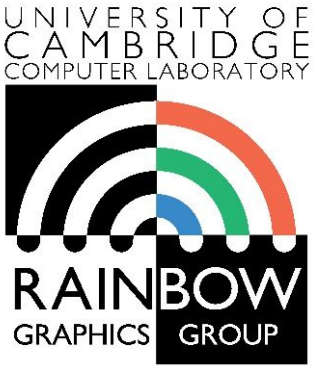


Advanced Graphics & Image Processing

Global Illumination

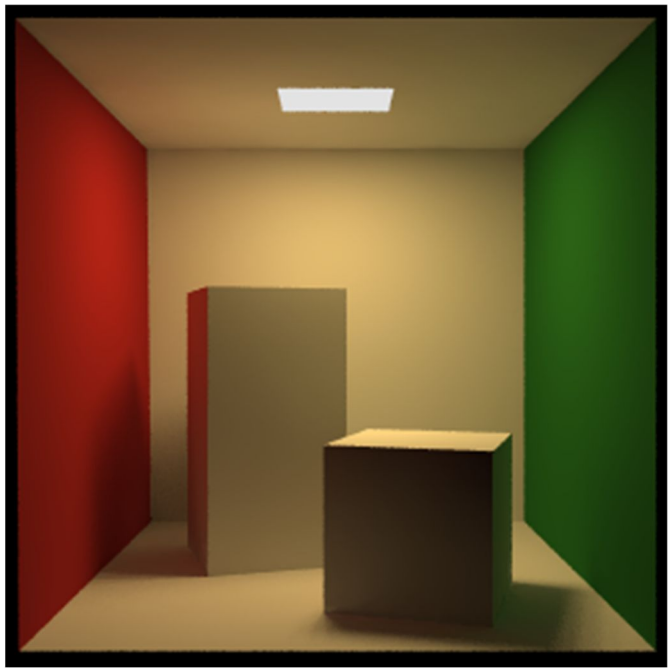
Rafał Mantiuk

Computer Laboratory, University of Cambridge

What's wrong with recursive raytracing?

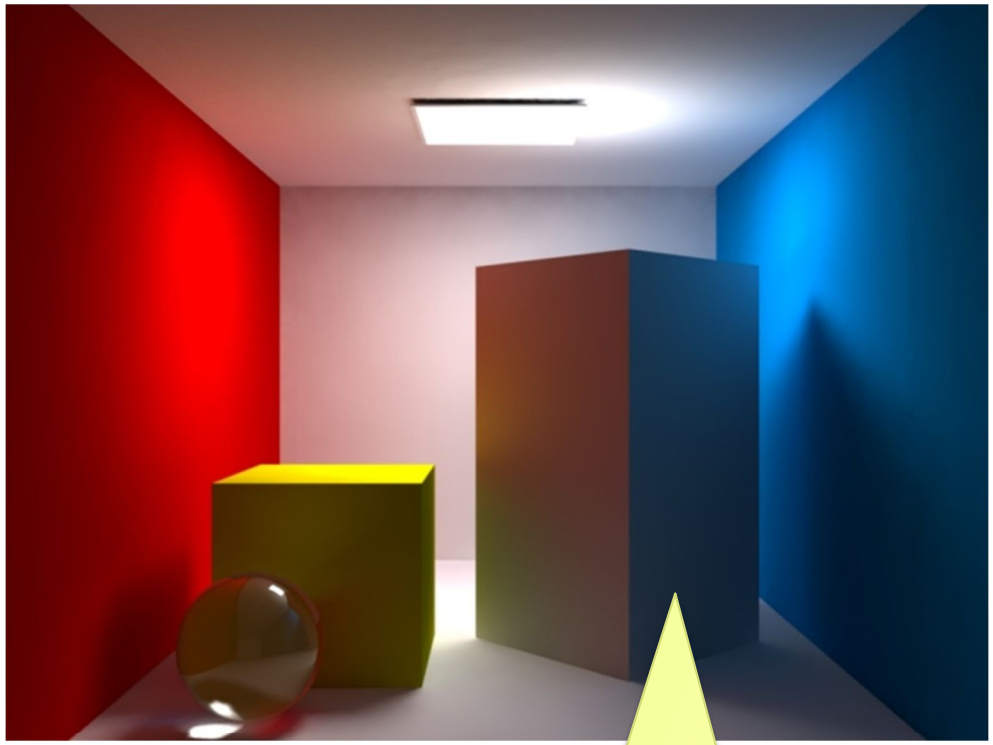
- Soft shadows are expensive
- Shadows of transparent objects require further coding or hacks
- Lighting off reflective objects follows different shadow rules from normal lighting
- Hard to implement diffuse reflection (color bleeding, such as in the Cornell Box—notice how the sides of the inner cubes are shaded red and green)

- Fundamentally, the ambient term is a hack and the diffuse term is only one step in what should be a recursive, self-reinforcing series.



The *Cornell Box* is a test for rendering Software, developed at Cornell University in 1984 by Don Greenberg. An actual box is built and photographed; an identical scene is then rendered in software and the two images are compared.

Global illumination examples



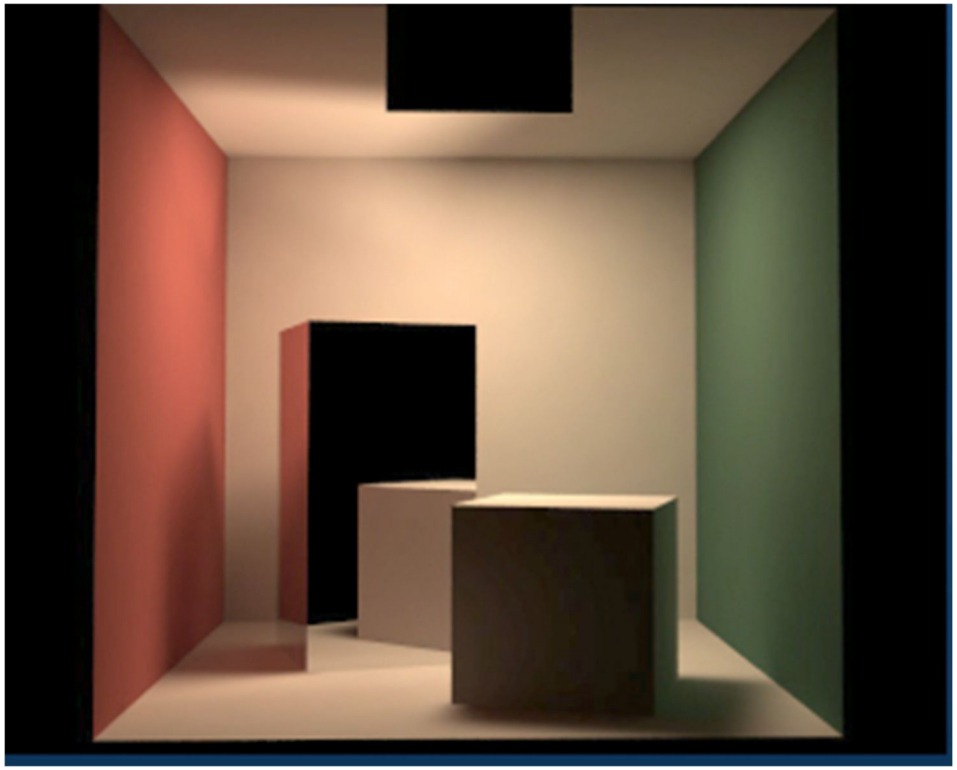
This box is white!

Global Illumination in real-time graphics

Pre-GI

Post-GI

Cornell Box: a rendering or photograph?



Rendering

Photograph

Rendering equation (revisited)

- Most rendering methods require solving an (approximation) of the rendering equation:

$$L_r(\boldsymbol{\omega_r}) = \int_{\Omega} \rho(\boldsymbol{\omega_i}, \boldsymbol{\omega_r}) L_i(\omega_i) cos\theta_i d\boldsymbol{\omega_i}$$

