

Introduction to Computer Graphics

11. Advanced Rendering

I-Chen Lin

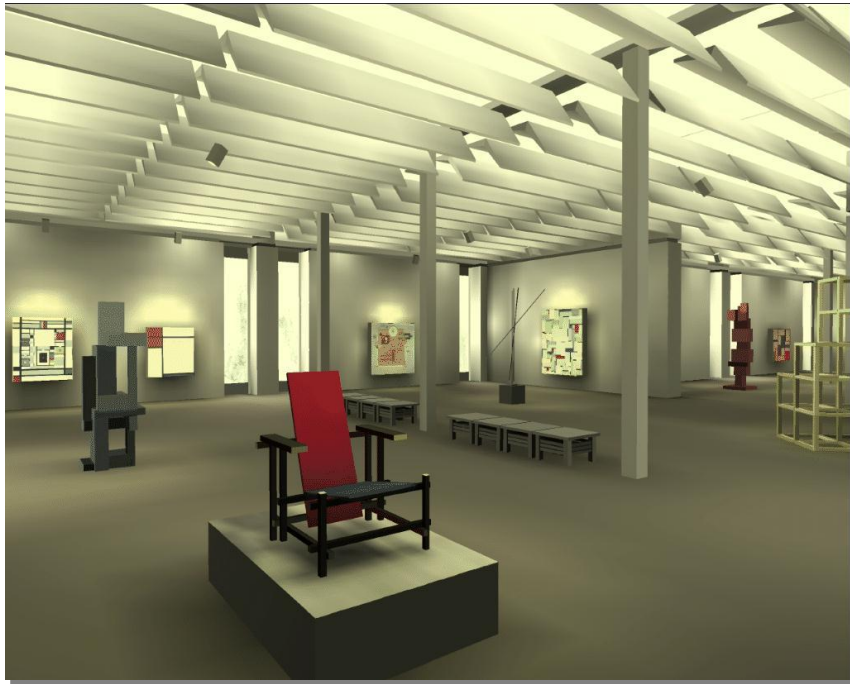
National Chiao Tung University

Textbook: E. Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson
John C. Hart, slides of Advanced Topics in Computer Graphics
H.W. Jenson, Realistic Image Synthesis using Photon mapping

Outline

- ▶ Going beyond pipeline rendering
- ▶ Ray tracing
- ▶ Rendering equation
- ▶ Radiosity
- ▶ Photon mapping

Can We Render Images Like These?

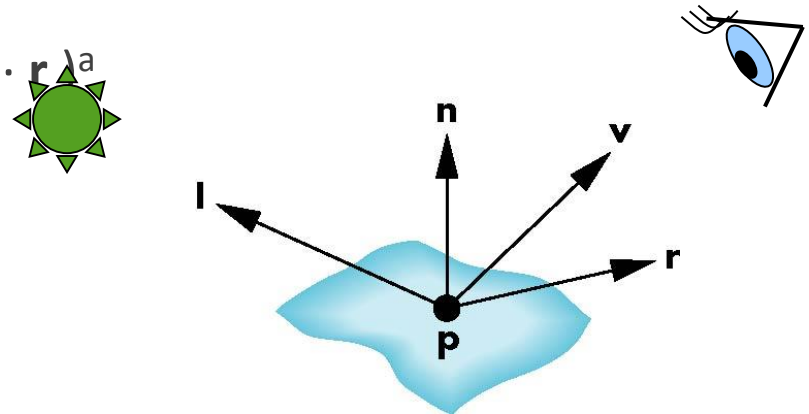


Picture from <http://www.graphics.cornell.edu/online/realistic/>

Local Illumination

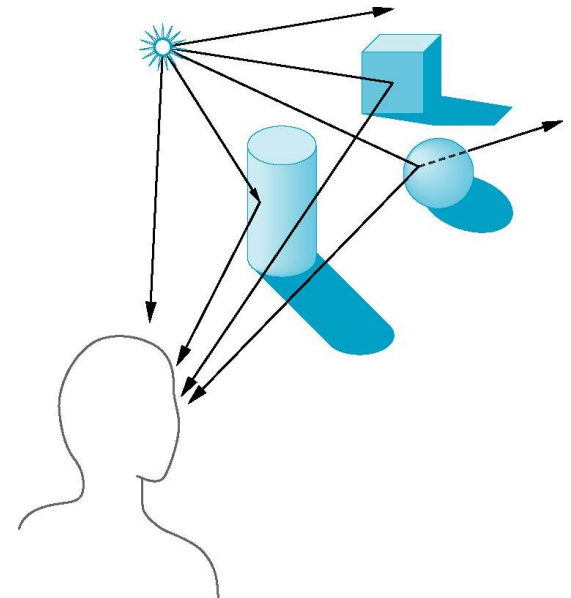
- ▶ The Phong model is a local illumination model
 - ▶ Shaded color depends only on
 - ▶ Surface normal, viewing direction, light direction
 - ▶ Ambient, diffuse, and specular reflectances

- ▶
$$I = I_{ambient} + I_{diffuse} + I_{specular}$$
$$= k_a I_a + k_d I_d (I \cdot n) + k_s I_s (v \cdot r)^a$$



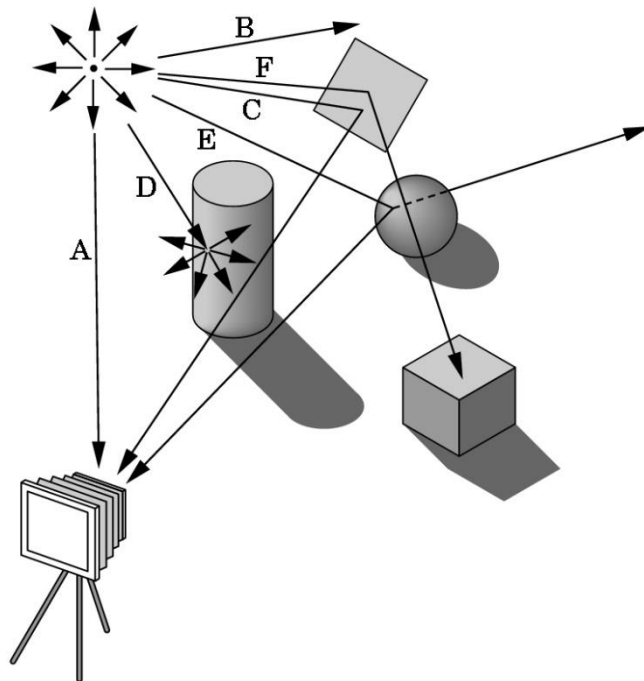
Local Illumination (cont.)

- ▶ Don't take other surfaces into account !
 - ▶ Other surfaces cannot block light (no shadows)
 - ▶ Omitting light from reflection or refraction of other objects.
- ▶ These interactions happen in reality!



Forward Ray Tracing

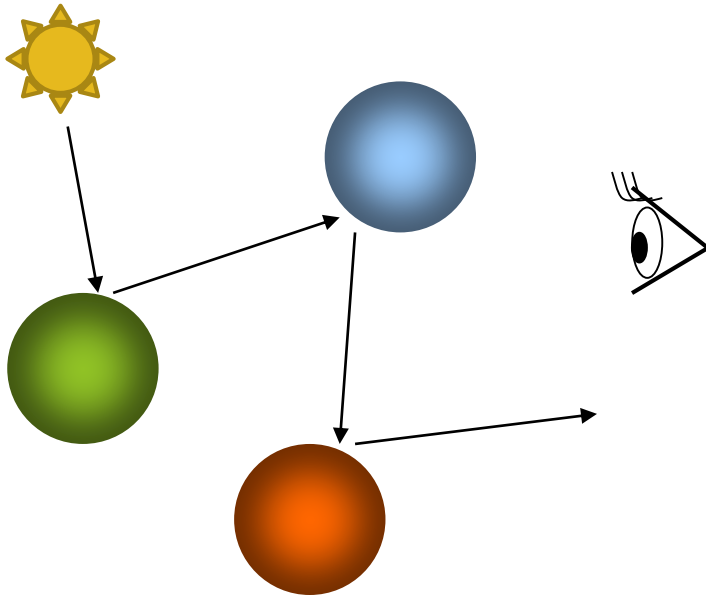
- Rays emanate from light sources and bounce around in the scene.
- Rays that pass through the projection plane and contribute to the final image.



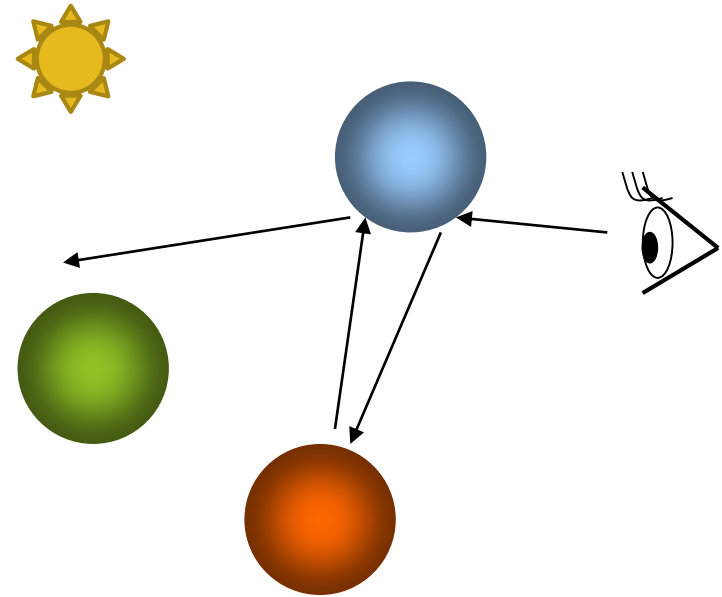
What's wrong with this method?

Forward vs. Backward

Starting at the light



Starting at the eye

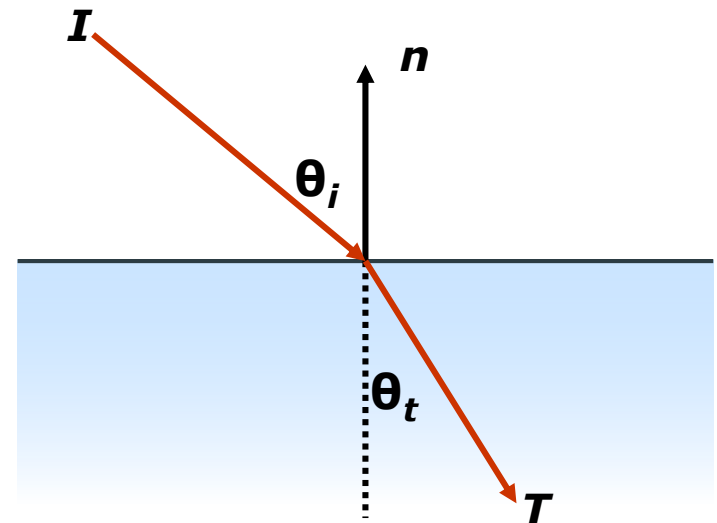


Refraction of Light

- ▶ Rays transitioning between materials are bent around normal
 - ▶ every material has an index of refraction
- ▶ Angles with surface normal obey Snell's Law

$$\frac{\sin \theta_i}{\sin \theta_t} = \eta_{ti} = \frac{\eta_t}{\eta_i} \quad \text{Where } \eta \text{ is the indices of refraction}$$

Material	Index of Refraction
<i>Vacuum</i>	1.0
<i>Ice</i>	1.309
<i>Water</i>	1.333
<i>ethyl alcohol</i>	1.36
<i>Glass</i>	1.5–1.6
<i>Diamond</i>	2.417

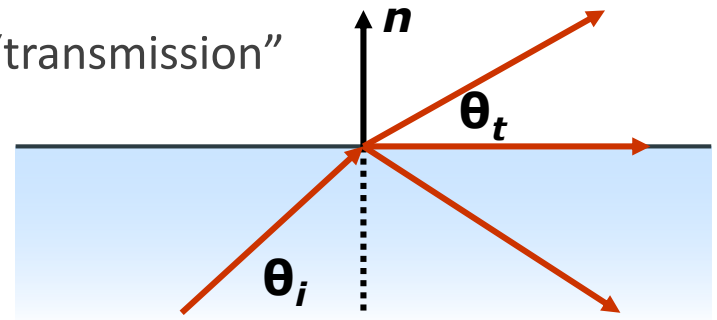


Refraction of Light (cont.)

