Lighting

CS425: Computer Graphics I

Fabio Miranda

https://fmiranda.me

Overview

- Light and shading
- Rendering equation
- Light-material interaction
- Reflection models:
  - Phong,
  - Blinn-Phong
- Shading models:
  - Flat
  - Gouraud
  - Phong

Lighting and shading

- Light is emitted by a light source.
- Light interacts with objects in the scene:
  - Part is absorbed, part is scattered in new directions.
- Finally, light is absorbed by a sensor (e.g., human eye, film).

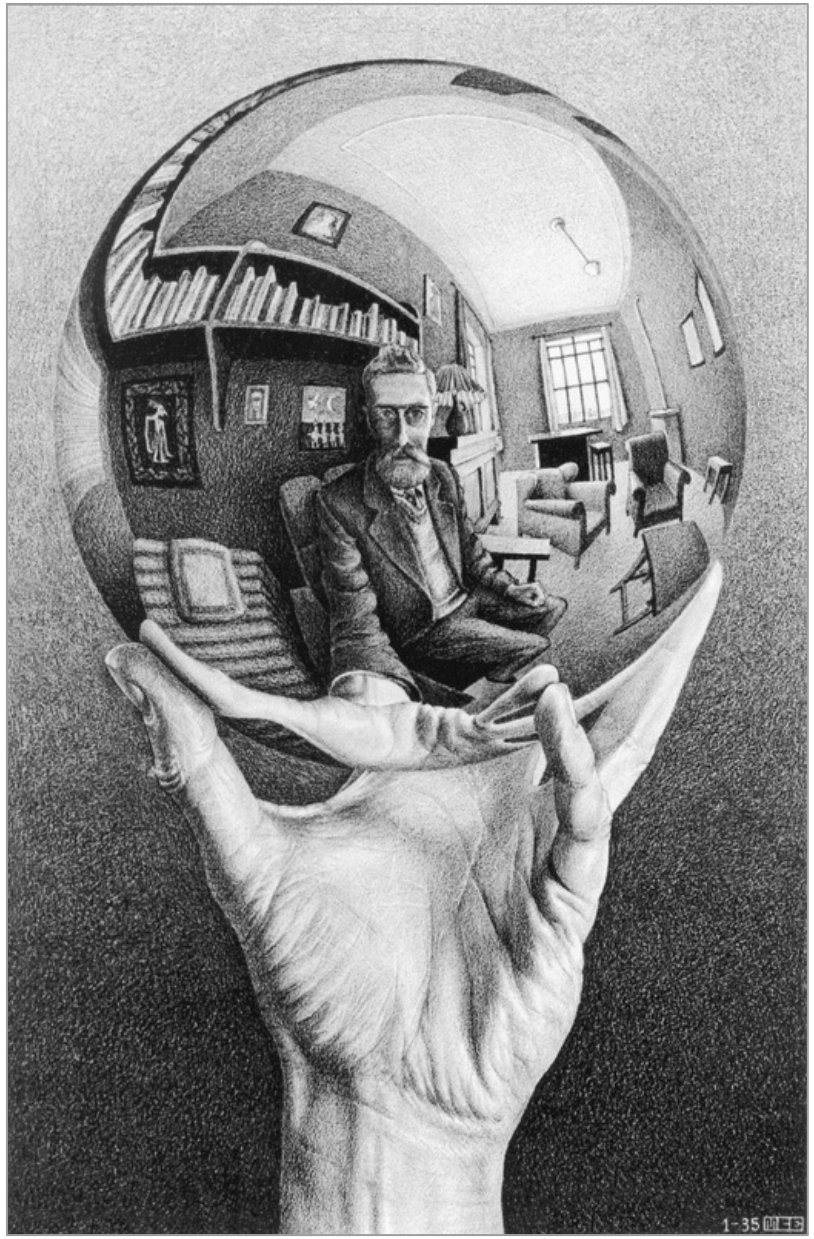
Lighting and shading

- Shading objects so their images appear three-dimensional.

- How can we model light-material interactions?
- We will see how to build a simple reflection model (Phong model) that can be used with real-time graphics hardware.

Why we need shading?

- Appearance of surfaces, taking into account:
  - Surface material
  - Lighting conditions



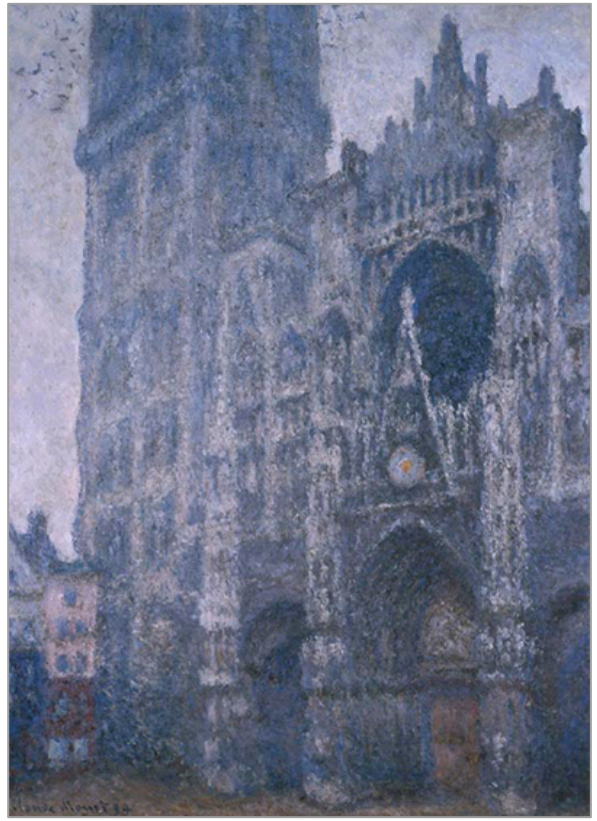
MC Escher

Why we need shading?

Full Sun

Morning Sun

Setting Sun



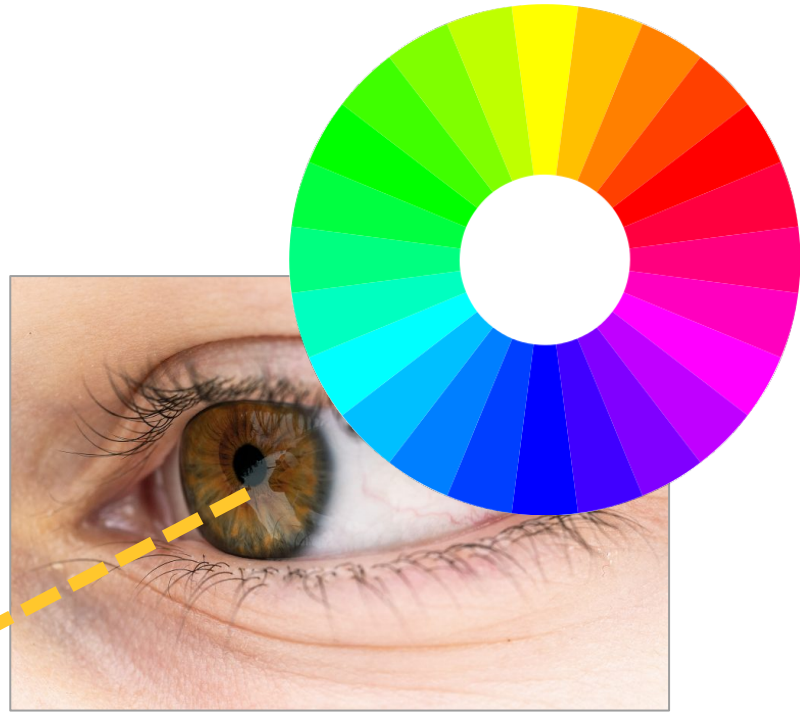
Grey Weather

Rouen Cathedral (Monet series)

Light and shading

1. Light source emits photons

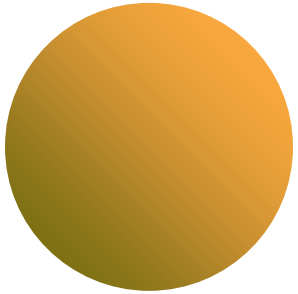
2. Photons interact with the environment: absorption, reflection



3. Some are captured by eye / camera

Lighting and shading

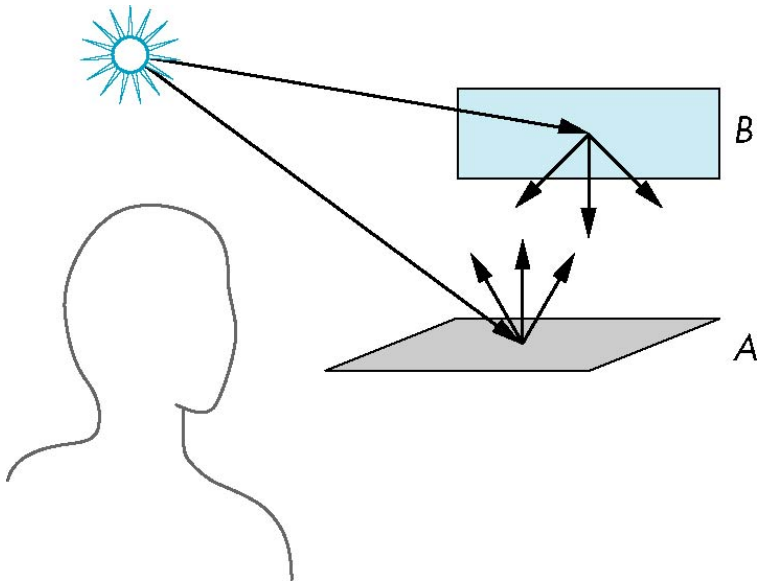
- Light-material interactions cause each point to have a different color or shade.