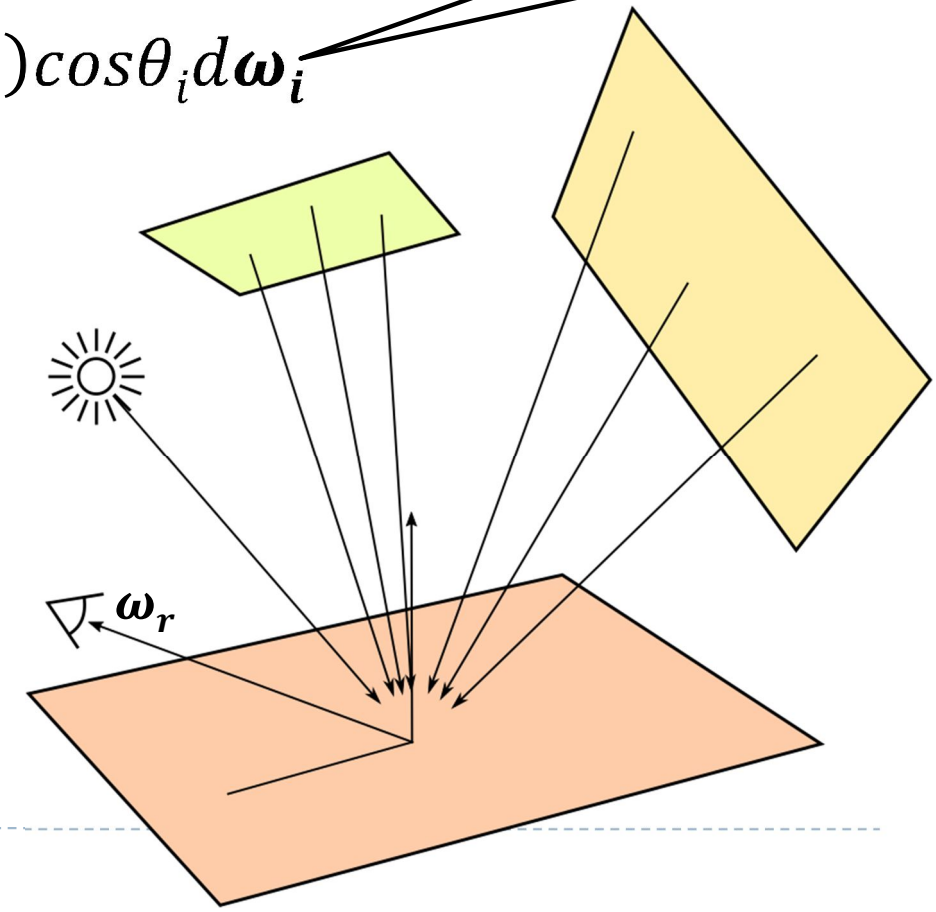
Incident light

Integral over the hemisphere of incident light

Reflected light

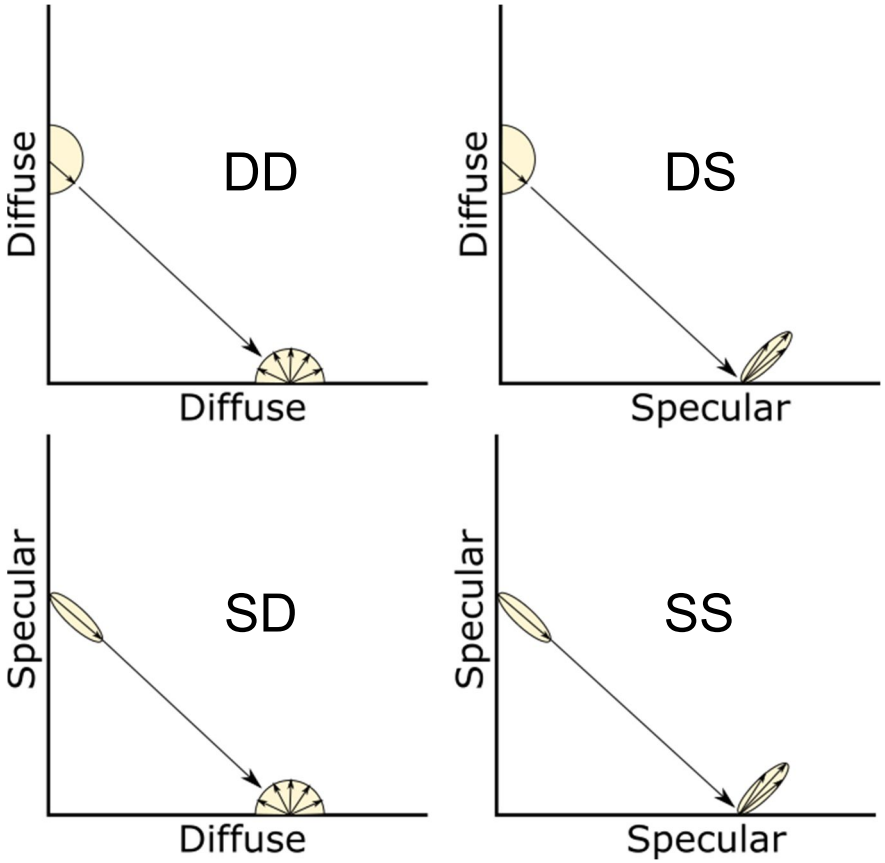
BRDF

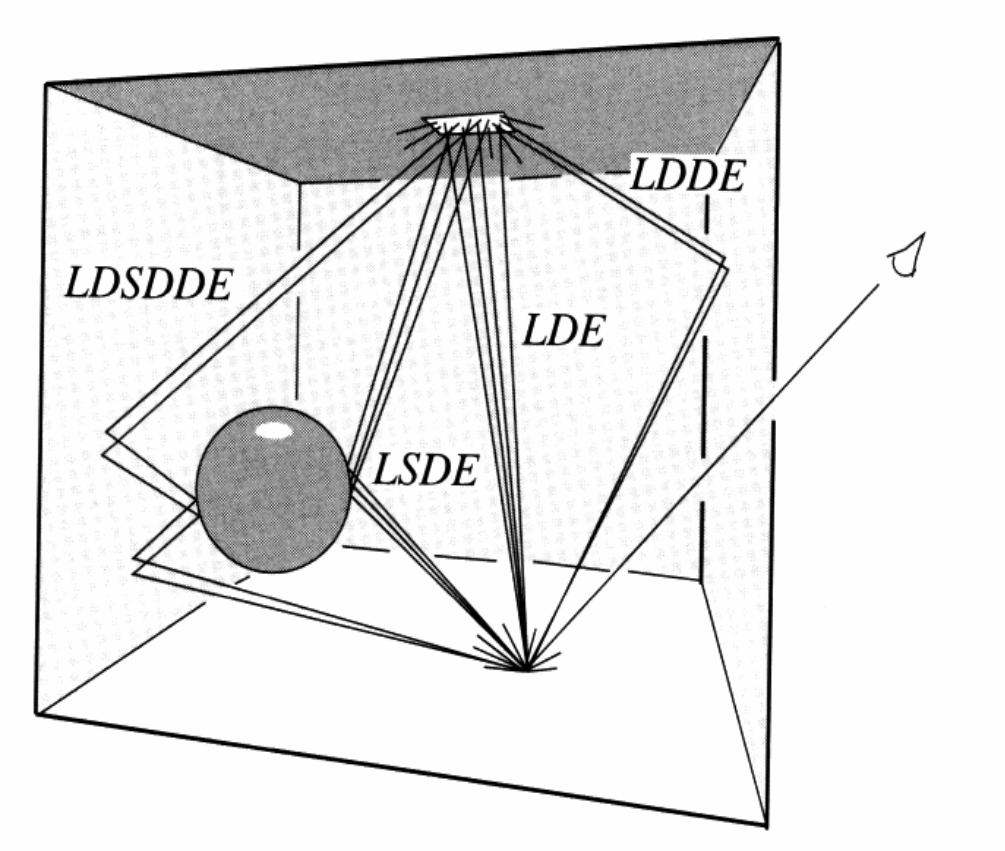
$$\boldsymbol{\omega_i} = [\phi_i, \theta_i]$$



- The solution is trivial for point light sources
- Much harder to estimate the contribution of other surfaces

Light transport





Shadows, refraction and caustics

- Problem: shadow ray strikes transparent, refractive object.
  - Refracted shadow ray will now miss the light.
  - This destroys the validity of the boolean shadow test.
- Problem: light passing through a refractive object will sometimes form *caustics* (right), artifacts where the envelope of a collection of rays falling on the surface is bright enough to be visible.

This is a photo of a real pepper-shaker.
Note the caustics to the left of the shaker, in and outside of its shadow.
Photo credit: Jan Zankowski

Shadows, refraction and caustics

- Solutions for shadows of transparent objects:
  - Backwards ray tracing (Arvo)
    - *Very* computationally heavy
    - Improved by stencil mapping (Shenya et al)
  - Shadow attenuation (Pierce)
    - Low refraction, no caustics

- More general solution:
  - *Path tracing*
  - *Photon mapping* (Jensen)→

Image from http://graphics.ucsd.edu/~henrik/
Generated with photon mapping

Path tracing

- Trace the rays from the camera (as in recursive ray tracing)
- [Russian roulette] When a surface is hit, either (randomly):
  - shoot another ray in the random direction sampled using the BRDF [importance sampling];
  - or terminate
- For each hit sample sample light sources (direct illumination) and other directions (indirect illumination)
- 40-1000s rays must be traced for each pixel
- The method converges to the exact solution of the rendering equation
  - But very slowly
  - Monte Carlo approach to solving the rendering equation

