An introduction to Global Illumination

Tomas Akenine-Möller
Department of Computer Engineering
Chalmers University of Technology

Modified by Ulf Assarsson

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
  - Modeling and animating using Maya/3DSMax/Blender

GFX Companies Gothenburg

3D software development + some need of artists:
Autodesk,
EON,
Spark Vision
Simbin
MindArk
Mentice
Vizendo
Surgical Science
Combitech
Fraunhofer (Chalmers Teknikpark)
Electronic Arts (new in Gbg)
RD&T Technology
Smart Eye AB
(Spotfire)

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

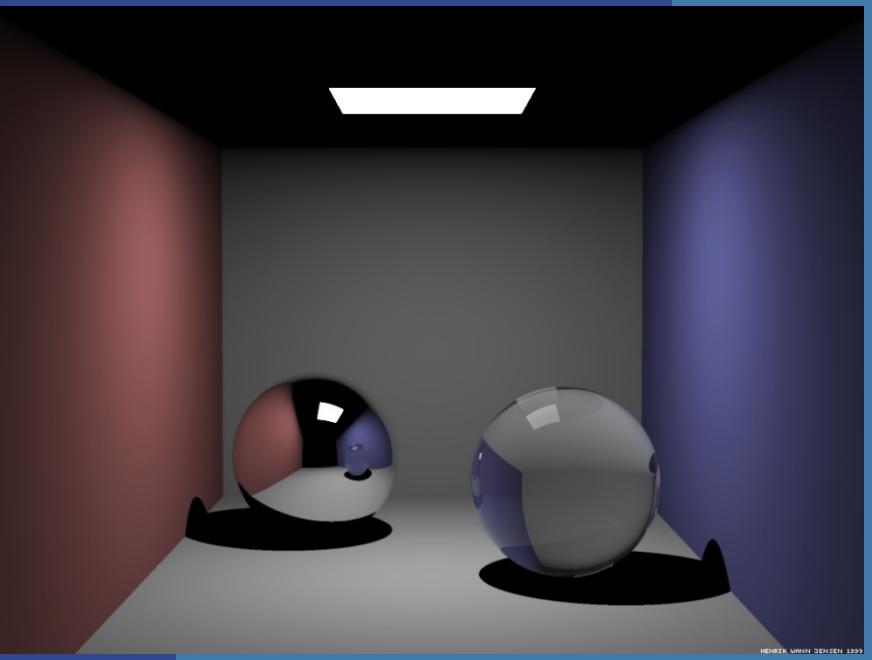
Non-Gothenburg

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

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

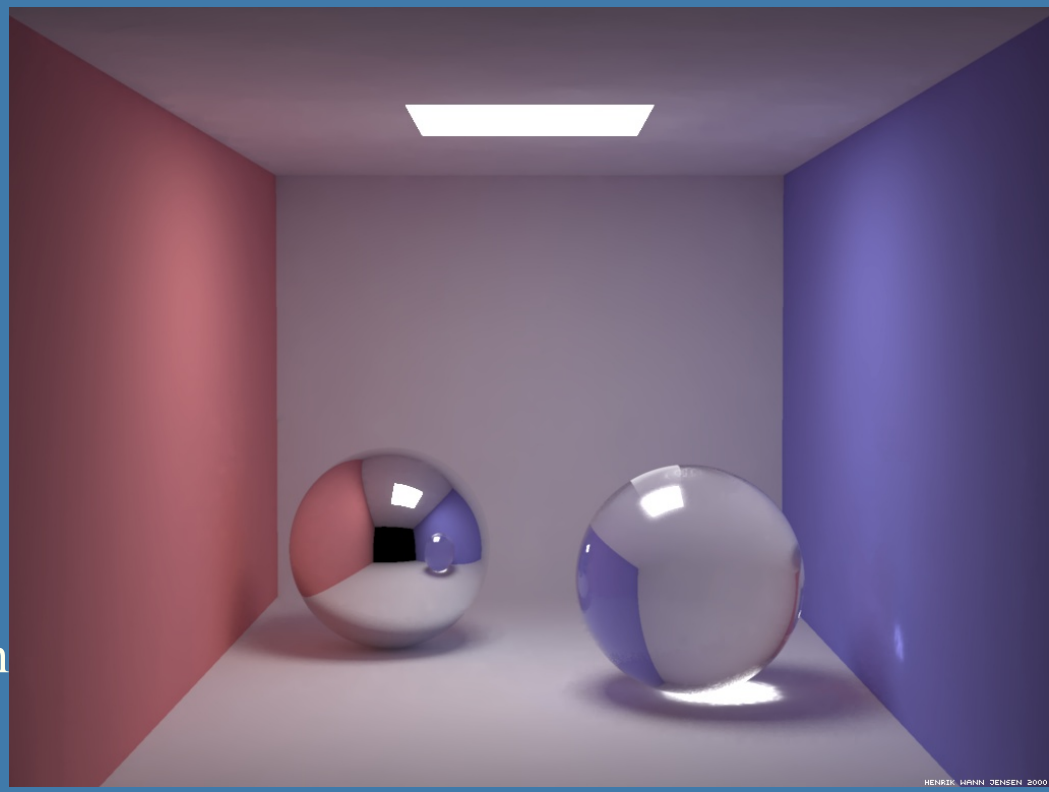
Architects, graphics artists:
White
Wingårdhs
Volvo Personvagnar
Semcon
Ramböll
Zynka
CAP AB
Grafia

Isn't ray tracing enough?



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



Which are the differences?

Global Illumination

Images courtesy of Henrik Wann Jensen

Global Illumination

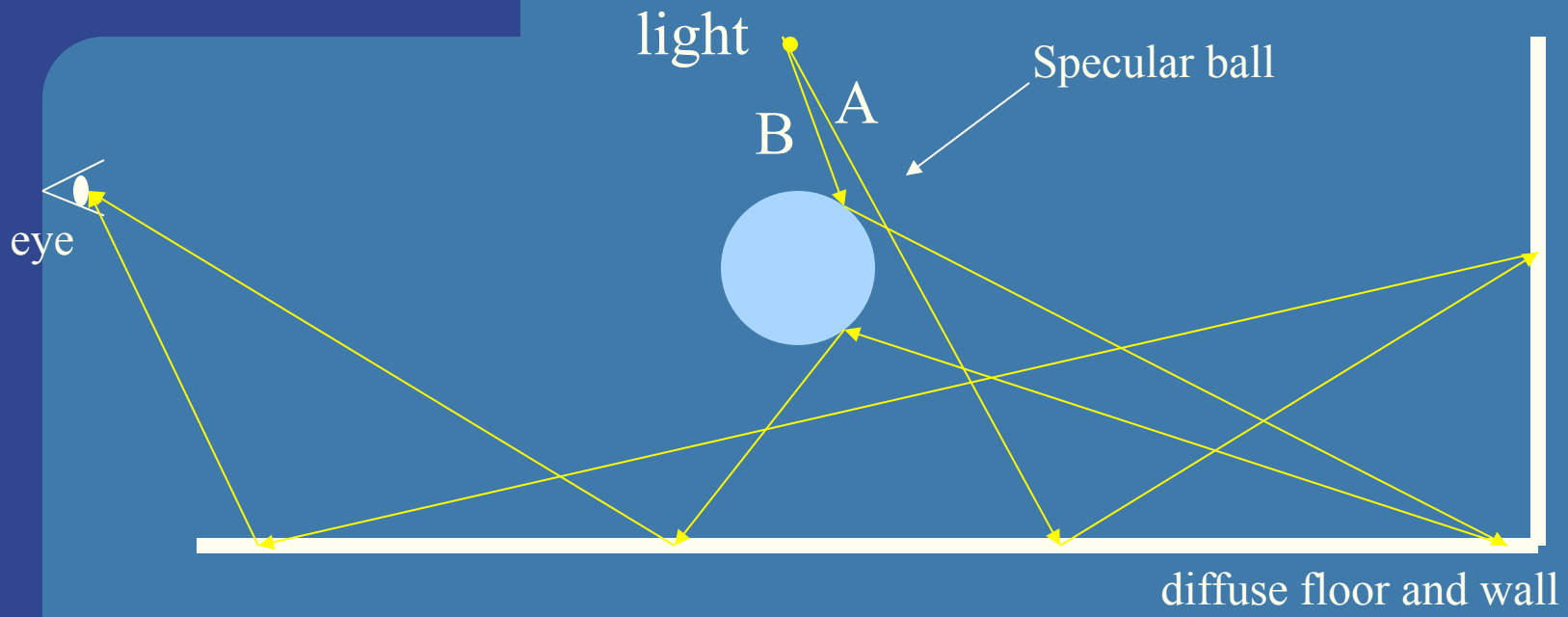
- The goal: follow all photons through a scene, in order to render images with all light paths
- This will give incredibly realistic images
- This lecture will treat:
  - Background
  - Montecarlo ray tracing
  - Path tracing
  - Photon mapping
  - Final Gather
- Great book on global illumination and photon mapping:
  - 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 | 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?

