Programming Project Report

Student Name
Student UAID

Problem Statement:

The goal of this assignment was to implement basic OpenGL operations and understand how the main loop and drawing of objects occurs. Normal inputs were hardcoded in range of what our random generator should make and allowed for easy testing. The program will create a window and in that window my number of polygons with some number of vertices chosen by me. Error handling was done with simple cout statements to see when operations failed and why.

Design:

I decided to separate the classes into a header and .cpp file to make my code easier to understand, define, and begin implementing. I also designed a testing function using random device that utilizes hardware to select a random number. I used vectors to store polygons and vertices, as well as floats to store color and position data, and lastly integers to handle looping and iterating through data. There were no specific algorithms I used, but for each loop were necessary to traverse my vectors of data. Pros include how easy it is to read my data, and cons include having to use global variables to test features.

Implementation:

I started with no code and created the structure of my code with a main program, two header files, and two class files to define everything we needed. This code was extended by implementing the Vertex & Polygon classes, as well as implementing test features within the main program. I worked on this project over the course of two days and spend one day working on all of the C++ files, then one day converting it to OpenGL and adding rotation mechanics.
AI Prompts & Answers:
PROMPT 1
In this chat, I will be working on a computer graphics assignment where a requirement is to keep track of all AI prompts in a separate document to submit with my project report. For task one I must implement a polygon class in C++ that stores information about 3D polygons (x, y, z, r, g, b). Don't give me the solution, just help guide me in the steps.
AI RESPONSE (Summary):
Outlined a step-by-step design plan for implementing Vertex and Polygon classes, emphasizing data modeling, class responsibilities, readable output methods, and avoiding direct solutions while explaining the reasoning behind each step.

PROMPT 2
This project will be transformed to work with OpenGL. What data types for coordinates and colors are used by convention?

AI RESPONSE (Summary):
Explained that OpenGL conventionally uses `float` values for vertex positions (x, y, z) and normalized `float` values in the range [0,1] for colors (r, g, b), as these map directly to GPU pipelines and shaders.

PROMPT 3
I am making my vertices class. Is it important to keep all variables private and use getters/setters? Where should the presentInformation function live?
AI RESPONSE (Summary):
Explained that vertices are typically treated as simple data containers in graphics programs, so strict encapsulation is not required. Recommended keeping output methods public and const, and avoiding unnecessary getters/setters unless required by the assignment.

PROMPT 4
Should the set methods for (x,y,z) and (r,g,b) be implemented in vertices.cpp or polygon.cpp?
AI RESPONSE (Summary):
Clarified that setting position and color belongs in the Vertex class because these values are intrinsic to each vertex, while Polygon should manage collections of vertices rather than individual vertex attributes.

PROMPT 5
Should color belong to the whole polygon instead of each vertex? Also, can you show an example of C++ inheritance?
AI RESPONSE (Summary):
Explained that although a polygon may conceptually have one color, OpenGL treats color as a per-vertex attribute. Provided a conceptual explanation of inheritance using a base Shape class and a derived Polygon class.

PROMPT 6
Will these classes need constructors and destructors?
AI RESPONSE (Summary):
Explained that constructors are necessary to properly initialize Vertex and Polygon objects, but destructors are unnecessary unless the class manages dynamic memory or OpenGL resources.

PROMPT 7
What data type should the Polygon class use to store vertices?
AI RESPONSE (Summary):
Recommended using `std::vector<Vertex>` because it supports variable-sized polygons, provides contiguous memory (ideal for OpenGL), and avoids manual memory management.

PROMPT 8
How do I properly loop through each vertex in a vector?

AI RESPONSE (Summary):
Explained correct looping techniques using range-based for loops and index-based loops, emphasizing passing vertices by reference to avoid copying and ensure correctness.

PROMPT 9
Should random values be completely random or hard-coded so I know the output?
AI RESPONSE (Summary):
Explained that the assignment expects true randomness and recommended using real random number generation, optionally with a fixed seed during debugging and a random seed for final submission.

PROMPT 10
Is my random number generation efficient, and how do I get three values each time?
AI RESPONSE (Summary):
Explained that one random generator should be reused with multiple uniform real distributions, and that each call to the distribution produces one value—so three calls are needed for x, y, z and r, g, b.

PROMPT 11
Can you show a working example that assigns random values to x, y, z, r, g, b?
AI RESPONSE (Summary):
Provided a minimal example demonstrating correct usage of `std::mt19937` with `std::uniform_real_distribution<float>` to generate valid OpenGL-friendly values.

PROMPT 12
Should each polygon have a random number of vertices?
AI RESPONSE (Summary):
Advised against randomizing vertex count unless explicitly required, recommending a fixed vertex count (e.g., triangles) to simplify grading and OpenGL compatibility.

