

GRK 5

dr Wojciech Palubicki

Simplified Rendering Pipeline

Model Matrix



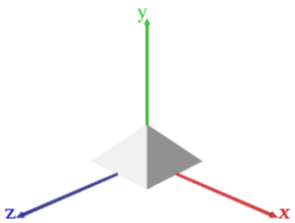
View Matrix



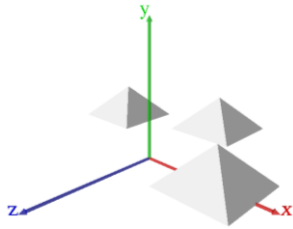
Projection Matrix



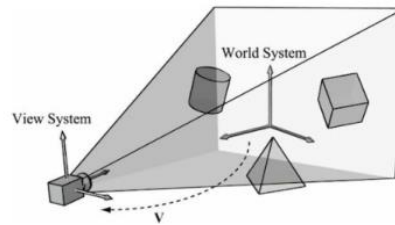
Viewport Transform



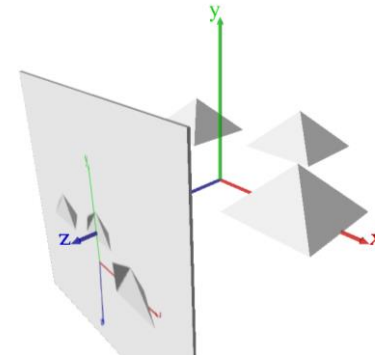
Object Space



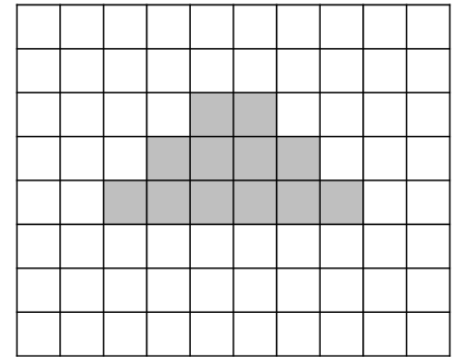
World Space



View Space



Clip Space



Screen Space

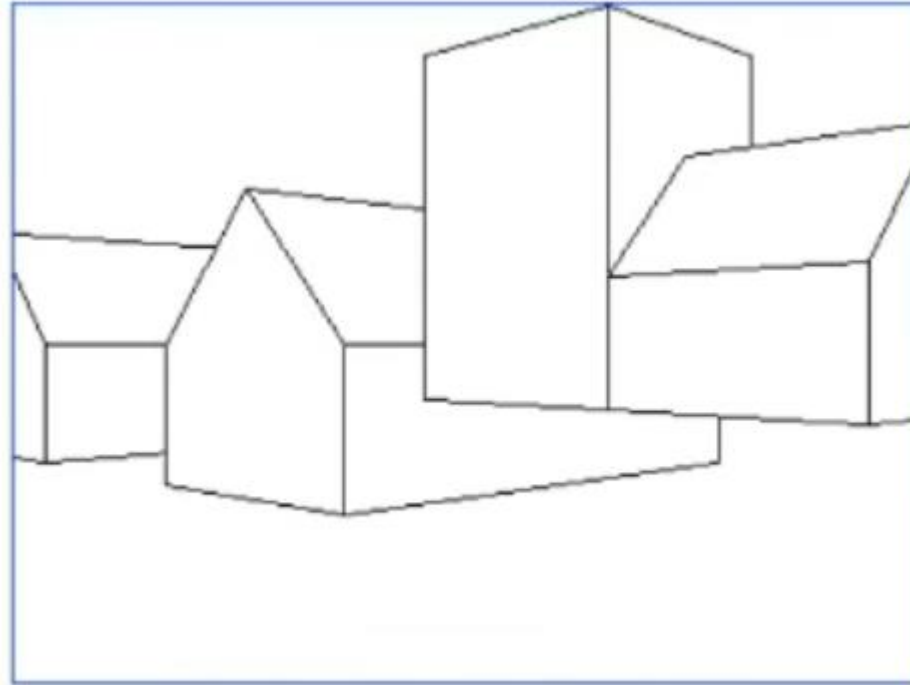
Drawing on the Display

- The following pseudo-code illustrates how to draw a line between two points a and b

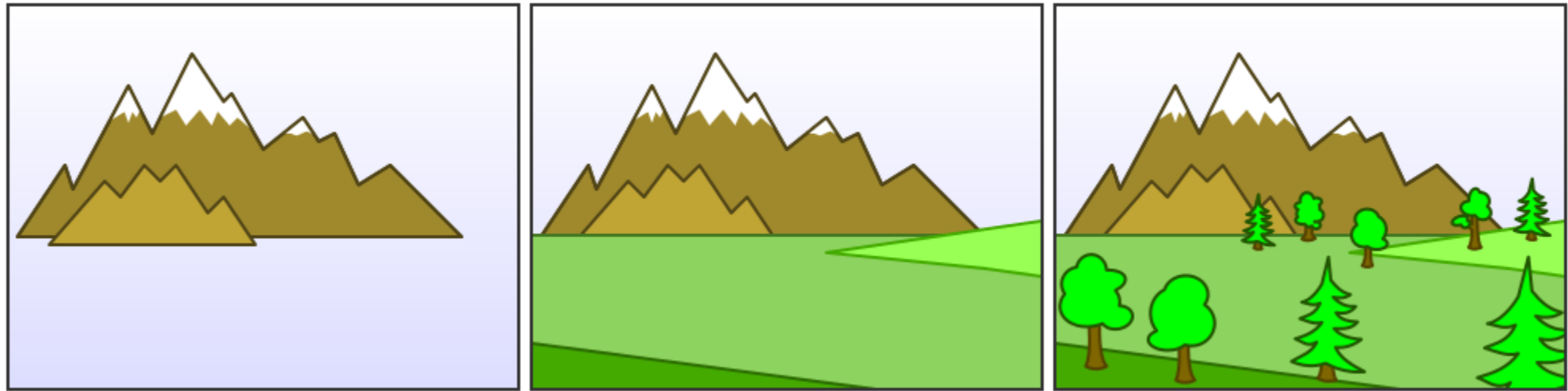
```
compute  $M_{vp}$ 
compute  $M_{per}$ 
compute  $M_{cam}$ 
 $M = M_{vp} M_{per} M_{cam}$ 

for each line segment  $(a, b)$  do
     $p = M a$ 
     $q = M b$ 
    drawline( $p_x/p_w, p_y/p_w, q_x/q_w, q_y/q_w$ )
```

Drawing Order Problem



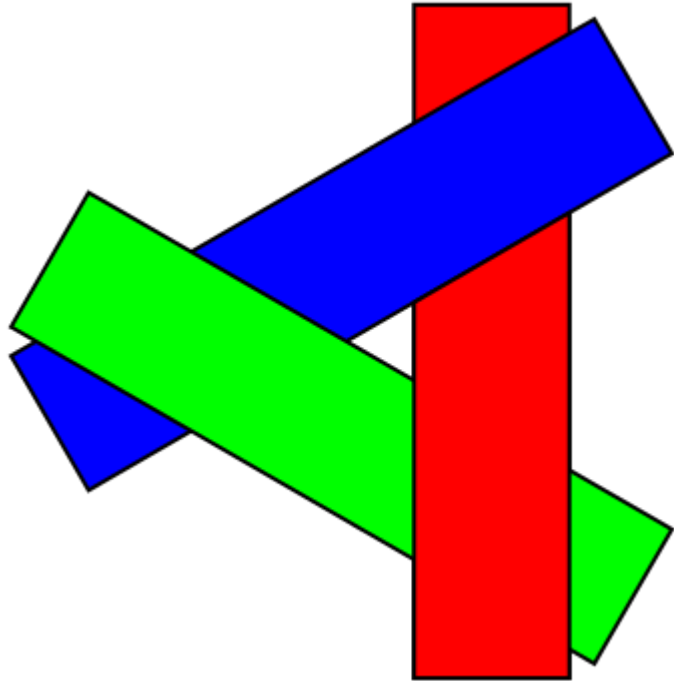
Painter's algorithm



Painter's algorithm

- Algorithm draws polygons in relation to distance from camera
- Smallest coordinate z_s of all polygons is used to determine the distance
- Closer polygons are drawn on top of further ones

Partial polygon overlay



z-buffer / depth buffer

- Z-buffer algorithm is similar to painter's algorithm but at the scale of pixels
- Two buffers (arrays) are created where each pixel corresponds to one element of the array
- **depth buffer** stores distance **z** from the nearest surface in world space for each pixel
- **frame buffer** stores indices of polygons (to later on select corresponding colors)

z-buffer / depth buffer

Algorithm 1 Z buffer

Require: a set of polygons P , a depth buffer array Z and a frame buffer array F
initialise Z to z_{\max}
for all polygons in P **do**
 for all pixels in the current polygon **do**
 calculate the z co-ordinate of the point corresponding to the current pixel
 if $z < Z(x, y)$ **then**
 Replace $Z(x, y)$ with z
 Replace $F(x, y)$ with the colour of the current polygon
 end if
 end for
end for
Display F on screen

∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Z buffer

Frame buffer

∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	1	2	3	4	5	6	7	∞	∞
∞	1	2	3	4	5	6	∞	∞	∞
∞	1	2	3	4	5	∞	∞	∞	∞
∞	1	2	3	4	∞	∞	∞	∞	∞
∞	1	2	3	∞	∞	∞	∞	∞	∞
∞	1	2	∞	∞	∞	∞	∞	∞	∞
∞	1	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Z buffer

Frame buffer

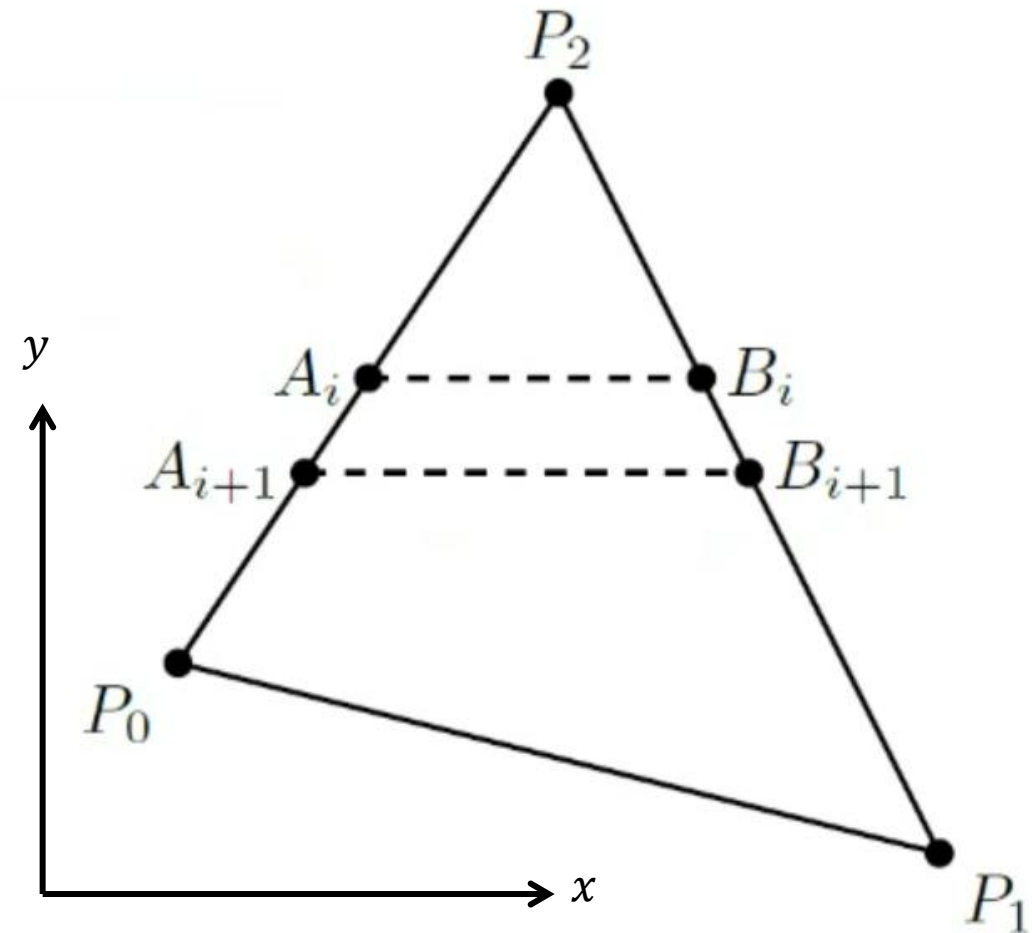
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	1	2	3	4	5	6	7	∞	∞
∞	1	2	3	4	4	3	2	1	∞
∞	1	2	3	4	4	3	2	1	∞
∞	1	2	3	4	4	3	2	1	∞
∞	1	2	3	5	4	3	2	1	∞
∞	1	2	6	5	4	3	2	1	∞
∞	1	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Z buffer

Frame buffer

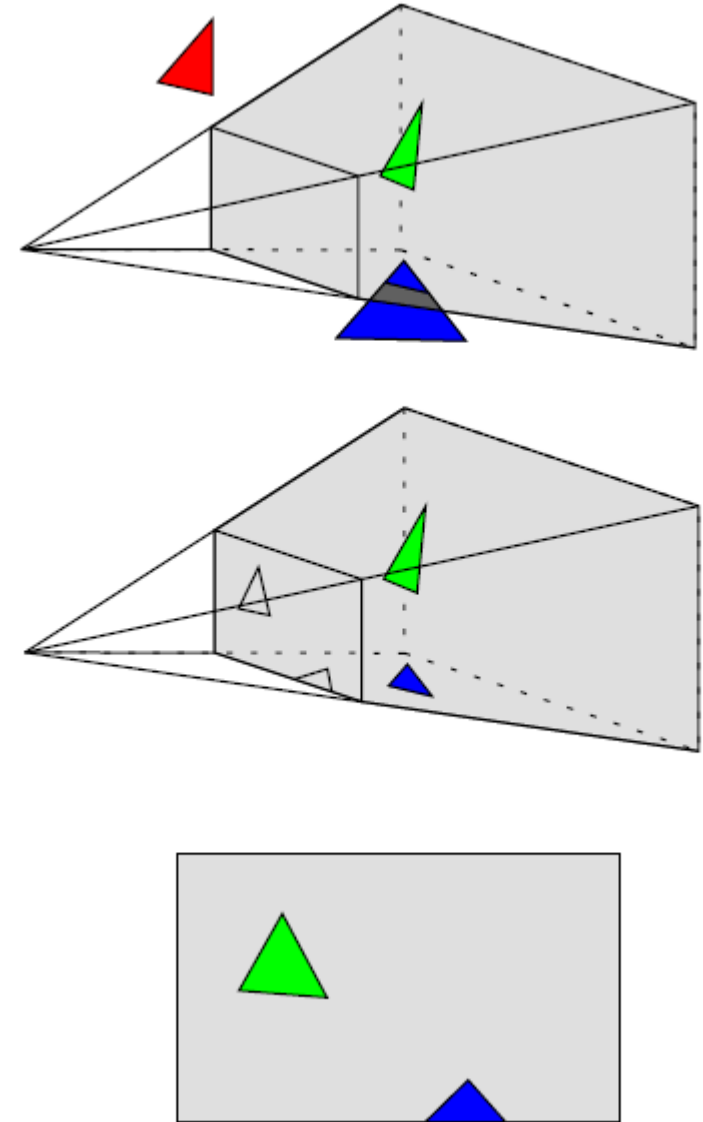
Interpolating between vertices P

- Loop through horizontal rows of pixels (**scan lines**) from top to bottom
- Calculate pixels on the left and right of scan line A and B by **interpolation**
- Calculate the pixels on the scan line by **interpolating** between A and B



Clipping/Culling

- Triangles that lie (partly) outside of the view frustum don't have to be projected and are culled (**clipping/culling**)
- The remaining triangles are projected on the view plane



Lighting



Without lighting

Lighting



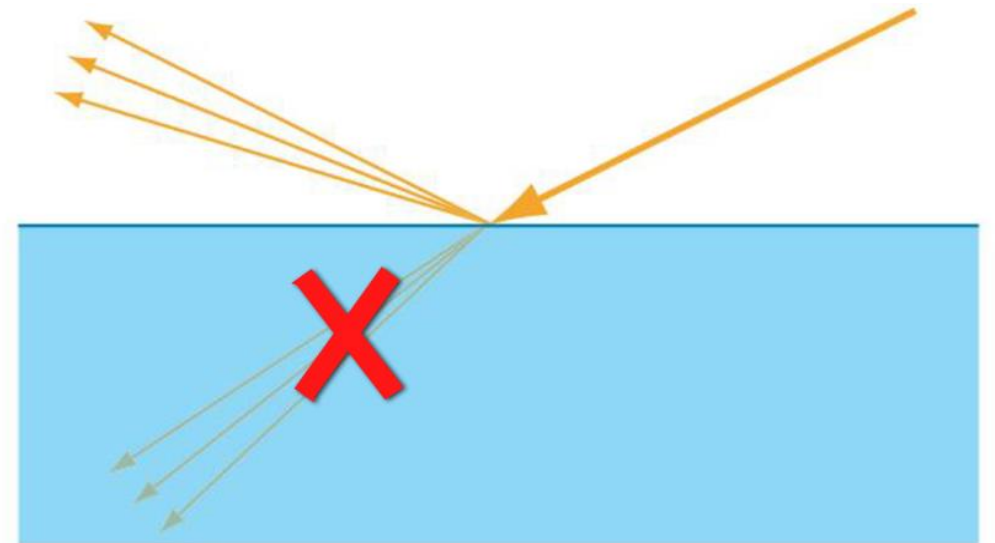
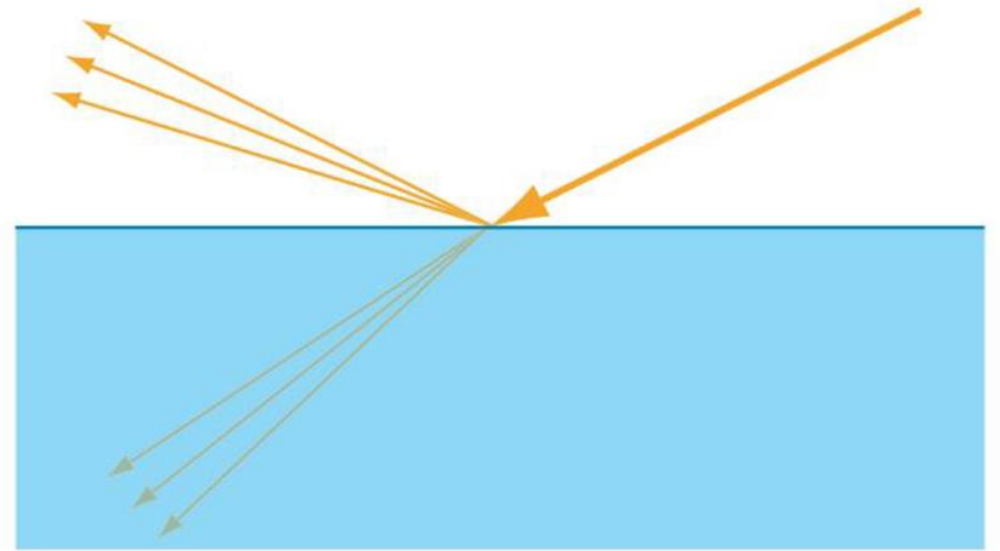
Without lighting



