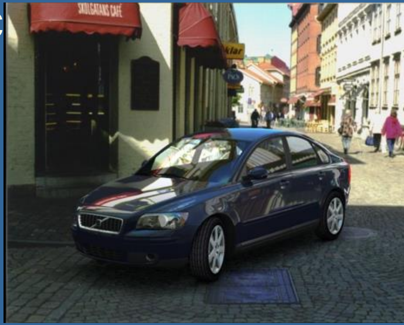
An introduction to Global Illumination

Tomas Akenine-Möller
Modified by Ulf Assarsson
Department of Computer Engineering
Chalmers University of Technology

DAT295/DIT221 Advanced Computer Graphics - Seminar Course, 7.5p

- If you are interested, register to that course
- http://www.cse.chalmers.se/edu/course/TDA362/Advanced Computer Graphics/
- ~13 seminars in total, sp4
- Project (no exam)
  - Self or in groups
- Project examples include:
  - GPU ray tracing (Vulkan), AI denoising
  - realistic explosions, clouds, smoke, procedural textures
  - fractal mountains, CUDA program, Spherical Harmonic Displacement mapping, Collision detection
  - 3D Game
  - real-time ray tracer, enhanced path tracing.
  - or anything else you can come up with…



GFX Companies Gothenburg

3D software development:
Rapid Images
EA Frostbite (filial i Göteborg)
TTK Games (Gbg + Sthlm)
Epic Games
NVIDIA – Lund/Göteborg
Smart Eye AB,
EON Reality,
Spark Vision
MindArk
Mentice
Vizendo
Surgical Science
Combitech
Fraunhofer (Chalmers Teknikpark)
RD&T Technology
Qualisys
Volvo Trucks, Volvo Cars
Zenseact
Berge Consulting / Berge Group

And many more that I have forgotten now…

For graphics artists:
Rapid Images
AFRY
Zoink games
Industriromantik
Stark Film
Edit House
Bobby Works
Filmgate
Ord och bild
Magoo 3D Studios
Tenjin Visual
Silverbullet Film
Tengbom
MFX – www.mfx.se

Non-Gothenburg

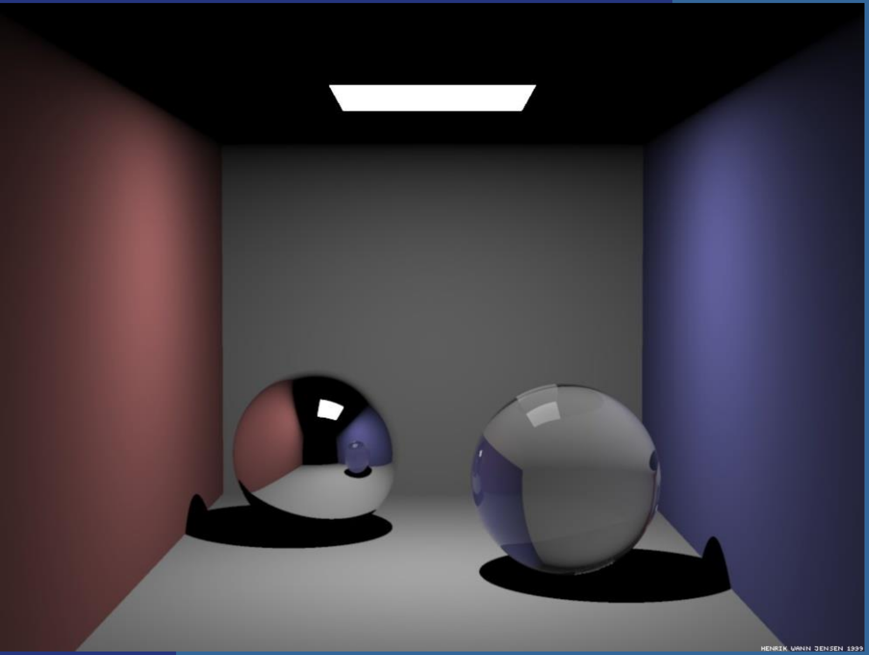
Game Studios:
Avalanche studios (Sthlm)
DICE / EA (Sthlm)
Massive (Malmö)
Frostbite (Sthlm)

Architects
Arcitec – (Sthlm)– visualization of buildings for architects

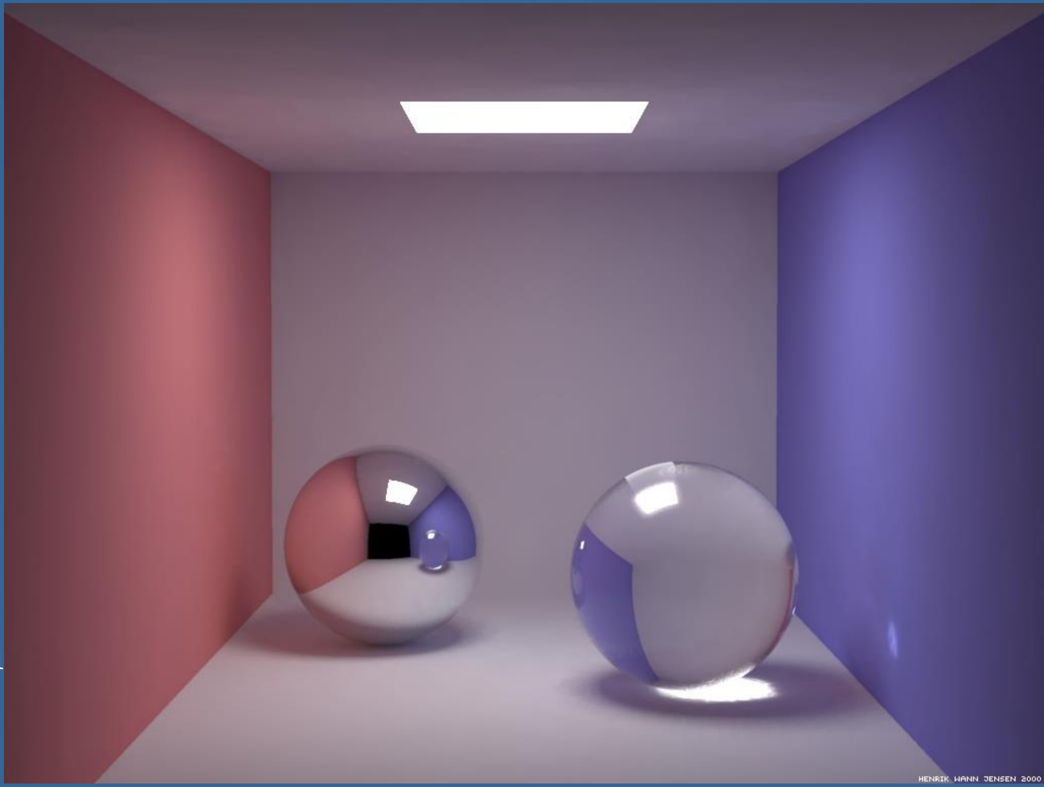
Architects, graphics artists:
White
Wingårdhs
Volvo Personvagnar
Semcon
Ramböll
Zynka
CAP AB

Isn't classic ray tracing enough?

Effects to note in Global Illumination image:
1) Indirect lighting (light reaches the roof)
2) Color bleeding (example: roof is red near red wall)
3) Materials have no ambient component
4) Caustics (concentration of refracted light through glass ball)
5) Soft shadows (light source has area)
Others: volumetric effects, e.g., participating media



Whitted Ray tracing
(reflections, refractions, shadows)



Which are
the differences?

Global
Illumination

Images courtesy of Henrik Wann Jensen

Global Illumination

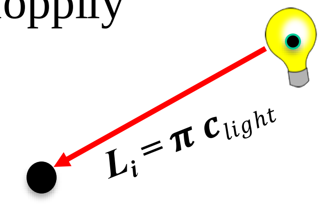
- The goal: **follow** all **photon/ray** bounces through a scene, in order to render images with all kinds of light paths.
- This will give incredibly realistic images
- This lecture will treat:
  - Background:
    - radiance
    - *the rendering equation*
  - How to solve the rendering equation by Monte Carlo ray tracing:
    - Path tracing
    - Bidirectional Path tracing
    - Adding denoising - Final Gathering or AI denoising
    - Photon mapping
- Great book on global illumination:
  - Pharr, Humphreys, Physically Based Rendering, 2010
    - With source code.

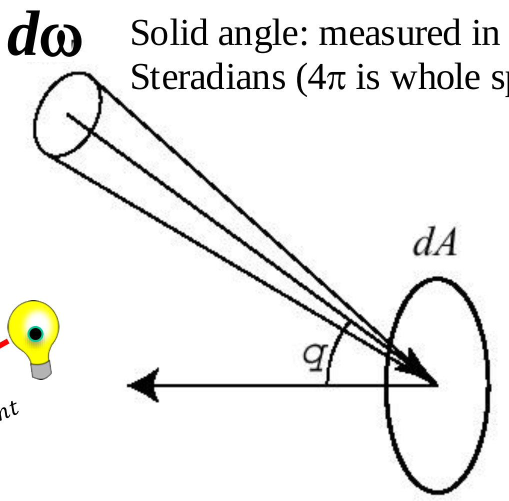
Radiance

- In graphics, we typically use rgb-colors $\boldsymbol{c} = (c_r, c_g, c_b)$ and mean the intensity or ***radiance*** for the red, green, and blue light.
- Radiance, L : a radiometric term. What we store in a pixel is the radiance towards the eye: a tripplet $\boldsymbol{L} = (L_r, L_g, L_b)$
  - Radiance = the amount of electromagnetic radiation leaving or arriving at a point on a surface (per unit solid angle per unit projected area)
- $L_o(\mathbf{x},\boldsymbol{\omega})$ is often five-dimensional (or 6, including wavelength):
  - Position (3)
  - Direction (2) – horizontal + vertical angle
- Radiance is ”power per unit projected area per unit solid angle”

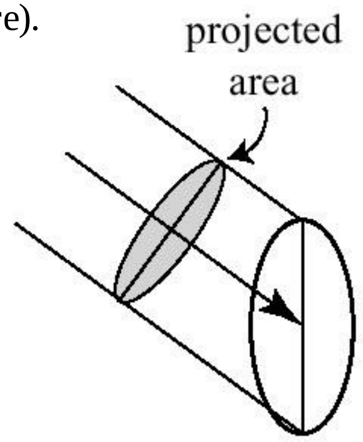
Radiance from a specific *direction* uses differentials, so the cone of the solid angle becomes an infinitesmally thin ray.

Hence, in graphics we often sloppily talk about the radiance from a direction to a surface point





Solid angle: measured in Steradians (4π is whole sphere).



Background: The rendering equation

- Paper by Kajiya, 1986 (see course website).
- Is the basis for all rendering, but especially for global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$ (slightly different terminology than Kajiya)
  - outgoing=emitted+reflected radiance
  - $\mathbf{x}$ is position on surface, ω is direction vector
- Extend the last term $L_r(\mathbf{x},\omega)$

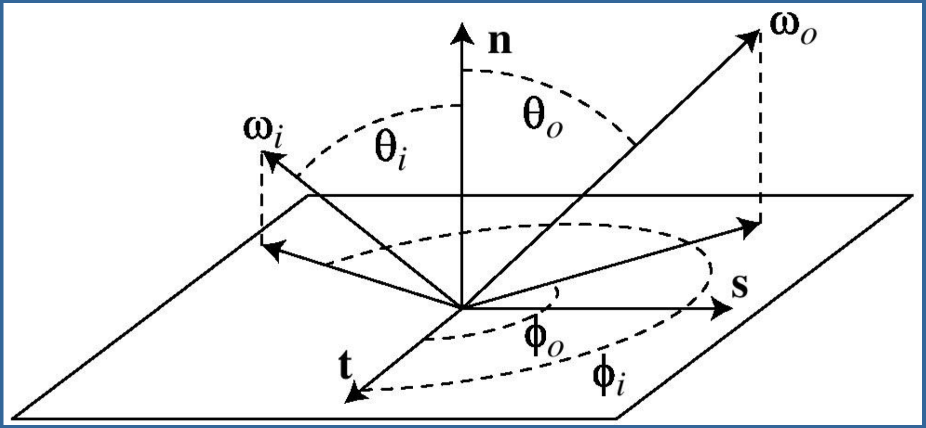
$$L_o = L_e + \int_\Omega f_r(\mathbf{x},\omega,\omega')L_i(\mathbf{x},\omega')(\omega'\cdot\mathbf{n})\,d\omega'$$

- f_r is the BRDF (next slide), ω' is incoming direction, $\mathbf{n}$ is normal at point $\mathbf{x}$, Ω is hemisphere ”around” $\mathbf{x}$ and $\mathbf{n}$, L_i is incoming radiance

Background: Briefly about BRDFs

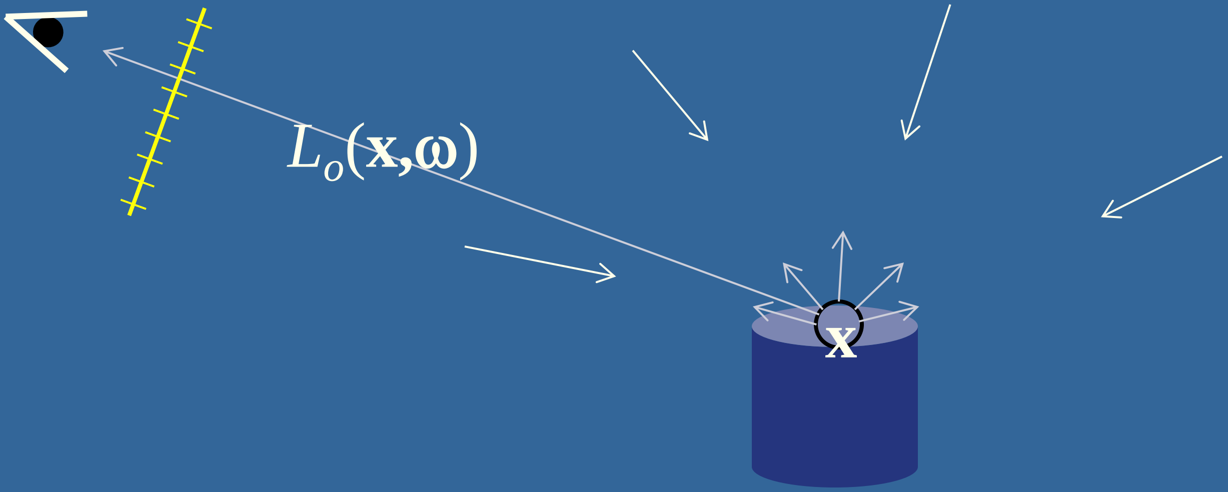
- Bidirectional Reflection Distribution Function
- A more accurate description of material properties
- What it describes:
  - How much of the incoming radiance L_i from a given direction ω_i that will leave in a given outgoing direction ω_o.
  - It is wavelength and polarization dependent.

- i is incoming direction
- o is outgoing direction
- Many different ways to get BRDF:s
  - Measurement
  - Models:
    - Simple: amb+diff+spec
    - Physically-based: metalness (vs dielectric), shininess, Fresnel, base color



Radiance/strålning

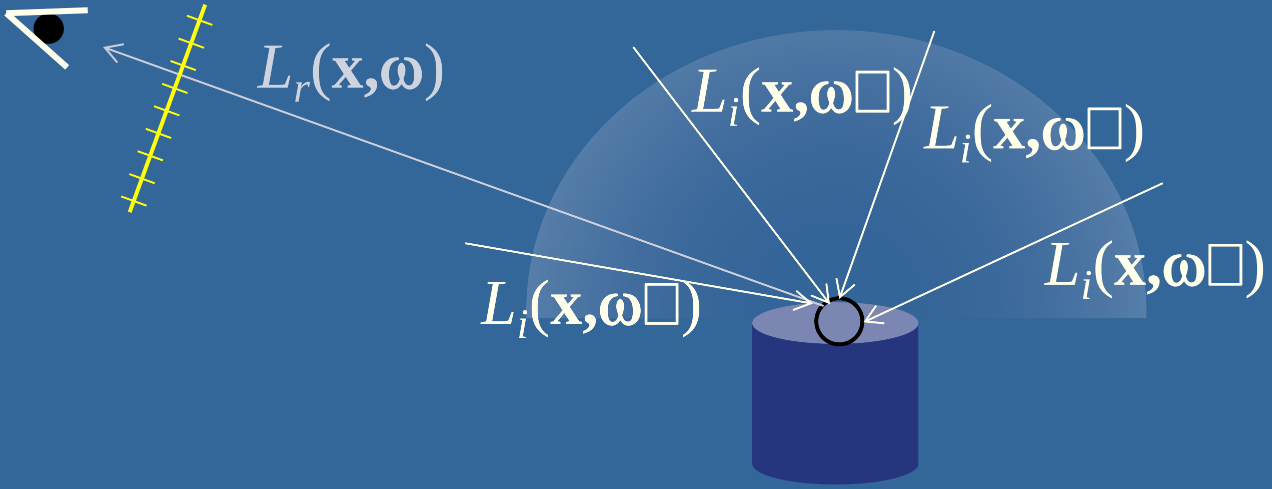
- Radiance, L : a radiometric term. What we store in a pixel is the radiance towards the eye
  - the amount of electromagnetic radiation leaving or arriving at a point on a surface



- L_o= outgoing radiation from a point to a certain direction
- Radiation = color and its intensity, i.e., rbg-value
- $\mathbf{x}$ = x,y,z-position in space
- ω = outgoing direction

The rendering equation - Summary

- Paper by Kajiya, 1986.
- Is the basis for all global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$
  - outgoing=emitted+reflected radiance


$L_r(\mathbf{x},\omega)$
$L_i(\mathbf{x},\omega□)$
$L_i(\mathbf{x},\omega□)$
$L_i(\mathbf{x},\omega□)$
$L_i(\mathbf{x},\omega□)$

Integrate over all incoming directions **ω**□to get how much radiance is reflected in outgoing direction **ω**.

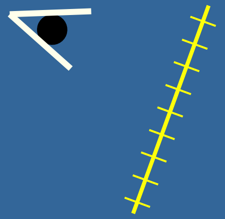
$$L_o = L_e + \int_{\Omega} \underline{f_r(\mathbf{x},\omega,\omega')L_i(\mathbf{x},\omega')(\omega'\cdot\mathbf{n})d\omega'}$$

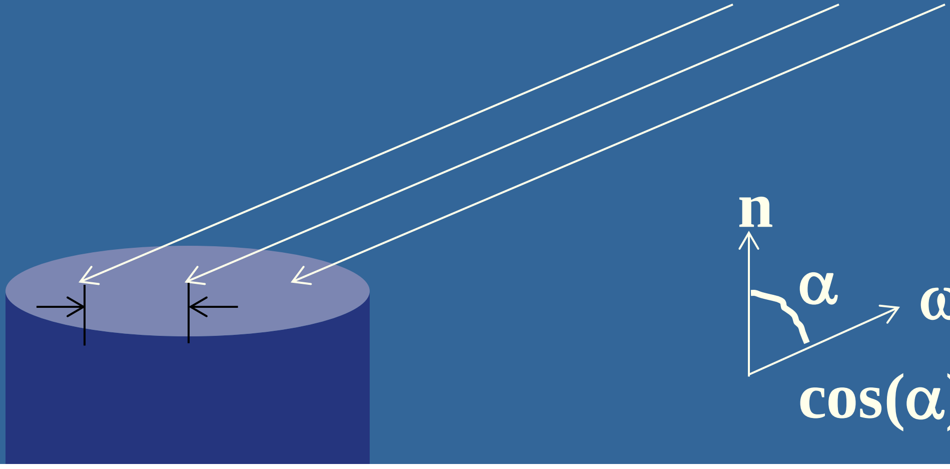
- f_r is the BRDF, ω' is incoming direction, $\mathbf{n}$ is normal at point $\mathbf{x}$, Ω is hemisphere "around" $\mathbf{x}$ and $\mathbf{n}$, L_i is incoming radiance

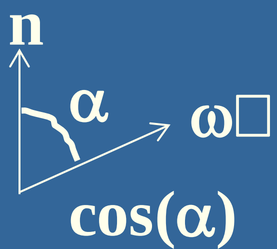
The rendering equation

Scale incoming radiance with cosine of the incoming angle

- Paper by Kajiya, 1986.
- Is the basis for all global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$
  - outgoing=emitted+reflected radiance





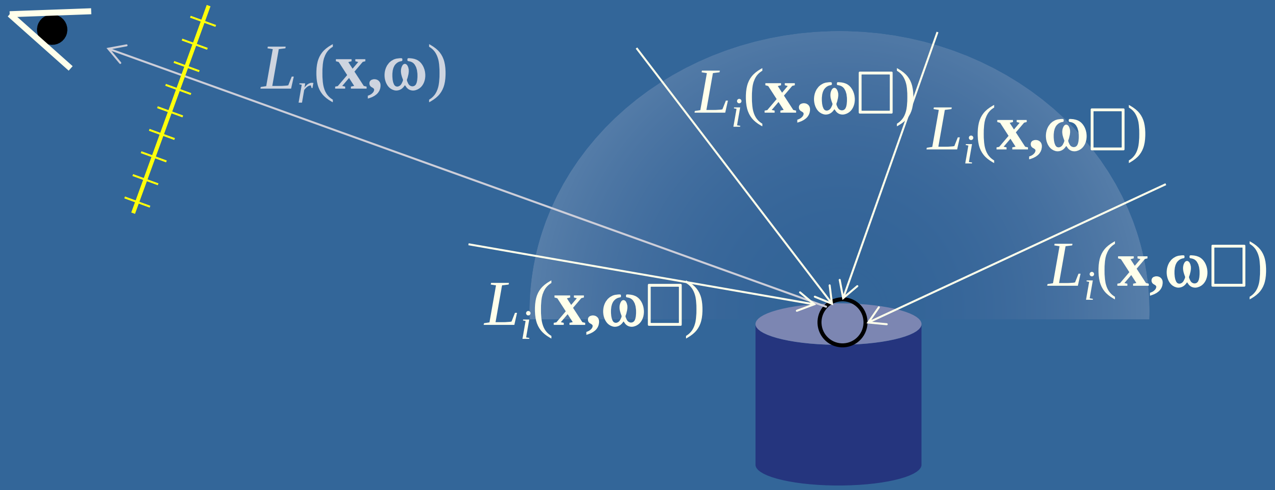


$$L_o = L_e + \int_{\Omega} f_r(\mathbf{x},\omega,\omega')L_i(\mathbf{x},\omega')\underline{(\omega'\cdot\mathbf{n})}d\omega'$$

