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/TDA361/Advanced Computer Graphics/
- ~13 seminars in total, sp3+4
- 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, ray tracing with photon mapping.
  - HDRI

GFX Companies Gothenburg

3D software development:
Ghost Games (DICE)
Autodesk,
EON,
Spark Vision
Simbin
MindArk
Mentice
Vizendo
Surgical Science
Combitech
Fraunhofer (Chalmers Teknikpark)
RD&T Technology
Smart Eye AB
Rapid Images

And many more that I have forgotten now…

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

Non-Gothenburg

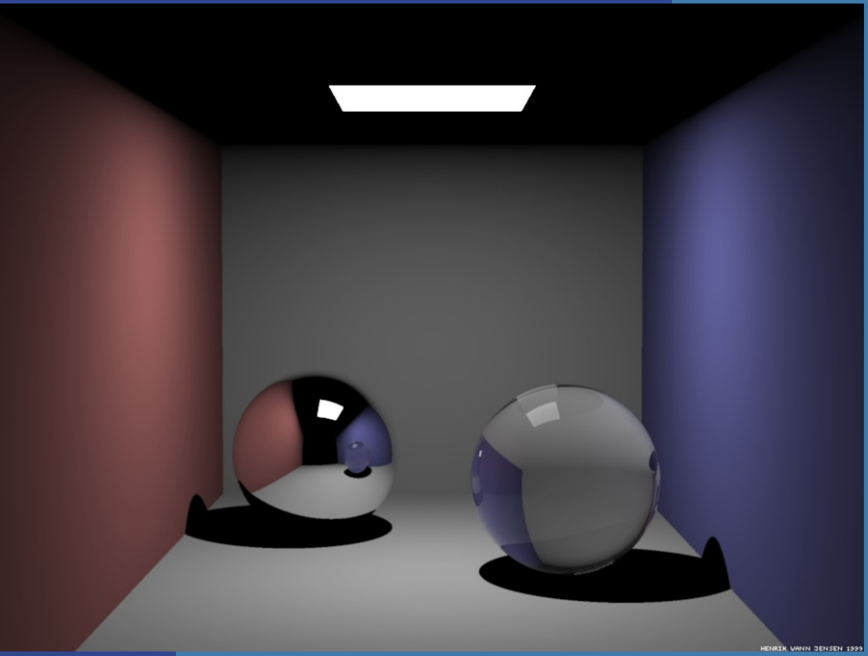
Game Studios:
Avalanche studios (Sthlm)
DICE (Sthlm)
Massive (Malmö)

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

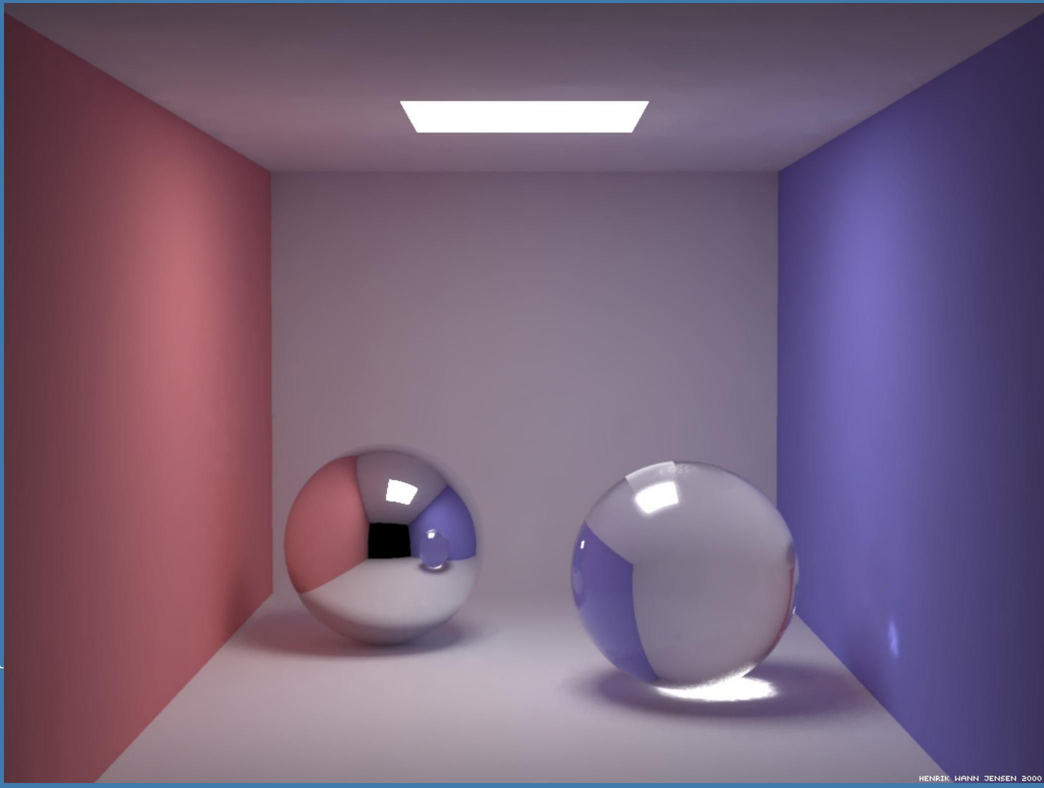
Architects, graphics artists:
White
Wingårdhs
Volvo Personvagnar
Semcon
Ramböll
Zynka
CAP AB
Grafia

Isn't ray tracing enough?

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



Ray tracing



Global Illumination

Which are the differences?

Images courtesy of Henrik Wann Jensen

Global Illumination

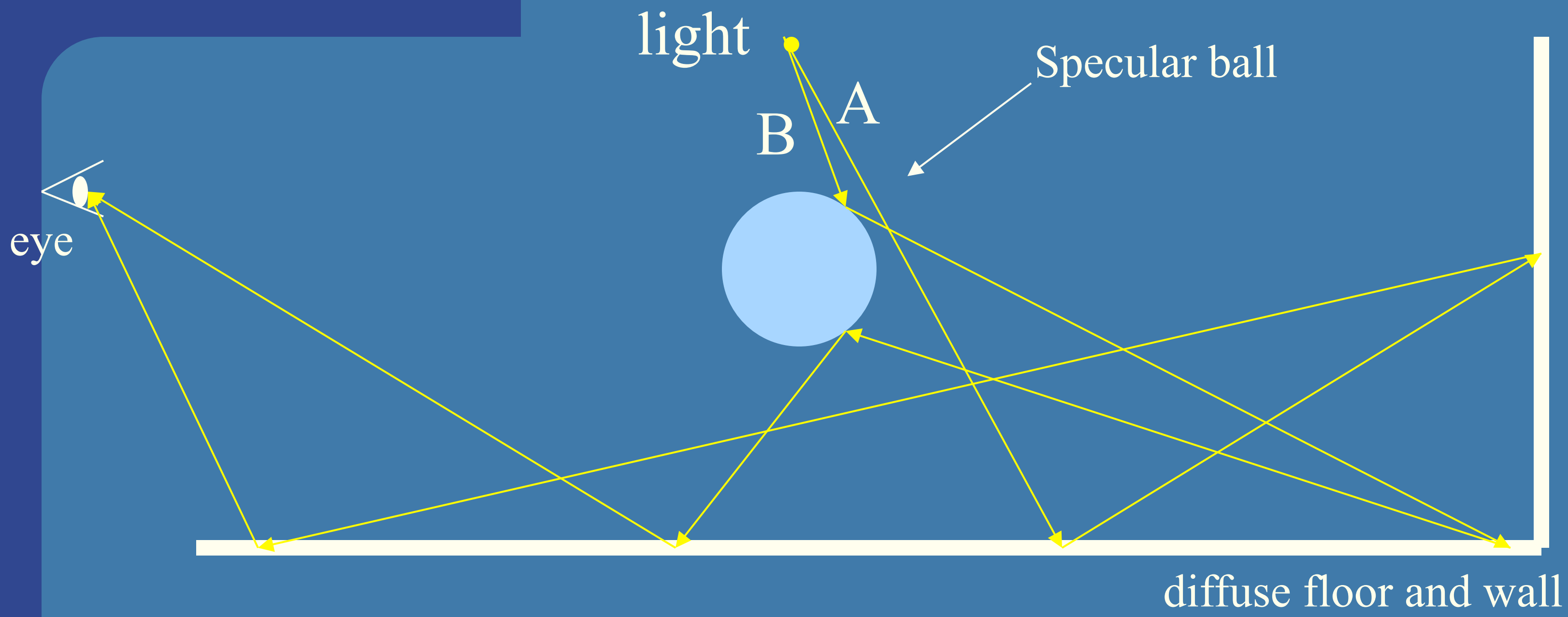
- The goal: follow all photons through a scene, in order to render images with all light paths
- This will give incredibly realistic images
- This lecture will treat:
  - Background
  - Montecarlo ray tracing
  - Path tracing
  - Photon mapping
  - Final Gather
- Great books on global illumination and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

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 | M) : a K or an M event

Notation after Paul Heckbert, SIGGRAPH 90

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?

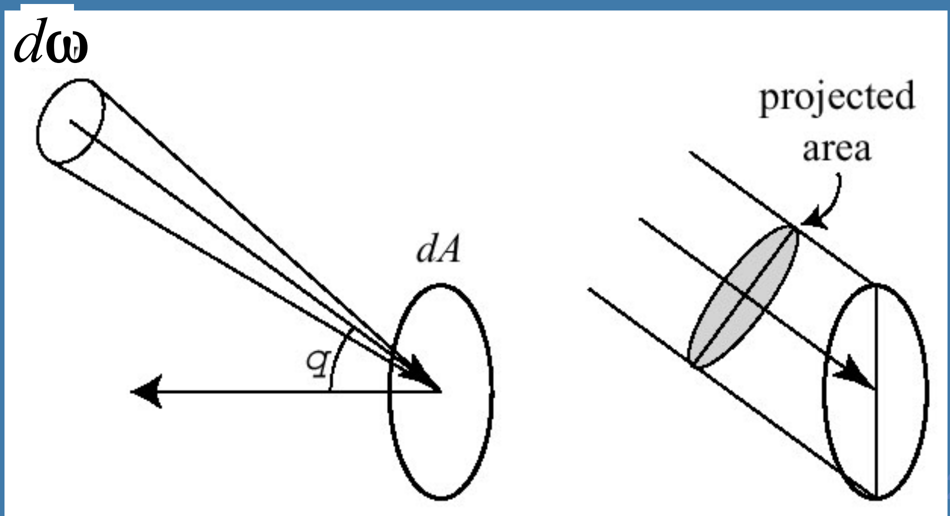
- The ultimate goal is to simulate all light paths: L(S|D|G)*E
- Using this notation, we can find what ray tracing can handle:
  - LDS*E | LS*E = LD?S*E
  - This is clearly not L(S|D|G)*E

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
- 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 infinitesmally small: a ray



