**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 \(\cos(\Phi)\)

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**: \(K_d \cos(\theta)\)
- **Specular**: \(K_s \cos(\Phi)\)

**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

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.;
float s = 0.;
if( dot(Normal,Light) > 0. ) // only do specular if the light can see the point
    d = dot(Normal,Light);
vec3 ref = normalize( reflect( -Light, Normal ) ); // reflection vector
s = pow( max( dot(Eye,ref), 0. ), uShininess);
vec3 diffuse = uKd * d * myColor;
vec3 specular = uKs * s * mySpecularColor;

vec3 color = ambient + diffuse + specular;

gl_FragColor = vec4( color, 1. );
```

Here's where we apply lighting to that color.

Per-fragment A-D-S Lighting with Flat Interpolation

Vertex shader:

```glsl
flat in vec2 vST; // texture cords
flat in vec3 vN; // normal vector
flat in vec3 vL; // vector from point to light
flat in vec3 vE; // vector from point to eye

flat out vec2 vST; // texture coords
flat out vec3 vN; // normal vector
flat out vec3 vL; // vector from point to light
flat out vec3 vE; // vector from point to eye
```

Fragment shader:

```glsl
flat in vec2 vST; // texture cords
flat in vec3 vN; // normal vector
flat in vec3 vL; // vector from point to light
flat in vec3 vE; // vector from point to eye

flat in vec2 vST; // texture cords
flat in vec3 vN; // normal vector
flat in vec3 vL; // vector from point to light
flat in vec3 vE; // vector from point to eye
```

Each polygon has a single lighting value applied to every pixel within it.

Flat Shading

Smooth Shading
What the eye sees

\[ E_R = L_R \times M \]

\[ E_G = L_G \times M \]

\[ E_B = L_B \times M \]

This is how you implement subtractive coloring.

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.


Summary

Flat Smooth Anisotropic

Besides Hair, Other Real World Examples of Anisotropic Lighting Behavior

Information Desk Elevator Doors