CMSC 23700 **Introduction to Computer Graphics** **Project 2**
Winter 2014 **January 29**

Lighting Effects

Due: Monday, February 3 at 10pm

1 Description

In this project, you will shader code to support point lights, texturing, and bump mapping. The rendering framework is a modification of Project 1 and will be provided.

1.1 Lighting

In this project, you will support a third lighting mode: point lighting. Point lights model light sources that are near or within the scene being rendered. For this project, your shader should support both directional and point lighting. Unlike in Project 1, the lighting position information will be passed to the shader in uniform variables. Remember that point lights are attenuated based on the distance between the light and the surface. For this project we recommend using an attenuation factor of $\frac{1}{distance}$.

For directional and point lighting, you should use an ambient lighting level of $\langle 0.25, 0.25, 0.25 \rangle$ and an intensity of $\langle 1.0, 1.0, 1.0 \rangle$ for the light source.

1.2 Texturing

The second step of the project is to use a texture for the walls. The `proj-2/data` directory contains a PNG (Portable Network Graphics) file called `uchic-rgb.png` that contains the image that you will use to texture the walls (it is the seal of the University of Chicago). The CMSC23700 common library code contains support for loading PNG image files and for making OpenGL textures from them. In a shader program, a texture is called a *sampler* and must be a uniform variable.

1.3 Bump-mapping

Bump mapping (sometimes called normal mapping) is a technique for creating the illusion of an uneven surface by perturbing the surface normals. The `proj-2/data` directory contains a PNG file called `uchic-norm08.png` that contains a three-channel texture that represents the normal vectors on the surface. These normals are stored in the tangent space of the surface, so you will have to map the light and eye vectors to tangent space before you can compute the lighting information. Specifically, the red and green channels hold the X and Y components of the normal vector, which should coincide with the S and T axes of the texture. The blue channel holds the Z component of the normal. Note that the normal vectors stored in the bump texture are not unit vectors.

1.4 User interface

The sample code extends the Project 1 commands with two additional operations:

- `l` cycle through lighting modes (no lighting, directional lighting, point lighting).
- `b` toggle bump-mapping on/off

1.5 Shader interface

For this project, you should implement a single unified shader program (*i.e.*, one vertex shader and one fragment shader). The shader source should be put into files `proj-2/data/shader.vsh` and `proj-2/data/shader.fsh`. The full set of uniforms and vertex attributes used to communicate to the shader is as follows:

```
const int NO_LIGHT = 0;          // lighting disabled
const int DIR_LIGHT = 1;         // directional lighting
const int POINT_LIGHT = 2;       // point lighting

const int NO_TEXTURE = 0;        // texturing disabled
const int TEXTURE = 1;           // walls are textured
const int BUMP_MAPPING = 2;      // walls are textured and bump
                                 // mapping is enabled

uniform int lightingMode;        // NO_LIGHT, DIR_LIGHT, or POINT_LIGHT
uniform int texturingMode;       // NO_TEXTURE, TEXTURE, BUMP_MAPPING

uniform mat4 modelView;          // model-view transform
uniform mat4 projection;         // projection transform

uniform sampler2D colorMap;      // color texture for walls
uniform sampler2D bumpMap;       // bump-map texture for walls
uniform vec3 primColor;          // the color of a non-texture-mapped
uniform vec3 ecLight;            // primitive light vector; for DIR_LIGHT,
                                 // this is the light's direction vector,
                                 // while for POINT_LIGHT this is the
                                 // light's position. In both cases, the
                                 // vector is in eye coordinates.

layout (location = 0) in vec3 coord;    // object coords for vertex position
layout (location = 1) in vec3 norm;     // object coords for vertex normal
layout (location = 2) in vec4 tanVec;   // tangent vector with bitangent
                                        // sign
layout (location = 3) in vec2 texCoord; // texture coordinate
```

The two uniforms `lightingMode` and `texturingMode` control the function of the shader.

2 Hints

There are a number of different rendering paths depending on the lighting mode, the object (balls do not have a texture), and whether bump mapping is enabled. The computation of the fragment color has four parts:

1. Calculation of the surface normal (depends on `lightingMode` and `texturingMode`).
2. Calculation of the lighting intensity (depends on `lightingMode` and the normal computed in Part 1).
3. Calculation of the surface color (depends on `texturingMode`).
4. Combining the lighting intensity from Part 2 with the surface color from Part 3 to compute the fragment color.

Note that the first two of these are trivial when the lighting mode is `NO_LIGHT`.

3 Submission

We will create a directory `proj-2` in your CMSC 23700 Phoenix Forge repository. The projects will be collected at 10:00pm on Monday February 3 from the repositories, so make sure that you have committed your final version before then.

History

2014-01-29 Clarified lighting.

2014-01-29 Added missing sampler uniforms.

2014-01-29 Original version.