## **Milestone Two Narrative (Software Design Only)**

**Artifact & When:**I enhanced my **CS 330 OpenGL/C++ desk scene** project (originally created Fall 2024) to improve **software design and engineering** quality, focusing on resource lifecycle, encapsulation, configuration, and defensive error handling.

**Why include it:** This artifact now demonstrates professional design practices in a real graphics codebase: correct GPU resource management, improved ownership semantics, encapsulated view/camera state, and centralized configuration, not just working output. These are the habits that make graphics applications stable and maintainable.

**What I changed & how it improved:**

* Fixed a critical lifecycle bug by replacing a mistaken glGenTextures call with glDeleteTextures in DestroyGLTextures(), ensuring textures are properly freed and internal counters reset.
* Replaced file-scope global state in ViewManager (window size, camera pointer, timing, projection flag) with a **ViewConfig** struct and **class members**. This improves encapsulation, testability, and clarity of ownership.
* Adopted **std::unique\_ptr** for manager ownership in main, removing manual delete calls and clarifying lifetime.
* Strengthened **defensive error handling** at initialization and uniform-setting boundaries to fail fast and avoid undefined state.

**Course outcome progress:**

* **Innovative techniques/tools (software engineering/design):** Applied RAII and configuration patterns to a graphics codebase.
* **Professional communication:** The code now documents ownership and lifecycle guarantees, and the enhancement is explained in this narrative.
* **Security mindset (design flaws):** While not a security feature, defensive checks reduce crash surfaces and undefined behavior typical in graphics apps.

**Reflection:**The process reinforced that **design correctness precedes performance**: subtle resource mistakes can survive for a long time and cause intermittent failures. Encapsulating view/camera state removed hidden coupling and made the code easier to reason about. Small, targeted improvements produced a disproportionately large quality gain, and I’m confident I can apply these patterns to larger systems.