For my 3D scene, I chose four objects that were easy to deconstruct into simple primitive shapes so that it would be easier to represent them in a 3D graphical environment. The four objects I chose were Christmas ornaments that were a cotton candy cart, a cupcake, an ice cream bar, and a donut. I chose these objects because they were easy to deconstruct into simple shapes. The donut was the easiest, since it does not need to be broken down as a donut is simply a torus. The cupcake and ice cream bar were the next easiest to deconstruct, as both were only two primitive shapes. The cupcake consists of a cone and a cylinder and the ice cream bar of two (scaled) cubes. The last object, the cotton candy cart, seemed a little more tricky but it too was easy to deconstruct. The cotton candy cart deconstructed into a cube for the base, a sphere for the glass ball, a cylinder for the topper, and two cylinder for the wheels. Breaking all of these objects down into primitive shapes made it possible to recreate them in a low-polygon 3D environment.

A user can navigate the 3D scene by using a keyboard and mouse for input devices. The keyboard takes input from the following keys: W, S, A, D, Q, and E. These keys control the camera movement in the 3D space. For forward movement you press the W key and for backwards movement you press the S key. Likewise for leftward movement your press the A key and for rightward movement you press the D key. And for upward movement you press the Q key and downward movement you press the E key. The mouse provides additional camera control through the additional mouse movement and scroll inputs. Through the movement of the mouse, you can change the orientation of the camera and scrolling the mouse changes the camera movement speed. Combining all of these inputs together, I find the easiest way to move the camera through the 3D scene is by mainly using the W key for forward movement and the mouse movement to change the orientation of the camera. This makes it easy to move around the 3D scene, while you can still use the other keys to make small corrections to your movement.

I have several user-defined methods that make my code more modular and organized. I will go through each method I created and explain whether it makes my code more modular and/or organized. My first method is the *initOpenGL()* method. This method contains all of logic to initialize and run OpenGL libraries and registers the following callback methods I defined as well: *resizeWindow(), mousePositionCallback(), mouseScrollCallback()*. The *initOpenGL()* method is only run once, so it does not provide modularity to my code, but it does help organize it and make it more readable. The callback methods, however, are called several times. They are called specifically whenever the window changes sizes, the mouse moves, or the mouse scroll wheel moves which help modularize and organize my code. The *processInput()* method is modular and is called once per frame to check if a key event was pressed on the keyboard. The *createMesh()* and *destroyMesh()* methods are only run once, but they help organize the creation and deletion of all VAOs and VBOs for my objects. The *createAndBindBuffers()* modularizes my code by abstracting away the code necessary to set up each objects VAO and VBO. The *createTexture()* and *destroyTexture()* methods are run once for each primitive shape in the 3D scene, which make these methods modular and also help organize my code. Lastly, the *render()* method is called once per frame and contains all the logic to render a single frame of the 3D scene. This method is not so much modular, as it only has one use case (rendering a single frame in the while loop), but it does keep my code very organized and abstracts most of the functionality of rendering a single frame into once method.