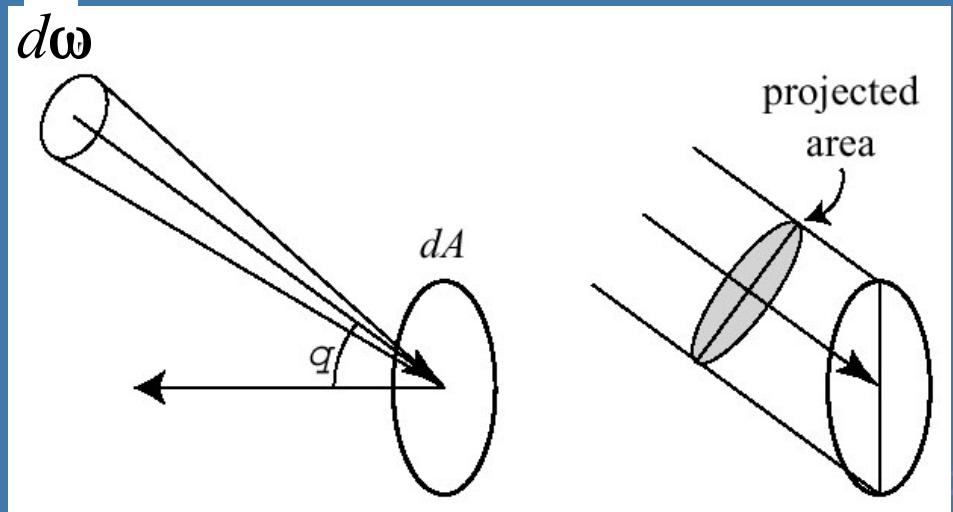
- The ultimate goal is to simulate all light paths: L(S|D|G)*E
- Using this notation, we can find what ray tracing can handle:
  - LDS*E | LS*E = LD?S*E
  - This is clearly not L(S|D|G)*E

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

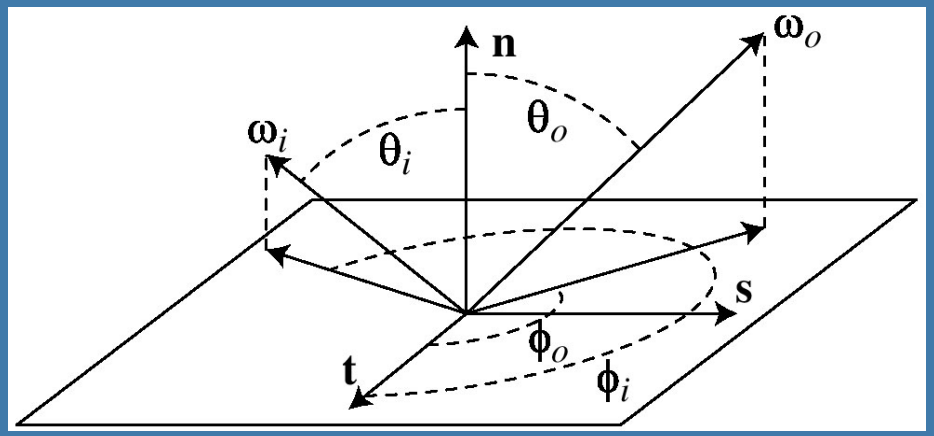
- Paper by Kajiya, 1986 (see course website).
- Is the basis for all rendering, but especially for global illumination algorithms
- $L_o(\mathbf{x},\omega)=L_e(\mathbf{x},\omega)+L_r(\mathbf{x},\omega)$ (slightly different terminology than Kajiya)
  - outgoing=emitted+reflected radiance
  - $\mathbf{x}$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



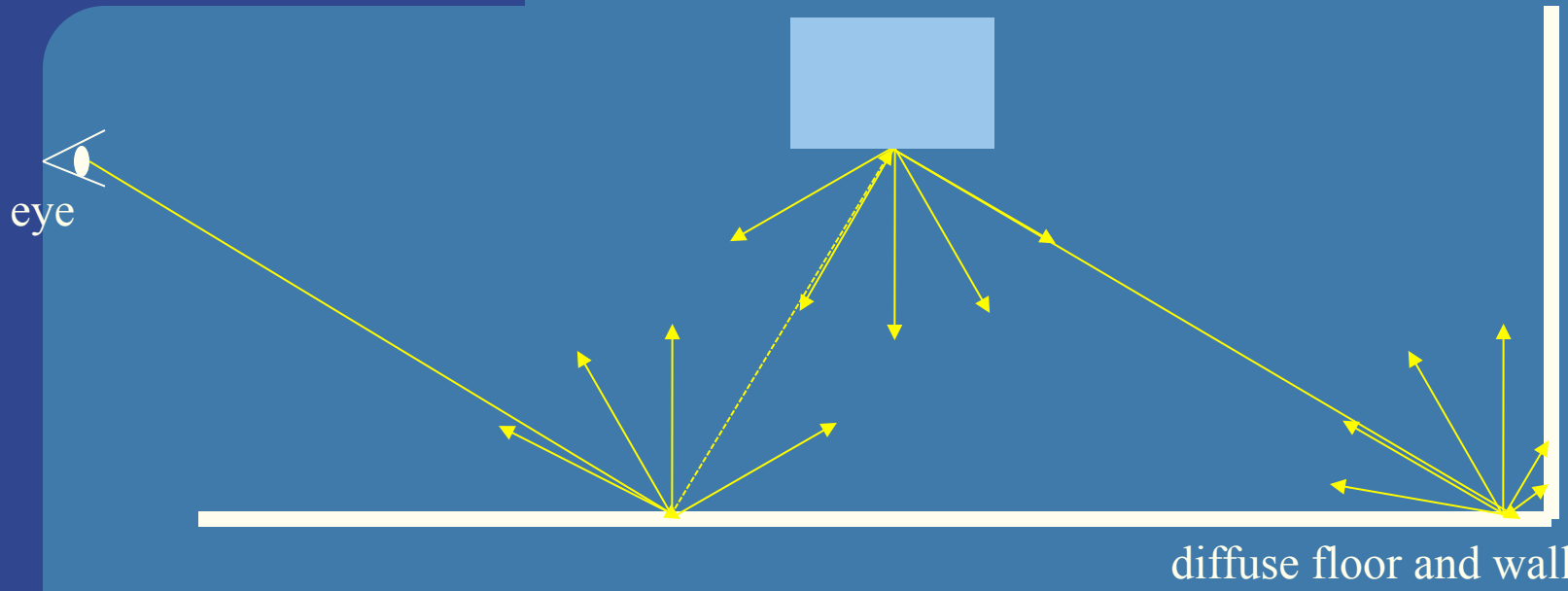
Many GI algorithms is built on Monte Carlo Integration

- 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!!