- We need to consider:
  - Light sources.
  - Material properties.
  - Location of viewer.
  - Surface orientation.

Lighting and shading

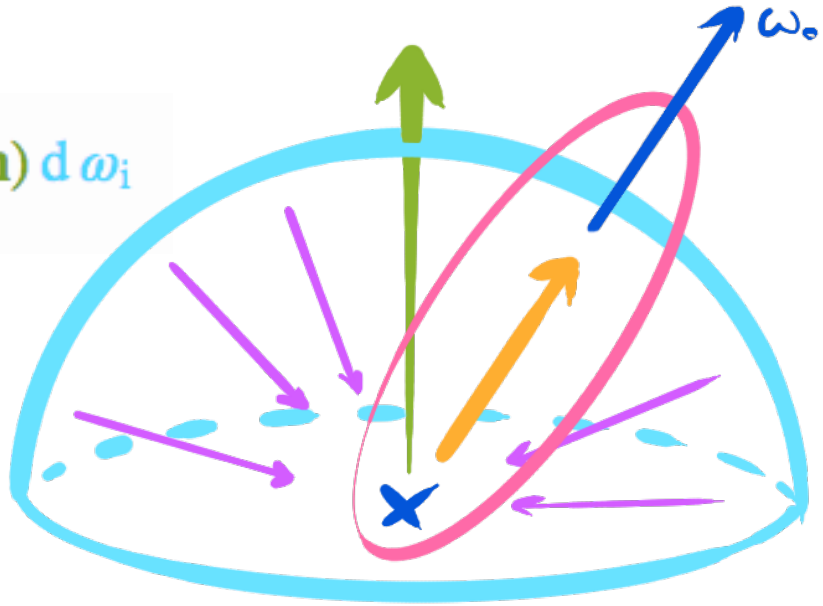
- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A and so on…



Rendering equation

- The infinite scattering and absorption of light can be described by the rendering equation:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) \, \mathrm{d}\omega_i$$

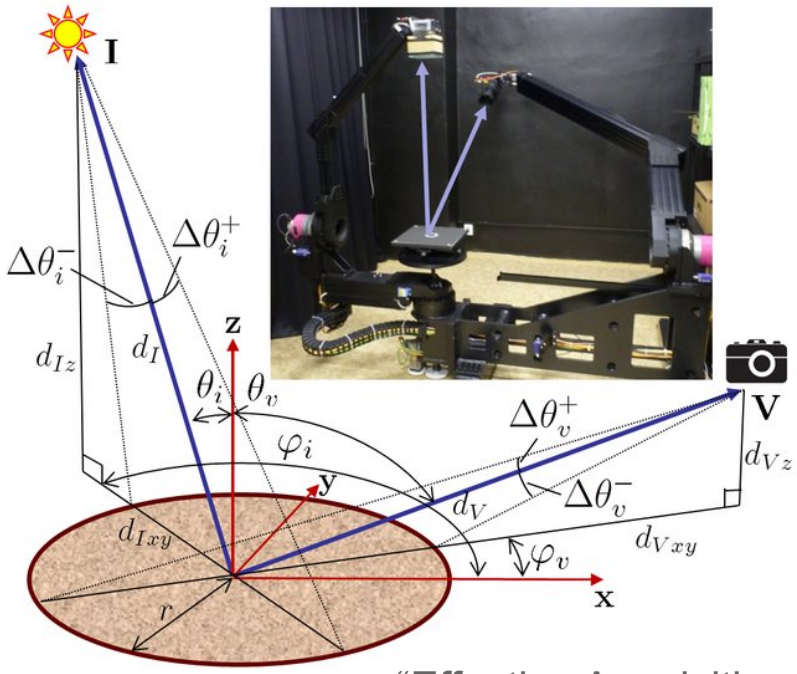


To find the light towards the viewer from a specific point, we sum the light emitted from such point plus the integral within the unit hemisphere of the light coming from a any given direction multiplied by the chances of such light rays bouncing towards the viewer (BRDF) and also by the irradiance factor over the normal at the point.

https://chuckleplant.github.io/2017/05/28/light-shafts.html

Bidirectional reflectance distribution function

- Function that defines how light is reflected at an opaque surface.



"Effective Acquisition of Dense Anisotropic BRDF", Filip et al.

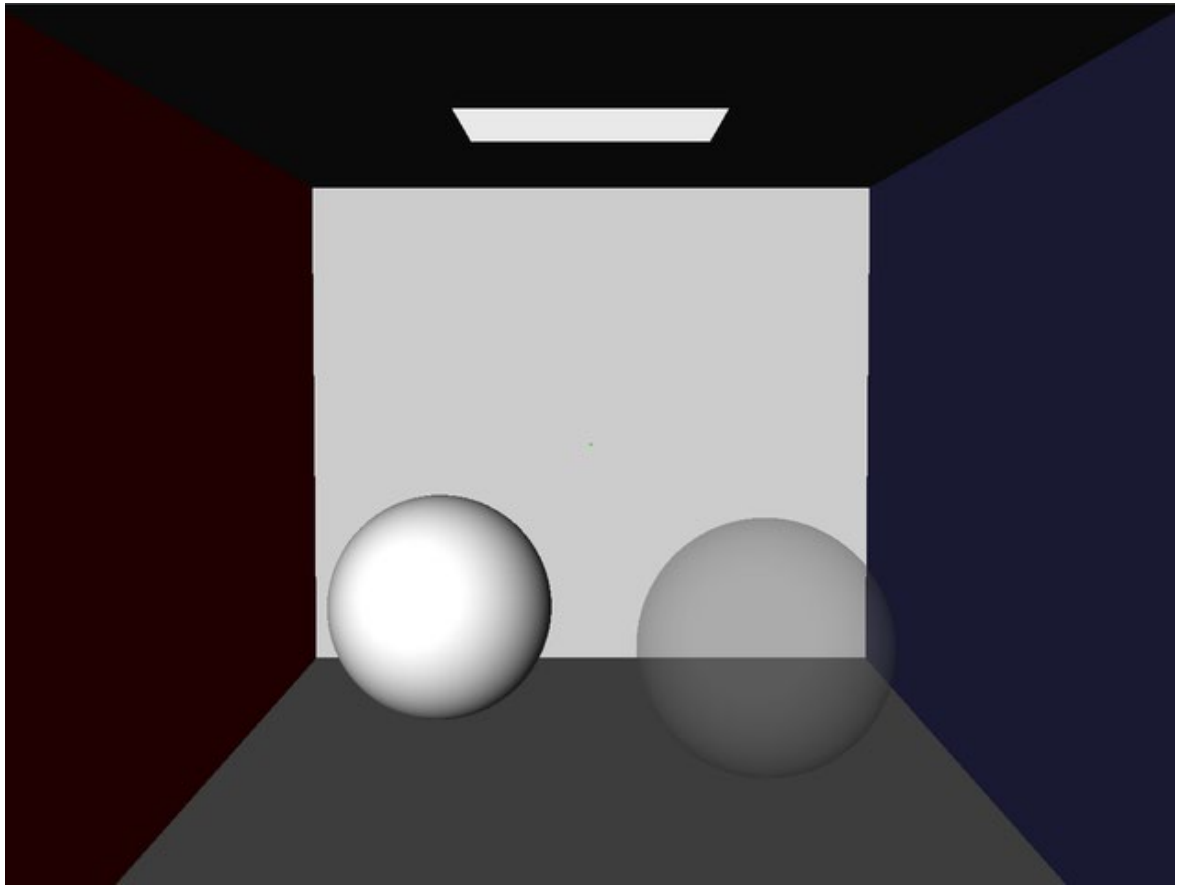
Illumination models

- Global illumination
  - Takes into account light coming from source lights (direct illumination), as well as light reflected by other surfaces (indirect illumination).
  - Takes into account shadow, reflection, refraction.
  - Algorithms:
    - Ray tracing
    - Path tracing
    - Radiosity

