OpenGL Viewer Warmup Report

Joshua Lum Shue Chan

Compiling Guide

See README.txt.
Ensure that the cube.obj file is with the others.
Ensure that the glm include library (no lib / .a files) is in the same directory as GL and GLFW include directories.
Compile with normal commands. Presumably,
g++ -lglfw -lglew Source.cpp Shader.cpp ObjectLoader.cpp
will work, but I was unable to test it.
To call an object file other than “cube.obj”, please edit the value in line 83 of Source.cpp.

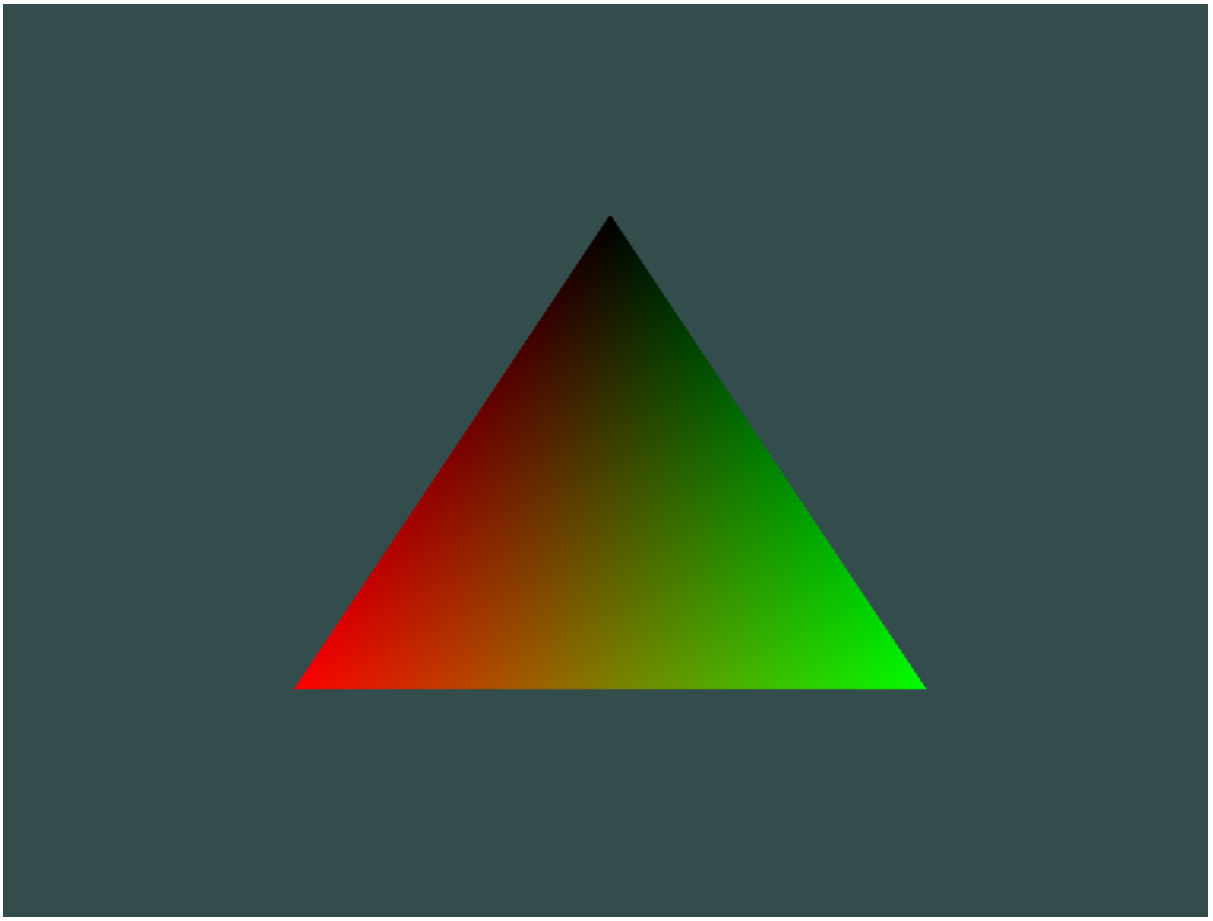
Operational Guide

Press ‘0’ to see the wireframe.
Press ‘1’ to zoom out.
Press ‘2’ to zoom in.
Press ‘3’ to rotate the model.
Press ‘Esc’ to exit the program.

