IMT2521 - Assignment 2A

Due 31st October 2012, 4:30pm.
Worth 20% of total course mark. (i.e. half of the 40% internal assessment)

Part 1: Load objects from a file

You can use the Assignment2-Starting_code.zip on Fronter as a beginning.

In a class called Object make a loadTxtFile method. This method should load should load the vertices and triangles from the file specified as the parameter.

The file format to be used is as follows:

```
<numVertices> <numTriangles>
<x1> <y1> <z1>                    //3 values defining a vertex. 1 vertex per line
<x2> <y2> <z2>
…
<indexA1> <indexB1> <indexC1>     //The 3 indices of a triangle. 1 triangle per line
<indexA2> <indexB2> <indexC2>
…
```

Note that the index value for the first vertex is 1 (not 0). So you will need to subtract 1 before indexing into your vertex array.

There are some test files given: quad.txt, cube.txt, teapot.txt, monkey.txt

Hint: In lab01 we looked at reading values from files. You can find my completed Lab01 code on Fronter, at the end of the lab01 exercise.

Also, I recommend using a `std::vector<glm::vec3>` for you vertices. See the Appendix at the end of this document for more details on using GLM math library.

Part 2: Draw the object using glDrawArrays

In order to do flat shading later, you need to duplicate the shared vertices.
To do this create another (empty) vector, then for every triangle push back 3 vertices.
I.e. For each triangle push_back the 3 corresponding original vertices based on their triangle indices.

Then in your draw() method, use glDrawArrays(GL_TRIANGLES, …) to render the object to the screen.
(If you like you can use Vertex Buffer Objects, but it's not requirement)
You should set the colors of all the vertices to white

Part 3: Calculate normals and flat shade the object

For each triangle face, calculate the normal. The same normal should be used for all 3 vertices in a given triangle.
Hint: Use the cross product of (B - A) and (C - A). And don't forget to normalise it. The GLM library has a glm::cross(...) and glm::normalize(...) function you can use.

Use these normals to render the object with flat shading.
Hint: Remember to enable lighting

Part 4: Display normal vector lines

Now for each vertex on each triangle, render a line from the vertex in the direction of the normal. The line should be of length 0.3. Color the lines red.
GL_LINES can be used to render these

Make the spacebar toggle the drawing of these normal lines on and off.
The lines should be turned off when the program first runs.

Advanced:

For those finish early. Learn how to use the parameters passed in from the command line (e.g argc, argv), and use them to set the model that is loaded.
Note: You can see command line parameters in Visual Studio under the Project Properties: Debugging->Command arguments.
Note: this will also work if you drag the .txt file onto the .exe file to run it.

Expert:

Add a loadObjFile method which can load vertices and faces from simple .obj files.
See the .obj definition at http://en.wikipedia.org/wiki/Wavefront_.obj_file

Hand in

Put your files into a zip file of your name and student number.
Make sure you include all the .cpp and .h source files and the .exe (zipping the whole project folder is fine).
Make sure your code is tidy, and has suitable comments - such that another programmer can follow it.

Appendix: Using the GLM math library

GLM is useful little math library you may find very useful. It provides OpenGL compatible vector and matrix routines.
You can find it at http://glm.g-truc.net/index.html
It's not a requirement to use it, but I recommend it, as it should make things easier for you.

Installing GLM

Click on the download link on the site, and unzip the files together with you other libraries.
This is a header file only library, which means all you need to do is add the base folder to the "Include directories" to use it.
(There are no .lib files or "Library directories" to worry about)

Usage

Look at the "code samples" link on the glm web page.
The vec3 is a useful class for grouping together 3 float values - like a 3D position.
We can create a vec3 called n, for example with `glm::vec3 n(1.0f, 0.0f, 0.0f);`
You can access elements with for example the x component with `n.x`
We can create and push_back a vec3 onto over vector like
`vertices.push_back(glm::vec3(x, y, z));` //where x, y and z are floats
We can add two vec3's with `a = b + c;` //where a, b and c are all vec3's
We can scale a vec3 with `n = n * 2.0f;` or just `n *= 2.0f;`

A std::vector of vec3 will pack continuously in memory [x, y, z, x, y, z, x, y, z …] so can be used in glDrawArray calls directly. You can get the data pointer for example with &vertices.front()
The number of bytes of the vector data would be `vertices.size()*sizeof(glm::vec3)`