University of Western Ontario, Computer Science Department
CS3388B, Computer Graphics

Assignment 4 **Due: Friday, March 10, 2023**

General Instructions: This assignment consists of 5 pages, 1 exercise, and is marked out of 100. For any question involving calculations you must provide your workings. *Correct final answers without workings will be marked as incorrect.* Each assignment submission and the answers within are to be solely your individual work and completed independently. Any plagiarism found will be taken seriously and may result in a mark of 0 on this assignment, removal from the course, or more serious consequences.

Submission Instructions: The answers to this assignment (code, workings, written answers) are to be submitted electronically to OWL. Ideally, any workings or written answers are to be typed. At the very least, clearly *scanned* copies (no photographs) of hand-written work. If the person correcting your assignment is unable to easily read or interpret your written answers then it may be marked as incorrect without the possibility of remarking. Include a **README** with any code.

Install Tips: I highly recommend downloading GLEW: `https://glew.sourceforge.net/` and (if working in C++) GLM: `https://www.opengl.org/sdk/libs/GLM/`. My example code (**which you should pillage** to help do this assignment) use GLEW and GLM.

Exercise 1. The goal of this assignment is get comfortable with triangular meshes and working with the camera in world space.

At a high-level, your program will:

- Manipulate the view matrix so that the camera appears to move around in world space
- Read triangle mesh data from a file
- Render textured triangle meshes using VBOs and VAOs

The end product of this assignment is to have a program which loads triangle mesh data and textures from files, renders them, and allows the user to explore that rendered world by moving the camera around via arrow keys.

The Camera.

1. We are going to use a create a limited "first-person camera", where you (the character) are viewing the world as if you were in it. You will directly be controlling the camera.
2. Begin with the camera being positioned at $(0.5, 0.4, 0.5)$ in world space and looking in the direction of $(0, 0, -1)$.
3. You will use the arrow keys (up, down, left, right), to move around the camera.
    - Pressing the up key should make the camera move forward in the direction the camera is currently facing
    - Pressing the down key should make the camera move backward in the exact opposite direction the camera is currently facing
    - Pressing the left key should make the camera rotate counter-clockwise, without moving; it should "rotate in place".
    - Pressing the right key should make the camera rotate clockwise, without moving; it should "rotate in place".
4. The camera should move relatively smoothly. Consider implementing an algorithm something like "for each frame the up arrow key is held down, I move 0.05 units forward" and "for each frame the left arrow is held down, I rotate 3 degrees counter-clockwise", etc.
5. Let your camera move freely based on the above rules. Don't worry about running into or through walls. Free roam and fly like a ghost.

The File Reading.

Create a function which, given a file name, returns a list vertices and a list of faces. It should work as follows:

1. Implement a `VertexData` class/struct. It should have instance variables for position (x, y, z), a normal vector (nx, ny, nz), color (r, g, b), and texture coordinates (u, v). It does not necessarily need to have valid values for all of vertex attributes. Only position is mandatory.

2. Implement a `TriData` class/struct. It should have three instance variables for the indices of three vertices making up the triangle. You can also use one instance variable which is a list of length 3. Your choice.

3. Implement a `readPLYFile` function whose argument is a string representing a file path to a PLY file. This function returns to lists: a list of `VertexData` objects (one object for each vertex specified in the PLY file), and a list of `TriData` objects (one object for each face specified in the PLY file). In python, consider returning a tuple of lists. In C/C++ consider returning be reference using the following function prototype:

```
void readPLYFile(std::string fname, std::vector<VertexData>& vertices,
    std::vector<TriData>& faces);
```

4. Your `readPLYFile` function must handle the vertex properties `x, y, z, nx, ny, nz, red, green, blue, u, v` and any number of vertices. Assume there are no other vertex properties. You can assume `x, y, z` will be specified but any other property is optional.

5. Your `readPLYFile` function must handle faces with a property list `vertex_indices`. You can assume this list is always of size 3.

There are a total of 10 PLY files to read, each with their own bitmap texture. Find all of these files on OWL alongside this PDF.

The Code Structure.

1. Create a class/struct `TexturedMesh` which encapsulates a textured triangle mesh.

2. The constructor of this class should take two file paths: one for a PLY file, and another for a bitmap image file. Each object should use your `readPLYFile` function to read the vertices and faces data from the PLY file. In C/C++ you can use the provided `loadARGB_BMP` function to load the bitmap image. In Python, you can use the PIL library (see this Stackoverflow post).

3. Your class should have instance variables for:

   - An integer ID for the VBO to store the vertex positions
   - An integer ID for the VBO to store the texture coordinates
   - An integer ID for the VBO to store the face's vertex indices
   - An integer ID for the Texture Object created to store the bitmap image.

- An integer ID for the VAO used to render the texture mesh (see the next part below for rendering).
- An integer ID for the shader program created and linked to render the particular textured mesh.

4. The `TextureMesh` class should have a `draw(glm::mat4 MVP)` instance method which is called by the `main` function to render that `TextureMesh` object.

5. In the `main` method, create all the `TexturedMesh` objects and setup a projection matrix with a vertical field of view of 45°. Then, in the render loop: (i) handle use input to move the camera around; and (ii) call `TextureMesh.draw` for each `TextureMesh` object.

The Rendering.

1. Each `TexturedMesh` object should be rendered using a VAO, `glDrawElements`, and custom vertex and fragment shaders. Thus, the `glVertexAttribPointer` and buffer bindings should only occur once, in the constructor, and then never again. The VAO will have captured that state.

2. The vertex and fragment shaders can be very simple. You can use the ones supplied with `L10texture.cpp`.

3. Use blending and use `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`.

4. Make sure you **render transparent objects last**. This includes `DoorBackground.ply`, `MetalObjects.ply`, and `Curtains.ply`

The Submission.

Submit to OWL:

1. Your source code. (One or more files depending on how you defined the classes).

2. A README explaining what you did and why you did it like that, any known bugs, instructions for compiling. It doesn't have to be long, but it should be informative to the person marking.

3. A couple of screenshots of your program rendering the inside of your house. Similar to those on page 2 of this assignment instructions. (Although maybe from different camera angles; your choice).