Using Shaders for Lighting

Mike Bailey
mjb@cs.oregonstate.edu
Oregon State University

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: \( K_s \cos(n \phi) \)

Vertex Shader

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

Fragment Shader

\[
\text{gl_FragColor.rgb} = \text{K_a} \times \text{ambient} + \text{K_d} \times \text{diffuse} + \text{K_s} \times \text{spec};
\]

Flat Rasterize ambient, diffuse, spec

What the light can produce

What the eye sees

What the material can reflect

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

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.

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

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

**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.

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

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

**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, given a stronger impression that the surface is shiny.

\[
\text{vec3 ambient} = \text{Color.rgb} \\
\text{vec3 diffuse} = \text{max}\left(\text{dot}(\text{L}, \text{N}), 0.0\right) \times \text{Color.rgb} \\
\text{vec3 R} = \text{normalize}\left(\text{reflect}\left(-\text{L}, \text{N}\right)\right) \\
\text{vec3 spec} = \text{LightColor} \times \text{pow}\left(\text{max}\left(\text{dot}\left(\text{R}, \text{E}\right), 0.0\right), \text{Shininess}\right)
\]

\[
\text{vec3 reflcolor} = \text{textureCube}(\text{ReflectUnit}, \text{R}).\text{rgb};
\]

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

**Vertex Shader**

**Fragment Shader**
vec3 ambient = Color.rgb;
float d = max( dot(Normal,Light), 0. );
vec4 diffuse = uKd * d * uColor;
float s = 0.;