Nathaniel Holcombe

CS 330

Project Reflection

02/21/2021

In the process of completing this project, which was the culmination of the entire term’s content, I made many development choices, some of which have resulted in a sub-par end result. Namely, my adherence to the Github tutorial style encapsulation and modularity seemed to have added some unhelpful complication which also worked against reusability. I noticed during the course that the instructor-led videos showed a better-organized, simpler and differently-structured approach which abstracted the mesh initialization, rendering, and texturing functions in what seemed to me to be a superior manner.

The simplicity of the approach of Joey de Vries of LearnOpenGL also proved superior to the overcomplication of the Github tutorials. While I believe that I have benefitted from being exposed to multiple approaches, I had accumulated over 1700 lines of code by the end of the term, which became more of a burden to reorganize with each new milestone. While my project isn’t what I’d consider “spaghetti code”, it has obscured the order in which the graphics pipeline operates. Meanwhile, I was also concerned from week 3 onward whether I would be better off diverging from the GitHub tutorials, but I feared that this venture would bring a high risk of failure later in the class. I chose to “trust the class tutorials” because I was afraid of what could happen if I didn’t; if I’ve learned anything in my education, it is that nothing is ever as straightforward as I expect, and so taking risks with projects is not wise.

If I were to do this project over again, I would mostly ignore the GitHub tutorial material and stick with de Vries’ and the instructor videos as references, which would combine the encapsulation of object rendering with the sequential function stack used by de Vries. While de Vries placed everything in the main method and did not encapsulate, I would instead make my main method call a set of functions for each step in the graphics pipeline, for each object. These functions would be defined elsewhere in the file, and would take world space location parameters to more easily place them in the viewport. Likewise, this would make it clear how to add further functionality, such as changing perspectives.

I chose my objects because I thought the process of recreating them would be interesting and an appropriate challenge for my capabilities. For the most part, I was successful, as the only deficits are that I could not cause the sphere (exercise ball) and subsequently the lighting objects to appear, and I cannot figure out why. I believe if I had followed the other approach that I mentioned earlier, the project would have been more easily troubleshot. The lighting objects had appeared before I attempted to create the sphere, and it may be that I inserted or deleted something that was integral to rendering the lighting objects in the fragment shader, or perhaps elsewhere. Nevertheless, the project does include the functionality to navigate the scene, set the camera location, render with a perspective display (not orthogonal), and apply lighting effects to the textures.

The scene is navigated using a combination of the keyboard and mouse. The keyboard w, s, a, d, q, and e keys control cardinal movement, while the mouse movement controls the view direction and speed of movement (mouse wheel). These are methods of control with which most people who have played 3D video games are familiar. The combination of movement capabilities was made possible by OpenGL’s built-in GLFW getKey and mouse callback functions. These work by monitoring for user feedback at each frame rendering, using the input to calculate the resulting camera position and thereby the image in the viewport.

As I mentioned, the custom functions in the project were not well organized or reusable. The URender function contained the rendering logic for all of the objects in the scene, whereas separating them by including them in their respective “draw” functions would have been more readable, orderly, reusable, and efficient to create. The UCreate functions seemed to be an effective way to encapsulate the objects, as were the fragment shaders. The UDestroy functions were straightforward in their purpose and easy to use.

Overall, as a somewhat artistic and scientific person, I found this project and the process of gaining the skills to complete it to be very interesting, and something that I will continue to work at. It was a considerable challenge to absorb the many concepts that comprise OpenGL, but I have no doubt that this was a good way to establish a foundation of knowledge in order to understand graphics suites and editing software in the future.