Computer Graphics - Assignment 1

1 Overview

For this assignment, you will have the opportunity to familiarize yourself with basic OpenGL concepts and real-time graphics programming. You will implement basic camera and lighting functionality, and render a 3D mesh.

2 Objective

This assignment is designed to give you an opportunity to implement some of the theoretical ideas that are discussed in the lectures and a chance to program graphics applications. Your task is to use the OpenGL shading language (GLSL) to render a simple 3D mesh of a teapot object.

Input: We provide you with a 3D model of the Utah Teapot, a `Shader` class for loading your shader programs, and an example shader with a simple program for loading and rendering the scene using OpenGL. Please see the sections below for more details on each item.
Output: The 3D model does not contain information on the vertex normals. You must compute the normals and use these to render the 3D mesh with three separate shaders that implement the lighting techniques discussed in the lectures. These are flat shading, Gouraud shading, and Phong shading. You must also write a readme file explaining the techniques you have used in the assignment and provide screenshots of your work.
Requirements:

- Calculate the normals for the provided teapot mesh.
- Write three shader programs that implement flat shading, Gouraud shading, and Phong shading.
- Submit a few screen shots of your program's renderings.

- Use good code style and document well. We *will* read your code.
- Create a file named "readme.txt" containing the details of your implementation and instructions for compiling and running your code.
- The program must compile and run on DICE. If it does not, you run the risk of getting 0 marks.

Should you manage to achieve all of these requirements then you will receive a good mark. The following is a list of possible additional techniques that can be implemented for further marks:

- Control of the camera/teapot model using keyboard/mouse input. This can be achieved by manipulating the view and model matrices respectively.
- Alternative shading e.g. Toon shading
- Bump,light,displacement maps.
- Texture mapping. UVs are provided in the teapot.obj file.
- Anti-aliasing.
- Shadows.
- Environment Mapping.
- Reflection and Refraction.

A number of these techniques are taught at the sites listed in the "Notes on GLSL" section. If they are not there then a quick internet search for "technique name" + "GLSL" will normally find a decent tutorial for you.

3 DICE Instructions for use

This demonstration program is written in C++. It needs to be compiled into executable code before running.

To compile the demo program, open up a terminal window and navigate to the directory containing the source code. Here you can run the command:

```
make run
```

This will compile the default source code and run the demo program with the teapot mesh as input. You can perform these actions separately by first compiling with:

```
g++ -o demo1 demo1.cpp Shader.cpp -lglut -lGLU -lGL -lGLEW
```

This compiles the code and links libraries glut, GLU, GLEW and openGL. You may then run the program with the following command:

```
./demo1 teapot.obj
```

Please note the .obj loading function (`TriangeMesh::LoadFile(char *)`) does not account for .obj files containing information for normal vectors. If you should wish to load other .obj files then you will need to adjust this method accordingly.

4 Notes on the OpenGL Shading Language (GLSL)

The DICE environment should run OpenGL 2.1 and therefore will be working with at least GLSL version 1.2 according to the OpenGL documentation. It appears that shaders written for GLSL version 1.3 can also be compiled and run on DICE and will offer some functionality that will be useful for completion of the coursework. The version number for the shader code is specified in the vertex and fragment shader files by the command `#version 130` (change to `#version 120` to work with GLSL 1.2). This is not the most up-to-date version of GLSL but the syntax for GLSL1.3 is more similar to the current version than GLSL1.2. Please **DO NOT** write shaders for any version higher than this as it will likely not run on the marker's computer. If the shaders do not compile or run we cannot see your work and you run the risk of receiving a zero mark. The full reference for GLSL can be found here: `http://www.opengl.org/sdk/docs/manglsl/`. Be sure to check that the function or variable is available in the GLSL version you are working with. Debug output from the code should help you with this. There are several resources online that teach basic GLSL concepts. These include:

- `http://www.swiftless.com/glsltuts.html` - Covers a lot of the basics for using GLSL shaders, relies a lot on using the GLSL built-in variables
- `http://en.wikibooks.org/wiki/OpenGL_Programming` and `http://www.lighthouse3d.com/opengl/glsl/` - These sites are better for

showing how to provide your own structures and values to the shader. They do use a slightly older version of GLSL however the concepts still hold. Be particularly careful of their description of "varying" and "attribute" variables. This syntax is used in GLSL 1.2 but not in GLSL 1.3 where it has been replaced with "out" and "in" variables respectively.