- ▶ When entering material of lower index
 - ▶ Ray bends outward from normal
 - ▶ What if the angle is more than 90° ?
 - ▶ Ray is actually reflected off the boundary
 - ▶ this is called total internal reflection (like fiber optics)
- ▶ Total internal reflection occurs when

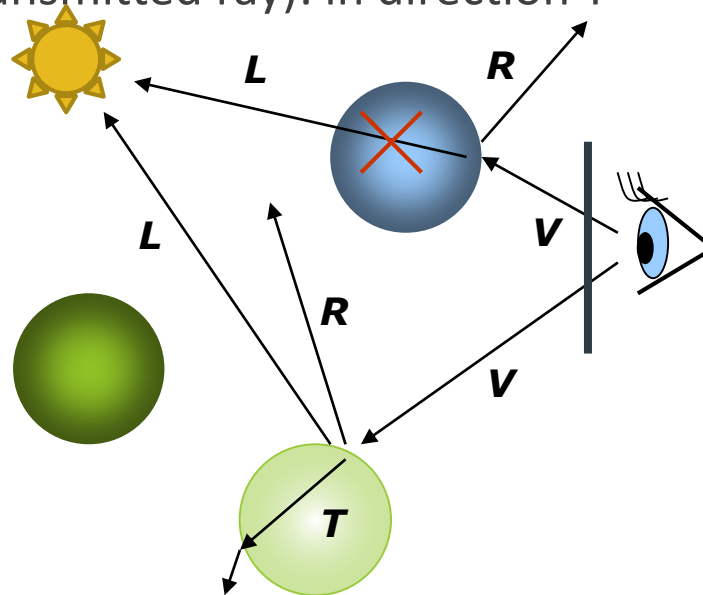
$$\theta_i > \theta_{critical}, \text{ where } \theta_{critical} = \sin^{-1} \frac{\eta_t}{\eta_i}$$

- ▶ just need to check for this critical angle
- ▶ if above it, use specular reflection for “transmission”



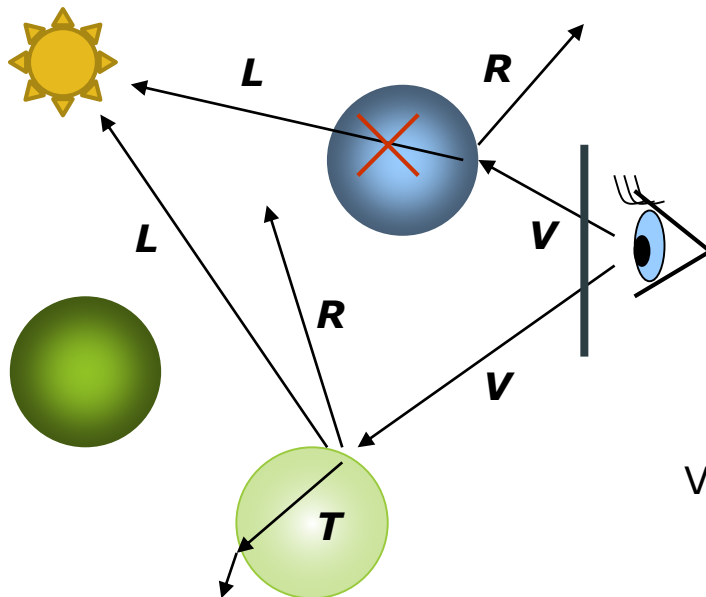
Whitted Ray-tracing

1. For each pixel, trace a **primary ray** (eye ray) to the first visible surface.
2. For each intersection trace secondary rays:
 - ▶ **Shadow rays**: in directions L to light sources
 - ▶ **Reflected ray**: in direction R
 - ▶ **Refracted ray** (transmitted ray): in direction T



Whitted Ray-tracing (cont.)

- ▶ Every surface intersection spawns
 - ▶ 1 reflected ray
 - ▶ 1 transmitted ray
 - ▶ 1 shadow ray per light
 - ▶ Shaded color of $V_i = \text{Valid}(L_i) \times \text{PhongModel} + \text{ReflectedRay} + \text{TransmittedRay}$



$\text{Valid}(L_i) = 1$, if visible to the light source
0, otherwise

A Simple Ray Tracer

```
void raytrace()  
    for all pixels (x,y)  
        image(x,y) = trace(compute_eye_ray(x,y))
```

```
rgbColor trace(ray r)  
    for all surfaces s  
        t = compute_intersection(r, s)  
        closest_t = MIN(closest_t, t)  
  
    if( hit_an_object )  
        return shade(s, r, t)  
    else  
        return background_color
```

A Simple Ray Tracer (cont.)

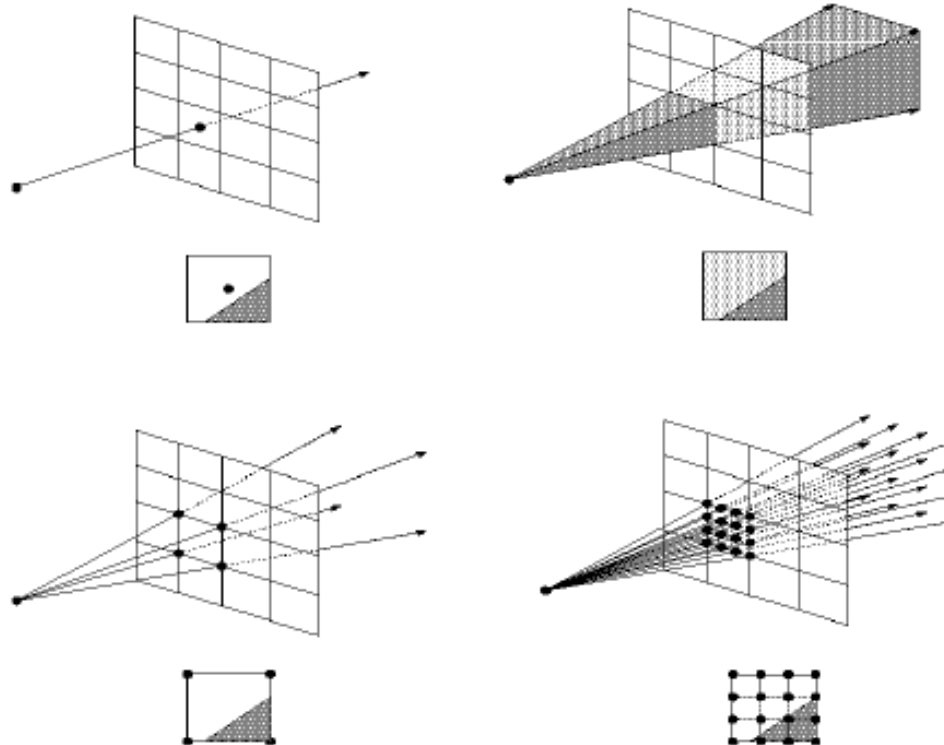
```
rgbColor shade(surface s, ray r, double t)
    point x = r(t)
    rgbColor color = black

    for each light source L
        if( closest_hit(shadow_ray(x, L)) >= distance(L) ) {
            color += shade_phong(s, x)
        }
    color += k_specular * trace(reflected_ray(s,r,x))
    color += k_transmit * trace(transmitted_ray(s,r,x))

    return color
```

Supersampling

- ▶ Aliasing problems.
- ▶ We can approximate the average color of a pixel's area by firing multiple rays and averaging the result.



Efficiency of Ray Tracing

- ▶ Consider this example
 - ▶ image resolution of $1024 \times 768 = 786,432$ pixels
 - ▶ 3x3 supersampling = 7 million eye rays
 - ▶ recursion depth 5 = $63 \times 7 = 441$ million rays
 - ▶ each tested against 10,000 polygons
 - ▶ 4.4 trillion intersection tests (ignoring shadow rays)

Most of the time is spent in the calculation of intersection !

Efficiency of Ray Tracing (cont.)

- ▶ How to efficiently calculate intersections?
 - ▶ Efficient representation of an object.
 - ▶ Bounding boxes
 - ▶ Space partitioning
 - ▶ Octree, BSP tree, etc.
 - ▶ Distributed ray tracing (non-uniform ray distribution)
 - ▶

Efficiency of Ray Tracing (cont.)

- ▶ How to utilize more than 1 computer?

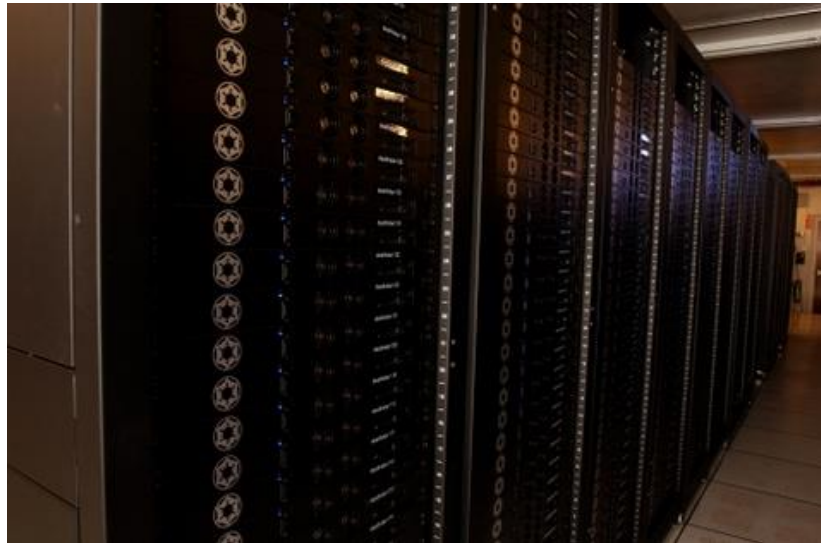


Figure from <http://www.linuxjournal.com>

ILM RackSaver Linux render farm has 1,500 processors in 2003.

- ▶ The efficiency of realistic synthetic image rendering (in movie quality)

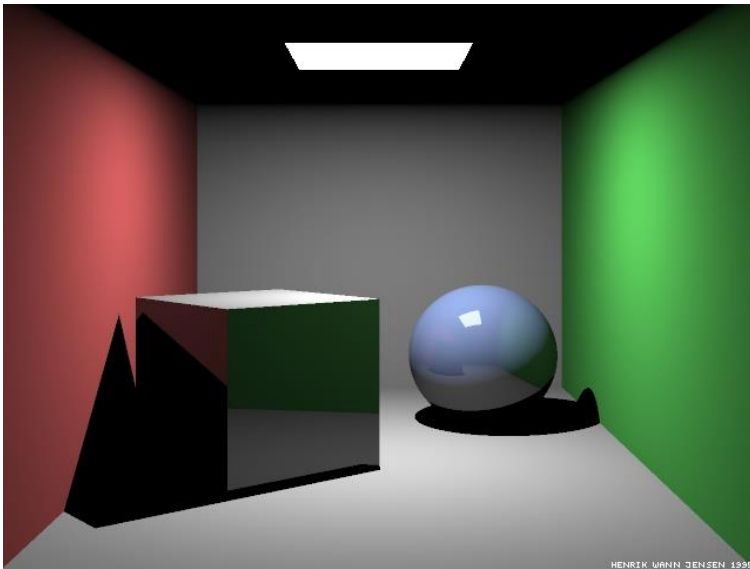
Ray Tracing Example



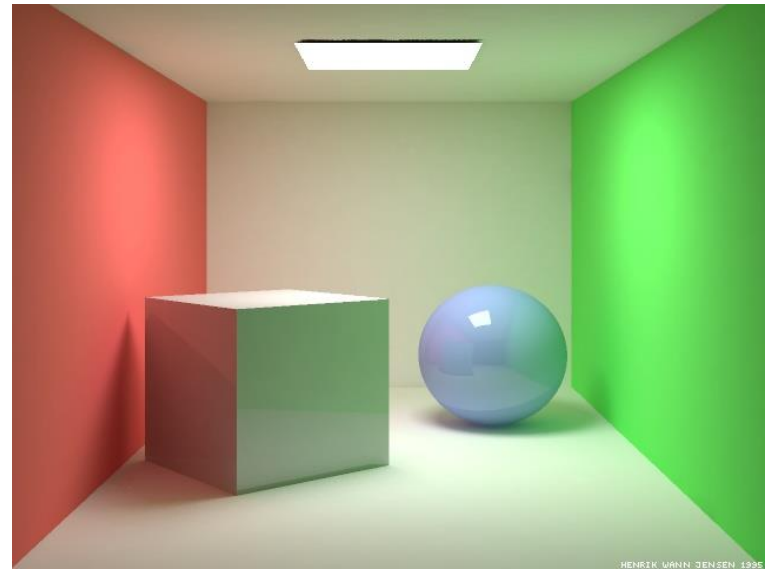
Kettle, Mike Miller, POV-Ray

Ray Tracing Example

What're missing?