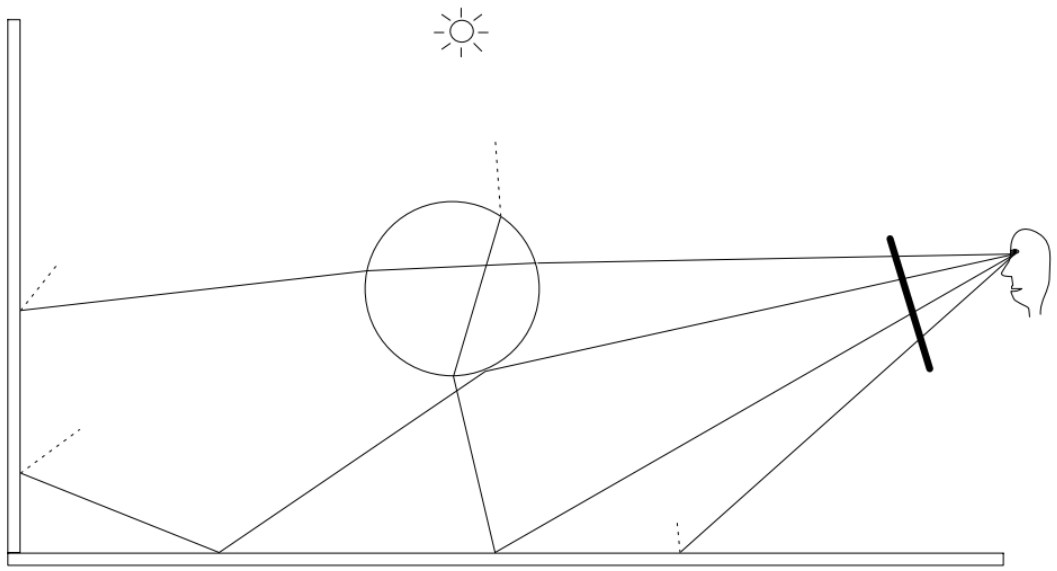


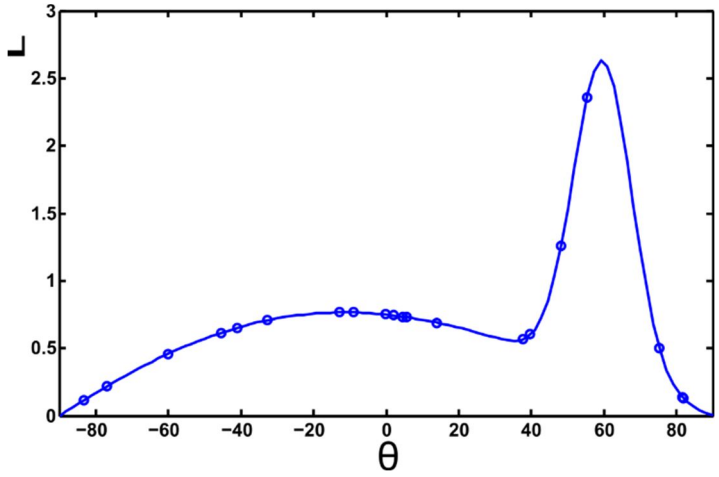
Image from A Practical Guide to Global Illumination using Photon Maps by Henrik Jensen (2000)

Monte-Carlo methods

- Path tracing estimates rendering equation by shooting rays in random directions (sampling) and averaging the contributions
- This is equivalent to estimating integral using Monte-Carlo sampling

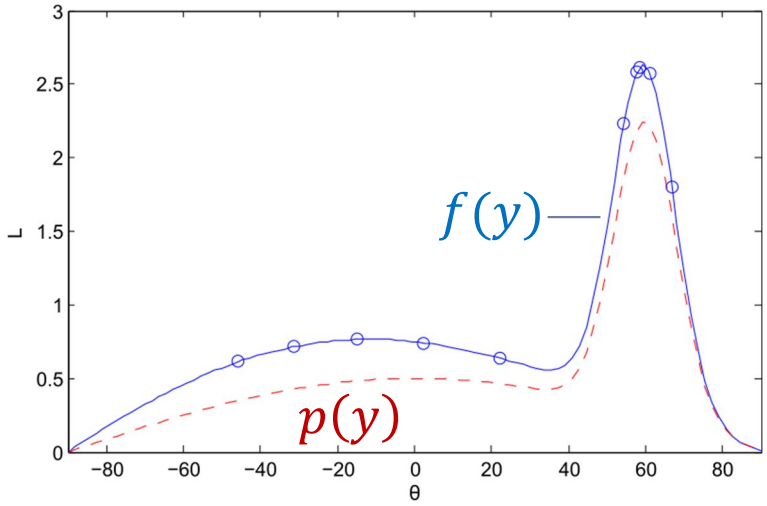
$$\int_a^b f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)(b-a)$$

where x_i are randomly drawn from Uniform(a,b)



Importance sampling

- Monte-Carlo sampling converges faster if ray directions with dominant contribution are sampled more often
- Dominant directions are unknown
  - But BRDF could be used as an estimate of importance
- When the sampling distribution is non-uniform, we need to use different estimator:

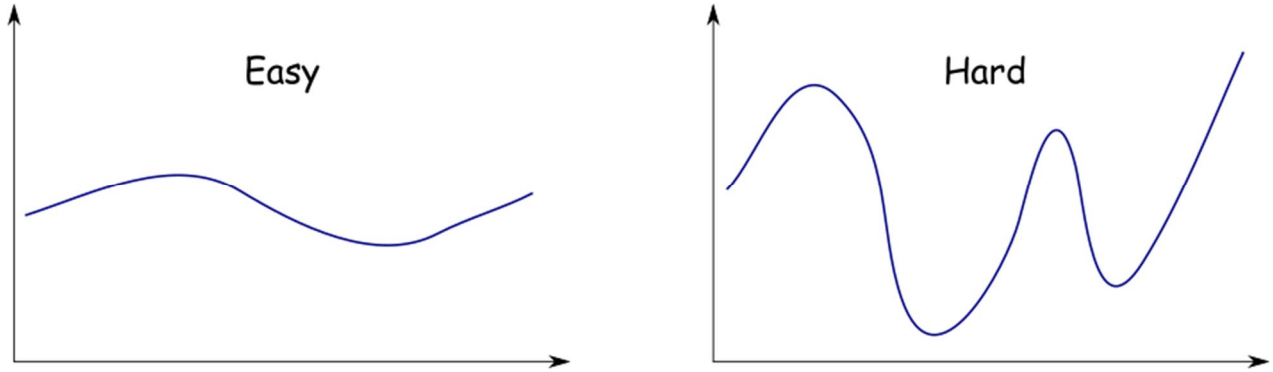


$$\int f(x)dx = \int \frac{f(x)}{p(x)} p(x)dx = E\left[\frac{f(y)}{p(y)}\right] \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(y_i)}{p(y_i)}$$

Where y is sampled from the distribution $p(y)$ - shown as red-dashed line in the plot

Importance sampling (intuition)

- Monte-carlo integration requires less samples when the integrated function varies less



- One way to make the integrated function vary less: divide by an approximation of the integrated function

$$\hat{f}(x) = \frac{f(x)}{p(x)}$$

Russian roulette

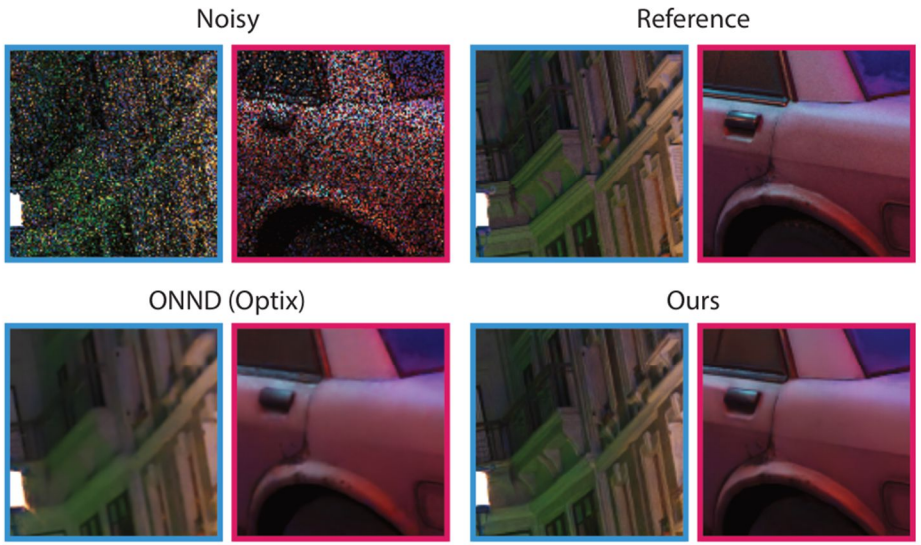
- Intuition: consecutive light bounces contribute less and less to the final color
  - But we cannot stop after N bounces as it will introduce bias (under-estimation)
- Instead: (after the first one or two bounces) terminate the current path with the probability q
  - Then, the estimator becomes

$$F' = \begin{cases} \frac{F}{1-q} & if\ \tau > q \\ 0 & otherwise \end{cases}, \quad \tau \sim Uniform(0,1), \quad F - \text{next bounce radiance}$$

  - Longer paths (with more vertices) become unlikely
  - Works the best if we know the contribution of F is likely to be small
  - If q is too large, we may end up with high variance and *fireflies*

Denoising for Monte-Carlo rendering

- Instead of tracing 1000s of rays, we can trace 4-8 rays per pixel and employ a denoiser
  - Modern denoisers are (convolutional) neural networks that take as input sample radiance, geometric and material features (G-buffer) and warped samples from the previous frame(s)



From: Balint et al. 2023 http://dx.doi.org/10.1145/3588432.3591562

Photon mapping

Photon mapping is the process of emitting photons into a scene and tracing their paths probabilistically to build a *photon map*, a data structure which describes the illumination of the scene independently of its geometry.