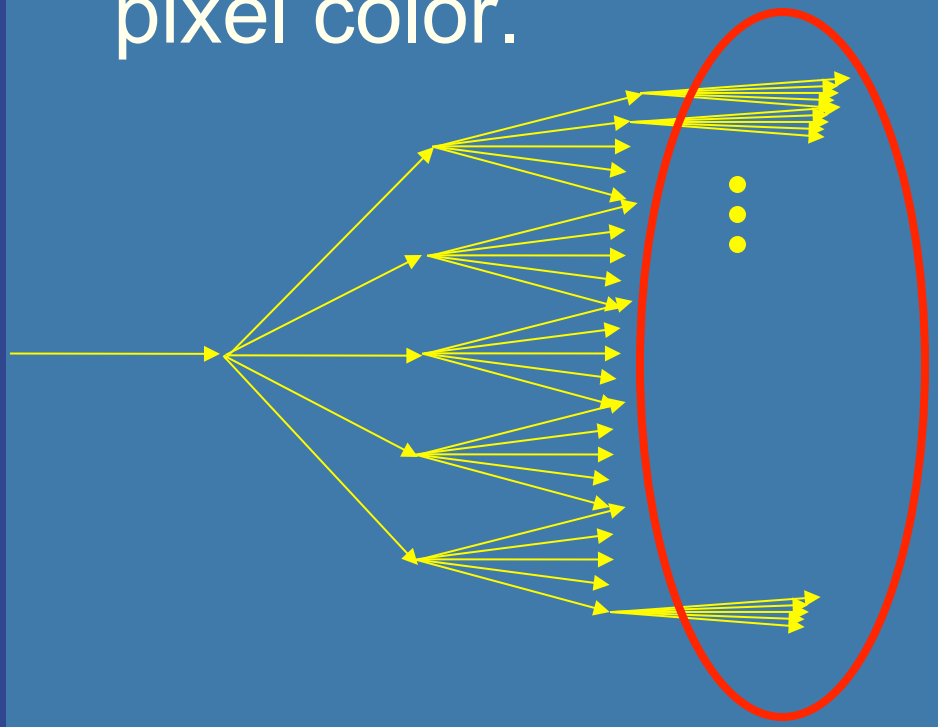
Monte Carlo Ray Tracing



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

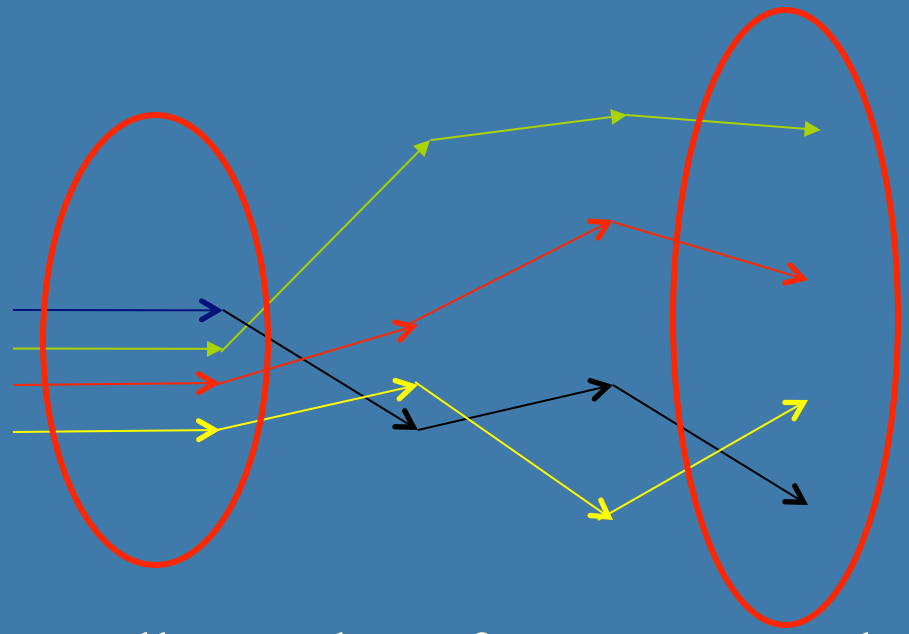
Monte Carlo Ray Tracing

- 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

- 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.



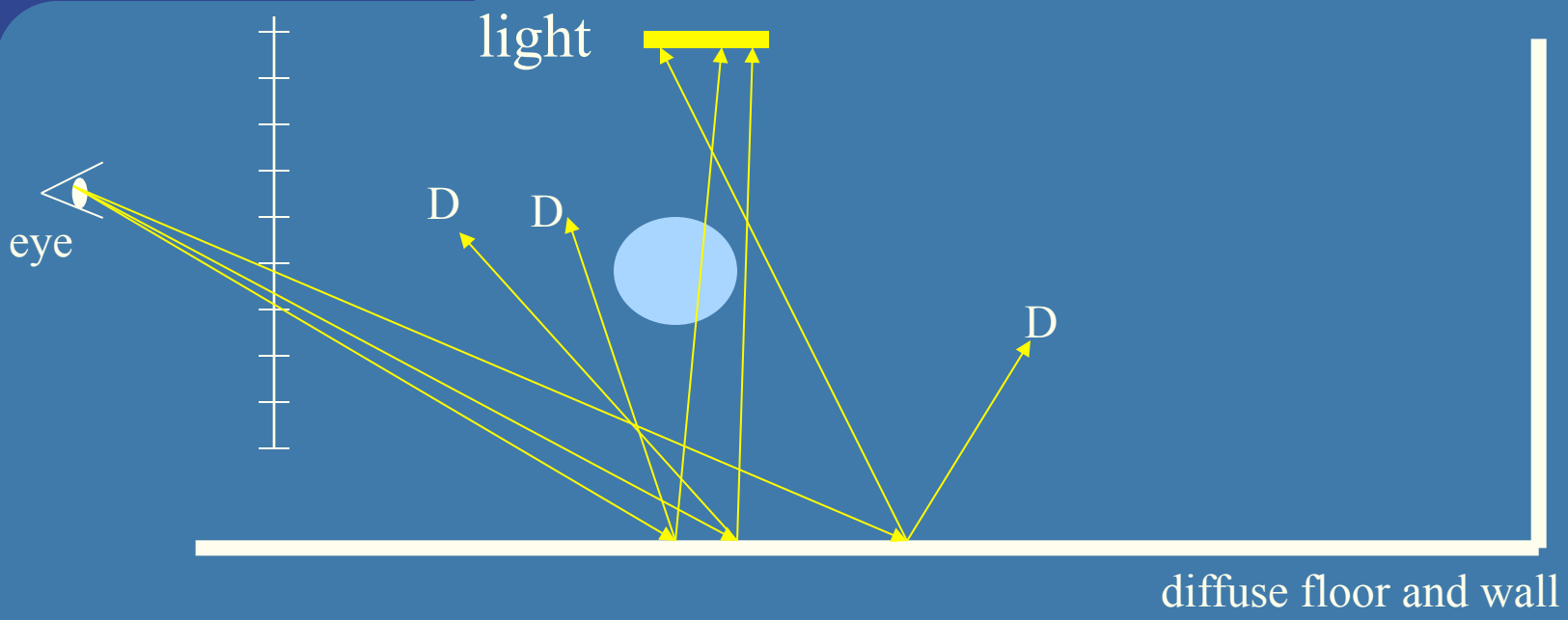
Equally number of rays are traced at each level

Path tracing: One solution to GI

See section 6 in Kajiya's paper

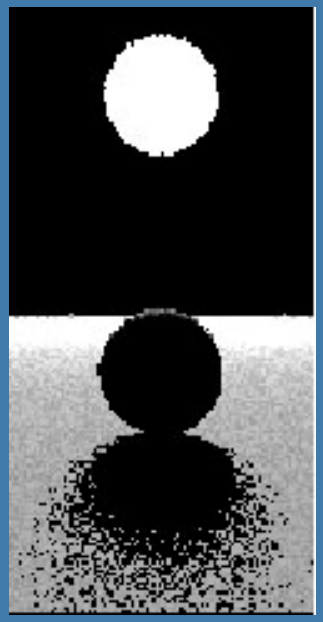
- Uses Monte Carlo sampling to solve integration: just shoot many random rays over the integral domain
- Example: ray hits a diffuse surface
  - Shoot many rays distributed randomly over the possible reflection directions
  - Gives color bleeding effects (and the ambient part of lighting)
- Algorithm: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

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

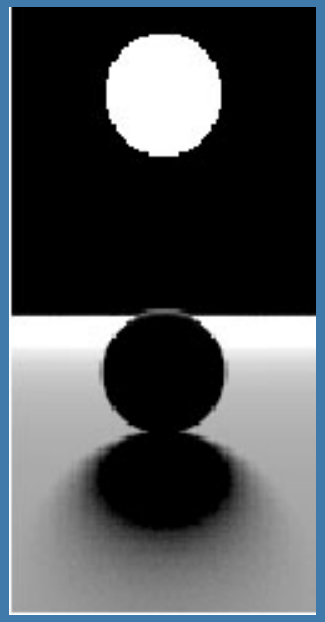


- Example: Three rays for one pixel
- All three rays hits diffuse floor
- Pick ***one*** random position on light source
- Sends **one** random diffuse ray (D's above)
- Reason for not sending more rays: their contribution gets smaller and smaller, after more bounces

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

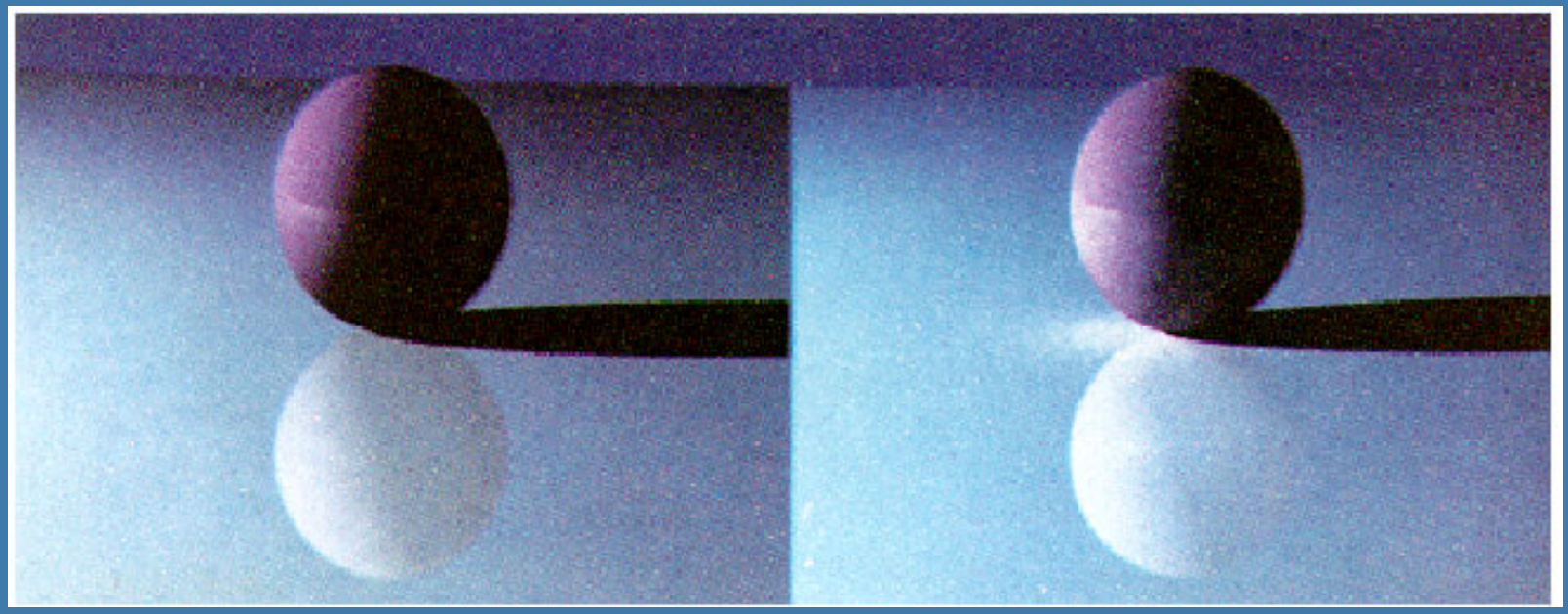
Images courtesy of Peter Shirley

Diffuse and Specular surfaces in path tracing

- Assume $k_{diff}+k_{spec}<=1$
  - Comes from that energy cannot be created, but can be absorbed
  - k_{diff} is sum of diffuse color, (R+G+B)/3, etc.
- 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