Background: The rendering equation

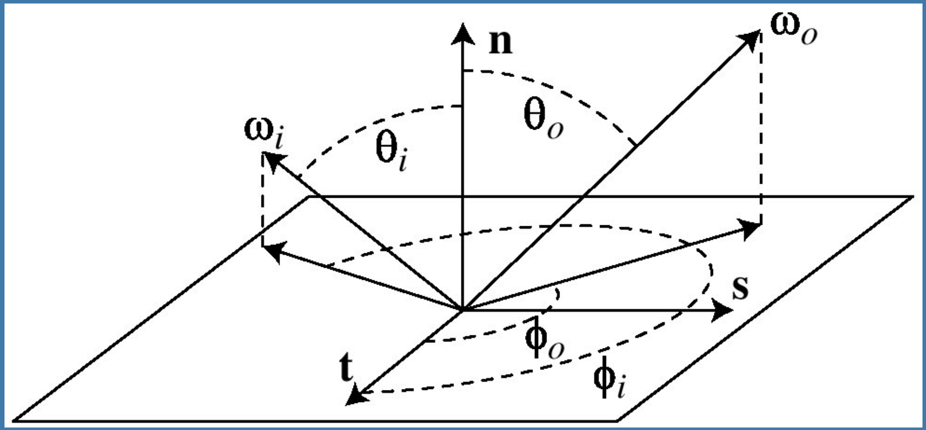
- Paper by Kajiya, 1986 (see course website).
- Is the basis for all rendering, but especially for global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$ (slightly different terminology than Kajiya)
  - outgoing=emitted+reflected radiance
  - $\mathbf{x}$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

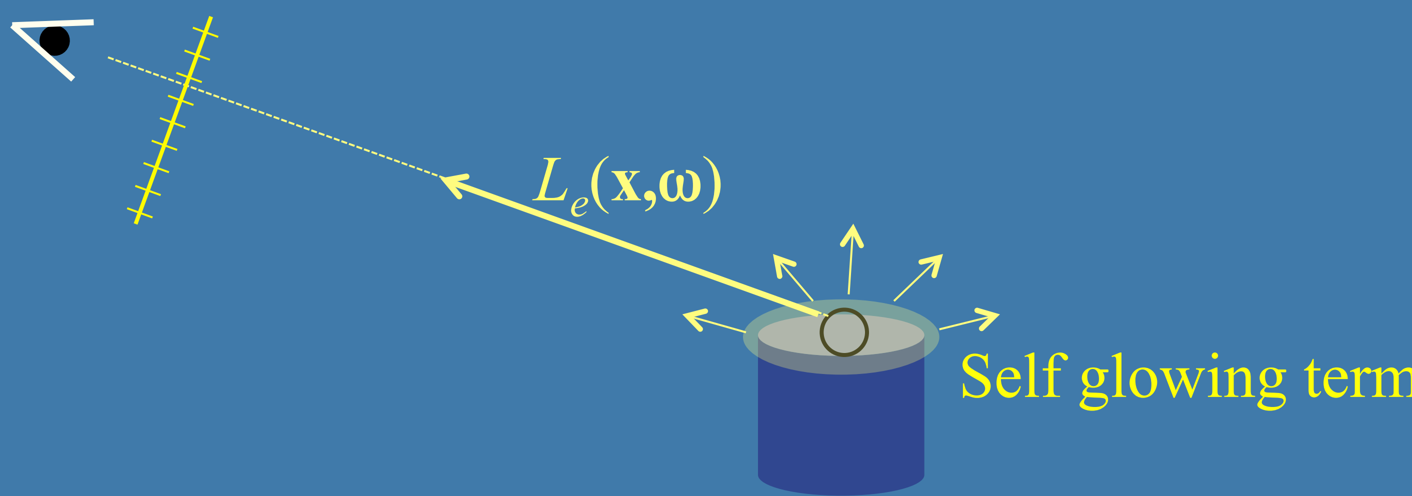
Background: Briefly about BRDFs

- 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



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

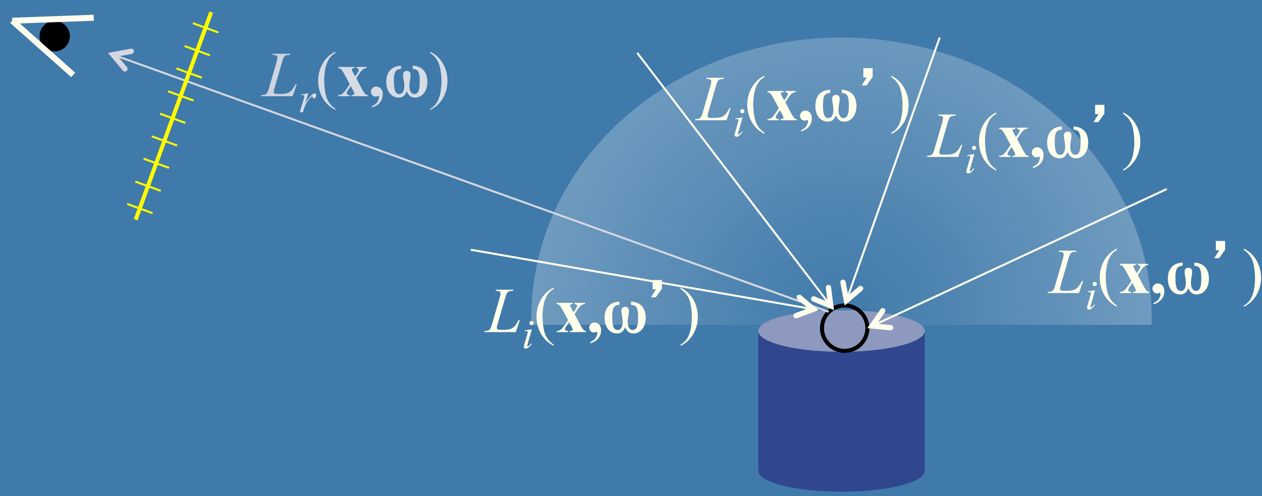


$$L_o = \underline{L_e} + \int_{\Omega} f_r(\mathbf{x},\omega,\omega')L_i(\mathbf{x},\omega')(\omega'\cdot\mathbf{n})d\omega'$$

- 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

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**.

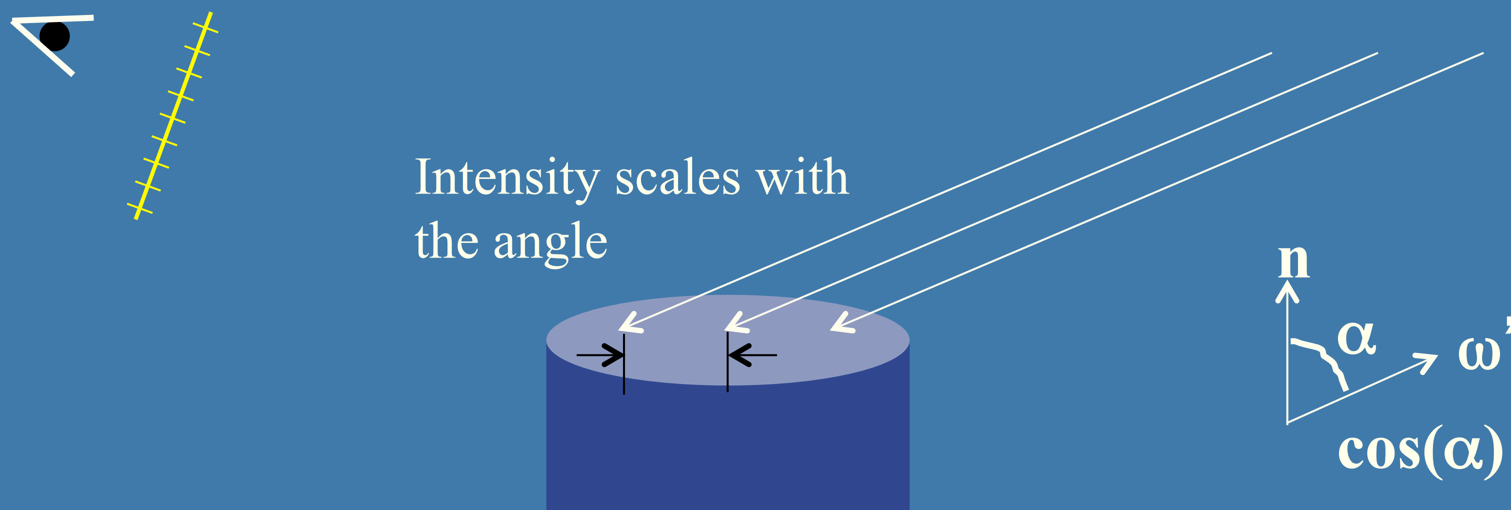
$$L_o = L_e + \int_\Omega f_r(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')L_i(\mathbf{x},\boldsymbol{\omega}')(\boldsymbol{\omega}'\cdot\mathbf{n})d\boldsymbol{\omega}'$$

- 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

Many GI algorithms are built on Monte Carlo Integration

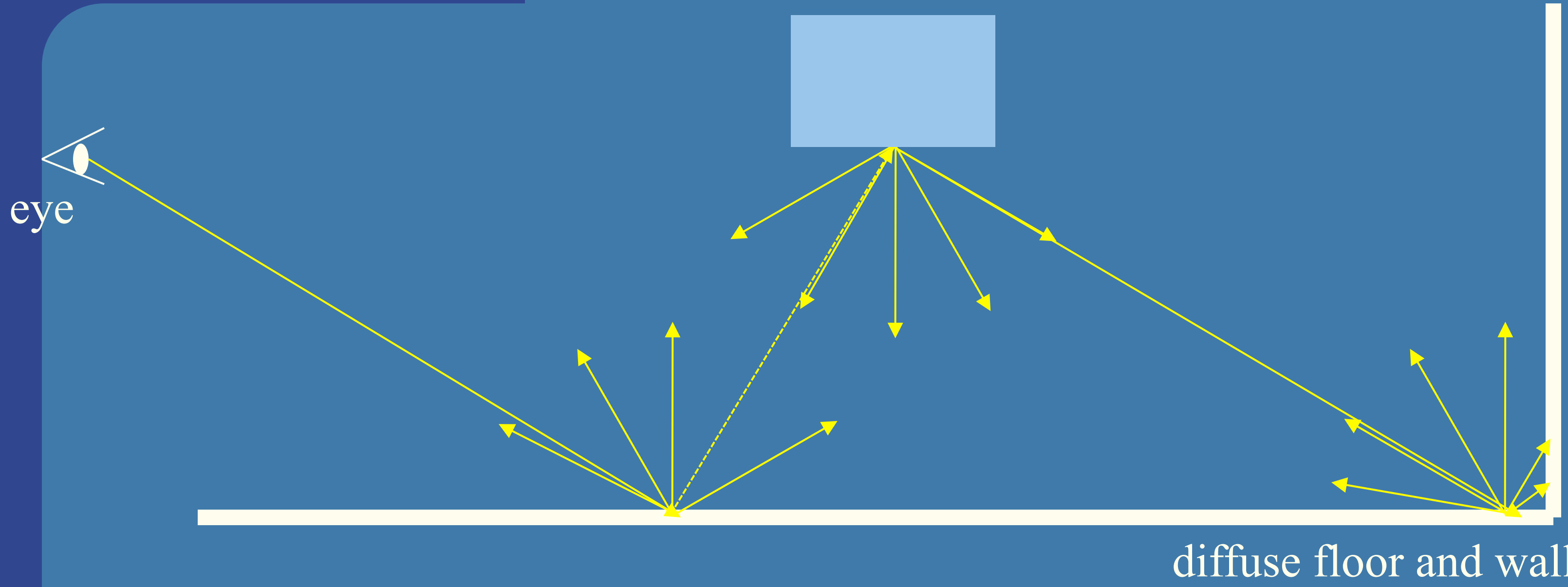
- Integral in rendering equation
- Hard to evaluate
- MC can estimate integrals: $I = \int_a^b f(x)dx$
- Assume we can compute the mean of $f(x)$ over the interval $[a,b]$
  - Then the integral is mean*(b-a)
- Thus, focus on estimating mean of $f(x)$
- Idea: sample f at n uniformly distributed random 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 (naively)

