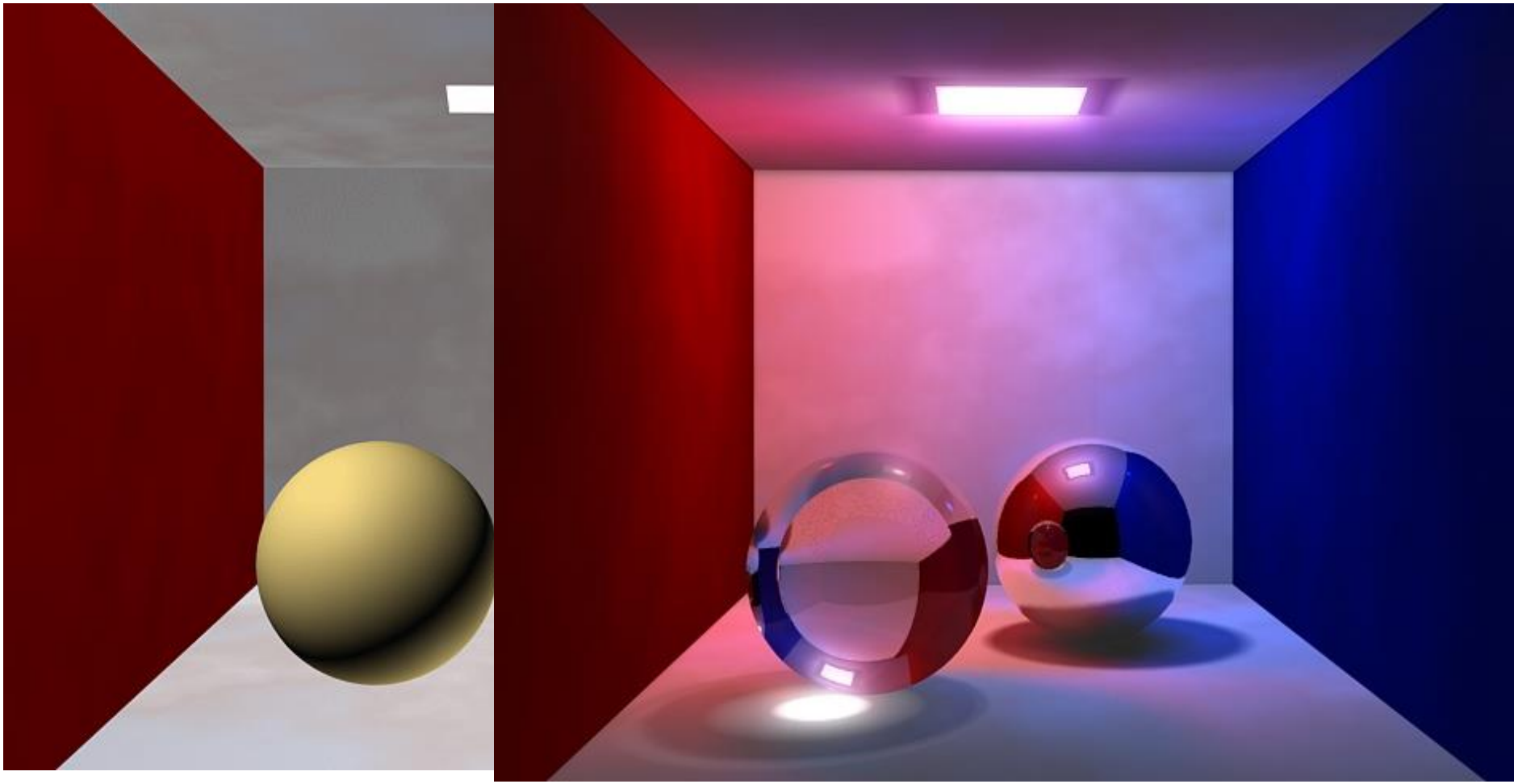
Lighting and shading

Part I

Without light … we don't see much of our scene!
Without shading, objects do not look three dimensional!

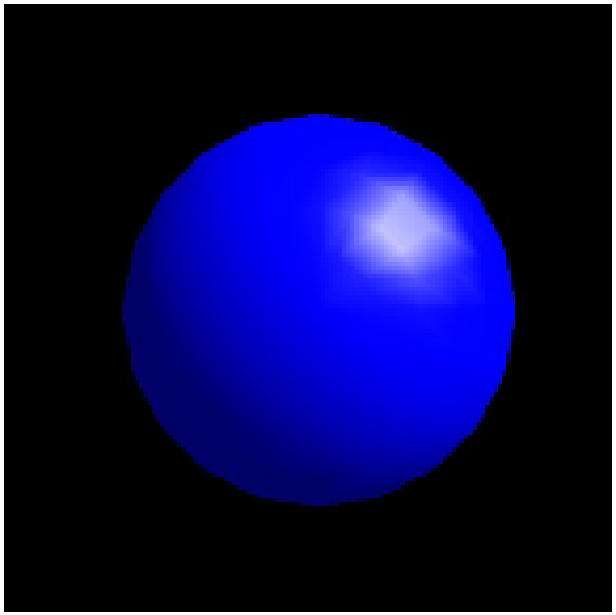
Color, depth and photorealism..

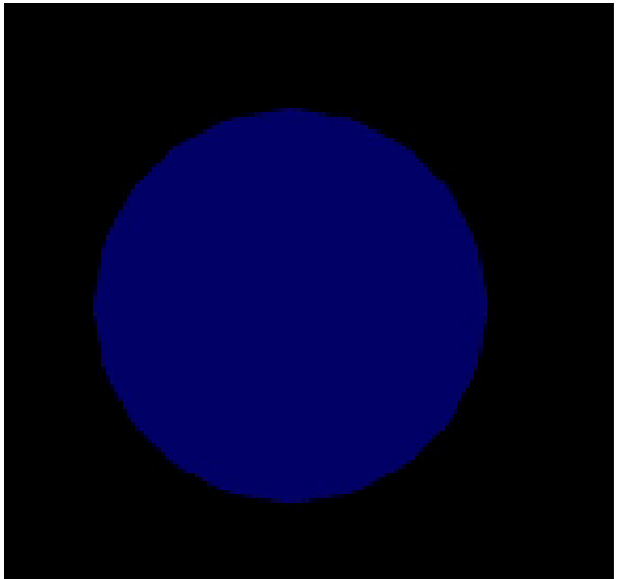


From the geometric model..... Toward its realistic visualization

Lighting

- In real world the colour of the objects that we see is the result of the incident light which hits the objects and of the way the object surface reflects the light.
- Wrong assumption: each surface is colored with a single color-> the object appear flat
- Many gradations or shades of color give 2D images the appearance of being 3D



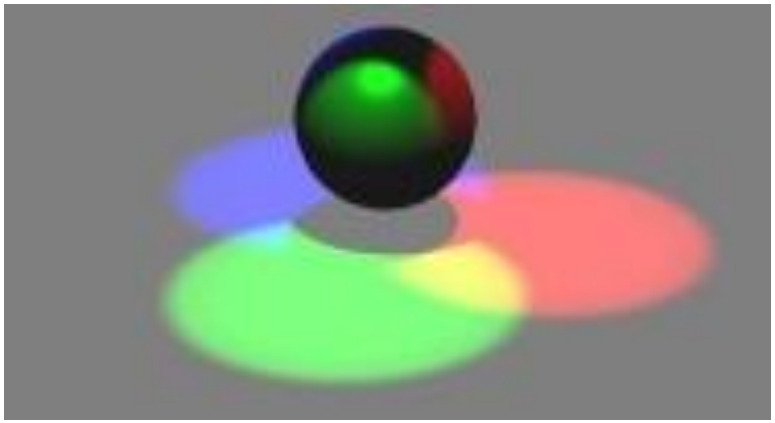


Two Components of Lighting

When we look at a point on an object, the color that we see is determined by multiple interactions between light sources and reflective surfaces.

Light Sources:

- Emittance spectrum (colour)
- Geometry (position and colour)
- Directional attenuation

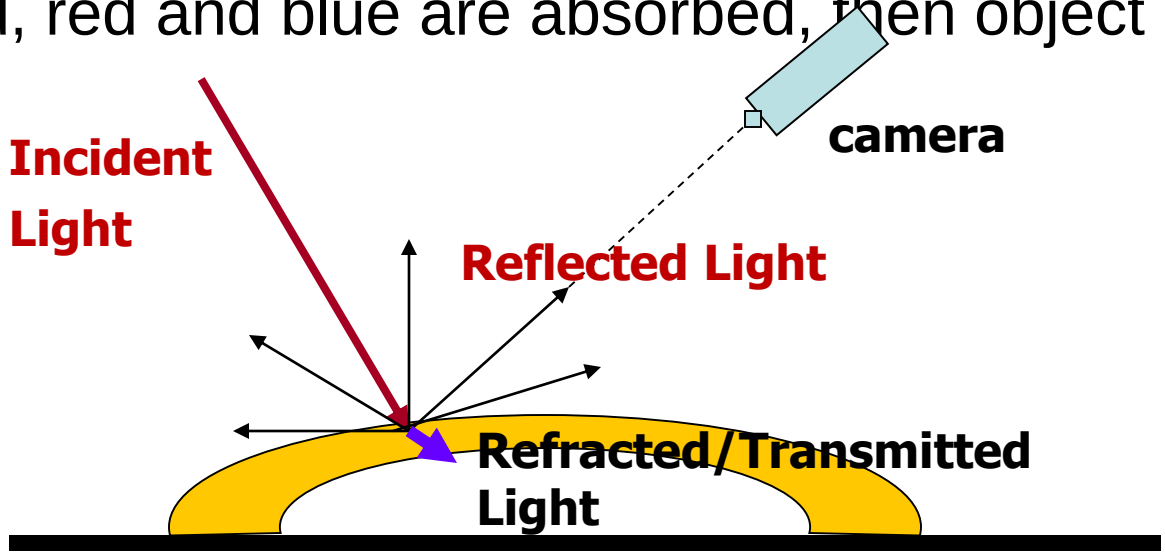


Surface Properties:

- Reflectance spectrum (colour)
- Geometry (position, orientation, and micro-structure)
- Absorption

Real World Lighting

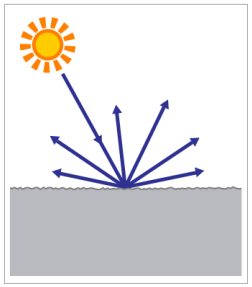
- Light sources emit photons.
- The incoming light from light sources is partially **absorbed**, partially **reflected** and partially **refracted**
- The reflected light defines the object color: if only green is reflected, red and blue are absorbed, then object appears green

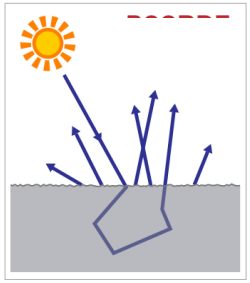


- Reflectance – a fraction of the incident light that is reflected

From conservation of energy:

Light incident at surface = reflected + absorbed + transmitted/refracted





- **Opaque object -** the majority of incident light is either the reflected or absorbed (transmitted light 0)
- **Translucent object** - significant light transmission

We do not consider: absorption, transmission, diffraction

Lighting models

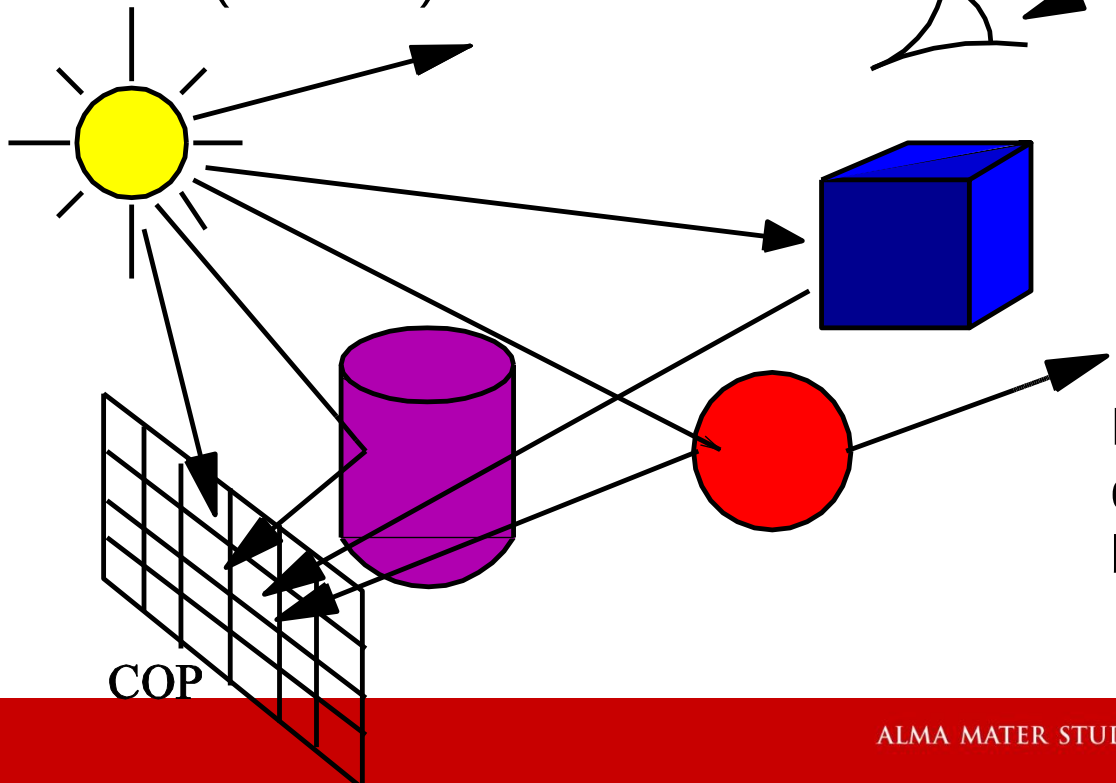
Empirical: simple approximations to observed phenomena.

Physically-based: model actual physics of light

- **Rendering Equation**: consider a global energy balance, it is a model of the conservation of energy that describe how light intercats with materials
- **Empirical model**: we follow rays of lights from a point sources that reach the viewer both directly, and by interacting with reflecting surfaces in the scene

To determine the color of a point we need to model

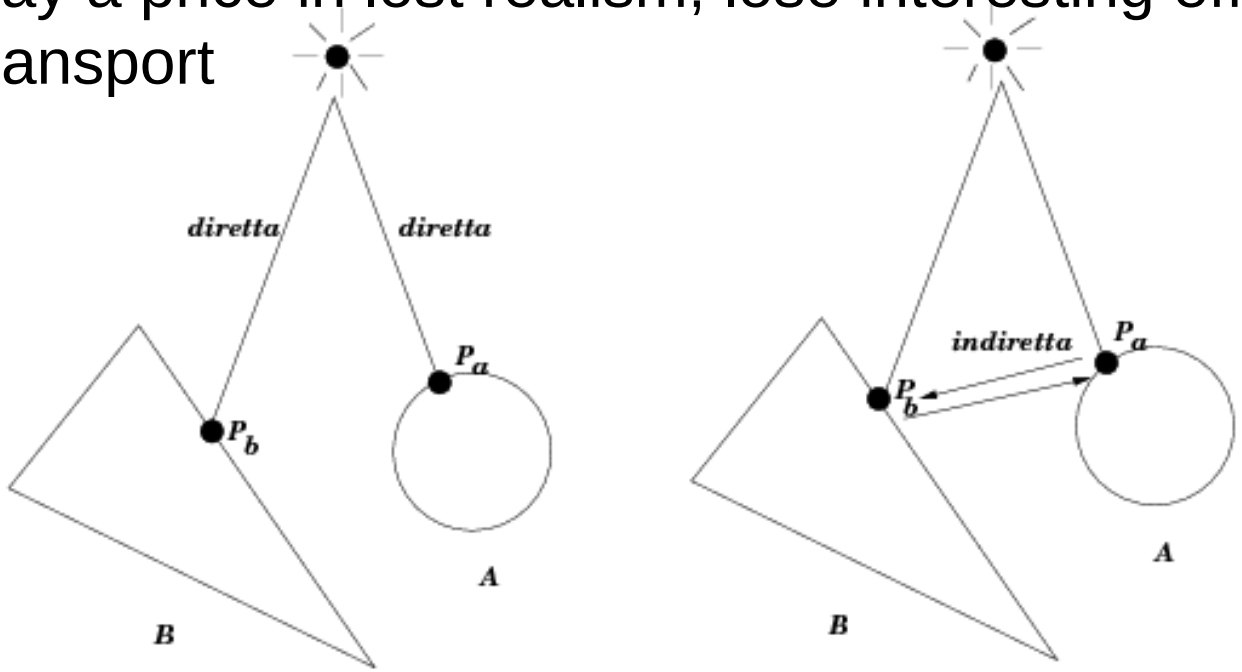
- the light sources in the scene
- the reflection properties of the surface (material)



Build an illumination model to describe the interaction between materials and lights

Local and Global Models

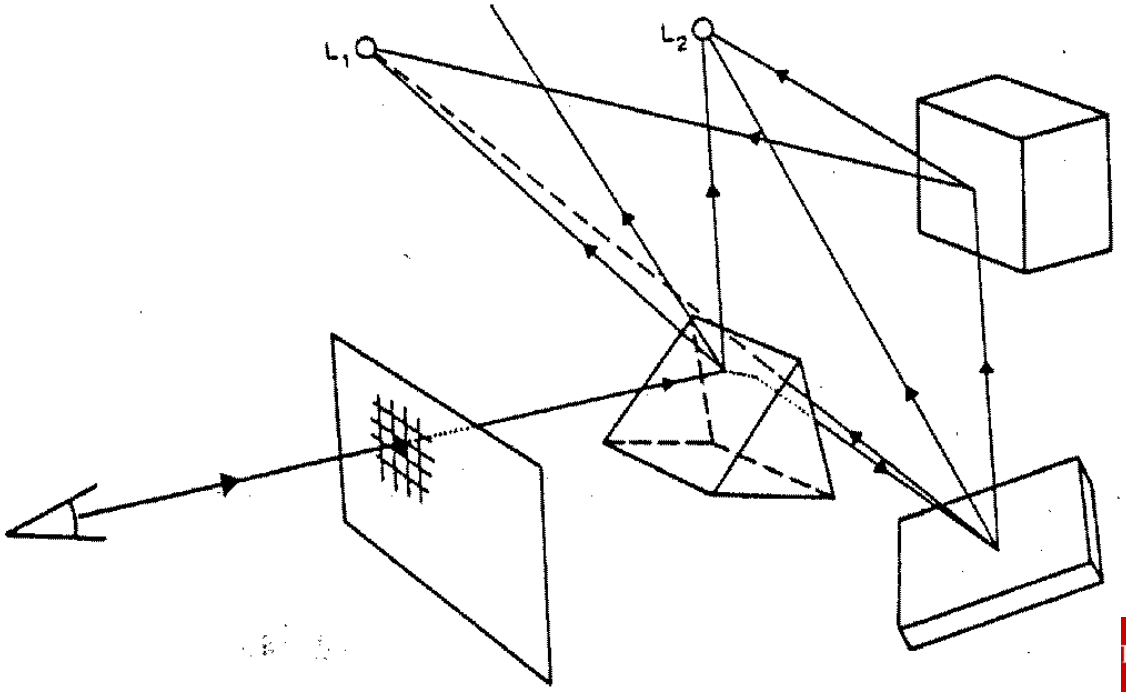
- **Local model:** the color of a surface element depends only on light from light sources and ignores effects of all other objects in the scene
  - pro: scene can be rendered much faster
  - con: pay a price in lost realism; lose interesting effects of light transport

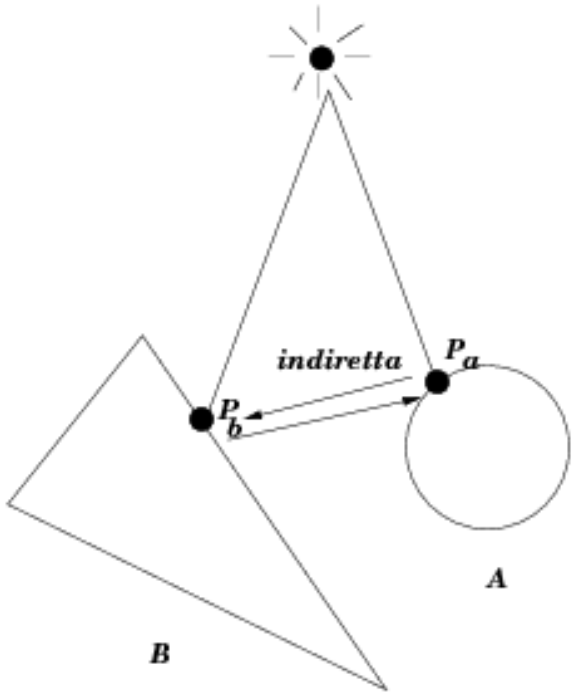


Local and Global Models

- **Global model:** the color of a surface element depends on the light from light sources and from light bounces off other objects toward our surface element

- This is not compatible with a rendering pipeline





Light-Material Interactions

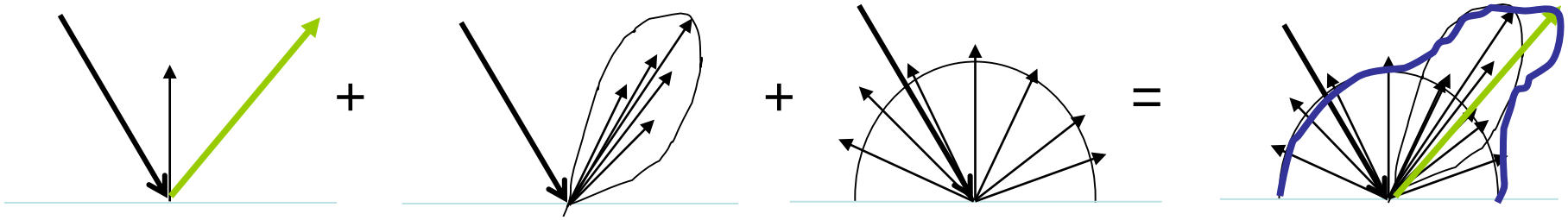
The light reflected depends on the surface material

How the object surface reflects the light

Specular: light perfectly reflected along one reflection direction

Gloss (Lucide): mixed, partial specularity

Diffuse: light scattered into all directions



specular + gloss + diffuse = reflectance distribution

Material Properties

Material Properties

- K_a ambient $K_{a,d,s,e}$ in [0,1]
- K_d diffuse - diffuse reflection coefficient:

Defines the fraction of light scattered into all directions,
Defines the color of the surface

- K_s specular - specular reflection coefficient:

Defines the fraction of light reflected near the mirror reflection direction

- K_e emissive
- n_s specular reflection exponent

(grado di specularità per superfici parzialmente speculari)

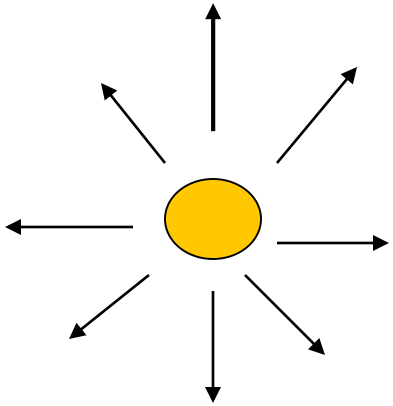
Light Sources

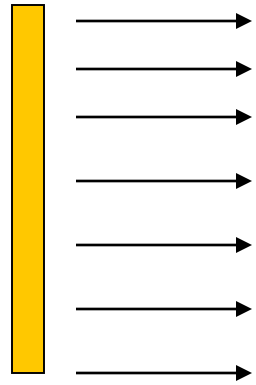
Each light in the scene needs to have its type specified (position, direction) as well as any other relevant properties:

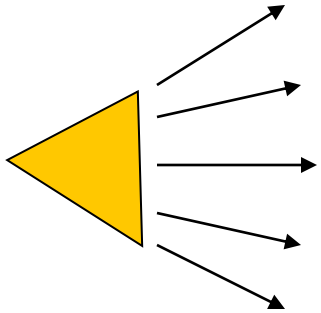
color (RGB), energy (light intensity)

Different Kinds of Light Sources:

1) Ambient 2) Point 3) Directional 4) Spot



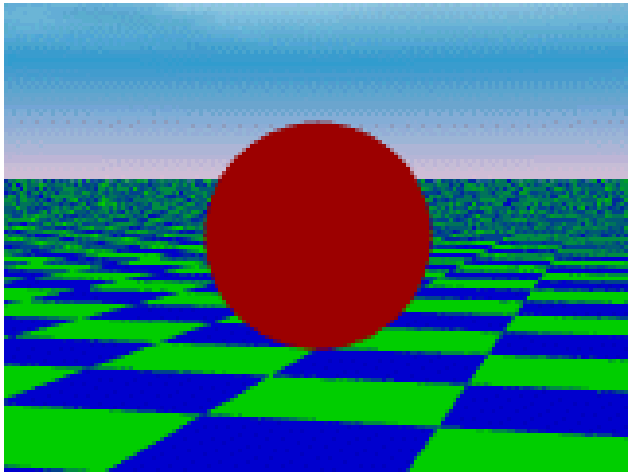




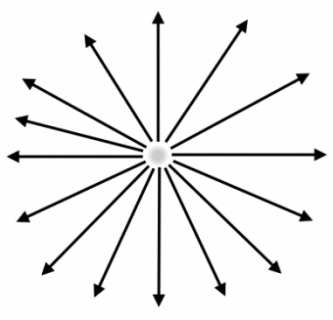
1) Ambient Light

Represents a fixed-intensity light source that hits all objects equally

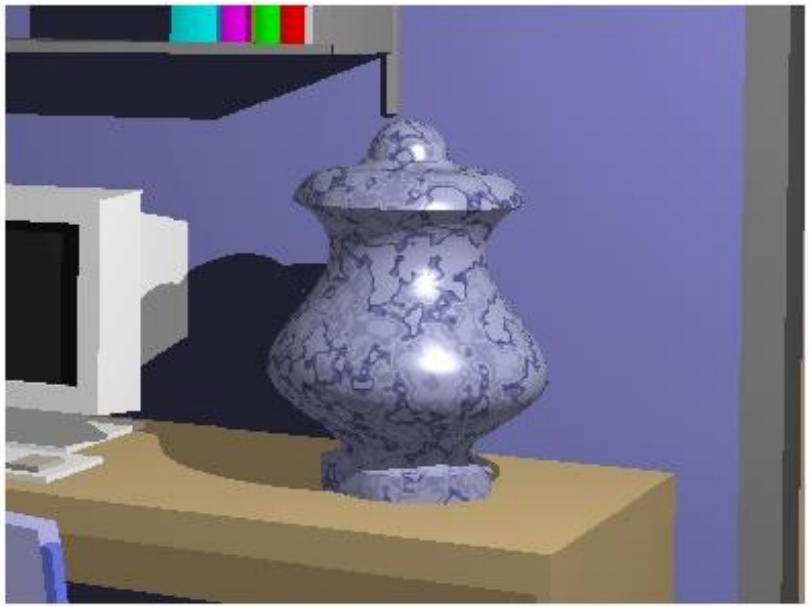
- Used to uniformly brighten/darken the entire scene
- Simulate the indirect light
- Coming from all directions
- Useful as a “hacky” model for indirect lighting so shadows aren’t completely



