Final Project Reflection

I chose a small, believable vignette instead of a sprawling environment: an upright piano with bench, a table lamp, a framed poster, and wood floor, and a painted back wall. These objects give me a mix of box like, round, and tapered primitives (boxes, planes, cylinders, cones, spheres) so I can demonstrate transformation composition, UV mapping, materials, and shadows without importing external models. They also echo the reference photo, a realistic anchor that guides proportions and layout.

Lighting is a two spot setup. Light 0 is a warm, ceiling mounted key light aimed at the bench and keybed; Light 1 is a cool, wide fill that prevents pure black in the shadows. This arrangement mirrors a simple photographic lighting rig and gives me clear, observable interactions across glossy (piano wood, black metal) and rough (wall paint, floor) materials. The lamp geometry is present for realism and to cast shadows, but it is not an emissive light source; keeping illumination limited to the two spotlights helps me isolate and debug shading and shadowing behavior.

Textures were chosen to carry most of the visual storytelling: a piano wood map with high specular response, a light wood floor with broad grain for scale cues, painted wall for a diffuse, rough background, and black metal / stainless for the lamp. I added three subtle "scratch / wear" overlays that can be toggled and tiled independently to break up repetition on large surfaces (e.g., the piano cabinet and benches).

Most of the required functionality is delivered through a sequence of reusable steps that repeat for every draw: set transforms, set material and textures, then issue the mesh draw call. Mesh preparation occurs only once, the program loads a shared set of primitives (box, plane, cylinder, cone, sphere, and tapered variants) and then reuses them throughout the scene.

CreateGLTexture handles image loading via stb_image, sets filtering and wrapping, generates mipmaps, and registers a string tag for later lookup. During rendering, objects refer to textures by tag rather than by numeric handle, which keeps the code easier to read and modify. Material control is similarly centralized. A single function pushes ambient, diffuse, specular, and shininess values to the shader just before each draw, so switching from a matte wall to a glossy piano becomes a one line change instead of scattered constant edits.

Shadow mapping is implemented with a dedicated depth pass. InitShadowResources builds a framebuffer object with a depth only texture that serves as the shadow map. At the start of each frame, UpdateLightSpaceAndBindShadow computes a light space matrix from the key lights position and direction and binds the shadow texture for later sampling. The program then renders all scene geometry again with a bShadowPass uniform enabled, which makes the vertex shader output light space positions and writes only depth. Because the depth pass reuses the exact same transformation calls as the visible pass, the shadow edges remain stable and self consistent. After the depth pass, the program restores normal rendering, samples the shadow map in the fragment shader, and applies a soft comparison to darken fragments that are occluded from the light. This approach is robust and widely used, and it fits the scene well because the piano, bench, and lamp all produce crisp, readable case shadows.

Navigation is intentionally familiar so users can evaluate lighting and materials from any vantage without learning a custom tool. The keyboard translates the camera forward, back, and laterally, with a vertical rise and lower for quick reframing. Mouse drag adjusts yaw and pitch, and the scroll wheel speeds up or slows down the camera speed. Each frame the view and projection matrices are updated and the camera position is sent to the shader so specular response tracks the viewer.

Modularity comes from a small set of reusable helpers that encapsulate the render steps. Set transformation composes scale, rotation, and translation into a model matrix and derives a normal matrix, which keeps lighting correct after any transform. Texture utilities load images once, look them up by tag, and expose per object UV tiling; a lightweight scratch overlay can be toggled and retiled to add wear to any surface without changing geometry. Lighting and shadows are likewise factored out: one function establishes the two spotlights, another initializes the shadow map, and a per frame updater binds the light-space transform. Because every draw call goes through the same pipeline, set transforms, set material, draw mesh, the code stays readable and easy to extend.

These choices balance realism and scope. A compact set of everyday objects, piano, bench, lamp, poster, wall, and floor, provides varied geometry and a mix of matte and glossy materials while remaining tractable to tune. The two light rig and shadow pass deliver clear speculars and anchored contact shadows without the overhead of a full global illumination system. With more time, I would expose light parameters in a small UI, add percentage closer filtering for softer penumbras, give the lamp shade thin shell translucency, and migrate to a physically based material model so wood, metal, and paint behave consistently across different lighting setups.

References

- de Vries, J. (2023). *LearnOpenGL*. https://learnopengl.com
- G-Truc Creation. (2024). *OpenGL Mathematics (GLM)* [Computer software].

https://github.com/g-truc/glm

- Barrett, S. (2017). *stb_image.h* [Computer software]. https://github.com/nothings/stb