

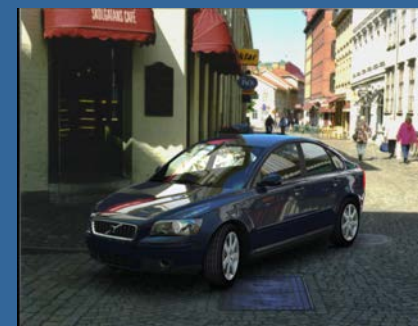
An introduction to Global Illumination



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/](http://www.cse.chalmers.se/edu/course/TDA362/Advanced%20Computer%20Graphics/)
- ~13 seminars in total, sp4
- Project (no exam)
 - Self or in groups
- Project examples include:
 - realistic explosions, clouds, smoke, procedural textures
 - fractal mountains, CUDA program, Spherical Harmonics, SSAO, Displacement mapping, Collision detection
 - 3D Game
 - real-time ray tracer, enhanced path tracing.
 - HDRI



GFX Companies Gothenburg

3D software development:

Rapid Images
Epic Games
Smart Eye AB,
EON Reality,
Spark Vision
(Autodesk)
MindArk
Mentice
Vizendo
Surgical Science
Combitech
Fraunhofer (Chalmers Teknikpark)
RD&T Technology

And many more that I have forgotten
now...

For graphics artists:

Rapid Images
zoink
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 (Sthlm)
Massive (Malmö)

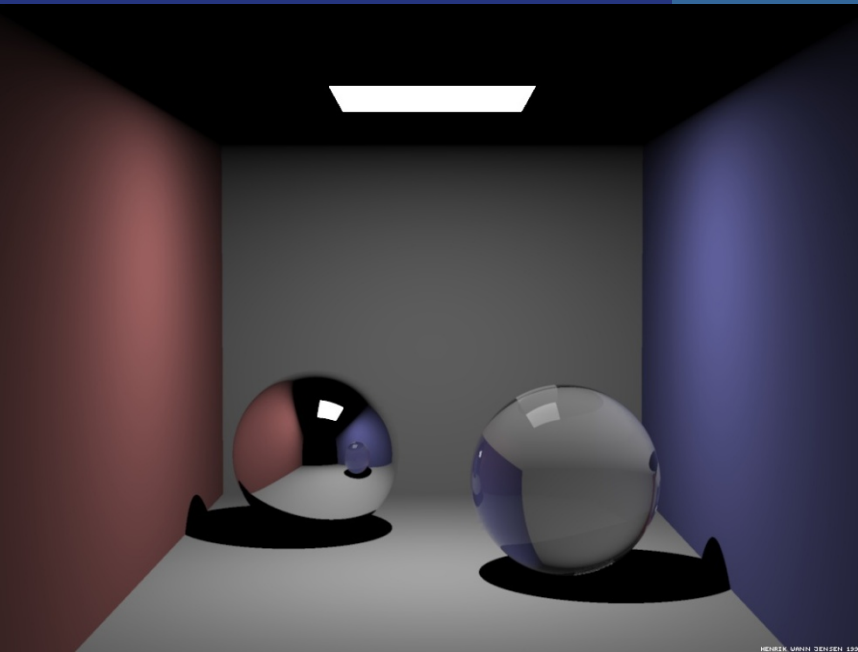
Architects

Arcitec – (Sthlm)–
visualization of buildings for
architects

Architects, graphics artists:

White
Wingårdhs
Volvo Personvagnar
Semcon
Ramböll
Zynka
CAP AB

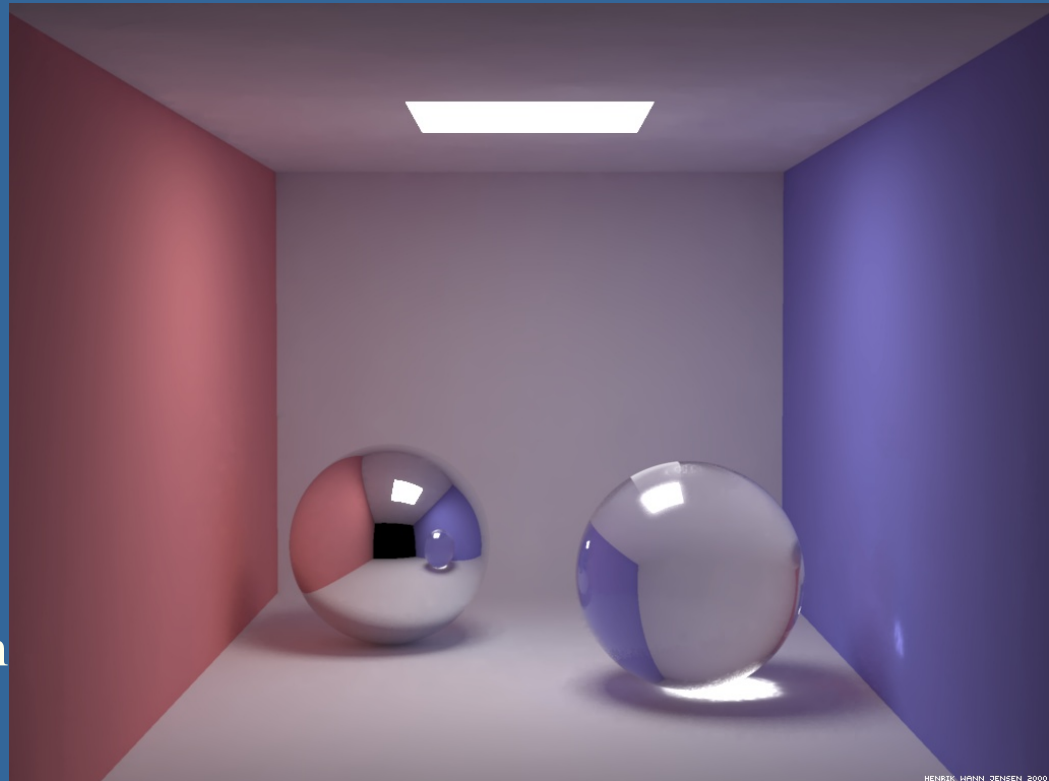
Isn't classic ray tracing enough?



Whitted Ray tracing
(reflections, refractions, shadows)

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



Which are
the differences?

Global
Illumination

Global Illumination

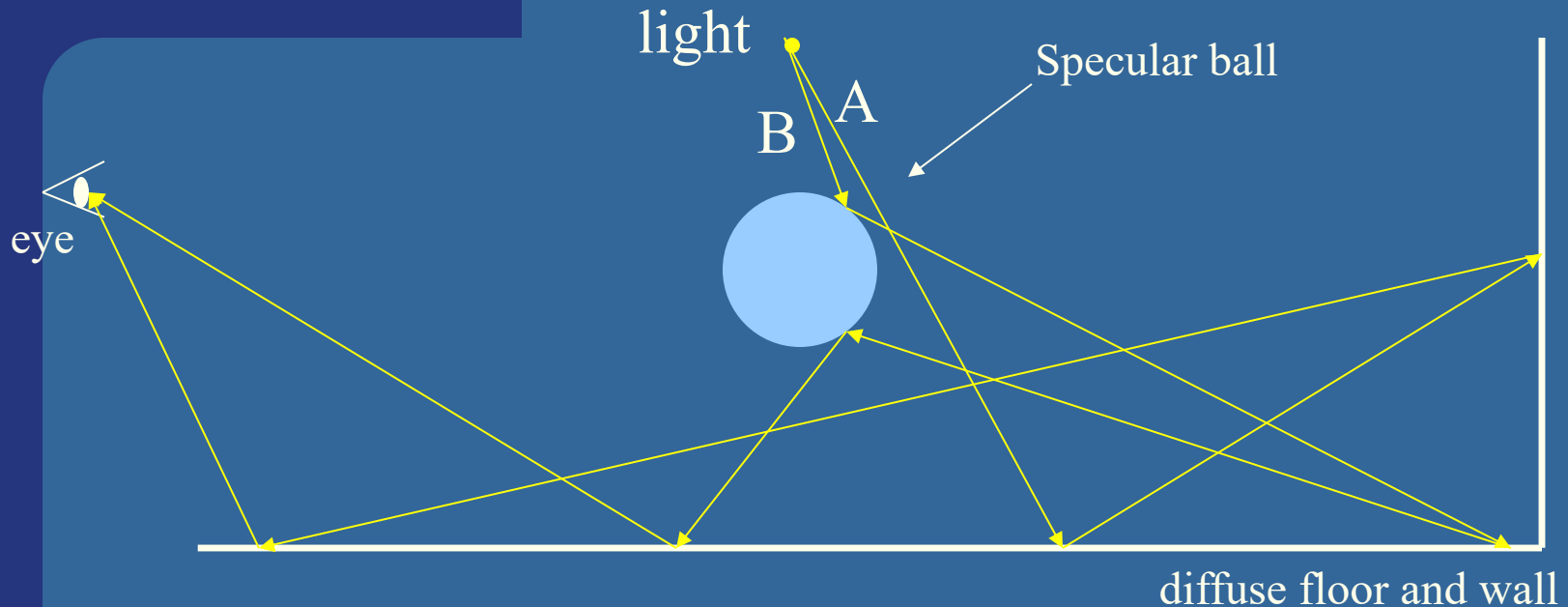
- The goal: **follow** all **photon/ray** bounces through a scene, in order to render images with all kinds of light paths.
- This will give incredibly realistic images
- This lecture will treat:
 - Background
 - **Monte Carlo** ray tracing
 - Path tracing
 - Bidirectional Path tracing
 - Photon mapping
 - Final Gather
- Great books on global illumination and photon mapping:
 - Pharr, Humphreys, Physically Based Rendering, 2010
 - With source code.
 - Henrik Wann Jensen, *Realistic Image Synthesis using Photon Mapping*, AK Peters, 2001.

Light transport notation

Useful tool for thinking about global illumination (GI)

- Follow light paths
- The endpoints of straight paths can be:
 - L : light source
 - E : the eye
 - S : a specular reflection
 - D: a diffuse reflection
 - G: a glossy reflection
- Regular expressions can be used:
 - $(K)^+$: one or more of K
 - $(K)^*$: zero or more of K
 - $(K)?$: zero or one of K
 - $(K \mid M)$: a K or an M event

Examples of light transport notation



- The following expression describes all light paths to the eye in this scene: $L(S|D)^*E$
- Path A: LDDDE
- Path B: LSDSDE

Light transportation

What for?

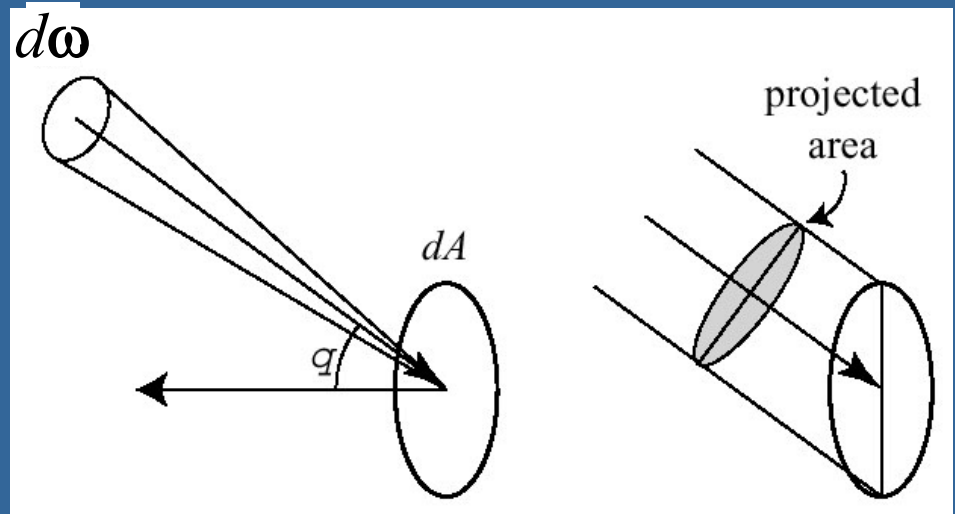
- The ultimate goal is to simulate all light paths: $L(S|D|G)^*E$
- Using this notation, we can find what ray tracing can handle:
 - $LDS^*E \mid LS^*E = LD?S^*E$
 - This is clearly not $L(S|D|G)^*E$

Background: Radiance

- 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 (per unit solid angle per unit projected area)
- Five-dimensional (or 6, including wavelength):
 - Position (3)
 - Direction (2) – horizontal + vertical angle
- Radiance is "power per unit projected area per unit solid angle"

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

Uses differentials, so the cone of the solid angle becomes infinitesimally small: a ray



Background:

The rendering equation

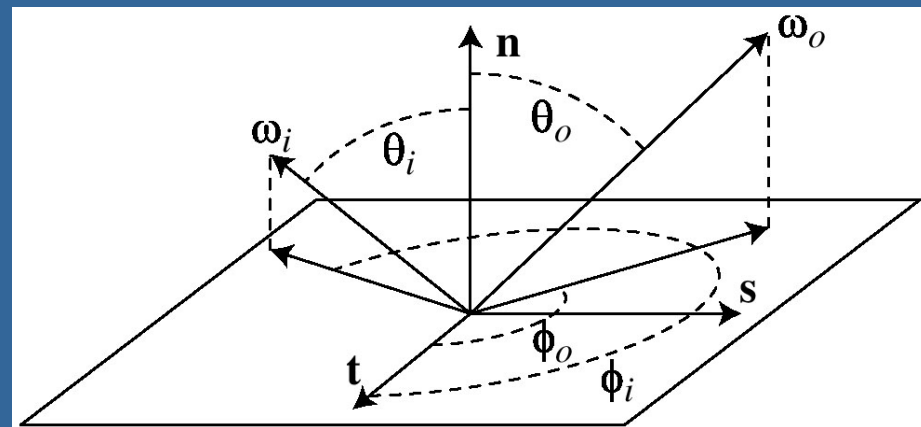
- Paper by Kajiya, 1986 (see course website).
- Is the basis for all rendering, but especially for global illumination algorithms
- $L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + L_r(\mathbf{x}, \omega)$ (slightly different terminology than Kajiya)
 - outgoing = emitted + reflected radiance
 -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

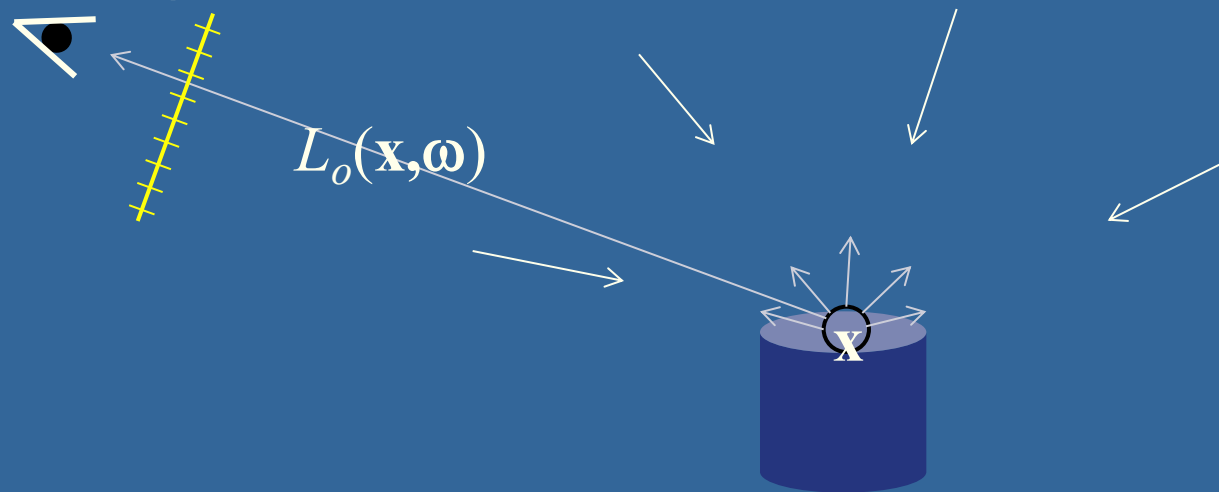
Background: Briefly about BRDFs

- Bidirectional Reflection Distribution Function
- A more accurate description of material properties
- What it describes: the probability that an incoming photon will leave in a particular outgoing direction
- i is incoming
- o is outgoing
- Huge topic!
- Many different ways to get these
 - Measurement
 - Hacks: amb+diff+spec



