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: $K_s \cdot \cos^s \phi$

Ambient-only
Diffuse-only
Specular-only

ADS – Shininess=50
ADS – Shininess=1000
ADS – Shininess=1000 -- Flat
The Difference Between Per-Vertex Lighting and Per-Fragment Lighting

Per-vertex

Per-fragment

The Difference Between Per-Vertex Lighting and Per-Fragment Lighting

Per-vertex

Per-fragment
Per-fragment A-D-S Lighting

Visualization of smooth rasterization of N, L, E

Computer Graphics

#version 330 compatibility

uniform vec3 uLightPosition;

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

void main( )
{
    vST = gl_MultiTexCoord0.st;

    vec4 ECposition = gl_ModelViewMatrix * gl_Vertex; // eye coordinate position
    vN = normalize( gl_NormalMatrix * gl_Normal ); // normal vector
    vL = uLightPosition - ECposition.xyz; // vector from the point to the light position
    vE = vec3( 0., 0., 0. ) - ECposition.xyz; // vector from the point to eye position

    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}

Vertex shader:
Applying Per-Fragment Lighting

Rasterizer
**Applying Per-Fragment Lighting**

Fragment shader:

```glsl
#version 330 compatibility
uniform vec3  uColor;
uniform vec3  uSpecularColor;
uniform float   uKa, uKd, uKs; // coefficients of each type of lighting
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 = uColor; // default color
    vec3 ambient = uKa * myColor;
    float d = 0., s = 0.
    if (dot(Normal,Light) > 0.) // only do specular if the light can see the point
    {
        d = dot(Normal,Light);
        vec3 R = normalize( reflect( -Light, Normal ) ); // reflection vector
        s = pow( max( dot(Eye,R), 0. ), uShininess );
    }
    vec3 diffuse = uKd * d * myColor;
    vec3 specular = uKs * s * uSpecularColor;
    gl_FragColor = vec4( ambient + diffuse + specular, 1.);
}
```

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

Each polygon has a single lighting value applied to every pixel within it.
Per-fragment A-D-S Lighting with Flat Interpolation

Vertex shader:

```glsl
flat out vec2 vST; // texture cords
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 Shading

Smooth Shading
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.

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