Design Decisions

CS-330-T1190 Comp Graphic and Visualization 22EW1

Valerie J. Smith

October 22, 2022

The 2-D image I have chosen to recreate in 3-D is an image that is very special to me, as it is an image of my house, which is over 100 years old. The house holds special memories to me as not only did I hang onto the house no matter what state other than Pennsylvania I was living in; I also remember driving past the house as a child. As a child I would gaze up at the house from the road and always felt drawn in by the looming structure. The bank of the house facing the main highway always (and still does, although not shown in this image) contained wild lilies that were in bloom in the late spring and early summer. When I purchased the house, I was surprised to find that it had belonged to a cousin of mine and had been in my father’s family for many years. I have owned this house for over 20 years and have recently moved back to reside in it here in Pennsylvania to fulfill my dream of securing the structure of this house, hopefully for the next 100 years to come, and in hopes that there will be someone after me who loves it as much as I do.



The development choices for this 3D scene were quite difficult as I had chosen an outdoor scene. Coming into the course with no prior knowledge in 3D imaging was a barrier for me, and in retrospect, I would’ve chosen a different image from the start if I had any idea of what was expected as an outdoor scene required quite a bit of texture work in order to make the scene appear to be outdoors. Therefore, the selected objects for this project consist of items necessary to configure the exterior view of a house, which is a combined cube and inverted pyramid, along with a foundation block that is also a cube. A pyramid is also shown in the image, which was not originally planned for. I had intended it to be a tree but was at the limit for adding additional images for texture. There is also a lamppost with a lamplight included in the scene, which are two 3D combined cubes. The scene sits upon a plane that is textured with leaves.

Programming for this scene was difficult at best. The vertices are abstracted from the view by the use of a Coordinates.cpp file and a Coordinates.h file. This abstraction helped to work with the individual objects faster and helps to organize the code. As all of the objects contain texture, a lot of effort was put into finding images and ensuring that the coordinates would be correct in order to accurately reflect the proper appearance of the texture to the objects.

A user can navigate around the 3D scene by using the WASDQ and E keyboard characters as well as by using the mouse for the same functionality. The user can also use the P keyboard character to get the orthographic projection of the scene. The camera functionality is organized by the inclusion of the camera.h file. The camera is controlled by setting the view matrix for the camera in the view, with the perspective of the camera set in radians based on the width of the window of the scene. These coordinates are passed to the shader program. The GLFW library contains keyboard shortcuts that will manipulate the view of the scene in order to mimic the camera movement and camera speed. The GLFW library also contains a similar functionality in order to listen to and provide a callback for mouse movements.

There are a lot of custom functions in this code that help to make the code more modular and organized. The UProcessInput function gets the camera up vector and uses this vector multiplied by the camera offset in order to find provide the up and down functionality and it is very handy to have this function written once and used throughout the scene. The URender function sets the basis for what is needed to render the view. Once the rotation, model, camera view, and projection are defined, it passes the model view and projection information to the shader program. The next set of functions tell the program details about how to draw the objects in the view. If another object is to be added, it is simple to add another one as they are set up in the same manner.

The UCreateMesh function is organized into separate sections depending on which set of vertices are being used to draw the objects. As the vertices are abstracted to their own file, each mesh is assigned one of the sets of vertices from that file. The coordinates, or vertices, are assigned to a vector in this function with the number of vertices defined in an array. This function will loop through the vertices and draw the objects as instructed. This abstraction helps to keep the code cleaner and easier to understand, and also helps when testing the objects as the implementation can be worked on in a different file without affecting the main source code. This also makes it easier to add additional objects as it is clearly defined on where the objects need to be placed in the code.

Almost all of the functions in the code can be considered reusable as this code could be refactored simply by reorganizing the vertices in the Coordinates file and ensuring that the objects have the correct number of vertices reflected in the mesh in the corresponding source file. The images can also be easily swapped out simply by adding an image file to the project and changing the file name in the source code. The code that is written to control the camera, keyboard, and mouse functionality can also be reused in another project. The code that creates the lighting for this project could also be reused as a basis for lighting another project.

