Using Shaders for Lighting

Lighting Definitions

- **N** = Normal vector
- **L** = Vector from Point to the Light
- **R** = Light reflection vector
- **E** = Vector from the Point to the eye

Ambient = Light intensity that is "everywhere"

Diffuse = Light intensity proportional to \( \cos(\theta) \)

Specular = Light intensity proportional to \( \cos^2(\phi) \)

A-D-S = Lighting model that includes Ambient, Diffuse, and Specular

Flat Interpolation = Use a single polygon normal to compute one A-D-S for the entire polygon

Smooth Interpolation = Use a normal at each vertex to compute one A-D-S at each vertex

Per-fragment lighting = Interpolate the vectors across the entire polygon and then compute A-D-S at each fragment

A-D-S Lighting

- Ambient: \( K_a \)
- Diffuse: \( K_d \cdot \cos(\theta) \)
- Specular: \( K_s \cdot \cos^2(\phi) \)
The Difference Between Per-Vertex Lighting and Per-Fragment Lighting

Per-vertex

Per-fragment

Applying Per-Fragment Lighting

Vertex shader:

```glsl
#version 330 compatibility
out vec2 vST; // texture coords
out vec3 vN; // normal vector
out vec3 vL; // vector from point to light
out vec3 vE; // vector from point to eye
const vec3 LIGHTPOSITION = vec3( 5., 5., 0. );

void main( )
{
    vST = gl_MultiTexCoord0.st;
    vec4 ECposition = gl_ModelViewMatrix * gl_Vertex; // eye coordinate position
    vN = normalize( gl_NormalMatrix * gl_Normal ); // normal vector
    vL = LIGHTPOSITION - ECposition.xyz; // vector from the point to the light position
    vE = vec3(0., 0., 0.) - ECposition.xyz; // vector from the point to the eye position
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Fragment shader:

```glsl
#version 330 compatibility
uniform float uKa, uKd, uKs; // coefficients of each type of lighting
uniform float uShininess; // specular exponent
in vec2 vST; // texture cords
in vec3 vN; // normal vector
in vec3 vL; // vector from point to light
in vec3 vE; // vector from point to eye

void main( )
{
    vec3 Normal = normalize(vN);
    vec3 Light = normalize(vL);
    vec3 Eye = normalize(vE);
    vec3 myColor = vec3( 1.0, 0.5, 0.0 ); // default color
    vec3 mySpecularColor = vec3( 1.0, 1.0, 1.0 ); // specular highlight color
    vec3 ambient = uKa * myColor;
    float d = 0.;
    if( dot(Normal,Light) > 0. ) // only do specular if the light can see the point
    {
        vec3 refl = normalize( reflect( -Light, Normal ) );
        float s = pow( max( dot(Eye,refl),0. ), uShininess );
        vec3 diffuse = uKd * d * myColor;
        vec3 specular = uKs * s * mySpecularColor;
        gl_FragColor = vec4( ambient + diffuse + specular, 1. );
    }
}
```
Each polygon has a single lighting value applied to every pixel within it.
What you see depends on the light color and the material color:

\[
\begin{align*}
E_R &= L_R \cdot M_R \\
E_G &= L_G \cdot M_G \\
E_B &= L_B \cdot M_B
\end{align*}
\]

This is how you implement subtractive coloring.

Summary:

- Flat
- Smooth
- Anisotropic

A-D-S Anisotropic Lighting with Normal Interpolation:

Note: The bright spot is not circular because the material has different properties in different directions. Materials such as fur, hair, and brushed metal behave this way.


Besides Hair, Other Real World Examples of Anisotropic Lighting Behavior:

- Information Desk
- Elevator Doors