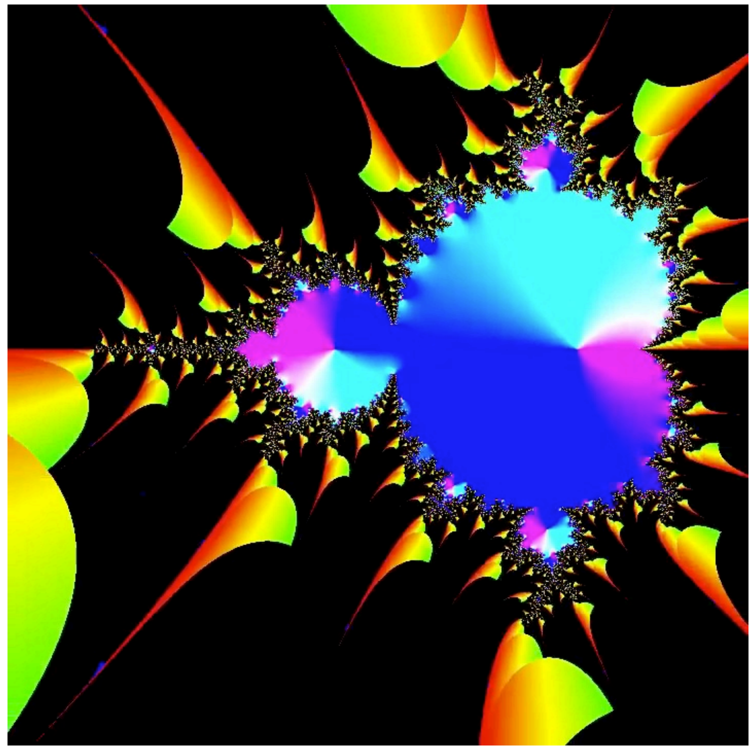
TSBK 07

Computer Graphics

Ingemar Ragnemalm, ISY



Lecture 5
3D graphics part 3

Illumination

Illumination applied: Shading

Surface detail: Mappings
Texture mapping

...

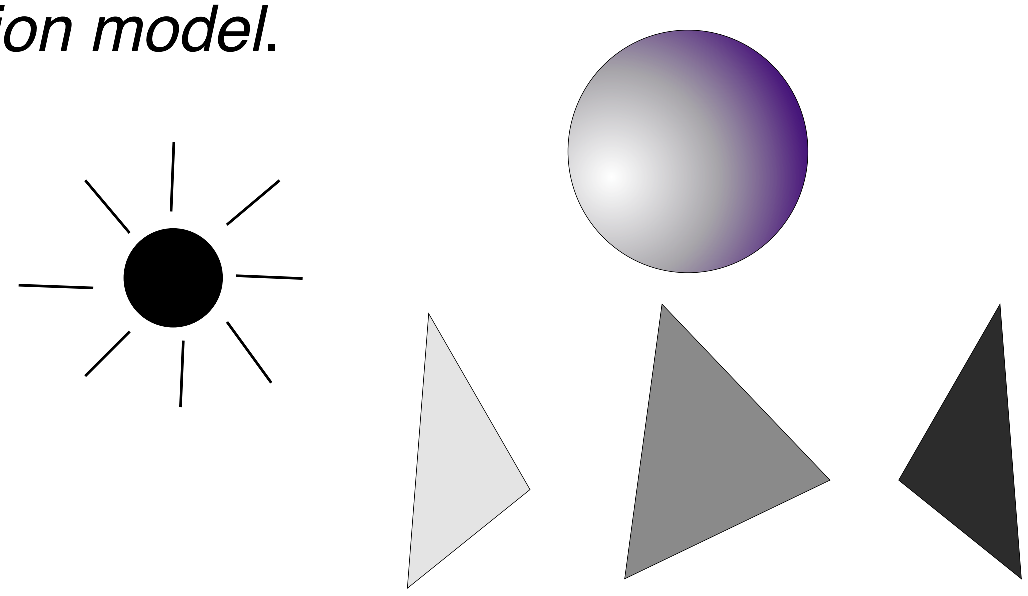
Illumination

We know *where* to put a polygon. Now, *what* should we fill it with? What pixel value should we choose?

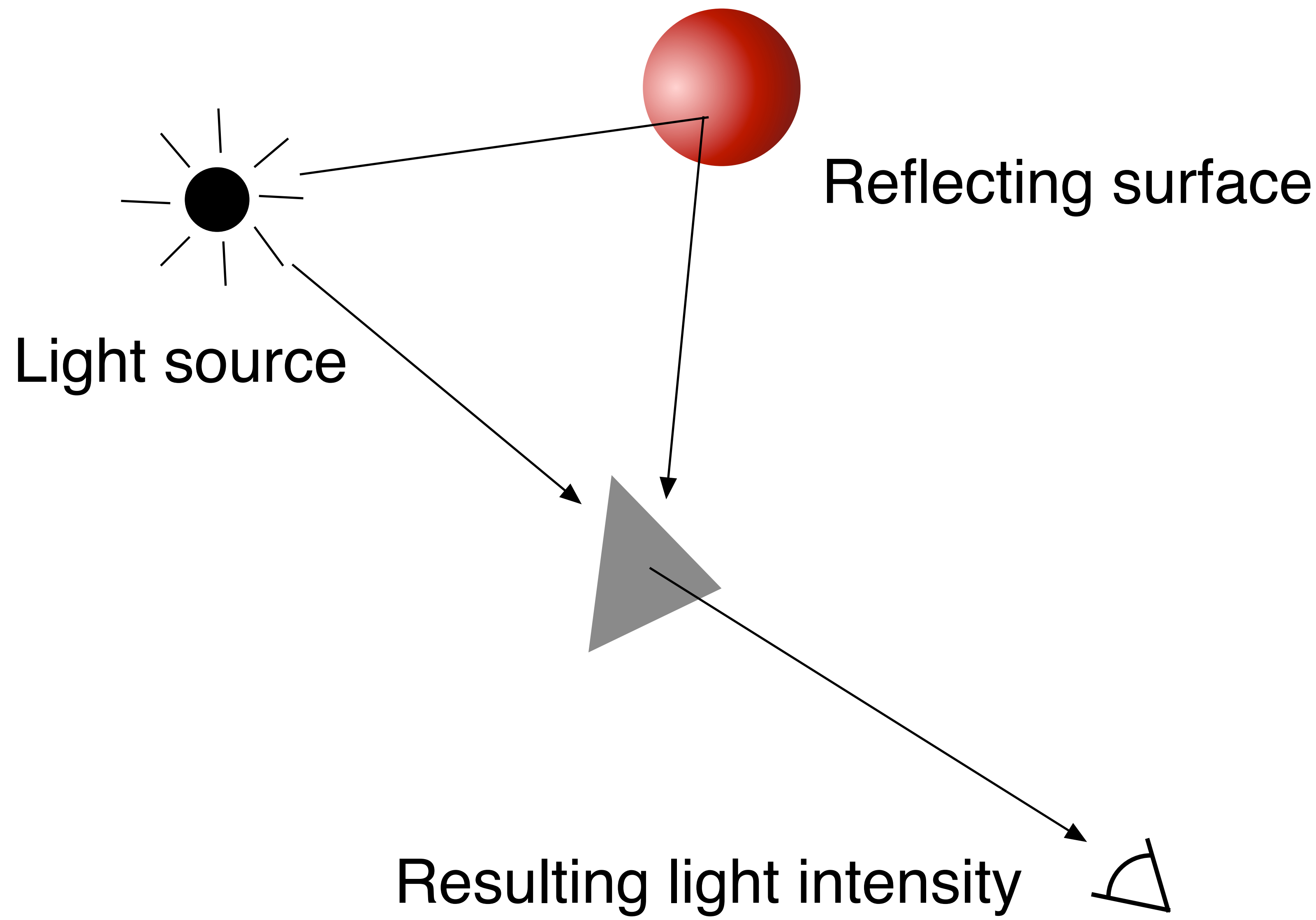
Several factors to take into account. The most important ones:

- Shading, illumination.
- Texture mapping.