Radiance/strålning

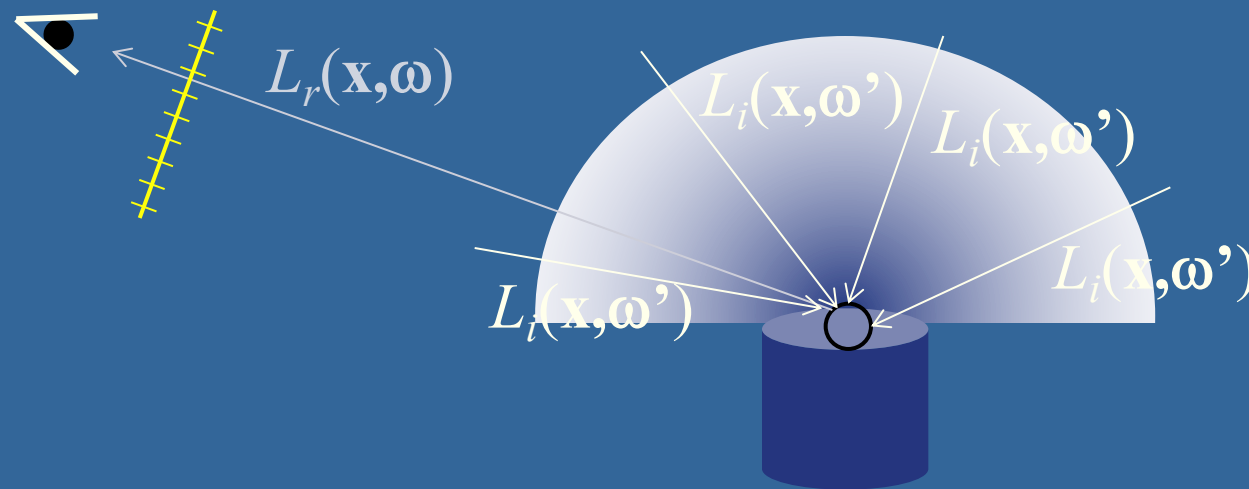
- Radiance, L : a radiometric term. What we store in a pixel is the radiance towards the eye
 - the amount of electromagnetic radiation leaving or arriving at a point on a surface



- L_o = outgoing radiation from a point to a certain direction
- Radiation = color and its intensity, i.e., rgb-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



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

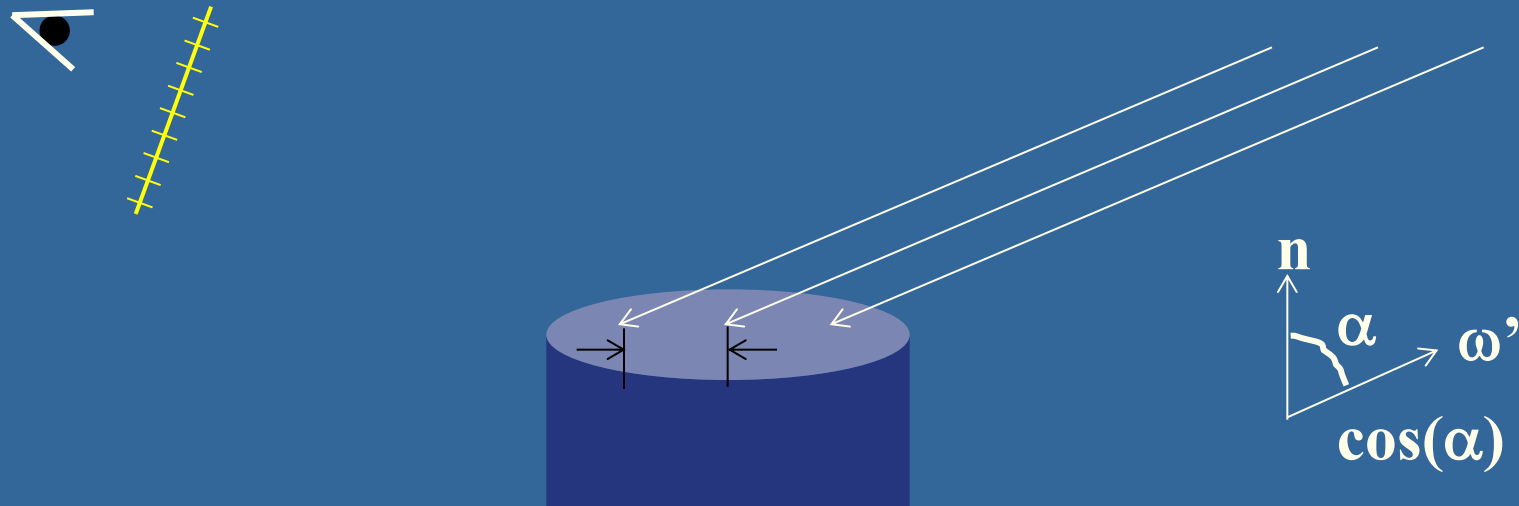
$$L_o = L_e + \int_{\Omega} \underline{f_r(\mathbf{x}, \omega, \omega') L_i(\mathbf{x}, \omega') (\omega' \cdot \mathbf{n})} d\omega'$$

- 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

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

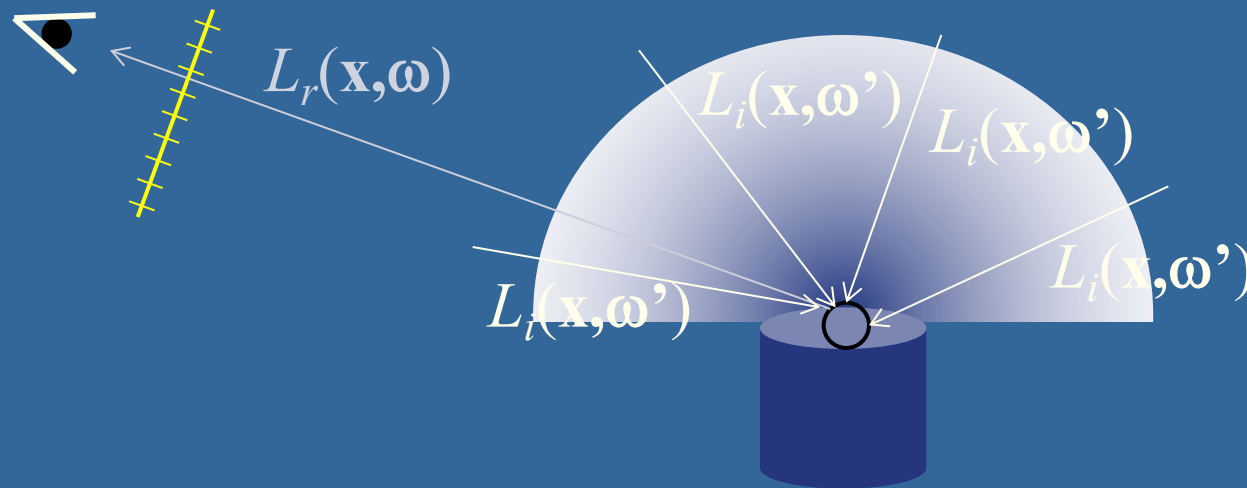


$$L_o = L_e + \int_{\Omega} f_r(\mathbf{x}, \omega, \omega') L_i(\mathbf{x}, \omega') (\omega' \cdot \mathbf{n}) d\omega'$$

- 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

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



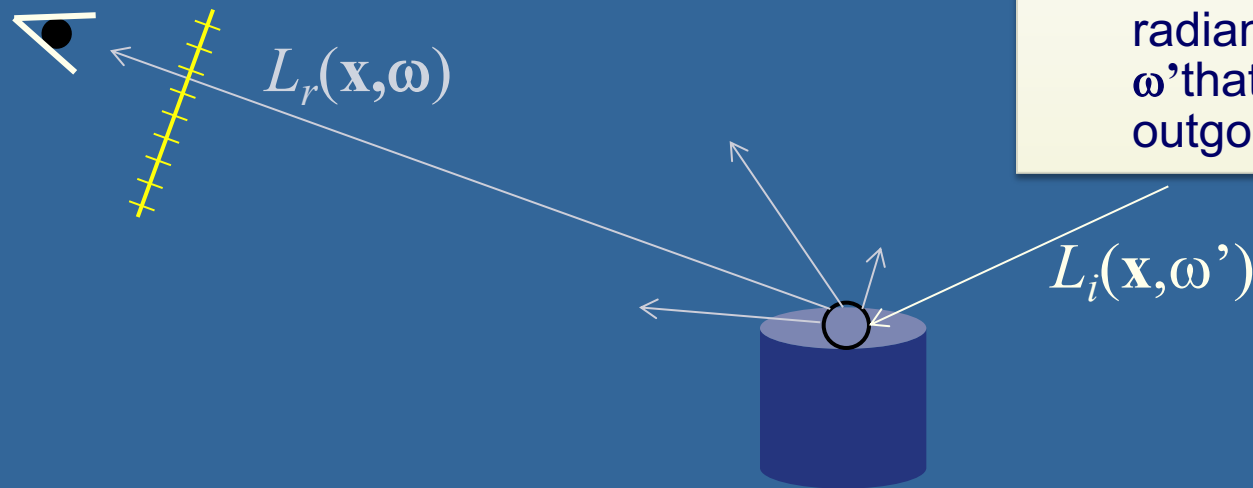
$$L_o = L_e + \int_{\Omega} f_r(\mathbf{x}, \omega, \omega') \underline{L_i(\mathbf{x}, \omega')} (\omega' \cdot \mathbf{n}) d\omega'$$

- 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:

$f_r(\mathbf{x}, \omega, \omega') =$

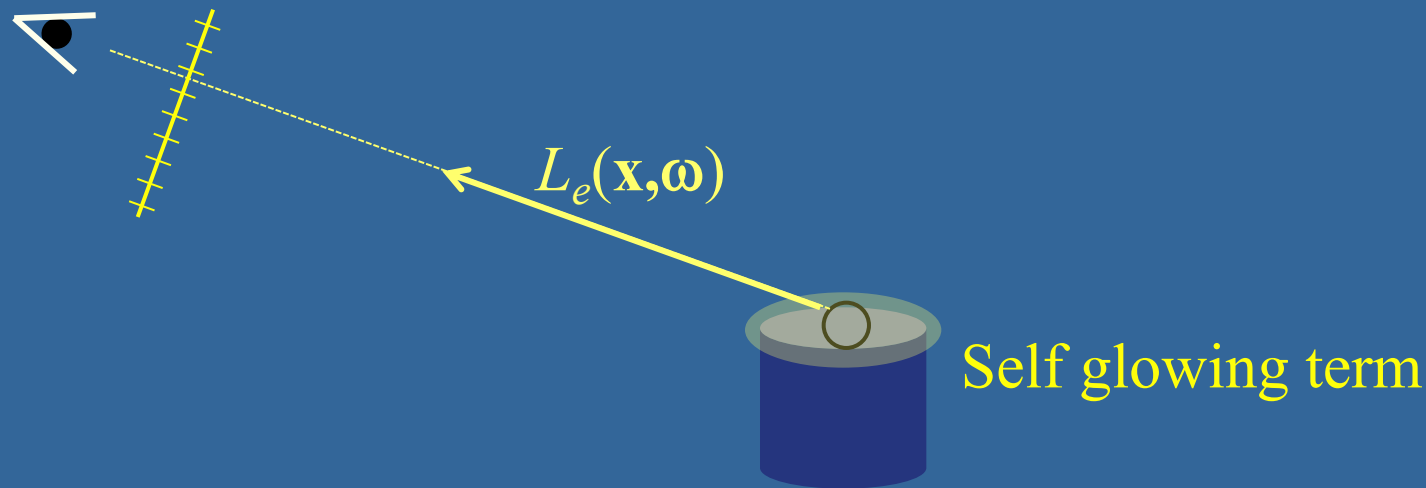
"How much of incoming radiance, L_i , from direction ω' that leaves in an outgoing direction ω "

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

- 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

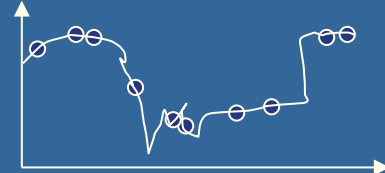
Many GI algorithms are built on Monte Carlo Integration

- Integral in rendering equation:

- Hard to evaluate numerically
- But we can sample it.

