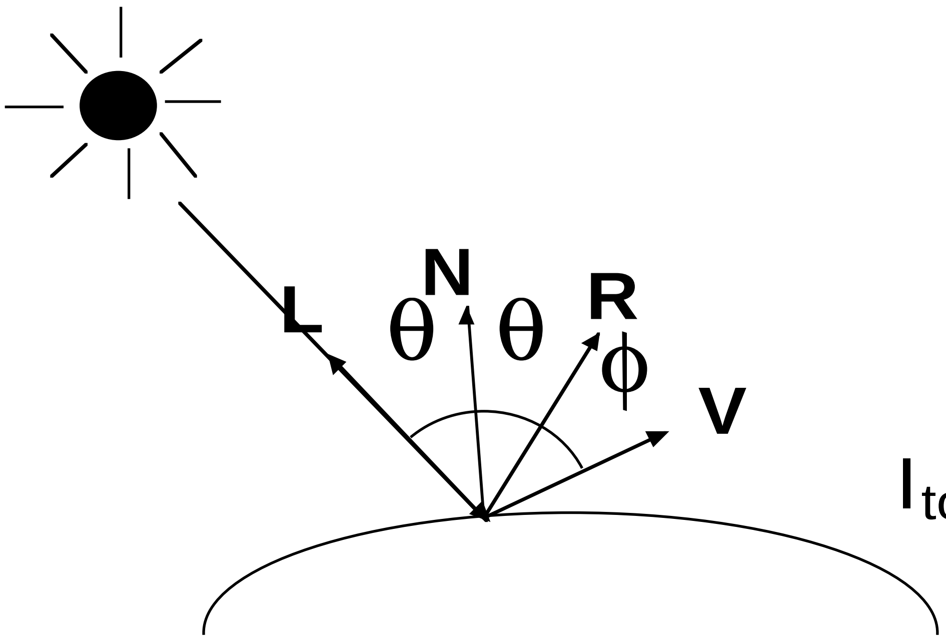
Total contribution from one surface



$$I_{amb} = k_d * I_a$$

$$I_{diff} = k_d * I_l * \mathbf{L} \cdot \mathbf{N}$$

$$I_{spec} = k_s * I_l * (\mathbf{R} \cdot \mathbf{V})^{\mathbf{n}}$$

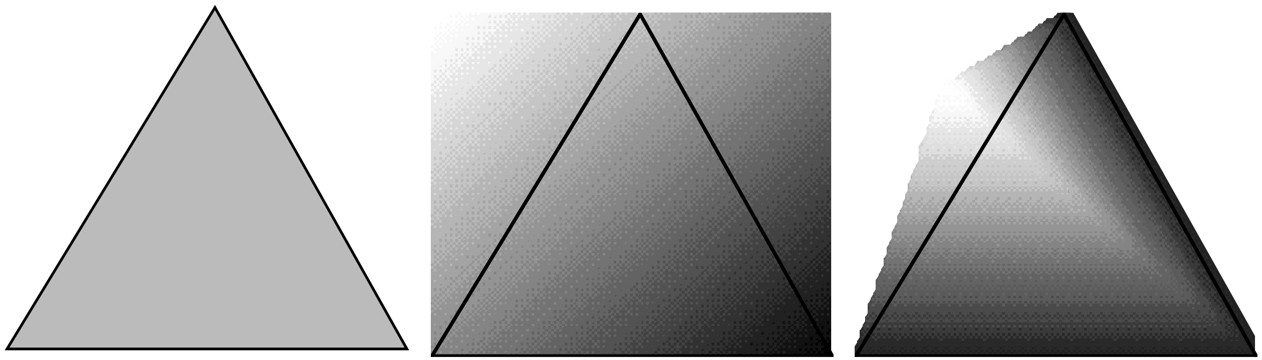
$$I_{total} = I_{amb} + I_{diff} + I_{spec}$$

Polygon shading

Using the illumination models in high-speed polygon rendering

Three ways to render a shaded polygon:

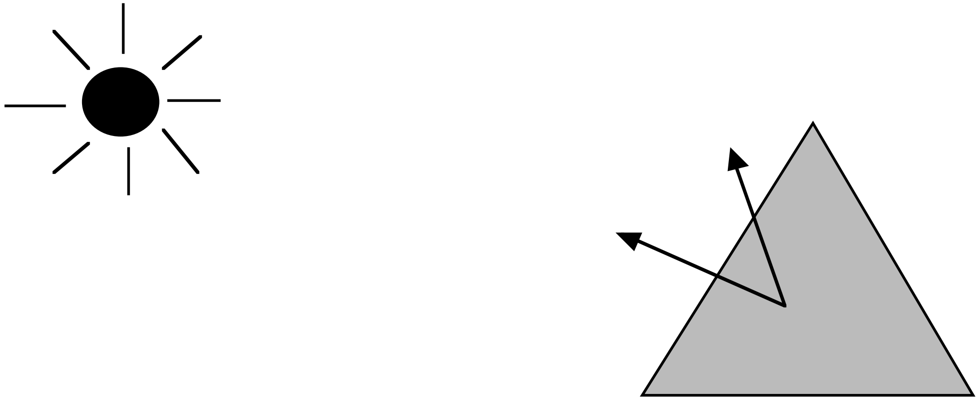
Flat shading
Gouraud shading
Phong shading



Flat shading

Intensity calculated once and for all for the whole polygon

E.g. Ip = **N·L**



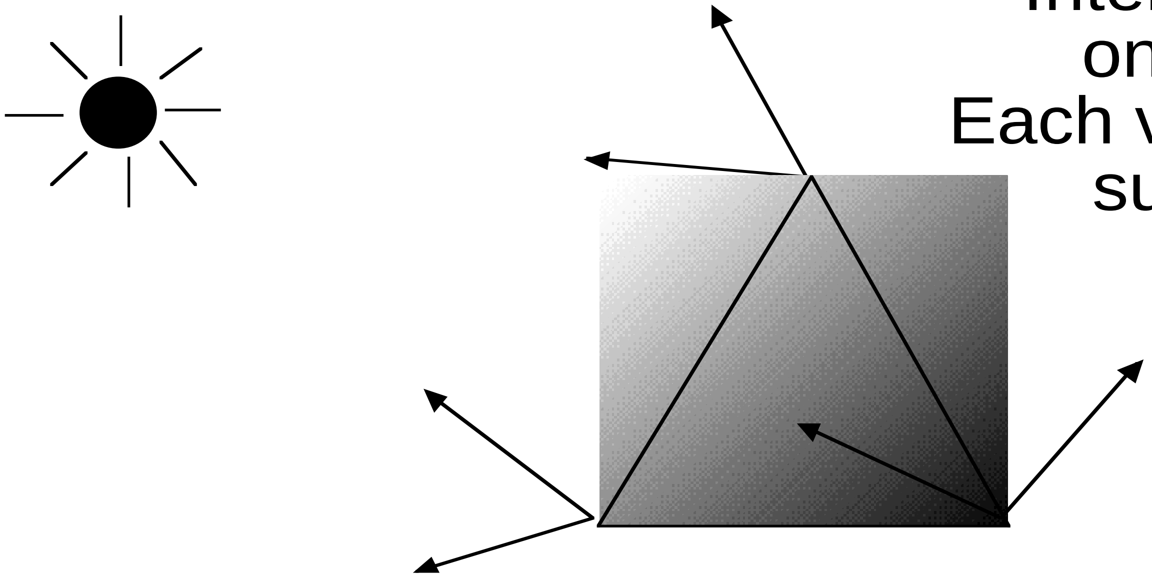
Flat shading is “correct” when:

1) The surfaces should be flat, not approximating a curved surface
2) Distance to light source high => N·L constant
3) Distance to camera high => V·R constant

and in particular

4) When the problem is not lighting, but something else! (Rendering surface identifications)