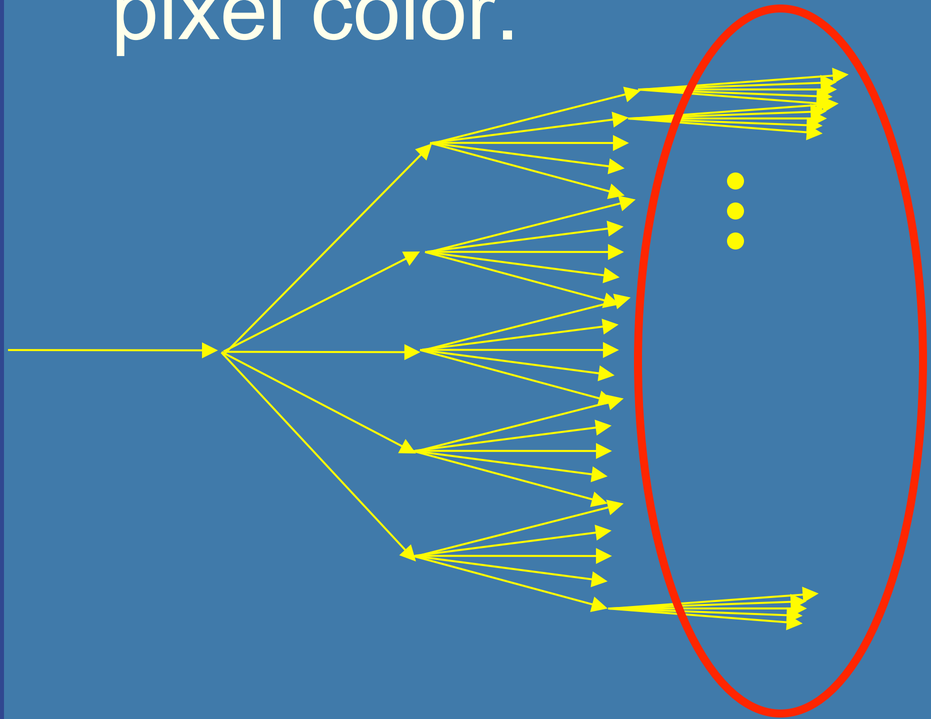


- Sample indirect illumination by shooting sample rays over the hemisphere, at each hit.

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

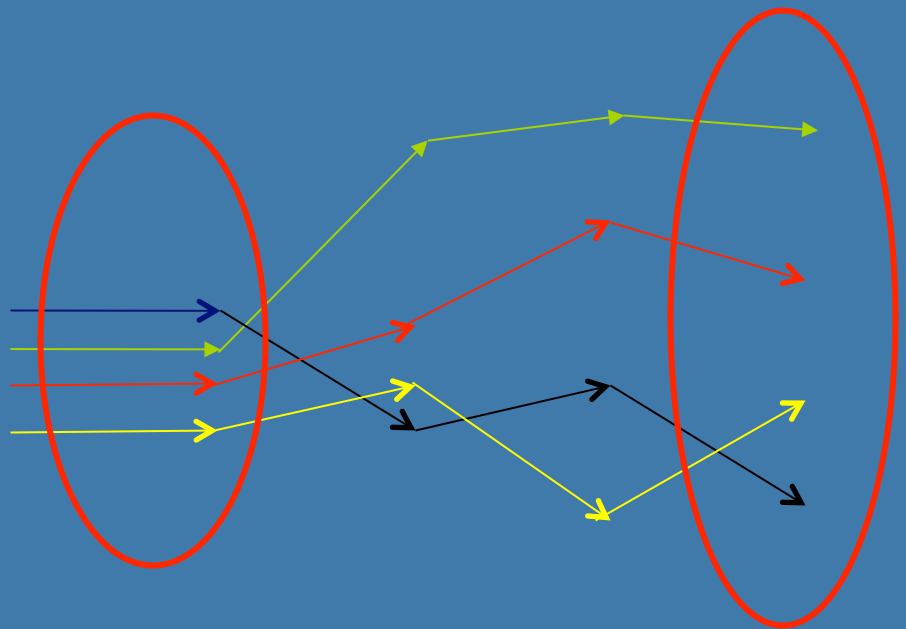
Monte Carlo Ray Tracing (naively)

- 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.



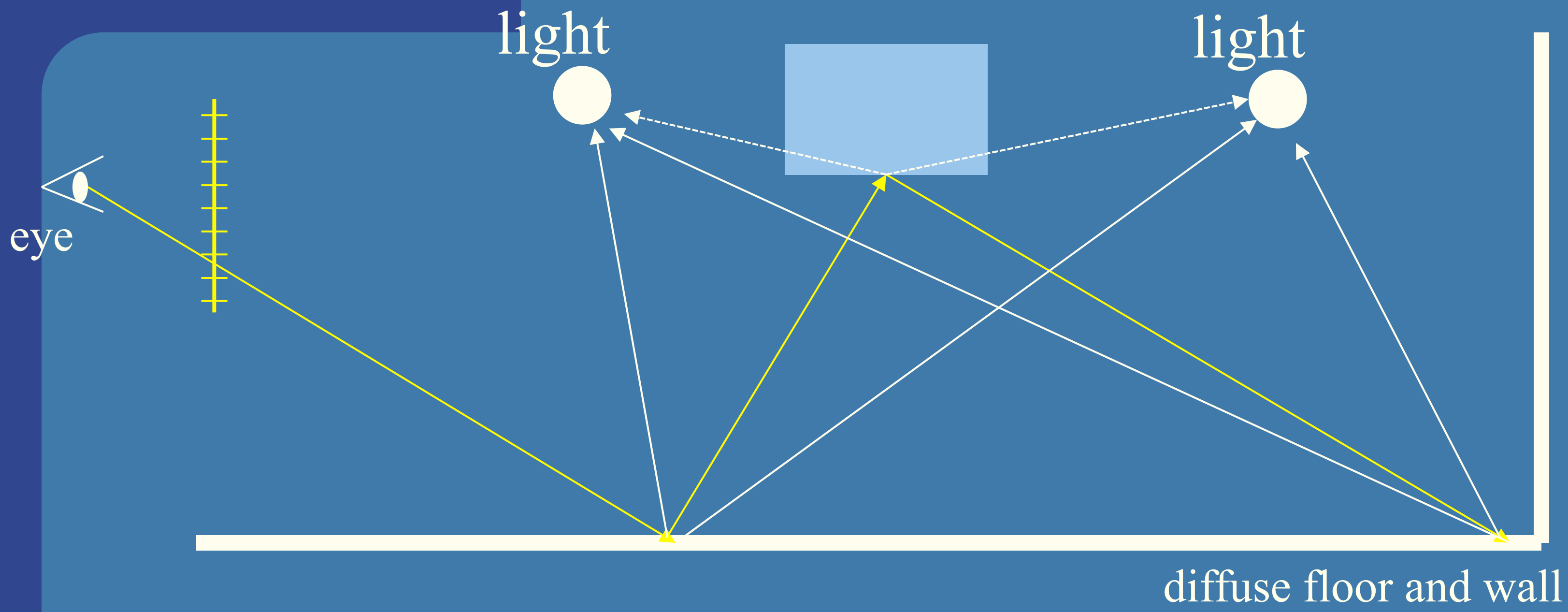
PathTracing
– one efficient Monte-Carlo Ray Tracing solution

- 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.



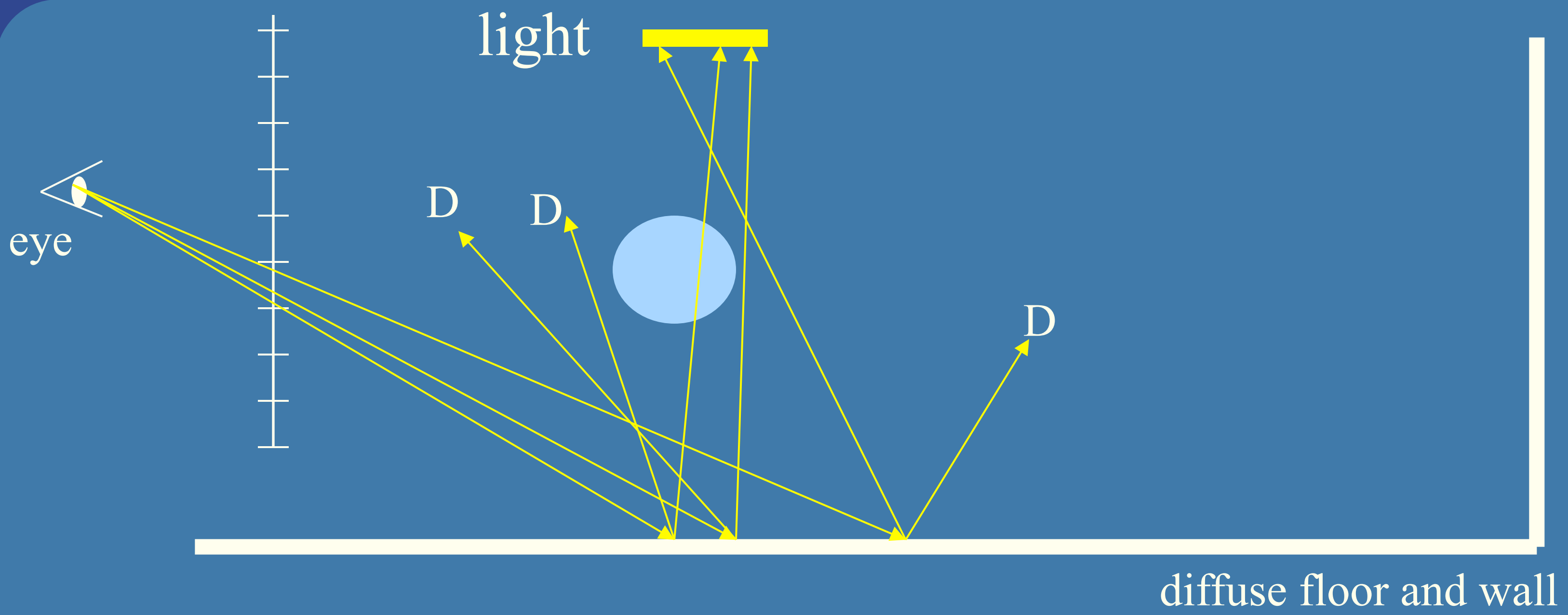
Equally number of rays are traced at each level

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 then randomly select one new ray direction.

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

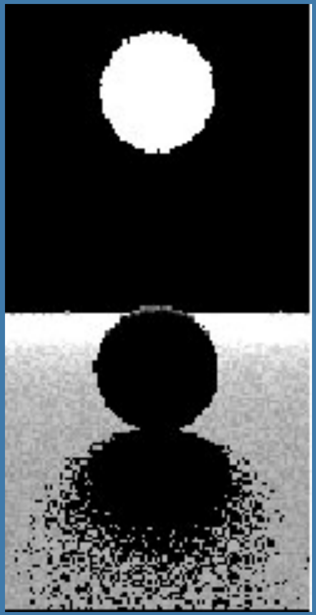


- Example: Three rays for one pixel
- All three rays hits diffuse floor
- Pick ***one*** random position on light source
- Send one random diffuse ray (D's above)
  - in order to continue the path...