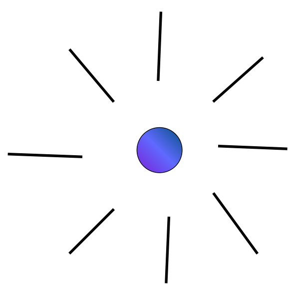
Shading is determined according to an *illumination model*.



Light sources



Light sources



Small sources can be modelled as **point light source**

Others can be modelled as **distributed light sources**

Reflections

Diffuse reflection

Specular reflection

Paper | Plastic | Metal | Mirror

3-component illumination model

A common simple illumination model is built from three components:

Ambient light

Diffuse reflections

Specular reflections

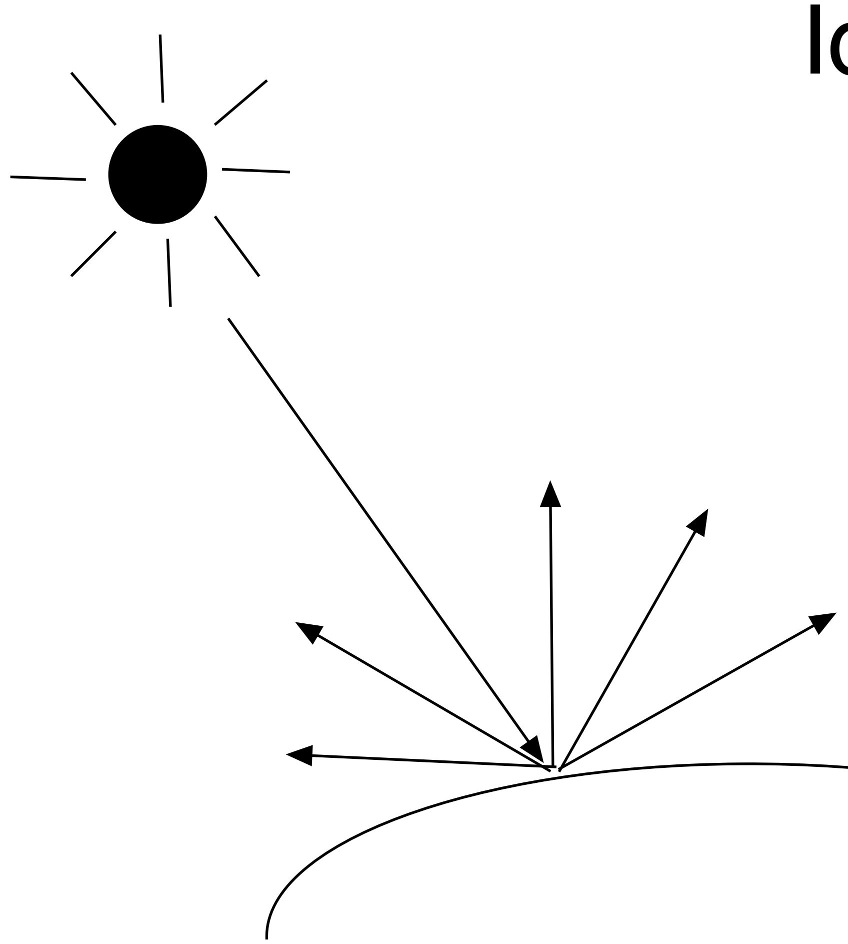
Ambient light

Same everywhere!

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

- I_{amb}: Light emitted from surface
- k_d: Diffuse reflectivity
- I_a: Ambient light level of scene

Diffuse reflections

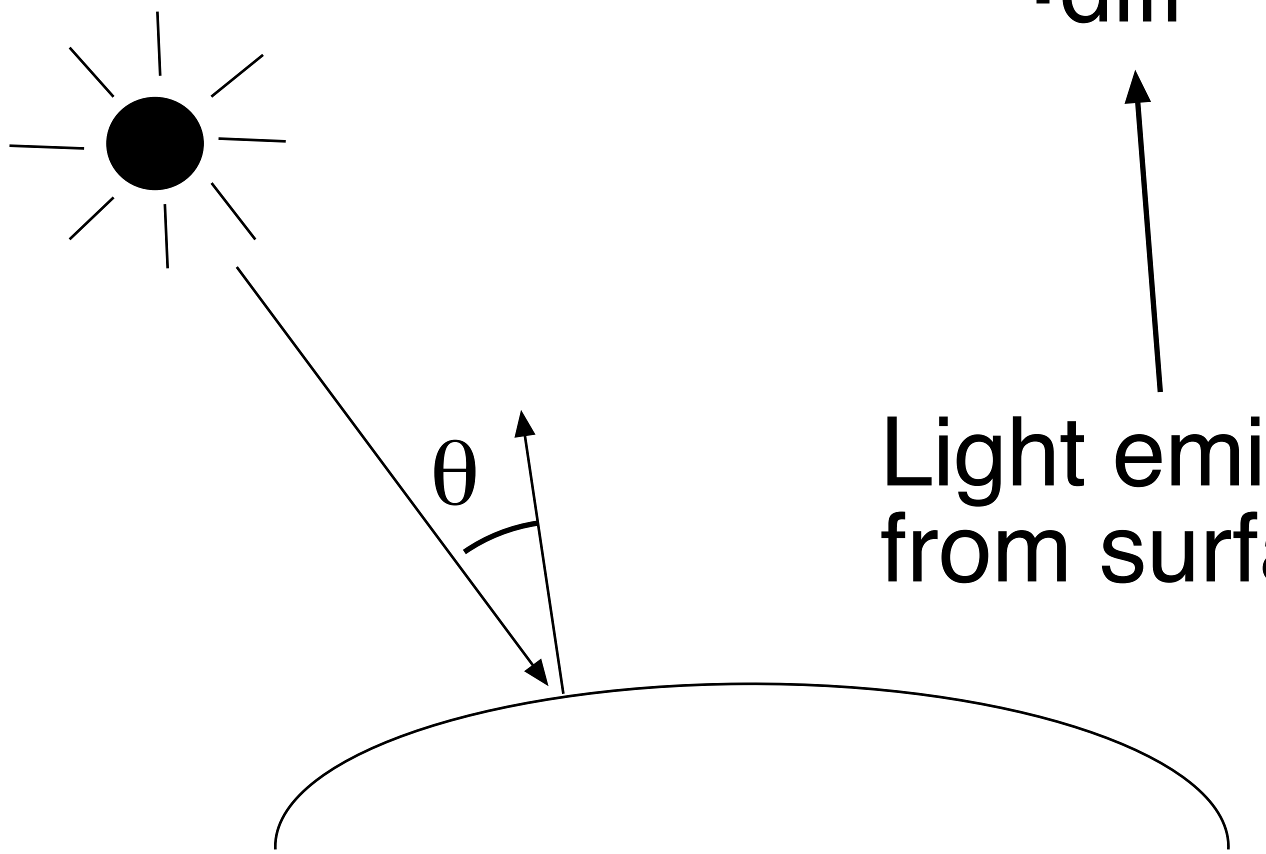


Ideal diffuse reflector =
Lambertian surface

Incoming light produces same
intensity in all directions!

