Global Illumination

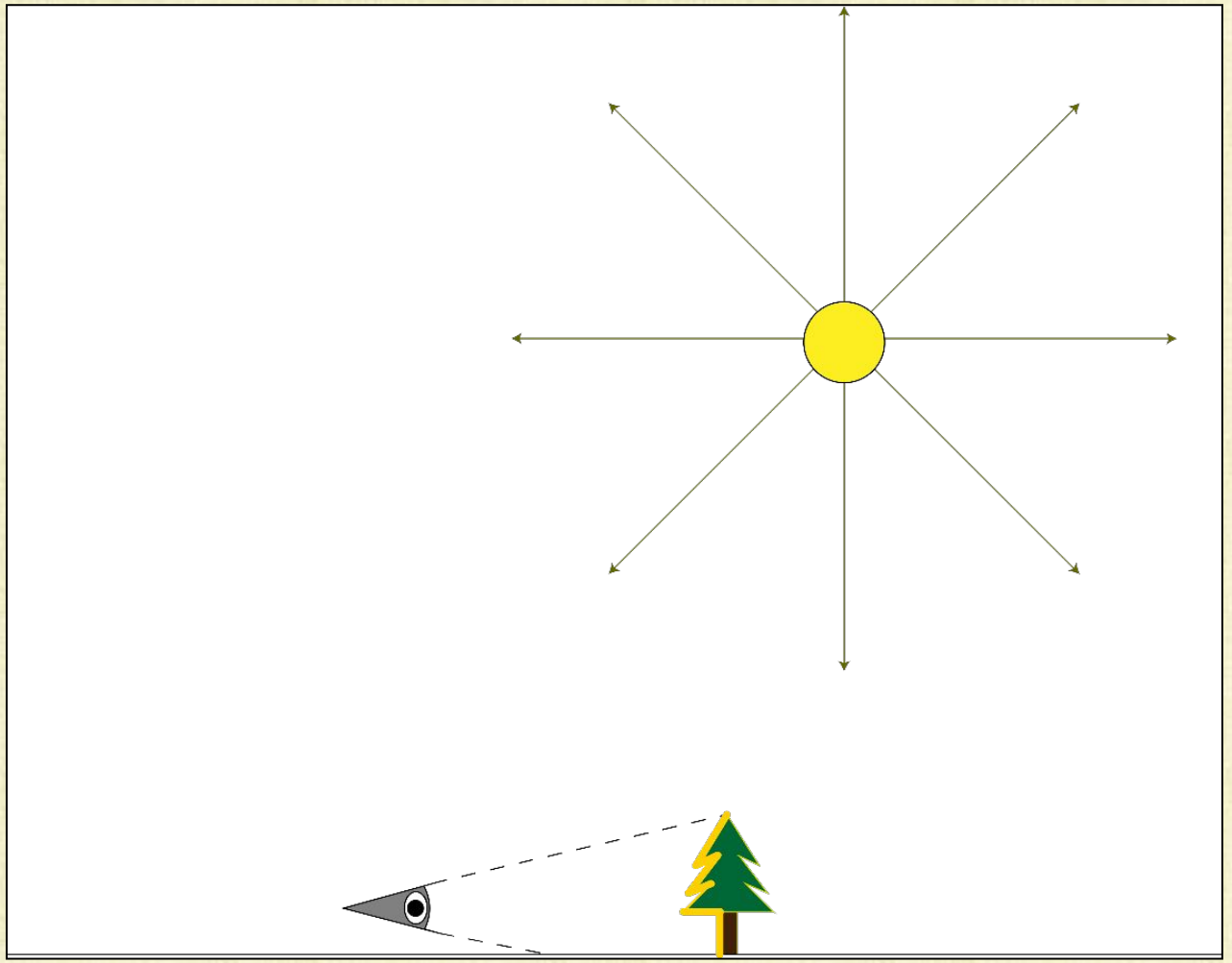
Capturing Realistic Lighting

Following Photons (Photon Tracing)

- For each light, choose a number of directions to sample the outgoing hemisphere (or sphere); emit a photon in each direction
- Photon travels in a straight line, until it intersects an object
- Then, check:
  - Absorbed: terminate photon (does not make it to the camera's film)
  - Reflected/Transmitted/Scattered (by the BRDF): send photon off in a new direction
- Follow photon around the scene, until it's absorbed
- Photons that go through the camera aperture and hit the film activate pixel sensors and contribute to the final image
- N.B. very few photons ever hit the film (most of the light in the world is not seen by your camera/eyes!)

