# **Final Project Self-Assessment**

**CS 499 – Computer Science Capstone** **Student:** Shoh Janobilov

### **Project Overview**

My final project unified all major areas of computer science, **software design**, **data structures and algorithms**, and **databases**, into a single, production-style artifact. The base artifact, created in **CS 330 (Computational Graphics and Visualization)**, is a C++ OpenGL application that renders an interactive 3-D desk scene containing textured objects such as a keyboard, mouse, wrist rest, and glasses.  
 Across Milestones Two through Four, I progressively evolved this artifact from a working prototype into a professional-quality application that demonstrates scalability, maintainability, and persistence. Each enhancement targeted a specific outcome and showed measurable growth in my engineering skill set.

## **Category 1 – Software Design and Engineering (Milestone Two)**

**Enhancement Focus:** Code structure, memory safety, and resource lifecycle  
**Files Touched:** SceneManager.cpp/.h, ViewManager.cpp/.h, MainCode.cpp

I replaced unsafe manual memory management with **RAII patterns** using std::unique\_ptr, refactored global state into encapsulated class members, and corrected a critical OpenGL resource-lifecycle bug (replacing an incorrect glGenTextures call with glDeleteTextures).  
 These improvements eliminated hidden coupling, clarified ownership semantics, and ensured that all textures and buffers are released exactly once. The result is a codebase that follows professional C++ design conventions, clean interfaces, explicit lifetime control, and defensive error handling.

**Outcome Demonstrated:**

*Use well-founded and innovative software-engineering techniques to deliver solutions that meet industry goals.*

## **Category 2 – Algorithms and Data Structures (Milestone Three)**

**Enhancement Focus:** Performance and scalability of texture lookups  
**Files Touched:** SceneManager.cpp/.h

Originally, texture management relied on a fixed-size array and linear searches, yielding O(n) lookup time and an arbitrary 16-texture limit. I refactored this logic to use an **std::unordered\_map<std::string, GLuint>**, achieving **O(1)** average lookup complexity and dynamic scalability.  
 This change simplified code paths, reduced complexity, and eliminated the need for manual bookkeeping variables like m\_loadedTextures. It also modernized the artifact’s use of STL containers and demonstrated sound algorithmic decision-making.

**Outcome Demonstrated:**

*Design and evaluate computing solutions using algorithmic principles and data-structure trade-offs.*

## **Category 3 – Databases and Persistence (Milestone Four)**

**Enhancement Focus:** Data persistence, telemetry logging, and error auditing  
**Files Added:** DbHelper.h/.cpp  
**Files Integrated:** MainCode.cpp, ViewManager.cpp, SceneManager.cpp

To extend the project beyond transient graphics data, I embedded an **SQLite database layer** through a dedicated DbHelper class that uses prepared statements for safety.

* **Profiles table:** Stores camera position, zoom, and projection settings for reproducible sessions.
* **Telemetry table:** Logs frames-per-second and frame-time metrics each second for performance analysis.
* **Errors table:** Records texture-load or shader-uniform failures for post-run diagnostics.

This integration demonstrates full-stack proficiency, linking low-level C++ logic with a robust data-management layer, and reflects real-world engineering practices such as persistence, instrumentation, and graceful error handling.

**Outcome Demonstrated:**

*Employ innovative tools and techniques in software engineering and databases to deliver maintainable, data-driven solutions.*

## **Reflection and Growth**

This capstone forced me to think like a professional engineer, not just a student. I learned to:

* Balance **code safety, performance, and readability** when choosing patterns or data structures.
* **Refactor incrementally** while preserving existing behavior and avoiding regressions.
* Implement **persistent storage and telemetry** to support maintainability and long-term observability.
* Communicate design intent clearly through comments, commit messages, and structured documentation.

The most significant challenge was integrating SQLite cleanly into a graphics-bound application without disrupting render performance or cross-platform compatibility. Through research and experimentation, I achieved a lightweight solution that enhanced functionality without sacrificing speed.

## **Alignment to Program Outcomes**

| **Program Outcome** | **Evidence of Progress** |
| --- | --- |
| **Collaborative Environments** | Followed modular design patterns that make team integration straightforward. |
| **Professional Communication** | Produced consistent documentation, naming conventions, and explanatory narratives for each milestone. |
| **Algorithmic Design & Evaluation** | Replaced linear search with hash-based structures; reasoned about time/space trade-offs. |
| **Software Engineering / Database** | Implemented RAII, encapsulation, and database persistence with industry-standard libraries. |
| **Security Mindset** | Added error handling and validated inputs to mitigate undefined behavior and data corruption. |

### **Conclusion**

My final project now embodies the depth of a complete computer-science solution, clean architecture, efficient algorithms, and secure, persistent data management. The transformation from a static 3-D scene to an instrumented, data-aware application illustrates the evolution of my skills from **developer** to **software engineer**.  
 This ePortfolio artifact stands as evidence of my readiness to contribute to complex, performance-sensitive, and data-driven systems in professional environments.