Global Illumination


rspective
Lit
Show
TING NEEDS TO BE REBUILT (2,086 unbuilt objects)
ECTION CAPTURES NEED TO BE REBUILT (1 unbuilt)
10
10°
0.25
4
Level: NewMap_01 (Persistent)

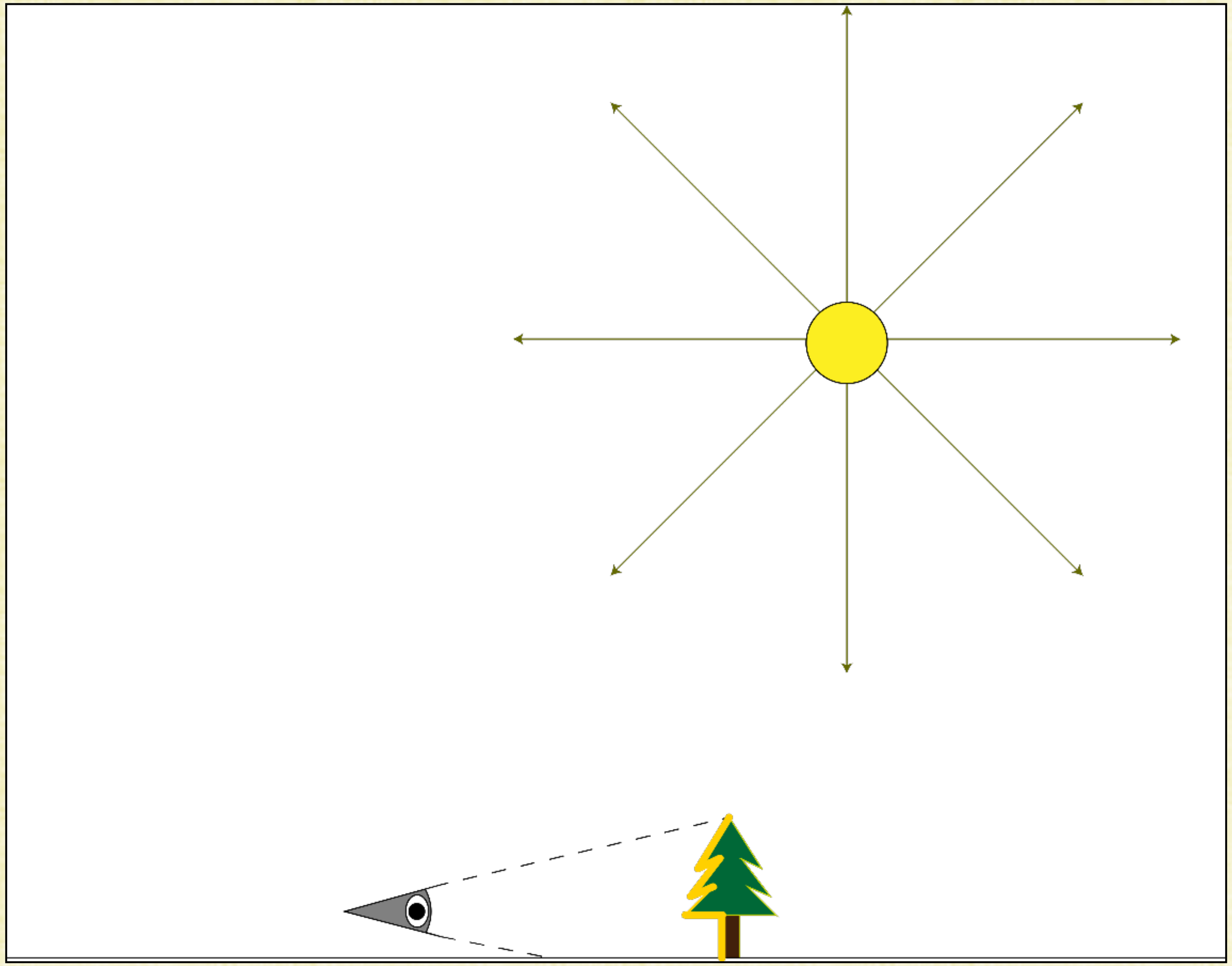
Photon Tracing

- For each light, choose a number of outgoing directions (on the hemisphere or sphere); emit a photon in each direction
- Each photon travels in a straight line, until it intersects an object
- If Absorbed: terminate photon (it doesn’t get to the film)
- If Reflected/Transmitted/Scattered: photon goes off in a new direction (until it again intersects an object)

- If a photon goes through the camera aperture and hits the film, it contributes to the final image

Photon Tracing

- Most of the light never hits the film (far too inefficient, impractical)

