good source: https://www.youtube.com/watch?v=LKXAIuCaKAQ

Exercise 4

Computer Graphics - COMP.CE.430

Lighting and Shading

In this week's exercise, we will start from where we left off last week and add lighting and shading to our 3D model. If you have a 3D model on the screen with the normal vectors available in the fragment shader, you can follow along with this exercise. If you lost your code or didn't complete last week's mandatory parts, you can start from the 3D template (the default view when you open the desktop version, or the *3D GLSL* template in SHADERed Lite) in SHADERed and modify it to output the normal vectors to the fragment shader.

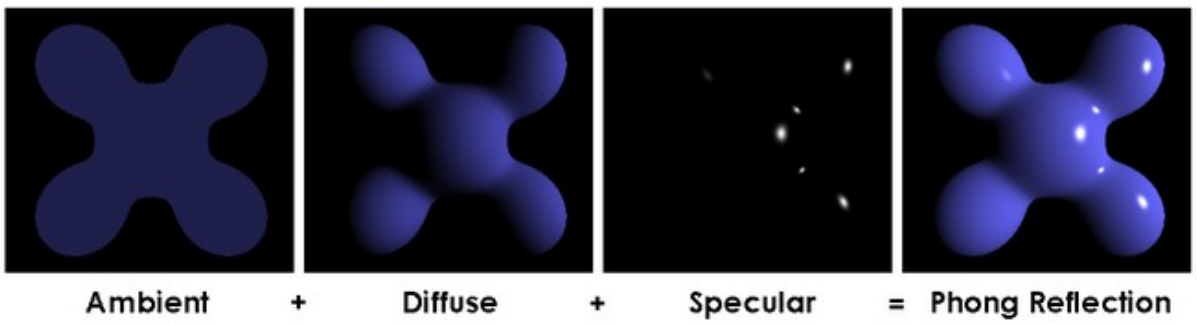
The specific shading model we'll implement this week is called the Phong reflection model. Before we implement shading, however, we will need some way of modeling lights.
The two simplest forms of lights used in computer graphics are directional lights and ambient light.

Directional lights are defined by a direction. They are treated as light sources that are infinitely far away and therefore all of the light rays propagating from that light source are parallel. This is a reasonable approximation for sunlight.

Ambient light is a constant term added to all surfaces so that surfaces that don't receive direct light don't appear completely dark. This is a very crude approximation for indirect lighting, where light reaches the surface after bouncing off of other surfaces.

The Phong Lighting Model

The most commonly used local lighting model used by older game rendering engines is the *Phong* reflection model. It models light reflected from a surface as the sum of three distinct terms; a *diffuse* term, a *specular* term, and an *ambient* term.



The following explanation is somewhat math-heavy. Here is a rundown of all the mathematical notation used in this chapter:

I	The color and intensity of the light leaving the surface in the direction of the virtual camera. Represented by an RGB vector.
A	The color and intensity of ambient lighting in the scene. Represented by an RGB vector.
C_L	The color and intensity of direct light hitting the surface. Represented by an RGB vector.
k_A	The ambient reflectivity of the surface. Represented by an RGB vector.
k_D	The diffuse reflectivity of the surface. Represented by an RGB vector.
k_S	The specular reflectivity of the surface. Represented by an RGB vector.
N	The unit-length vector that is normal to the surface.
L	The unit-length vector pointing from the surface toward the light source.
R	The unit-length vector representing the reflection of the incident light direction about the surface normal.
V	The unit-length vector pointing from the surface toward the virtual camera.
$a \otimes b$	Component-wise multiplication of vectors *a* and *b*.
$a \cdot b$	The dot product between vectors *a* and *b*. Recall the two definitions of the dot product: Algebraic definition: $a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$ Geometric definition: $a \cdot b = \|a\|\|b\|cos\theta$, where θ is the angle between vectors *a* and *b*.

The *ambient* term models the overall lighting level of the scene. It is a gross approximation of the amount of indirect light that reaches the shading point by bouncing off of other surfaces. The equation for the ambient term is

$I_{ambient} = k_A \otimes A$,

where k_A is an RGB vector representing the ambient reflectivity of the surface, A is an RGB vector representing the color and intensity of the ambient lighting in the scene and the $\otimes$ symbol denotes component-wise multiplication.

