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(Θ)
Specular = Light intensity proportional to cos²(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

**Ambient:** $K_a$

**Diffuse:** $K_d \cdot \cos \theta$

**Specular:** $K_s \cdot \cos \varphi$

---

**Images:**

- **Ambient-only**
- **Diffuse-only**
- **Specular-only**

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

Each facet 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

Vertex Shader
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 );
flat rasterize ambient, diffuse, specular

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

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

What the light can produce
ER = LR * MR
EG = LG * MG
EB = LB * MB

What the eye sees
ER = LR * MR
EG = LG * MG
EB = LB * MB

White Light

Green Light
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.

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

**Vertex Shader**

\[
\text{Smooth-rasterize ambient, diffuse, spec}
\]

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

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

**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.

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

**Fragment Shader**

\[
\text{Smooth-rasterize N, L, E}
\]

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

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

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

Fragment Shader

```
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;
```

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

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 brushed metal behave this way.


Fragment Shader

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;

Summary

Flat

Normal

Anisotropic

Smooth

Reflection
#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 v Ef;
out vec3 vEs;

vec3 eyeLightPosition = vec3( uLightX, uLightY, uLightZ );

void
main()
{

vec4 ECposition = uModelViewMatrix * aVertex;

N f = normalize( uNormalMatrix * aNormal ); // surface normal vector
Ns = N f;

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

}
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;

fFragColor = vec4( ambient.rgb + diffuse.rgb + specular.rgb, 1. );