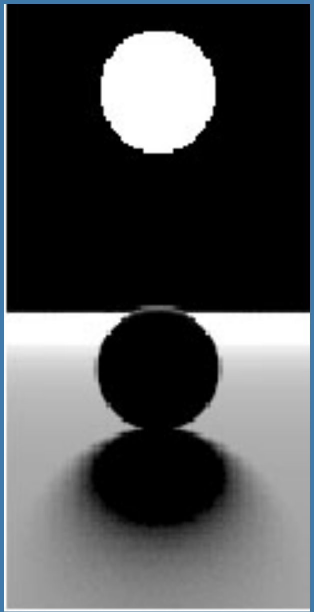
Path tracing: One solution to GI
See section 6 in Kajiya's paper

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

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

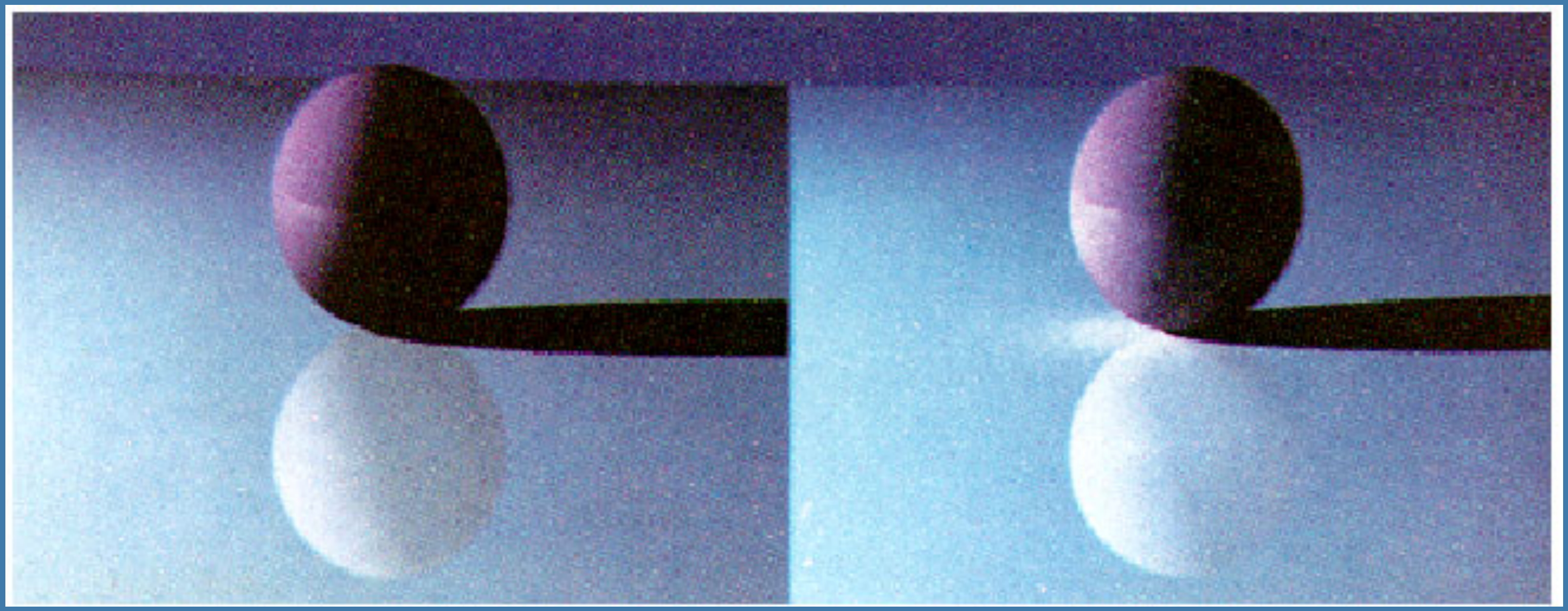
Images courtesy of Peter Shirley

Perfectly Diffuse and Perfectly Specular surfaces in path tracing

- Assume $k_{diff}+k_{spec}$<=1
  - Comes from that energy cannot be created, but can be absorbed
  - k_{diff} can be sum of diffuse color, (R+G+B)/3, and same for k_{spec}.
- When a ray hits such a surface
  - Pick a random number, r in [0,1]
  - If($r < k_{diff}$) → send diffuse ray (e.g. in random direction)
  - Else if($r < k_{diff}+k_{spec}$) → send specular ray (e.g. along reflection direction)
  - Else absorb ray.
- This is often called Russian roulette

A classical example – spec+diff surface + hard shadow

- Path tracing was introduced in 1986 by Jim Kajiya



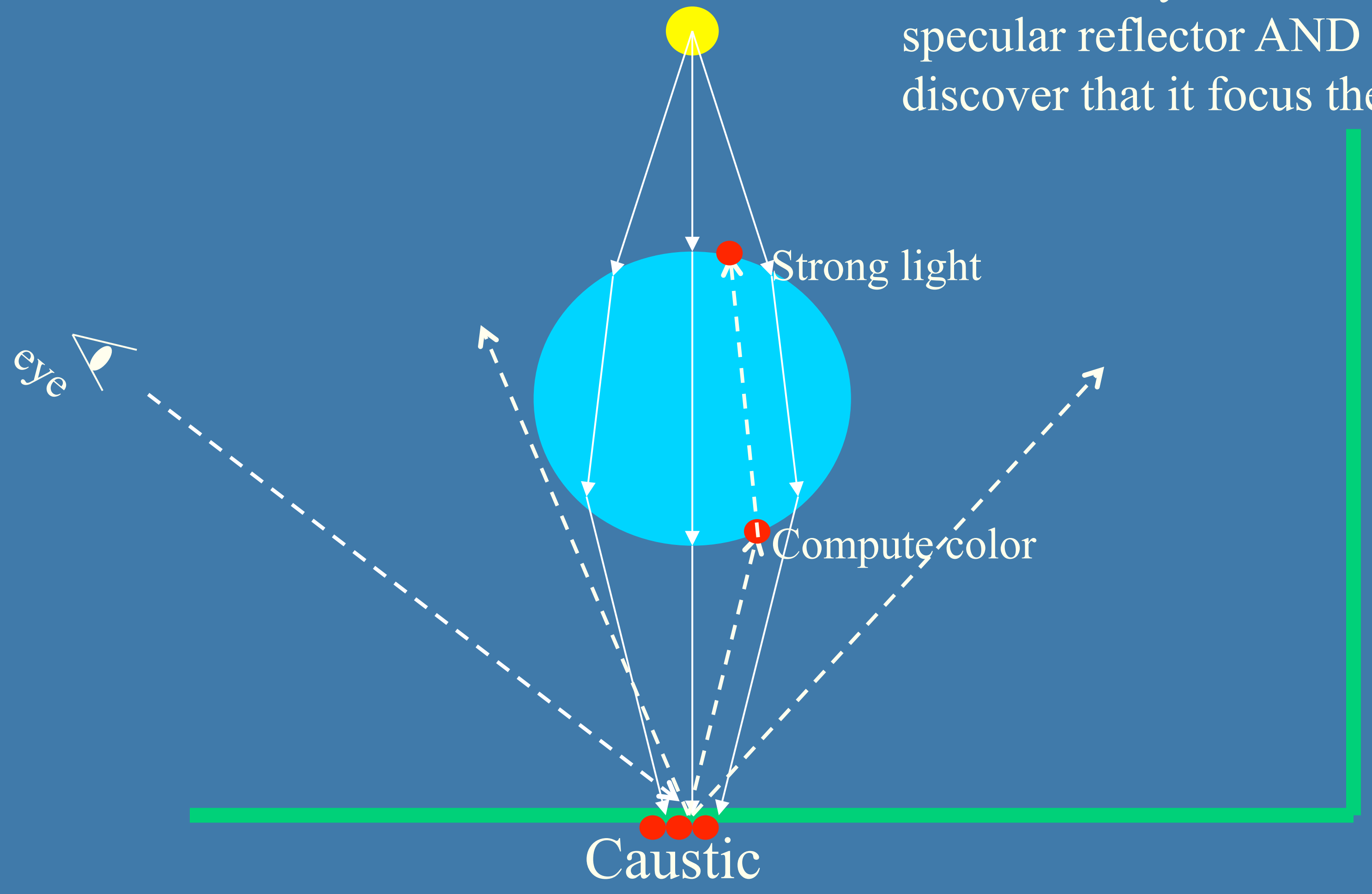
- Note how the right sphere reflects light, and so the ground under the sphere is brighter

What is Caustics?

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

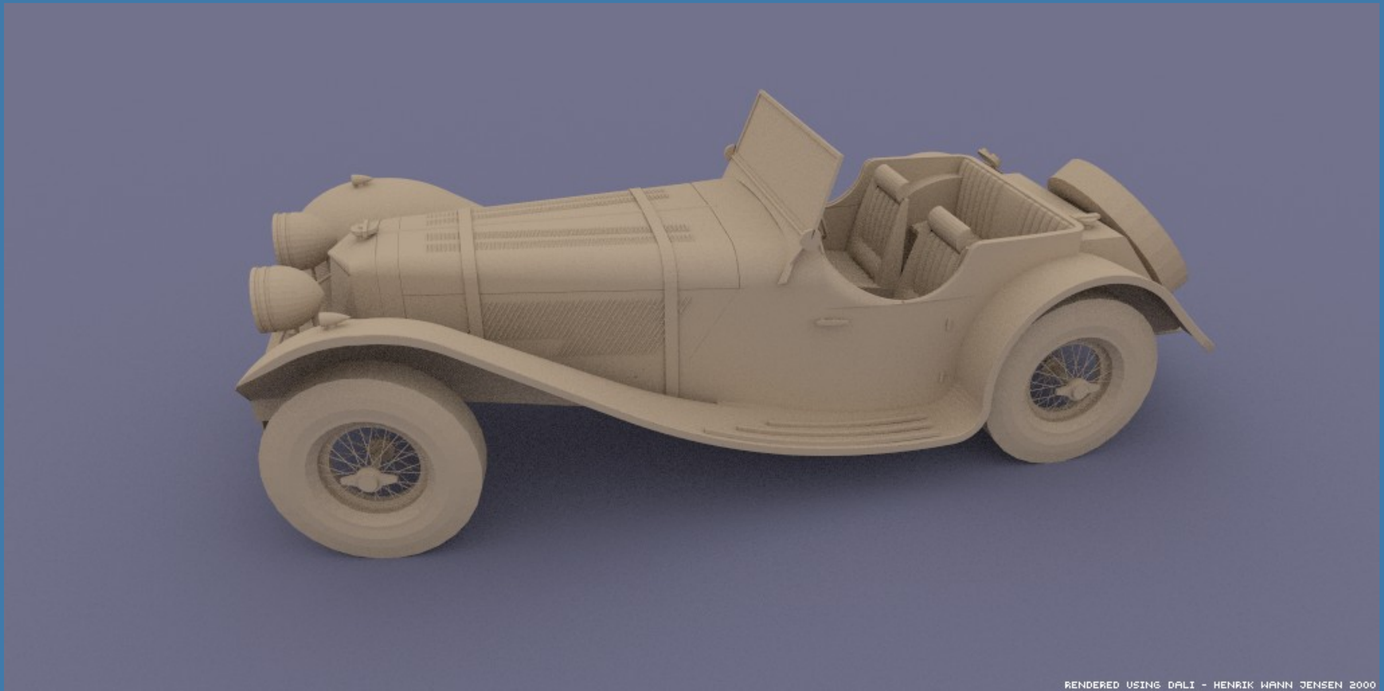
Reason why forward ray tracing fails to capture caustics well