- `http://www.opengl-tutorial.org/` - Along with the other sites, this set of tutorials covers some more advanced techniques with shaders.

5 Notes on the source code

Below is a short description of the source code files provided to you for this assignment. The files themselves contain a number of comments on functionality of the program and should be largely self-explanatory. Should you have any issues then please contact the course teaching assistant (see contacts below).

- *demo1.h* - Used to include necessary utility code and defines the `TriangleMesh` class. Feel free to change this to your needs.
- *demo1.cpp* - Defines the entry point for the program. Sets up the OpenGL window and callback methods for displaying the rendered image and handling mouse input. Loads the provided .obj file and renders it using a basic shader. Along with the shader files this will be where you implement the majority of your work.
- *Shader.h* - Defines the `Shader` class interface. You should not need to change this file.
- *Shader.cpp* - An implementation of the `Shader` class interface for loading and running OpenGL (GLSL) shaders. This file should not be changed.
- *shaders/exampleShader.vert* - Defines the behaviour of the OpenGL vertex shader. This file is loaded by the `Shader` class. You should do most of your work here.
- *shaders/exampleShader.frag* - Defines the behaviour of the OpenGL fragment shader. This file is loaded by the `Shader` class. You should do most of your work here.

The source code includes headers from the OpenGL Mathematics library (`http://glm.g-truc.net/0.9.4/index.html`). These headers define basic vector and matrix classes that correspond to the classes used in OpenGL. Of particular interest will be the `glm::vec3`, `glm::vec4`, and `glm::mat4` classes as well as the methods for producing specific matrices relevant to computer graphics: `glm::ortho`, `glm::perspective`, and `glm::lookAt`.

The vertices of the mesh are rearranged according to the triangle order (`demo1.cpp:lines 99-105`). This is done so that the call to `glDrawArrays` can be made. When calculating the normals for your mesh please be sure to assign each normal to its corresponding vertex. Alternatively, you can remove the code that rearranges the vertices and use `glDrawElements`. This will require creating an array of indexes for the triangle primitives. Use the `Triangle` class to help you with this.

The source code uses Vertex Buffer Objects (VBOs) to store the mesh information for the shader program. The code could be made much cleaner if you were to use Vertex Array Objects (essentially collections of VBOs). While this is not necessary, it may help make the process clearer.

6 Hints

- Read and understand demo1.h and demo1.cpp first. If you don't understand, contact the course's teaching assistant (see below for details) or ask a fellow student.

- If you have never experienced GLSL before, take some time to go over it. Read through some of the basic tutorials at the websites listed in the "Notes on GLSL" section. If you are having trouble then don't be afraid to ask!

- Read and understand how the source code interacts with the shader programs. Find out how variables are passed to the program. In particular, look at the use of Vertex Buffer Objects for providing positions, normals and color info to the shader.

- Understand the difference between "uniform" variables,"in" variables and "out" variables in shaders. These are useful for knowing how you can get the right information to the shaders as well as how information is passed from the vertex shader to the fragment shader.

- Be careful when writing your shader code. A lot of the interaction with the shader program relies on case-sensitive matching of variable

names. If something is misspelt it can prevent your shader program from compiling and you will render nothing to the screen. The `Shader` class outputs debug info to the terminal so make sure to check everything is compiling correctly.

- Make sure you understand how the lighting equations work. The actual implementation of the three shaders will be very similar but they all rely on knowing these equations.

- Document your work well. Be sure to include details in your readme about the approach you took towards the coursework. A perfect solution with no documentation will not receive as high a mark. If you cannot get your shaders to work try to explain why you think this might be happening.

7 Notes on .obj files

The obj file is ascii encoded, and so can be viewed by a text editor. Lines starting with v are vertices. Lines starting with vn are vertex normals. Lines starting with f are faces. In the supplied file (teapot.obj), all faces are triangles.

This is a subset of the specification. The full specification for obj files is not commonly followed in parsers or exporters. The simple importer supplied does not deal with many aspects of the full specification including vertex normals. Further information can be found here: `http://www.martinreddy.net/gfx/3d/OBJ.spec` and here:`http://en.wikipedia.org/wiki/Wavefront_.obj_file`

8 Contacts

Should you have any questions regarding the practical please feel free to contact the course teaching assistant at: j.henry@ed.ac.uk