Using Shaders for Lighting

Oregon State University
Mike Bailey

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 \( \cos^s(\varphi) \)

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: \( K_s \cos^s(\varphi) \)
### A-D-S Lighting with Flat Interpolation

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

- \( N = \text{Normal} \)
- \( L = \text{Light vector} \)
- \( E = \text{Eye vector} \)
- \( R = \text{Light reflection vector} \)
- \( ER = \text{Eye reflection vector} \)
- \( \text{Color} = \text{LightColor} \times \text{MaterialColor} \)

#### Vertex Shader

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

#### Fragment Shader

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

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

What the eye sees

\[
ER = L_B \times M_R
\]
\[
EG = L_B \times M_G
\]
\[
EB = L_B \times M_B
\]

White Light

Green Light

### 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.

- \( N = \text{Normal} \)
- \( L = \text{Light vector} \)
- \( E = \text{Eye vector} \)
- \( R = \text{Light reflection vector} \)
- \( ER = \text{Eye reflection vector} \)
- \( \text{Color} = \text{LightColor} \times \text{MaterialColor} \)

#### Vertex Shader

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

#### Fragment Shader

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

### 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.

- \( N = \text{Normal} \)
- \( L = \text{Light vector} \)
- \( E = \text{Eye vector} \)
- \( R = \text{Light reflection vector} \)
- \( ER = \text{Eye reflection vector} \)
- \( \text{Color} = \text{LightColor} \times \text{MaterialColor} \)

#### Fragment Shader

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

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

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.


#version 330

Per-fragment lighting: the vertex shader

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;

    if( uFlat )
    {
        Normal = normalize(vNf);
        Light = normalize(uL1);
        Eye = normalize(uE);
    }
    else
    {
        Normal = normalize(vNs);
        Light = normalize(uLs);
        Eye = normalize(uEs);
    }
}

Per-fragment lighting: the fragment shader, I

Per-fragment lighting: the fragment shader, I
Per-fragment lighting: the fragment shader, II

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

float s = 0.0;
if (dot(Normal, Light) > 0.0) // only do specular if the light can see the point
{
    vec3 ref = normalize(2.0 * Normal * dot(Normal, Light) - Light);
    s = pow(max(dot(Eye, ref), 0.0), uShininess);
}

vec4 specular = uKs * s * uSpecularColor;

gl_FragColor = vec4(ambient.rgb + diffuse.rgb + specular.rgb, 1.0);
```