With lighting

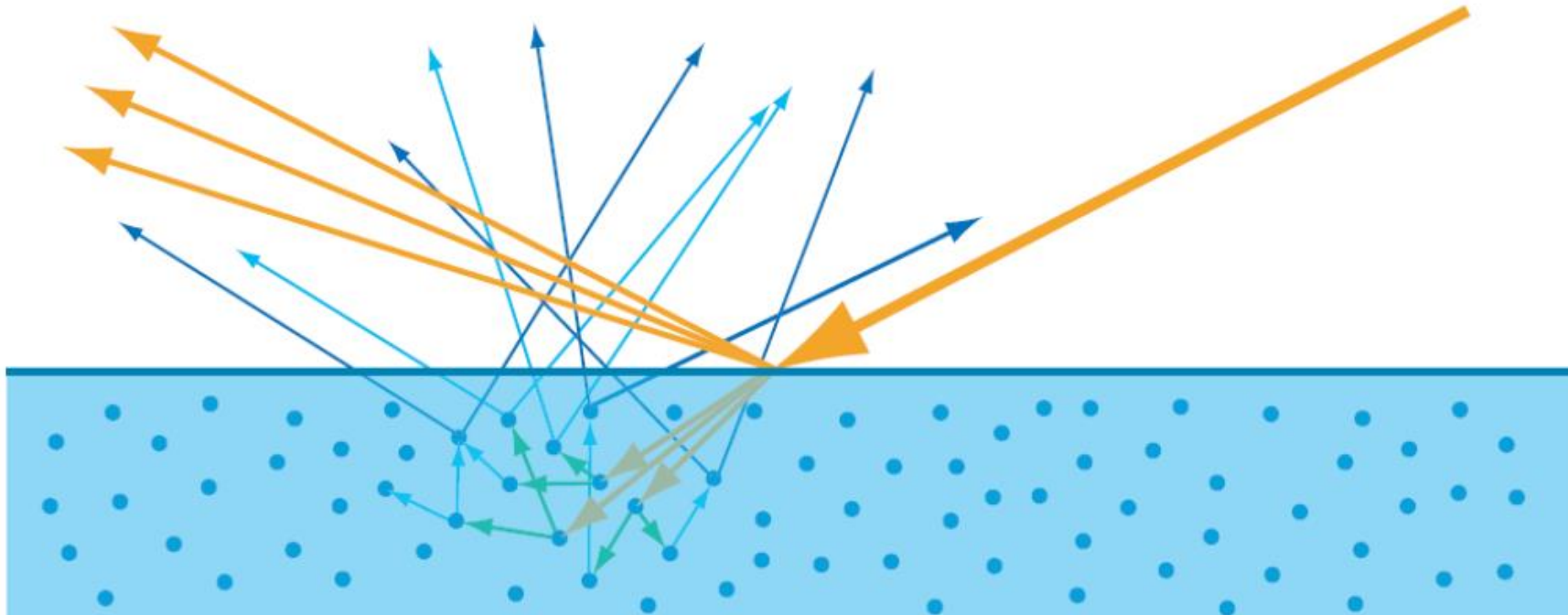
Light

- Non-metal material
 - A portion of light is reflected
 - Another portion enters the material
- Metal
 - A portion of light is reflected
 - Another portion is absorbed



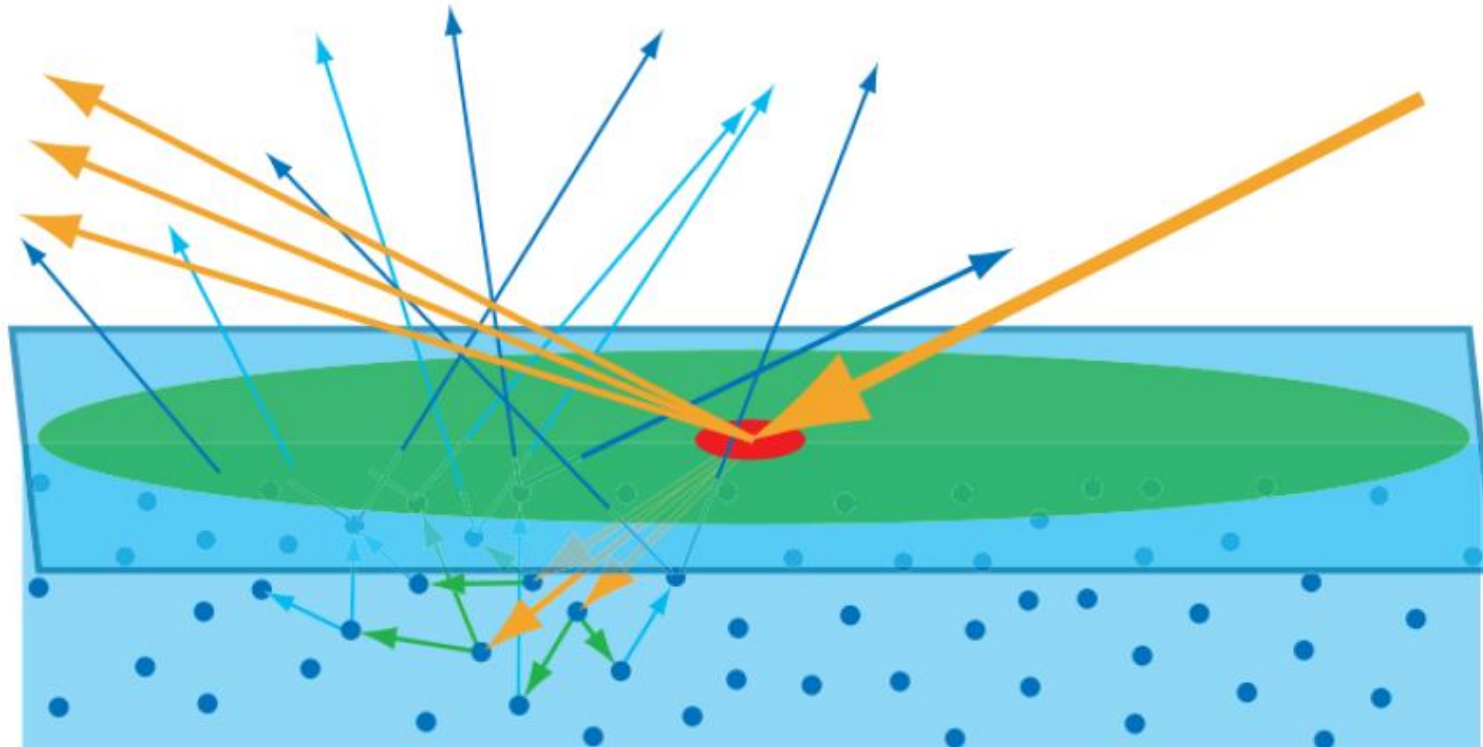
Non-metal materials

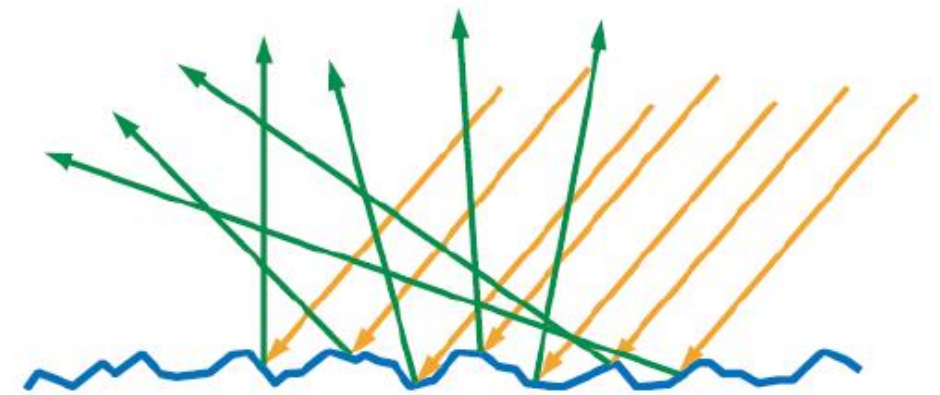
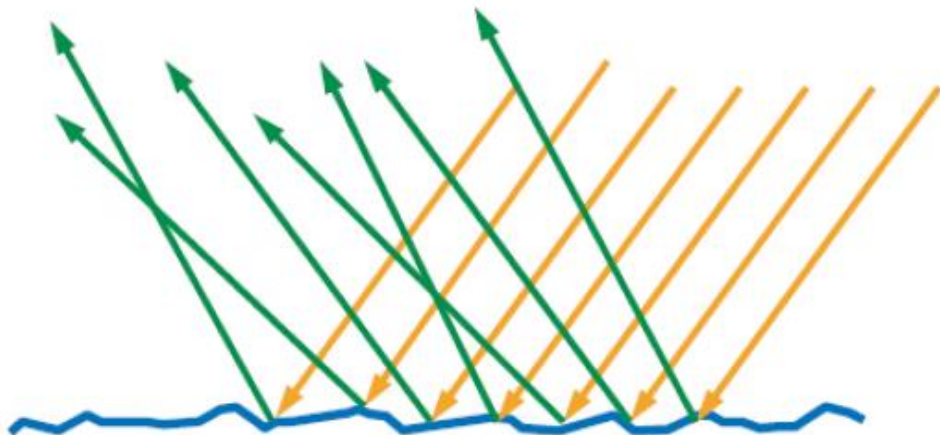
- Light that is entering the material
 - Is absorbed
 - Or re-emitted after a while



Subsurface scattering

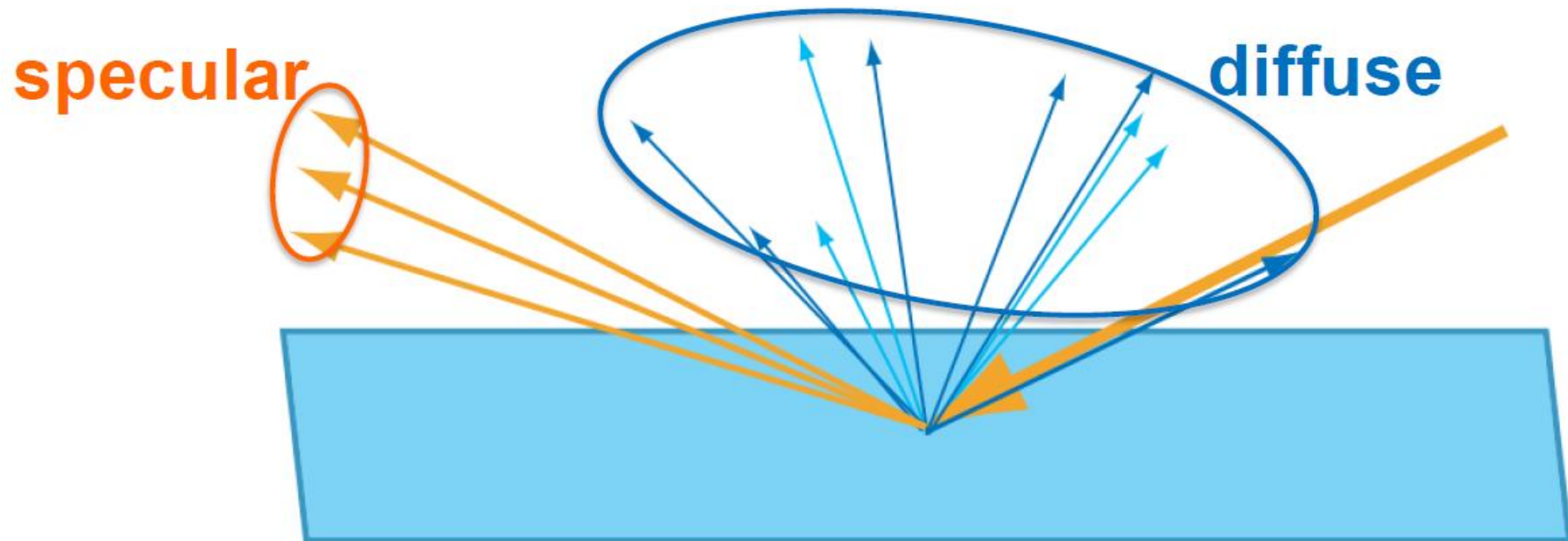
- The distance between entry and exit points of light rays is specified by the material