Must be lucky to hit the specular reflector AND discover that it focus the light



Example when path tracing works well

- When indirect illumination varies slowly and no specularity
  - An example with strong indirect illumination is caustics (concentrated refracted light)
- Example from Henrik Wann Jensen
- 100 paths per pixel
- 140,000 triangles
- 1024x512 in 20 min. on a PIII-500

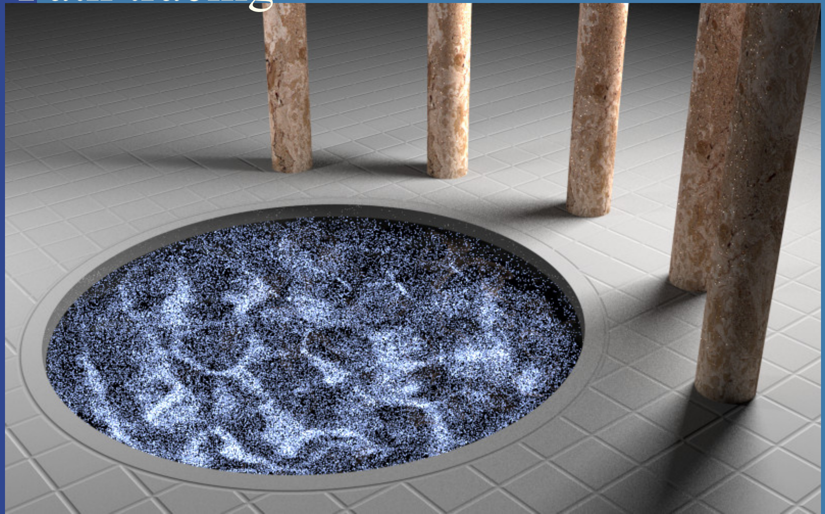


Extensions to path tracing

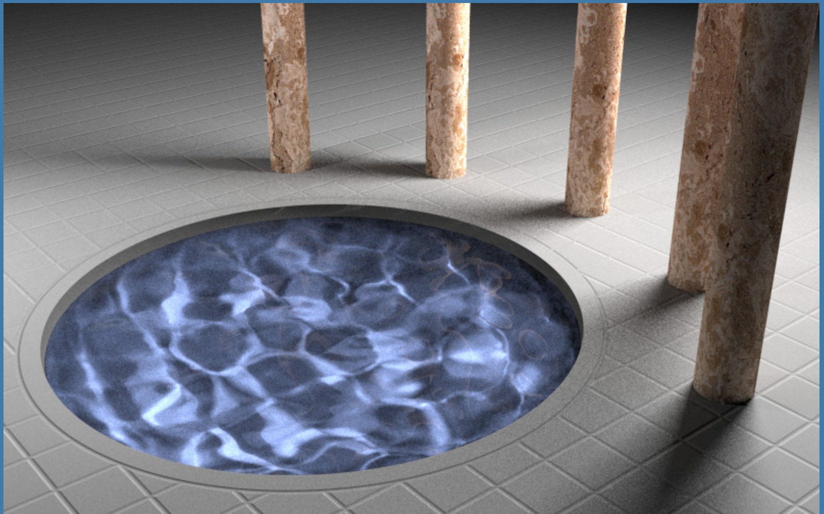
- Bidirectional path tracing
  - Developed in1993-1994
  - Sends light paths, both from eye and from the light
  - Faster, but still noisy images.
- Metropolis light transport
  - 1997
  - Ray distribution is proportional to unknown function
  - Means that more rays will be sent where they are needed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

Images courtesy of Eric Veach

Path tracing



Metropolis (same rendering time)



Bidirectional Path tracing

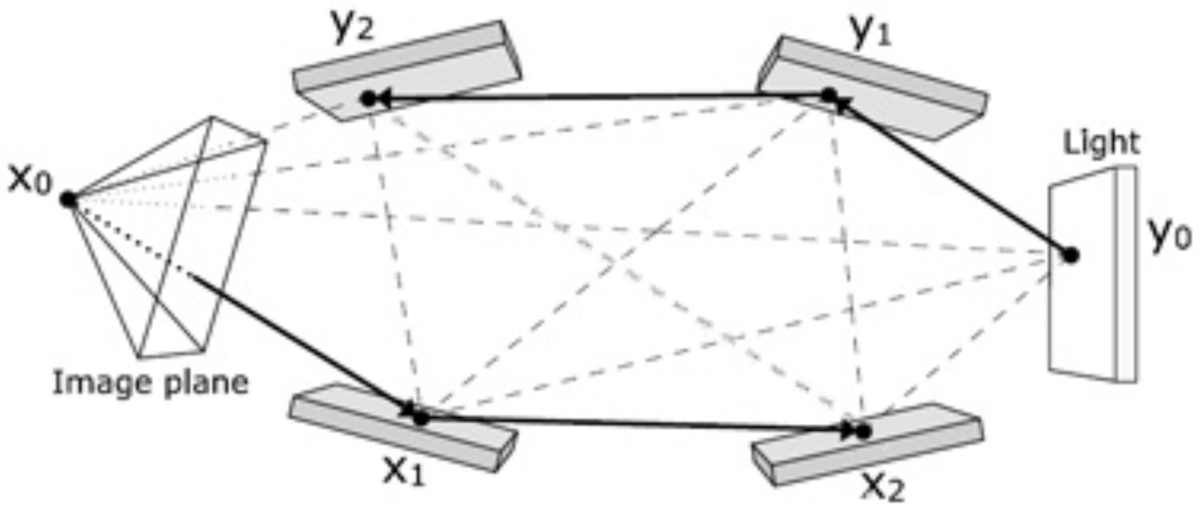


Figure 1: The different ways to combine the eye path and the light path. The eye path starts at the eye in X0 and moves through X1 and X2. The light path starts at the light source in Y0 and moves through Y1 and Y2. To create a complete path, a shadow ray is used to connect X2 and Y2. More paths can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

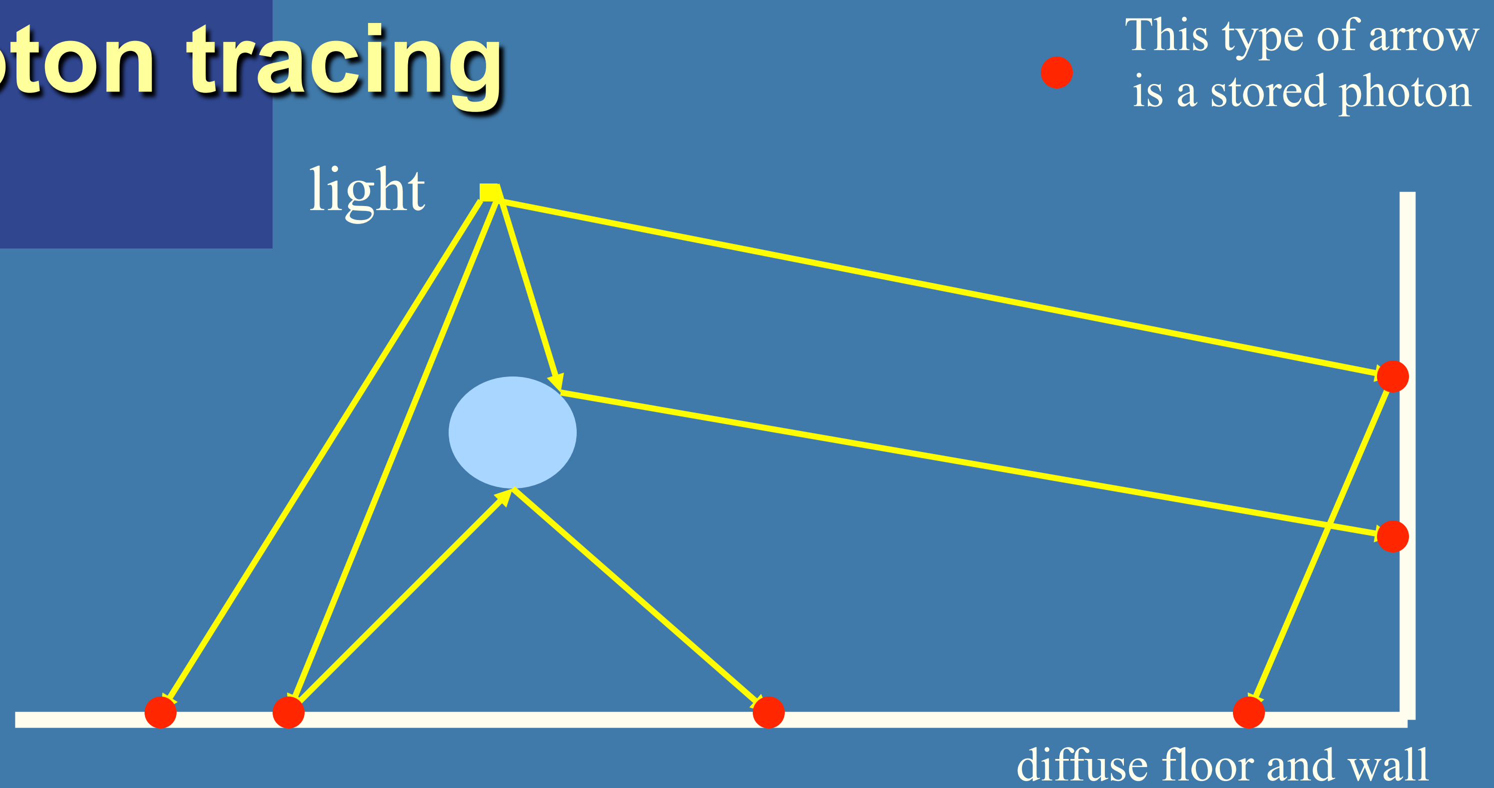
Photon mapping

- 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



- Should not store photon at specular surfaces, because these effects are view dependent
  - only diffuse effect is view independent
- At hit, photon gets colored (looses intensity)
  - E.g., white photon (1,1,1) becomes pink (0.8, 0.5, 0.5), so looses intensity.
  - Instead of decreasing intensity, decrease probability of further scatter the photon. (E.g., probability of absorbing photon = 1/5)
  - Why not just decrease the photon's intensity?
    - Harder to get good filtering by expanding spheres

The photon map data structure

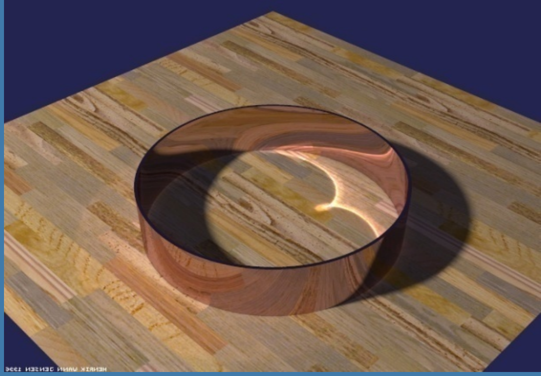
- Keep them in a separate (from geometry) structure
- Store all photons in kD-tree
  - Essentially an axis-aligned BSP tree, since we must alter splitting axis: x,y,z,x,y,z,x,y,z, etc.
  - Each node stores a photon
  - Needed because the algorithm needs to locate the n closest photons to a point
- A 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

