Modified Phong Model

\[
I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a I_a \\
I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^\beta + k_a I_a
\]

- Blinn proposed using \textbf{halfway vector}, more efficient
- \textbf{h} is normalized vector halfway between \textbf{l} and \textbf{v}

\[h = (\mathbf{l} + \mathbf{v})/|\mathbf{l} + \mathbf{v}|\]
Example

Modified Phong model gives similar results as original Phong
Computer Graphics (CS 4731)
Lecture 17: Lighting, Shading and Materials (Part 2)

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)
Computation of Vectors

- To calculate lighting at vertex P
  Need $\mathbf{l}$, $\mathbf{n}$, $\mathbf{r}$ and $\mathbf{v}$ vectors at vertex P

- User specifies:
  - Light position
  - Viewer (camera) position
  - Vertex (mesh position)
- $\mathbf{l}$: Light position – vertex position
- $\mathbf{v}$: Viewer position – vertex position
- Normalize all vectors!
Specifying a Point Light Source

- For each light source component, set RGBA and position
- alpha = transparency

```
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

vec4 light0_pos = vec4(1.0, 2.0, 3.0, 1.0);

- Position is in homogeneous coordinates
Recall: Mirror Direction Vector $r$

- Can compute $r$ from $l$ and $n$
- $l$, $n$ and $r$ are co-planar
- What about determining vertex normal $n$?

$$r = 2 \left( l \cdot n \right) n - l$$
Finding Normal, $n$

- Normal calculation in application, passed to vertex shader
Recall: Newell Method for Normal Vectors

- Formulae: Normal \( N = (mx, my, mz) \)

\[
m_x = \sum_{i=0}^{N-1} \left( y_i - y_{\text{next}(i)} \right) \left( z_i + z_{\text{next}(i)} \right) \\
m_y = \sum_{i=0}^{N-1} \left( z_i - z_{\text{next}(i)} \right) \left( x_i + x_{\text{next}(i)} \right) \\
m_z = \sum_{i=0}^{N-1} \left( x_i - x_{\text{next}(i)} \right) \left( y_i + y_{\text{next}(i)} \right)
\]
OpenGL shading

- **Need**
  - Normals
  - Material properties
  - Lights

- **State-based shading functions now deprecated**
  - (glNormal, glMaterial, glLight) deprecated
Material Properties

- Need to specify material properties of scene objects
- Material properties also has ambient, diffuse, specular
- Material properties specified as RGBA + reflectivities
- w component gives opacity (transparency)
- **Default?** all surfaces are opaque

```c
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
```
Recall: CTM Matrix passed into Shader

- Recall: CTM matrix concatenated in application
  
  \[ \text{mat4 ctm} = \text{ctm } \ast \text{LookAt}(\text{vec4 eye}, \text{vec4 at}, \text{vec4 up}); \]

- CTM matrix passed in contains object transform + Camera

- Connected to matrix \textbf{ModelView} in shader

```cpp
in vec4 vPosition;
Uniform mat4 ModelView;

main( )
{
    // Transform vertex position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;
    ............
}
```
Computation of Vectors

- CTM transforms vertex position into eye coordinates
  - Eye coordinates? Object, light distances measured from eye
- Normalize all vectors! (magnitude = 1)
- GLSL has a `normalize` function
- **Note:** vector lengths affected by scaling

```cpp
// Transform vertex position into eye coordinates
vec3 pos = (ModelView * vPosition).xyz;

vec3 L = normalize( LightPosition.xyz - pos ); // light vector
vec3 E = normalize( -pos );                      // view vector
vec3 H = normalize( L + E );                     // Halfway vector
```
Spotlights

- Derive from point source
  - **Direction I** (of lobe center)
  - **Cutoff**: No light outside $\theta$
  - **Attenuation**: Proportional to $\cos^\alpha \phi$
Recall: Lighting Calculated Per Vertex

- Phong model (ambient+diffuse+specular) calculated at each vertex to determine vertex color
- Per vertex calculation? Usually done in vertex shader
Per-Vertex Lighting Shaders I

// 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;

Ambient, diffuse, specular (light * reflectivity) specified by user

kd Id ks Is

 exponent of specular term

k_a I_a  k_d I_d  k_s I_s
void main( )
{
  // Transform vertex position into eye coordinates
  vec3 pos = (ModelView * vPosition).xyz;

  vec3 L = normalize( LightPosition.xyz - pos );
  vec3 E = normalize( -pos );
  vec3 H = normalize( L + E ); // halfway Vector

  // Transform vertex normal into eye coordinates
  vec3 N = normalize( ModelView*vec4(vNormal, 0.0 ) ).xyz;
Per-Vertex Lighting Shaders III

// Compute terms in the illumination equation
vec4 ambient = AmbientProduct;  \( k_a I_a \)

float cos_theta = max( dot(L, N), 0.0 );
vec4 diffuse = cos_theta * DiffuseProduct;  \( k_d I_d l \cdot n \)

float cos_phi = pow( max(dot(N, H), 0.0), Shininess );
vec4 specular = cos_phi * SpecularProduct;  \( k_s I_s (n \cdot h)^\beta \)

if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);
gl_Position = Projection * ModelView * vPosition;

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

\[
I = k_d I_d l \cdot n + k_s I_s (n \cdot h)^\beta + k_a I_a
\]
Per-Vertex Lighting Shaders IV

// in vertex shader, we declared color as out, set it

......

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

// in fragment shader (in vec4 color;

void main()
{
    gl_FragColor = color;
}

color set in vertex shader

Graphics Hardware

color used in fragment shader
References