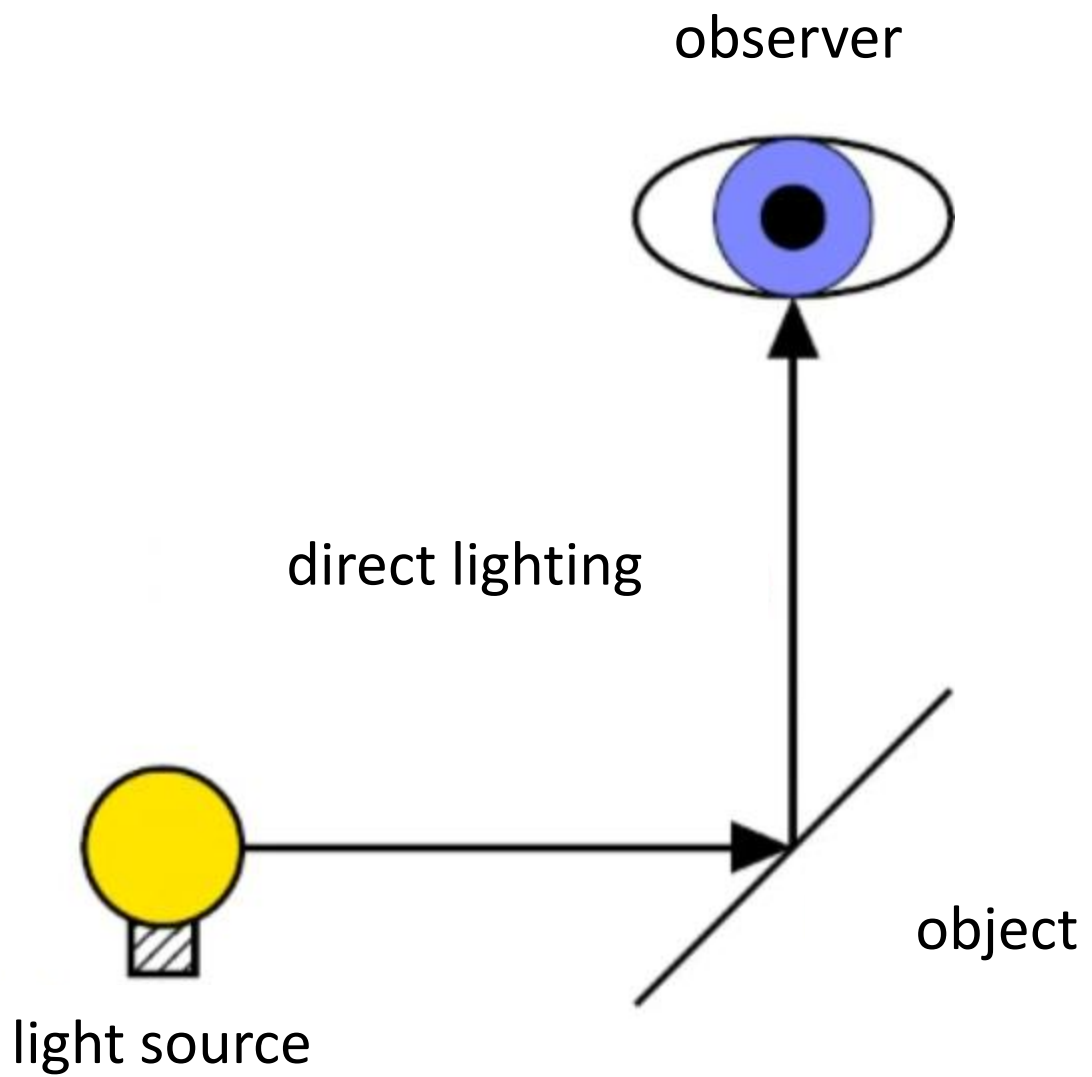


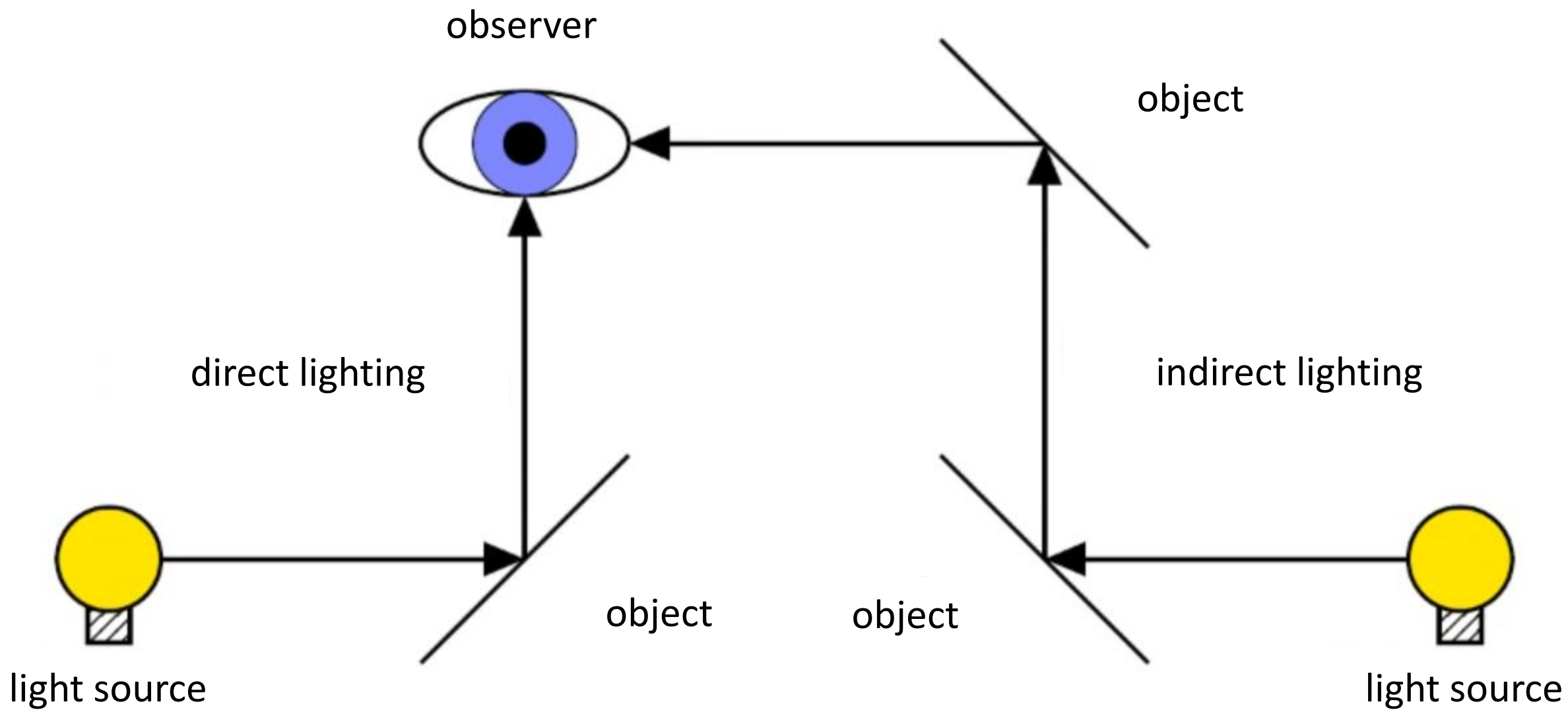


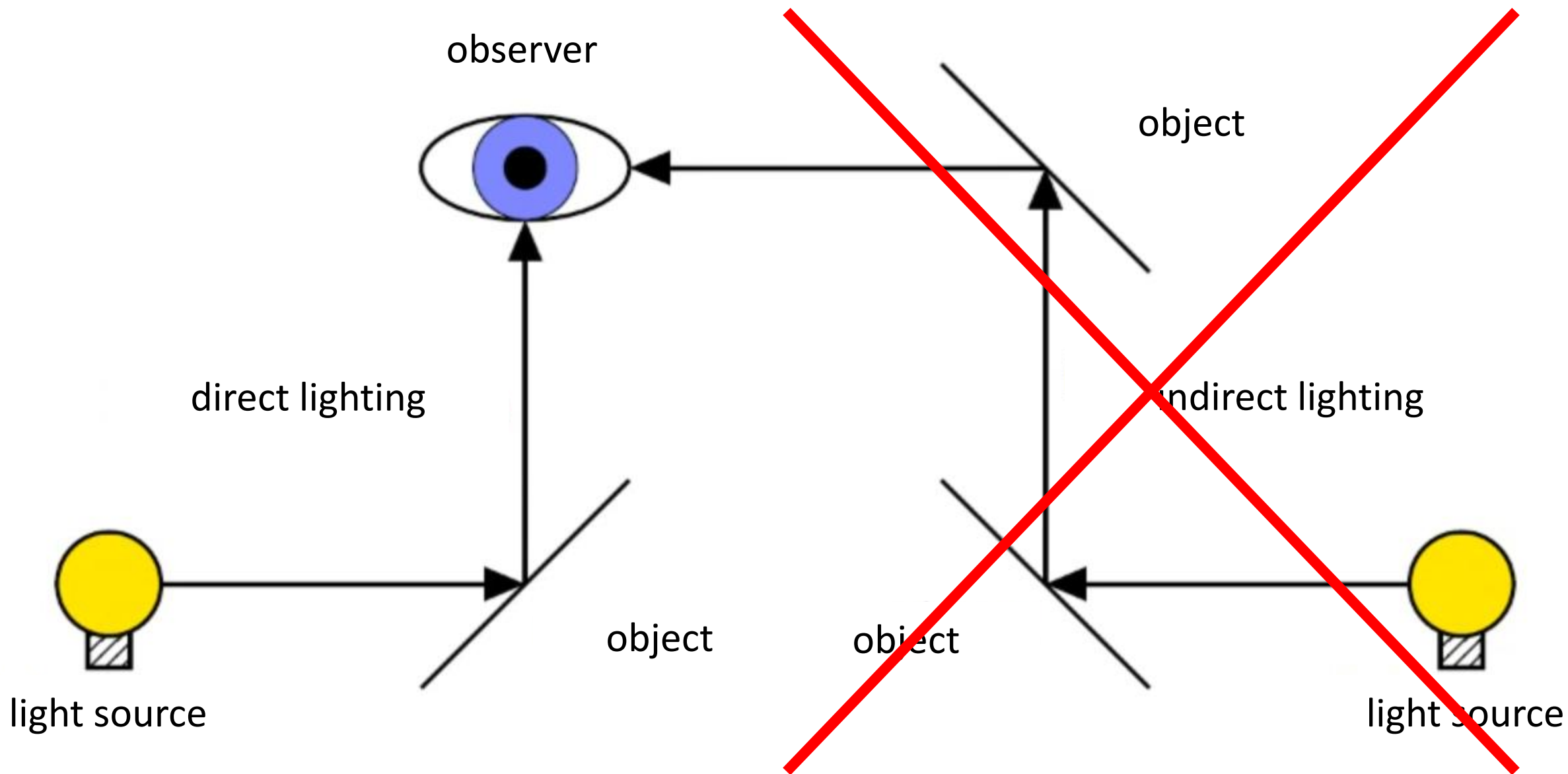
Modeling Light with BRDF's

- BRDF stands for *Bidirectional Reflectance Distribution Function*
- BRDF's are modeling light by ignoring the difference between entry and exit points of a material

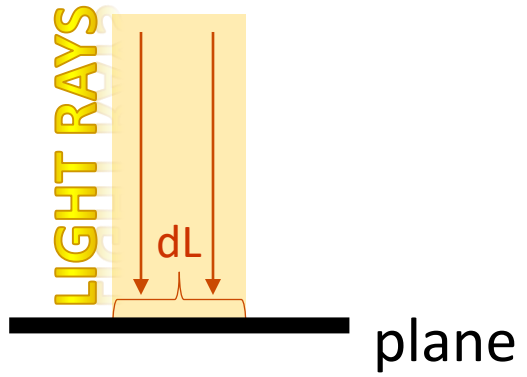




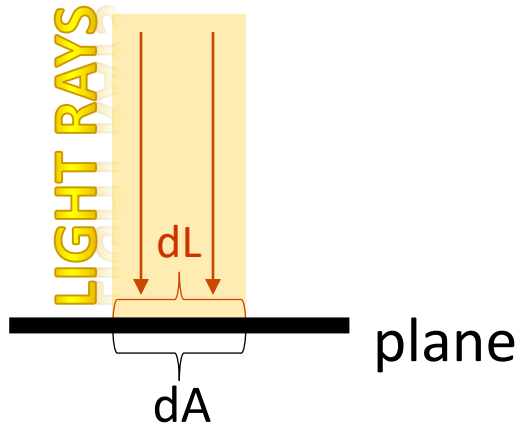




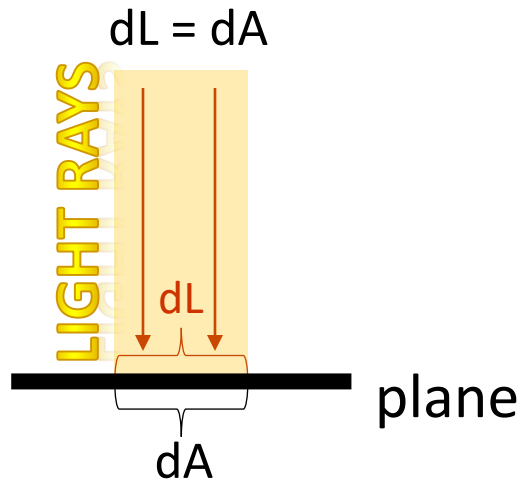
Observation



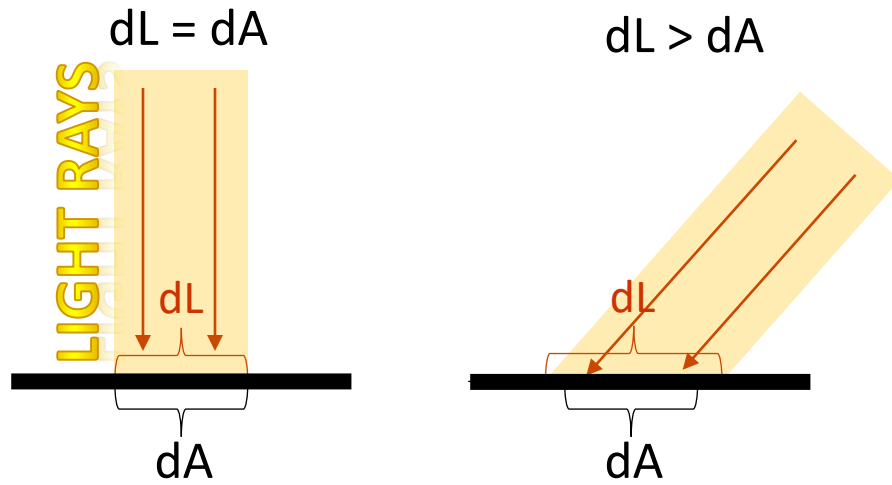
Observation



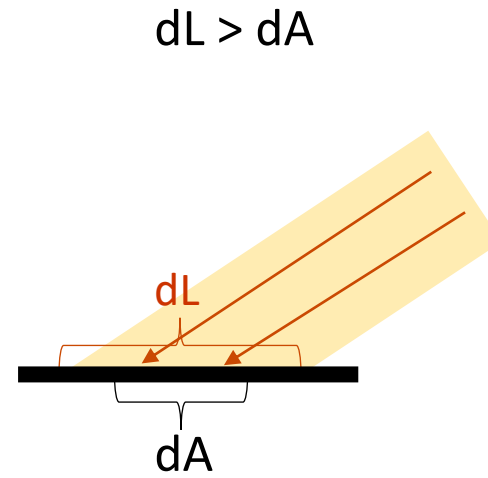
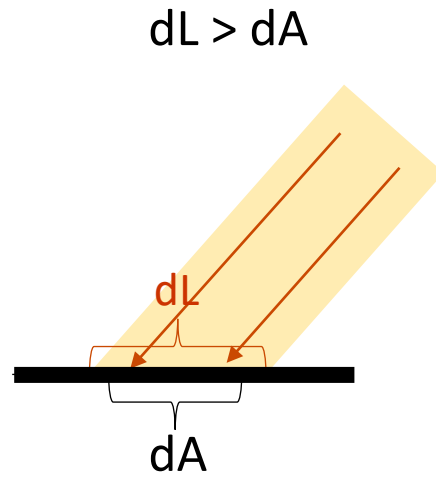
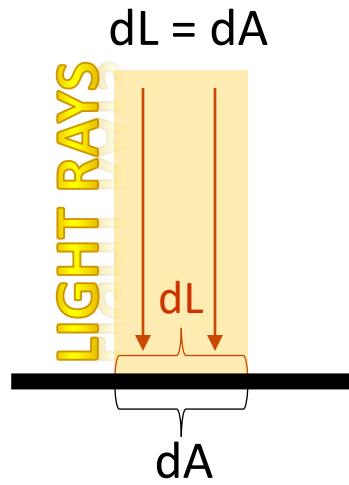
Observation



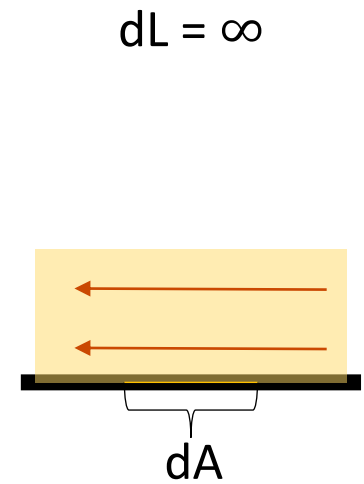
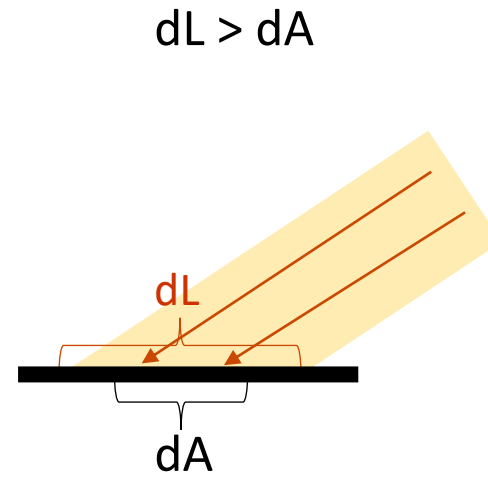
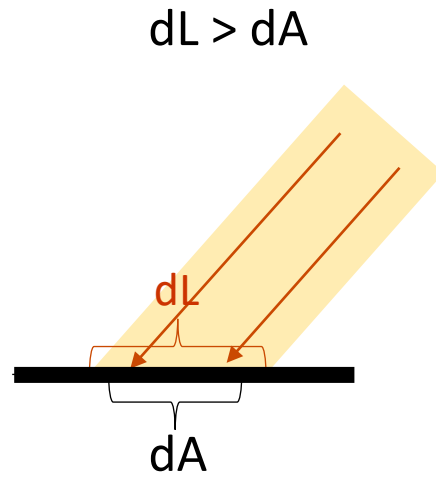
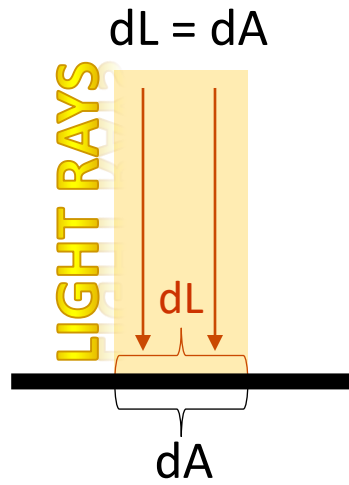
Observation



