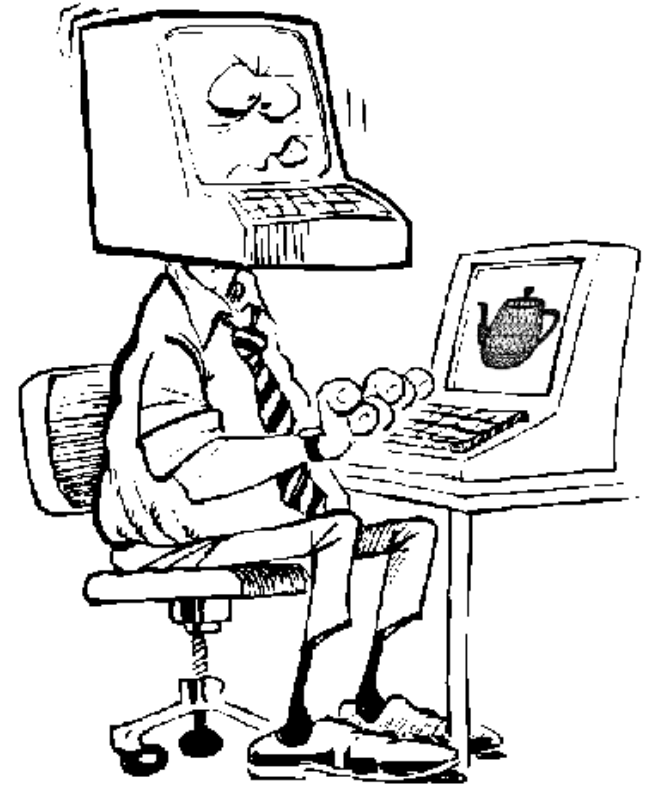




Introduction

Diego Gutiérrez

Introducción

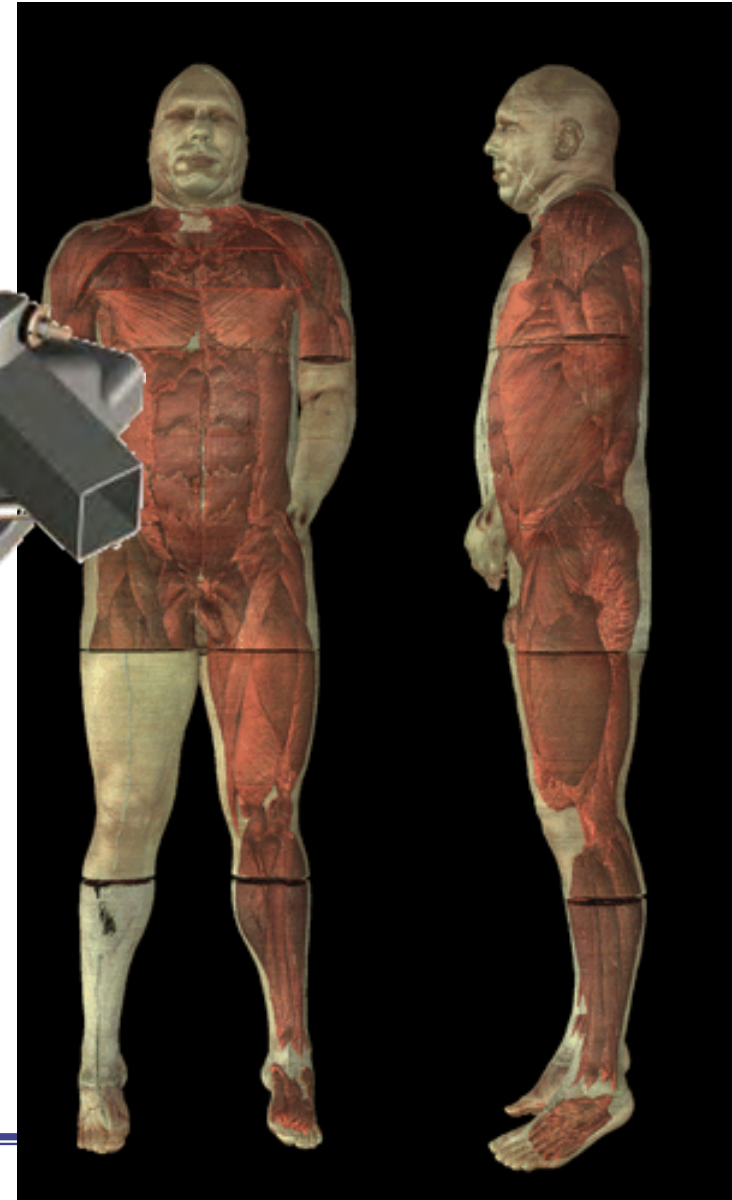
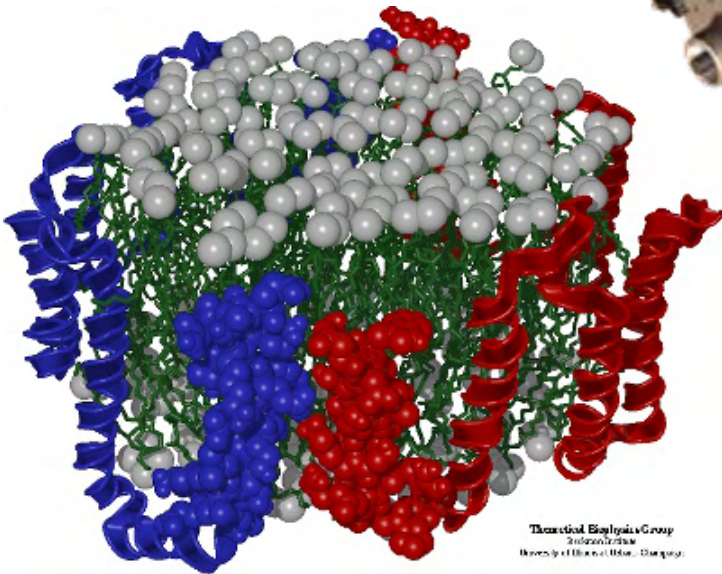
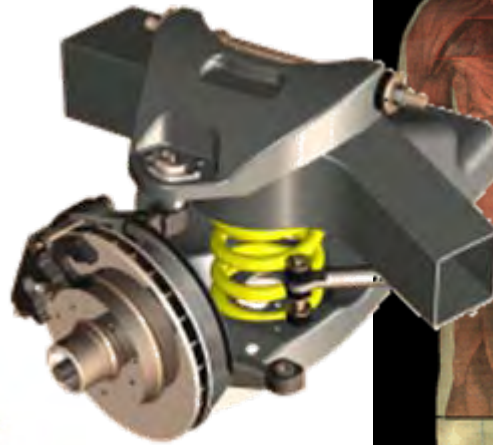
- ◆ CG is a technology to present information
- ◆ *If you can imagine it, you can do it in CG*
- ◆ It's... Everywhere!



Introducción

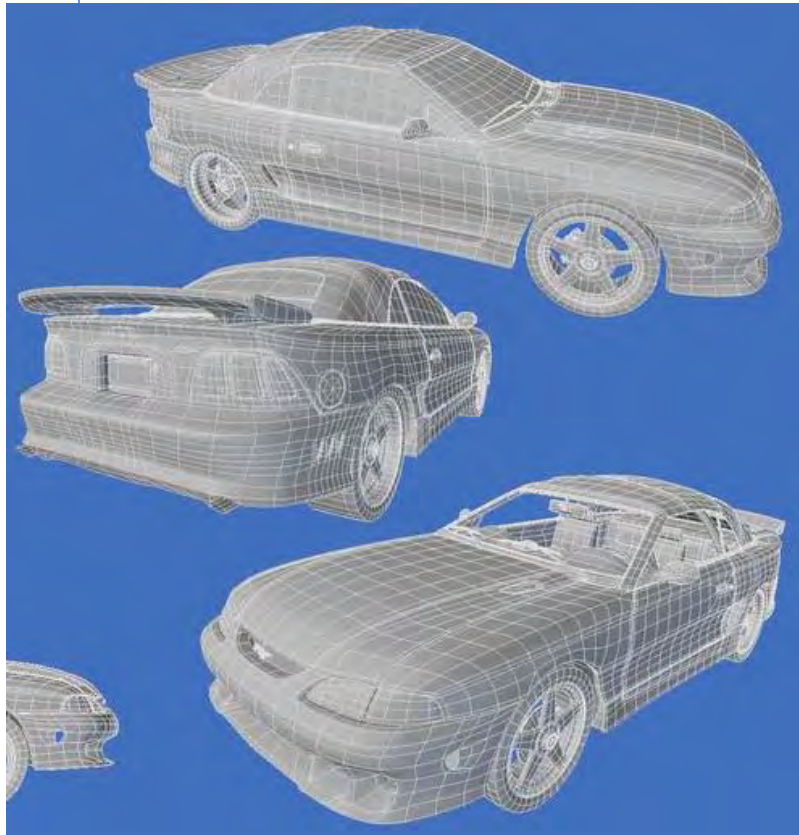


Ambitos



Introducción

- ◆ Given a 3D scene... How do we obtain a realistic image?



Introducción

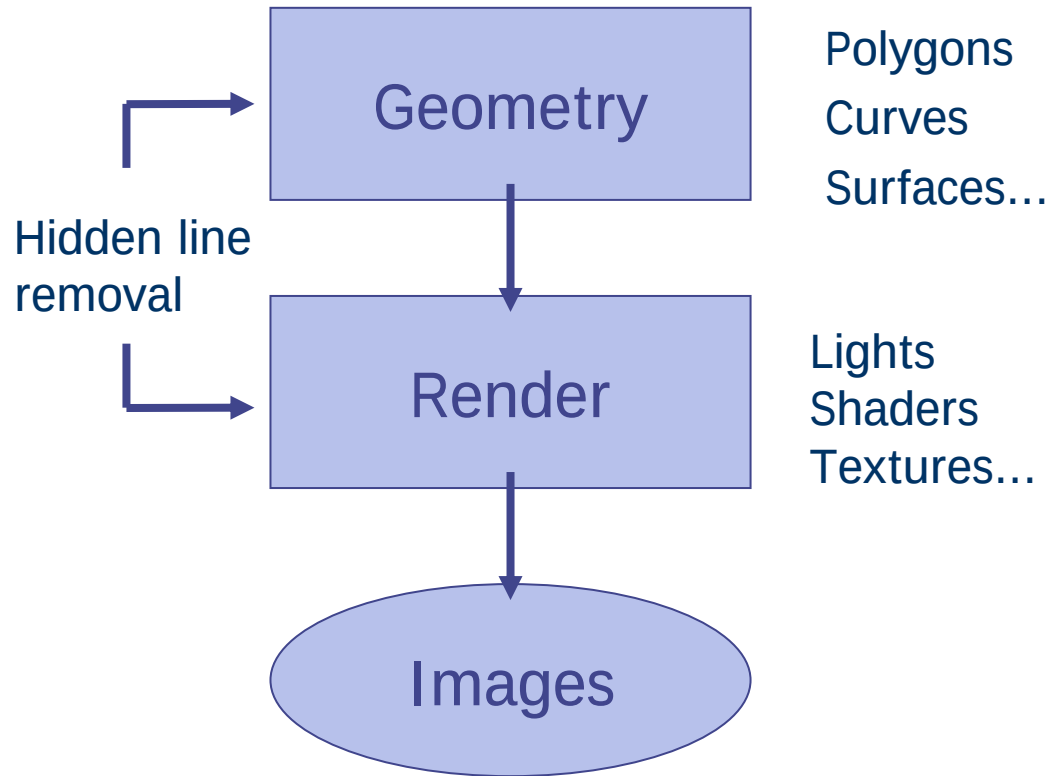
◆ We need:

- Scene description:
 - ◆ Shapes of objects
 - ◆ Localization and orientation
 - ◆ Surface properties
 - ◆ Description of light sources
 - ◆ ...
- Visibility information:
 - ◆ Which points are visible?
 - ◆ How are they lit?
 - ◆ ...

◆ We're going to mainly focus on the latter question

Introducción

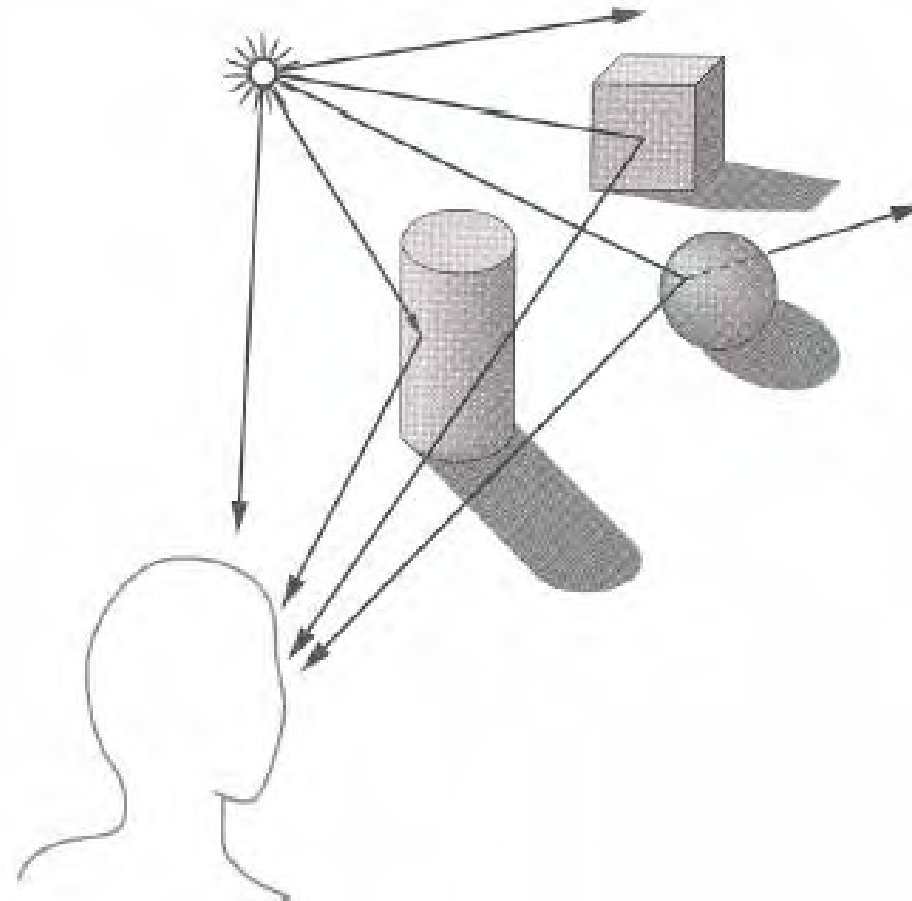
◆ (highly oversimplified) Rendering pipeline



Introducción

- ◆ Once we chose a POV, the goal is to know how each surface (point) is lit
- ◆ We'll consider two types of illumination for each point:
 - Direct lighting
 - Indirect lighting
- ◆ ... and two types of algorithms:
 - Local illumination
 - Global illumination (GI)

Introducción



Introducción

- ◆ Two angles to solve the same problem:
 - Heuristics: use all the tricks you can pull off to make it look good (*top-down*)
 - ◆ Interpolation
 - ◆ Gouraud
 - ◆ Phong
 - Physics-based: model the physical processes that govern light transport and let the image appear (*bottom-up*)
 - ◆ Ray tracing
 - ◆ Photon mapping
- ◆ The latter are more expensive but more realistic

Introducción

- ◆ These two algorithms use very different approaches to begin with:
 - Ray tracing: image space. Find out which points are visible, then worry about how they're lit
 - Photon mapping: first worry about light, then about visibility



Local Illumination Models

Diego Gutiérrez

Index

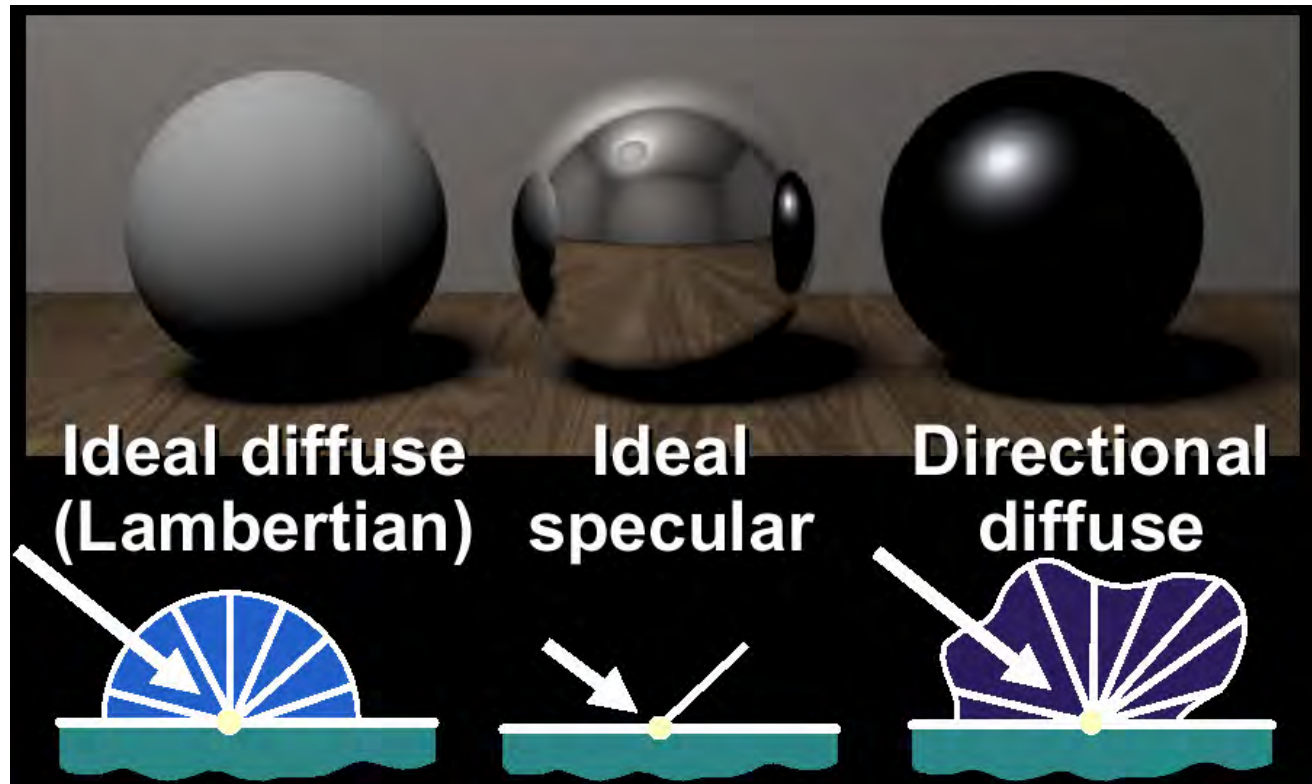
- ◆ Intro
- ◆ Types of reflections
- ◆ A simple illumination model
- ◆ Diffuse reflection: a second model
- ◆ Specular reflection: a third model
- ◆ A final model
- ◆ Transparency
- ◆ Color



Introduction

- ◆ We start with **local illumination models**
- ◆ In this models, only light coming directly from light sources is calculated
- ◆ Reflection (specular, diffuse) of light on objects is what allows us to perceive them
- ◆ Even though color is a very important aspect...
 - In the definition of light sources
 - In material properties
 - In the generation of effects
- ◆ ...it will not be discussed here for reasons of clarity

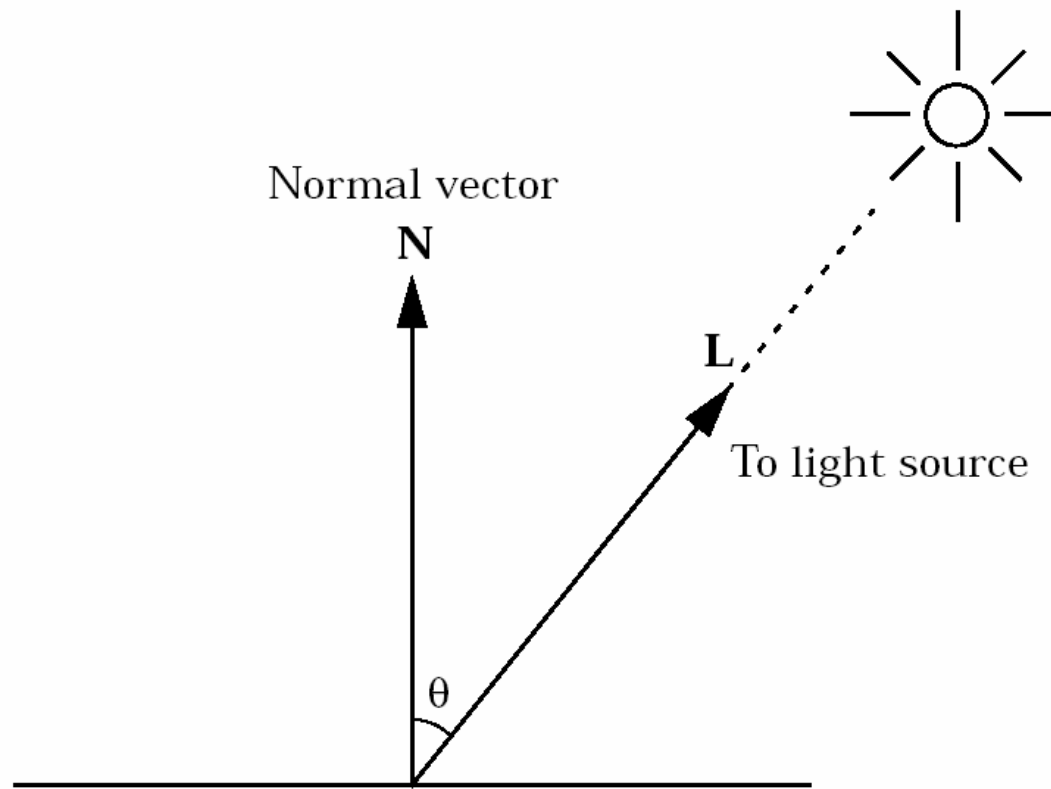
Types of reflections



- ◆ In the real world all materials are, to a greater or lesser extent, directional diffuse (except for perfect mirrors)

Surface orientation

- ◆ **N** and **L** are unitary vectors
- ◆ **Theta** is the angle of incidence



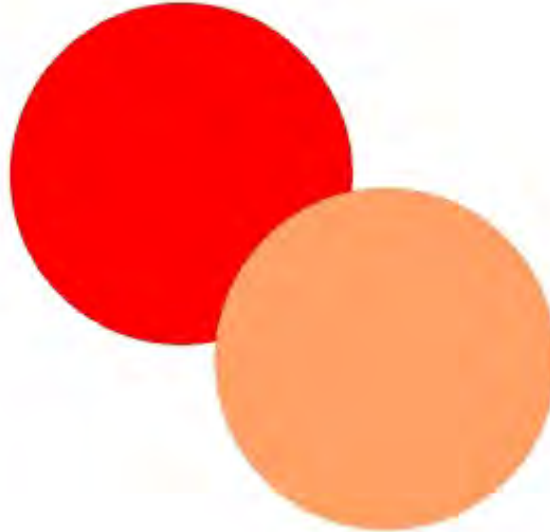
A simple illumination model

- ◆ We will consider just **ambient illumination**:
 - Uniform from all directions (intensity I_a)
 - Is the result of multiple reflections
 - $I = I_a K_d$
 - K_d measures reflectivity of a surface to diffuse (ambient) light. Named *diffuse reflection coefficient*.
 - K_d takes values between 0 and 1

◆ Problems?

A simple illumination model

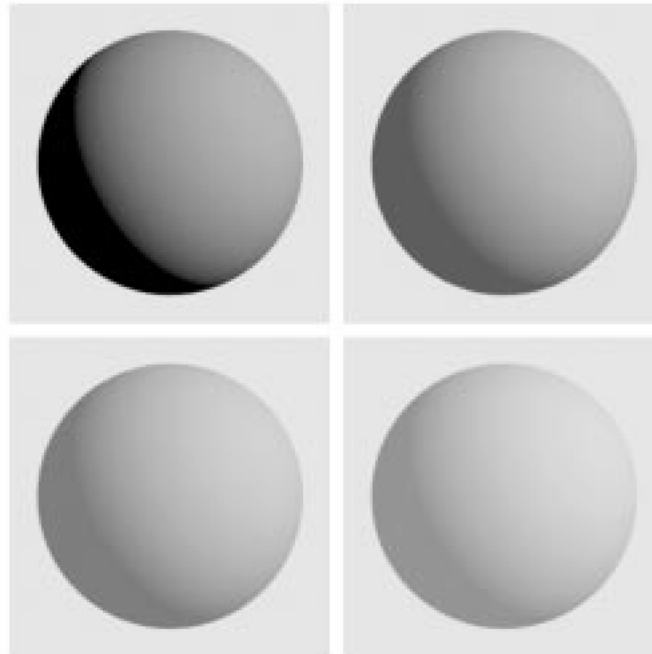
- ◆ If we use the previous equation, every point of the object will have the same intensity!
- ◆ We lose tridimensionality



- ◆ Are they spheres or circles?