- f_r is the BRDF, ω' is incoming direction, $\mathbf{n}$ is normal at point $\mathbf{x}$, Ω is hemisphere "around" $\mathbf{x}$ and $\mathbf{n}$, L_i is incoming radiance

The rendering equation

- Paper by Kajiya, 1986.
- Is the basis for all global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$
  - outgoing=emitted+reflected radiance



$L_r(\mathbf{x},\omega)$
$L_i(\mathbf{x},\omega□)$
$L_i(\mathbf{x},\omega□)$
$L_i(\mathbf{x},\omega□)$
$L_i(\mathbf{x},\omega□)$

$$L_o = L_e + \int_\Omega f_r(\mathbf{x},\omega,\omega')\underline{L_i(\mathbf{x},\omega')}(\omega'\cdot\mathbf{n})d\omega'$$

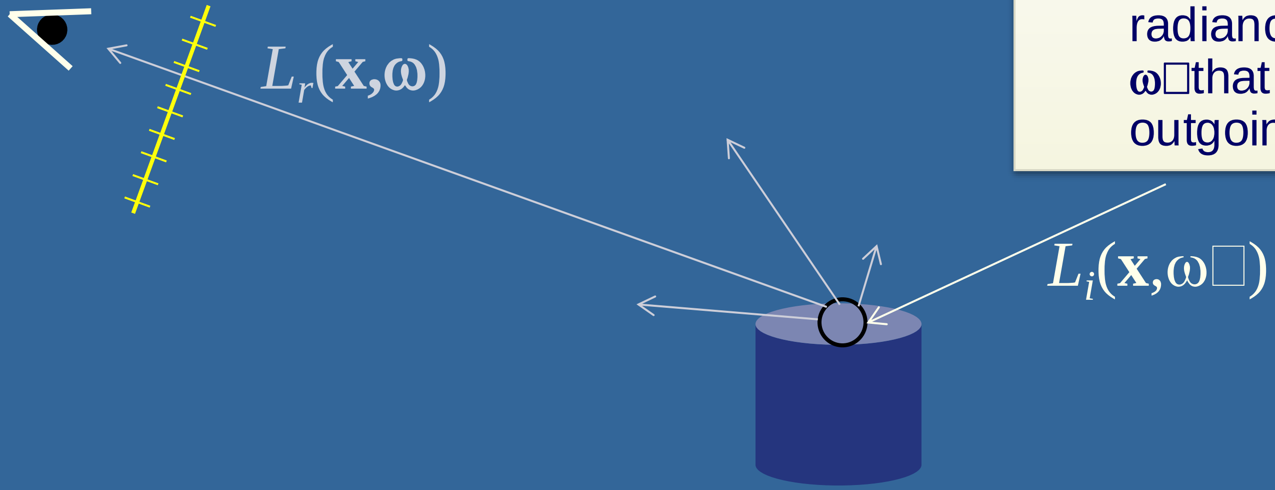
- f_r is the BRDF, ω' is incoming direction, $\mathbf{n}$ is normal at point $\mathbf{x}$, Ω is hemisphere ”around” $\mathbf{x}$ and $\mathbf{n}$, L_i is incoming radiance

The rendering equation

BRDF = Bidirectional Reflection Distribution Function

- Paper by Kajiya, 1986.
- Is the basis for all global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$
  - outgoing=emitted+reflected radiance

BRDF:
$f_r(\mathbf{x}, \omega, \omega□) =$
”How much of incoming radiance, L_i, from direction ω□that leaves in an outgoing direction ω□

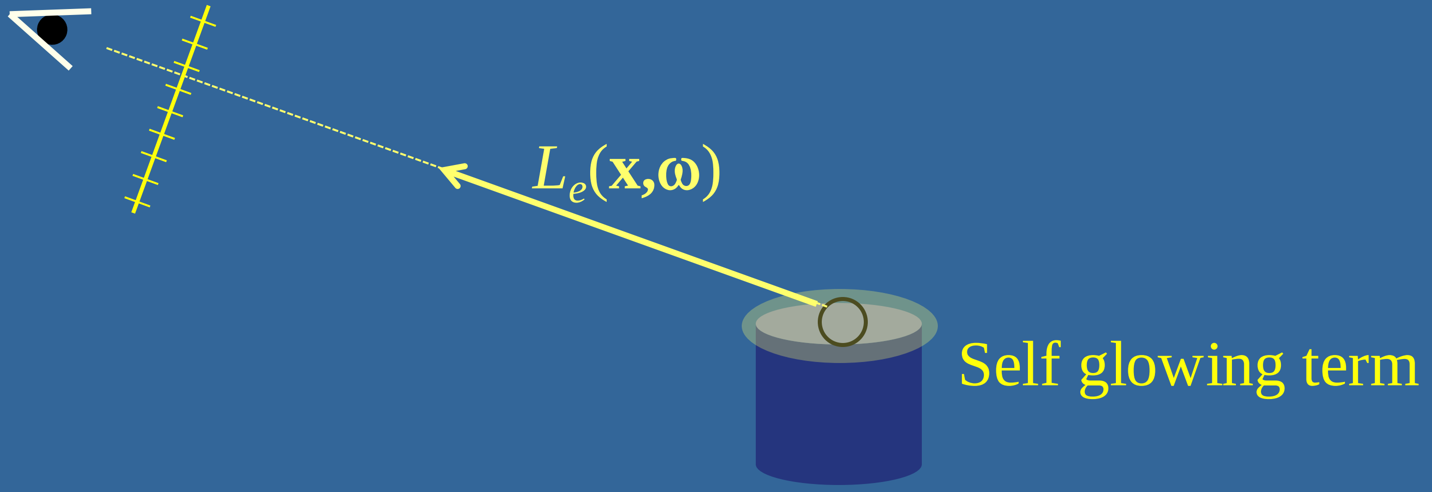


$$L_o = L_e + \int_\Omega \underline{f_r(\mathbf{x},\omega,\omega')} L_i(\mathbf{x},\omega')(\omega'\cdot\mathbf{n})d\omega'$$

- f_r is the BRDF, ω’ is incoming direction, $\mathbf{n}$ is normal at point $\mathbf{x}$, Ω is hemisphere ”around” $\mathbf{x}$ and $\mathbf{n}$, L_i is incoming radiance

The rendering equation - Summary

- Paper by Kajiya, 1986.
- Is the basis for all global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$
  - outgoing=emitted+reflected radiance



$$L_o = \underline{L_e} + \int_{\Omega} f_r(\mathbf{x},\omega,\omega')L_i(\mathbf{x},\omega')(\omega'\cdot\mathbf{n})d\omega'$$

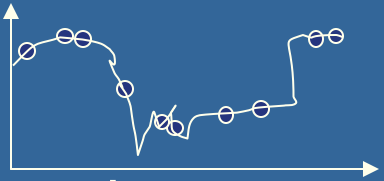
- f_r is the BRDF, ω' is incoming direction, $\mathbf{n}$ is normal at point $\mathbf{x}$, Ω is hemisphere ”around” $\mathbf{x}$ and $\mathbf{n}$, L_i is incoming radiance

Many GI algorithms are built on Monte Carlo Integration

- Integral in rendering equation:
  - Hard to evaluate numerically
  - But we can sample it.

$$L_o = L_e + \int_\Omega f_r(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')L_i(\mathbf{x},\boldsymbol{\omega}')(\boldsymbol{\omega}'\cdot\mathbf{n})d\boldsymbol{\omega}'$$

- MC can estimate integrals: $I = \int_a^b f(x)dx$



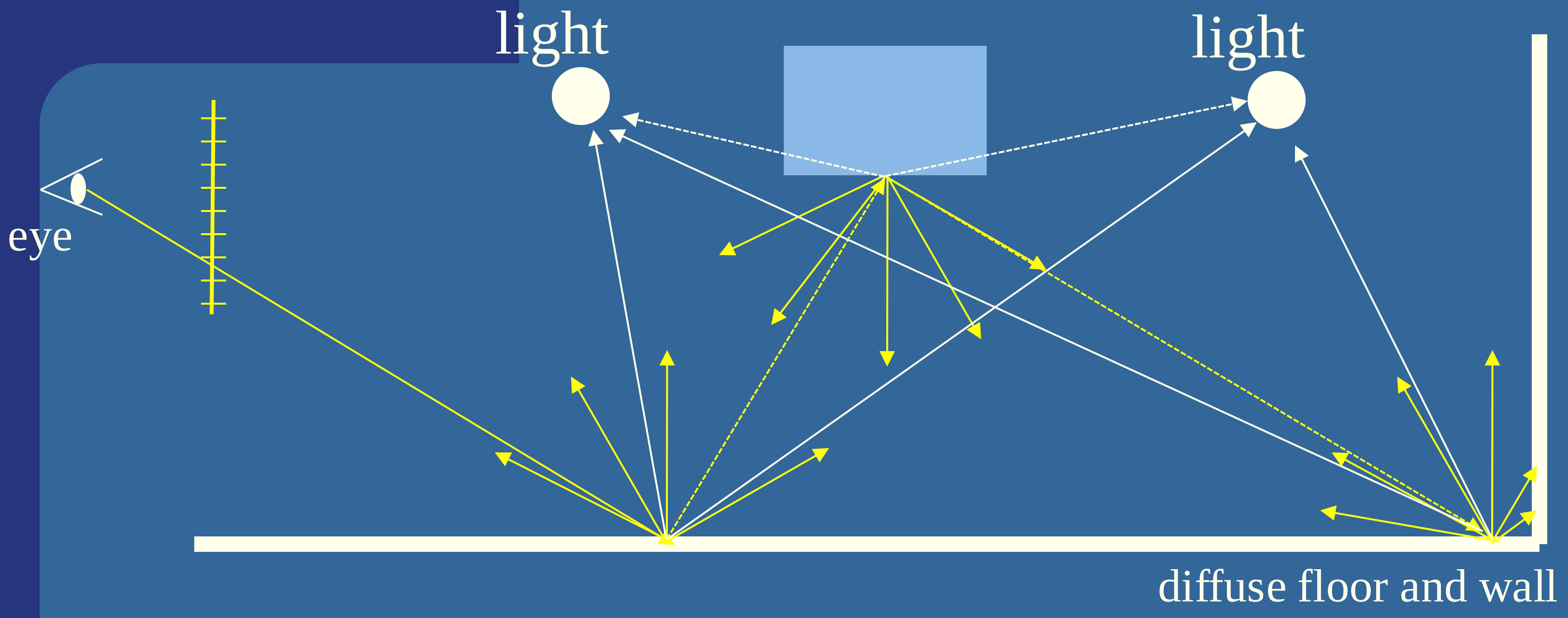
- Assume we can compute the mean of $f(x)$ over the interval $[a,b]$
  - Then the integral is mean*(b-a)
- Thus, focus on estimating mean of $f(x)$
- Idea: sample f at n uniformly distributed random locations, x_i:

$$I_{MC} = (b-a)\frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

Monte Carlo estimate

- When $n \rightarrow infinity$, $I_{MC} \rightarrow I$
- Standard deviation convergence is slow: $\sigma \propto \frac{1}{\sqrt{n}}$
- Thus, to halve error, must use 4x number of samples!

Monte Carlo Ray Tracing (naïvely)

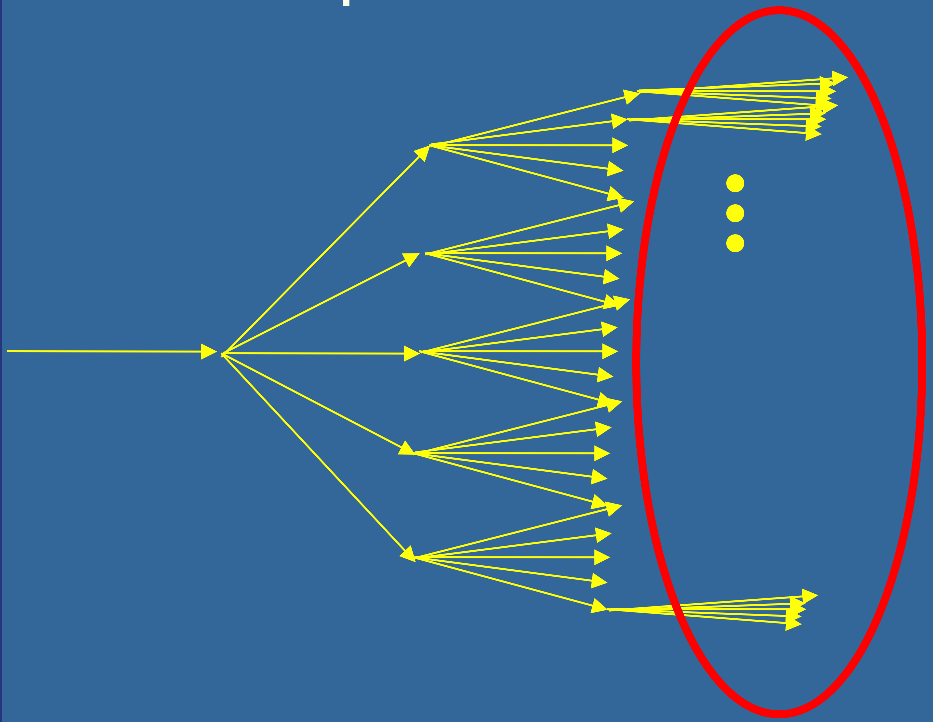


- But we separate direct lighting from indirect, since direct lighting is so dominant (when not in shadow), by always shooting a ray to light sources.
  - I.e., compute local lighting as usual, with a shadow ray per light.
- Then, sample indirect illumination by shooting sample rays over the hemisphere, at each hit.
- This separation of local vs global lighting works without getting math biasing issues.

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

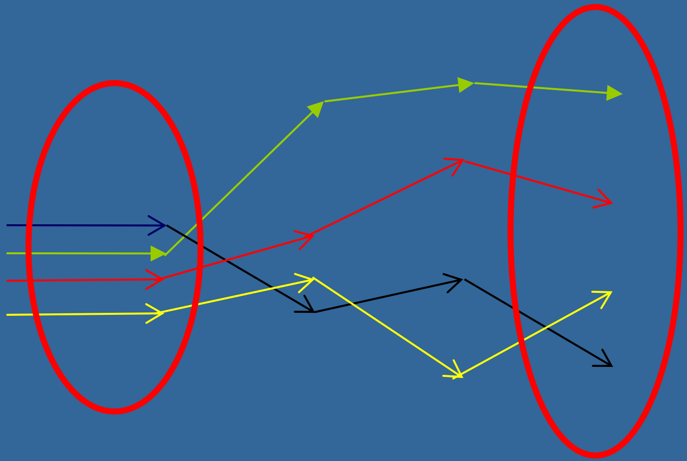
Monte Carlo Ray Tracing (naïvely)

- The indirect-illumination sampling gives a ray tree with most rays at the bottom level. This is bad since these rays have the lowest influence on the pixel color.



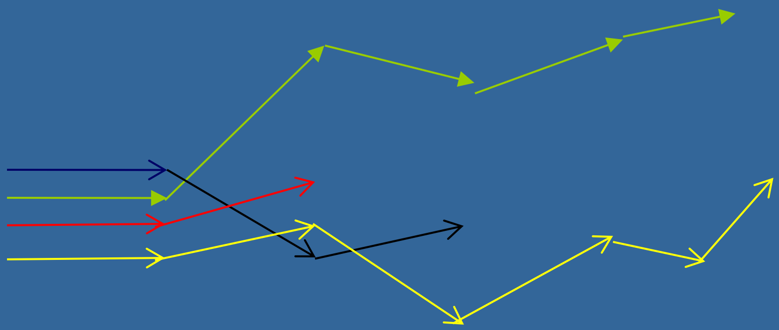
PathTracing
– one efficient Monte-Carlo Ray-Tracing solution

- Path Tracing instead only traces one of the possible ray paths at a time. This is done by randomly selecting only one sample direction at a bounce. Hundreds of paths per pixel are traced.



Equally number of rays are traced at each level

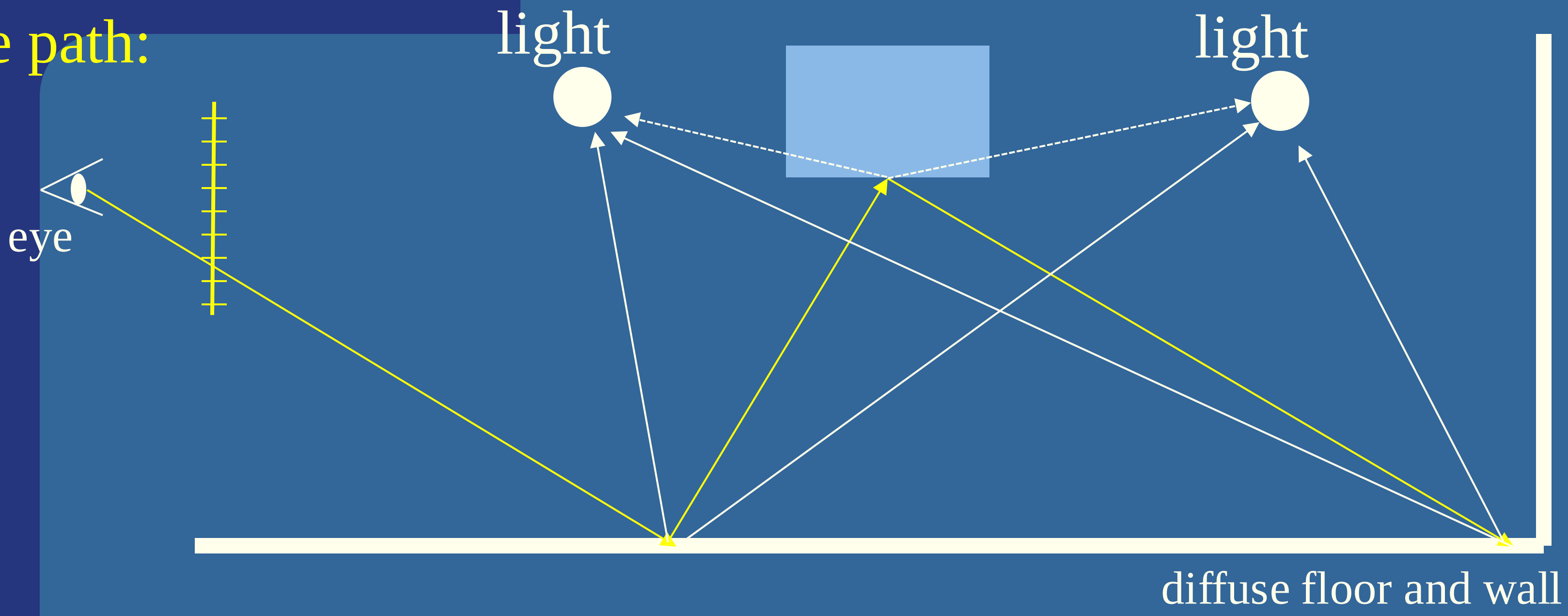
Or:



Even smarter: terminate path with some probablility after each level, since they have decreasing importance to final pixel color.

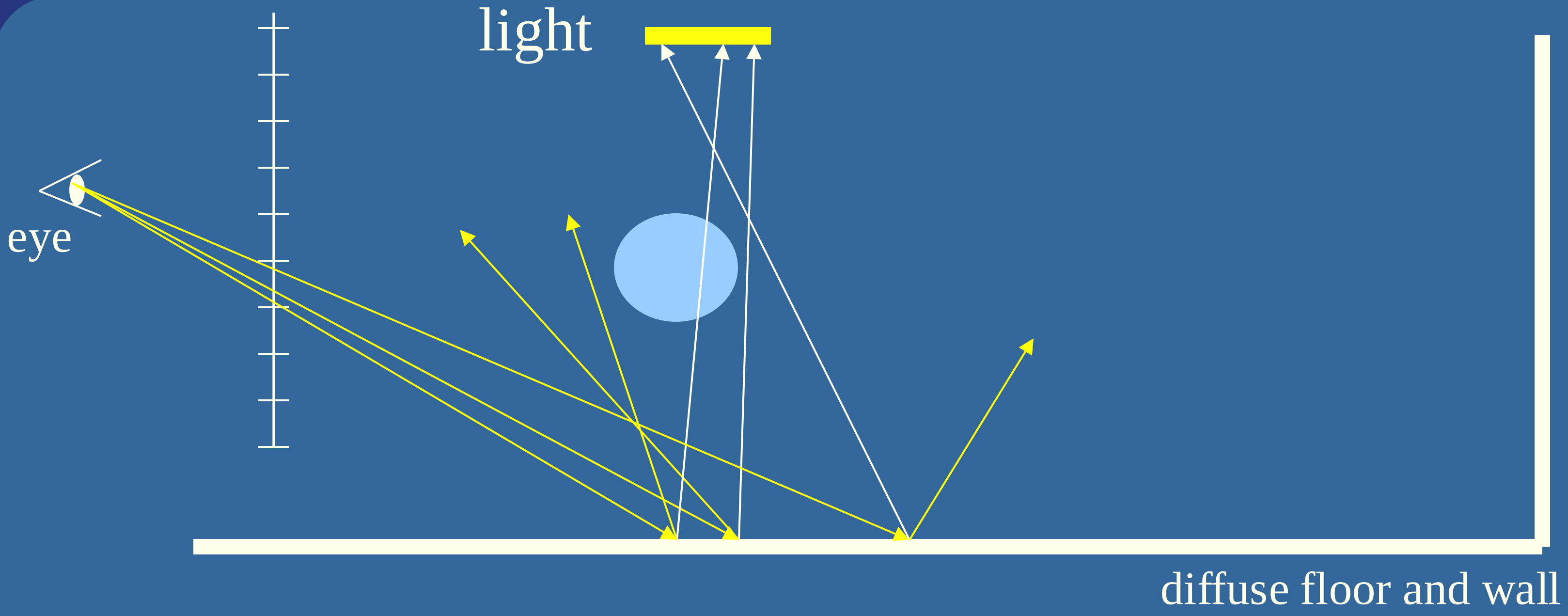
Path Tracing – indirect + direct illumination.