2) Point light
(Luce omnidirezionale-punto luce)

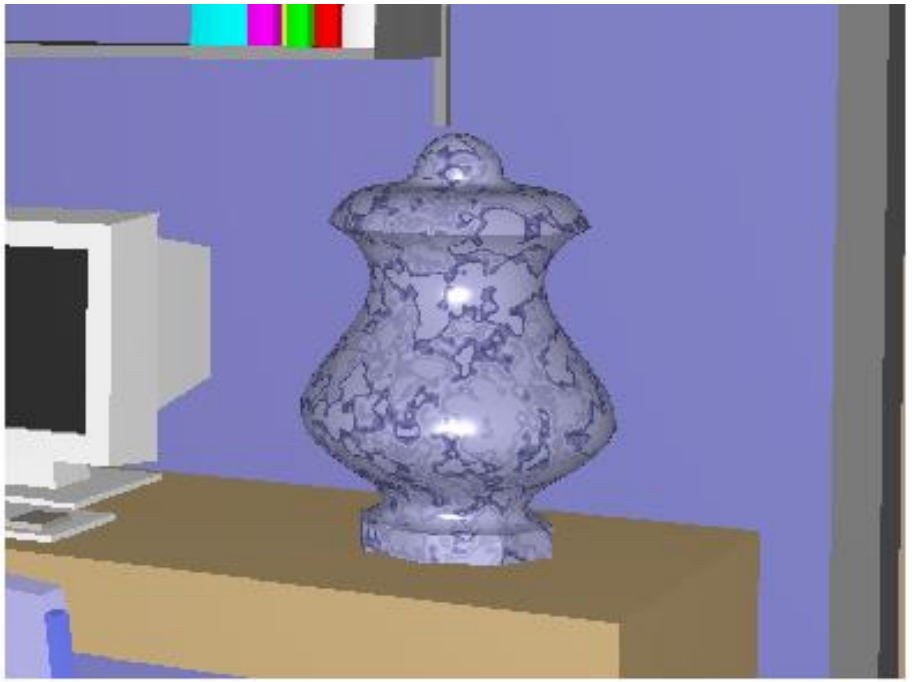


- Point (positional) light source P=[x,y,z,1]
- By default, radiate in all directions with constant intensity (lamp)

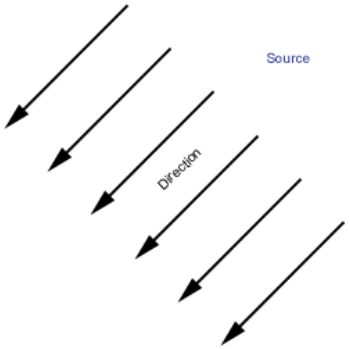


Point light located near the vase

Point light located at the viewer position, there are no shadows

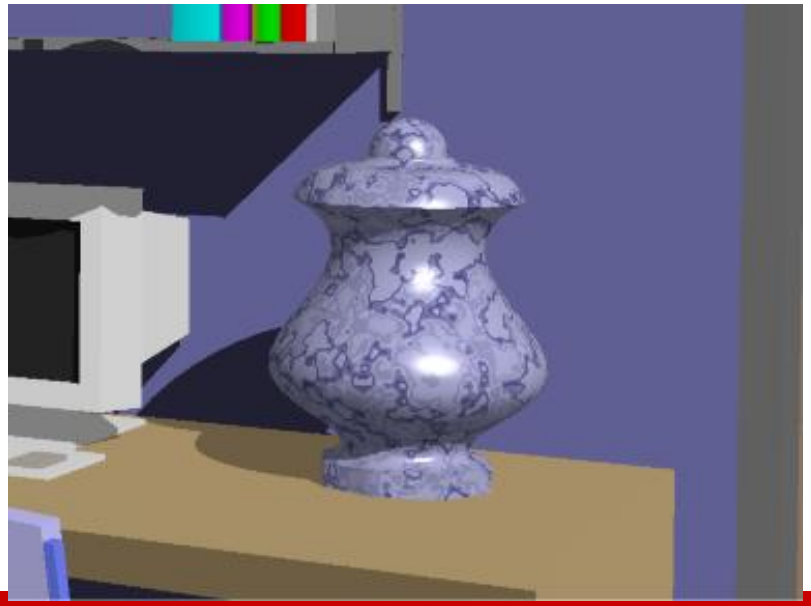


3) Directional light



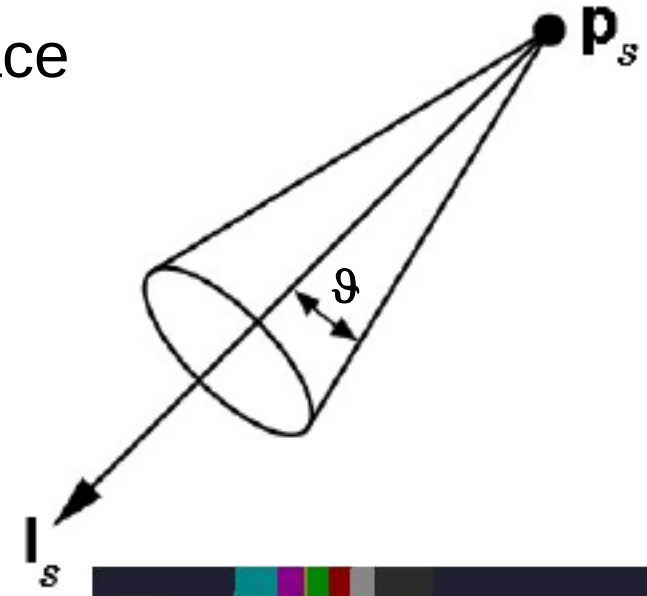
Lights all objects equally, but only from a specified direction

- Located infinitely far away
- Simulate distant light sources (Sun)
- Specified as a vector P=[x,y,z,0]



4) Spotlight (luce riflettore)

Light emitted from a single point in space

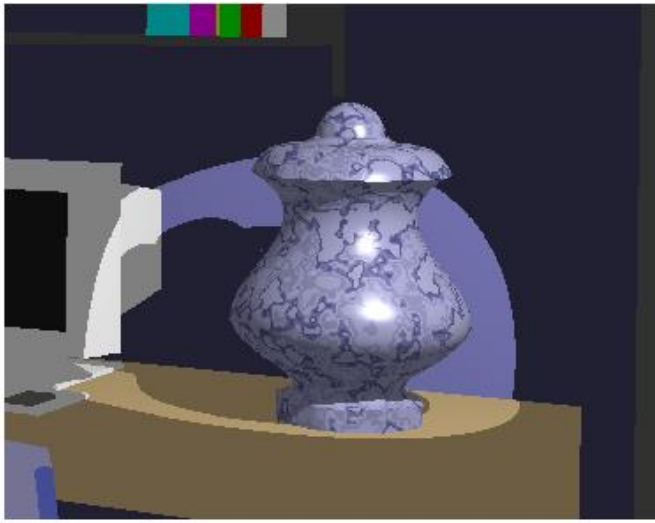


- Spreads outwards in a cone

whose apex is at $\mathbf{P_s}$,
which points in the direction $\mathbf{l_s}$, and
whose width is determined by an angle ***teta.***

- Angular subset of a point light

The light intensity is concentrated in the center of the cone and it drops off towards (ϑ,-ϑ)
(exponent of attenuation of a spotlight)



Illumination model (Reflection Model)

- Different materials reflect light in different patterns
- The way a particular material reflects light is referred to as a *reflection model*
- The reflection model takes the direction and color of the incident light coming from a particular light source and computes what color of light is going to be reflected in the direction of the camera
- *Phong's local illumination model*

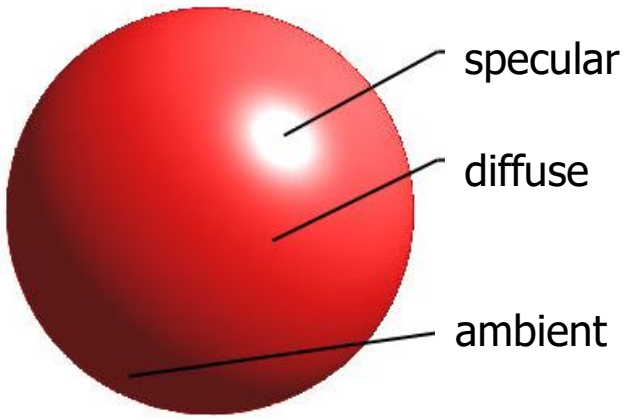
Phong's local illumination model (Phong Bui-Tuong 1975)

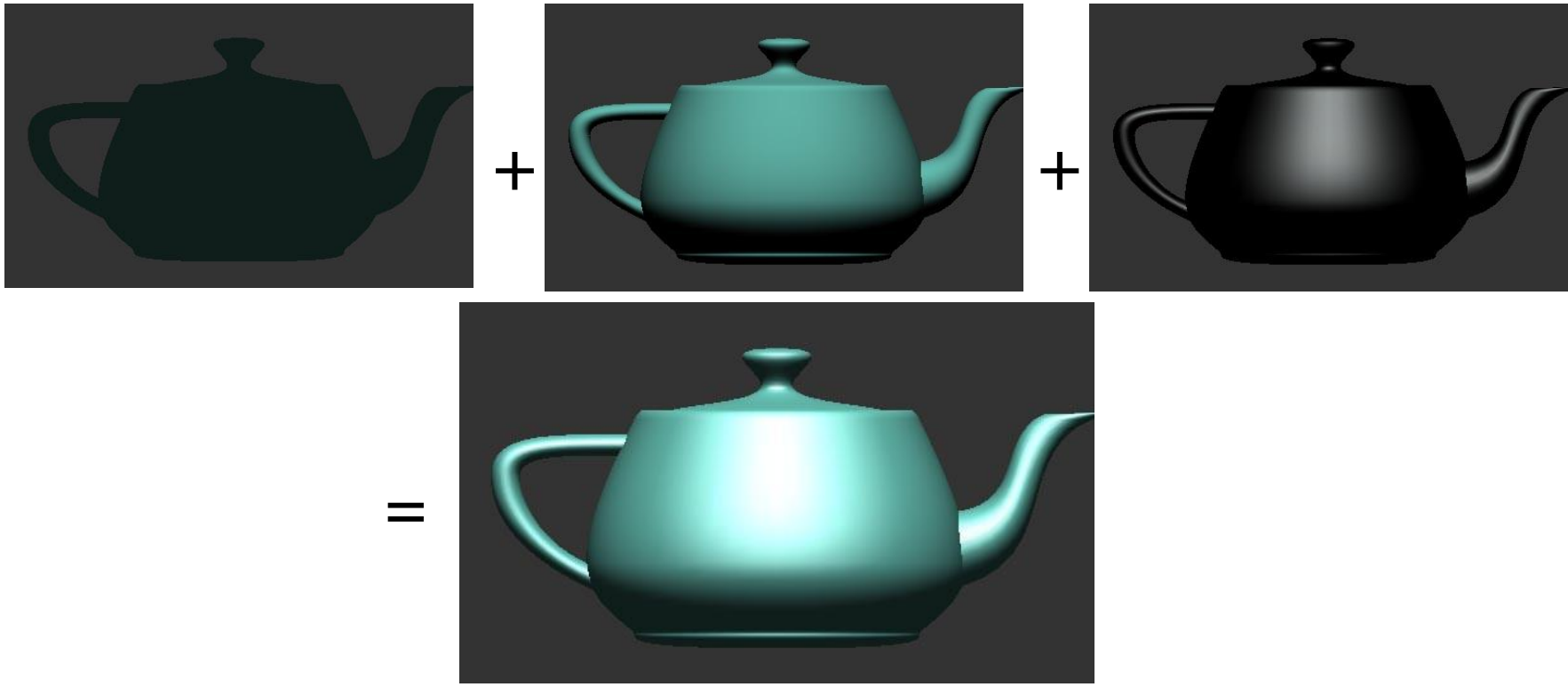
The model only considers light coming directly from light sources (called *direct light*)
Sum of three components:

diffuse reflection +specular reflection +"ambient" + emissive

Phong's Equation at a surface point:

$$I_\lambda = k_e + I_a k_a + I_d k_d + I_s k_s$$
$$0 \le k_e, k_a, k_d, k_s \le 1$$





Lighting: Emissive component

- Light sources: sun, fire, light bulbs etc.
- Objects can emit light by self-emission
- The object is a point light source that emits light uniformly in all directions
- k_e emissive coefficient

(luminosità intrinseca dell'oggetto)

$$I = k_e \qquad 0 \leq k_e \leq 1$$

Lighting: Ambient component

- Ambient light component accounts for non-specific global light (indirect light from other objects) scattered in all directions, to guarantee a minimum uniform light level in the scene
- It does not depends on the viewer position or surface position, it is constant

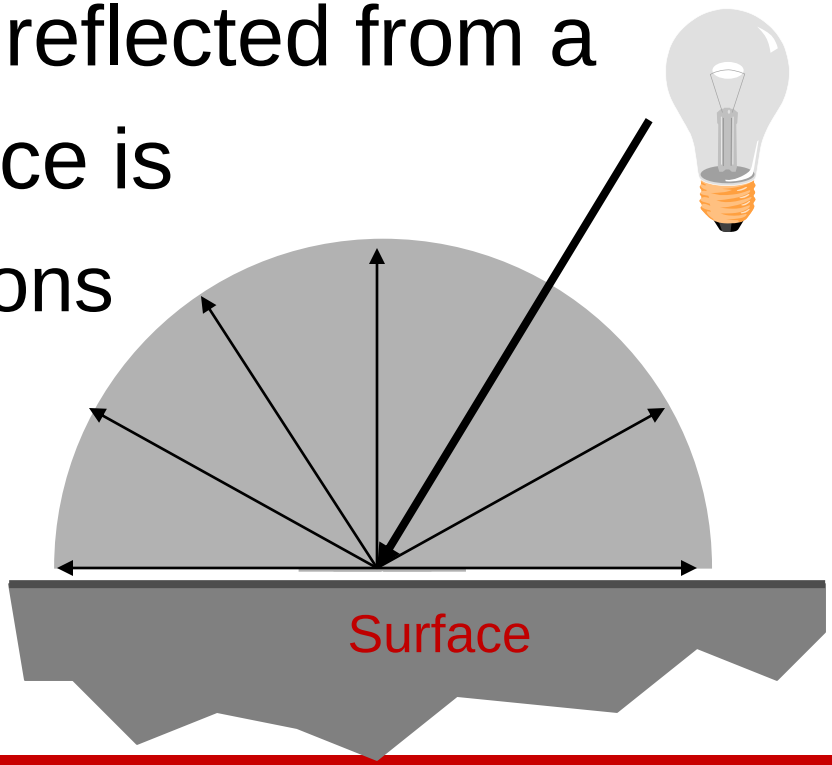
$$I = I_a k_a \qquad 0 \leq k_a \leq 1$$

I_a ambient intensity, identical at every point in the scene, and it is the same at every point on the surface. Some of this light is absorbed and some is reflected. The amount reflected is given by

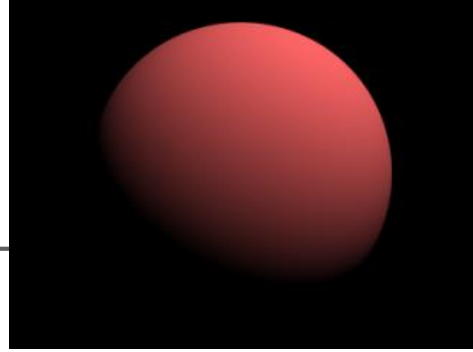
k_a ambient reflection coefficient and is characteristic of a given material

Lighting: Diffuse component

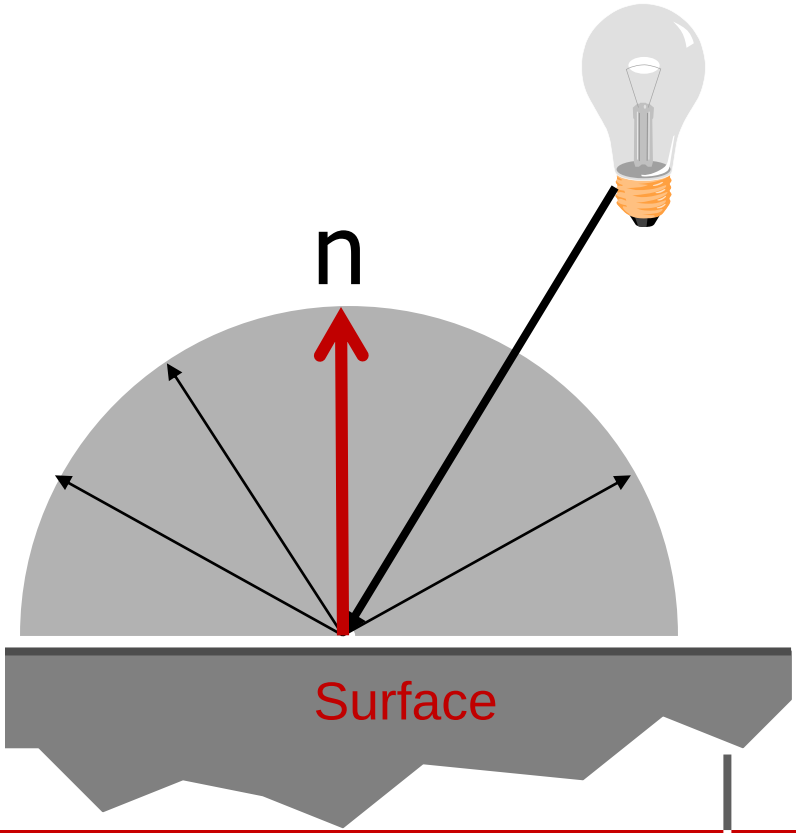
- An ideal diffuse surface is, at the microscopic level, a very rough surface.
  - Example: chalk, clay, some paints, wood
- Amount of light reflected from a point on the surface is equal in all directions



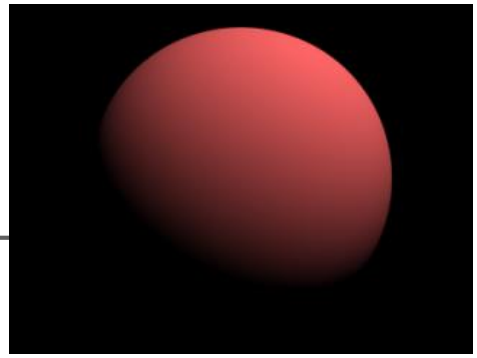
Lighting: Diffuse component



- Reflection dependent on
  - orientation of surface
  - light source position
- light is reflected equally in all directions
- The brightness is independent of the viewing direction (camera position)



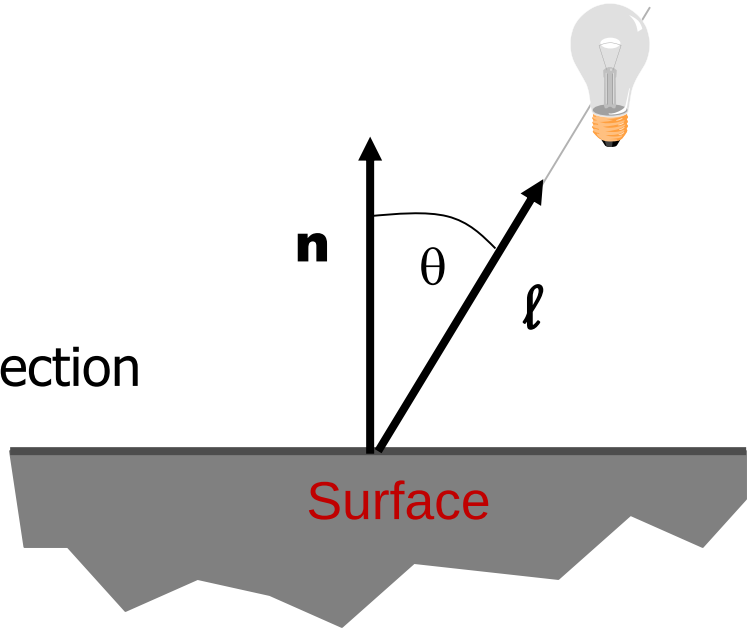
Lighting: Diffuse component



Ideal diffuse reflectors reflect light according to Lambert's cosine law. Lambert's law determines how much of the incoming light energy is reflected. Reflected intensity:

$$I_\lambda = I_d k_d = k_d \left[I_\lambda \cos\theta \right] =$$

$$= k_d (l \cdot n) I_\ell$$



θ Angle between normal and light direction

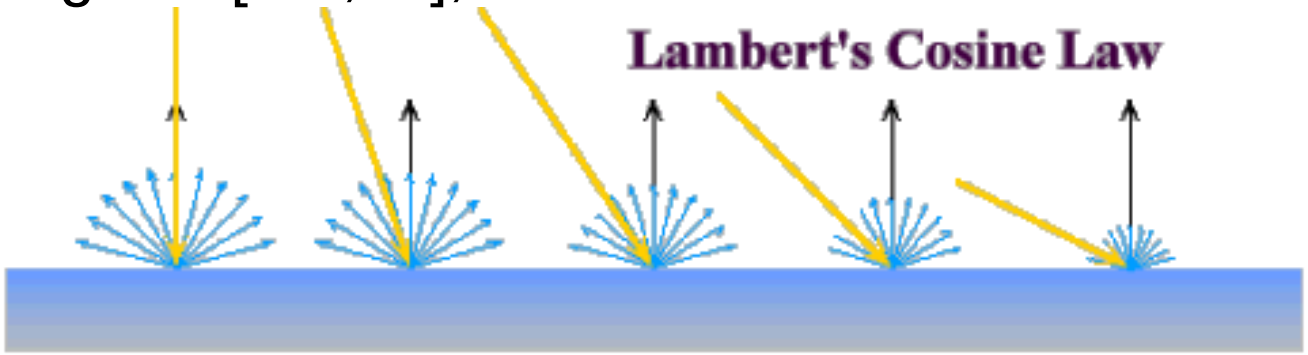
I_ℓ Source light intensity

k_d Material diffuse coefficient

Normal n = (nx, ny, nz)
Direction of the light source l = (lx, ly, lz)

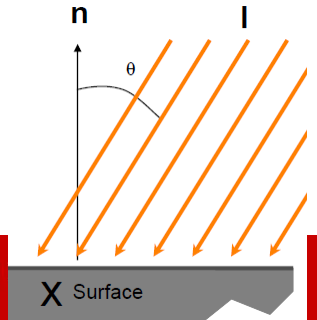
Lighting: Diffuse component

- For the Lambert's law the *reflected intensity* increases for angles approach to zero.
- Angle in [-90,90];



$$I_\lambda = k_d \max(0, l \cdot n) I_\ell$$

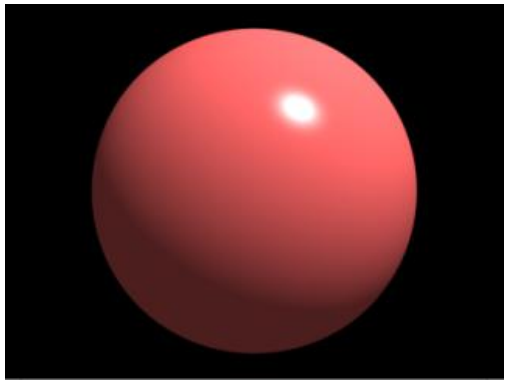
- <u>if the dot product is negative</u>, indicating that the light is on the wrong side of the surface, we clamp it to zero
- For **point light** the vector l changes from point to point on the surface
- For **directional light** is constant

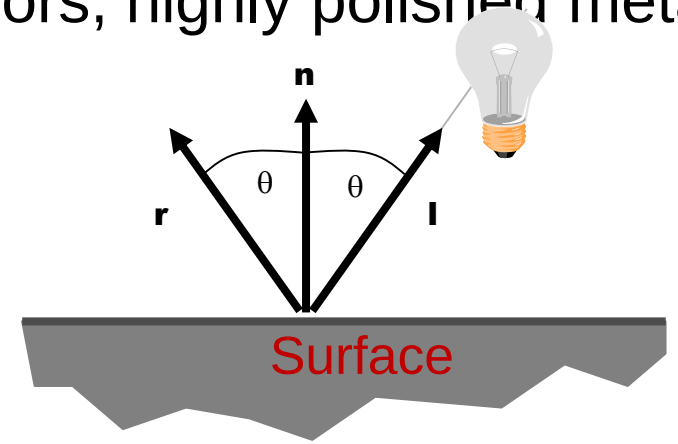


Lighting: Specular component

- Highlights that we see reflected from shiny objects
- An ideal mirror surface reflects the light in the ideal reflection direction only.
- Ideal mirror reflector

  the light is reflected in the ideal reflection direction **r** , which is obtained by reflecting **l** (light direction) with respect to **n** (surface normal)
- Examples: mirrors, highly polished metals, plastics.





