Project Two Reflection

CS 330

Daniel Lowery

I chose to build a familiar desk scene populated with everyday itemsa pen, monitor, keyboard, mouse, mug, and a handful of pencilsto ground the viewer in a recognizable environment while also showcasing a variety of primitive meshes. By arranging cylinders, cones, boxes, spheres, and a planar floor in logical positions (for example, pencils clustered in a cup, notebooks stacked neatly), I was able to verify that my transformation routines correctly scale, rotate, and translate each object into world space. Using realistic proportions helped ensure that each mesh’s parameters felt intuitive and made debugging easier whenever something looked out of place.

To satisfy the project’s interactivity requirements, I implemented a first person camera system with both keyboard and mouse controls. Movement along the forward/backward axis (W/S), strafing left and right (A/D), and vertical translation (Q/E) allows the user to explore the scene freely. Mouse look reads cursor deltas to adjust yaw and pitchclamped between –89° and +89° to prevent gimbal lock and the scroll wheel dynamically modifies the movement speed, giving finer control when navigating tight spaces or zooming quickly across the desk. Toggling between perspective and orthographic projections using the P and O keys provides two distinct ways to view the scene: perspective mode delivers natural depth and foreshortening, while orthographic mode flattens the view and is useful for examining object relationships without perspective distortion. Altogether, this input scheme combines the familiarity of an FPSstyle interface with simple projection controls that let users understand the effects of different viewing models firsthand.

Overall I kept the rendering code organized through a set of modular helper functions in the SceneManager class. The SetTransformations() function centralizes modelmatrix construction from scale, rotation, and translation parameters, so every mesh simply calls it before drawing. The SetShaderColor() routine handles uniform uploads for RGBA color and toggles texturing and lighting flags, streamlining material setup across shapes. PrepareScene() consolidates all mesh loading into one place to avoid redundancy, and RenderScene() sequences each object’s draw call in a clear, readable fashion. Thanks to these abstractions, adding new objects or experimenting with alternate materials requires minimal changesjust adjust parameters or insert additional draw calls without rewriting transformation logic.

Through this project I reinforced best practices in modular code design, efficient shader uniform management, and intuitive camera navigation. The end result is an interactive 3D desk environment that feels cohesive, performant, and easy to extend.