One path:



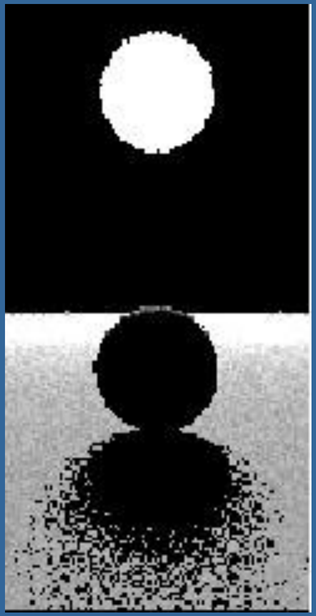
- Shoot many paths per pixel (the image just shows one light path).
  - At each intersection,
    - Shoot one shadow ray per light source
      - at random position on light, for area/volumetric light sources
    - and randomly select one new ray direction.

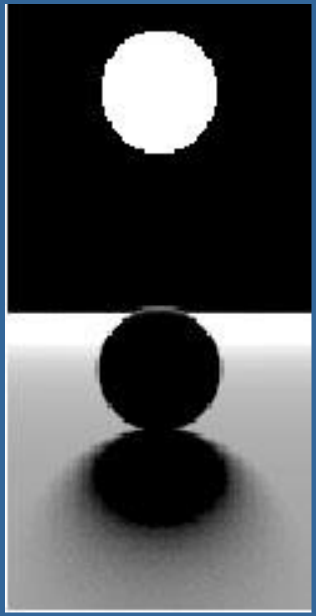
Path Tracing and area lights



- For area light sources, shoot the shadow ray to one random position on the area light. This gives soft shadows when many paths are averaged for the pixel.
- Example: Three paths for one pixel
  - At each ray intersection,
    - Pick ***one*** random position on light source
    - Send one random ray bounce to continue the path...

Example of diffuse surface + soft shadows


One sample per pixel


100 samples per pixel

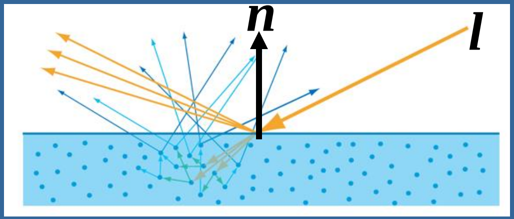
- Need to send many many paths to avoid noisy images
  - Perhaps 10,000 or more paths are needed per pixel
    - But eventually you often denoise by Final Gather or AI denoising.
- Still, it is a simple method to generate high quality images. Will converge to a statistically correct result.

Images courtesy of Peter Shirley

Path tracing: Summary

- Uses Monte Carlo sampling to solve integration:
  - by shooting many random ray ***paths*** over the integral domain.
  - Algorithm:
    - For each pixel, // we will shoot a number of paths:
      - For each path, generate the primary ray:
      - Repeat {
        1. Trace the ray. At hitpoint:
        2. Shoot one shadow ray (per light) to compute direct lighting.
        3. Sample indirect illumination randomly over the possible reflection/refraction directions by generating **one** new ray to continue the path.
      - } until the path is randomly terminated (or the ray does not hit anything).
- Shorter summary: shoot many paths per pixel, by randomly choosing **one** new ray at each interaction with surface + **one** shadow ray per light. Terminate the path with a random probability

Russian Roulette



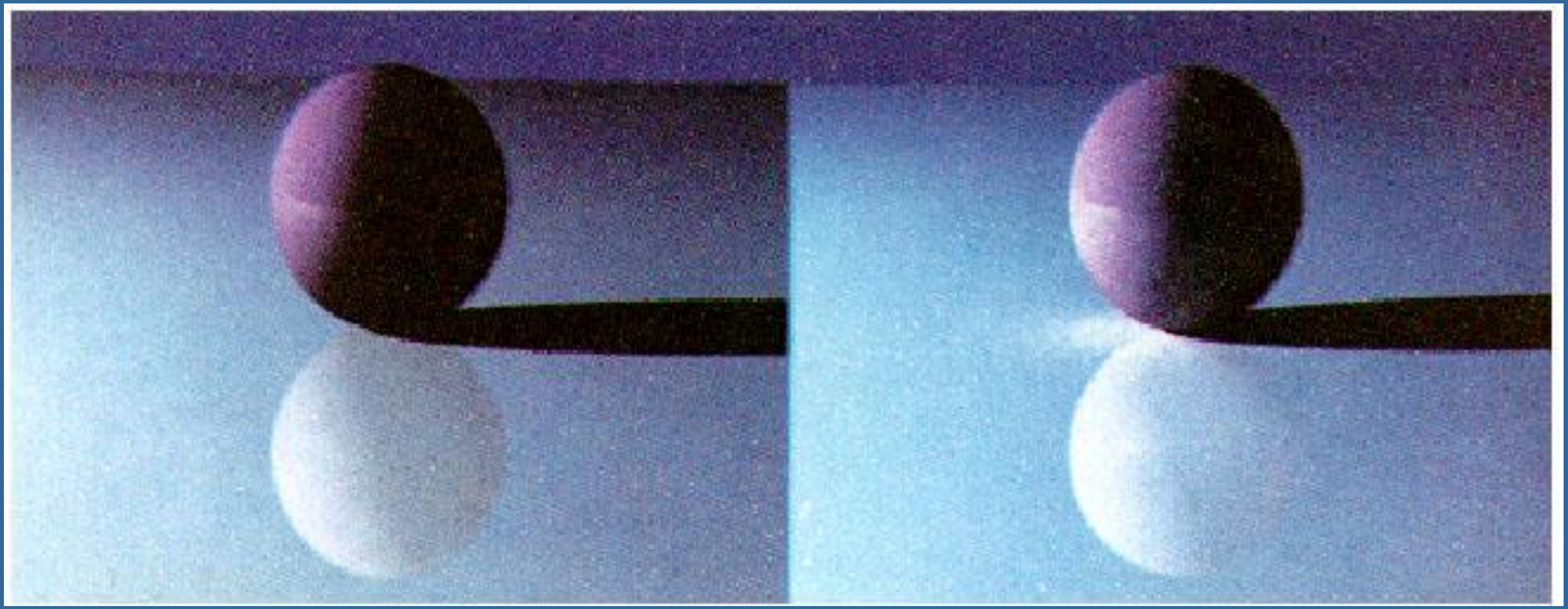
Use randomness to decide whether to trace a diffuse or specular ray:

- Assume $k_{diff}+k_{spec}$<=1 (since energy cannot be created but can be absorbed)
  - In Physically-based Shading – use the mtrl brdf, e.g., f_{spec}= D()G()F():
    - Let k_{spec} = *%reflectivity* for the ray w.r.t incoming angle
    - Let k_{diff} = *%refraction* for the ray w.r.t incoming angle
      (If transparent mtrl., then also randomly select between diffuse ray and transparency ray based on material's %transparency.)

- When a ray hits such a surface
  - Pick a random number, r in [0,1]
  - If($r < k_{diff}$) $\rightarrow$ send diffuse ray (e.g. in random direction)
  - Else if($r < k_{diff} + k_{spec}$) $\rightarrow$ send specular ray (e.g. along reflection dir.)
  - Else absorb ray, i.e., terminate ray.
- This is called **Russian roulette**.
  - Common for layered materials.
  - and for BRDF's, see path-tracer lab.
- Point: this selects just one ray so we get a path instead of a tree.

A classical example – spec+diff surface + hard shadow

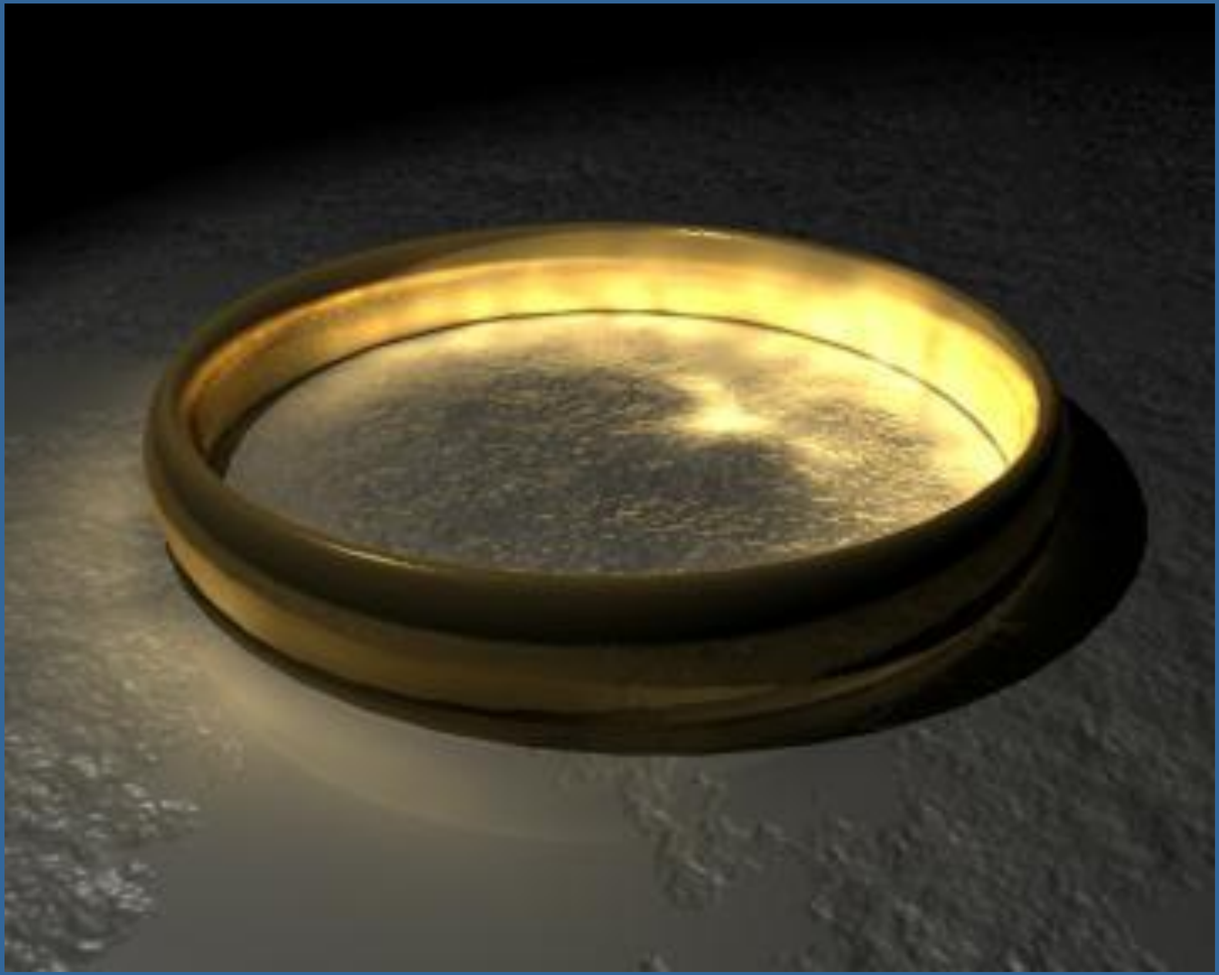
- Path tracing was introduced in 1986 by Jim Kajiya



- Note how the right sphere reflects light, and so the ground under the sphere is brighter

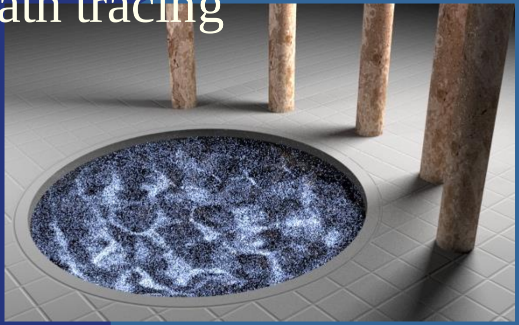
What is Caustics?

- Caustic’s don’t work well for path tracing

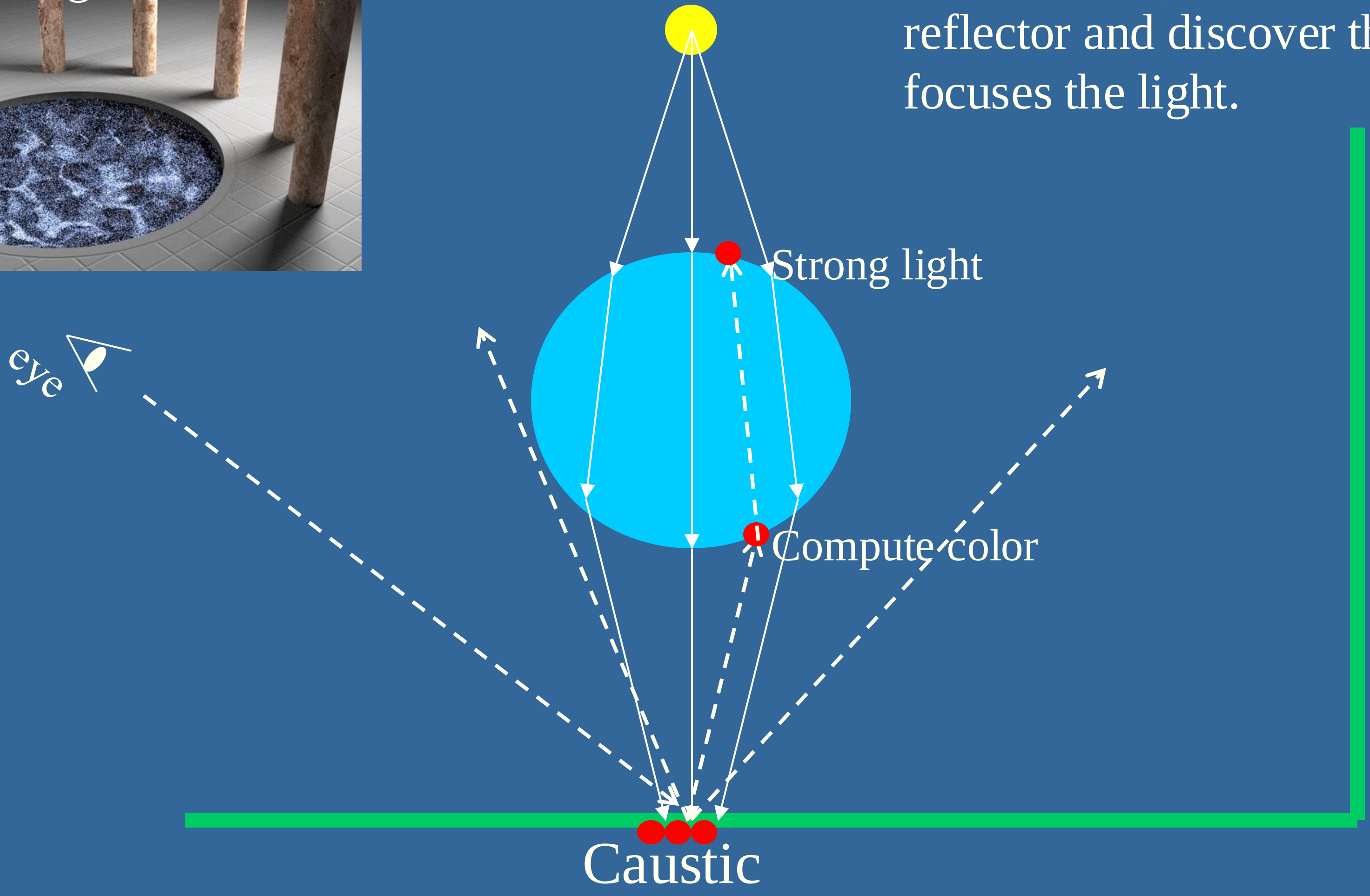


Reason why forward ray tracing fails to capture caustics well

Path tracing



Must be lucky to hit the specular reflector and discover that it focuses the light.

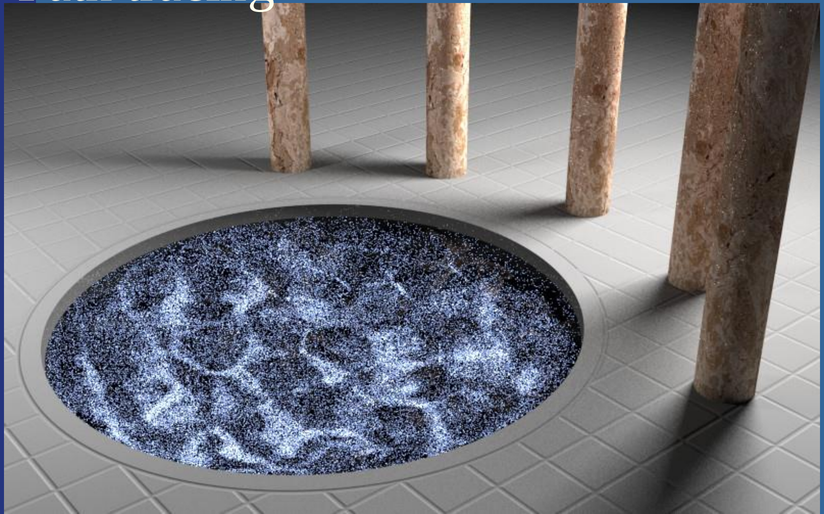


Extensions to path tracing

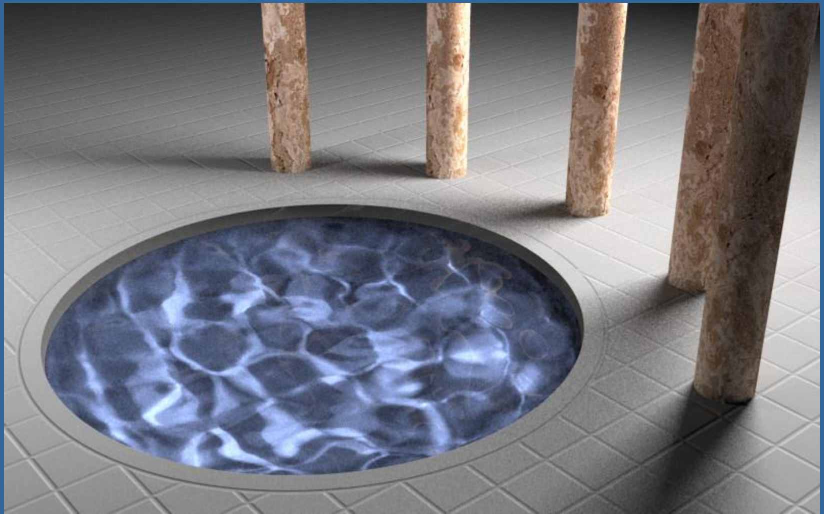
- Bidirectional path tracing
  - Developed in1993-1994
  - Sends light paths, both from eye and from the light
  - Faster, but still noisy images.
- Metropolis light transport
  - 1997
  - Ray distribution is proportional to unknown function
  - Means that more rays will be sent where they are needed
  - Faster convergence in certain cases (see below)

Images courtesy of Eric Veach

Path tracing



Metropolis (same rendering time)