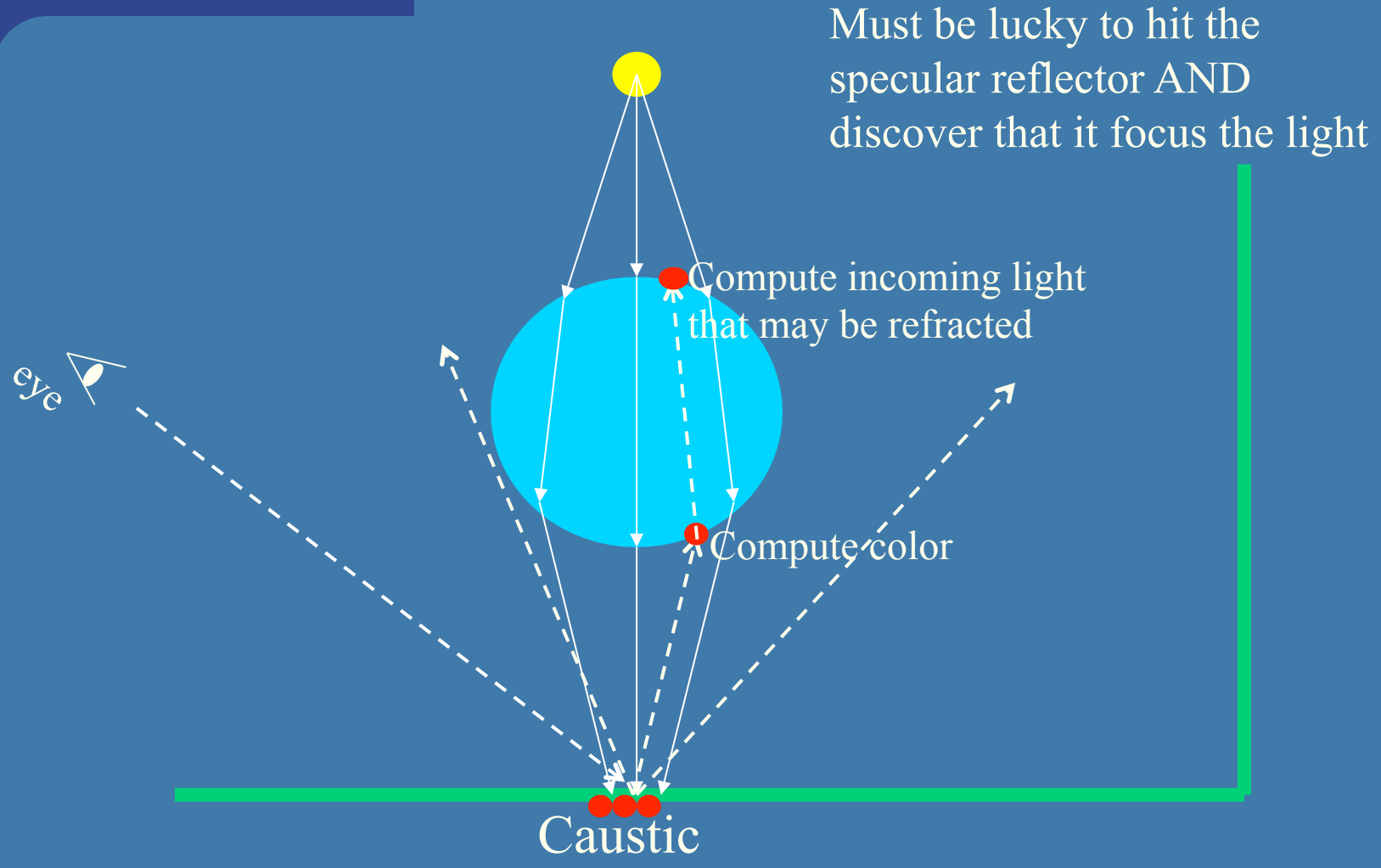


- Note how the right sphere reflects light, and so the ground under the sphere is brighter

What is Caustics?

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

Reason why forward ray tracing fails to capture caustics well

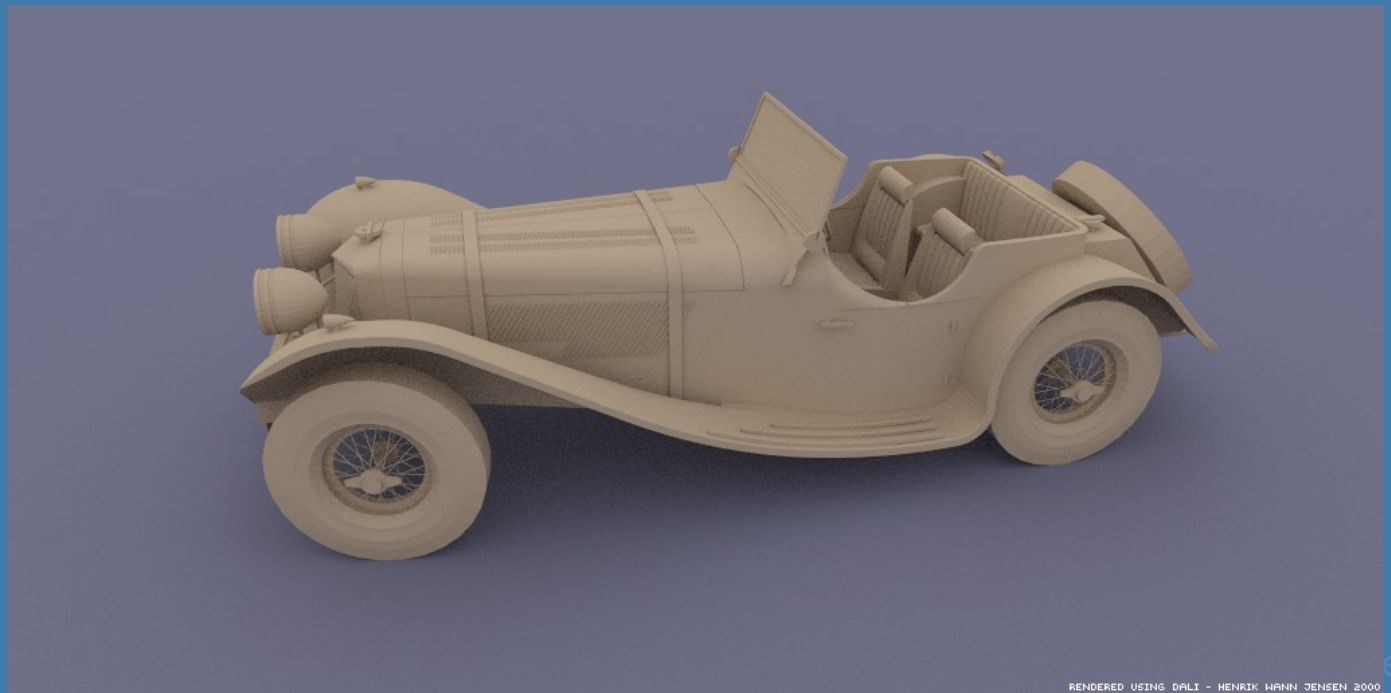


Path tracing implemented using `trace()` and `shade()` framework

- In **`RayTraceImage()`**:
  - Shoot n rays per pixel
  - New random position inside pixel for each ray
  - Possibly also a random time and lens position
- In **`trace()`**:
  - Nothing
- In **`shade()`**:
  - For area light sources: pick random position on light source
  - When calling trace(), choose one random ray direction (use russian roulette)