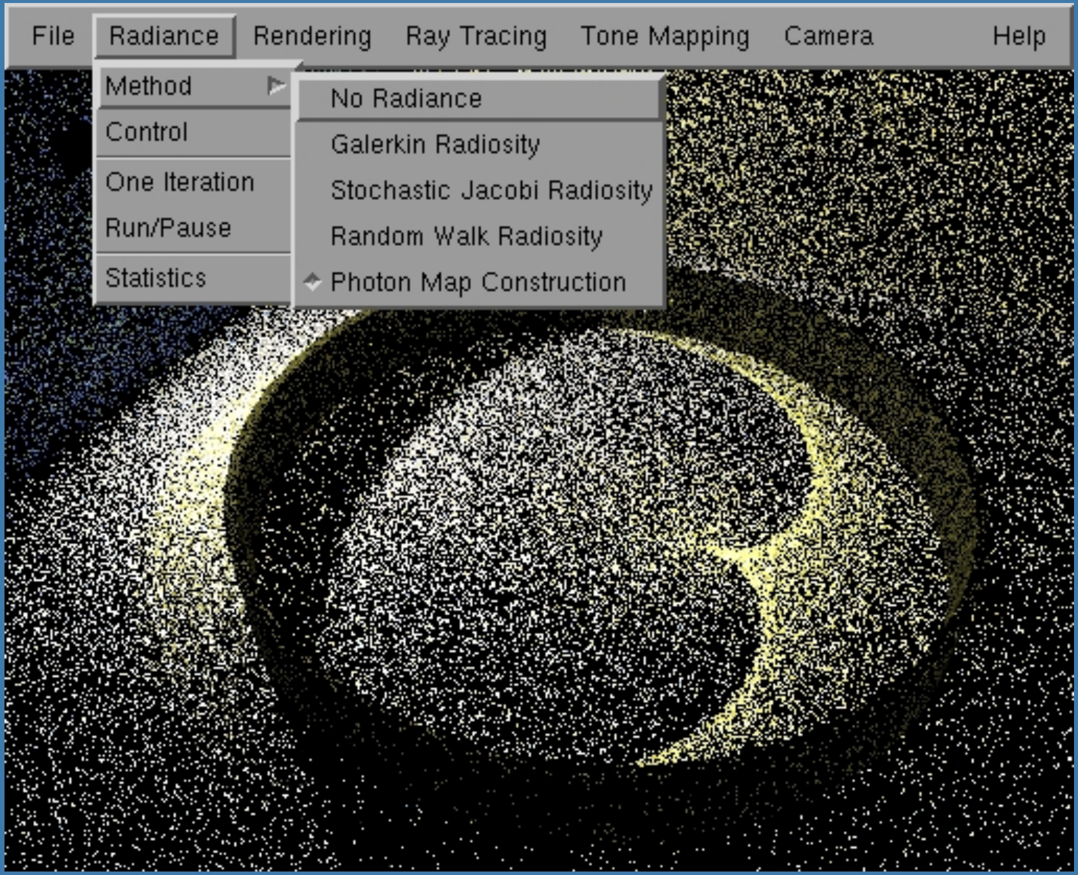
After Henrik Wann Jensen

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

What does it look like?



- Stored photons displayed:



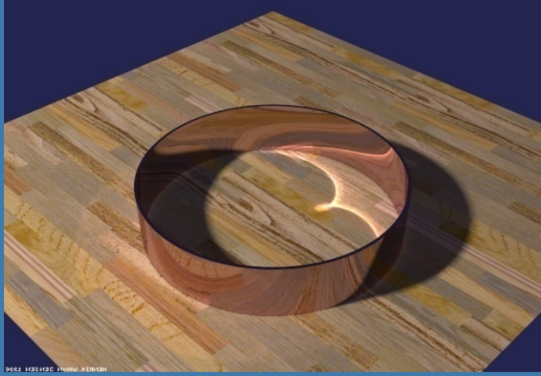
Density estimation

- 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
- ω_p is the direction of the stored photon
- Φ_p is the stored power of the photon
- f_r is the BRDF

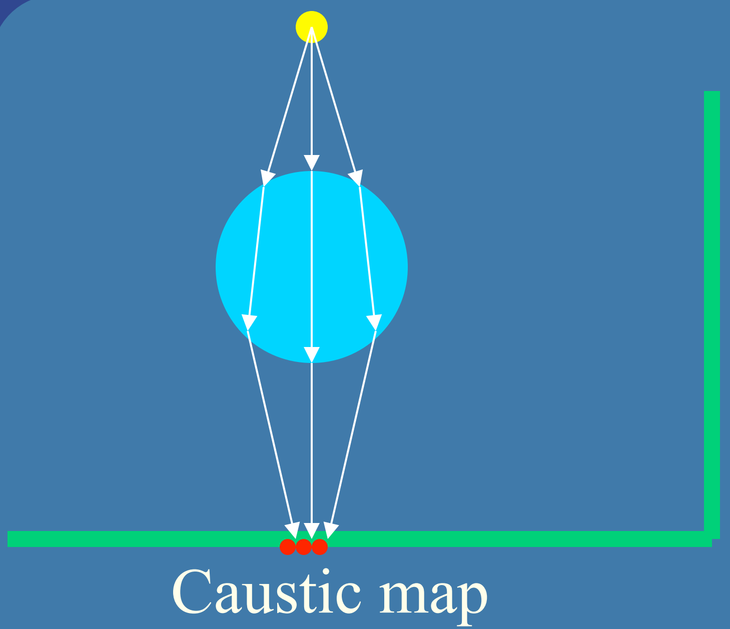
Two-pass algorithm

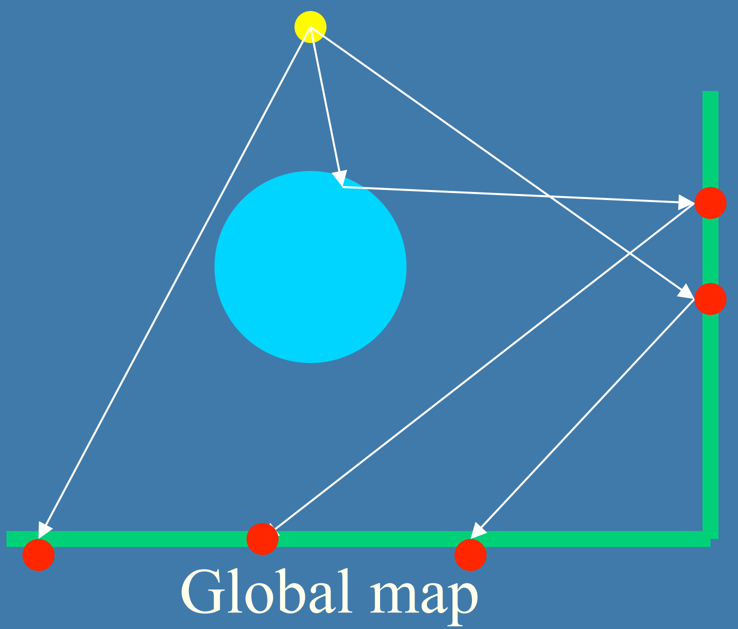


- Already said:
  - 1) Photon tracing, to build photon maps
  - 2) Rendering from the eye using photon maps
- Pass 1 (create photon maps):
  - Use two photon maps
  - A caustics photon map (for caustics)
    - Stores photons that have been reflected or refracted (via a specular/transparent surface) to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



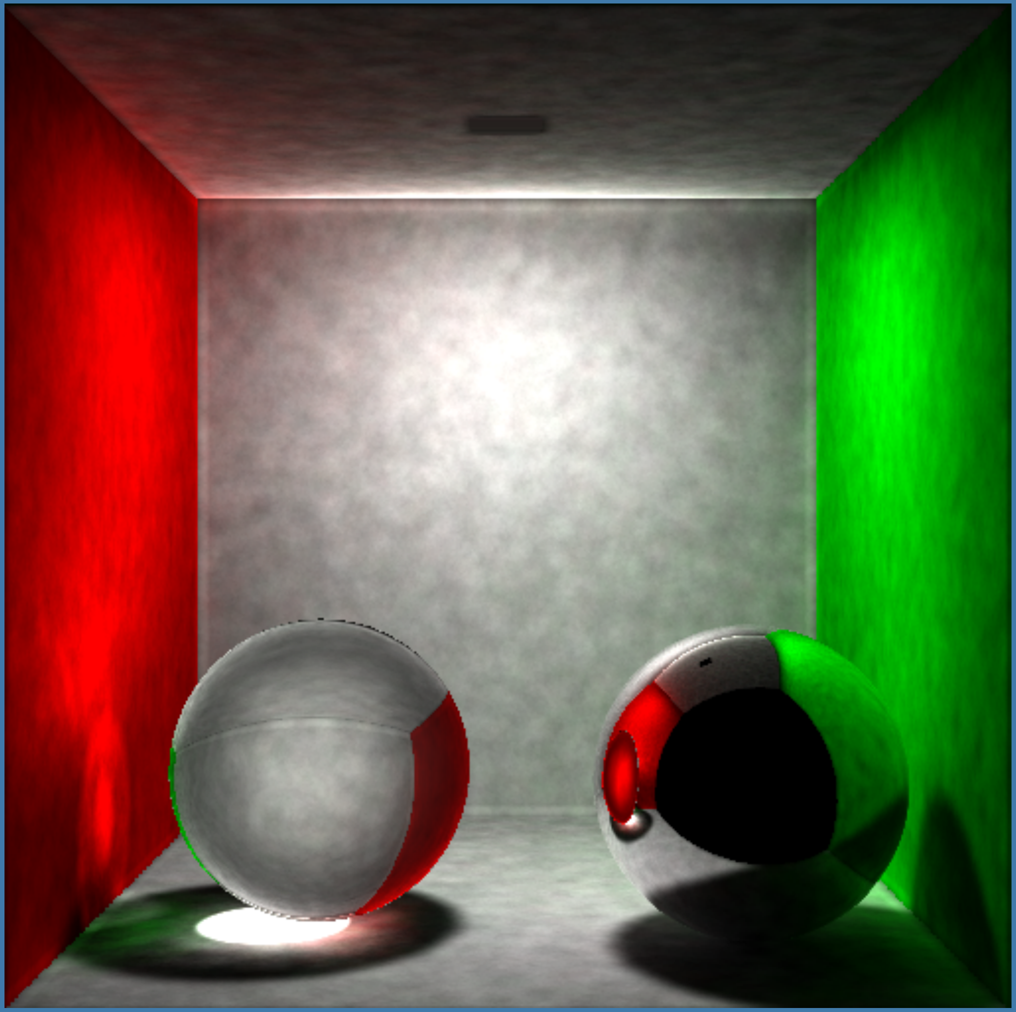


- Caustic map: send photons only towards reflective and refractive surfaces
  - Caustics is a high frequency component of illumination
  - Therefore, need many photons to represent accurately
- Global map - assumption: illumination varies more slowly