Lambert's cosine law

$$I_{diff} = k_d * I_l * \cos\theta$$



Light emitted from surface

Diffuse reflectivity

Light level of light source

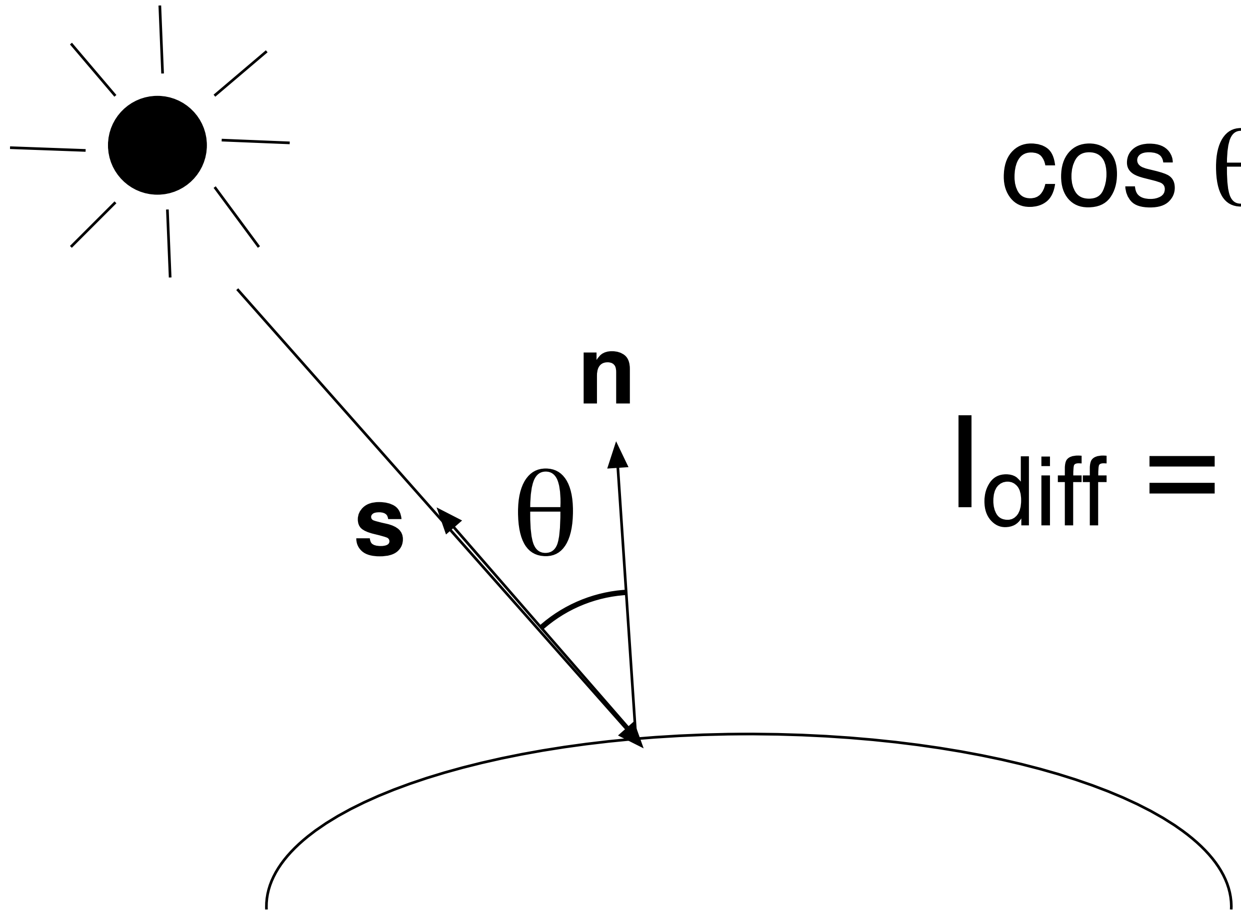
Example: Diffuse sphere

$k_d = 1$

$I_l = 0.9$

$I_a = 0.1$

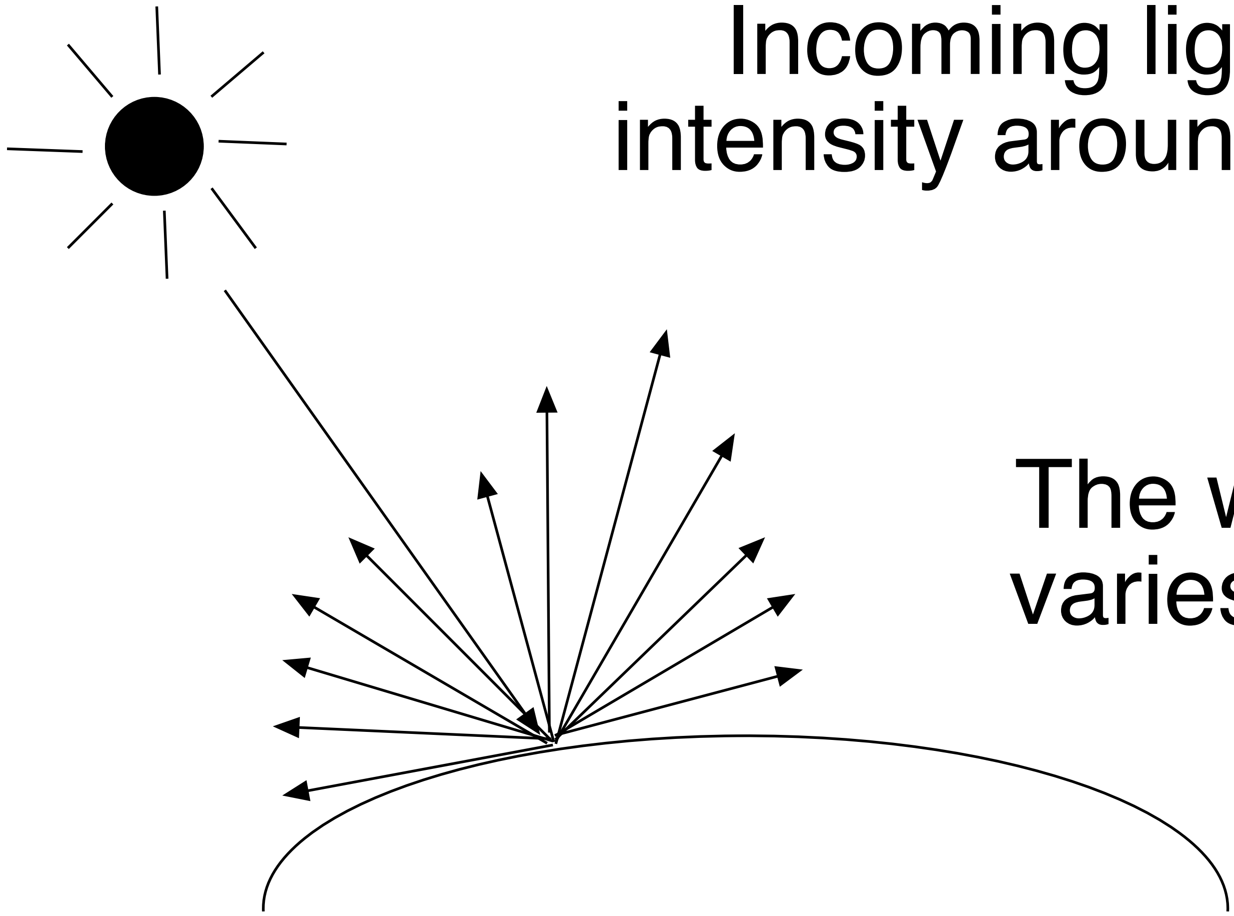
Dot product is nicer than angles!



