Global Illumination
Global Illumination

• More faithfully (less hacks) tracking the physical process:
  • Emit light from light sources
  • Follow the light from the light sources throughout the scene
  • Account for all reflection and absorption along the way
  • Follow the light through the aperture into the camera
  • Record the final pixel colors when the light hits the film
Soft shadows...
Color Bleeding...
Bidirectional Ray-Tracing
Light Tracing

- Consider following photons
- For each light, choose a number of directions to evenly sample the outgoing hemisphere (or sphere depending on the type of light), and emit a photon in each sample direction
- Follow this photon in a straight line, until it intersects an object
- Then check to see if it is:
  - Absorbed - terminate the photon; it does not make it to the film inside the camera
  - Reflected/Transmitted/Scattered by the BRDF - create a new ray and send the photon off in that direction
- Continue bouncing photons around the scene, terminating them when absorbed
- The photons that make it through the aperture and hit the film inside the camera activate the sensor and contribute to the lighting of the scene
- N.B. very few photons ever hit the film, which is the case in reality – most of the light in the world is not seen by your eyes!
Light Tracing

most of the light never hits the film (light tracing fails)
Path Tracing

- Trace photons backward in the opposite direction from the film to the light
- For each pixel send out a ray through the aperture into the scene to trace the photon that hit that pixel backwards along its path
  - (This is how we do ray tracing already!)
- If the ray hits a light, use the outgoing light in the ray direction to determine how much light hits the pixel
- If the ray hits an object, send out reflected and transmitted rays as usual (for ray tracing)
- Continue following all these rays terminating them when they hit a light
- A terminated ray only gives us a path
  - Still need to emit a photon in this direction and bounce it off all the objects along the path, checking to see if it is absorbed along the way before it makes it to the pixel on the film
  - Typically one emits many photons in this direction as some percentage of them are absorbed resulting in a specific color and brightness of light hitting the pixel
Path Tracing

most of the scene cannot see the light
(path tracing fails)
BRDF Path Tracing

- Almost all ray paths from the camera bounce around and end up eventually going out a window and off into outer space never finding a light.
- This is because reflected and transmitted rays do not properly represent the backwards path of the photons.
- Instead, given a point on the surface of an object, photons bombard that point from every direction of the hemisphere.
- Thus, we cannot just spawn one reflected and one transmitted ray, but rather need to spawn rays in all directions of the hemisphere to trace all the photons backwards from every incoming direction.
- This means spawning many, many rays at every intersection point, which is expensive and impractical.
- And every one of these rays that hits another point on another surface will spawn an entirely new hemisphere of rays (exponential growth).
BRDF Path Tracing

most of the scene STILL cannot see the light
(BRDF path tracing fails)
Ray Tracing

• Basic ray tracing assumes that all the incoming directions on the hemisphere are unimportant except for those that go directly towards the light source – and thus sends out only those rays
  • Except for reflection/transmission where these particular ray directions of the hemisphere (or sphere) are also considered important
• Ignoring all the other directions of the hemisphere is called direct illumination (only considering light directly from the light source)
• This ignores global illumination and its desired effects
• Worse yet, if the light is occluded in some way, most of the scene cannot directly see it and ray tracing will fail to find any illumination whatsoever
  • Everything occluded will be in shadow with only ambient lighting
Bidirectional Ray Tracing

- Combines Light Tracing and Path Tracing
- First, bathe objects in photons from the light and record their lighting information in a light map (similar to a texture map)
- This allows photons to bounce around the scene illuminating shadows, bleeding color, etc., etc.
- Then, when ray tracing the scene, use the light map at each point to estimate the indirect light coming in from all the directions of the hemisphere
- Still sample the most important directions on the hemisphere explicitly:
  - Still sends rays directly to the light sources to gather direct illumination, and still send out reflected and transmitted rays to gather lighting information directly in these important directions as well
Light Maps

- Light maps work great for soft shadows, color bleeding, etc.
- But they can also be used to generate many other interesting effects...
Lighting Equation
Strategy