Pass 2: Rendering using the photon map

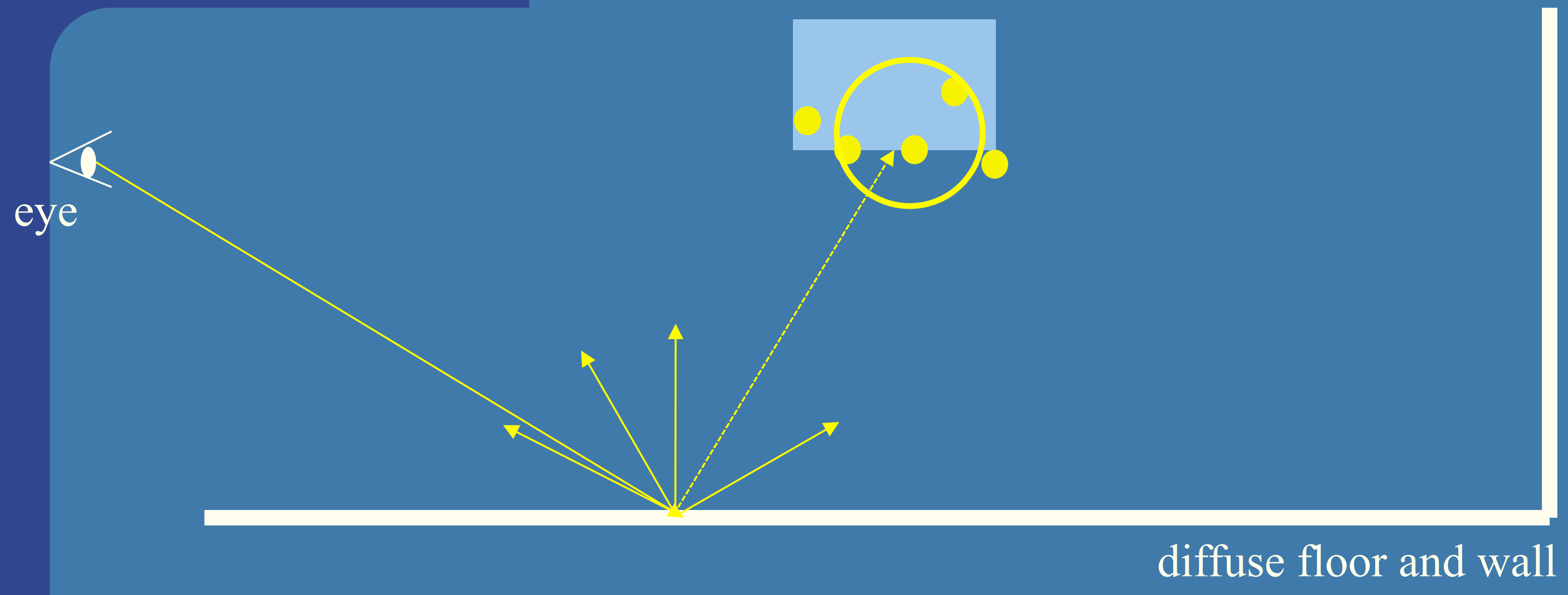
- Render from the eye using a modified ray tracer
  - A number of rays are sent per pixel
  - For each ray, evaluate four terms
    - **Direct illumination** (light that reaches a surface directly from light source)… may need to send many rays to area lights. Done using standard ray tracing.
    - **Specular reflection** (also evaluted 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



Solution: use Final Gather instead...

A modification for indirect Illumination – Final Gather

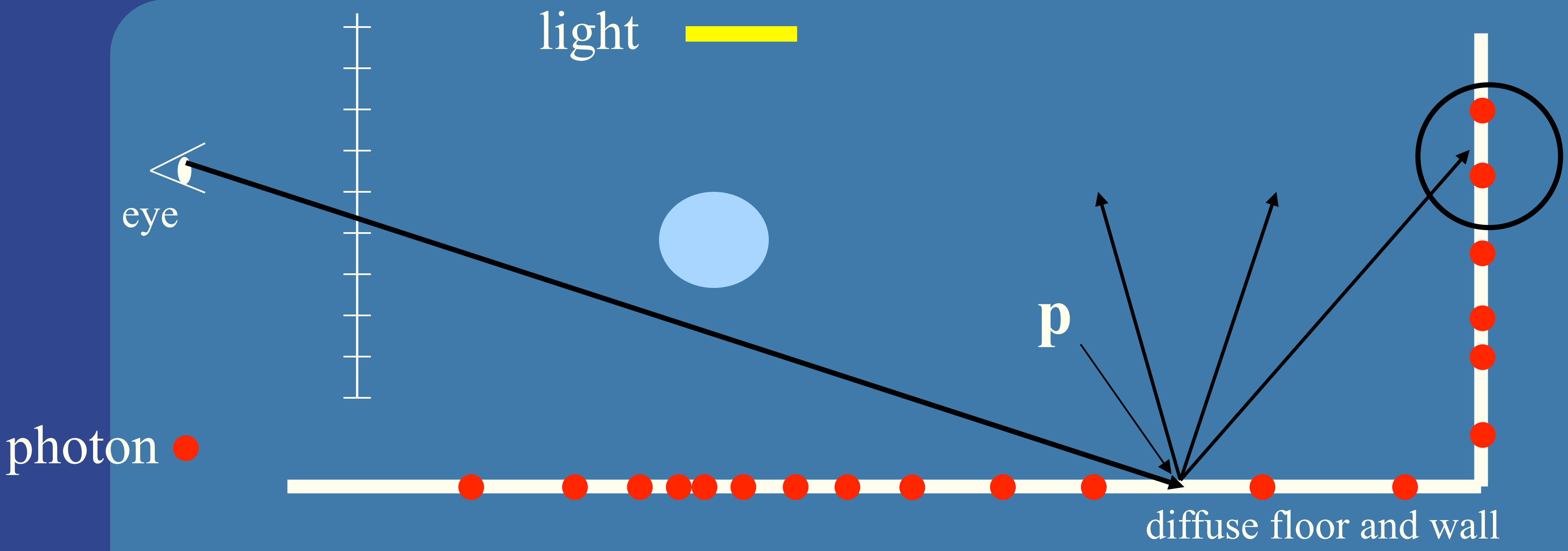


- Too noicy to use the global map for direct visualization
- Remember: eye rays are recursively traced (via reflections/ refractions) until a diffuse hit, **p**. There, we want to estimate slow-varying indirect illumination.
  - Instead of growing sphere in global map at **p**, Final Gather shoots 100-1000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

Final Gather

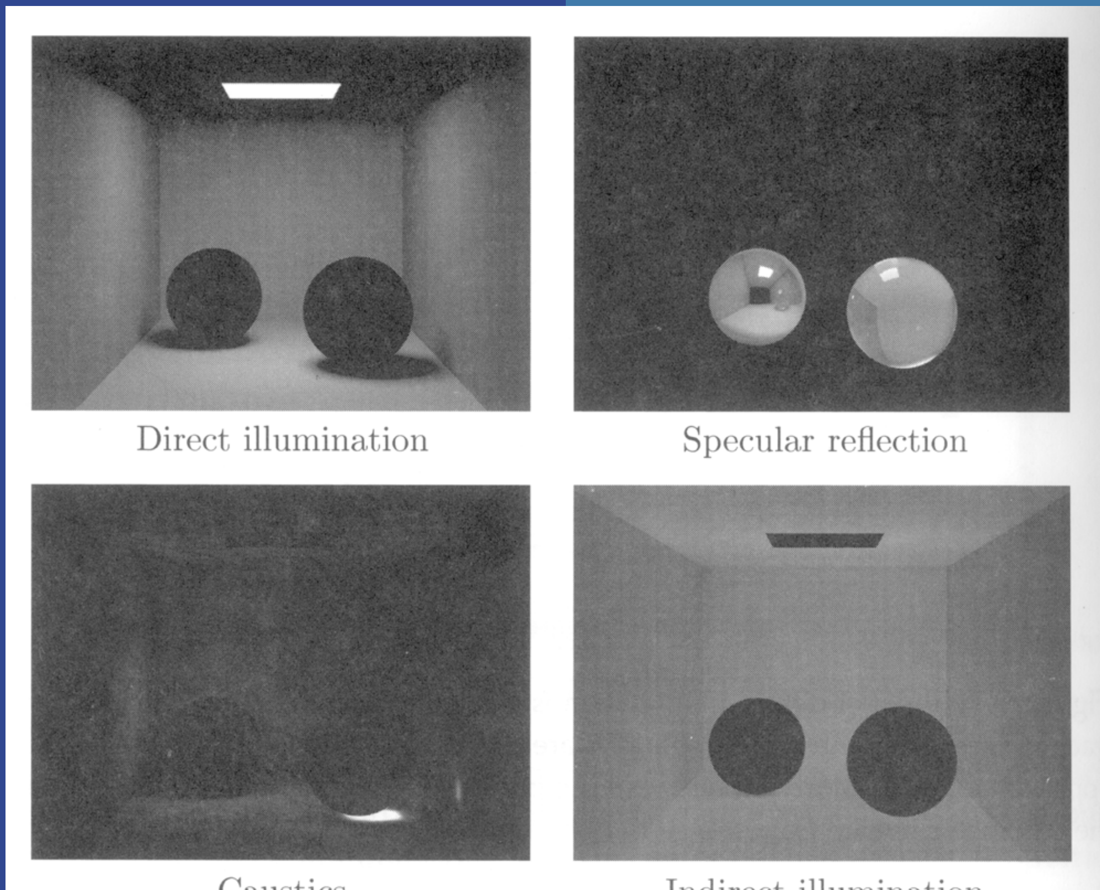
- Final gathering is a technique for estimating global illumination for a given point by either sampling a number of directions in the hemisphere over that point (such a sample set is called a *final gather point*), or by averaging a number of final gather points nearby since final gather points are 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

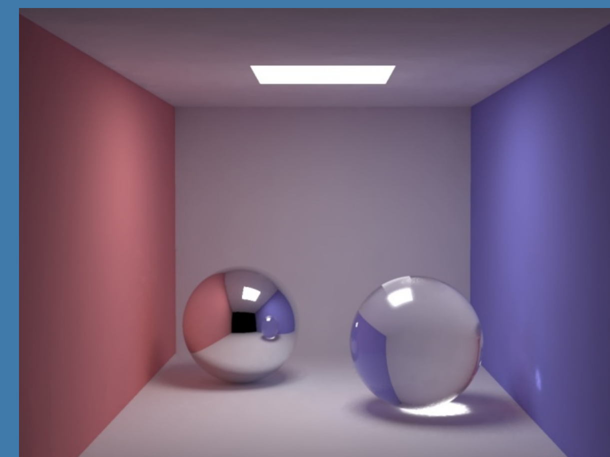


- To evaluate indirect illumination at point **p**:
  - Send several random rays out from **p**, and grow spheres at contacts
  - May need several hundreds of rays to get good results.

Images of the four components



=



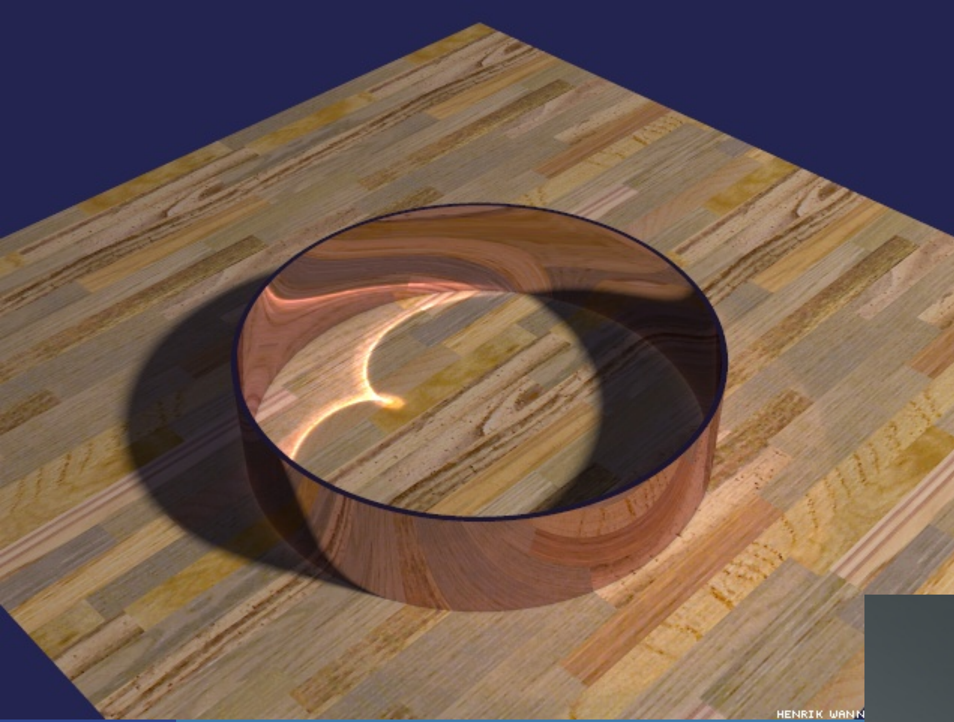
- These together solves the entire rendering equation!