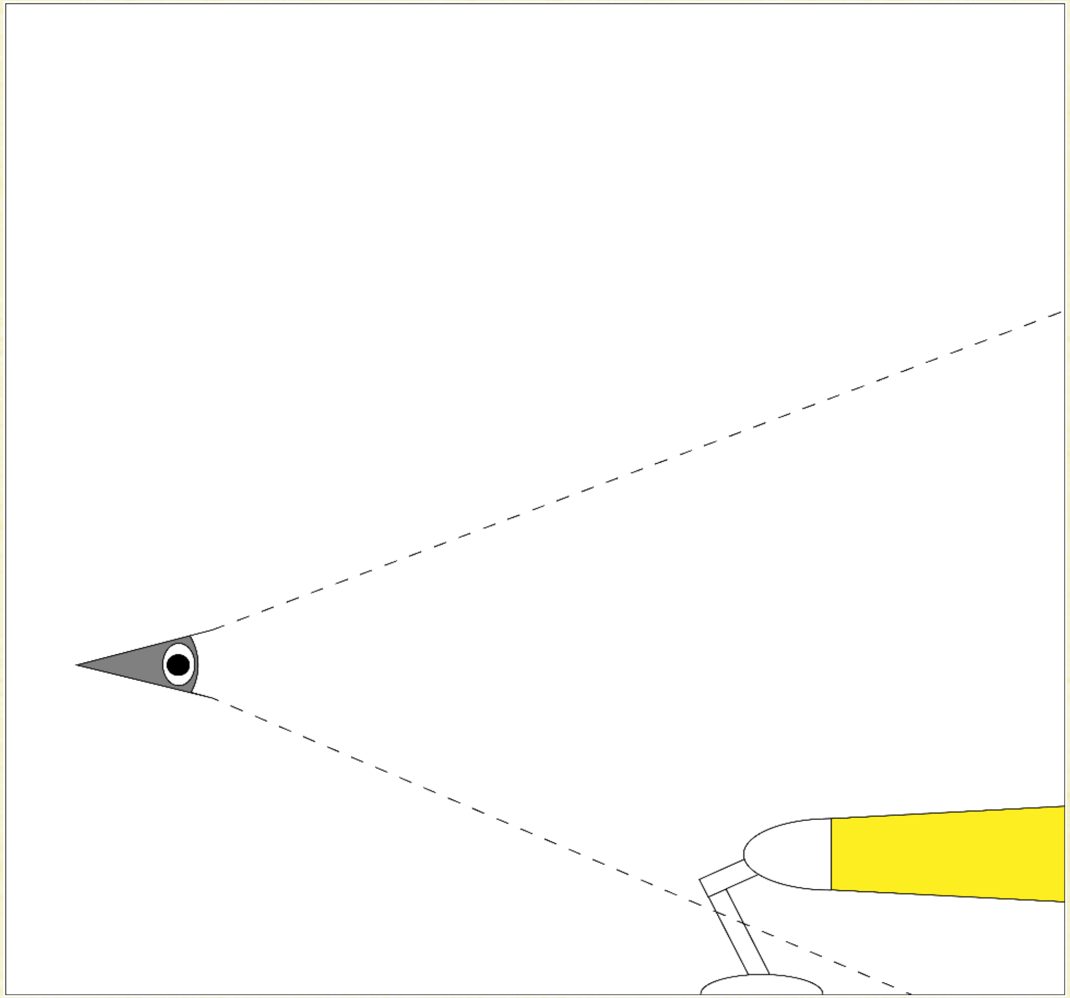


(Backward) Path Tracing

- For each pixel, send a ray through the aperture to backward trace a photon that would hit the pixel (same as ray tracing)
- If the ray hits an object, cast rays in all directions of the hemisphere in order to backwards trace incoming photons
  - Every new ray that hits another surface spawns an entire hemisphere of rays of its own (exponential growth, impractical)
- Follow all rays until they hit a light source (and terminate)
- A terminated ray (only) gives a path from the light source to the pixel
  - Emit photons along this path, bounce them off all the objects along the path, check to see if absorbed (otherwise, continue on towards the pixel)
  - Some percentage of the photons are absorbed resulting in a specific color/brightness of light hitting the pixel (along that path)

(Backward) Path Tracing

- Most paths take too long to find their way back to the light source (inefficient)



Ray Tracing (a more efficient Path Tracing)

- Ignore most incoming directions on the hemisphere, only keeping the most important ones:

- Rays incoming directly from the light source have a lot of photons
  - A Shadow Ray is used to account for this incoming light
  - Called direct illumination (since light is coming directly from a light source)
- Reflective objects bounce a lot of photons in the mirror reflection direction
  - This incoming light is accounted for with a Reflected Ray
- Transparent objects transmit a lot of photons along the transmitted ray direction
  - This incoming light is accounted for with a Transmitted Ray

- Downside: ignoring a lot of the light, and its visual effects

Bidirectional Ray Tracing

- Combine Photon Tracing and Ray Tracing
- Step 1: Emit photons from the light, bathe objects in those photons, and record the result in a light map
  - Photons bounce around illuminating shadowed regions, bleeding color, etc.
  - Note: light maps don't change when the camera moves (so they can be precomputed)
- Step 2: Ray trace the scene, using the light map to estimate indirect light (from the ignored directions of the hemisphere)

- IMPORTANT: Still treat the most important directions (on the hemisphere) explicitly, for increased accuracy
  - Shadow Rays for direct illumination
  - Reflected Rays
  - Transmitted Rays

Light Maps

- Light maps work great for soft shadows, color bleeding, etc.
- They can also generate many other interesting effects:

Recall: Lighting Equation

- Multiplying the BRDF by an incoming irradiance gives the outgoing radiance

$$dL_{o\ due\ to\ i}(\omega_i, \omega_o) = BRDF(\omega_i, \omega_o) dE_i(\omega_i)$$

- For even more realistic lighting, we'll bounce light all around the scene
- It's tedious to convert between E and L, so use $dE = L d\omega \cos\theta$ to obtain:

$$dL_{o\ due\ to\ i}(\omega_i, \omega_o) = BRDF(\omega_i, \omega_o) L_i d\omega_i \cos\theta_i$$

- Then,

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

Lighting Equation

- Explicitly add the dependencies on the surface location x and incoming angle ω_i
- Change $i \in in$ for "incoming directions" to $i \in hemi$ for "hemisphere"
- Add an emission term L_e, so x can be a location on the surface of actual lights too

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{i \in hemi} BRDF(x, \omega_i, \omega_o) L_i(x, \omega_i) \cos \theta_i \, d\omega_i$$

- Incoming light from direction ω_i left some other surface point x' going in direction $-\omega_i$
- So, replace $L_i(x, \omega_i)$ with $L_o(x', -\omega_i)$

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{i \in hemi} BRDF(x, \omega_i, \omega_o) L_o(x', -\omega_i) \cos \theta_i \, d\omega_i$$

An Implicit Equation

- Computing the outgoing radiance $L_o(x, \omega_o)$ on a particular surface requires knowing the outgoing radiance $L_o(x', -\omega_i)$ from all the other (relevant) surfaces
- But the outgoing radiance from those other surfaces (typically) depends on the outgoing radiance from the surface under consideration (circular dependencies)

