Using Shaders for Lighting

Computer Graphics

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

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

\[
\text{Ambient: } K_a \\
\text{Diffuse: } K_d \cdot \cos(\theta) \\
\text{Specular: } K_s \cdot \cos^s(\phi)
\]

A-D-S Lighting with Flat Interpolation

Vertex Shader

- \( R \) = Normal
- \( L \) = Light vector
- \( E \) = Eye vector
- \( R \) = Light reflection vector
- \( ER \) = Eye reflection vector

Color = LightColor \* MaterialColor

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

Fragment Shader

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

- White Light
- Green Light

\[
\begin{align*}
E_E & = L_E \cdot M_E \\
E_R & = L_R \cdot M_R \\
E_S & = L_S \cdot M_S
\end{align*}
\]
**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.

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

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

Vertext Shader

**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.

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

\[
\text{gl_FragColor.rgb} = \text{Ka} \times \text{ambient} + \text{Kd} \times \text{diffuse} + \text{Ks} \times \text{spec} + \text{Kr} \times \text{reflcolor.rgb};
\]

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

**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, giving a stronger impression that the surface is shiny.
A-D-S Anisotropic Lighting with Normal Interpolation

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;

James Kajiya and Timothy Kay, "Rendering Fur with Three
Dimensional Textures", Proceedings of SIGGRAPH 1989,

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.

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

Summary

Flat
Smooth
Reflection
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;

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
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}

Vertex shader

#version 330 compatibility
uniform float uKa, uKd, uKs;
uniform vec4 uColor;
uniform vec4 uSpecularColor;
uniform float uShininess;
uniform bool uFlat;
flat in vec3 vNf;
in vec3 vNs;
flat in vec3 vLf;
in vec3 vLs;
flat in vec3 vEf;
in vec3 vEs;

void
main( )
{
vec3 Normal;
vec3 Light;
vec3 Eye;
if( uFlat )
{
Normal = normalize(vNf);
Light = normalize(vLf);
Eye = normalize(vEf);
}
else
{
Normal = normalize(vNs);
Light = normalize(vLs);
Eye = normalize(vEs);
}
}

Fragment shader, I

vec4 ambient = uKa * uColor;
float d = max( dot(Normal,Light), 0. );
vec4 diffuse = uKd * d * uColor;
float s = 0.0;
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. );

Fragment shader, II

#version 330 compatibility
uniform float uR0, ua, ub, ac,
uniform vec3 uSpecularColor;
uniform float uShininess,
uniform float uAlpha;
flat in vec3 vNf;
in vec3 vNs;
flat in vec3 vLf;
in vec3 vLs;
in vec3 vEf;
in vec3 vEs;

void
main( )
{
vec3 Normal;
vec3 Light;
vec3 Eye;
if( uAlpha )
{
Normal = normalize(vNf);
Light = normalize(vLf);
Eye = normalize(vEf);
}
else
{
Normal = normalize(vNs);
Light = normalize(vLs);
Eye = normalize(vEs);
}
}