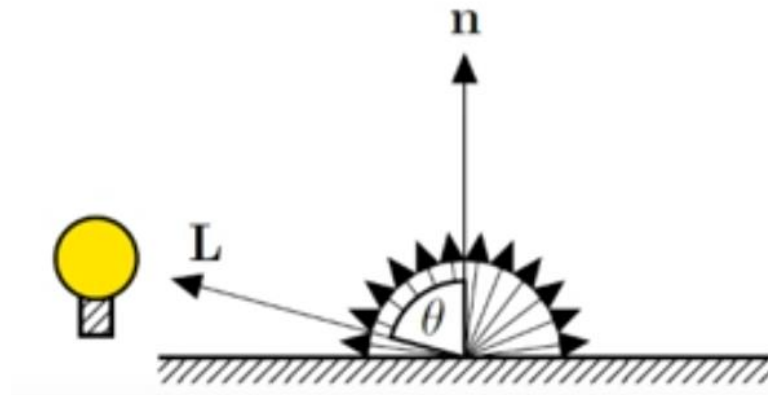
Observation



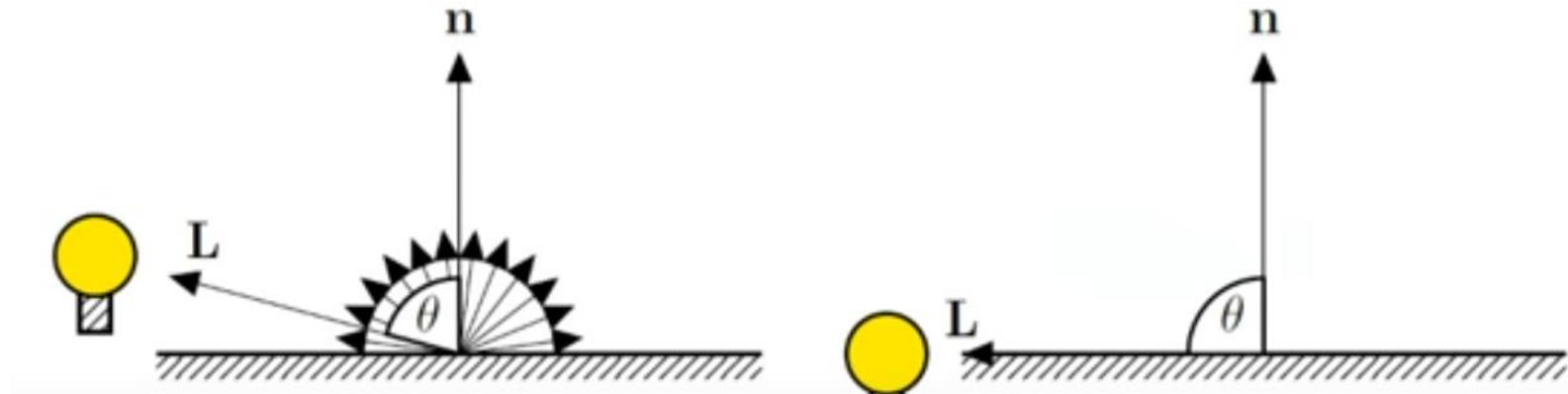
Observation



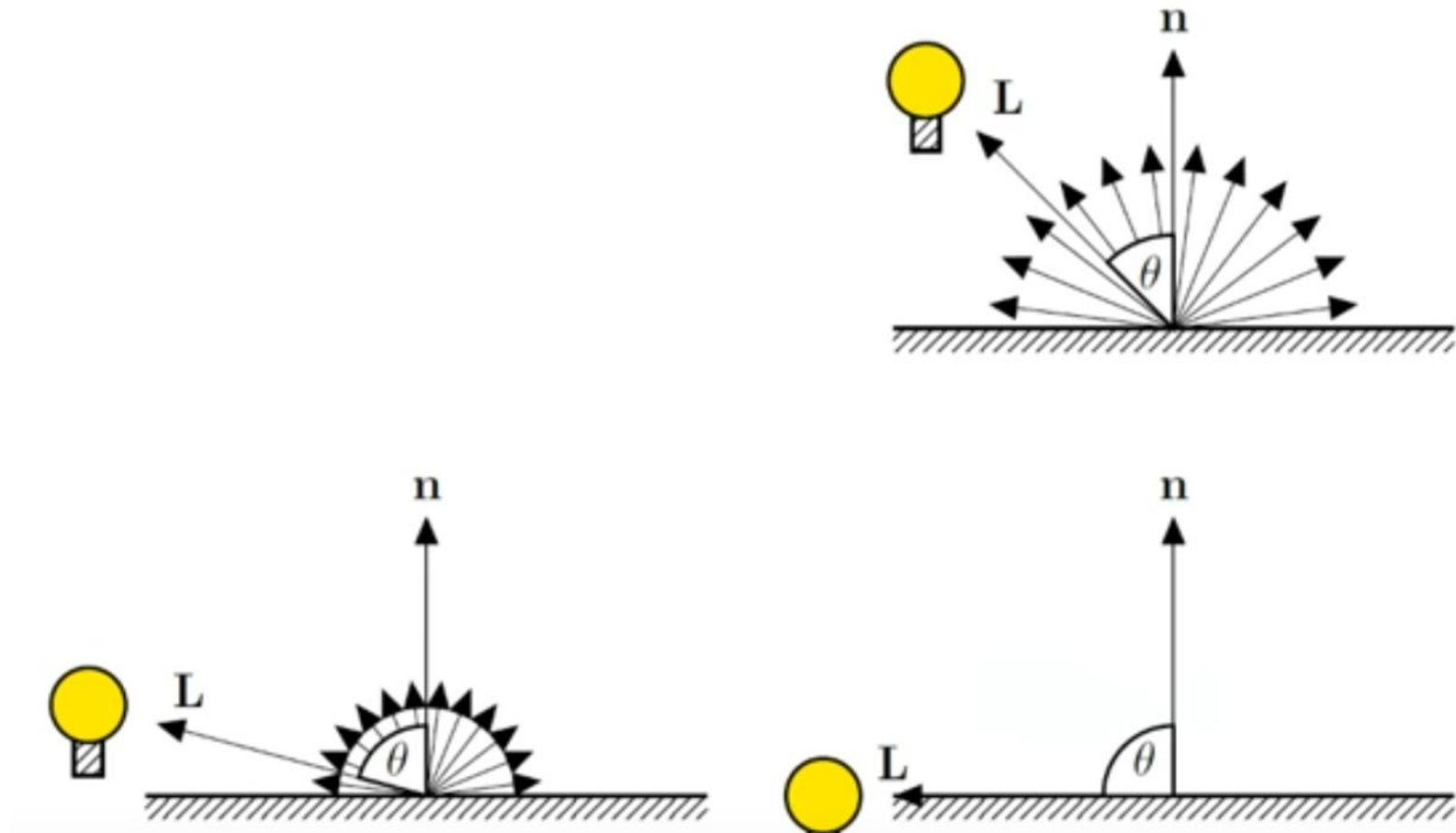
Phong model of diffuse lighting



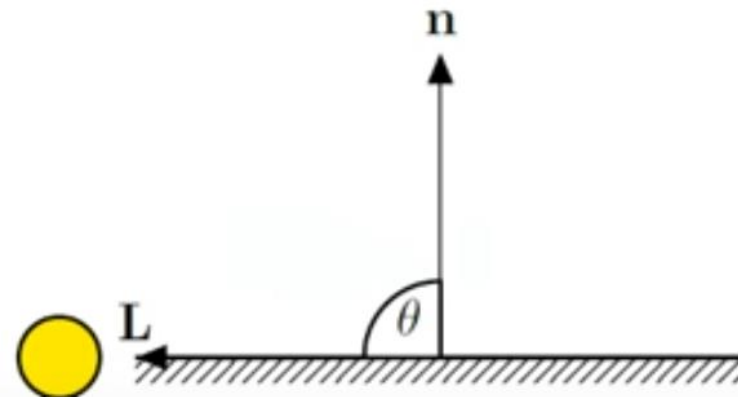
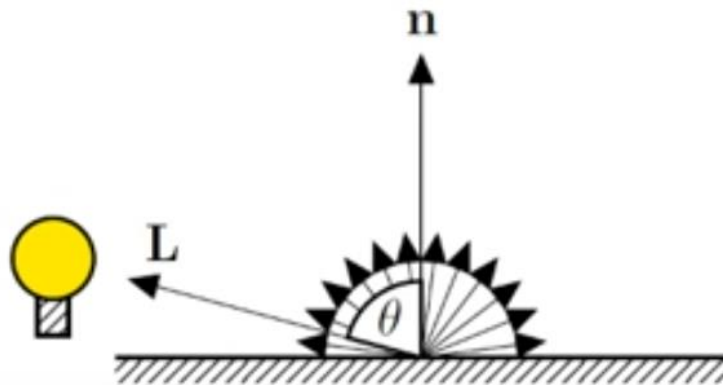
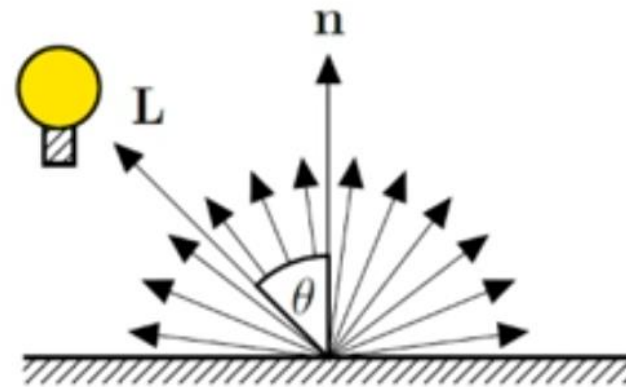
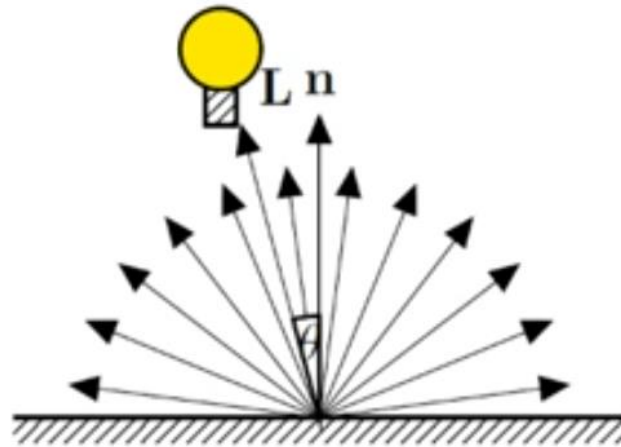
Phong model of diffuse lighting



Phong model of diffuse lighting



Phong model of diffuse lighting



Phong model of diffuse lighting

- Phong model of diffuse lighting depends on the position of the light source relative to the object surface
- $D = I_p k_d \max[\cos(\theta), 0]$

Phong model of diffuse lighting

- Phong model of diffuse lighting depends on the position of the light source relative to the object surface
- $D = I_p k_d \max[\cos(\theta), 0]$
- Where
 - I_p is the light intensity of the light source
 - $k_d \in [0, 1]$ is a diffuse lighting coefficient
 - θ is the angle between L and the normal of the object surface n

Phong model of diffuse lighting

- Phong model of diffuse lighting depends on the position of the light source relative to the object surface
- $D = I_p k_d \max[\cos(\theta), 0]$
- Where
 - I_p is the light intensity of the light source
 - $k_d \in [0, 1]$ is a diffuse lighting coefficient
 - θ is the angle between L and the normal of the object surface n
- Function $\max[\cos(\theta), 0]$ is used so that light is not reflected when the light source is behind the object surface



$k_d = 0.25$



$k_d = 0.5$

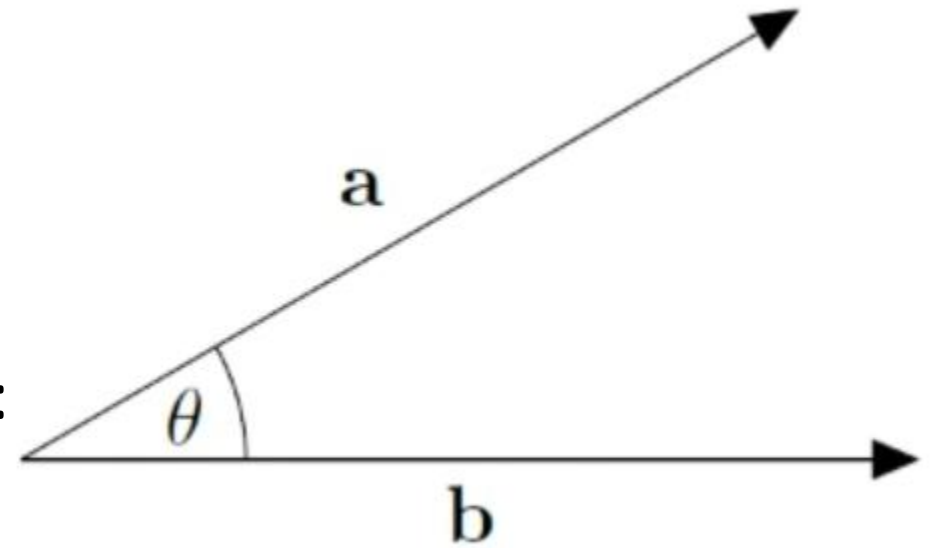


$k_d = 0.75$

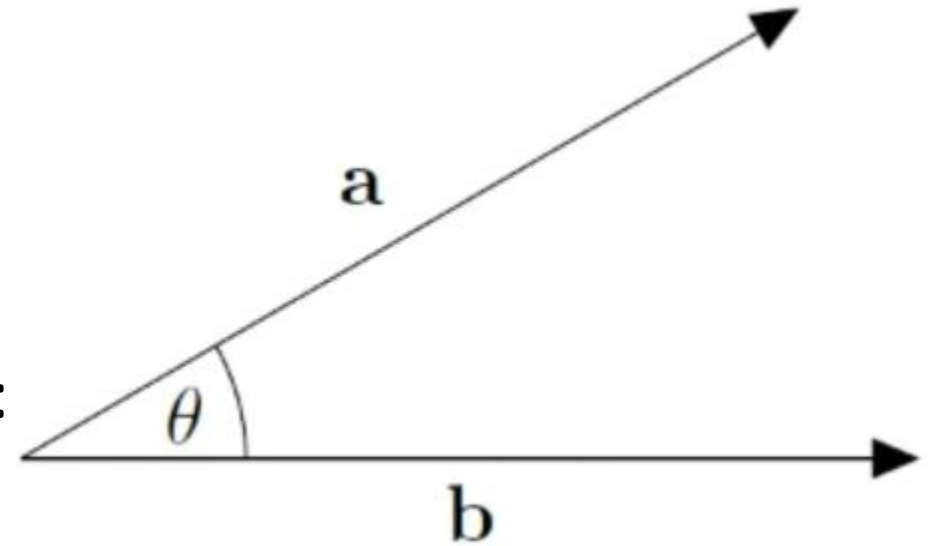


$k_d = 1$

- Dot product between two vectors:
 - $a \cdot b = |a| |b| \cos(\theta)$
- However if the vectors are of unit length:
 - $L \cdot n = \cos(\theta)$



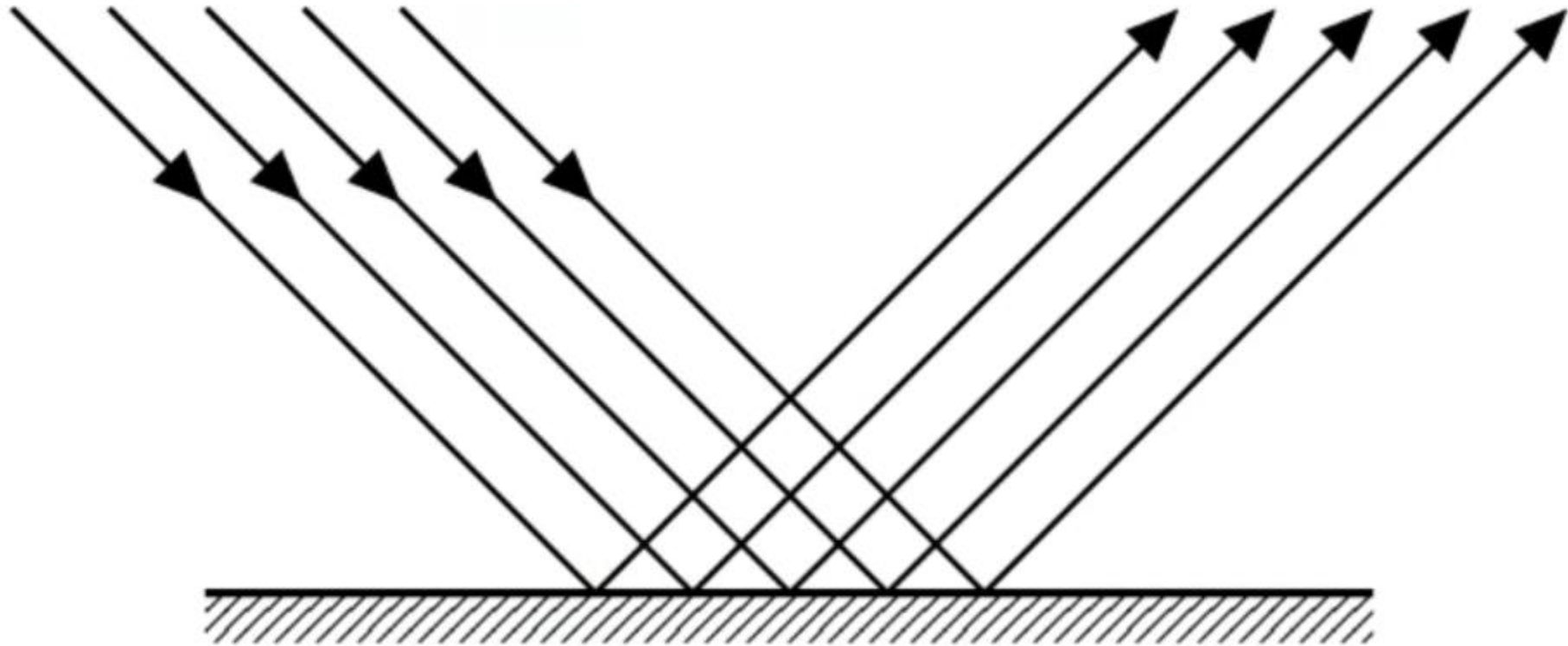
- Dot product between two vectors:
 - $a \cdot b = |a| |b| \cos(\theta)$
- However if the vectors are of unit length:
 - $L \cdot n = \cos(\theta)$
- We can substitute the costly cosine calculation with the dot product
 - $D = I_p k_d \max[L \cdot n, 0]$



