Global Illumination

Advanced Graphics

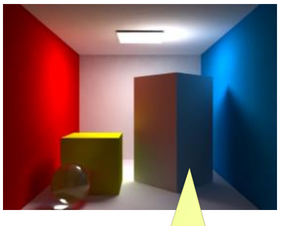
Rafał Mantiuk and Alex Benton
Computer Laboratory, University of Cambridge

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!

Rendering equation (revisited)

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

Incident light

Integral over the hemisphere of incident light

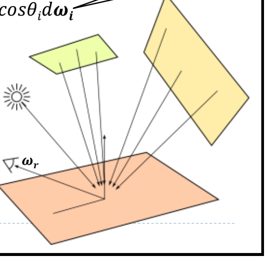
$$L_r(\boldsymbol{\omega}_r) = \int_{\Omega} \rho(\boldsymbol{\omega}_i, \boldsymbol{\omega}_r) L_i(\omega_i) cos\theta_i d\boldsymbol{\omega}_i$$

Reflected light

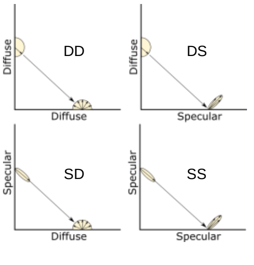
BRDF

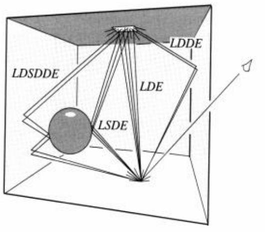
$$\boldsymbol{\omega}_i = [\phi_i, \theta_i]$$

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



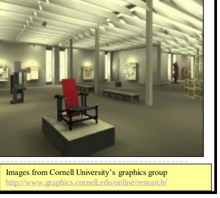
Light transport





Radiosity

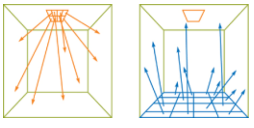
- *Radiosity* is an illumination method which simulates the global dispersion and reflection of diffuse light.
  - First developed for describing spectral heat transfer (1950s)
  - Adapted to graphics in the 1980s at Cornell University
- Radiosity is a finite-element approach to global illumination: it breaks the scene into many small elements ('*patches*') and calculates the energy transfer between them.



Images from Cornell University's graphics group
http://www.graphics.cornell.edu/online/research/

Radiosity—algorithm

- Surfaces in the scene are divided into *patches*, small subsections of each polygon or object.
- For every pair of *patches* A, B, compute a *view factor* (also called a *form factor)* describing how much energy from patch A reaches patch B.
  - The further apart two patches are in space or orientation, the less light they shed on each other, giving lower view factors.
- Calculate the lighting of all directly-lit patches.
- Bounce the light from all lit patches to all those they light, carrying more light to patches with higher relative view factors. Repeating this step will distribute the total light across the scene, producing a global diffuse illumination model.



Radiosity—mathematical support

The 'radiosity' of a single patch is the amount of energy leaving the patch per discrete time interval.

This energy is the total light being emitted directly from the patch combined with the total light being reflected by the patch:

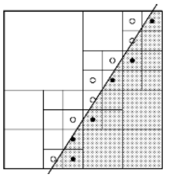
$$B_i = E_i + \rho_i \sum_{\substack{j=1..n,\\ j\neq i}} B_j F_{ij} \quad \Rightarrow \quad \begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \cdots & 1-\rho_N F_{NN} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix}$$

This forms a system of linear equations, where…

- B_i is the radiosity of patch i;
- B_j is the radiosity of each of the other patches ($j \neq i$)
- E_i is the emitted energy of the patch
- ρ_i is the reflectivity of the patch
- F_{ij} is the view factor of energy from patch i to patch j.

Radiosity—form factors

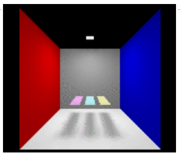
- Finding form factors can be done procedurally or dynamically
  - Can subdivide every surface into small patches of similar size
  - Can dynamically subdivide wherever the 1st derivative of calculated intensity rises above some threshold.
- Computing cost for a general radiosity solution goes up as the square of the number of patches, so try to keep patches down.
  - Subdividing a large flat white wall could be a waste.
- Patches should ideally closely aligned with lines of shadow.



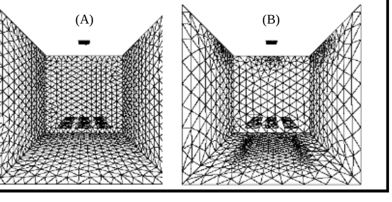
Radiosity—implementation

(A) Simple patch triangulation

(B) Adaptive patch generation: the floor and walls of the room are dynamically subdivided to produce more patches where shadow detail is higher.



