Using Shaders for Lighting

Lighting Definitions

- **N** = Normal
- **L** = Light vector
- **E** = Eye vector
- **R** = Light reflection vector
- **ER** = Eye reflection vector
- **Color** = LightColor * MaterialColor

**Ambient** = Light intensity that is “everywhere”

**Diffuse** = Light intensity proportional to $\cos(\Theta)$

**Specular** = Light intensity proportional to $\cos^S(\Phi)$

**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 for each vertex

**Per-vertex lighting** = Compute A-D-S using each vertex normal and then interpolate the summed intensity over the entire polygon

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

**CubeMap Reflection** = Using the Eye Reflection Vector (ER) to look-up the reflection of a “wall texture”

A-D-S Lighting

- Ambient: $K_a$
- Diffuse: $K_d \cdot \cos(\Theta)$
- Specular: $K_s \cdot \cos^S(\Phi)$

A-D-S Lighting with Flat Interpolation

$$\text{gl}_\text{FragColor}.\text{rgb} = K_a \cdot \text{ambient} + K_d \cdot \text{diffuse} + K_s \cdot \text{spec};$$

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

This is how you implement subtractive coloring.

Ambient-only

Diffuse-only

Specular-only

**Complements**

- **ADS – Shininess=50**
- **ADS – Shininess=1000**
- **ADS – Shininess=1000 – Flat**

White Light

Green Light

Write Light

Find light's color

Write light's color

L_i = $L_i \cdot M_i$

L_f = $L_f \cdot M_f$

This is how you implement subtractive coloring.
A-D-S Lighting with Smooth Interpolation

Note: In per-vertex lighting, the light intensity is computed at each vertex and interpolated throughout the polygon. This creates artifacts such as Mach Banding and the fact that the bright spot is "jagged". You can do this in stock OpenGL or in a shader.

N = Normal
L = Light vector
E = Eye vector
R = Light reflection vector
ER = Eye reflection vector
Color = LightColor * MaterialColor

\[ \text{gl}_\text{FragColor.rgb} = \text{Ka*ambient + Kd*diffuse + Ks*spec}; \]

Vertex Shader

\[ \text{vec3 ambient = Color.rgb; \]
\[ \text{diffuse} = \max(\, \text{dot}(\text{L},\text{N}), 0.\, ) * \text{Color.rgb}; \]
\[ \text{vec3 R = normalize(\, \text{reflect(\, -\text{L}, \text{N})\, }); \]
\[ \text{vec3 spec} = \text{LightColor} * \text{pow(\, \max(\, \text{dot}(\, \text{R}, \text{E}\, ), 0.\, ), \text{Shininess}\, )); \]

Fragment Shader

Smooth-rasterize ambient, diffuse, spec

In per-fragment lighting, the normal is interpolated throughout the polygon. The light intensity is computed at each fragment. This avoids Mach Banding and makes the bright spot smooth. You can only do this in a shader.

N = Normal
L = Light vector
E = Eye vector
R = Light reflection vector
ER = Eye reflection vector
Color = LightColor * MaterialColor

\[ \text{gl}_\text{FragColor.rgb} = \text{Ka*ambient + Kd*diffuse + Ks*spec}; \]

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

Per-vertex

Per-fragment

A-D-S Lighting with Normal Interpolation and a CubeMap Reflection

Note: A cube map reflection is blended in, giving a stronger impression that the surface is shiny.

N = Normal
L = Light vector
E = Eye vector
R = Light reflection vector
ER = Eye reflection vector
Color = LightColor * MaterialColor

\[ \text{vec3 reflcolor} = \text{textureCube( ReflectUnit, \text{R} \, \text{.rgb}); \]
\[ \text{gl}_\text{FragColor.rgb} = \text{Ka*ambient + Kd*diffuse + Ks*spec + Kr*reflcolor.rgb}; \]

Fragment Shader

Smooth-rasterize N, L, E

Flat shading

Normal interpolation

A-D-S Lighting with Normal Interpolation

In per-fragment lighting, the normal is interpolated throughout the polygon. The light intensity is computed at each fragment. This avoids Mach Banding and makes the bright spot smooth. You can only do this in a shader.

N = Normal
L = Light vector
E = Eye vector
R = Light reflection vector
ER = Eye reflection vector
Color = LightColor * MaterialColor

\[ \text{gl}_\text{FragColor.rgb} = \text{Ka*ambient + Kd*diffuse + Ks*spec}; \]

Vertex Shader

\[ \text{vec3 ambient = Color.rgb; \]
\[ \text{diffuse} = \max(\, \text{dot}(\text{L},\text{N}), 0.\, ) * \text{Color.rgb}; \]
\[ \text{vec3 R = normalize(\, \text{reflect(\, -\text{L}, \text{N})\, }); \]
\[ \text{vec3 spec} = \text{LightColor} * \text{pow(\, \max(\, \text{dot}(\, \text{R}, \text{E}\, ), 0.\, ), \text{Shininess}\, )); \]

Fragment Shader

Smooth-rasterize N, L, E

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

Per-vertex

Per-fragment

A-D-S Lighting with Normal Interpolation and a CubeMap Reflection

Note: A cube map reflection is blended in, giving a stronger impression that the surface is shiny.

N = Normal
L = Light vector
E = Eye vector
R = Light reflection vector
ER = Eye reflection vector
Color = LightColor * MaterialColor

\[ \text{vec3 reflcolor} = \text{textureCube( ReflectUnit, \text{R} \, \text{.rgb}); \]
\[ \text{gl}_\text{FragColor.rgb} = \text{Ka*ambient + Kd*diffuse + Ks*spec + Kr*reflcolor.rgb}; \]
vec3 ambient = Color.rgb;
float dl = dot( T, L );
vec3 diffuse = sqrt( 1. - dl*dl ) * Color.rgb;
float de = dot( T, E );
vec3 spec = LightColor * pow(  dl * de + sqrt( 1. - dl*dl ) * sqrt( 1. - de*de ), Shininess );

gl_FragColor.rgb = Ka*ambient + Kd*diffuse + Ks*spec;