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

- 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 $\text{mean} \cdot (b-a)$
- Thus, focus on estimating mean of $f(x)$
- Idea: sample f at n uniformly distributed random locations, x_i :

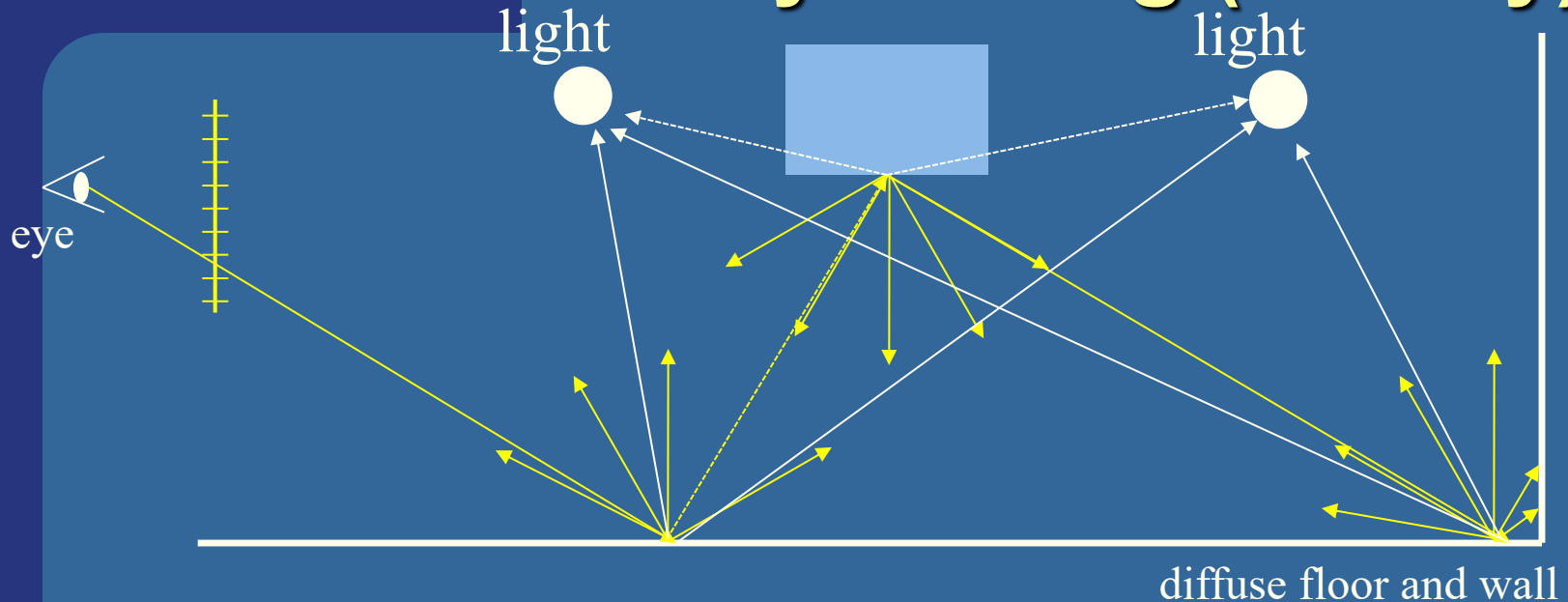
$$I_{MC} = (b-a) \frac{1}{n} \sum_{i=1}^n f(x_i)$$

Monte Carlo estimate

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

$$\sigma \propto \frac{1}{\sqrt{n}}$$

Monte Carlo Ray Tracing (naïvely)

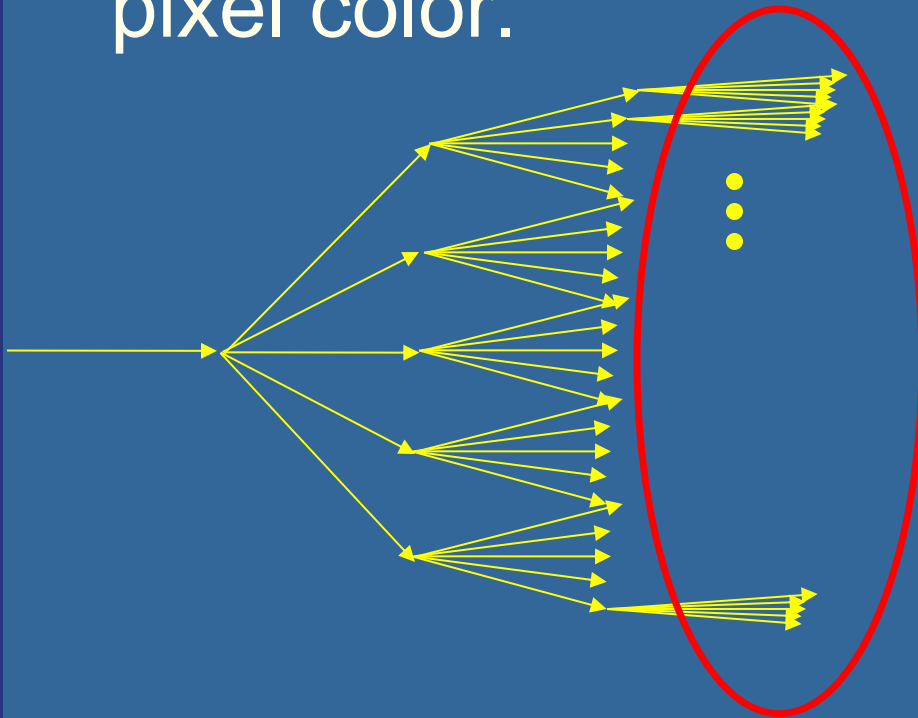


- (Compute local lighting as usual, with a shadow ray per light.)
- Sample indirect illumination by shooting sample rays over the hemisphere, at each hit.

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

Monte Carlo Ray Tracing (naïvely)

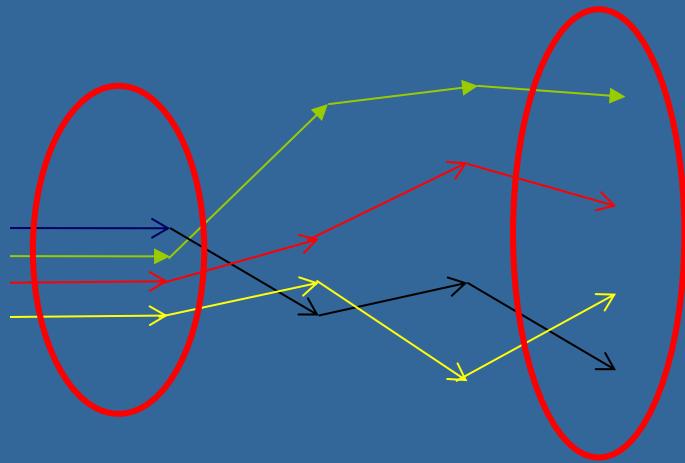
- This gives a ray tree with most rays at the bottom level. This is bad since these rays have the lowest influence on the pixel color.



PathTracing

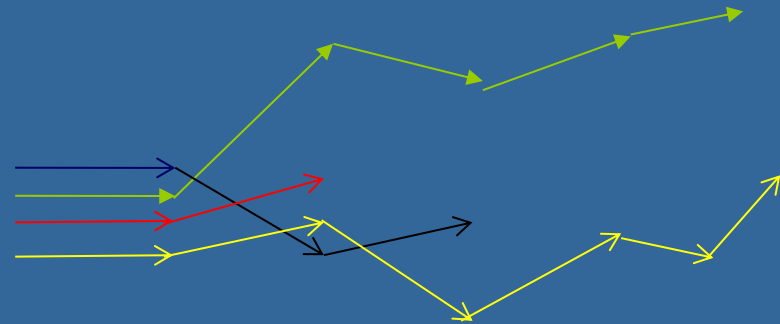
– one efficient Monte-Carlo Ray-Tracing solution

- 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

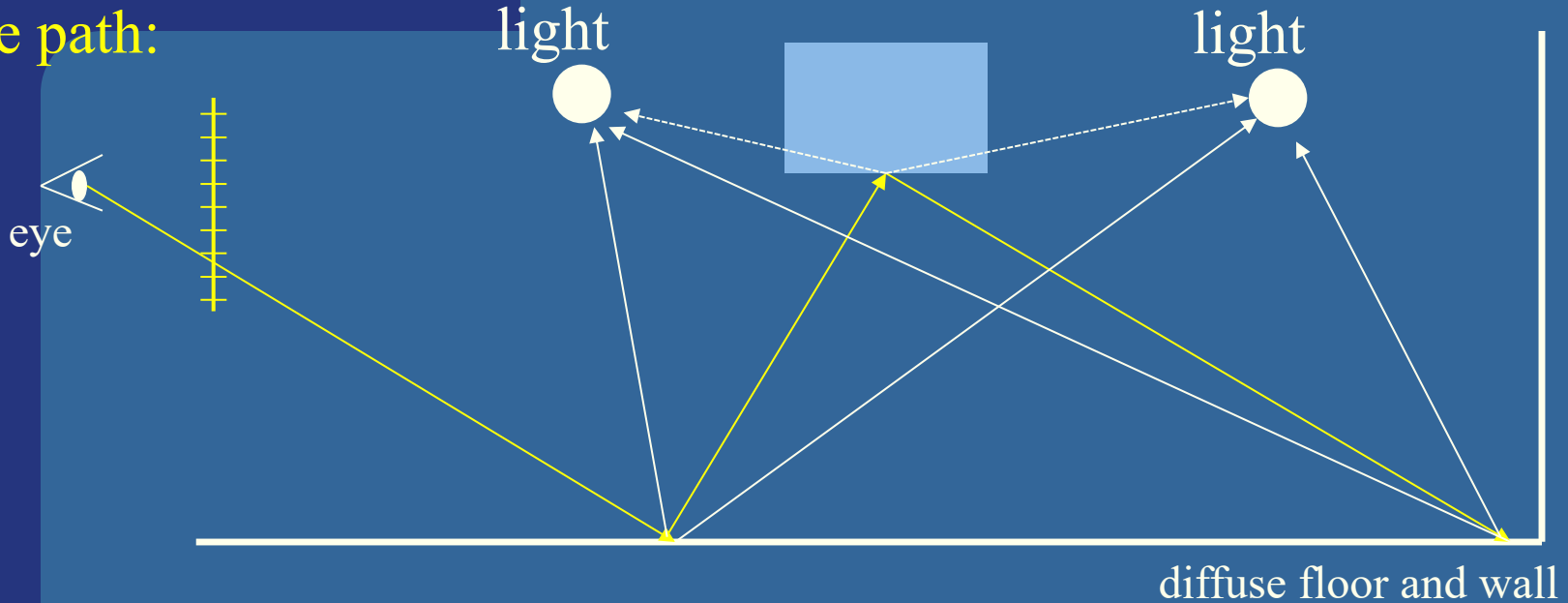
Or:



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

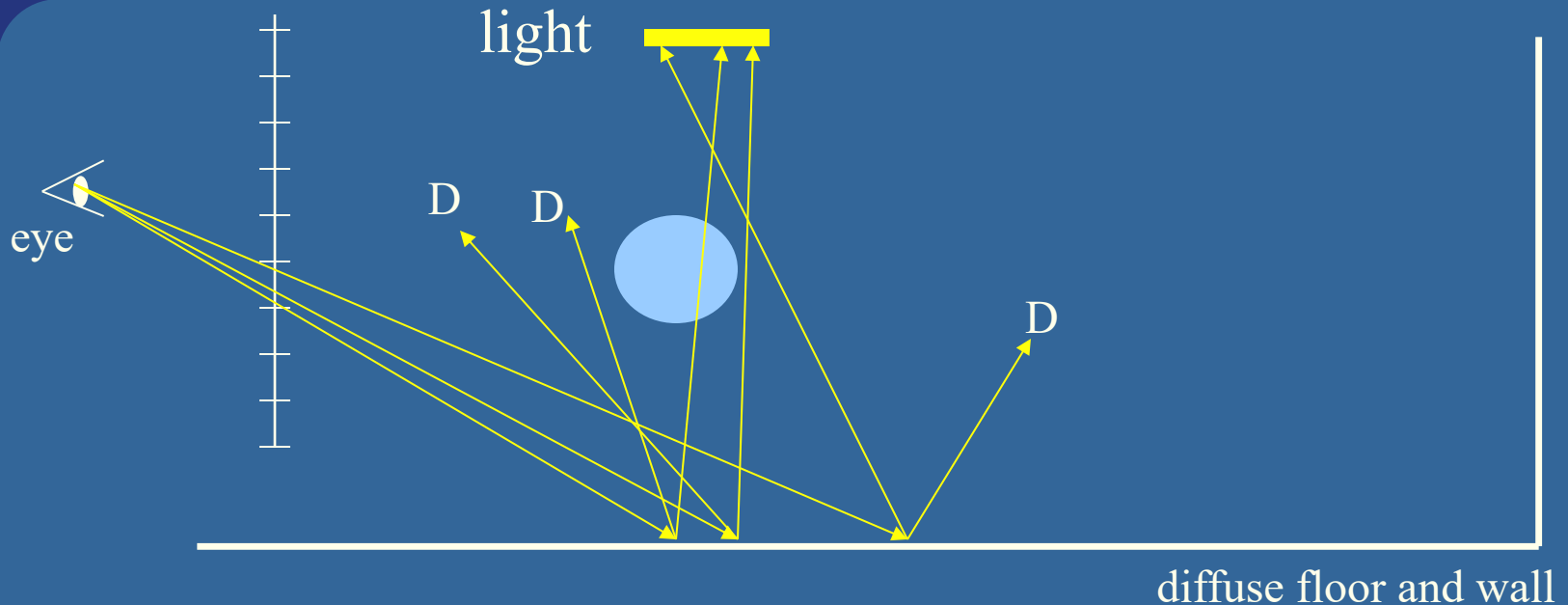
Path Tracing – indirect + direct illumination

One path:



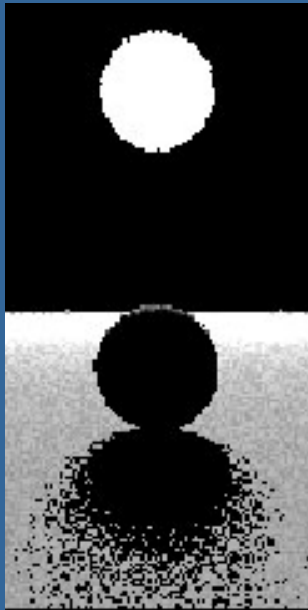
- Shoot many paths per pixel (the image just shows one light path).
 - At each intersection,
 - Shoot one shadow ray per light source
 - at random position on light, for area/volumetric light sources
 - and randomly select one new ray direction.

Example of soft shadows on a diffuse surface (with path tracing)

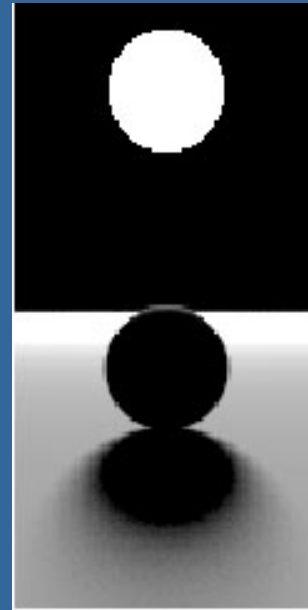


- Example: Three rays for one pixel
- All three rays hits diffuse floor
- Pick **one** random position on light source
- Send one random diffuse ray (D's above)
 - To continue the path...

Example of diffuse surface + soft shadows



One sample
per pixel



100 samples
per pixel

- Need to send many many rays to avoid noisy images
 - Sometimes 1000 or 10,000 rays are needed per pixel!
- Still, it is a simple method to generate high quality images

Path tracing: Summary

See section 6 in Kajiya's paper

- Uses Monte Carlo sampling to solve integration: just shoot many random ray paths over the integral domain
- Example: ray hits a diffuse surface
 - Sample randomly over the possible reflection directions by following **one** such ray.
 - Gives color bleeding effects (and the ambient part of lighting)
- Algorithm: shoot many paths per pixel, by randomly choosing **one** new ray at each interaction with surface + **one** shadow ray per light.