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{i \in hemi} L_o(x', -\omega_i) \boxed{BRDF(x, \omega_i, \omega_o) \cos\theta_i \, d\omega_i}$$

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

- Fredholm Integral Equation of the second kind (extensively studied) given in canonical form with kernel $k(u, v)$ by:

$$l(u) = e(u) + \int l(v) \boxed{k(u, v) dv}$$

Aside: Participating Media

- "Air" typically contains participating media (e.g. dust, droplets, smoke, etc.)
- L should be defined over all of 3D space
- The incoming light should be considered in a sphere centered around each point in 3D space

- Neglecting this assumes that "air" is a vacuum
- This restricts L to surfaces

Discretization (of the integral equation)

- Choose p points, each representing a chunk of surface area (or chunk of volume for participating media), which is a 2D (or 3D) discretization
- For each of the p points: Choose q outgoing directions, each representing a chunk of solid angles of the hemisphere (or sphere), which is a 2D discretization
  - q can vary from surface chunk to surface chunk
- L_o and L_e then each have $p * q$ unknowns, a 4D (or 5D) discretization
  - They can thus be represented by vectors: L and E, each with length $p * q$
- The light transport “kernel” matrix K has size $p * q$ by $p * q$

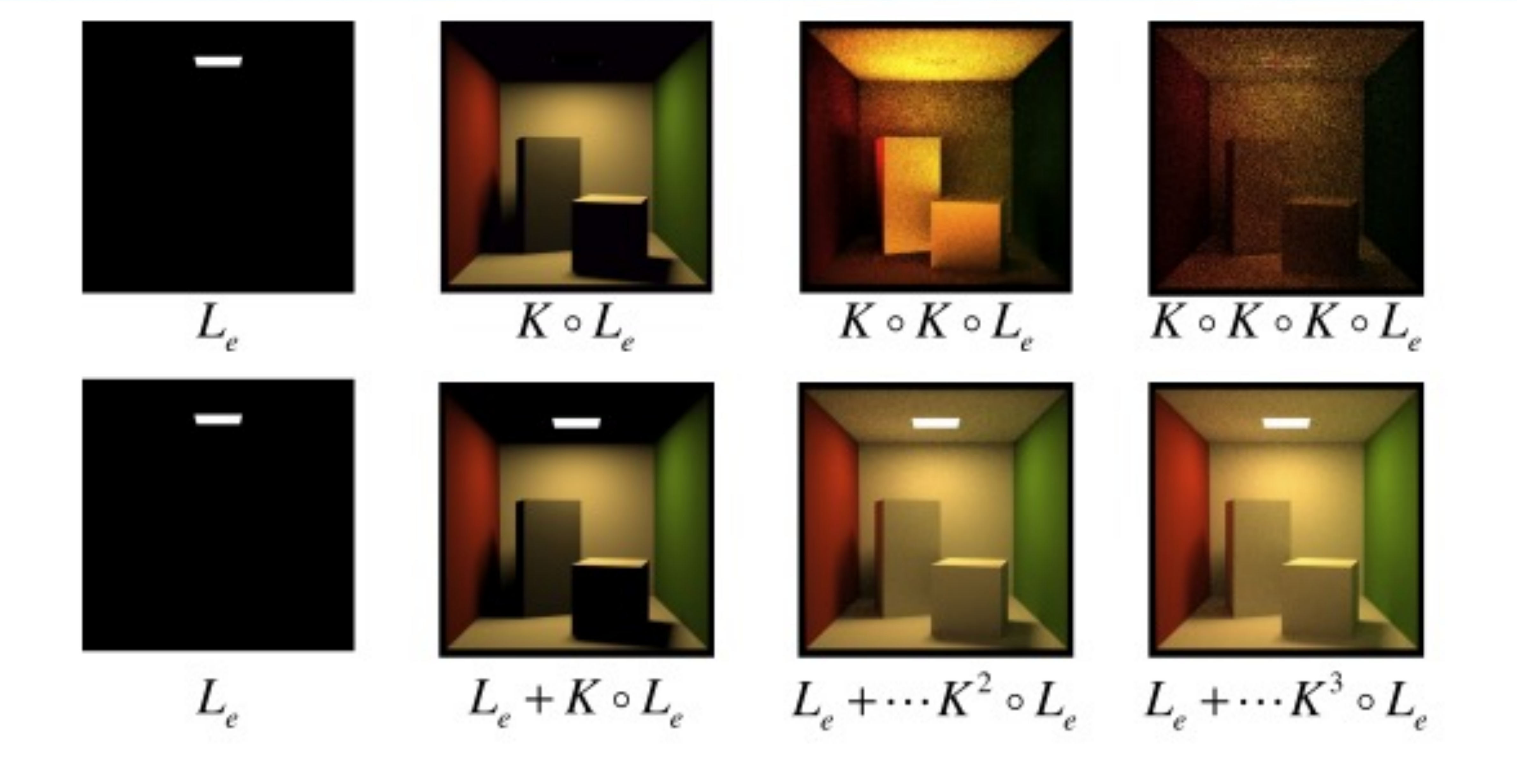
- The linear system of equations is: $L = E + KL$ or $(I - K)L = E$
- Solution: $L = (I - K)^{-1}E = (I + K + K^2 + \cdots)E$
- Since K bounces only a fraction of the light (the rest is absorbed), higher powers are smaller (and the series can be truncated)

Power Series

$$L = E + KE + K^2E + K^3E + \cdots$$

- E: Emission directly from Light Sources
- KE: Direct Illumination (light bounces only once)
- K^2E: Global Illumination (indirect lighting, two bounces)
- K^3E: Global Illumination (indirect lighting, three bounces)
- $\cdots$: Etc.

