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 of 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 (revised)

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

\[ \iiint_{\Omega} \iiint_{\Omega} L_r(\omega_i, \omega_o)\Omega_{\omega} d\omega_i d\omega_o \]

- 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.
Radiosity—algorithm

- Surfaces in the scene are divided into patches, small subsections of each polygon or object.
- For every pair of patches A and 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—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 intensity 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 align with lines of shadow.

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:

$$ B_i = E_i + \sum_{j=1}^{N} B_j F_{ij} $$

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 ($\forall j \neq i$).
- $E_i$ is the emitted energy of the patch.
- $\rho_i$ is the reflectivity of the patch
- $F_{ij}$ is the view factor of energy from patch $i$ to patch $j$.

Radiosity—implementation

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


Radiosity—calculating visibility

- Calculating $V_{ij}$ can be slow.
- One method is the hemisphere, 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 hemisphere; $V_{ij}$ is computed for each patch based on which patches it can see (and at what percentage) in its hemicube. The scene is then 'rendered' from the point of view of the patch, through the walls of the hemisphere; $V_{ij}$ 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.
### Radiosity gallery

- **Image**: A photograph of a room with a window and a table, showing a scene with a light source and a shadow.

- **Description**: This image demonstrates a typical radiosity scene, where light is reflected and absorbed by various surfaces.

---

### 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.

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

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

---

### Shadows, refraction and caustics—algorithm (1/2)

**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.

---

### 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.

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

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

---

### 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

---

### Photon mapping—algorithm (1/2)

**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 is often used.
Photon mapping—algorithm (2/2)

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

- 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:
    
    ![Probability Map Diagram]

    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.

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

Ambient occlusion in action

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

Ambient occlusion

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

\[ A_p = \frac{1}{\pi} \int_{\Omega} V_p(\mathbf{n} \cdot \omega) d\omega \]

\( A_p \) occlusion at point \( p \)
\( \mathbf{n} \) normal at point \( p \)
\( V_p \) visibility from \( p \) in direction \( \omega \)
\( \Omega \) 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...

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

References


Ambient occlusion and SSAO

- John Hable’s presentation at GDC 2010, "Uncharted 2: HDR Lighting".


Radiosity

- Cornell: http://www.graphics.cornell.edu/online/research/


Photon mapping

- Henrik Jensen, "Realistic Image Synthesis Using Photon Mapping"