The *diffuse* term accounts for light that is reflected uniformly in all directions. This is a good approximation of how matte surfaces behave in the real world, such as a block of wood or a block of white chalk. The brightness of the diffuse term depends only on the intensity and

color of the light and the cosine of the angle between the light direction and the surface normal. The cosine of the angle between two unit vectors can be computed by using the dot product: $cos\theta = N \cdot L$, where *N* is the surface normal and *L* is the normalized direction vector from the point on the surface towards the light source. To avoid adding contribution from lights that reach the shading point from *below* the surface, we clamp this cosine factor to positive values. The diffuse term is

$$I_{diffuse} = k_D (N \cdot L) \otimes C_L,$$

where k_D is an RGB vector representing the diffuse reflectivity of the surface, C_L is an RGB vector representing the light's color and intensity and the $\otimes$ symbol denotes component-wise multiplication.

The *specular* term models the bright highlights we sometimes see when viewing a glossy surface. Plastics and metals are common examples. The brightness of the specular highlight depends both on the light direction and the view direction. The *Phong* reflection model models this by taking the clamped dot product between the view vector *V* and a reflection vector *R* and raising the result to a power. The view vector *V* is the normalized direction vector pointing from the shading point towards the virtual camera. The reflection vector *R* is the reflection of the light ray's direction vector *L* about the surface normal *N*. Note that in this case the vector *L* points from the light source towards the shading point. The complete specular term is

$$I_{specular} = k_S(R \cdot V)^{\alpha} \otimes C_L,$$

where α denotes the specular exponent, often called *shininess* or *glossiness*.

The combined equation representing the intensity *I* of light reflected from a point towards the camera for a single light source is

$$I = (k_A \otimes A) + [k_D(N \cdot L) + k_S(R \cdot V)^{\alpha}] \otimes C_L.$$

Problem set

For the following exercises you'll probably want to declare some uniform variables. You can define uniform variables in SHADERed by right-clicking the *Shader Pass* and selecting *Variables*. You can also set uniforms to track other variables, such as an object's position, by going to the edit window (the pen icon on the left side of a variable), then clicking the square root icon on the top right and selecting *ObjectProperty* as the function. You may find this useful for experimenting with light sources.

1. Implement the *ambient* term of the *Phong* reflection model.

You will need to define a *vec3* variable representing the color and intensity of the ambient lighting in the scene. You can either use a *uniform* or a hard-coded global variable in the fragment shader. You'll also need to define a *vec3* variable representing the ambient reflectivity of the surface. Fairly low values are typically used for the ambient light's color and intensity (eg. 0.1). After you've done this, you can compute the ambient term by multiplying the ambient lighting with the ambient reflectivity. Output the result as the fragment's color. **(1p)**

2. Define a directional light and implement the *diffuse* term of the *Phong* reflection model.

   A directional light is defined by a normalized direction vector pointing toward the light source and an RGB vector representing the light's color and intensity. Define these variables (again, you can either use uniforms or hardcode these in the shader). You can use a value of (1.0, 1.0, 1.0) for the intensity and any normalized direction for the direction. You will also need a variable representing the diffuse reflectivity of the surface.

   Now you can compute the cosine term by taking the dot product between the light direction vector and the normal vector. GLSL has a built-in function for this: dot. Remember that the cosine term should be clamped to only include positive values. You can use either max or clamp for this. Multiply the cosine term with the diffuse reflectivity of the surface and the directional light's intensity to compute the diffuse term. Output the sum of the diffuse term and the ambient term from step 1 from the fragment shader. **(1p)**

   Note: In general, the normal vector passed to the fragment shader will no longer be normalized after interpolation (the graphics hardware interpolates the values passed from the vertex shader to the fragment shader). You should always renormalize the normal vector in the fragment shader before using it in shading equations.

3. Implement the *specular* term of the Phong reflection model.

   You'll need two extra variables for this: A *vec3* representing the specular reflectivity and a *float* representing the "shininess" value α. The specular reflectivity for a typical non-metallic surface is usually white. For a metallic surface, it would be the color of the metal.

   To compute the specular term, you'll need to compute the reflection vector. GLSL has a built-in function for this that you can use: reflect, which takes an incident vector and a normal vector as input. We defined the light direction vector to point *towards* the light source, so the reflection vector can be computed as

```
vec3 r = reflect(-light_dir, normal);
```

   The view vector can be computed from the position vector of the shading point and the camera position. SHADERed has a built-in uniform that you can use to retrieve

the camera position. To use the built-in uniform, select *vec3* as the *Type* and *CameraPosition3* from the *System* drop-down menu in the *Variable Manager* window.

