Dre’ Scheetz

10/23/2022

CS-330

Kurt Diesch

Project Reflection

Reflecting on the decisions made during the development of the 3D scene utilizing OpenGL and C++, the most significant takeaway is my underestimation of implementing code beyond my scope of understanding and experience. Attempting to debug where the issues and conflicts resided took a lot of trial and error and reinstallation. However, I am confident with the result and believe my mistake has helped me grow as a computer science developer and student. The scene features a simple desk with a globe, a laptop, a pencil, a beverage, and a book.

Development Justification:

While I could not anticipate the arduous nature of using two separate code bases and integrating a library, the scene benefits from the inclusion greatly. With the project in its current state, including more objects or functionality would be simple. Assimp, a library for importing models, and learnOpenGL header files, allow more complex shapes to be used. By incorporating .obj files, manually inputting every object's vertex data is no longer necessary. The original cylinder included in the milestone was created in Blender then the vertex data for the mesh was manually formatted. Including objects is much faster now, without cluttering the project's code base.

Navigation of Scene:

Relying on learnOpenGL, movement and camera control are primarily handled by learnOpenGL's camera header file. To navigate the OpenGL scene, a user must use the keyboard's A, S, D, Q, W, E, and P keys. Using the provided code of module 3, adding the upward and downward controls required changing a few variables in the UProcessInput function:

if (glfwGetKey(window, GLFW\_KEY\_Q) == GLFW\_PRESS)

gCamera.ProcessKeyboard(UP, gDeltaTime);

if (glfwGetKey(window, GLFW\_KEY\_E) == GLFW\_PRESS)

gCamera.ProcessKeyboard(DOWN, gDeltaTime);

Projection is handled by the keyboard input GLFW\_KEY\_P to toggle a bool value by assigning its inverse and changing the projection with an if/else statement.

Organization and Modularity:

One of the most significant changes to the codebase is the inclusion of model loading. Model textures and mesh data are stored inside the objects folder in separate folders. The change enhances the modularity of the project and allows a user to quickly incorporate any object or remove them. Another change worth discussing is the change to the model class. The object's individual lighting needs, scale, position, and rotation are kept neatly within the constructor's arguments by keeping model transformations and data as variables in the class.

Lastly, many functions have been migrated into individual components or where they are needed. UcreateMesh, UDestroyMesh, UCreateTexture, and UDestroyTexture are now handled by the mesh and model class. Shader methods are now contained and easily modified in the shader header. URender has been altered to iterate over the models constructed, contained in the model class vector. It also sets up the shader values to be used on the models. While there is room for improvement by allowing for different shader uses, the changes have been beneficial overall.