Christopher Keenan

06/22/2025

Computer Graphics and Visualization

Professor Graham

*Final Summary*

For this final project in Comp Graphics and Visualization, I aimed to recreate a symbolic 3D scene which was inspired by Iron Man’s arc reactor display, featured in part in the popular Marvel series. The goal was not just to build a functioning scene with lighting, camera control, and interactive capabilities, but also to creatively express an homage to technological legacy through appropriate modeling, shading, and placement. The selected objects such as the base platform for the reactor, the reactor core itself, and the representation of the miniaturized iron man helmet within the reference image, as well as the stylized display placard that looks like something out of Stark labs augmented reality computer displays were chosen for their iconic presence in Marvel’s depiction of Iron Man and the subsequent story. They were also chosen for the modeling challenge they presented using basic geometric primitives available to me in the Module 7 file structure.

The platform and base serve as the foundation of the scene. I utilized a scaled cylinder mesh to achieve a disk like shape, which I placed at ground level using the SetTransformations function with appropriate vector values relative to the included plane which defined ‘ground level’. The reactor, represented as a smaller, scaled cylinder, was textured with SetShaderTexture(“reactor\_tex”), and was then precisely positioned to rest centered on top of the wooden platform. The helmer was the most intricate modeling challenge of the projects exercise, as it was built using a combination of a red sphere for the dome and elongated, reshaped boxes and cylinders for the jaw and side components. While it was admittedly abstract in detail, the positioning and color selection using functions like SetShaderColor() created a clearly recognizable comparison. I adjusted the components z positions to appear slightly behind the reactor but in the foreground of the display placard. To finalize the scene structure, I aligned the display placard base and nameplate using corrected translations to ensure their positions rested on the external base, completing the display structure shown in the selected reference image.

Camera movement and viewing were handled in ViewManager.cpp, using the mouse for yaw/pitch orientation and the keyboard for translation. Mouse control was implemented in Mouse\_Position\_Callback(), where the offsets between frames updated g\_pCamera->Yaw and pitch, then recalculated the front vector. Zoom control was added via scroll input using Mouse\_Scroll\_Callback(), which modifies movement speed dynamically. Standard keyboard inputs such as W, A, S, D were implemented to move around the scene. Q & E were added to handle movement vertically in the scene, Q for movement up and E for movement down. These allow the camera to move freely around the scene. This was all handled in ProcessKeyboardEvents() by modifying the camera’s position relative to the front and right vectors. This gives the user 6 degrees of movement with all controls included. Perspective mode toggling was also implemented, utilizing keys P & O to switch between perspective and orthographic views. This is reflected during rendering in PrepareSceneView() by setting up an appropriate projection matrix with glm::perspective() or glm::ortho().

To maintain a clear and modular structure, I utilized custom transformation and rendering wrapper functions such as SetTransformations() and SetShaderTexture(). SetTransformations allowed reuse across all 3D components, ensuring consistent object placement and ultimately simplified any further modification. Similarly, SetShaderTexture() modularized assignment of necessary textures. This design separated texture binding from the actual draw calls and allowed for rapid prototyping of different texture models for use in the final render. Coloring was also handled through SetShaderColor, which allowed me to utilize a preset for assigning colors onto 3D objects within the scene quickly and efficiently. These functions all contributed to making the code reusable and helped avoid making syntax errors. Instead of repeatedly writing transformation matrices or shader bindings, each mesh component needed only a few intuitive function calls, keeping the SceneManager source file relatively clean and easy to read.

Overall, the scene balances technical implementation with artistic vision. While more refined modeling could enhance the realism to a near replication of the originally chosen reference image, the current build successfully met the project objectives, with proper structure, modularity, and interaction.