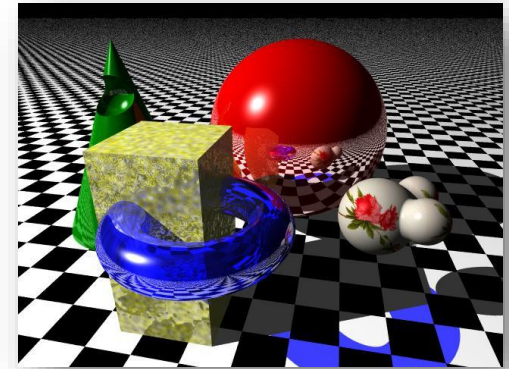
VS



Ray-traced Cornell box by Henrik Jensen,
<http://www.gk.dtu.dk/~hwj>

Ray Tracing vs. Radiosity

- ▶ Ray tracing
 - ▶ An image space algorithm
 - ▶ View-dependent
 - ▶ Rendering scenes with perfect specular reflection and refraction.
 - ▶ Point light sources.
 - ▶ Ideas from the path of light flow



- ▶ Radiosity
 - ▶ An object space algorithm
 - ▶ View-independent (can be pre-computed)
 - ▶ Rendering perfect **diffuse** scenes.
 - ▶ Light sources are **polygonal patches**.
 - ▶ Ideas from the conservation of energy.



The Rendering Equation

- ▶ Regarding the light as a form of energy.
- ▶ In a closed environment, we do not see how the rays have bounced around.
- ▶ What we see is at an equilibrium state.
 - ▶ $[outgoing] = [emitted] + [reflected] + [transmitted]$
 - ▶ (We usually omit the “transmitted” terms)

The Rendering Equation (cont.)

$$I(p, p') = v(p, p') \left[\varepsilon(p, p') + \int \rho(p, p', p'') \cdot I(p', p'') dp'' \right]$$

■ $I(p, p')$: intensity passing from p' to p .

■ $\varepsilon(p, p')$: emitted light intensity from p' to p .

■ $\rho(p, p', p'')$: reflection function at point p' .

■ $v(p, p')$: visibility function

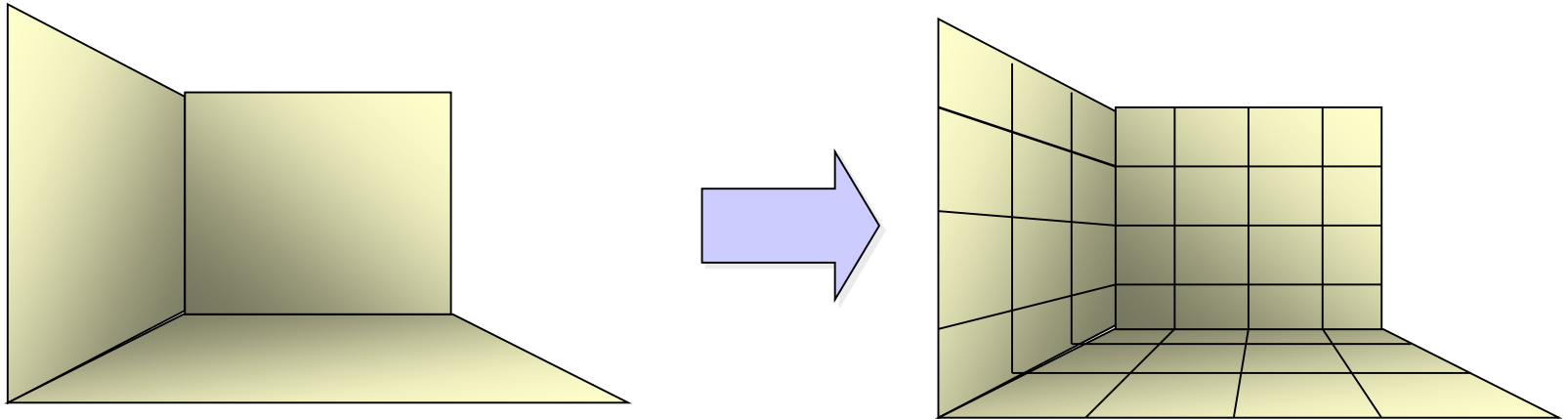
0: if p' is invisible from p .

$1/r^2$: if p' is visible from p .

■ r : distance between p and p' .

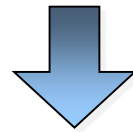
Radiosity

- ▶ One way to simplify the rendering equation.
 - ▶ All surfaces are perfectly diffuse reflectors.
 - ▶ Dealing with diffuse-diffuse interactions.
- ▶ A scene is divided into “patches”.



Radiosity (cont.)

$$I(p, p') = v(p, p') \left[\varepsilon(p, p') + \int \rho(p, p', p'') \cdot I(p', p'') dp'' \right]$$

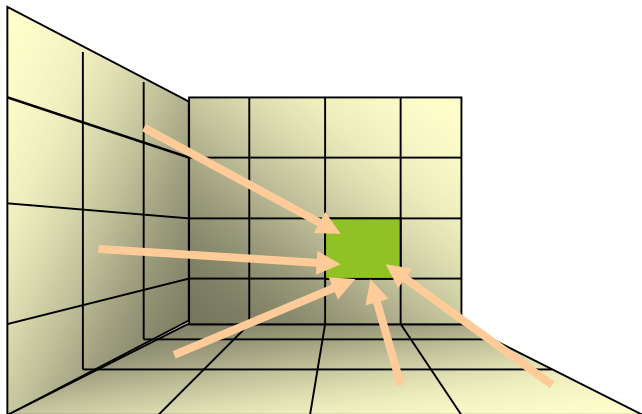


$$b_i a_i = e_i a_i + \rho_i \sum_{j=0}^n f_{ji} b_j a_j$$

The light intensity of i

The emissive intensity

The reflective intensity due to
intensity of all other patches



- ρ_i : reflectance of element i (given)
- b_i : the color of patch i (unknown)
- a_i : the area of patch i (computable)
- e_i : the emissive component (given)
- f_{ji} : the form factor ($j \rightarrow i$) (computable)

Form Factor

- ▶ F_{ji} : Fraction of light leaving element j and arriving at element i
- ▶ Depends on
 - ▶ Shape of patches i and j
 - ▶ Relative orientation of both patches
 - ▶ Distance between patches
 - ▶ Visibility or occlusion by other patches

$$dF_{12} = \frac{\cos \theta_1 \cos \theta_2}{\pi S^2} dA_2$$

$$F_{12} = \frac{1}{A_1} \int_{A_1} \int_{A_2} \frac{\cos \theta_1 \cos \theta_2}{\pi S^2} dA_2 dA_1$$

$$\sum_j F_{ij} = 1 \quad A_1 F_{12} = A_2 F_{21}$$

reciprocity

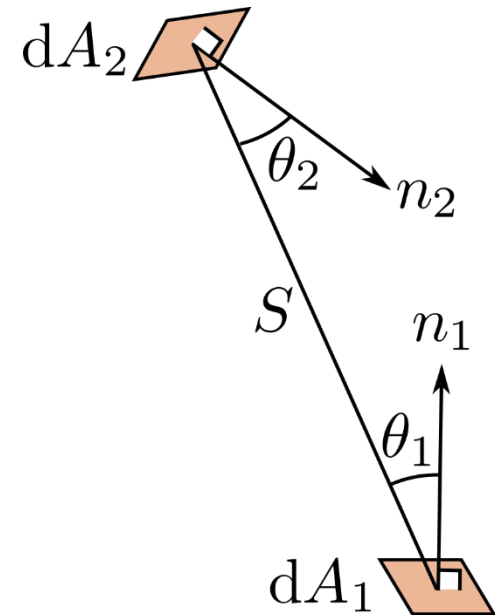


Fig. from: en.wikipedia.org/wiki/View_factor

Radiosity (cont.)

- The reciprocity equation

- $f_{ij} a_i = f_{ji} a_j$

$$b_i a_i = e_i a_i + \rho_i \sum_{j=0}^n f_{ji} b_j a_j$$



$$b_i a_i = e_i a_i + \rho_i \sum_{j=0}^n f_{ij} b_j a_i$$



$$b_i = e_i + \rho_i \sum_{j=0}^n f_{ij} b_j$$

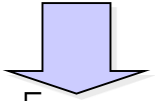


The radiosity equation

Radiosity (cont.)

- Put the equations in matrix form.

$$b_i = e_i + \rho_i \sum_{j=0}^n f_{ij} b_j$$



$$\begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} - \begin{bmatrix} \rho_0 & 0 & \cdots & 0 \\ 0 & \rho_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \rho_n \end{bmatrix} \begin{bmatrix} f_{00} & f_{01} & \cdots & f_{0n} \\ f_{10} & f_{11} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ f_{n0} & \cdots & \cdots & f_{nn} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

$\mathbf{e} = \mathbf{b} - \mathbf{R} \mathbf{F} \mathbf{b}$

The solution is

$$\mathbf{b} = [\mathbf{I} - \mathbf{R}\mathbf{F}]^{-1} \mathbf{e}$$

↩ Is it feasible?

Solving the Radiosity Equation

► Direct inverse: dimensional problem.

$$[I - RF]b = e$$

► Jacobi method

$$A = [I - RF]$$

$$Ax = c, A = D + O$$

$$e = c$$

(D : diagonal matrix, O : residual)

unknown b

$$(D + O)x = c$$

$$Dx = c - Ox$$

$$x^{t+1} = D^{-1}(c - Ox^t)$$

► Gauss-Seidal method

$$Ax = c, A = L + U$$

(L : lower triangle plus diagonal matrix, U : upper triangle matrix)

$$(L + U)x = c$$

$$Lx = c - Ux$$

$$x^{t+1} = L^{-1}(c - Ux^t)$$

Solving the Radiosity Equation

- ▶ Solving the equation by a direct method (e.g. Gaussian elimination) is infeasible.
 - ▶ \mathbf{F} is too large.

- ▶ Solving by iterative numerical methods

- ▶ Jacobi's method

$$b_i^{k+1} = \frac{1}{1 - \rho_i f_{ii}} \left(e_i + \sum_{j=1, j \neq i}^n \rho_i f_{ij} b_j^k \right)$$

- ▶ the Gauss-Seidel method

$$b_i^{k+1} = \frac{1}{1 - \rho_i f_{ii}} \left(e_i + \sum_{j=1}^{i-1} \rho_i f_{ij} b_j^{k+1} + \sum_{j=i+1}^n \rho_i f_{ij} b_j^k \right)$$

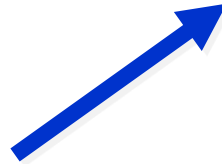
Solving the Radiosity Equation

► Jacobi's method

- need two copies of radiosity vector **B**
- doesn't always converge quickly

► the Gauss-Seidel method

- no additional copies
- it converges more quickly



```
// Make an initial guess
for all i {  $b_i = e_i$  }

// Iteratively improve guess
while( not converged )
{
    for each i
    {
        sum = 0;
        for all j except i
            sum +=  $\rho_j b_j * f_{ij}$ ;
         $b_i = e_i + \rho_i$  sum;
    }
}
```

Calculating Form Factors

- One simple way uses ray tracing & point-to-area form factors

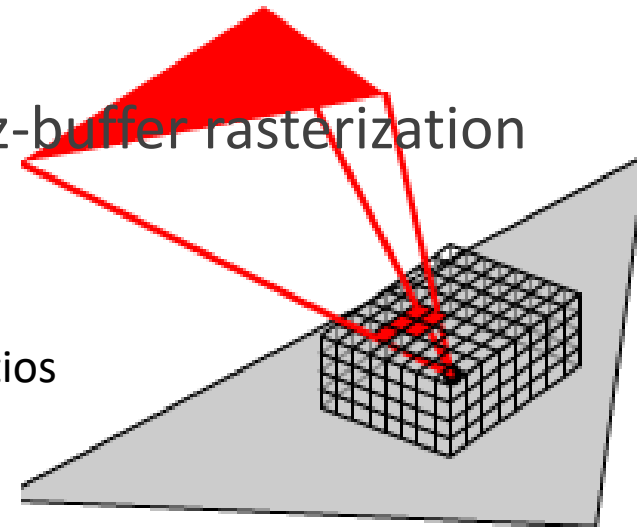
```
p = center of  $a_i$ ;  
 $f_{ij} = 0$ ;  
for k = 1 to N (separate  $a_j$  into N pieces)  
{  
    q = point on  $a_j$ ;  
    if( is_visible(p,q) )  
        // Trace ray to test visibility  
         $f_{ij} += \cos(\dots) * \cos(\dots) / (\pi * r * r) * (a_j / N)$ ;  
}
```