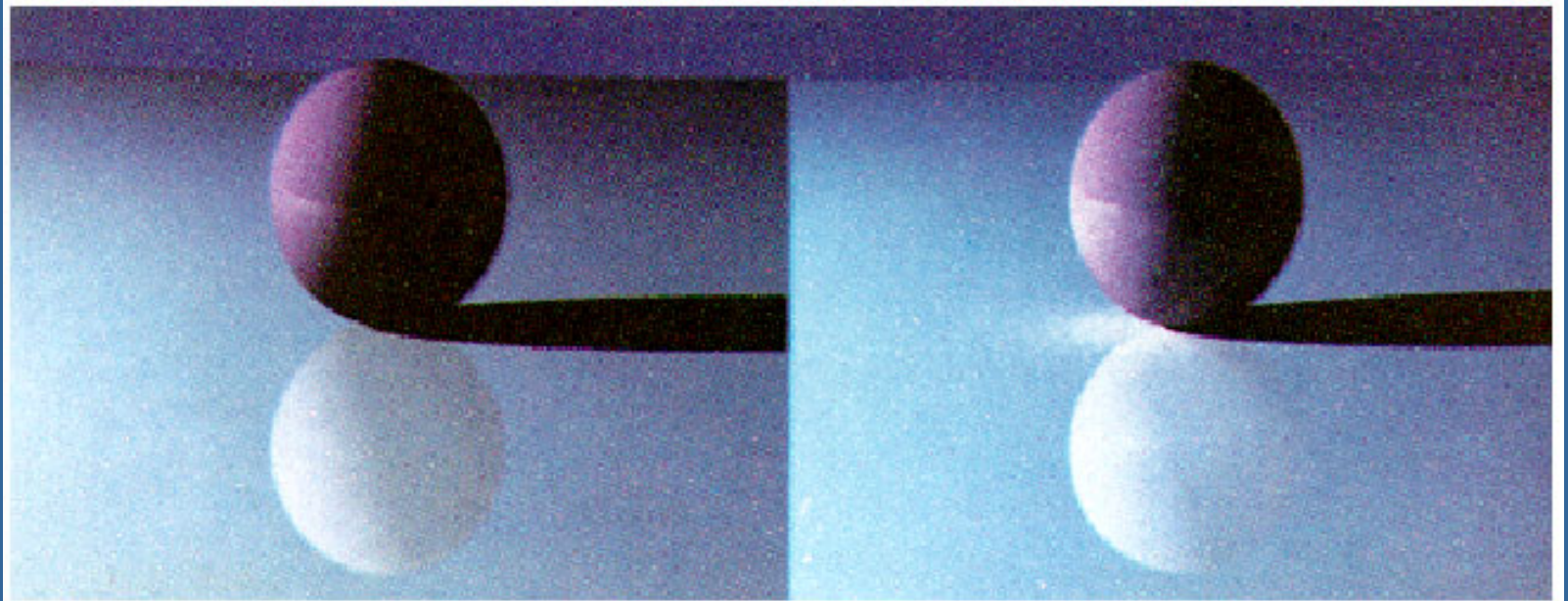
Russian Roulette

E.g., assuming perfectly diffuse and perfectly specular surfaces:

- Assume $k_{diff} + k_{spec} \leq 1$
 - Comes from that energy cannot be created, but can be absorbed
 - (E.g., k_{diff} could be sum of the diffuse color, $(R+G+B)/3$, and similar for k_{spec} .)
- 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 direction)
 - Else absorb ray.
- This is often called Russian roulette.
 - Also use for reflection vs refraction ray with probabilities based on Fresnel's eq..
 - and 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. Also terminates path with a physically-based probability.

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

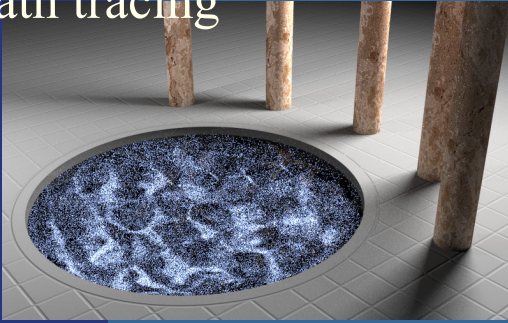
What is Caustics?

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

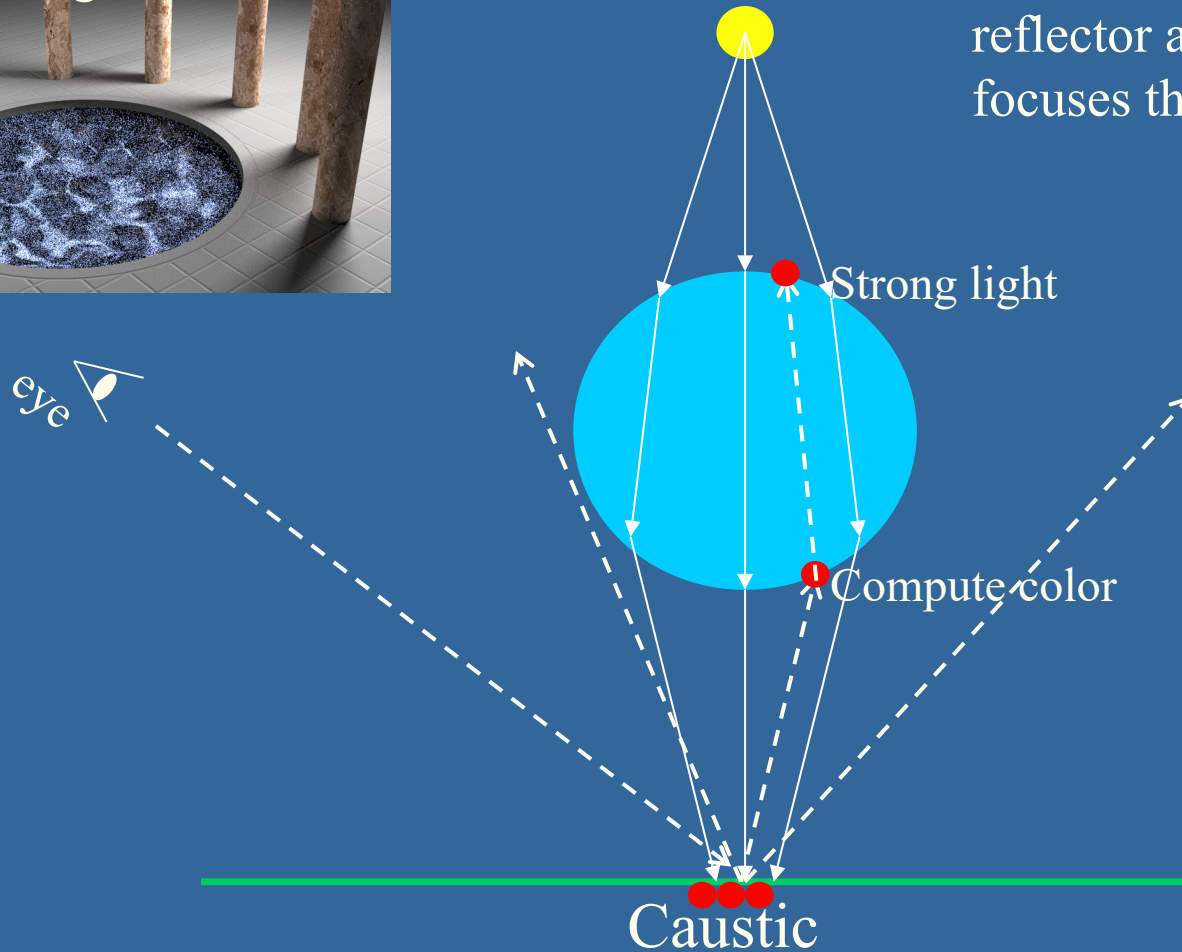


Reason why forward ray tracing fails to capture caustics well

Path tracing



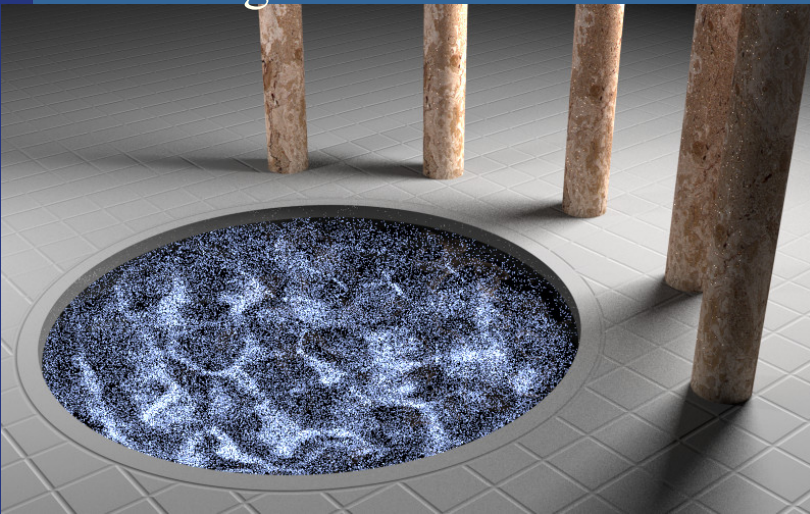
Must be lucky to hit the specular reflector and discover that it focuses the light.



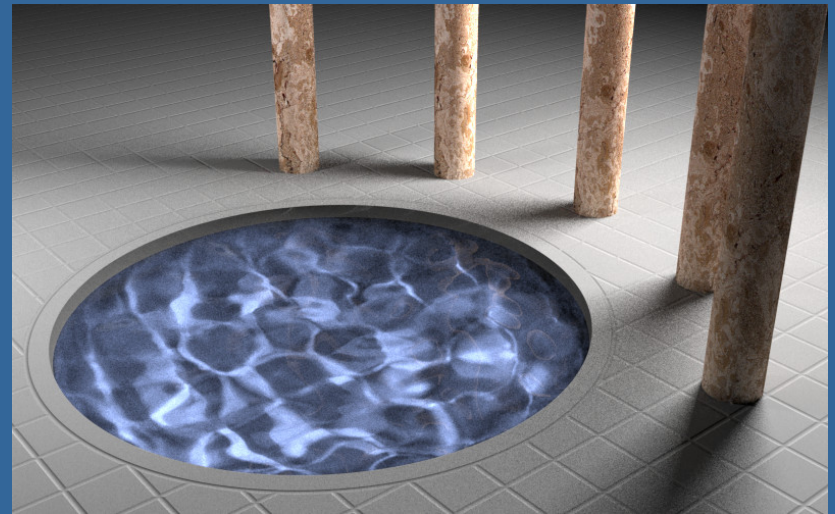
Extensions to path tracing

- Bidirectional path tracing
 - Developed in 1993-1994
 - 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)

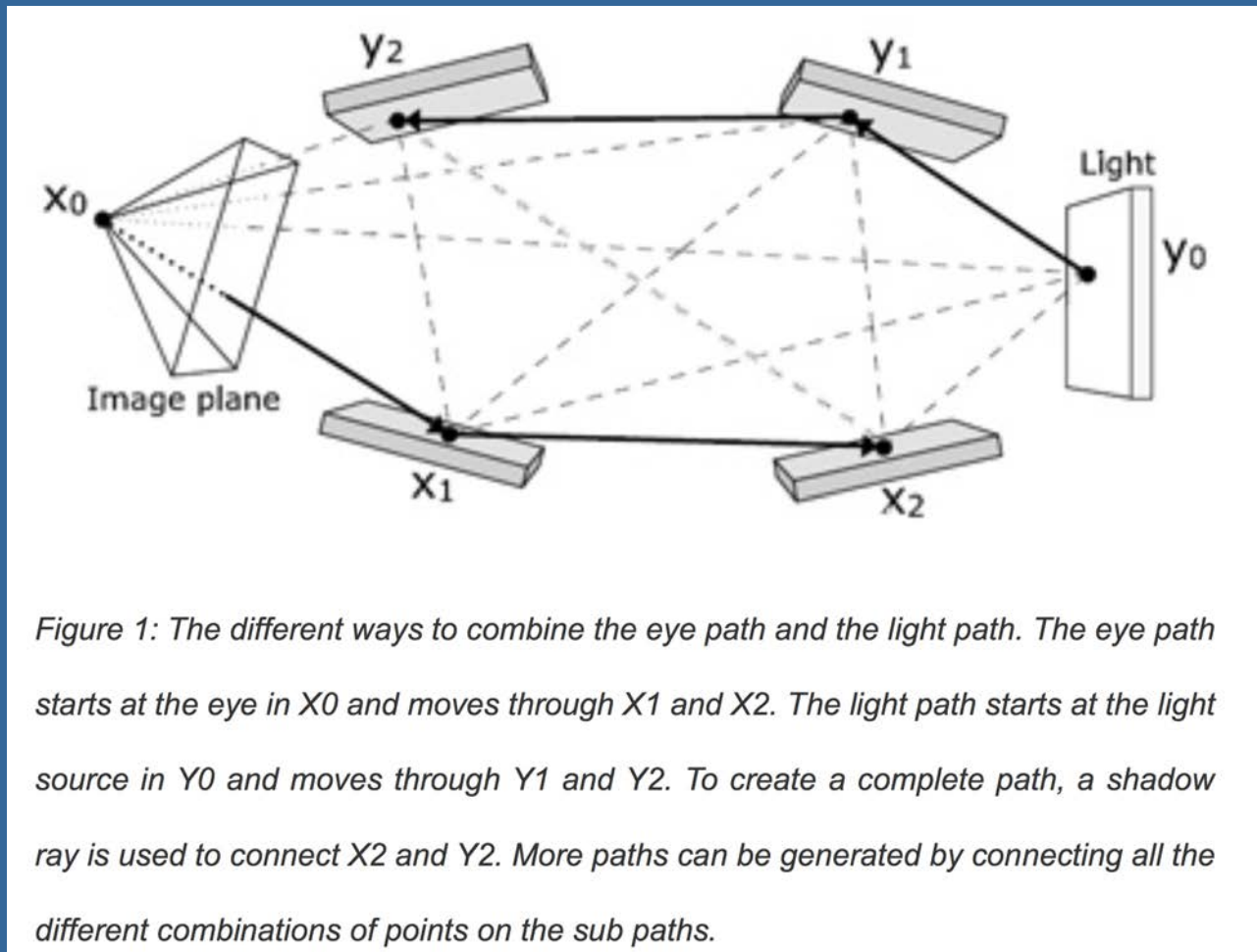
Path tracing



Metropolis (same rendering time)



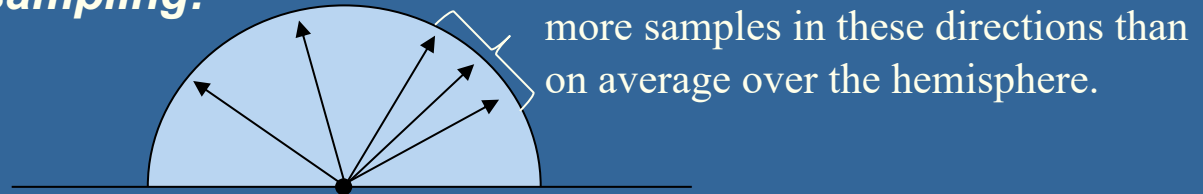
Bidirectional Path tracing



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**:



- 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.
 - $w_i = 1/(n * p(\omega_i))$, $p(\omega_i) = \text{"probability_bias_of_the_chosen_direction"}$

How much more/less likely (e.g., *2) direction ω_i is compared to the average direction (avg. should be =1).



