Quintin B. Rozelle

10/11/2023

7-1 Project

CS-330

|  |  |
| --- | --- |
| Original | Rendered Scene |
|  | A screenshot of a computer  Description automatically generated |

**Justify development choices for your 3D scene**

I chose the objects that I did largely because they fit the requirements of the scene and seemed relatively straightforward to build and render with primitive objects. My complex object (the large Lego brick) was built from a cube and cylinders. The salt container was also built using a cylinder. The pyramidal Rubix’s cube was built using a three-sided pyramid. And the watermelon toy was a sphere. At first, I assumed I could position the sphere such that the bottom half extended below the plane, effectively hiding it from view. But after learning more about OpenGL and hot it works, I realized that I could just render only the top half.

I chose the lighting that I did (two point lights above with a slight yellow tint and one directional light from the left) as I thought this mimicked the original lighting fairly closely. The two point lights above are meant to simulate the lightbulbs in the real world scene while the directional light is meant to simulate sunlight coming in from a nearby window.

Some of the textures I chose were meant to mimic the original scene while others were chosen to add a bit of artistic license to my rendered scene. The Rubix’s cube and watermelon are meant to be close approximations of their real-world counterparts, the countertop is meant to look like marble but with a different color, and the Lego brick and salt container were given new textures to try something different. Additionally, the countertop and the Rubix’s cube use specular map to help add some realism to these shapes by creating more realistic specular reflections:

* The Rubix’s cube doesn’t reflect specular light on the black lines (where the subsections of the Rubix’s cube would be separated in the real world)
* The counter top doesn’t reflect specular light in the grout lines as those wouldn’t be shiny in the real world. Additionally, the counter top has other random areas that specular light is reflected less well to add to the illusion that the marble isn’t perfectly smooth and reflective.

**Explain how a user can navigate your 3D scene**

My scene uses a combination of callback functions and an input checking function within the render loop to control navigation:

* mouseCallback – This callback function controls the orientation of the camera. Using the movement of the mouse, it will adjust the yaw and pitch of the camera. Pitch is capped at 89 to -89 degrees to prevent the camera from flipping by rotating too far.
* scrollCallback – This callback function controls the speed of the camera movement. Scrolling up will increase the speed while scrolling down will decrease the speed. When the mouse wheel is moved, a variable is updated which is used in the UProcessInput function to control the rate of camera movement. This speed is capped at 1 to 20.
* keyCallback – This callback function controls the toggling between orthographic and perspective projections. The scene defaults to a perspective projection but pressing the “O” button will swap between a Front, Right, and Top orthographic projection. Camera yaw and pitch is disabled while in orthographic view. Additionally, camera position, pitch, and yaw variables are saved with swapping to an orthographic projection and restored when moving back to a perspective projection.
* UProcessInput – This is an input checking function called during each loop of the render function. It controls the camera movement along the X, Y, and Z axes. It uses the movement speed set with the scrollCallback function to adjust camera speed. Additionally, it normalizes speed among different computers by measuring the time between different frames and factoring that into movement speed. It was chosen to put this functionality into a different function (as opposed to a callback function) in order to allow continued camera movement if the movement keys are continually pressed. The callback functions are currently designed to run only when a key is pressed and not again until that key is pressed again.

**Explain the custom functions in your program that you are using to make your code more modular and organized**

My program uses a variety of custom functions to help organize and modularize my code. For instance, UInitialize is used to run the standard initialization steps for OpenGL and URender is used to render the scene. Beyond the more boilerplate custom functions, I have also created a few more customized ones to assist with creating this specific scene:

* UCreateTextures – This is based off a typical texture loading sequence using stb\_image.h but adds a few features to improve the usability when multiple textures need to be loaded. I have put the standard sequence within a loop and a vector of file names is passed to it. Additionally, another vector is passed by reference to which the loaded texture locations is saved for future use. Now, instead of needing to call a load textures function for every texture to load, it can be passed a vector of file names and another for storing the locations and textures can then be access by referencing the appropriate element in the storage vector. Further, this also allows additional textures to be added at a future point without needing to modify texture loading code as this function will always cycle through the entire vector of file names (assuming a texture doesn’t fail to load part way through).
* uDrawLegoStud – My scene uses eight different cylinders to model the studs on the Lego brick. Apart from the location in the scene, they are identical. To prevent needing to retype the code to render these cylinders eight different times, I put that code within its own function. This function is then called eight times and the final stud location is passed as a parameter. This improves the code by making it smaller and more organized while also allowing for easy modification of the studs. Instead of needing to modify each individually, the uDrawLegoStud code can be modified once and affect all eight studs.