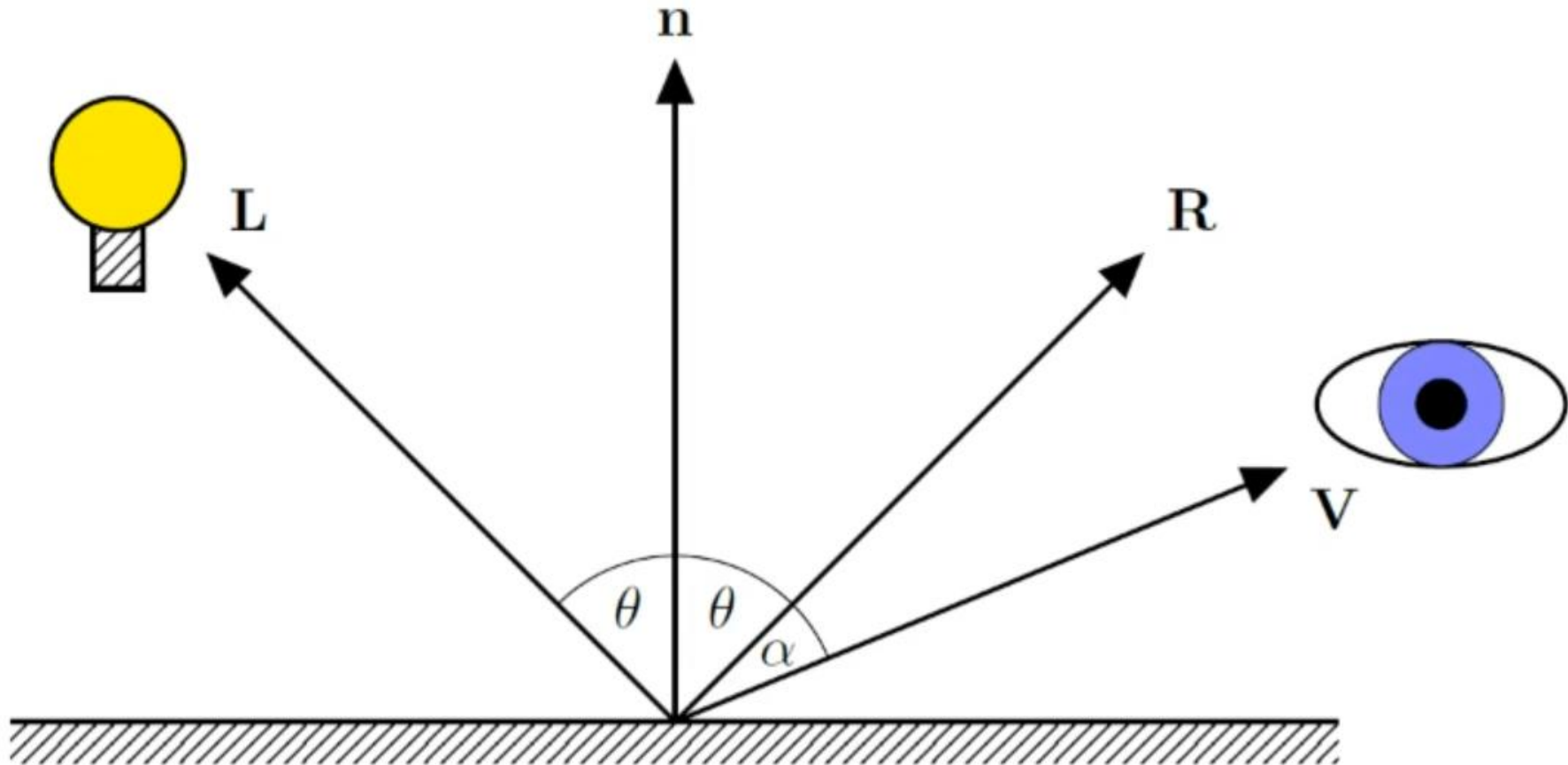
Light reflection

Incoming light rays

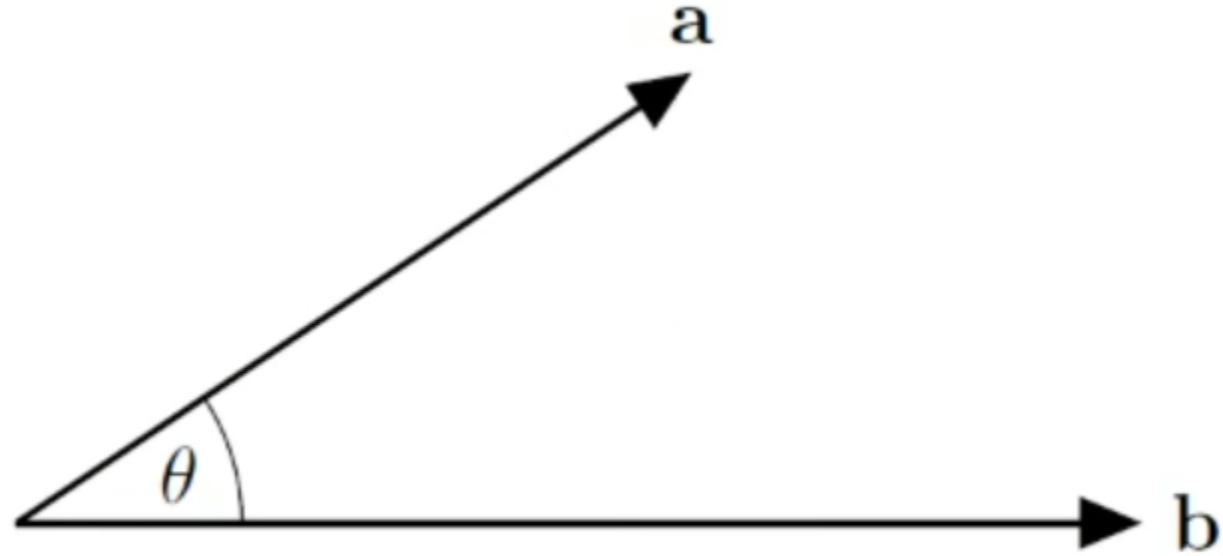
Reflected light rays



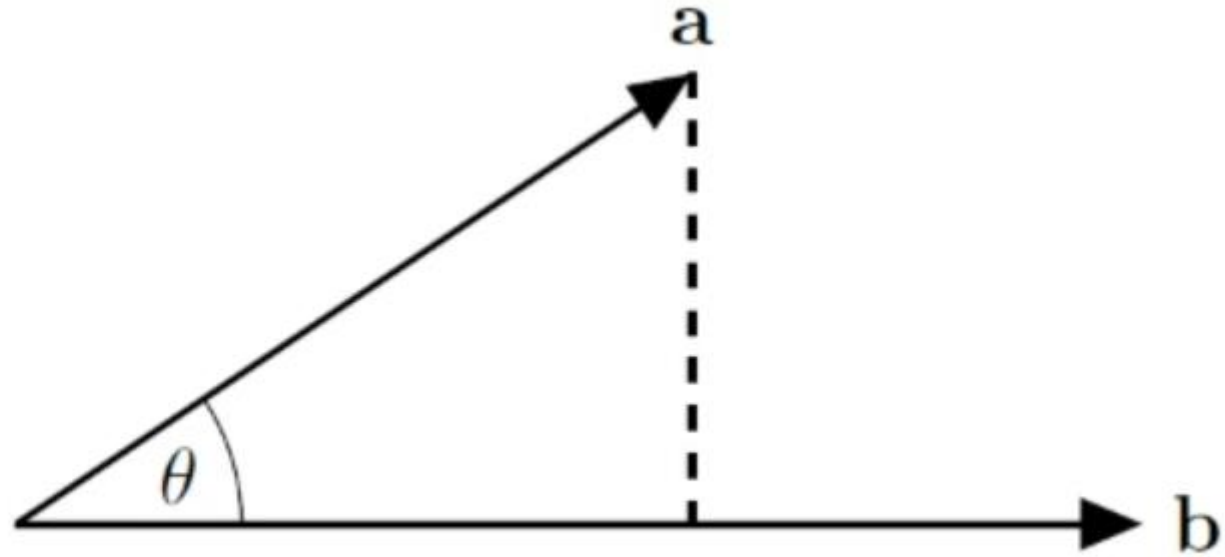
Specular lighting - vectors



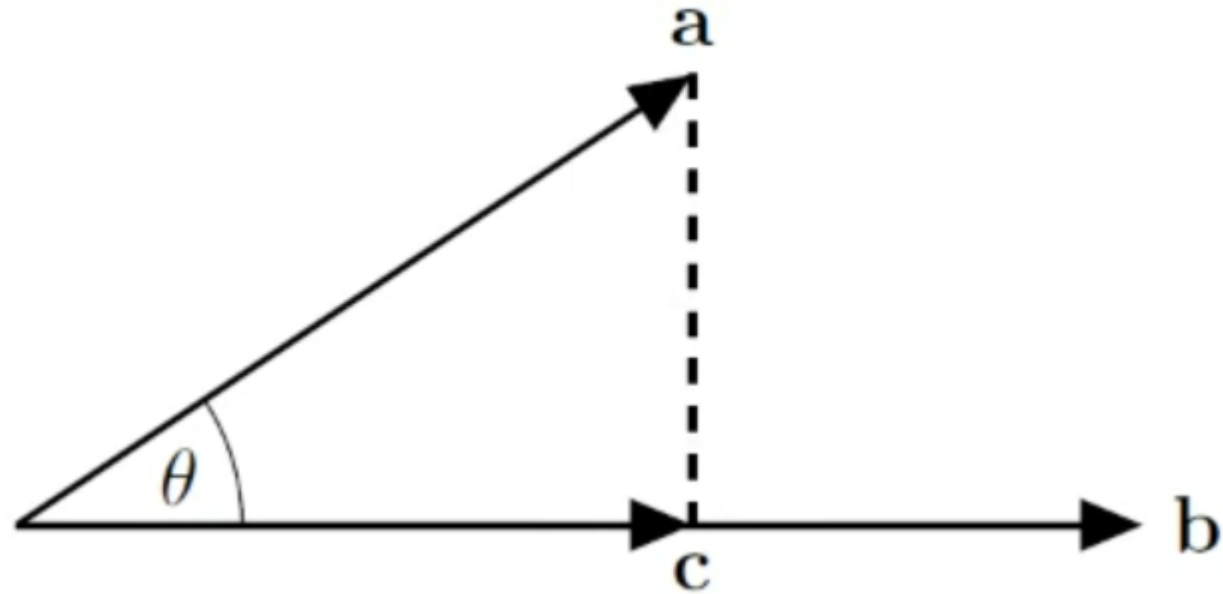
Vector projection

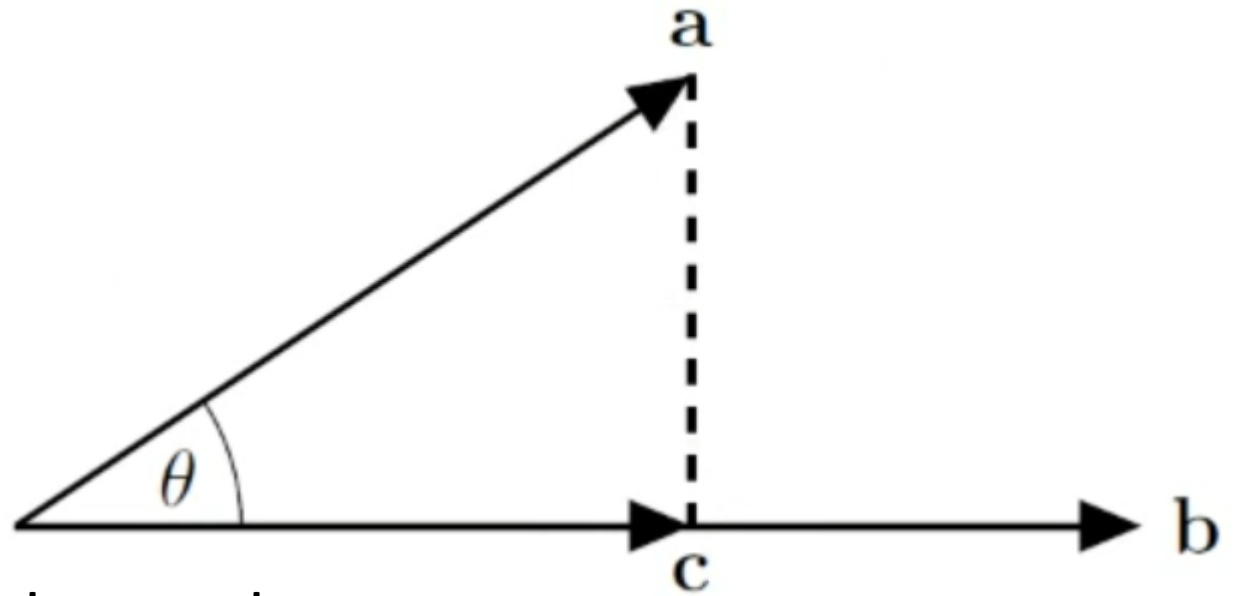


Vector projection

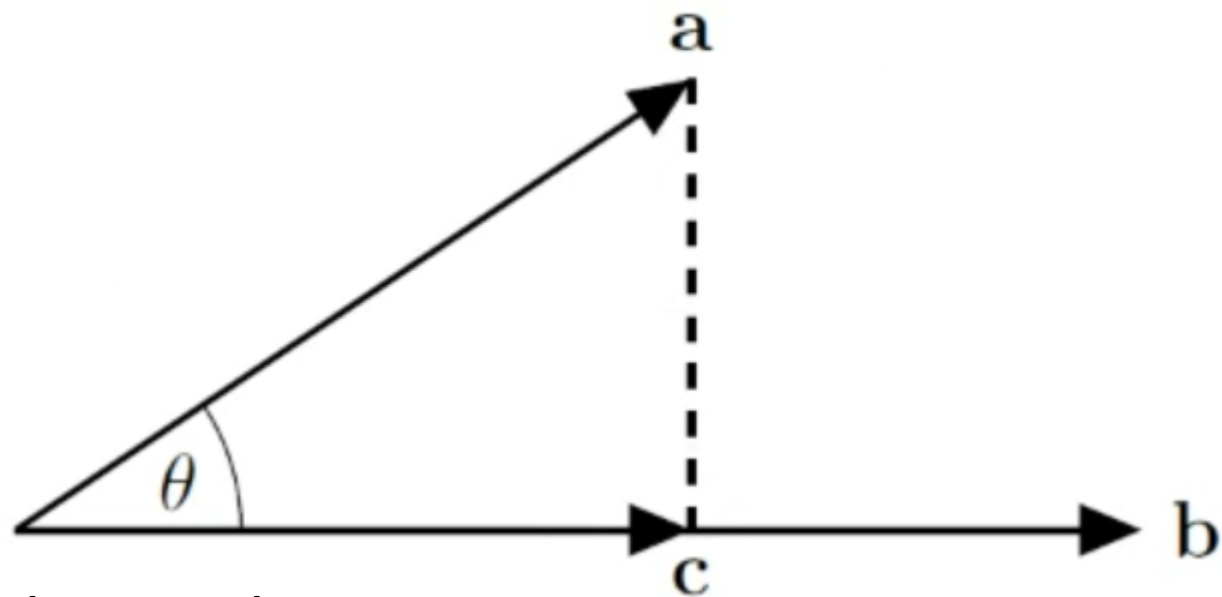


Vector projection



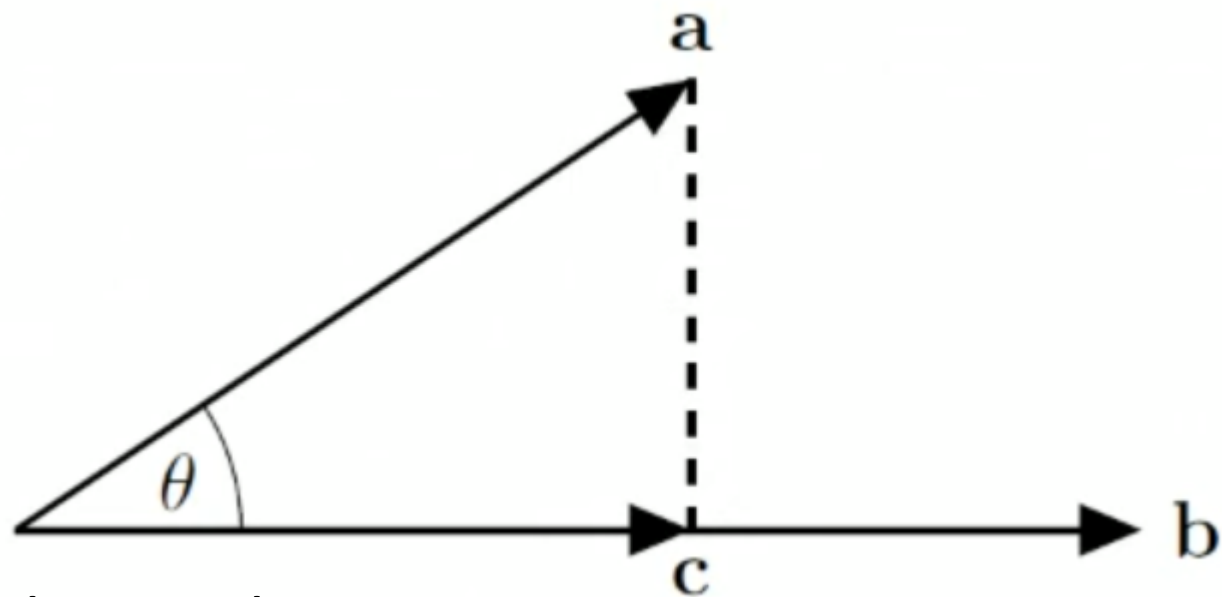


- From the definition of the dot product:
- $a \cdot b = |a||b| \cos(\alpha)$



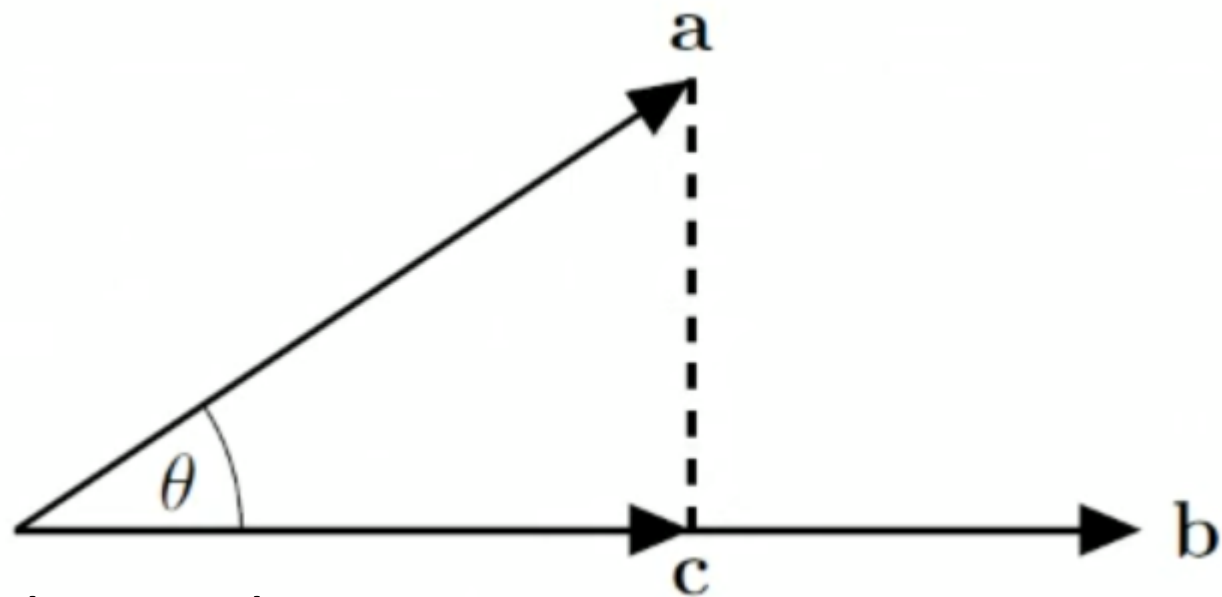
- From the definition of the dot product:

- $a \cdot b = |a||b| \cos(\alpha) = |a||b| \frac{|c|}{|a|}$



- From the definition of the dot product:

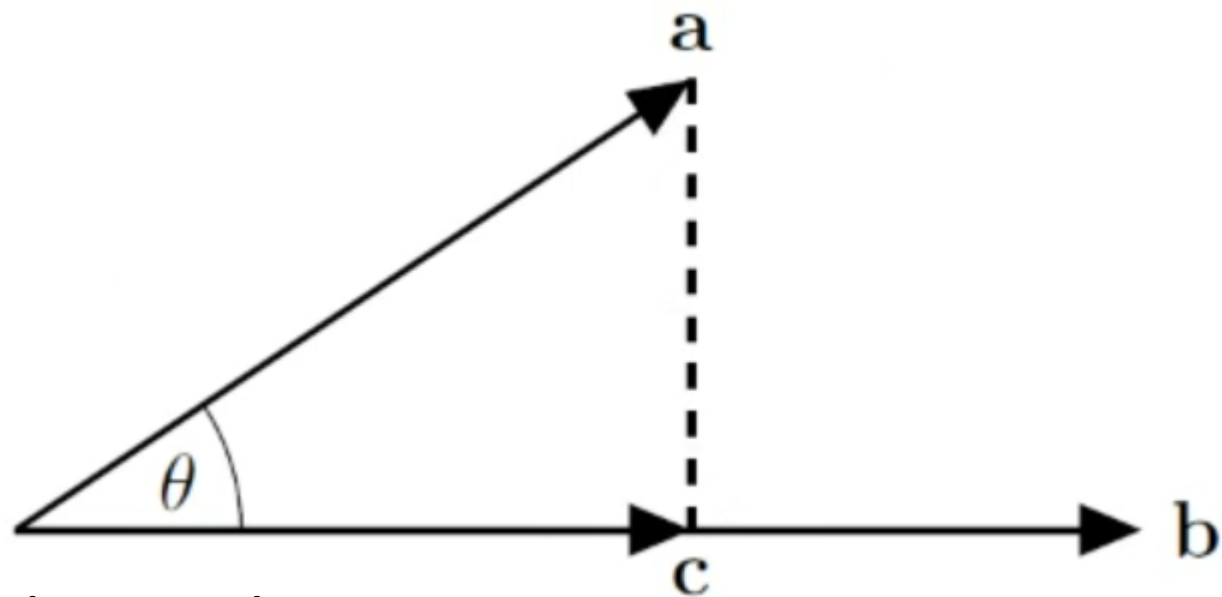
- $a \cdot b = |a||b| \cos(\alpha) = |a||b| \frac{|c|}{|a|} = |b||c|$



- From the definition of the dot product:

- $a \cdot b = |a||b| \cos(\alpha) = |a||b| \frac{|c|}{|a|} = |b||c|$

- $|c| = \frac{a \cdot b}{|b|}$



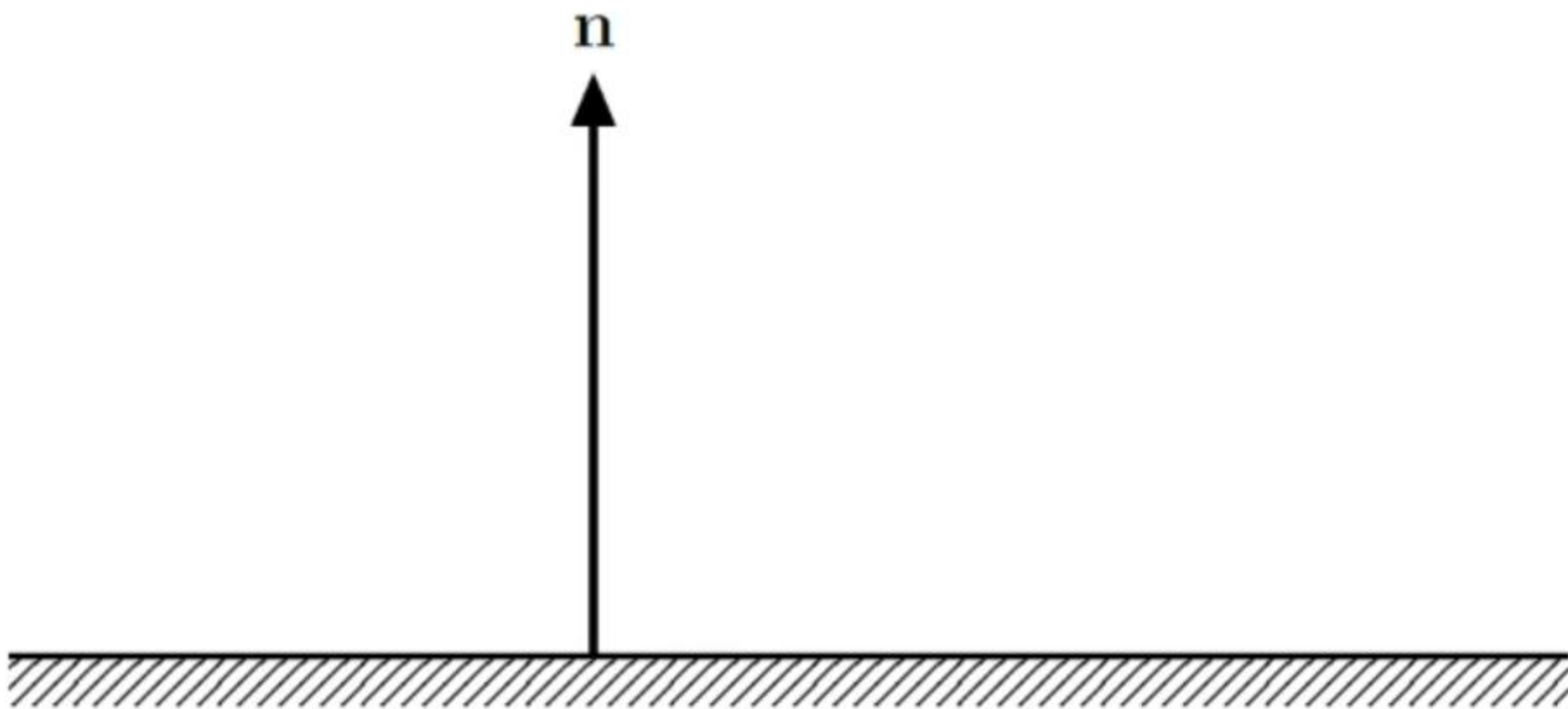
- From the definition of the dot product:

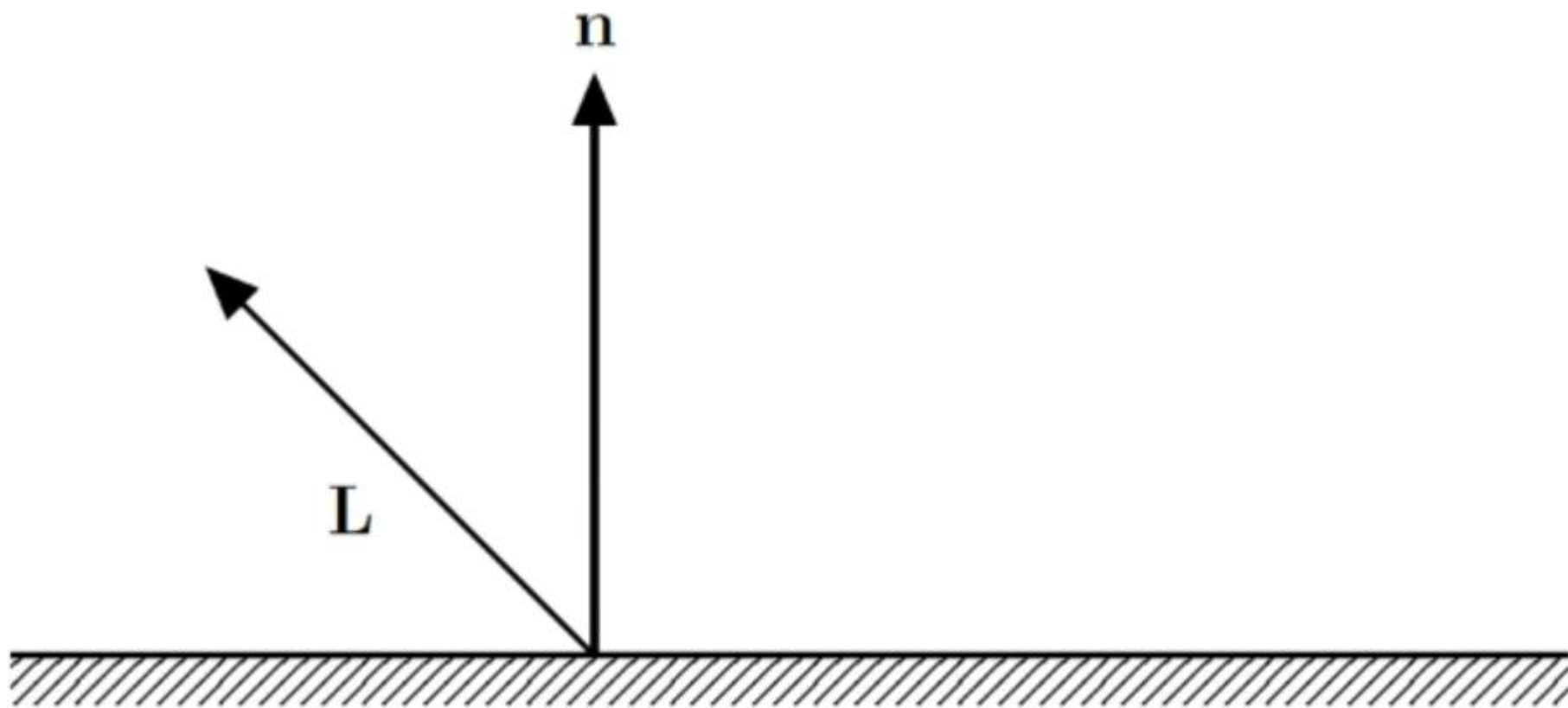
- $a \cdot b = |a||b| \cos(\alpha) = |a||b| \frac{|c|}{|a|} = |b||c|$

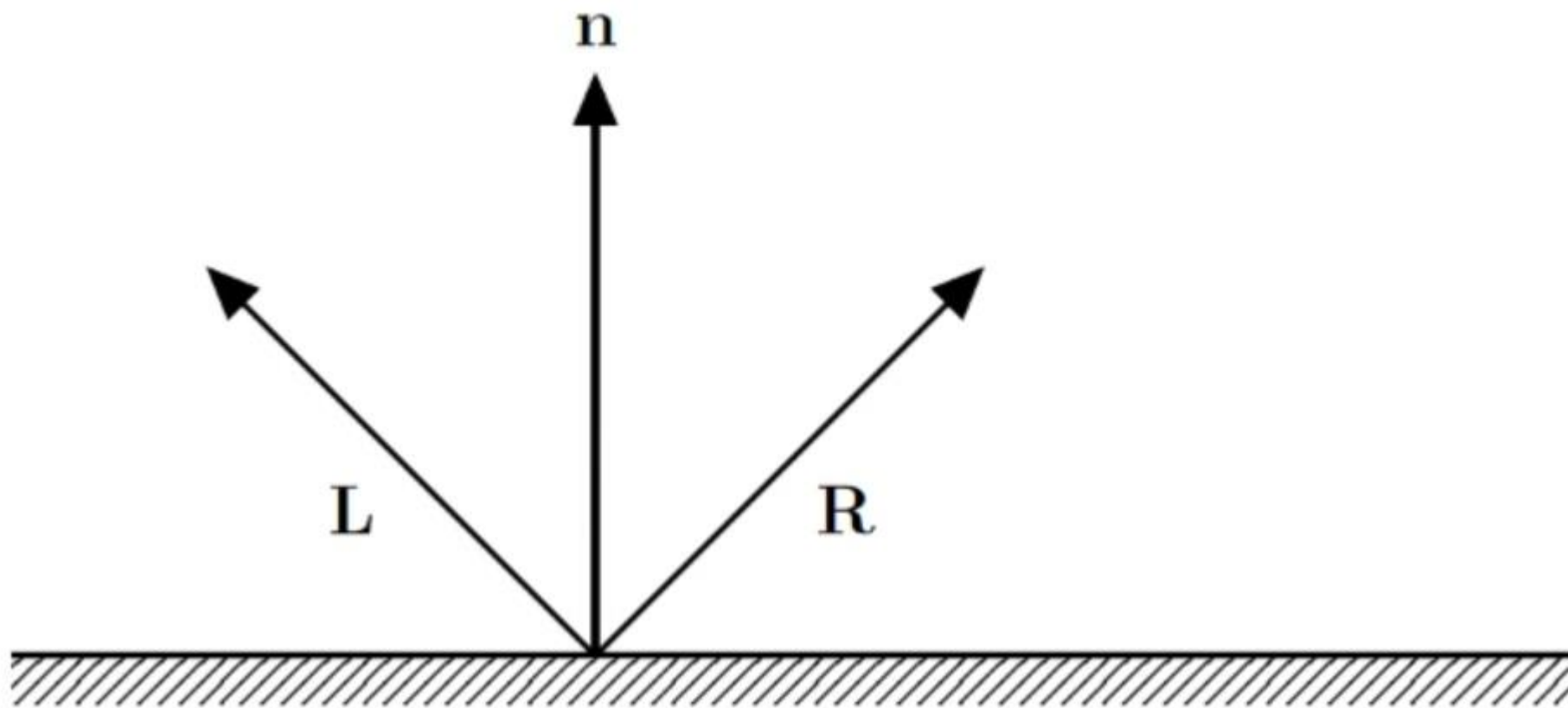
- $|c| = \frac{a \cdot b}{|b|}$

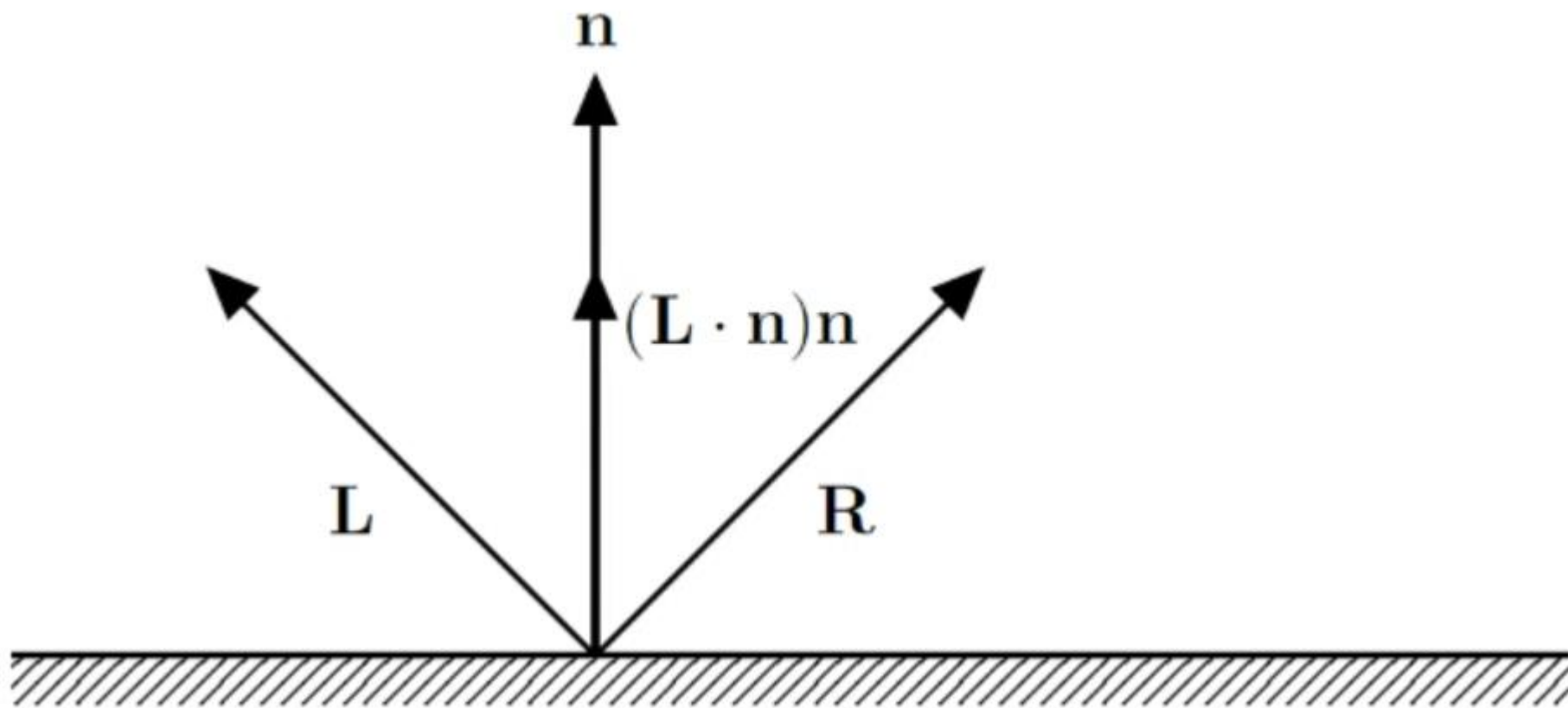
- If **b** is a unit vector $|c| = a \cdot b$ then