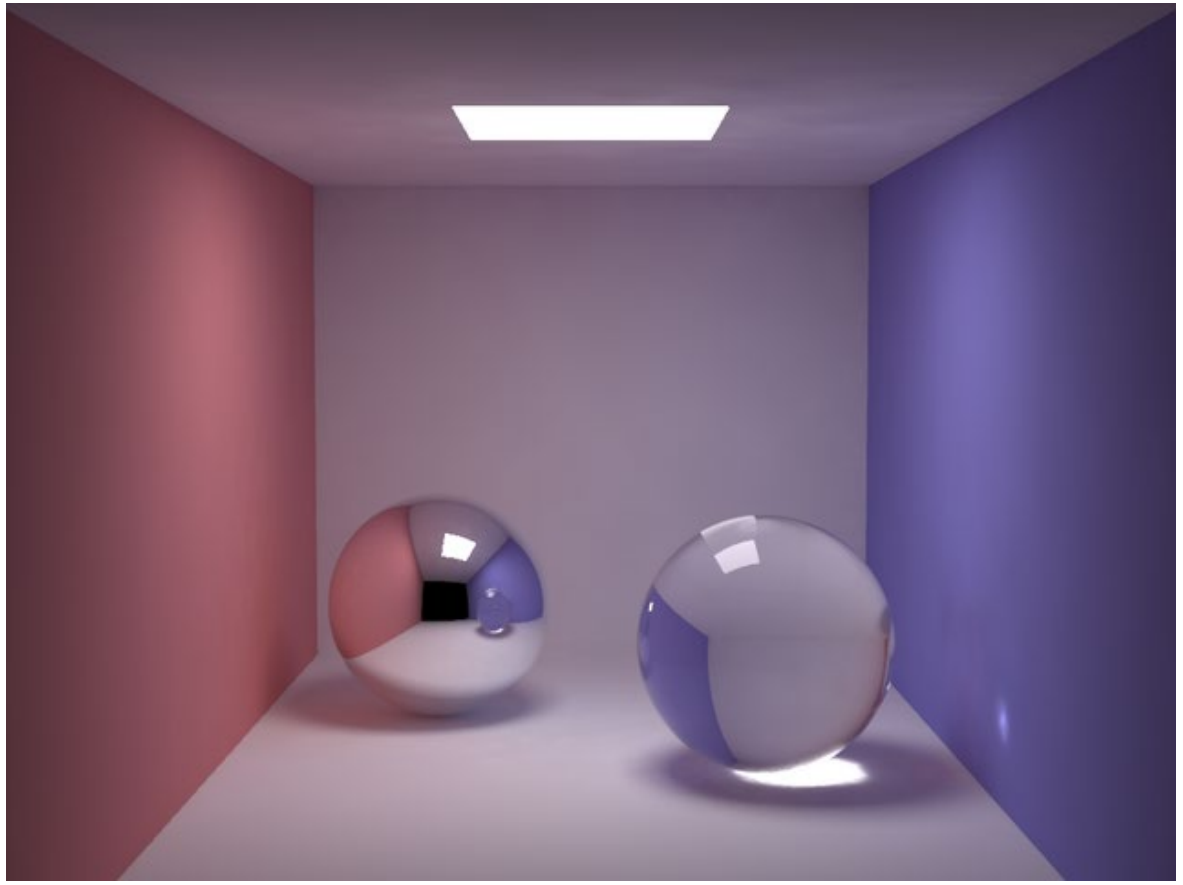
Illumination models

- Local illumination
  - Takes into account light from light sources.
  - Does not consider other objects in the scene (illumination at a point depends only of its own properties).
  - Shadow, reflection, and refraction handled by additional methods.

Illumination models

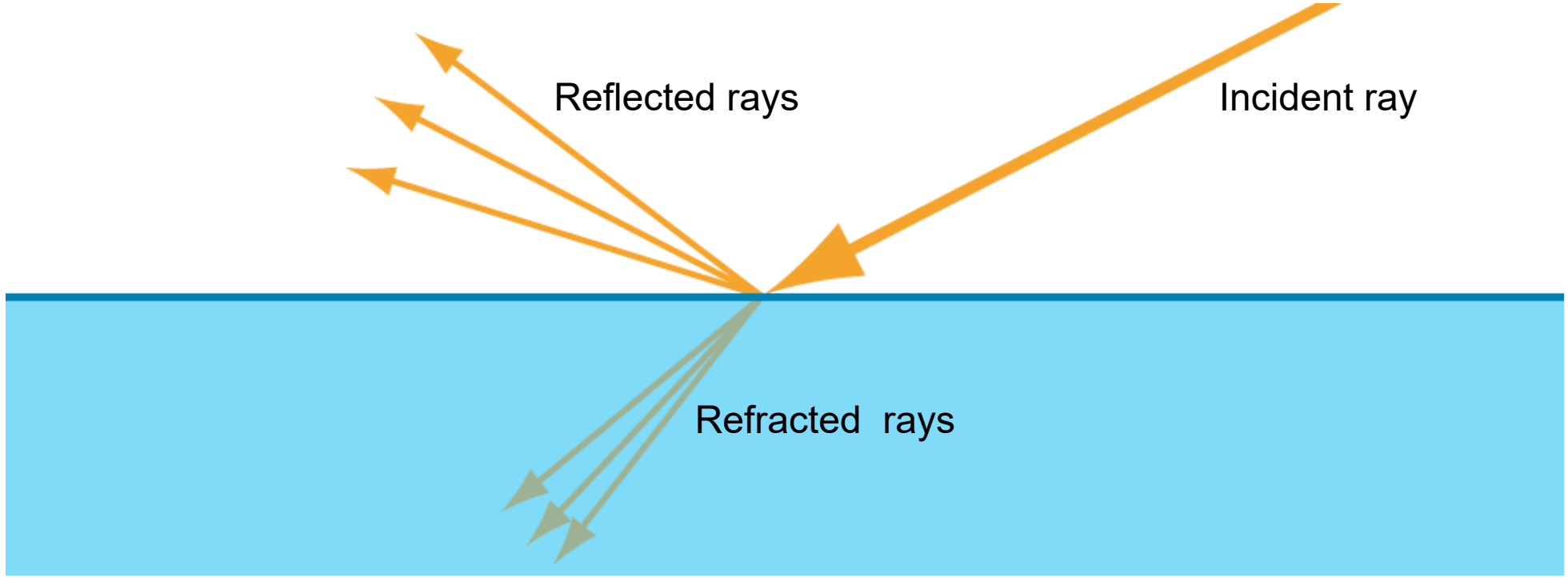


Local illumination

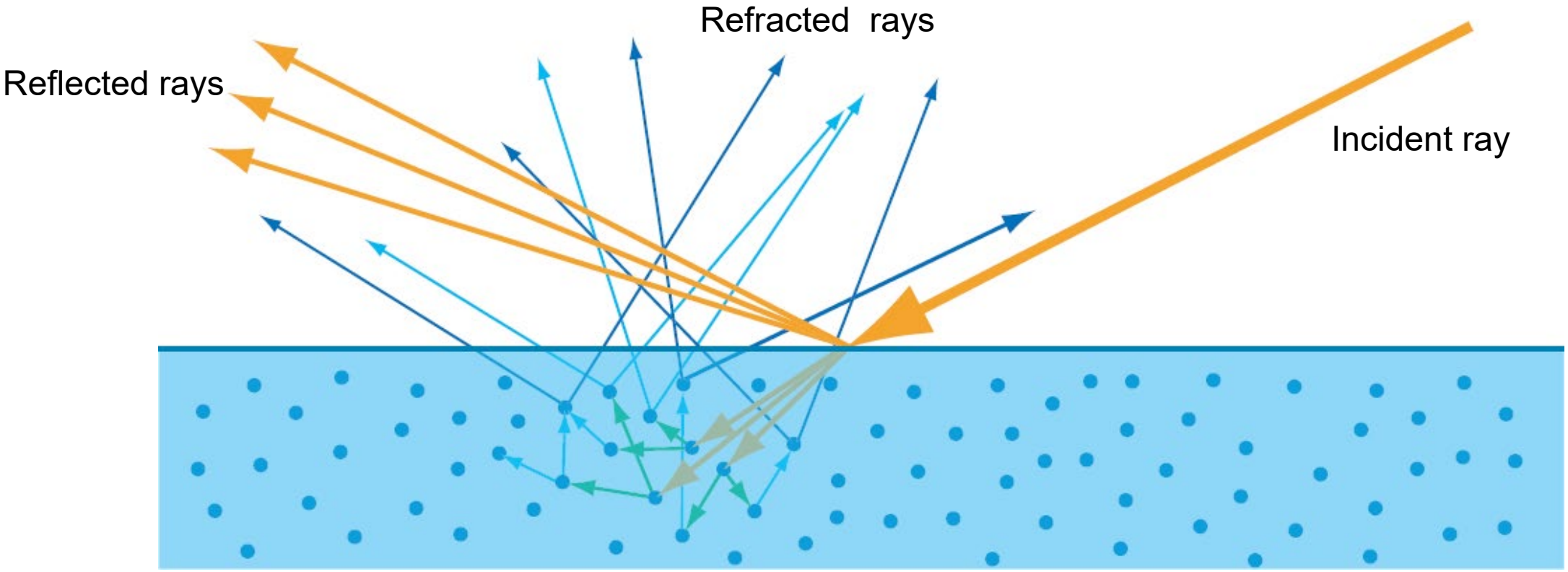


Global illumination

Light-material interaction



Light-material interaction

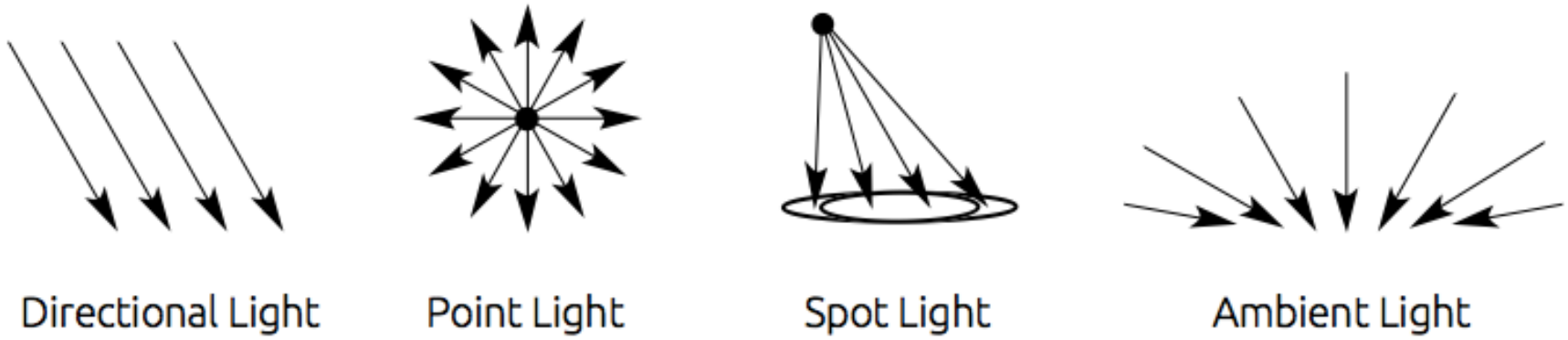


Local illumination

- Light-material interaction is modeled through two components:
- Specular component
  - Reflected rays
- Diffuse component
  - Refracted rays
- Ambient component