Photon mapping

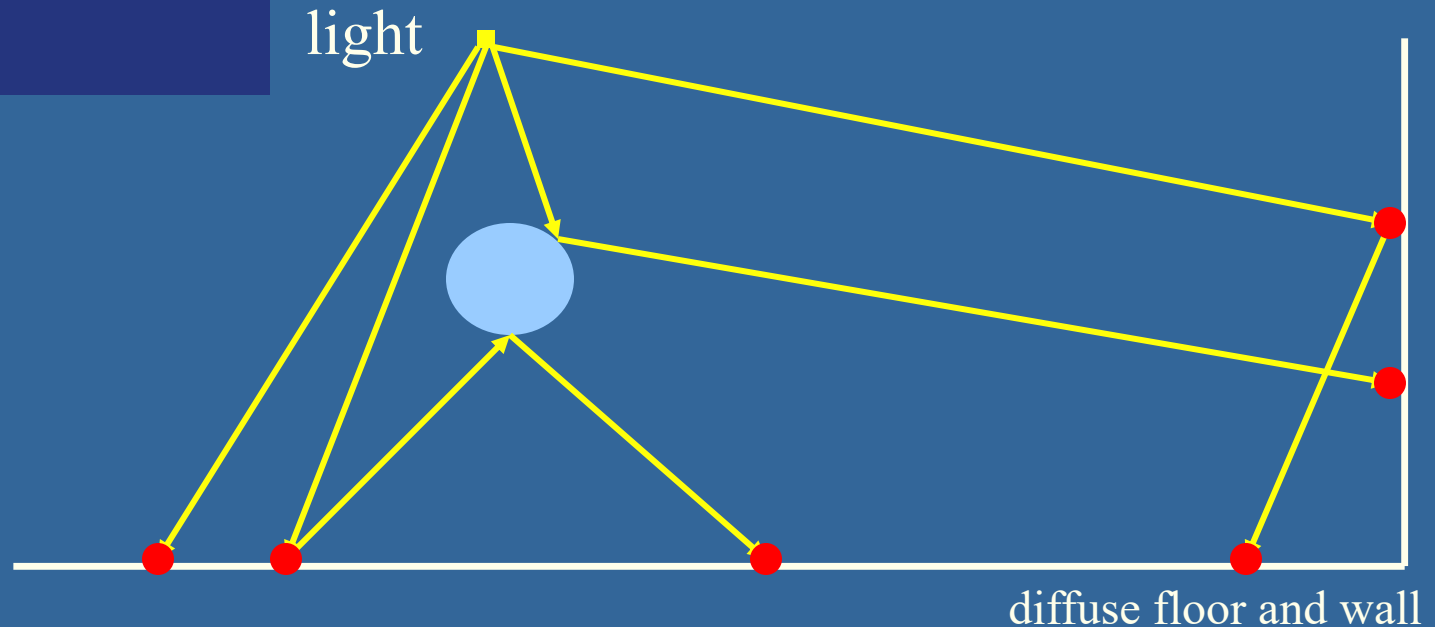
- Developed by Henrik Wann Jensen (started 1993)
- A clever two-pass algorithm:
 - 1: Shoot photons from light source, and let them bounce around in the scene, and store them where they land
 - 2: "Ray tracing"-like pass from the eye, but gather the photons from the previous pass
- Advantages:
 - Fast
 - Handles arbitrary geometry (as do path tracing)
 - All global illumination effects can be seen
 - Little noise (in still images)

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 has a diffuse component), store the photon in photon map
 - Then use Russian roulette to find out whether the photon is absorbed or reflected
 - If reflected, then shoot photon in new random direction

Photon tracing

● This is a stored photon



- Should not store photon at purely specular surfaces, because these effects are fully view dependent
 - only diffuse effect is view independent
- At hit, photon gets colored (loses intensity)
 - E.g., white photon (1,1,1) becomes pink (0.8, 0.5, 0.5), so loses intensity.
 - Instead of decreasing intensity, decrease probability of further scatter the photon. (E.g., probability of absorbing photon ≈ 0.4)
 - Why not just decrease the photon's intensity?
 - Harder to get good filtering by expanding spheres

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

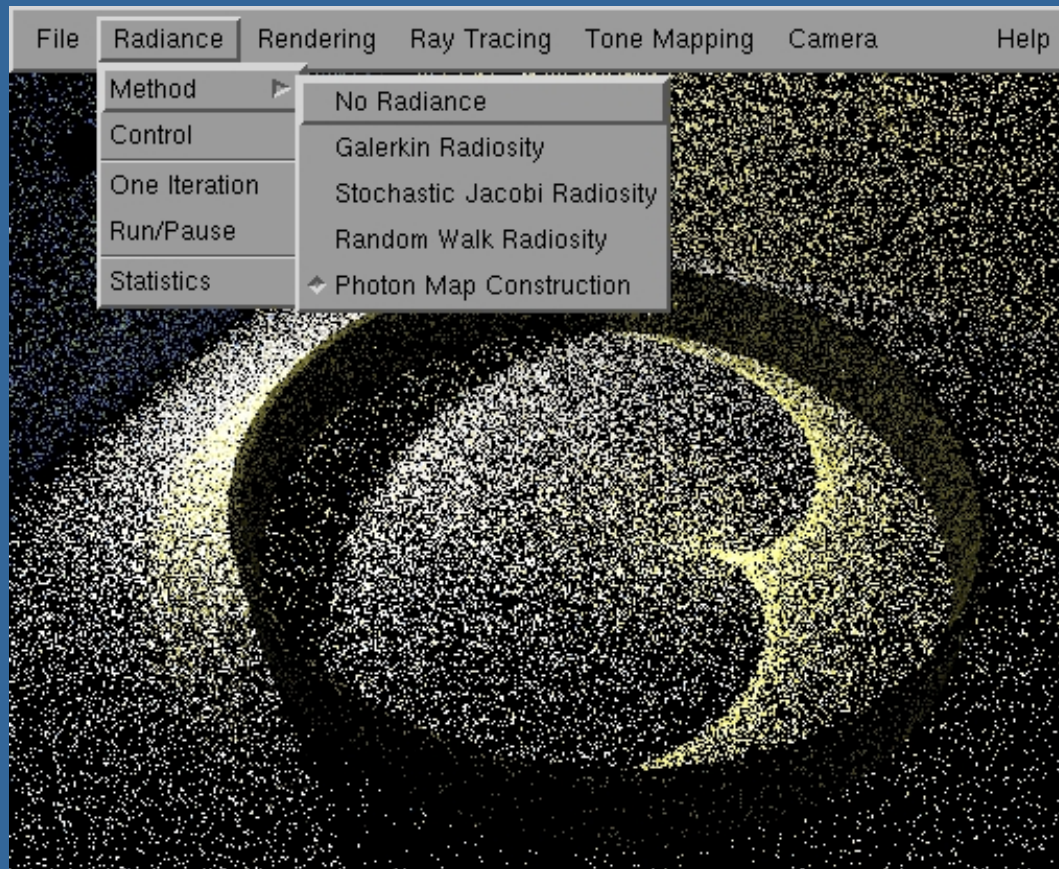
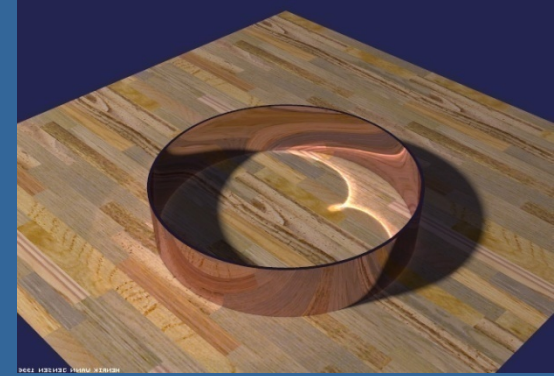
After Henrik Wann Jensen

```
// 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 expanding when n closest
// photons have been found
```

What does it look like?

- Stored photons displayed:



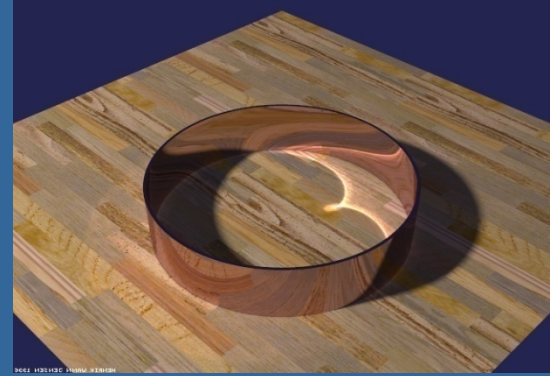
Density estimation

- 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 \mathbf{x} :

$$L(\mathbf{x}, \omega) \approx \frac{1}{\pi r^2} \sum_1^n f_r(\mathbf{x}, \omega_p, \omega) \Phi_p(\mathbf{x}, \omega_p)$$

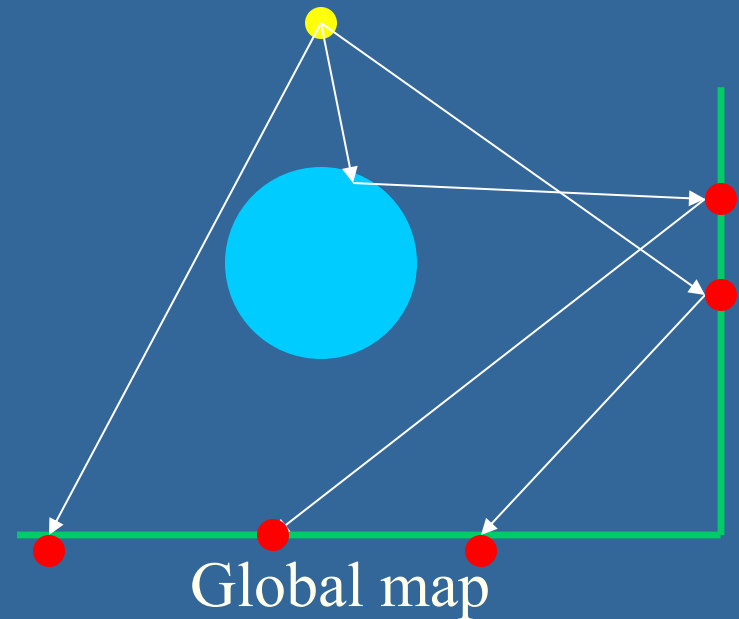
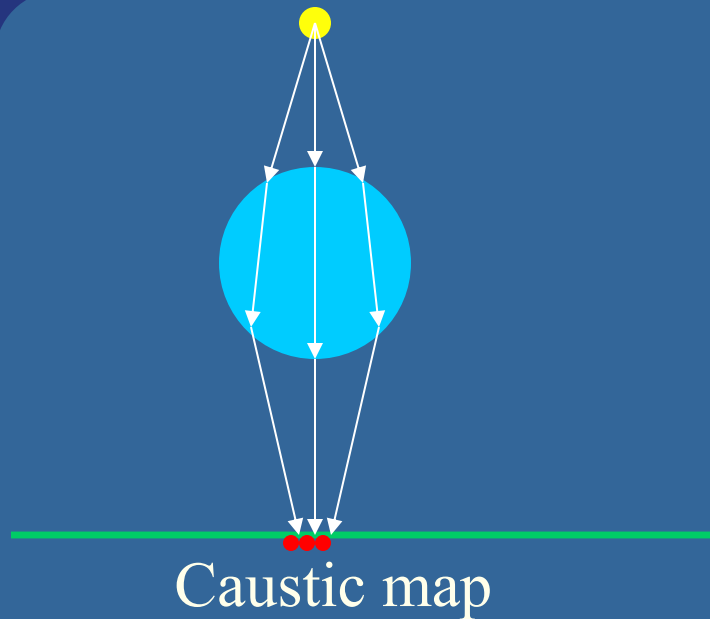
- L is radiance in \mathbf{x} in the direction of ω
- r is radius of expanded sphere
- ω_p is the direction of the stored photon
- Φ_p is the stored power of the photon
- f_r is the BRDF

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
 - (Light transport notation: $LS+D$)
 - +
 - A global photon map (for all illumination)
 - All photons that landed on diffuse surfaces
 - $L(S | D)*D$

Caustic map and global map



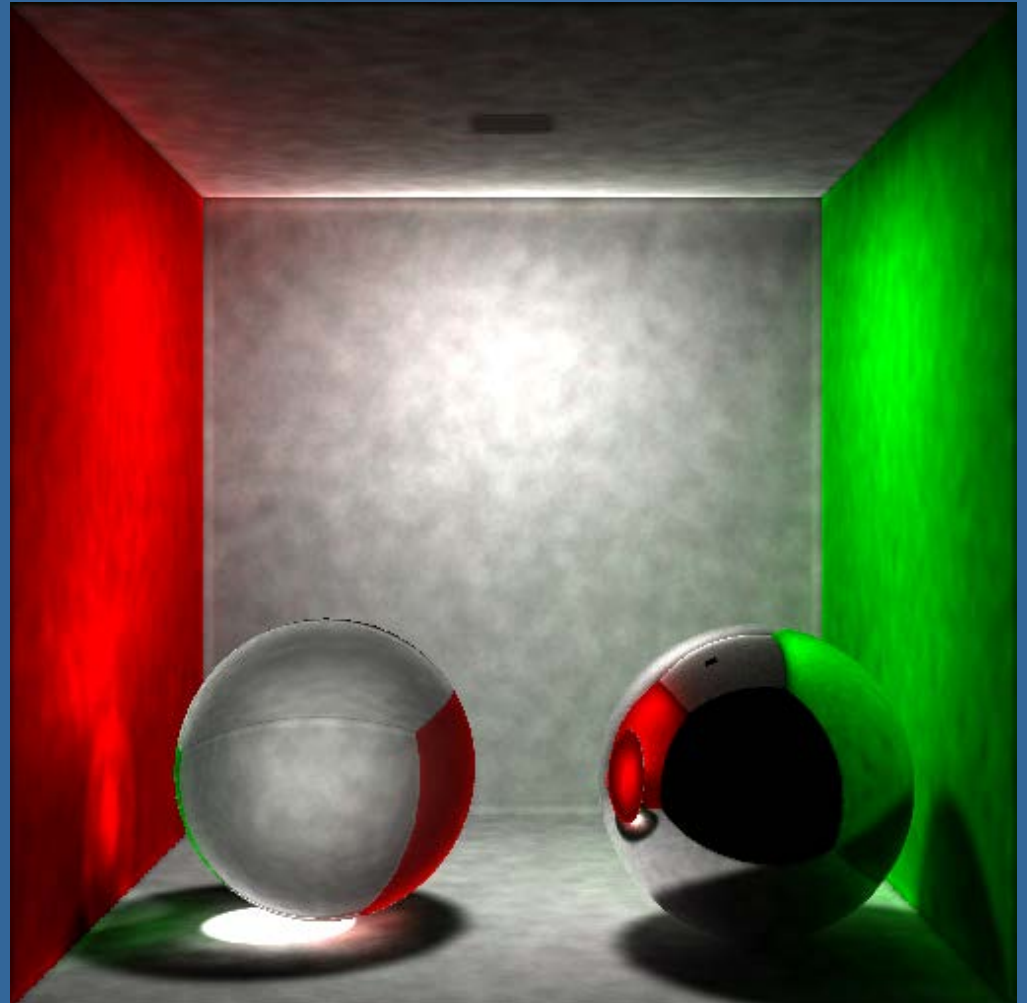
- Caustic map: send photons only towards reflective and refractive surfaces
 - Caustics is a high frequency component of illumination
 - Therefore, need many photons to represent accurately
- Global map - assumption: illumination varies more slowly