Bidirectional Path tracing

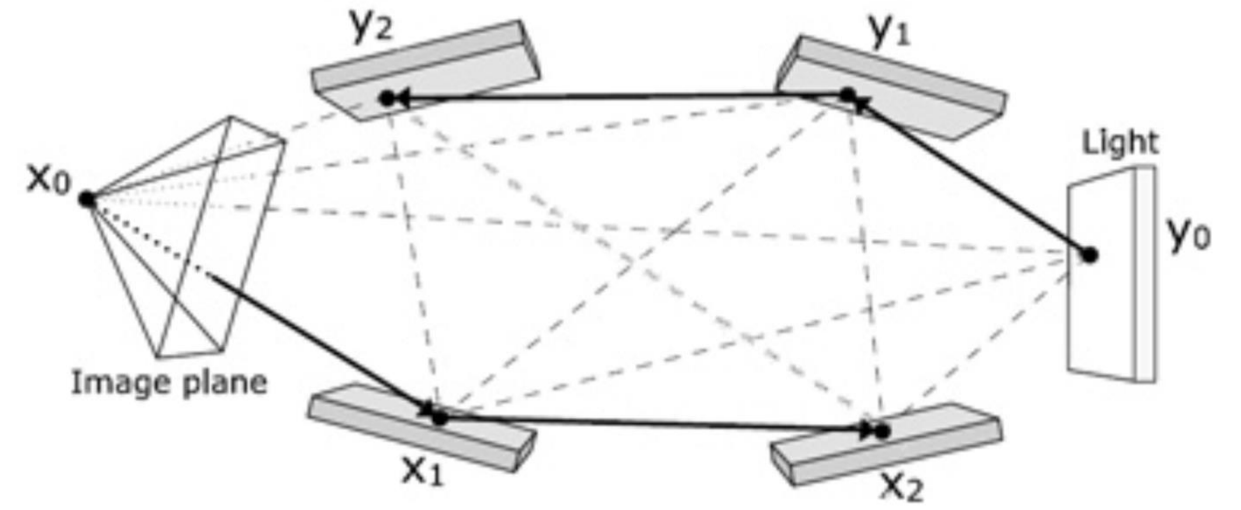
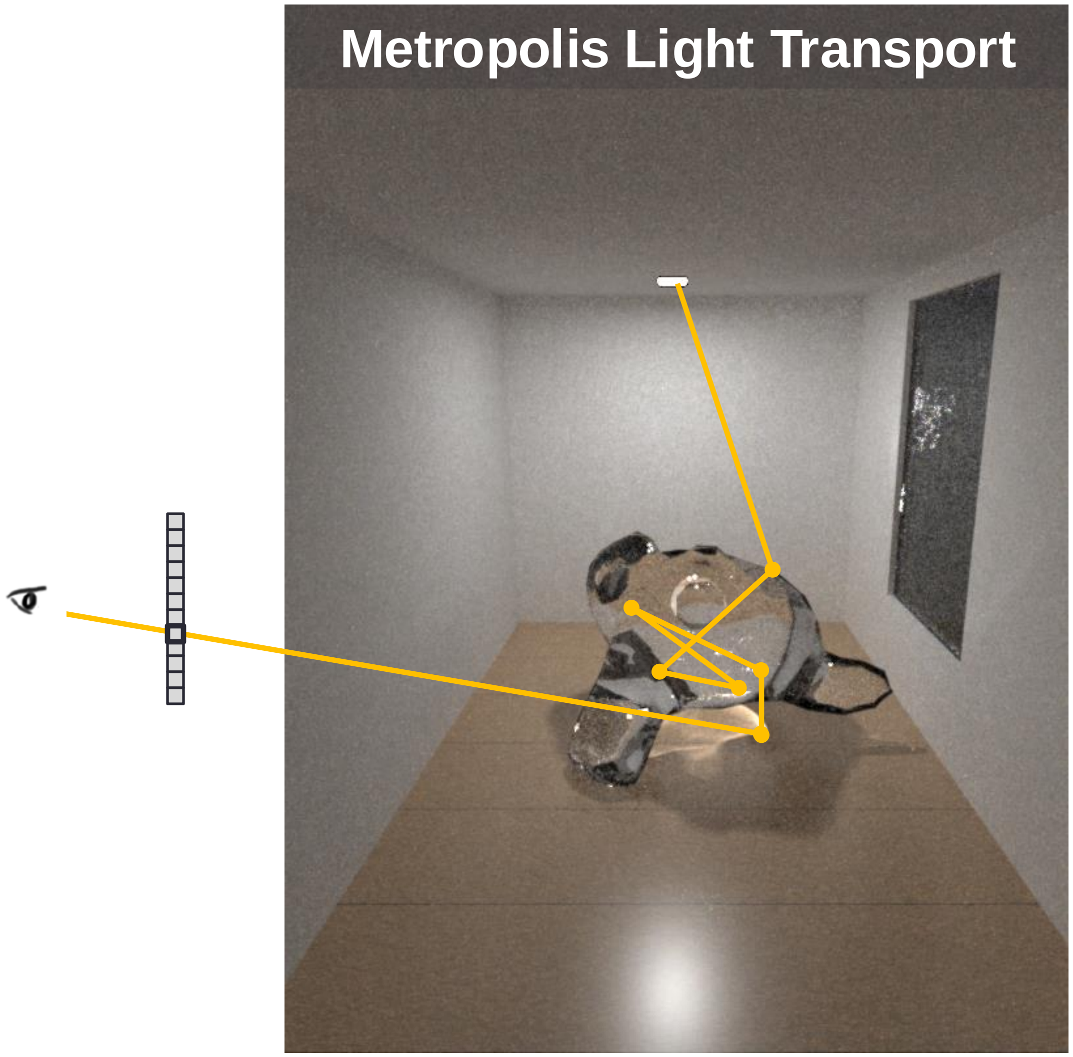
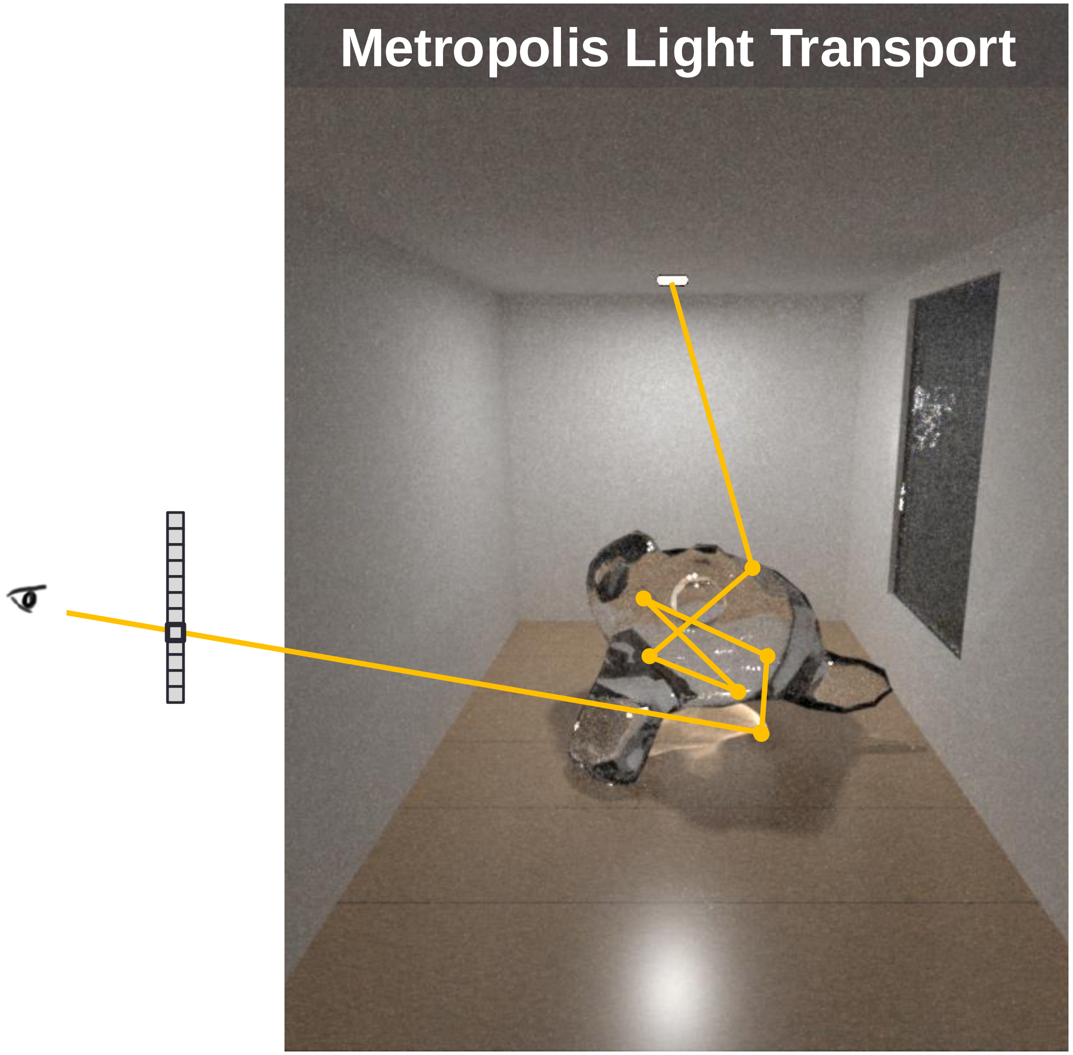


Figure 1: The different ways to combine the eye path and the light path. The eye path starts at the eye in X0 and moves through X1 and X2. The light path starts at the light source in Y0 and moves through Y1 and Y2. To create a complete path, a shadow ray is used to connect X2 and Y2. More paths can be generated by connecting all the different combinations of points on the sub paths.

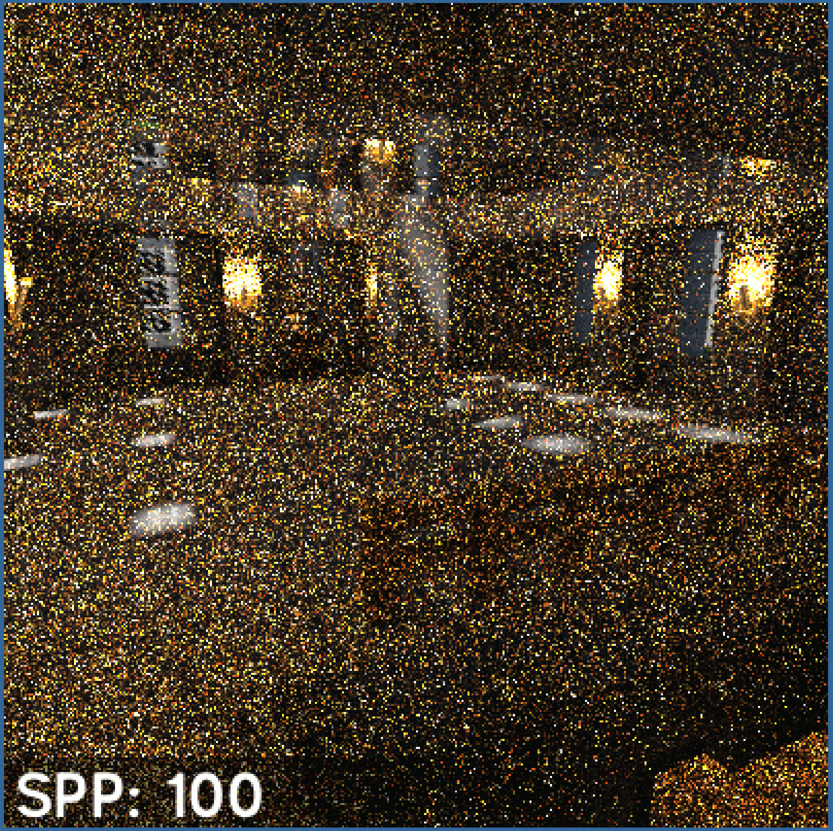

Metropolis Light Transport

Metropolis Light Transport



Denoising

- Monte Carlo ray tracing is typically slow or noisy.
- You can denoise by using:
  - Final Gather (older)
  - or AI denoising (new).
    - E.g., a machine learning autoencoder that takes in 3 images: albedo (=diffuse), first bounce normals, and the input noisy image. Outputs a filtered image.
      - OIDN – Intel Open Image Denoise library
      - OptiX Recurrent Denoising Autoencoder, NVIDIA
      - DLSS, NVIDIA

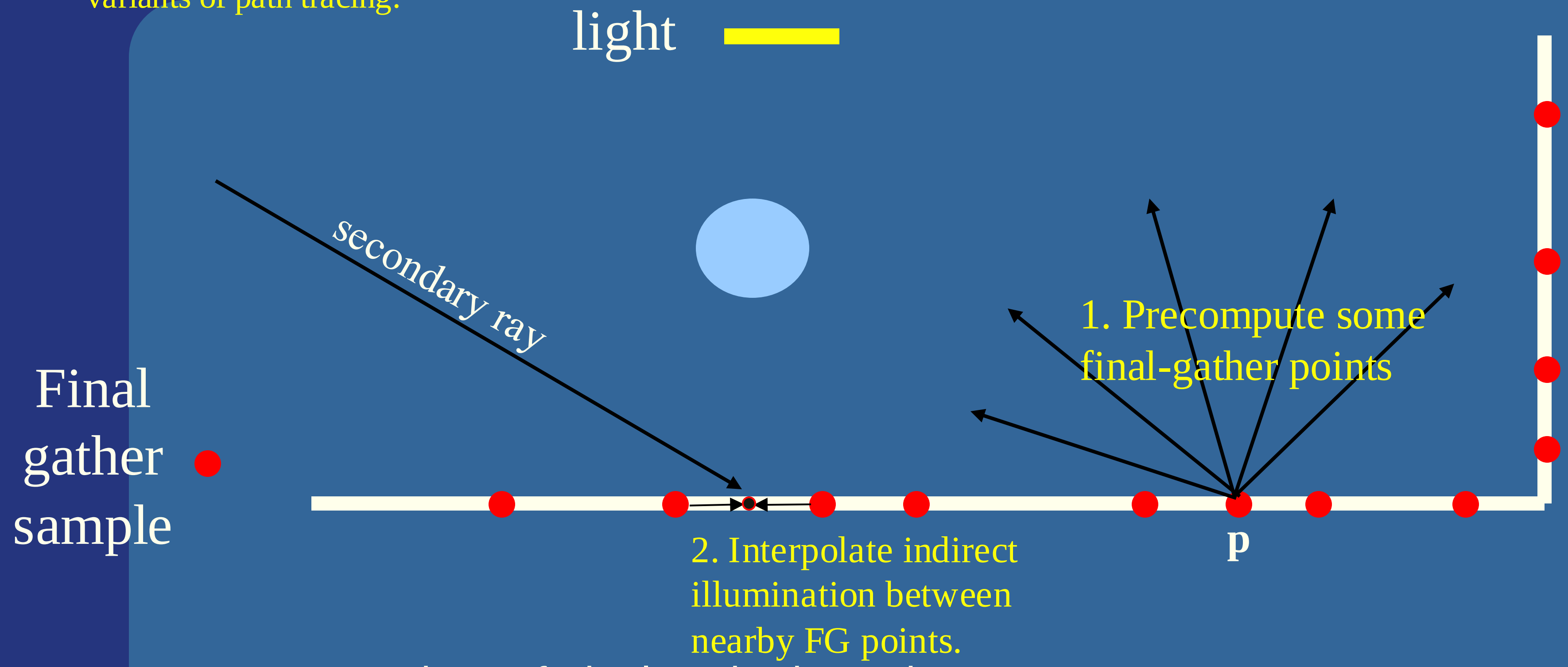


For more info, see https://alain.xyz/blog/ray-tracing-denoising

Final Gather

Popular for naïve monte carlo ray tracing and photon mapping but not for variants of path tracing.

Idea and good answer:
- Compute indirect illumination somehow, but only at sparse set of positions (final gather points) in the scene.
- Estimate indirect illumination for other positions by interpolation from nearby final-gather points

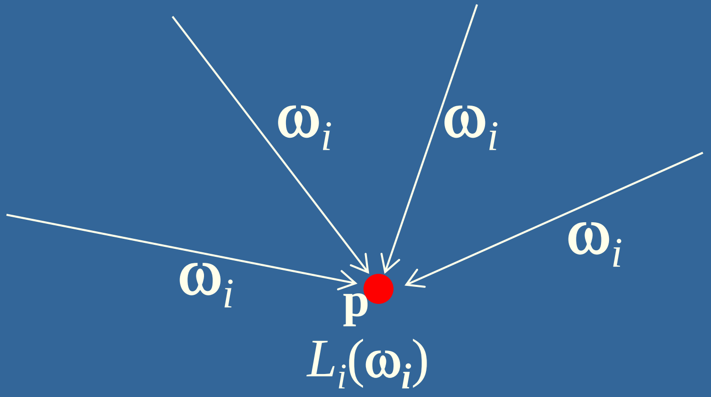

light
secondary ray
Final gather sample
1. Precompute some final-gather points
2. Interpolate indirect illumination between nearby FG points.
p

- Many versions of Final Gathering exist.
- E.g., to compute final-gather point **p**:
  - Send thousand(s) random rays out from **p** to sample indirect illumination
- To use during ray tracing: interpolate global illumination between nearby Final Gather points, to estimate incoming radiance at the ray's intersection point.
- Does not matter much if indirect illumination is blotchy for secondary rays.

Bonus

Final Gather – sample representation

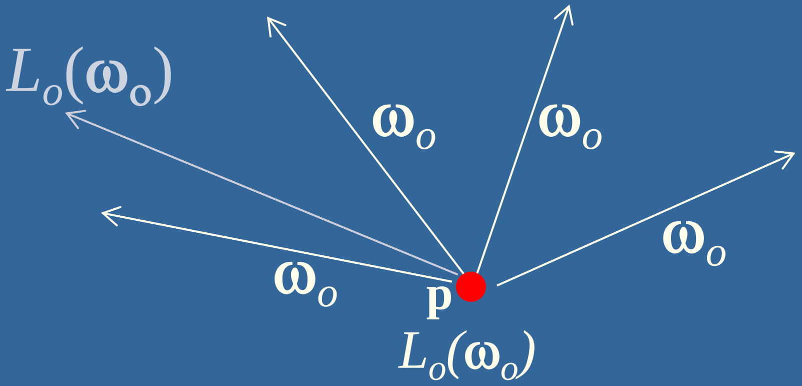
– The FG samples typically need to store directional information about radiance $L_i(\omega)$.



directional *incoming* radiance

$L_i(\omega_i)$ independent of **p**'s brdf . Better for interpolation between nearby positions.

Or:

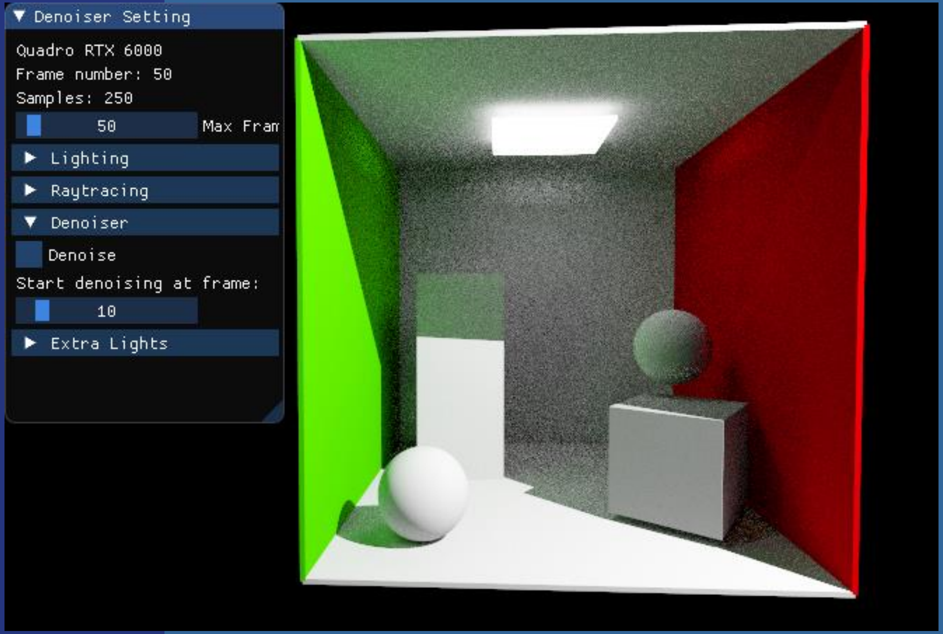


directional *outgoing* radiance

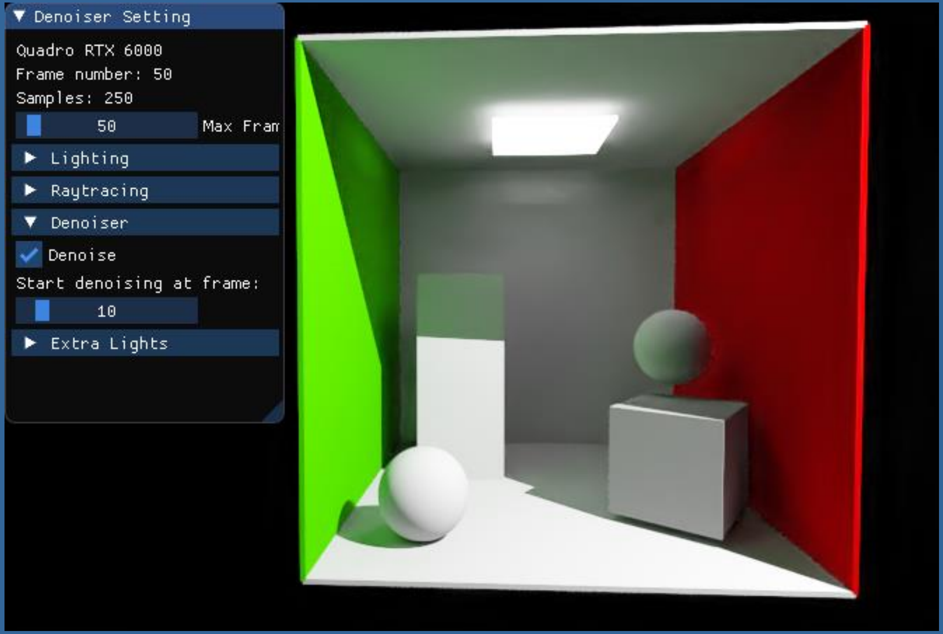
$L_o(\omega_o)$ only valid for **p**'s brdf. E.g., bad for textured surfaces.

