For my projects development I chose to go with a progressive approach where I found out what worked and then organized the code after. For the most part I tinkered around and tried to figure out how to do something before I figured out how to optimize it or make the code clean and modular. This often involved me building some method, that started with zero input parameters and that returned whatever I needed (fully hard coded) then, once I figured out the solution or the issues, I would restructure the code until I had a reusable product. From there I often modularized the code and modified it to be as general as I could, depending on other code until I determined the functionality was required more often than not. A great example of this was my shapes class where I held the entirety of my mesh code for several weeks. I tried to find the simplest way to recreate an object before moving onto combining shapes, building new shape functions as needed when I determined another shape could not reasonably fill the requirements for some reason. For example, I initially used circles as a way to build spheres, cones, and cylinders, but later realized these were better left to other functions specifically tailored for the job. From there I ended up finding half spheres (or rather only drawing half of a full sphere) to be the best alternative for many different shapes. Especially when it came to normal vectors, half spheres were significantly easier to work with than full spheres. Like I said in the beginning of the class I chose my objects partially because they were available, but partially because I viewed them as a challenge. While If I removed 1 or 2 of my bullets, I would have been able to hit the 1000 total triangle’s goal, I decided to leave them in as they better demonstrated the scene and the capabilities of the code than the majority of the other objects. For combining meshes I needed a better approach then using shapes alone and repositioning them, let alone texturing and normalizing them. For this I chose to build an Assets class, with a slightly different struct to handle multiple meshes. In this way I was able to combine multiple shapes and textures into one mesh and selectively draw each shape and texture while keeping the objects attached. After the portion of combining textures and meshes together was solved, it made sense to handle each in a single vector as individual changes were easily understood whereas in on large mesh the code would have become convoluted and difficult for anyone to quickly read and understand.

The user can navigate the scene with several keyboard keys and can position their view with the use of the mouse. While I will not take a significant amount of the credit for this as the foundations for the movement were built out of the tutorial, I did make some changes as I deemed necessary or reasonable. The use of keyboard keys and mouse movement seemed obvious as this is how most first, and third person games are developed, and this control scheme is effectively a second nature for a good portion of the gaming community. The intention was to replicate the camera control of these games by allowing users to control the directional movement with WASD, altitude with QE and speed with the mouse wheel (alternatively shift). From here the mouse can be primarily focused on changing the angle of view with its 4 major directions of UP, DOWN, LEFT and, RIGHT. These directions or a mixture of them allowed the camera view position to remain the same while the camera angle changed to match the movement of the mouse.

As I discussed above, I made a number of custom functions to facilitate the building of this project. While the camera and shader classes are more from the tutorials than anything this code is effectively entirely reusable in other OpenGL projects and is definitely a good base to use for other projects in the 3D graphics space as well. The shapes class is almost entirely reusable in another project or framework that uses similar concepts, and the assets class as an extension of shapes is as well. Finally, the shaders written for this project are likely going to remain relevant (with some modification) for many projects to come as they generically handle input data that conforms to their requirements. Effectively, with the shapes class I can use it in 3D graphics and mathematical applications as I can use the normal calculations to determine the which way a face of a shape is facing. The Assets class, while more specific towards this project provides a fairly straightforward framework on how to combine primitives into more complicated shapes and while specific for the assets here, the code included could help build other projects for years to come.