Artifact Title	3DTriangleOpenGL
Author	April Nixon
Creation Date	September 2024

Introduction

This document explains the numerous improvements made to a 3D Pyramid rendering program using OpenGL, which I developed at the end of my CS-330 graphics and visualization class. The artifact originally illustrated basic concepts of computer graphics, such as vertex and fragment shaders, buffer management, and 3D transformations. In this iteration, I focused primarily on improving the usage of algorithms and data structures to create cleaner, more efficient, and maintainable code.

Description of the Artifact

The artifact is a C++ program that uses OpenGL to render a 3D pyramid. Vertex and fragment shaders are used for coloring the sides of the pyramid, and transformation matrices are applied for real-time rotation of the pyramid. The artifact covers important basics of computer graphics, including VBO (Vertex Buffer Object), EBO (Element Buffer Object), and shader-based rendering.

We received a single-file application as an artifact where everything, including shader compilation, buffer handling, and rendering code, was in one file with very little modularity or error handling. Because of this approach, it was difficult to demonstrate how one would write modular interfaces or efficient code, which is important when writing production ready applications at scale.

Justification for Inclusion in the ePortfolio

I have choosen this artifact for my ePortfolio because it represents my ability to use complex algorithms and data structures in the context of a real application. The changes made to this artifact shows my ability to apply object-oriented programming (OOP), efficient memory management, and algorithmic efficiency — all skills that any competent computer scientist should possess. The Buffer class shows how I was able to use an abstract data type in graphics programming. In addition to helping keep the code organized and easy to read, it also makes management of our OpenGL resources more error-resistant. I have also used Object3D class and std::vector (Dynamic array) so that multiple 3D-objects can be read, which demonstrates my skills for handling dynamic data structure and memory management in a real-time application. I have also shown some errors during shader compilation and program linking. That means I'm aware of how reliable our code/algorithms should be for the proper working of the program. That's why writing reliable and maintainable code is my first priority. I used the GLM library to apply matrix transformations (e.g., model, view and projection matrices) to show I had an understanding of linear algebra algorithms that uses computer graphics. Ultimately, this made the rendering look better as well as improved the performance in the calculation of transformations.

These updates match what the course was aiming for, especially in terms of focusing on algorithms and data structures. The improvements I made also help make the code more professional and easier to maintain. Adding this project to my ePortfolio shows that I can take what I've learned in theory and apply it to real-world problems.

Enhancements Made

The following enhancements were made to the original artifact to better align it with best practices in algorithms and data structures. The buffer management code was summarized in a newly created Buffer class, which is responsible for creating, binding and deleting OpenGL buffers. This change increased the modularity of the code and made it easier to read, maintain and extend. Also, by centralizing the buffer related operations in a single place, the overall program structure became cleaner, and we could manage resources more effectively. I added an Object3D class which covers all the required data (such as vertex array object handles and buffer handles) and functions (such as bind() to bind an object for rendering) needed to deal with a 3D object. A std::vector is used to keep track of multiple objects with complex scenic scenarios. With this feature, we can easily add many objects without any significant changes in our scene rendering application. The shader compilation process has been restructured to perform a thorough error check. It makes sure that any failure during shader linking or shader compilation is detected and reported as soon as possible in the code. Writing maintainable and debugable code requires proper error handling, which becomes more important in graphics programming, since an incorrect result of a shader compilation may manifest as a very difficult-to-trace bug. With this enhancement, we used the GLM library to perform matrix transformations, such as rotation or perspective projection. The GLM library is well-optimized and provides a consistent interface for dealing with linear algebra operations. This allows not only the simplification of transformation logic but also better optimization opportunities for compilers when performing matrix operations inside a rendering pipeline.

These improvements were used for increasing the performance, readability and maintainability of the code. After these small but important changes, I could improve its availability a lot to professional standards and best practices for software development.

Reflection on the Enhancement Process

Working on this project taught me a lot about how data structures and algorithms fit into computer graphics. When I created the buffer management class, it helped me keep the code tidy and showed me how important it is to manage resources, especially in real-time situations. Making the code more modular also made it easier to update and expand the program. Using std::vector for handling multiple objects was important also. It showed me how strong dynamic data structures can be. We needed to manage all those objects without hitting memory limits, especially when adding just one more could push things over the edge. One of the most important things I learned was about catching errors in shader compilation. The more errors I caught, the easier it became to fix and improve the program later. This experience revealed the need to write solid, reliable code. Also, using GLM for matrix transformations helped me get a better handle on the math behind computer graphics. Designing algorithms is one thing, but making them work in real code is another. This process really showed me how much the way you apply algorithms affects the program's performance and quality.

Conclusion

This artifact, along with the enhancements I made, shows my ability to apply algorithms and data structures in a practical, real-world context. The improvements made to the original code not only align with best practices in software engineering but also showcase my growth as a computer science professional. This project is a testament to my ability to tackle complex problems, optimize code for performance, and write maintainable, scalable software. Including this artifact in my ePortfolio allows me to effectively show my technical skills and my ability to apply theoretical knowledge in practical applications.