Power Series



Tractability

- A (typical) scene might warrant thousands or tens of thousands of area chunks
    - So, p could be 1e3, 1e4, 1e5, 1e6, etc.
- Incoming light could vary significantly across the hemisphere
    - So, q might need to be 1e2, 1e3, 1e4, etc.
- L and E would then range in length from 1e5 to 1e10
- The matrix K would then range in size from 1e5 by 1e5 up to 1e10 by 1e10

- K would have between 1e10 and 1e20 entries!

- This tractability analysis is for the 4D problem (5D is even worse)
- The curse of dimensionality makes problems in 4D and 5D (and higher) hard to discretize (with numerical quadrature)

Addressing Tractability

- Idea: separate the diffuse and specular contributions (to be treated separately)

Diffuse:

- Assume all materials are purely diffuse (i.e. no specular contributions)
- Compute the view-independent global illumination for the entire scene
- This can be done in a pre-processing step

Specular:

- Compute (view-dependent) specular illumination on-the-fly as the camera moves
    - Use Phong Shading (or any other model)

Radiosity and Albedo

- Radiosity: power per unit surface area leaving a surface (similar to irradiance, but outgoing instead of incoming):

$$B(x) = \frac{d\Phi}{dA} = \int_{hemi} L_o(x, \omega_o) \cos\theta_o \, d\omega_o$$

- When L_o is independent of ω_o (i.e. purely diffuse):

$$B(x) = \frac{d\Phi}{dA} = L(x) \int_{hemi} \cos\theta_o \, d\omega_o = \pi L(x)$$

- Albedo: a "reflection coefficient" relating incoming light hitting a surface patch (irradiance E_i) to outgoing light emitted in all possible directions

$$\rho(x) = \int_{hemi} BRDF(x, \omega_o, \omega_i) \cos\theta_o \, d\omega_o$$

- When the BRDF is independent of ω_o and ω_i (i.e. purely diffuse):

$$\rho(x) = BRDF(x) \int_{hemi} \cos\theta_o \, d\omega_o = \pi \, BRDF(x)$$

(Purely Diffuse) Lighting Equation

- Given $L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{i\in hemi} L_o(x',-\omega_i) BRDF(x,\omega_i,\omega_o)\cos\theta_i\, d\omega_i$, multiply through by $\cos\theta_o d\omega_o$ and integrate over the hemisphere (i.e. $d\omega_o$) to obtain:

$$B(x) = E(x) + \int_{i\in hemi} B(x') BRDF(x,\omega_i,\omega_o)\cos\theta_i\, d\omega_i$$

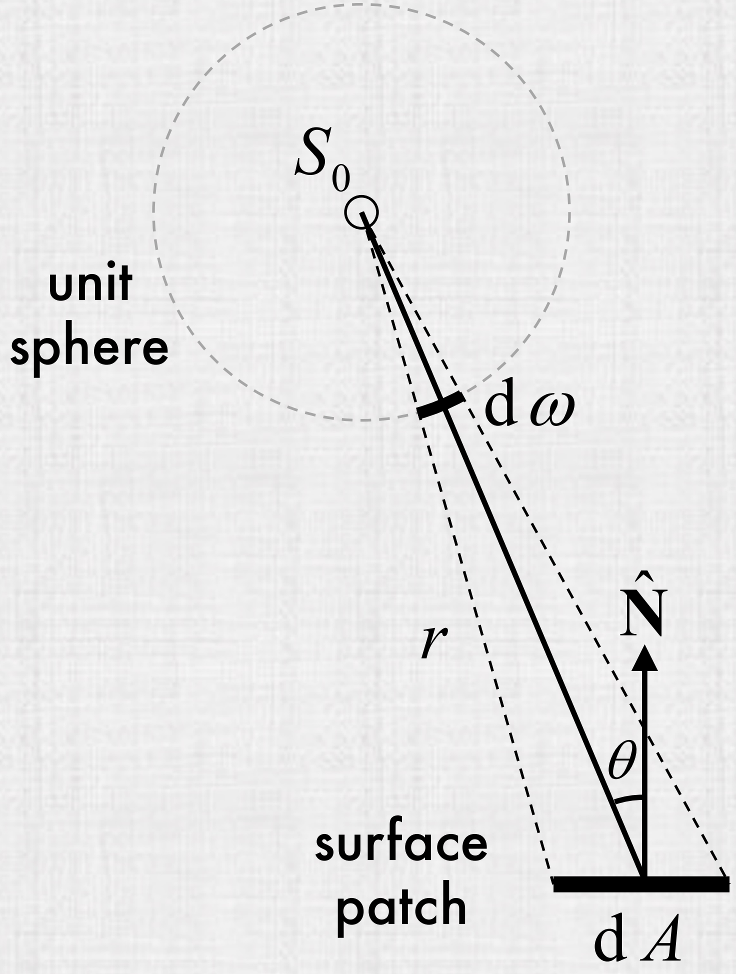
- B is a 2D function (of x), whereas L was a 4D function (of x and ω_o)

- Then, assume that all surfaces have a diffuse BRDF independent of angle:

$$B(x) = E(x) + \frac{\rho(x)}{\pi}\int_{i\in hemi} B(x')\cos\theta_i\, d\omega_i$$

Recall: Solid Angle vs. Cross-Sectional Area

- The (orthogonal) cross-sectional area is $dA\ cos\theta$
- So, $d\omega = \frac{dA_{sphere}}{r^2} = \frac{dA\ cos\theta}{r^2}$ (solid angle varies with tilting θ and distance r)