Hemicube Algorithm

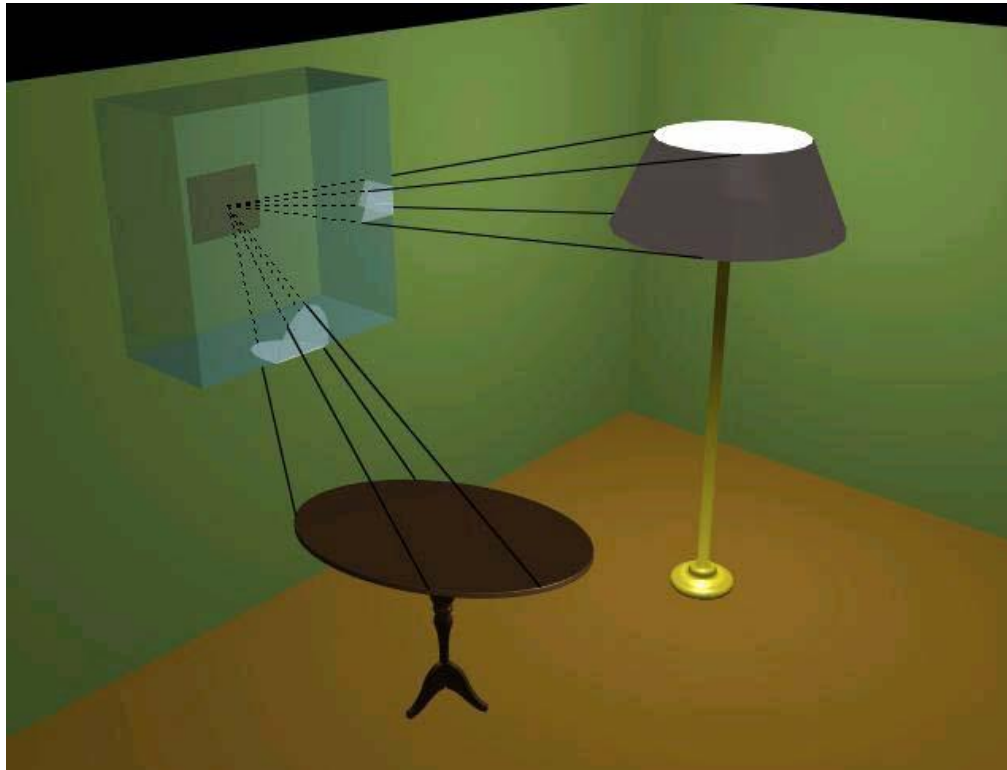
- ▶ A hemicube is constructed around the center of each patch.
- ▶ Faces of the hemicube are divided into "pixels"
- ▶ Each patch is projected (rasterized) onto the faces of the hemicube.
- ▶ Each pixel stores its **pre-computed** form factor.
- ▶ The form factor for a particular patch is just the sum of the pixels it overlaps.
- ▶ Patch occlusions are handled similar to z-buffer rasterization

In a closed scene, $\sum_{j=1}^n f_{ij} = 1$, because the radiosity-ratios affecting a patch must add up to 100%.

http://commons.wikimedia.org/wiki/File:Hemicube_Radiosity.png

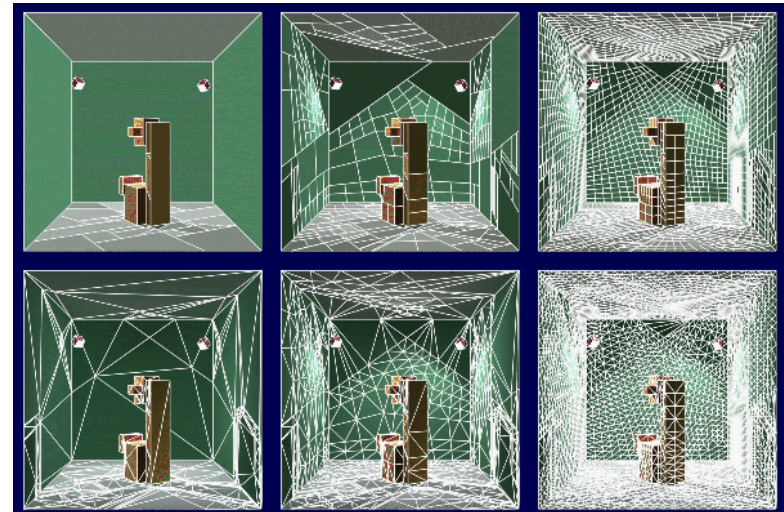
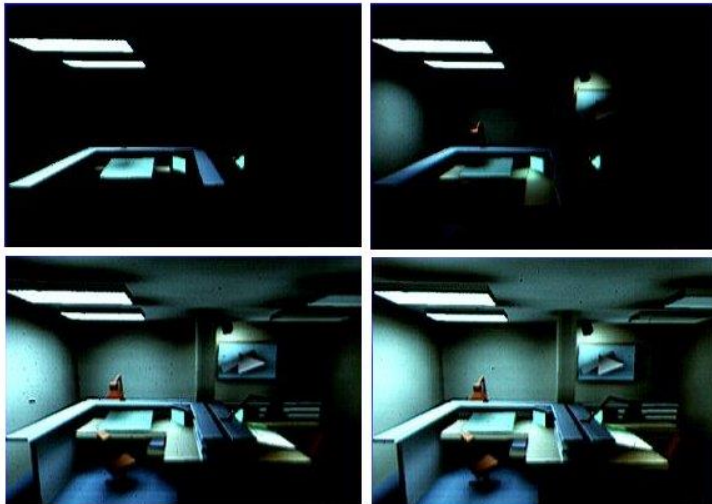
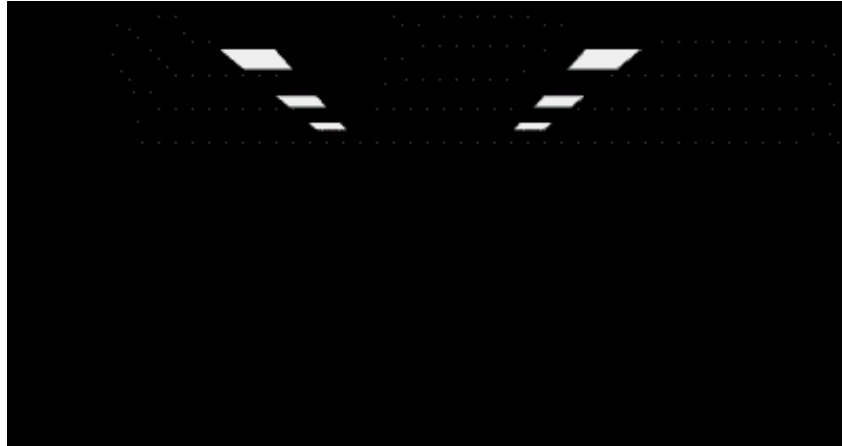


Hemicube Algorithm



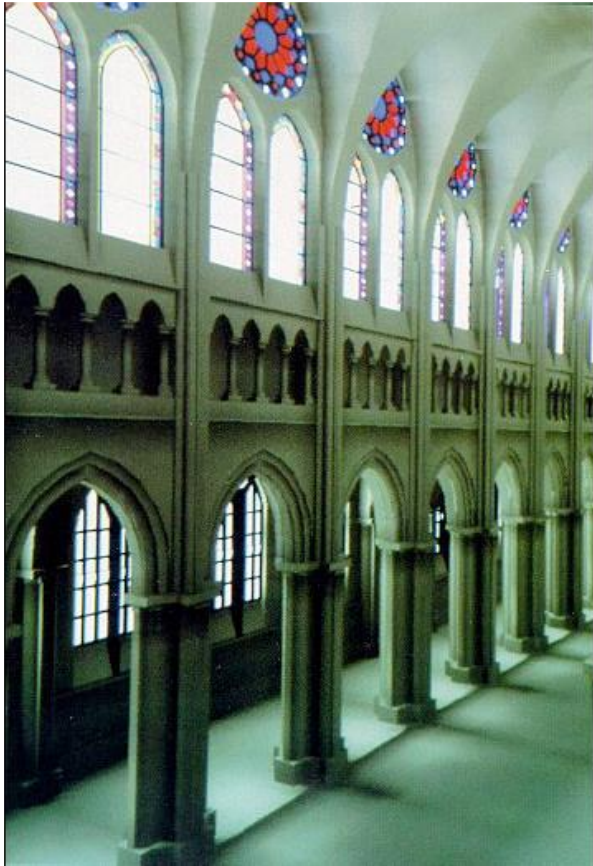
http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_2.htm

Radiosity



http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_3.htm

Radiosity Examples



the Chartres Cathedral



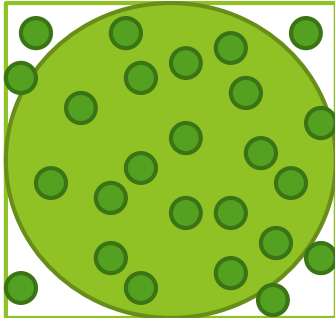
The Factory

www.graphics.cornell.edu

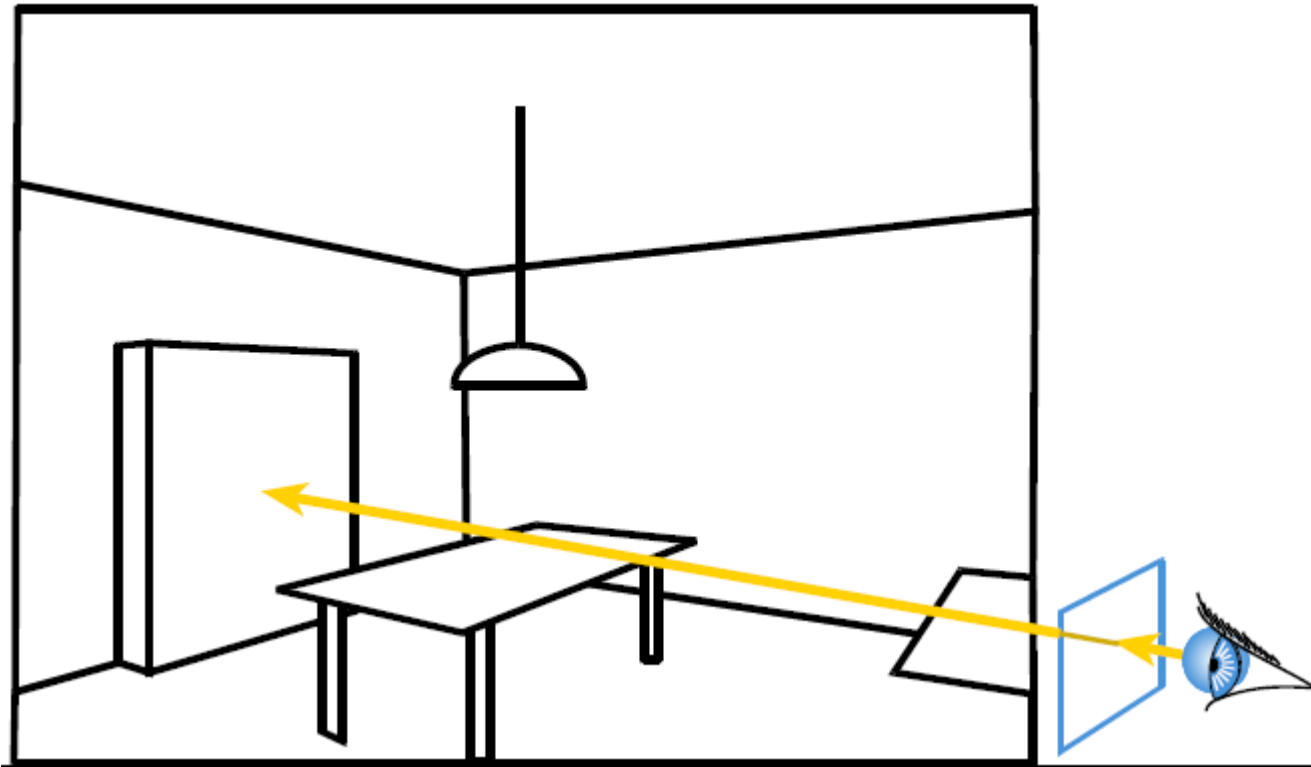
What are the limitations of Radiosity ?