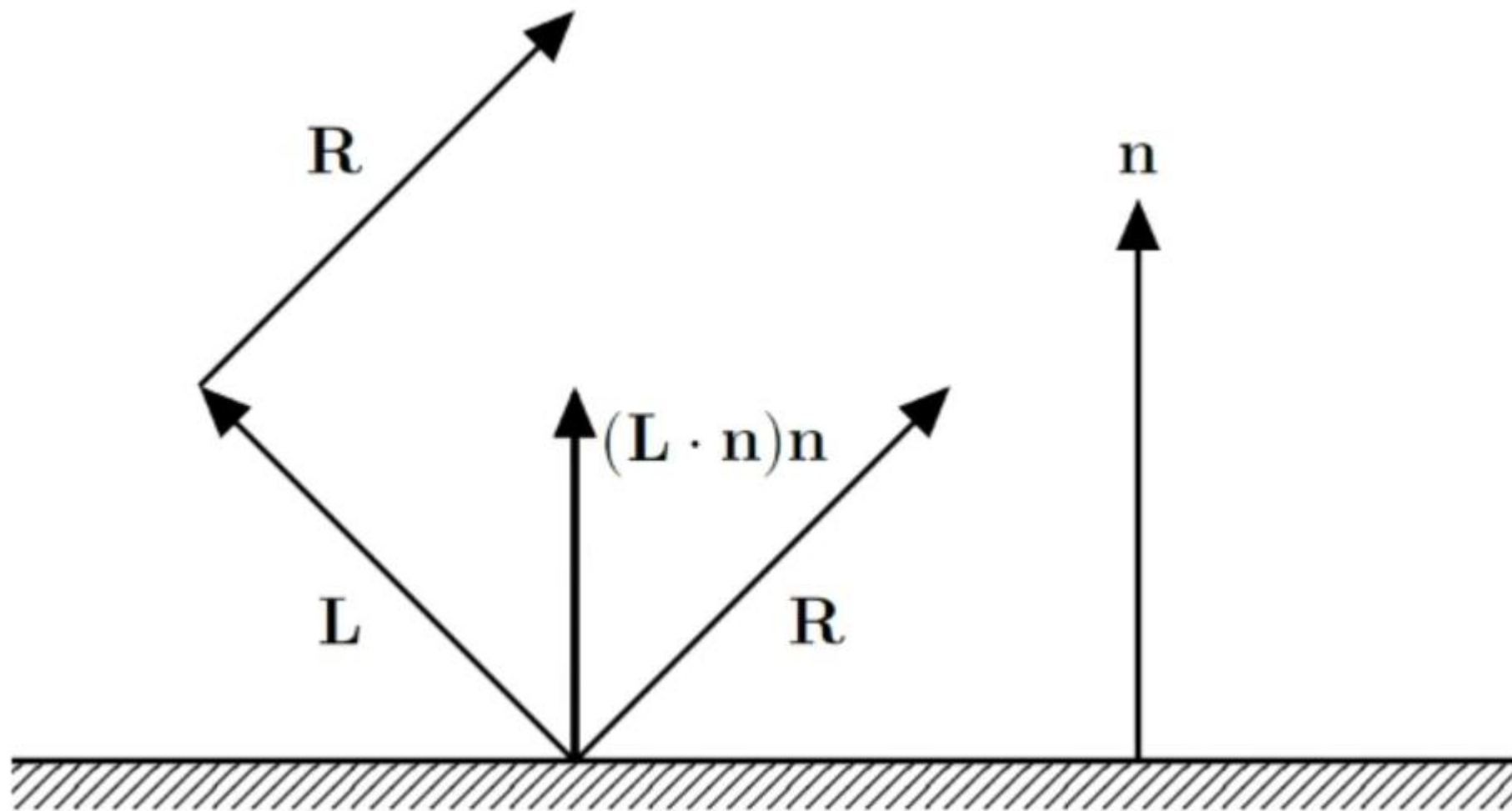
- $c = (a \cdot b)b$

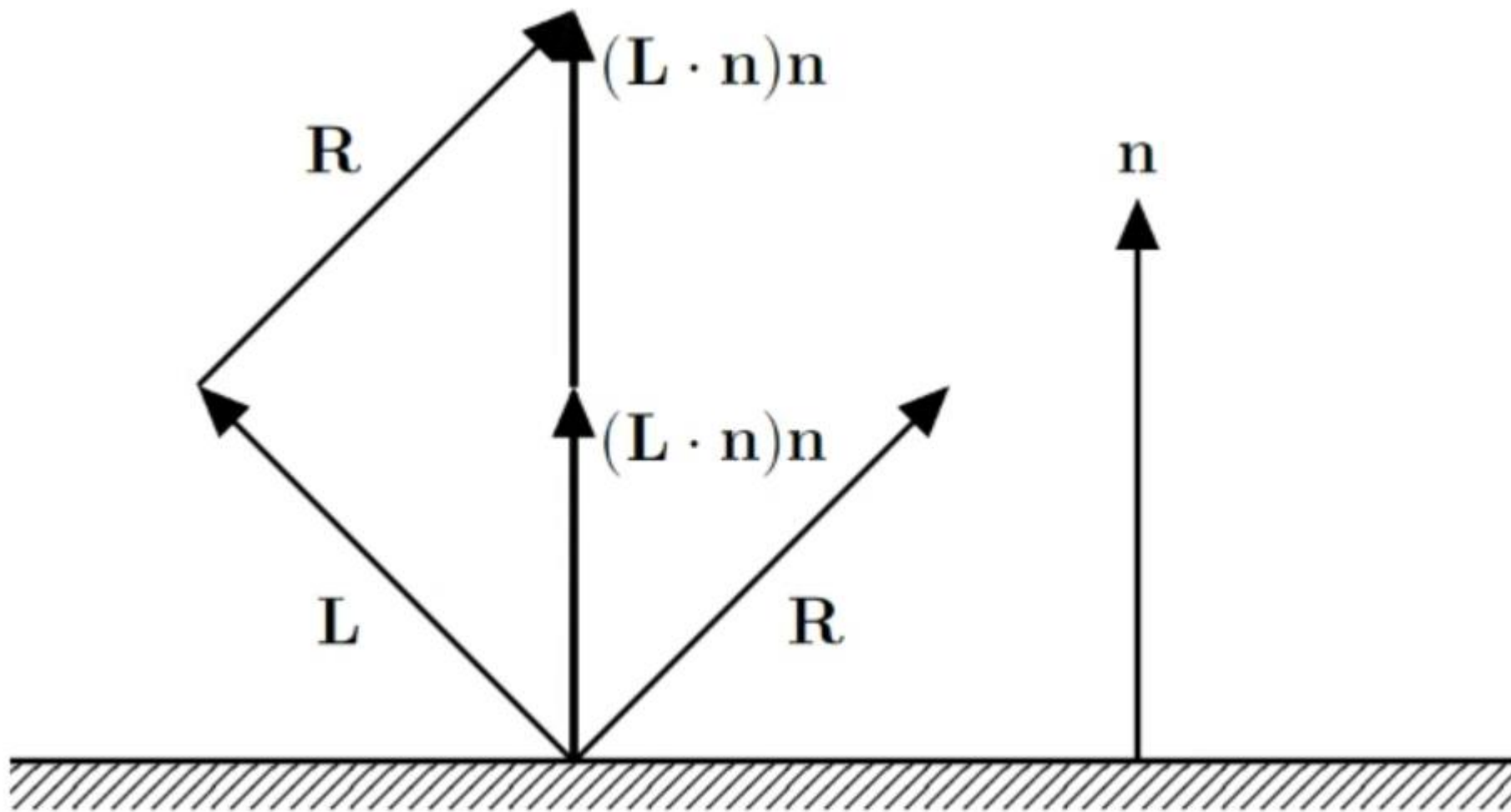


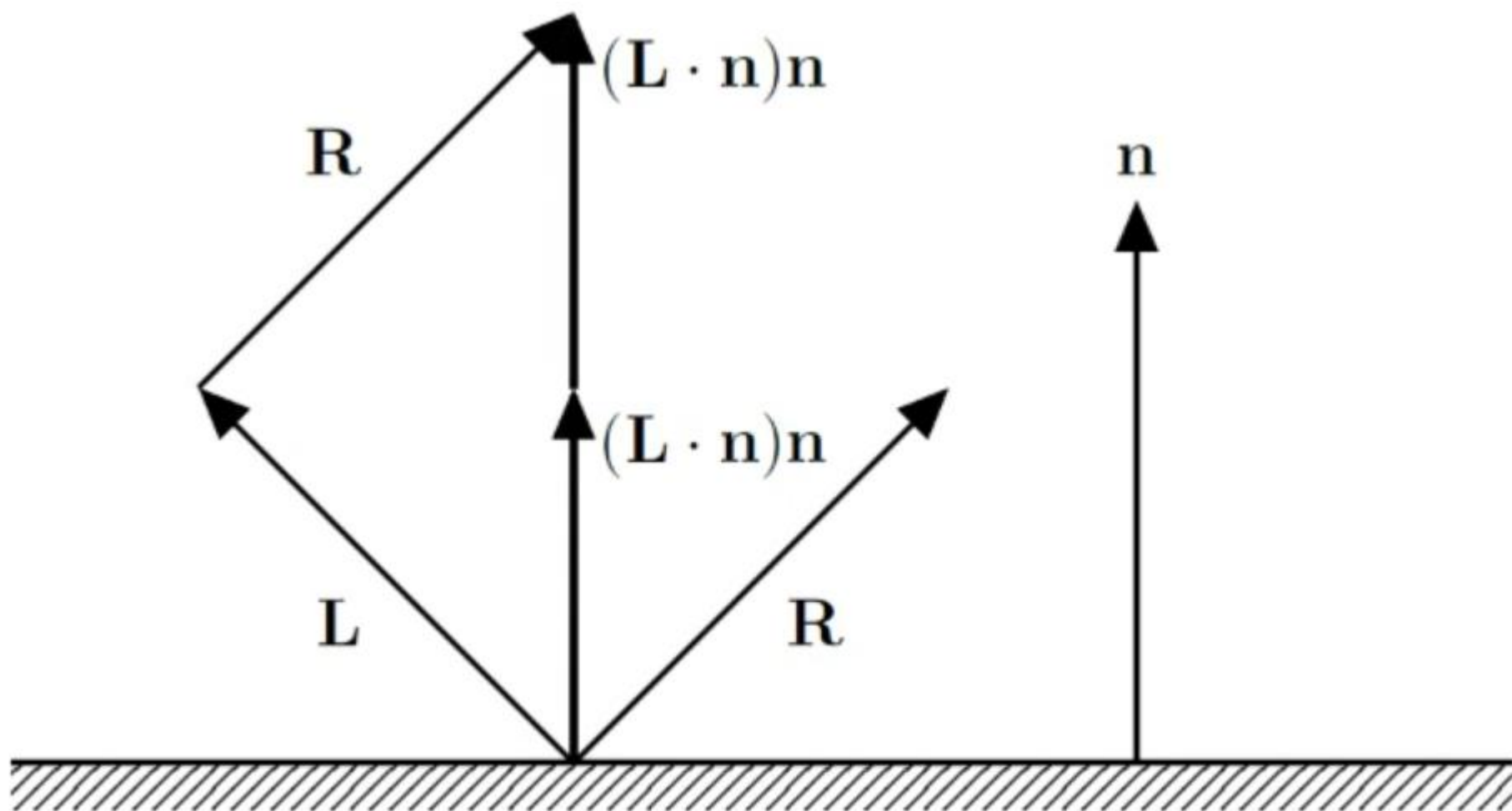




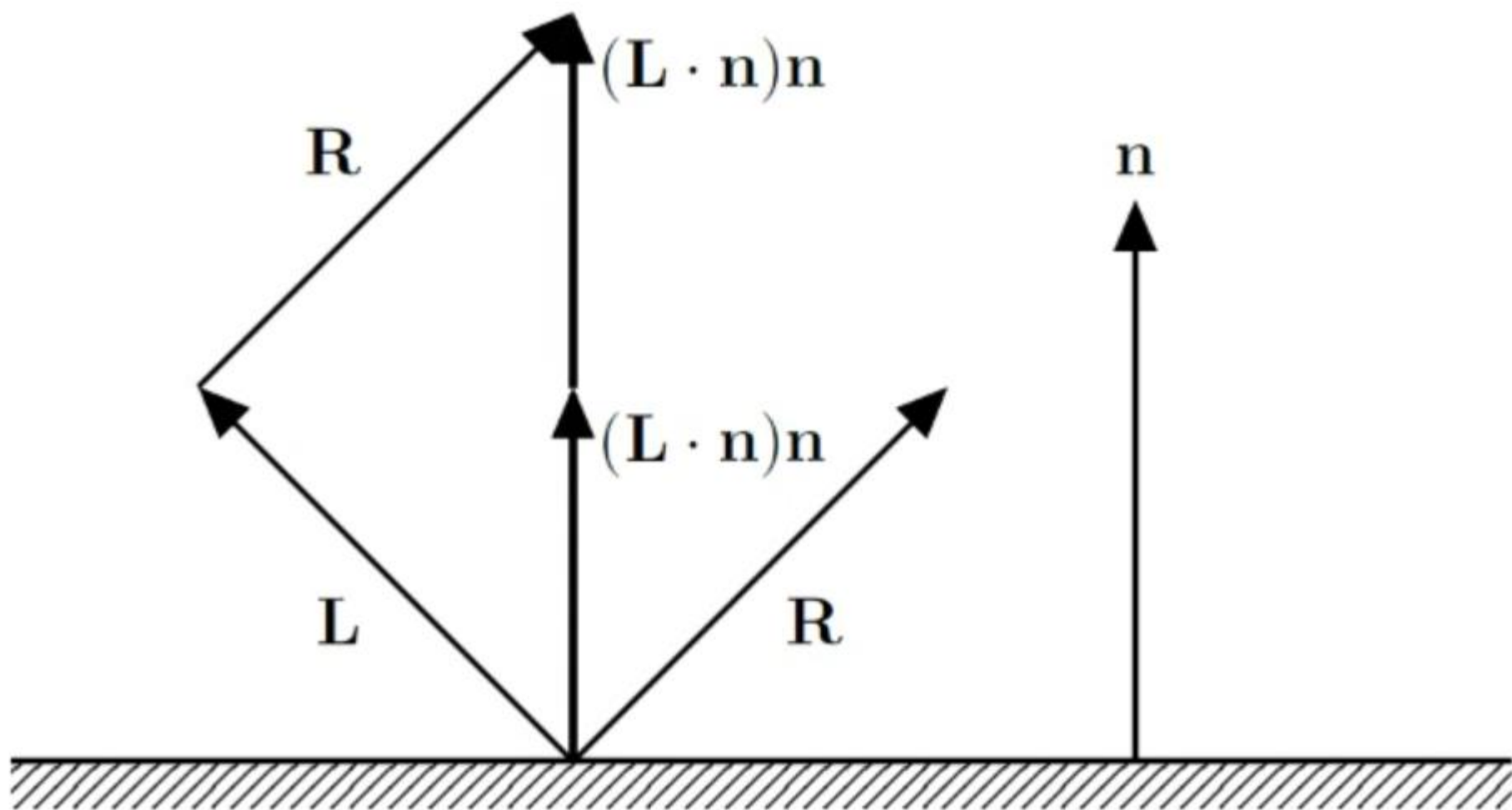








$$\mathbf{L} + \mathbf{R} = 2(\mathbf{L} \cdot \mathbf{n})\mathbf{n}$$



$$\mathbf{L} + \mathbf{R} = 2(\mathbf{L} \cdot \mathbf{n})\mathbf{n}$$

$$\therefore \mathbf{R} = 2(\mathbf{L} \cdot \mathbf{n})\mathbf{n} - \mathbf{L}$$

GLSL

$r = \text{reflect}(I, n)$

Name

reflect — calculate the reflection direction for an incident vector

Declaration

```
genType reflect( genType I,  
                  genType N) ;  
  
genDType reflect( genDType I,  
                  genDType N) ;
```

Parameters

I

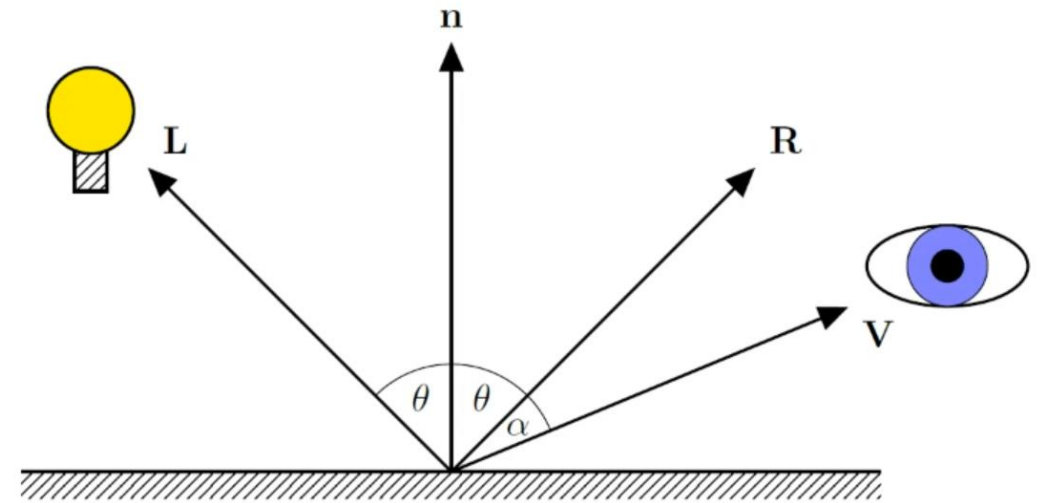
Specifies the incident vector.

N

Specifies the normal vector.