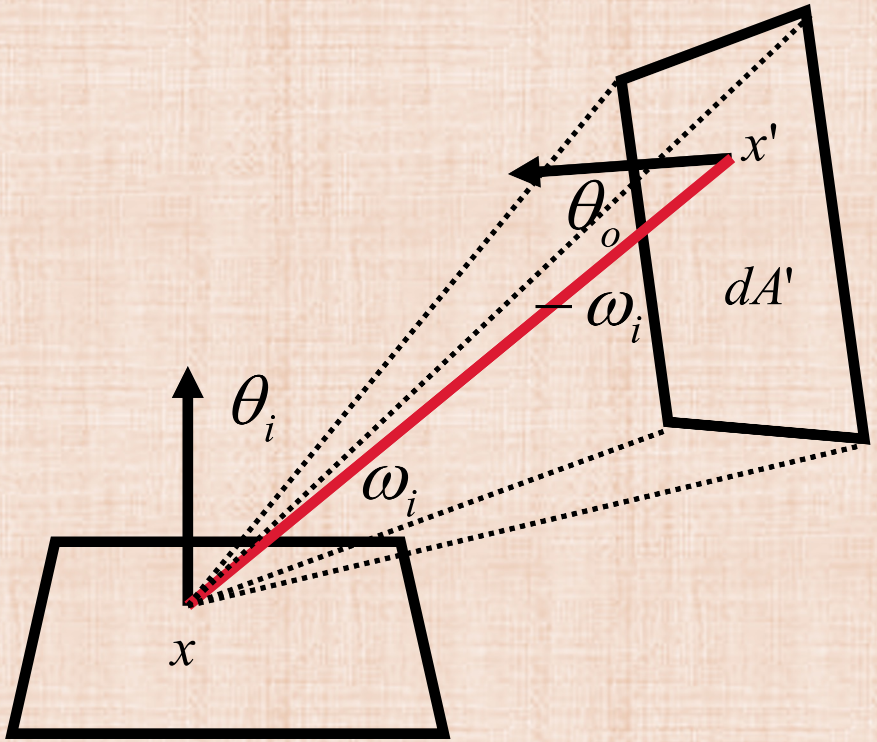
Interchange Solid Angle and Surface Area

- Note: $d\omega = \frac{dA\, cos\theta}{r^2}$ gives $d\omega_i = \frac{dA'\, cos\theta_o}{\|x-x'\|_2^2}$
- So, $B(x) = E(x) + \frac{\rho(x)}{\pi}\int_{i \in hemi} B(x') \cos\theta_i \, d\omega_i$ is:

$$B(x) = E(x) + \rho(x)\int_{i \in hemi} B(x')\frac{\cos\theta_i \cos\theta_o}{\pi\|x - x'\|_2^2} dA'$$

- Let $V(x, x') = 1$ when x and x' are mutually visible (and $V(x, x') = 0$ otherwise), then:

$$B(x) = E(x) + \rho(x)\int_{all\ x'} B(x')V(x, x')\frac{\cos\theta_i \cos\theta_o}{\pi\|x - x'\|_2^2} dA'$$



A Tractable Discretization

- Choose p points, each representing a chunk of surface area (a 2D discretization)
- Then $B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$ with a purely geometric $F_{ij} = V(x_i, x_j) \frac{\cos\theta_i \cos\theta_j}{\pi \|x_i - x_j\|_2^2} A_j$
- Rearrange to $B_i - \rho_i \sum_{j \neq i} B_j F_{ij} = E_i$ and put into matrix form:

$$\begin{pmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1p} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_p F_{p1} & -\rho_p F_{p2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_p \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_p \end{pmatrix}$$

- For p ranging from 1e3 to 1e6: B and E have the same size, and the matrix has 1e6 to 1e12 entries (still large, but 1e4 to 1e8 times smaller than previously)

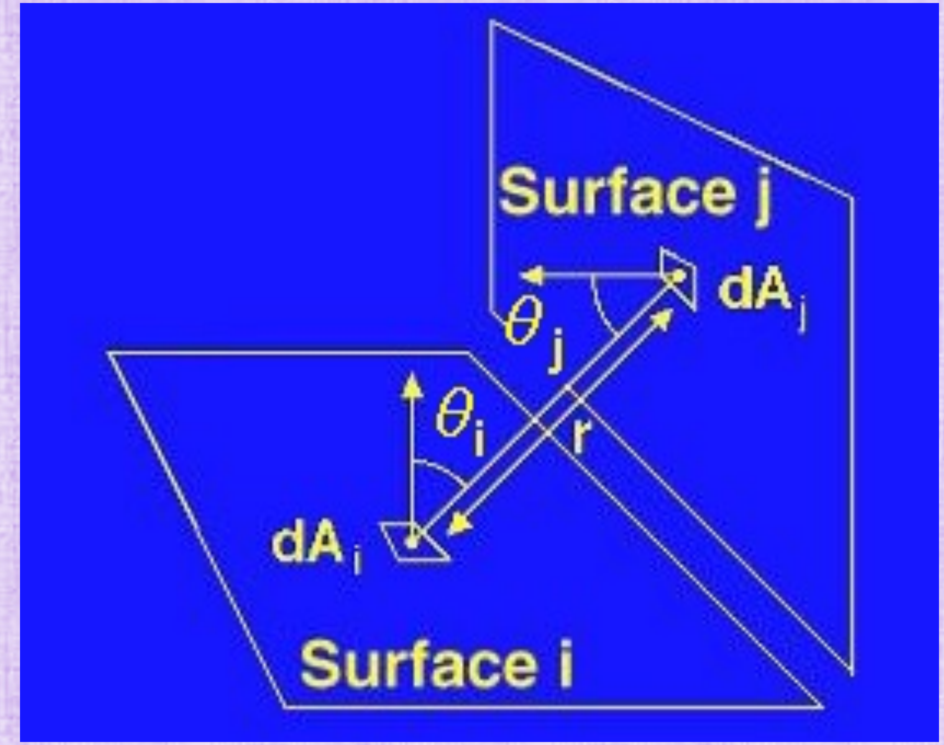
Form Factor

• Write $F_{ij} = V(x_i, x_j)\frac{\hat{F}_{ij}}{A_i}$ and $F_{ji} = V(x_i, x_j)\frac{\hat{F}_{ij}}{A_j}$ with (symmetric) form factor:

$$\hat{F}_{ij} = \frac{\cos\theta_i \cos\theta_j}{\pi\|x_i - x_j\|_2^2} A_i A_j$$