Monte-Carlo computation of π

- ▶ Take a random point (x,y) in unit square
- ▶ Test if it is inside the disc $(x^2 + y^2 < 1)$
- ▶ Probability of being inside disc
= (area of unit circle) / (area of 2x2 square) = $\pi/4$
- ▶ $\pi \approx 4 * \text{number inside disc} / \text{total number}$
- ▶ The error depends on the number or trials

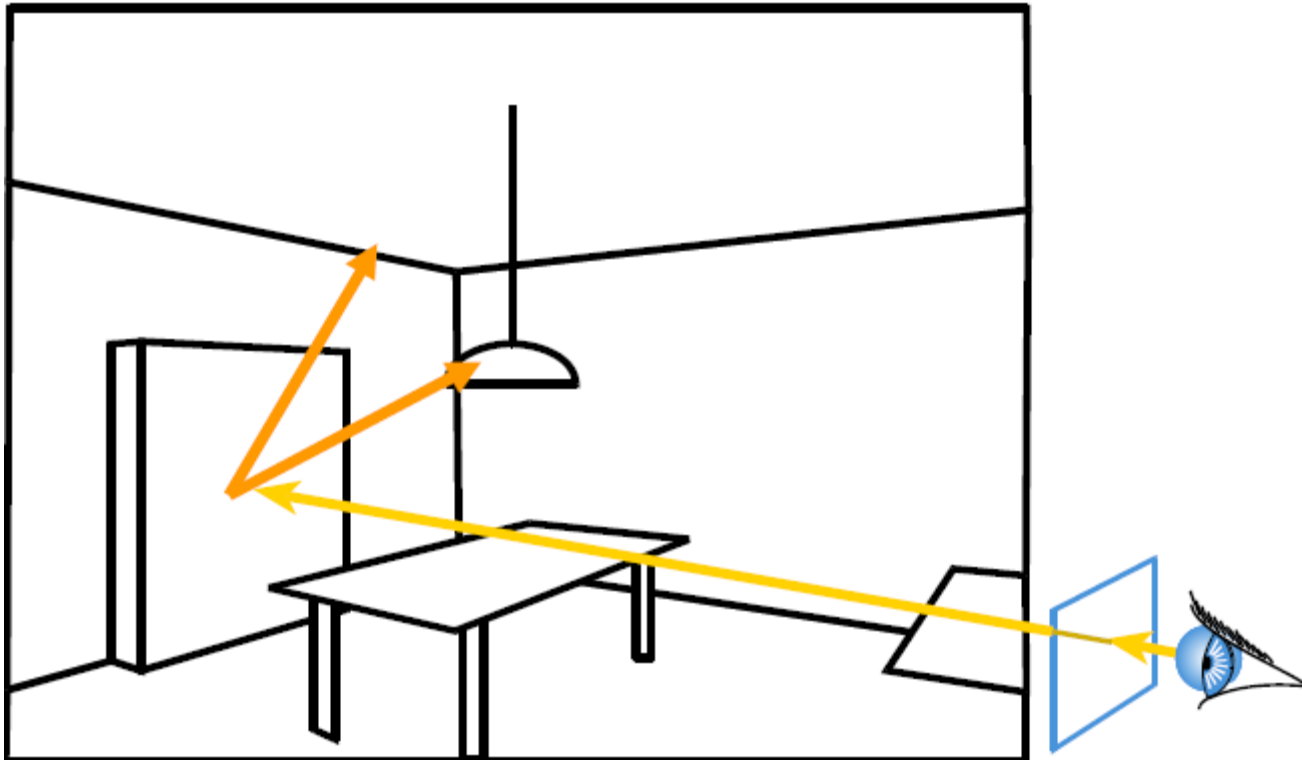


Ray casting



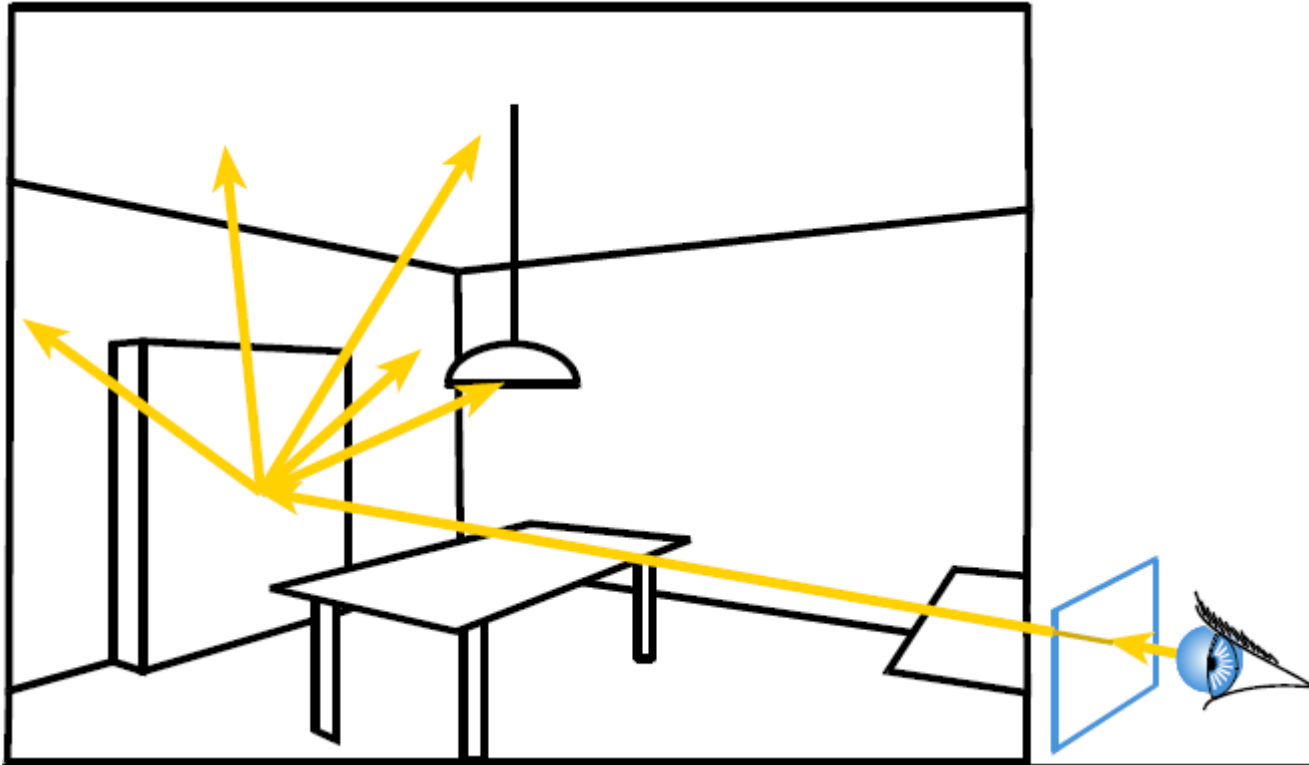
Ray tracing

- ▶ Cast a ray from the eye through each pixel
- ▶ Trace secondary rays (light, reflection, refraction)



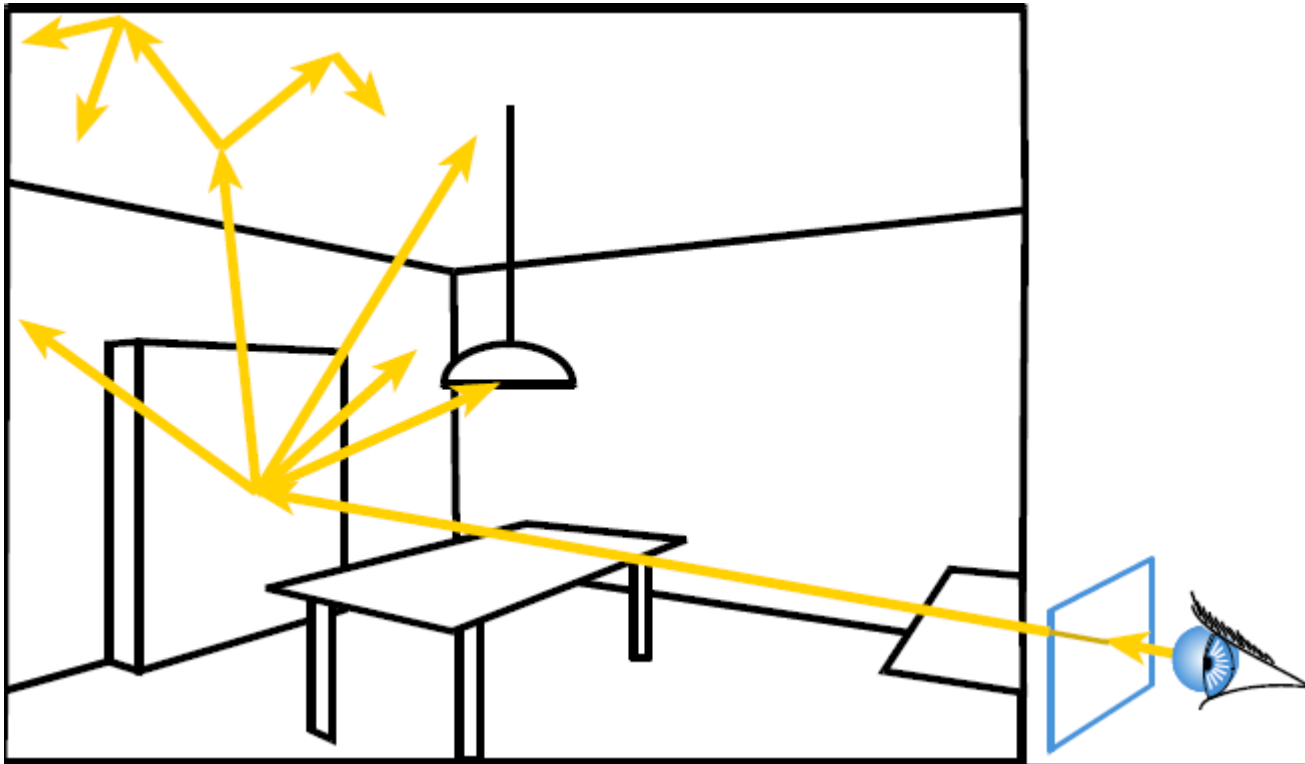
Monte-Carlo ray tracing

- ▶ Cast a ray from the eye through each pixel
- ▶ Cast random rays from the visible point
 - ▶ Accumulate radiance contribution