• Evaluate the lighting equation everywhere in the scene, irrespective of whether those locations are visible from the camera or not

• Disadvantage: Expensive
  • Requires consideration of all incoming light and all outgoing light at every point in the scene
  • Requires a lot more work(!) to compute lighting everywhere as opposed to calculating it only at points visible to camera

• Advantage: Far more realistic
  • Allows us to track the full path of light and account for the multiple bounces in indirect lighting
  • Once this computation is performed, one can place the camera anywhere in the scene and use the already computed light maps
Recall: The Lighting Equation

Multiplying the BRDF by an incoming irradiance gives a resulting outgoing radiance

\[ L_{o\,due\,to\,i}(\omega_i, \omega_o) = BRDF(\omega_i, \omega_o) dE_i \]

For more complex lighting we will bounce light all around the scene, and it is tedious to convert between irradiance and radiance so we use (from two slides earlier) to obtain:

\[ L_{o\,due\,to\,i}(\omega_i, \omega_o) = BRDF(\omega_i, \omega_o)L_i d\omega_i \cos \theta_i \]

Finally, the outgoing radiance considering the light coming from all incoming directions is

\[ L_o = \int_{i\in in} BRDF(\omega_i, \omega_o)L_i \cos \theta_i d\omega_i \]

completing the derivatic


Lighting Equation

\[ L_o = \int_{i \in \text{in}} BRDF(\omega_i, \omega_o) L_i(\omega_i) \cos \theta_i \, d\omega_i \]

- Explicitly write the angular dependence for L:

\[ L_o(\omega_o) = \int_{i \in \text{in}} BRDF(\omega_i, \omega_o) L_i(\omega_i) \cos \theta_i \, d\omega_i \]

- This equation is valid throughout space, so explicitly write L and the BRDF as continuous functions of space:

\[ L_o(x, \omega_o) = \int_{i \in \text{in}} BRDF(x, \omega_i, \omega_o) L_i(x, \omega_i) \cos \theta_i \, d\omega_i \]

- Write the BRDF( ) as f( ) for brevity:

\[ L_o(x, \omega_o) = \int_{i \in \text{in}} L_i(x, \omega_i) f(x, \omega_i, \omega_o) \cos \theta_i \, d\omega_i \]
Aside: Participating Media

• L can be defined over all 3D space, not just on 2D surfaces
• On surfaces $L_o$ defines the outgoing light and it depends on the incoming light in a hemisphere above the surface centered at that point
• For points not on surfaces, e.g. in the “air”, $L_o$ accounts for participating media, e.g. dust particles, and depends on the incoming light in a sphere centered at that point
• If neglecting participating media, e.g. in a vacuum, then we only need to compute L on surfaces
Treating Surfaces & Lights Uniformly

Surface inter-reflection: \( L_i(x, \omega_i) = L_o(x', -\omega_i) \)

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{i \in \text{hemi}} L_o(x'_i, -\omega_i) f(x, \omega_i, \omega_o) \cos \theta_i \, d\omega_i
\]

Emission (from a light)  All Light (emitted and reflected)
Implicit Equation

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{i \in \text{hemi}} L_o(x_i', -\omega_i) f(x, \omega_i, \omega_o) \cos \theta_i \, d\omega_i \]

- Computing reflected radiance on a particular surface requires knowing the incoming radiance from all other surfaces (and lights)
- But the incoming radiance from all those other surfaces typically depends on the outgoing radiance from the surface under consideration
Integral Equation

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{i \in \text{hemi}} L_o(x'_i, -\omega_i) f(x, \omega_i, \omega_o) \cos \theta_i \, d\omega_i \]

Reflected Light  Emission  Reflected Light  BRDF  incident angle
UNKNOWN  KNOWN  UNKNOWN  KNOWN  KNOWN