$$\cos\theta = \mathbf{s}\cdot\mathbf{n}$$

$$I_{diff} = k_d * I_l * \mathbf{s}\bullet\mathbf{n}$$

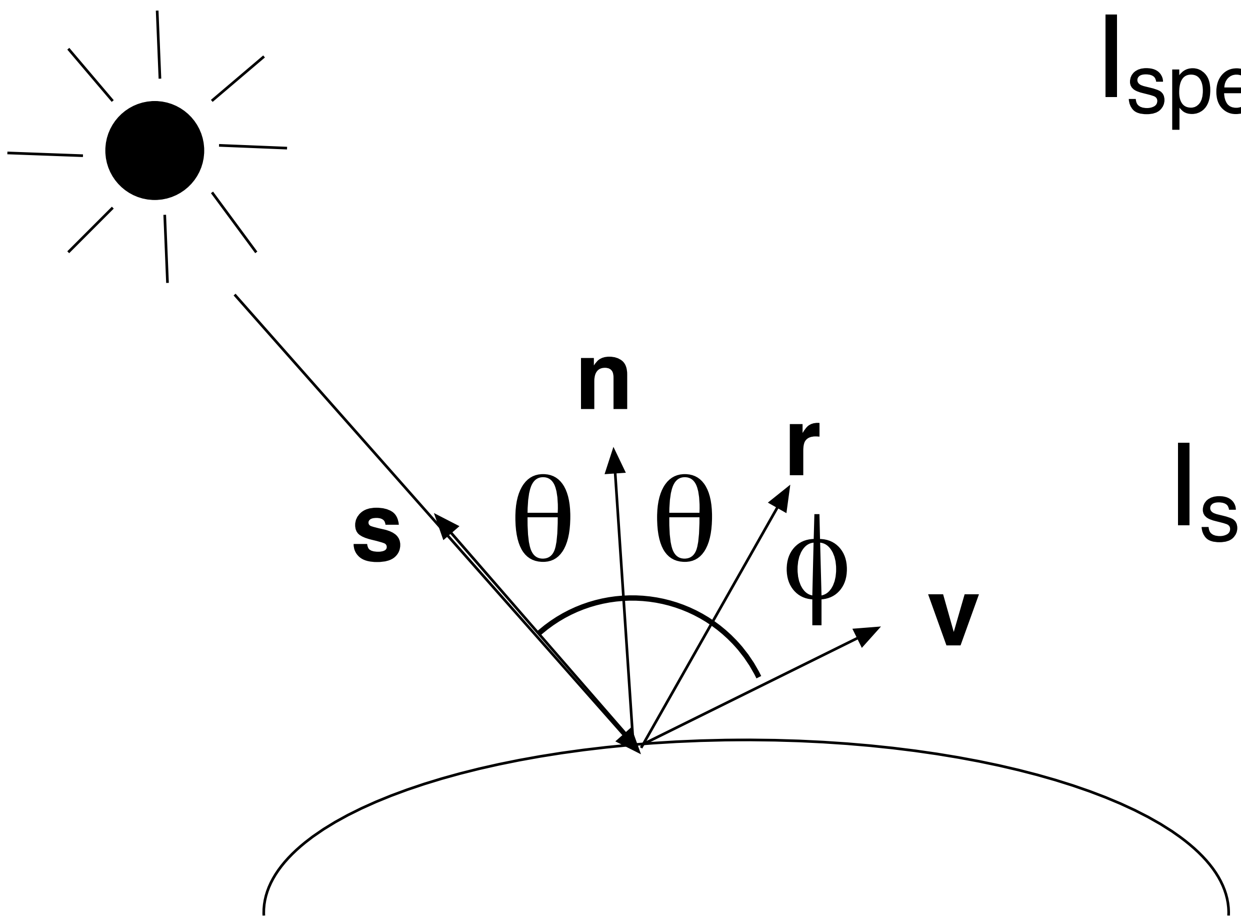
Specular reflections



Incoming light produces higher intensity around the mirroring angle!

The width of the highlight varies with surface types!

The Phong model



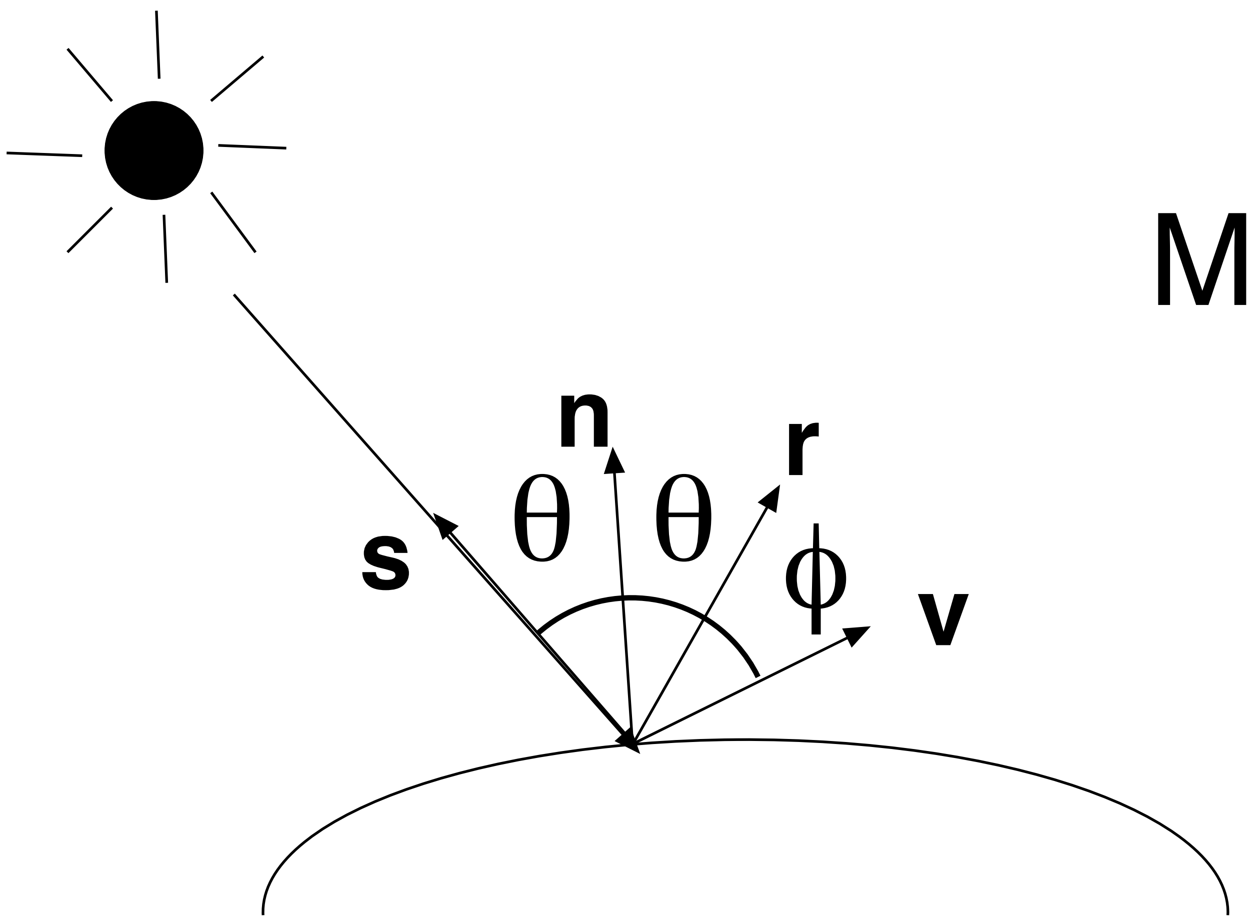
$$I_{spec} = W(\theta) * I_l * \cos^{\alpha} \phi$$