Following Photons (Photon Tracing)

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

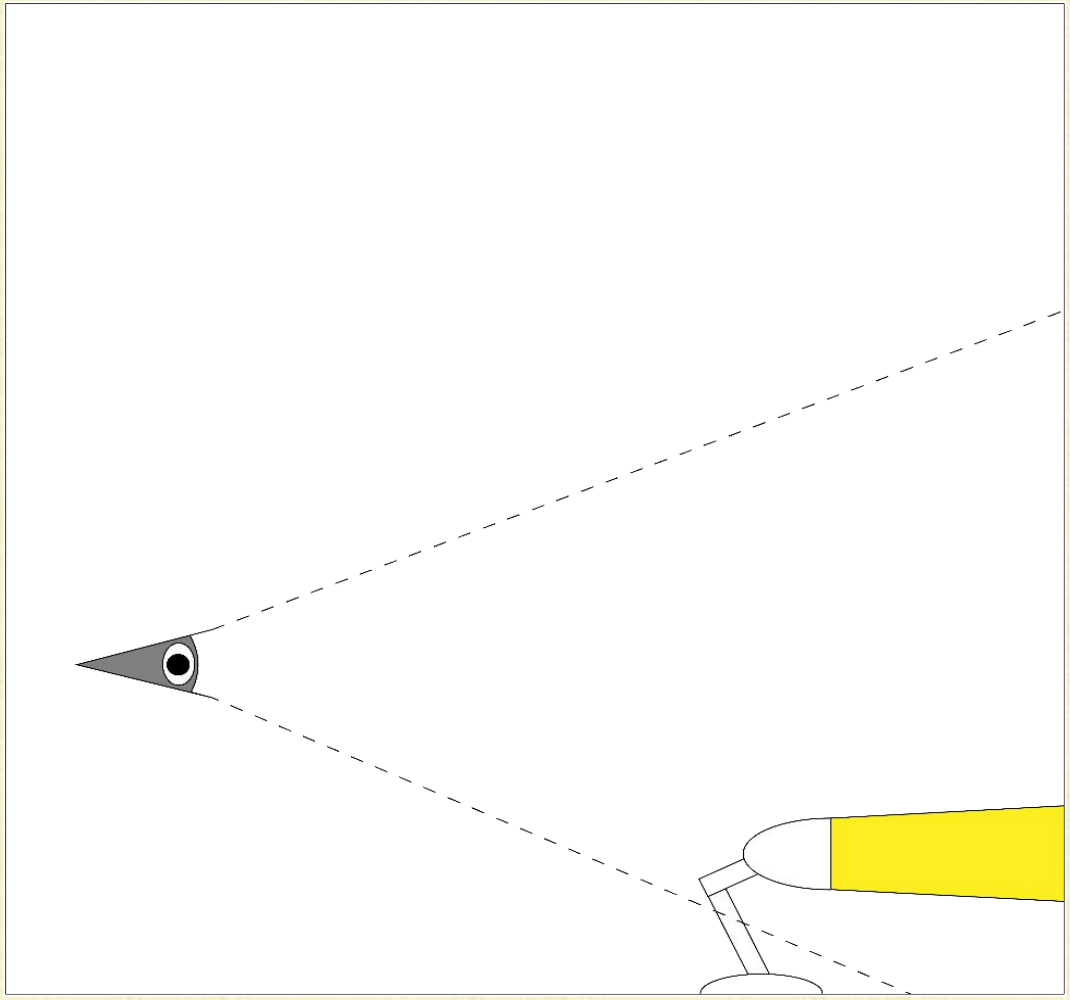


Path Tracing

- For each pixel, send a ray through the aperture to backward trace a photon path that would hit the pixel (same as ray tracing)
- If the ray hits a light source, use outgoing light in the ray direction to determine how many photons hit the pixel
- If the ray hits an object, send out reflected/transmitted rays (same as ray tracing)
- Following all rays until they hit a light source
- A terminated ray only gives a path from the pixel to a light source
  - Need to emit photons along this (path) direction, bounce them off objects along the path, and check to see how many are absorbed before they hit the film
  - Some percentage of the photons are absorbed resulting in a specific color/brightness of light hitting the pixel (along that path)