This data is then combined with ray tracing to compute the global illumination of the scene.

Image by Henrik Jensen (2000)

Photon mapping—algorithm (1/2)

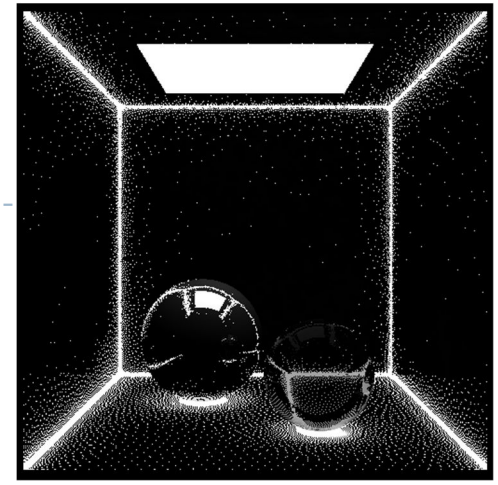


Image by Zack Waters

Photon mapping is a two-pass algorithm:

1. Photon scattering
   A. Photons are fired from each light source, scattered in randomly-chosen directions. The number of photons per light is a function of its surface area and brightness.
   B. Photons fire through the scene (re-use that raytracer). Where they strike a surface they are either absorbed, reflected or refracted.
   C. Wherever energy is absorbed, cache the location, direction and energy of the photon in the *photon map*. The photon map data structure must support fast insertion and fast nearest-neighbor lookup; a *kd-tree*[1] is often used.

Photon mapping—algorithm (2/2)

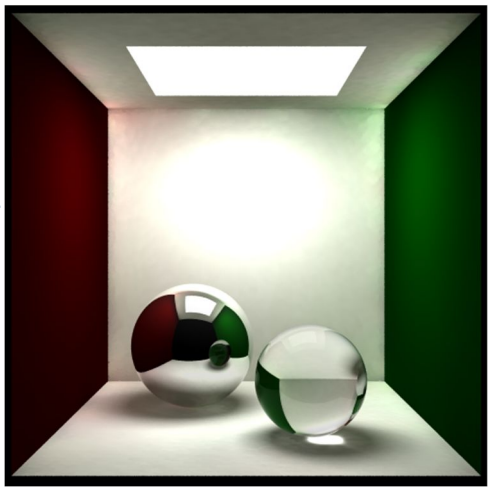


Image by Zack Waters

Photon mapping is a two-pass algorithm:

2. Rendering
    A. Ray trace the scene from the point of view of the camera.
    B. For each first contact point *P* use the ray tracer for specular but compute diffuse from the photon map.
    C. Compute radiant illumination by summing the contribution along the eye ray of all photons within a sphere of radius *r* of *P*.
    D. Caustics can be calculated directly here from the photon map. For accuracy, the caustic map is usually distinct from the radiance map.

Photon mapping is probabilistic

This method is a great example of *Monte Carlo integration*, in which a difficult integral (the lighting equation) is simulated by randomly sampling values from within the integral's domain until enough samples average out to about the right answer.

- This means that you're going to be firing *millions* of photons. Your data structure is going to have to be very space-efficient.

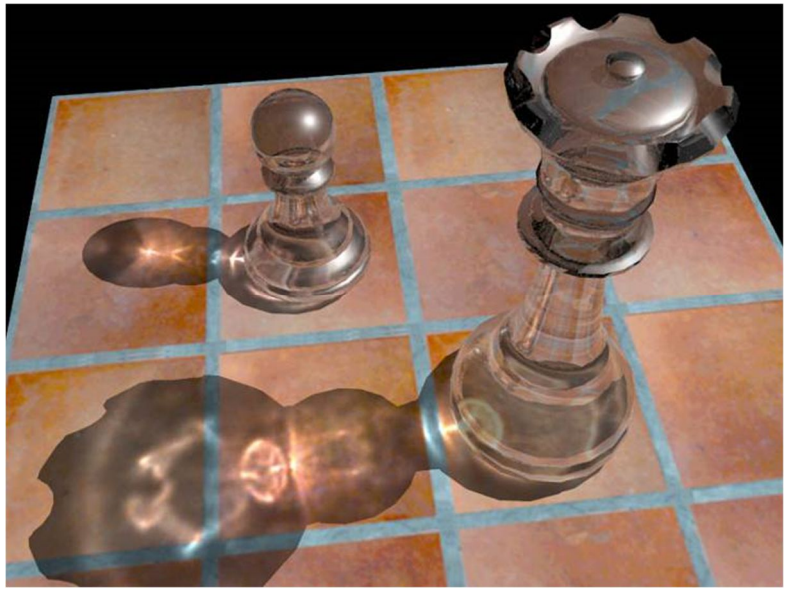
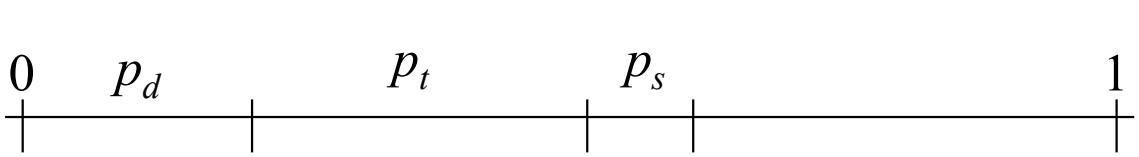


Image credit: http://www.okino.com/conv/imp_jt.htm

Photon mapping is probabilistic

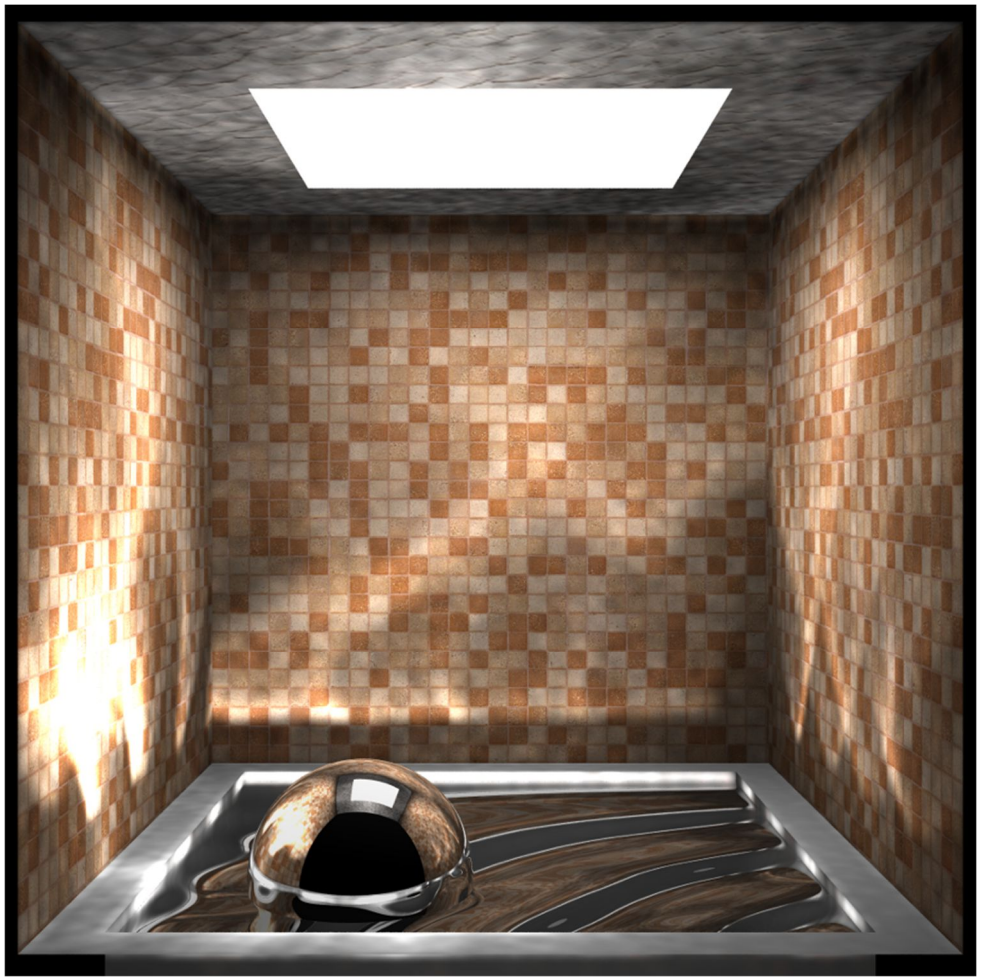
- Initial photon direction is random. Constrained by light shape, but random.
- What exactly happens each time a photon hits a solid also has a random component:
  - Based on the diffuse reflectance, specular reflectance and transparency of the surface, compute probabilities p_d, p_s and p_t where $(p_d+p_s+p_t)\leq 1$. This gives a probability map:



This surface would have minimal specular highlight.

  - Choose a random value $p \in [0,1]$. Where p falls in the probability map of the surface determines whether the photon is reflected, refracted or absorbed.