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

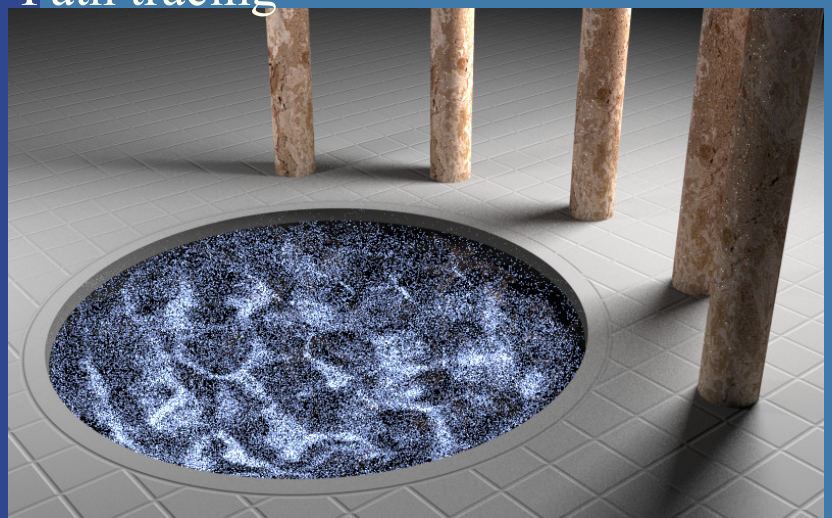


Extensions to path tracing

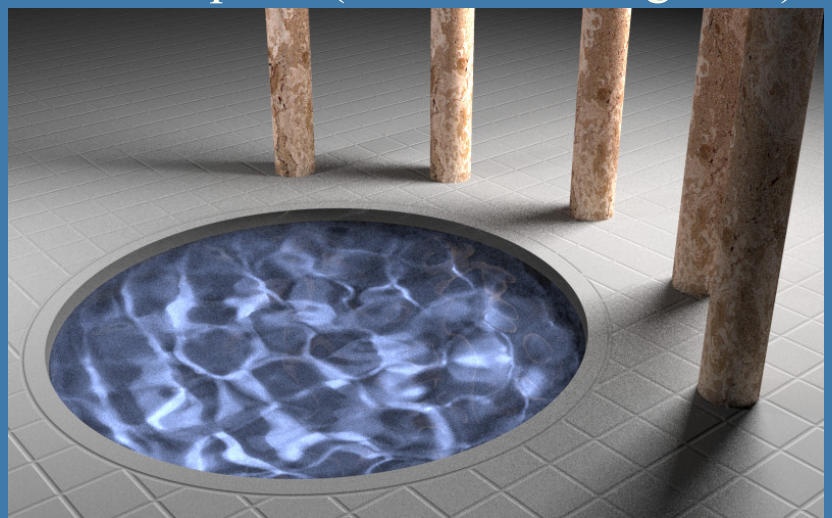
- Bidirectional path tracing
  - Developed in1993-1994
  - Sends light paths, both from eye and from the light
  - Faster, but still noisy images.
- Metropolis light transport
  - 1997
  - Ray distribution is proportional to unknown function
  - Means that more rays will be sent where they are needed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

Images courtesy of Eric Veach

Path tracing



Metropolis (same rendering time)



Photon mapping
State-of-the-art in GI

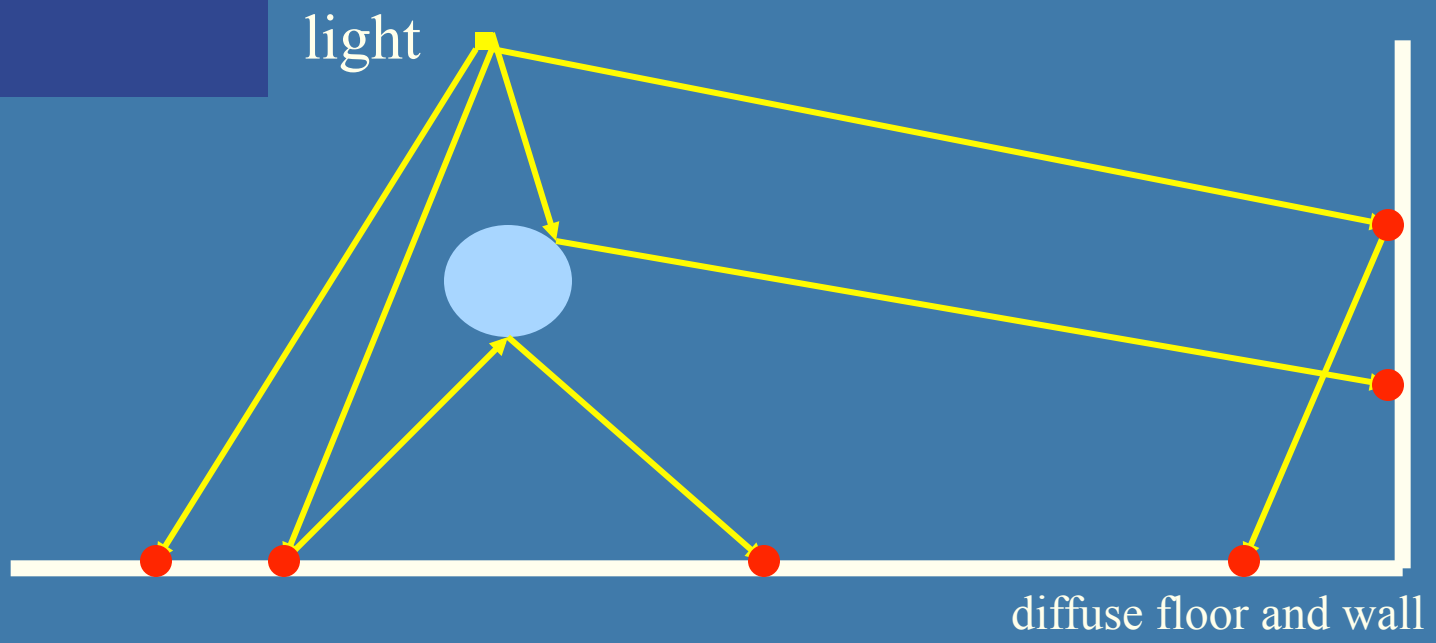
- Developed by Henrik Wann Jensen (started 1993)
- A clever two-pass algorithm:
  - 1: Shoot photons from light source, and let them bounce around in the scene, and store them where they land
  - 2: ”Ray tracing”-like pass from the eye, but gather the photons from the previous pass
- Advantages:
  -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

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 Russion roulette to find out whether the photon is absorbed or reflected
  - If reflected, then shoot photon in new random direction

Photon tracing

This type of arrow is a stored photon



- Should not store photon at specular surfaces, because these effects are view dependent
  - only diffuse effect is view independent
- Some diffuse photons are absorbed, some are scattered further
- A photon = the incoming illumination at a point
- (Power of photon is decreased at bounces)

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, but 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