• $\hat{F}_{ij}$ represents how the light energy leaving one surface impacts the other surface, and vice versa (and only depends on the geometry, not on the light)

• The visibility between between x_i and x_j, i.e. $V(x_i, x_j)$, also only depends on the geometry (and can be included into $\hat{F}_{ij}$ if desired)



Understanding the Form Factor

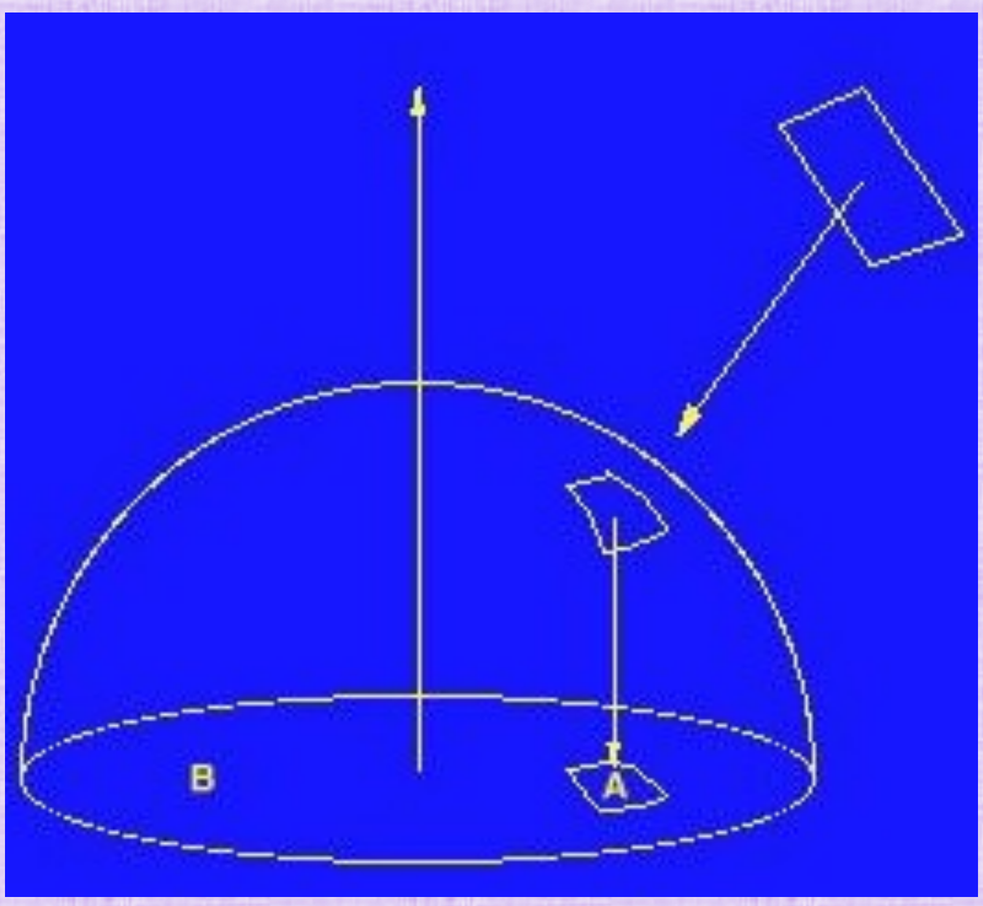
- Place a unit hemisphere at a surface point x_i

- Project the other surface onto the hemisphere, noting that $d\omega = \frac{dA\, cos\theta}{r^2}$ gives $\frac{A_j \cos\theta_j}{\|x_i - x_j\|_2^2}$ as the result

- Project the result downwards onto the circular base of the hemisphere, which multiples by $\cos\theta_i$
    - Recall $\int_{i \in hemi} \cos\theta_i \, d\omega_i = \pi$, the area of the unit circle

- Divide the result by the total area π to get the fraction of the circle occupied

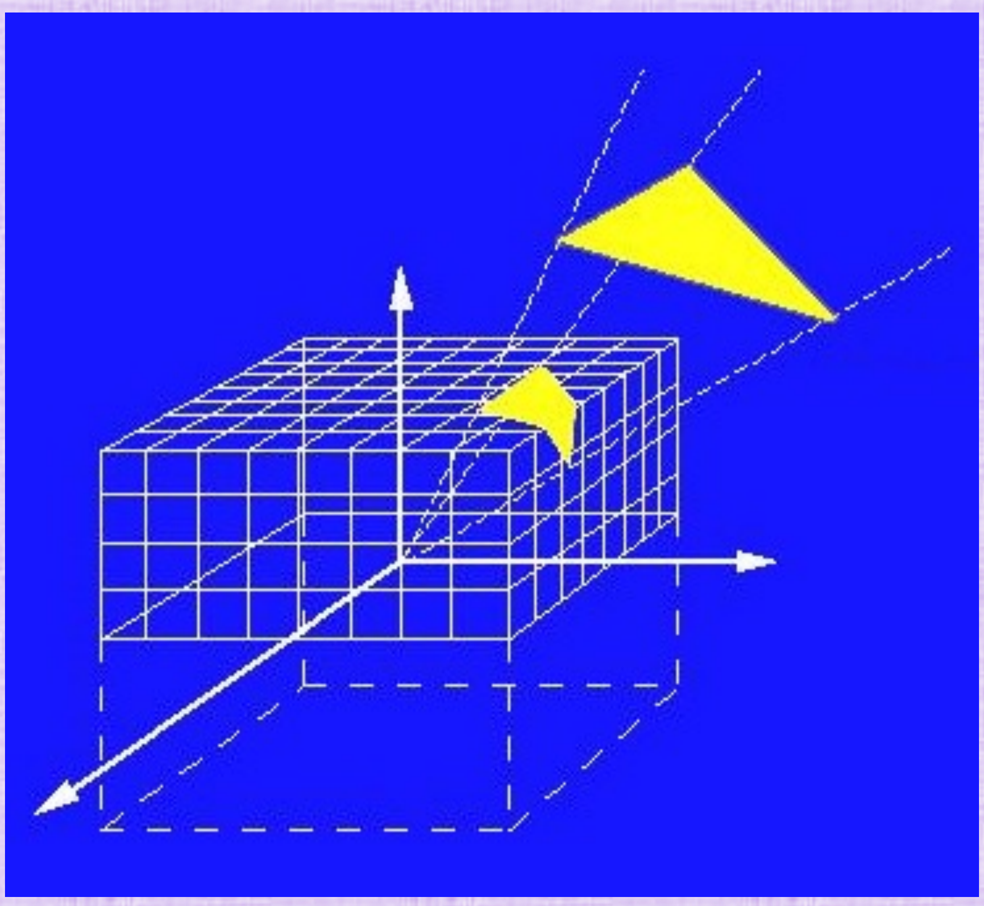
- Overall, this gives: $F_{ij} = \frac{\cos\theta_i \cos\theta_j}{\pi \|x_i - x_j\|_2^2} A_j$



