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 \( \cos(\Phi) \)

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 at 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: \( K_d \cos(\theta) \)
Specular: \( K_s \cos(\Phi) \)

 ADS – Shininess=50
 ADS – Shininess=1000
 ADS – Shininess=1000 – Flat
A-D-S Lighting with Flat Interpolation

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

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 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

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

Vertex Shader

Fragment Shader

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

Smooth-rasterize N, L, E

Vertex Shader

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, giving 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.

N = Normal
L = Light vector
E = Eye vector
R = Light reflection vector
ER = Eye reflection vector

Color = LightColor * MaterialColor

#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;
  vNf = normalize(gl_NormalMatrix * gl_Normal); // surface normal vector
  vNs = vNf;
  vLf = eyeLightPosition - ECposition.xyz; // vector from the point
  vLs = vLf; // to the light position
  vEf = vec3(0., 0., 0.) - ECposition.xyz; // vector from the point
  vEs = vEf; // to the eye position
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
  Per-fragment lighting: the 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);
  } Per-fragment lighting: the fragment shader, I
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. );