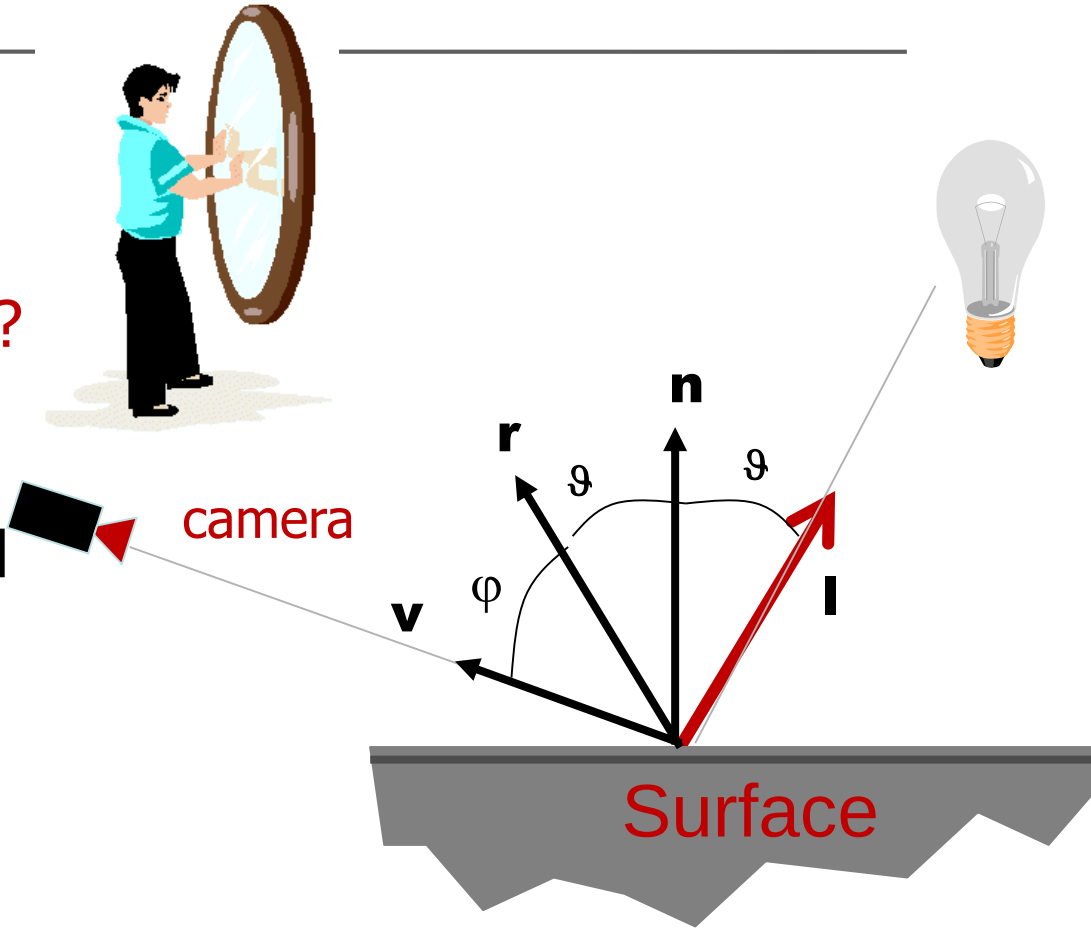
Lighting: Specular component

View dependent

How much light is reflected?

Depends on the angle between the ideal reflection direction r and the viewer direction v.

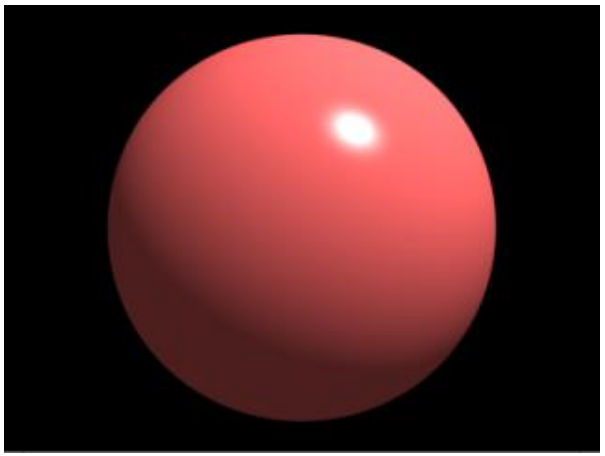
Max φ=0 (r=v) and it decreases for increasing angle φ.



$$I_\lambda = I_s k_s = k_s I_\ell \cos^{n_s} \varphi = k_s I_\ell (v \cdot r)^{n_s}$$

Non-ideal Reflectors

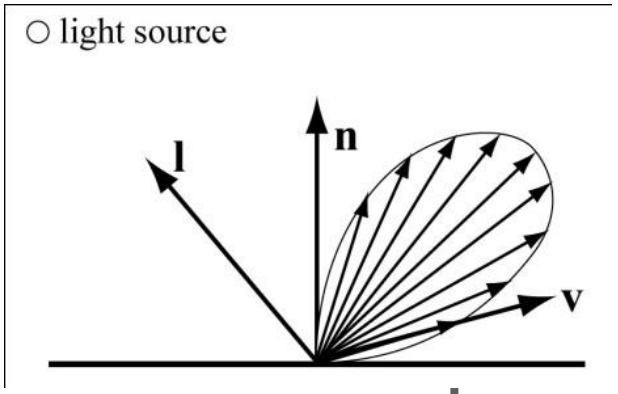
Specular: simulates a highlight



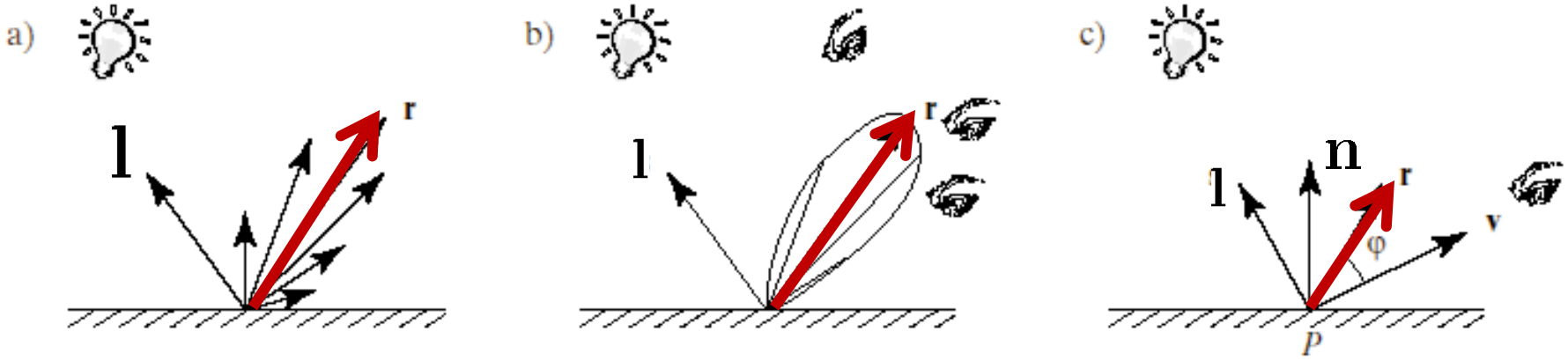
HIGHLIGHT (bright spot) in shiny surface: area which reflects the incident light colour and is not affected by the material colour

Real materials tend to deviate significantly from ideal mirror reflectors because of microscopic surface variations.

We might expect some of the light to be reflected just slightly offset from the ideal reflected ray.



Non-ideal Reflectors

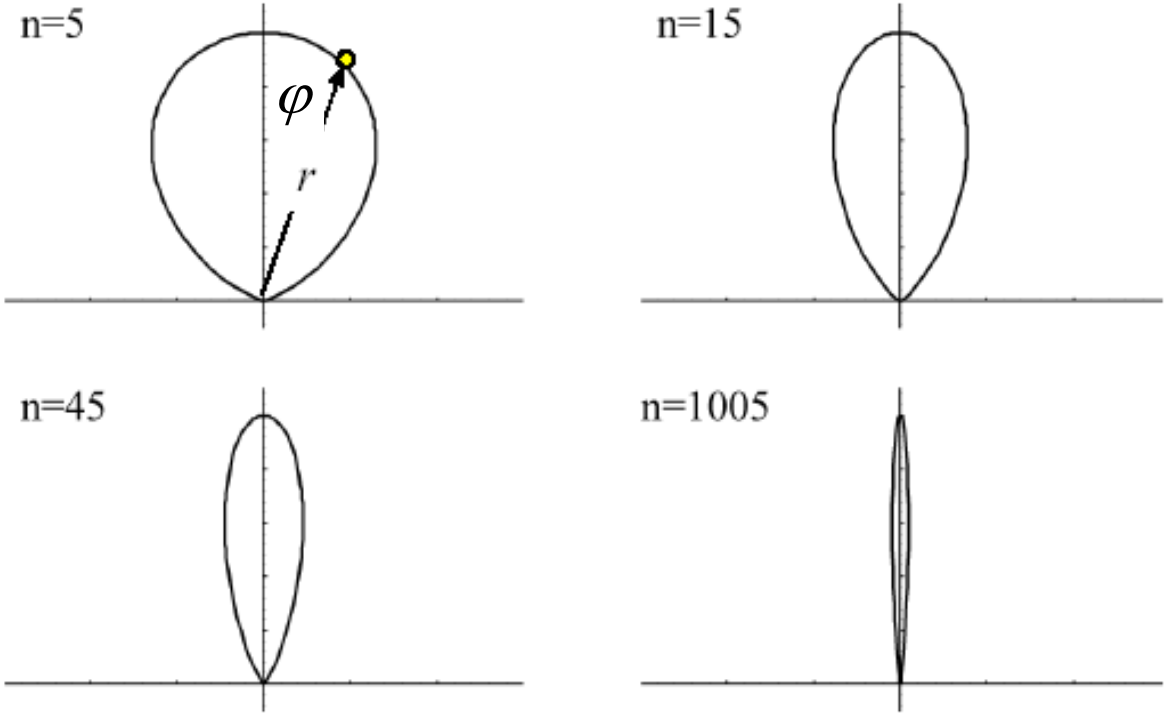


- The reflected light is scattered near the ideal reflection direction – Highlight is blurry
- **Specular lobe**: The farther away the viewing direction **v** is from the reflection direction **r**, the less light reflected is visible (the highlight descreses)

Shininess coefficient or Phong's exponent n_s

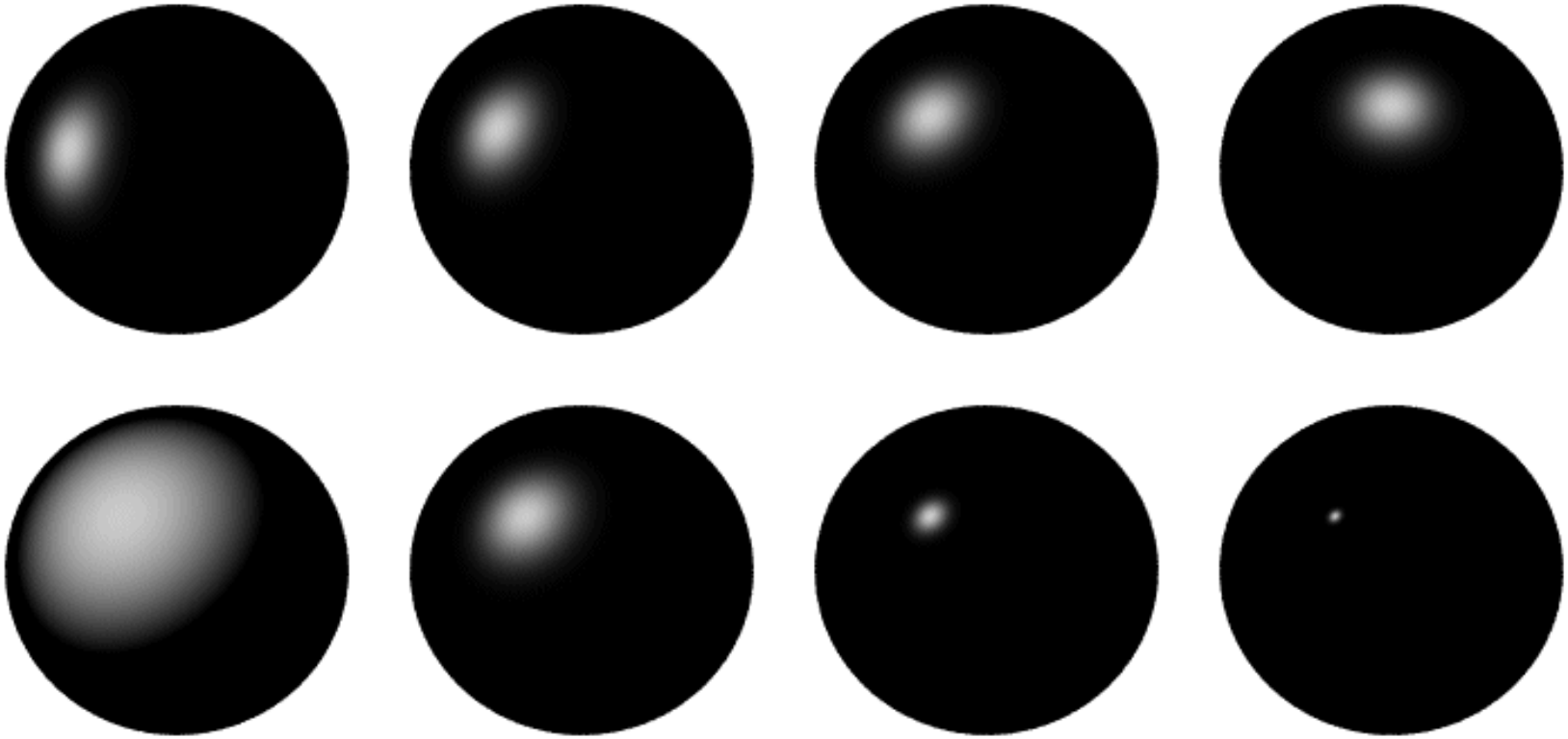
As n_s is increasing, the reflected light is concentrated in a narrower region centered on the perfect reflector

$r = \cos^n \varphi$

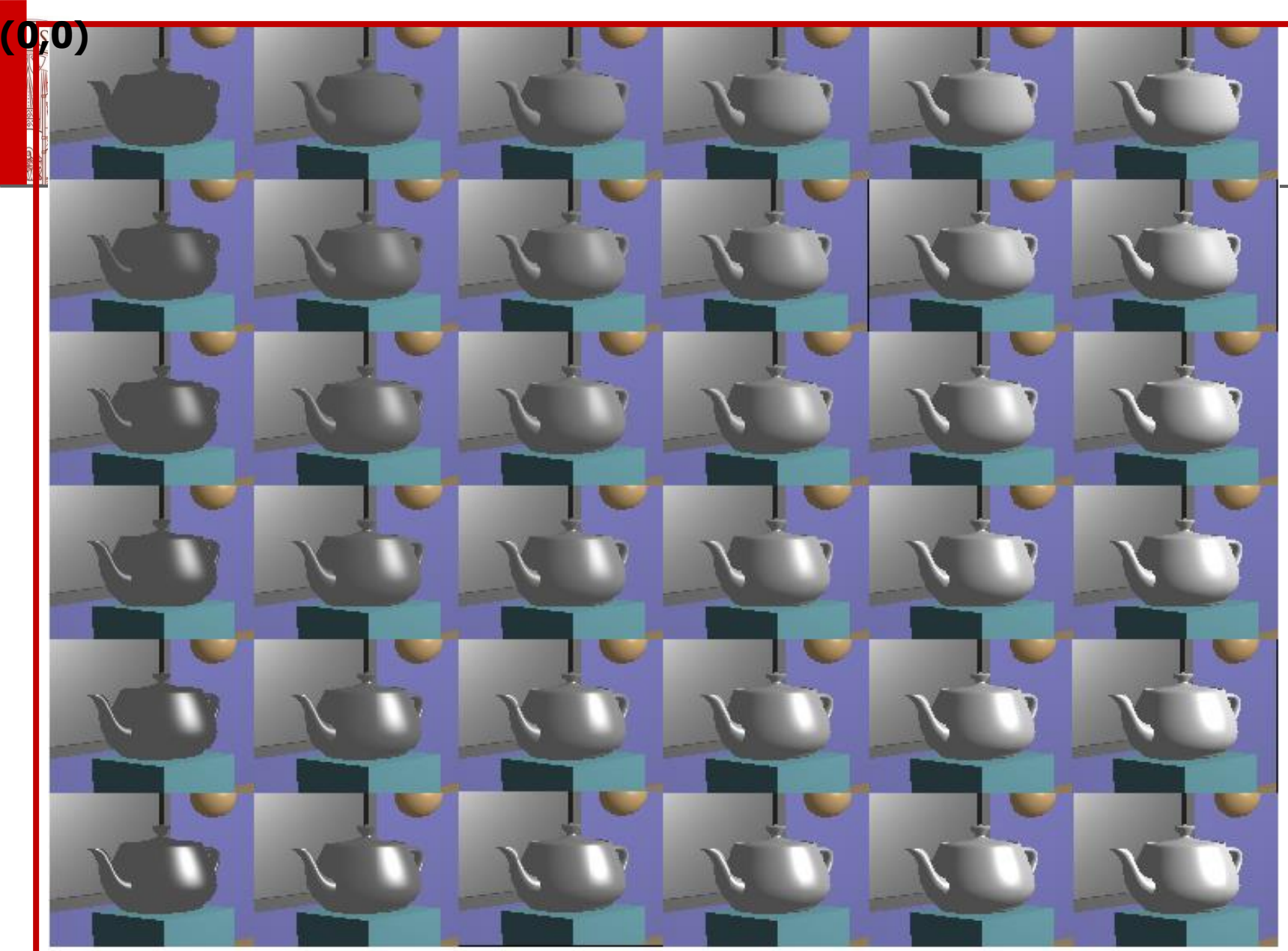


n_s shininess coefficient of the material (Specular exponent)

Phong model – the effect of n_s



As n_s goes to infinity, we get a mirror,
n_s in [100,500] metallic surfaces, n_s <100 broad highlights



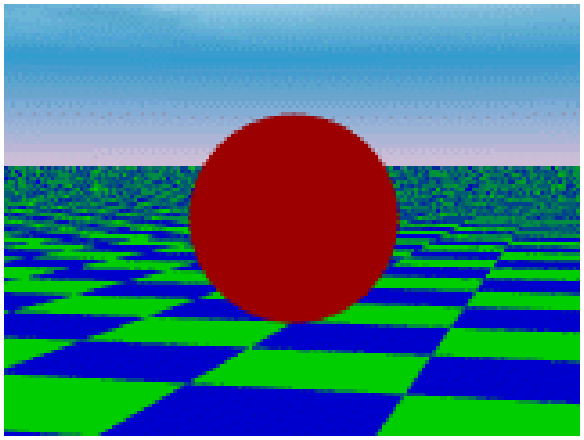
Ka =0.7

Putting it all together..

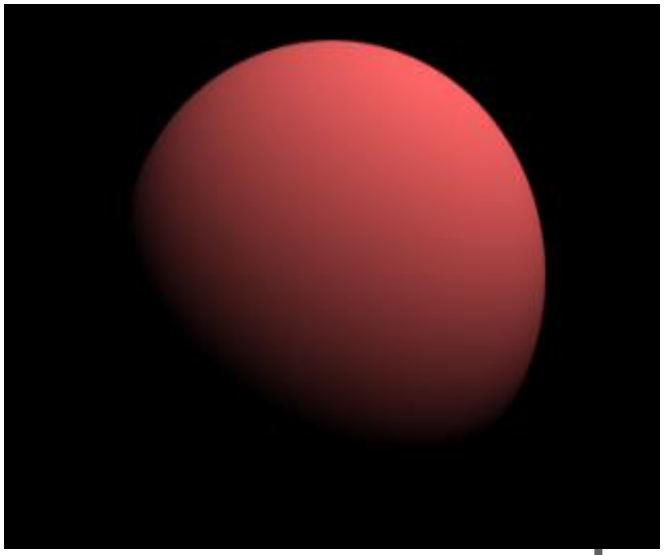
The complete Phong Illumination Model

$$I_\lambda = I_a k_a + I_l [k_d \cos\theta + k_s \cos^{n_s} \varphi]$$

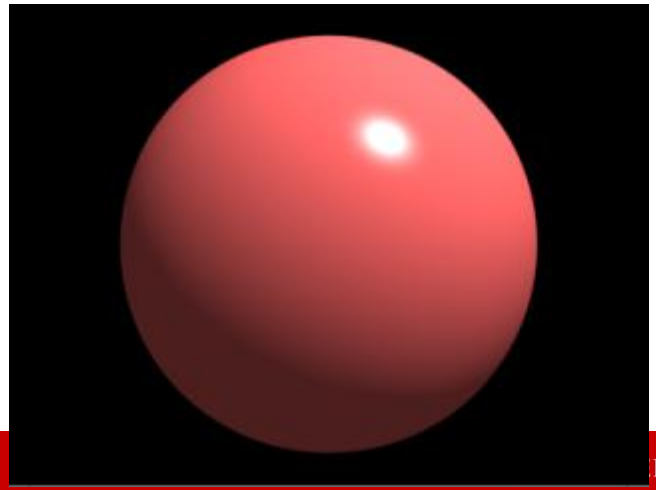
Ambient



Diffuse



Specular



Extend the Phong's model

To account for multiple lights, we just sum up the contribution from each individual light

$$I_\lambda = I_a k_a + \sum_{l=1}^{lights} I_l [k_d (n \cdot l) + k_s (v \cdot r)^{n_s}]$$

Extend to colored light source and object

$$I_{\lambda r} = I_{a_r} k_a + \sum_{l=1}^{lights} I_{l_r} [k_{dr} (n \cdot l) + k_s (v \cdot r)^{n_s}]$$

$$I_{\lambda g} = I_{a_g} k_a + \sum_{l=1}^{lights} I_{l_g} [k_{dg} (n \cdot l) + k_s (v \cdot r)^{n_s}]$$

$$I_{\lambda b} = I_{a_b} k_a + \sum_{l=1}^{lights} I_{l_b} [k_{db} (n \cdot l) + k_s (v \cdot r)^{n_s}]$$

The specular reflected coefficient Ks is constant for white light

Attenuation term

For point lights:

- To account for attenuation of the intensity light as the light travels a distance **s** from the source to the surface,
- Attenuation is inversely proportional to the quadratic of the distance

$$I_\lambda = I_a k_a + \sum_{l=1}^{lights} \frac{I_l}{s^2}[k_d(n \cdot l) + k_s(v \cdot r)^{n_s}]$$

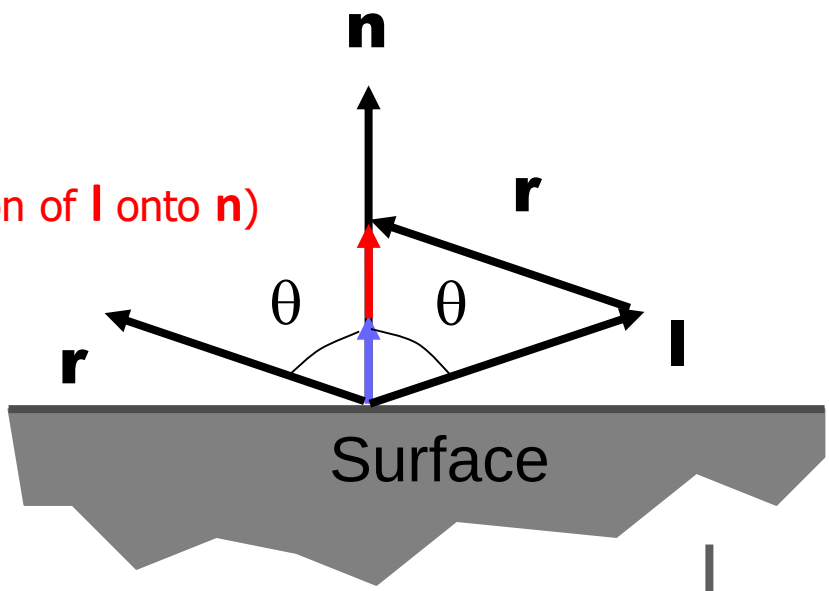
Determination of the Reflected Vector

Calculate **r** from **n**, **l** via

$$\mathbf{r} + \mathbf{l} = 2 \left\| l \right\| \cos\theta \ \mathbf{n}$$

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

(twice the projection of **l** onto **n**)



Blinn-Torrance Variation of Phong

In Phong shading, one must continually recalculate the dot product (***r,v***). If instead one uses the "halfway vector" **h** between **l** and **v:**