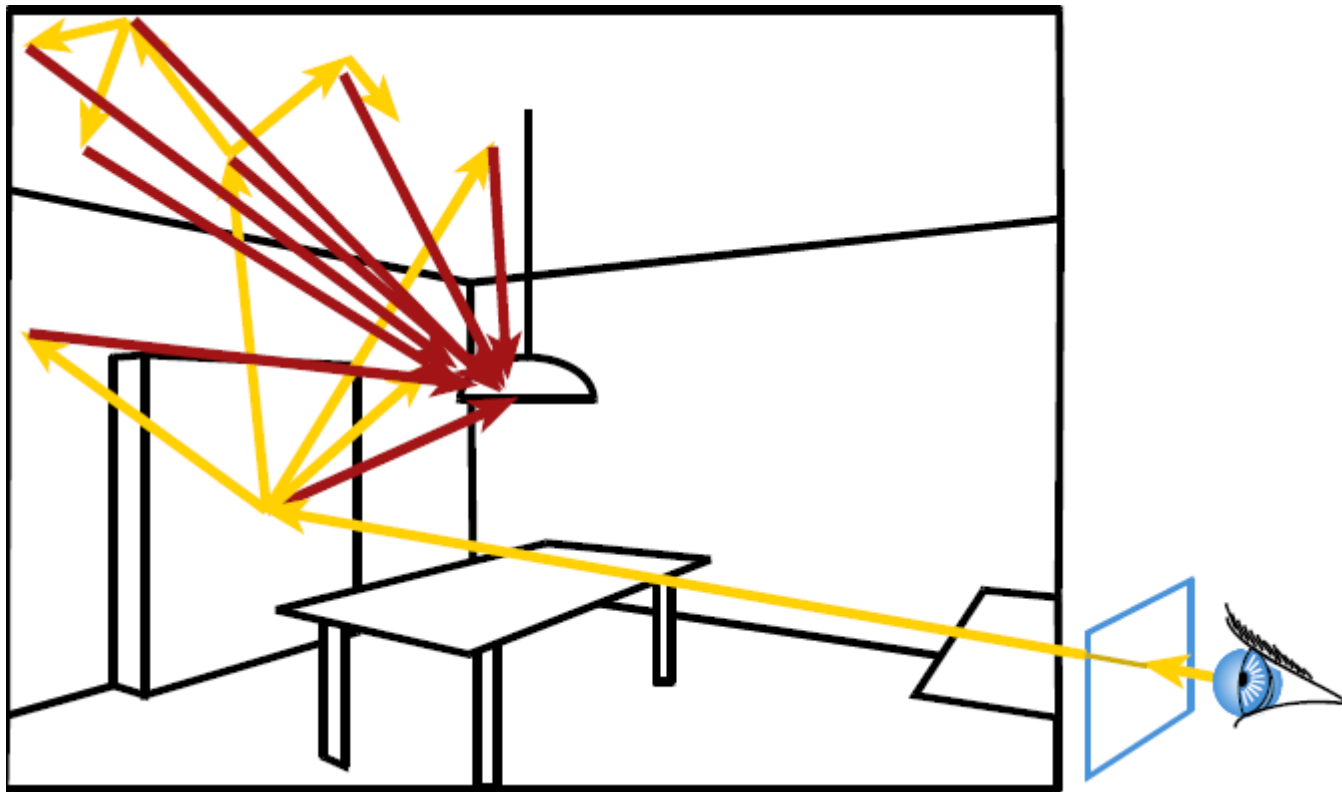
Monte-Carlo ray tracing

► Recursion



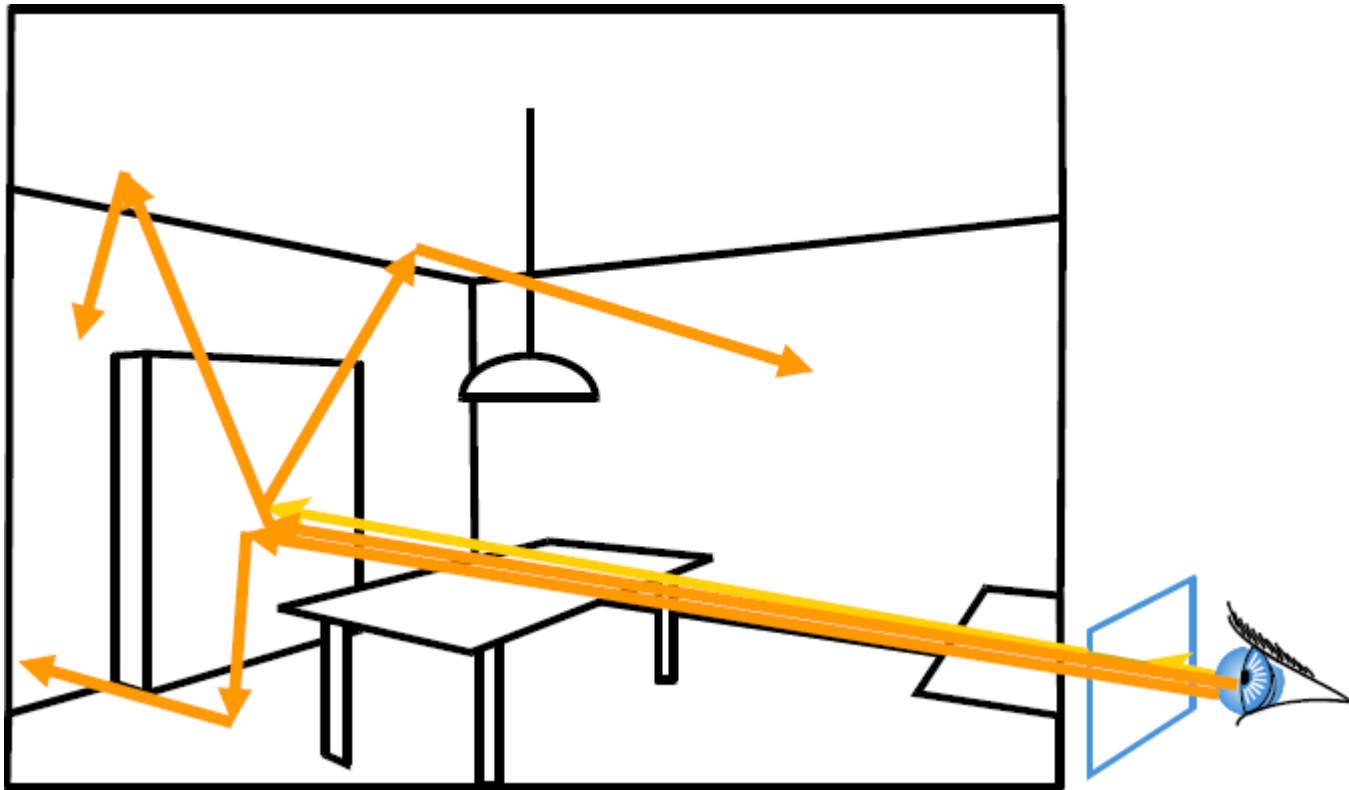
Monte-Carlo ray tracing

- Systematically sample primary light

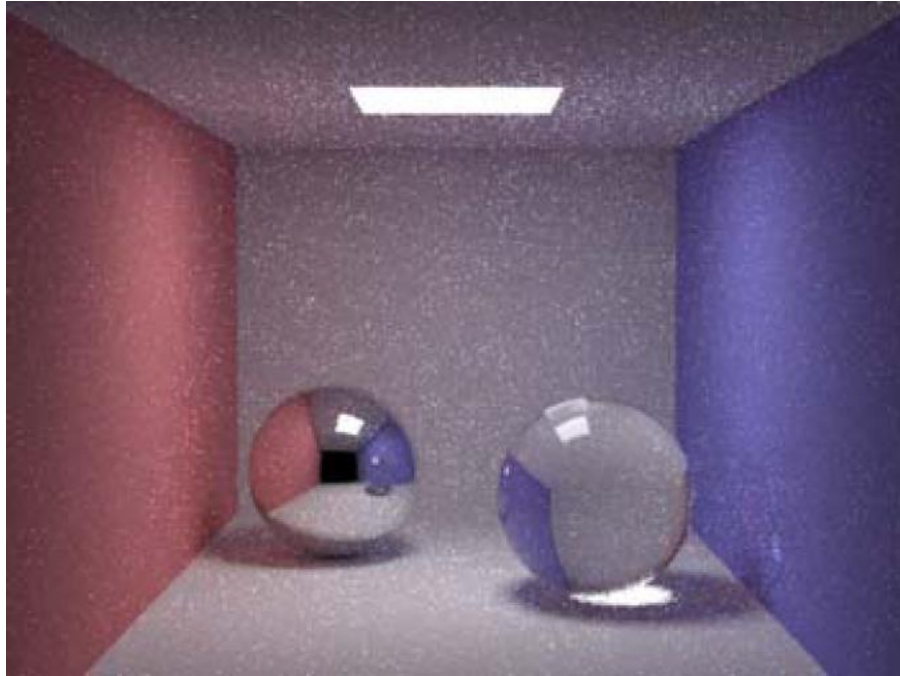


Monte Carlo path tracing

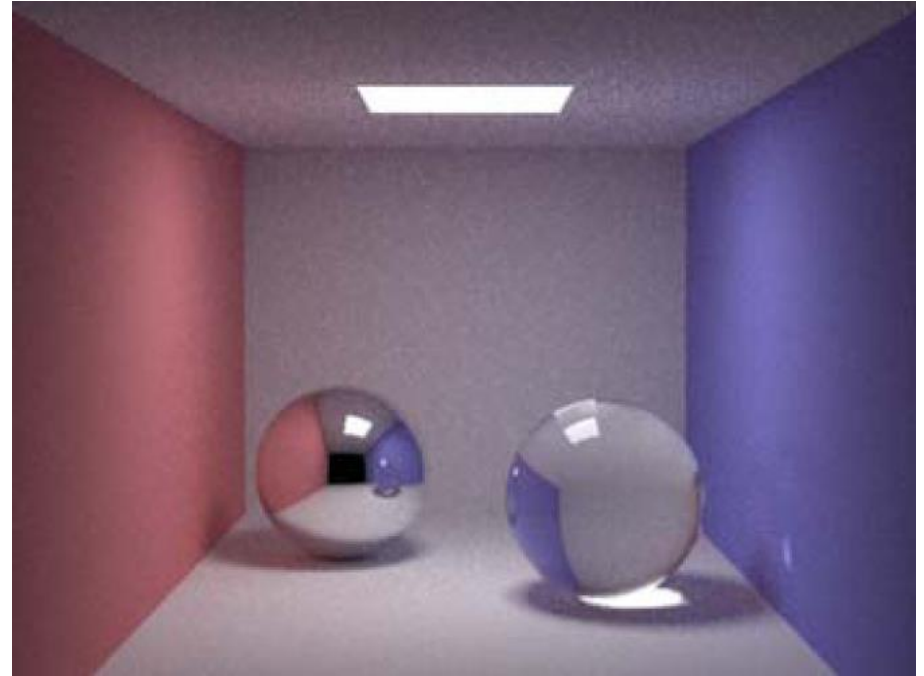
- ▶ Trace only one secondary ray per recursion
- ▶ But send many primary rays per pixel



Monte Carlo path tracing



10 paths/pixel

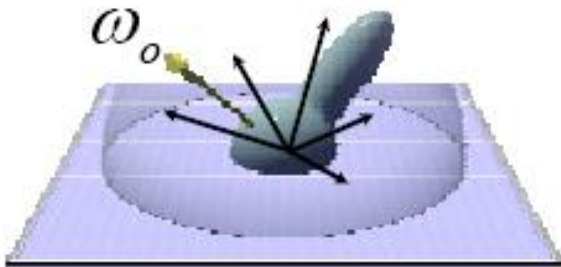


100 paths/pixel

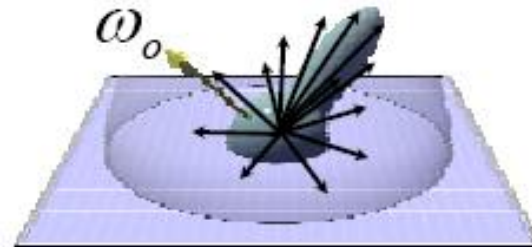
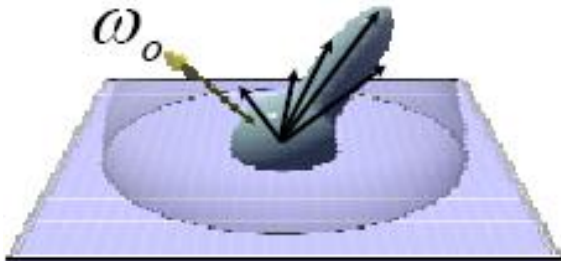
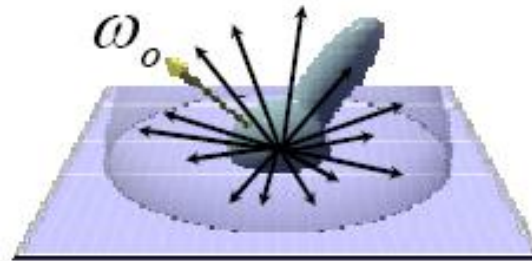
Importance sampling