Light sources

- Point: position
- Spotlight: position and direction
- Directional: direction

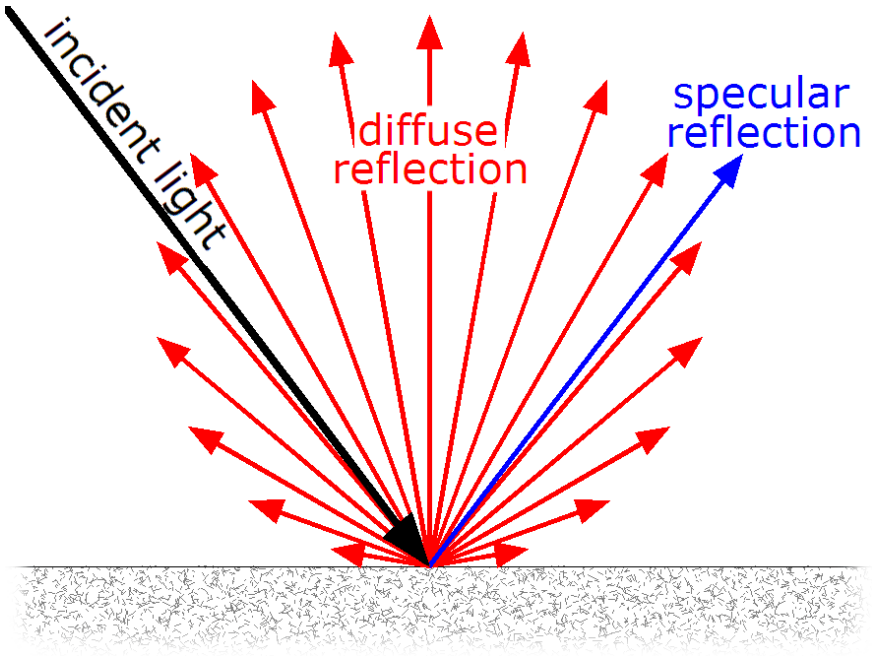


truvis.ch

Phong model

- Computes the local illumination of points on a surface.
- No physical basis. Reflection model.
- Combination of diffuse reflection with specular reflection.
- Rendering equation approximation:
  - Considers direct illumination from light sources
  - Indirect illumination mostly ignored
- Developed by Bui Tuong Phong in his 1975 Ph.D. dissertation.

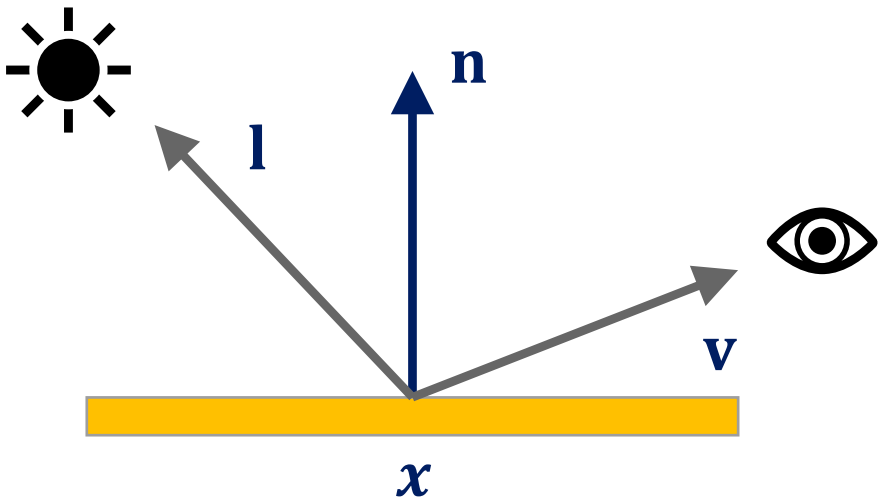
Phong model



$$L(\mathbf{x}, \mathbf{v}) = f_{Phong}(L_{light}, \mathbf{p}, \mathbf{l}, \mathbf{v}, \mathbf{n})$$

- f_{Phong} computes reflected light into direction $\mathbf{v}$ towards the sensor.

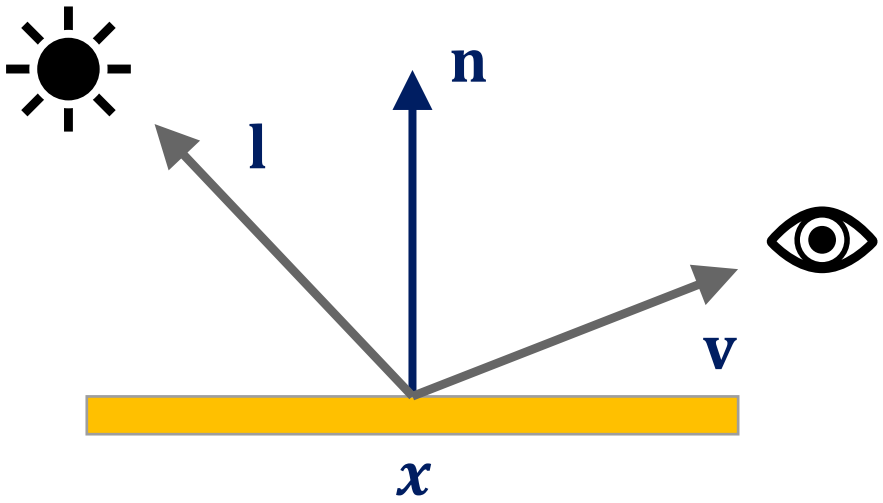
Phong model



$$L(\mathbf{x}, \mathbf{v}) = f_{Phong}(\mathbf{L}_{light}, \mathbf{p}, \mathbf{l}, \mathbf{v}, \mathbf{n})$$

- f_{Phong} computes reflected light into direction $\mathbf{v}$ towards the sensor.

Phong model

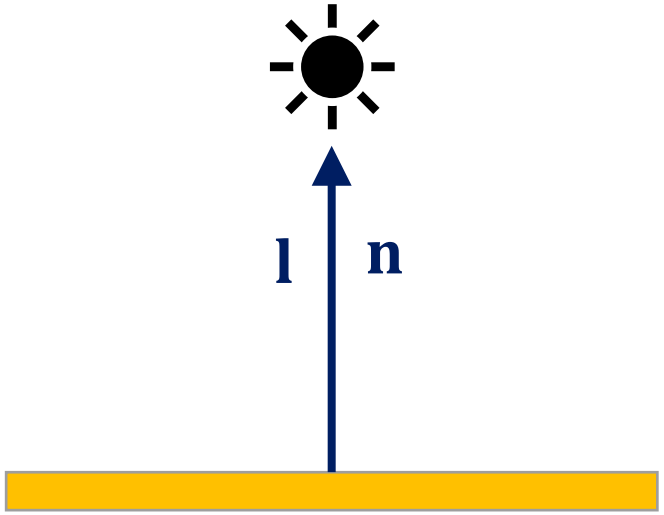


$$L(\mathbf{x}, \mathbf{v}) = f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n})$$

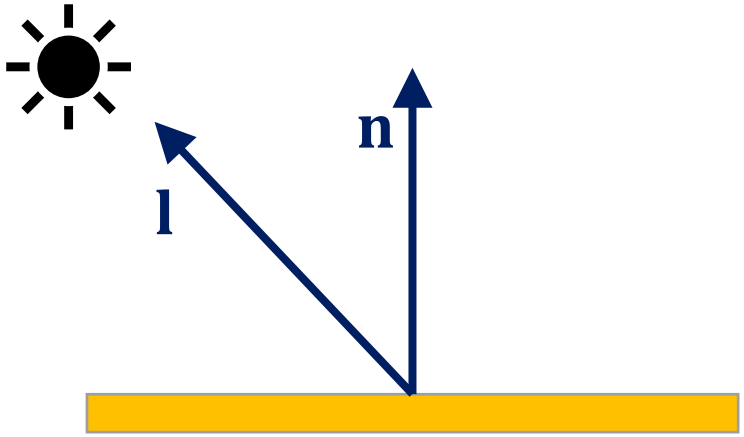
- Light L_{light} is emitted by a source, with some color and intensity.
- $L_{diffuse}$ and $L_{specular}$
  - Depends on angle between $\mathbf{l}$ and $\mathbf{n}$.
  - Depends of material.

Surface illumination

- Angle between surface normal and light surface direction influences the surface brightness.



