Design Decisions  
Joshua Lewis

08/19/2024

The objects I chose for my scene consist of objects that I have a somewhat sentimental attachment to. These objects consist of a Rubik’s Cube and Dice Set. The Rubik’s Cube I chose because I keep one on my desk to fidget with often and the Dice I keep on hand as I am a Tabletop RPG fan (Dungeons & Dragons, Pathfinder etc.) The dice have always piqued my interest as the various polyhedrons are geometrically (and mathematically) stimulating. Additionally, I have seen dice sets made of various materials, from plastic, to metal, or even gold and gemstones. I knew the dice would be a fun, yet challenging task to recreate.  
 To program the required functionality, I began by analyzing the code that I had been provided. I noticed that the code itself needed various improvements in terms of resource management, resource / memory management and overall structure. I began segmenting the large code base into smaller, more manageable scripts that each served their own purpose. From there I improved the memory management via the use of smart pointers and adjusted the existing functionality by ensuring that the various classes / resources were being used and released as necessary. I also created a class to generate shapes based on passed parameters. This substantially shortened the overall code required to generate the meshes within the SceneManager class and render the scene. It also helped to make the code clean, concise and clear. Creating the various meshes for the polyhedrons (the Tetrahedron, Octahedron, and Decahedron) proved to be quite the challenge, additionally, I adjusted function for the sphere mesh to calculate vertices dynamically based on passed parameters. The Rubik’s Cube itself was rather simple overall, using a For loop I was able to dynamically generate the positions of the smaller cubies and set the transformations using my Generate Shape class. Although the Rubik’s cube still has a long way to go, as I found that it is a complex structure to recreate.  
 Users can navigate the 3D scene with a mouse and keyboard by using W, A, S, D, Q, R (Forward, Backward, Left, Right, Up, Down) and can use the scroll wheel to increase or decrease the speed of the camera. Pressing TAB will toggle between perspective and orthographic view. To implement this, I first began by locking / hiding the mouse cursor to the center of the generated window (this is an important, often overlooked step) Using the ProcessMouseMovement function, I was able to use the offset of the current mouse position compared to the last frame to determine the amount the mouse had moved between frames. The camera position is then updated accordingly based on the offset. The ProcessMouseScroll function works similarly as it processes the offset of the scroll wheel and adjusts the offset of the camera on the negative y axis (simulating increasing / decreasing the “Speed” of the camera.) Using the glfwGetKey function, I created a bool and check to use to determine if the Tab key had been pressed and toggled the view between Orthographic & Perspective views accordingly using glm::ortho & glm::perspective.

Overall, my changes have made the code cleaner, reduced resource usage, improved memory management, improved compilation, and ensured that the code will be modular and easily adaptable for future projects. The functions / methods and classes are each designed to handle a specific task. This encapsulates the necessary methods of a class which makes the code robust, and less prone to errors, yet still allows for ease of use, and modification.