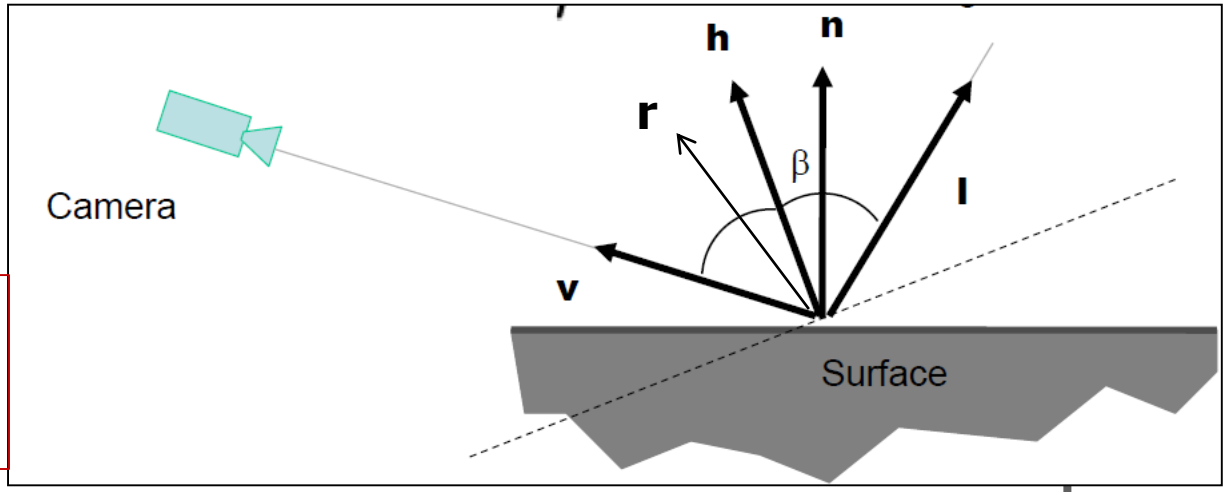
$$h = \frac{l + v}{\|l + v\|}$$

then the reflected light in Phong's model is given by

$$I_l k_s (n \cdot h)^{n_s}$$

$$I_l k_s (\cos \beta)^{n_s}$$

The angle β is half of the angle represented by Phong's dot product

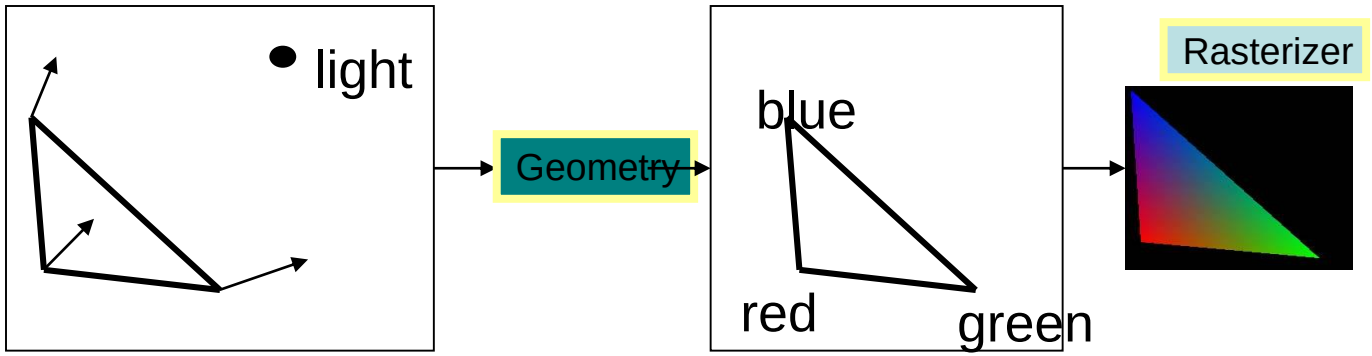


Lighting vs shading

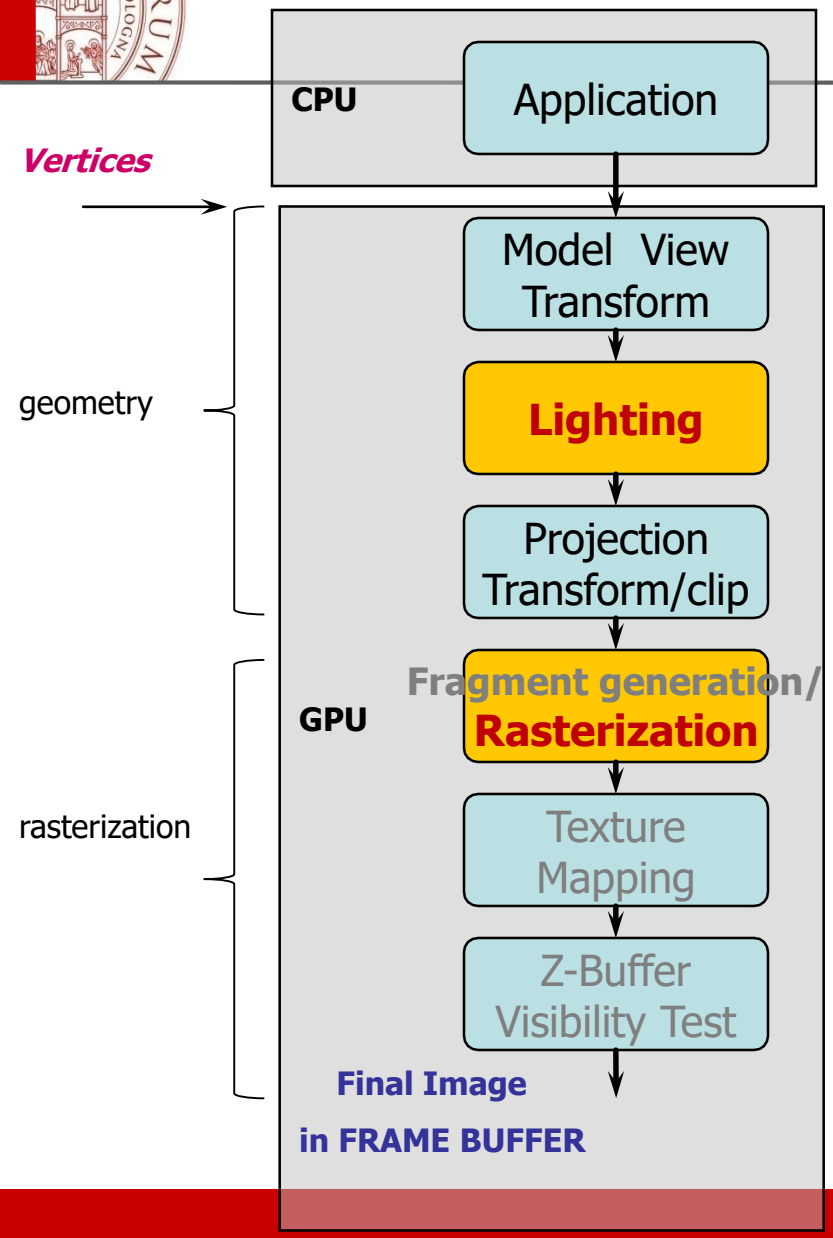
Illumination: transport of luminous flux from light sources via direct & indirect paths.

Lighting (per vertex): process of computing the luminous intensity (i.e., outgoing light) at a particular 3-D point, usually on a surface. On a curved surface the lighting changes from point to point

Shading (per fragment): assigning pixel colour. (How the lighting is used to color the pixels)



Geometry stage: Lighting/Shading



Lighting

Before Projection Transform which could deforme normal vectors; done in VCS coord.

Shading In rasterization a color per pixel is computed by interpolating the colors of the vertices

Shading for polygons

- Shading Models give a technique to determine the colors of all the pixels covered by a surface using appropriate illumination model
- IDEAL:
  - determine surface visible at each pixel
  - compute normal of the surface
  - evaluate light intensity and color using an illumination model

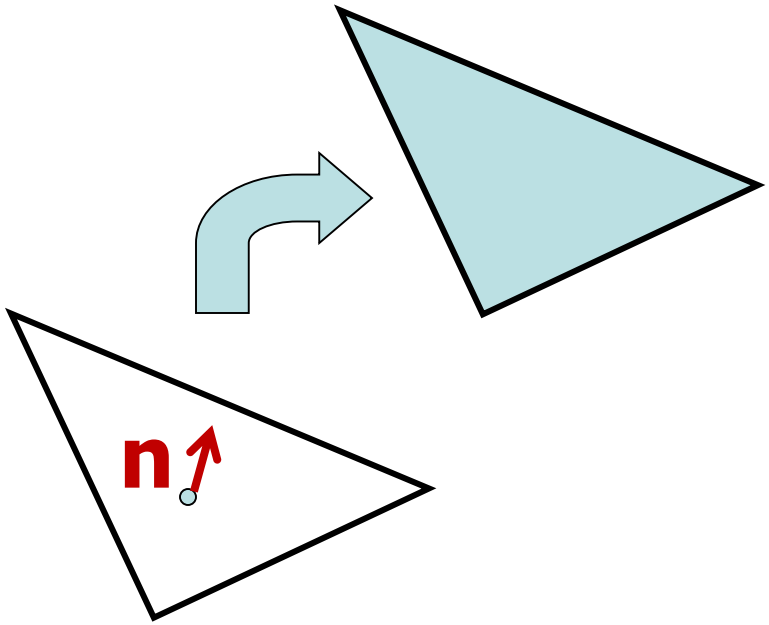
  This is quite expensive!

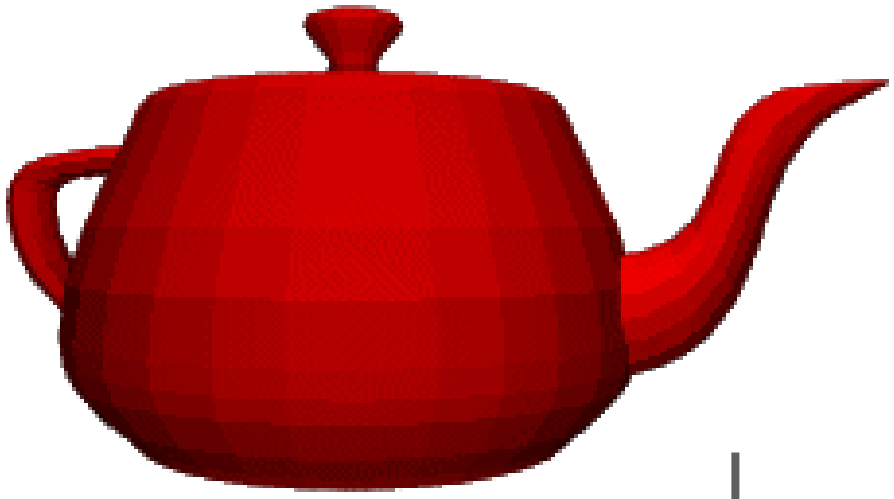
- REAL: approximate with **shading.**

  if surface defined as mesh of polygonal facets, *which surface points should we use ?*

Flat Shading

- Compute lighting at one vertex per polygon, use same value for every pixel in polygon
- obviously inaccurate for smooth surfaces

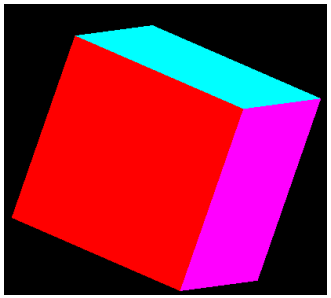


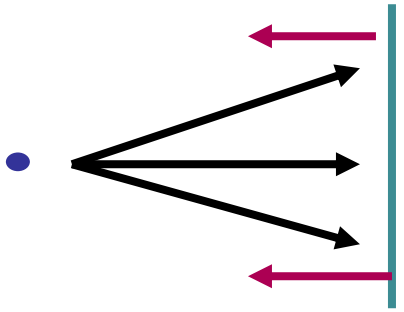


Flat Shading Approximations

- if an object really is faceted, is this accurate?
- no! It supposes constant l,n,v on a polygon
  - KO for point sources, the direction to light **l** varies across the facet

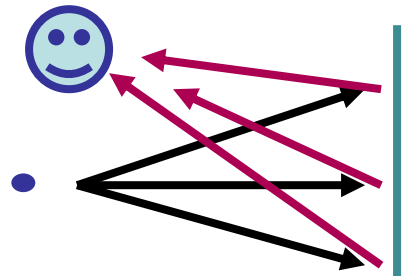
    (OK for directional light)





  - KO for specular reflectance, direction to eye **v** varies across the faces

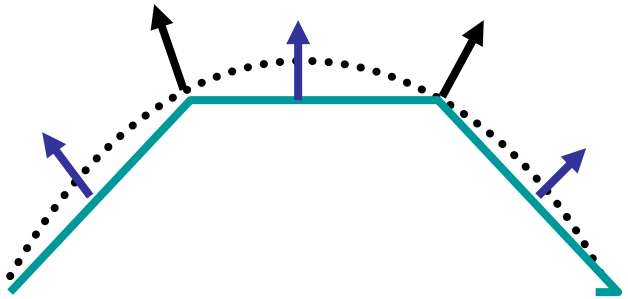
    (OK viewer at infinity – parallel proj.)



Improving Flat Shading

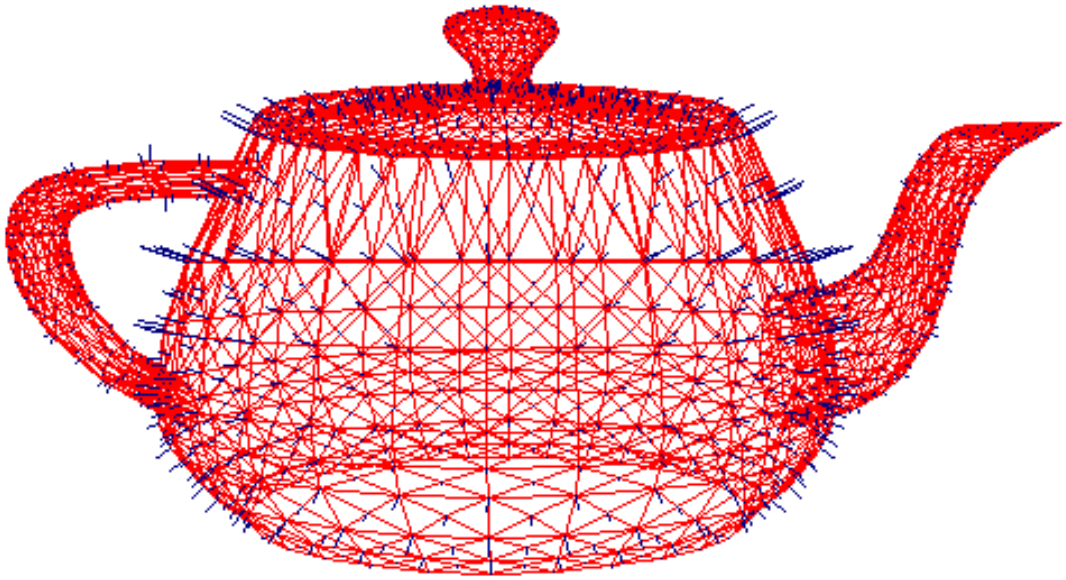
- what if evaluate Phong lighting model at each pixel of the polygon?
  - better, but result still clearly faceted

- we introduce *vertex normals* at each vertex
  - usually different from facet normal
  - used *only* for shading
  - Target: make a better approximation of the *real* surface that the polygons approximate

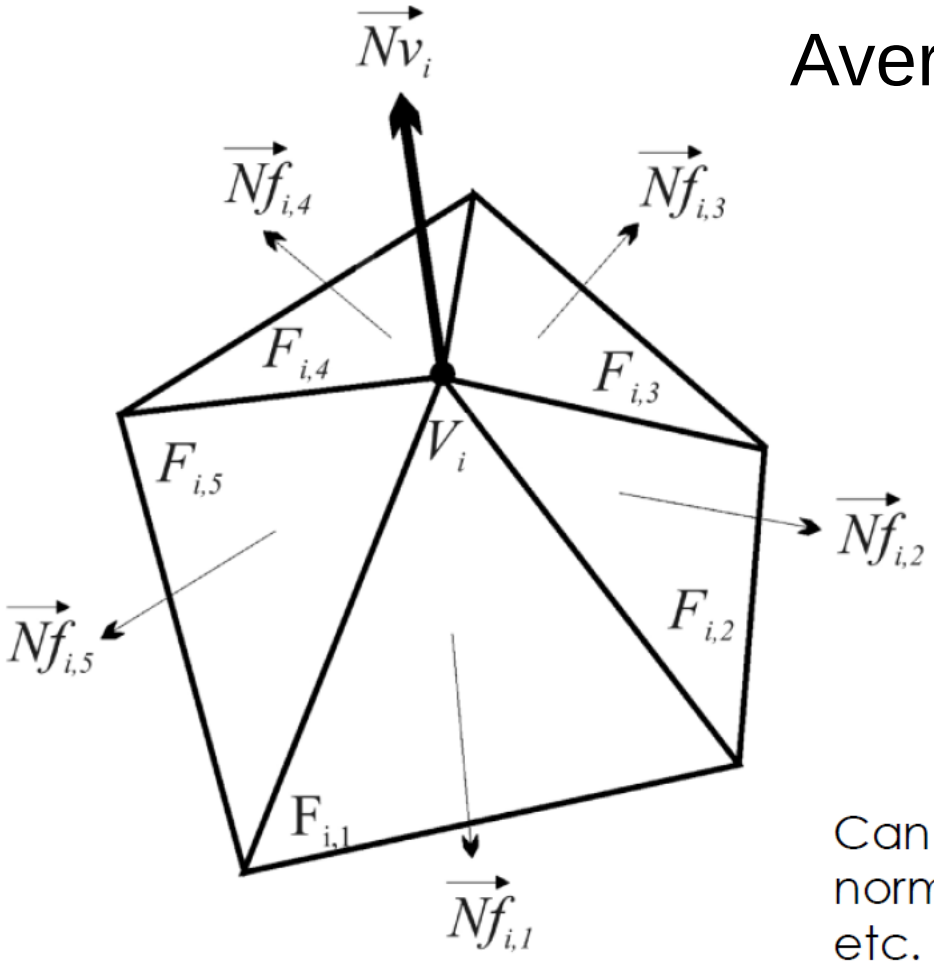


Vertex Normals

- vertex normals may be
  - provided with the model
  - computed exactly on the model (es. Spline)
  - approximated by averaging the normals of the facets that share the vertex



Approximate the vertex normals



Average of face normals

$$\overrightarrow{Nv_i} = \frac{\sum_{k=1}^{n} \overrightarrow{Nf_{i,k}}}{n}$$

Can weight triangle normals by angle, area, etc.

Assume to know which faces share a given vertex

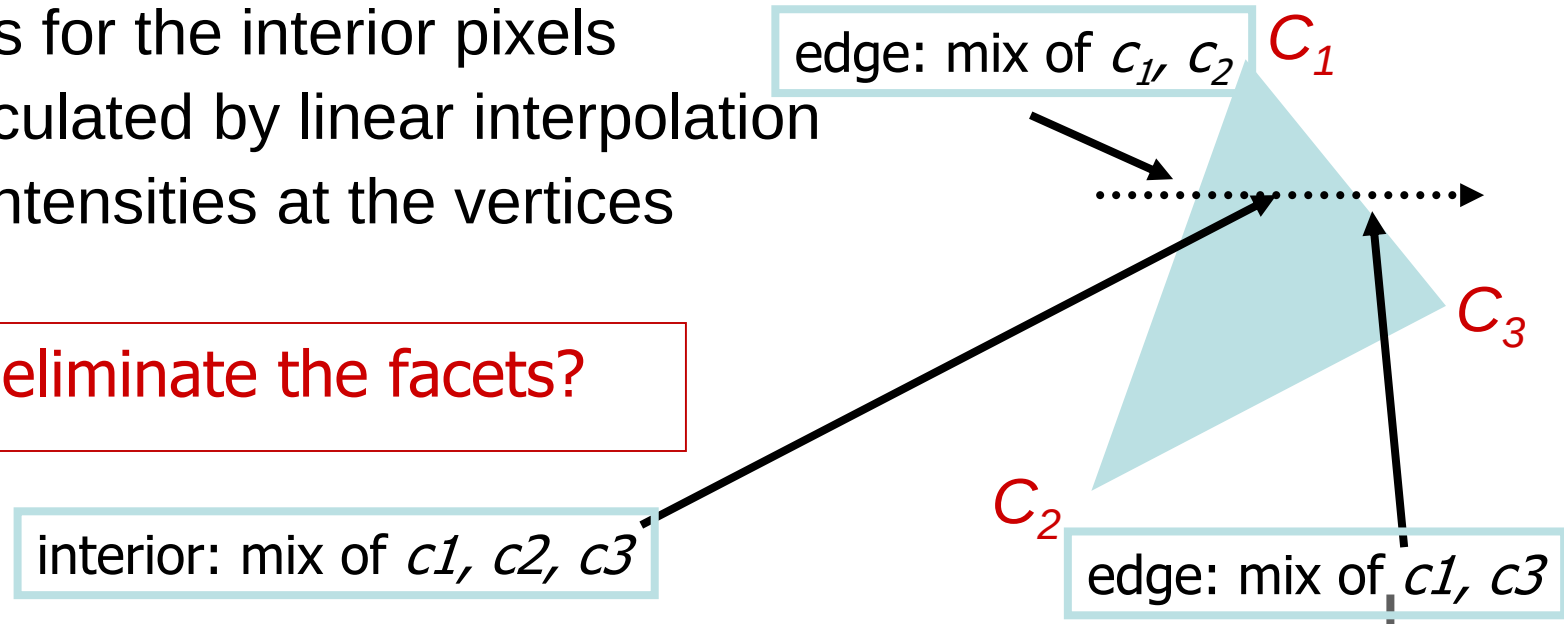
Gouraud Shading

- most common approach, and what OpenGL does
  - Normals are computed at the vertex (computed in VCS)
  - Intensity at each vertex is calculated using the normal and an illumination model (Phong lighting)
  - In rasterization stage: for each polygon the intensity values for the interior pixels

  are calculated by linear interpolation

  of the intensities at the vertices

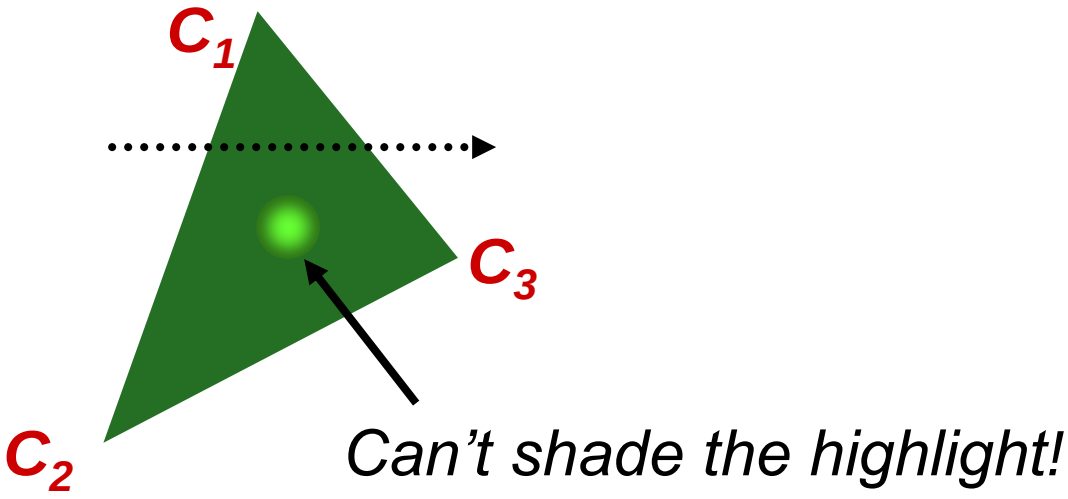
does this eliminate the facets?