Image scanned from Henrik Wann Jensen's book

Photon Mapping

- 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 **p**, compute direct illumination (shadow rays + shading).
  - Also grow sphere around each **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 **p** by sampling the hemisphere with ~100 − 1000 rays and use the two photon maps where those rays hit a surface.
- Growing sphere:
  - Uses the kd-tree to expand a sphere around **p** until a fixed amount (e.g. 50) photons are inside the sphere. The radius is a measure of the intensity of indirect light at **p**. The BRDF at **p** could also be used to get a more accurate color and intensity value.

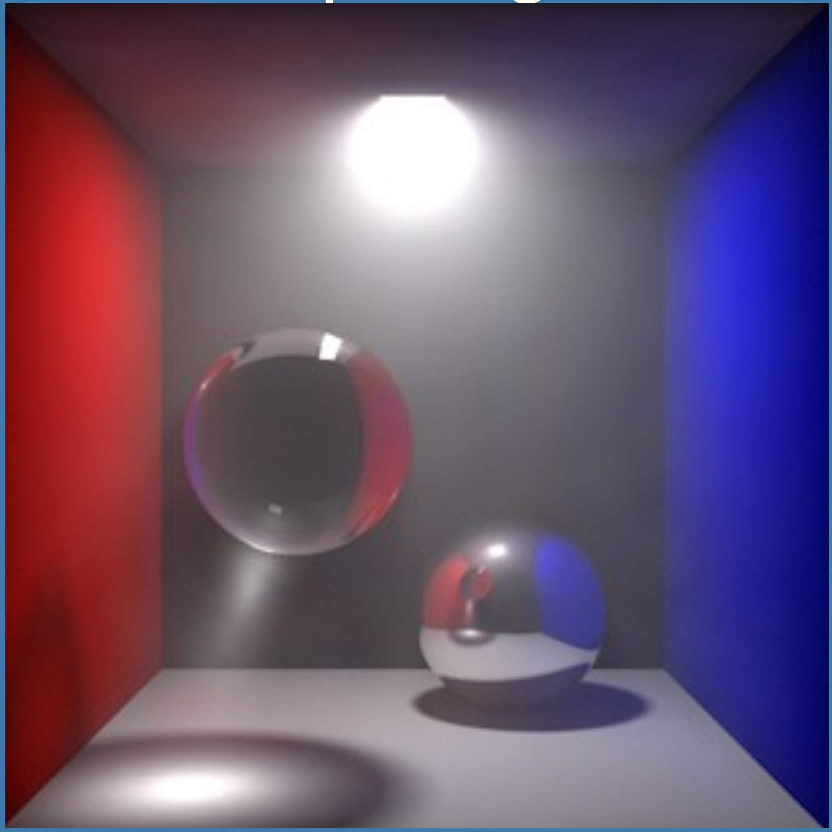
Standard photon mapping

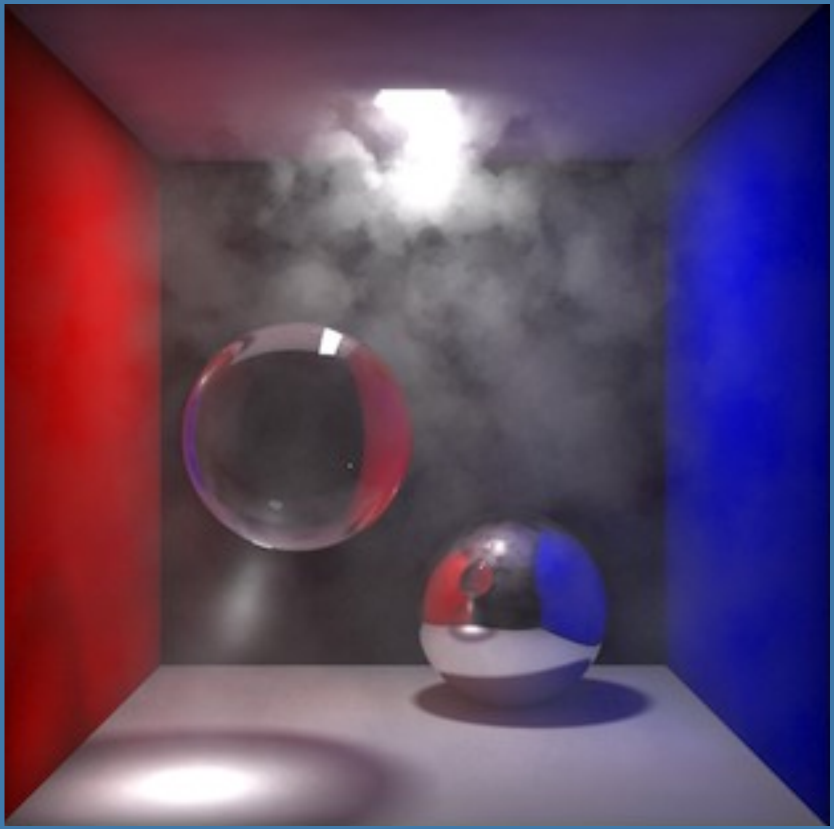


Caustics: concentrated reflected or refracted light

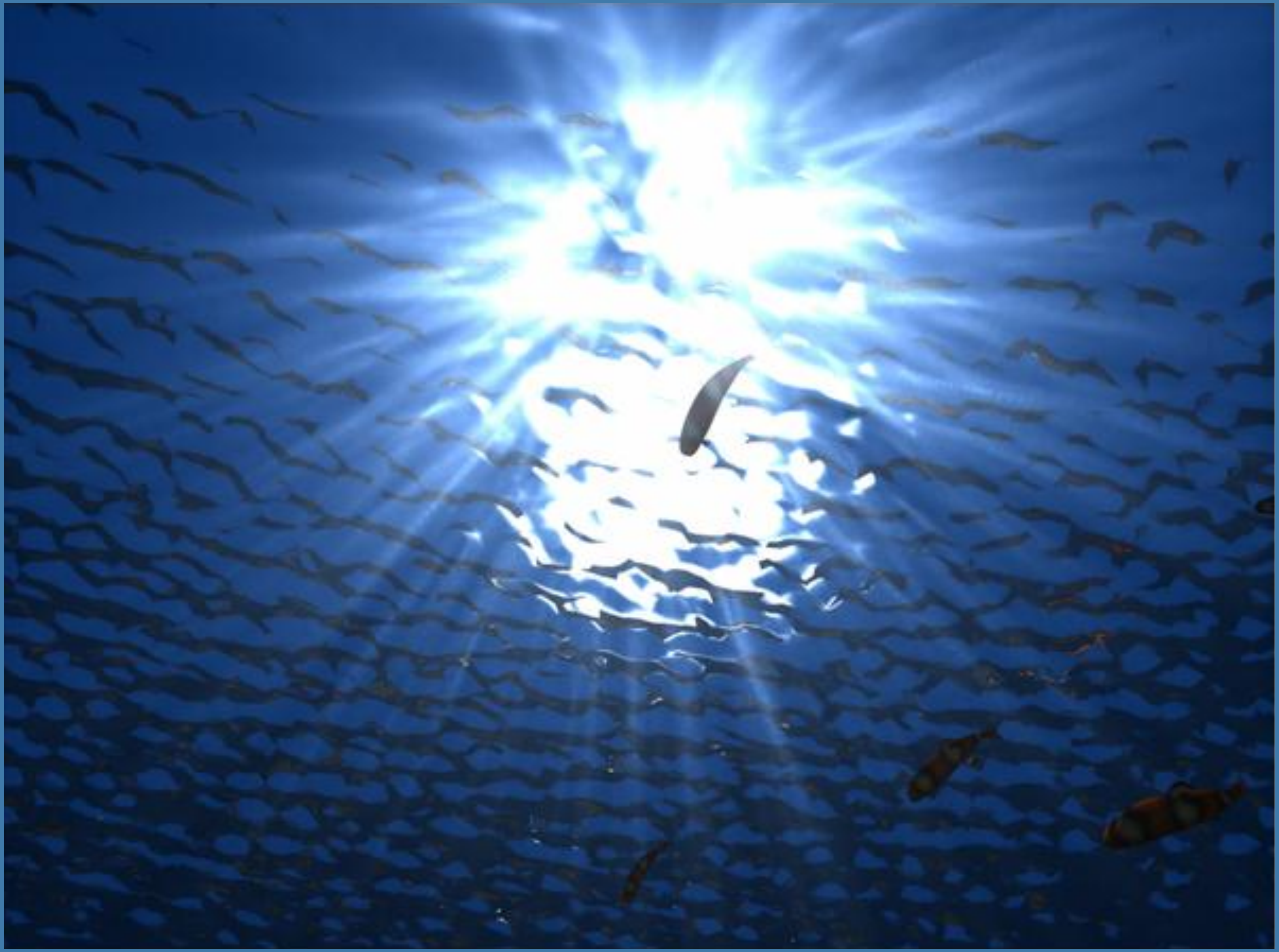
Extensions to photon mapping

- Participating media

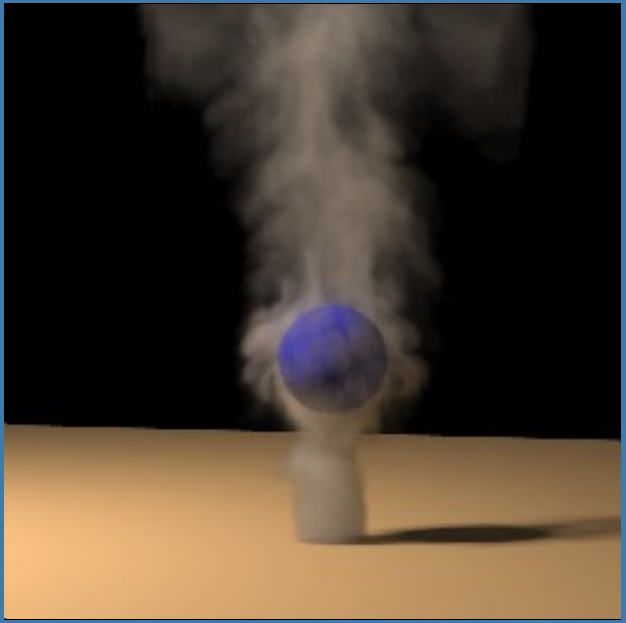




Another one on participating media



Smoke and photon mapping



Press for a movie

Much more details to photon mapping…

- Henrik Wann Jensen, *Realistic Image Synthesis using Photon Mapping*, AK Peters, 2001.
- Check out: Henrik's home page:

  http://graphics.stanford.edu/~henrik/

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)
- More advanced alternatives:
  - Bidirectional path tracing
  - Photon Mapping
  - Metropolis Light Transport

What you need to know

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