Graphics Assignment 1 Report

- by Alan Babu and Varre Suman Chaitanya

Implementation

Window (window.hpp, window.cpp)

The GLFW window creation, resizing, termination and also input polling, output buffers are handled by this class.

Event (event.hpp)

A struct for holding event info that is passed to event handlers.

Shader (shader.hpp, shader.cpp)

Creates shader objects from glsl shader files. It has functions to set uniforms.

Scene (scene.hpp, scene.cpp)

This class holds all the entities in the scene - **grid**, **cursor**, **model** and an orthographic **camera**. The functions for handling inputs for scene entities and *window size callback* function(for updating the camera aspect) are bound as member variables of **window** class. The **window** object calls these functions on getting the corresponding input. The scene's input handler in turn calls the input handlers of its different entities.

Scene Entities

Camera(camera.hpp, camera.cpp)

Instead of rotating each model model independently we keep the rotation data in this object. It provides a view projection matrix, calculated using this info ,to be set as uniforms in shaders.

Grid (grid.hpp, grid.cpp)

Creates and initializes a vertex buffer containing positions of endpoints of lines that makes up the grid of given resolution(no of subdivisions).

Rendering:- It uses the **simple_vs.glsl** and **simple_fs.glsl** shaders and a view projection matrix provided by the **camera** to render.

Cursor (cursor.hpp, cursor.cpp)

Contains the position of cursor, current drawing color, buffer(vertex array, vertex buffer, index buffer) objects for its cube mesh and a reference to the **model** in the scene. It updates its position and the current drawing color through its input handler. It also calls the *addCube* and *deleteCube* functions of the model with position and color as arguments.

Rendering:-Uses **cursor_vs.glsl** and **cursor_fs.glsl** shaders to render a cube mesh.

Model (model.hpp, model.cpp)

It contains the vertex array, vertex buffer and index buffer of a cube. It contains arrays of size $(voxel\ resolution)^3$ with info about each grid cell(position, color, filled or not). A copy of this data is stored as buffers in the graphics memory. This buffer data is also managed by the **model** class. addCube, deleteCube, reselModel, loadModel updates these buffers accordingly. saveModel writes the data to file.

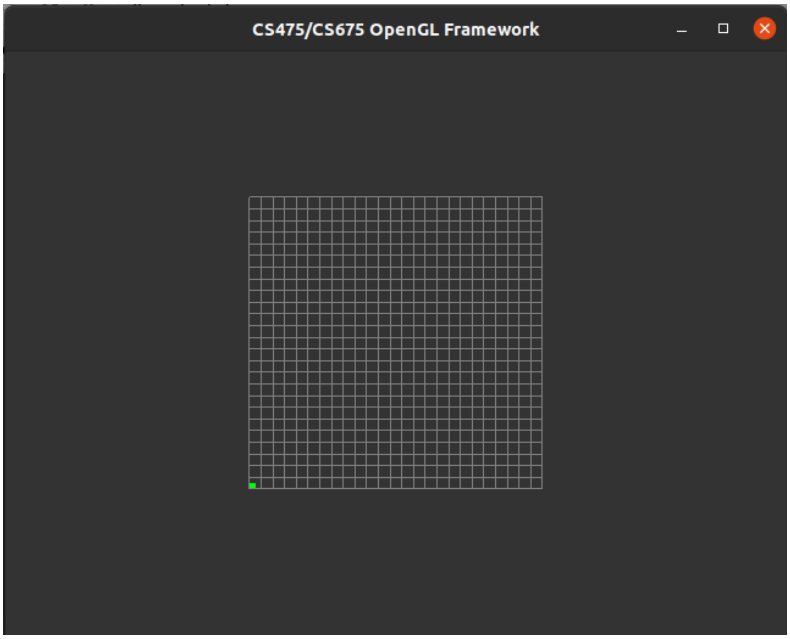
Rendering:-Uses **modelcube_vs.glsl** and **modelcube_fs.glsl.** The model is rendered by instancing the cube mesh. Each instance is rendered with its position and color. If the grid cell is not currently filled the mesh vertices are translated to a single point.

Control flow

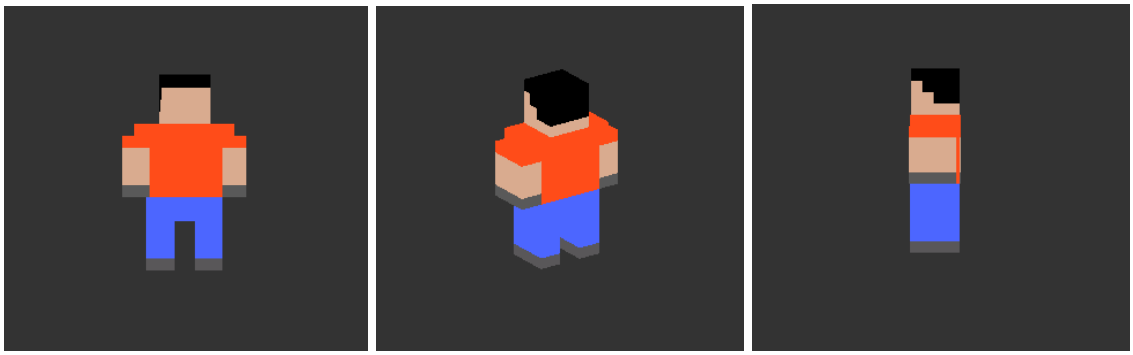
The main function in **main.cpp** creates a window and scene object and binds the corresponding callback function and input handlers. The **scene** object's constructor initializes the entities in the scene and also the shaders required. Here, we can change the resolution of the grid and size of the cursor. It then makes the calls for setting up the buffers of these objects for rendering.

The main loop runs while *window.shouldClose() == 0* . In the loop it calls the render function of the scene, swaps buffers and polls events. The render function calls the draw function of each of its entities after setting shader uniforms and binding the required shader.

Grid and Cursor



Model-1(Man)



Model-2(Flag)