Sampling a BRDF

5 Samples/Pixel



25 Samples/Pixel



Problem of path tracing



1000 paths/pixel

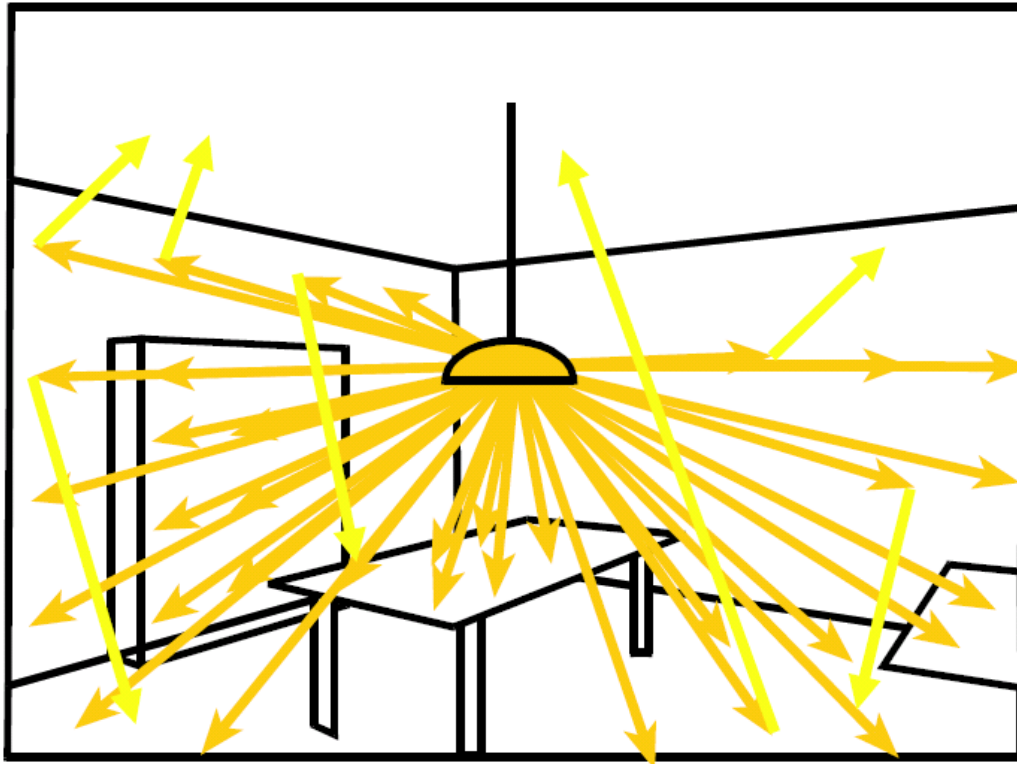
Figure by H.W. Jensen

Photon mapping

- ▶ Bi-directional paths
 - ▶ Construct paths not only from the eye, but also from the light sources
- ▶ Caching
 - ▶ Cache photons distributed along paths from the light sources
- ▶ Interpolation
 - ▶ Interpolate radiance from cached photons

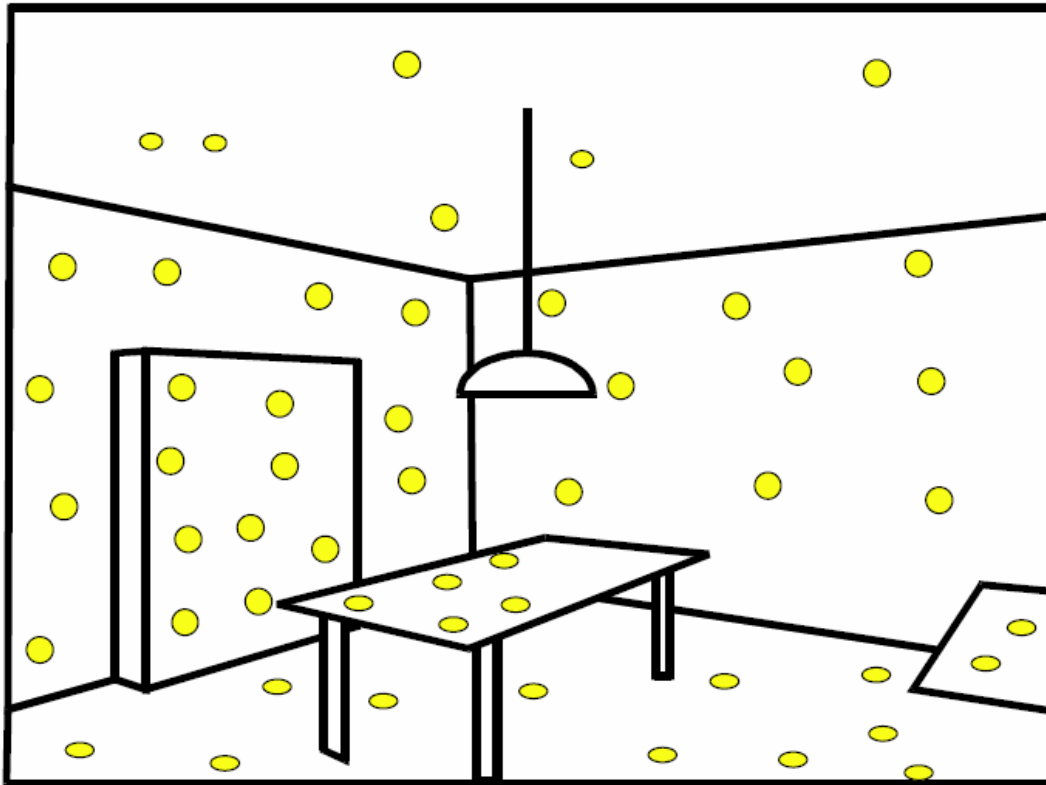
Photon mapping

► Photon emission and transport



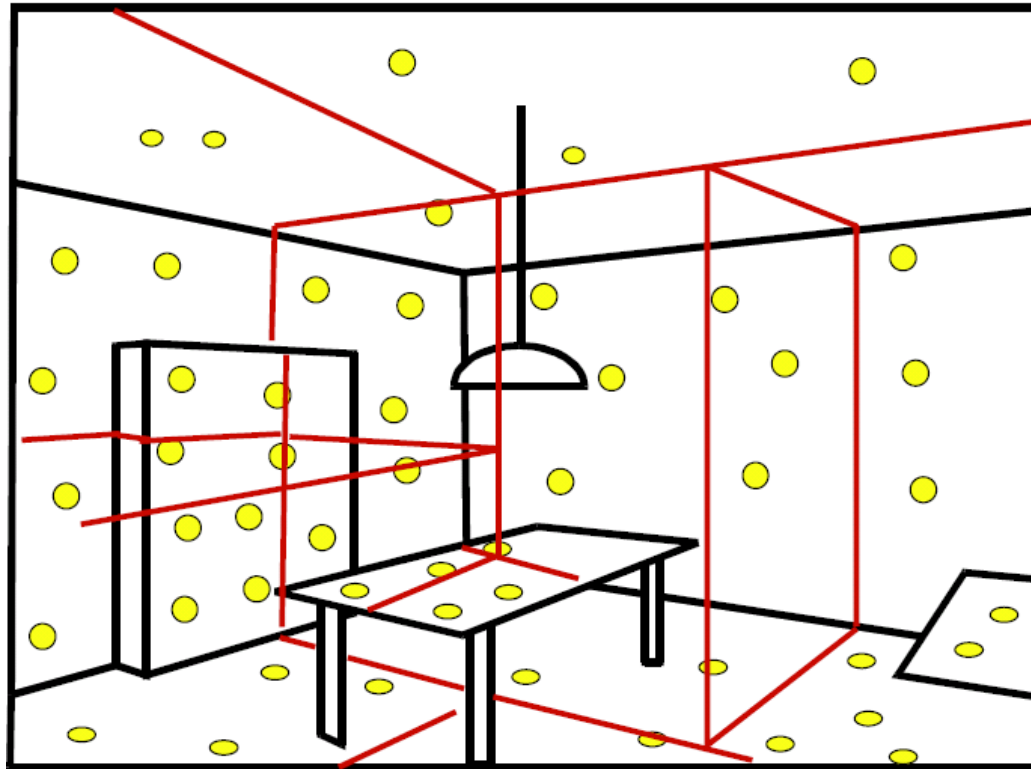
Photon mapping

► Photon caching



Photon mapping

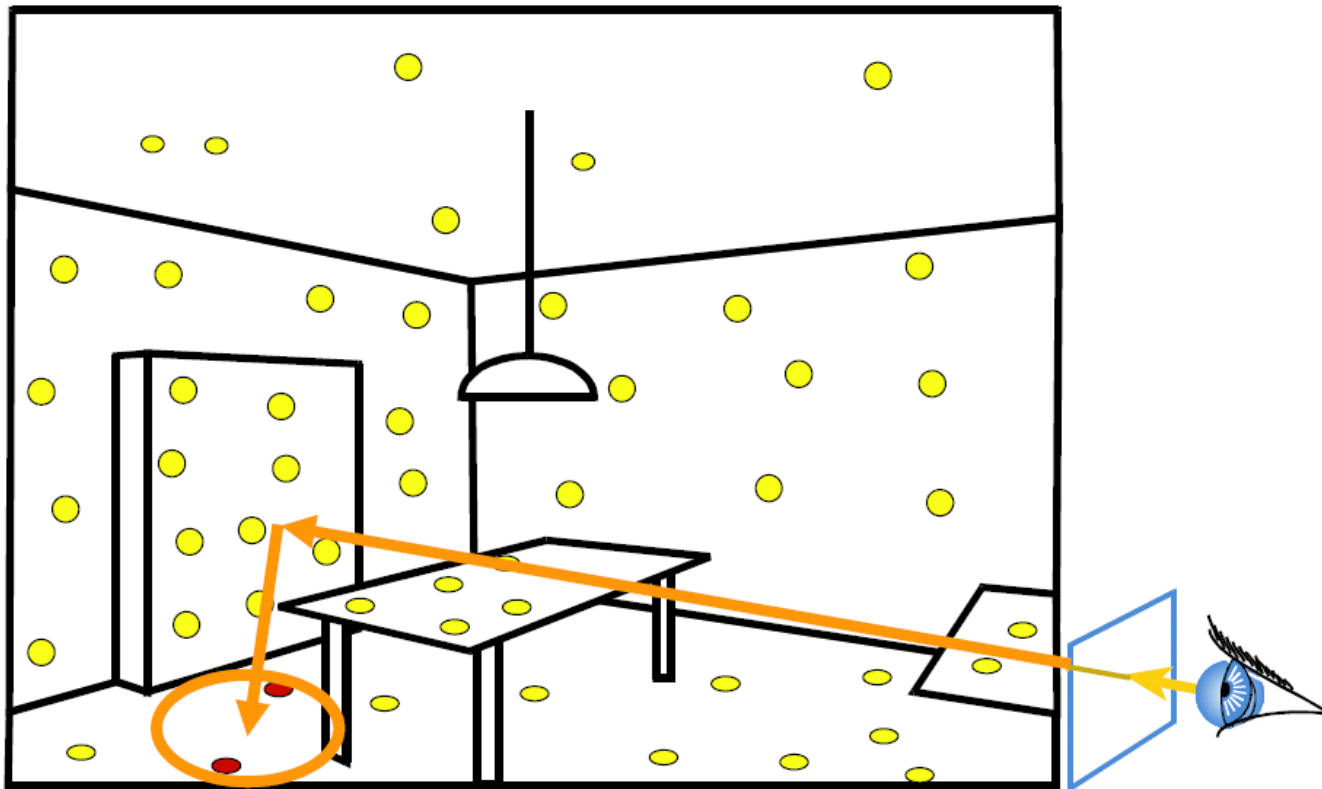
- Spatial data structure for fast access



[Cutler, Durand]

Photon mapping

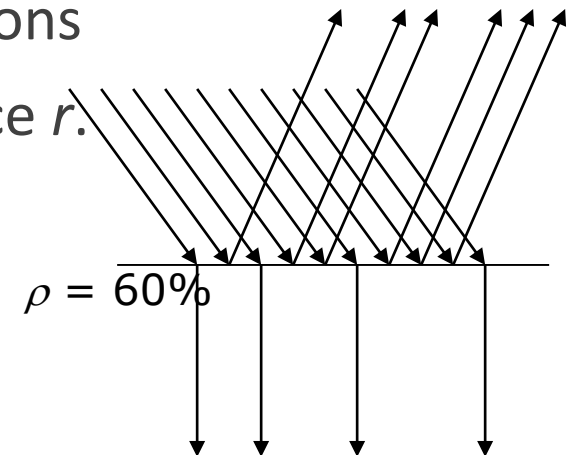
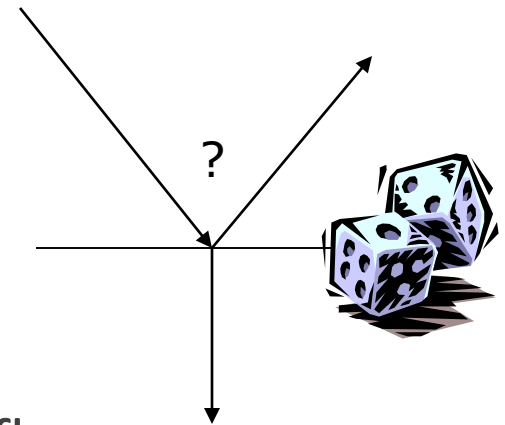
► Radiance estimation



[Cutler, Durand]

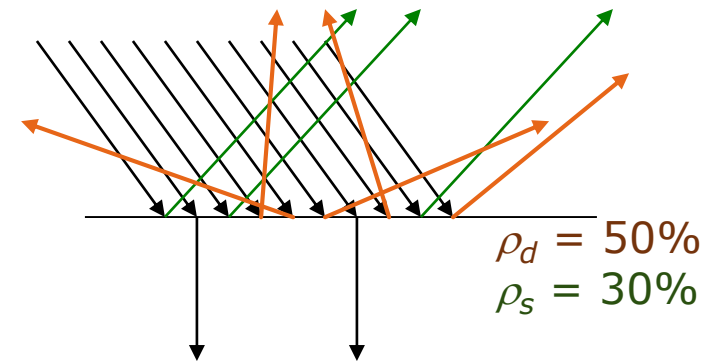
Russian Roulette

- ▶ Arvo & Kirk, S90
- ▶ Reflected flux only a fraction of incident flux
- ▶ After several reflections, spending a lot of time keeping track of very little flux
- ▶ Instead, completely absorb some photons and completely reflect others at full power
- ▶ Spend time tracing fewer full power photons
- ▶ Probability of reflectance is the reflectance r .
- ▶ Probability of absorption is $1 - r$.



