Assignment 2: 3D Geometry, Lighting, and Shading

CS148 Fall 2015-2016

Introduction

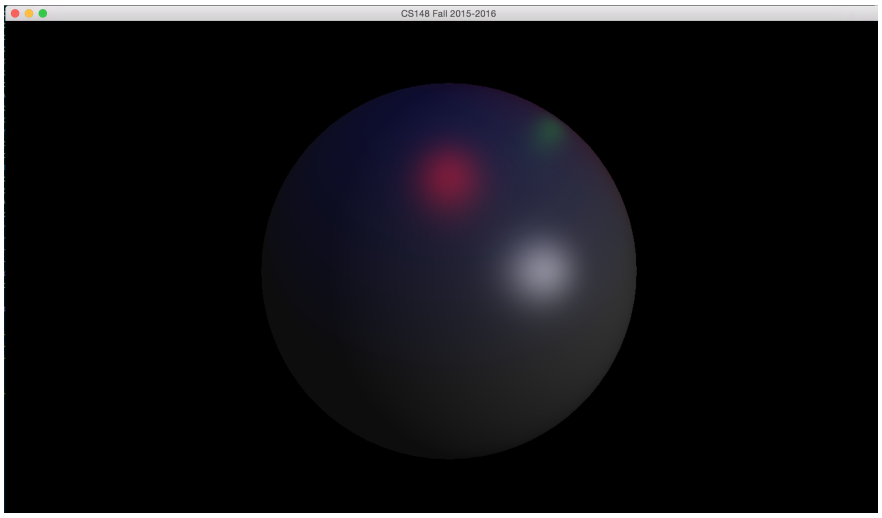

Figure 1: A Sphere with Three Moving Lights using Blinn-Phong Shading

In this assignment we will explore the process of getting a 3D object onto the screen, lit, and shaded. You will also become more familiar with GLSL (the OpenGL shading language). Do note that there are many ways to light and shade a 3D scene! The method we chose in the assignment framework is just one way of doing so. Feel free to implement your own way of handling lights and creating your own shaders!

Finally, you will begin the process of creating your final image (for the OpenGL assignments)! You will need to find 3D geometry and place it in your scene, adjust the camera, and add and adjust lights to get a nice image! You may have to iterate this process many times until you get an image to your liking. Remembmer, start early and ask questions if you get stuck!

Useful Resources

1. OpenGL Programming Guide Chapters 3-5, 7 (the book is not required, but recommended!)
2. OpenGL 4 Documentation: here. Note that these pages are for OpenGL 4.5 but most should be equivalent (unless otherwise noted on the function's page).
3. SDL 2 Documentation: here.
4. GLSL Tutorial: here.
5. GLM: Manual and Documentation.
6. C++ Reference: here

7. Open Asset Import Library Documentation: here
8. Assignment Framework Documentation: This can be found in "source/doxygen"

Finding Meshes Online (For Free!)

1. TurboSquid.
2. PolyCount. Note that it may be useful to Google "Polycount free models".
3. TF3DM.

Note that it is hard to find high quality models and textures online. You may find it useful to look for people who have used "Zbrush" to create high-poly models and then trying to find a way to get an OBJ from that.

Creating your Own Meshes (For Free!)

1. Autodesk Maya
2. Autodesk 3ds Max
3. Blender

I personally find Blender very hard and uninuitive to use (it has a pretty steep learning curve, in my opinion) compared to Maya and 3DS Max, but the choice is up to you! Additionally, you can generate your models procedurally in code.

Assignment

In this assignment's code, you will see the use of "DISABLE_OPENGL_SUBROUTINES" which is only defined on Mac OS X. This is due to OpenGL subroutines being broken on Mac OS X (at least with a NVIDIA graphics card). This does not affect Windows and Linux users. As a result, I have two versions of the shaders used, one with subroutines and one without subroutines. If you are on Mac, make sure you look at/modify the shaders in the "noSubroutine" folders.