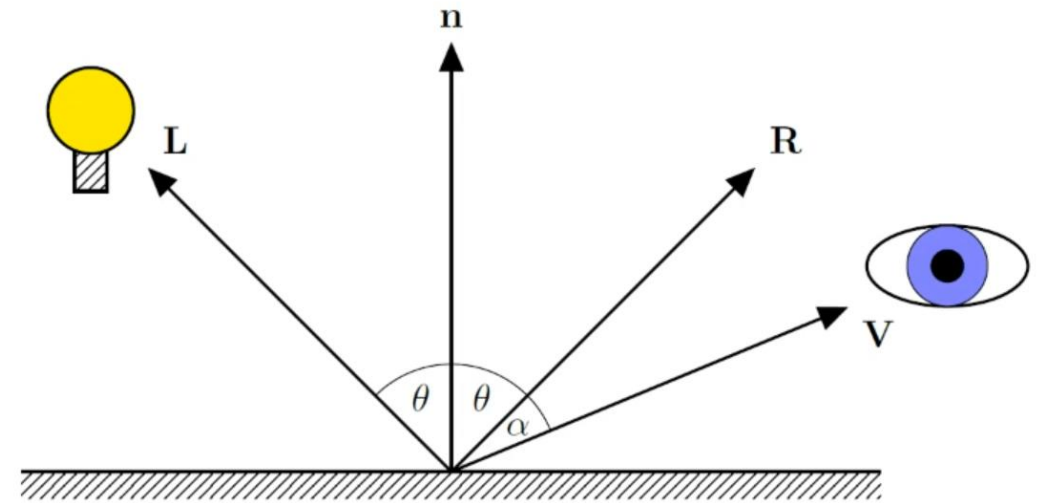
Specular lighting - vectors

- The Phong model of specular lighting is:
- $S = I_p k_s \cos^n(\alpha) = I_p k_s (V \cdot R)^n$



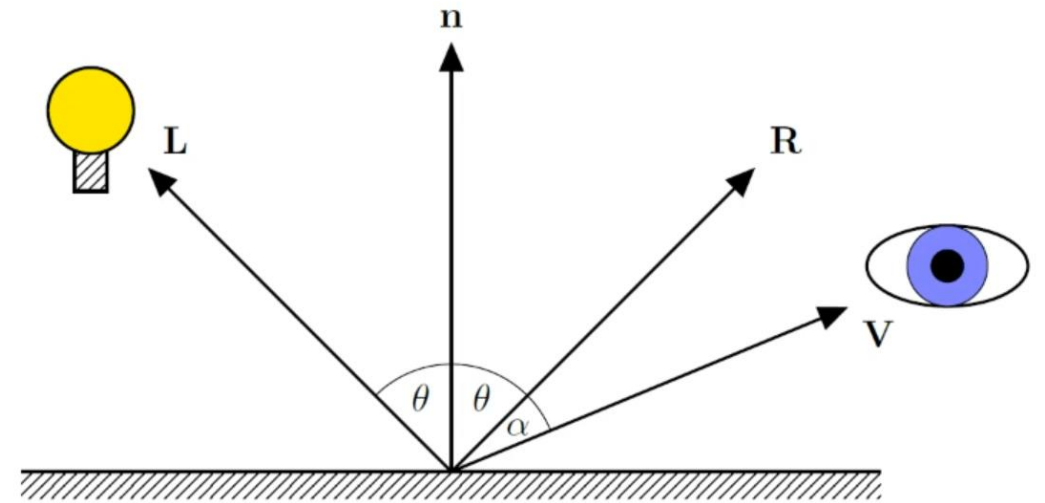
Specular lighting - vectors

- The Phong model of specular lighting is:
- $S = I_p k_s \cos^n(\alpha) = I_p k_s (V \cdot R)^n$
- Where
 - $k_s \in [0, 1]$ is the specular lighting coefficient
 - n is the specular lighting exponent
 - α is the angle between R and V

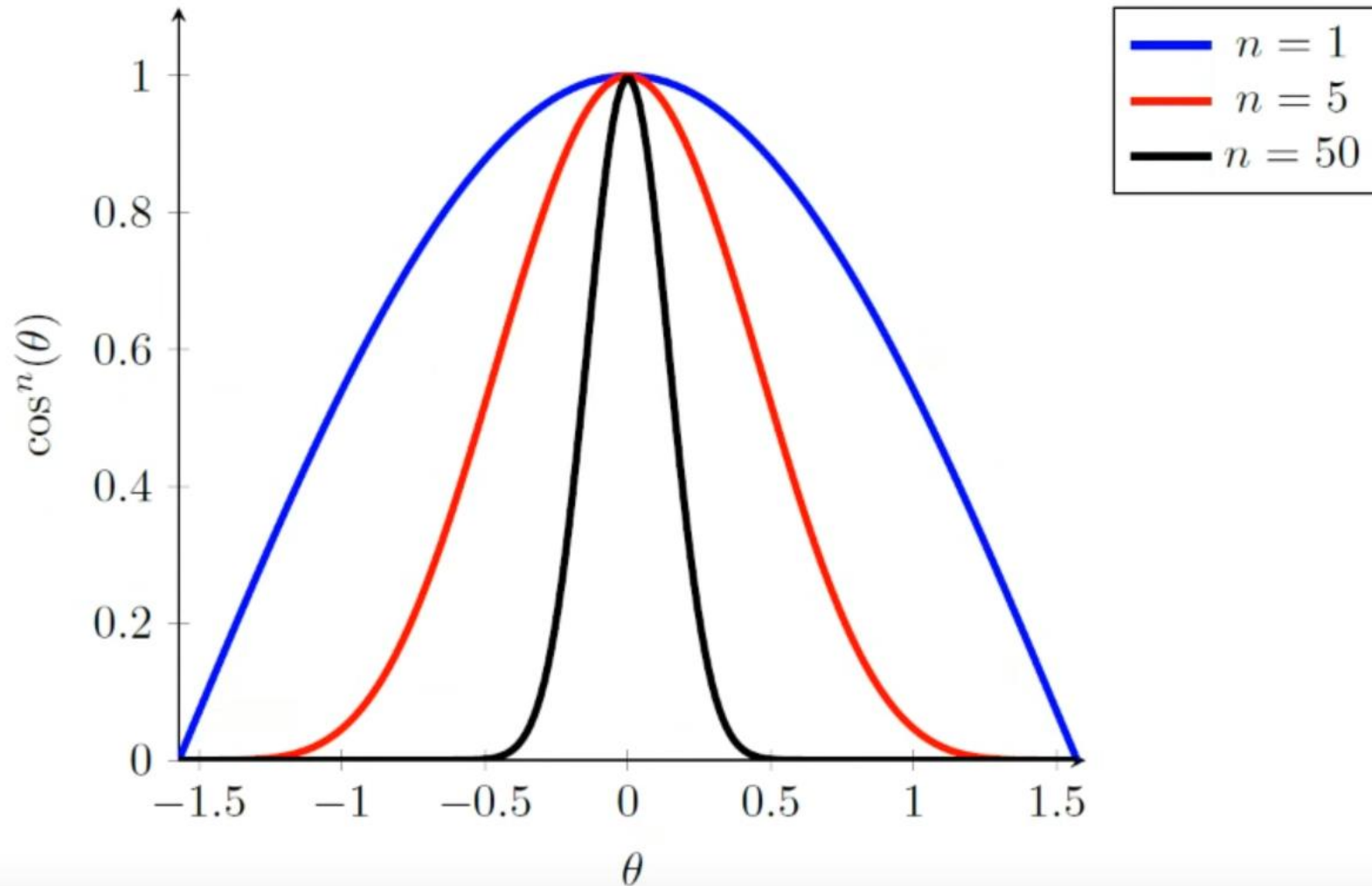


Specular lighting - vectors

- The Phong model of specular lighting is:
- $S = I_p k_s \cos^n(\alpha) = I_p k_s (V \cdot R)^n$
- Where
 - $k_s \in [0, 1]$ is the specular lighting coefficient
 - n is the specular lighting exponent
 - α is the angle between R and V
- $\cos^n(\alpha)$ determines how much light is reflected by a given material

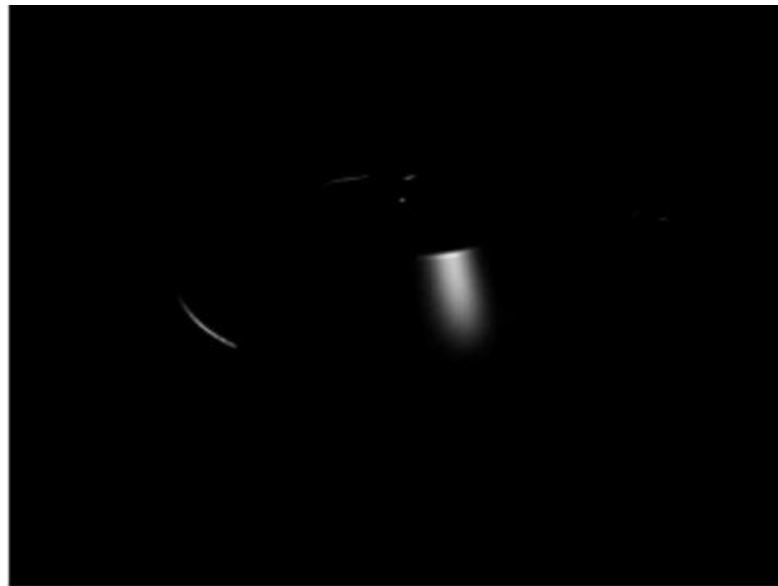


Specular lighting coefficient

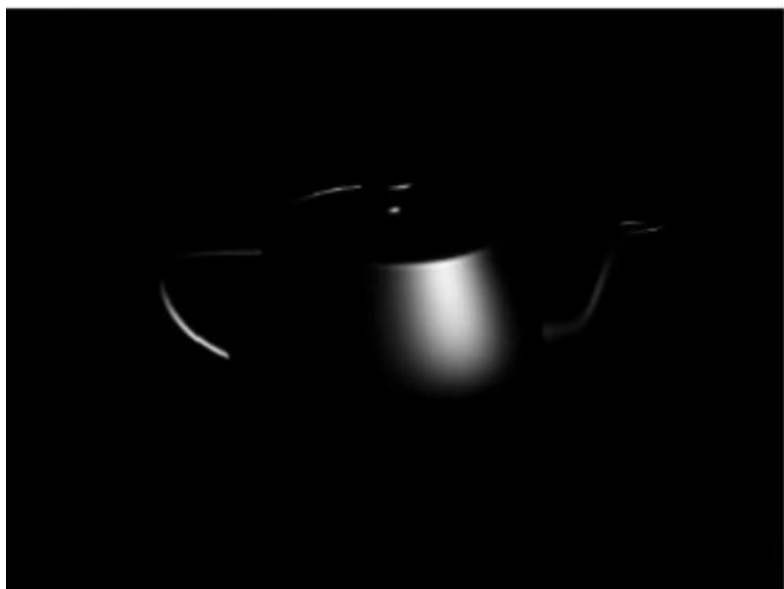




$n = 50$



$n = 20$



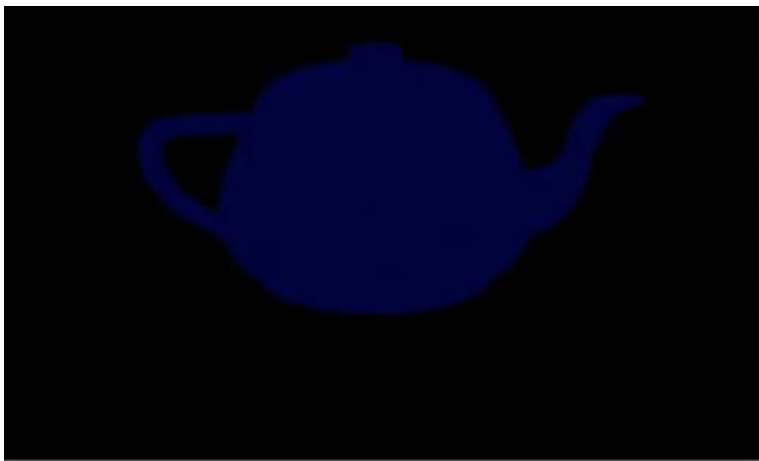
$n = 5$



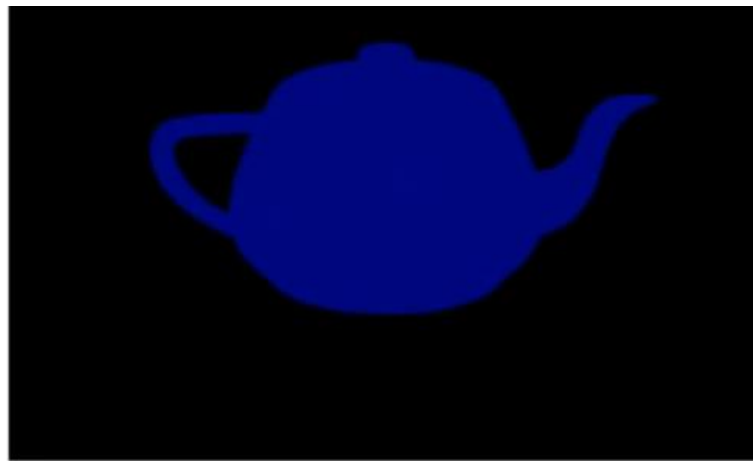
$n = 1$

Ambient lighting

- Ambient lighting, is the reflected light not coming directly from the light source
- $A = I_a k_a$
- Where I_a denotes light intensity $k_a \in [0, 1]$ is the coefficient of the environment
- k_a allows to specify a minimum amount of light available for all objects in a scene, n.p. $k_a \rightarrow 1$ for bright $k_a \rightarrow 0$ for dark scenes



$k_a = 0.25$



$k_a = 0.5$



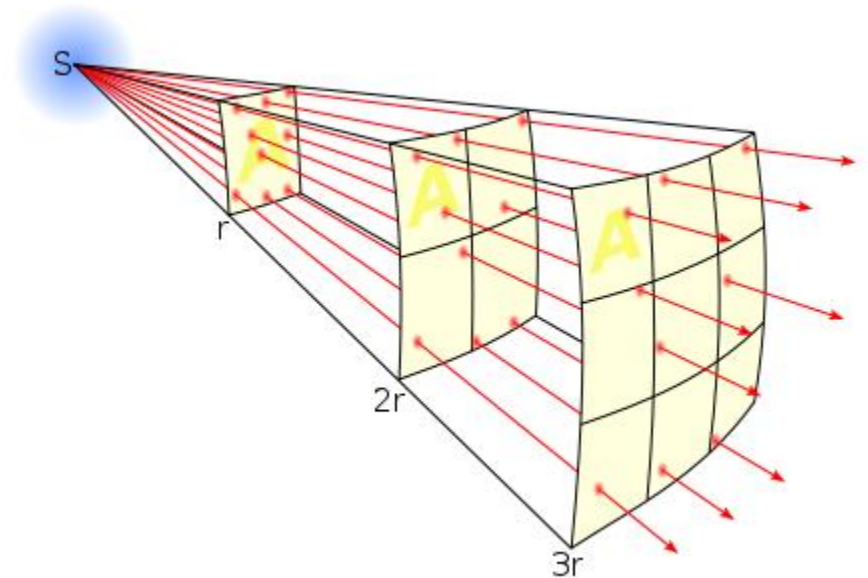
$k_a = 0.75$



$k_a = 1.0$

Attenuation

- Attenuation describes the diminishing of light energy in space
- In the Phong model the coefficient f_{att} models this phenomenon
- Physically-based models propose the following relation
- $f_{att} \sim \frac{1}{d^2}$



Attenuation

- Because the Phong model is a direct lighting model not accounting for reflected light rays usually too much light would be removed using the physically-based formulation. Instead, the following relation is used:
- $f_{att} = 1 - \left(\frac{d}{r}\right)^2$
- Where r is the radius of the sphere surrounding the light source that represents the furthest possible light

Attenuation

- Adding all the individual parts of the Phong model gives the full Phong light reflectance model
- $I = I_a k_a + f_{att} I_p k_d \max[L \cdot n, 0] + f_{att} I_p k_s (V \cdot R)^n$

Attenuation

- Adding all the individual parts of the Phong model gives the full Phong light reflectance model
- $I = I_a k_a + f_{att} I_p k_d \max[L \cdot n, 0] + f_{att} I_p k_s (V \cdot R)^n$
- For scenes with more than one light source, the individual intensities are simply added together
- $I = I_a k_a + \sum_{i=1}^m f_{att} I_{p,i} [k_d \max(L_i \cdot n, 0) + k_s (V \cdot R_i)^n]$

Light intensity and RGB colors

- We can simply scale the **color vectors** in the fragment shader:

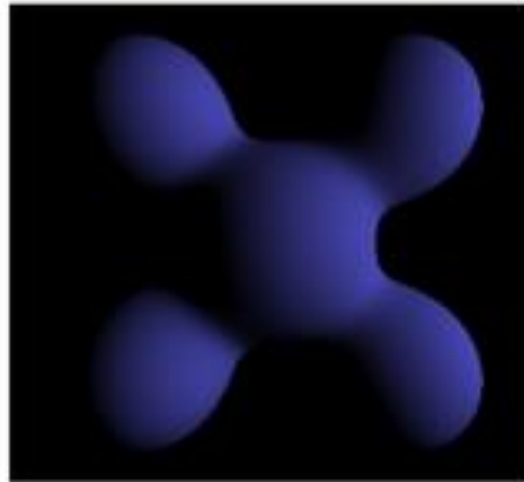
$$v_{new} = v \cdot I$$

- For example: the red color $v = (1, 0, 0)$ with 50% intensity becomes
 $v_{new} = (0.5, 0, 0)$

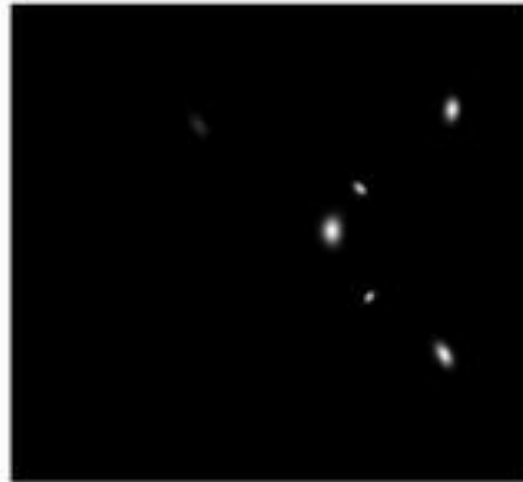
Result



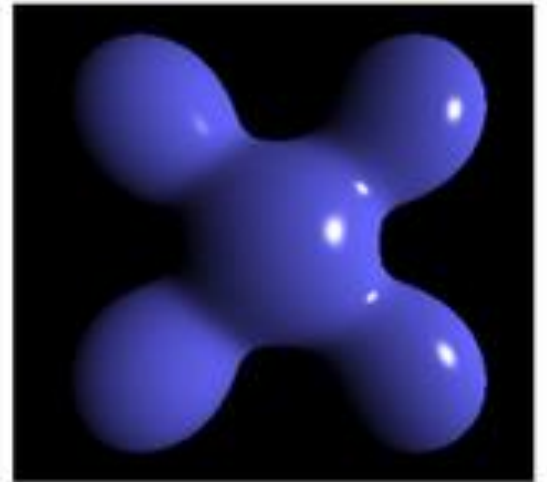
Ambient light



Diffuse light



Specular light



Phong lighting model