Pass 2: Rendering using the photon map

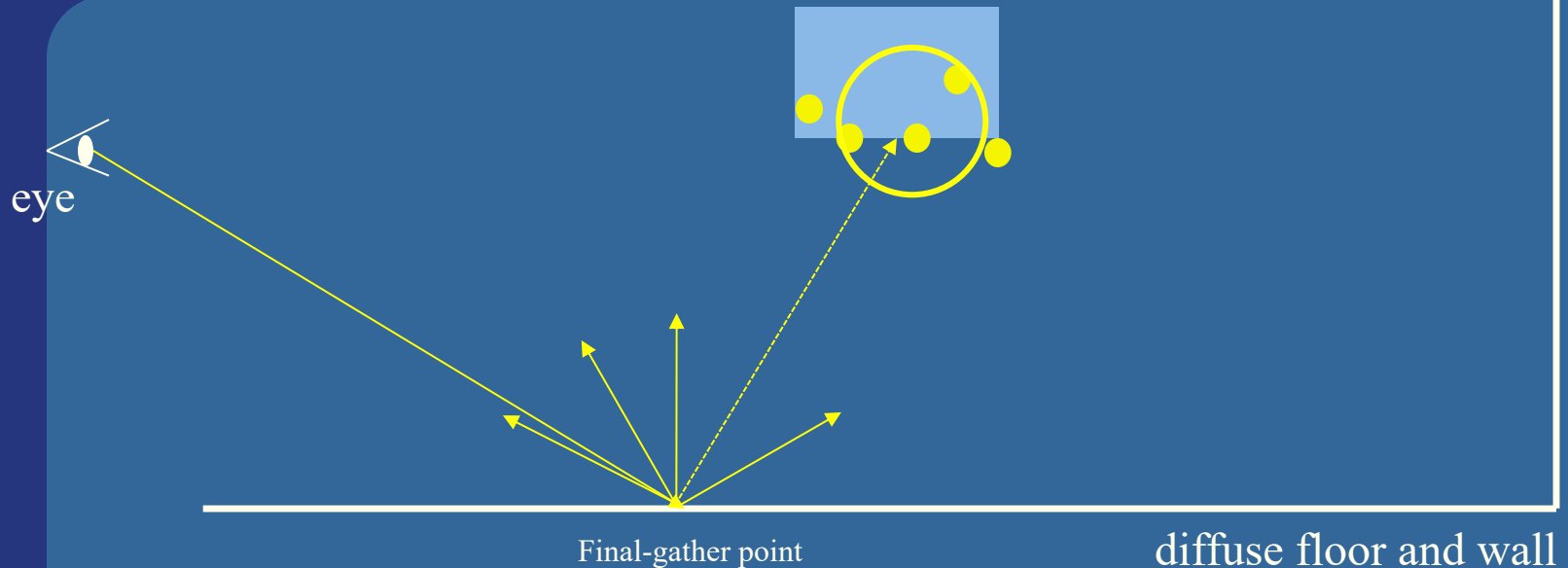
- Render from the eye using a modified ray tracer
 - A number of rays are sent per pixel
 - For each ray, evaluate four terms
 - **Direct illumination** (light that reaches a surface directly from light source)... may need to send many rays to area lights. Done using standard ray tracing.
 - **Specular reflection** (also evaluated using ray tracing, possibly with many rays sent around the reflection direction)
 - **Caustics**: use caustics photon map
 - **Indirect illumination**: use the global photonmap
 - Or *Final Gather* + global photon map...

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

- Ugly noise:
- Solution:
 - for the primary rays: use Final Gather instead of using the global photon map



A modification for indirect Illumination – Final Gather

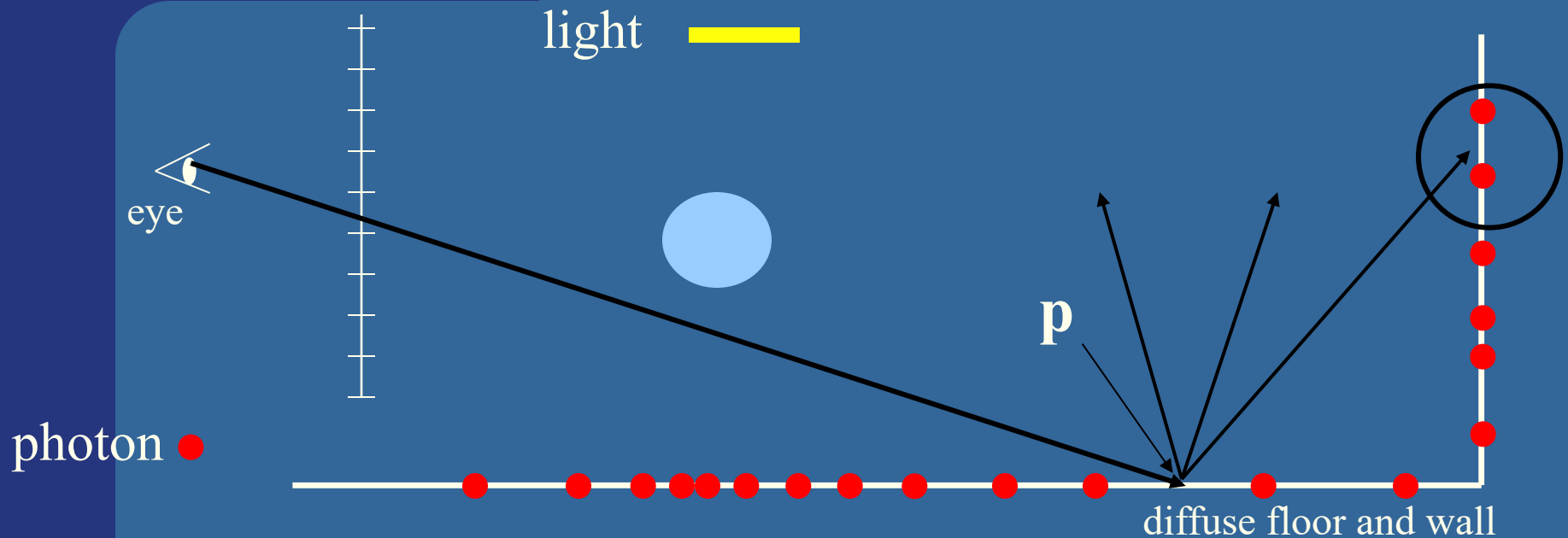


- Too noisy to use the global map for direct visualization
- Remember: eye rays are recursively traced (via reflections/refractions) until a diffuse hit, \mathbf{p} . There, we want to estimate slow-varying indirect illumination.
 - Instead of growing sphere in global map at \mathbf{p} , Final Gather shoots 100-1000 indirect rays from \mathbf{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.

Final Gather

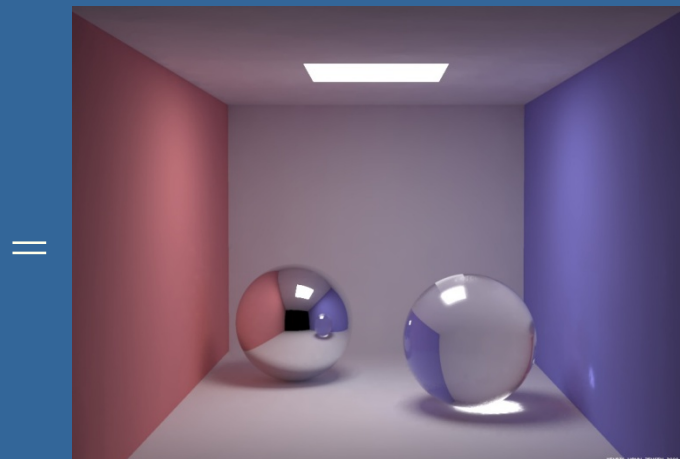
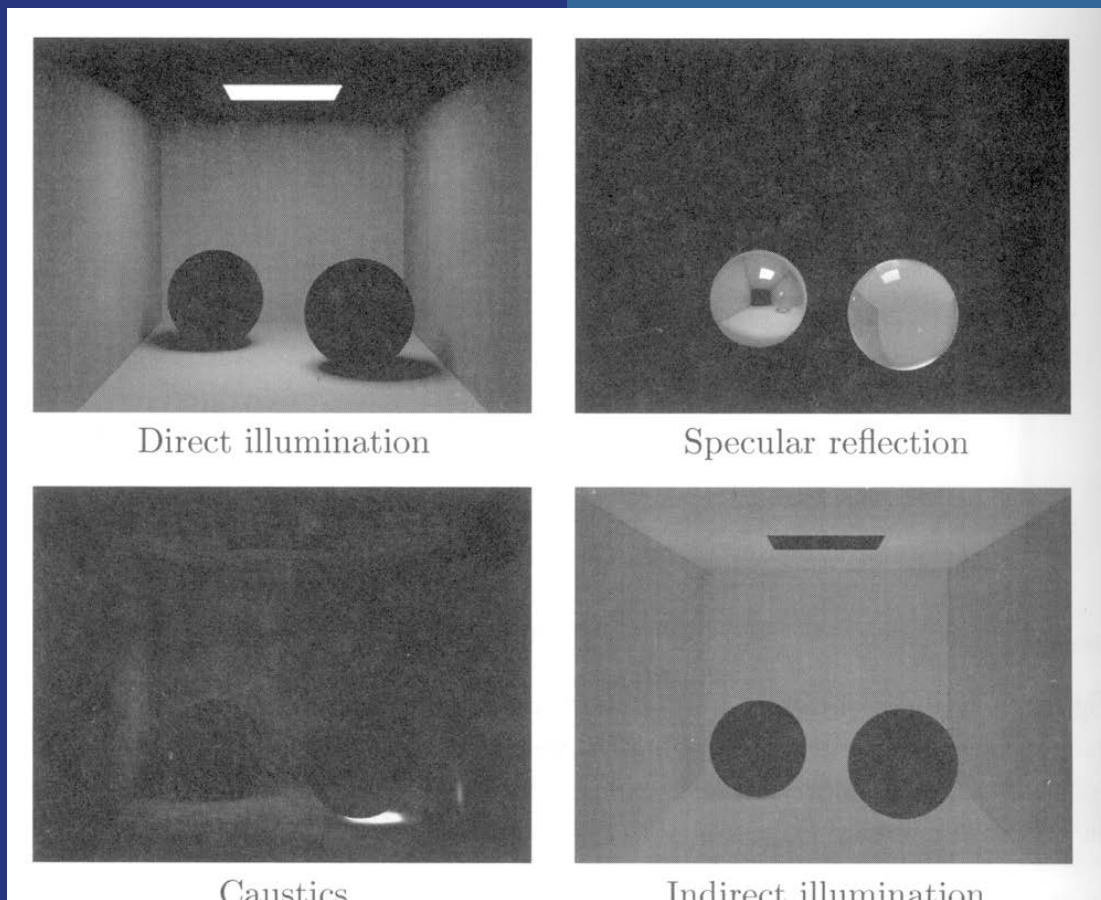
- Final gathering is a technique for estimating global illumination for a given point by either sampling a number of directions in the hemisphere over that point (such a sample set is called a *final gather point*), or by averaging a number of final gather points nearby since final gather points are too expensive to compute for every illuminated point.
- For diffuse hits, final gathering often improves the quality of the global illumination solution. Without final gathering, the global illumination on a diffuse surface is computed by estimating the photon density (and energy) near that point. With final gathering, many new rays are sent out to sample the hemisphere above the point to determine the incident illumination. Some of these rays hit diffuse surfaces; the global illumination at those points is then computed by the material shaders at those sample point, using illumination from the photon maps and other material properties. Other rays hit specular surfaces and do not contribute to the final gather color (since that type of light transport is a secondary caustic). Tracing many rays (each with photon-map lookups) is very time-consuming so it is only done when necessary – in most cases, interpolation and extrapolation from previous nearby final gatherings is sufficient.

Indirect illumination: Use the global photon map



- To evaluate indirect illumination at point p :
 - Send several random rays out from p , and grow spheres at contacts
 - May need several hundreds of rays to get good results.

Images of the four components



- These together solves the entire rendering equation!

Photon Mapping - Summary

- **Creating Photon Maps:**

- Trace photons (~100K-1M) from light source. Store them in kd-tree when they hit diffuse surface. 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.

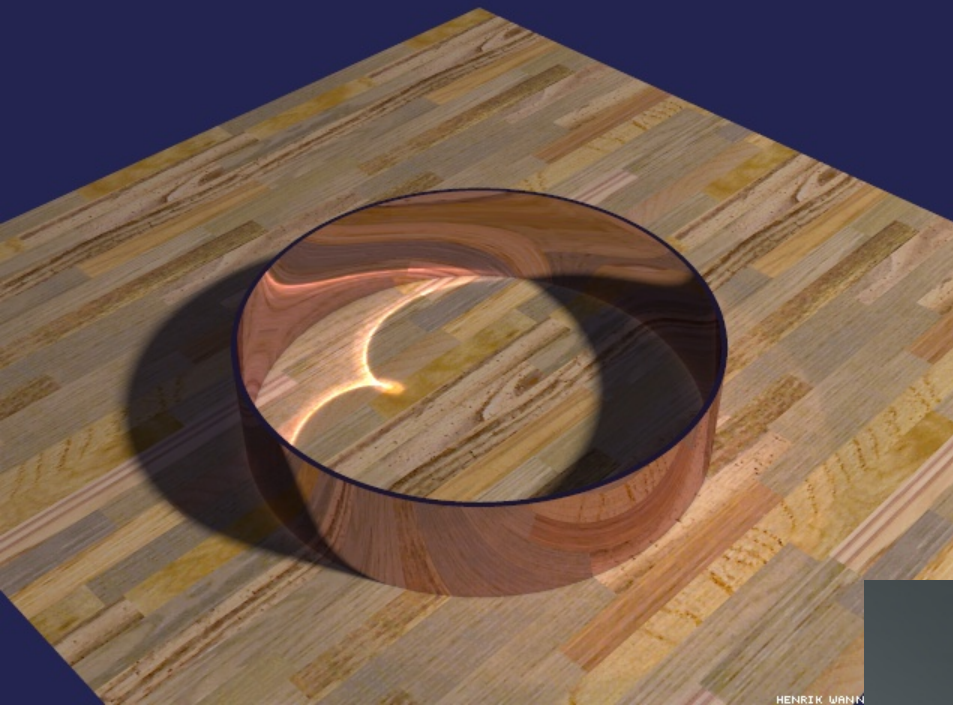
- **Ray trace from eye:**

- As usual: I.e., shooting primary rays and recursively shooting reflection/refraction rays, and at each intersection point \mathbf{p} , compute direct illumination (shadow rays + shading).
- Also grow sphere around each \mathbf{p} in caustics map to get caustics contribution and in global map to get slow-varying indirect illumination.
- If final gather is used: At the first diffuse hit, instead of using global map directly, sample indirect slow varying light around \mathbf{p} by sampling the hemisphere with ~100 – 1000 rays and use the two photon maps where those rays hit a surface. Or interpolate from nearby final-gather points.

- **Growing sphere:**

- Uses the kd-tree to expand a sphere around \mathbf{p} until a fixed amount (e.g. 50) photons are inside the sphere. The radius is an inverse measure of the intensity of indirect light at \mathbf{p} . The BRDF at \mathbf{p} could also be used to get a more accurate color and intensity value.

