Using Shaders for Lighting

Lighting Definitions

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

Ambient = Light intensity that is “everywhere”
Diffuse = Light intensity proportional to cos(θ)
Specular = Light intensity proportional to cos^n(φ)
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
Per-vertex lighting = Compute A-D-S using each vertex normal and then interpolate the sum over the entire polygon
Per-fragment lighting = Interpolate the vertex normals across the entire polygon and 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

\[
\begin{align*}
\text{Ambient: } & K_a \\
\text{Diffuse: } & K_d \times \text{cos}(θ) \\
\text{Specular: } & K_s \times \text{cos}^n(φ)
\end{align*}
\]
A-D-S Lighting with Flat Interpolation

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

\[
\text{gl\_FragColor.rg} = K_a \times \text{ambient} + K_d \times \text{diffuse} + K_s \times \text{spec};
\]

Vertex Shader

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

Fragment Shader

What you see depends on the light color and the material color

What the light can produce

What the eye sees

What the material can reflect

White Light

Green Light

What the light can produce

A-D-S Lighting with Smooth Interpolation

Note: The light intensity is computed at each vertex and interpolated throughout the facet. This creates artifacts such as Mach Banding and the fact that the bright spot is not circular. You can do this in stock OpenGL or in a shader.

\[
\text{gl\_FragColor.rg} = K_a \times \text{ambient} + K_d \times \text{diffuse} + K_s \times \text{spec};
\]

Vertex Shader

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

Fragment Shader

A-D-S Lighting with Normal Interpolation

Note: The normal is interpolated throughout the facet. This avoids Mach Banding and makes the bright spot circular. You can only do this in a shader.

\[
\text{gl\_FragColor.rg} = K_a \times \text{ambient} + K_d \times \text{diffuse} + K_s \times \text{spec};
\]

Vertex Shader

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

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.
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;


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.

Fragment Shader

### Summary

<table>
<thead>
<tr>
<th>Flat</th>
<th>Smooth</th>
<th>Anisotropic</th>
</tr>
</thead>
</table>

#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 = uModelViewMatrix * aVertex;
    Nf = normalize( uNormalMatrix * aNormal ); // surface normal vector
    Ns = Nf;
    LF = eyeLightPosition - ECposition.xyz; // vector from the point to the light position
    La = LF;
    EF = vec3( 0., 0., 0. ) - ECposition.xyz; // vector from the point to the eye position
    Es = EF;
    gl_Position = uModelViewProjectionMatrix * aVertex;
}
vec4 ambient = uKa * uColor;
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;
FragColor = vec4(ambient.rgb + diffuse.rgb + specular.rgb, 1.);