Coms 331 **Assignment 10** **Fall 2019**

Introduction

In this assignment, we will implement ambient and diffuse lighting. This will make the 3-dimensional walls recognizable as 3-dimensional objects

The User Interface

The user interface is the same as in Assignment 9. The lighting effects should not change with a change in the viewpoint because neither the ambient nor the diffuse lighting depend on the viewpoint.

The Lighting Model

This model will include the ambient light and the diffuse light. Each is a vector, normally set to a gray level (bright or dim white light). The ambient light is usually low and the diffuse light is usually high. We will need to introduce several new variables:

```
vec3 light_amb;
vec3 light_diff;
vec3 light_pos;

GLint light_amb_loc;
GLint light_diff_loc;
GLint light_pos_loc;
```

These variables should be initialized and used in the usual way.

The `setLight()` Function

Write a function named `setLight()` that will send the values of the uniform variables `light_amb`, `light_diff`, and `light_pos` to the shaders.

The Normal Matrix

The 4×4 model matrix that transforms vertices from model coordinates to world coordinates will not correctly transform normal vectors. Using only the space coordinates x, y, and z, the correct transformation matrix for normal vectors is the inverse transpose of the (upper-left) 3×3 model matrix. The `ModelStack` class computes this matrix in the private member function `invtran()`, with prototype

```
GLfloat* invtrans(mat4& m)
```

and it sends the matrix to the shaders. What you need to do is to set up a location for the normal matrix and then pass it to the `ModelStack` object, using the member function `setNormalLoc()`, in the same way that the location of the model matrix is sent, using the member function `setModelLoc()`.

The Vertex Shader

There will be several changes in the vertex shader, all having to do with the vertex's position and normal vector. (The color will still be pass-through.) The fragment shader must compute the dot product of the light vector and the normal vector. The light position is defined in world coordinates and it is a uniform variable in the fragment shader. The vertex's position was defined in model coordinates and was passed to the vertex shader as a vertex attribute. We need to send it from there to the fragment shader in world coordinates to be compatible with the light position. Thus, we compute the product

```
pos = model*vec4(vPosition, 1.0f);
```

and send `pos` to the fragment shader. In the meantime, we go on to compute

```
gl_Position = proj*view*pos;
```

Also, the vertex normal vector is received as `vNormal` (a `vec3`). You need to multiply it by the normal matrix (`uniform mat3`) and output the resulting normal vector to the fragment shader. (Why can this not be done in the fragment shader?)

The Fragment Shader

It is in the fragment shader where we do the lighting calculations. We have available the object's color, the ambient light, the diffuse light, the light position, the vertex position, and the normal vector. The light vector is the vector from the vertex to the light source. We have the formulas for calculating the ambient contribution and the diffuse contribution. Add them together, convert to a `vec4`, and assign to `fragColor`.

Keep in mind that any value that is passed from the vertex shader to the fragment shader is interpolated across the primitive. For points, this is no problem; it is exactly what we want. For vectors, there is a minor problem: the interpolated vectors are no longer unit vectors. Thus, we need to use the `normalize()` function to normalize them.

One further (minor) issue is that the light's position is a 3-dimensional vector while the vertex's position is a 4-dimensional vector, with the 4th coordinate the homogeneous coordinate w. When we subtract it from the light position, we want to use only the x, y, and z coordinates. The shader language provides a mechanism for this, called the *swizzle* operator. Any `vec4` object may be appended with `.` followed by up to four of the letters `x`, `y`, `z`, `w`, including repetitions (e.g., `.xzx`). The operator will create a new vector using only the specified coordinates in the specified order. The same can be done with colors using the letters `r`, `g`, `b`, and `a`, and with textures using `s`, `t`, `p`, `q`.

Due Date

This assignment will not be collected, but you should finish it by Monday, October 21. The next assignment will implement the specular lighting, which will be collected.