**Using Shaders for Lighting**

**Lighting Definitions**

- **N** = Normal vector
- **L** = Vector from Point to the Light
- **R** = Light reflection vector
- **E** = Vector from the Point to the eye

- **Ambient** = Light intensity that is “everywhere”
- **Diffuse** = Light intensity proportional to \( \cos(\theta) \)
- **Specular** = Light intensity proportional to \( \cos^s(\phi) \)

**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 at each vertex
- **Per-fragment lighting** = Interpolate the vectors across the entire polygon and then compute A-D-S at each fragment

**A-D-S Lighting**

- **Ambient**: \( K_a \)
- **Diffuse**: \( K_d \cdot \cos(\theta) \)
- **Specular**: \( K_s \cdot \cos^s(\phi) \)
Applying Per-Fragment Lighting

Fragment shader:

```glsl
#version 330 compatibility
uniform vec3 uColor;
uniform vec3 uSpecularColor;
uniform float uKa, uKd, uKs; // coefficients of each type of lighting
uniform float uShininess; // specular exponent
in vec2 vST; // texture cords
in vec3 vN; // normal vector
in vec3 vL; // vector from point to light
in vec3 vE; // vector from point to eye
void main()
{
    vec3 Normal = normalize(vN);
    vec3 Eye = normalize(vE);
    vec3 Light = normalize(vL);
    vec3 myColor = uColor; // default color
    vec3 ambient = uKa * myColor;
    float d = 0.;
    float s = 0.;
    if (dot(Normal, Light) > 0.) // only do specular if the light can see the point
    {
        d = dot(Normal, Light);
        vec3 R = normalize(reflect(-Light, Normal));
        s = pow(max(dot(Eye, R), 0.), uShininess);
    }
    vec3 diffuse = uKd * d * myColor;
    vec3 specular = uKs * s * uSpecularColor;
    gl_FragColor = vec4(ambient + diffuse + specular, 1.);
}
```

Per-fragment A-D-S Lighting with Flat Interpolation

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

Flat Shading

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

\[ E_R = L_R \times M_R \]
\[ E_G = L_G \times M_G \]
\[ E_B = L_B \times M_B \]

This is how you implement subtractive coloring.

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.


Summary

Flat Smooth Anisotropic