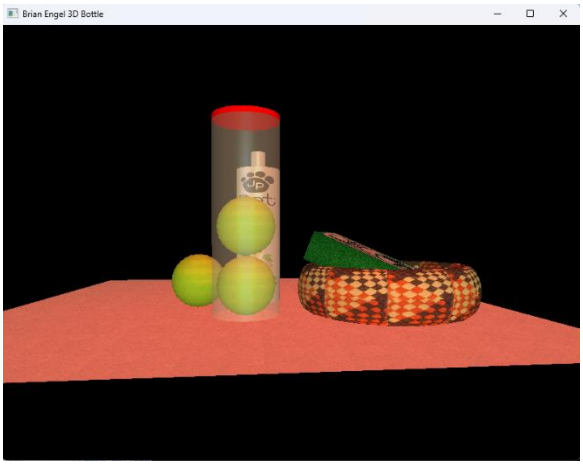
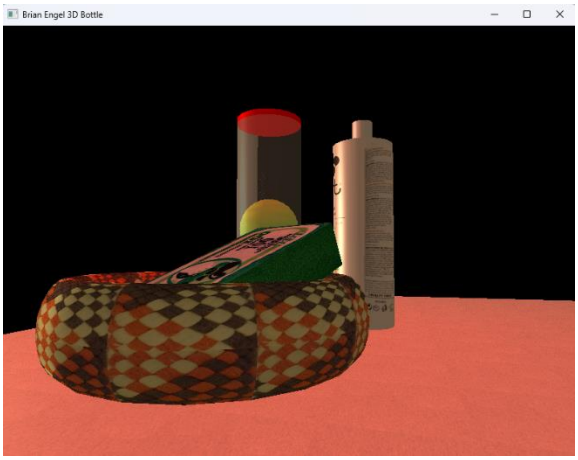
Software Design and Engineering Narrative

Artifact Description: The artifact used to demonstrate software design and engineering is a project from CS 330: Computational Graphics and Visualization, completed in the fall of 2023. This project utilizes OpenGL and C++ to create a navigable 3D scene with textures, lighting, and shading.





Justify the inclusion of the artifact: The project was selected for inclusion in my ePortfolio because, while the end result achieved my goals, the initial structure of the artifact was disorganized. The original implementation involved creating numerous vertex and index generators for various shapes, which were all housed within the main .cpp file. This cluttered approach made the code difficult to read, manage, and extend.

Improvements and Enhancements: To improve the artifact, I refactored the code by moving the vertex and index generators to separate header files for each shape, significantly cleaning up the code structure. During this process, I realized I could encapsulate each shape into its own class. This encapsulation allowed me to define private variables for each instance, which could be accessed through getter functions, replacing the numerous public global variables that were

originally necessary in the main file. This change not only improved the code's readability and maintainability but also enhanced the program's security.

Furthermore, I created constructors for each class that accepted arguments to generate different-sized shapes. This enhancement provided two ways of creating varied shapes in the scene: by generating different instances of varying sizes or by scaling the shapes using OpenGL. Although the current implementation retains the OpenGL scaling method, future projects or extensions of this project would benefit from using new instances, as this approach offers greater flexibility and ease of use.

Course Outcomes: By enhancing and modifying this artifact, I successfully met and exceeded the course outcomes I had planned for. Originally, I had planned to demonstrate building collaborative coding environments through code refactoring and modularization, using computer science practices and standards by creating the header files, and adding value by improving code maintainability and scalability, which all was accomplished. After these I also enhanced the quality of communication within the project by creating structured and well-documented code, along with clear class definitions and modularity and moving to a class-based structure and reducing public global variables directly addresses the development of a security mindset. It anticipates potential vulnerabilities and ensures better security of data and resources.

Reflection on Process: One of the key lessons learned was the importance of modular design in managing complex systems. By encapsulating shapes into individual classes, I was able to isolate functionality, which made the code easier to understand and extend. This approach not only improved the readability of the code but also facilitated debugging and future enhancements. In hindsight, I would spend more time in the planning phase to avoid structural issues from the start. A major challenge was deciding between retaining the existing scaling method or adopting

the new instance-based approach. The flexibility to use either method now simplifies the creation of future shapes. Additionally, refactoring removed numerous magic numbers and replaced global variables with instances created with well-named constants, contributing to a cleaner and more maintainable codebase.