Tasks

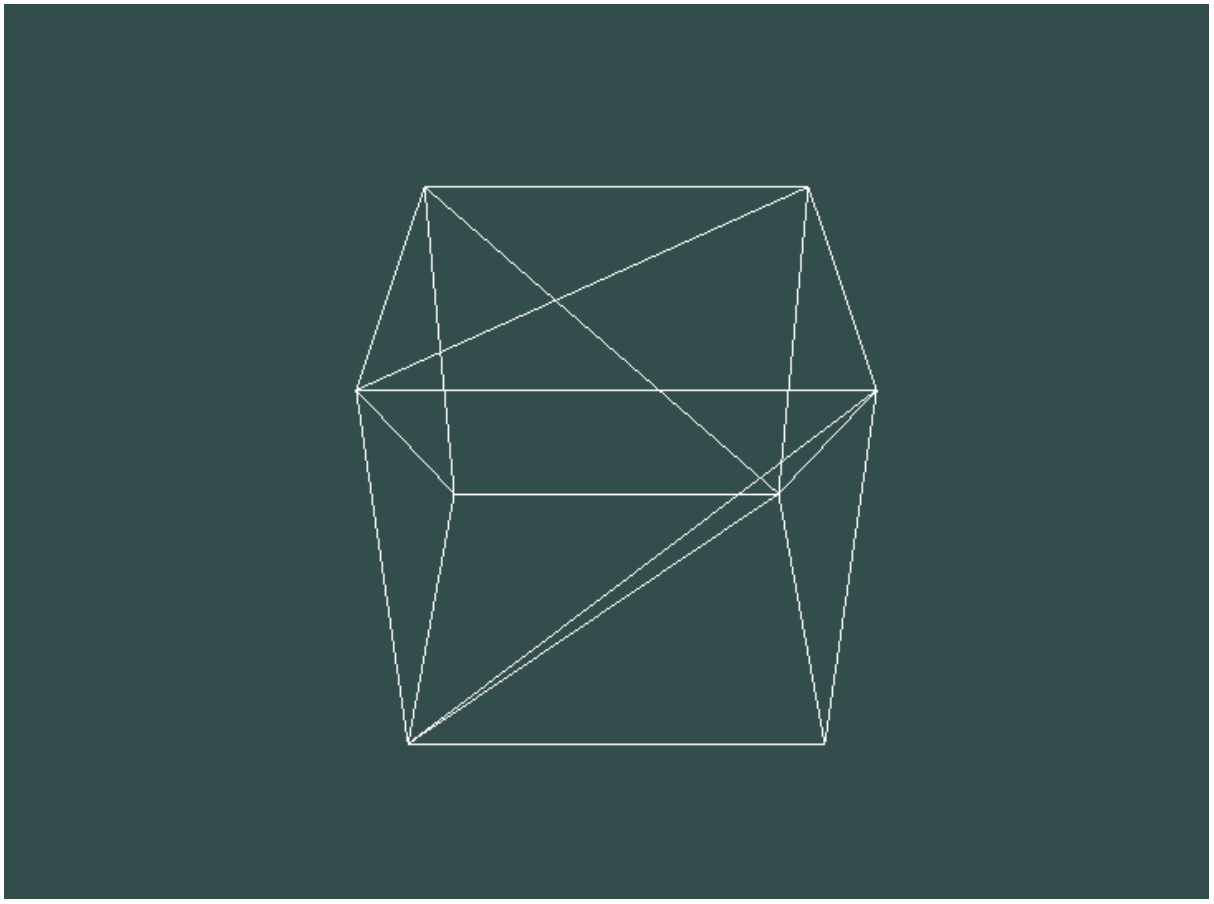
Optional tasks:
I implemented the Shader class to have the functionality of reading in and compiling the vertex and fragment shader code.
While following the guide at learnopengl.com, I implemented the indexed triangle data structure by simply fixing the vertices array and adding the indices array that lists all the indices of each triangle. All examples shown are using the indexed triangle data structures.

1. Re-coloring the triangle.
I edited the vertices array to include 3 extra fields (RGB) for each vertex. Then, I added another vertex attribute pointer to gather the extra data in the array. Finally, I changed the fragment and shader codes to use the new color data.