Assignment 2 Code Explanation

Look in the Assignment2 class documentation in the assignment framework documentation found at "source/-doxygen".

Asset Loading

Assets (models, textures) are stored in the "assets" folder. When the program is configured and compiled, a C++ preprocessor definition is added to tell the program where to look for this folder. Much like the shaders, when you load an asset (whether it be a mesh or a texture), make sure the path is specified relative to the "assets" folder.

Pointers to Get You Started

- The code to generate 'RenderingObject's from a file (OBJ, FBX, whatever) is found in "common/U-tility/Mesh/Loading/MeshLoader.h/cpp". Note that we make use of the Open Asset Import Library which takes care of actually parsing the file; all we do is read in the data given to us by the library.

Note that currently the LoadMesh function does not support reading in materials from the file so you will have to create the material parameters yourself!

- The shaders "shaders/brdf/blinnphong/vert" demonstrate flat shading while the shaders in "source/shaders/brdf/blinnphong/frag" demonstrate smooth shading. Run the assignment to see the different between the two.
- The OpenGL shaders are tightly coupled to the corresponding shader class in the assignment framework. Look at the GLSL shaders found in "source/shaders/brdf/blinnphong/vert" (or frag) and look at "common/Rendering/Shaders/BlinnPhongShader.h/cpp" to see how certain variables correspond.
- We have provided an implementation of Blinn-Phong shading. You can read more about it here. You may find the Wiki to be more useful.
- The code has a distinction between a "Light" and its properties ("LightProperties"). "Light" will control physical properties such as its location, direction its facing, etc. "LightProperties" on the other hand will control the actual shading properties like its diffuse color, specular color, etc. As a result, the type of "LightProperties" is dependent on which "ShaderProgram" subclass you use (i.e. BlinnPhongShader).
- As an addendum to the previous point, to add a new type of light (i.e. a directional light), you will want to have a subclass of "Light", add a new light type to "LightType" (in Light.h), and then make sure you handle the new type of light in BlinnPhongShader.cpp in the "SetupShaderLighting" function. In your new light class, make sure you call the Light constructor with the new enum type! Additionally, make sure you have everything setup in the GLSL shader to handle your changes. "SetShaderSubroutine" and "SetShaderUniform" will fail silently if they can't find something.

Going Beyond the Assignment (Optional)

- The type of rendering implemented currently is a multi-pass forward renderer. What this means is that for every object, we go through each light and run the vertex/fragment shader once. To compute the final image, we blend the colors together to get the final result! This technique does not scale, however. As the number of lights increases and as the complexity of the scene grows, your fragment shaders will be doing a lot of work (a lot of which is unnecessary). A solution to this is to use deferred shading.
- Try your hard at implementing even more advanced rendering techniques such as screen space ambient occlusion! (Wiki Page).
- The Blinn-Phong shading model isn't the only shading model that exists! It is merely one example of a BRDF. Check out Epic's shading presentation at SIGGRAPH 2013 to check out how they do shading in Unreal Engine 4 here. The Disney presentation they reference can be found here.

Questions

1. In the context of a vertex shader, what is the difference between a 'uniform' variable and a variable passed to the shader via the 'in' keyword.
2. When would you expect diffuse lighting for a vertex to be strongest in Blinn-Phong shading (i.e. when X is pointing towards Y).
3. What is the difference between a directional light and a point light?

Feedback (Optional)

1. On a scale of 1-10 [10 being the most work]: How much work was this assignment?
2. Do you wish the assignment framework abstracted less of the underlying OpenGL API? More? Just right?

Grading

This assignment will be graded on the following requirements

- There are at least two different geometric models in your scene.
- There are at least two types of lights in the scene (i.e. a point light and a directional light).
- The student is able to adequetely answer questions asked by the CAs.

according to the following rubric.

- + – Exceeds the requirements via one or more artistic/technical contributions
- ✓ – Meets all of the requirements
- − – Does not meet the requirements but still produces a drawing.
- 0 – The submitted solution does not produce a drawing.