PROMPT 13
Would my loop generate identical vertices or randomized ones?
AI RESPONSE (Summary):
Explained that calling the distribution inside the loop generates new random values each iteration, but emphasized correct loop structure and storing vertices in a container.

PROMPT 14
Why am I getting compiler warnings about the comma operator?
AI RESPONSE (Summary):
Explained that `(x, y, z)` in C++ uses the comma operator rather than forming a tuple, causing values to be discarded. Clarified that grouped data must be stored using multiple variables or a struct.

PROMPT 15
How do I represent grouped data and set its values in C++?

AI RESPONSE (Summary):
Outlined valid approaches including storing separate floats, using small structs (Vec3/Color3), or `std::array<float,3>`, explaining tradeoffs and OpenGL friendliness.

PROMPT 16
After testing, each blue color value is always 0. Why?
AI RESPONSE (Summary):
Identified common causes such as integer distributions, assignment mistakes, printing the wrong variable, or modifying copies instead of references.

PROMPT 17
What must I know to convert my program to OpenGL?
AI RESPONSE (Summary):
Explained the OpenGL render loop model, the importance of display callbacks, GPU context requirements, and how Polygon drawing fits inside a rendering loop.

PROMPT 18
I am using GLUT—how do I call display from main?
AI RESPONSE (Summary):
Explained GLUT's callback-based architecture and that drawing must occur inside the registered display callback, not directly from main.

PROMPT 19
What order must functions be called to avoid conflicts?
AI RESPONSE (Summary):
Outlined correct GLUT initialization order: initialize GLUT, create window, initialize OpenGL state, register callbacks, then enter `glutMainLoop`.

PROMPT 20
Why does only one polygon draw?
AI RESPONSE (Summary):
Explained that clearing the color buffer inside each polygon draw erased previous polygons, and that clearing must occur once per frame.

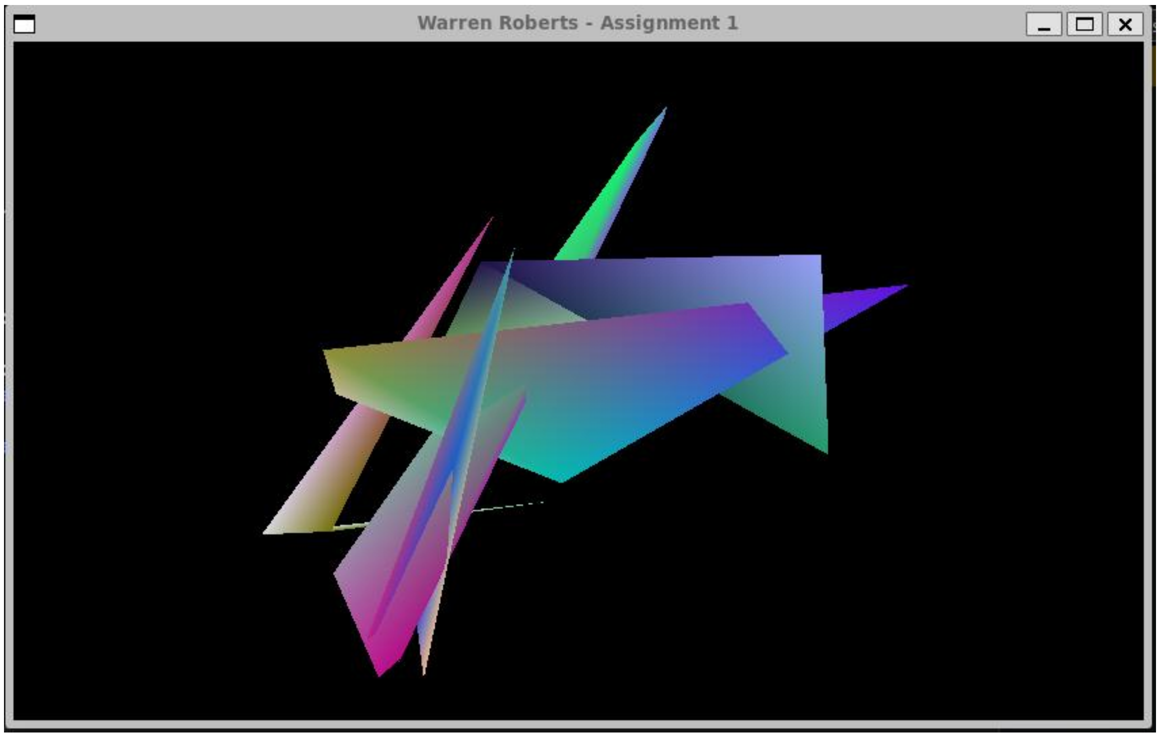
PROMPT 21
How do I rotate polygons when resizing the window?
AI RESPONSE (Summary):
Explained use of the GLUT reshape callback to update projection, modify rotation angle, and request redraws.