Gouraud Shading **Artifacts**

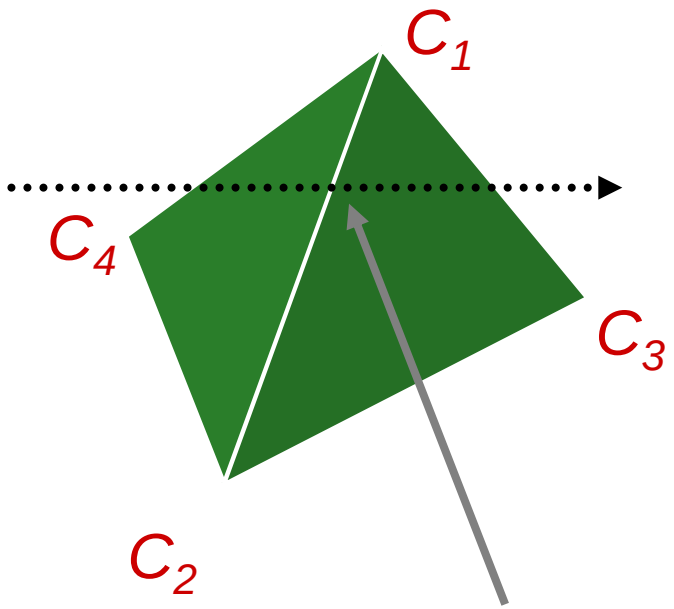
(I) MISSING HIGHLIGHT and SPOTLIGHT effects

- The surface is dark;
- Might miss specular highlights, if the highlight doesn't fall at the vertex. if the highlight is included, will be averaged over entire polygon

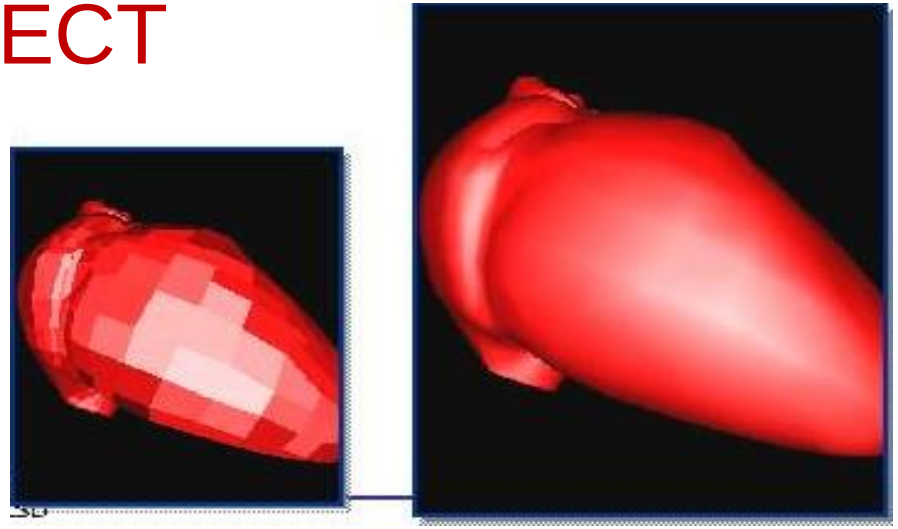


Gouraud Shading **Artifacts**

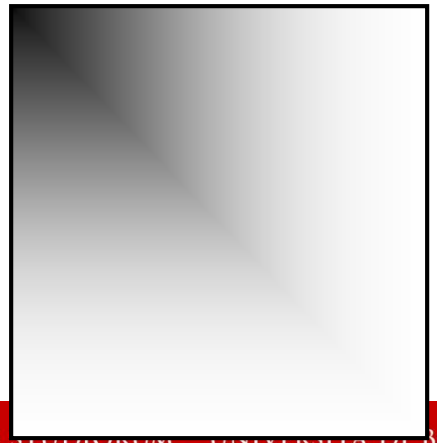
(II) Mach banding EFFECT



Discontinuity in rate of color change occurs here



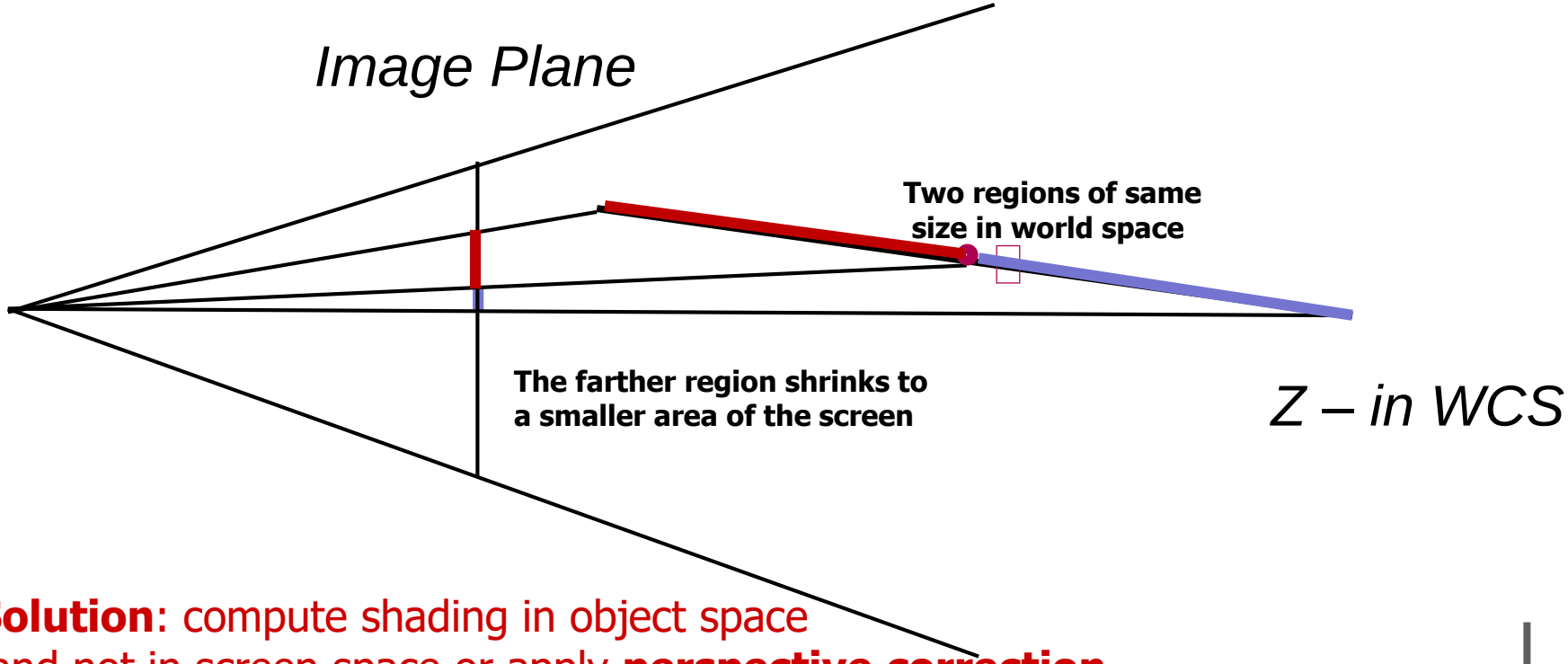
The normal vector is the same for the vertices C1 and C2 in both triangles



Gouraud Shading **Artifacts**

Interpolation of colors in Screen Space

- Linear interpolation is performed in screen space and it does not correspond to linear interpolation in WCS!
- Perspective projection alters the linear interpolation!

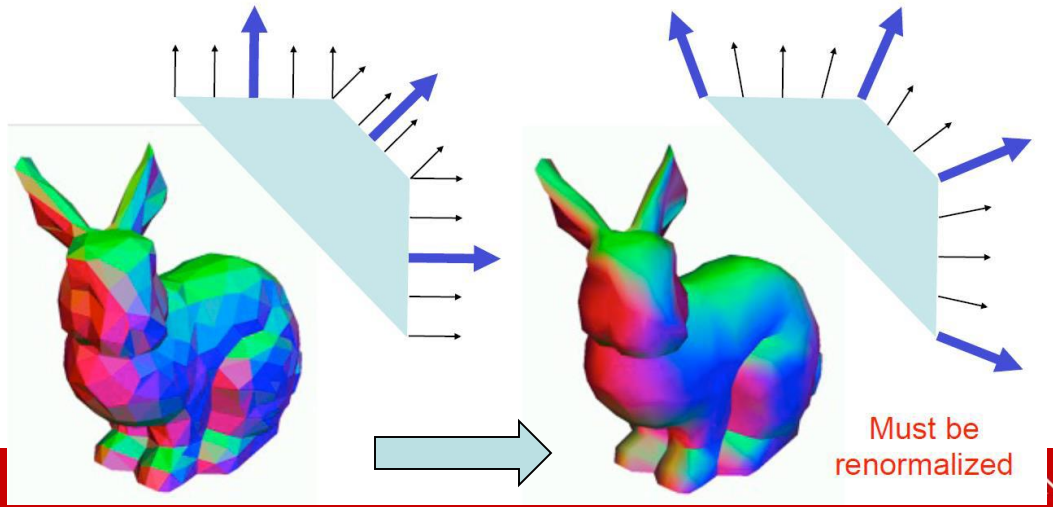


Solution: compute shading in object space and not in screen space or apply **perspective correction**.

Phong Shading

- Normals are calculated for each vertex.
- Vectors are then interpolated across the face.
- Apply Phong's light model at every pixel inside face using interpolated normal vector
  - Better handling of specularities
  - Slower than Gouraud shading
- Not built into OpenGL

Interpolated Normals

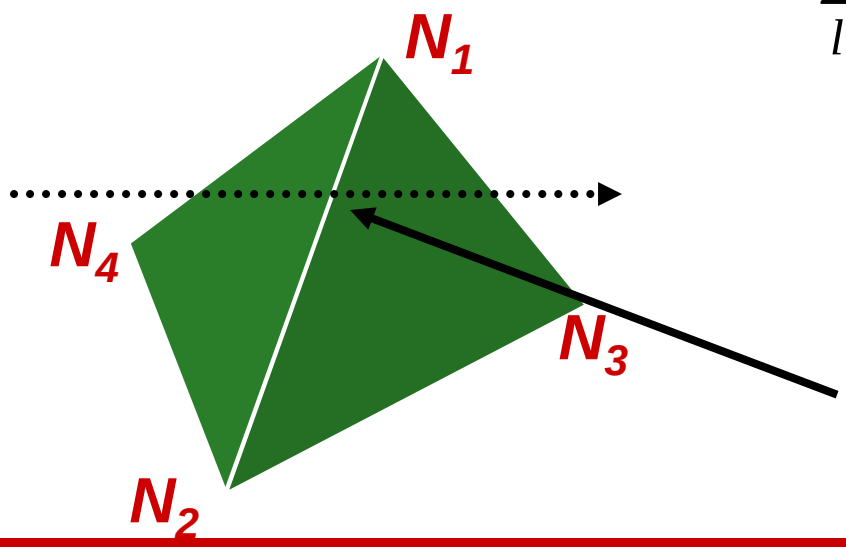


Phong Shading

- linearly interpolate the vertex normals
  - apply Phong's light model at every pixel

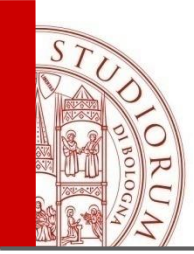
  (Specular reflections are also incorporated)

$$I_{total} = k_e + k_a I_{ambient} + \sum_{l=1}^{lights} I_l \left(k_d \left(n \cdot l \right) + k_s \left(v \cdot r \right)^{n_s} \right)$$



remember: normals used in diffuse and specular terms

discontinuity in normal's rate of change harder to detect…



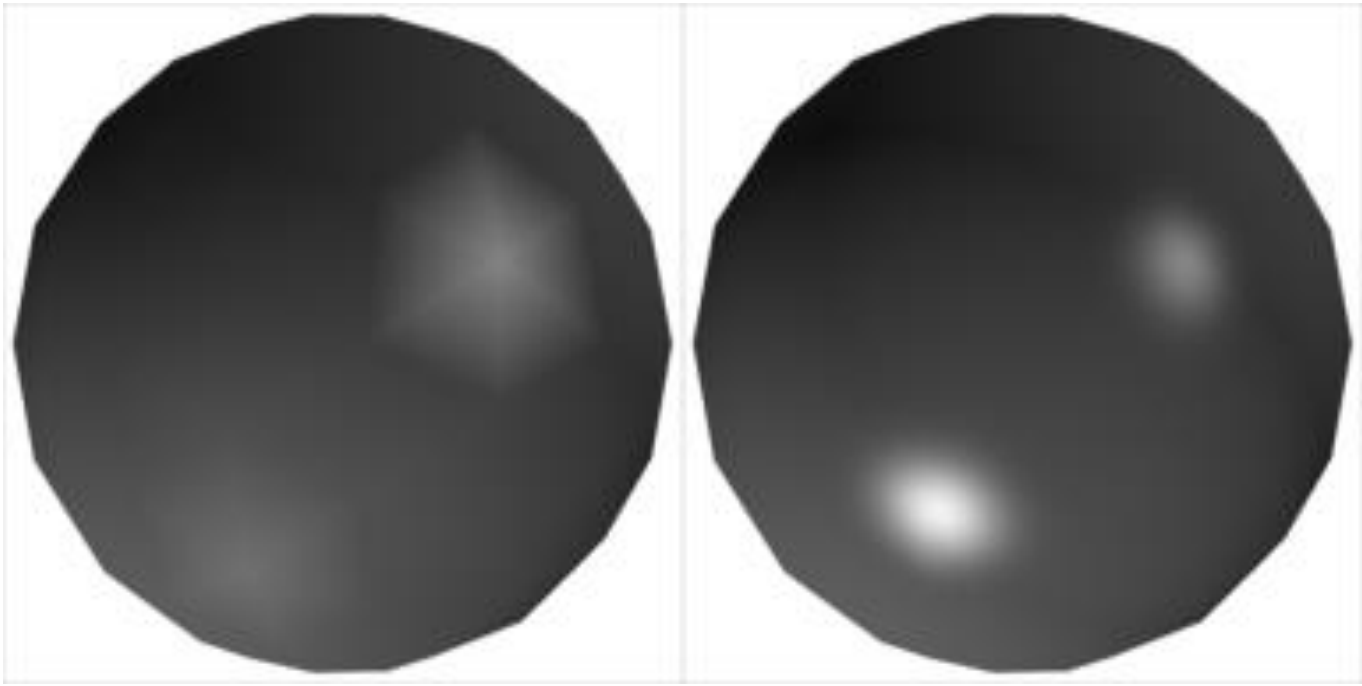
Phong Shading Difficulties

- computationally expensive
  - per-pixel vector normalization and lighting computation!
  - floating point operations required
- lighting after perspective projection
- no **direct** support in hardware
  - Apply using shader

PHONG Shading **ARTIFACTS**

Silhouettes

- polygonal silhouettes remain



Gouraud *Phong*

PHONG Per-fragment shading

- Vertex shader

  -compute vectors: **n,l,v** in VCS for vertices
- Rasterizer (generates fragment):

  interpolate values at vertices to produce values per pixel
- Fragment shader (lighting computation)

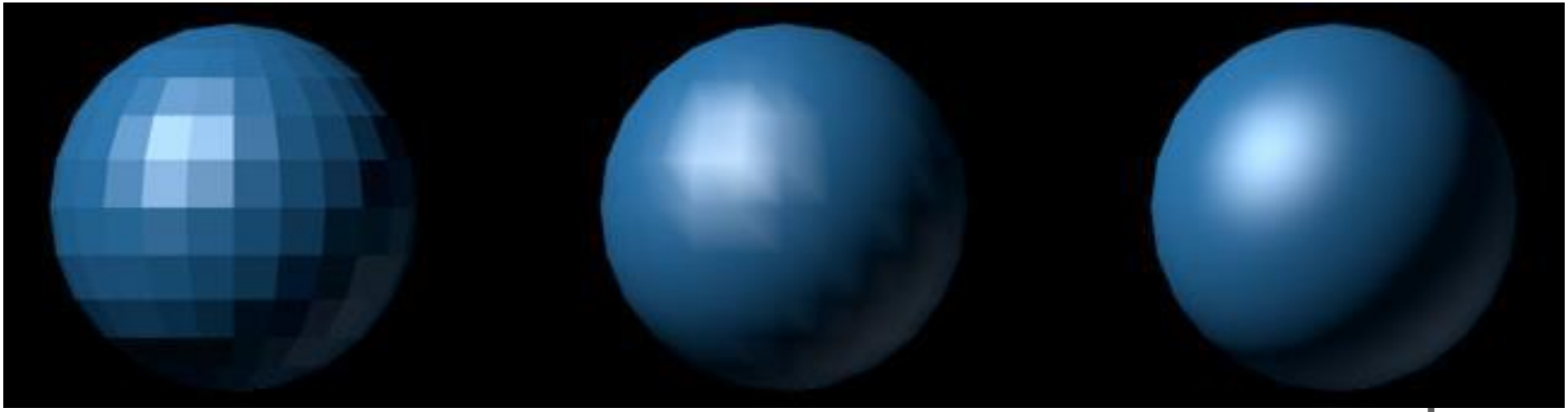
  -apply Phong's light model to pixels

- Flat **(uses actual triangle normals)**
- Gouraud

(uses vertex normals, one lighting evaluation per vertex)

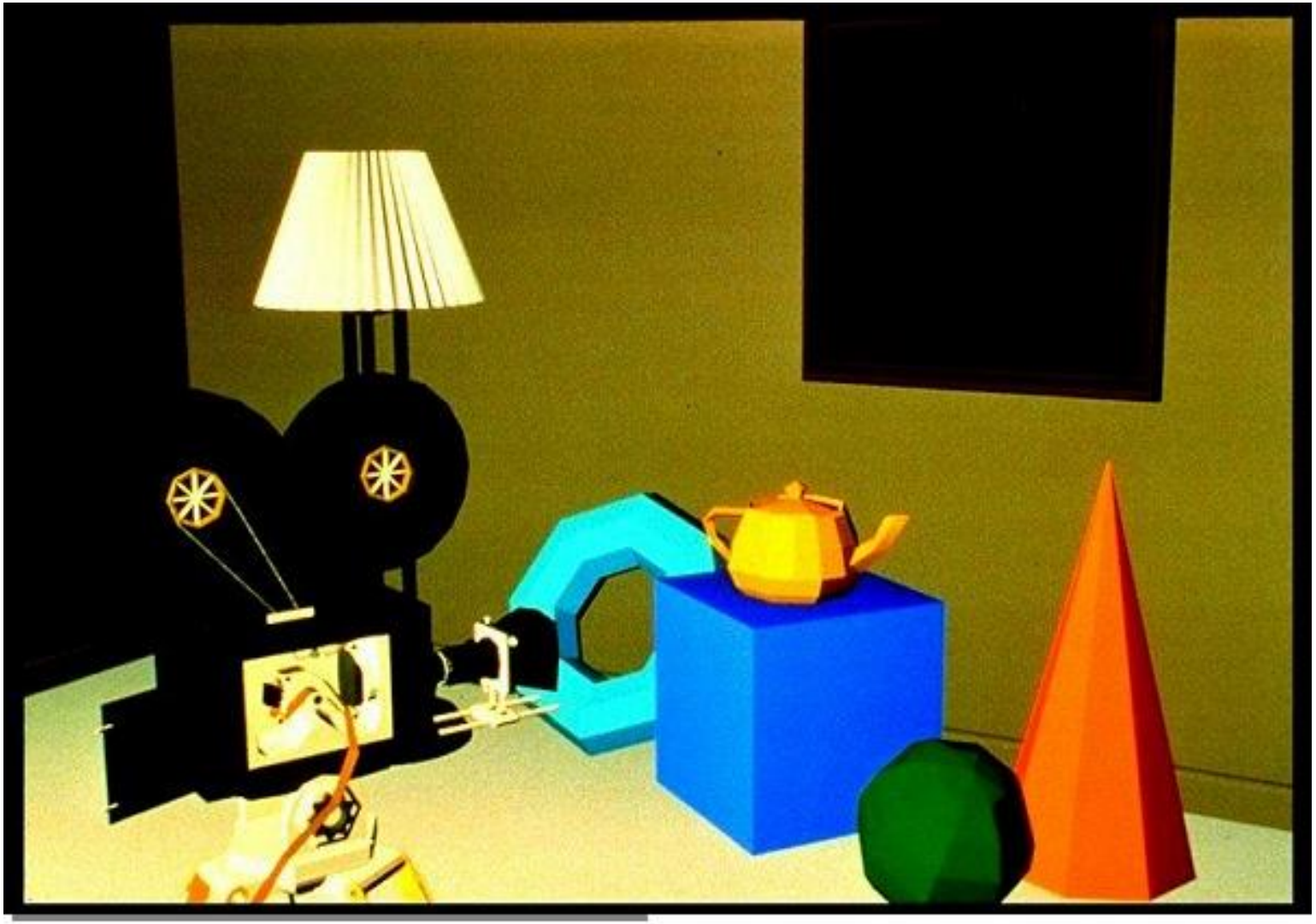
- Phong

(uses vertex normals, one lighting evaluation per pixel)



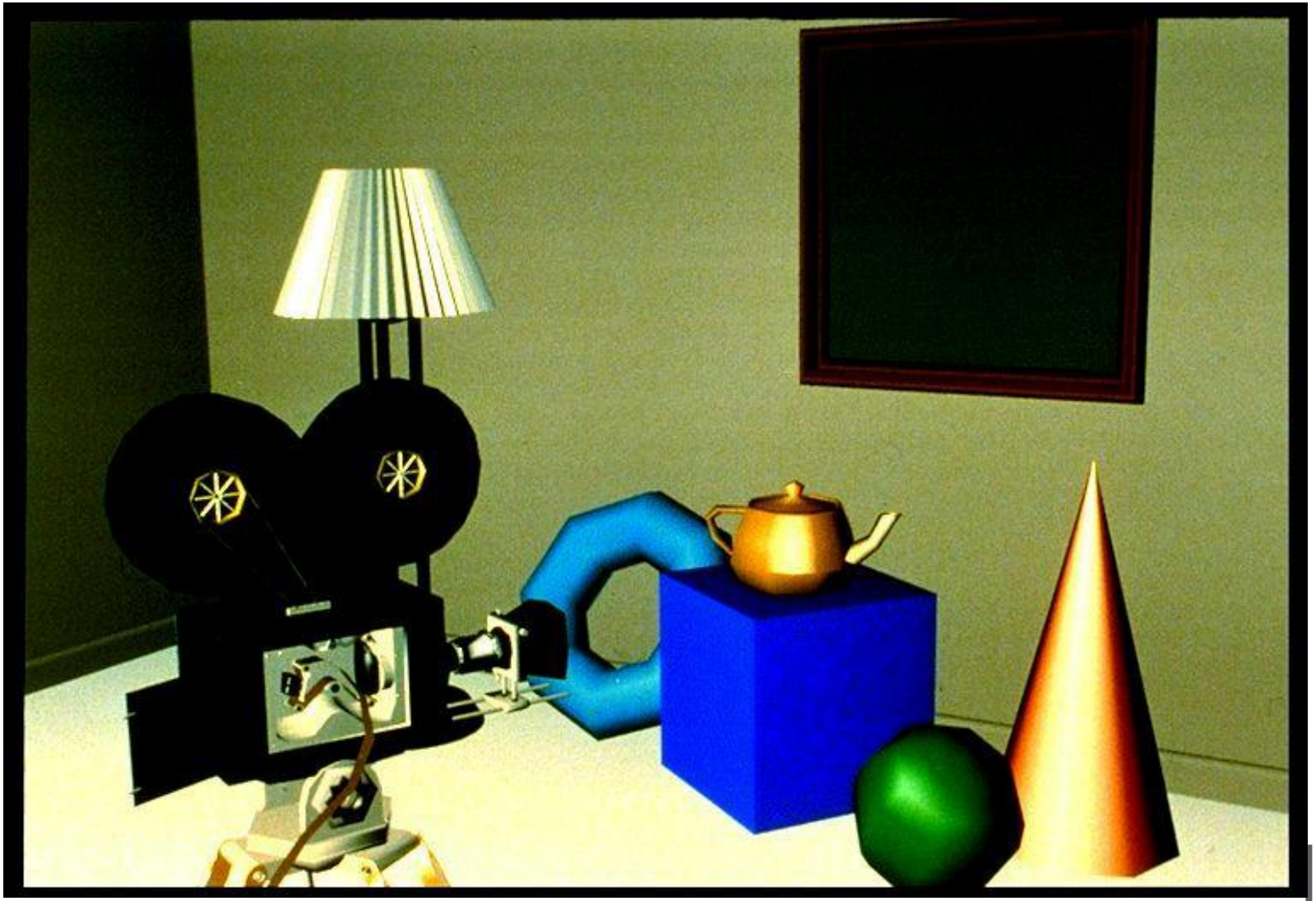
Increase the number of faces

Flat shading

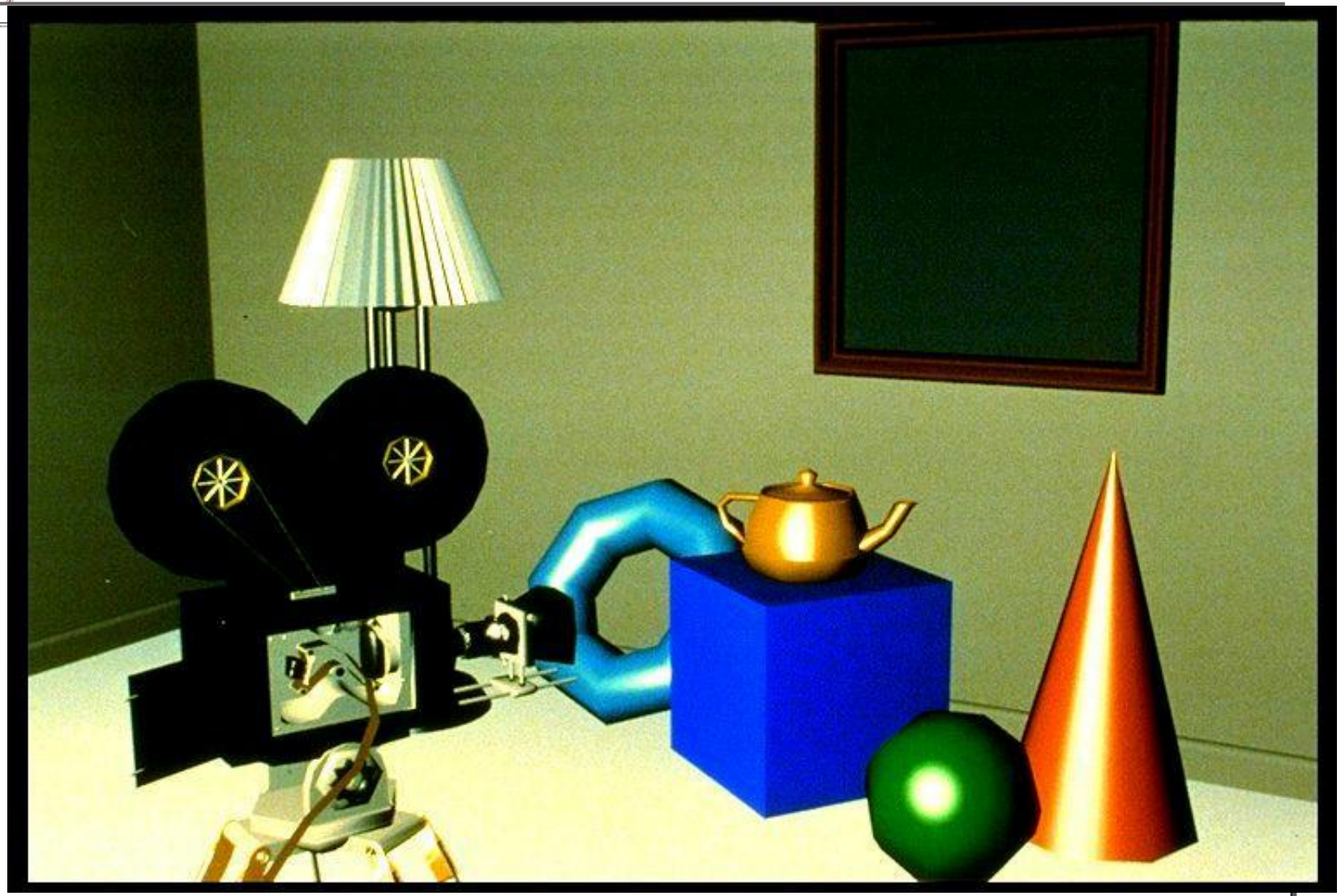


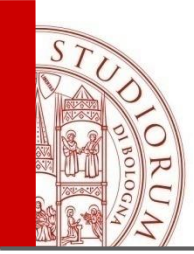
Foley Van Dam

Gouraud shading



Phong shading





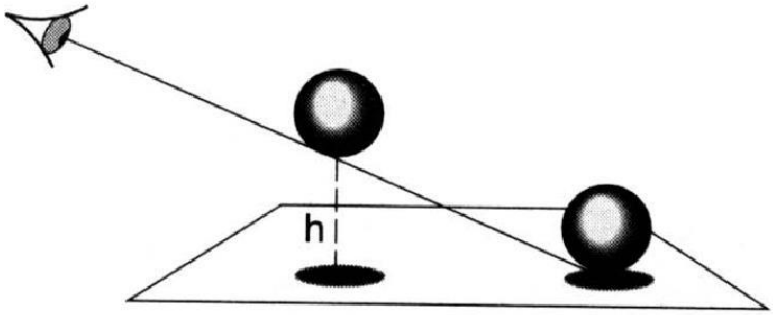
Shadow...ombre!