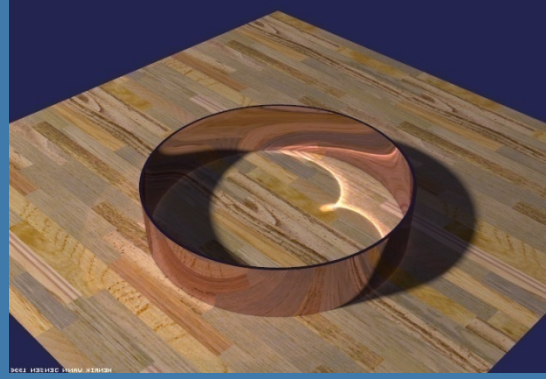
Bonus

Locate n closest photons

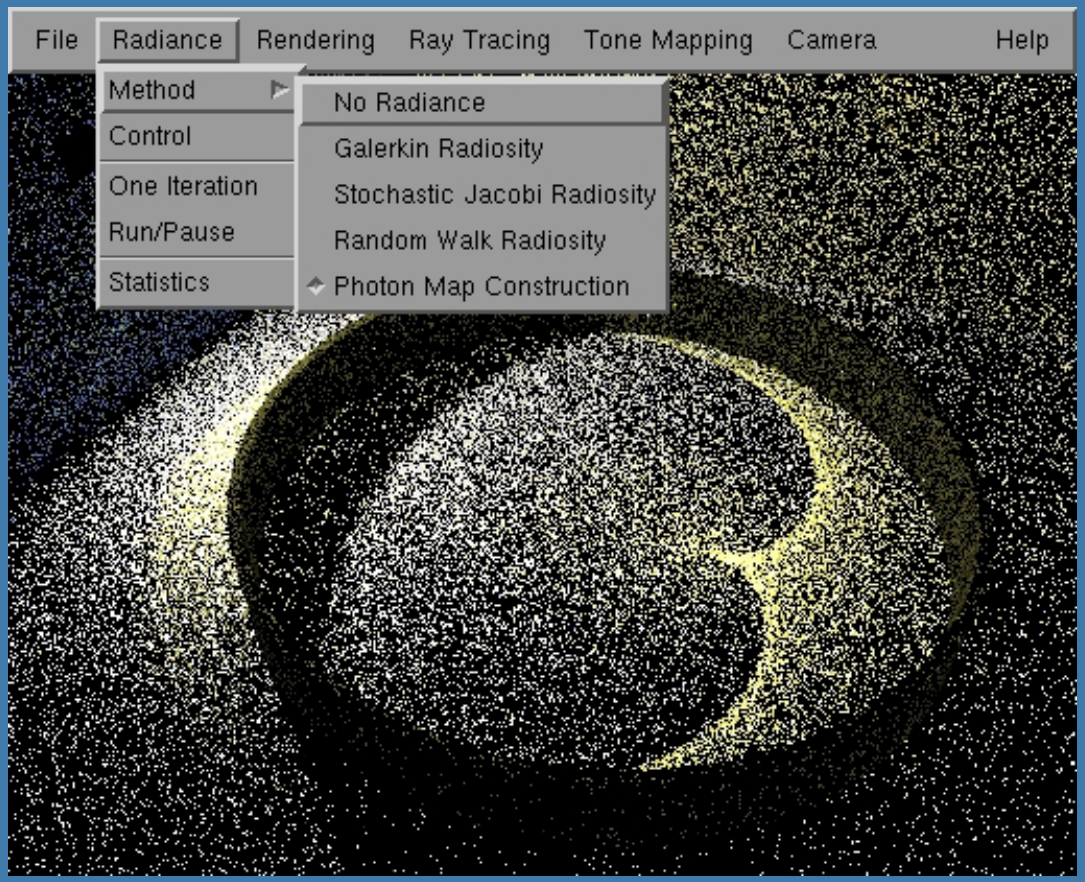
After Henrik Wann Jensen

```
// 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:

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

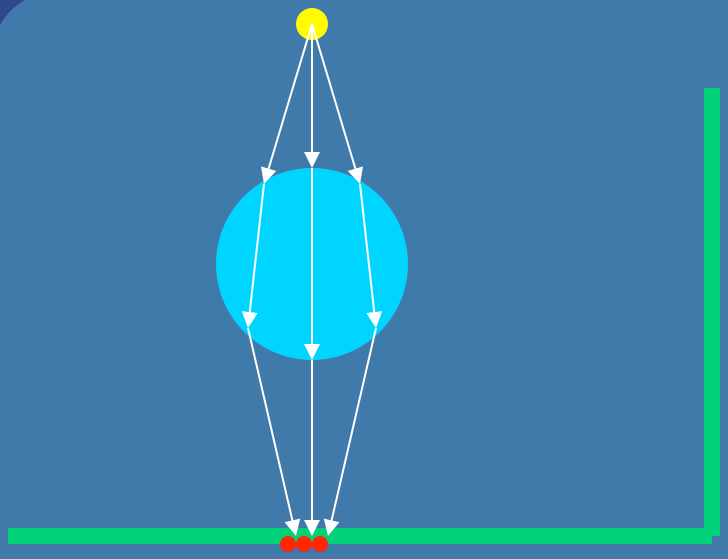
- L is radiance in $\mathbf{x}$ in the direction of $\mathbf{w}$
- r is radius of expanded sphere
- ω_p is the direction of the stored photon
- Φ_p is the stored power of the photon
- f_r is the BRDF

Two-pass algorithm

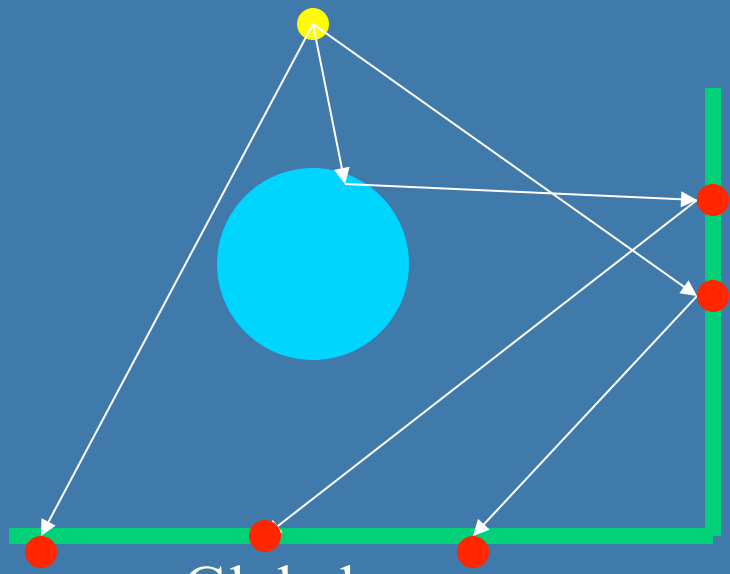
- Already said:
  - 1) Photon tracing, to build photon maps
  - 2) Rendering from the eye using photon maps
- Pass 1 (create photon maps):
  - Use two photon maps
  - A caustics photon map (for caustics)
    - Stores photons that are reflected or refracted via a specular/transparent surface to a diffuse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



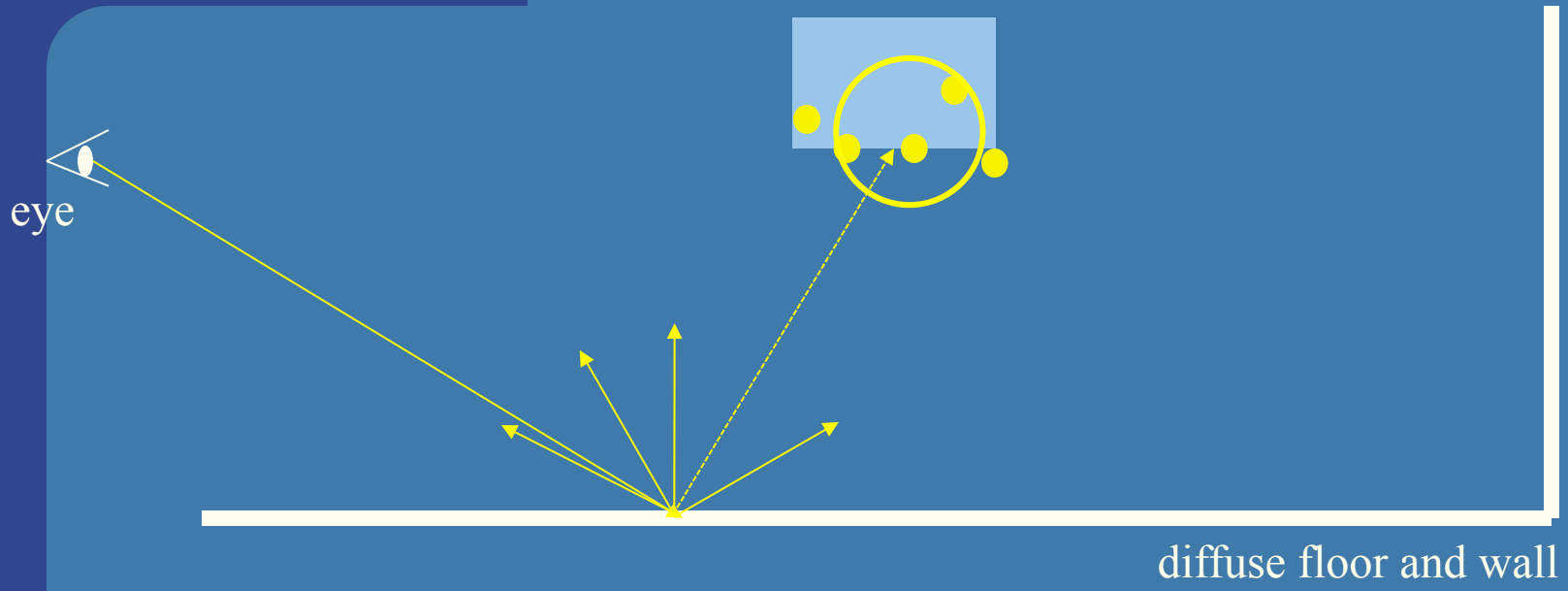
Global map

- 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 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...

A modification for indirect Illumination – Final Gather