Photon mapping gallery

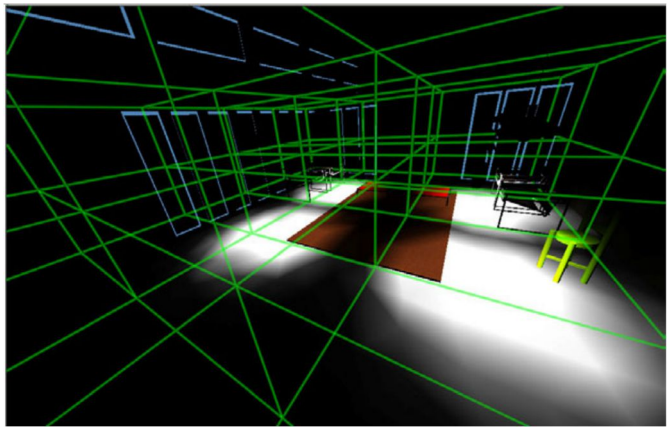


http://graphics.ucsd.edu/~henrik/images/global.html

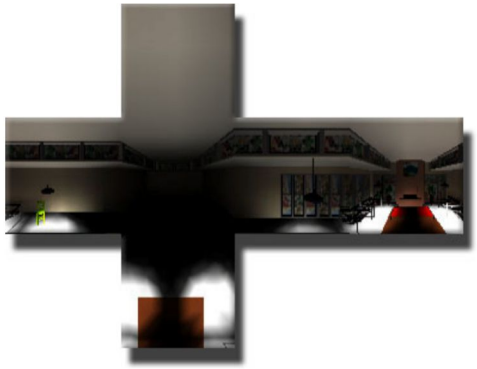
http://www.pbrt.org/gallery.php

http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/photon_mapping/PhotonMapping.html

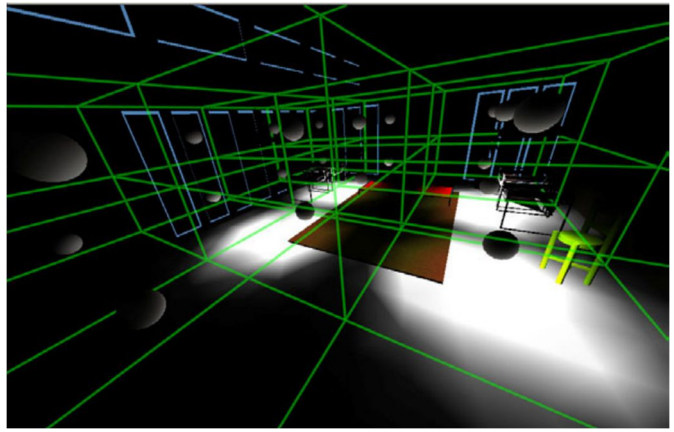
Real-time global illumination: irradiance probes (diffuse GI only)



Step 1: Create a voxel grid



Step 2: For each voxel centre, render a cube map (or sample with a ray-tracer). For the first bounce, render direct illumination only.

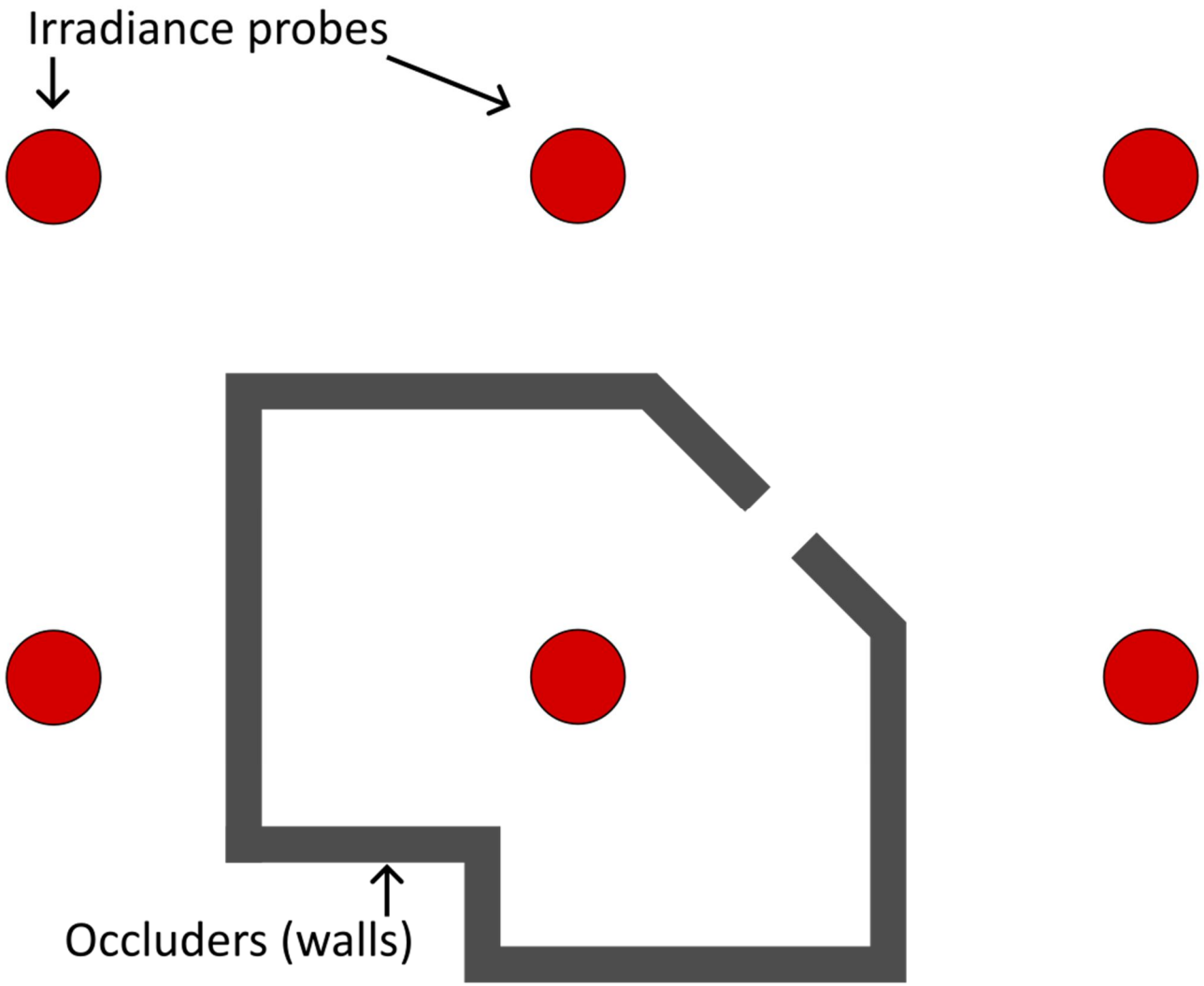


Step 3: Integrate incoming light over a hemisphere (compute irradiance probes)

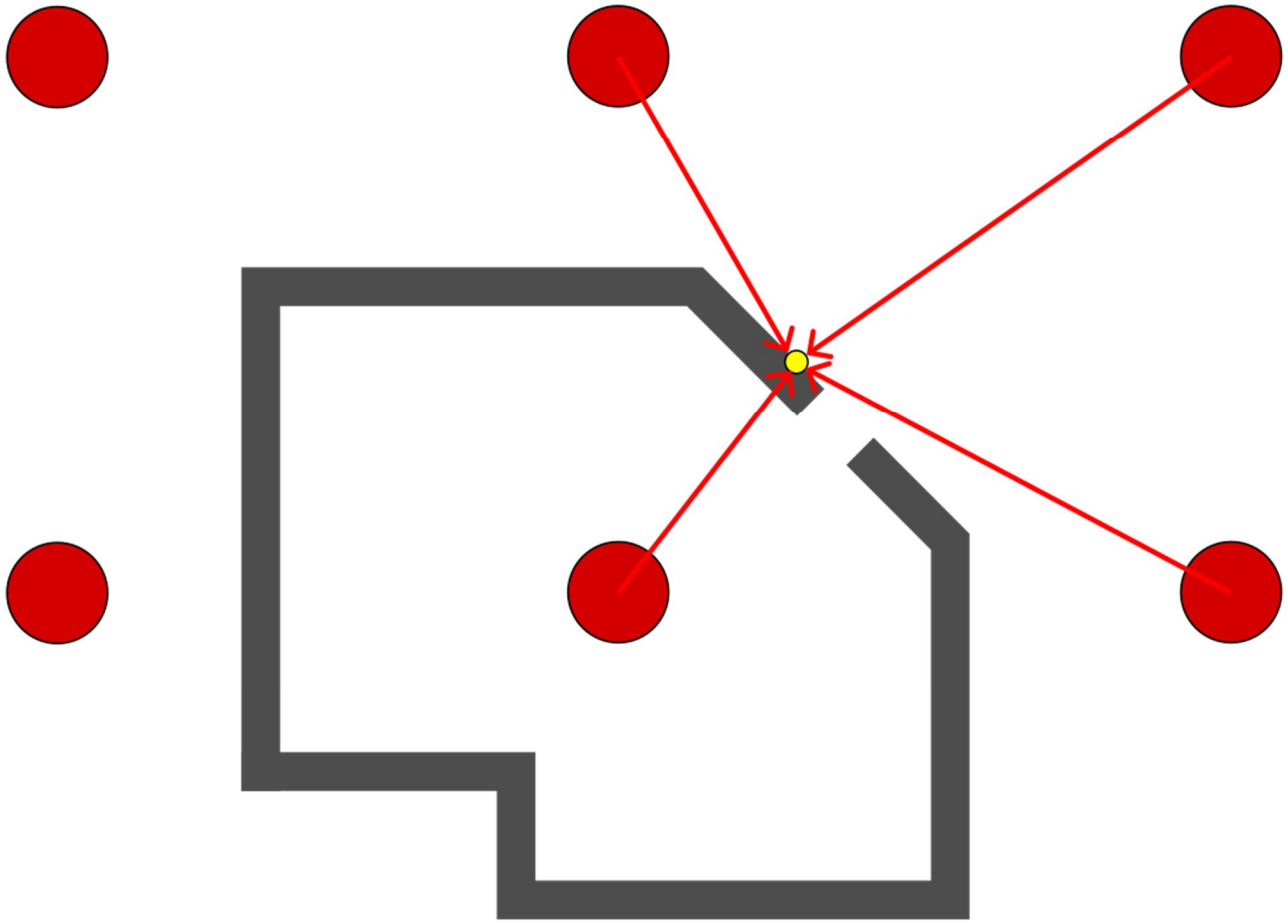
Step 4: Render the scene using interpolated values from the irradiance probes to look up indirect illumination