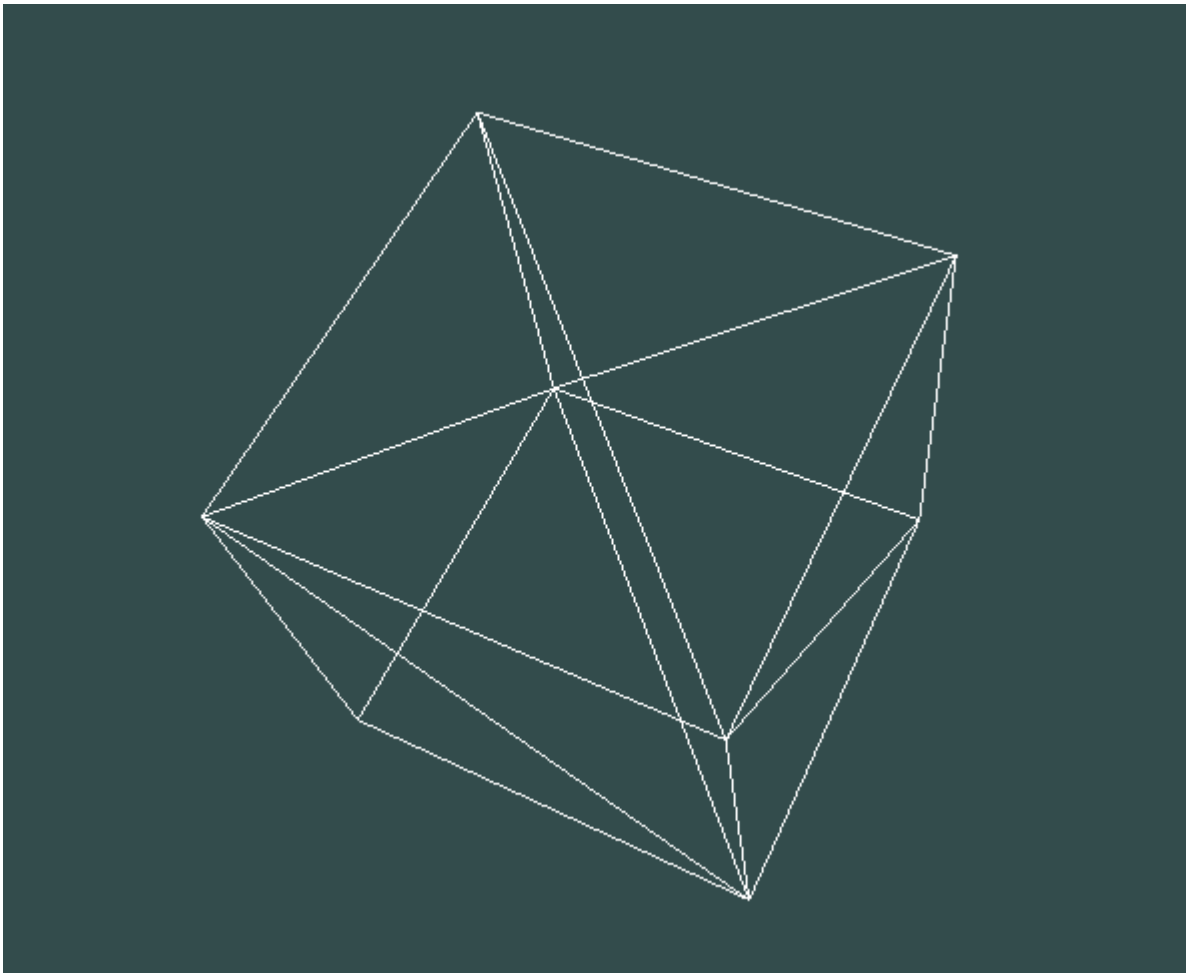
2. Read in from .obj and build a 3d object.
Note: I did this with indexed triangles since I already had it implemented by this point, and the .obj files are directly in indexed format.
I built a class that reads in the data and puts it into vertices and indices arrays. I then got the arrays from the object in the main function and passed them to the VAO and EBO. I also changed the perspective of the camera using transformation functions in the glm math library, and by multiplying those transformation matrices by the input to gl_position in the vertex shader. I did not implement this for non-triangular meshes, or meshes that don't have index lists.

Top view



Same view with frames

Diagonal view



Diagonal view with frames

3. Simple changes to shaders

The below show is at the same “position” as the above top view shot, but since I flipped the coordinates, translated it in the x direction by 2, and changed the colors, it appears different.

Program Logic
The program first uses the Shader class to compile shader files, and the ObjectLoader class to read in the vertex and index information from the .obj file. Then, the VAO, VBO, and EBO are initialized, bound, and given data. Then, the shader program receives the data. After that, the initial viewing transformations are calculated. Finally, the render loop first checks for input, then sends the modified viewing transformations to the shader program, and finally draws the image.

Requirement Checklist
1. “Change the Vertex Buffer Objects.. “ Finished.
2. “Add the functionality to read…” Finished, except my implementation never used the “separate triangles” data structure. I used indexed triangles for the whole project.
3. “Currently the vertex shader…” Implemented in Shader class..
4. “Make simple changes to vertex shader…” I experimented with adding and flipping coordinates in the vertex shader and changing colors in the fragment shader.
5. “Bonus: read up…” I implemented this at the start of the project.

Acknowledgements
https://learnopengl.com/ I followed their tutorial all the way through shaders, and used other bits of their tutorial for changing perspective, etc.
https://www.glfw.org/ for glfw libraries
http://glew.sourceforge.net/ for glew libraries

https://github.com/g-truc/glm for glm libraries
https://www.javatpoint.com/how-to-split-strings-in-cpp for string splitting method
Lecture slides, Professor Entezari for teaching me.
The given code in HelloTriangle.cpp
https://www.youtube.com/c/TheChernoProject for help with setting up glew and glfw