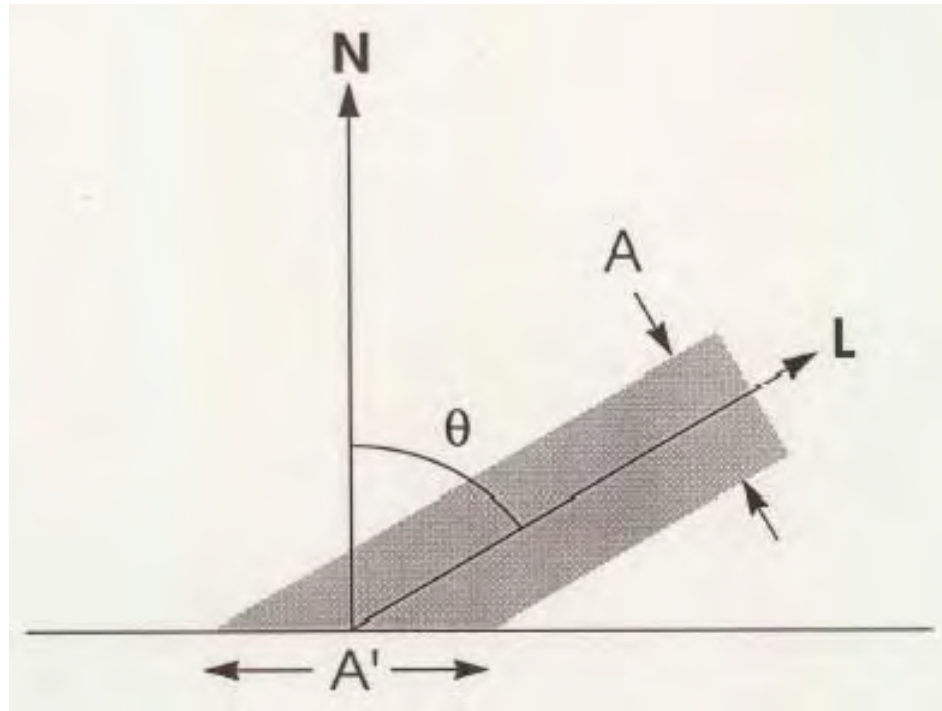
A simple illumination model

- ◆ Ambient illumination is bad, bad, bad



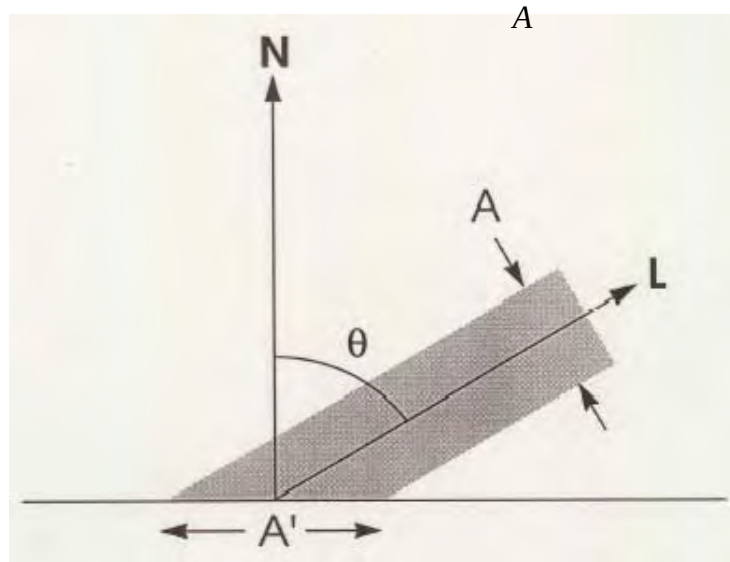
Lambert's cosine law

- ◆ The intensity of the light reflected by a surface depends on the intensity of the incident light, which in turn depends on the light source and the orientation of the surface with respect to it



Lambert's cosine law

- ◆ Light intensity I_p of area A
- ◆ Light is dispersed over an area A'
- ◆ $A' = A / \cos \theta$
- ◆ $A = A' \cos \theta$
- ◆ Thus, the effective intensity on the surface is $I_e = I_p \cos \theta$

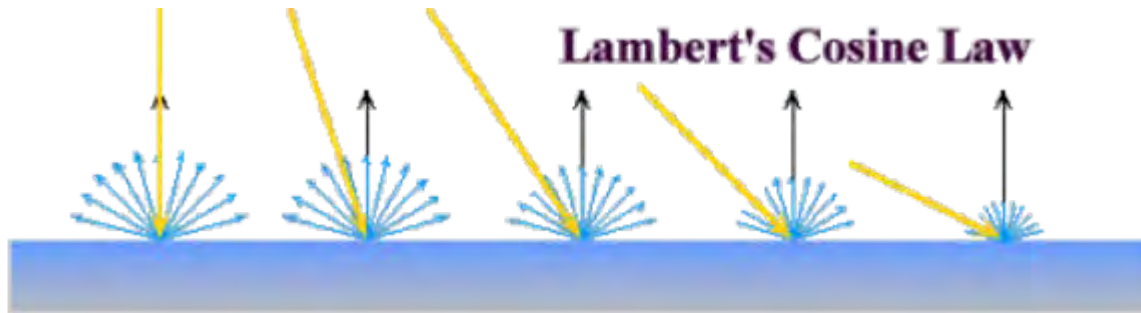


Lambert's cosine law

- ◆ Why do our shoulders get more sun burnt in summer?
- ◆ Ideal diffusers behave according to this law (*Lambertian* materials)
- ◆ Reflected intensity is **independent of the visualization direction**
- ◆ It only depends on the **orientation of the surface with respect to the incident light**



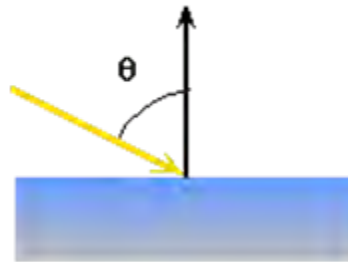
Lambert's cosine law



Diffuse reflection

- ◆ Intensity of diffusely reflected light can be calculated as:

$$I_{diffuse} = k_d I_{light} \cos \theta$$



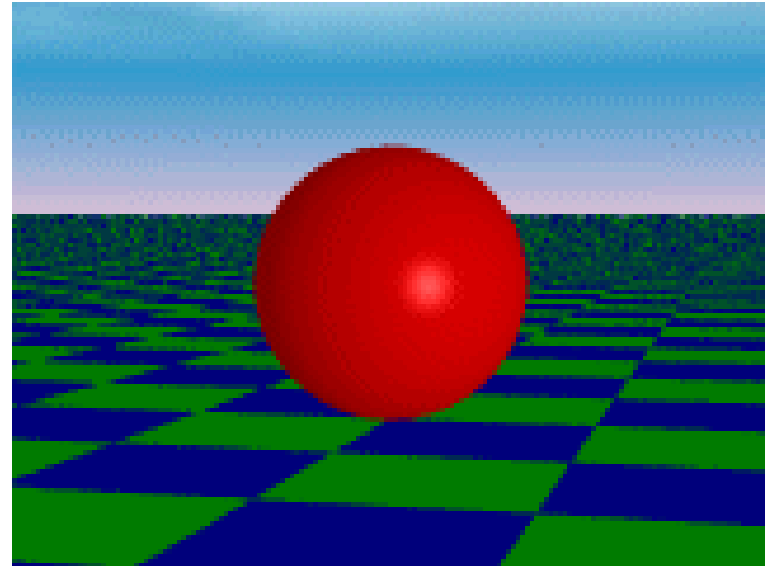
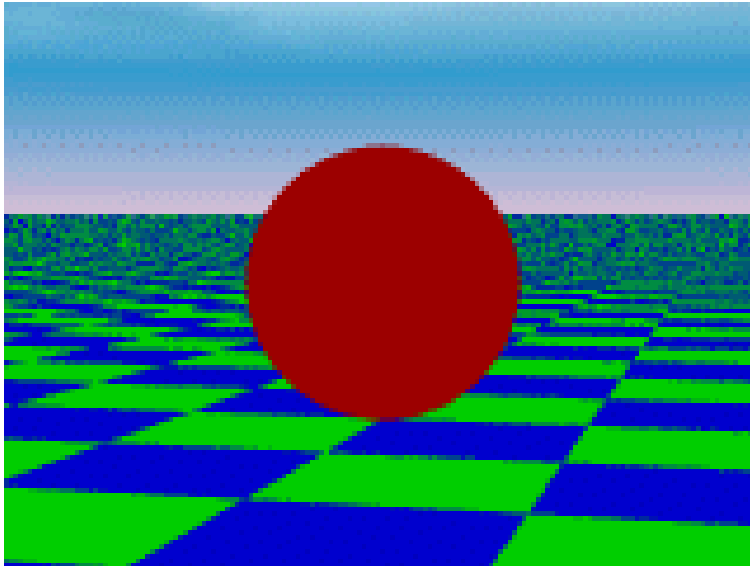
- ◆ Given that **N** and **L** are unitary vectors, and thus:

$$\cos \theta = \mathbf{N} \cdot \mathbf{L}$$

- ◆ Intensity of diffusely reflected light is then:

$$I_{diffuse} = k_d I_{light} (\bar{n} \cdot \bar{l})$$

Diffuse reflection



Diffuse reflection: a second model

- ◆ We will now consider the ambient component plus the diffuse component (coming from a point light source)

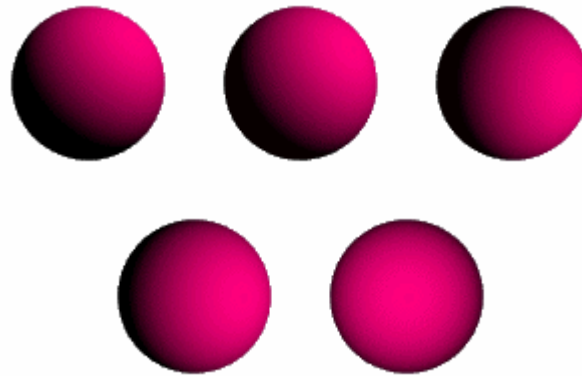
$$I = \text{ambient} + \text{diffuse}$$

$$I = k_d I_a + k_d I_p (\mathbf{N} \cdot \mathbf{L})$$

- ◆ For the diffuse component, when the scalar product is negative (angles greater than 90°), the surface does not “see” the light and the reflected energy is zero

Diffuse reflection: a second model

- ◆ Examples of lambertian spheres with light sources placed at different angles:



- ◆ Why are spheres used for illumination and shading tests?

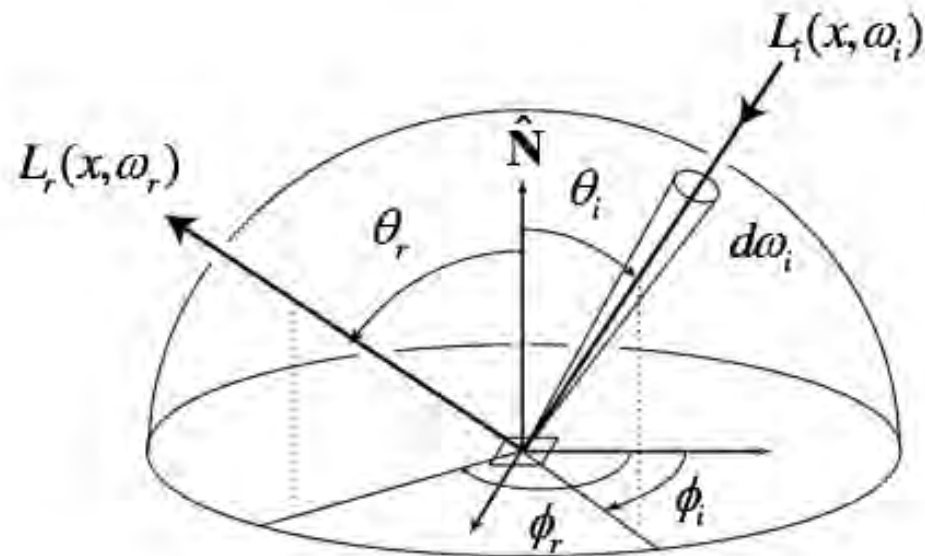


Specular reflection

- ◆ Only matt objects can be modeled with the Lambertian model
- ◆ Bright surfaces offer specular reflections (*highlights*) under certain viewing angles.
- ◆ Caution with the terminology! (specular reflection vs. *highlight*)
- ◆ The position of the *highlight* is a function of the viewing point of the scene: we introduce **dependance on the visualization direction**
- ◆ *Highlights* occur over a narrow range of angles
- ◆ The color of the *highlight* is usually the same as the color of the light causing that *highlight*
- ◆ Mirrors are an example of perfect specular reflection: angle of reflection equal to angle of incidence

Specular reflection

◆ Reflection equation

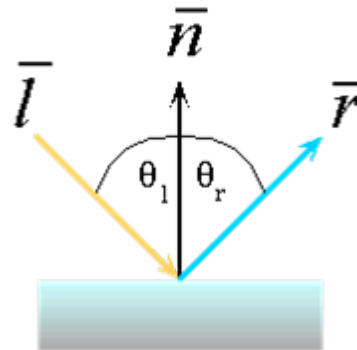


$$L_r(x, \omega_r) = \int_{H^2} f_r(x, \omega_i \rightarrow \omega_r) L_i(x, \omega_i) \cos \theta_i d\omega_i$$

Specular reflection

- ◆ The ray, the normal and the reflected ray belong to the same plane
- ◆ Snell's law:

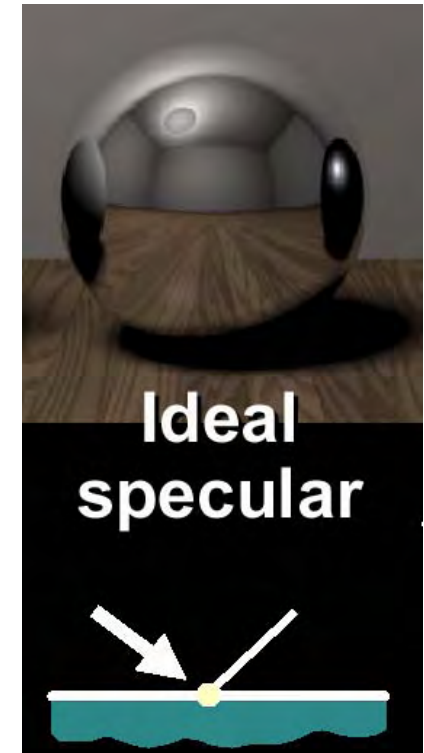
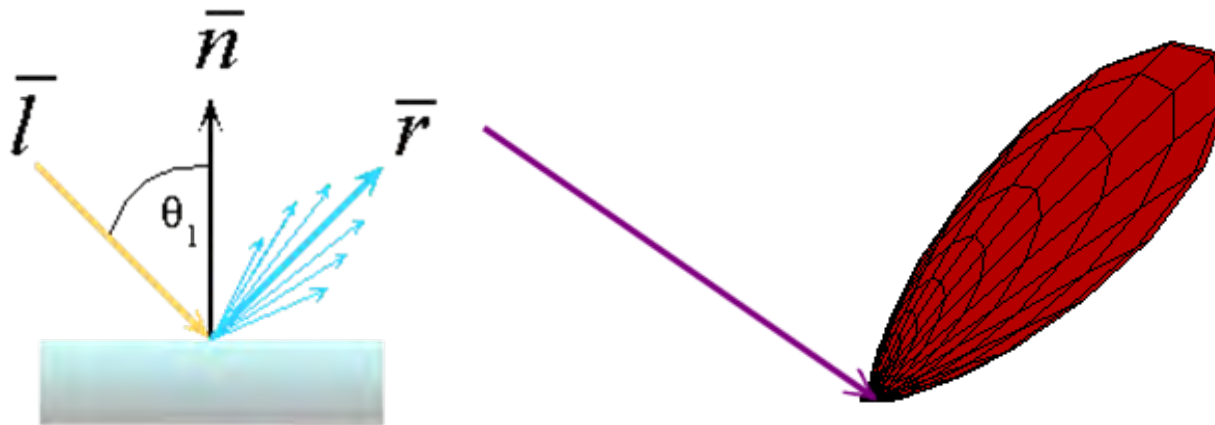
$$n_i \sin \theta_i = n_r \sin \theta_r$$



- ◆ In specular reflection: $\theta_i = \theta_r$

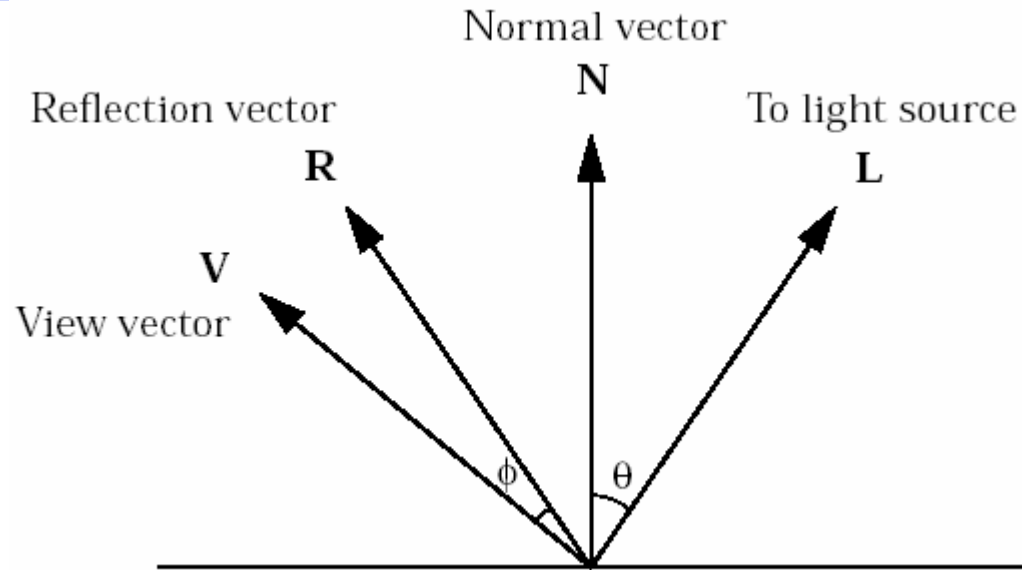
Specular reflection

- ◆ No real material is perfect specular
- ◆ Imperfections of the surface cause directions of reflected rays to differ from the theoretical direction



- ◆ We will introduce now an empirical model to calculate the specular reflection: **Phong**

Phong illumination model



◆ We introduce more notation:

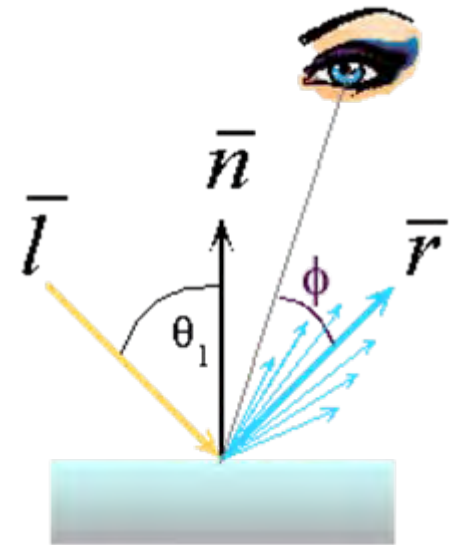
- **N** is the normal of the surface
- **L** is the direction towards the light source
- **V** is the direction towards the viewing point
- **R** is the direction of the ideal specular reflection
- As we will see, intensity of specular reflection depends on the angle between **V** and **R**

Phong illumination model

- Warning: do not confuse with the *Phong interpolation model*
- The Phong illumination model is heuristic, it has no physical basis...
- ...and it is one of the most used in CG

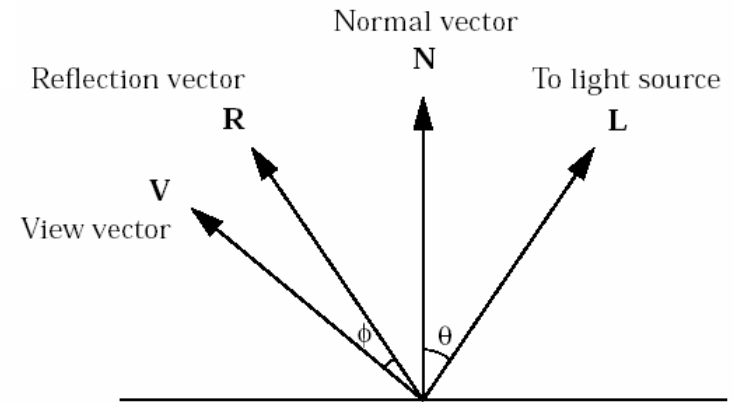
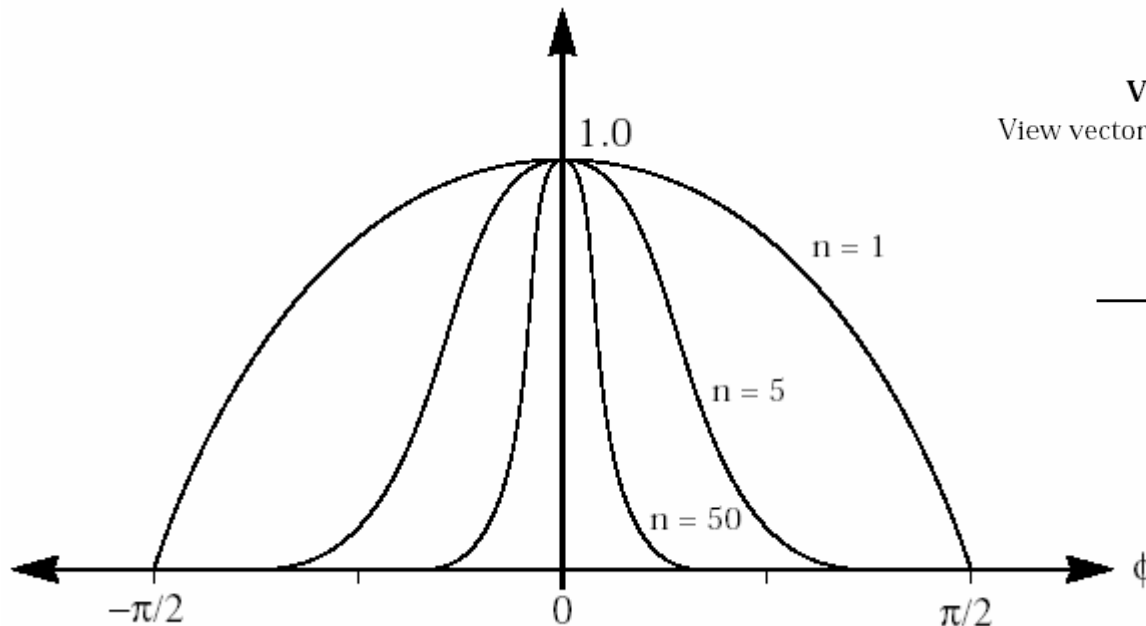
$$I_{\text{specular}} = I_{\text{light}} (\cos \phi)^{n_{\text{shiny}}}$$

- The term $\cos \phi$ reaches its maximum when the surface is seen from the specular direction, and is zero when seen from a direction forming 90° with the specular direction. The term n_{shiny} controls that decay.



Phong illumination model

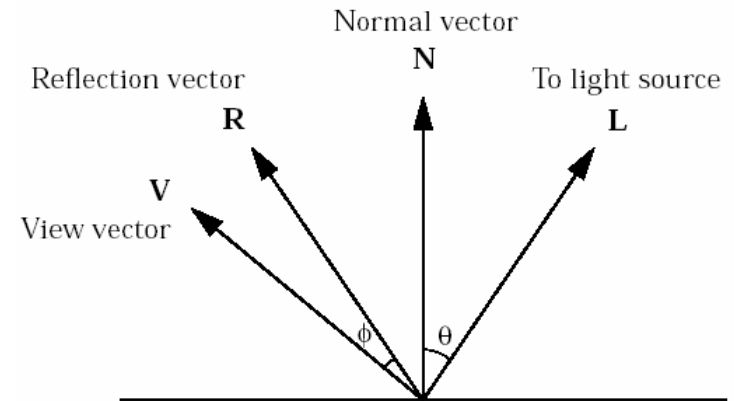
- ◆ n (or n_{shiny}) depends on the surface:
 - Ideal reflector: $n=\infty$
 - Nearly null reflector (a sponge): $n=1$
 - In practice $n[1..200]$



Phong illumination model

- ◆ We can evaluate $\cos \phi$ vectorially:
 - $\cos \phi = \mathbf{R} \cdot \mathbf{V}$ (as both vectors are unitary)
- ◆ Specular reflection also depends on θ
- ◆ So:

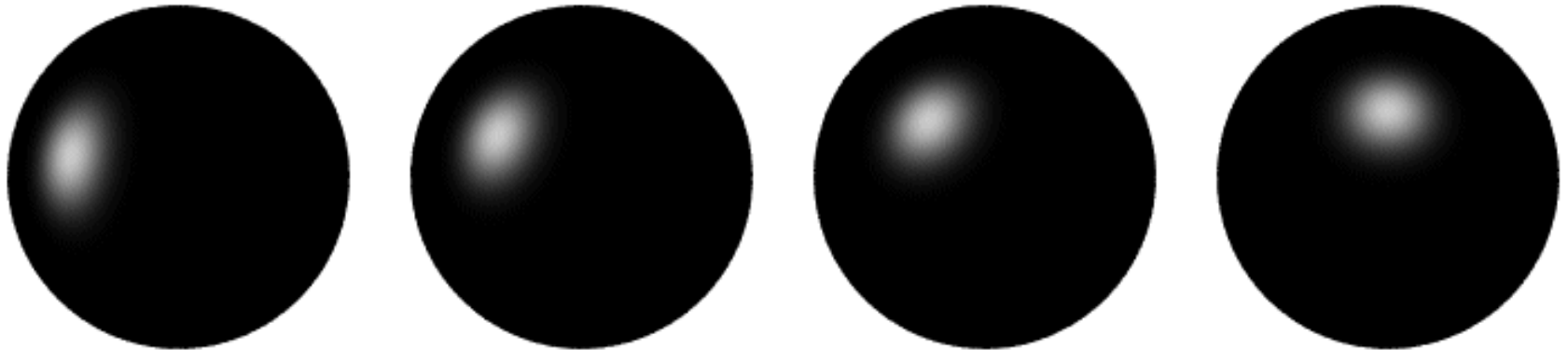
$$I_s \propto W(\theta) (\mathbf{R} \cdot \mathbf{V})^n$$



- ◆ In practice, K_s is used instead of $W(\theta)$
- ◆ K_s is the **specular reflection coefficient**
- ◆ $K_s[0..1]$

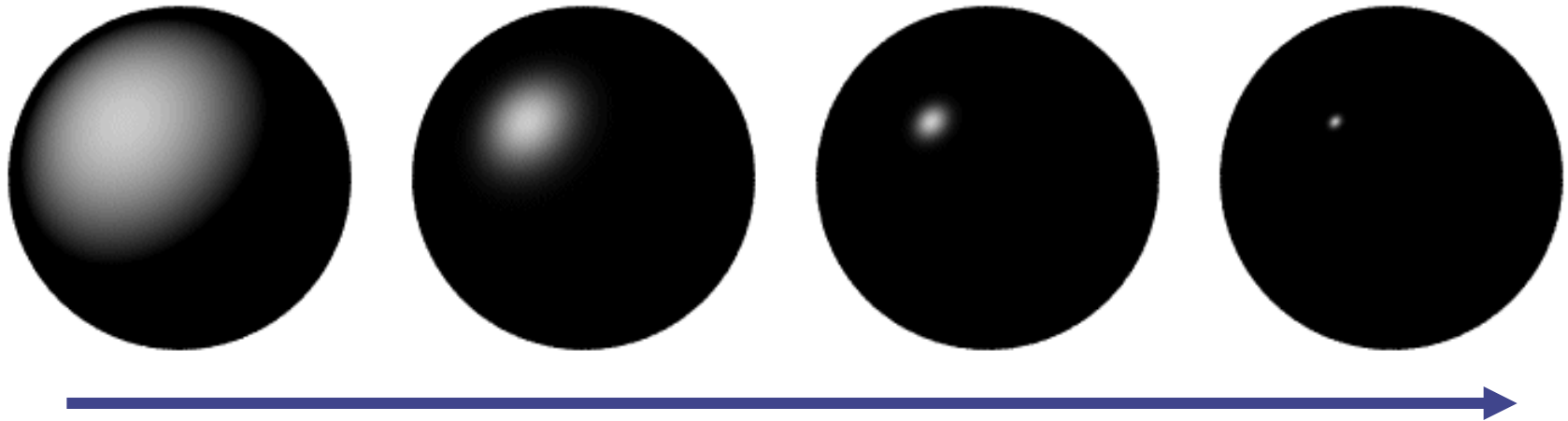
Phong illumination model

- ◆ Specular reflections (Phong's model) with constant n and varying the position of the light source.



Phong illumination model

- ◆ Specular reflections (Phong's model) increasing n and with the position of the light source fixed



A third illumination model

◆ We had:

$$I = \text{ambient} + \text{diffuse}$$

$$I = k_d I_a + k_d I_p (\mathbf{N} \cdot \mathbf{L})$$

◆ We continue to complete the model:

$$I = \text{ambient} + \text{diffuse} + \text{specular}$$

$$I = k_d I_a + I_p [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{R} \cdot \mathbf{V})^n]$$

A third illumination model

Examples:

➤ $K_d = 0.4$

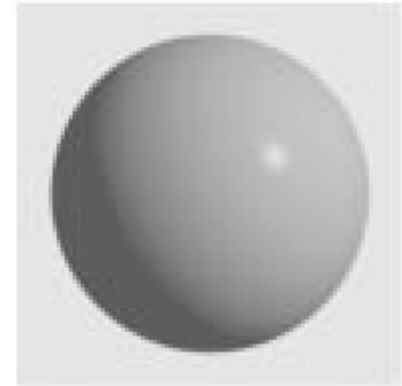
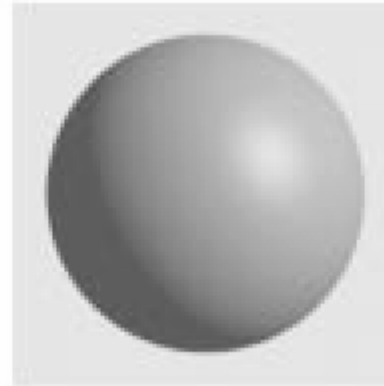
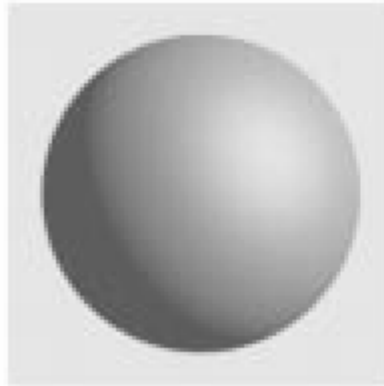
$$I_a = 0.2 I_p$$

$n=3$

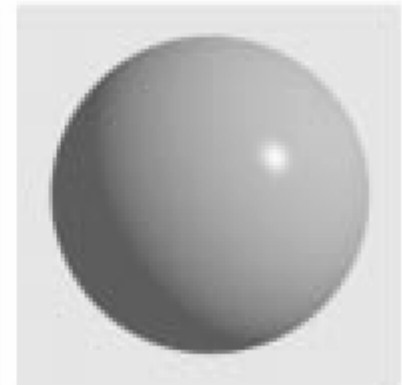
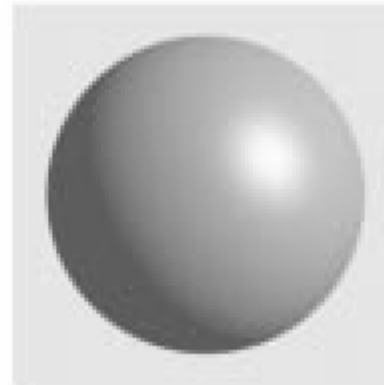
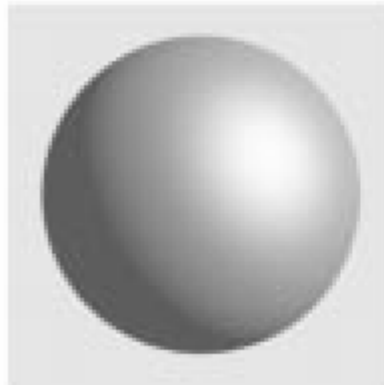
$n=10$

$n=100$

$K_s = 0.25$



$K_s = 0.5$



A third illumination model

- ◆ The intensity of a light source decreases with the square of the distance:

- $I = I_p / (4\pi d^2)$

- ◆ This is usually replaced with this experimental expression:

- $I = I_p / (d + d_0)$

...where d_0 is an empirically determined constant

- ◆ Once again an empirical trick (*hack*) substitutes for a model of the physical phenomenon
- ◆ There are many “empirically determined constants” in CG

A fourth illumination model

- ◆ Yet one more model: we introduce attenuation of light with distance and we get:

$$I = k_d I_a + \frac{I_p}{d + d_0} [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s \cos^n \phi]$$

- ◆ Or in vectorial form (unitary vectors!):

$$I = k_d I_a + \frac{I_p}{d + d_0} [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{R} \cdot \mathbf{V})^n]$$