For the position vector, you'll need to output this from the vertex shader in a similar manner as you did for the normal vector in last week's exercise. The view vector can then be computed as

```
vec3 view = normalize(camera_pos - fragment_pos);
```

Now you can take the dot product between the reflection vector and the view vector. Clamp the resulting value to positive values and raise it to the power α that represents the "shininess" value. You can use the pow function for this. Multiply the result with the specular reflectivity of the surface and the color and intensity of the directional light. Output the sum of the ambient term from step 1, the diffuse term from step 2, and the specular term from step 3. This is the complete *Phong* reflection model. **(1p)**

Bonus

1. Use the *Blinn-Phong* reflection model instead of the *Phong* reflection model.

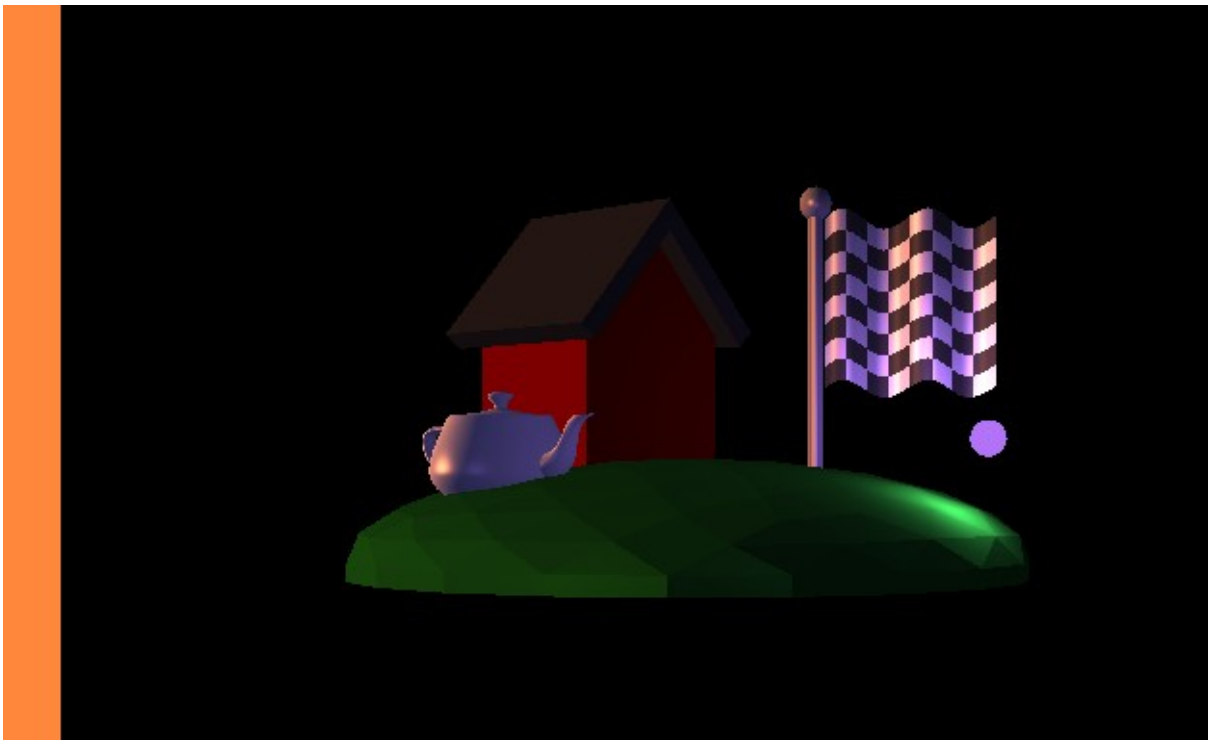
   The only difference between the two is that the specular term of the *Blinn-Phong* reflection model is formulated in terms of the dot product between the surface normal and a *half vector* as opposed to the dot product between the view vector and the reflection vector. The half vector is a vector that is halfway between the light direction vector and the view vector. It can be computed as $H = \frac{L+V}{|L+V|}$. Replace $R \cdot V$ with $H \cdot N$ in the computation of the specular term.

   Note that the impact of the specular exponent will change. Try comparing the specular highlights between both reflection models using different values for the "shininess" exponent. **(1p)**

2. Add a point light to the scene.

   A point light is defined by a position vector that defines its point in space and the light's color and intensity, so add the appropriate variables. Unlike directional lights, the light direction vector for a point light varies for each fragment, so you will need to compute it in the fragment shader. For point lights to be physically accurate, you'll also need to scale the light intensity by the inverse of the squared distance. You can compute the distance from the fragment position to the point light by using the built-in distance function. The intensity falls off very quickly as the distance increases, so you might need to set a higher intensity for the point light in comparison to the directional light to see its effects properly. You can combine the contribution from multiple light sources by summing the diffuse and specular terms from all of the light sources. **(1p)**

Example scene



This scene is an example of all the features implemented in this exercise. Some meshes were added to aid with visualizing the lights. The big orange box on the left represents the directional light and the sphere on the right represents the point light.