Images from "Automatic generation of node spacing function", IBM (1998) http://www.trl.ibm.com/projects/meshing/nsp/nspE.htm



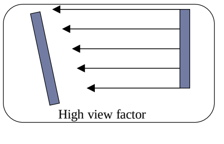
Radiosity—form factors

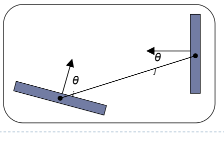
One equation for the view factor between patches i, j is:

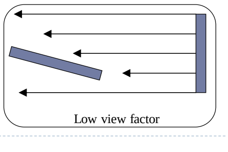
$$Fi \rightarrow j = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V(i,j)$$

Distance between patches

Visibility
1 – patches visible
0 - occluded

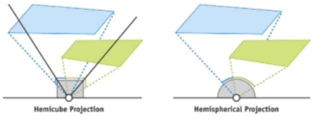






Radiosity—calculating visibility

- Calculating $V(i,j)$ can be slow.
- One method is the *hemicube*, in which each form factor is encased in a half-cube. The scene is then 'rendered' from the point of view of the patch, through the walls of the hemicube; $V(i,j)$ is computed for each patch based on which patches it can see (and at what percentage) in its hemicube.
- A purer method, but more computationally expensive, uses hemispheres.



Note: This method can be accelerated using GPU to render the scene. The scene is 'rendered' with the camera located in the centre of the patch. Use patch index instead of color.

Radiosity gallery

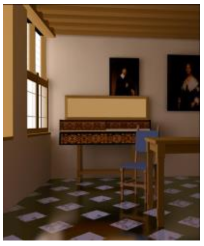


Image from *A Two Pass Solution to the Rendering Equation: a Synthesis of Ray Tracing and Radiosity Methods*, John R. Wallace, Michael F. Cohen and Donald P. Greenberg (Cornell University, 1987)

Image from *GPU Gems II*, nVidia

Teapot (wikipedia)

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)
- When a surface is hit, either (randomly):
  - shoot another ray in the random direction sampled using the BRDF (importance sampling);
  - or terminate
- For every hit-point shoot a shadow (light) ray and add the contribution of the light
- 40+ 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

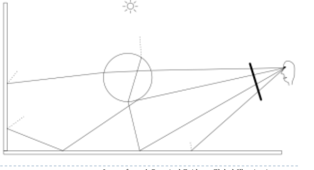


Image from A Practical Guide to Global Illumination using Photon Maps by Henrik Jensen (2000)

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)

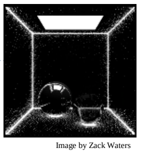


Image by Zack Waters

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, folks.) 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

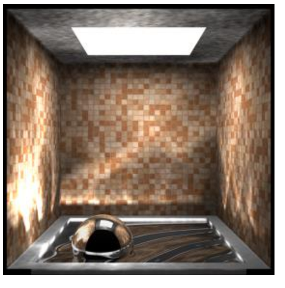
Photon mapping is probabilistic

- 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:

    0 p_d p_t p_s 1

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

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

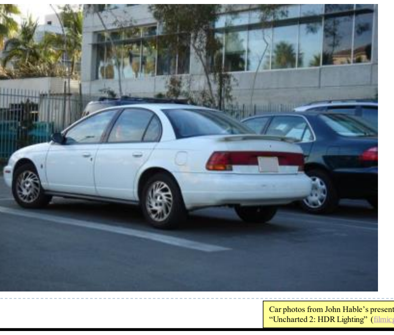
http://www.pbrt.org/gallery.php

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

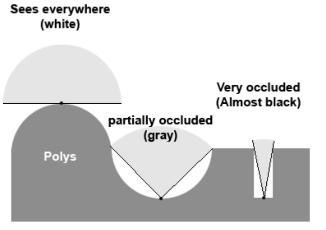
Car photos from John Hable's presentation at GDC 2010, "Uncharted 2: HDR Lighting" (filmicgames.com/archives/6)

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...

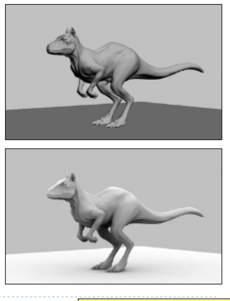


Image credit: "GPU Gems 1", nVidia, 2004.
Top: without AO. Bottom: with AO.

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

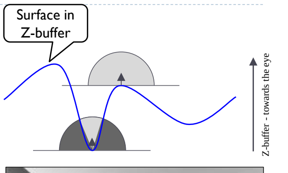


Image: CryEngine 2. M. Mittring, "Finding Next Gen – CryEngine 2.0, Chapter 8", SIGGRAPH 2007 Course 28

References

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

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)

Radiosity
- nVidia: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter39.html
- Cornell: http://www.graphics.cornell.edu/online/research/
- Wallace, J. R., K. A. Elmquist, and E. A. Haines. 1989, "A Ray Tracing Algorithm for Progressive Radiosity." In *Computer Graphics (Proceedings of SIGGRAPH 89)* 23(4), pp. 315–324.
- Buss, "3-D Computer Graphics: A Mathematical Introduction with OpenGL" (Chapter XI), Cambridge University Press (2003)

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