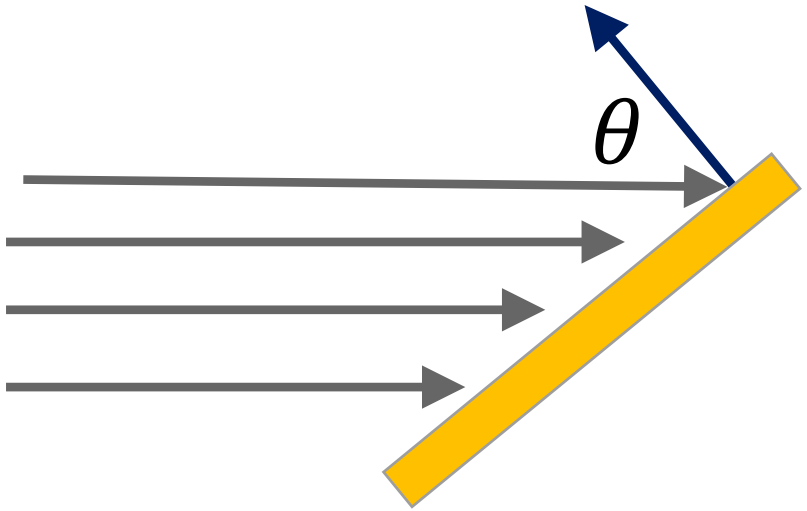
Surface receives more light per area, appears brighter.



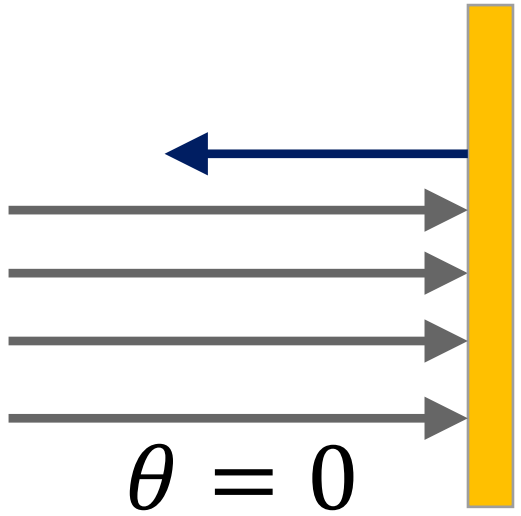
Surface receives less light per area, appears darker.

Lambert's cosine law

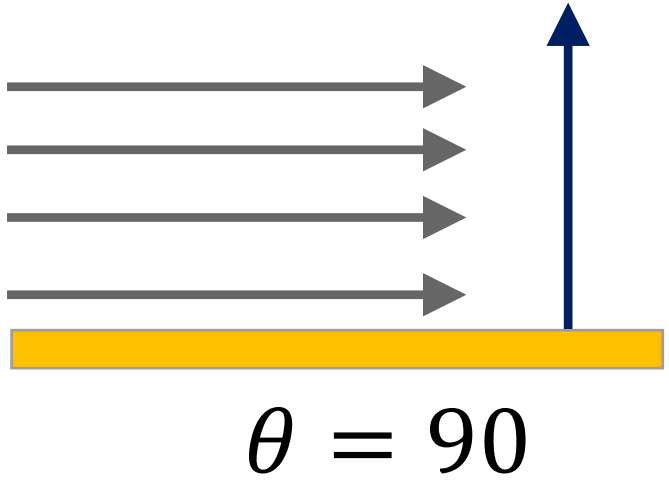
- Amount of light energy arriving at a surface is proportional to the cosine of the angle between the light direction and the surface normal.



$\mathbf{L}_{surface} = \mathbf{L}_{light} cos(\theta)$

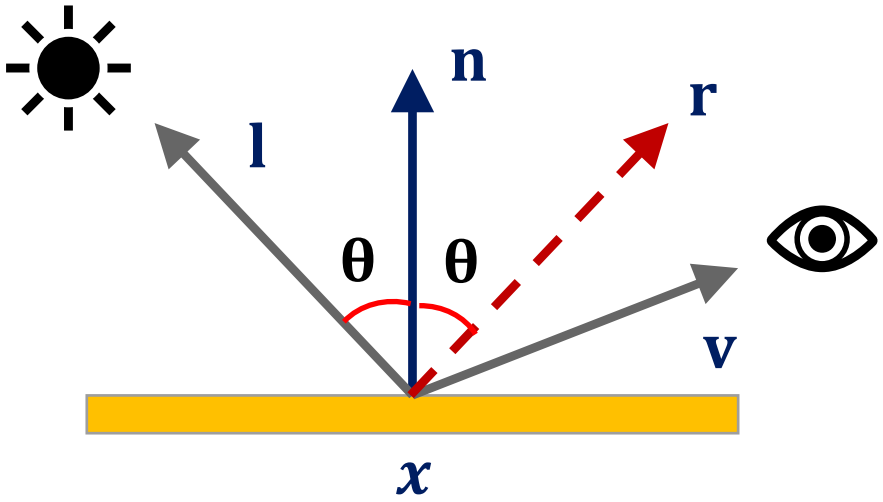


$\mathbf{L}_{surface} = \mathbf{L}_{light} cos(\theta) = \mathbf{L}_{light}$



$\mathbf{L}_{surface} = \mathbf{L}_{light} cos(\theta) = 0$

Law of reflection



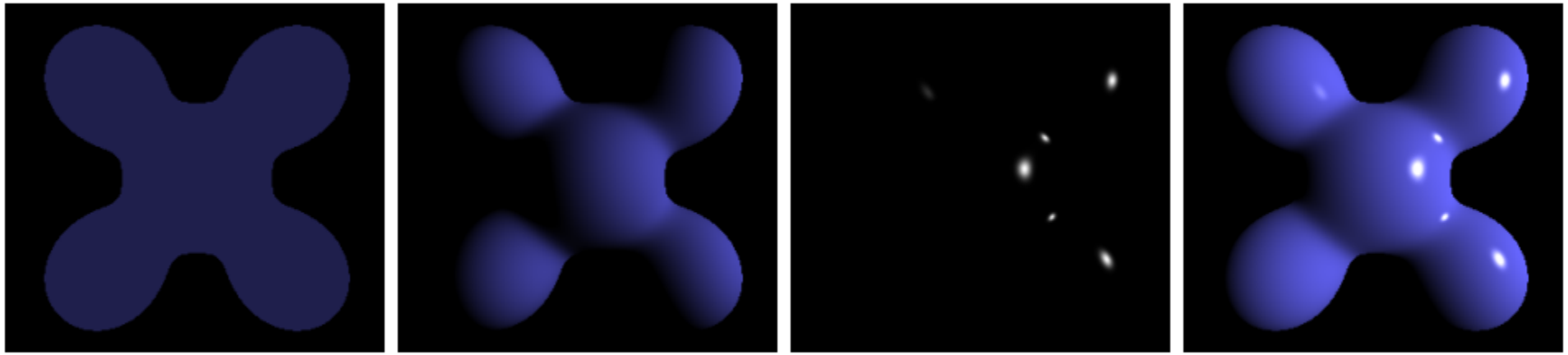
$$\hat{\mathbf{r}} + \hat{\mathbf{l}} = 2\hat{\mathbf{n}}cos\theta = 2(\hat{\mathbf{l}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}$$

$$\hat{\mathbf{r}} = 2(\hat{\mathbf{l}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} - \hat{\mathbf{l}}$$