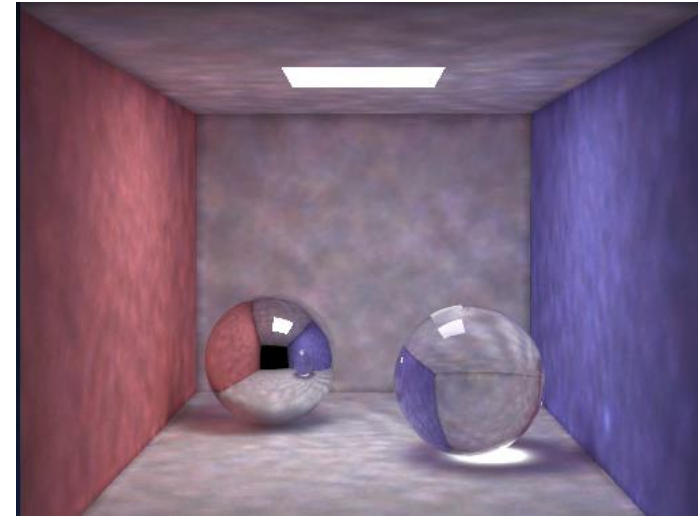
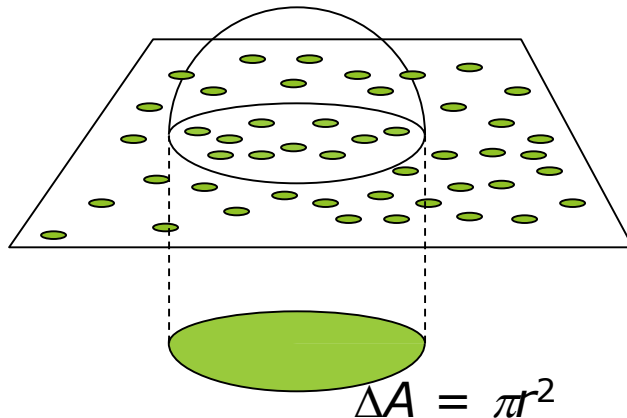
Distribution

- ▶ Surfaces have specular and diffuse components
 - ▶ r_d – diffuse reflectance
 - ▶ r_s – specular reflectance
 - ▶ $r_d + r_s < 1$ (conservation of energy)
- ▶ Let z be a uniform random value from 0 to 1
- ▶ If $z < r_d$ then reflect diffuse
- ▶ Else if $z < r_d + r_s$ then reflect specular
- ▶ Otherwise absorb

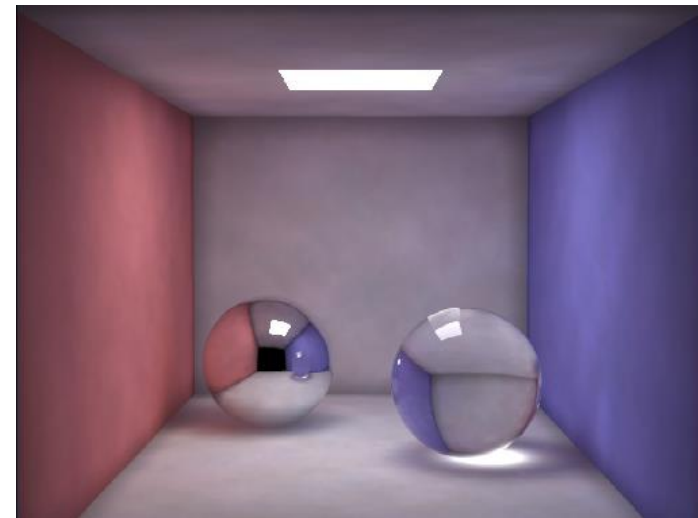


How many photons?

- ▶ How big is the disk radius r ?
- ▶ Large enough that the disk surrounds the n nearest photons.
- ▶ The number of photons used for a radiance estimate n is usually between 50 and 500.



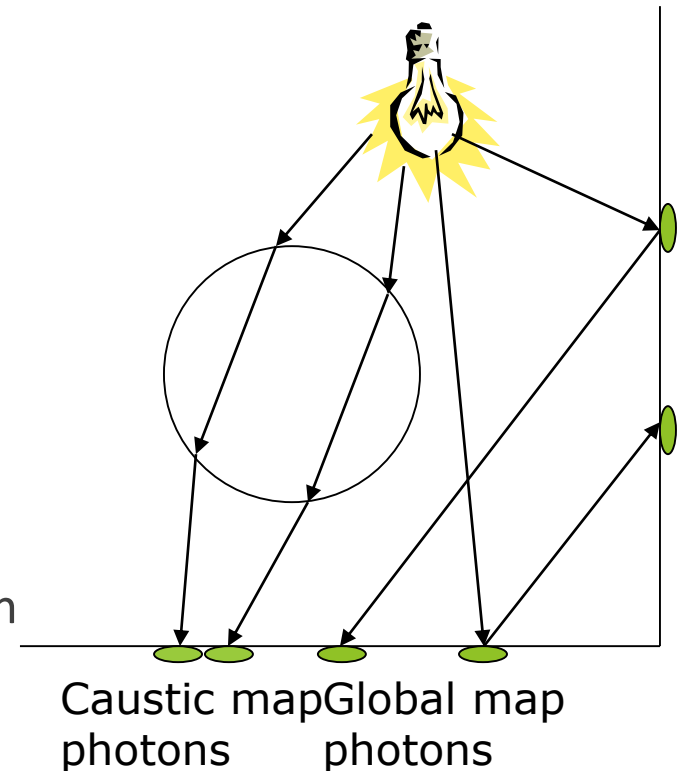
Radiance estimate using 50 photons



Radiance estimate using 500 photons

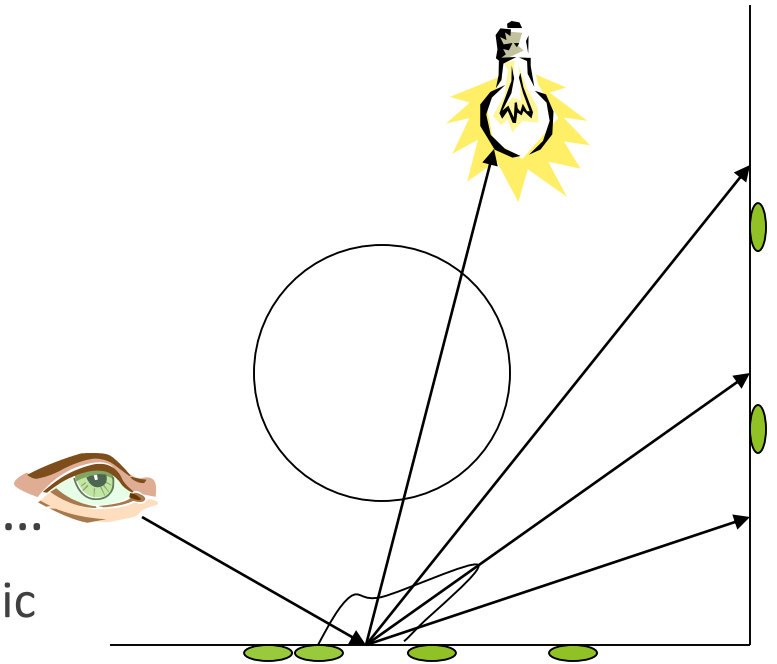
Multiple Photon Maps

- ▶ Global $L(S|D)*D$ photon map
 - ▶ Photon sticks to diffuse surface *and* bounces to next surface (if it survives Russian roulette)
 - ▶ Photons don't stick to specular surfaces
- ▶ Caustic $LSS*D$ photon map
 - ▶ High resolution
 - ▶ Light source usually emits photons only in directions that hit the thing creating the caustic



Rendering

- ▶ Rendered by glossy-surface distributed ray tracing
- ▶ When ray hits first diffuse surface...
 - ▶ Compute reflected radiance of caustic map photons
 - ▶ Ignore global map photons
 - ▶ Importance sample BRDF f_r as usual
 - ▶ Use global photon map to importance sample incident radiance function L_i
 - ▶ Evaluate reflectance integral by casting rays and accumulating radiances from global photon map



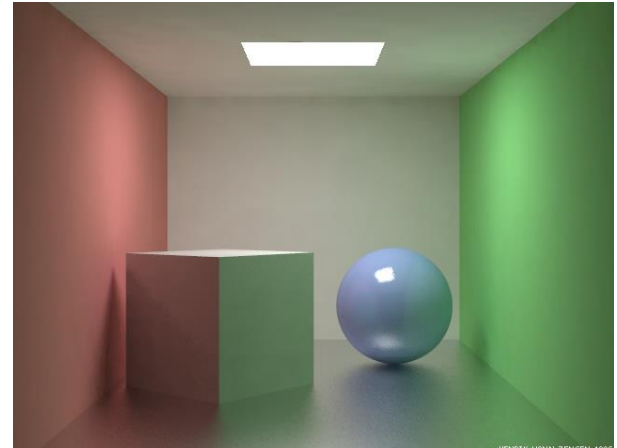
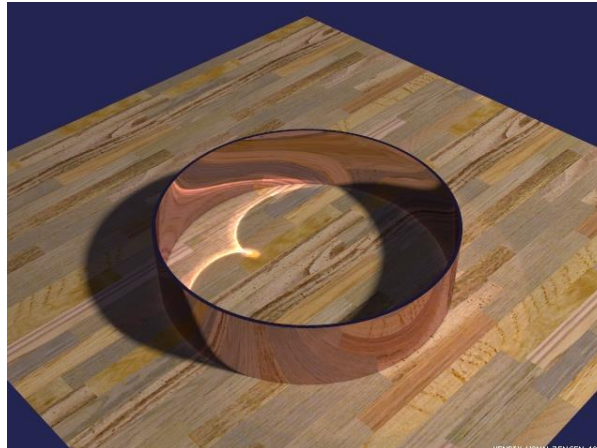
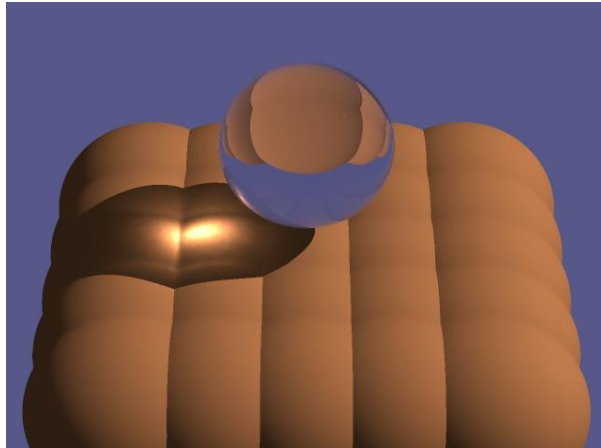
First diffuse intersection.
Return radiance of caustic
map photons here, but
ignore global map photons

Use global
map photons
to return
radiance
when
evaluating L_i
at first
diffuse
intersection.



Photon mapping

- ▶ [Jensen EGRW 95, 96]
 - ▶ The lower-left scene below contains glossy surfaces, and was rendered in 50 minutes using photon mapping. The same scene took 6 hours for render with Radiance that used radiosity for diffuse reflection and path tracing for glossy reflection.



The End of this Chapter