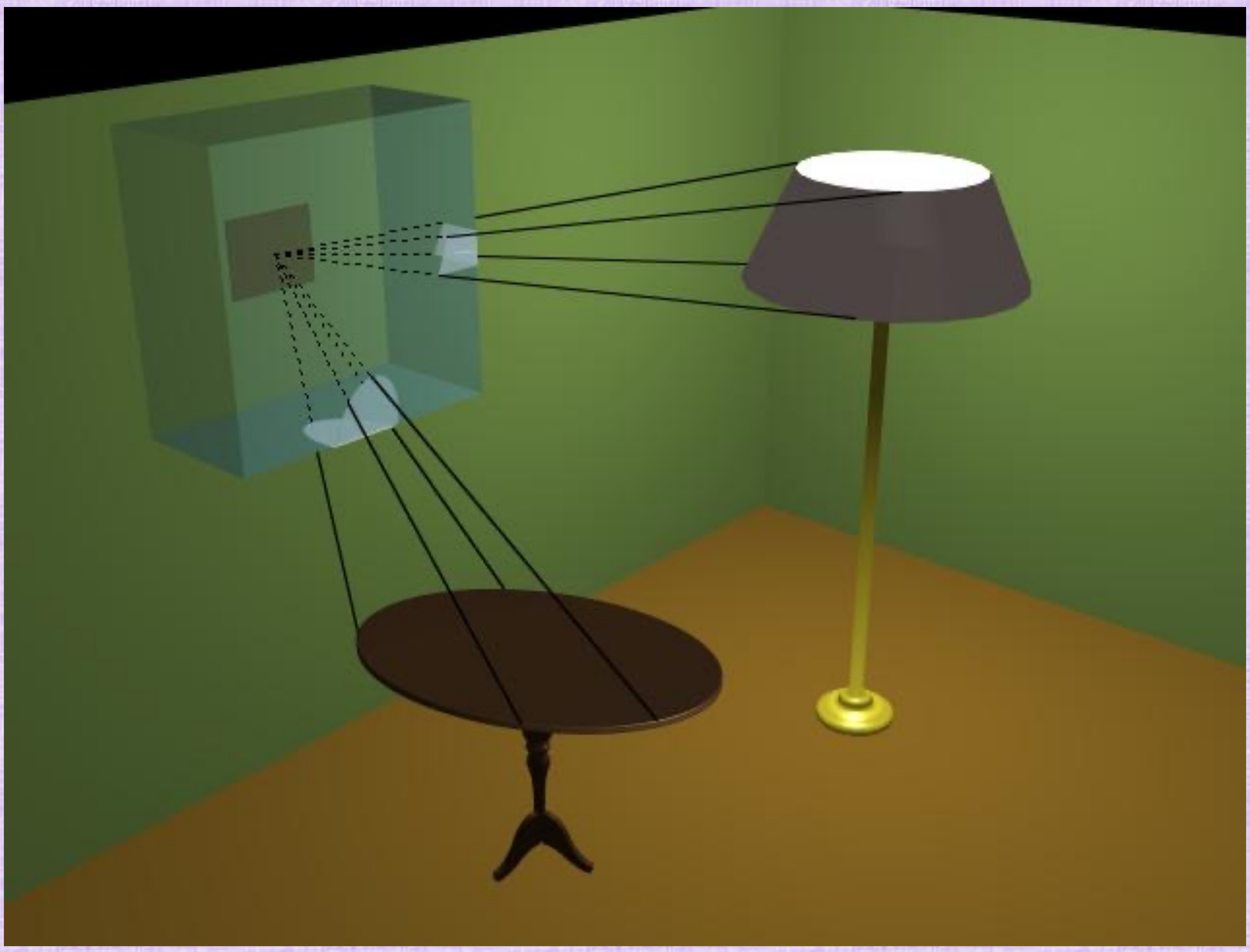
Implementation

- Create a hemicube, and divide each face into sub-squares (as small as desired)
- For each sub-square, use hemisphere projection (from the last slide) to pre-compute its contribution to F_{ij}

- Place the hemicube at a surface point x_i
- A surface patch (from another object) is projected onto the hemicube in order to approximate F_{ij} (using the pre-computed values for the sub-squares)
- The five hemicube faces can be treated as image planes and the sub-squares as pixels, making this equivalent to <u>scanline rasterization</u>
- The <u>depth buffer</u> can be used to detect occlusions, which are used the visibility term



Hemicube Scanline Rasterization



Iterative Solvers

- For large matrices, iterative solvers are typically far more accurate than direct methods (that compute an inverse)
- Iterative methods start with an initial guess, and subsequently iteratively improve it