- Fredholm Integral Equation of the second kind
- extensively studied numerically
- canonical form:

\[ l(u) = e(u) + \int l(v) \, k(u, v) \, dv \]

kernel \( k(u, v) \)
Linearity & Discretization
Linear Operator Theory

- Linear operators act on functions like matrices act on vectors:
  \[ h(u) = (M \circ f)(u) \]
  M is a linear operator, f and h are functions of u

- Basic linearity relations hold:
  \[ M \circ (af + bg) = a(M \circ f) + b(M \circ g) \]
  scalars: a, b
  functions: f, g

- Examples include differentiation and integration:
  \[ (D \circ f)(u) = \frac{\partial f}{\partial u}(u) \]
  \[ (K \circ f)(u) = \int k(u, v)f(v)dv \]
Linear Operator Theory

\[ l(u) = e(u) + \int l(v) k(u, v) \, dv \]

Light Transport Operator

\[ L = E + KL \]

**Discretization:**

- pick a number of points in the world p each representing an area chunk of a surface (2D discretization)
  - points can be chosen throughout the 3D volume for participating media
- for each point, pick a number of outgoing directions q each representing a subset of solid angles of the hemisphere or sphere (2D discretization)
  - generally speaking, q can vary from surface chunk to surface chunk
- then L and E are vectors of length p*q (4D or 5D discretization)
- K is the light transport matrix with size (p*q) by (p*q)
- K and E are known
- result is a set of linear equations for the unknown vector L
Linear Operator Theory

\[ L = E + KL \]

\[ IL - KL = E \]

\[ (I - K)L = E \]

\[ L = (I - K)^{-1}E \]

Using the Binomial Theorem:

\[ L = (I + K + K^2 + K^3 + \ldots)E \]

\[ L = E + KE + K^2E + K^3E + \ldots \]

• This power series expansion is only useful if the higher order terms vanish
• Term \( n \) corresponds to \( n-1 \) bounces of light, and at each bounce some light is absorbed
• Thus, the higher order terms do tend to vanish
Power Series

\[ L = E + KE + K^2E + K^3E + \ldots \]

- Emission directly from light sources
- Direct Illumination on surfaces
- Emission indirectly from light sources (one bounce indirect)
- Global Illumination (two bounce indirect)

OpenGL Shading & Simple Ray Tracer
Power Series

\[ L_e \]

\[ K \circ L_e \]

\[ K \circ K \circ L_e \]

\[ K \circ K \circ K \circ L_e \]

\[ L_e \]

\[ L_e + K \circ L_e \]

\[ L_e + \cdots K^2 \circ L_e \]

\[ L_e + \cdots K^3 \circ L_e \]
From left to right: 1 bounce, 2 bounces, 4 bounces
Interchange Solid Angle and Area

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{i \in \text{Hemi}} L_o(x_i', -\omega_i) f(x, \omega_i, \omega_o) \cos \theta_i \, d\omega_i \]

Recall: \[ d\omega = \frac{dA \cos \theta}{r^2} \]

So we can write and subsequently discretize the integral in terms of surface area patches using:

\[ d\omega_i = \frac{dA' \cos \theta_o}{|x - x'|^2} \]
Interchange Solid Angle and Area

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + \int L_o(x_i', -\omega_i) f(x, \omega_i, \omega_o) \frac{\cos \theta_i \cos \theta_o}{|x - x'|^2} dA' \]

all \( x' \) visible to \( x \)

Define a geometric term:

\[ G(x, x') = G(x', x) = \frac{\cos \theta_i \cos \theta_o}{|x - x'|^2} \]

Introduce a binary visibility function \( V \):

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + \int L_o(x_i', -\omega_i) f(x, \omega_i, \omega_o) G(x, x') V(x, x') dA' \]

all surfaces \( x' \)
Specular Highlights

• Equations are too hard to solve as is…

• Simplifying assumption:
  • Assume all materials are diffuse
  • Compute specular highlights using a separate method

• Step 1: compute the view independent, global ambient and diffuse illumination for the entire scene in a pre-processing step

• Step 2: compute the view dependent, specular illumination “on-the-fly” using that component of the Phong shading model or another method
Radiosity and Albedo

**Radiosity** - power per unit surface area leaving a surface

- same as irradiance but outgoing instead of incoming

\[ B(x) = \frac{d\Phi}{dA} = \int L(x, \omega) \cos \theta d\omega = \pi L(x) \]

The last equality holds if \( L \) is independent of angle, which is true for purely diffuse surfaces.