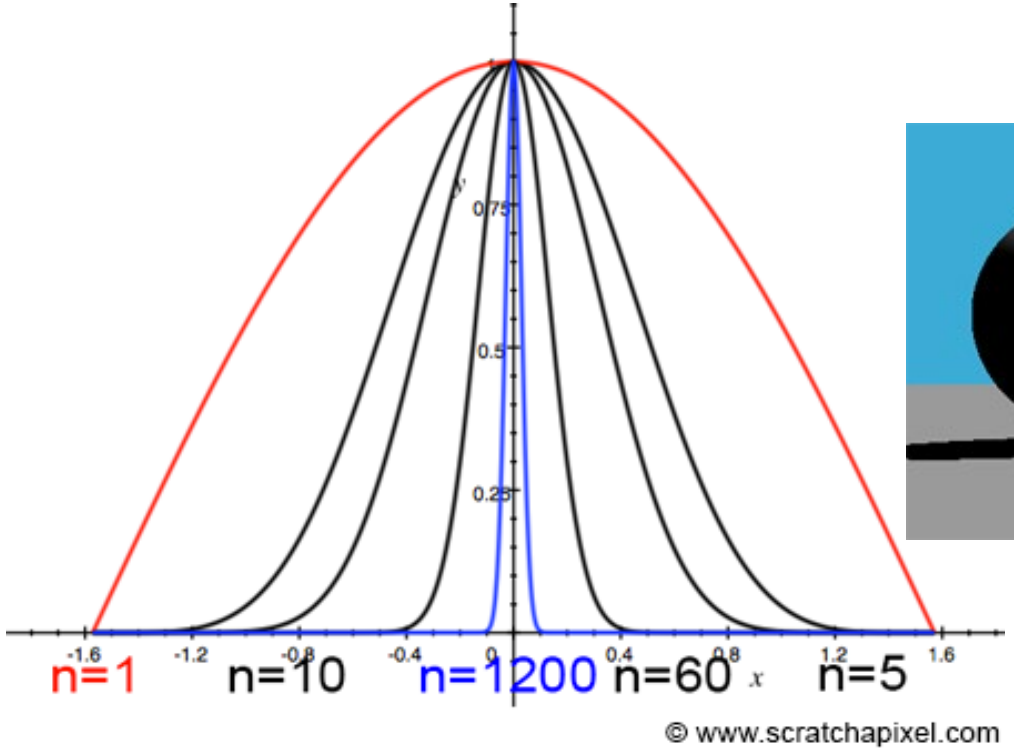
Phong model

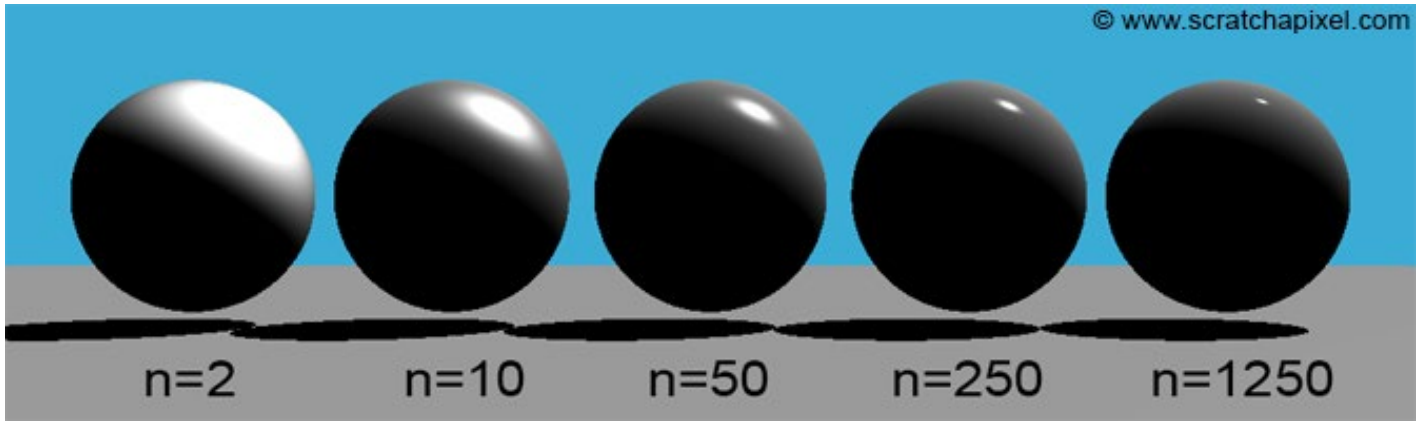
$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\mathbf{L}_{m,diffuse} + k_{specular}(\hat{\mathbf{r}}_{\boldsymbol{m}} \cdot \hat{\mathbf{v}})^{\boldsymbol{n}}\mathbf{L}_{m,specular}$$

$$\hat{\mathbf{r}}_{\boldsymbol{m}} = 2(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} - \hat{\mathbf{l}}_{\boldsymbol{m}}$$



Specular exponent



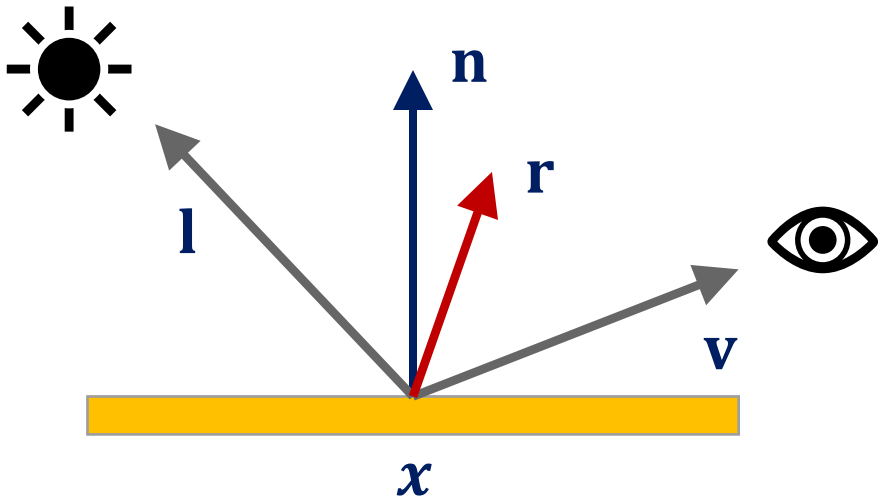


Blinn-Phong model

- A slight modification of the Phong reflection model.
- Purpose is to speed up the computation (not relevant today), and account for **v** and **r** greater than 90 degrees.
- Modification only affects specular reflection.
- Main goal: avoid computation of the reflected direction $\hat{\mathbf{r}}$.

Blinn-Phong model

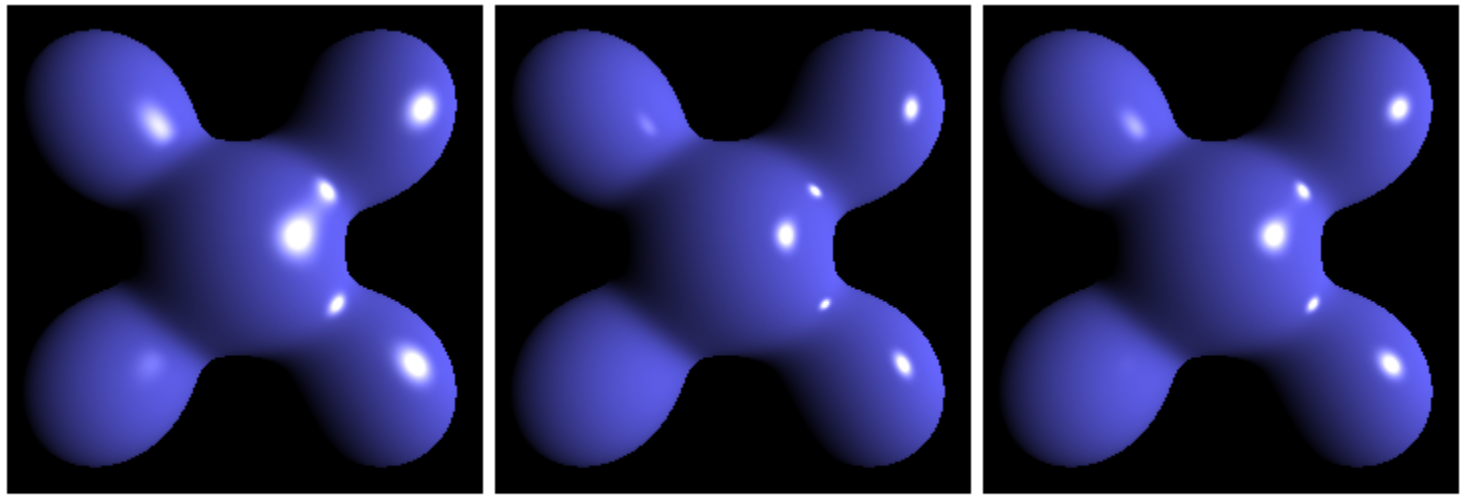
- Half-way vector:



$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{\|\mathbf{l} + \mathbf{v}\|}$$

Blinn-Phong model

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\mathbf{L}_{m,diffuse} + k_{specular}(\hat{\boldsymbol{h}}_{\boldsymbol{m}} \cdot \hat{\boldsymbol{n}})^{\boldsymbol{n}}\mathbf{L}_{m,specular}$$



Different components

Shading

- How do we compute the color for the whole surface?
- Illumination models (e.g., Phong, Blinn-Phong) compute the color of sample points. How do we color the entire object?

Shading

- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\mathbf{L}_{m,diffuse} + k_{specular}(\hat{\boldsymbol{h}}_{\boldsymbol{m}} \cdot \hat{\boldsymbol{n}})^{\boldsymbol{n}}\mathbf{L}_{m,specular}$$

  - Uniforms

Shading

- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\mathbf{L}_{m,diffuse} + k_{specular}(\hat{\boldsymbol{h}}_{\boldsymbol{m}} \cdot \hat{\boldsymbol{n}})^{\boldsymbol{n}}\mathbf{L}_{m,specular}$$

  - Uniforms
  - Normals

Shading

- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\mathbf{L}_{m,diffuse} + k_{specular}(\hat{\boldsymbol{h}}_{\boldsymbol{m}} \cdot \hat{\boldsymbol{n}})^{\boldsymbol{n}}\mathbf{L}_{m,specular}$$