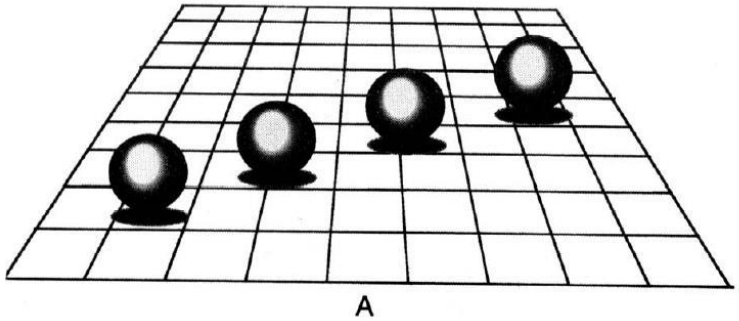
- Lighting and shading are computed using direct light
- That is we ignore effects due to:
  - Shadows for occlusions
  - Intensity of the shadows (points in shadows that receive light from other objects)
  - Light reflected by other objects

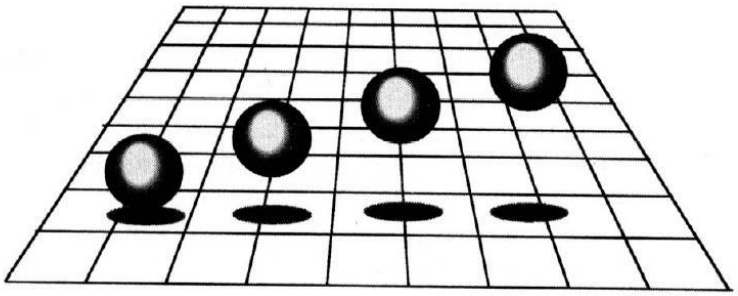
- Recursive reflection, Refract (Translucent objects) and shadows are only obtained by global illumination models

The shadow for an object depends on the object shape and on the position of the light source.

- Depth cue
- Scene Lighting
- Realism
- Contact points





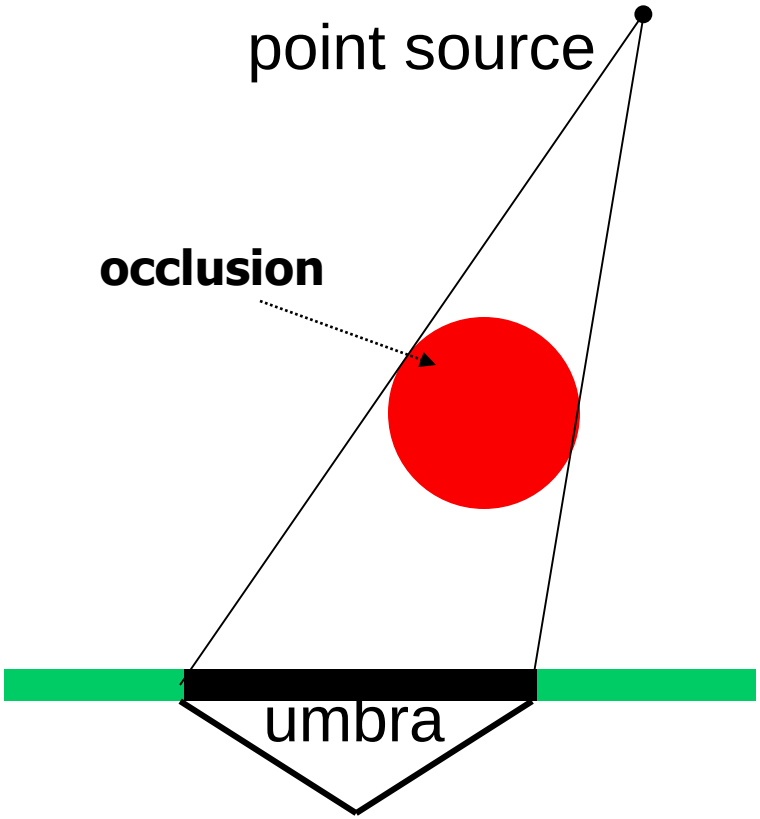


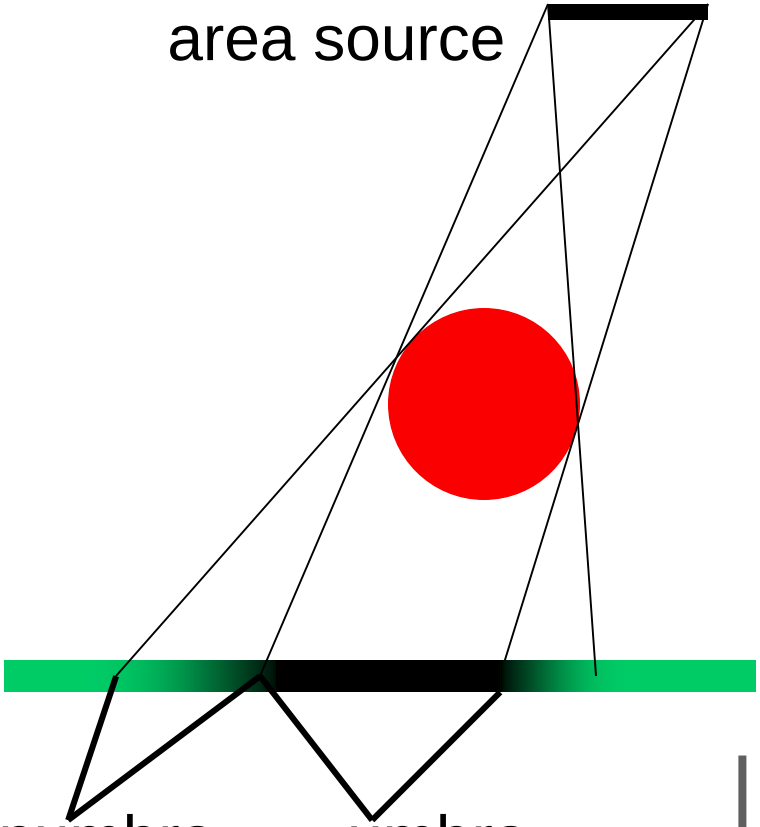
Soft and hard shadow

SOFT Shadows: caused by extended light sources

Umbra source completely occluded

Penumbra source partially occluded



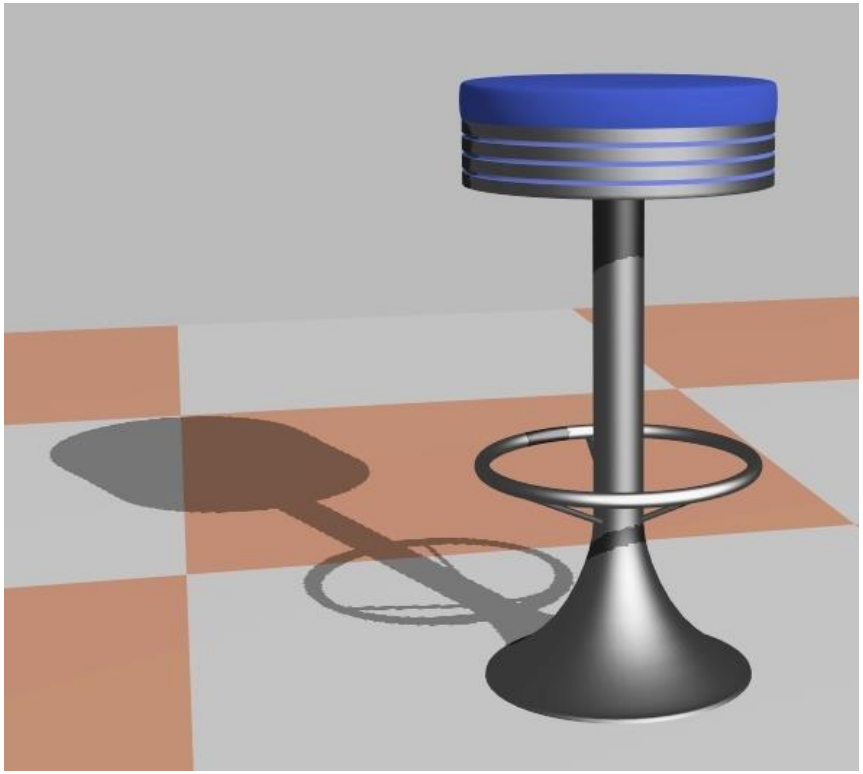


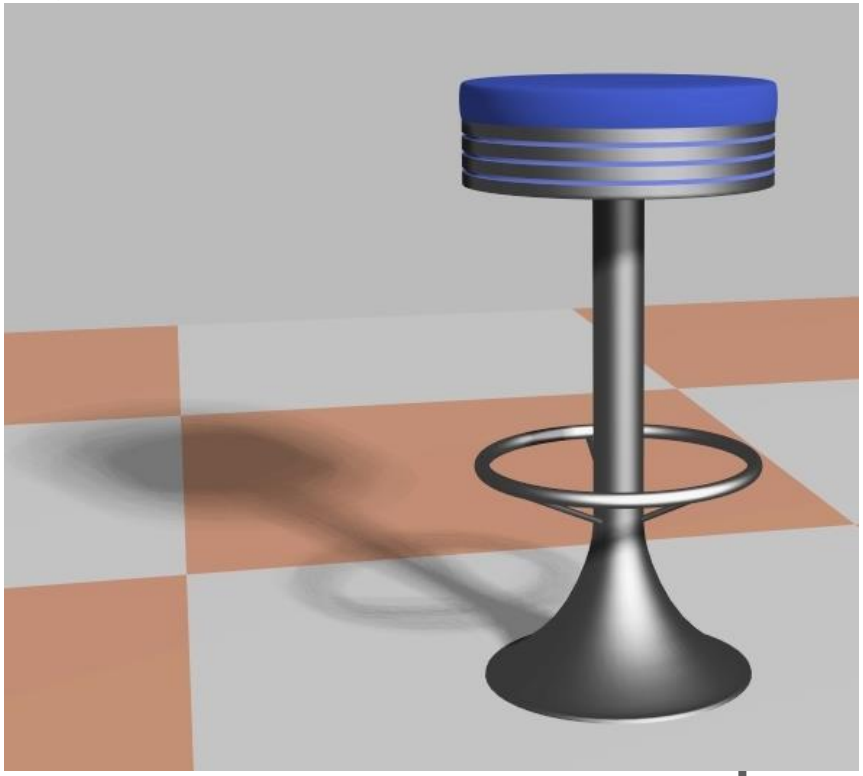
Soft and hard shadow

- Soft Shadow contains both **umbra** than **penumbra**
- Point lights produce only HARD Shadows

SOLUTION 1 Simulate area lights with lots of point lights Expensive

SOLUTION 2 Blur shadows in image space Cheap, inaccurate





Fake/Approximated Shadows

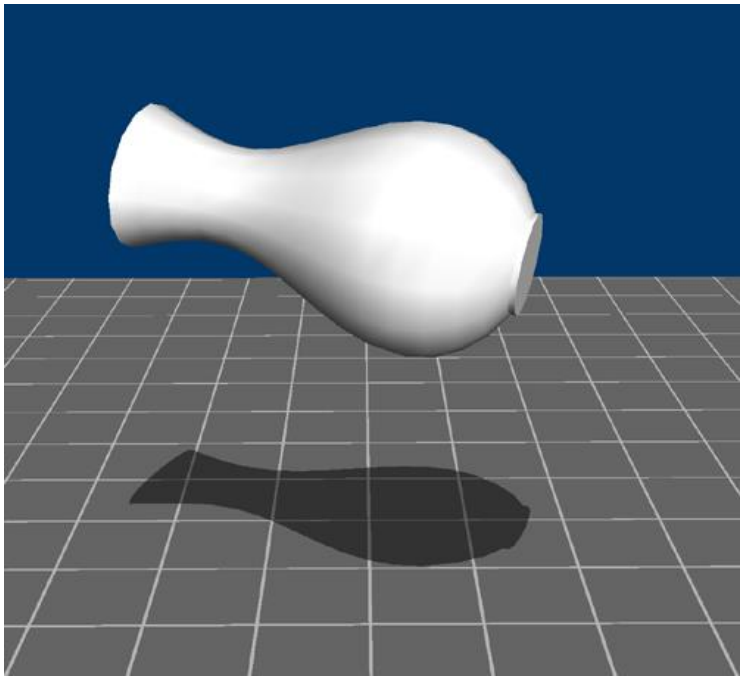
1) Planar Shadows

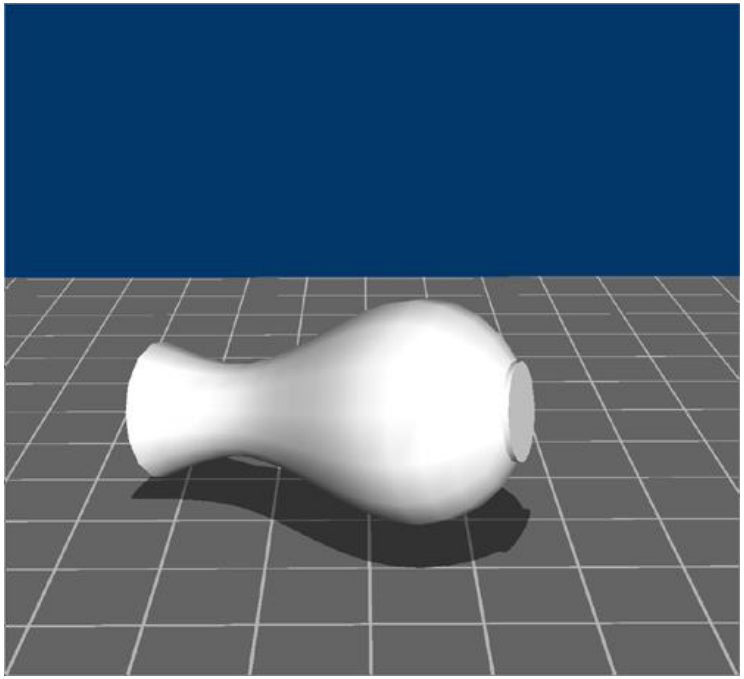
2) Shadow Maps

3) Shadow Volumes
   - Stencil Buffer

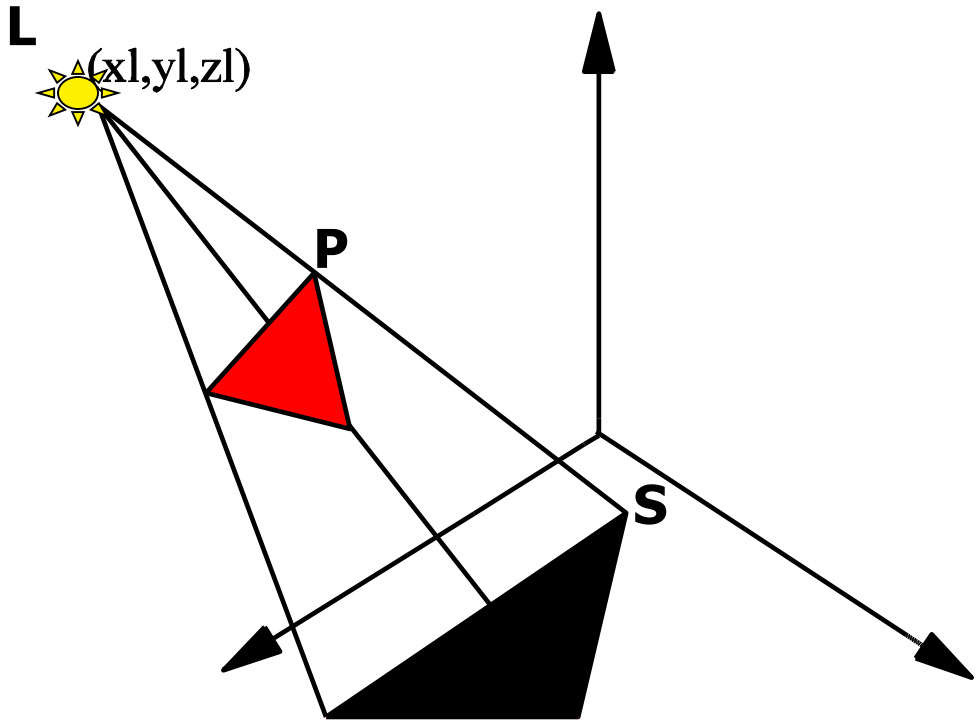
1) Shadows on Planar Surfaces

- Draw the object primitives a second time, projected to the ground plane
- The point light source is the Center of Projection (COP)





1) Shadows on Planar Surfaces (Blinn 1988)



Assume a single object on a plane

Draw the projection of the object onto the plane;

shadow polygon: it is the projection of the polygon on the surface (es. z=0) with COP in the light point L.

1) Shadows on Planar Surfaces (Blinn 1988)

For each vertex of the object $P(x_p,y_p,z_p)$, a shadow ray passes from the point light $L(x_l,y_l,z_l)$ to the plane intersecting the shadow polygon in $S(x_s,y_s,0)$

Compute the vertices of the *shadow polygon* and rending it as an added object with dark color.

$$S = P - \alpha L$$

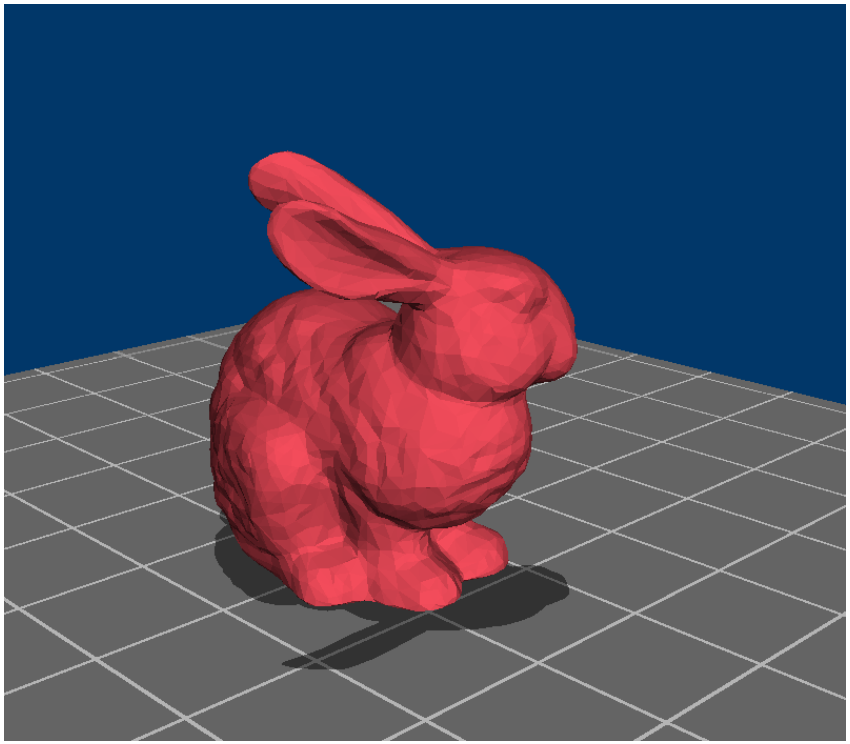
$$0 = z_p - \alpha z_l \quad \alpha = z_p / z_l$$

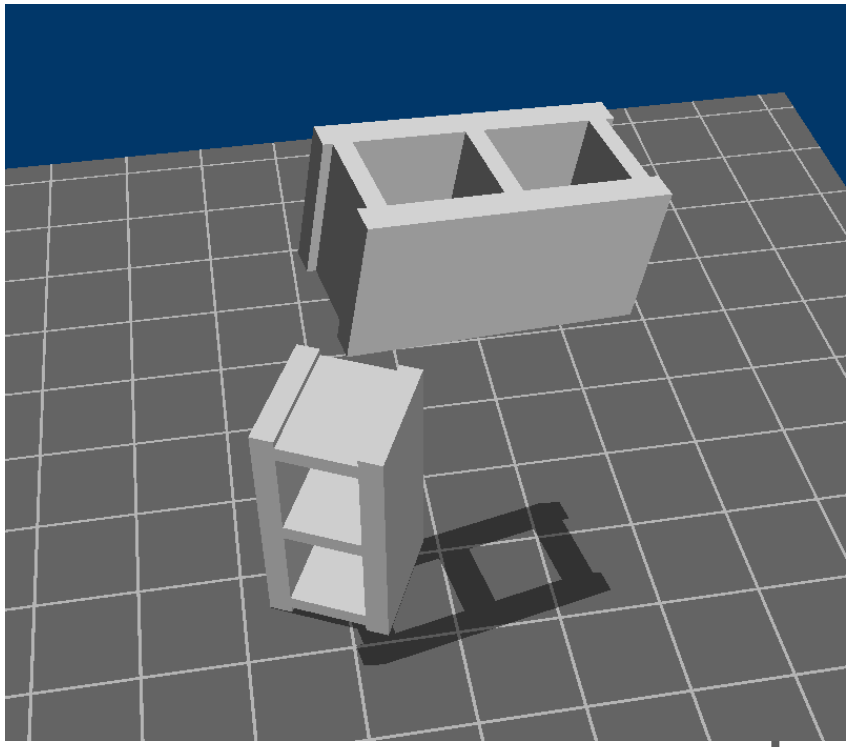
$$x_s = x_p - \frac{z_p}{z_l} x_l \quad y_s = y_p - \frac{z_p}{z_l} y_l$$

$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\frac{x_l}{z_l} & 0 \\ 0 & 1 & -\frac{y_l}{z_l} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

Limitations of Planar Shadows

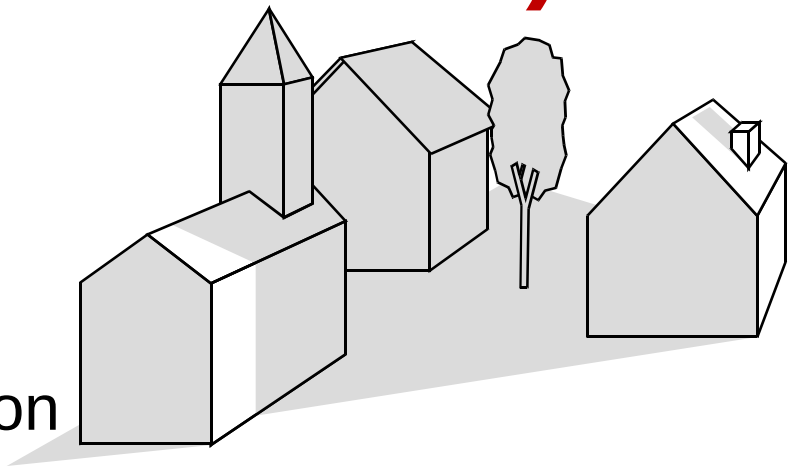
- Does not produce self-shadows, shadows cast on other objects, shadows on curved surfaces, etc.



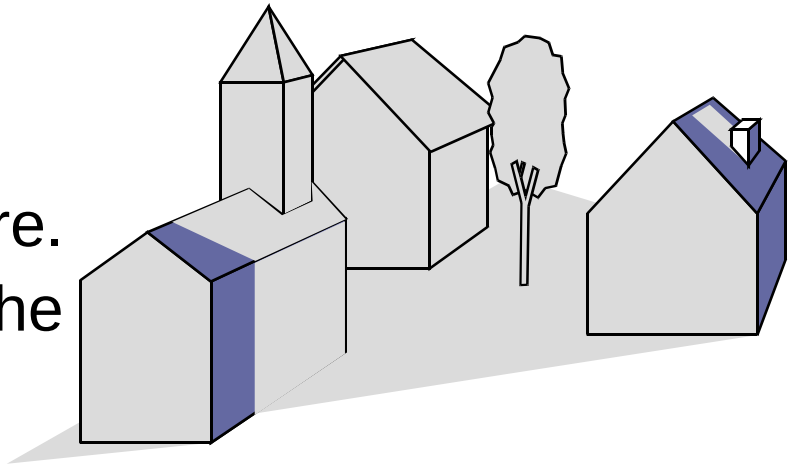


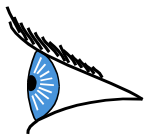
2) Shadow Map Algorithm (Williams 1978)

- A point is lit if it is visible from the light source



- Shadow computation similar to view computation (hidden surface removal)





- There are not shadows if the viewer coincides with the unique light point there.

  (shadows are hidden to the light source)

2) Shadow Map Algorithm

It needs a buffer (**shadow z-buffer**) for each light source.

