Sheet 1

Rendering Pipeline

Question 1.
You have learned about the structure of a rendering pipeline for creating interactive 2D and 3D graphics in the lecture. In this context, work on the following tasks.

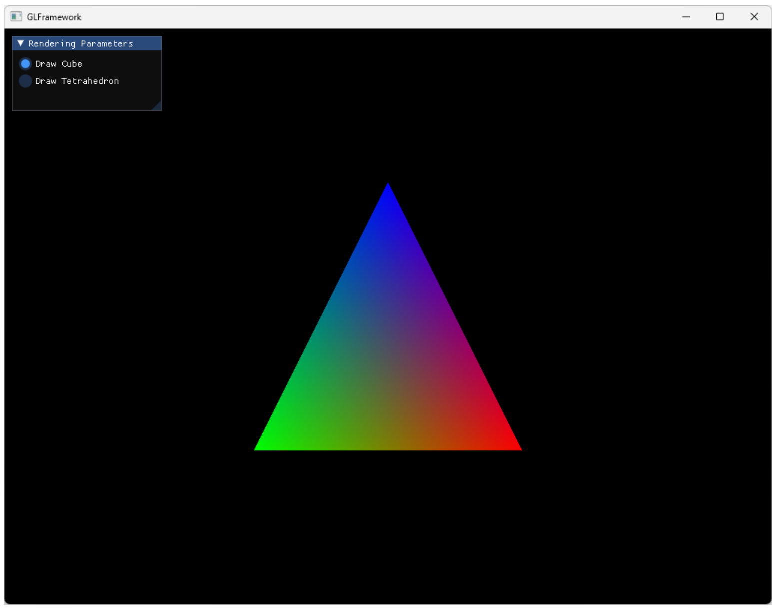
(a) (3 points) Describe in detail the three conceptual stages required in a generic rendering pipeline to produce an image that can then be written to the framebuffer.

(b) (2 points) Which stage processes the vertices into a rasterizable representation of the scene to be displayed? What substeps does this stage include and what are they used for?

(c) (3 points) OpenGL also follows the concept of the rendering pipeline, where the principle of a state machine is implemented, in which the behavior of each stage is determined by the current state. The state of the OpenGL machine can be changed using corresponding functions. Name two OpenGL functions for each of the conceptual stages described in part (a) that you can use to influence the behavior of these stages. Furthermore, briefly describe how these functions change the state.

Question 2.
With this exercise sheet, a framework is available for download in which the programming tasks are to be solved.

(a) (0 points) Download the framework and unpack it. Use 'cmake' to create a project for the platform you are using (operating system, compiler, IDE). Load this project, compile it, and start the program (can be terminated with 'ESC'). The running program should display a triangle as shown in the next figure, which can be rotated with the mouse.



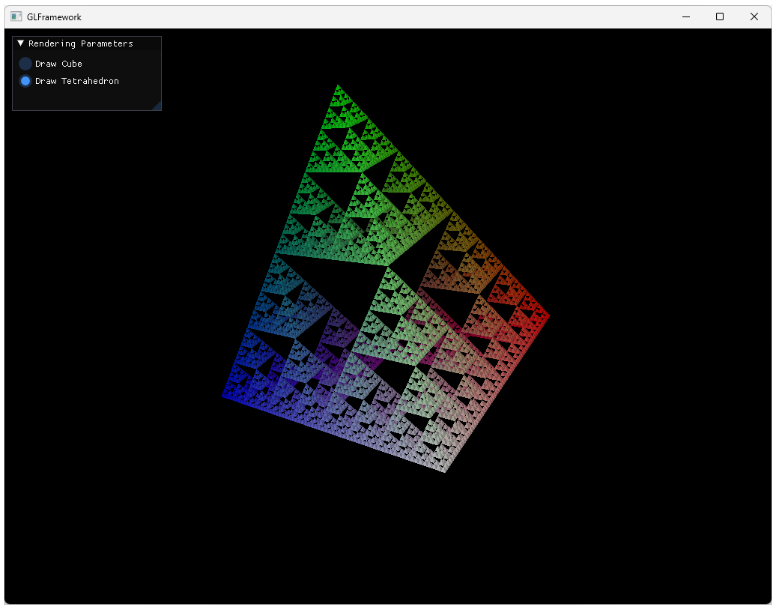
(b) (2 points) In the displayed scene, the back side of the triangle is not shown. Which OpenGL command can be used to achieve this? What are the benefits? How can you also specify that the front sides of triangles have a counterclockwise winding?

(c) (3 points) Edit the file **"src/main.cpp"**. Change the method **createCubeVertices** to display a cube in the window. Pay attention to the order of the vertices so that the front and back sides are correctly displayed. Correct the **glDrawArrays** call in the main loop of the method **main** to render the correct number of vertices. Set the colors of the cube so that three side faces are rendered red, green, and blue. The opposite sides should have the complementary colors (cyan, magenta, yellow).

(d) (2 points) Adjust the normal vectors of the cube so that they correspond to the surface normal. Use a different shader that simulates more realistic lighting so that this change is visible. At the beginning of the **main** function, a fragment shader ("flat.frag") is loaded. Change this to "light.frag" to render the cube with a light source.

(e) (5 points) Render a Sierpinski tetrahedron. This is a fractal structure created by subdividing tetrahedrons. Edit in the file **"src/main.cpp"** the methods **createFractalTetrahedronVertices** and **splitFractalTetrahedron**. Create an initial tetrahedron. Pay attention to the order of the vertices so that they are correct when creating the vertex buffer. Recursively replace each tetrahedron with four smaller tetrahedrons to create a Sierpinski tetrahedron. Set the correct normals for the triangles of the tetrahedrons when inserting them into the vertex list. Calculate the correct number of vertices and adjust the **glDrawArrays** call accordingly. The GUI in the top left corner can switch between the cube and the Sierpinski tetrahedron.

The result should look approximately like the next figure.



Submission All areas where you need to change source code are marked with a comment of the form `//TODO:`. Submit the written tasks, the framework you modified, and the result images for subtasks 2 c), 2 d), and 2 e) by Tuesday, May 20th, 2025, 11:55 PM via the Moodle platform in the form of a ZIP file.