- Directional radiance information can for instance be stored as *Spherical Harmonics* or a set of *Spherical Gaussians* (beyond this course).
- You may store directional *incoming* radiance, to then be multiplied for each incoming direction by the brdf to compute outgoing radiance for desired ray direction
- Or just store directional *outgoing* radiance directly, thereby baking in the surface brdf (faster but less general when interpolating FG samples)

Path tracing + AI Denoising


▼ Denoiser Setting
Quadro RTX 6000
Frame number: 50
Samples: 250

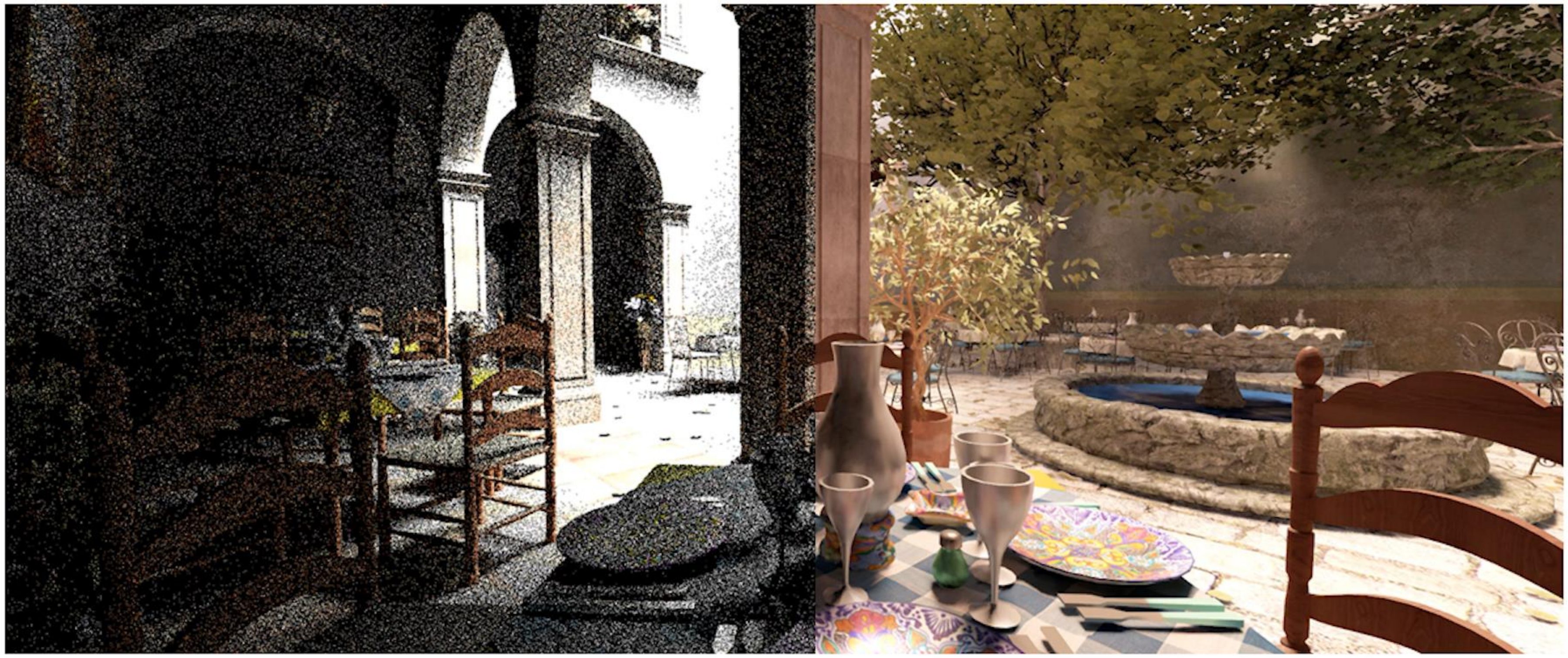
Before denoising


▼ Denoiser Setting
Quadro RTX 6000
Frame number: 50
Samples: 250

After denoising

Real-time Denoising - NVIDIA

Need to Start with a Noisy Result and Reconstruct



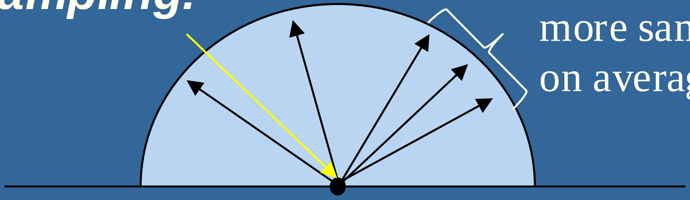
(8 min video) https://www.youtube.com/watch?v=6O2B9BZiZjQ

BONUS

Monte-Carlo Ray tracing – the maths

The weight of the radiance from each sampled ray direction:

- If hemispherical directions are **not** sampled perfectly **randomly**, then the weight for each of the *n* sampled rays is **not** just $w = 1/n$,
  - e.g., when shooting more sampling rays towards the more probable directions (by trying to somewhat regard the BRDF). This is called ***importance sampling:***



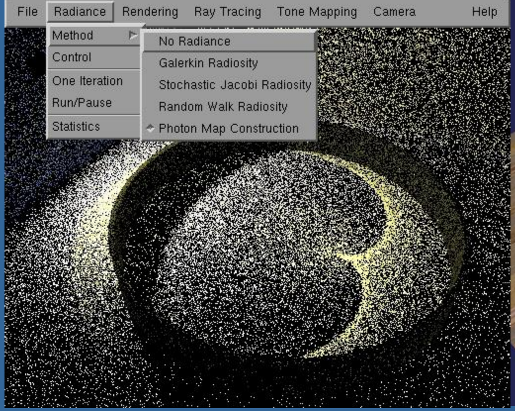
more samples in these directions than on average over the hemisphere.

- Solutions:
  - In theory, we could look at the actual taken sample directions, and estimate good weights. This is rarely used.
    - Does not work well for path tracing, since we only sample one direction per position.
  - Or, rely on probability theory, which will converge to correct weights when #samples, *n,* goes to infinity.
    - $w_i = 1/(n * p(\omega_i)), \quad p(\omega_i) = \text{"}probability\_bias\_of\_the\_choosen\_direction\text{"}$

    where function $p(\omega)$ is our Probability Density Function (PDF)
    - This is what people use today. See our path-tracing tutorial.

How much more/less likely direction ω_i is compared to the average direction (avg. should be =1).

Photon mapping



HENRIK WANN JENSEN 1996

- Developed by Henrik Wann Jensen (started 1993)
- A two-pass algorithm:
  - 1: Shoot photons from light source, and let them bounce around in the scene, and store them where they land (e.g. in a kD-tree).
  - 2: Ray- or path-tracing pass from the eye. Estimate photon density at each ray hit, by growing a sphere (at the hit point in the kD-tree) until it contains a predetermined #photons. Sphere radius is then the inverse measure of the light intensity at the point.
- Features:
  - Polurar in the 90'ies + 00'ies. Now again popular by combining bidirectional path tracing and progressive photon mapping.
  - Less noise than path tracing

The first pass: Photon tracing

- Store illumination as points (photons) in a ”photon map” data structure
- In the first pass: photon tracing
  - Emit photons from light sources
  - Trace them through scene
  - Store them in photon map data structure
- More details:
  - When a photon hits a surface (that is not too specular), store the photon in photon map
  - Then use Russian roulette to find out whether the photon is absorbed, reflected, or refracted
  - If reflected, then shoot photon in new random direction

The photon map data structure

- Keep them in a separate (from geometry) structure
- Store all photons in kD-tree
  - Essentially an axis-aligned BSP tree, since we must alter splitting axis: x,y,z,x,y,z,x,y,z, etc.
  - Each node stores a photon
  - Needed because the algorithm needs to locate the n closest photons to a point
- A photon:
  - float x,y,z;
  - char power[4]; // essentially the color, with more accuracy
  - char phi,theta; // compact representation of incoming direction
  - short flag; // used by KD-tree (stores which plane to split)
- Create balanced KD-tree – simple, done once.
- Photons are stored linearly in memory:
  - Parent node at index: p
  - Left child at: 2p , right child: 2p+1

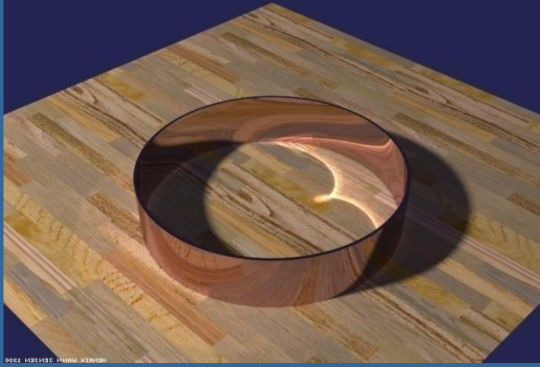
Locate n closest photons

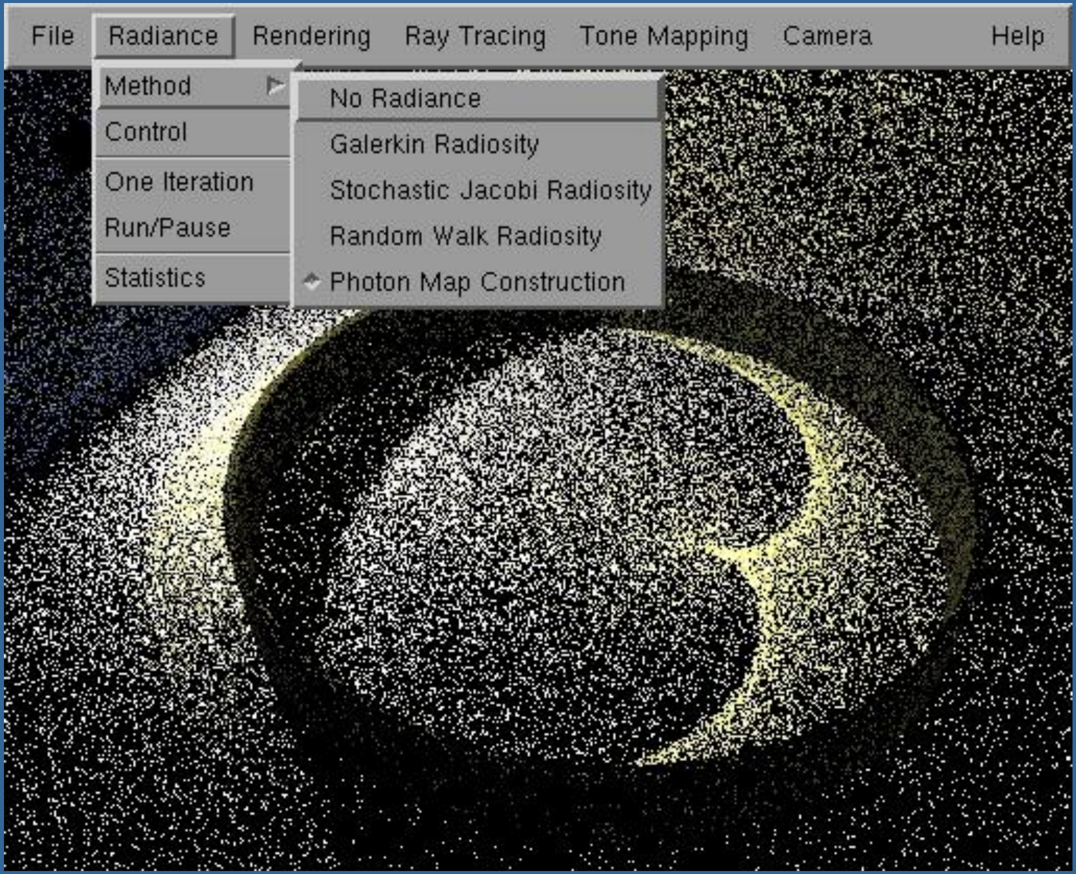
After Henrik Wann Jensen

```
// locate n closest photons around point "pos"
// call with "locate_photons(1)", i.e., with the root as in argument
locate_photons(p)
{
        if(2p+1 < number of photons in photon map structure)
        {       // examine child nodes
                delta=signed distance to plane of node n
                if(delta<0)
                {       // we're to the "left" of the plane
                        locate_photons(2p);
                        if(delta*delta < d*d)
                                locate_photons(2p+1); //right subtree
                }
                else
                {       // we're to the "right" of the plane
                        locate_photons(2p+1);
                        if(delta*delta < d*d)
                                locate_photons(2p);  // left subtree
                }
        }
        delta=real distance from photon p to pos
        if(delta*delta < d*d)
        {       // photon close enough?
                insert photon into priority queue h
                d=distance to photon in root node of h
        }
}
// think of it as an expanding sphere, that stops exanding when n closest
// photons have been found
```

What does it look like?

- Stored photons displayed:



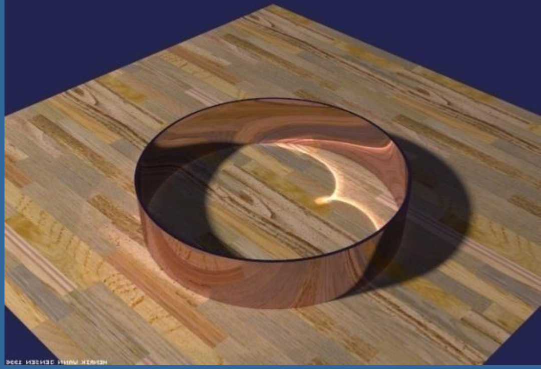


Density estimation

- The density of the photons indicate how much light that point receives
- Radiance is the term for what we display at a pixel
- Complex derivation skipped (see Jensen's book)…
- Reflected radiance at point x:

$$L(\mathbf{x},\boldsymbol{\omega}) \approx \frac{1}{\pi r^2} \sum_{1}^{n} f_r(\mathbf{x},\boldsymbol{\omega}_p,\boldsymbol{\omega})\Phi_p(\mathbf{x},\boldsymbol{\omega}_p)$$

- L is radiance in $\mathbf{x}$ in the direction of $\mathbf{w}$
- r is radius of expanded sphere
- $\boldsymbol{\omega}_p$ is the direction of the stored photon
- Φ_p is the stored power of the photon
- f_r is the BRDF



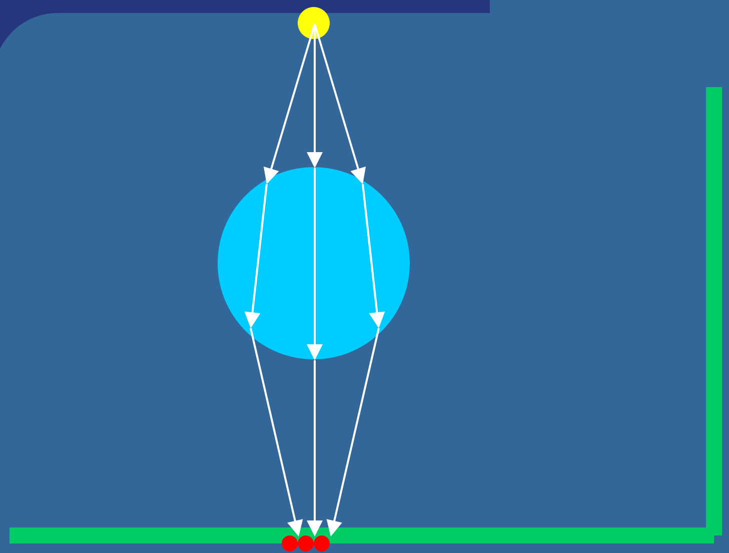
Two-pass algorithm

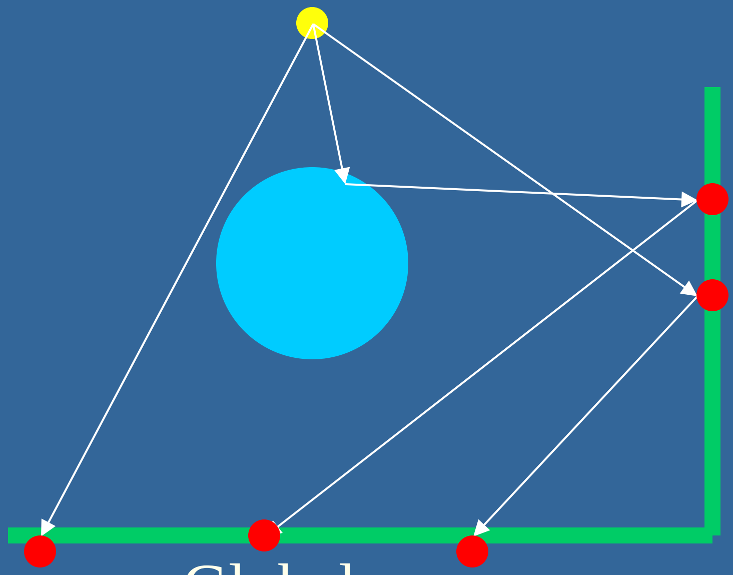
- Already said:
  - 1) Photon tracing, to build photon maps
  - 2) Rendering from the eye + using photon maps
- Pass 1 (create photon maps):
  - Use two photon maps
  - A caustics photon map (for caustics)
    - Stores photons that have been reflected or refracted (via a specular/transparent surface) to a diffuse surface

+

  - A global photon map (for all illumination)
    - All photons that landed on diffuse surfaces

Caustic map and global map


Caustic map


Global map

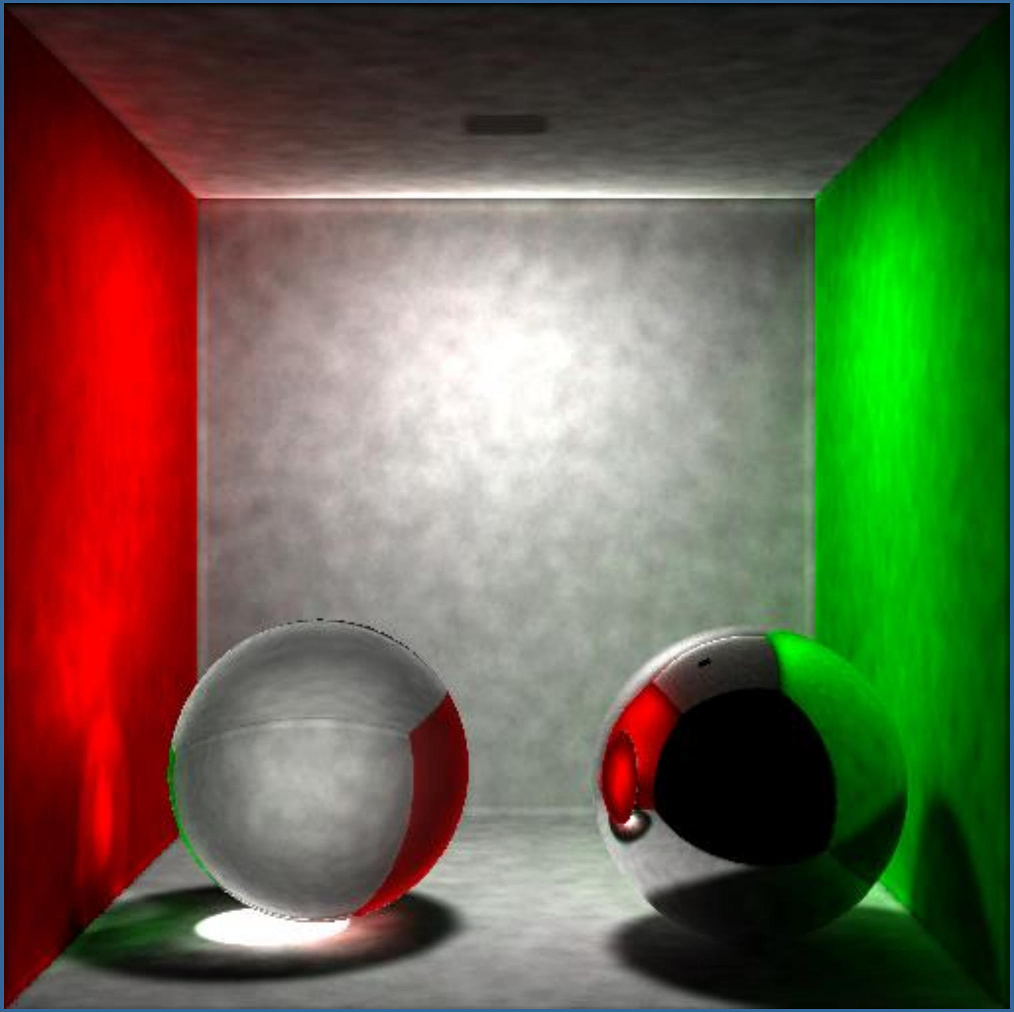
- Caustic map: send photons only towards reflective and refractive surfaces. Gives biased photon-density distribution but does not matter much:
    - Caustics is a high frequency component of illumination
    - Therefore, need many photons to represent accurately
- Global map - assumption: illumination varies more slowly.

Pass 2: Rendering using the photon map

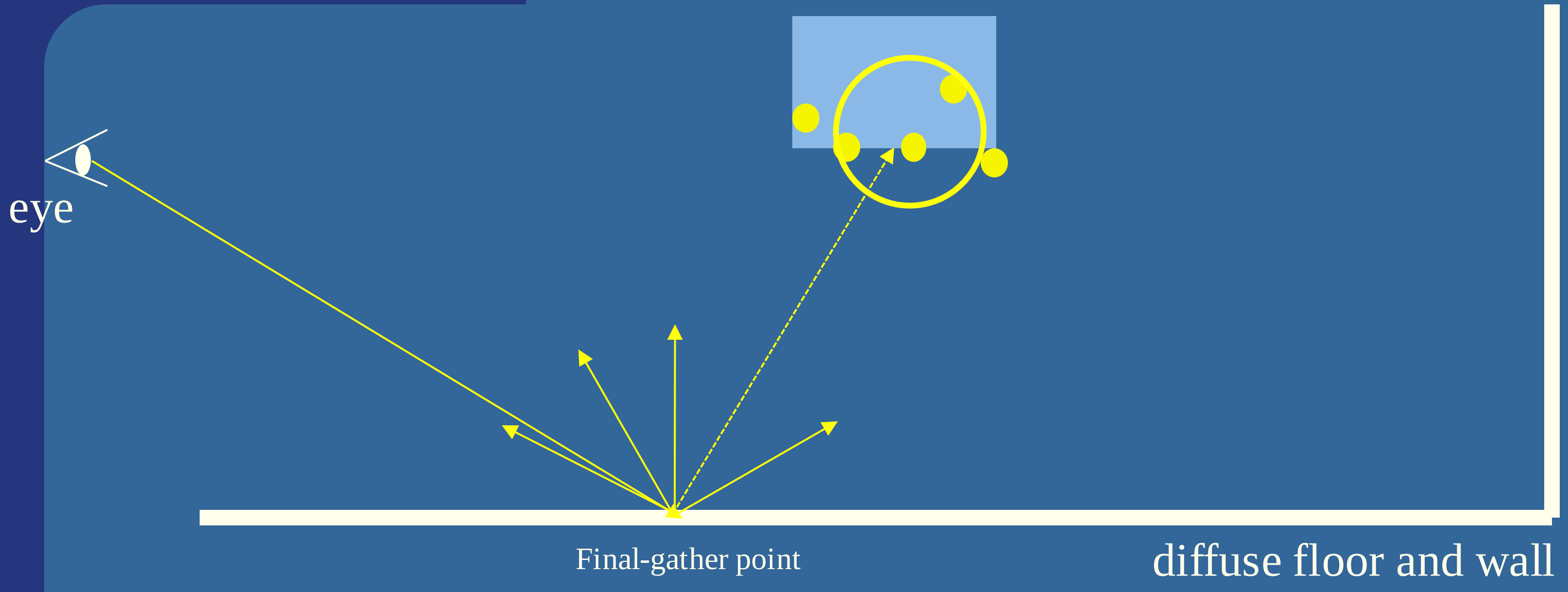
- Render from the eye using a modified ray or path tracer
  - Ray/path trace from the eye.
  - For each ray bounce (hit point **p**), compute:
    - **Direct illumination** (light that reaches a surface directly from light source): Compute local lighting with shadow rays and local shading.
    - **Indirect illumination** (options)**:**
      - Can grow sphere around **p** until it includes a predetermined #photons
        - in caustics map to get caustics contribution and
        - in global map to get slow-varying indirect illumination
      - Can use Final Gather
      - Can continue ray path (or ray-tracing recursion) a bit more.
  - Can use AI denoising as post process (can sometimes produce good caustics)

Example of noise when using the photon maps for the primary rays

- Ugly noise:
- Solution:
  - for the primary rays: don’t use the global map directly. E.g., instead use Final Gather.