Procedure uses 2 passes through the pipeline:

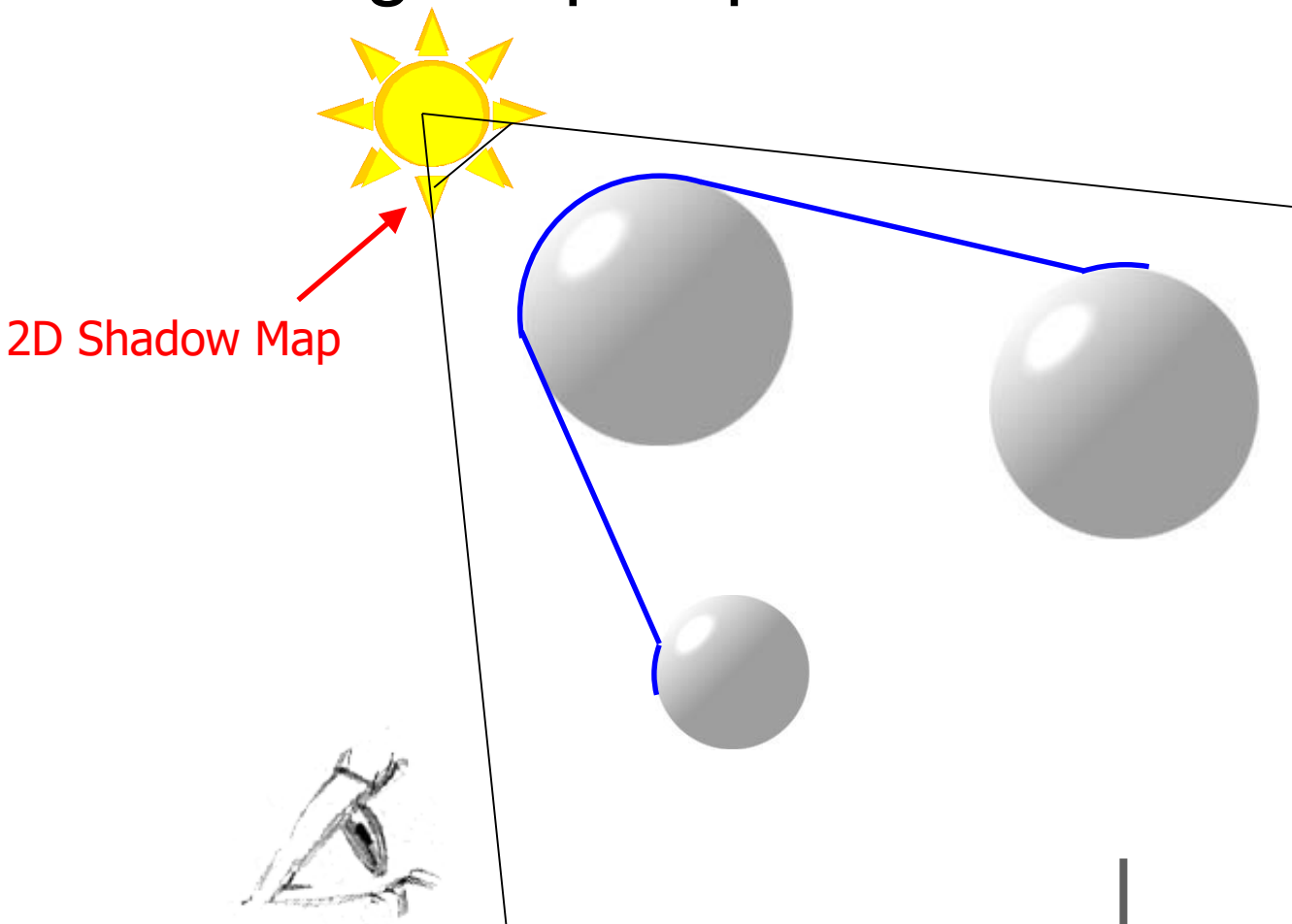
STEP 1: compute shadow map (depth from light source) first rendering of the scene using the light source as view reference point; store the result in the shadow Z-buffer

(Depth image of visible polygons from light source)

STEP 2: Render final image second rendering of the scene using Z-buffer algorithm and checking shadow map to see if points are in shadow

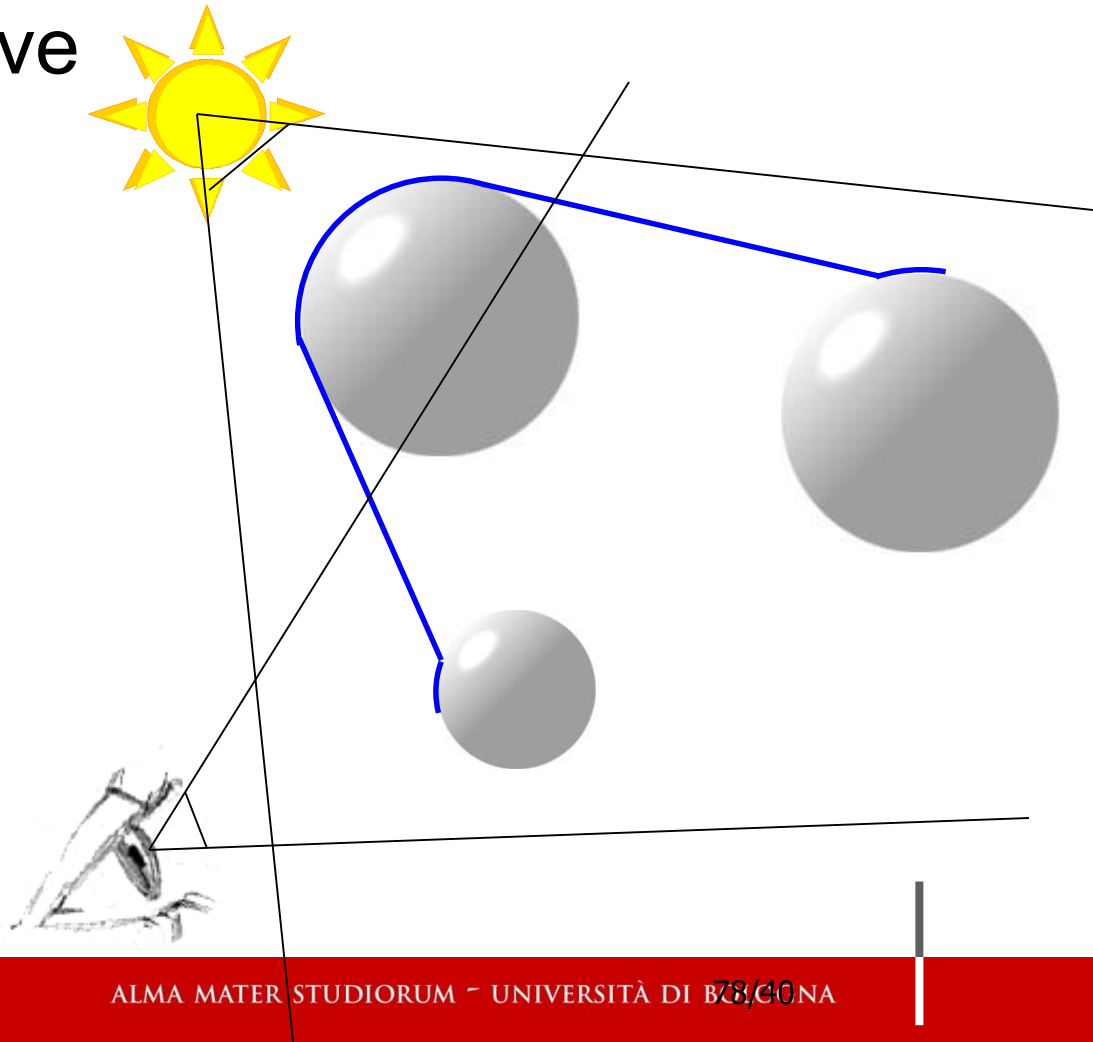
Shadow Maps: step 1

- Render scene from light's perspective



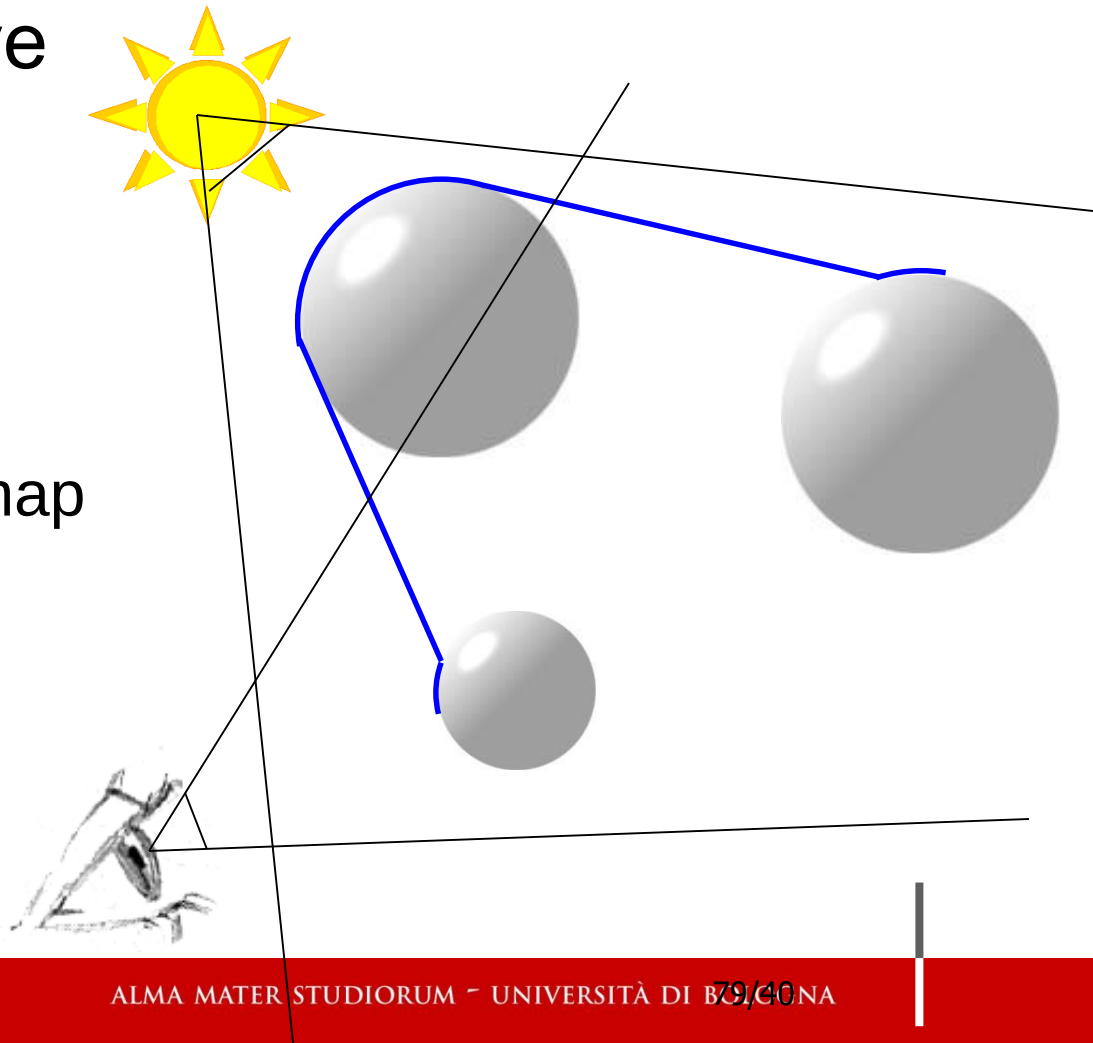
Shadow Maps: step 2

- Render scene from viewer's perspective



Shadow Maps: step 2

- Render scene from viewer's perspective
- For every pixel
  - Transform to light source space
  - Compare distance to value in shadow map

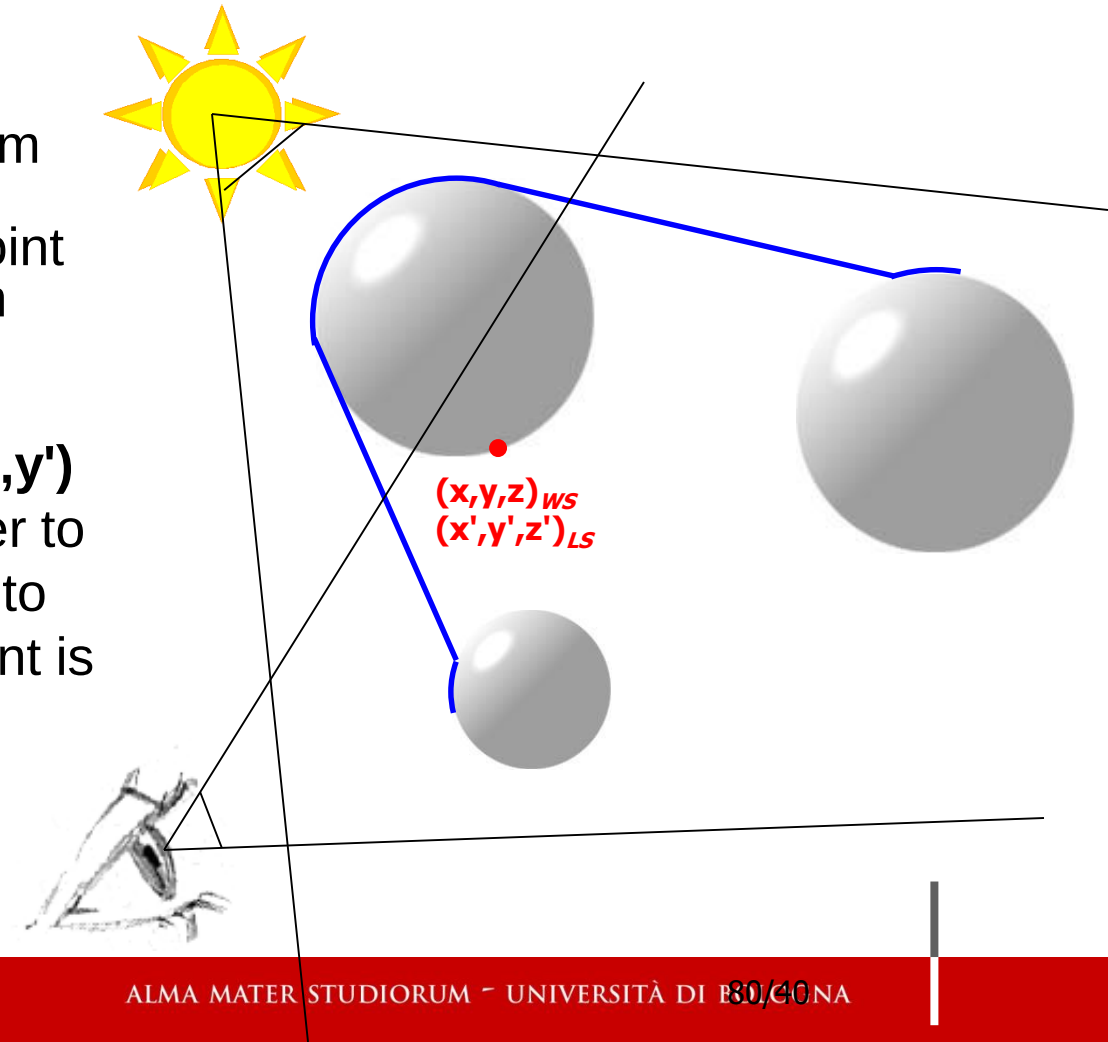


Shadow Maps : step 2

- Render scene from viewer's perspective
- For every pixel

If the point $(x,y,z)_{ws}$ is visible from the camera,
convert $(x,y,z)_{ws}$ in $(x',y',z')_{LS}$ point coord. in light source frame with origin at the source light;

- If **z' > shadow_z-buffer(x',y')** then another surface is closer to the light source with respect to the considered point, the point is shadow;
- Otherwise render the point.

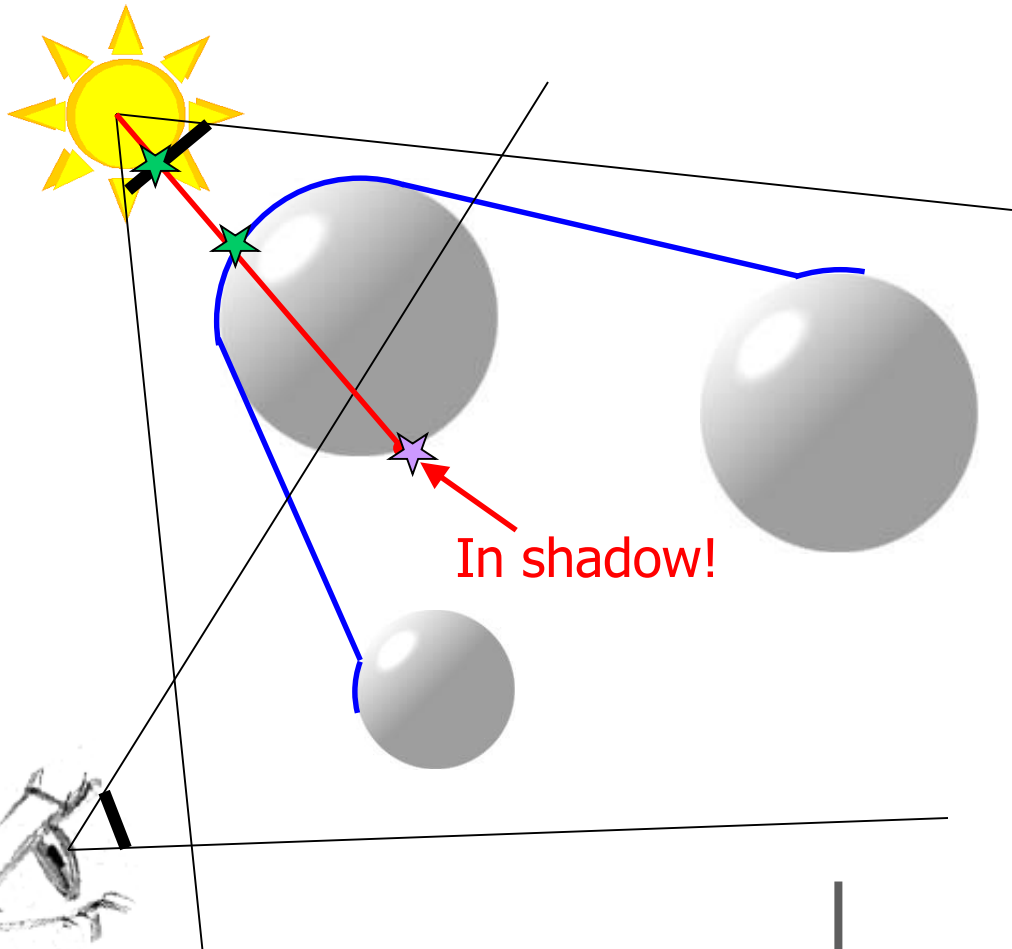


Shadow Maps : step 2

- Render scene from viewer's perspective
- For every pixel

If the point (x,y,z) ws is visible from the camera,
convert (x,y,z)ws in (x',y',z')ls point coord. in light source frame with origin at the source light;

- If **z' > shadow_z-buffer(x',y')** then another surface is closer to the light source with respect to the considered point, the point is shadow;
- Otherwise render the point.

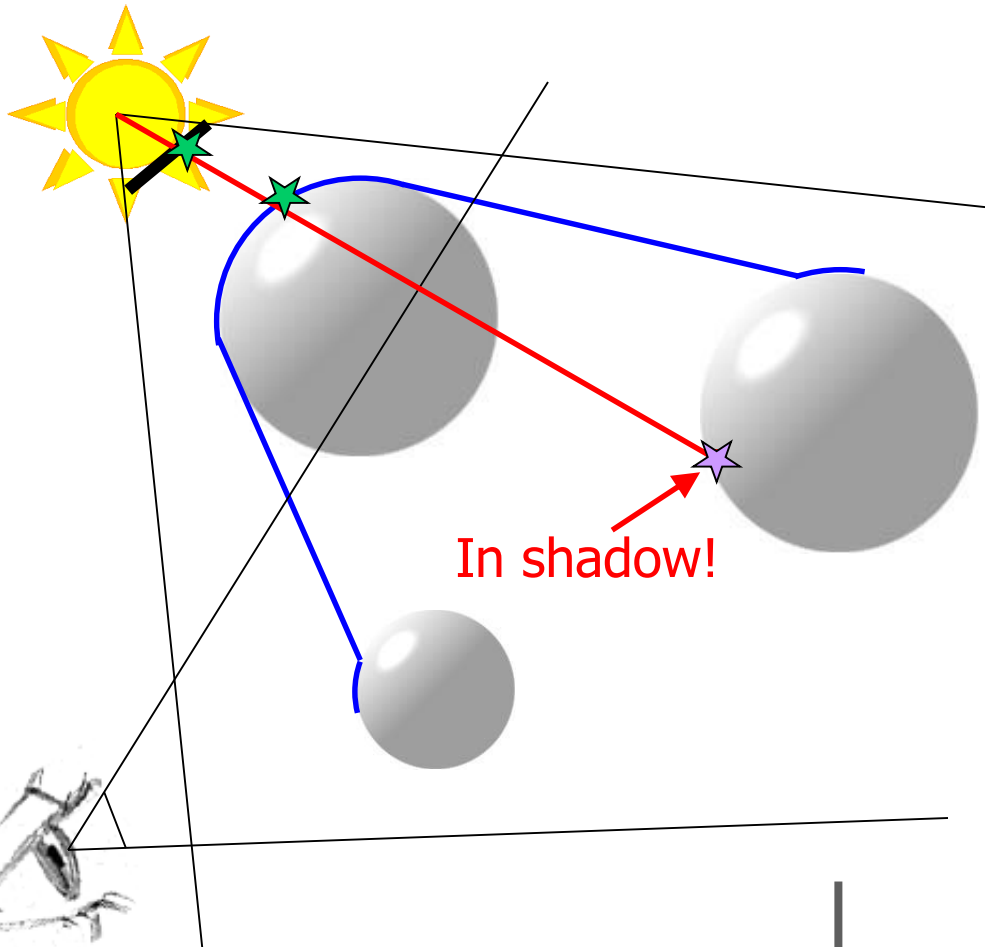


Shadow Maps : step 2

- Render scene from viewer's perspective
- For every pixel

If the point (x,y,z) ws is visible from the camera,
convert (x,y,z)ws in (x',y',z')ls point coord. in light source frame with origin at the source light;

- If **z' > shadow_z-buffer(x',y')** then another surface is closer to the light source with respect to the considered point, the point is shadow;
- Otherwise render the point.

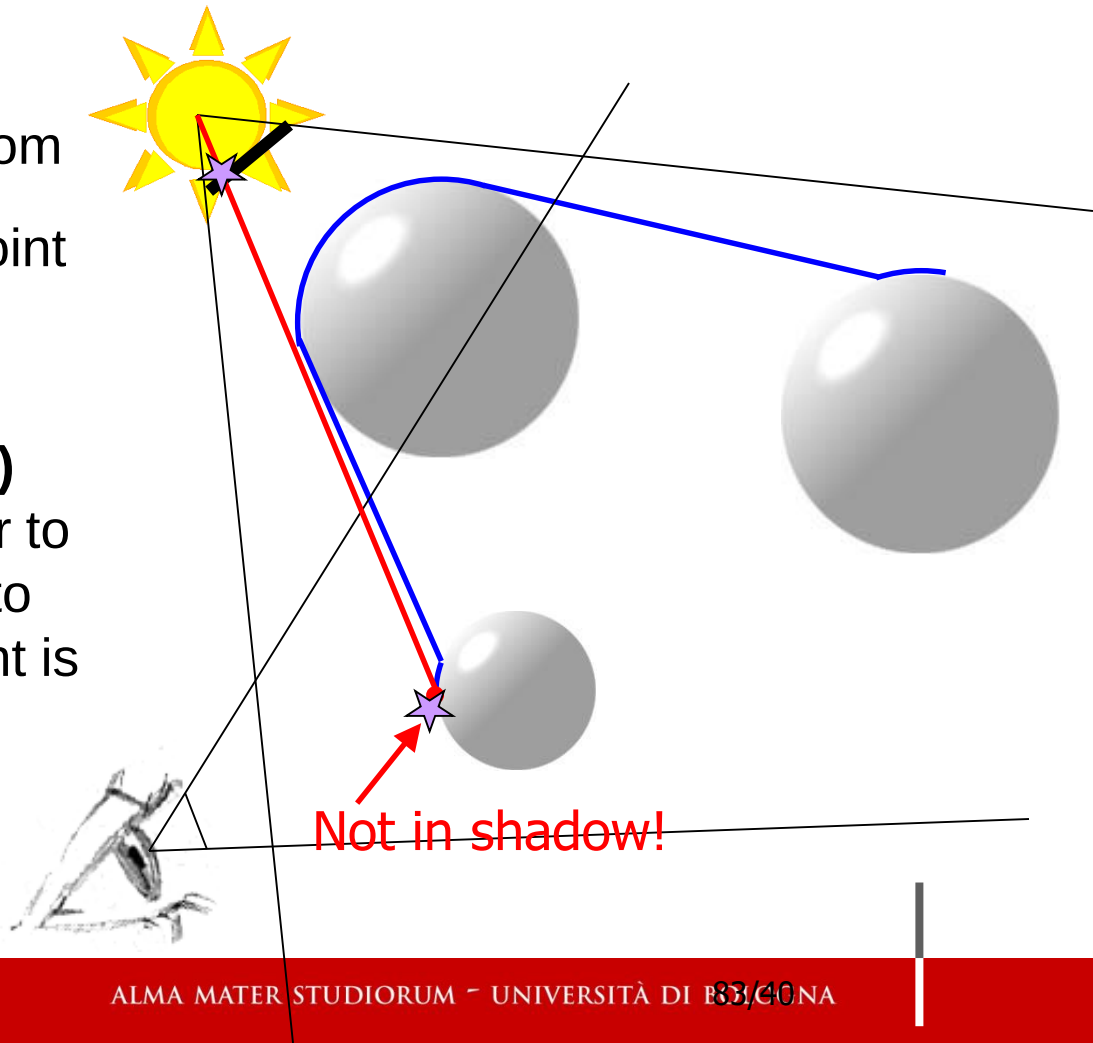


Shadow Maps : step 2

- Render scene from viewer's perspective
- For every pixel

If the point (x,y,z) ws is visible from the camera,
convert (x,y,z)ws in (x',y',z')ls point coord. in light source frame with origin at the source light;

- If **z' > shadow_z-buffer(x',y')** then another surface is closer to the light source with respect to the considered point, the point is shadow;
- Otherwise render the point.



