Using Shaders for Lighting

Lighting Definitions

\[ N = \text{Normal} \]
\[ L = \text{Light vector} \]
\[ E = \text{Eye vector} \]
\[ R = \text{Light reflection vector} \]
\[ \text{ER} = \text{Eye reflection vector} \]

\[ \text{Color} = \text{LightColor} \times \text{MaterialColor} \]

Ambient = Light intensity that is “everywhere”
Diffuse = Light intensity proportional to \( \cos(\theta) \)
Specular = Light intensity proportional to \( \cos^\alpha(\phi) \)

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: \( K_d \times \cos(\theta) \)
Specular: \( K_s \times \cos^\alpha(\phi) \)
A-D-S Lighting with Flat Interpolation

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

\[
\begin{align*}
N &= \text{Normal} \\
L &= \text{Light vector} \\
E &= \text{Eye vector} \\
R &= \text{Light reflection vector} \\
ER &= \text{Eye reflection vector} \\
\text{Color} &= \text{LightColor} \times \text{MaterialColor}
\end{align*}
\]

Vertex Shader

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

\[
\begin{align*}
\text{ambient} &= \text{Color.rgb} \\
\text{diffuse} &= \max(\text{dot}(L, N), 0) \times \text{Color.rgb} \\
\text{spec} &= \text{LightColor} \times \text{pow}(\max(\text{dot}(R, E), 0), \text{Shininess});
\end{align*}
\]

Fragment Shader

A-D-S Lighting with Smooth Interpolation

Note: 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 not jagged.

You can do this in stock OpenGL or in a shader.

\[
\begin{align*}
N &= \text{Normal} \\
L &= \text{Light vector} \\
E &= \text{Eye vector} \\
R &= \text{Light reflection vector} \\
ER &= \text{Eye reflection vector} \\
\text{Color} &= \text{LightColor} \times \text{MaterialColor}
\end{align*}
\]

Vertex Shader

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

\[
\begin{align*}
\text{ambient} &= \text{Color.rgb} \\
\text{diffuse} &= \max(\text{dot}(L, N), 0) \times \text{Color.rgb} \\
\text{spec} &= \text{LightColor} \times \text{pow}(\max(\text{dot}(R, E), 0), \text{Shininess});
\end{align*}
\]

Fragment Shader

A-D-S Lighting with Normal Interpolation

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.

\[
\begin{align*}
N &= \text{Normal} \\
L &= \text{Light vector} \\
E &= \text{Eye vector} \\
R &= \text{Light reflection vector} \\
ER &= \text{Eye reflection vector} \\
\text{Color} &= \text{LightColor} \times \text{MaterialColor}
\end{align*}
\]

Vertex Shader

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

\[
\begin{align*}
\text{ambient} &= \text{Color.rgb} \\
\text{diffuse} &= \max(\text{dot}(L, N), 0) \times \text{Color.rgb} \\
\text{spec} &= \text{LightColor} \times \text{pow}(\max(\text{dot}(R, E), 0), \text{Shininess});
\end{align*}
\]

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

Per-vertex: 

Per-fragment:

Flat shading: 

Normal interpolation:

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

Note: A cube map reflection is blended in, given 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 

(vec3 ambient = Color.rgb; 
diffuse = max(dot(L,N), 0.) * Color.rgb; 
vec3 R = normalize(reflect(-L, N)); 
vec3 spec = LightColor * pow(max(dot(R, E), 0.), Shininess); 
vec3 reflcolor = textureCube(ReflectUnit, R).rgb; 
gl_FragColor.rgb = Ka*ambient + Kd*diffuse + Ks*spec + Kr*reflcolor.rgb;)

Smooth rasterize N, L, E
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 tannish metal behave this way.


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

vec3 ambient = Color.rgb;
float dl = dot( N, L );
vec3 diffuse = sqrt( 1. - dl*dl ) * Color.rgb;
float de = dot( N, 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;

Summary

Flat
Smooth
Anisotropic

Fragment Shader

#version 330 compatibility
uniform float uLightX, uLightY, uLightZ; flat out vec3 vNf;
out vec3 vNs;
flat out vec3 vLf;
out vec3 vLs;
flat out vec3 vEf;
out vec3 vEs;
vec3 eyeLightPosition = vec3( uLightX, uLightY, uLightZ );
void main( ){ vec4 ECposition = gl_ModelViewMatrix * gl_Vertex;
Nf = normalize( gl_NormalMatrix * gl_Normal ); // surface normal vector
Ns = Nf;
Lf = eyeLightPosition - ECposition.xyz; // vector from the point
Ls = Lf; // to the light position
Ef = vec3( 0., 0., 0. ) - ECposition.xyz; // vector from the point
Es = Ef; // to the eye position
N = normalize( gl_ModelViewProjectionMatrix * gl_Vertex); }
vec4 ambient = uKa * uColor;
float d = max(dot(Normal,Light), 0.);
vec4 diffuse = uKd * d * uColor;
float s = 0.;
if( dot(Normal,Light) > 0. ) // only do specular if the light can see the point
{
    vec3 ref = normalize(2. * Normal * dot(Normal,Light) - Light);
    s = pow(max(dot(Eye,ref), 0.), uShininess);
}
vec4 specular = uKs * s * uSpecularColor;
gl_FragColor = vec4(ambient.rgb + diffuse.rgb + specular.rgb, 1.);