Repeat Steps 2 and 3 (potentially over consecutive frames) to simulate more bounces of light

Dynamic Diffuse Global Illumination

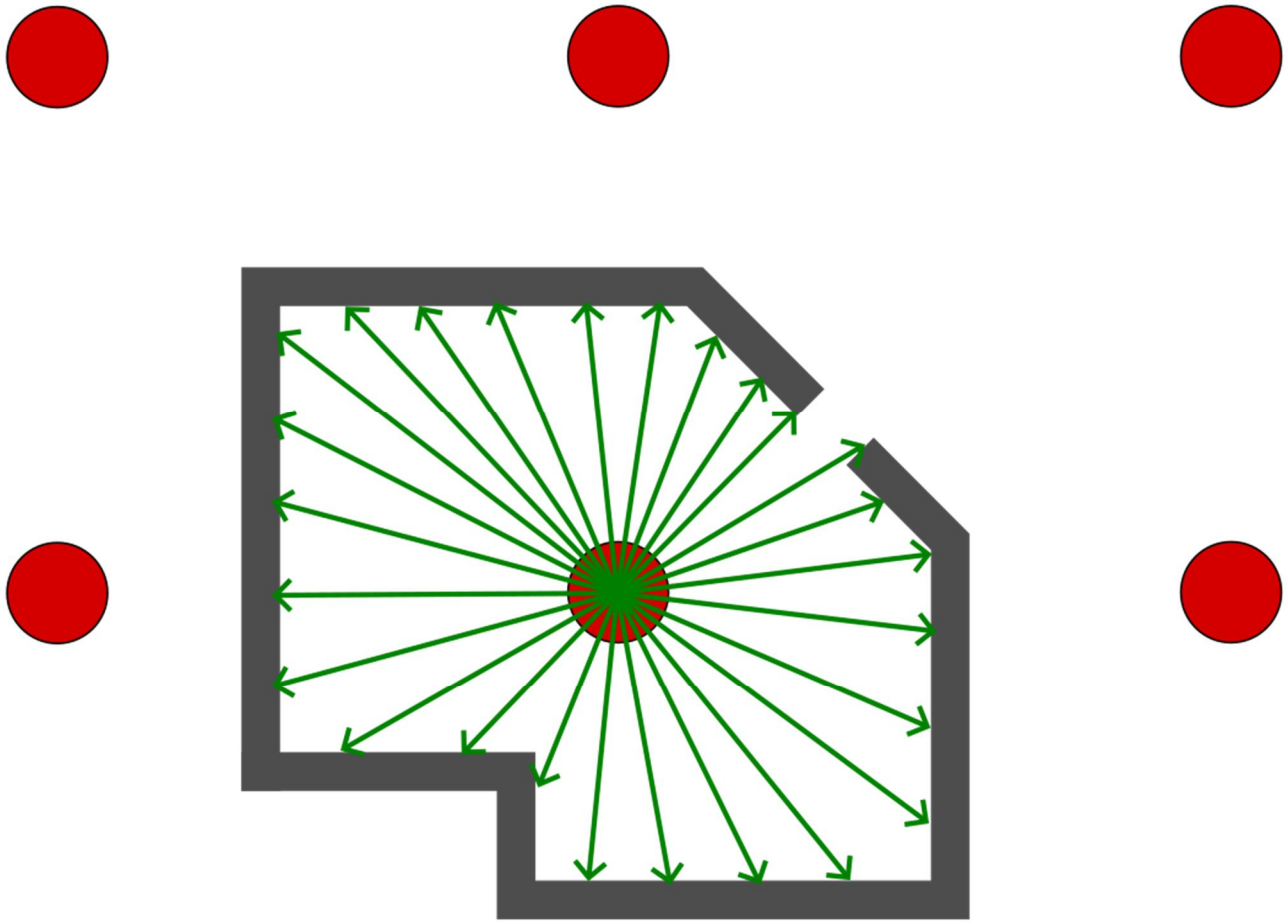


Dynamic Diffuse Global Illumination



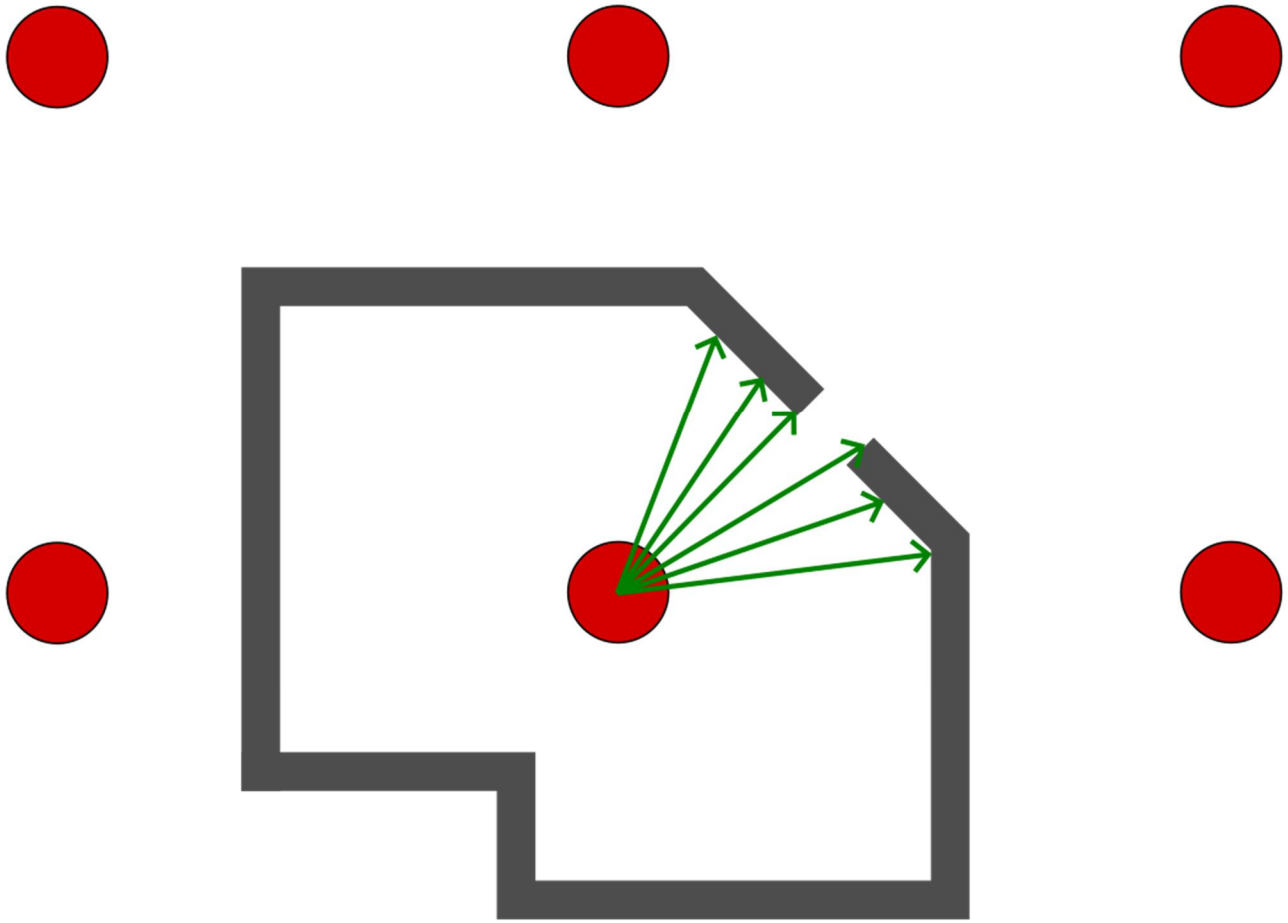
Main issue: how to discount the effect of the occluded probes

