Display of 3D Model Report

Théodore Billotte - 2025400572

April 12, 2026

1 Overview

This project is an interactive 3D model viewer written in C++ with OpenGL 3.3, GLFW, GLAD, and GLM. It loads the mesh file `eight.uniform.obj`, displays it in a window, and supports four display modes: vertex, wireframe, face, and face-with-edge. Keyboard input is used to switch modes, rotate and translate the model, and cycle the wireframe/edge color.

2 Assignment Requirements

Requirement	**Implementation**
Load and display a 3D mesh model	`objreader.h` reads vertices and triangular faces from `eight.uniform.obj`. The data is uploaded to OpenGL buffers and drawn in the GLFW window.
Support four display modes	`displayMode` selects vertex, wireframe, face, or face-with-edge mode. Keys `1-4` switch between them.
Different color for each face	`bindFaces` generates one HSV-based color per triangle and stores it with the triangle vertex data. The face shader uses `flat` color output so each face stays a single color.
Rotate and translate by keyboard	Arrow keys change `rotAngleX` and `rotAngleY`. Keys `W/A/S/D/Q/E` update the translation vector. These values are rebuilt into the model matrix every frame.
Change wireframe color	Pressing `C` cycles through the `wireColors` array. The selected color is used by wireframe mode and by the edge overlay in face-with-edge mode. Vertex mode uses the same color system for consistency.

3 Implementation

3.1 Mesh Loading

The OBJ reader is intentionally simple because the provided mesh only needs vertex positions and triangular faces. Lines beginning with `v` are stored as `Vertex` values, and lines beginning with `f` are stored as `TriFace` values. OBJ face indices are one-based, so the renderer subtracts one before looking up vertices in the C++ vector.

3.2 Buffers and Shaders

The program creates two VAO/VBO pairs. The first pair, built by `bindVertices`, stores only raw vertex positions and is used for vertex mode. The second pair, built by `bindFaces`, expands every triangle into three vertices and stores interleaved position and color data:

```
position: x y z
color:    r g b
```

Two shader programs are used. `pointShader` is used for vertices, wireframes, and edges. It reads vertex positions and receives a uniform color named `ourColor`. `faceShader` is used for filled faces. It reads both position and color attributes and passes the color with a `flat` qualifier, which prevents interpolation across a triangle.

3.3 Rendering Modes

In each frame, the program clears the color and depth buffers, updates the model matrix, and draws according to `displayMode`.

- **Vertex mode**: draws `VAO[0]` with `GL_POINTS`.
- **Wireframe mode**: draws `VAO[1]` with triangle data while using line polygon mode.
- **Face mode**: draws filled triangles with `faceShader`, so every triangle uses its generated face color.
- **Face-with-edge mode**: first draws filled faces, then draws wireframe edges on top. Polygon offset is used during the filled pass to reduce z-fighting.

3.4 Transformations

The initial model matrix rotates the mesh into a better viewing angle. During the render loop, the current model matrix is rebuilt from the keyboard-controlled translation and rotations. The view matrix moves the camera back along the `z` direction, and the projection matrix uses a perspective projection. All draw functions send the model, view, and projection matrices to the active shader before drawing.

3.5 Color Control

The selectable colors are stored in `wireColors`, and the active choice is tracked by `wireColorIndex`. Pressing `C` advances the index. The current color is passed into the vertex and wireframe draw functions, which upload it to the `ourColor` uniform.

4 Keyboard Controls

Key	Action
`1`	Vertex mode
`2`	Wireframe mode
`3`	Face mode
`4`	Face-with-edge mode
`C`	Cycle vertex/wireframe/edge color
Arrow keys	Rotate the model
`W`/`S`	Move up/down
`A`/`D`	Move left/right
`Q`/`E`	Move forward/backward
`Esc`	Close the window

5 Build and Run

The project uses CMake and requires a C++14 compiler, GLFW, GLM, OpenGL, and the bundled GLAD files in `third_party/glad`. On Ubuntu/Debian systems, the main external dependencies can be installed with:

```
sudo apt install cmake g++ libglfw3-dev libglm-dev libgl1-mesa-dev
```

Configure and build from the project root:

```
cmake -S . -B build
cmake --build build
```

Run the program from the project root:

```
./main
```

Running from the project root is important because the program loads the OBJ file and shader files using relative paths.

6 Submission

The submission archive contains the source code, shader files, OBJ model, GLAD dependency files, this report, and the screen recording. The screen recording is stored at `video/demo.mp4`.