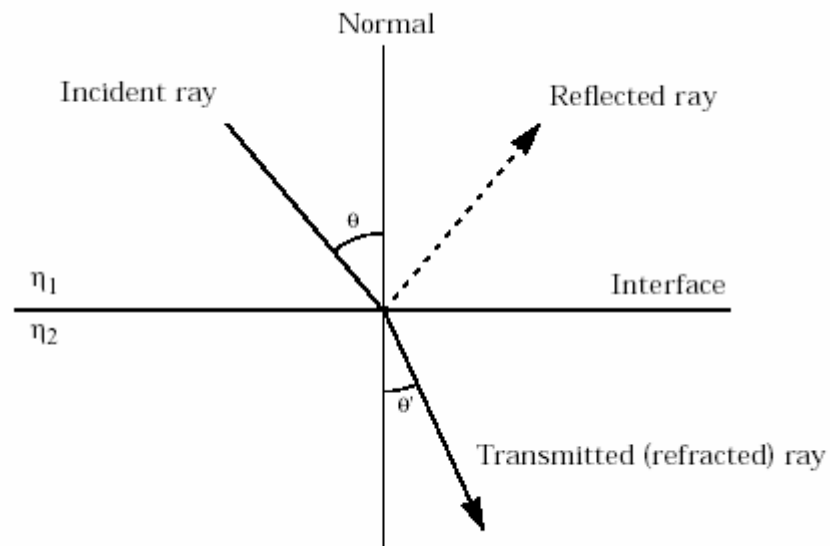
- ◆ If there is more than one light beam:

$$I = \text{ambient term} + \sum_{i=1}^n \text{diffuse term}_i + \sum_{i=1}^n \text{specular term}_i$$

Transparency

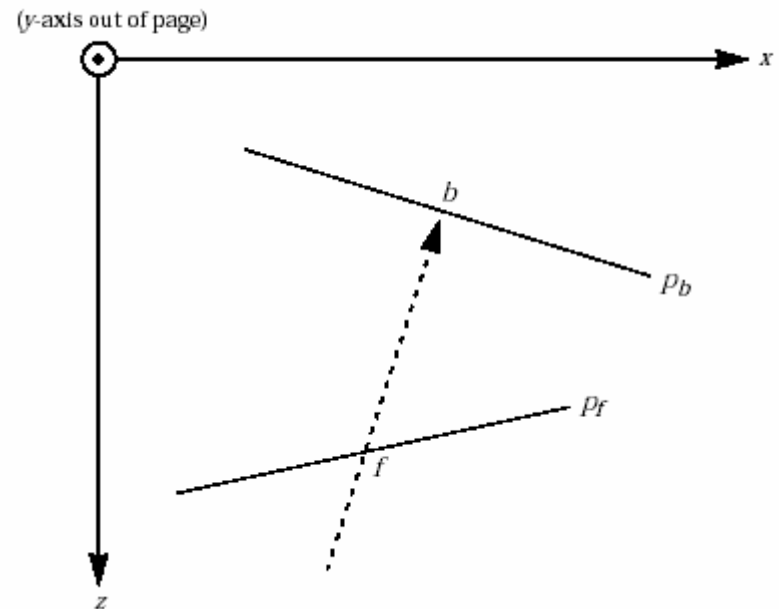
◆ Recall Snell's law:

$$n_i \sin \theta_i = n_r \sin \theta_r$$



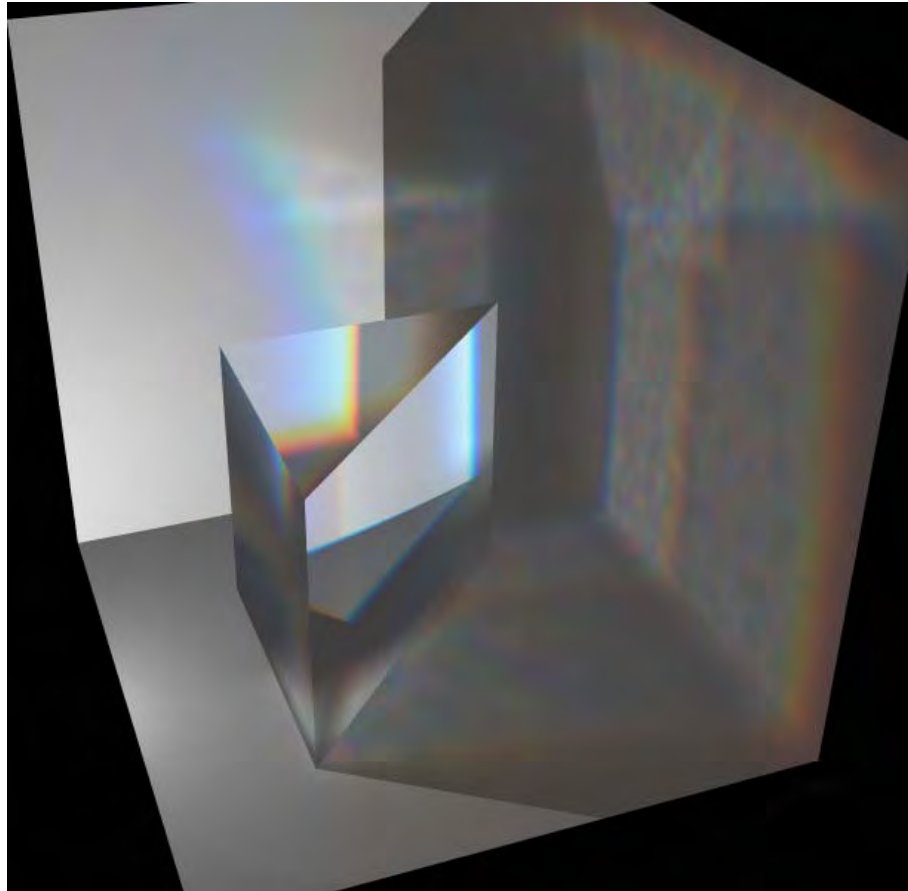
Transparency

- ◆ Transparency without refraction:
 - t is the **transmission coefficient**
 - Opaque object: $t=1$; Transparent object: $t=0$
 - t can be a function of color (e.g. church windows)
- ◆ In the example, observed intensity would be:
 - $I = tI_f + (1-t)I_b$



Dispersion

- ◆ The refractive index n is $f(\lambda)$: **dispersion**



Color

- ◆ Until now we have not considered color, we have just referred to light intensity
- ◆ The color model most vastly used in CG is RGB (which does not mean it is the best color model, not even the second best...)
- ◆ *We decouple the solution in three linearly independent wavelengths*

What?

Color

- ◆ We decouple the solution in three linearly independent wavelengths:

- R,G,B
- For Red:

$$I_R = k_{dR} I_{aR} + \frac{I_{pR}}{d + d_0} [k_{dR} (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{R} \cdot \mathbf{V})^n]$$

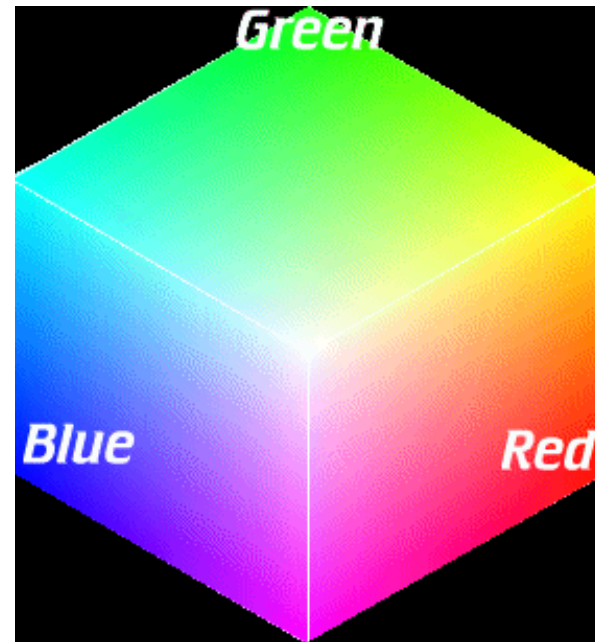
- It can be seen how we maintain the assumption that the specular highlight has the same color as the light source

- ◆ More complex spectral models deliver better solutions:

$$I_\lambda = k_{d\lambda} I_{a\lambda} + \frac{I_{p\lambda}}{d + d_0} [k_{d\lambda} (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{R} \cdot \mathbf{V})^n]$$

Color

- ◆ When working with RGB, the values obtained $[0..255]$ can be directly mapped to the RGB channels of a display
- ◆ And what happens if we exceed the maximum value?
- ◆ *Tone reproduction*

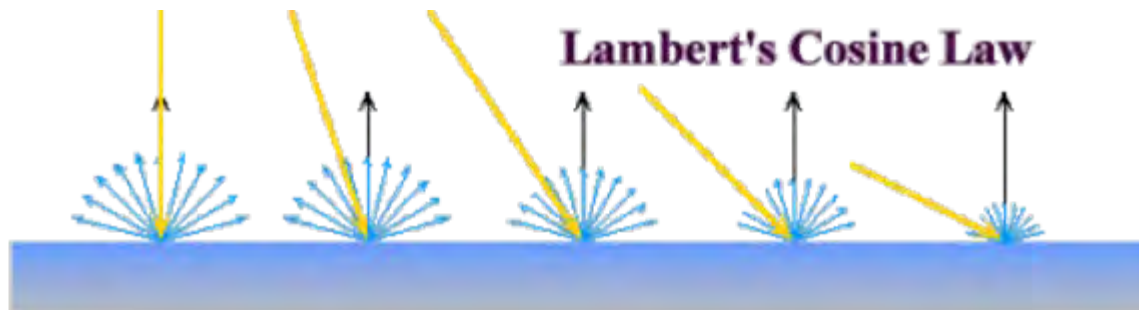
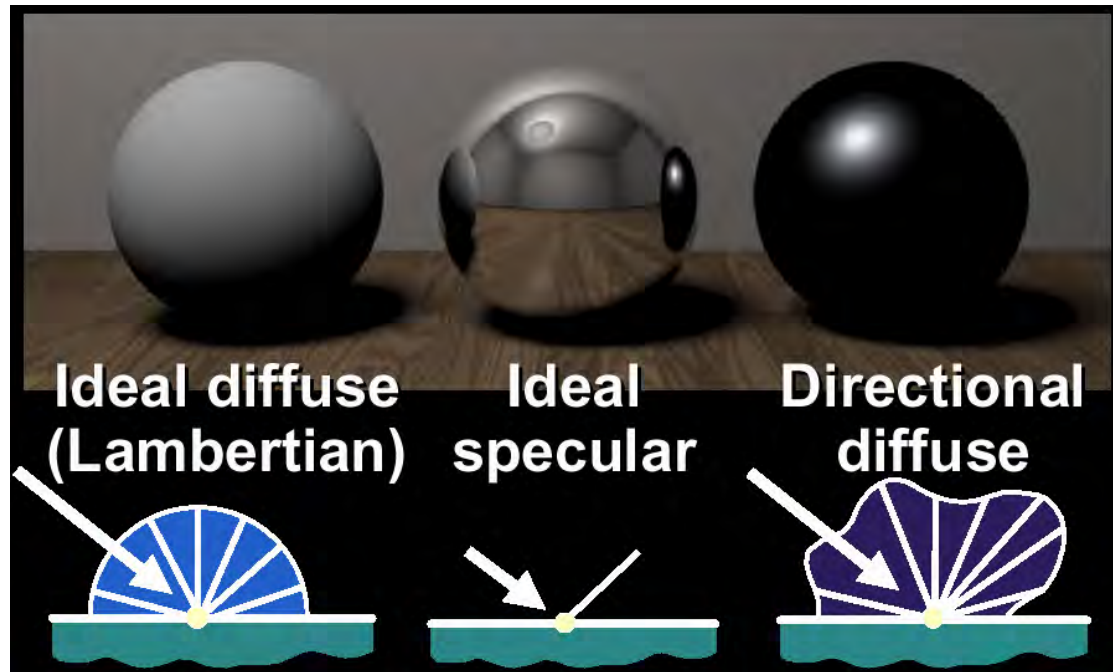




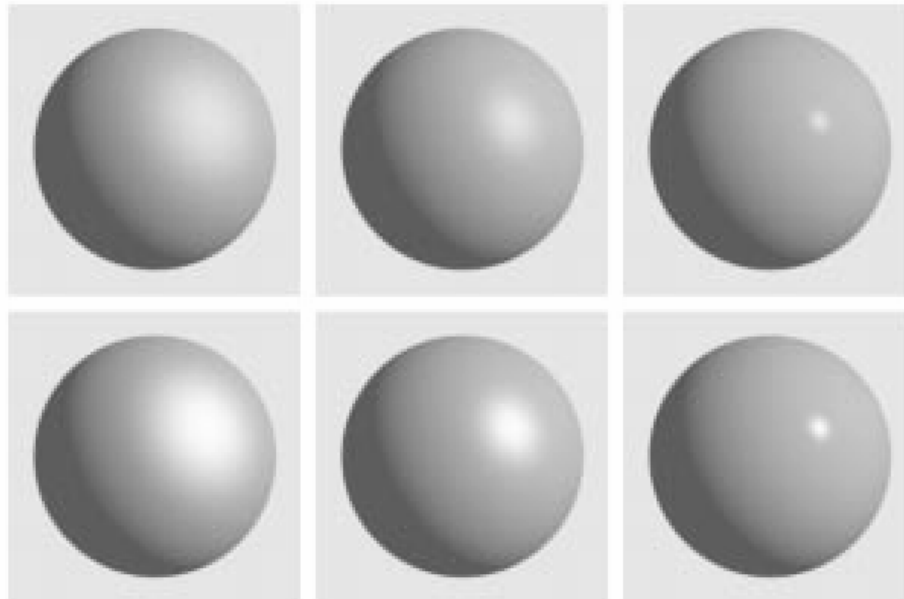
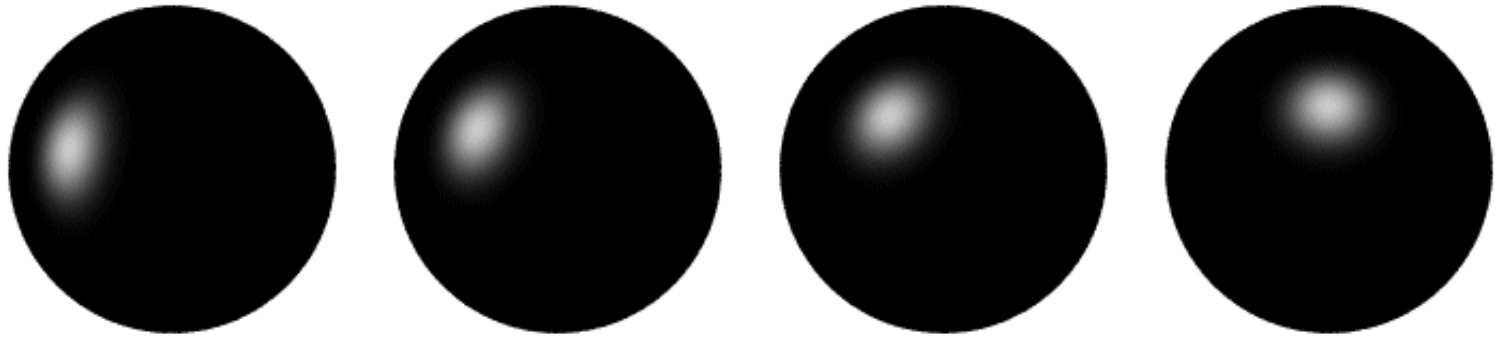
Local Illumination Models (in a nutshell)

Diego Gutiérrez

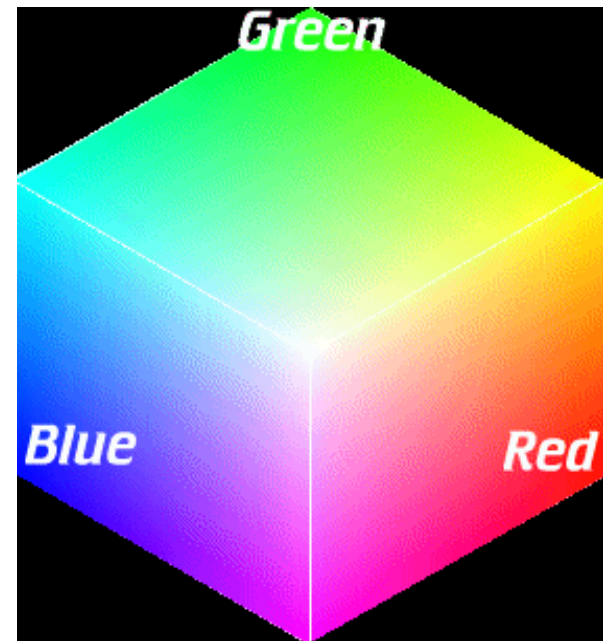
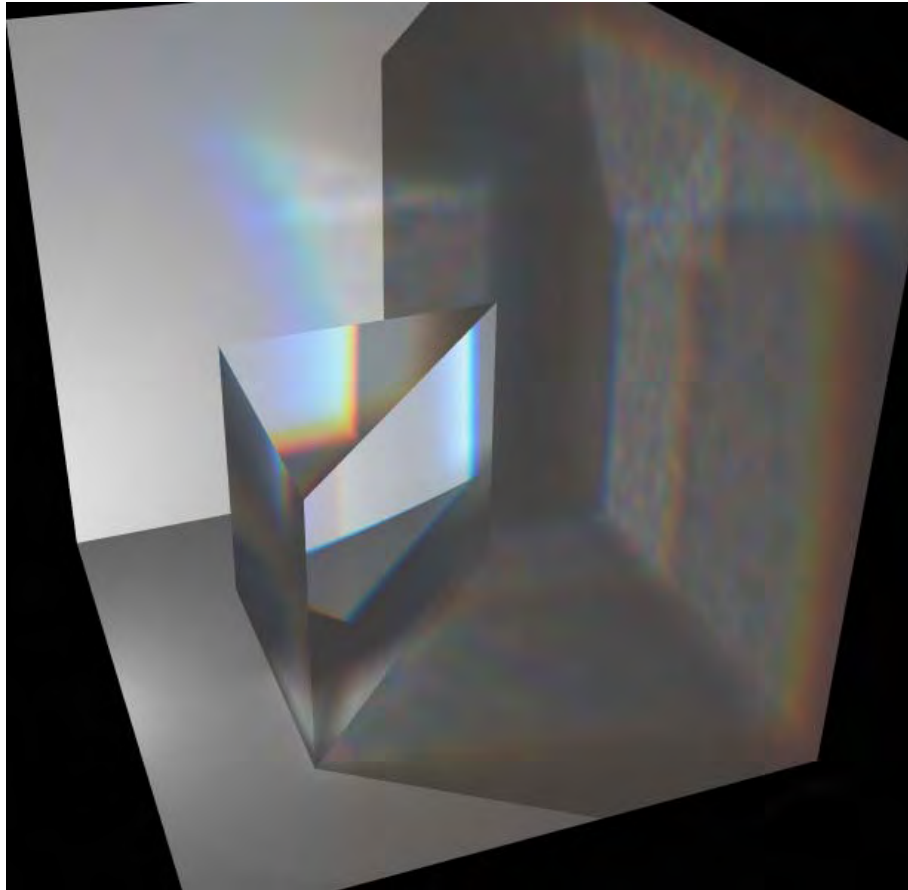
Local illumination (in a nutshell)



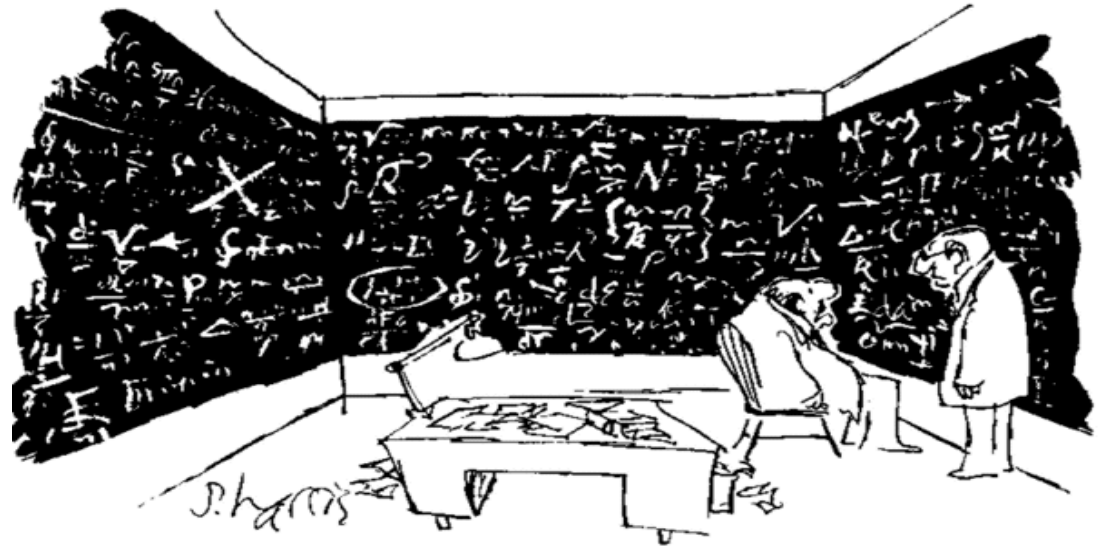
Local illumination (in a nutshell)



Local illumination (in a nutshell)



GIGA



"Whatever happened to *elegant* solutions?"

Global illumination

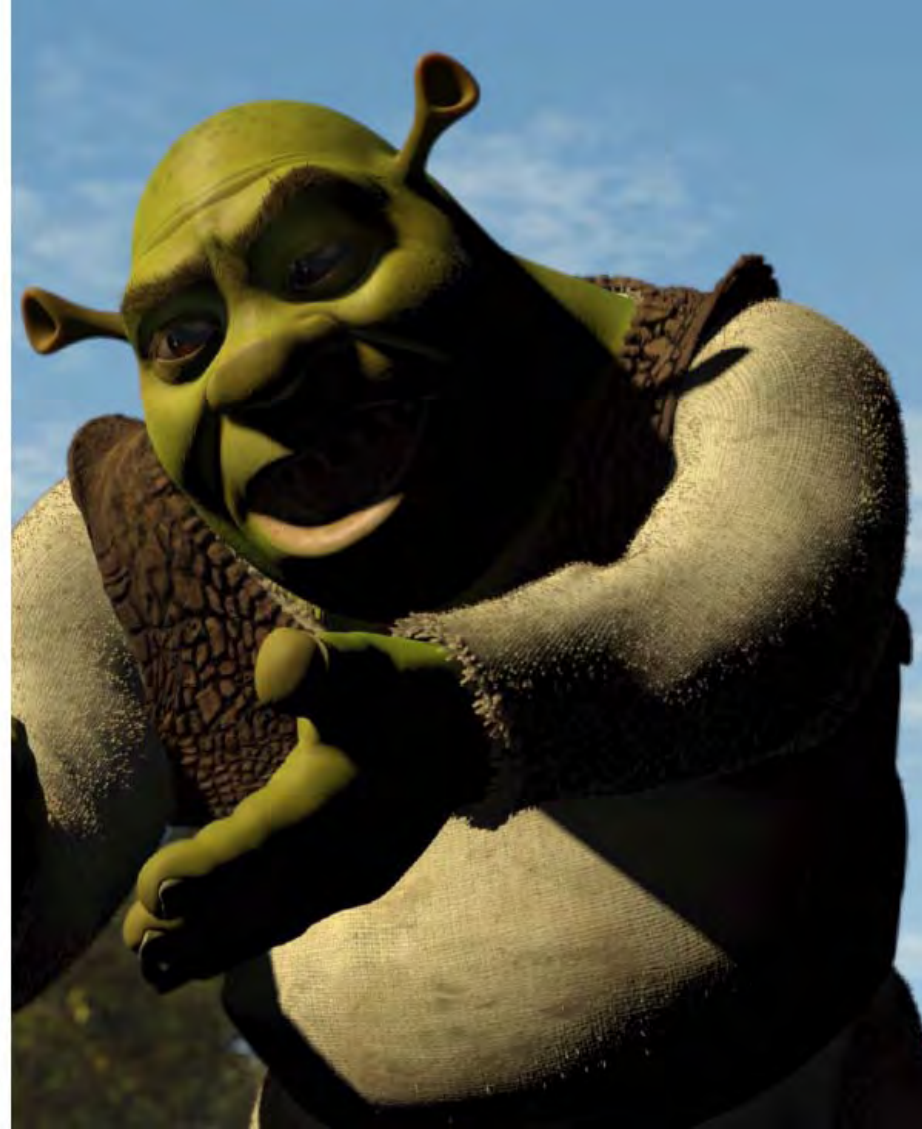
Diego Gutiérrez

Index

- ◆ What is GI
- ◆ Intro to some GI algorithms



What is GI?



What is GI?



What is GI?



What is GI?



What is GI?



What is GI?



What is GI?



What is GI?



What is GI?

- ◆ Ejemplo con photon mapping [Wann Jensen]

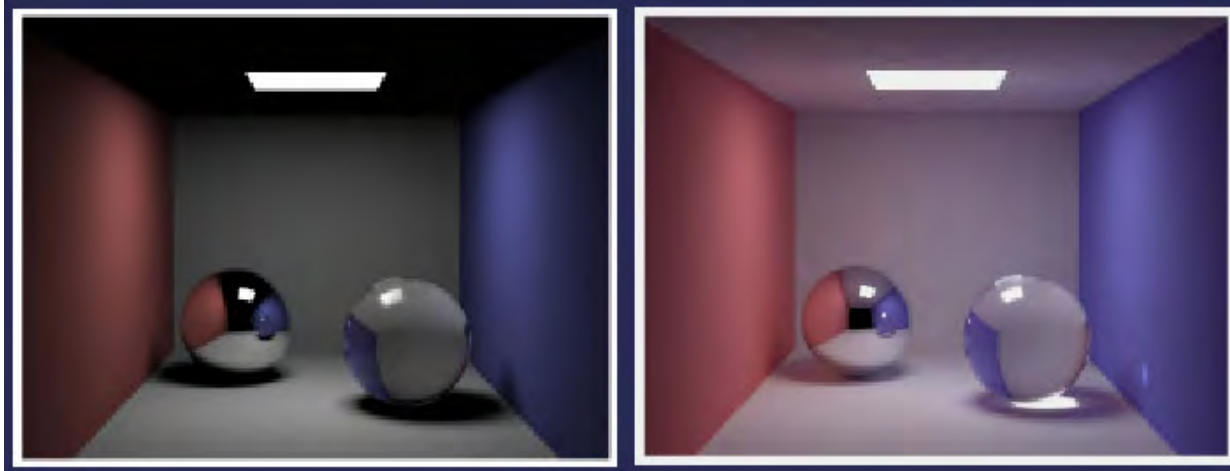




What is GI?

◆ Monte Carlo = Noise!

What is GI?



◆ Direct lighting

- Discrete light sources
- Efficient lighting calculations based on vectors and surfaces (hacks!)

◆ Indirect lighting

- Reflected or refracted light: color bleed, caustics, diffuse interreflection...
- Calculations based on physically-based light transport
- BRDF's (*Bidirectional Reflective Distribution Function*)

What is GI?