Standard photon mapping

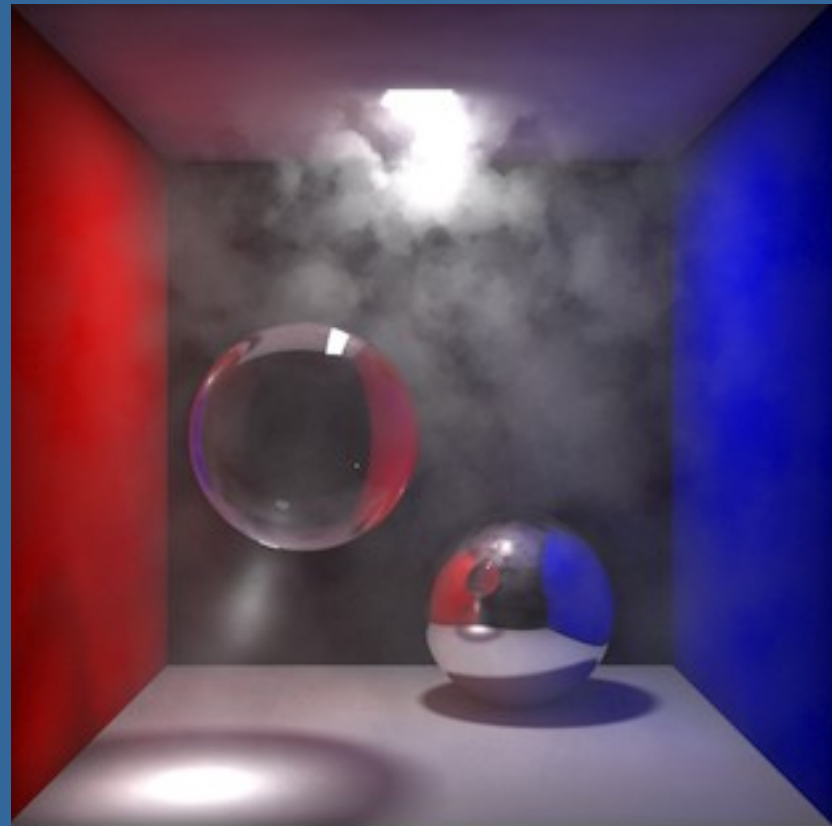
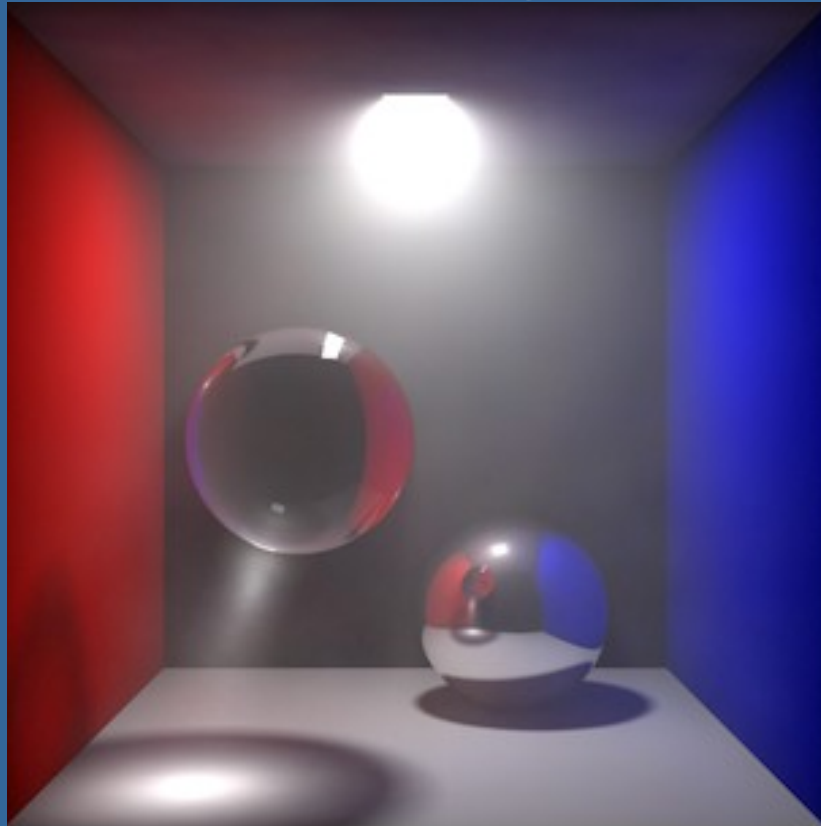


Caustics: concentrated
reflected or refracted light



Extensions to photon mapping

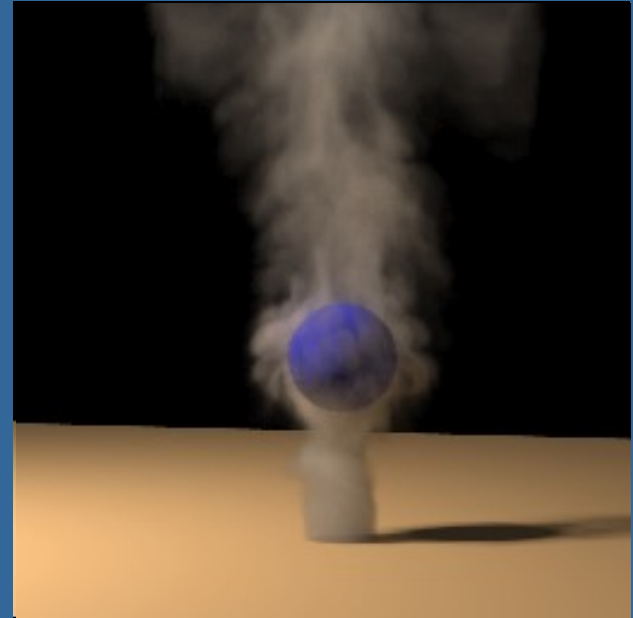
- Participating media



Another one on participating media



Smoke and photon mapping



Press for a movie

In conclusion

- If you want to get global illumination effects, then implement a path tracer
 - Simple to implement
 - Good results
 - Disadvantage: rendering times (many many rays per pixel). Advantage: fast preview.
- If you want a more advanced renderer:
 - Extend with bidirectional path tracing
 - Or Metropolis Light Transport, or Photon Mapping

What you need to know

- The rendering equation

- Be able to explain all its components

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

- 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 overall algorithm. See the summary slide on:

- Creating Photon Maps...
- Ray trace from eye...
- Growing spheres...

- Final Gather

- Why it is good. How it works:

- At the first diffuse hit, instead of using global map directly, sample indirect slow varying light around \mathbf{p} by sampling the hemisphere with ~ 1000 rays and use the two photon maps where those rays hit a diffuse surface.

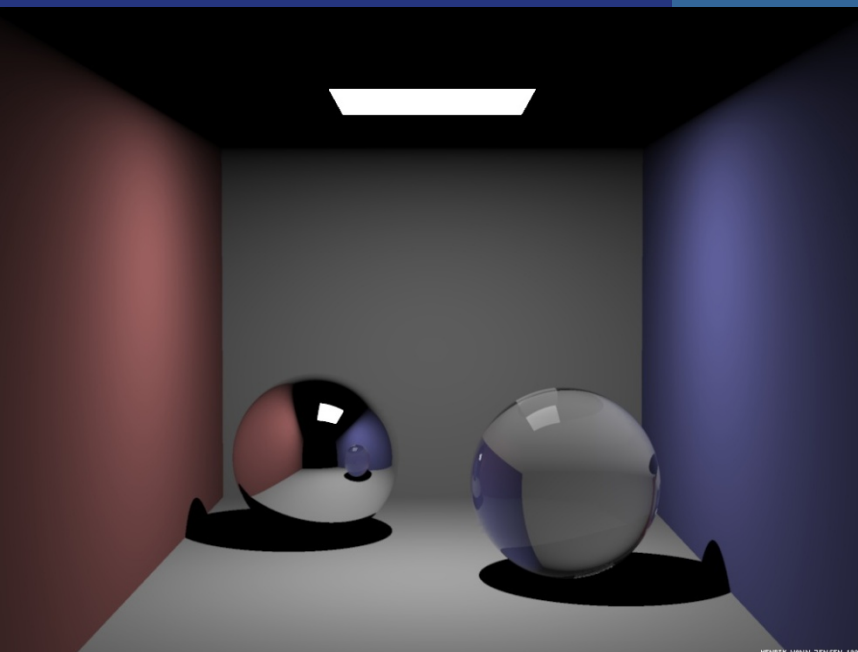
- Bidirectional Path Tracing, Metropolis Light Transport

- Just their names. Don't need to know the algorithms.

Photon Mapping - Summary

- **Creating Photon Maps:**
 - Trace photons ($\sim 100K-1M$) from light source. Store them in kd-tree when they hit diffuse surface. Then, use russian roulette to decide if the photon should be absorbed or specularly or diffusely reflected. Create both global map and caustics map. For the Caustics map, we send more of the photons towards reflective/refractive objects.
- **Ray trace from eye:**
 - As usual: i.e., shooting primary rays and recursively shooting reflection/refraction rays, and at each intersection point \mathbf{p} , compute direct illumination (shadow rays + shading).
 - Also grow sphere around each \mathbf{p} in caustics map to get caustics contribution and in global map to get slow-varying indirect illumination.
 - If final path is used: At the first diffuse hit, instead of using global map directly, sample indirect slow-varying light around \mathbf{p} by sampling the hemisphere with $\sim 100 - 1000$ rays and use the two photon maps where those rays hit a surface. Or interpolate from nearby final-path points.
- **Growing sphere:**
 - Using the kd-tree to expand a sphere around \mathbf{p} until a fixed amount (e.g. 50) photons are inside the sphere. The radius is an inverse measure of the intensity of indirect light at \mathbf{p} . The BRDF at \mathbf{p} could also be used to get a more accurate color and intensity value.

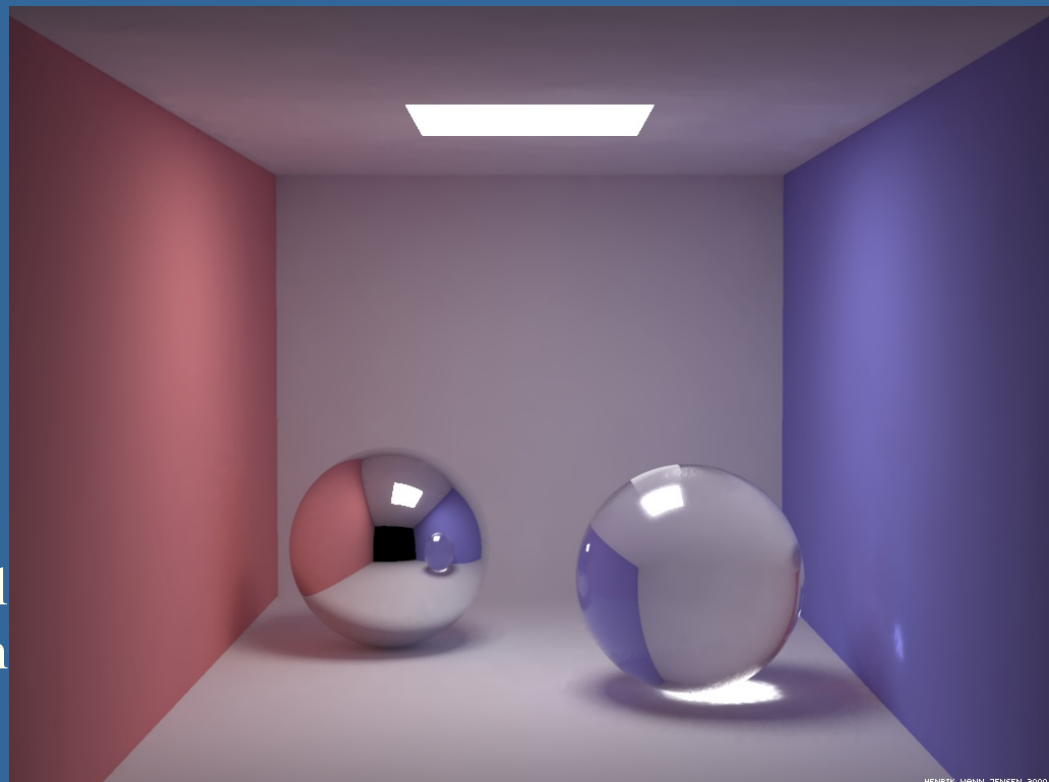
Isn't ray tracing enough?



Ray tracing

Effects to note in Global Illumination image:

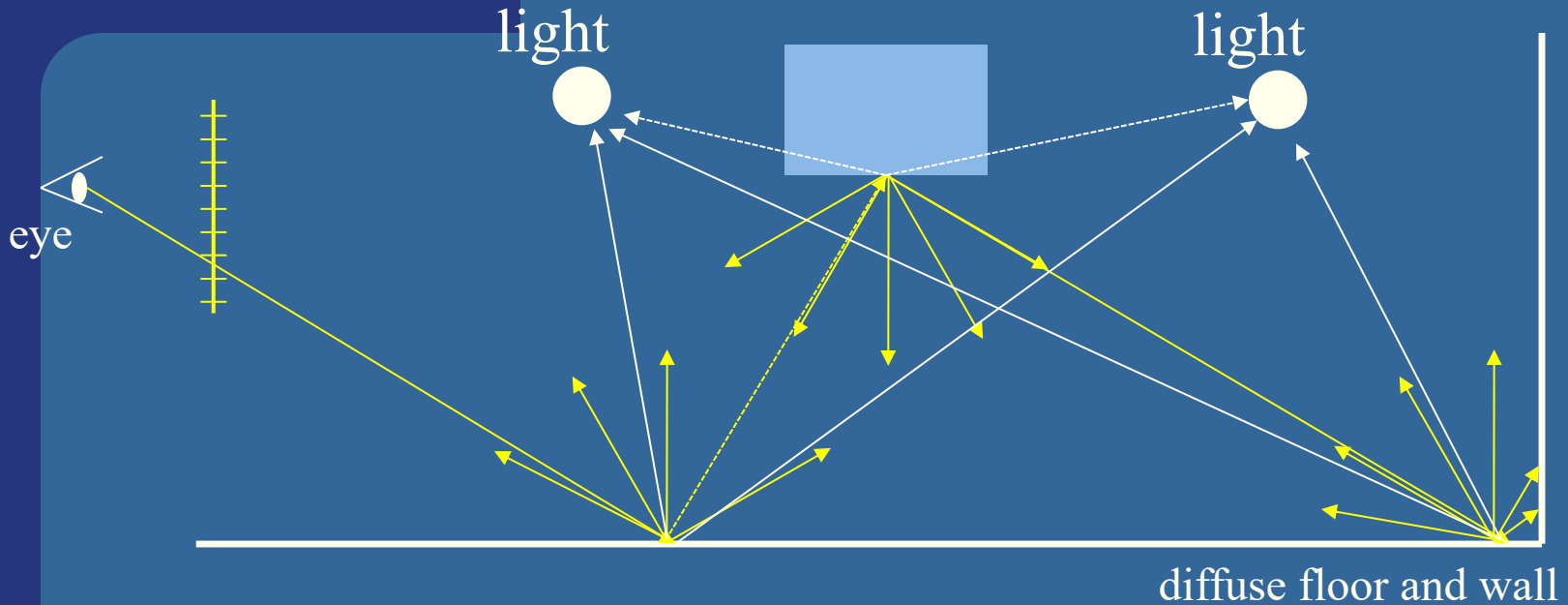
- 1) Indirect lighting (light reaches the roof)
- 2) Soft shadows (light source has area)
- 3) Color bleeding (example: roof is red near red wall) (same as 1)
- 4) Caustics (concentration of refracted light through glass ball)
- 5) Materials have no ambient component



Global
Illumination

Which are
the differences?