Gouraud shading



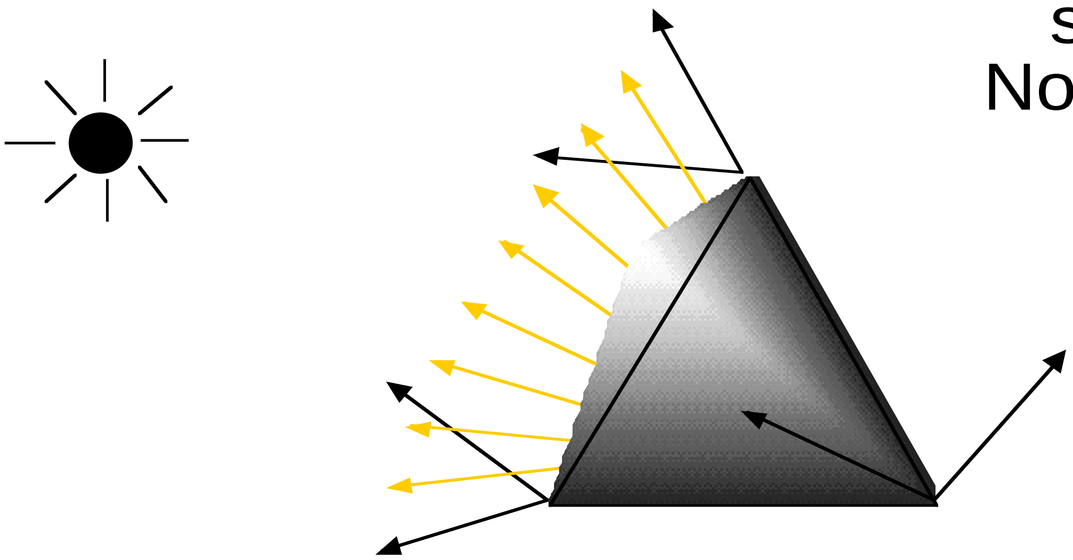
Intensity calculated
once per vertex
Each vertex has its own
surface normal

Gouraud shading

can simulate curved surfaces fairly well,
but many polygons may be needed, and edges remain visible

Built-in in the fixed pipeline - extremely fast

Phong shading



Each vertex has its own surface normal
Normal vectors are interpolated

Phong shading

can simulate curved surfaces very well, even with low polygon counts

can be fairly fast with “Fast Phong Shading”, an incremental method

Best implemented in shader programs

Phong shading
≠
The Phong model

Phong Shading doesn’t necessarily use specular reflections.

Phong Shading = normal-vector interpolation shading

Light sources in OpenGL

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Set position with glLightfv:

```
glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

Light source position:

```
GLfloat light_position[] = { 1.0, 1.0, 1.0, 1.0 };
```

Distant light source, direction:

```
GLfloat light_position[] = { 5.0, 5.0, 2.0, 0.0 };
```

Light source attributes

Set with glLightfv:

```
glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
glLightfv(GL_LIGHT0, GL_SPECULAR, amb);
```

Select attenuation:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, b);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, c);
```

Distance-attenuation model:

$$f(d) = 1 / (a + b*d + c*d^2)$$

Materials in OpenGL

glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);

Diffuse and specular values are often the same:

GL_DIFFUSE_AND_SPECULAR

The exponent in the Phong model:

glMaterialfv(GL_FRONT, GL_SHININESS, shininess);

A surface can also be self-illuminated:

glMaterialfv(GL_FRONT, GL_EMISSION, emission);

glMaterialfv
GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT_AND_DIFFUSE

glLightfv(

$$I_{amb} = k_d * I_a$$

GL_AMBIENT

GL_DIFFUSE, GL_SPECULAR

$$I_{diff} = k_d * I_l * \mathbf{L \cdot N}$$

$$I_{spec} = k_s * I_l * (\mathbf{R \cdot V})^{\mathbf{n}}$$

L_SHININESS

$$I_{total} = I_{amb} + I_{diff} + I_{spec}$$