◆ Caustics



Indirect lighting

- ◆ Museum of Modern Art, San Francisco



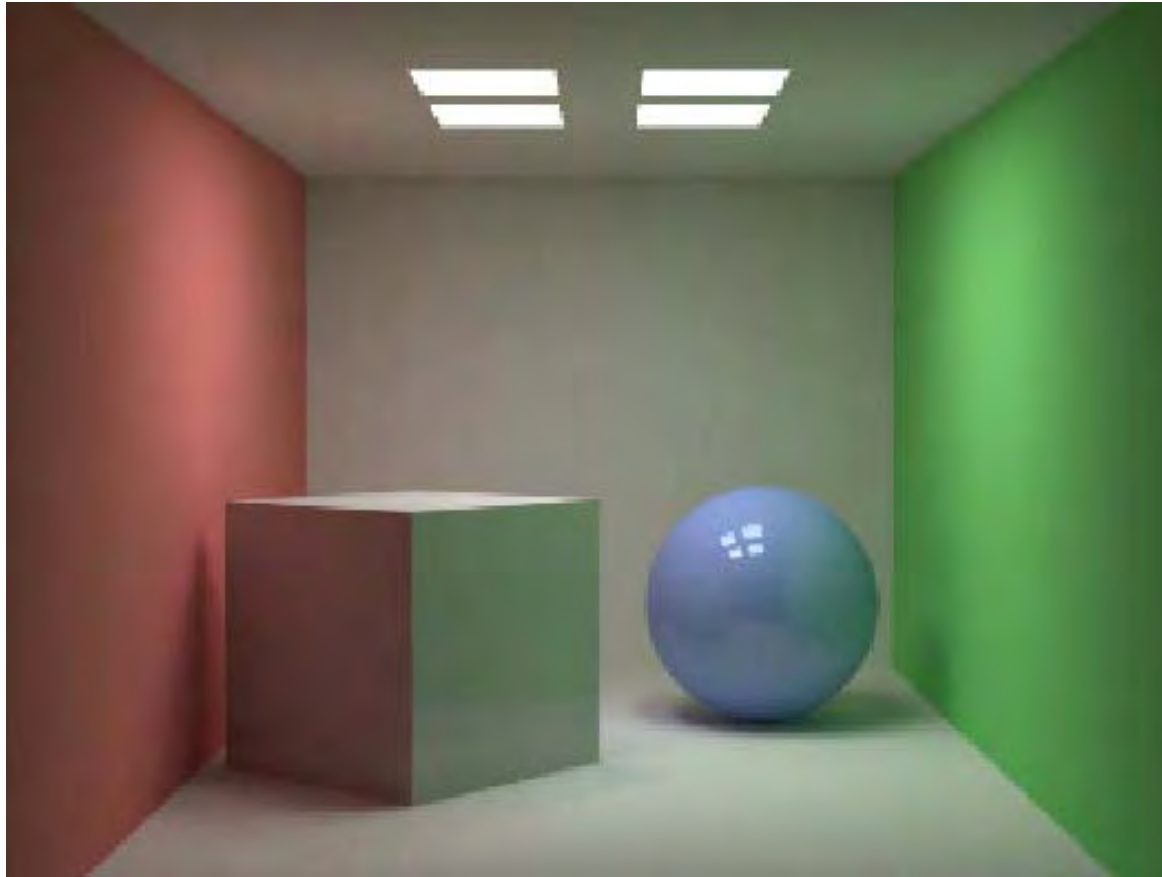
Indirect lighting

◆ *Color bleed*



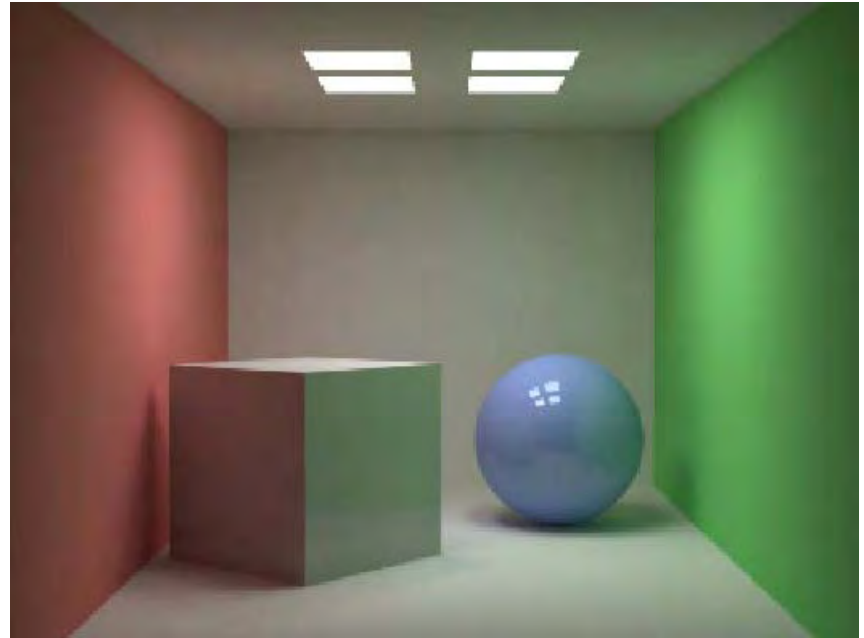
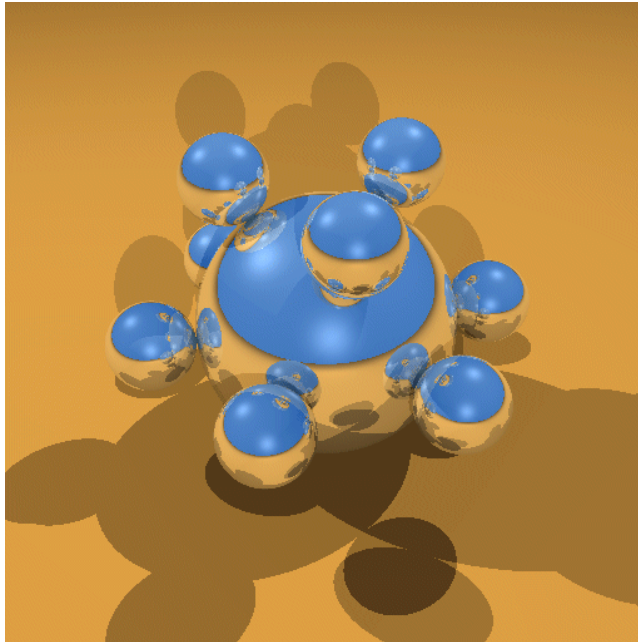
Indirect lighting

◆ *Color bleed*



Indirect lighting

- ◆ Compared with local models, there's a clear improvement...





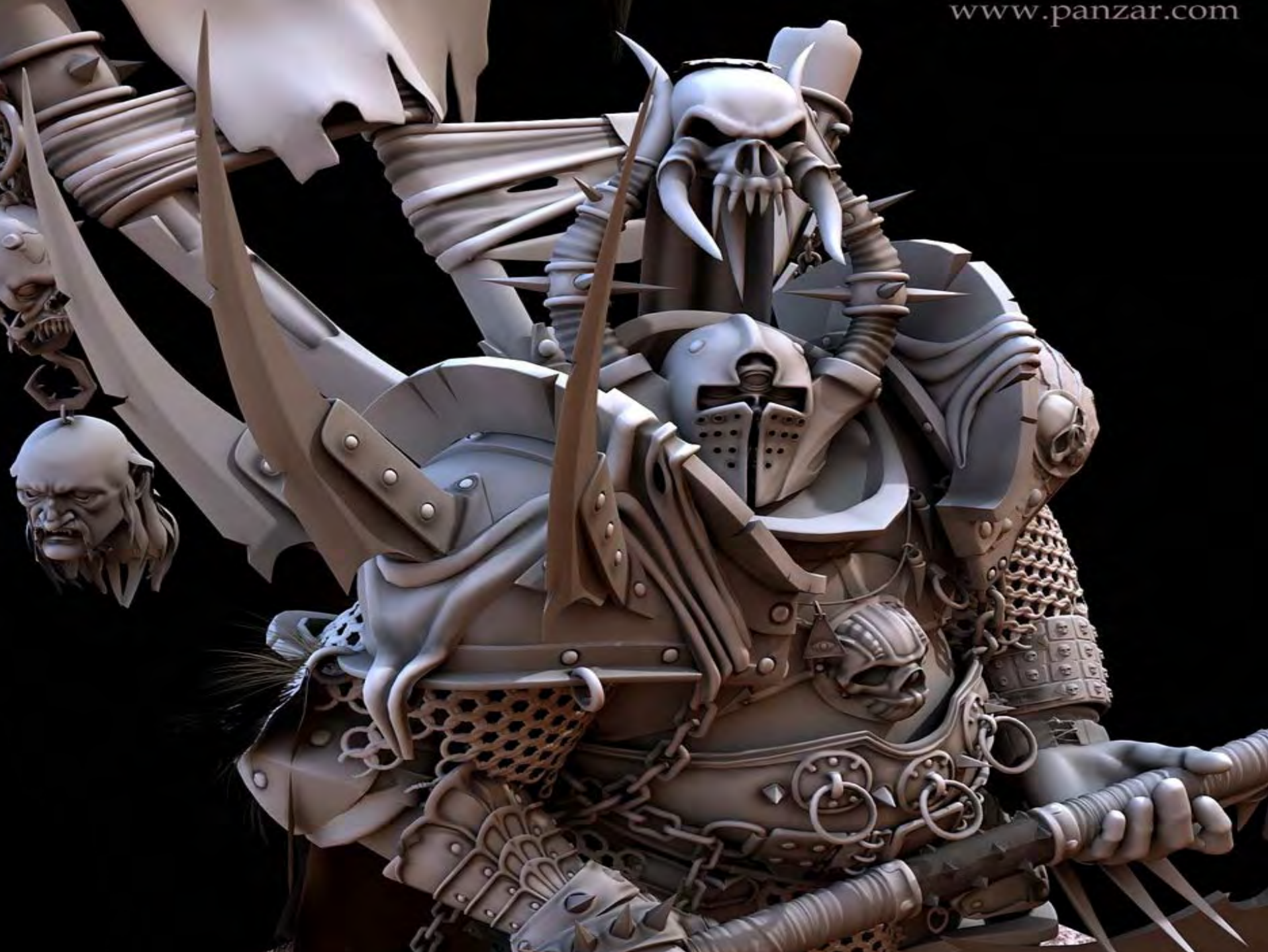




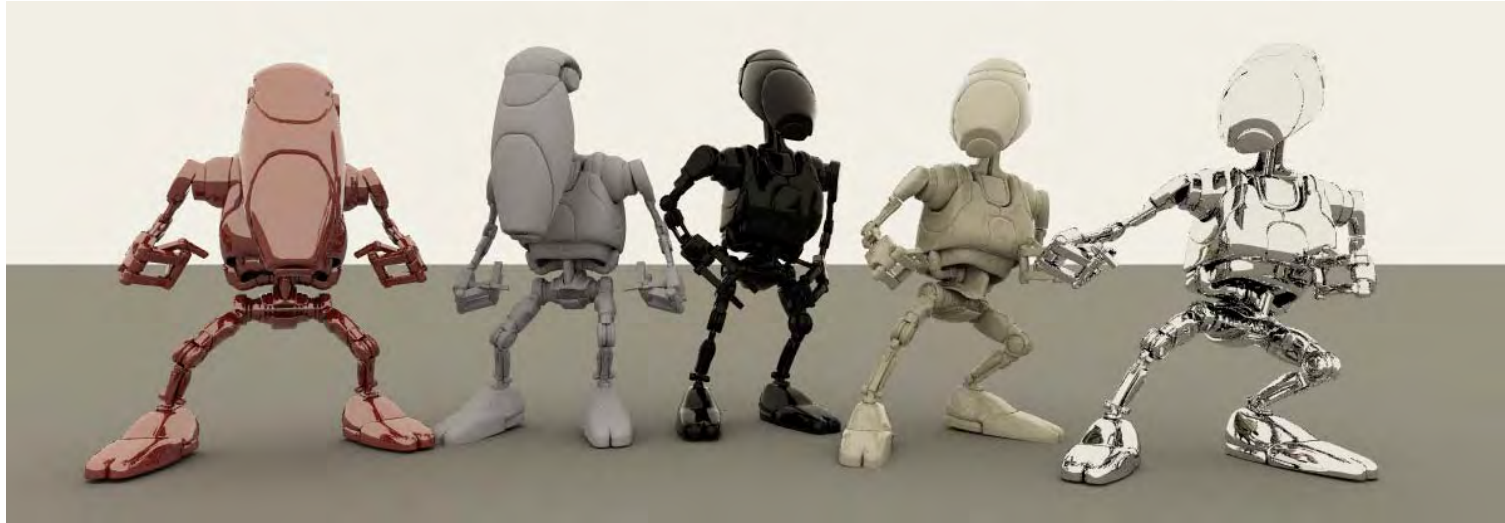
rendered in XSI
copyright umlaut 2000











Shading

Diego Gutiérrez

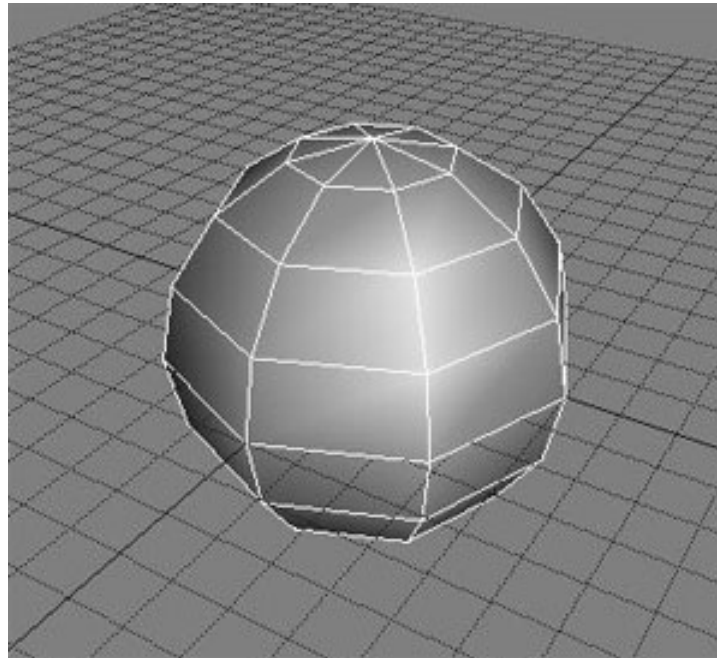
Index

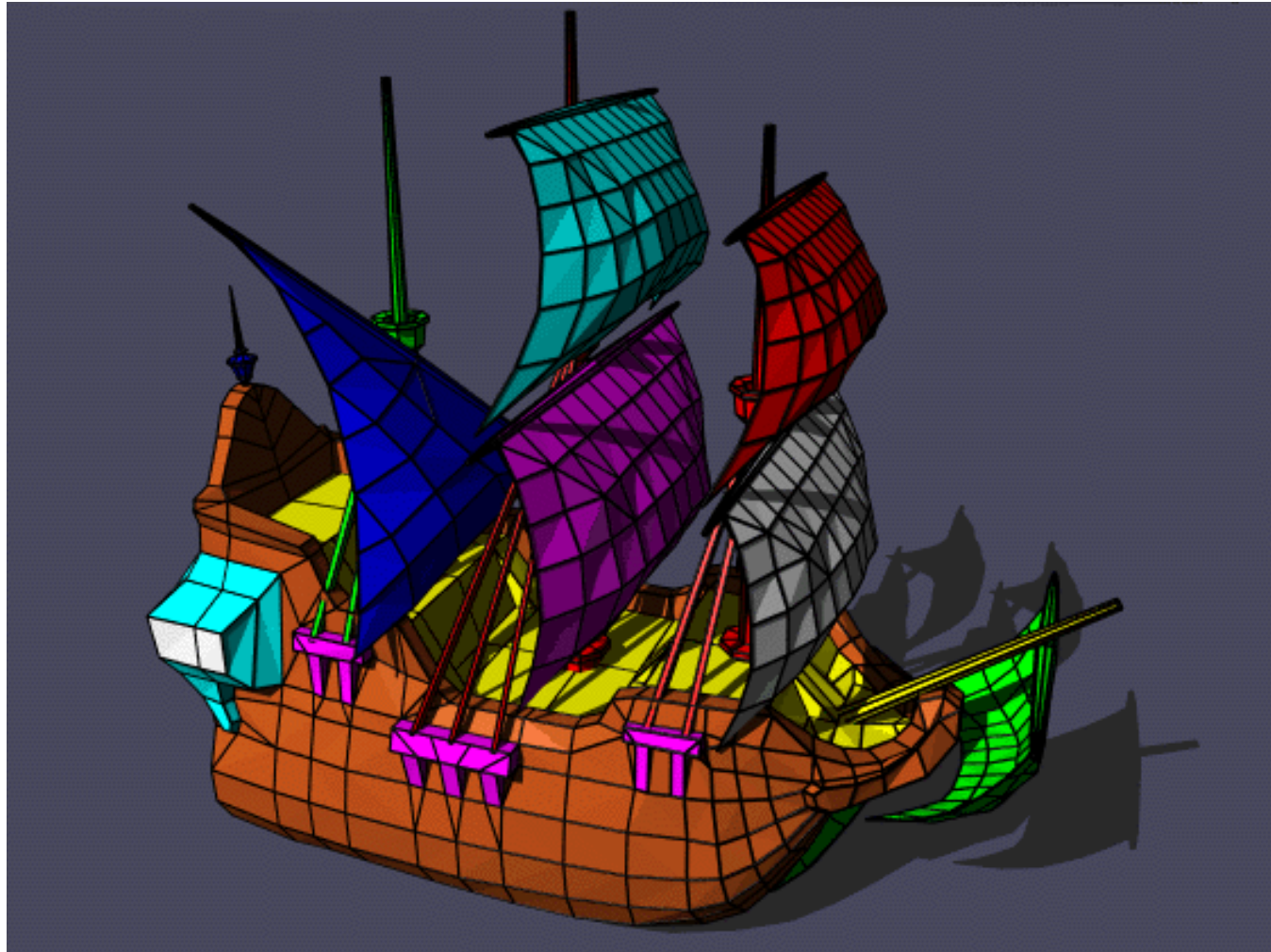
- ◆ Polygon shading
- ◆ Flat shading
- ◆ Gouraud shading
- ◆ Phong shading

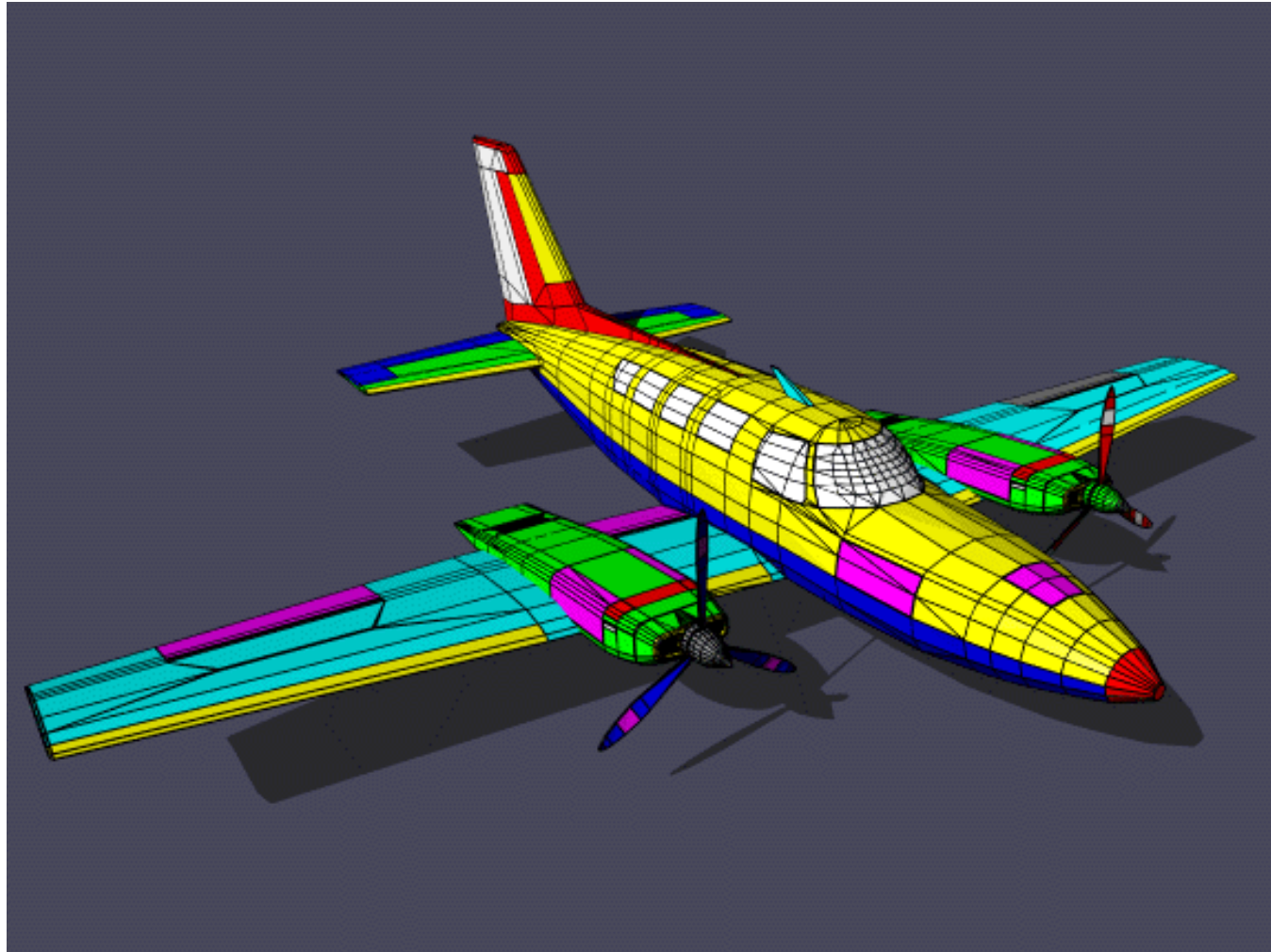


Why polygons?

- ◆ Most renderers work on polygons
- ◆ Most objects are represented as polygons
- ◆ Most algorithms are thought for polygons
- ◆ Graphics hardware works on polygons







Polygon shading

- ◆ Lighting models seen so far compute light for a given *point* on the surface
- ◆ How to obtain it for the whole polygon?
- ◆ Possibilities:
 - Repeat the same calculations for each point (unnecessary, maybe even stupid, inefficient)
 - *Flat shading*
 - *Gouraud y Phong*: vertices and interpolation

Flat shading

- ◆ Fastest, simplest
- ◆ Least realistic
- ◆ Problem: centroid only valid for convex polygons



$$centroid = \frac{1}{vertices} \sum_{i=1}^{vertices} \bar{P}_i$$

Flat shading

- ◆ Most times, light direction is NOT constant for the whole polygon
- ◆ For specular highlights, neither is the eye direction
- ◆ Polygonal inner structure too obvious!
- ◆ Subdivide?



Mach banding

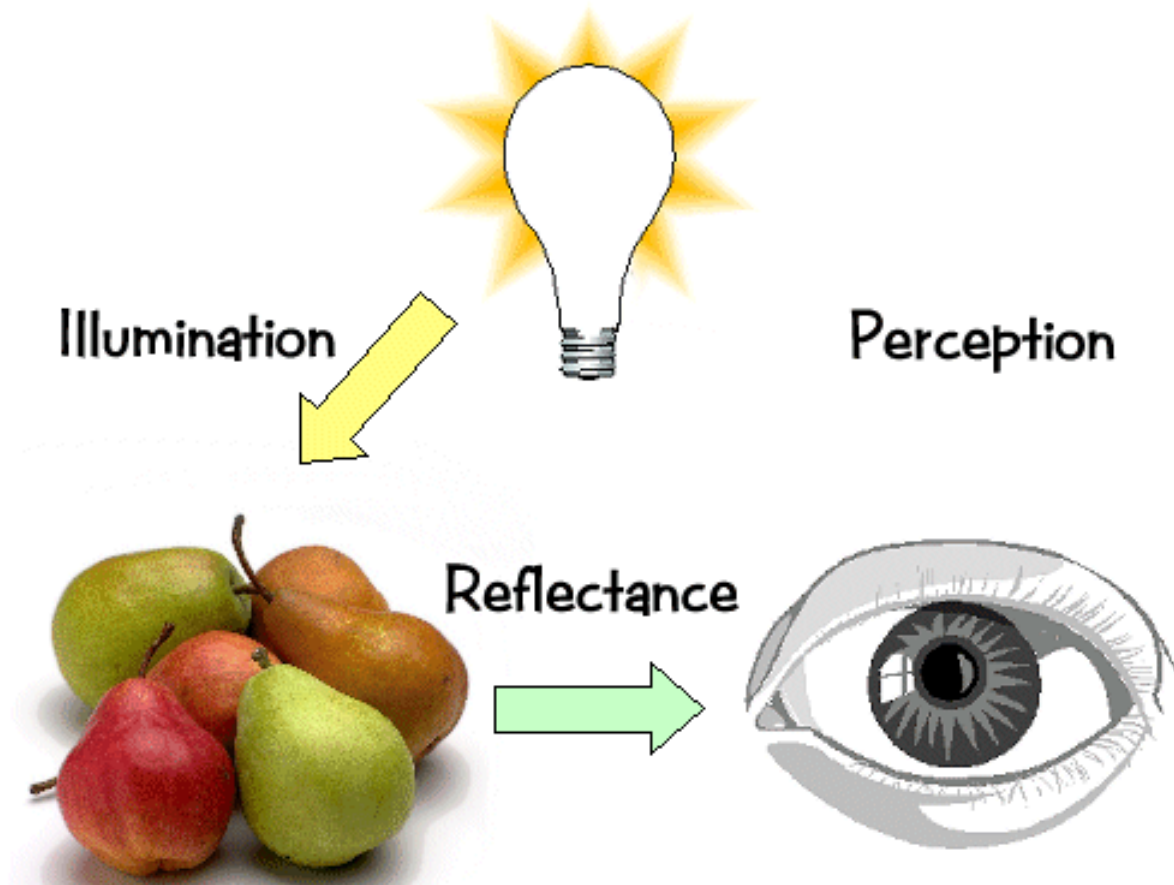
- ◆ Además, aparece el problema del *Mach banding*
 - HVS is very good at detecting edges... Even if these do not exist!



Mach banding

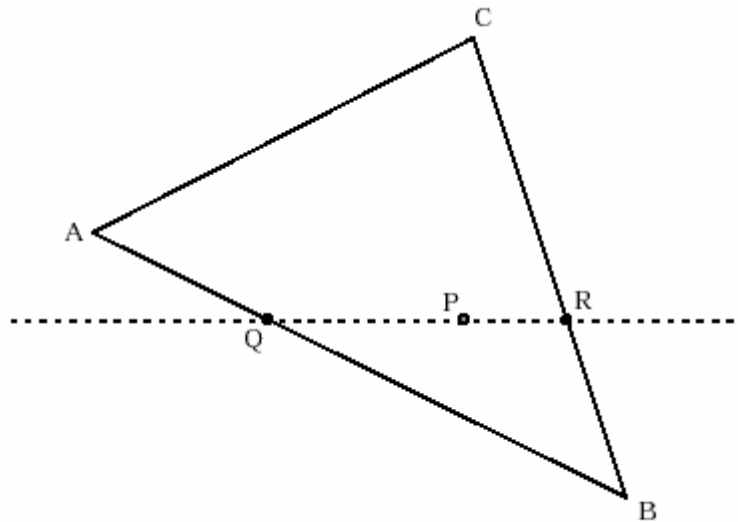


Mach banding



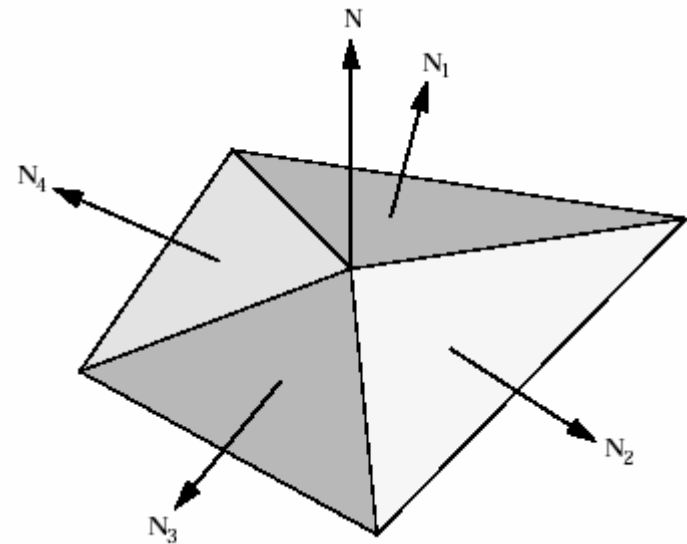
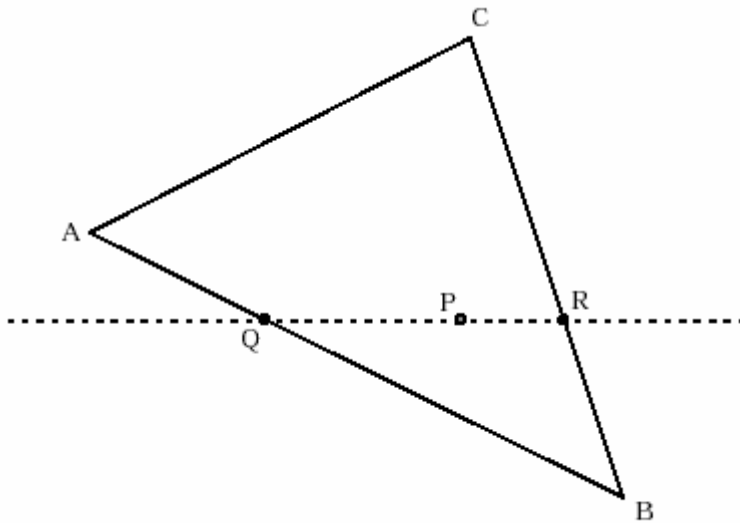
Interpolation

- ◆ Designed to avoid banding effects
- ◆ Problem is: given a polygon and a *scanline*, find out the intensity at any given point P inside the polygon



Gouraud Shading

- ◆ *Gouraud shading*: vertex interpolation
- ◆ Normal at a vertex: interpolation of normals for the polygons

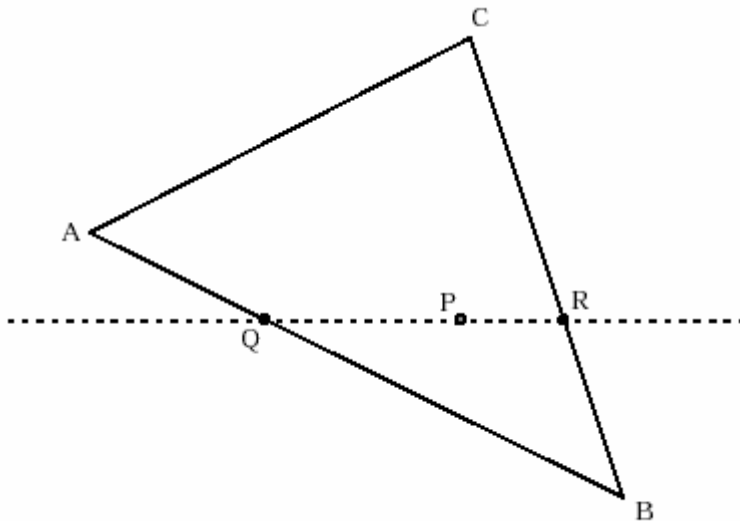


Gouraud

◆ $I = f(A, B, C)$

◆ $I = f(Q, R)$

◆ $I = f(P)$



$$I_Q = uI_B + (1 - u)I_A \quad \text{where } u = \frac{AQ}{AB}$$

$$I_R = wI_B + (1 - w)I_C \quad \text{where } w = \frac{CR}{CB}$$

$$I_P = vI_R + (1 - v)I_Q \quad \text{where } v = \frac{QP}{QR}$$

Gouraud

- ◆ We get rid of discontinuities: intensity at the end of one polygon is forced to be equal to the intensity at the beginning of the next





