COSC342 Assignment 2 Option2

Due: 5pm, 22th May 2017*

Building a render engine

For this assignment you are asked to write a render engine using OpenGL. We will provide some skeleton code. Some of the skeleton code will be further explained in the upcoming labs. It is contained within the

/home/cshome/coursework/342/pickup/Assignments/Assignment2OptionB directory.

The skeleton code is documented online.

1. Starting with the skeleton program that we have provided, write and test routines that render model files (.obj files), meeting the requirements below. The skeleton program creates the window and the OpenGL context, provides methods for basic model and material loading from obj and material (mtl) files using the Open Asset Import Library[1], and sets some useful data structures.

2. Test your program with our sample files (person.obj and earthobj.obj within the `Skeleton` directory) and with files of your choice.

3. Write a short report, describing how your program works and how you tested it. Describe any known flaws in your program.

4. Use the `submit342` script to provide us with a directory that contains:

   - Code that we can compile and run under the MacOS environment in the CS labs using the CMake file you provide. In order to avoid too large submissions, make sure that you only submit the CMakeList.txt, the **common** directory and the application directory where renderApp.cpp is located e.g. **Skeleton** directory. The best option is to create a clean directory and copy CMakeList.txt, **common** and your application directory and submit this one. Don't submit the binaries for your application and the external libraries.
   - A sample image file, which we will use in an art display for the course, and the model files you used to generate it (if you are using a different model or if you changed any of the provided models).
   - A short report as a pdf file. (This should usually be about a page in length, and not be more than three pages long.)

*The default late penalties apply.

[1] http://assimp.sourceforge.net

We will e-mail you to confirm that we have received your assignment within a week of the due date. Keep a copy of your work at least until the assessment has been returned to you.

Requirements for your render engine

We require that you complete the following tasks within the structure of the skeleton code provided.

1. Implement complete obj model loading (we provide help with assimp library in the lab and the skeleton code contains basic loading function for a single mesh without any shading). Main files for implementation: common/Objloader.cpp.

2. Implement complete model rendering (the skeleton code provides functionality to render a single mesh). Main files for implementation: common/Group.cpp.

3. Make sure all materials are loaded from the mtl file (Usually a mtl file contains multiple materials, material indexing allows to reuse materials between different meshes). Implement correct material indexing from the loaded mtl file. Main files for implementation: common/Objloader.cpp, common/Group.cpp.

4. At the moment only the diffuse parameter is read from the mtl file. Read additional parameters to support the material setttings for diffuse color, ambient color, specular color and specular exponent (parameters Ka, Kd, Ks, and Ns from the mtl file). Also read the texture settings for the diffuse texture map (map_Kd). Make sure that all materials for all meshes are processed. Main files for implementation: common/Objloader.cpp, common/Group.cpp.

5. Pass these additional material parameters to the shaders (main files for implementation: common/MTLShader.cpp and mtlShader.vert and mtlShader.frag in Skeleton).

6. Implement Phong shading and Blinn-Phong reflection model using the vertex and the fragment shader (main files: mtlShader.vert and mtlShader.frag in Skeleton).

7. Implement support for transparent material (use transparency setting from the mtl file - parameter d). Your result should look like Figure 1, Middle. The main files for implementation: renderApp.cpp to support blending and mtlShader.frag in the Skeleton directory.

8. Implement a Special Effect shader of your choice (Toon, Sepia, Black-and-White (BW)) - make this an option to select using an input key. The result for BW could like Figure 1, Right. Main files for implementation: renderApp.cpp and mtlShader.frag in Skeleton and common/MTLShader.cpp.

Figure 1: Left) Basic mesh renderer, Middle) Transparency. Right) Black-and-White effect

Marking scheme

3 marks	Report, with an emphasis on testing.
2 marks	Your sample scene image and the corresponding model file (obj).
2 marks	Complete model loading.
1 marks	Complete model rendering.
2 marks	Correct material indexing.
1 marks	Read other material parameters (Ka, Ks, Ns) and texture settings.
2 marks	Pass material parameters to shader.
3 marks	Phong shading and Blinn-Phong reflection model.
2 marks	Transparency option (mtl parameter d).
2 marks	Special effects option.

Hints

- There is no need to implement your own model loading or load anything manually in the main program. For finishing the model loading and rendering, you only need to complete the existing code for model loading and rendering in common/Objloader.cpp and common/Group.cpp.
- For implementing shading and reflection: remember the difference between Phong shading and Phong Reflection from the lecture. This will help you to implement Phong shading and Blinn-Phong reflection. Also have a look into the second OpenGL lab and how we implemented Phong shading and Phong reflection.
- Remember from the first OpenGL lab, how to use the swizzel operator, e.g output.r = 1.0. In the assignment we use a vec4 in the fragment shader to compute the output color. A vec4 allows you to set RGB as well as an alpha value (this is helpful for implementing the transparency). If you only want to write RGB for the first parts of the assignment, use e.g. color.rgb = vec3(1.0,1.0,1.0) to shade fragments with a white colour. If you want to manipulate the alpha value (the opacity), you would work with color.a = 1.0.
- In order to create transparency effects, you need to enable blending. Use glEnable and GL_BLEND for enabling it and setting up the way how to blend by using glBlendFunc. A hint from khronos.org:

"Transparency is best implemented using blend function (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)".

- You can access the transparency/opacity value using AI_MATKEY_OPACITY.
- There is already a *setRenderMode* method in the MTLShader class that can be used for switching on your implementation of a special effect for the last task. You still have to implement it in the fragment shader.

Final notes

Remember this is an assignment in graphics, not in production programming. You will not get any extra credit for extravagant solutions (although we may well smile while marking them). High marks will be given for a well explained solution that is easy to follow. Programs must be commented of course, but no more than necessary for us to be able to read them.

You may discuss conceptual issues relating to this assignment with others, but all the work you hand in must be your own, except for the parts of the given skeleton program.