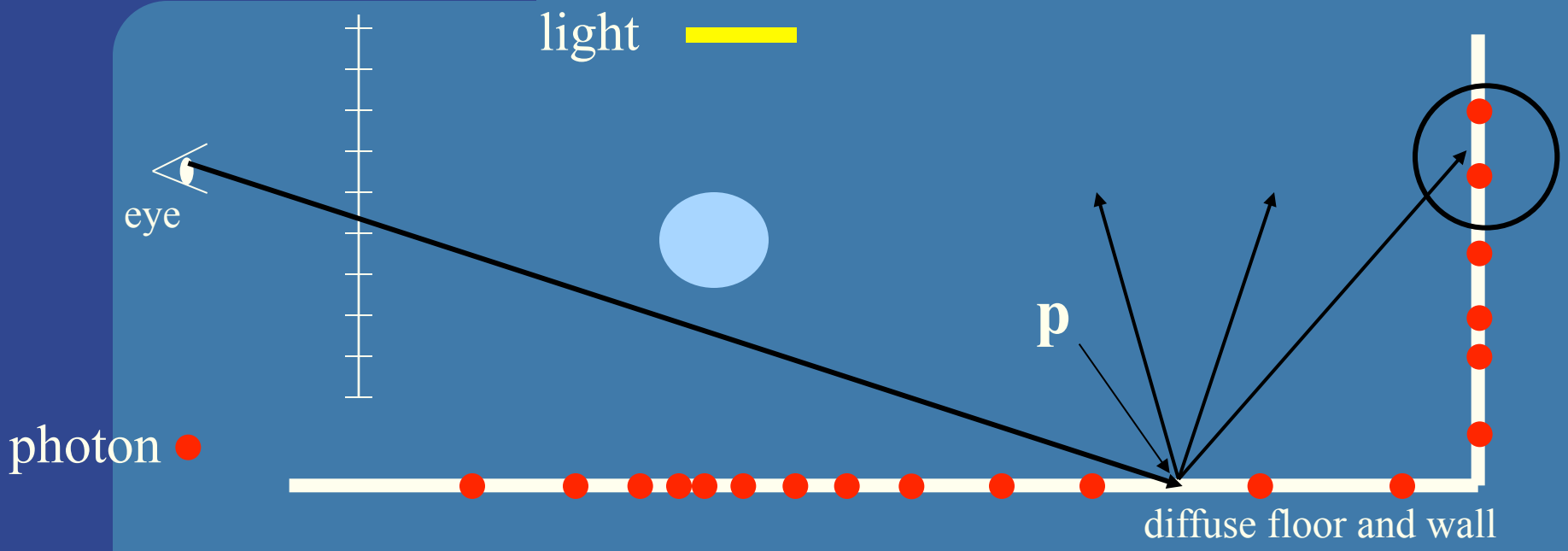


- Too noicy to use the global map for direct rays
- Use global map for indirect rays (shoot 100-1000 indirect rays per pixel)

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 scenes, 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 globillum photon map if available 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 a photon map lookup) is very time-consuming so it is only done when necessary - in most cases, interpolation and extrapolation from previous nearby final gatherings is sufficient.

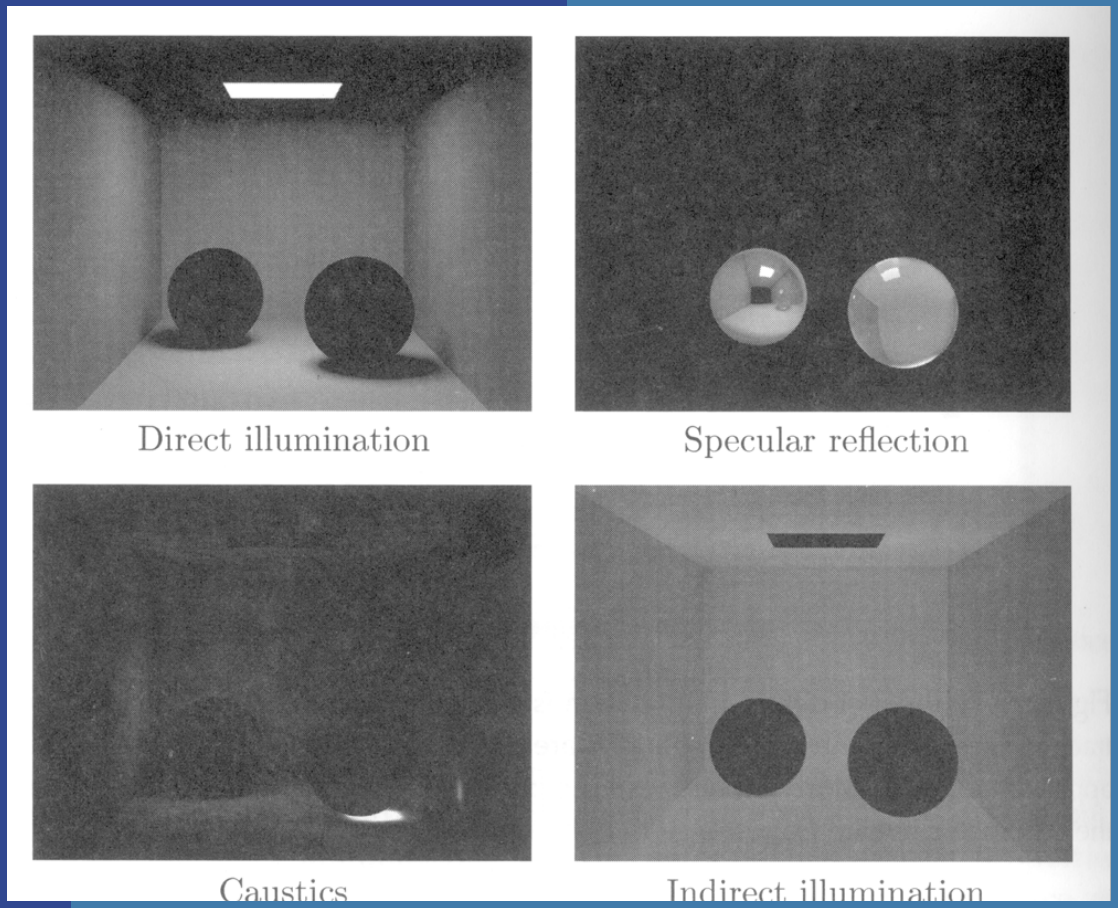
Indirect illumination: Use the global photon map



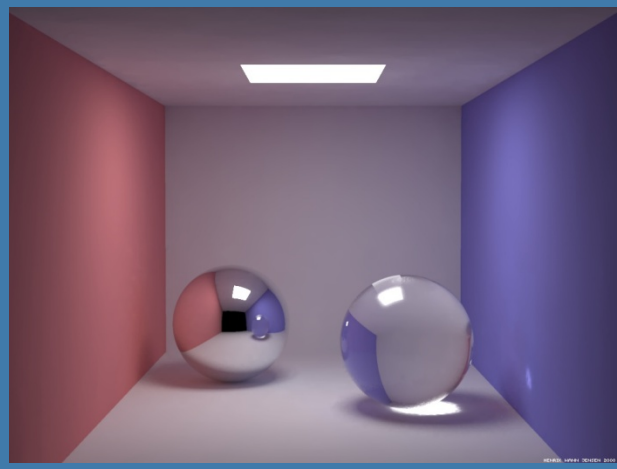
- To evaluate indirect illumination at point **p**:
  - Send several random rays out from **p**, and grow spheres at contacts
  - May need several hundreds of rays to get good results.

Image scanned from Henrik Wann Jensen's book

Images of the four components

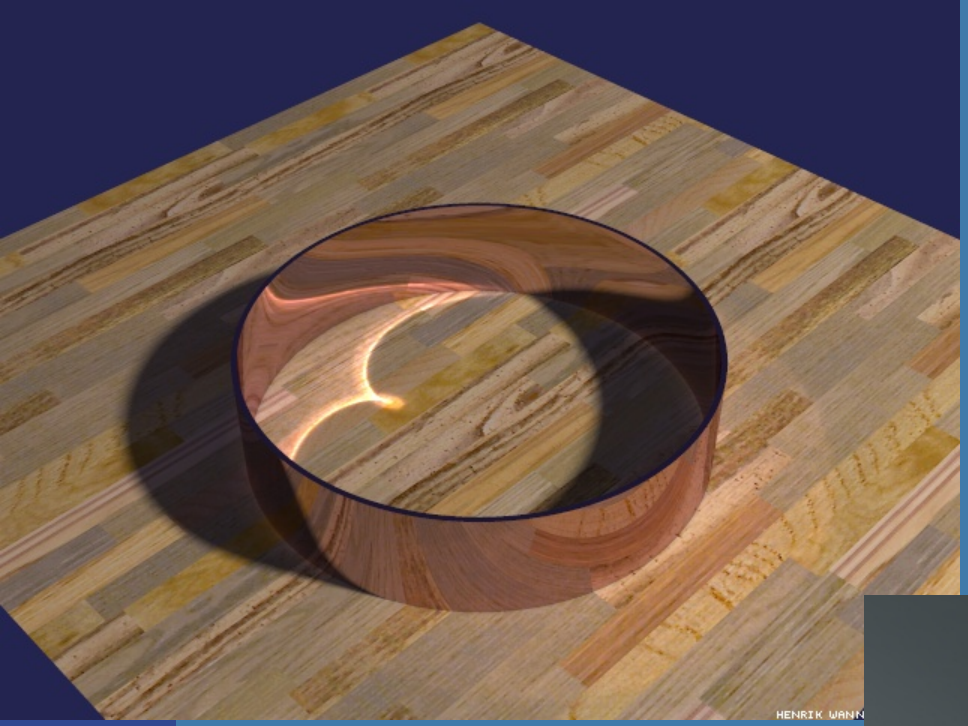


=

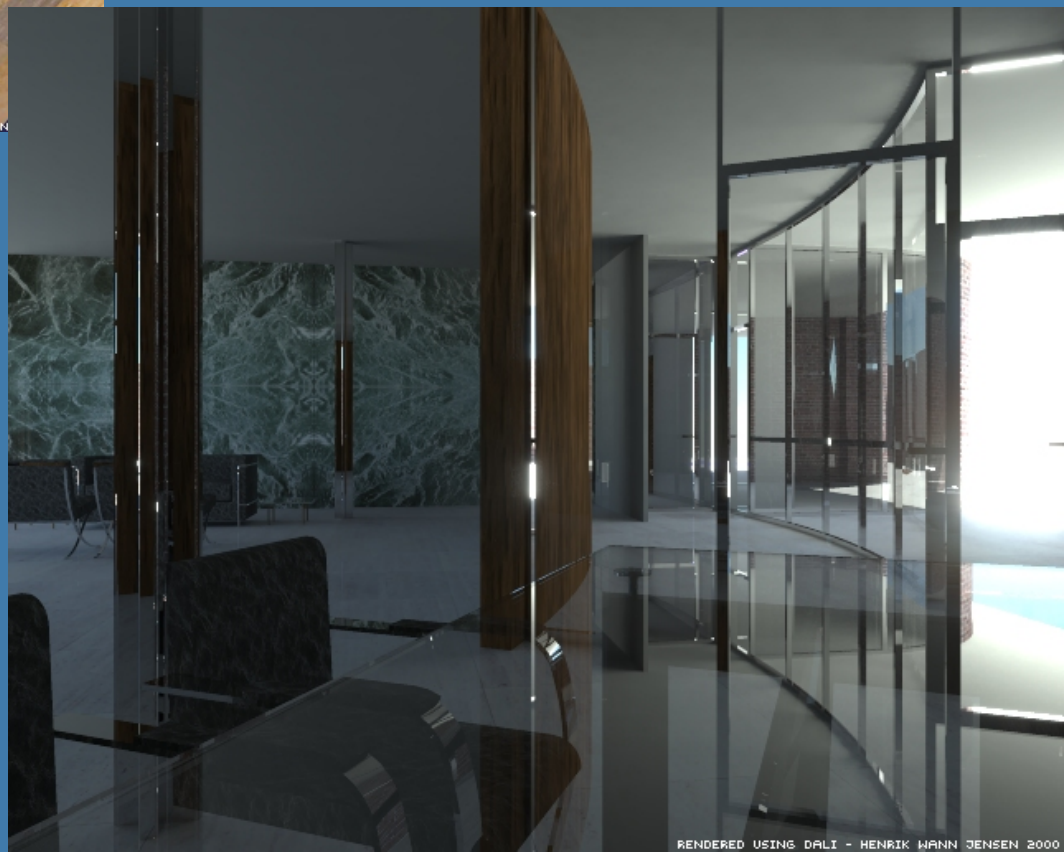


- These together solves the entire rendering equation!

Standard photon mapping

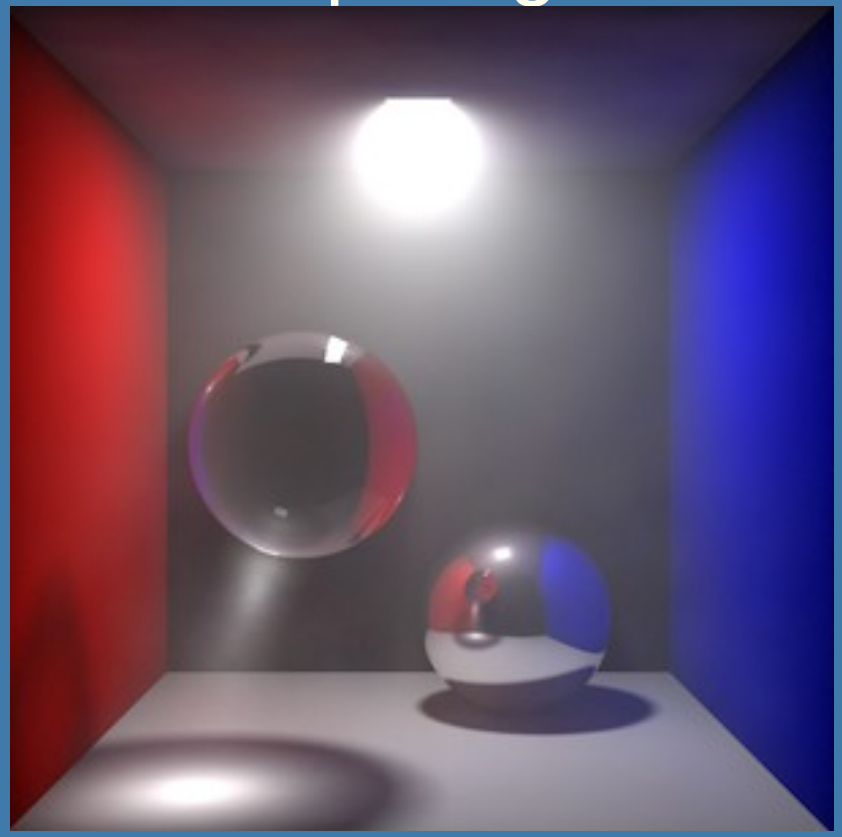