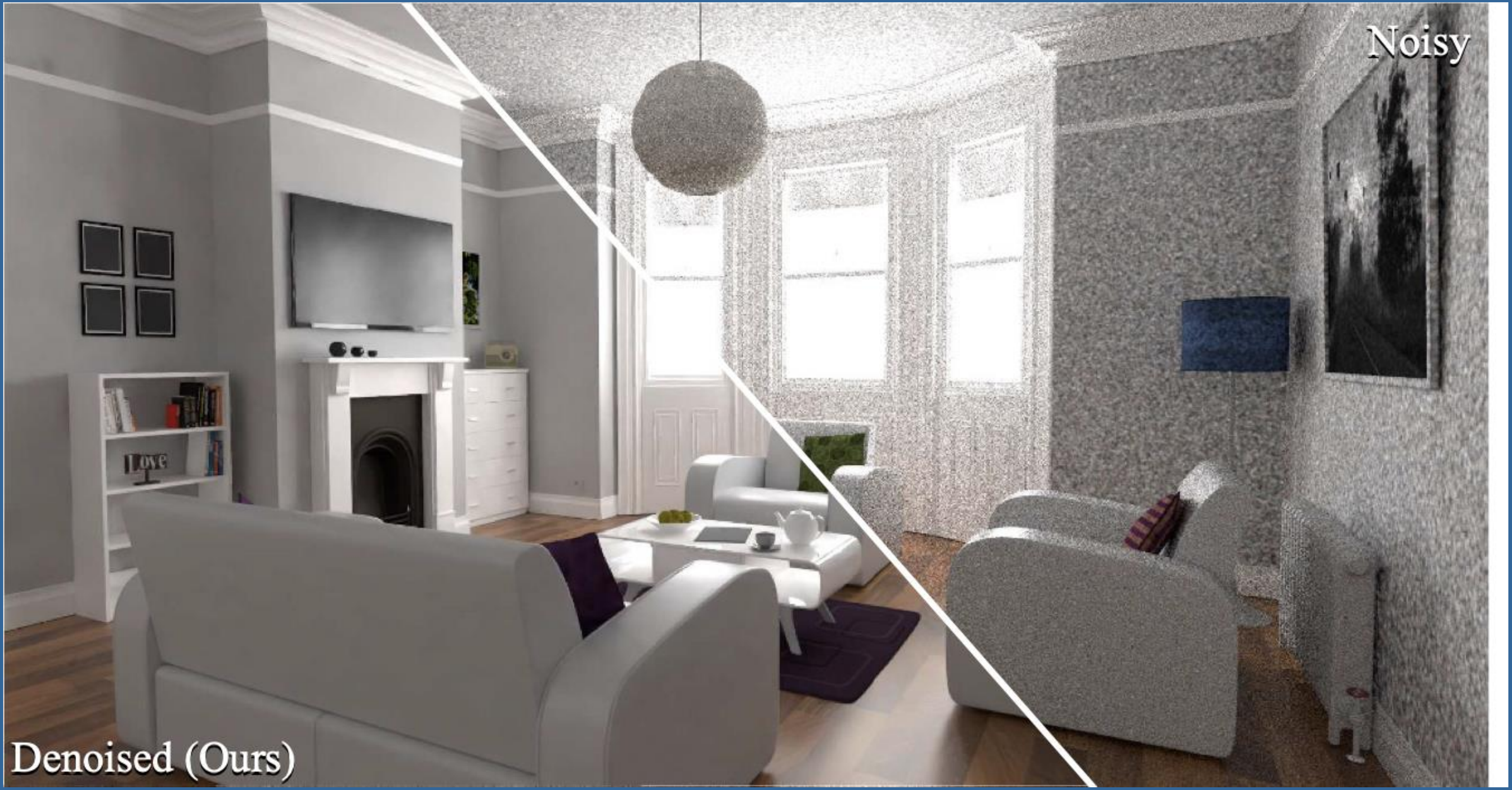
Final Gather with Photon Mapping



- Too noisy to use the global map for direct visualization
- Remember: eye rays are recursively traced (via reflections/refractions) until a diffuse enough hit, **p**. There, we want to estimate slow-varying indirect illumination.
  - Instead of growing sphere in global map at **p**, Final Gather shoots 100-1000 indirect rays from **p.** Where each of those rays end at a surface, grow a sphere in the global map and also caustics map, or interpolate from nearby already computed final-gather points.

Photon Mapping + AI Denoising

- Or use AI denoising instead of Final Gathering:



Photon Mapping + AI Denoising

- Or use AI denoising instead of Final Gathering:

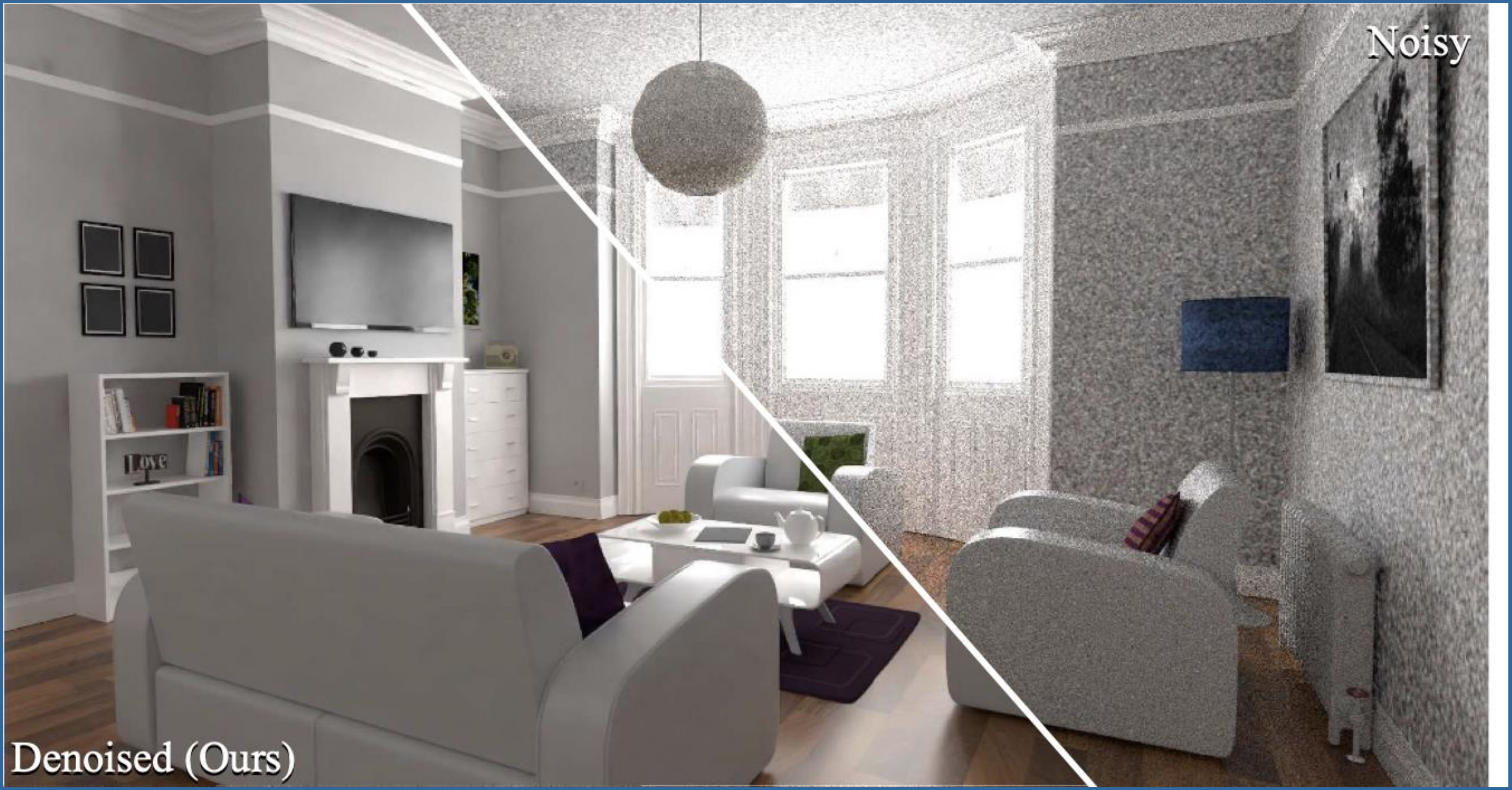
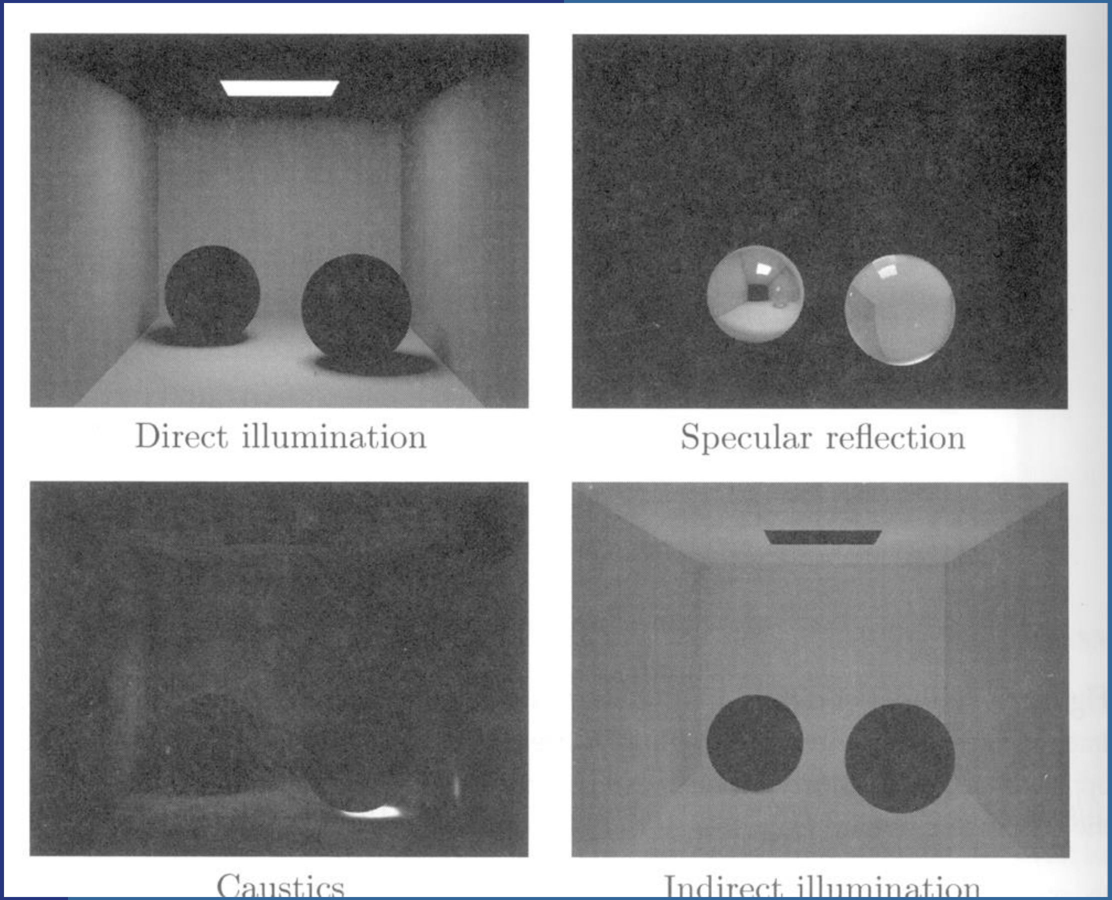
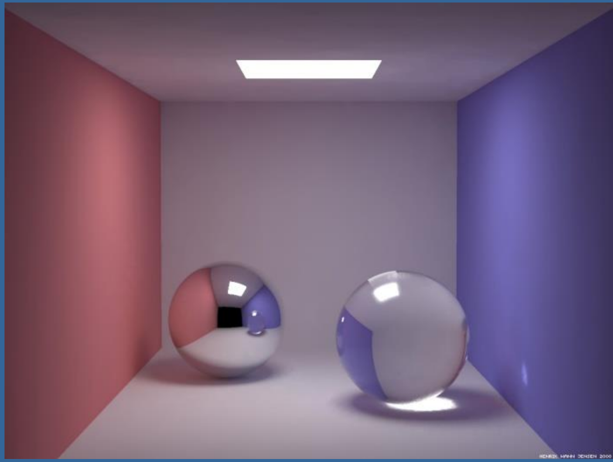


Image scanned from Henrik Wann Jensen's book

Images of the four components



=



Photon Mapping - Summary

Reality check:

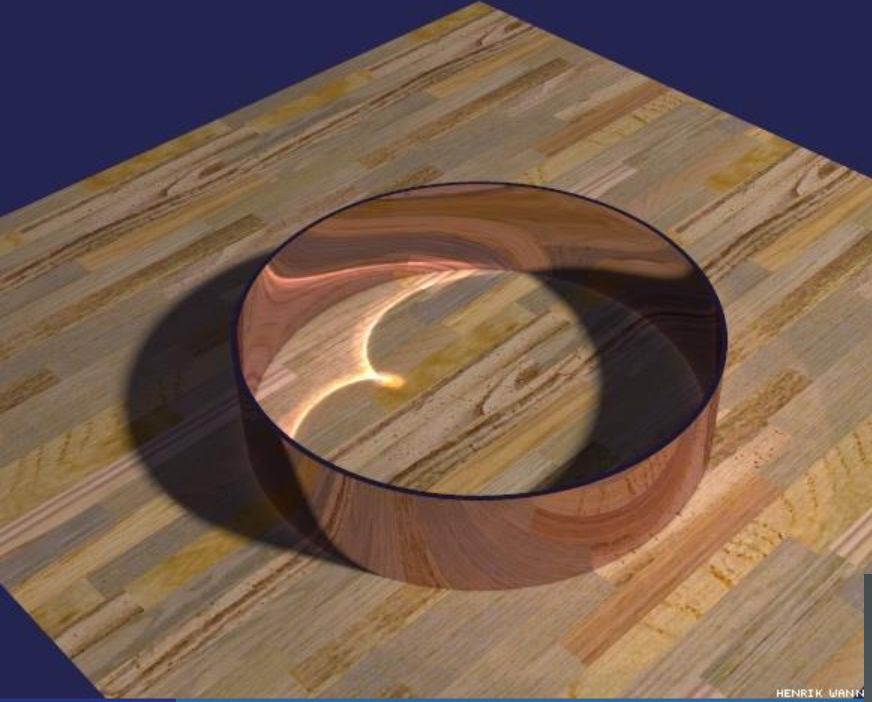
- In 2000-2010, e.g., trace 100K-10M photons. Grow sphere to include 20-100 photons. Artist select this.
- Today, e.g., combine progressive photon mapping with bidirectional path tracing.

- Creating Photon Maps:
  - Trace many many photons from light source. Store them in kd-tree where they hit surface (unless surface is very specular because standard ray tracing captures sharper reflections well). Then, use russian roulette to decide if the photon should be absorbed or specularly or diffusively reflected. Create both global map and caustics map. For the Caustics map, we send more of the photons towards reflective/refractive objects.
- Ray trace or path trace from eye:
  - At each intersection point **p**, compute direct illumination (shadow rays + local shading).
  - For indirect illumination: can grow sphere around **p** in caustics map to get caustics contribution and in global map to get slow-varying indirect illumination.
  - If final gather is used: instead of using global map directly, sample the indirect illumination around **p** by sampling the hemisphere with many many rays and **then** use the two photon maps where those rays hit a surface.
- Growing sphere:
  - Uses the kd-tree to grow a sphere around **p** until a fixed amount of photons are inside the sphere. Estimate outgoing radiance by using the material's brdf and the photons' powers and incoming directions.

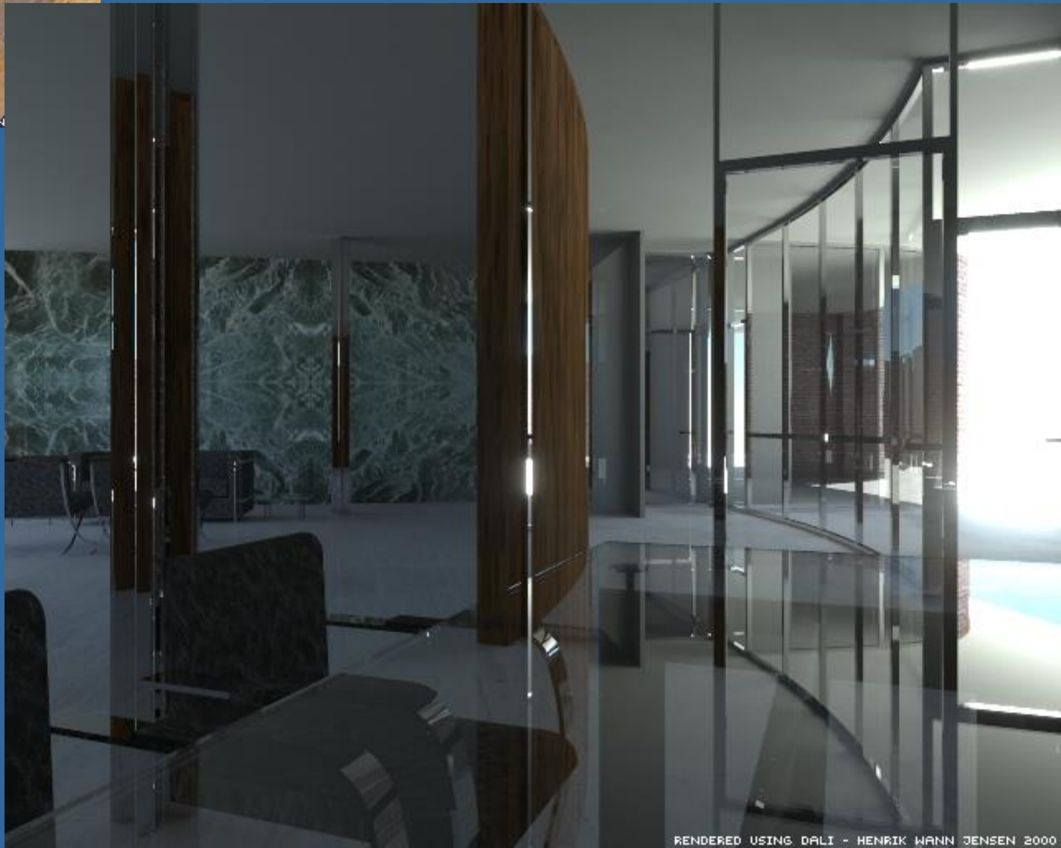
Or shorter summary:

1. Shoot photons from light source, and let them bounce around in the scene, and store them where they land (e.g. in a kD-tree).
2. Ray-tracing pass from the eye. Estimate radiance at each ray hit, by growing a sphere (at the hit point in the kD-tree) until it contains a predetermined #photons. Use the caustics map and the global map.

Standard photon mapping

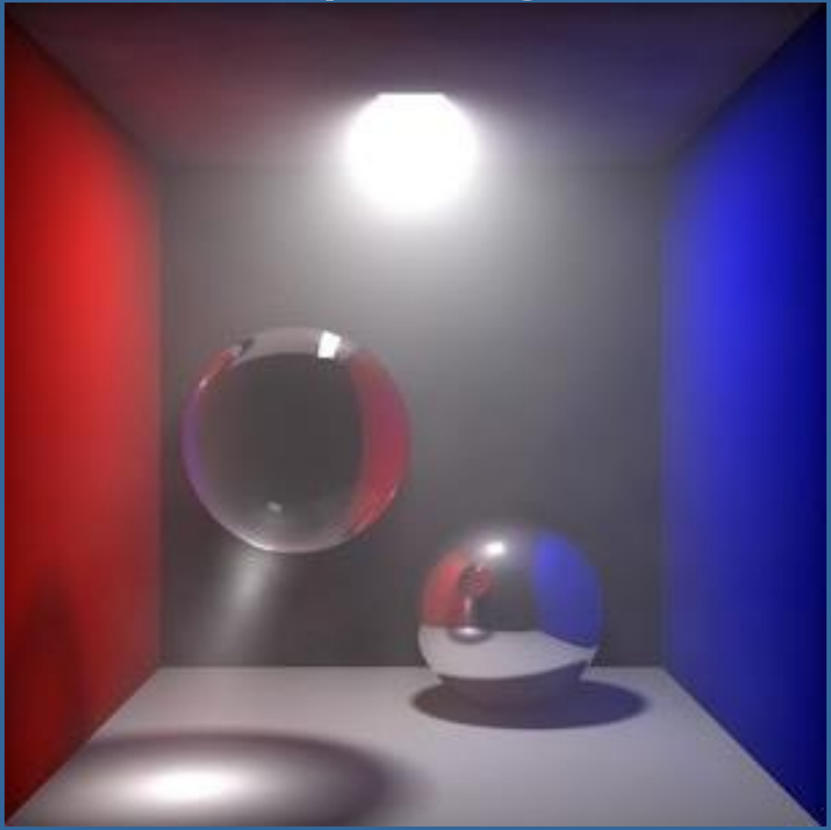


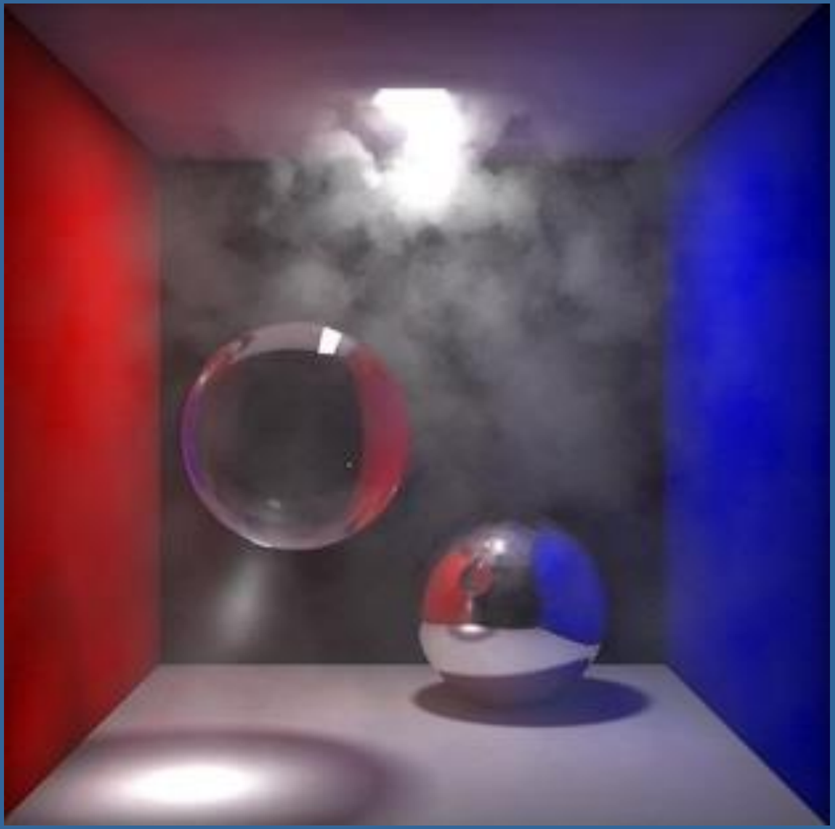
Caustics: concentrated reflected or refracted light