Dynamic Diffuse Global Illumination



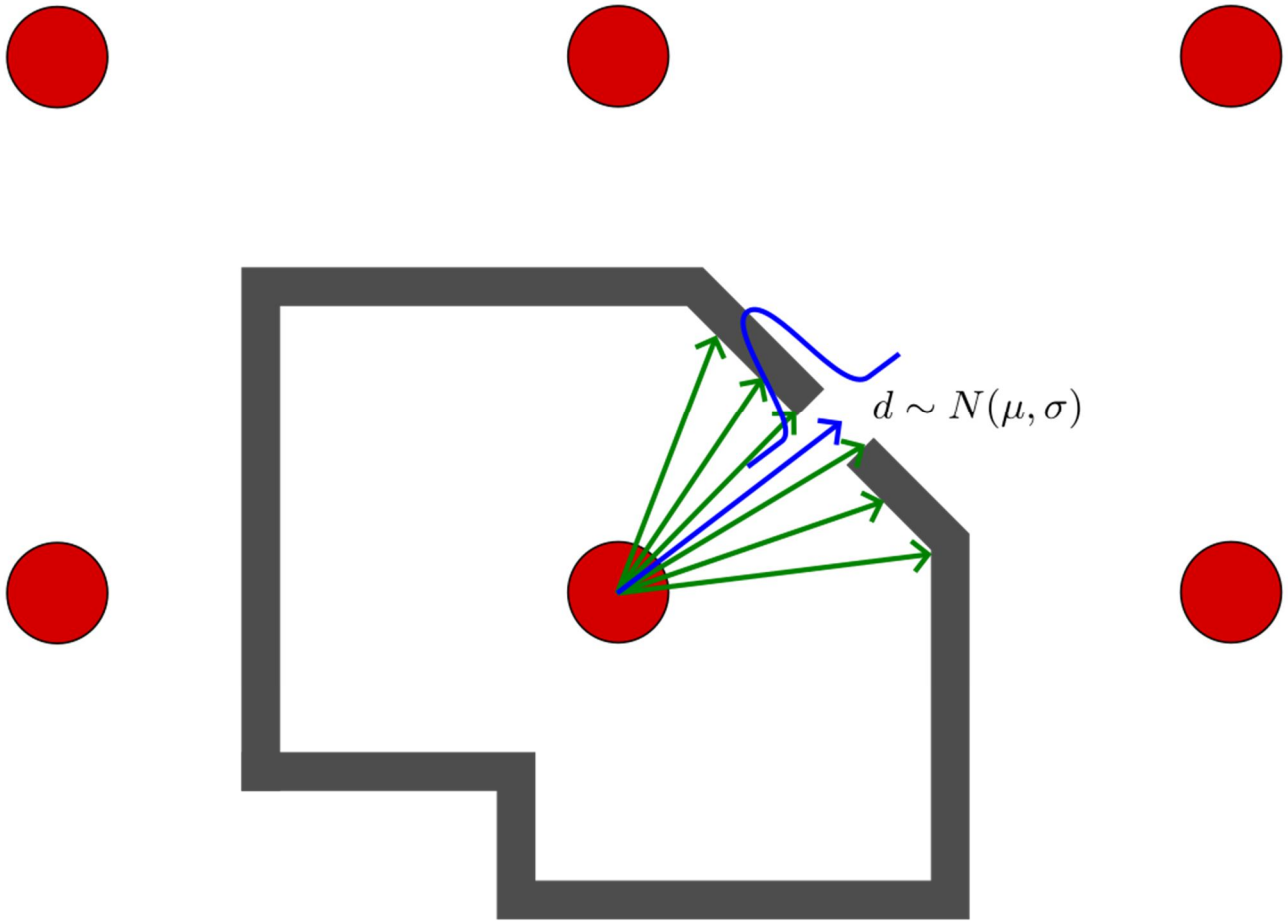
Storing depth per sample is too expensive

Dynamic Diffuse Global Illumination



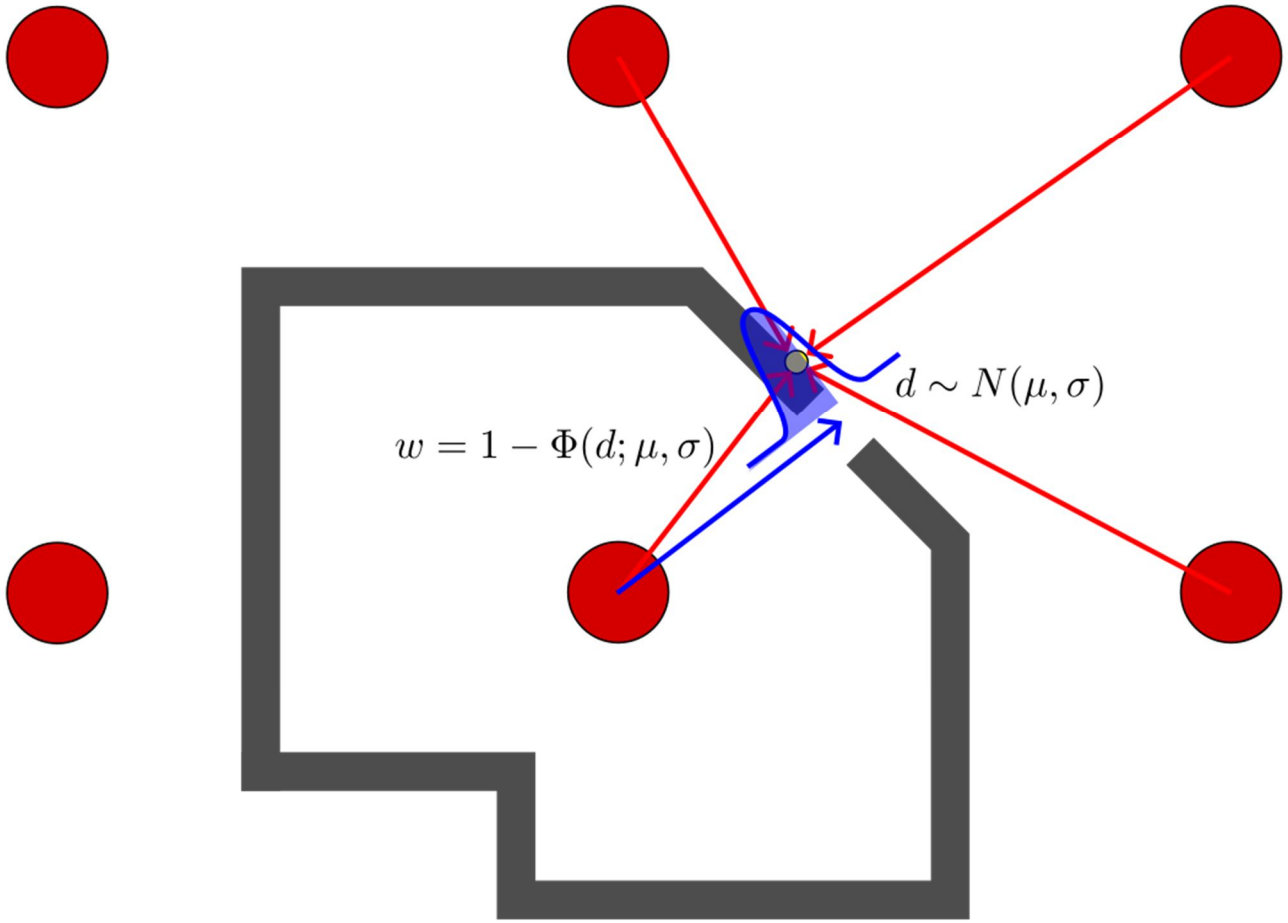
We want to store depth per a range of directions

Dynamic Diffuse Global Illumination



To encode the variation in depth, we store the distribution of depths (mean and variance)

Dynamic Diffuse Global Illumination



When interpolating irradiance from the probes, use the cumulative distribution to determine weight due to shadowing

Ambient occlusion

- Approximates global illumination
- Estimate how much occluded is each surface
  - And reduce the ambient light it receives accordingly
- Much faster than a full global illumination solution, yet appears very plausible
  - Commonly used in animation, where plausible solution is more important than physical accuracy

Image generated with ambient component only (no light) and modulated by ambient occlusion factor.