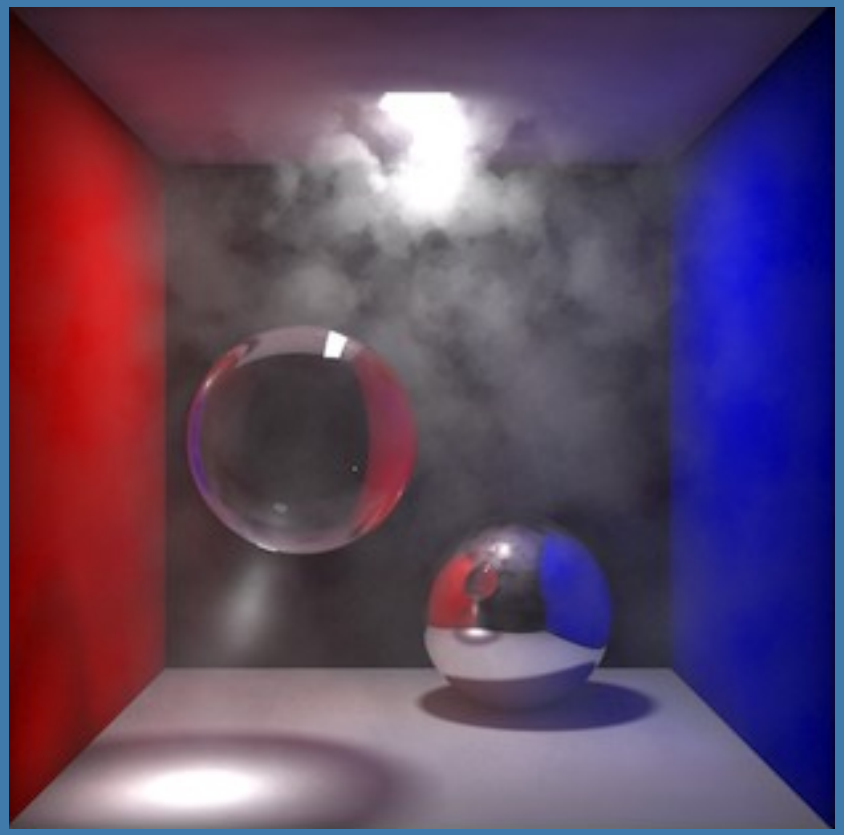
Caustics: concentrated reflected or refracted light



Extensions to photon mapping

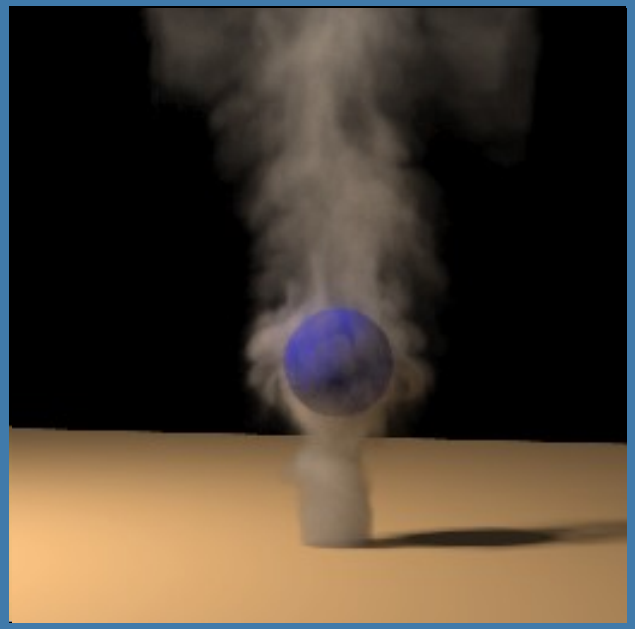
- Participating media





Another one on participating media

Smoke and photon mapping



Press for a movie

Much more details to photon mapping…

- "Global Illumination using Photon Maps", Henrik Wann Jensen, In "Rendering Techniques '96". Eds. X. Pueyo and P. Schröder. Springer-Verlag, pages 21-30, 1996
- "A Practical Model for Subsurface Light Transport", Henrik Wann Jensen, Steve Marschner, Marc Levoy, and Pat Hanrahan,Proceedings of SIGGRAPH'2001
- "Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps", Henrik Wann Jensen and Per H. Christensen, Proceedings of SIGGRAPH'98, pages 311-320, July 1998

In particular

- 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)
- If you want a professional renderer:
  - Read all the papers about photon mapping or the book
  - Implement!

What you need to know

- In schedule:
  Read globillumclarification.pdf
  - Montecarlo Ray Tracing
  - Path Tracing
  - Photon Mapping
- Final Gather

THE END