Monte Carlo Ray Tracing – direct + indirect illumination

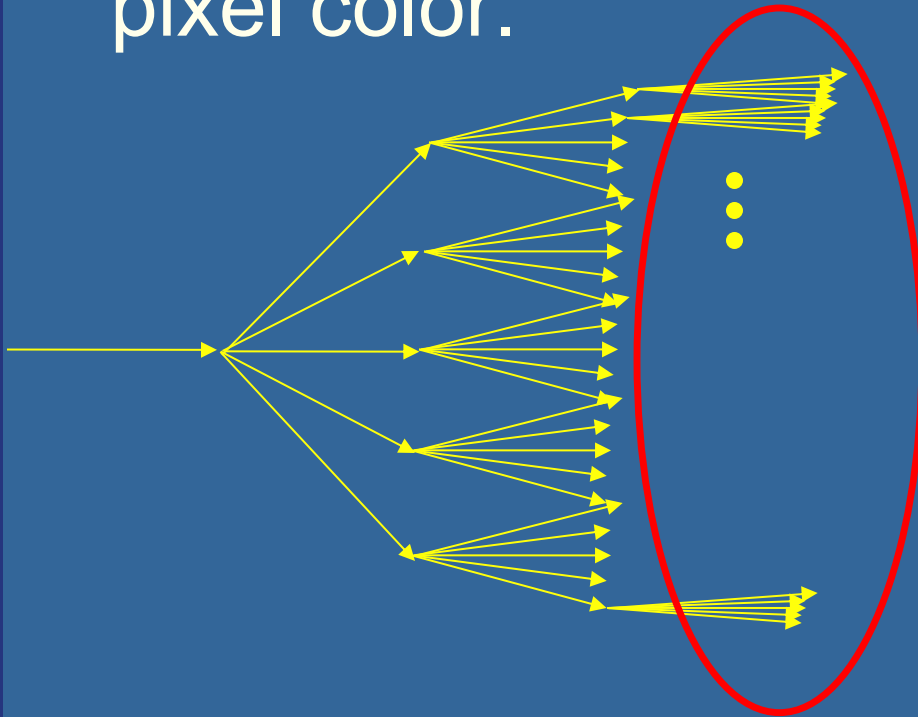


- Sample indirect illumination by shooting sample rays over the hemisphere, at each hit.

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

Monte Carlo Ray Tracing

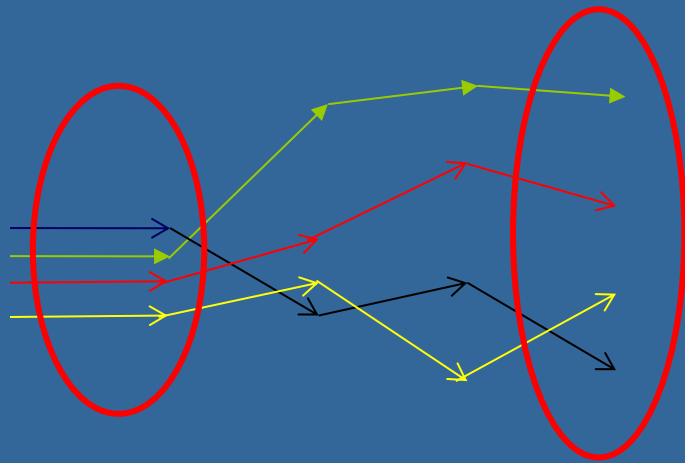
- This gives a ray tree with most rays at the bottom level. This is bad since these rays have the lowest influence on the pixel color.



PathTracing

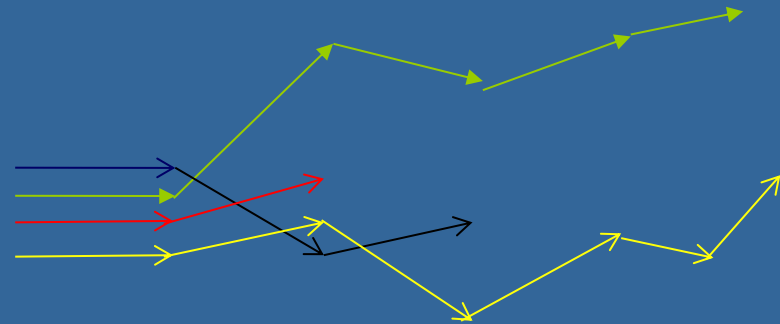
– one efficient Monte-Carlo Ray-Tracing solution

- 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.



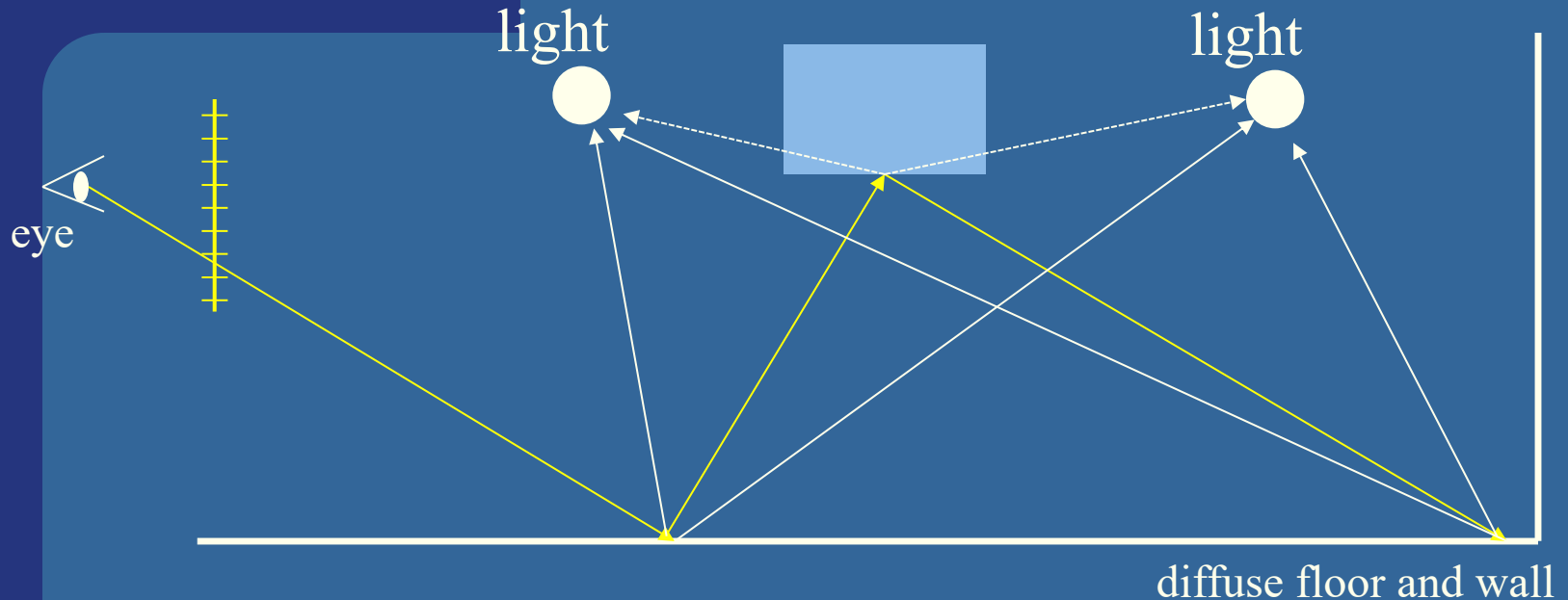
Equally number of rays
are traced at each level

Or:



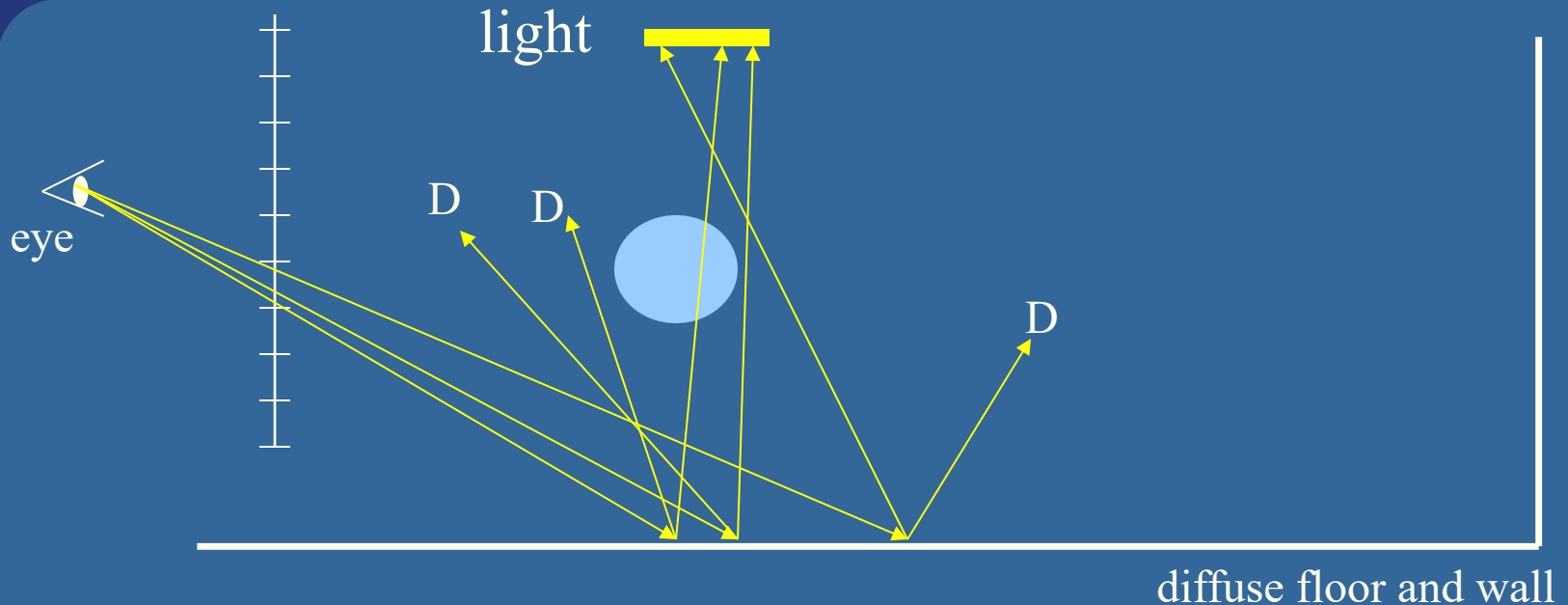
Even smarter: terminate path with
some probability after each level,
since they have decreasing
importance to final pixel color.

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.

Example of soft shadows on a diffuse surface (with path tracing)



- Example: Three rays for one pixel
- All three rays hits diffuse floor
- Pick **one** random position on light source
- Send one random diffuse ray (D's above)
 - To continue the path...

Photon Mapping - Summary

- **Creating Photon Maps:**

- Trace photons ($\sim 100\text{K}$ - 1M) from light source. Store them in kd-tree when they hit diffuse surface. 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.

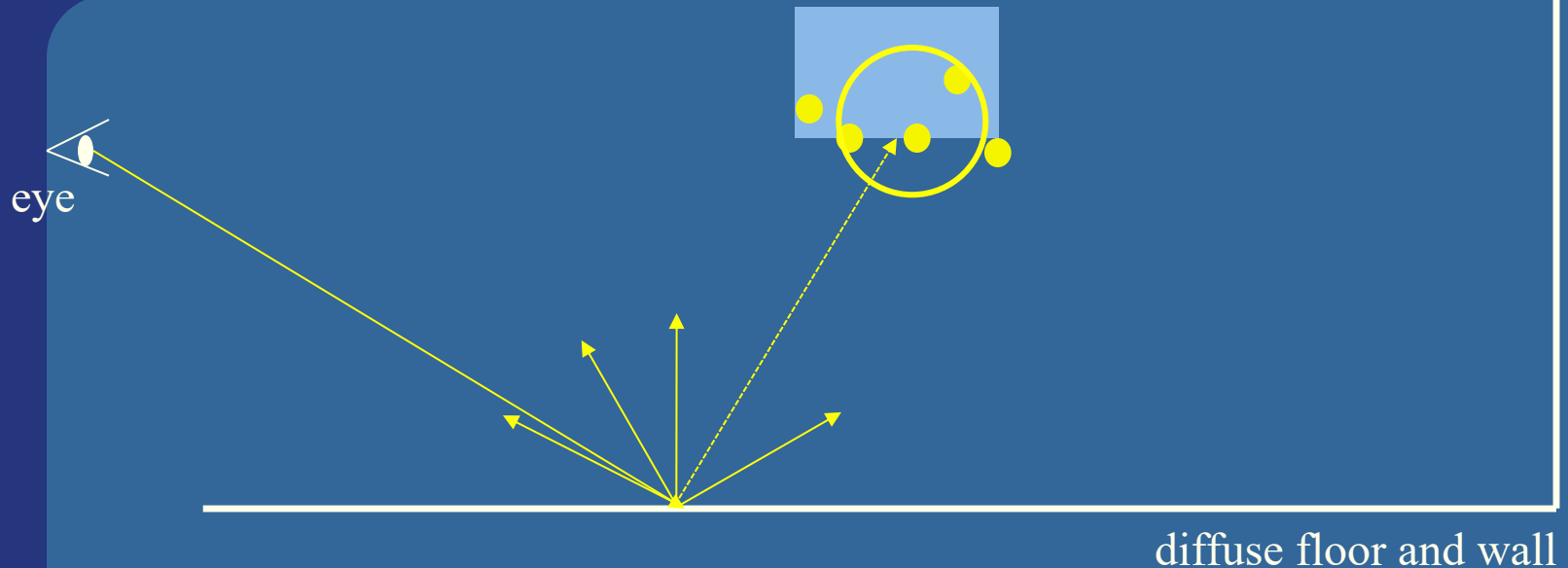
- **Ray trace from eye:**

- As usual: I.e., shooting primary rays and recursively shooting reflection/refraction rays, and at each intersection point \mathbf{p} , compute direct illumination (shadow rays + shading).
- Also grow sphere around each \mathbf{p} in caustics map to get caustics contribution and in global map to get slow-varying indirect illumination.
- If final gather is used: At the first diffuse hit, instead of using global map directly, sample indirect slow varying light around \mathbf{p} by sampling the hemisphere with $\sim 100 - 1000$ rays and use the two photon maps where those rays hit a surface. Or interpolate from nearby final-gather points.

- **Growing sphere:**

- Uses the kd-tree to expand a sphere around \mathbf{p} until a fixed amount (e.g. 50) photons are inside the sphere. The radius is an inverse measure of the intensity of indirect light at \mathbf{p} . The BRDF at \mathbf{p} could also be used to get a more accurate color and intensity value.

A modification for indirect Illumination – Final Gather



- Too noisy to use the global map for direct visualization
- Remember: eye rays are recursively traced (via reflections/refractions) until a diffuse hit, \mathbf{p} . There, we want to estimate slow-varying indirect illumination.
 - Instead of growing sphere in global map at \mathbf{p} , Final Gather shoots 100-1000 indirect rays from \mathbf{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 final-gather points.