CMSC 23700 **Introduction to Computer Graphics** **Project 2**
Winter 2010 **January 26**

Per-pixel lighting

Due: Friday, February 5 at 10pm

1 Description

In this project, you will replace the simple vertex shader from Project 1 with multiple shaders that implement more sophisticated lighting schemes. For the balls, you will use a per-pixel shader (*i.e.*, a Phong shader) that renders each fragment using ambient and diffuse lighting. For the walls, you will use two or more shaders that implement texture mapping, specular mapping, and bump mapping.

1.1 Per-pixel lighting

The first step of the project is to implement a shader that does per-pixel diffuse lighting. This shader will be similar to the one you implemented for Project 1, except that the color calculations will be in a fragment shader. The vertex shader will be much simpler: it needs to compute the transformed position and normal vectors and pass them on to the fragment shader as interpolated values (*i.e.*, `varying` variables). This shader will be used to render the balls in the final version of the project, but you can test it on the walls too.

1.2 Texturing

The second step of the project is to use a texture for the walls. The file `color.png` contains a 512×512 image that you should use as a texture for the walls. The file `image.h` defines functions for loading the image data and loading the data into an OpenGL texture.

To use a texture in a shader program, you first need to allocate a texture ID (using `glGenTextures`. Then bind the texture and load the data. You will also want to set various texture properties using the `glTexParameter` functions. In a shader program, a texture is called a *sampler* and must be a uniform variable.

1.3 Specular map

Your shader for the walls will also implement specular lighting. Recall that the luminance for specular lighting is given by the equation

$$I_s = \max(\mathbf{r} \cdot \mathbf{v}, 0))^n L_s S_s$$

where $\mathbf{r}$ is the unit reflected light vector, $\mathbf{v}$ is the unit view vector, n is the Phong exponent, L_s is the specular light source, and S_s is the surface's specular color.

Just as we use textures to In addition to supplying color information, textures can be used to specify material properties. A specular map is stored in a four-channel texture, where the RGB channels hold the specular color and the alpha channel has the phong exponent. Since the samples read by your shader are in the range $[0, 1]$ and the Phong exponent usually is in the range [0,128], you will need to scale it before using it as an exponent.

1.4 Bump-mapping

Bump mapping (sometimes called normal mapping) is a technique for creating the illusion of an uneven surface by perturbing the surface normals. We use a three-channel texture, to represent 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.

1.5 User interface

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

`b`	toggle bump-mapping on/off
`s`	toggle specular highlights

2 Hints

You will need at least two different shaders (one for the walls and one for the balls), but we recommend writing two shaders for the walls (i.e., one with bump mapping and one without).

To support multiple textures, you will need to use the `glActiveTexture` function to specify which texture unit subsequent calls to `glBindTexture`, *etc.* affect. To specify textures to the shader program, use `glUniform1i(loc, i)`, where `loc` is the location of the shader variable and `i` is the index of the texture unit (*e.g.*, use `1` for `GL_TEXTURE1`).

For bump mapping, you need to think carefully about what coordinate space things are in and how to transform them. Unlike the previous project, where lighting was computed in camera-space, for bump mapping you want to compute the lighting in the tangent space of the surface being rendered.

3 Submission

We will set up a **gforge** repository for each student on the Computer Science server (see the *Lab notes* (Handout 2) for more information on **gforge**). We will collect the projects at 10pm on Friday February 5th from the repositories, so make sure that you have committed your final version before then.

History

2010-01-31 Revised due date.

2010-01-30 Added additional hints.

2010-01-26 Original version.