**Albedo** - in a similar way, we define a “reflection coefficient” (the albedo) in terms of the BRDF:

\[ \rho(x) = \int f(x, \omega_o, \omega_i) \cos \theta d\omega_o = \pi f(x) \]

The last equality holds if the BRDF is independent of angle, which is true for purely diffuse surfaces.
Radiosity Equation

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + \int L_o(x', -\omega_i) f(x, \omega_i, \omega_o) G(x, x') V(x, x') dA' \]

- Assume a purely diffuse surface with a constant BRDF given by \( k_d \)

\[ L_o(x, \omega_o) = L_e(x, \omega_o) + k_d(x) \int L_o(x', -\omega_i) G(x, x') V(x, x') dA' \]

- Assume all surfaces look the same from all directions, which is only true for purely diffuse surfaces

- Then we can integrate over all incoming/outgoing angles removing the dependence on incoming/outgoing angles:

\[ B(x) = E(x) + \rho(x) \int B(x') \frac{G(x, x') V(x, x')}{\pi} dA' \]

Note B and E are only 2D now, instead of 4D!
System of Equations

\[ B_i = E_i + \rho_i \sum_j B_j F_{ij} \]

\[ B_i - \rho_i \sum_j B_j F_{ij} = E_i \]

Or in matrix form:

\[
\begin{bmatrix}
1 - \rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\
-\rho_2 F_{21} & 1 - \rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
-\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 - \rho_n F_{nn}
\end{bmatrix}
\begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_n
\end{bmatrix}
= 
\begin{bmatrix}
E_1 \\
E_2 \\
\vdots \\
E_n
\end{bmatrix}
\]
Form Factor
Form Factor

\[ B(x) = E(x) + \rho(x) \int_{all \ surfaces \ x'} B(x') \frac{G(x, x')V(x, x')}{\pi} \, dA' \]

Discretized over surface chunks:

\[ B_i = E_i + \rho_i \sum_j B_j F_{ij} \]

- The form factor is the fraction of energy leaving one surface patch that reaches another surface patch.
- It is a purely geometric relationship, independent of viewpoint or surface attributes.

\[ F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} \, dA_i dA_j \]
Nusselt developed a geometric analog of the differential form factor to aid in form factor calculations.

The "Nusselt analog" involves placing a hemispherical projection body, with unit radius, at a point on a surface.

The second surface is spherically projected onto the projection body, and then cylindrically projected onto the base of the hemisphere.

The form factor is the area projected on the base of the hemisphere divided by the area of the base.
Form Factor

• The hemicube (half a cube) strategy pre-computes the contribution to the form factor from each cell on the surface of a hemicube

• Then a surface patch is projected onto the hemicube, and the form factor is approximated using the pre-computed form factor values from the relevant portion of the hemicube

It’s faster to project onto a hemicube than onto a hemisphere, because a hemicube is made up of planes
Solving the Equations
**System of Equations**

Consider the system:

\[ \begin{align*}
2x + y &= 8 \\
x + 2y &= 10
\end{align*} \]

i.e. \[ \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 10 \end{bmatrix} \]

with the solution \[ \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \]

- We can invert the matrix and solve for \( x \) and \( y \), but this becomes more and more expensive as the system grows in size.
- Instead, use an iterative method that approximates the solution and iteratively improves the current approximation.
System of Equations

- Start with a guess (any guess) for the solution and try to improve it iteratively
- Initial guess:  \[ \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \]

