposed in creating or modifying any additional torus and could be overloaded or modified to further improve the coil, or in the implementation of any additional shapes like it.

The next custom function is a loadTexture() function. When working on lighting and the Phong model implementation it was important that I was able to quickly and efficiently load and map textures. This function uses a (char const\* path) argument to allow us to quickly bind a texture and generate a mipmap. This function can be utilized in implementing textures fully capable of complex lighting and is likely how I will add any new textures into the scene moving forward. This function returns the unsigned int textureID.

The last custom function is createObject(). Create object is a function which handles actually building the torus’ that are used extensively in this scene. Within createObject() there is a nested for loop that calls the setCoords() function to set vertex values that will be used in rendering. As with setCoords(), overloading or modifying this function could be used to create additional or more complex shapes.

I hope that you enjoy my 3D scene, and it was a learning experience for me. Moving forward I would like to explore other possibilities within my scene, such as animating motion and eventually integrating potentially a moving fire.

Dylan