Gouraud

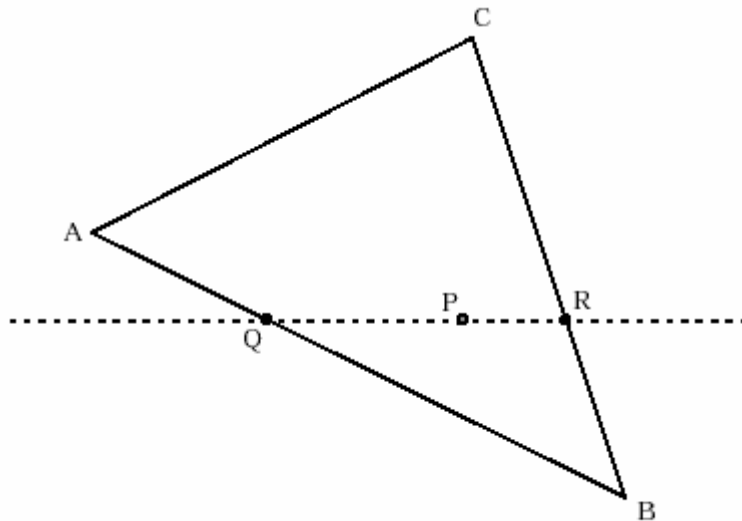


Problems:

- Edges tend to disappear
- Specular reflections lost sometimes

Phong Shading

- ◆ Not the same as Phong lighting model
- ◆ Same concept as Gouraud, but we interpolate the normals, not the intensities



$$\mathbf{n}_Q = u\mathbf{n}_B + (1 - u)\mathbf{n}_A$$

$$\mathbf{n}_R = w\mathbf{n}_B + (1 - w)\mathbf{n}_C$$

$$\mathbf{n}_P = v\mathbf{n}_R + (1 - v)\mathbf{n}_Q$$

Phong

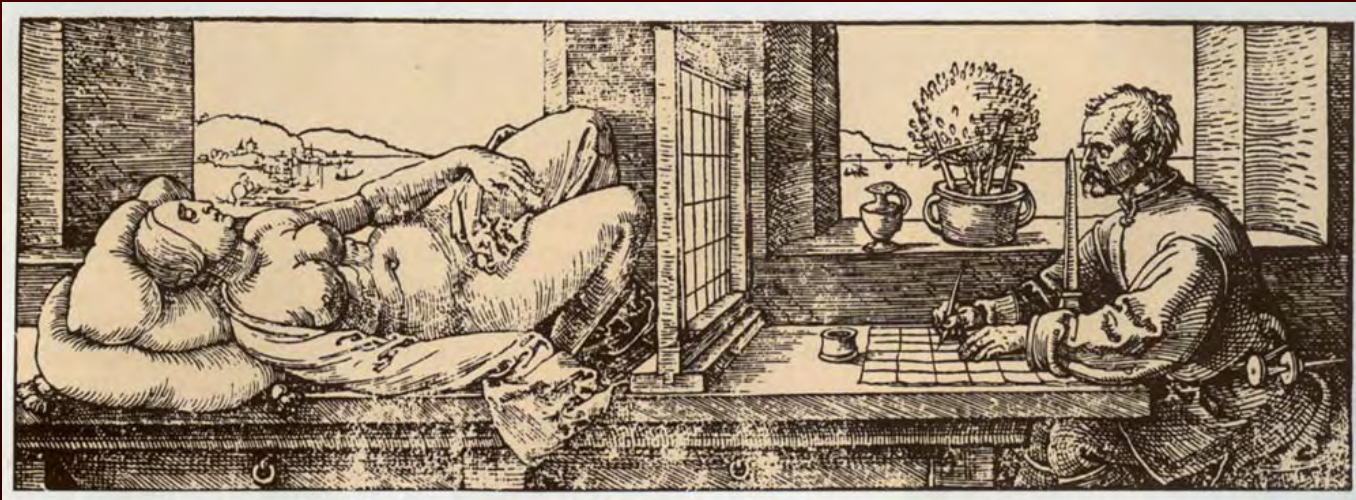
- ◆ Better: more expensive



Shading (in a nutshell)



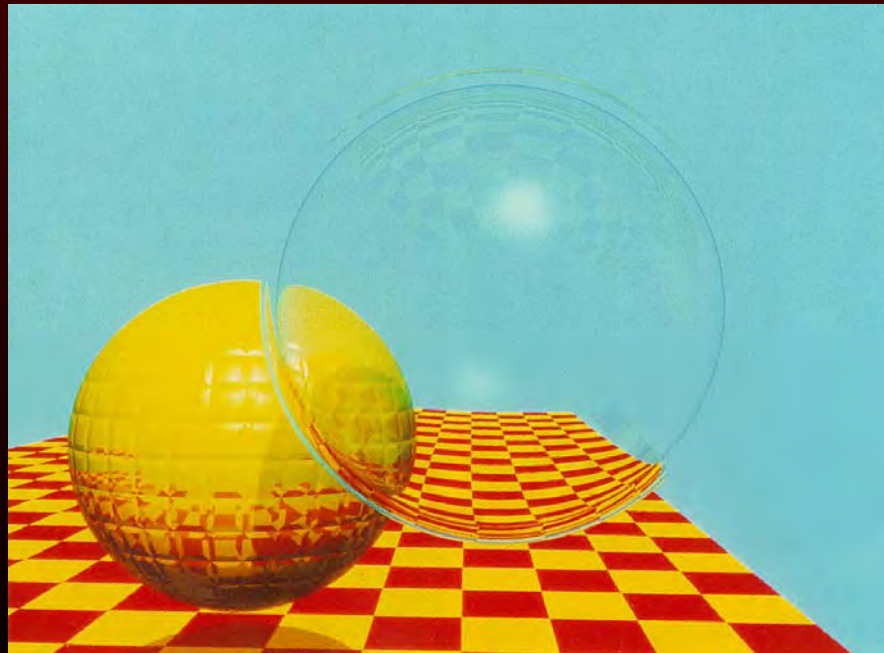
Ray Tracing I



Andrew Zaferakis and Diego Gutiérrez
University of North Carolina at Chapel Hill
Universidad de Zaragoza

Ray Tracing

- Ray tracing for graphics invented by Turner Whitted, while at Bell Labs, in his 1980 paper “*An Improved Illumination Model for Shaded Display*”.
- The idea of tracing photons from light sources has been around much longer.

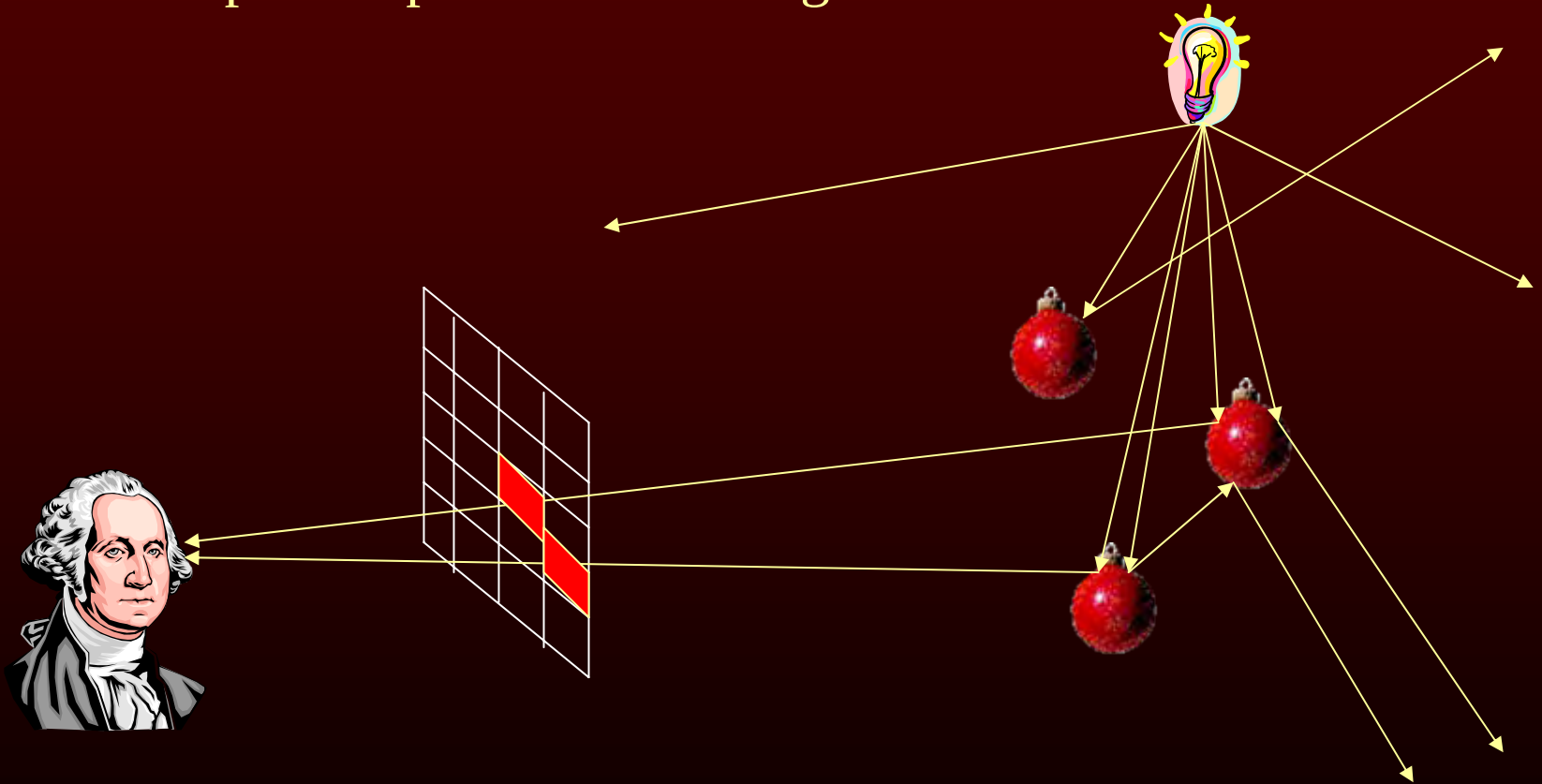


Overview

- Forward vs. Backward Ray Tracing
- A Ray Classification Scheme
- Geometric Representations
- Intersections
- Ray Propagation and Ray Trees
- Lighting
- Shadows
- Intersection Acceleration Techniques

Forward Ray Tracing

- Trace all photon paths from the light source.



Forward Ray Tracing

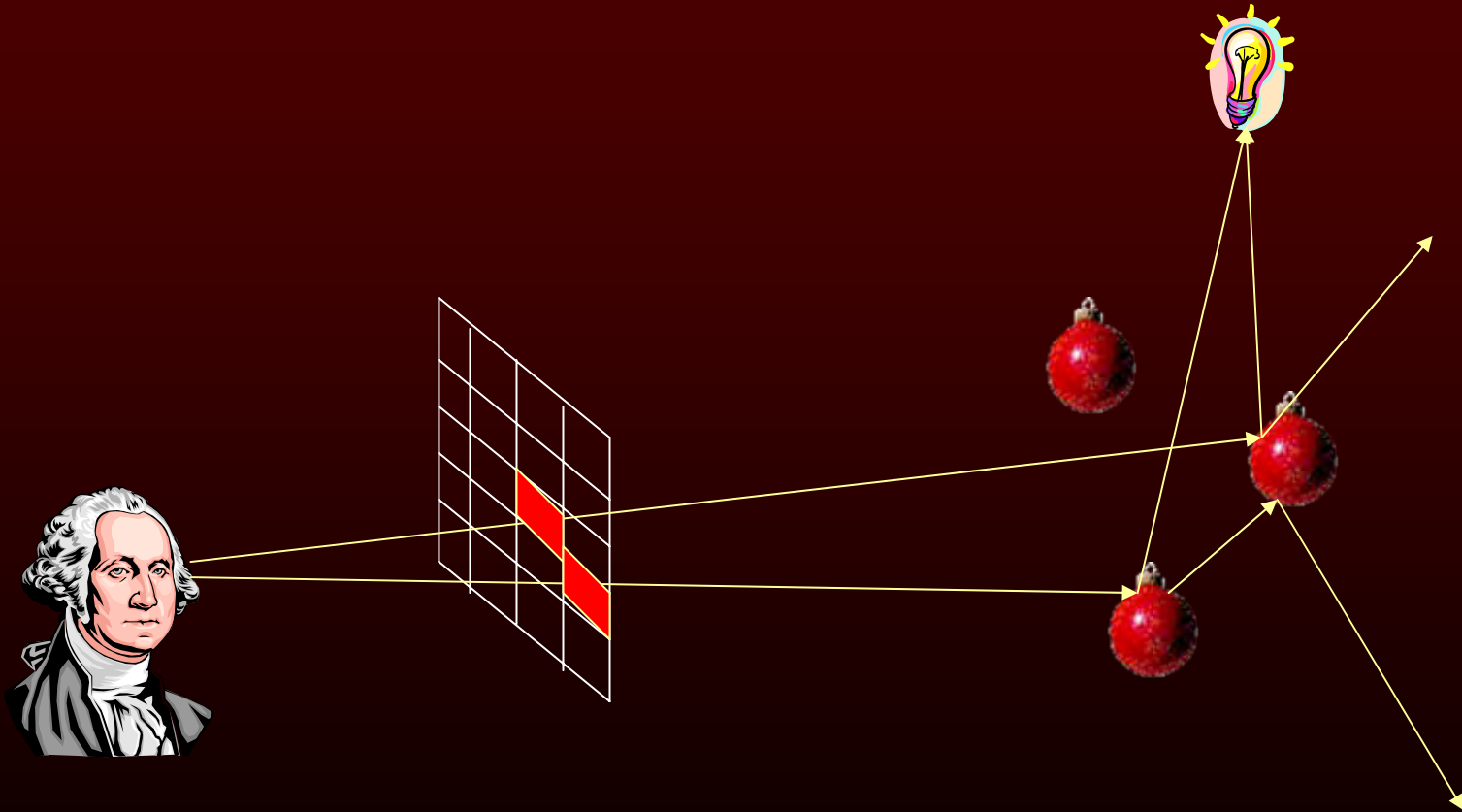
- Trace all the photons paths from the light sources through the scene.
- As a ray travels through the scene it gets dimmer, each interaction causes energy loss.
- Rays change color and direction with each interaction.
- Only photons that travel through the image plane into the eye contribute to the image color.
- Rays must be averaged when multiple rays intersect the same pixel on the image plane.

Forward Ray Tracing

- What is wrong with forward ray tracing?

Backward Ray Tracing

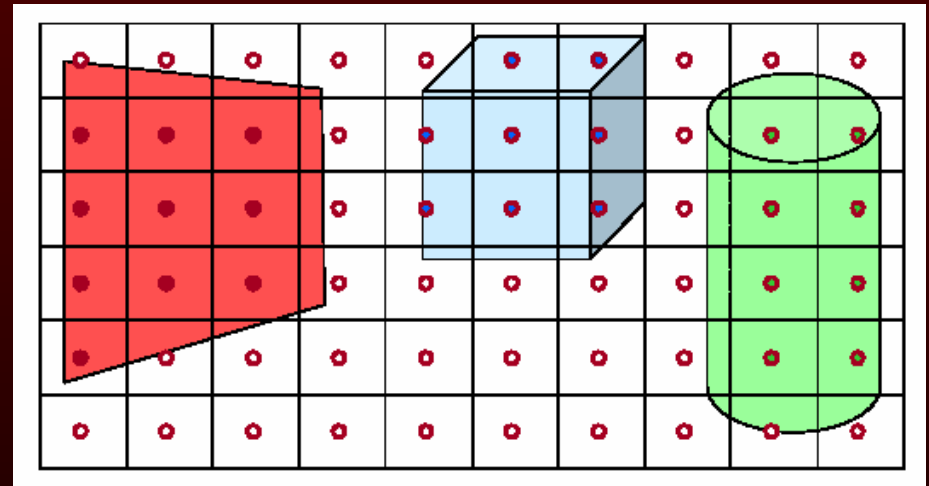
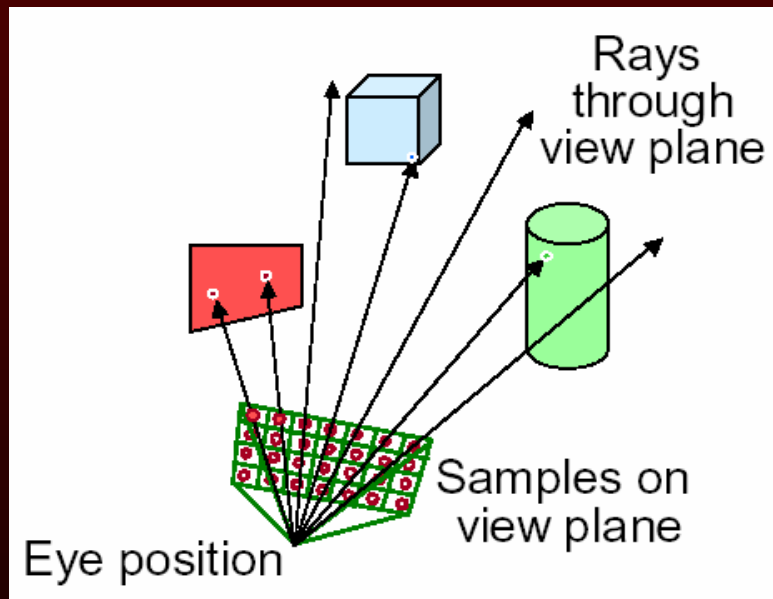
Trace rays from the eye through image plane

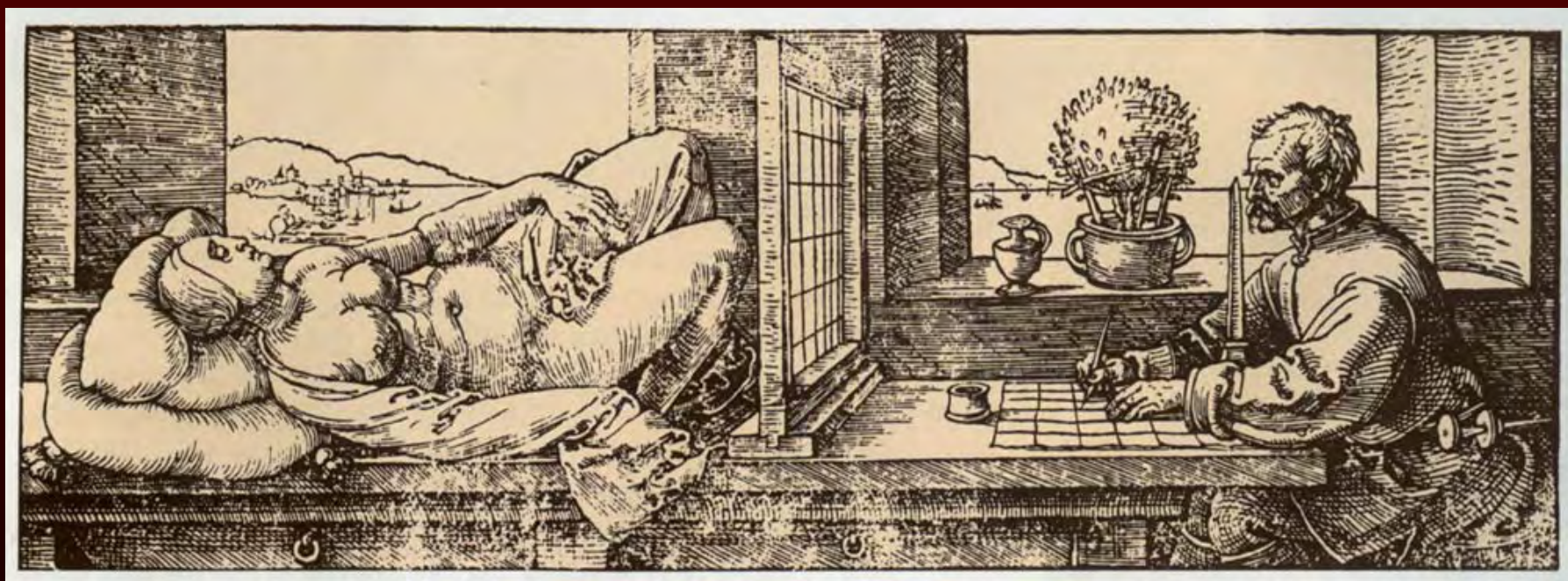


Backward Ray Tracing

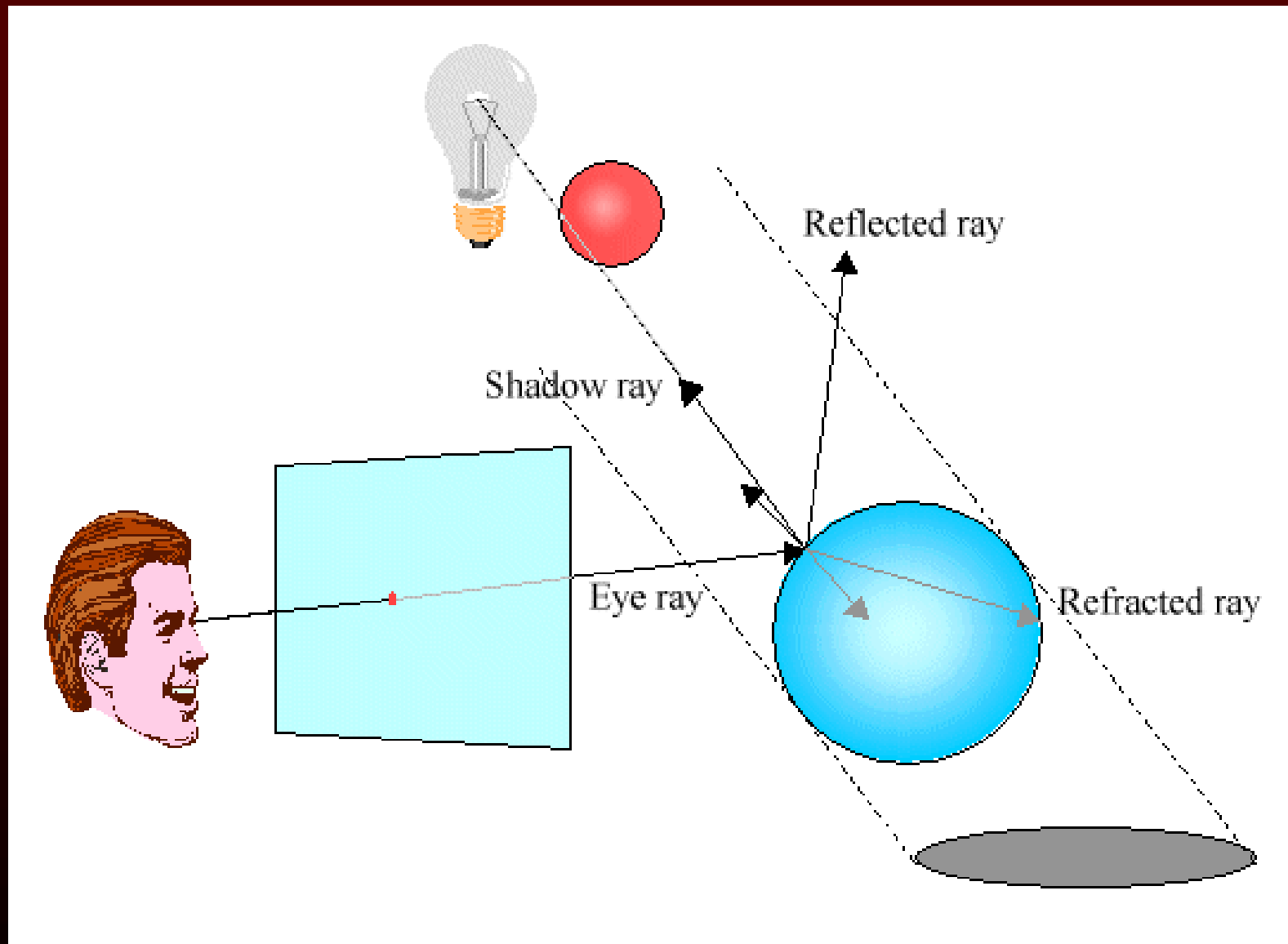
- Reverse the direction of the rays.
- Traverse the rays from the eye into the scene.
- Each ray is guaranteed to contribute to the image.
- Much faster!
- Must have different classifications of rays.

Ray casting





Ray tracing

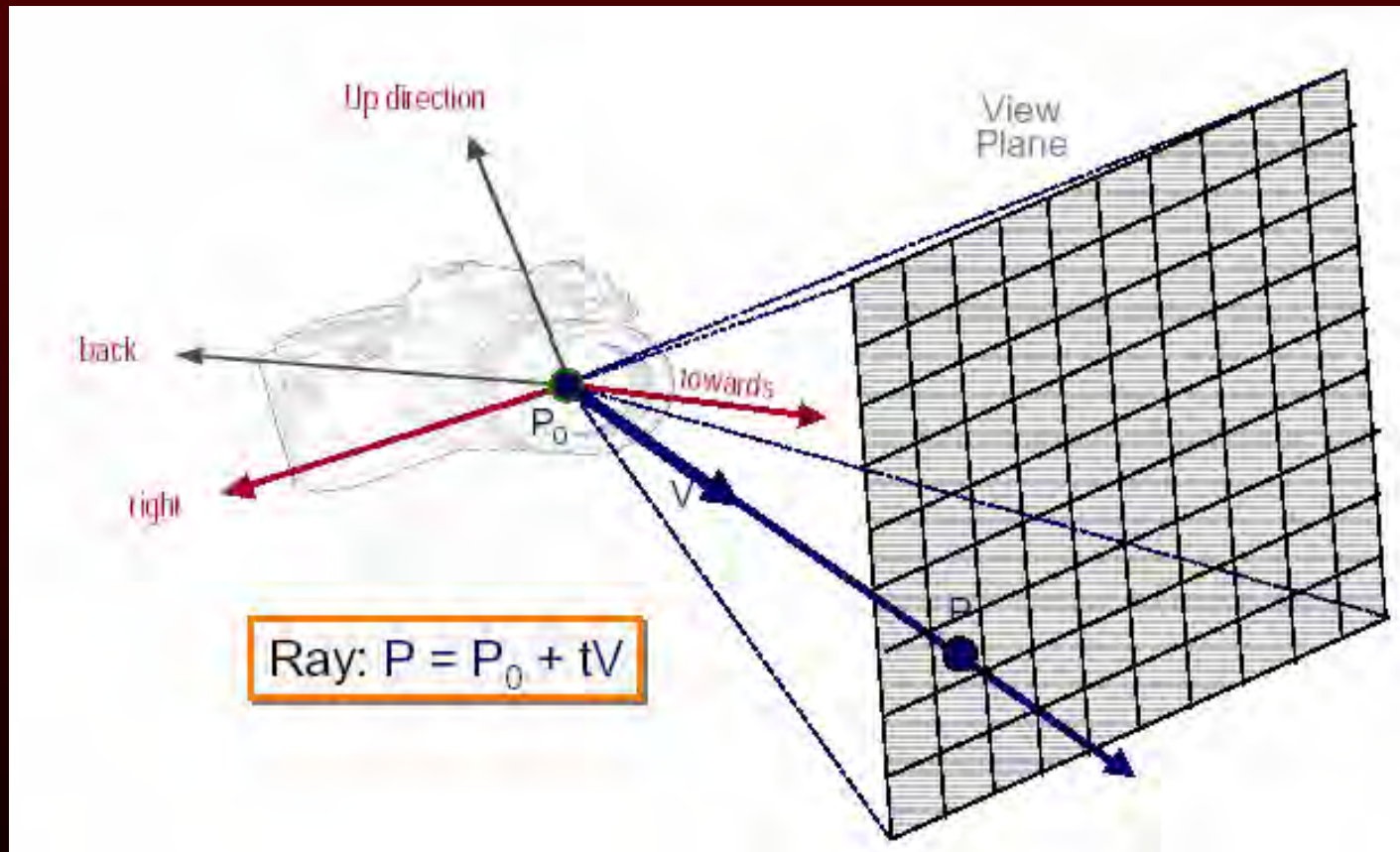


Ray Classification

- Eye Rays – Carry light to the eye
- Illumination / Shadow Rays – Carry light from a light source to an object's surface.
- Reflection Rays – Light reflected by an object.
- Transparency Rays – Light passing through an object.
- All rays are mathematically the *same*!

