Phong Illumination Shader Implementation
Phong Illumination Equation

\[ I_r = \sum_{i} R_{ira} L_{ira} + R_{ird} L_{ird} + R_{irs} L_{irs} \]

- Total red intensity
- Intensity of red ambient light (or how much red is coming in)
- Reflectivity of red ambient material (or how much red is going back out)
For each light source and each color component, the Phong model can be written (without the distance terms) as

\[ I = k_d L_d \mathbf{l} \cdot \mathbf{n} + k_s L_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a L_a \]

For each color component we add contributions from all sources
OpenGL shading

State-based shading functions have been deprecated (glNormal, glMaterial, glLight)

Need to define and send to shaders

- Normals
- material properties
- Lights
Normalization

Cosine terms in lighting calculations can be computed using dot product

Unit length vectors are necessary when using dot product replacement

GLSL has a normalization function: `normalize`
GLSL also has a reflection function: `reflect(I,N)`
For each light source, we can set an RGBA for the diffuse, specular, and ambient components, and specify the position.

Do this on the application side:

```cpp
vec4 diffuse0 = vec4(1.0, 0.0, 0.0, 1.0);
vec4 ambient0 = vec4(1.0, 0.0, 0.0, 1.0);
vec4 specular0 = vec4(1.0, 0.0, 0.0, 1.0);
vec4 light0_pos = vec4(1.0, 2.0, 3.0, 1.0);
```
Distance and Direction

The source colors are specified in RGBA
The position is given in homogeneous coordinates

If \( w = 1.0 \), we are specifying a finite location

If \( w = 0.0 \), we are specifying a directional light source with the given direction vector
Material Properties

Material properties represent the coefficients in the Phong Illumination model (e.g. the k’s)

```cpp
vec4 ambient = vec4(0.2, 0.2, 0.2, 1.0);
vec4 diffuse = vec4(1.0, 0.8, 0.0, 1.0);
vec4 specular = vec4(1.0, 1.0, 1.0, 1.0);
gLfloat shine = 100.0
```
Phong Illumination Model

L’s and k’s directly from the application for efficiency, precompute the product

e.g. diffuseProduct = lightDiffuse * materialDiffuse;

\[
I = k_d \cdot L_d \cdot n + k_s \cdot L_s \cdot (v \cdot r) + k_a \cdot L_a
\]
// vertex shader
in vec4 vPosition;
in vec3 vNormal;
out vec4 color; // vertex shader

// light and material properties
uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform mat4 Projection;
uniform vec4 LightPosition;
uniform float Shininess;
void main()
{
    // Transform vertex position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;

    vec3 L = normalize( (ModelView * LightPosition).xyz - pos );
    vec3 E = normalize( -pos ); //vector from pos to eye!
    vec3 H = normalize( L + E ); //halfway vector

    // Transform vertex normal into eye coordinates
    vec3 N = normalize( transpose(inverse(ModelView)) *vNormal ).xyz;
Vertex Lighting Shaders

// Compute terms in the illumination equation
vec4 ambient = AmbientProduct;  // ka*la

float dr = max( dot(L, N), 0.0 );
vec4 diffuse = dr*DiffuseProduct;

float sr = pow( max(dot(N, H), 0.0), Shininess );
vec4 specular = sr * SpecularProduct;
if( dot(L, N) < 0.0 )  //if light is behind surface!
    specular = vec4(0.0, 0.0, 0.0, 1.0);

gl_Position = Projection * ModelView * vPosition;

color = ambient + diffuse + specular;
color.a = 1.0;
}
Shading

Now that we have colors at vertices, using the phong illumination model, how are colors computed internally for each triangle = SHADING
// fragment shader

in vec4 color; //color will be interpolated over // the triangle
out vec4 fcolor;

void main()
{
    fColor = color;
}