or

$$I_{spec} = k_s * I_l * \cos^{\alpha} \phi$$

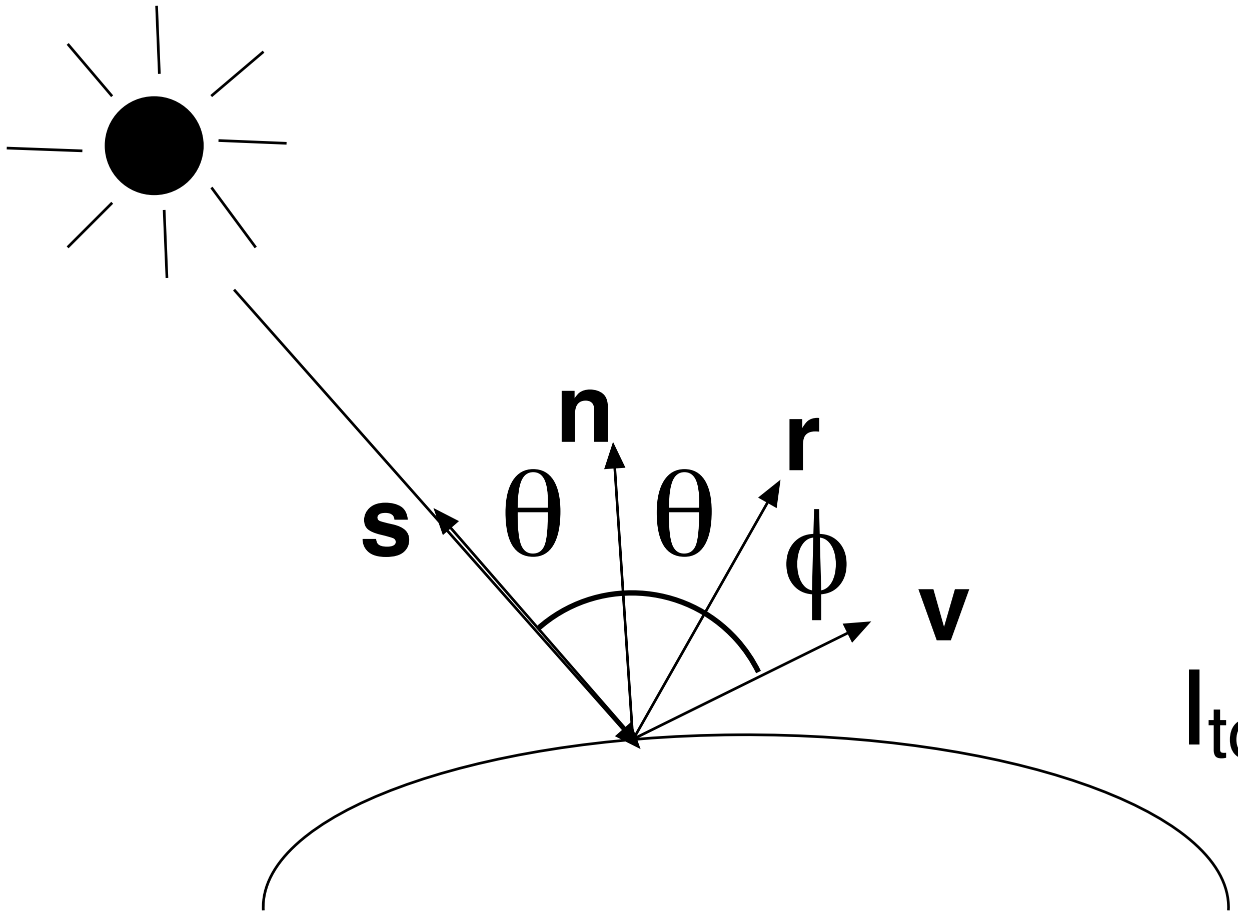
$$\cos \phi = \mathbf{r} \cdot \mathbf{v}$$

Calculation of R



Mirror **s** by **n**!

Total contribution from one light source



$I_{amb} = k_d * I_a$

$I_{diff} = k_d * I_l * \mathbf{s} \cdot \mathbf{n}$

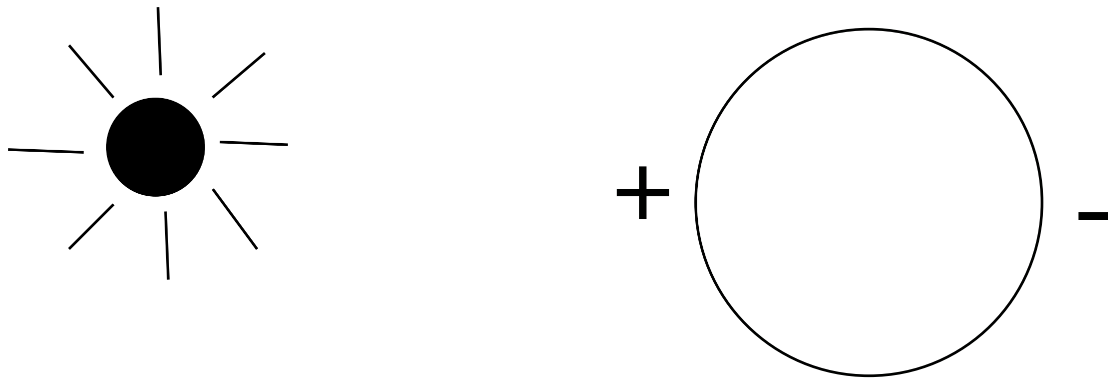
$I_{spec} = k_s * I_l * (\mathbf{r} \cdot \mathbf{v})^{\alpha}$

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

Clamping shading

Note that the three-component light model will produce negative light!

This will cause problems in scenes with several light sources!



To avoid problems, clamp the resulting value, e.g. max(0, light)

Complete formula

$$I = k_d * I_a + I_{diff} + \sum(k_d*I_l*\max(0, \mathbf{s}\cdot\mathbf{n}) + I_{spec} = k_s*I_l*\max(0, \mathbf{r}\cdot\mathbf{v})^{\alpha})\ )$$

