An introduction to Global Illumination

Tomas Akenine-Möller
Department of Computer Engineering
Chalmers University of Technology
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 (so shadows in ray tracing image are black)

Images courtesy of Henrik Wann Jensen

Tomas Akenine-Möller © 2002
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
  - Path tracing
  - Photon mapping

- Great book on global illumination and photon mapping:

Tomas Akenine-Möller © 2002
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
- 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
Examples of light transport notation

- The following expression describes all light paths in this scene: \( L(S|D) \times E \)
- Path A: LDDDE
- Path B: LSDSDE
The ultimate goal is to simulate all light paths: \( L(S|D)^*E \)

Using this notation, we can find what ray tracing can handle:
- \( LDS^*E \mid LS^*E \)
- This is clearly not \( L(S|D)^*E \)!
Background: Radiance

- Radiance, $L$: a radiometric term for what we store in a pixel
  - Most important term!
- Five-dimensional (or 6, including wavelength):
  - Position (3)
  - Direction (2)
- Radiance is "power per unit projected area per unit solid angle" $dA \, d\omega$

Solid angle: measured in Steradians ($4\pi$ is whole sphere).

Uses differentials, so the cone of the solid angle becomes infinitesimally small: a ray
Background: The rendering equation

- Is the basis for all rendering, but especially for global illumination algorithms

\[ L_o(x, \omega) = L_e(x, \omega) + L_r(x, \omega) \]
- outgoing = emitted + reflected radiance
- \( x \) is position on surface, \( \omega \) is direction vector

- Extend the last term \( L_r(x, \omega) \)

\[
L_o = L_e + \int_{\Omega} f_r(x, \omega, \omega') L_i(x, \omega')(\omega' \cdot n) d\omega'
\]
- \( f_r \) is the BRDF (next slide), \( \omega' \) is incoming direction, \( n \) is normal at point \( x \), \( \Omega \) is hemisphere "around" \( x \) and \( n \), \( L_i \) 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

- Many integrals 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 \( n \to \infty \), \( I_{MC} \to I \)
  - Standard deviation convergence is slow: \( \sigma \propto \frac{1}{\sqrt{n}} \)
  - Thus, to halve error, must use 4x number of samples!!
Path tracing
One solution to GI

- Uses Monte Carlo sampling to solve integration: just shoot many random rays over the integral domain

- Example: light 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 a new at each interaction with surface, light, etc.
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

Tomas Akenine-Möller © 2002
Diffuse and Specular surfaces in path tracing

- Assume $k_{\text{diff}} + k_{\text{spec}} \leq 1$
  - Comes from that energy cannot be created, but can be absorbed
  - $k_{\text{diff}}$ 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($r < k_{\text{diff}}$) → send diffuse ray
  - Else if($r < k_{\text{diff}} + k_{\text{spec}}$) → send specular ray
  - Else absorb ray.

- This is often called Russian roulette
Example

- 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
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
Path tracing implemented using \texttt{trace()} and \texttt{shade()} framework

- \texttt{RayTraceImage()}:  
  - Shoot \( n \) rays per pixel  
  - New random position inside pixel for each ray  
  - Possibly also a random time and lens position

- \texttt{trace()}:  
  - Nothing

- \texttt{shade()}:  
  - For area light sources: pick random position on light source  
  - When calling \texttt{trace()}, choose one random ray direction (use Russian roulette)
A classical example

- 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
Extensions to path tracing

- **Bidirectional path tracing**
  - Developed in 1993-1994
  - Sends light paths, both from eye and from the light
  - Faster, but still noisy images.

- **Metropolis light transfer**
  - 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 Russian roulette to find out whether the photon is absorbed or reflected
  - If reflected, the 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
- Some diffuse photons are absorbed, some are scattered further
- A photon = the incoming illumination at a point
- Power of photon is decreased at bounces

This type of arrow is a stored photon
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: $2p+1$
Locate n closest photons
After Henrik Wann Jensen

// locate n closest photons around point "pos"
// call with "locate_photons(1)", i.e., with the root as in argument
locate_photons(p)
{
    if(2p+1 < number of photons in photon map structure)
    {
        // examine child nodes
        delta=signed distance to plane of node n
        if(delta<0)
        {
            // we're to the "left" of the plane
            locate_photons(2p);
            if(delta*delta < d*d)
                locate_photons(2p+1); //right subtree
        }
        else
        {
            // we're to the "right" of the plane
            locate_photons(2p+1);
            if(delta*delta < d*d)
                locate_photons(2p); // left subtree
        }
    }
    delta=real distance from photon p to pos
    if(delta*delta < d*d)
    {
        // photon close enough?
        insert photon into priority queue h
        d=distance to photon in root node of h
    }
}
// think of it as an expanding sphere, that stops expanding when n closest
// photons have been found
What does it look like?

- Stored photons displayed:
Density estimation

- The density of the photons indicate how much light that point receives
- Radiance is the term for what we display at a pixel
- Complex derivation skipped (see Jensen’s book)…
- Reflected radiance at point $x$:

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

- $L$ is radiance in $x$ in the direction of $\omega$
- $r$ is radius of expanded sphere
- $\omega_p$ is the direction of the stored photon
- $\Phi_p$ 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:
  - Use two photon maps
  - A caustics photon map (for caustics)
    - Reflected or refracted via a surface to a diffuse surface
    - Light transport notation: LS+D
  - A global photon map (for all illumination)
    - All photons that landed on diffuse surfaces
    - \( L(S \mid D)^*D \)
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

Tomas Akenine-Möller © 2002
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
    - Specular reflection (also evaluated using ray tracing, possibly, with many rays send around the reflection direction)
    - Caustics: use caustics photon map
    - Multiple diffuse reflections (color bleeding): use the photonmap for reflected rays.
Images of the four components

- These together solves the entire rendering equation!
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
Photon mapping with subsurface scattering

- Photons enter the surface, and bounces around
Much more details to photon mapping...

- 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
  – Same results
  – Disadvantage: rendering times (several hundreds rays per pixel)

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