Using Shaders for Lighting

Mike Bailey
mjb@cs.oregonstate.edu

Computer Graphics

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 cos^nφ

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** = $K_s \cdot \cos^n(\phi)$

A-D-S Lighting with Flat Interpolation

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

```
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

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

Fragment Shader

```
gl_FragColor.rgb = ambient + diffuse + spec;
```

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

White Light

- $E_n = L_n \cdot M_n$
- $E_d = L_d \cdot M_d$
- $E_s = L_s \cdot M_s$

Green Light

- $E_n = L_n \cdot M_n$
- $E_d = L_d \cdot M_d$
- $E_s = L_s \cdot M_s$

What the light can produce

What the material can reflect

What the eye sees
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

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

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.

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

Color = LightColor * MaterialColor

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

Vertex Shader
Fragment Shader

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

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

Vertex Shader
Fragment Shader
vec3 ambient = Color.rgb;
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;
float fFragColor.rgb = vec4( ambient.rgb + diffuse.rgb + specular.rgb, 1. );