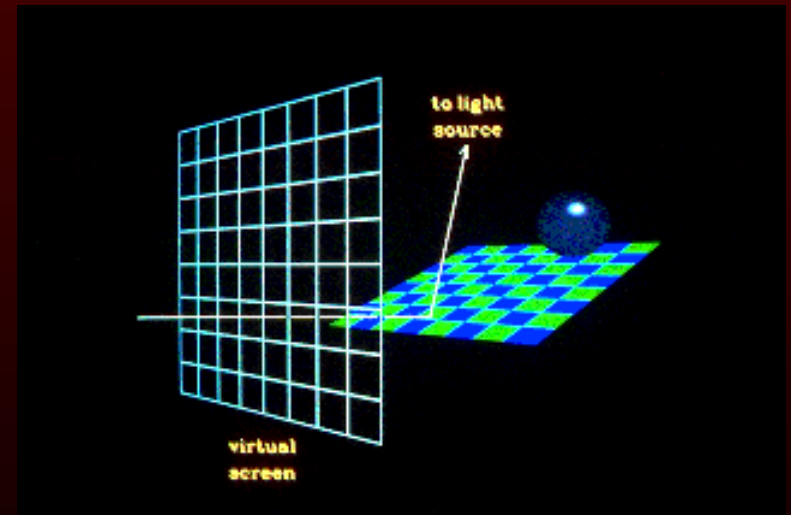
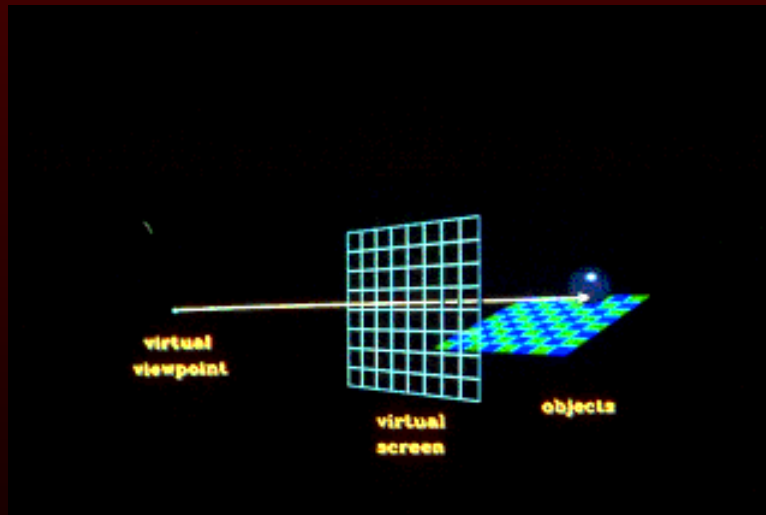
Procedure (I)

- A primary ray is shot through each pixel



Procedimiento (II)

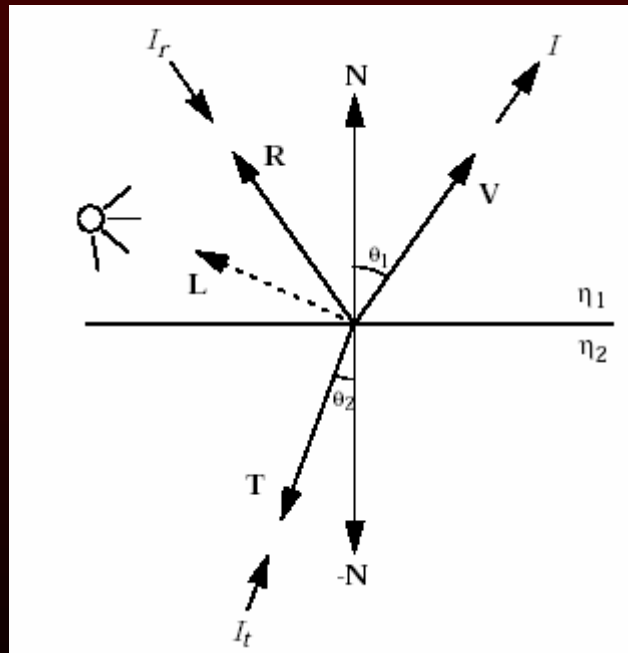
- Find the intersection with the closest object



Procedimiento (III)

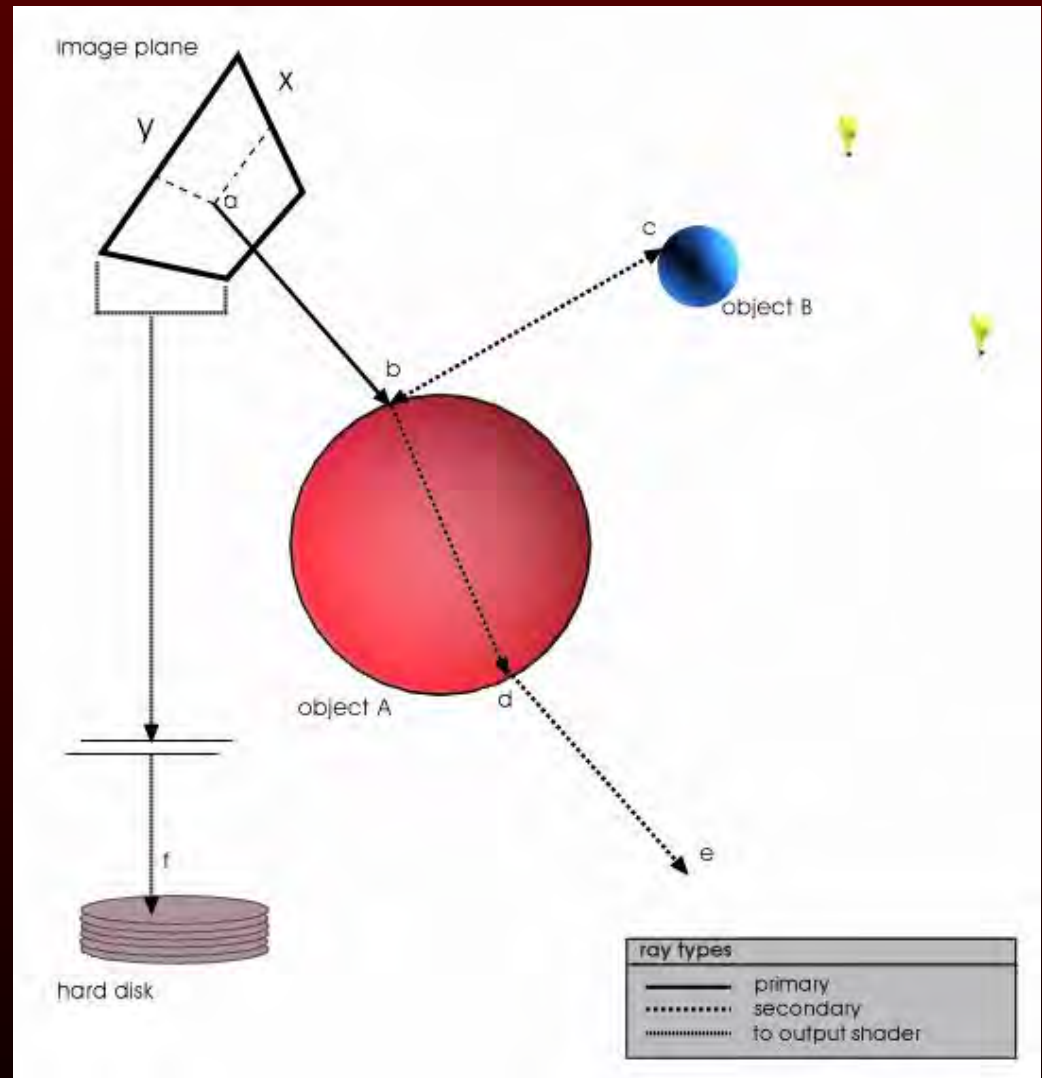
- Obtain light intensity at that point
 - Local + global components

$$I = k_d I_a + k_d \sum_{i=1}^m I_{p_i} \left[(\mathbf{N} \cdot \mathbf{L}_i) + k_s (\mathbf{R} \cdot \mathbf{V})^n \right] + k_r I_r + k_t I_t$$



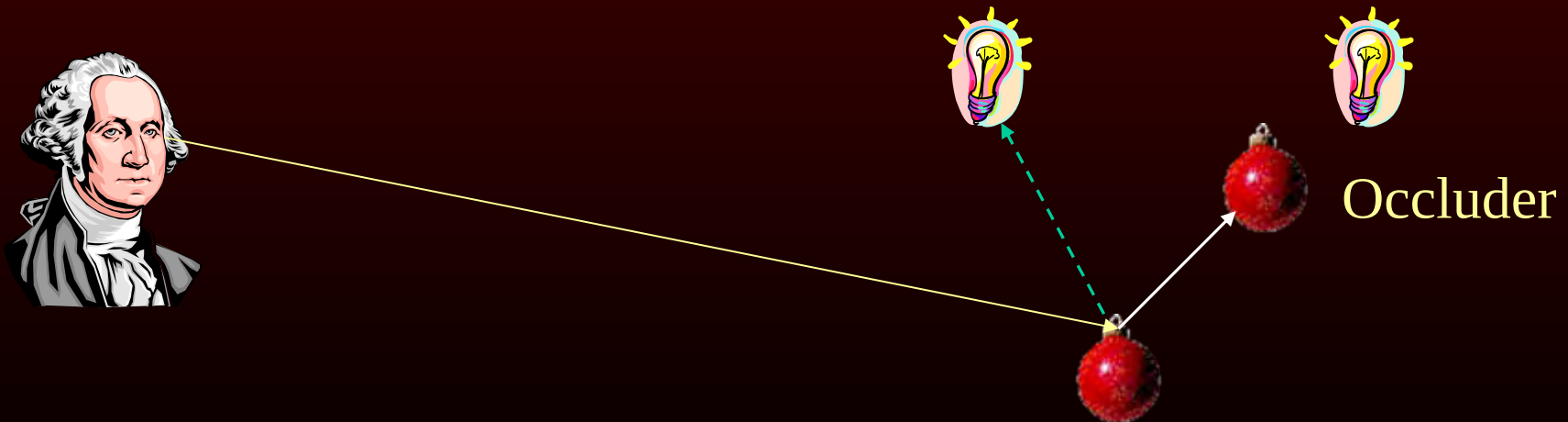
Procedimiento (IV)

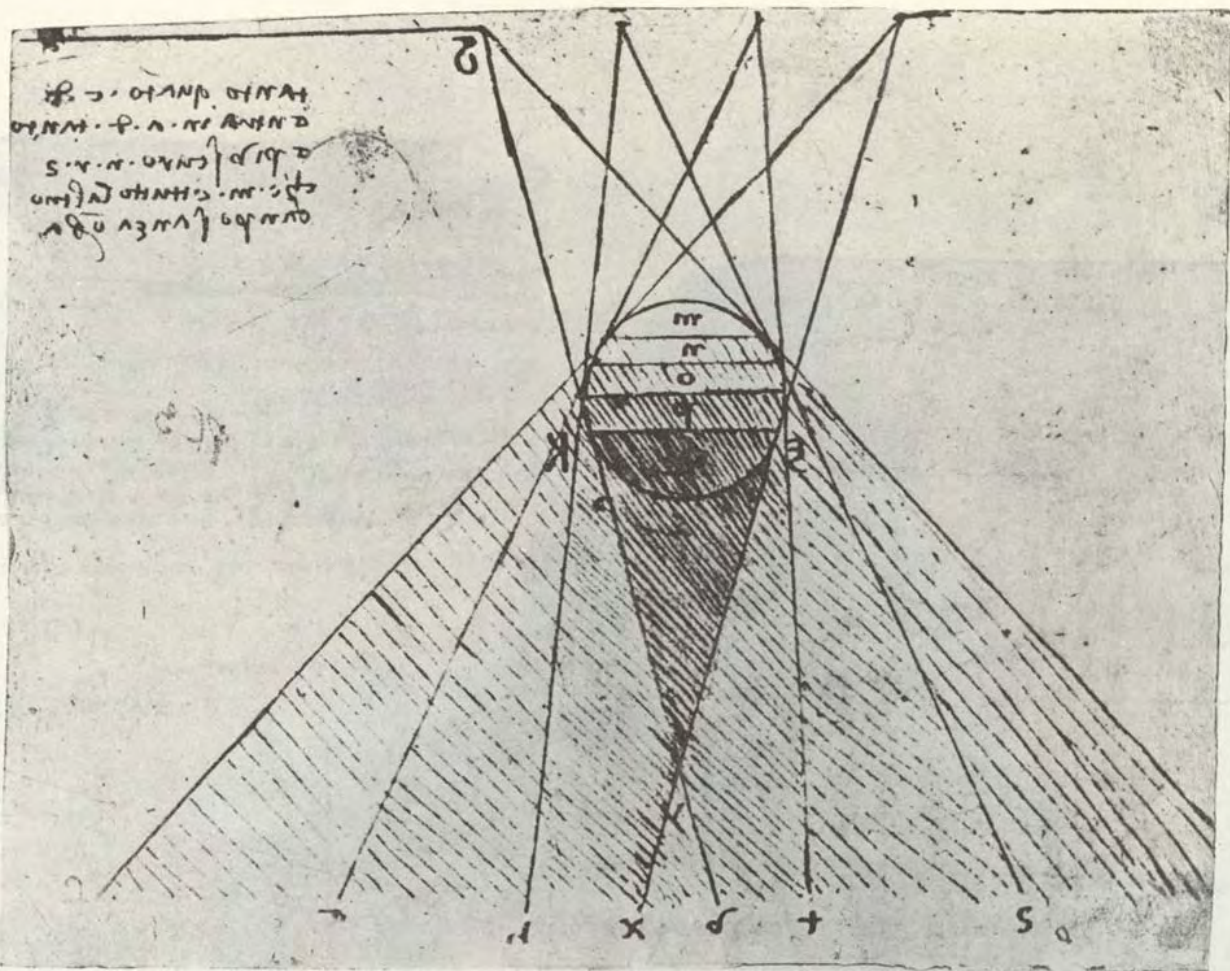
- Global components by shooting secondary rays



Illumination / Shadow Rays

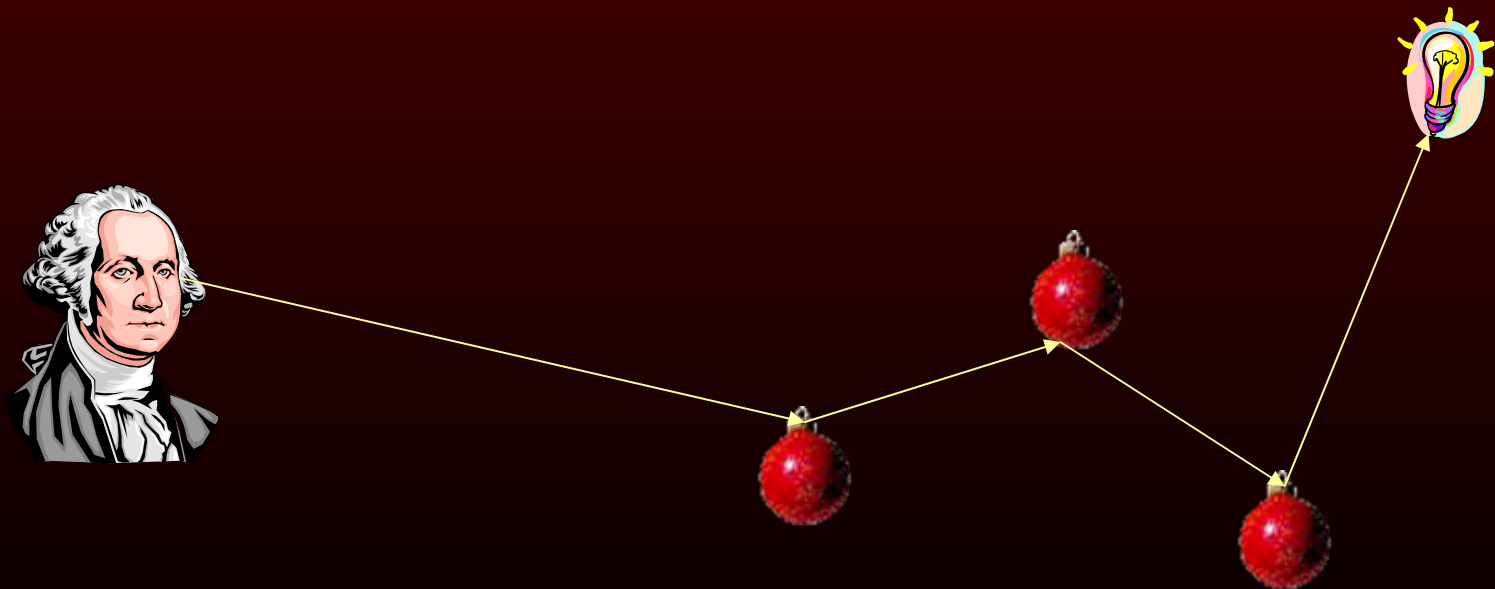
- A ray is cast from an object's surface towards a light.
- The shadow ray tests if the light is occluded
- If the light is not occluded, then the light contributes to the object's surface color.
- If the light is occluded, then the light does not contribute to the object's surface color.





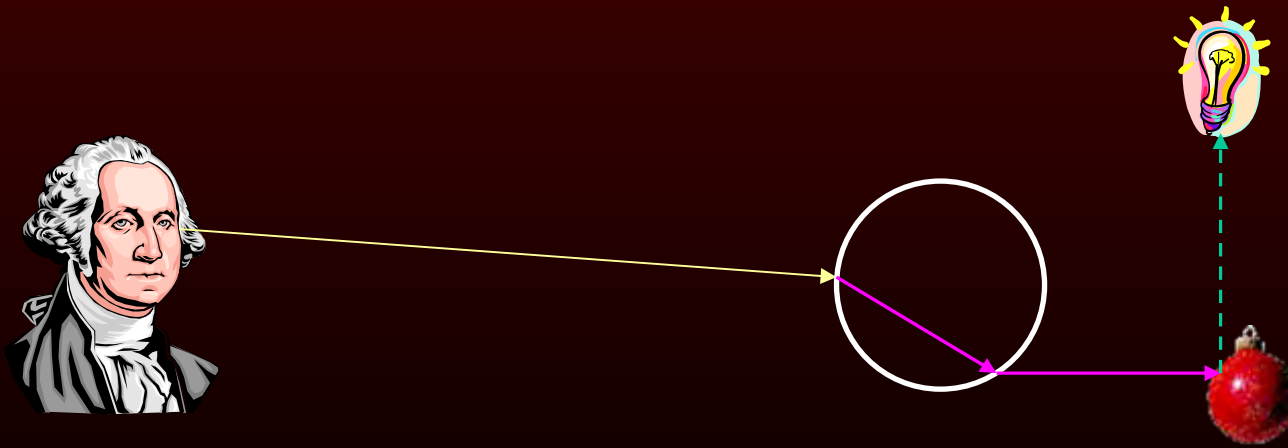
Reflected Rays

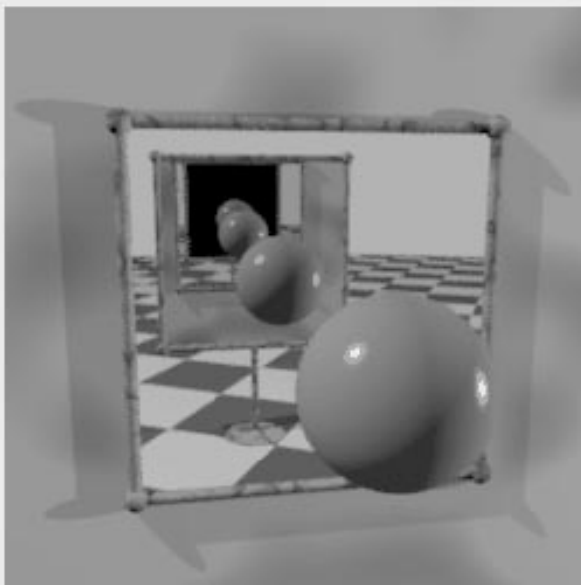
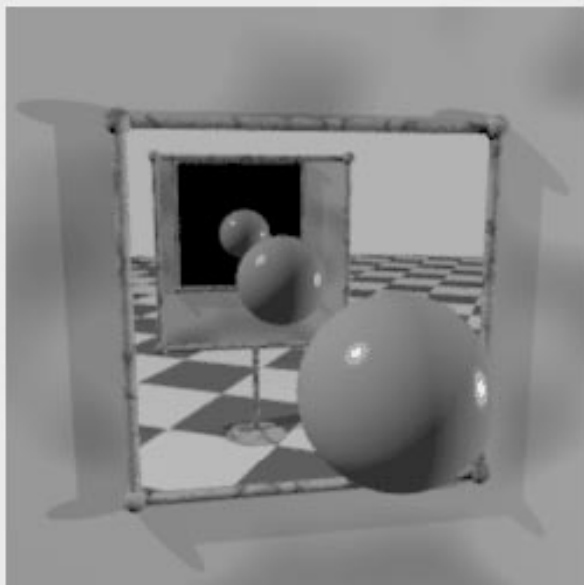
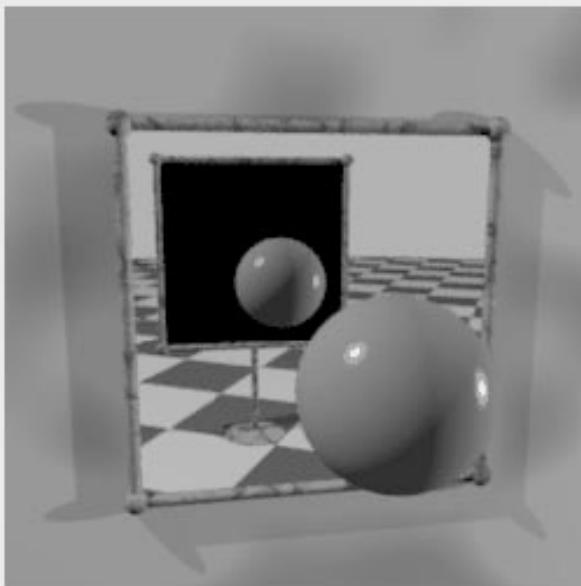
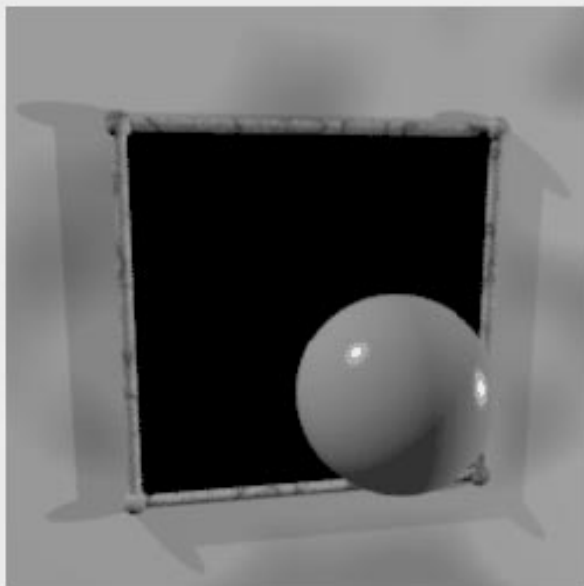
- A ray is cast from the surface of an object based on its material properties.
- The contribution results in the specular reflection.

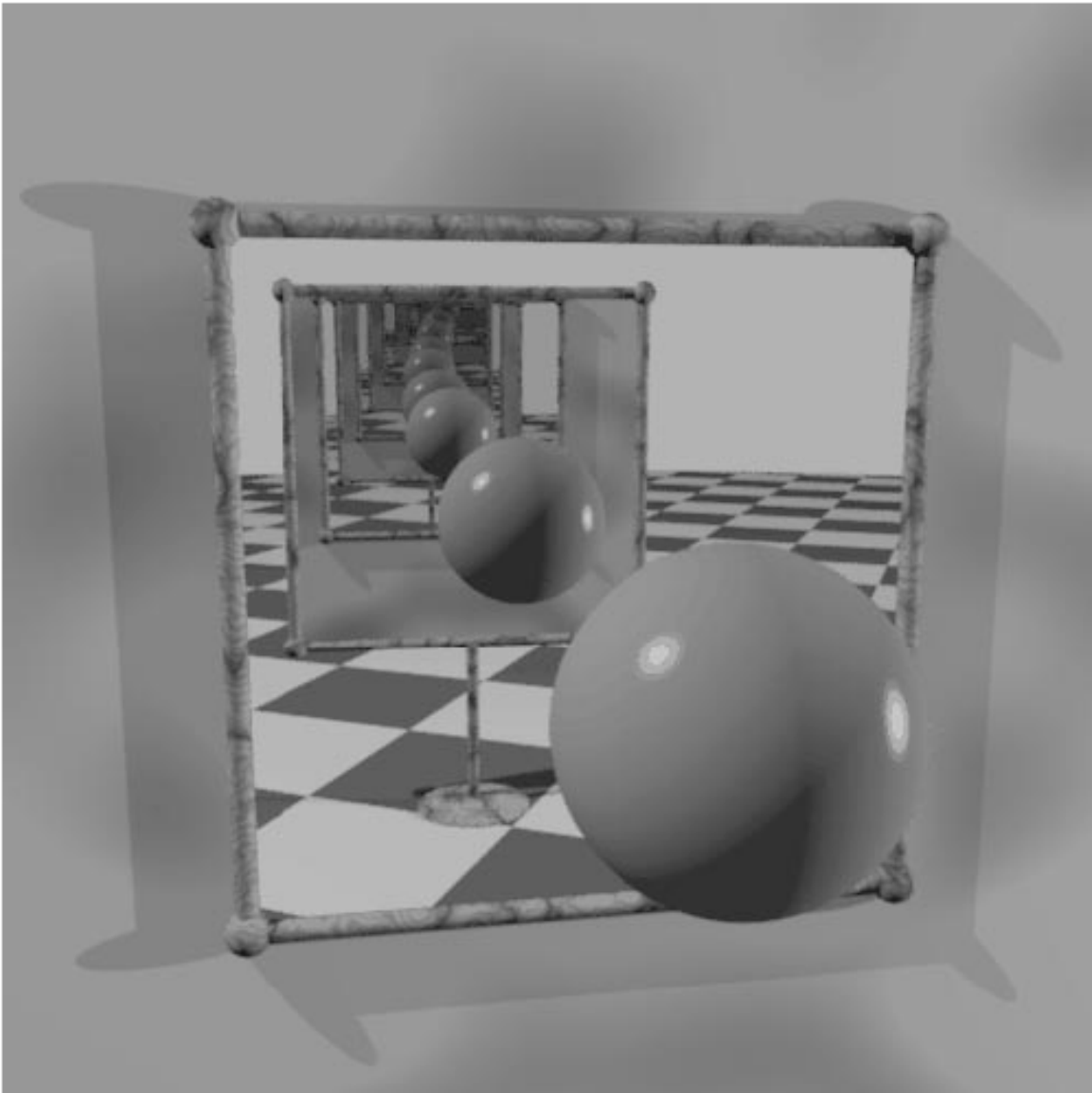


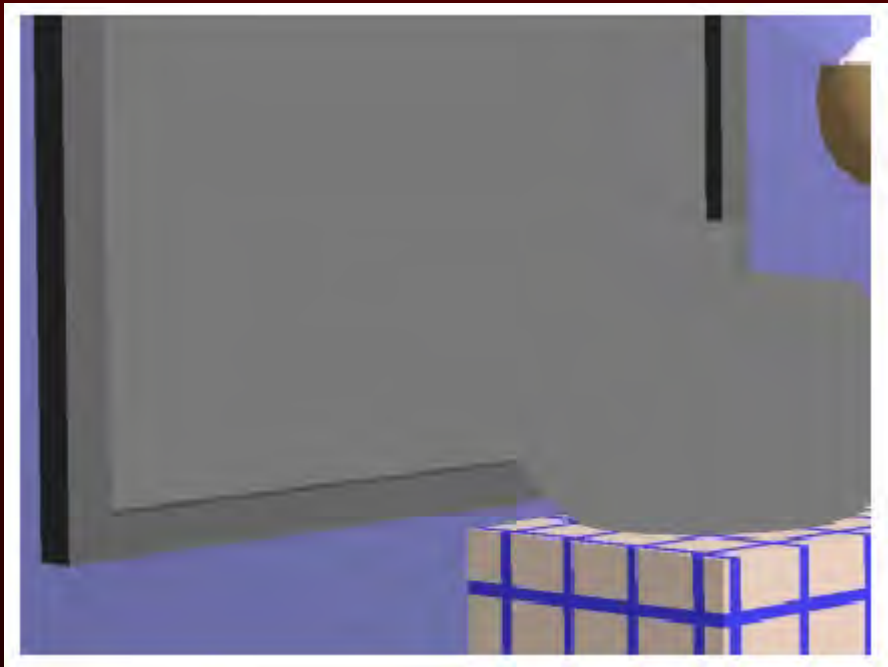
Transparency Rays

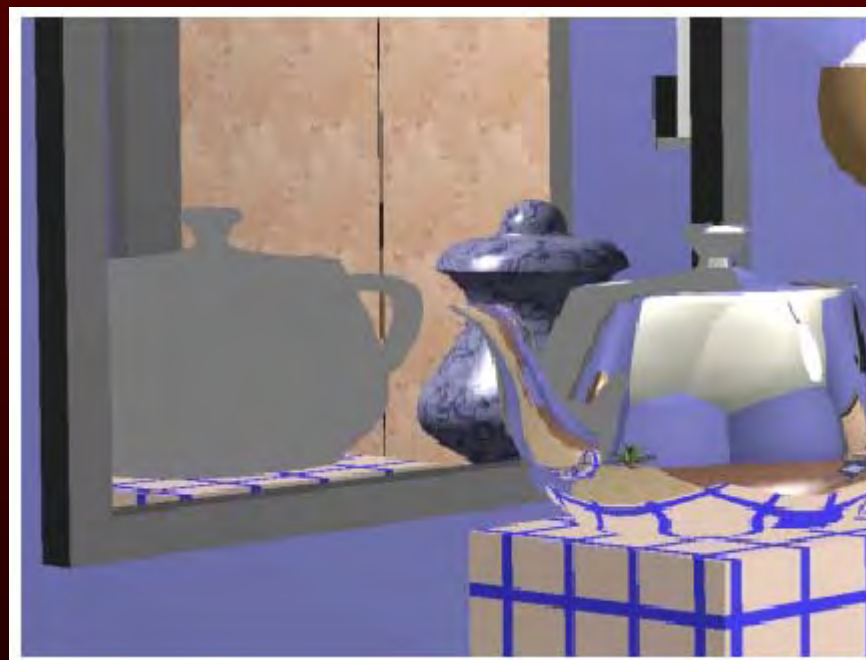
- Some objects are transparent or translucent.
- The transmitted light also contributes to the surface color, called specular transmission.
- The ray can be refracted based on the object's composition.