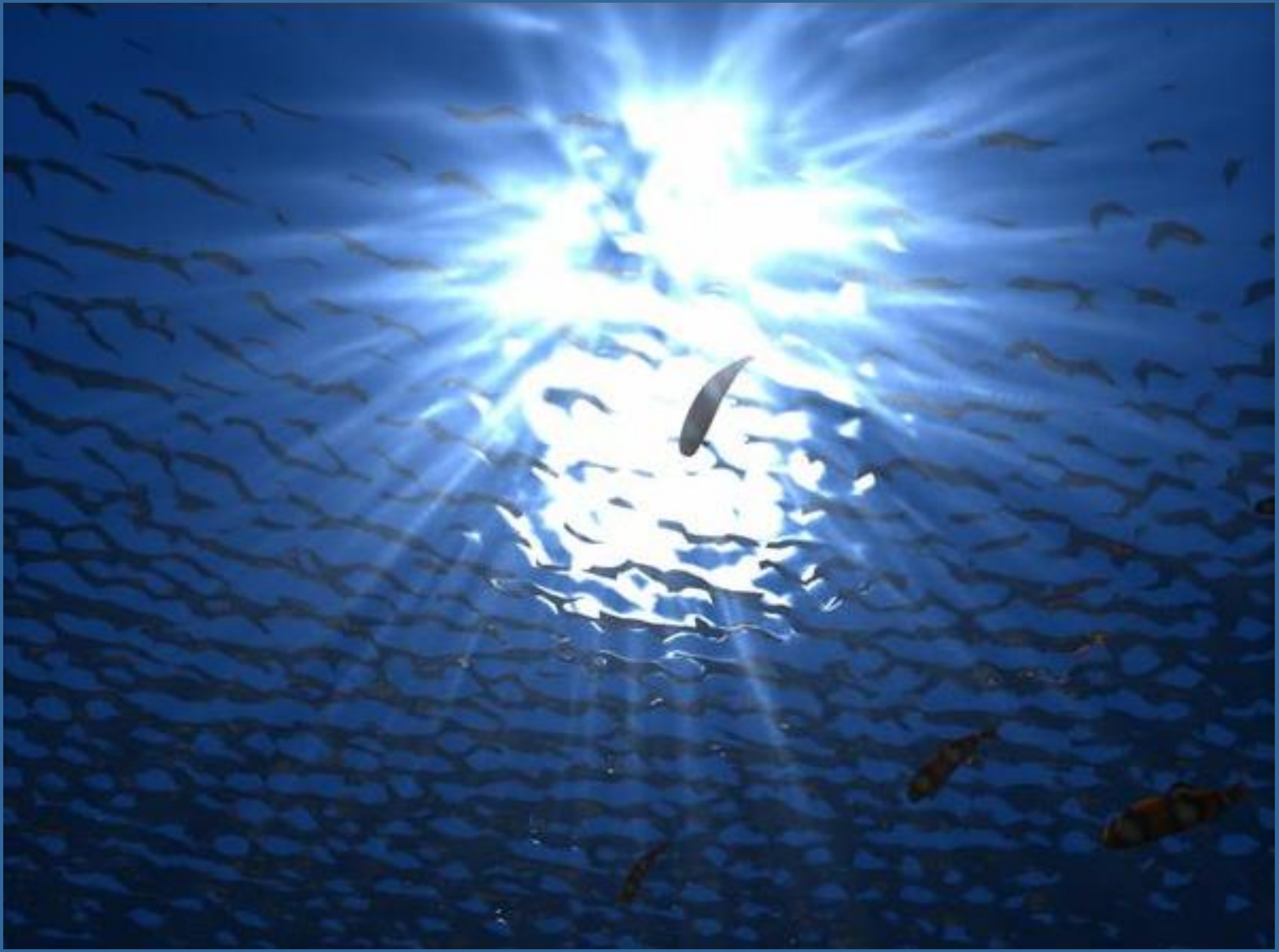
Extensions to photon mapping

- Participating media

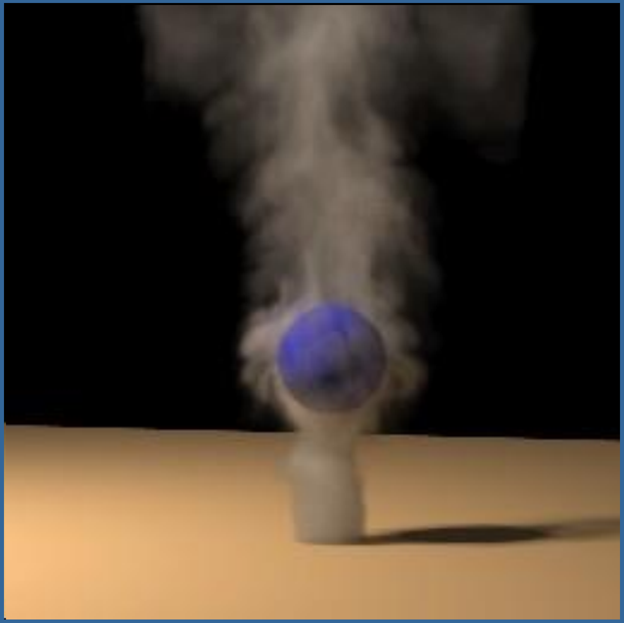




Another one on participating media



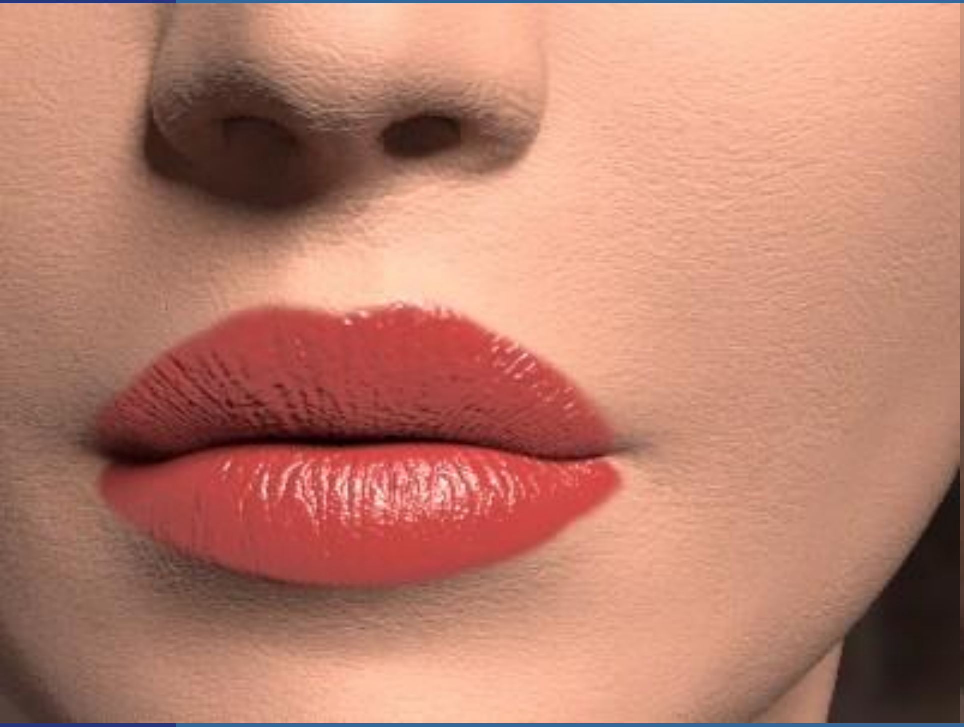
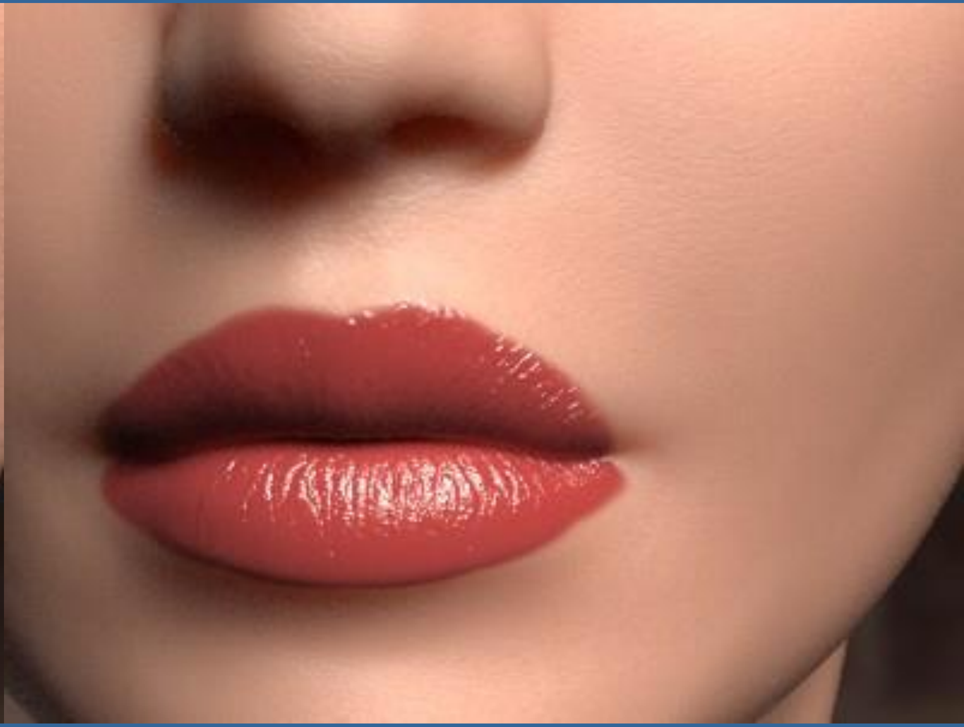
Smoke and photon mapping



Press for a movie

Photon mapping with subsurface scattering

- Photons enter the surface, and bounces around

Standard way

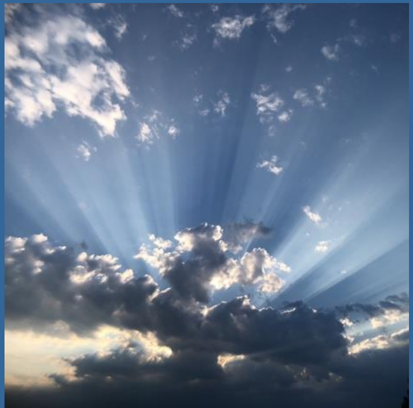
Subsurface scattering

Press for a movie

More GI methods…

Newer global-illumination methods:

- Vertex Connection and Merging
- Unified Path Sampling
  - Both are effectively identical techniques. They combine bidirectional path tracing and progressive photon mapping, and is particularly advantageous for specular- diffuse paths and specular-diffuse-specular paths (i.e. caustics and specular reflections of caustics)
  - Progressive photon mapping allows many photon-passes (e.g., to use more photons than fit in RAM).
- Unified Points, Beams, and Paths (UPBP). For volume rendering. Particular strength are crepuscular rays,volume caustics, and specular reflections of volume caustics. Implemented in Pixar’s RenderMan.
- See https://graphics.pixar.com/library/PathTracedMovies/paper.pdf
- Point-based Global Illumination, Tamy Boubekeur et al.



In conclusion

- If you want to get global illumination effects, then implement a path tracer
  - Very simple to implement
  - Good results – will eventually converge to correct result although may take very long time for caustics and hidden light sources (long light paths).
  - Advantage: fast for reasonable preview.
  - Noisy, so use together with AI denoising
- If you want a more advanced renderer:
  - Bidirectional path tracing – handles caustics and hidden light well.
  - Metropolis Light Transport –handles caustics even better but not popular for movie rendering due to temporal unstability - new specularities may be discovered and appear suddently
  - Photon Mapping – considered fast. Easy to implement for volumetric media. Use with bidirectional path tracing (”Vertex Connection and Merging” or ”Unified Path Sampling”)

THE END

What you need to know

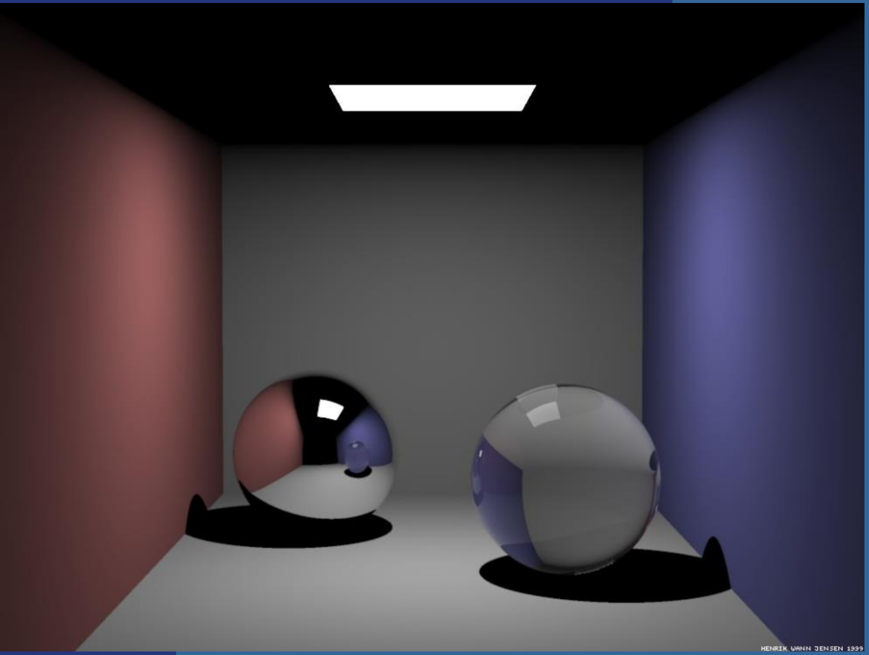
- The rendering equation
  - Be able to explain all its components

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

- Monte Carlo sampling:
  - The naïve way (an exponentially growing ray tree)
  - Path tracing
    - Why it is good, compared to naive monte-carlo sampling
    - The overall algorithm (on a high level as in these slides).
  - Photon Mapping
    - The short summary of the algorithm
    - Why 2 maps (global + caustics) are needed.
  - Bidirectional Path Tracing, Metropolis Light Transport
    - Just their names. Don't need to know the algorithms.
- Denoising by Final Gather or AI
    - Final Gather – sample indirect illumination at some positions in the world (these are the final-gather points). Then, at each ray hit, estimate indirect illumination by interpolation from nearby final-gather points.
    - AI: use some existing Deep Neural Network solution that denoises your images for your kind of scenes.

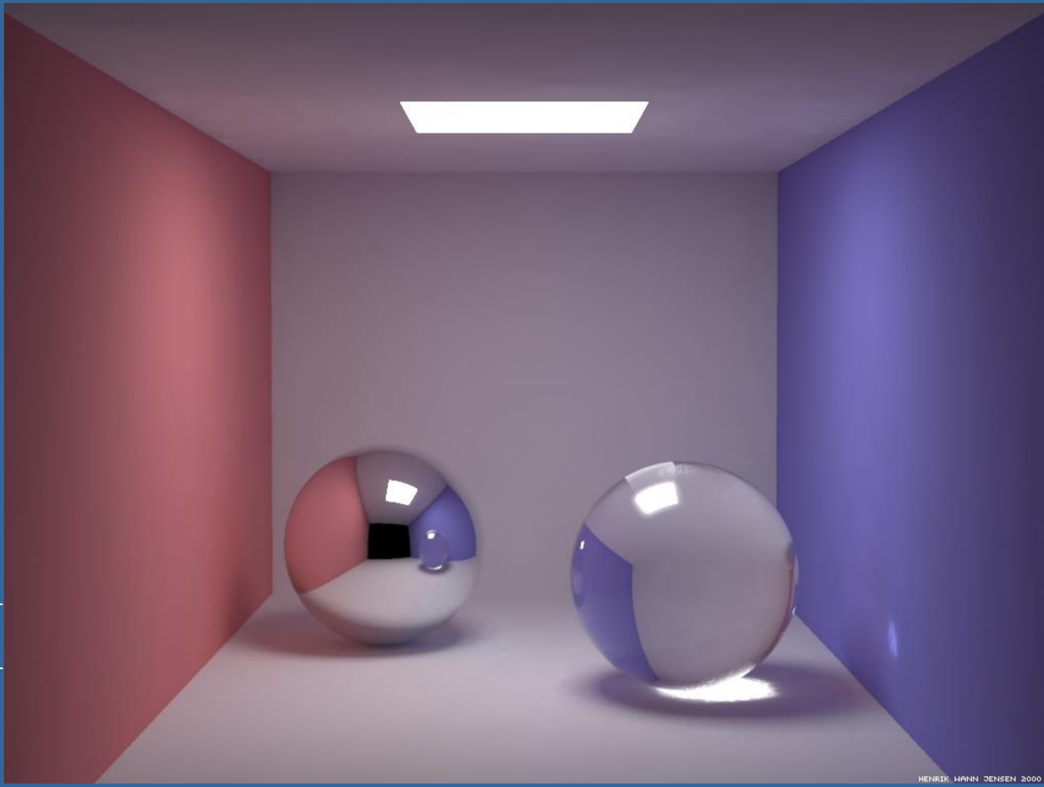
The most important slides from today's lecture →

Isn't ray tracing enough?



Ray tracing

Effects to note in Global Illumination image:
1) Indirect lighting (light reaches the roof)
2) Soft shadows (light source has area)
3) Color bleeding (example: roof is red near red wall) (same as 1)
4) Caustics (concentration of refracted light through glass ball)
5) Materials have no ambient component



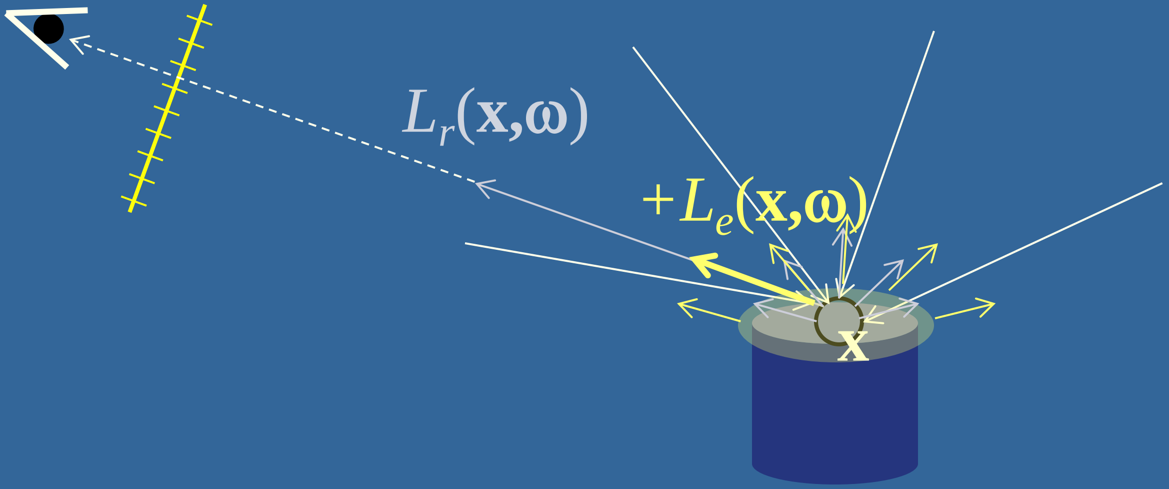
Global Illumination

Which are the differences?

