CMSC 23700 **Introduction to Computer Graphics** **Project 2**
Autumn 2017 **October 9, 2015**

Shading and lighting

Due: Wednesday October 18 at 10pm

1 Summary

This assignment builds on your Project 1 code by adding directional lighting and texturing to your renderer. You should start by fixing any issues with your Project 1 code.

2 Description

The focus of this project is on supporting two new rendering modes: diffuse rendering using a directional light and texturing. In the first mode, you will compute the surface color of objects based on the object's color (as before) combined with the ambient and directional light. In texturing mode, you will use texture mapping to determine the color, which will also be combined with the lighting information.

2.1 Scenes

To support the new features, the scene description format has been extended with additional information and resources. Specifically, the `scene.json` file now contains information about the scene's lighting and the object models now have associated *materials* that are used to specify the textures that will be applied to the objects. Figure 1 gives an example of the new scene description format. We will provide a enhanced version of the `Scene` class that supports the new features. As before, your code will use the scene object to initialize your view state and object representations.

The lighting field is a JSON object that has three fields:

1. The `direction` field is a vector that specifies the direction of the light ($\langle 0, -1, 0\rangle$ in the example in Figure 1, which means that the light is shining from directly overhead).

2. The `intensity` field is an RGB triple that specifies the intensity of the light ($\langle 0.8, 0.8, 0.8\rangle$ in the example).

3. The `ambient` field is an RGB triple that specifies the intensity of the scene's ambient light ($\langle 0.2, 0.2, 0.2\rangle$ in the example).

OBJ files have mechanisms for associating *materials* with groups in a model. The materials are defined in a separate file and are used to control the rendering of the triangles in the associated group.

```
{
  "lighting" : {
      "direction" : { "x" : 0, "y" : -1, "z" : 0},
      "intensity" : { "r" : 0.8, "b" : 0.8, "g" : 0.8},
      "ambient" : { "r" : 0.2, "b" : 0.2, "g" : 0.2}
    },
  "camera" : {
      "size" : { "wid" : 1024, "ht" : 768 },
      "fov" : 120,
      "pos" : { "x" : 0, "y" : 3, "z" : -10},
      "look-at" : { "x" : 0, "y" : 3, "z" : 0},
      "up" : { "x" : 0, "y" : 1, "z" : 0}
    },
  "objects" : [
      { "file" : "box.obj",
        "pos" : { "x" : 0, "y" : 0, "z" : 0},
        "color" : { "r" : 0, "b" : 1, "g" : 0}
      }
    ]
}
```

Figure 1: An example `scene.json` file

For the purposes of this project, you will continue to use the monochromatic shading of objects in wireframe, flat-shading, and diffuse rendering modes. In texturing mode, however, you will use a texture to define the color of the object. The texture's name is specified by the `diffuseMap` field of the `Material` structure. The `Scene` class provides a mapping from the texture names to 2D images (`cs237::image2d`). You will need to initialize an OpenGL texture object from the image data[1] that you can then use when rendering the associated object.

2.2 Meshes

In Project 1, you should have implemented a `Mesh` (or similarly named) class that contained the vertex-array object (VAO) ID, a vertex-buffer ID, the element-buffer ID, and the primitive type for a model. For Project 2, you will need to extend this representation to include two additional vertex-attribute buffers: a normal buffer and a texture-coordinate buffer. As before, you can initialize these buffers directly from the model's data.

2.3 Lighting

The scene description defines a single directional light. To compute the lighting at a point on an object, you need to consider the following factors:

- the light's direction, specified as a unit vector $\mathbf{l}$,
- the light's intensity I_l,

[1] The common code library provides the `cs237::texture2D` class to help with managing OpenGL texture objects.

- the ambient lighting intensity I_a,
- the color of the object C_{obj}, and
- the unit normal-vector to the surface at the point on the object $\mathbf{n}$.

Then the computed illumination for the point is given by the equation

$$C = (I_a + \max(0, -\mathbf{l} \cdot \mathbf{n})I_l)C_{obj}$$

where the product of colors is computed as a per-channel multiplication (this operation is sometimes called *modulation*).

Lighting is computed in the fragment shader, but you will need to supply the vertex normals to your shader program, transform the normal vector in the vertex shader, and pass it to the rasterizer. Vertex normals are provided as part of the OBJ file format. Note that interpolated normal vectors are **not** guaranteed to be unit-length, so you will need to renormalize them in the fragment shader. Also remember that the transformation of normal vectors uses a different matrix than the transformation of vertices.

2.4 Texturing

Texturing mode uses the same lighting equation from above, but with the exception that the object's color (C_{obj}) is determined by indexing into a texture. The OBJ file assigns texture coordinates to each vertex, so you can create a vertex buffer for these and pass them in as an additional attribute to the vertex shader. The vertex shader should then pass them on to the rasterizer, for linear interpolation. The interpolated coordinates are then used to index into the texture (called a *sampler* in GLSL) in the fragment shader.

2.5 Renderers

As in Project 1, we used subclasses of the abstract base class `Renderer` to represent different rendering modes (*i.e.*, `WireframeRenderer` and `FlatShadingRenderer`). For ths project, you will need to extend the class hierarchy with classes that support direct lighting (`DirectLightingRenderer`) and texturing (`TexturingRenderer`). Since both these modes involve lighting computations (and the associated state), we recommend defining an intermediate class (say `LightingRenderer`) that factors out common code and state.

2.6 User interface

You will add support for two new commands to the Project 1 user interface:

`d D`	switch to diffuse-rendering mode (no textures)
`t T`	switch to textured mode

In addition, your viewer should provide the camera controls that you implemented in Project 1.

3 Sample code

Once the Project 1 deadline has passed, we will seed a `proj2` directory with a copy of your source code and shaders from Project 1. We will update this code with a new implementation of the `Scene` class, but otherwise your source code will be unchanged. The seed code will also include a new `Makefile` in the build directory and new scenes.

4 Summary

For this project, you will have to do the following:

- Fix any issues with your Project 1 code.
- Modify your Project 1 data structures to support direct lighting and texture mapping.
- Write a shader that renders objects using the directional light.
- Write a shader that renders objects using texture mapping.
- Implement renderer classes for direct lighting and texture mapping.
- Add support for the new rendering modes to the UI.

5 Submission

We have set up an **svn** repository for each student on the `phoenixforge.cs.uchicago.edu` server and we will populate your repository with the sample code. You should commit the final version of your project by 10:00pm on Wednesday October 18. Remember to make sure that your final version **compiles** before committing!

History

2017-10-14 Removed sentence fragment.

2017-10-13 Extended deadline to Wednesday.

2017-10-08 Original version.