where ∑ sums over all light sources

Examples

Diffuse surface, $k_d = 0.9$

Specular surface, $n = 1$

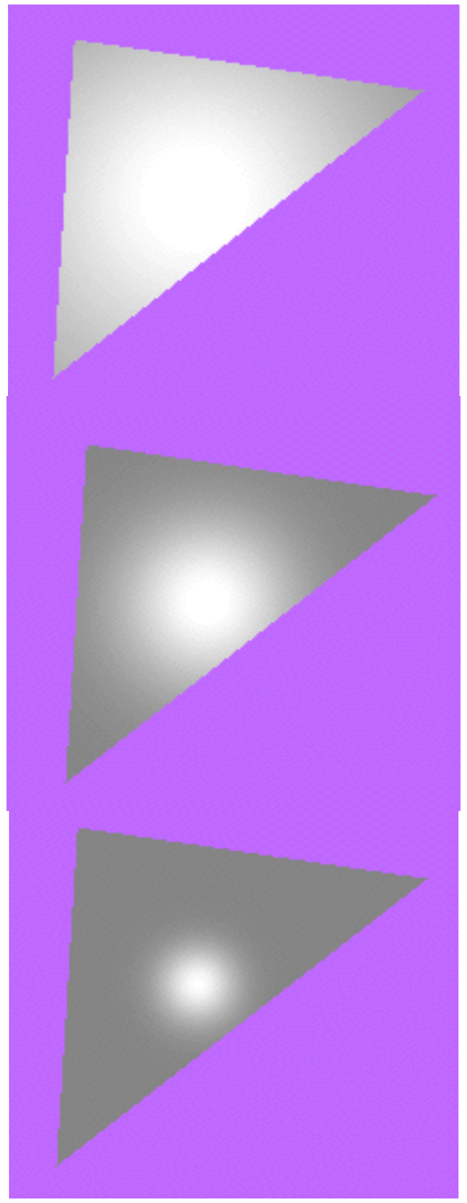
Specular surface, $n = 5$

Specular surface, $n = 25$

Specular surface, $n = 125$

$k_d = 0.45$
$k_s = 0.5$
$I_l = 1.0$
$I_a = 0.1$

Examples



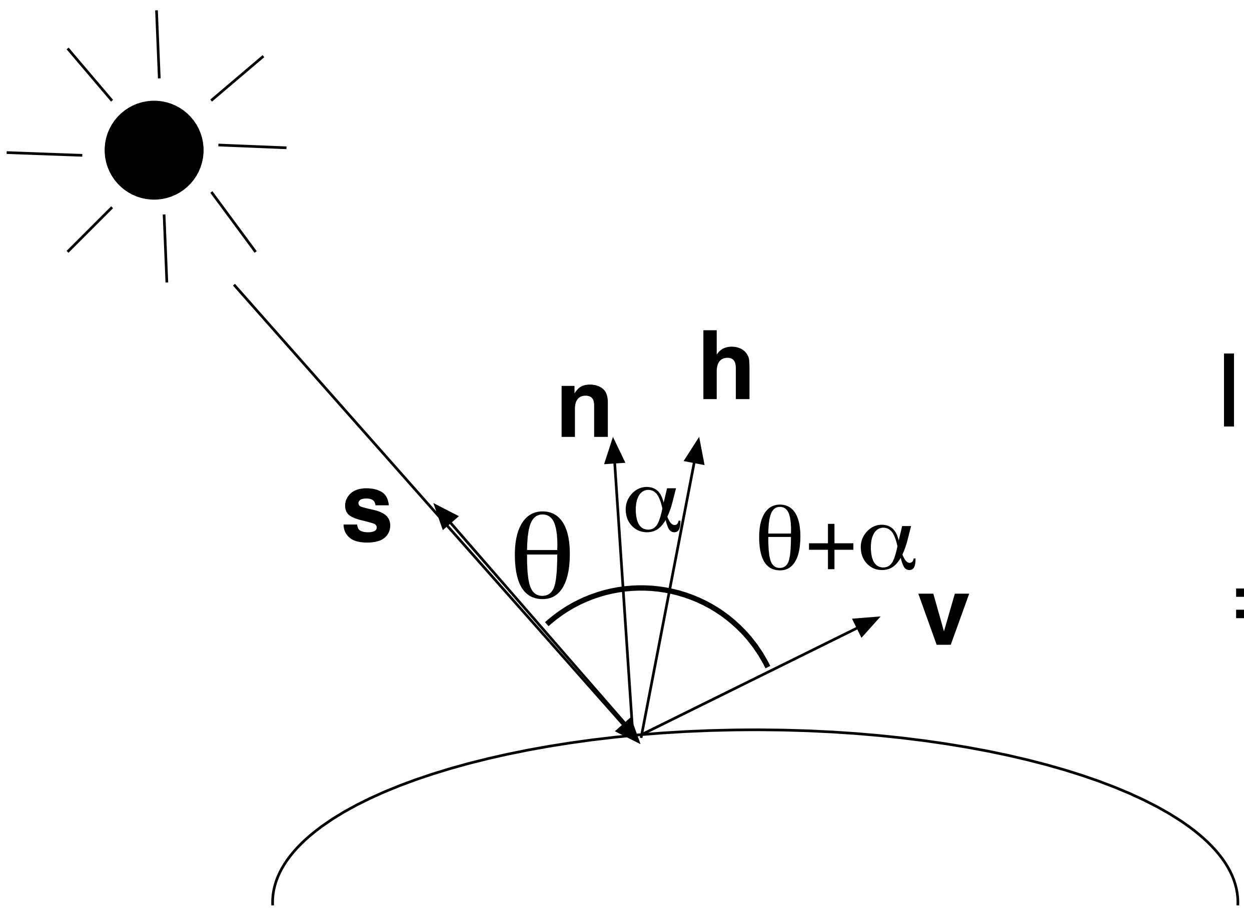
Specular surface, n = 5

Specular surface, n = 25

Specular surface, n = 125

$k_d = 0.45$
$k_s = 0.5$
$I_l = 1.0$
$I_a = 0.1$

Alternative formulation
Blinn-Phong



Halfway vector

$$\mathbf{h} = (\mathbf{s}+\mathbf{v}) / |\mathbf{s} + \mathbf{v}|$$

$$I_{spec} = k_s * I_l * \cos^n \alpha =$$

$$= I_{spec} = k_s * I_l * (\mathbf{n} \cdot \mathbf{h})$$

Advanced illumination models

Make k_s a function of the viewing angle - better modelling of glass and paper

BRDF - highly general multi-dimensional function

Global illumination models

Radiosity: Models light exchange recusively

Ray-tracing, trace viewing rays.

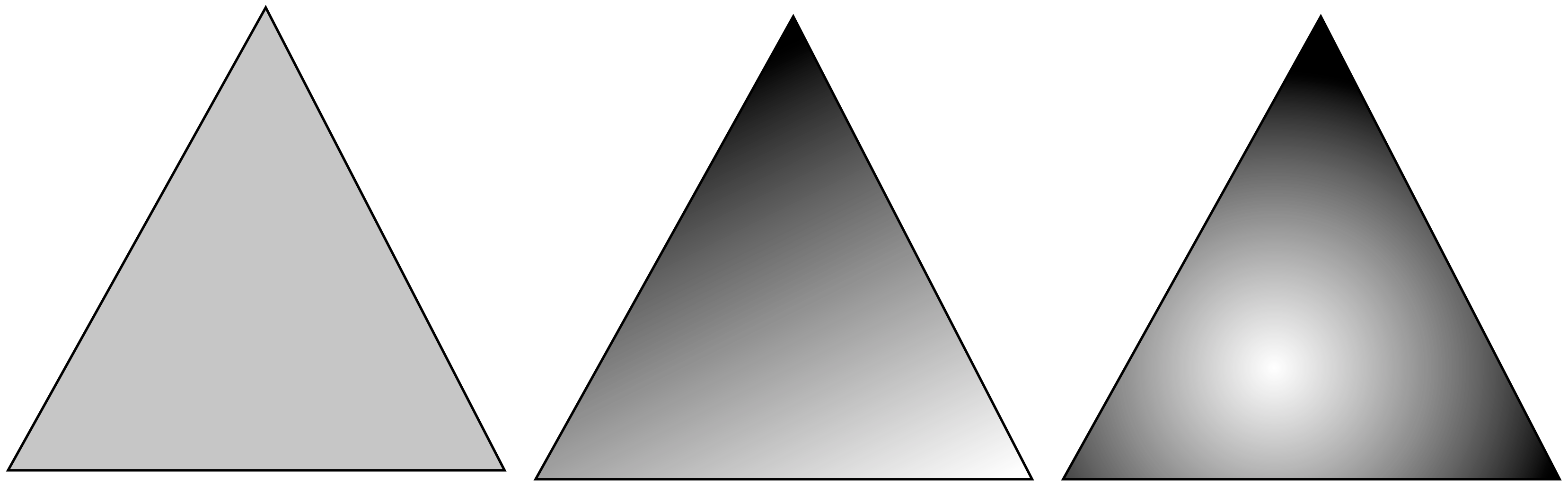
Photon mapping, “backwards ray-tracing”, trace lighting rays.

Polygon shading

Using the illumination models in high-speed polygon rendering

Three ways to render a shaded polygon:

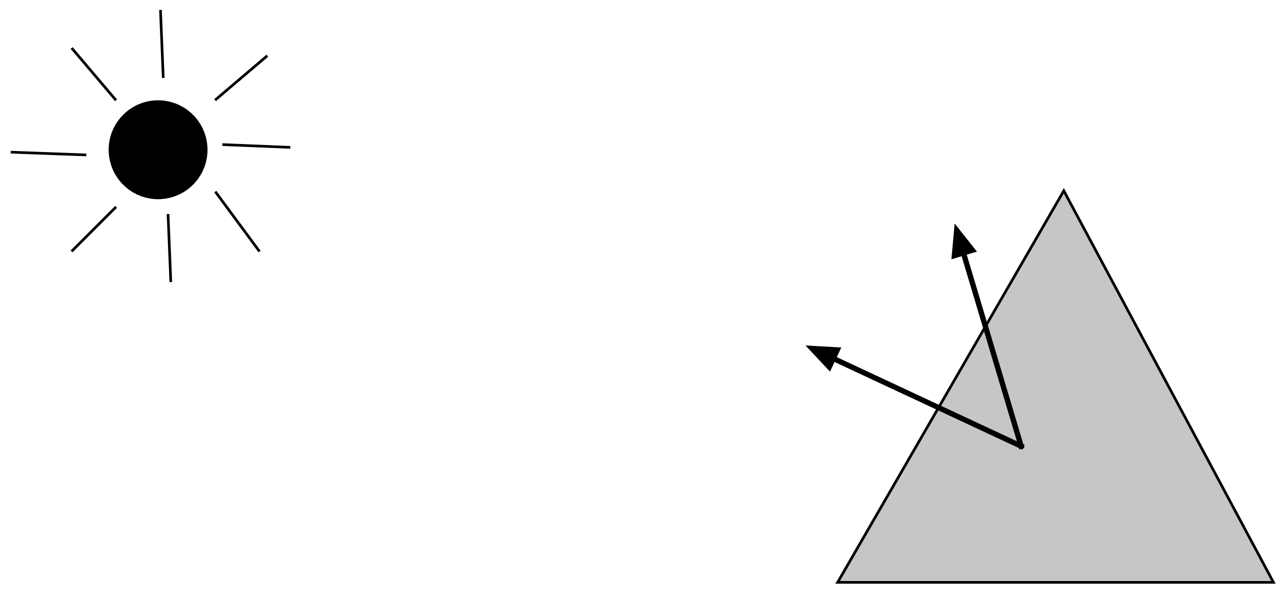
Flat shading
Gouraud shading
Phong shading