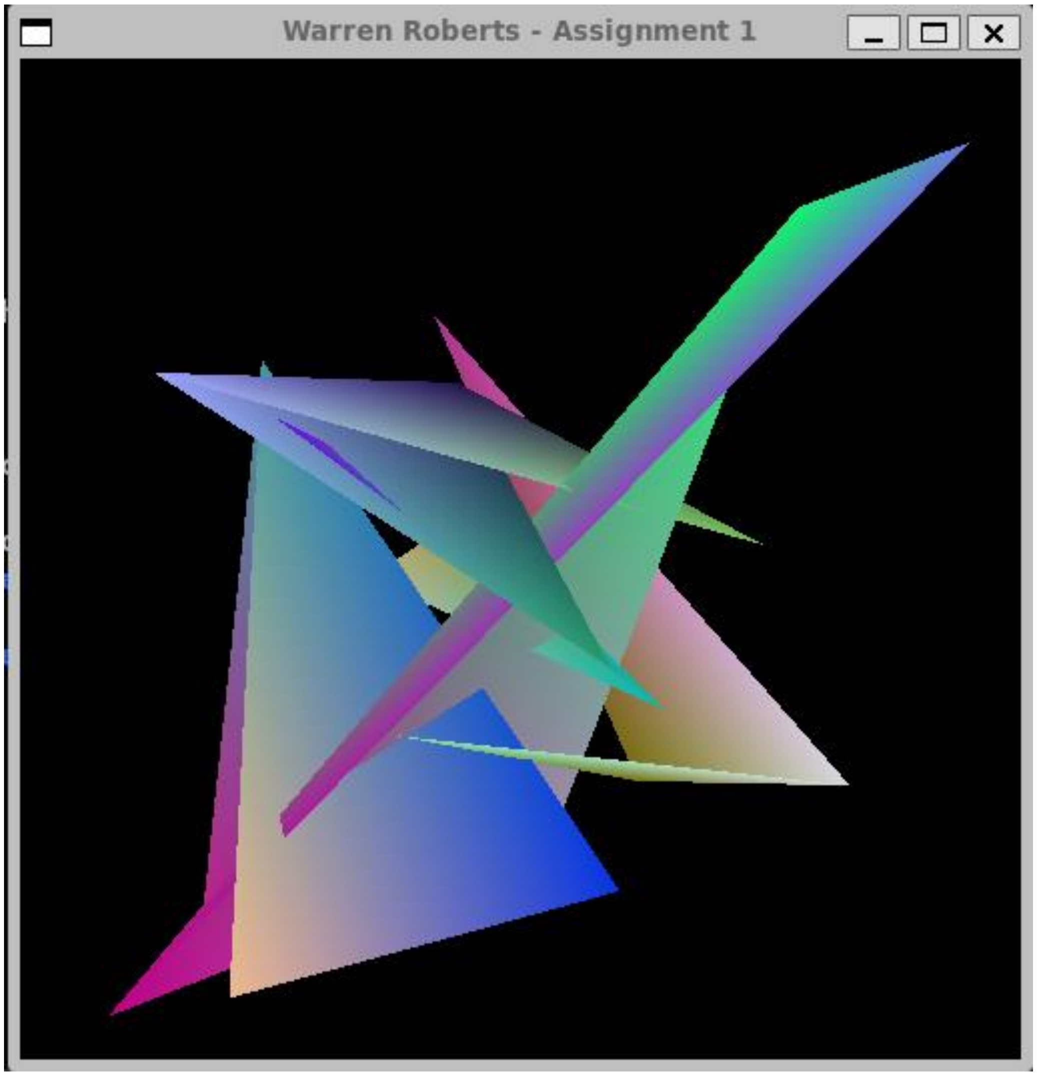
PROMPT 22
What OpenGL functions define an axis and rotation?
AI RESPONSE (Summary):
Identified required functions: `glMatrixMode`, `glLoadIdentity`, `glRotatef`, and explained how axis vectors define the rotation axis.

PROMPT 23
Is my rotation and reshape logic correct?
AI RESPONSE (Summary):
Confirmed correctness, suggested improvements (single axis rotation, depth buffer consistency), and alignment with assignment wording.

Testing:

I tested my program by printing information regarding each vertex of each polygon, before being able to install OpenGL and visually see my polygon. Normal inputs included hardcoding vertices and polygons to test in OpenGL, before making a randomizer to make a lot of polygons with random dimensions and color. Special input cases were ones that exited the bounds provided to be tested, and everything worked as expected


Warren Roberts - Assignment 1



Conclusions:

The project went very well, and it was extremely interesting to visualize the results of this lesson. This project was a success, and I was very happy with the outcome. If I were to do this again, I would probably make no class of vertices and just do polygons with structures to represent each vertex. It took about two days to complete this project, and I started the report the next day.