Assignment #3

CS 200, Fall 2024

Friday, September 27

I will provide you with a header file named `SolidRender.h`, which gives the interface to a simple 2D rendering class. You are to implement this interface using the code from the `cs200opengl_demo.cpp` program discussed in class. The (public and private) interface is:

```
namespace cs200 {
  class SolidRender {
    public:
      SolidRender(void);
      ~SolidRender(void);
      static void clearFrame(const glm::vec4 &c);
      void setTransform(const glm::mat4 &M);
      void loadMesh(const cs200::Mesh &m);
      void unloadMesh(void);
      void displayEdges(const glm::vec4 &c);
      void displayFaces(const glm::vec4 &c);
    private:
      GLint ucolor,
            utransform;
      GLuint program,
             vao_edges,
             vao_faces,
             vertex_buffer,
             edge_buffer,
             face_buffer;
      int mesh_edge_count,
          mesh_face_count;
  };
}
```

`SolidRender()` — (constructor) creates/initializes a 2D rendering object. Specifically, the constructor will need to perform several initialization tasks. (1) Compile the fragment and vertex shaders. You are required to use the fragment shader

```
#version 130
uniform vec4 color;
out vec4 frag_color;
void main(void) {
  frag_color = color;
}
```

and the vertex shader

```
#version 130
in vec4 position;
uniform mat4 transform;
void main() {
  gl_Position = transform * position;
}
```

You may not modify the code for either shader. (2) Link the compiled shaders into a shader program, and set the value of the private datum `program`. If either the fragment or vertex shader fails to compile, or if the shader program fails to link, the standard `runtime_error` exception should be thrown (do **not** print any text messages to the standard output here or in any of the member functions). (3) Set the values of the private data members `ucolor` and `utransform` by getting the locations of the shader uniform variables `color` and `transform`.

`~SolidRender()` — (destructor) destroys the 2D rendering object. You should delete all resources allocated in the constructor.

`clearFrame(c)` — clears the frame buffer with the color c. Here we are using a floating point RGBA color model, with all values in the range $[0, 1]$.

`setTransform(M)` — sets the value of `transform` in the vertex shader to M. The transformation M specifies how the vertices of the current mesh should map to normalized device coordinates.

`loadMesh(m)` — uploads the mesh data (the vertex array, the edge list, and the face list) to the GPU, and creates two vertex array objects (VAOs) for rendering the either the edges or faces of the mesh. Don't forget to enable the `position` attribute in the vertex shader! The mesh m becomes the current mesh.

`unloadMesh()` — removes the previously loaded mesh from GPU memory.

`displayEdges(c)` — draws the boundary lines of the currently loaded triangular mesh to the frame buffer using the solid color c.

`displayFaces(c)` — draws the faces of the currently loaded triangular mesh to the frame buffer using the solid color c.

Remark

OpenGL architecture is modeled on a state machine. For example, the call `glBindBuffer` is used set a state: the current GPU buffer to use. A subsequent call to `glBufferData` transfers data from the CPU onto into the currently selected GPU buffer. A graphics application may use multiple shader programs. The call `glUseProgram` is used to select the a shader program

to use, and a call such as `glUniform4f` sets the value of a uniform variable of the *currently selected shader program.* When implementing the member functions of the Render class, be aware that a different shader program than the one created in the constructor may currently be selected.

What to turn in

Your submission for this assignment should consist of a single source file, an implementation file named `SolidRender.cpp`. In addition to `SolidRender.h`, you may include any standard `C++` header file. In particular, you will need to include the `stdexcept` standard header file. Note that `SolidRender.h` includes the header files `GL/glew.h`, `GL/gl.h`, `glm/glm.hpp`, and `Mesh.h`.