- Consider $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 8 \\ 10 \end{pmatrix}$ with exact solution $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$
- Start with an initial guess of $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

- Jacobi iteration (solve both equations using the current guess):
  - $x^{new} = \frac{8-y^{old}}{2}$ and $y^{new} = \frac{10-x^{old}}{2}$

- Gauss Seidal iteration (always use the most up to date values):
  - $x^{current} = \frac{8-y^{current}}{2}$ and $y^{current} = \frac{10-x^{current}}{2}$

Jacobi vs. Gauss-Seidal

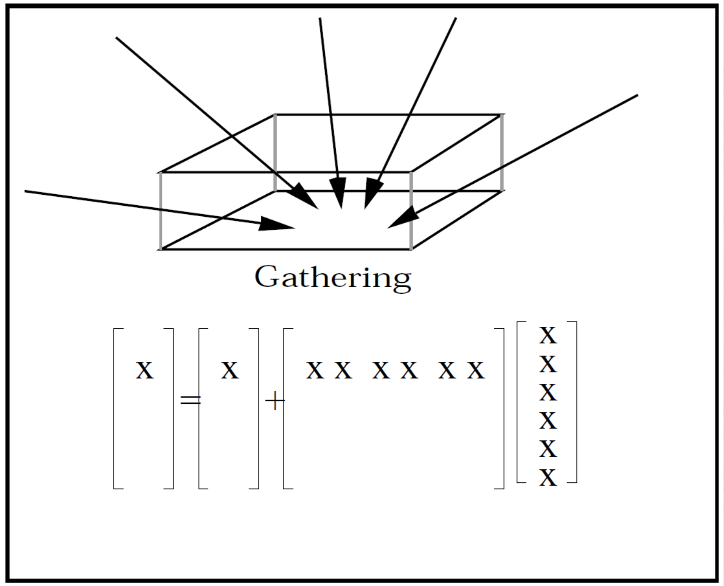
Iteration	Jacobi		Gauss Seidel	
	x	y	x	y
1	0	0	0	0
2	4	5	4	3
3	1.5	3	2.5	3.75
4	2.5	4.25	2.125	3.9375
5	1.875	3.75	2.03125	3.984375
6	2.125	4.0625	2.007813	3.996094
7	1.96875	3.9375	2.001953	3.999023
8	2.03125	4.015625	2.000488	3.999756
9	1.9921875	3.984375	2.000122	3.999939
10	2.0078125	4.00390625	2.000031	3.999985
11	1.998046875	3.99609375	2.000008	3.999996
12	2.001953125	4.000976563	2.000002	3.999999
13	1.999511719	3.999023438	2	4
14	2.000488281	4.000244141	2	4
15	1.99987793	3.999755859	2	4
16	2.00012207	4.000061035	2	4
17	1.999969482	3.999938965	2	4
18	2.000030518	4.000015259	2	4
19	1.999992371	3.999984741	2	4
20	2.000007629	4.000003815	2	4

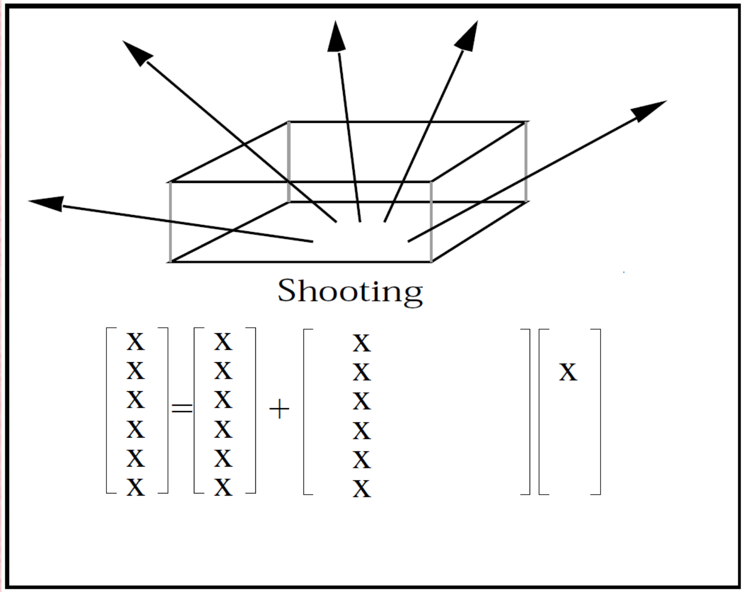
Better Initial Guess

Iteration	Jacobi		Gauss Seidal	
	x	y	x	y
1	2	3	2	3
2	2.5	4	2.5	3.75
3	2	3.75	2.125	3.9375
4	2.125	4	2.03125	3.984375
5	2	3.9375	2.007813	3.996094
6	2.03125	4	2.001953	3.999023
7	2	3.984375	2.000488	3.999756
8	2.0078125	4	2.000122	3.999939
9	2	3.99609375	2.000031	3.999985
10	2.001953125	4	2.000008	3.999996
11	2	3.999023438	2.000002	3.999999
12	2.000488281	4	2	4
13	2	3.999755859	2	4
14	2.00012207	4	2	4
15	2	3.999938965	2	4
16	2.000030518	4	2	4
17	2	3.999984741	2	4
18	2.000007629	4	2	4
19	2	3.999996185	2	4
20	2.000001907	4	2	4

Iterative Radiosity

- Gathering - update one surface by collecting light energy from all surfaces
- Shooting - update all surfaces by distributing light energy from one surface
- Sorting and Shooting - choose the surface with the greatest un-shot light energy and use shooting to distribute it to other surfaces
    - start by shooting light energy out of the lights onto objects (the brightest light goes first)
    - then the object that would reflect the most light goes next, etc.
- Sorting and Shooting with Ambient - start with an initial guess for ambient lighting and do sorting and shooting afterwards





Iterative Radiosity