Images courtesy of Henrik Wann Jensen

The rendering equation

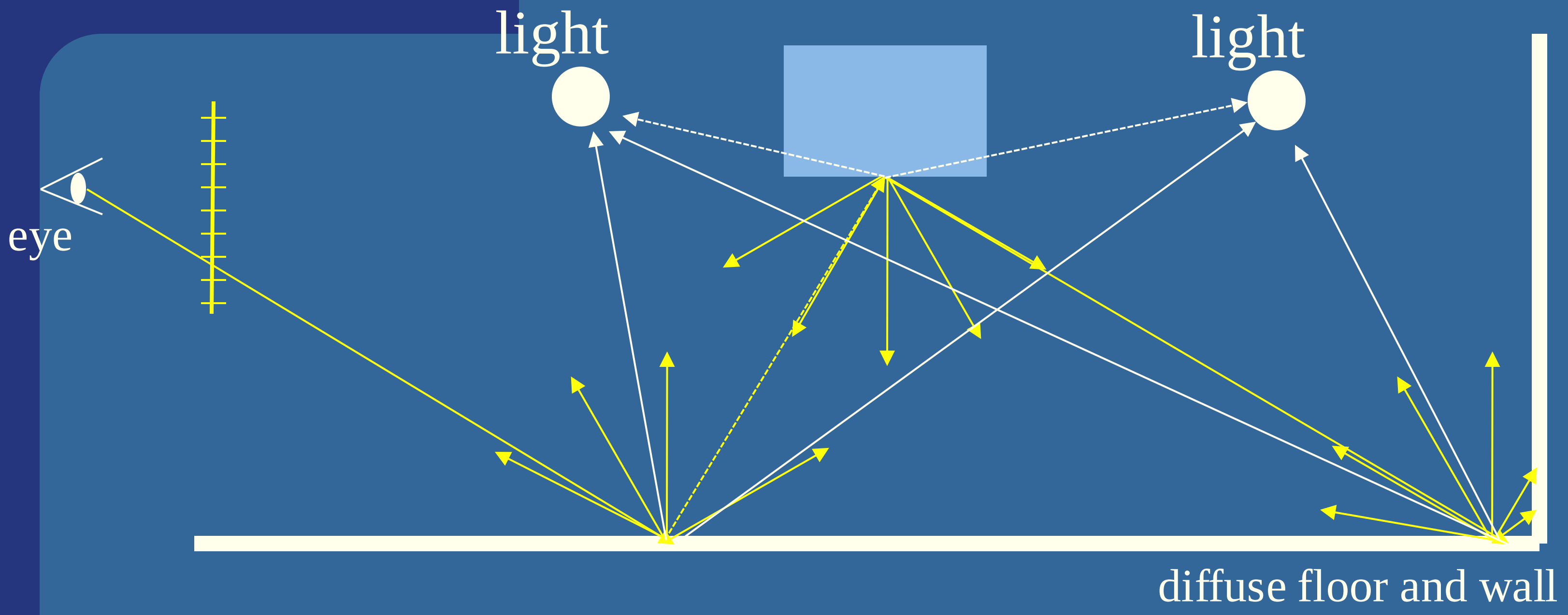
- Paper by Kajiya, 1986.
- Is the basis for all global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$
  - outgoing=emitted+reflected radiance



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

- f_r is the BRDF, ω' is incoming direction, $\mathbf{n}$ is normal at point $\mathbf{x}$, Ω is hemisphere "around" $\mathbf{x}$ and $\mathbf{n}$, L_i is incoming radiance

Monte Carlo Ray Tracing – direct + indirect illumination

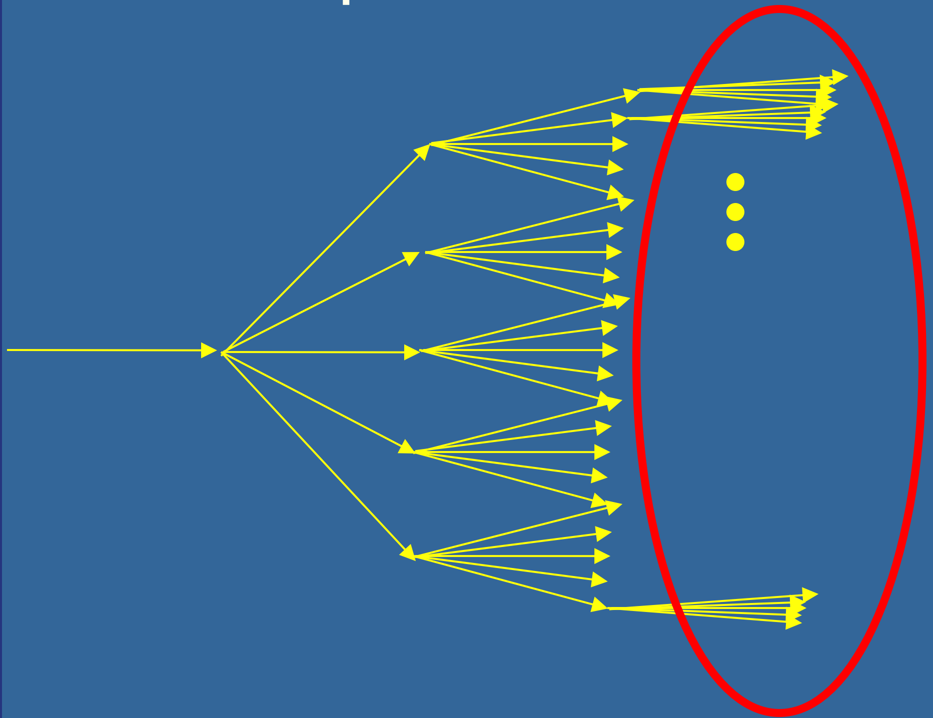


- Sample indirect illumination by shooting sample rays over the hemisphere, at each hit.

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

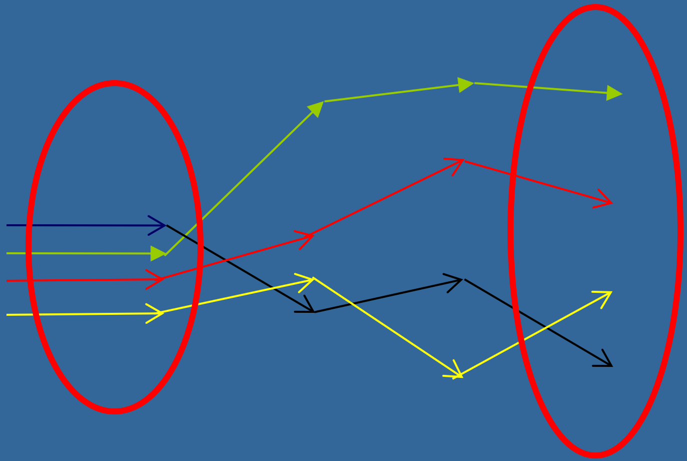
Monte Carlo Ray Tracing (naïvely)

- The indirect-illumination sampling gives a ray tree with most rays at the bottom level. This is bad since these rays have the lowest influence on the pixel color.



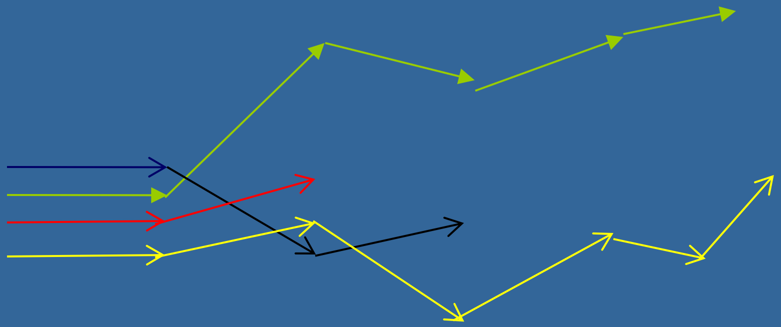
PathTracing
– one efficient Monte-Carlo Ray-Tracing solution

- Path Tracing instead only traces one of the possible ray paths at a time. This is done by randomly selecting only one sample direction at a bounce. Hundreds of paths per pixel are traced.



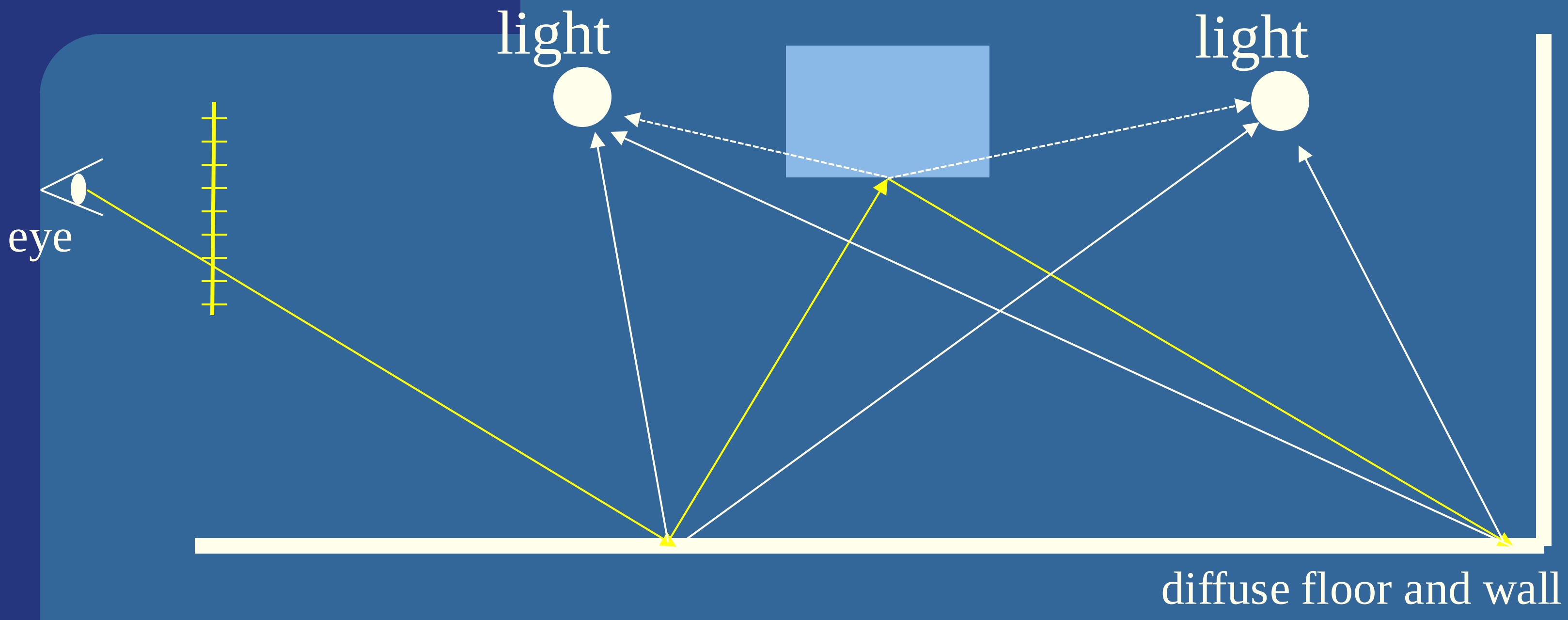
Equally number of rays are traced at each level

Or:



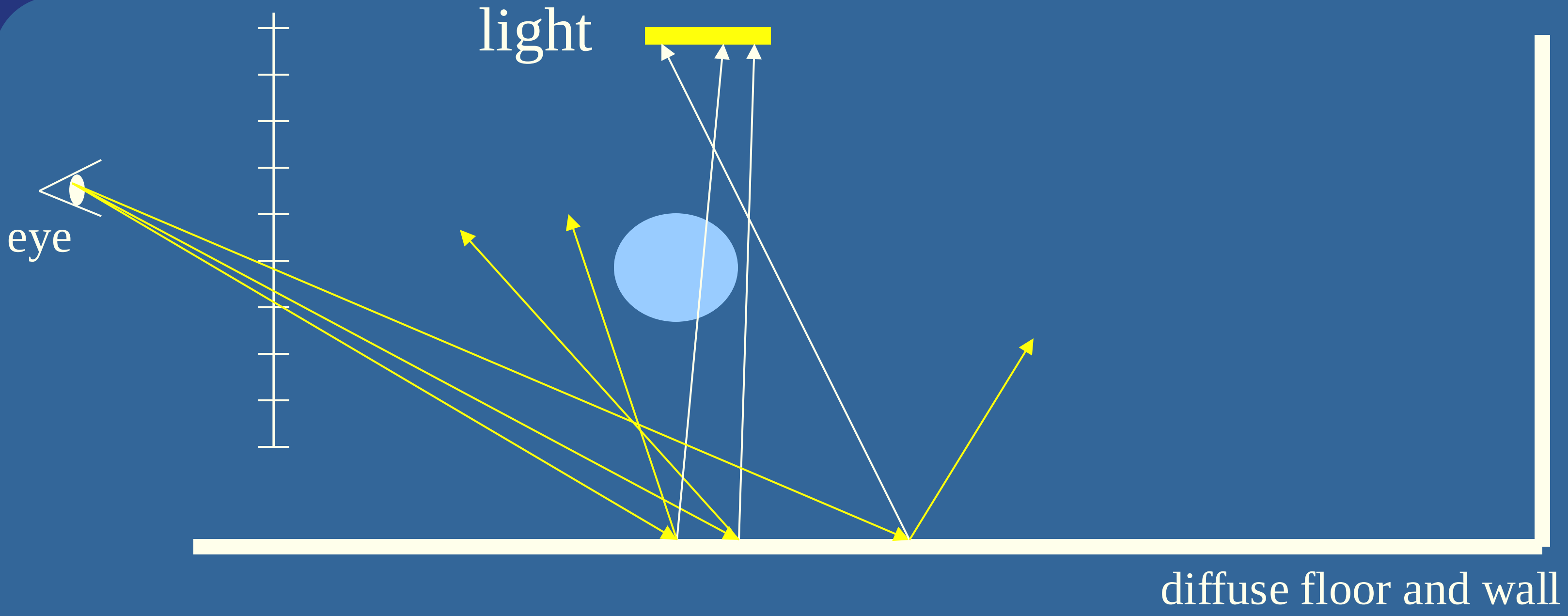
Even smarter: terminate path with some probablility after each level, since they have decreasing importance to final pixel color.

Path Tracing – indirect + direct illumination



- Shoot many paths per pixel (the image just shows one light path).
  - At each intersection,
    - Shoot one shadow ray per light source
      - at random position on light, for area/volumetric light sources
    - and randomly select one new ray direction.

Path Tracing and area lights



- For area light sources, shoot the shadow ray to one random position on the area light. This gives soft shadows when many paths are averaged for the pixel.
- Example: Three paths for one pixel
  - At each ray intersection,
    - Pick ***one*** random position on light source
    - Send one random ray bounce to continue the path...

Path tracing: Summary

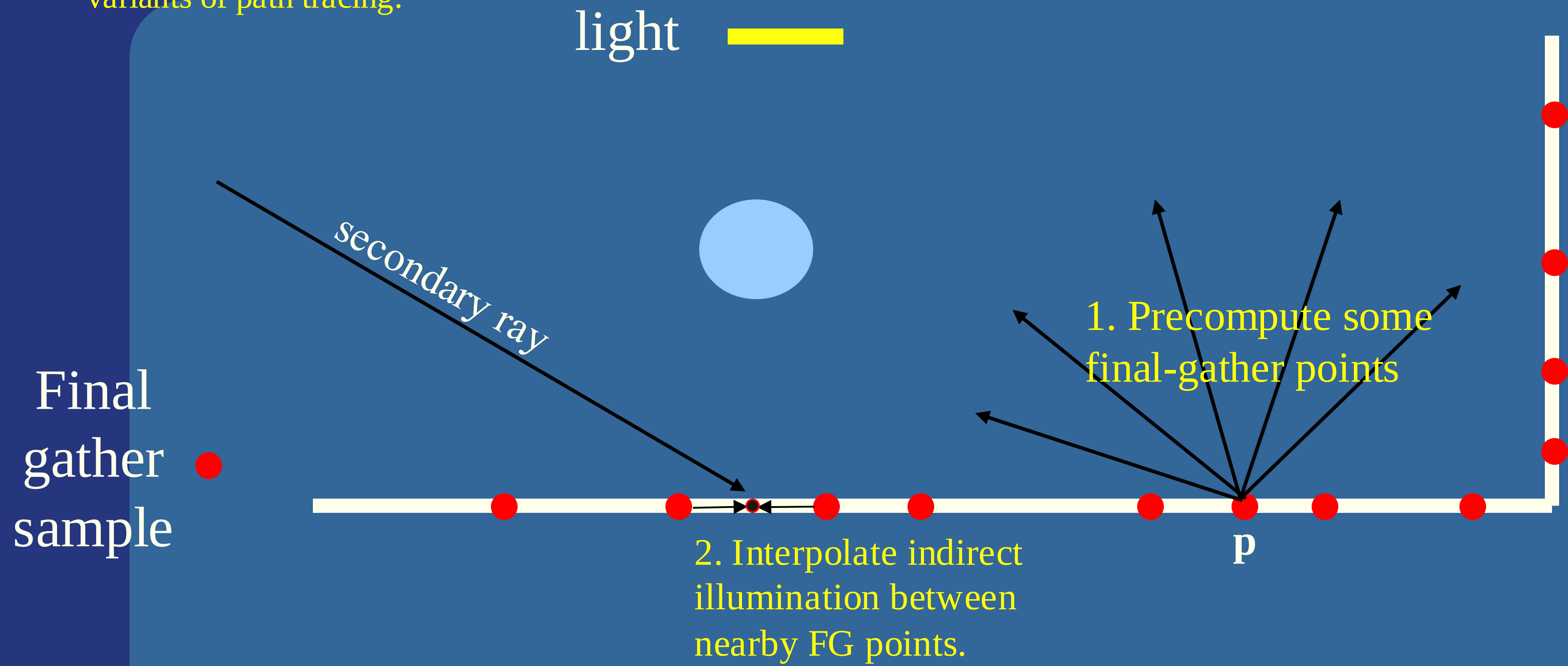
- Uses Monte Carlo sampling to solve integration:
  - by shooting many random ray ***paths*** over the integral domain.
  - Algorithm:
    - For each pixel, // we will shoot a number of paths:
      - For each path, generate the primary ray:
      - Repeat {
        1. Trace the ray. At hitpoint:
        2. Shoot one shadow ray and compute local lighting.
        3. Sample indirect illumination randomly over the possible reflection/refraction directions by generating **one** such new ray.
      - } until the path is randomly terminated (or the ray does not hit anything).
- Shorter summary: shoot many paths per pixel, by randomly choosing **one** new ray at each interaction with surface + **one** shadow ray per light. Terminate the path with a random probability

Final Gather

Popular for naïve monte carlo ray tracing and photon mapping but not for variants of path tracing.

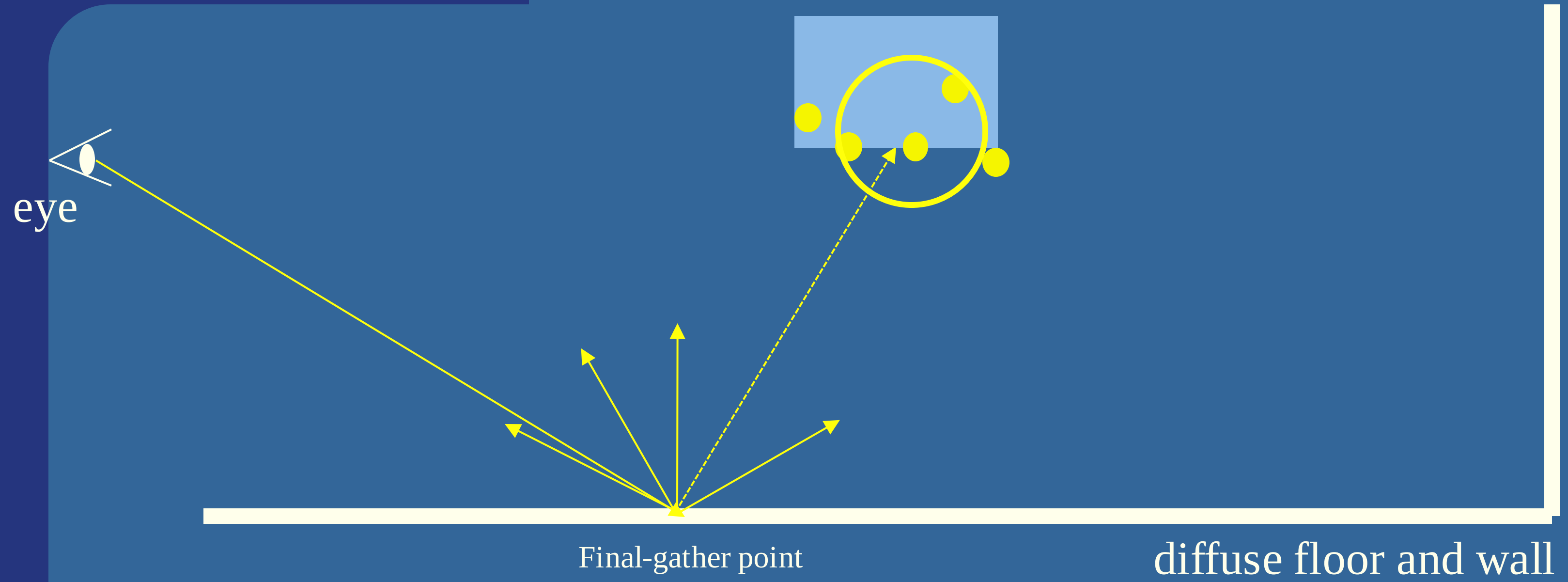
Idea and good answer:

- Compute indirect illumination somehow, but only at sparse set of positions (final gather points) in the scene.
- Estimate indirect illumination for other positions by interpolation from nearby final-gather points


light
secondary ray
1. Precompute some final-gather points
Final gather sample
p
2. Interpolate indirect illumination between nearby FG points.

- Many versions of Final Gathering exist.
- E.g., to compute final-gather point **p**:
  - Send thousand(s) random rays out from **p** to sample indirect illumination
- To use during ray tracing: interpolate global illumination between nearby Final Gather points, to estimate incoming radiance at the ray's intersection point.
- Does not matter much if indirect illumination is blotchy for secondary rays.

Final Gather with Photon Mapping



- Too noicy to use the <u>global</u> map for direct visualization
- Remember: eye rays are recursively traced (via reflections/refractions) until a diffuse hit, **p**. There, we want to estimate slow-varying indirect illumination.
  - Instead of growing sphere in global map at **p**, Final Gather shoots 100-1000 indirect rays from **p** and grows sphere in the global map and also caustics map, where each of those rays end at a diffuse surface. Or interpolate from nearby already computed final-gather points.

Photon Mapping - Summary

- Creating Photon Maps:
  - Trace many many photons from light source. Store them in kd-tree where they hit surface (unless surface is very specular because standard ray tracing captures sharper reflections well). Then, use russian roulette to decide if the photon should be absorbed or specularly or diffusively reflected. Create both global map and caustics map. For the Caustics map, we send more of the photons towards reflective/refractive objects.
- Ray trace or path trace from eye:
  - At each intersection point **p**, compute direct illumination (shadow rays + local shading).
  - For indirect illumination: can grow sphere around **p** in caustics map to get caustics contribution and in global map to get slow-varying indirect illumination.
  - If final gather is used: instead of using global map directly, sample the indirect illumination around **p** by sampling the hemisphere with many many rays and **then** use the two photon maps where those rays hit a surface.
- Growing sphere:
  - Uses the kd-tree to grow a sphere around **p** until a fixed amount of photons are inside the sphere. Estimate outgoing radiance by using the material's brdf and the photons' powers and incoming directions.

Or shorter summary:

1. Shoot photons from light source, and let them bounce around in the scene, and store them where they land (e.g. in a kD-tree).
2. Ray-tracing pass from the eye. Estimate radiance at each ray hit, by growing a sphere (at the hit point in the kD-tree) until it contains a predetermined #photons. Use the caustics map and the global map.