Ambient occlusion in action

Car photos from John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Ambient occlusion in action

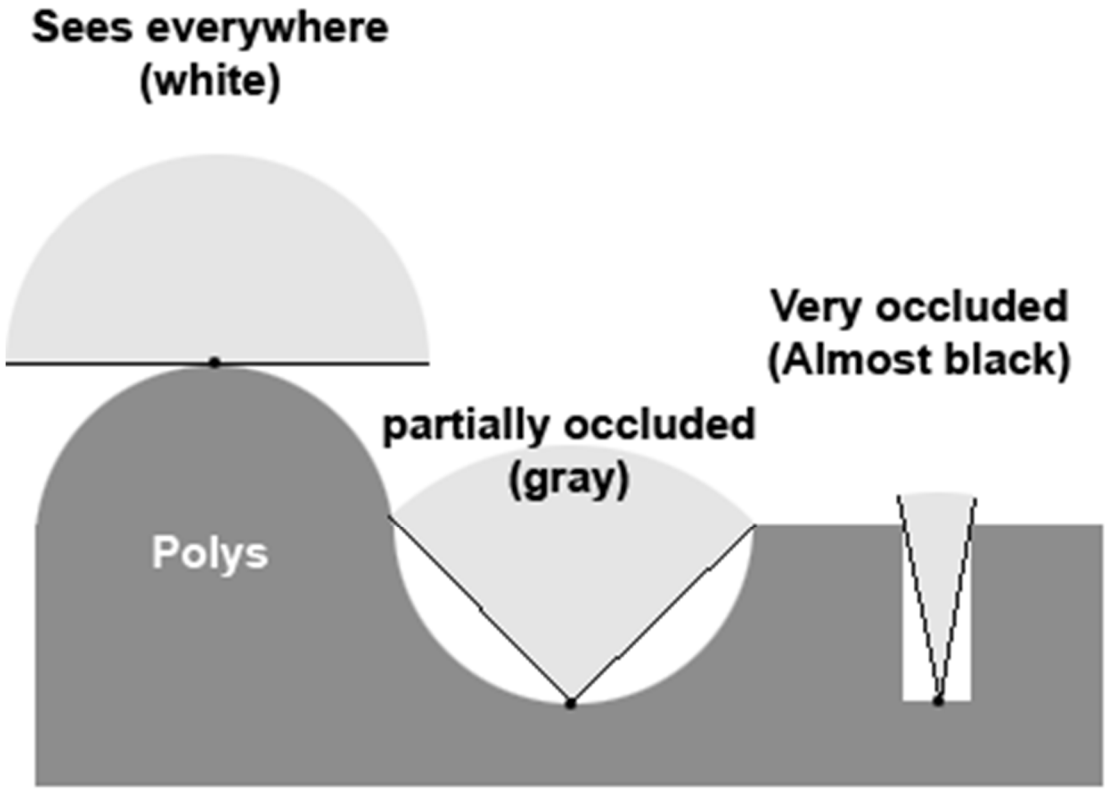
Car photos from John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Ambient occlusion

- For a point on a surface, shoot rays in random directions
- Count how many of these rays hit objects
- The more rays hit other objects, the more occluded is that point
  - The darker is the ambient component

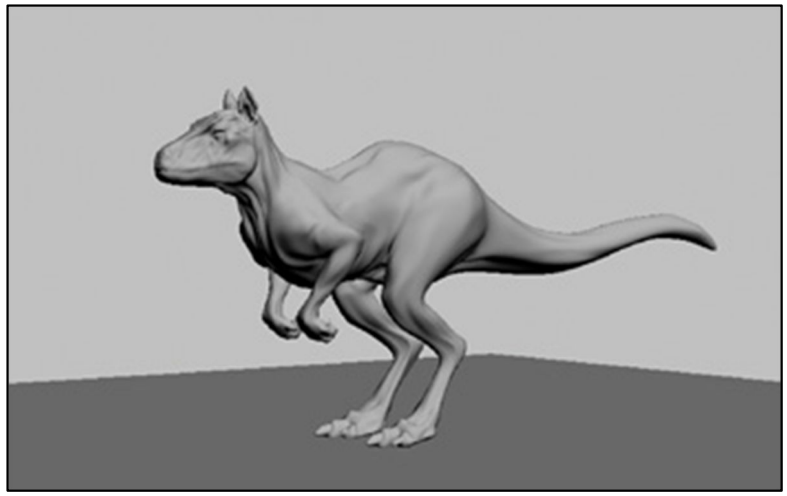
$$A_{\bar{p}} = \frac{1}{\pi} \int_{\Omega} V_{\bar{p},\hat{\omega}} (\hat{n} \cdot \hat{\omega}) d\omega$$

A_p	occlusion at point *p*
n	normal at point *p*
$V_{p,\omega}$	visibility from *p* in direction ω
Ω	integrate over a hemisphere



Ambient occlusion - Theory

- This approach is very flexible
- Also very expensive!
- To speed up computation, randomly sample rays cast out from each polygon or vertex (this is a *Monte-Carlo* method)
- Alternatively, render the scene from the point of view of each vertex and count the background pixels in the render
- Best used to pre-compute per-object "*occlusion maps*", texture maps of shadow to overlay onto each object
- But pre-computed maps fare poorly on animated models...



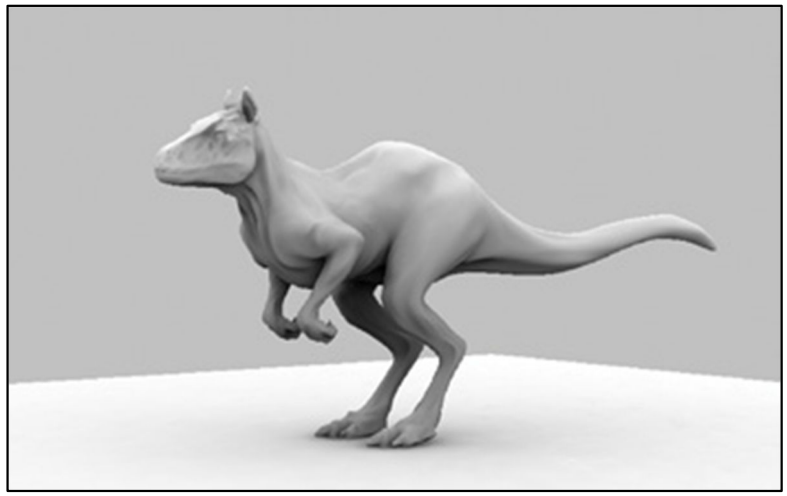


Image credit: "GPU Gems 1", nVidia, 2004.
Top: without AO. Bottom: with AO.

Screen Space Ambient Occlusion - SSAO

"True ambient occlusion is hard, let's go hacking."

- Approximate ambient occlusion by comparing z-buffer values in screen space!
- Open plane = unoccluded
- Closed 'valley' in depth buffer = shadowed by nearby geometry
- Multi-pass algorithm
- Runs entirely on the GPU

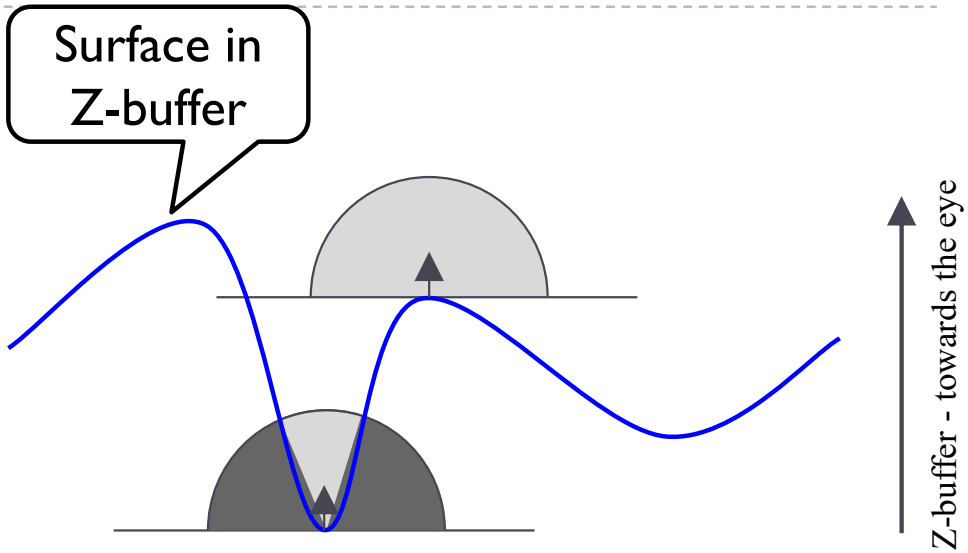


Image: CryEngine 2. M. Mittring, "Finding Next Gen – CryEngine 2.0, Chapter 8", SIGGRAPH 2007 Course 28

References

Shirley and Marschner, "Fundamentals of Computer Graphics", Chapter 24 (2009)

Matt Pharr, Wenzel Jakob, Greg Humphreys, "Physically Based Rendering From Theory to Implementation" (2017)

Dynamic Diffuse Global Illumination

- Majercic et al. "Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields"

Ambient occlusion and SSAO

- "GPU Gems 2", nVidia, 2005. Vertices mapped to illumination. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter14.html
- MITTRING, M. 2007. Finding Next Gen – CryEngine 2.0, Chapter 8, SIGGRAPH 2007 Course 28 – Advanced Real-Time Rendering in 3D Graphics and Games, Siggraph 2007, San Diego, CA, August 2007. http://developer.amd.com/wordpress/media/2012/10/Chapter8-Mittring-Finding_NextGen_CryEngine2.pdf
- John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

Photon mapping

- Henrik Jensen, "Global Illumination using Photon Maps": http://graphics.ucsd.edu/~henrik/
- Henrik Jensen, "Realistic Image Synthesis Using Photon Mapping"
- Zack Waters, "Photon Mapping": http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/photon_mapping/PhotonMapping.html

Some slides are the curtesy of Alex Benton