  - Uniforms
  - Normals
  - Light position

Shading

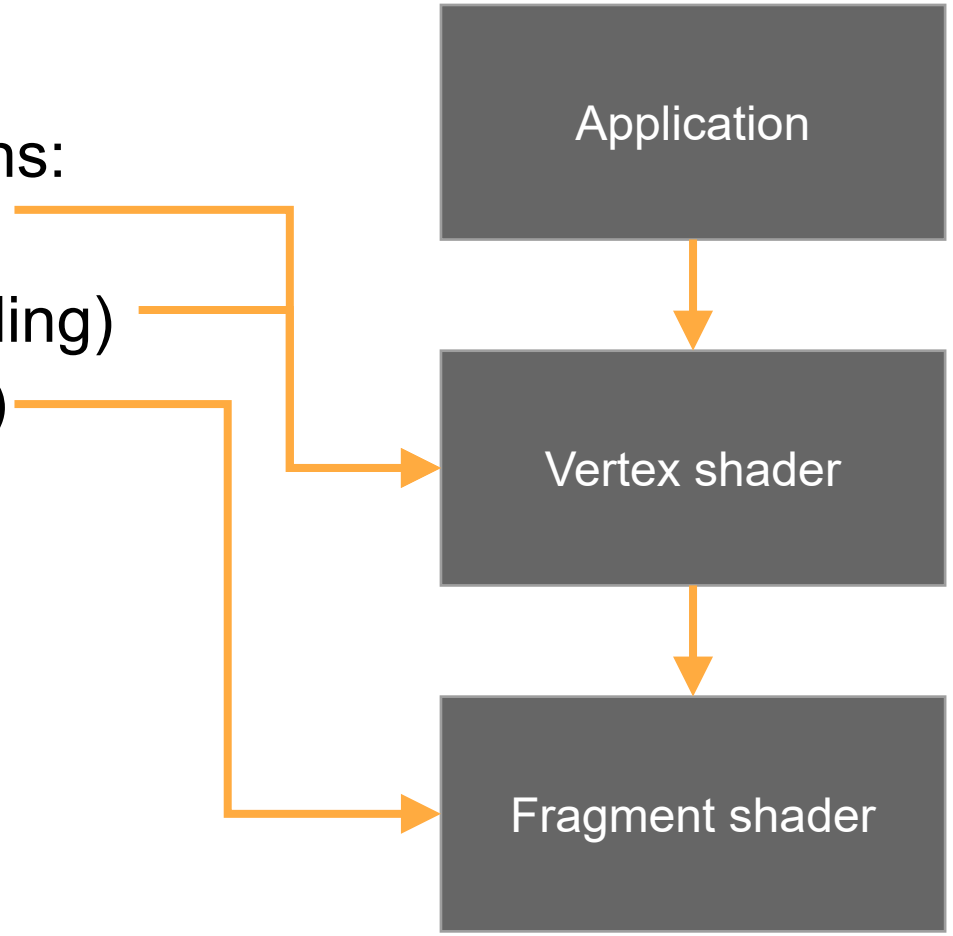
- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\mathbf{L}_{m,diffuse} + k_{specular}(\hat{\boldsymbol{h}}_{\boldsymbol{m}} \cdot \hat{\boldsymbol{n}})^{\boldsymbol{n}}\mathbf{L}_{m,specular}$$

  - Uniforms
  - Normals
  - Light position
  - Half-vector: depends on $\mathbf{l}$ and $\mathbf{v}$
    - Do we really need to send $\mathbf{v}$? We can work on camera space!

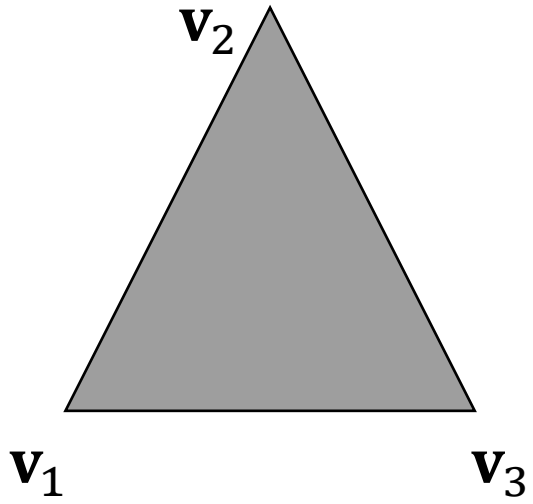
Shading: per-triangle, per-vertex, per-pixel

- We can compute shading operations:
  - Once per triangle (flat shading)
  - Once per vertex (Gouraud shading)
  - Once per pixel (Phong shading)

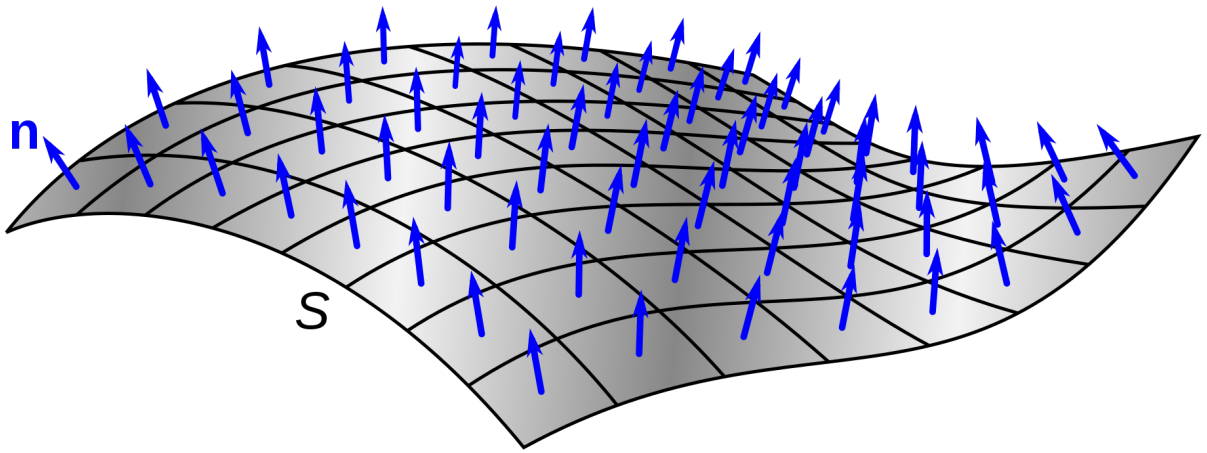


Computing normals

Triangle with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$:

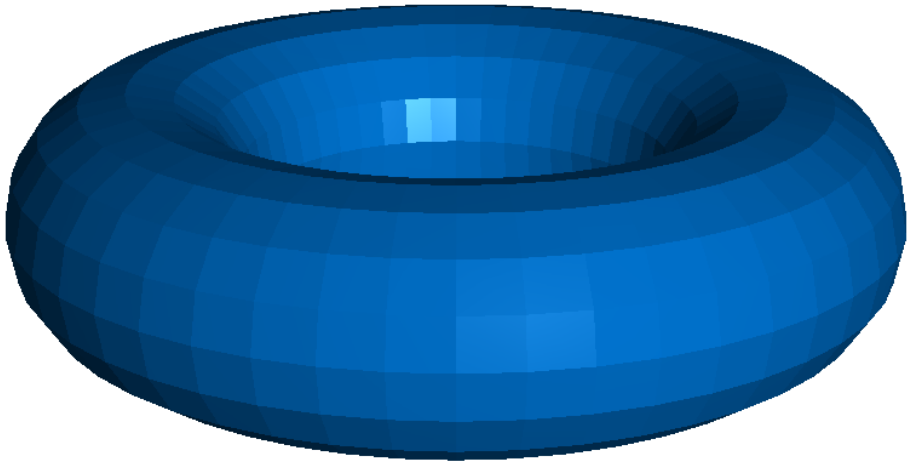


$$\mathbf{n} = (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)$$



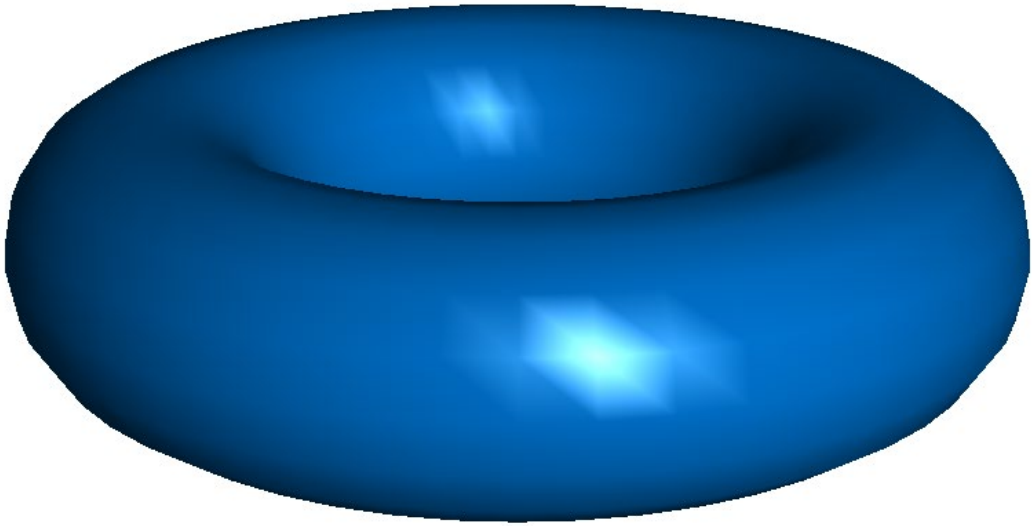
Flat shading

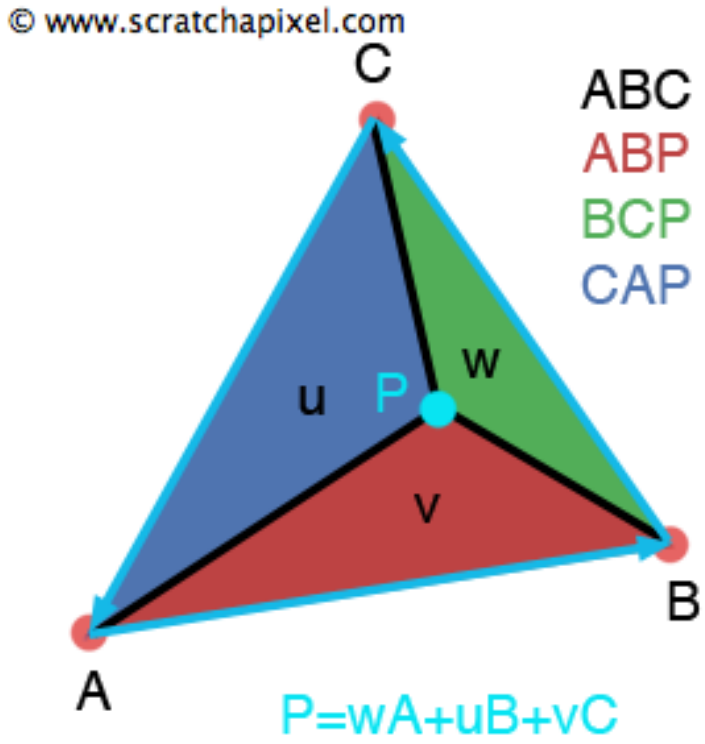
- Compute one color per polygon.
- All pixels in the same polygon are colored by the same color.