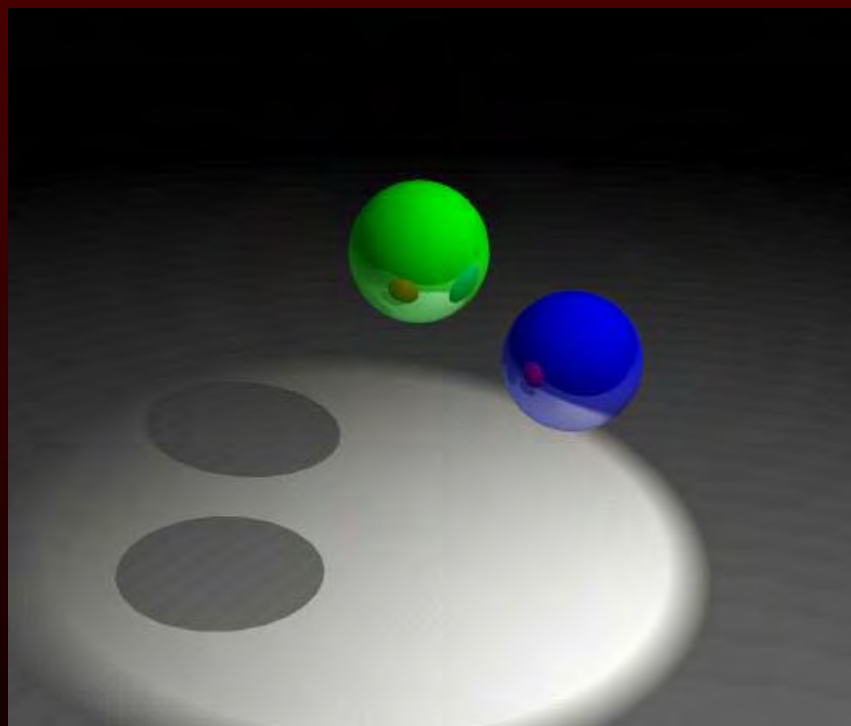












Characteristics

- Follows light energy through rays
- Recursive algorithm
- Depends on POV
- Can handle most kinds of geometry (not just polygons)

Characteristics

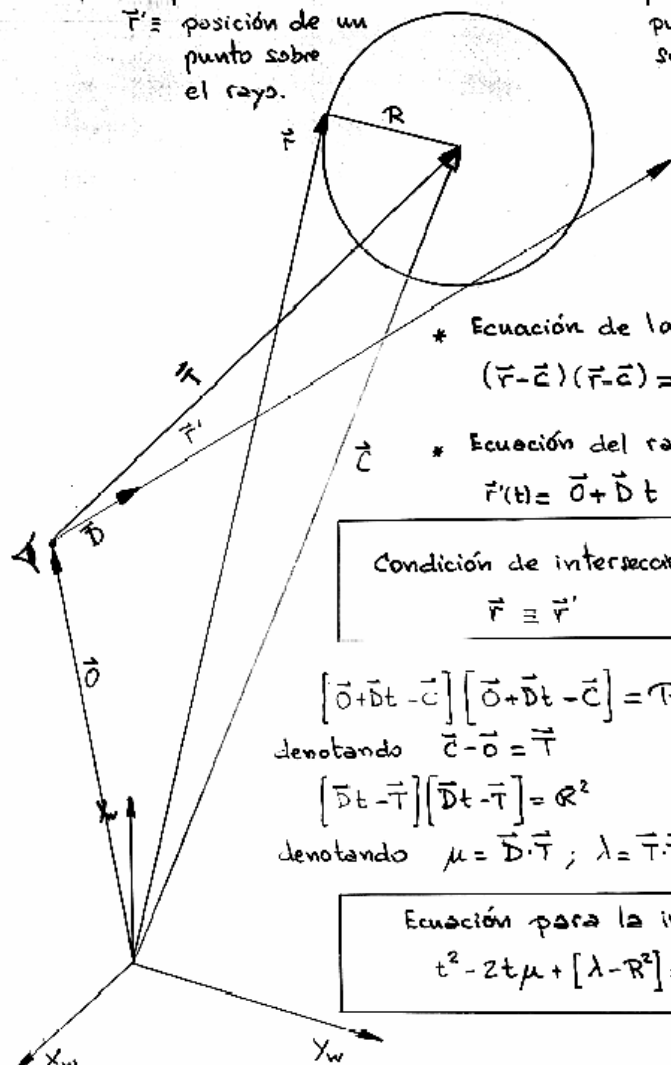
- **Pros:**
 - Reflections
 - Transparencies and refractions
 - Shadows
 - Based on an easy concept
- **Inconvenientes:**
 - Doesn't allow for a full GI solution
 - Based on approximations of the real phenomena

Ray-Object Intersection

- RT bottleneck!
- In principle, each ray needs to be tested against all objects
- If intersections are found, which is the closest to the camera?
- Need for efficient intersection algorithms
 - Spheres, cones, torii...
- Any way to avoid testing with all the objects, or at least do it in an efficient way?

$\vec{D}$ = dirección del rayo
 $\vec{O}$ = posición del observador
 $\vec{r}$ = posición de un punto sobre el rayo.

$\vec{C}$ = Centro de la esfera.
 $\vec{r}$ = posición de un punto sobre la superficie.



* Ecuación de la superficie
 $(\vec{r} - \vec{C})(\vec{r} - \vec{C}) = R^2$

* Ecuación del rayo
 $\vec{r}'(t) = \vec{O} + \vec{D}t \quad t \geq 0$

Condición de intersección
 $\vec{r} \equiv \vec{r}'$

$[\vec{O} + \vec{D}t - \vec{C}][\vec{O} + \vec{D}t - \vec{C}] = R^2$
 denotando $\vec{C} - \vec{O} = \vec{T}$
 $[\vec{D}t - \vec{T}][\vec{D}t - \vec{T}] = R^2$
 denotando $\mu = \vec{D} \cdot \vec{T}; \lambda = \vec{T} \cdot \vec{T}$

Ecuación para la intersección
 $t^2 - 2t\mu + [\lambda - R^2] = 0$

AN EFFICIENT RAY - POLYGON INTERSECTION

ECUACION DEL PLANO EN FORMA NORMAL

$$x \cos \alpha + y \cos \beta + z \cos \gamma = d$$

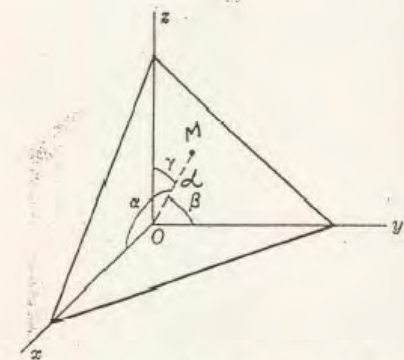
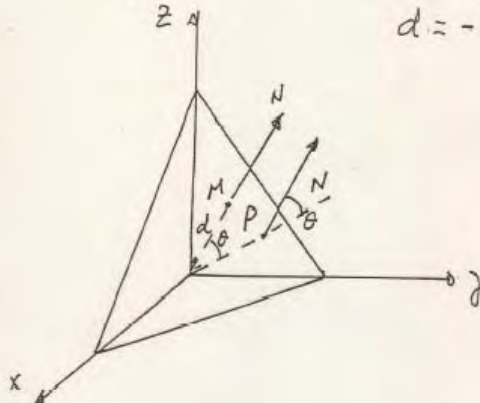
donde d = distancia perpendicular desde O hasta el punto M del plano, mientras que α, β, γ son los ángulos que forma OP con los ejes positivos x, y, z .

$$\vec{N} (\cos \alpha, \cos \beta, \cos \gamma)$$

$$\vec{OM} \parallel \vec{N}$$

Si $V_0 \in \text{plano}$

$$d = -\vec{V}_0 \cdot \vec{N}$$



$$\vec{P} \cdot \vec{N} = |\vec{P}| \cdot |\vec{N}| \cdot \cos \theta = \hat{P} \cos \theta = d$$

ECUACION IMPLICITA

$$\vec{N} \cdot \vec{P} + d = 0$$

Intersection Acceleration Techniques

- What makes ray tracing slow?
 - Number of ray-object intersections performed?
 - Cost of ray-object intersection computation?

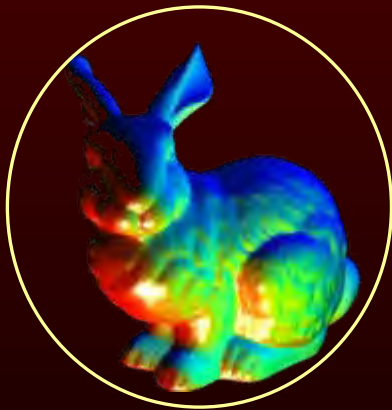
- RT as explained can be veeeeeeeeeeeeery slow
- 90% of time: intersections
 - The more objects in the scene, the longer the time
 - The larger the image, the longer the time
- Acceleration techniques:
 - Less rays to test intersections
 - Try to avoid expensive tests until really needed
 - ...
- Bounding volumes, spatial subdivision...

Bounding volumes

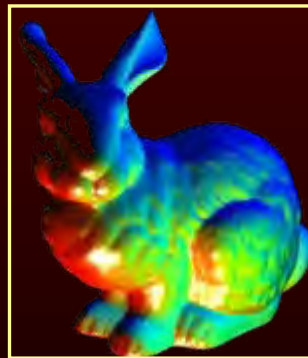
- Concept: to enclose a complex geometry into a simpler one.
 - No easy intersection? No complex one
 - Easy intersection? Check complex one
- *The closer the BV to the object, the better*

Object Bounding Volume

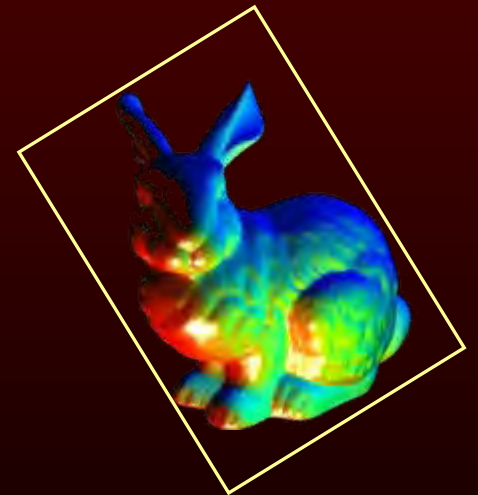
- Surround objects with a bounding volume.
- Fast determination if object needs to be intersected further.



Sphere:
Fastest, poor fit



AABB:
Fast, good fit



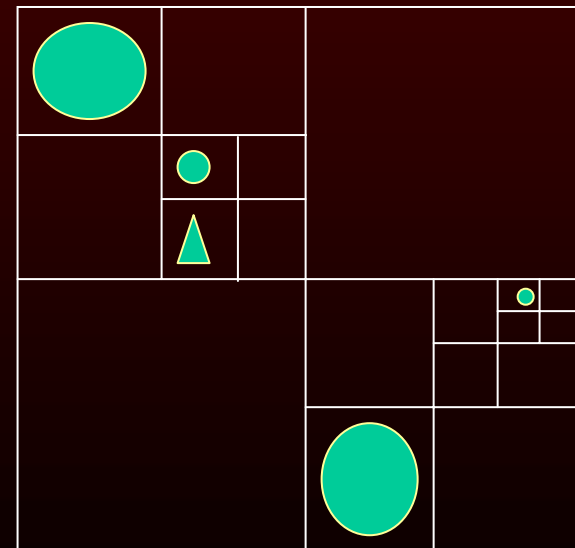
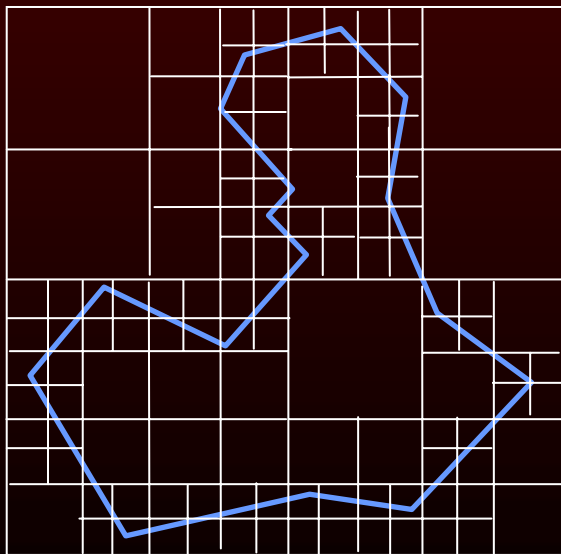
OBB:
Fast, better fit

Bounding volumes

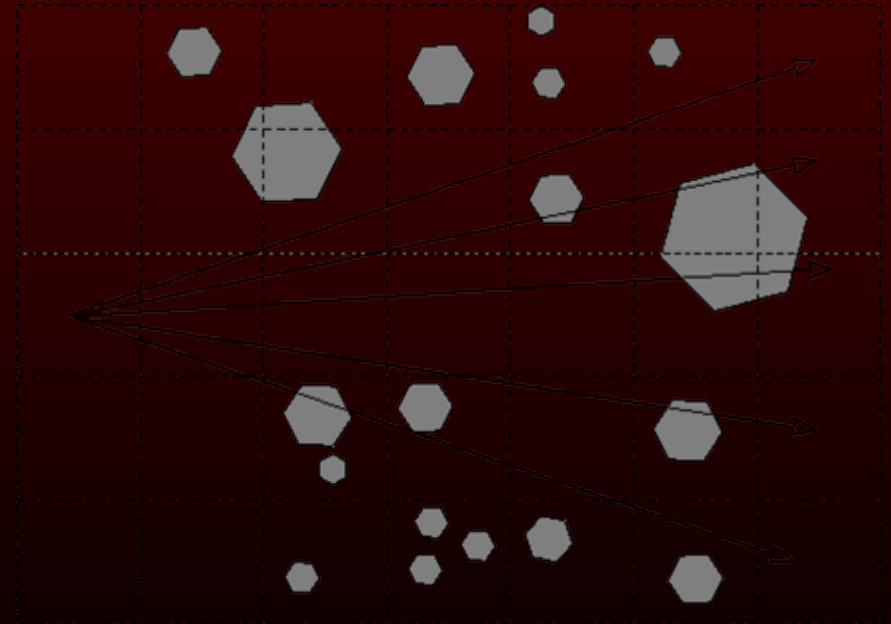
- If there's a BVH, this process can be repeated iteratively digging deeper into the hierarchy

Bounding Volume Hierarchies

- Create a hierarchy of bounding volumes.
- Can be applied to the objects and/or the scene.
- Rays first test the top level, then descend into the tree testing until an object intersection is found or the ray passes through the scene without touching anything.

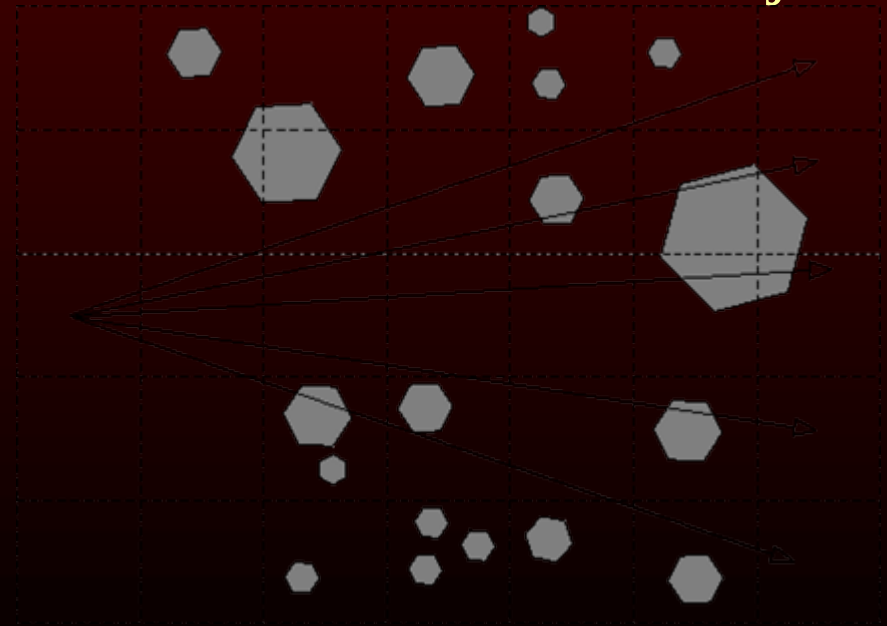


- No overlapping regions
- Non-uniform



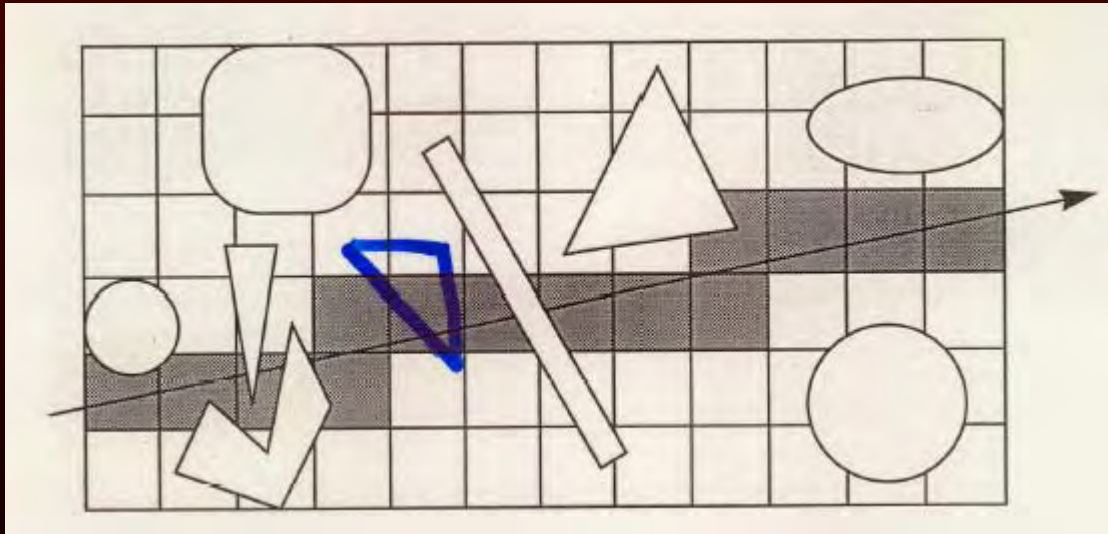
Spatial subdivision

- Subdivide space in regions
- Make a list of objects in each region
- Only test intersections in those regions visited by the ray
- Avoid computing twice the same intersection if an object lies on two or more regions

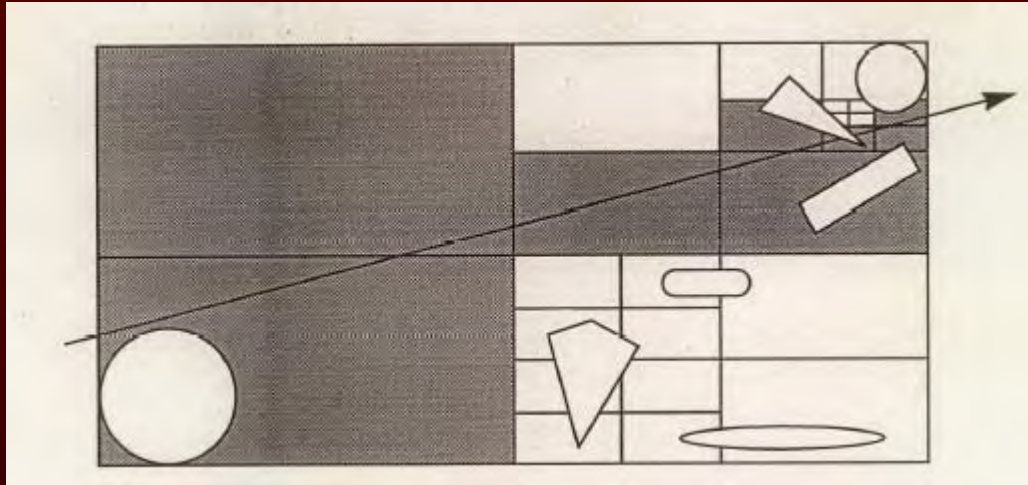


Spatial subdivision

- Basically, we only test in the ray's near surroundings



Non-uniform Spatial subdivision



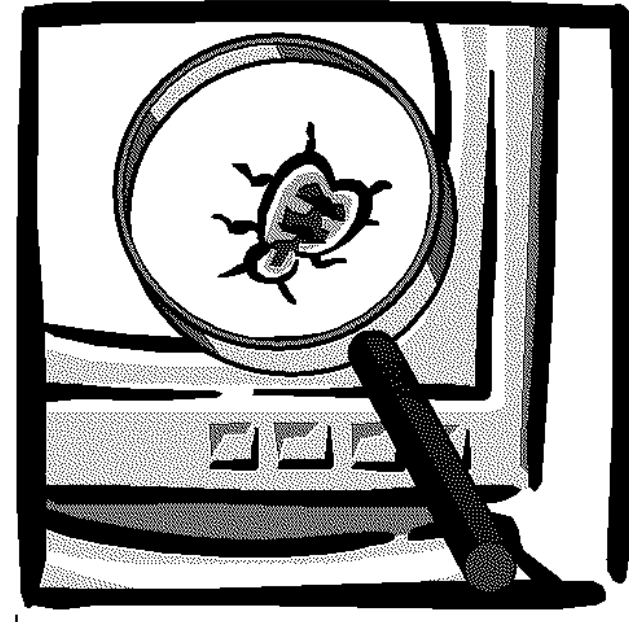
- Void zones are skipped faster
- Real work only done on high-density areas

Conclusions

- Trade-off between subdivision and scene complexity on a per-scene basis
- **Example:** 10.000 objects took 52 seconds with non-uniform subdivision, 15 minutes with uniform subdivision and 40 days with no subdivision!

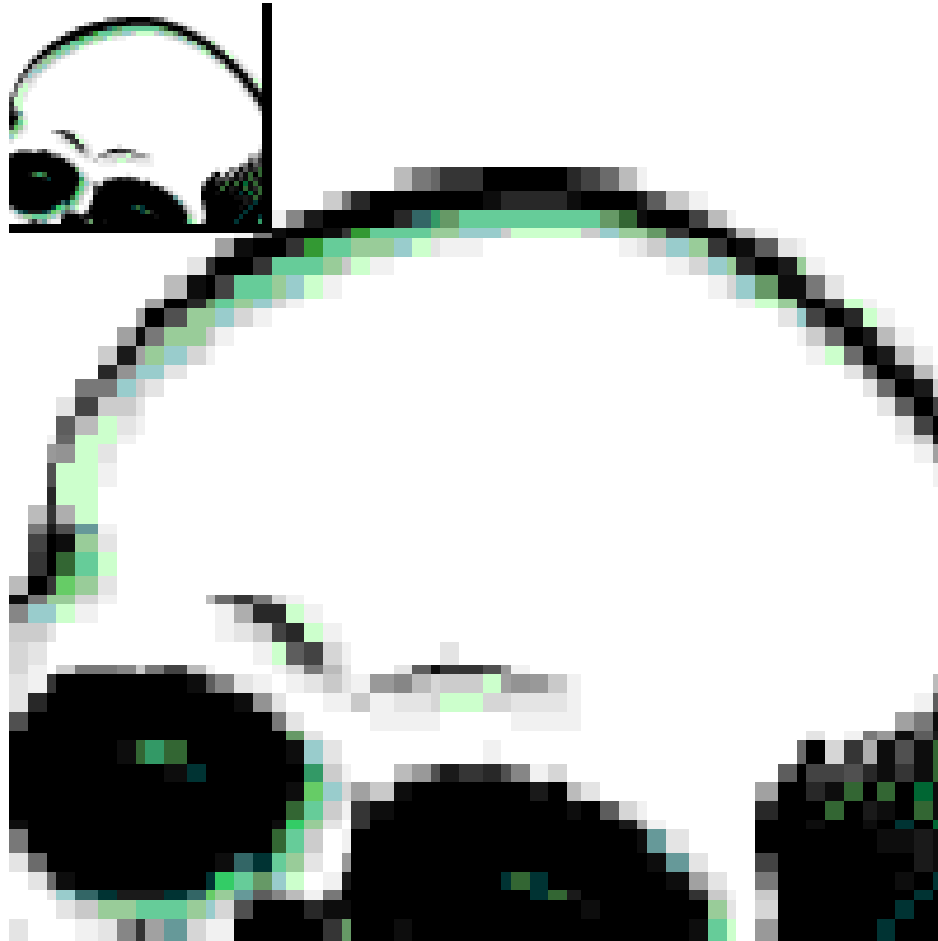
References

- Peter Shirley, “*Realistic Ray Tracing*”, ISBN 1-56881-110-1
- Andrew Glassner, “*An Introduction to Ray Tracing*”, ISBN 0-12-286160-4
- Steve Pettifer (<http://www.cs.man.ac.uk/people/srp>)
- Anselmo Lastra (<http://www.cs.unc.edu/~lastra>)
- Paul Rademacher (<http://www.cs.unc.edu/~rademach>)
- Mark Harris (<http://www.cs.unc.edu/~harrism>)
- Kenny Hoff (<http://www.cs.unc.edu/~hoff>)
- POV-Ray (<http://www.povray.org>)



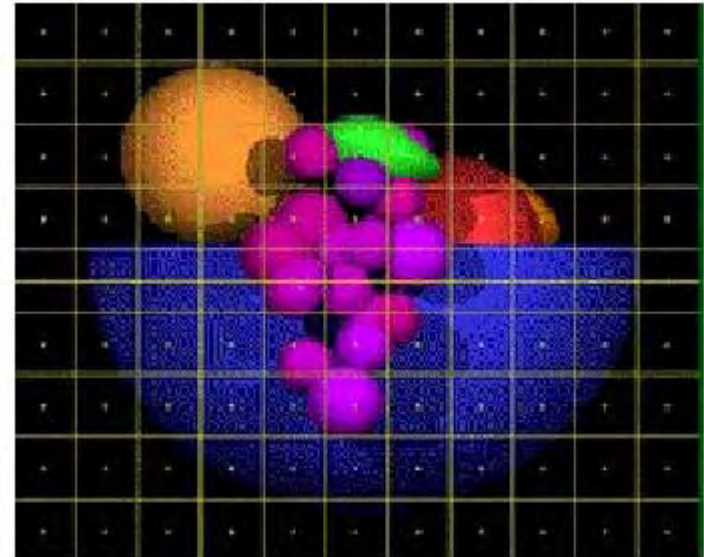
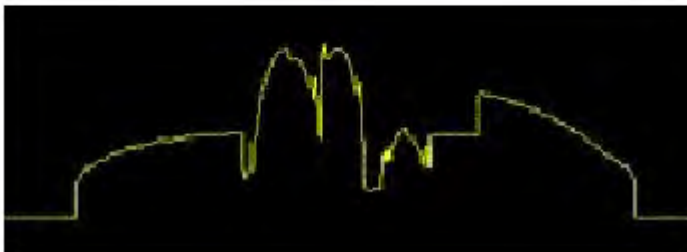
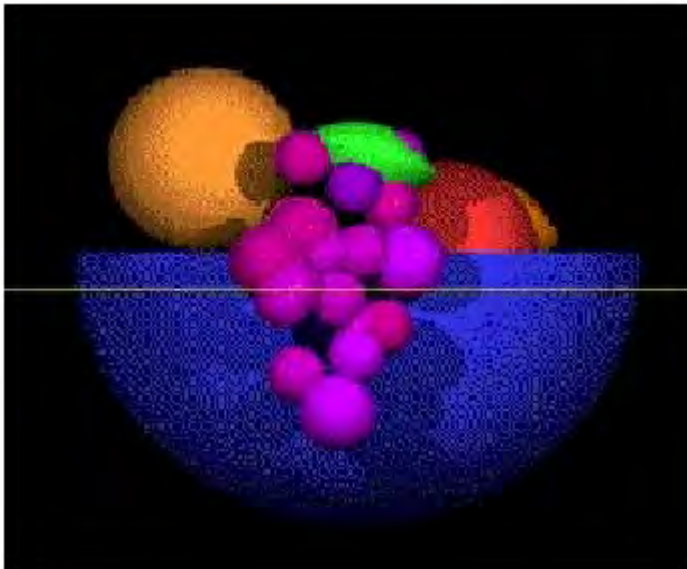
Sampling and aliasing

