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^n(\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
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

Ambient: \( K_a \)
Diffuse: \( K_d \cos \theta \)
Specular: \( K_s \cos^n \phi \)

A-D-S Lighting with Flat Interpolation

```
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 );
```

Vertex Shader
Fragment Shader

What the light can produce
What the eye sees

ER = LR * MR
EG = LG * MG
EB = LB * MB

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

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

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

Vertex Shader

Fragment Shader

A-D-S Lighting with Normal Interpolation

Note: The normal is interpolated throughout the facet. The light intensity is computed at each fragment. This avoids Mach Banding and makes the bright spot circular. 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

Smooth-rasterize N, L, E

Fragment Shader

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

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

Smooth-rasterize N, L, E

Fragment Shader
A-D-S Anisotropic Lighting with Normal Interpolation

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 );
vec3 gl_FragColor.rgb = Ka*ambient + Kd*diffuse + Ks*spec;

Summary

Flat

Normal

Anisotropic

#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;
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 );
}
vec4 ambient = uKa * uColor;
float d = max( dot( Normal, Light ), 0. );
vec4 diffuse = uKd * d * uColor;
float s = 0.;
if( dot( Normal, Light ) > 0. )
{
vec3 ref = normalize( 2. * Normal * dot( Normal, Light ) - Light );
s = pow( max( dot( Eye, ref ), 0. ), uShininess );
}
vec4 specular = uKs * s * uSpecularColor;
vec4 fFragColor = vec4( ambient.rgb + diffuse.rgb + specular.rgb, 1. );
}