For the development of my 3D scene using OpenGL, I started by selecting a few objects that contained various primitive shapes. In OpenGL, objects can be constructed using primitive shapes such as triangles. A complex object is an object that is created by using more than one primitive shape. Two of the objects I selected to model in my 3D scene are complex objects. The first complex object is a replica of a video game item known as an energy tank. I was able to recreate this object using three cylinders stacked on top of each other. The top and bottom cylinders are the same size in height and diameter. The middle cylinder has a diameter slightly smaller than the diameter and larger height compared to the top and bottom cylinders. The object is blue and has a black letter “E” on it. My second complex object is a coffee mug with a round handle attached to it. I was able to create this by making a cylinder without a cap on the top to represent the mug. I rendered a smaller cylinder without a top or bottom cap and rotated it on its side to model the mug’s handle. The other two items I chose to model in this scene are a lithium-ion battery and a pinball. The lithium-ion battery could be modeled using a cube and the pinball was modeled as a sphere. I chose these objects because they represented a mix of simple and complex objects that could be constructed using various primitive shapes.

The scene I created is dynamic and can be interacted with. A user can interact with the scene using various input devices that are commonly used with a computer such as a mouse and keyboard. There are several keyboard keys that are used to navigate the scene as well as toggle options. For example, pressing the ‘p’ button on the keyboard toggles the viewing perspective between perspective and orthographic views. You can press this key at any time while navigating the scene to change the perspective type. For navigation, the user can zoom or move forward and backward using the ‘w’ and ‘s’ keys. The ‘a’ and ‘d’ keys can be used to move the camera left and right respectively. The ‘q’ and ‘e’ keys can be used to move the camera up and down. Using the keyboard alone is not always the most ideal configuration for navigating a 3D scene. The mouse can be used to change the camera’s viewing angle without moving the camera’s position. This can enable a user to stand still and move the camera around as if they are turning their head to look from a different angle. Lastly, I added some controls to the mouse scroll wheel that allow a user to increase or decrease the speed at which the camera can move around the scene. Pressing the scroll wheel down like a button resets the camera speed to its initial value.

During the development of my 3D scene, I implemented several custom classes and functions that allowed me to modularize my code. For example, I created a Cube class that allows me to create Cube objects. The cube object holds all the information needed to render a cube on the screen. This information includes the vertices that will be drawn and texture images that will be used when displaying the object in the scene. I used a sphere class that I found on an OpenGL tutorial website and customized it to make it work the way I wanted it to so I could easily render Spheres. Similarly, I used code from a tutorial we reviewed in this course that could generate the vertices and other information required to draw a cylinder. I also customized the Cylinder class and added new features such as flags that can be set during the construction of a Cylinder object that determines if the top and bottom cylinder covers should be rendered. For my coffee mug, several of the cylinders do not need either top or bottom covers. I changed the Cylinder class function called draw to include true or false parameters that indicate whether the top and bottom caps are rendered when the function is called. Another example of a custom function I implemented in my main source file is called USetDefaultTextureParameters(). This function was extremely helpful because it made my code shorter and easier to read and maintain. The code is shorter because instead of writing the same few lines of code multiple times, I can call the function which only requires one line of code per object. Keeping the code modularized and well organized allowed me to keep code related to certain objects or functions in one place. Having organized code makes the code easier to maintain. For example, if my scene is not rendering a cube correctly, I know that there is a mistake somewhere in the cube class. This saves me from having to look in other places when trying to determine what the issue is and how best to resolve it. I learned a lot about coding, OpenGL, 3D graphics, and debugging throughout the course of this project. These skills will help me further my academic and career goals by helping me become a better programmer.