Flat shading

Intensity calculated once and for all for the whole polygon

E.g. $Ip = k_d \cdot \mathbf{N} \cdot \mathbf{L}$



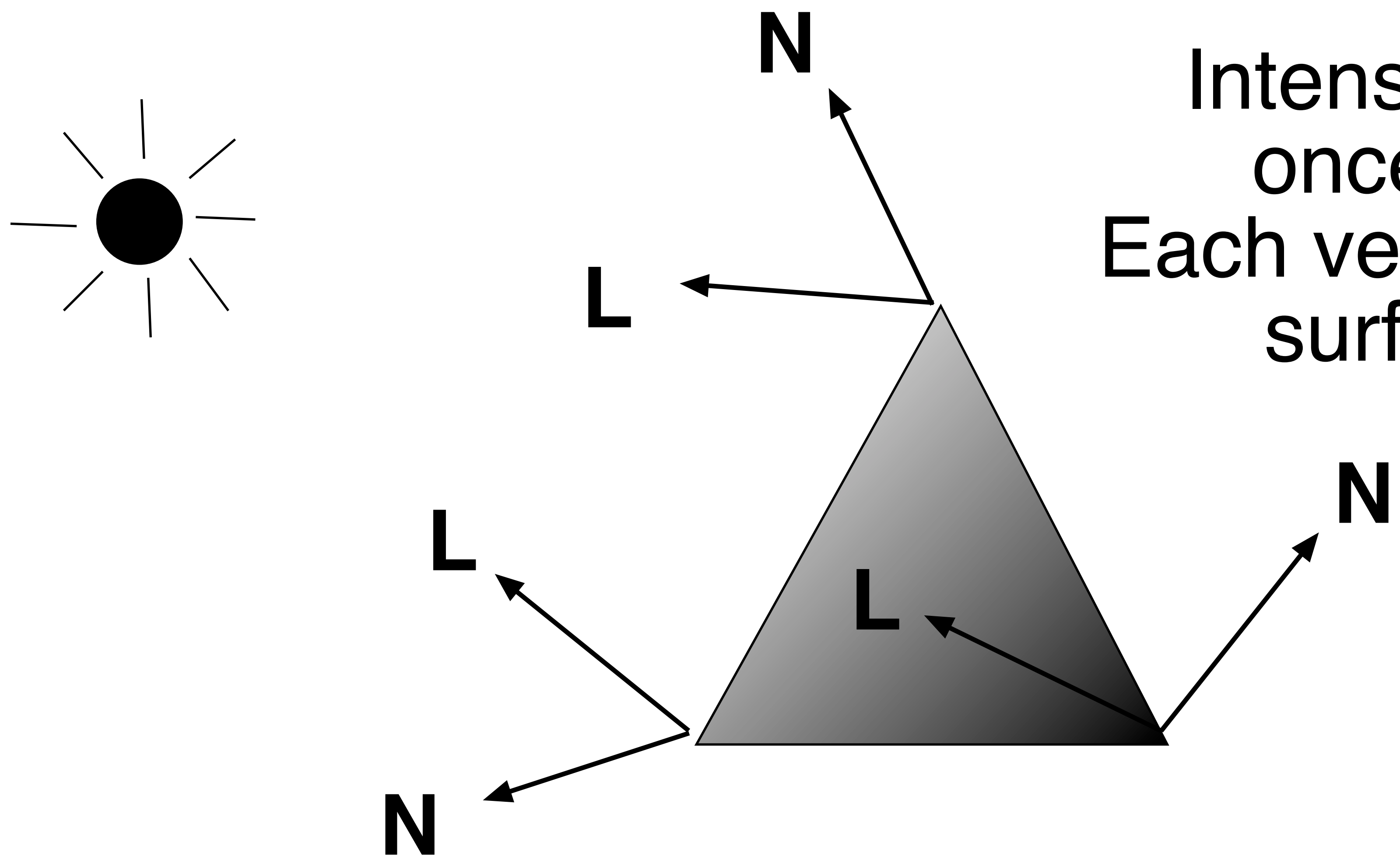
Flat shading is “correct” when:

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

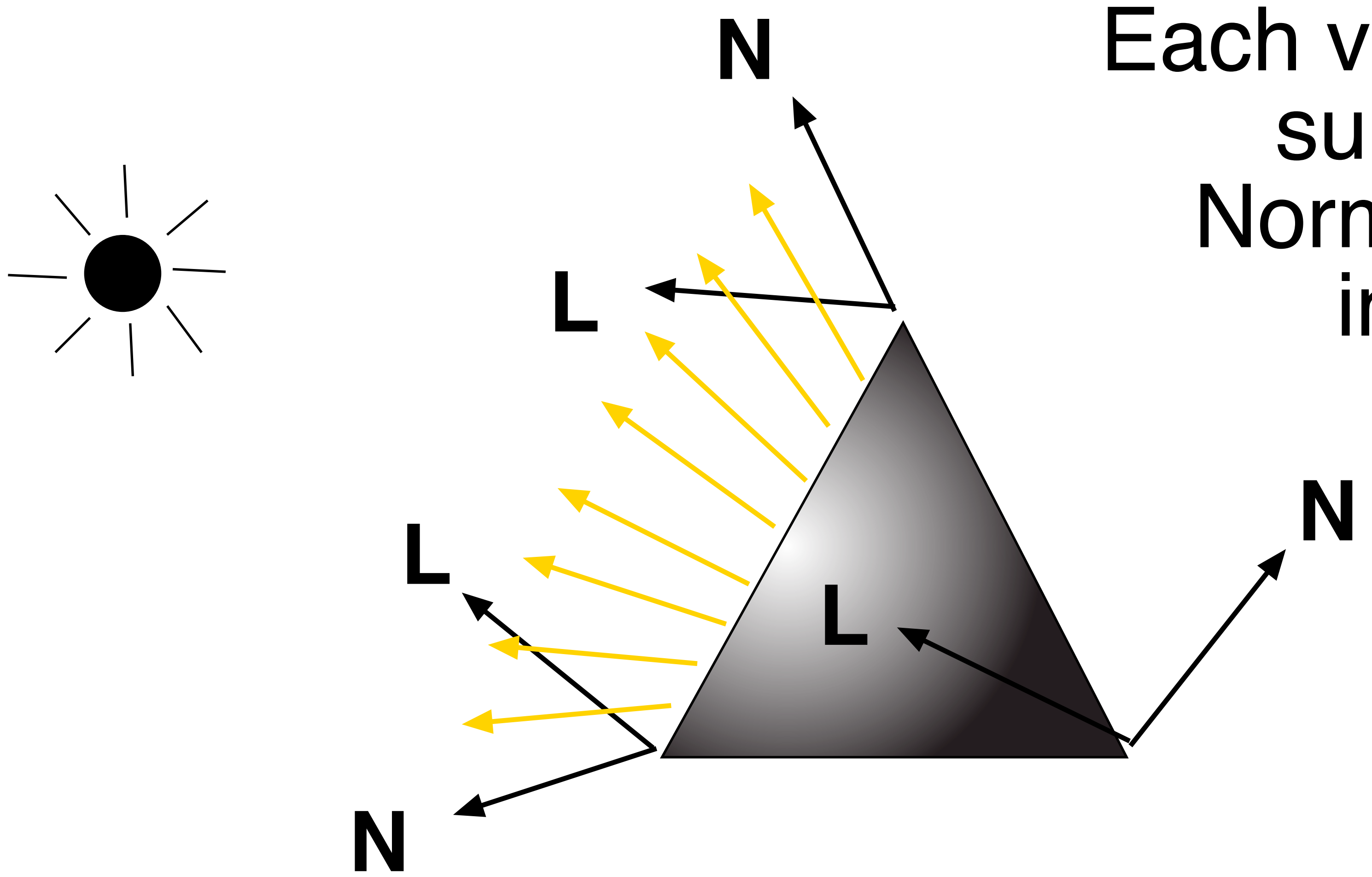
Interpolate *intensities!*

Gouraud shading

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

Calculations in vertex shader - 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

Calculate the light in the fragment shader

Computationally heavier

Phong shading
≠
The Phong model

Phong Shading doesn't necessarily use specular reflections.

Phong Shading = normal-vector interpolation shading

Example: Gouraud shader

- Transform normal vectors
- Calculate shading value per vertex, (here using diffuse only), by dot product with light direction
- Interpolate between vertices

Gouraud shader - vertex shader

```
#version 150
in  vec3 inPosition;
in  vec3 inNormal;
out vec3 exColor;

void main(void)
{
const vec3 light = vec3(0.58, 0.58, 0.58);
float shade;
shade = dot(normalize(inNormal), light);
shade = clamp(shade, 0, 1);
exColor = vec3(shade);
gl_Position = vec4(inPosition, 1.0);
}
```

Gouraud shader - fragment shader

```
#version 150
in  vec3 exColor;
out vec4 outColor;
void main(void)
{
outColor = vec4(exColor,1.0);
}
```

Gouraud shader

Note:

The variable “exColor” is interpolated between vertices!

dot() och normalize() do what you expect.

inNormal is the normal vector in model coordinates
(Should be transformed in a real program!)

The constant vector ”light” is here hard coded

Typical Gouraud shaded bunny

Version 2: Add specular lighting to vertex shader

```
// Specular
vec3 reflectedLightDirection = reflect(-light, norm);
vec3 eyeDirection = vec3(normalize(-inPosition));

float specularStrength = 0.0;
specularStrength = dot(reflectedLightDirection, eyeDirection);
float exponent = 8.0;
specularStrength = max(specularStrength, 0.01);
specularStrength = pow(specularStrength, exponent);

shade = (0.3*diffuseshade + 0.9*specularStrength);
}
```

(Again some transformations skipped.)

Specular Gouraud shaded bunny

A bit polygonal...

Example:
Phong shader

Better shading!

- **Interpolate normal vectors between vertices**
- **Calculate shading value per fragment**

Practically the same operations, but the light calculation are done in the fragment shader

Phong shader

Vertex shader

```
#version 150
in  vec3 inPosition;
in  vec3 inNormal;
out vec3 exNormal;
out vec3 surf;
void main(void)
{
exNormal = inNormal;
surf = inPosition; // For specular
gl_Position = vec4(inPosition, 1.0);
}
```

Phong shader

Fragment shader

```
#version 150
out vec4 outColor;
in vec3 exNormal;
in vec3 surf;
void main(void)
{
const vec3 light = vec3(0.58, 0.58, 0.58);
float shade;
shade = dot(normalize(exNormal), light);
shade = clamp(shade, 0, 1);
outColor = vec4(shade, shade, shade, 1.0);
}
```

..and add specular part

```
// Specular
vec3 reflectedLightDirection = reflect(-lightDirection, n);
vec3 eyeDirection = normalize(-surf);

float specularStrength = 0.0;
if (dot(lightDirection, n) > 0.0)
{
specularStrength = dot(reflectedLightDirection, eyeDirection);
float exponent = 200.0;
specularStrength = max(specularStrength, 0.01);
specularStrength = pow(specularStrength, exponent);
}

outColor = vec4(diffuseStrength*0.5 + specularStrength*0.5);
}
```

Specular Phong shaded bunny

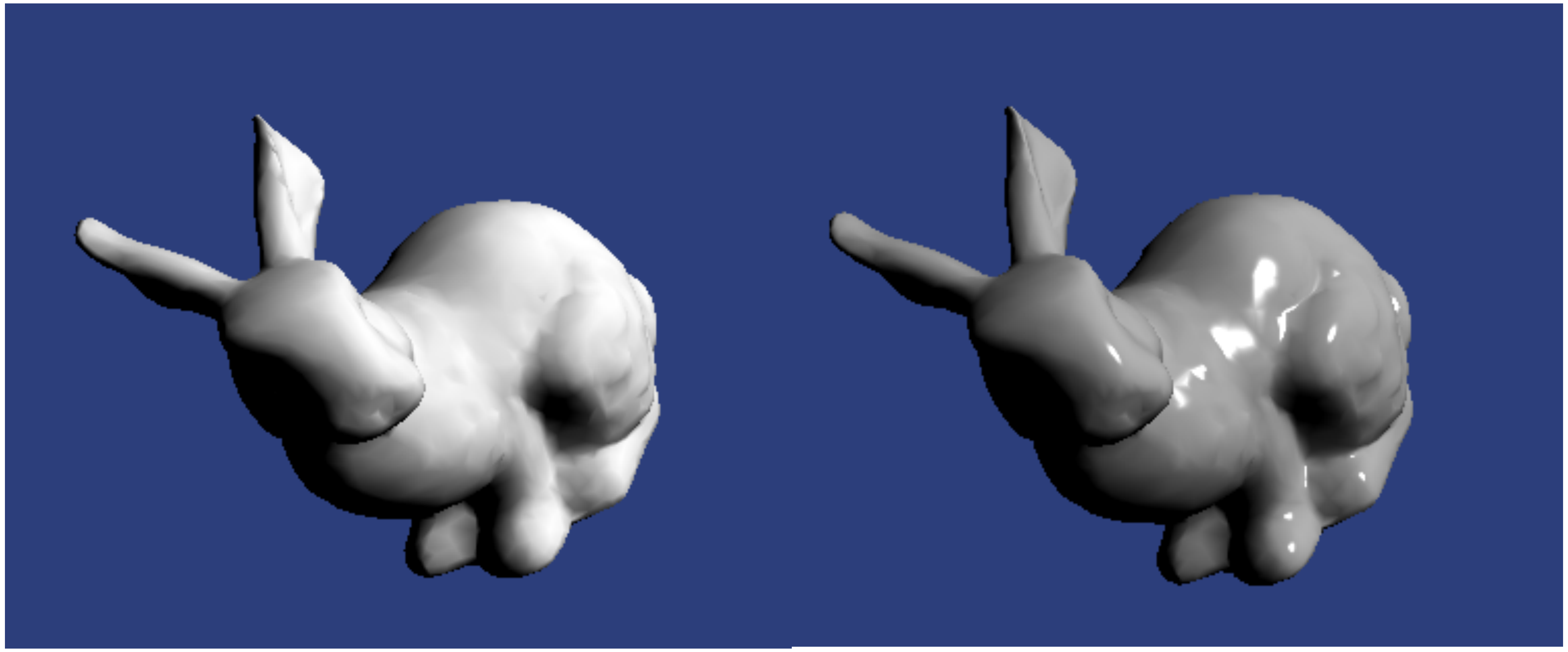
Now we're talking!

Use the shading you need, balance computing and quality

Gouraud

Phong

Same for Stanford bunny



Gouraud Phong