Gouraud shading

- Compute one color per vertex.
- Interpolate vertex **colors** across triangles.





Gouraud shading + WebGL

What uniforms in the vertex shader?

```
uniform float kAmbient;
uniform float kDiffuse;
uniform float kSpecular;
uniform float specExponent;
uniform vec3 colorAmbient;
uniform vec3 colorDiffuse;
uniform vec3 colorSpecular;
```

```
uniform vec3 lightPos; // or lightDir
```

What attributes in the vertex shader?

```
in vec3 pos;
in vec4 color;
in vec3 normal;
```

One normal per vertex.

Vertex shader:

- Blinn-Phong operations, send color result to fragment shader.

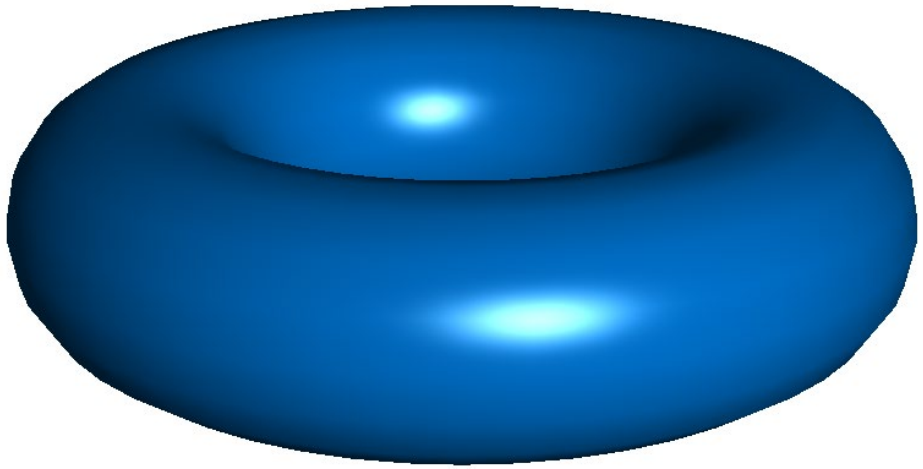
What about eye pos?

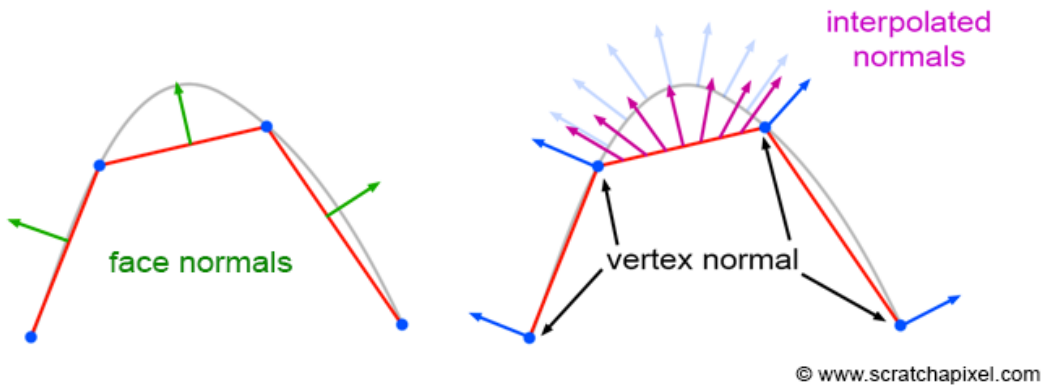
- No, if we perform everything in camera space.

```
posEyeSpace = modelViewMatrix * pos;
```

Phong shading

- One color per pixel.
- Interpolates vertex **normals** across triangles.
- Illumination model evaluated at each pixel.





Phong shading + WebGL

What uniforms in the frag shader?

```
uniform float kAmbient;
uniform float kDiffuse;
uniform float kSpecular;
uniform float specExponent;
uniform vec3 colorAmbient;
uniform vec3 colorDiffuse;
uniform vec3 colorSpecular;
```

```
uniform vec3 lightPos; // or lightDir
```

What varying in the frag shader?

```
in vec3 position;
in vec3 normal;
```

Fragment shader:

- Blinn-Phong operations, send color result to framebuffer.

Again: camera space.

```
posEyeSpace = modelViewMatrix * pos;
```

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\hat{\mathbf{l}}_{\boldsymbol{m}} \cdot \hat{\mathbf{n}})\mathbf{L}_{m,diffuse} + k_{specular}(\hat{\boldsymbol{h}}_{\boldsymbol{m}} \cdot \hat{\boldsymbol{n}})^{\boldsymbol{n}}\mathbf{L}_{m,specular}$$

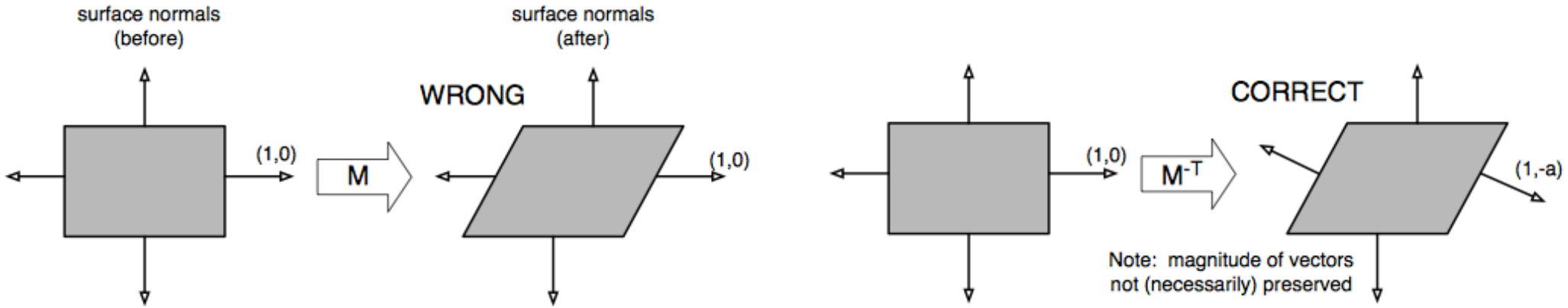
How can we transform normals?

- Vertex:

```
v = modelViewMatrix * vec4(position, 1);
```

- Normal:

```
n = uNormal * vec4(normal, 1);
```



$$\mathbf{NormalMatrix} = (\mathbf{ModelView}^{-1})^{\mathbf{T}}$$

How can we transform normals?

$\mathbf{t} = \mathbf{p_2} - \mathbf{p_1}$

$\mathbf{t} * \mathbf{M} = (\mathbf{p_2} - \mathbf{p_1}) * \mathbf{M}$

$\mathbf{t}' = \mathbf{p_2}' - \mathbf{p_1}'$

$\mathbf{n}$ and $\mathbf{t}$ are perpendicular

$\mathbf{n}'$and $\mathbf{t}'$must also be perpendicular

We find normal matrix by solving: $\mathbf{n}'.\mathbf{t}' = \mathbf{0}$

What is $\mathbf{n}'$? $\mathbf{G} * \mathbf{n}$

What is $\mathbf{t}'$? $\mathbf{M} * \mathbf{t}$

$(\mathbf{G} * \mathbf{n}).(\mathbf{M} * \mathbf{t}) = \mathbf{0}$

$\mathbf{G} = (\mathbf{M}^{-1})^{\mathbf{T}}$

Point and directional light

Directional Light

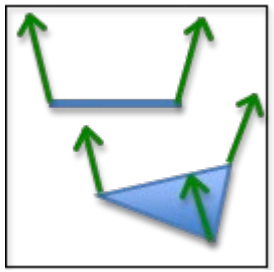
Point Light

```
uniform vec3 lightDir;
```

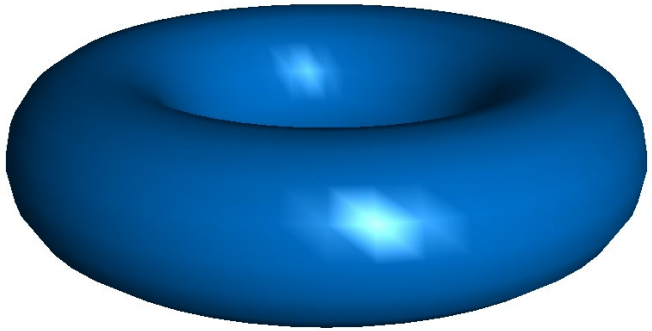
```
lightDir = normalize(lightPos - pos);
```

Comparison

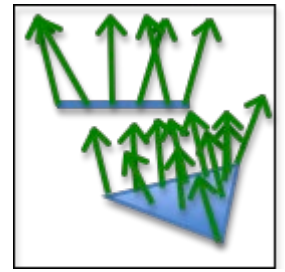
Gouraud shading



Vertex shader → color → Fragment shader



Phong shading



Vertex shader → normal position → Fragment shader