Milestone 3 Narrative

Sean Born

CS 499

October 5, 2025

**Artifact Description**

The artifact that I selected for this category is a 3D graphics rendering engine that I originally developed in *CS-330* Computational Graphics and Visualization in 2023. The system I used was OpenGL to display 3D objects and manage scene rendering through a series of interconnected classes. For this enhancement, I focused on algorithmic efficiency and data structure optimization. Specifically, I implemented a quadtree spatial partitioning algorithm to improve rendering and collision detection performance and refactored the code to remove reliance on global variables by introducing an ApplicationManager class for centralized control. These modifications chnaged the program into a more scalable, efficient, and maintainable system aligned with professional software engineering principles.

**Justification**

I selected this artifact to include in my ePortfolio because it shows my ability to design, implement, and analyze algorithms that improve computational efficiency. The original code performed rendering and collision detection with O(n²) complexity, which limited scalability as the number of objects grew. By integrating a quadtree data structure, I reduced scene traversal and object query complexity to approximately O(log n). This greatly improved performance for larger datasets. Creating the ApplicationManager class replaced multiple global variables such as g\_Window, g\_SceneManager, g\_ViewManager, and g\_ShaderManager with a structured, object-oriented architecture. This change improved encapsulation and supported better modularity and testing practices. These enhancements collectively showcase my ability in algorithmic design, modular software architecture, and performance optimization which are all important skills that directly reflect the 3rd program outcome on designing and evaluating computing solutions using algorithmic principles.

**Outcome Alignment**

This enhancement demonstrates achievement across multiple computer science program outcomes by integrating algorithmic principles, software design, and secure coding practices into a cohesive and optimized solution. By implementing a quadtree spatial partitioning algorithm, I effectively applied computational and algorithmic reasoning to reduce the time complexity of rendering and collision detection from O(n²) to about O(log n) This directly aligns with the outcome of designing and evaluating computing solutions using algorithmic principles. Creating the ApplicationManager class and refactoring of global variables into a modular structure shows my ability to use well-founded and innovative software engineering techniques to design scalable and maintainable systems. I also incorporated secure coding measures such as input validation, file existence checks, and smart pointers for automated resource management, reflecting a security-focused mindset that anticipates and mitigates potential vulnerabilities. The expanded Doxygen-style documentation and clear class organization also highlights my capability to produce professional, technical communication.

**Reflection**

The enhancement process increased my understanding of how data structures and algorithmic complexity directly influence software performance. Implementing a quadtree required recursive reasoning, spatial partitioning logic, and careful balancing of memory efficiency with lookup speed. I gained practical experience translating theoretical computer science concepts into an operational, visual system that improved both accuracy and efficiency. Refactoring the code to remove global variables was another valuable learning experience. I encountered the challenge of redesigning initialization and dependency management across multiple subsystems of SceneManager, ViewManager, and ShaderManager. This taught me how architectural design choices such as modularity, encapsulation, and dependency inversion affect scalability and maintainability in large applications. Debugging OpenGL resource handling such as texture deletion and context state also reinforced the importance of robust error handling, secure resource management, and defensive programming. Lastly, writing unit test scaffolds for matrix math, resource loading, and spatial traversal brought more attention to the role of validation and testing in making sure the code has long-term reliability.