Shadow Maps

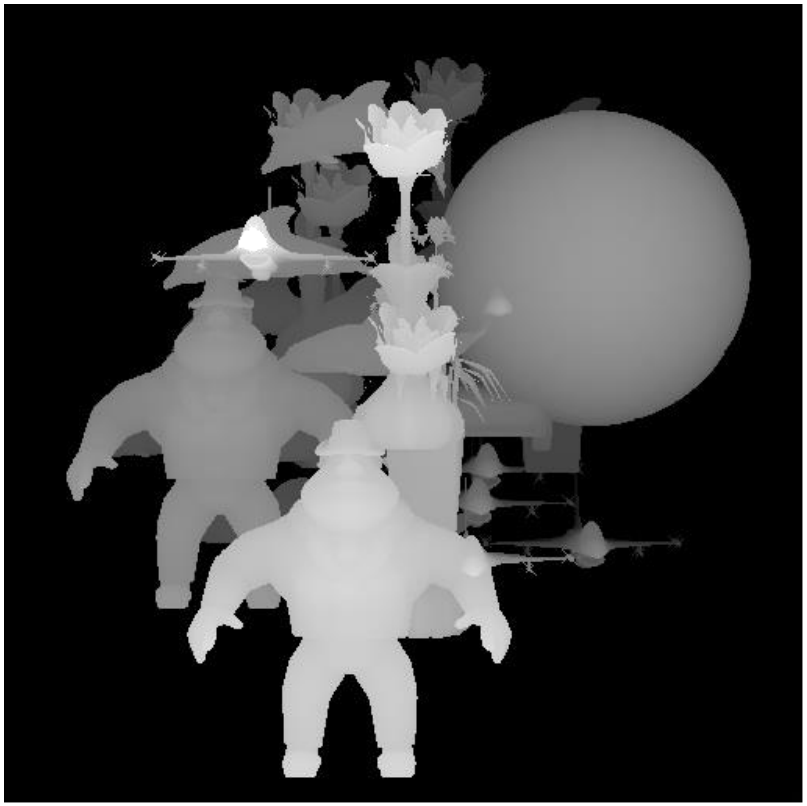
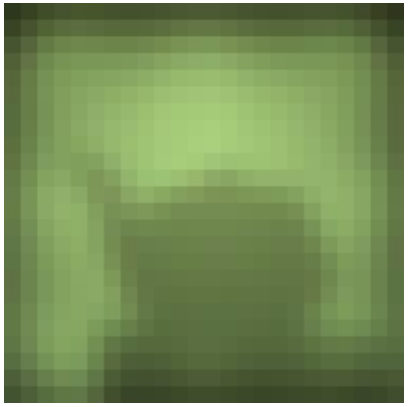


Image taken from http://www.cse.ohio-state.edu/~haleyb/Hardware/ggDepthBuffer.jpg

Shadow Map

Pre-computation of shadows as texture



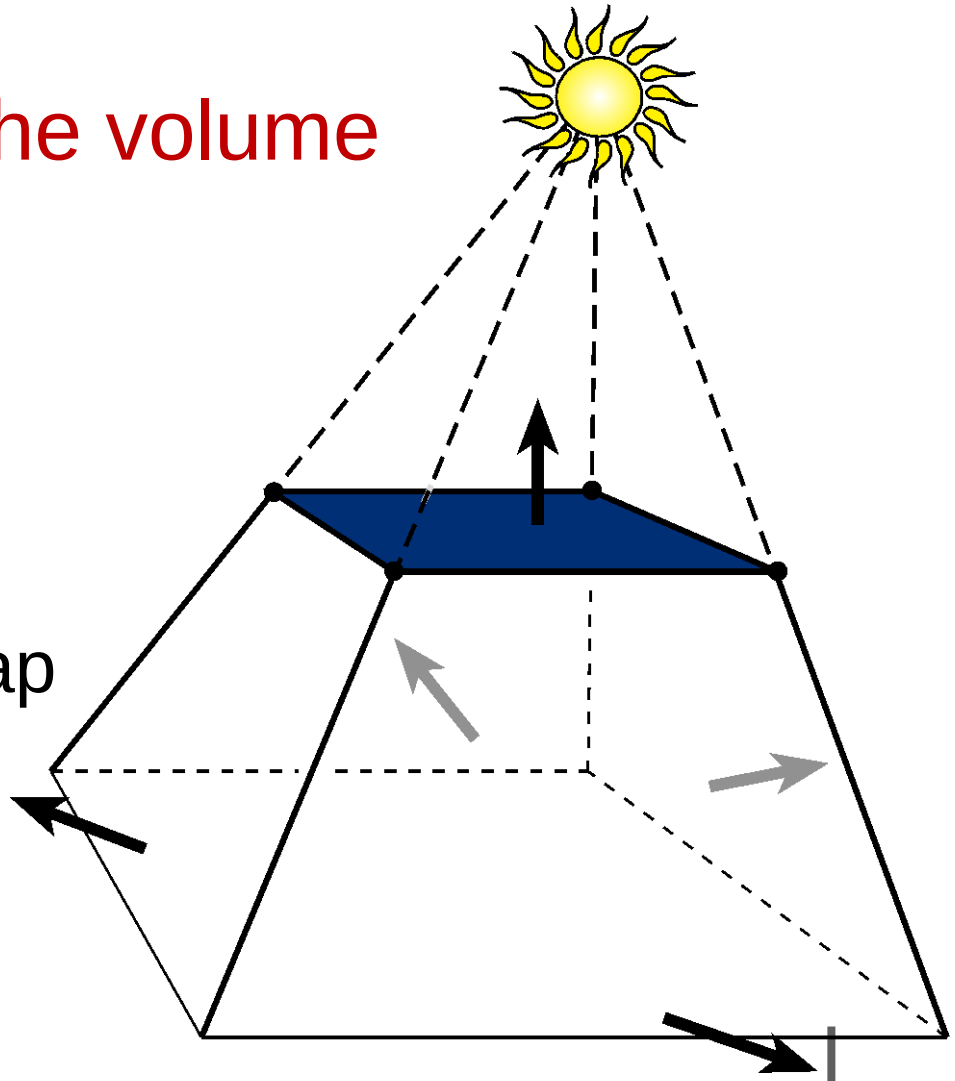
Shadows are independent on the view reference point.

Shadow-Maps: generate shadow texture by capturing silhouettes of objects as seen from the light source. Project texture onto scene.
Note: must recalculate for moving lights

Images courtesy of Kasper Høy Nielsen.

3) Shadow Volumes

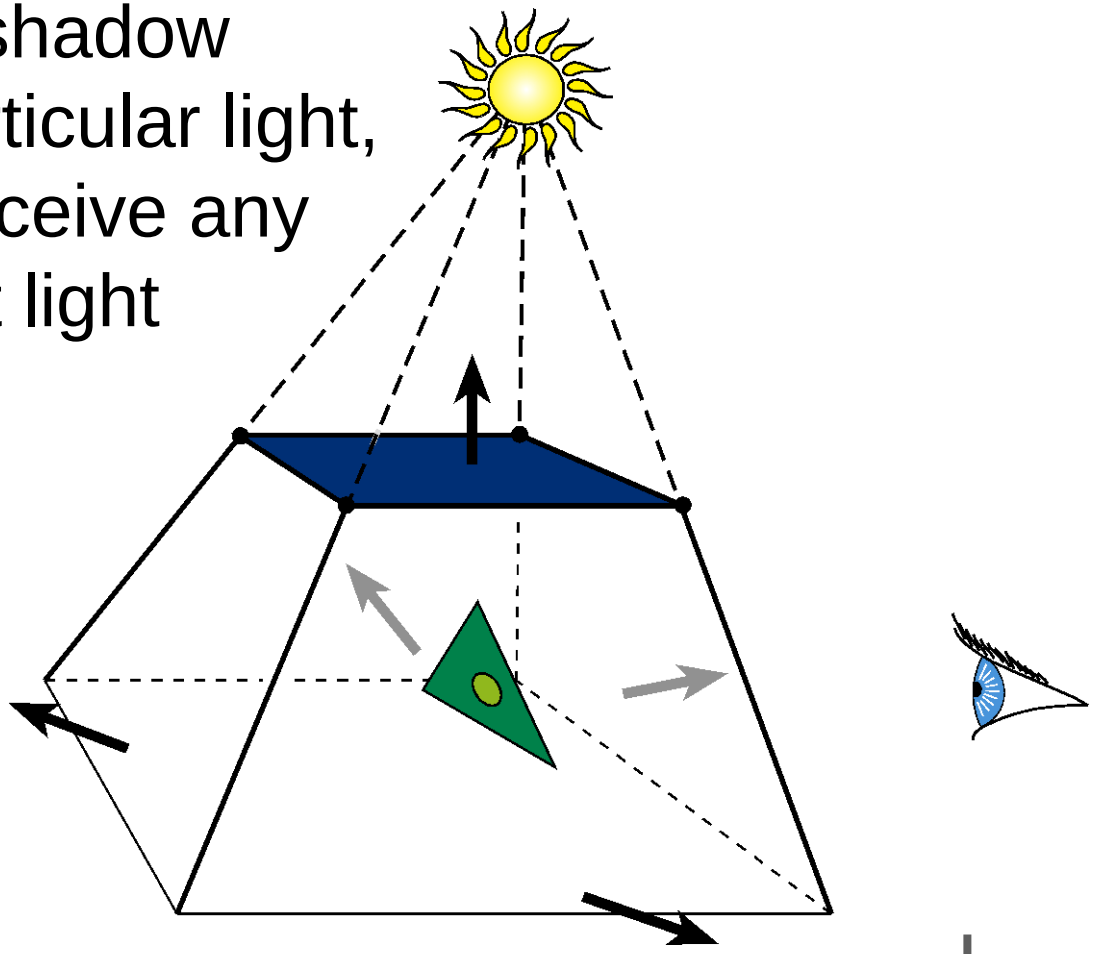
- Explicitly represent the volume of space in shadow
- For each polygon
  - Pyramid with point light as apex
  - Include polygon to cap
- Shadow test similar to clipping



Shadow Volumes

- If a point is inside a shadow volume cast by a particular light, the point does not receive any illumination from that light

- Cost of Naive implementation:
  #polygons * #lights



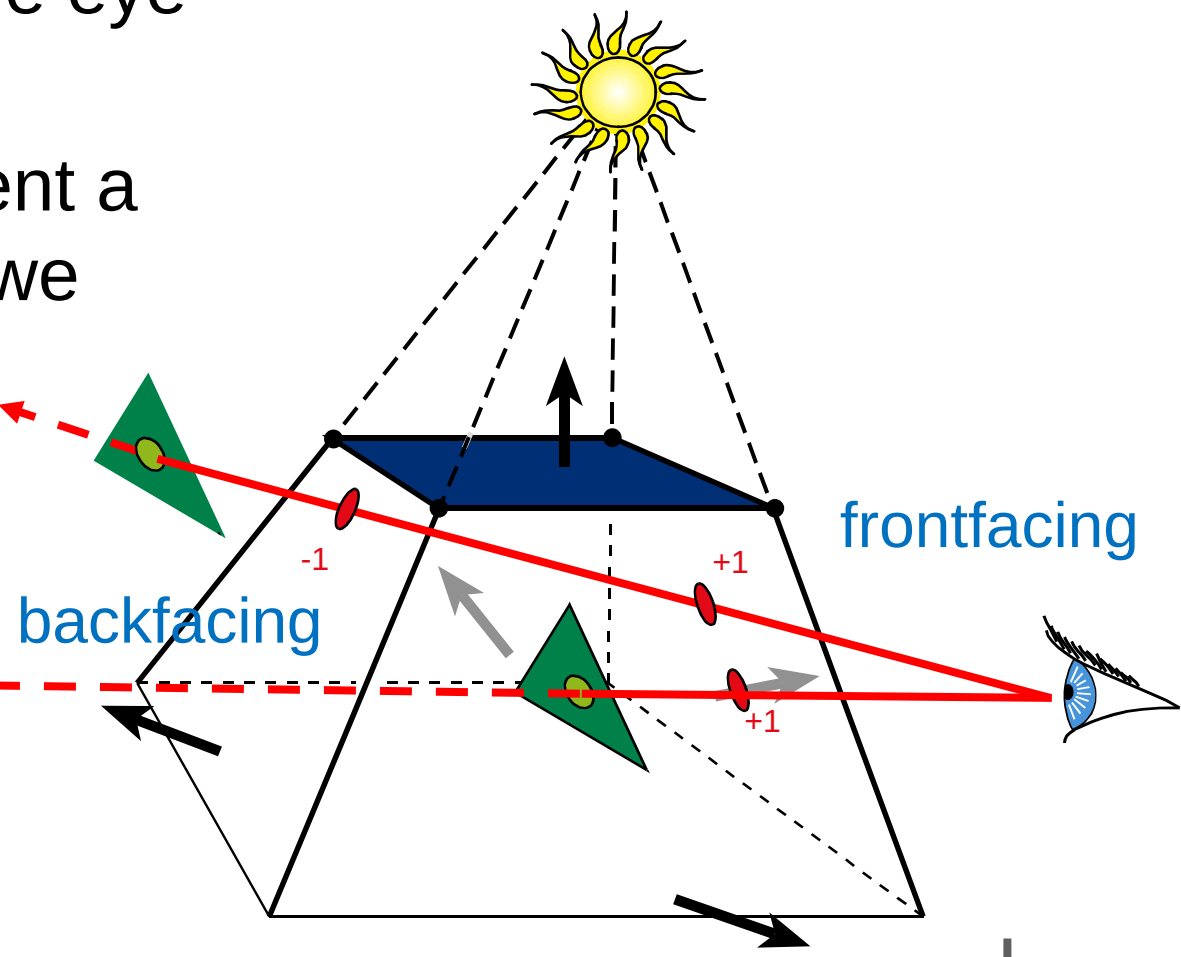
MIT Durand and Cutler

Shadow Volumes

- Shoot a ray from the eye to the visible point
- Increment/decrement a counter each time we intersect a shadow volume polygon *(check z buffer)*

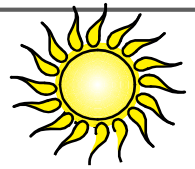
use front/back culling

- If the counter ≠ 0, the point is in shadow



Shadow Volumes with the Stencil Buffer

A four pass process [Heidmann91]:



1) Initialize stencil buffer to 0

Draw scene with ambient light only

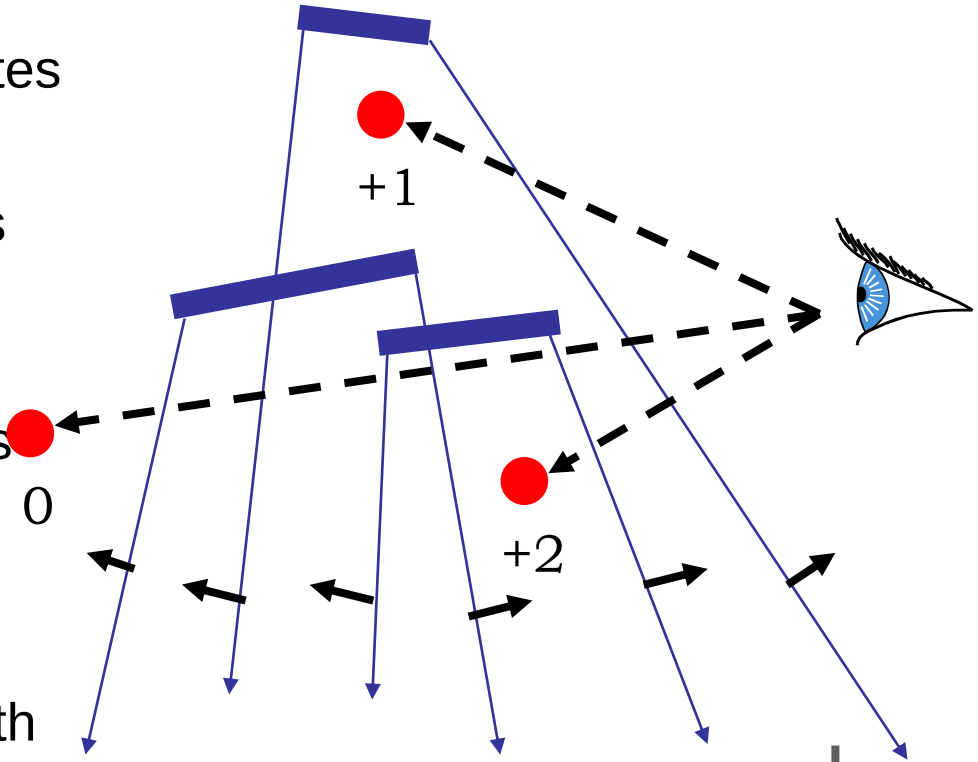
Turn off frame buffer & z-buffer updates

2)Draw front-facing shadow polygons

If z-pass → increment counter

3)Draw back-facing shadow polygons

If z-pass → decrement counter



4)Turn on frame buffer updates

Turn on lighting and redraw pixels with stencil buffer counter = 0

Image courtesy of NVIDIA Inc.

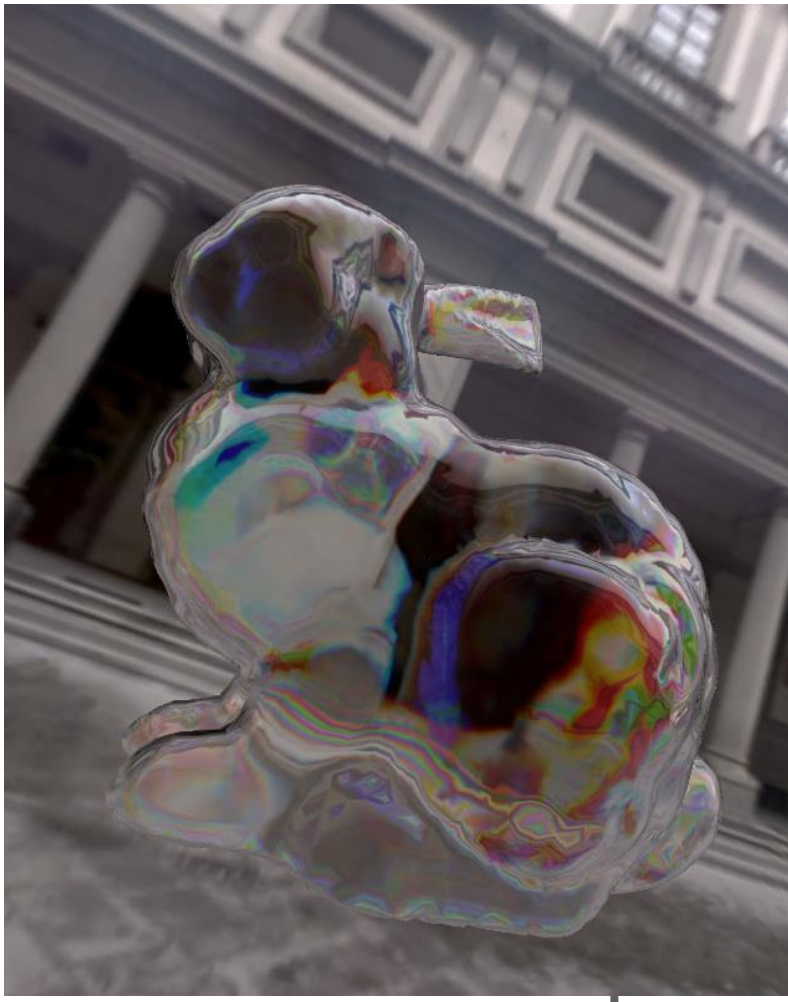
http://www-sop.inria.fr/reves/personnel/Marc.Stamminger/psm/

Alpha for Transparency

- Add an additional property to material colors called *alpha*
- Colors are actually (***r g b*** *a)*
- Alpha actually represents the *opacity* of an object and ranges from 0 (totally invisible) to 1 (fully opaque).

A value between 0 and 1 will be partially transparent

- The alpha value is a 8 bit quantity, so a color is a 32 bit rgba value. (default alpha =1 - fully opaque)



Alpha Blending: Computing Transparency

- When we are rasterize a transparent triangle, we end up with an alpha quantity per pixel

- We also end up with a color value per pixel which comes from the Gouraud interpolated color per vertex, or the texture map, or both. We refer to this as the ***source color***
- We can also read the color value already stored in the pixel. We call this the ***destination color***

- The final color we render into the pixel will be a linear blend of the source color and destination color. The alpha quantity controls how much we blend each one

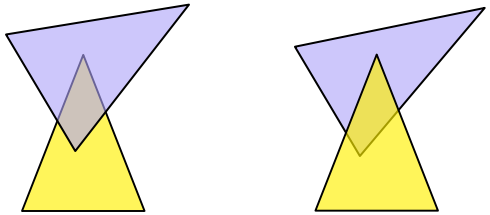
$$\mathbf{c}_{final} = \alpha \mathbf{c}_{src} + (1-\alpha)\mathbf{c}_{dest}$$

Basic Alpha Blending

- Blending Operator is not commutative

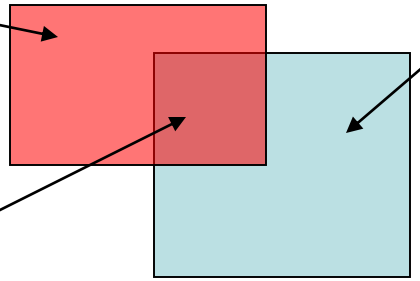
Front to Back

Back to Front



Transparent front color (src) with opacity α

Solid back color (dest)



$$\mathbf{c}_{final} = \alpha \mathbf{c}_{src} + (1-\alpha)\mathbf{c}_{dest}$$

Rendering with transparency

- Because of the zbuffer, we can generally render triangles in any order without affecting the final image
- With transparency, however, we actually need to render transparent surfaces in a back to front (far to near) order
- This is required because the transparent surface will modify the color already stored at the pixel

Rendering with transparency

- If we have a blue tinted glass in front of a brick wall, we render the brick wall first, then when we render the blue glass, we need to apply a blue tinting to the brick pixels already in the framebuffer

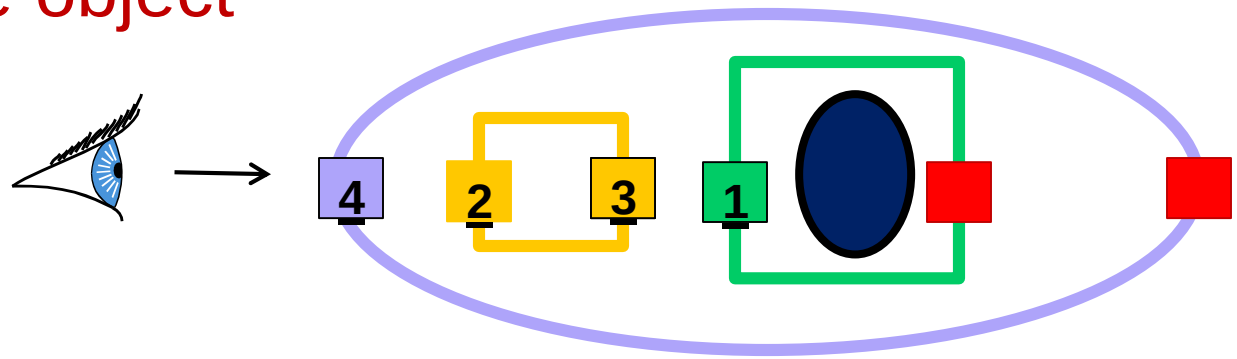
- We should render all opaque surfaces in a scene before rendering the transparent surfaces.
- For best quality, the transparent triangles should be sorted back to front, which can be an expensive operation

→**Order-Independent Transparency** (OIT) algorithms!

Rendering with transparency

1) Render Opaque object

- Depth-buffer rejects occluded fragments



2) Render Transparent object

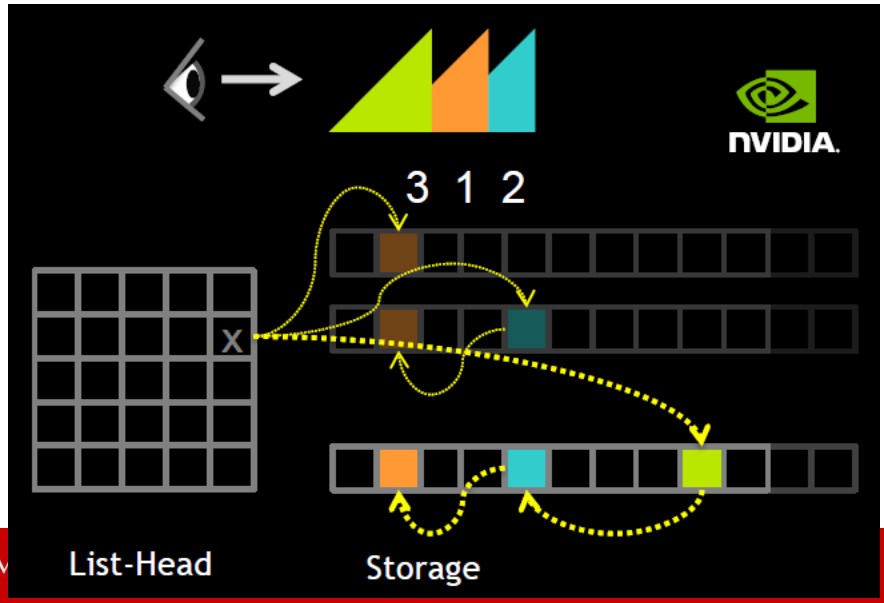
All fragments are stored using per-pixel linked lists

Linked List [Yang et al.]

Store fragments:

Color+alpha+depth (only the first K)

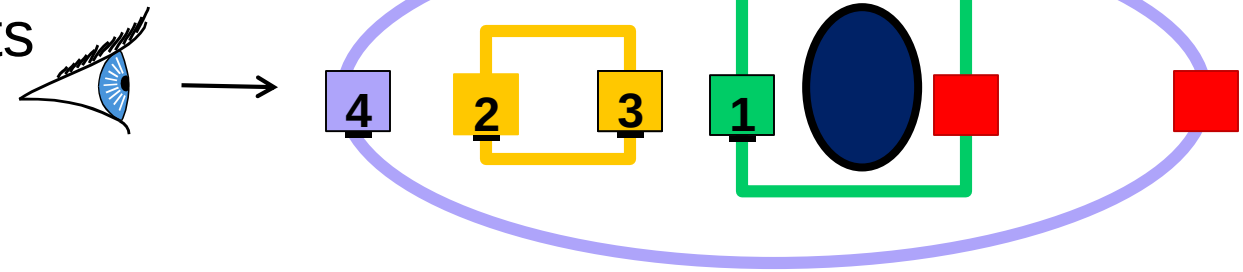
- Storage buffer: fragment + previous
  - Per-pixel list-head



Rendering with transparency

1) Render Opaque object

- Depth-buffer rejects occluded fragments



2) Render Transparent object

– Store fragments: Color+alpha+depth (only the first K)

3) Resolve Transparent

– Fullscreen sort & blend per pixel

Not efficient to globally sort all fragments per pixel

– Sort K nearest correctly via register array

– Blend fullscreen on top of framebuffer

Serena Morigi

Dipartimento di Matematica

serena.morigi@unibo.it

http://www.dm.unibo.it/~morigi