### Jacobi method

\[
\begin{align*}
x^{(k)} &= (8 - y^{(k-1)}) / 2 \\
y^{(k)} &= (10 - x^{(k-1)}) / 2
\end{align*}
\]

use old values to compute new values

### Gauss Seidel method

\[
\begin{align*}
x^{(k)} &= (8 - y^{(k-1)}) / 2 \\
y^{(k)} &= (10 - x^{(k)}) / 2
\end{align*}
\]

always use the most up to date values possible
# System of Equations

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Jacobi</th>
<th>Gauss Seidel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2.5</td>
<td>4.25</td>
</tr>
<tr>
<td>5</td>
<td>1.875</td>
<td>3.75</td>
</tr>
<tr>
<td>6</td>
<td>2.125</td>
<td>4.0625</td>
</tr>
<tr>
<td>7</td>
<td>1.96875</td>
<td>3.9375</td>
</tr>
<tr>
<td>8</td>
<td>2.03125</td>
<td>4.015625</td>
</tr>
<tr>
<td>9</td>
<td>1.9921875</td>
<td>3.984375</td>
</tr>
<tr>
<td>10</td>
<td>2.0078125</td>
<td>4.00390625</td>
</tr>
<tr>
<td>11</td>
<td>1.998046875</td>
<td>3.99609375</td>
</tr>
<tr>
<td>12</td>
<td>2.001953125</td>
<td>4.000976563</td>
</tr>
<tr>
<td>13</td>
<td>1.999511719</td>
<td>3.999023438</td>
</tr>
<tr>
<td>14</td>
<td>2.000488281</td>
<td>4.000244141</td>
</tr>
<tr>
<td>15</td>
<td>1.99987793</td>
<td>3.999755859</td>
</tr>
<tr>
<td>16</td>
<td>2.00012207</td>
<td>4.000061035</td>
</tr>
<tr>
<td>17</td>
<td>1.999969482</td>
<td>3.999938965</td>
</tr>
<tr>
<td>18</td>
<td>2.000030518</td>
<td>4.000015259</td>
</tr>
<tr>
<td>19</td>
<td>1.999992371</td>
<td>3.999984741</td>
</tr>
<tr>
<td>20</td>
<td>2.000007629</td>
<td>4.000003815</td>
</tr>
</tbody>
</table>
## Better Initial Guess

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Jacobi</th>
<th>Gauss Seidal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2.5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3.75</td>
</tr>
<tr>
<td>4</td>
<td>2.125</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3.9375</td>
</tr>
<tr>
<td>6</td>
<td>2.03125</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3.984375</td>
</tr>
<tr>
<td>8</td>
<td>2.0078125</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>3.99609375</td>
</tr>
<tr>
<td>10</td>
<td>2.001953125</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>3.999023438</td>
</tr>
<tr>
<td>12</td>
<td>2.000488281</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>3.999755859</td>
</tr>
<tr>
<td>14</td>
<td>2.00012207</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>3.999938965</td>
</tr>
<tr>
<td>16</td>
<td>2.000030518</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>3.999984741</td>
</tr>
<tr>
<td>18</td>
<td>2.000007629</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>3.999996185</td>
</tr>
<tr>
<td>20</td>
<td>2.000001907</td>
<td>4</td>
</tr>
</tbody>
</table>
Iterative Radiosity

- **Gathering** - updates one surface by collecting light energy from all surfaces.
- **Shooting** - updates all surfaces by distributing light energy from one surface.
Iterative Radiosity

• Sorting and shooting:
  • chooses the surface with the greatest unshot light energy and uses shooting to distribute its light energy to other surfaces in the environment
  • so it starts by shooting light energy out of the lights onto objects (the brightest light goes first)
  • then the object that reflects the most light goes next, etc.

• Sorting and shooting with ambient:
  • starts with an initial guess for ambient lighting and does sorting and shooting afterwards
Finally...
Question 1 (short/long)

Tell me about global illumination...
(Long: 2 pages)