Diego Gutiérrez

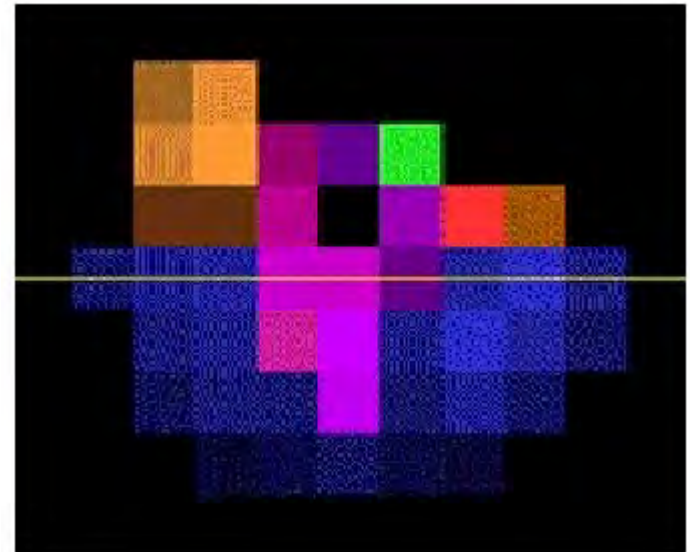
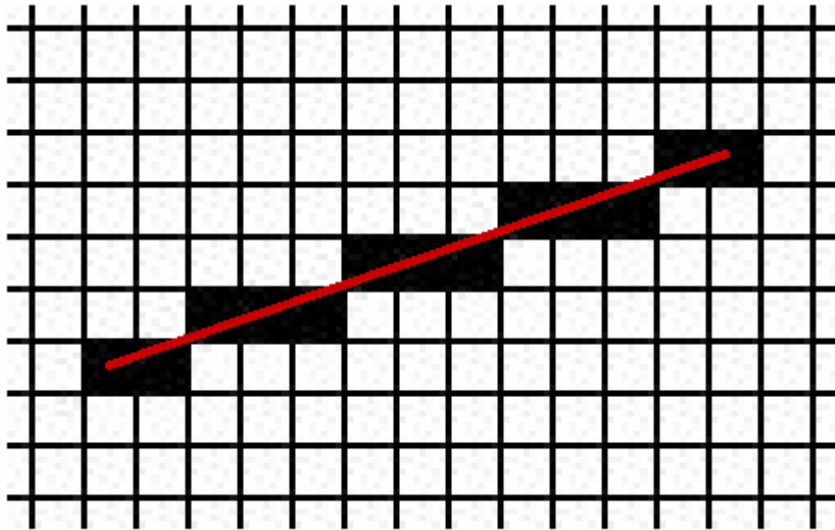
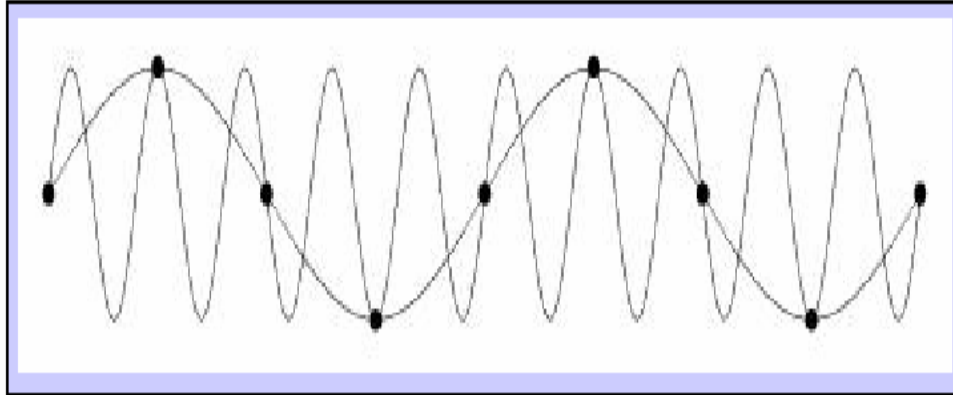


Qué es el *aliasing*

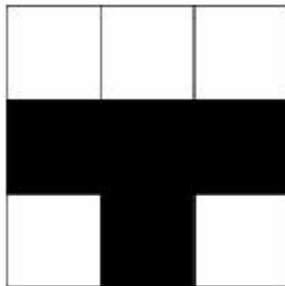
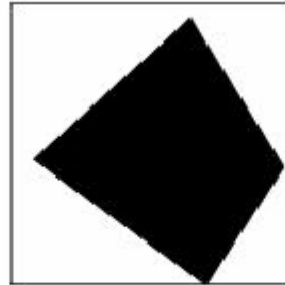
◆ Continuous vs. sampled



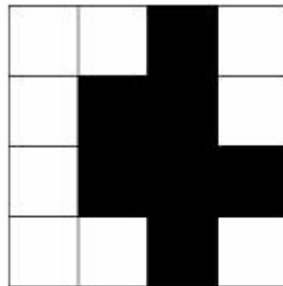
Qué es el *aliasing*



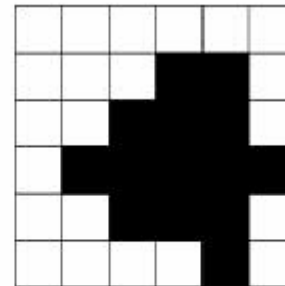
Qué es el *aliasing*



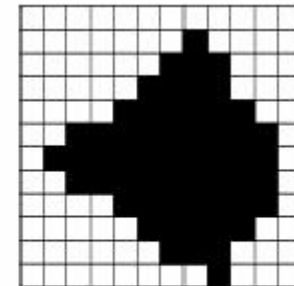
3x3



4x4

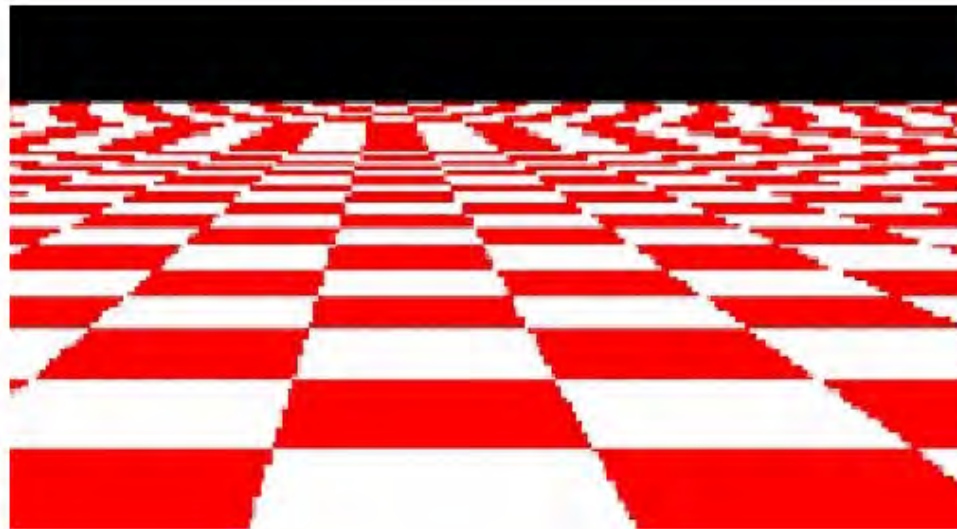
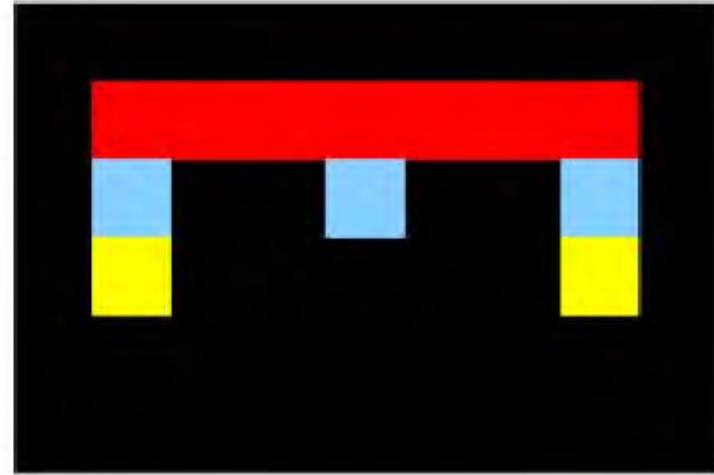
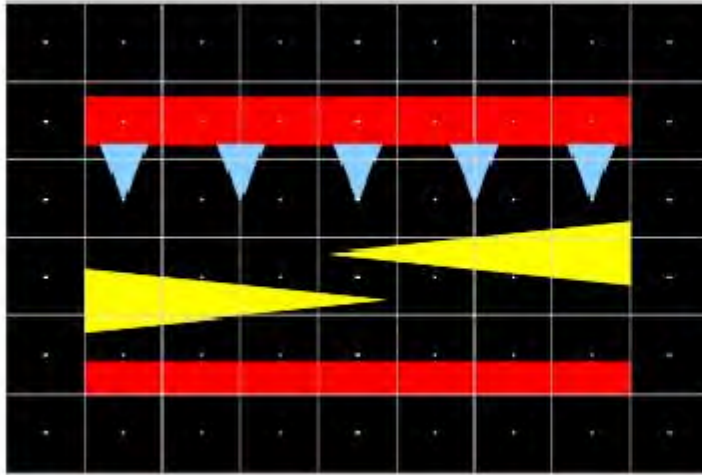


6x6



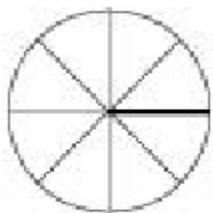
12x12

Qué es el *aliasing*

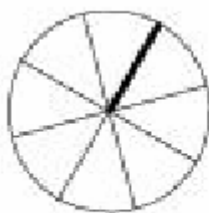


Qué es el *aliasing*

◆ Temporal aliasing



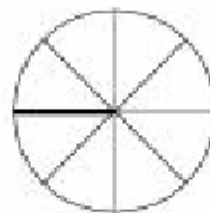
0



$5/6$



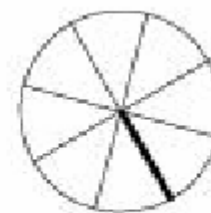
$10/6$



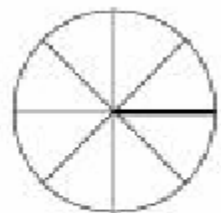
$15/6$



$20/6$

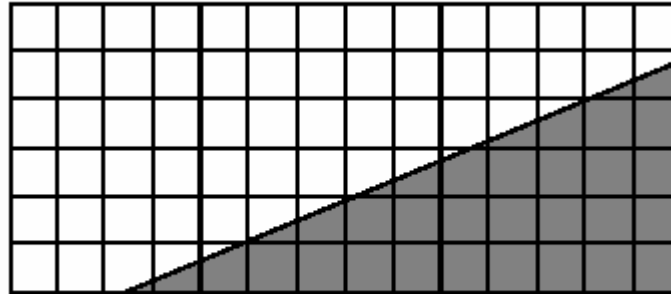


$25/6$

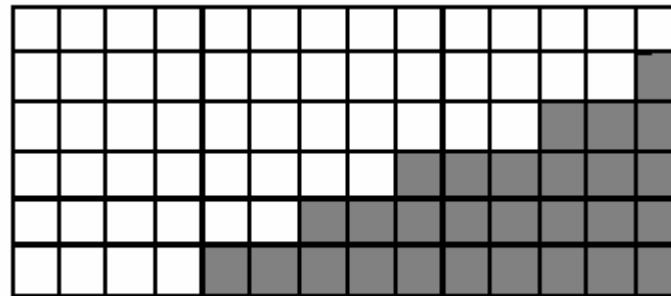


5

Mundo real vs. Mundo digital



(a) reality



(b) image

Sampling

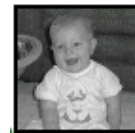
- ◆ An image is a continuous signal $I(x,y)$, with an intensity value for each coordinate (x,y)



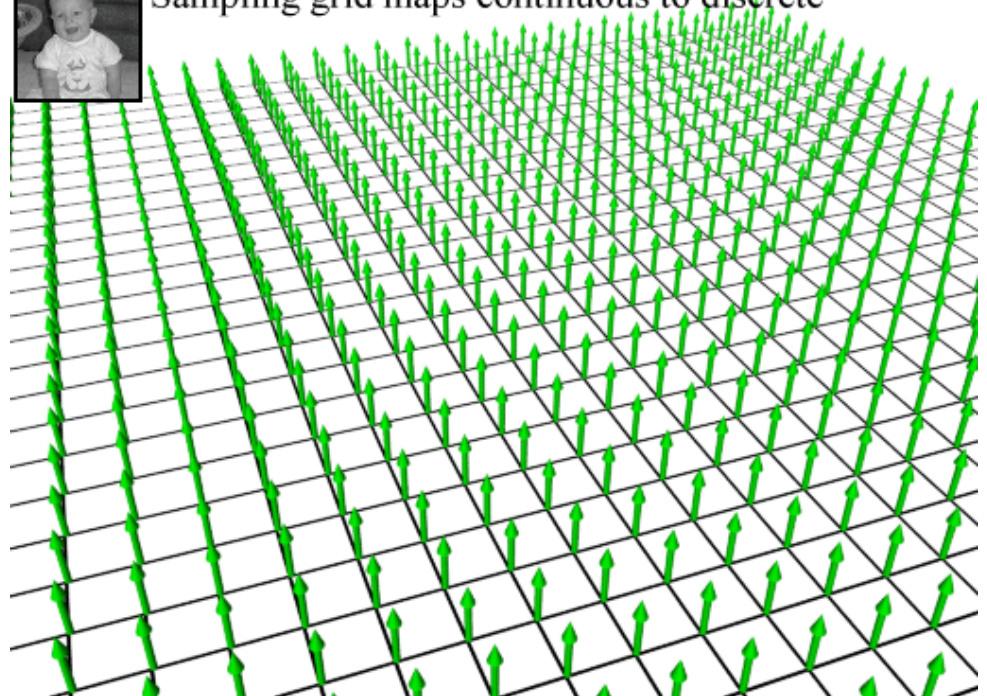
Sampling

- ◆ We sample it by multiplying it by a sampling grid (deltas)

$$\delta(x, y) = \begin{cases} 1, & (x, y) = (0, 0) \\ 0, & \text{otherwise} \end{cases}$$

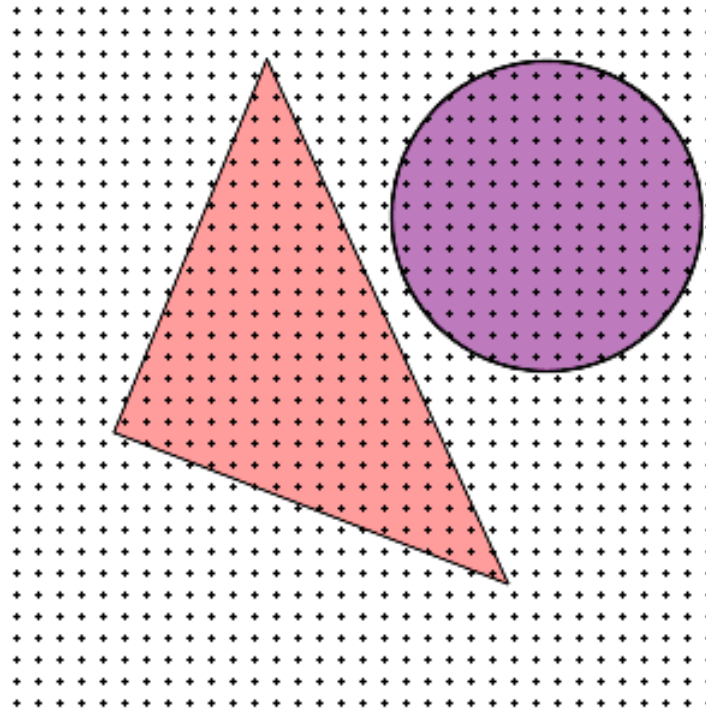


Sampling grid maps continuous to discrete



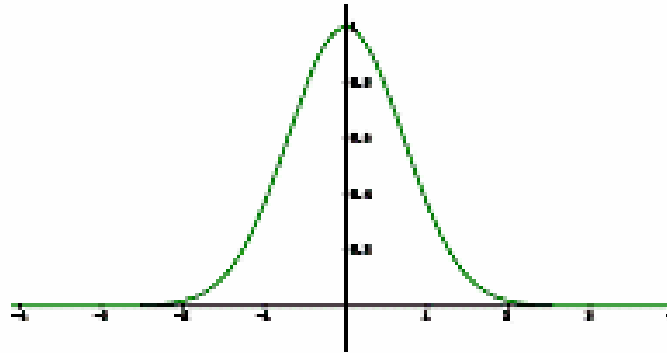
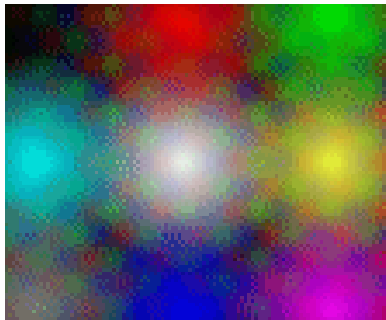
Sampling

- ◆ We save that as what we call “an image”



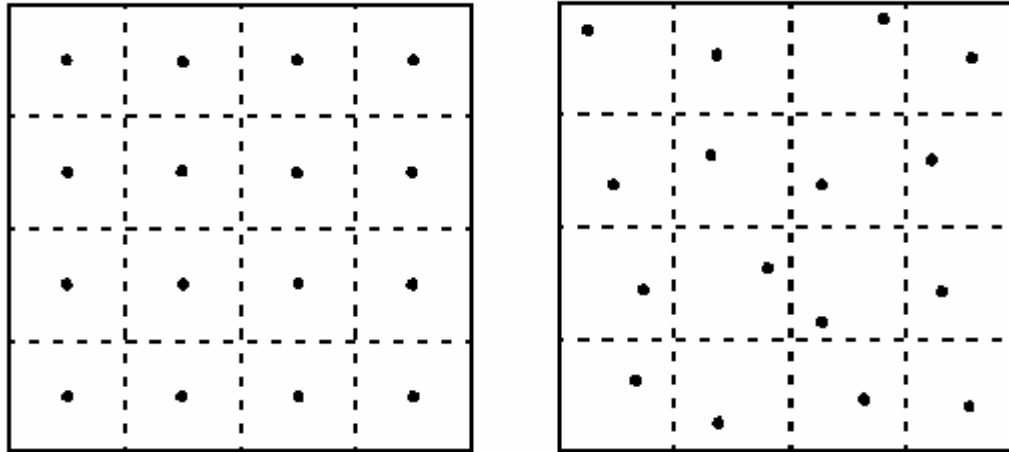
What is a pixel?

- ◆ The way we've described it, a pixel is a point
- ◆ Not a small square, not a small circle
- ◆ A point. No dimensions
- ◆ So.. How do we see it?
- ◆ Fortunately our display technology is far from mathematically-perfect!



Antialiasing

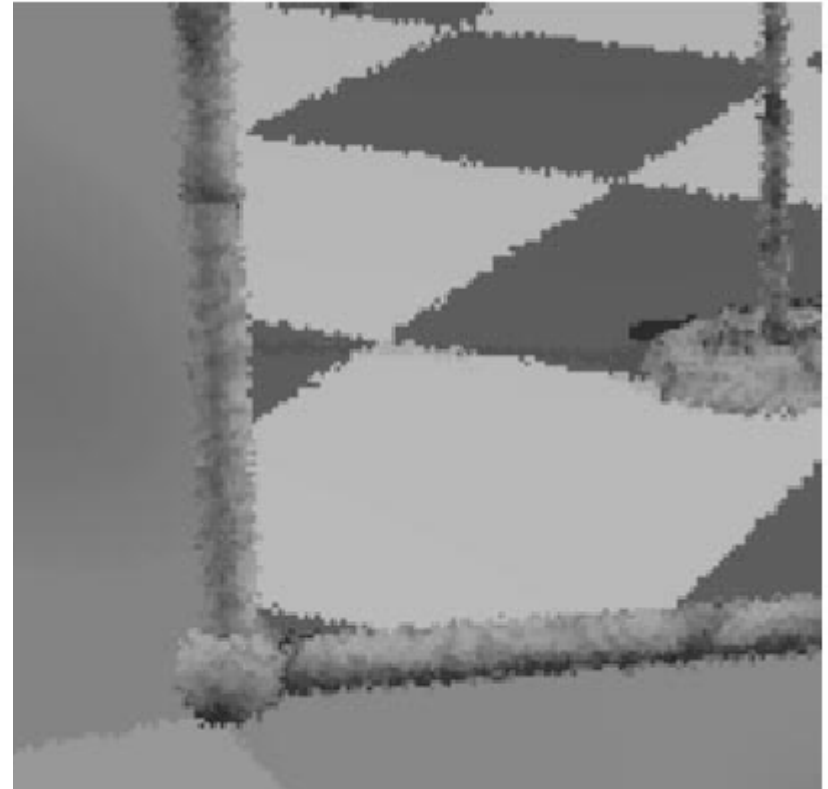
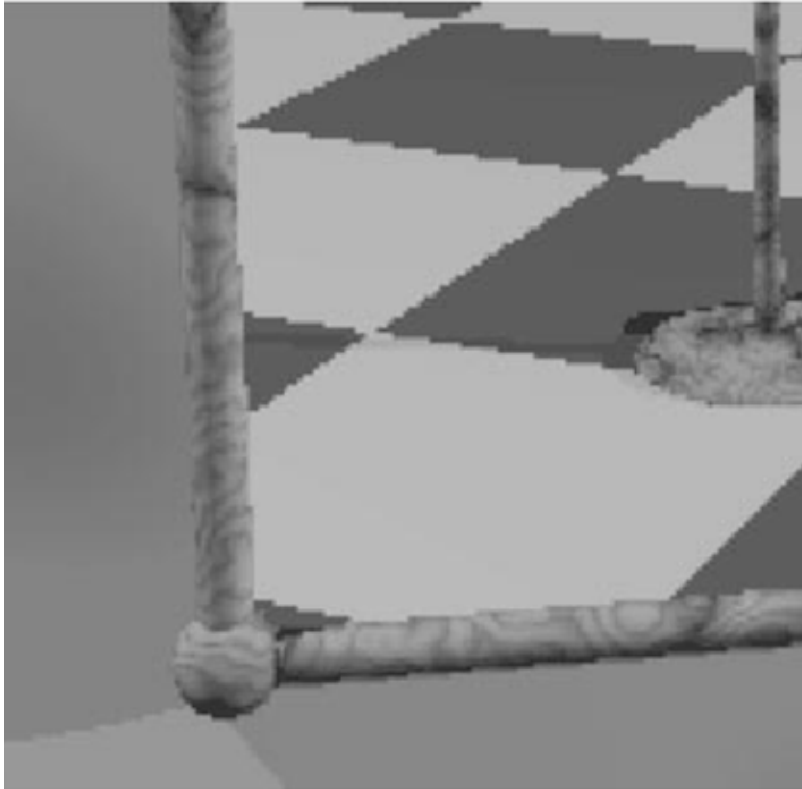
- ◆ Antialiasing: to reduce aliasing effects
- ◆ Supersampling
 - *Uniform supersampling*
 - *Jittered supersampling*



- ◆ Works best: *jittered*

Antialiasing

◆ Supersampling: uniform vs *jittered*



Antialiasing

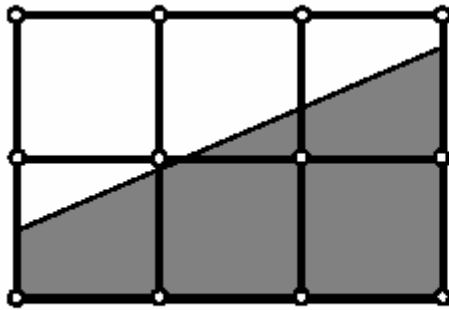
◆ Adaptive antialiasing:

- Supersampling per pixel: highly inefficient!
- Aliasing only visible in high-contrast areas

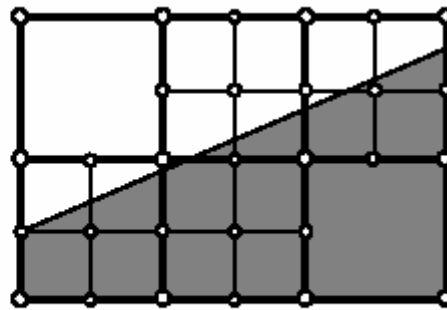
◆ Process:

- 1. Shoot rays to the pixel's corners
- 2. Examine difference in values
- 3. If higher than a threshold, subdivide, repeat

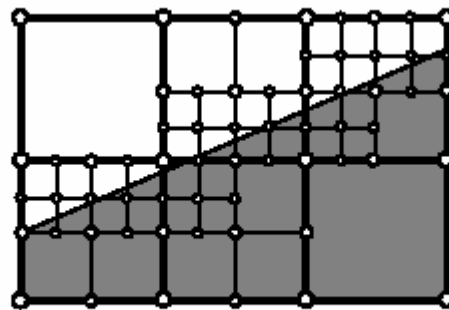
Antialiasing



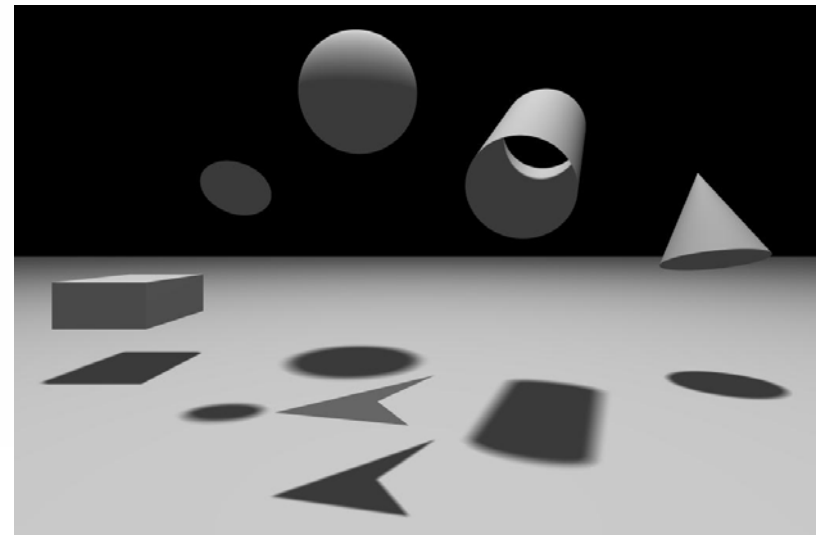
(a)



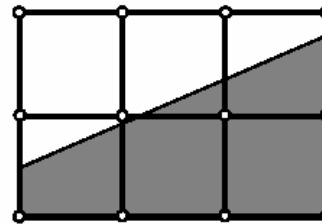
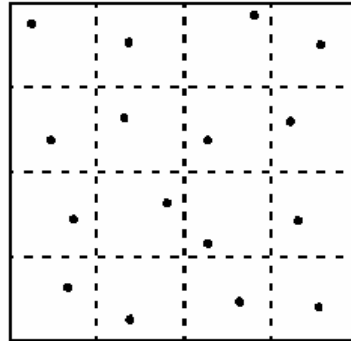
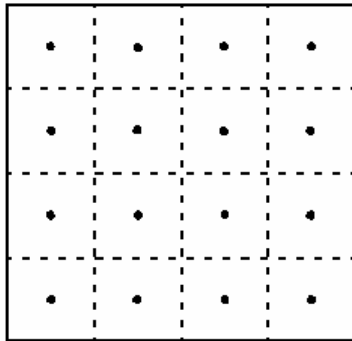
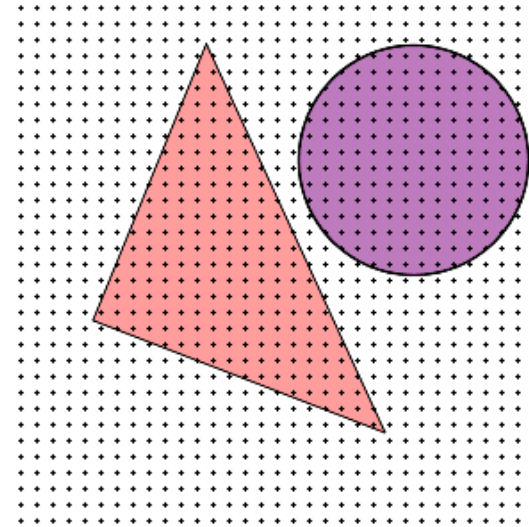
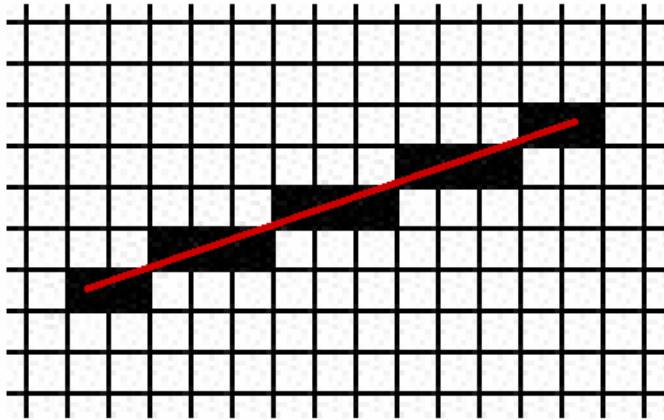
(b)



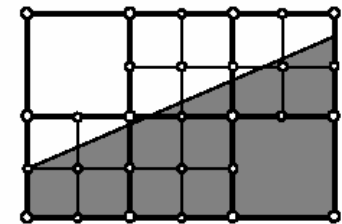
(c)



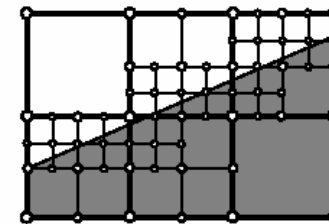
Sampling y aliasing (in a nutshell)



(a)



(b)



(c)