Path Tracing

- Most of the scene cannot see the light (inefficient)



Path Tracing with BRDFs

- Most paths bounce around, go out a window, and then continue off to outer space (never finding a light source)
  - This is because reflected/transmitted ray directions do not adequately represent the actual backwards path of incoming photons
- Photons bombard a point on a surface from every direction of the hemisphere
- Need to cast rays in all directions of the hemisphere in order to backwards trace all the photons incoming from every direction
- But, spawning many rays at every intersection point is expensive/impractical
- Every spawned ray that hits another surface spawns an entirely new hemisphere of rays of its own (exponential growth)

Ray Tracing is an Optimization (of BRDF Path Tracing)

- Throws out most of the incoming directions on the hemisphere, only keeping those deemed to be most important
- Rays incoming directly from the light source have a lot more photons than other directions (of the hemisphere)
  - Thus, shadow rays are used to track this incoming light
  - Ignoring all other directions is called direct illumination, as only light directly from light sources is considered (recall the need for ambient lighting in this case)
- Reflective objects have a lot more photons from the mirror reflection direction (than from other directions) bouncing to go along the path in question
  - Thus, incoming light from the mirror reflection direction is accounted for
- Transparent objects have a lot more photons from the transmitted ray direction (than from other directions) bouncing to go along the path in question
  - Thus, incoming light from the transmitted ray direction is accounted for

Bidirectional Ray Tracing

- Combine Photon Tracing (following photons) and Path Tracing
- First, emit photons from the light, and bathe objects in those photons recording their lighting information in a light map
  - Allows photons to bounce around the scene illuminating shadows, bleeding color, etc.
- Second, ray trace the scene, while using the light maps to estimate indirect light coming in from all the directions of the hemisphere
- IMPORTANT: Still need to sample the most important directions on the hemisphere explicitly (for increased accuracy!):
  - Shadow rays for direct illumination
  - Reflected and transmitted rays

Light Maps

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

Strategy

Evaluate the lighting equation everywhere:

- Disadvantage: Expensive
  - Requires consideration of all incoming/outgoing light at every point in the (relevant) world
  - Requires a lot more work(!) to compute lighting *everywhere* as opposed to only at points visible to the camera
- Advantage: Far more realistic
  - Can track the full path of light, accounting for the multiple bounces in indirect lighting
  - Once this computation is performed, the camera can be placed anywhere in the scene and can use the already (pre-)computed light maps

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 more realistic lighting, we bounce light all around the scene, and it is tedious to convert between irradiance and radiance, so we use $dE = Ld\omega \cos\theta$ to obtain:

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

- The outgoing radiance considering the light coming from all incoming directions is:

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

Lighting Equation

- Explicitly add the angular dependence for $L_i(\omega_i)$
- Valid throughout all of space, so explicitly add dependence on x

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

- Aside: L can be defined over all of 3D space (points in the "air" may contain participating media, e.g. dust particles), and depend on the incoming light in a sphere centered around a point in 3D (as opposed to a hemisphere around a surface point)

- Neglecting participating media (e.g. assuming a vacuum) restricts x to (only) be on surfaces

Treating Surfaces/Lights Similarly

- Add emission $L_e(x, \omega_o)$, so x can be on (the surface of) light sources
- Incoming light in direction ω_i left a point x' in direction $-\omega_i$, so $L_i(x, \omega_i) = L_o(x', -\omega_i)$

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

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

- Computing reflected radiance on a particular surface requires knowing the incoming radiance from <u>all</u> other (relevant) surfaces (and lights), i.e. knowing $L_i(x, \omega_i) = L_o(x', -\omega_i)$
- But the incoming radiance from all those other surfaces (typically) depends on the outgoing radiance from the surface under consideration (an implicit equation)

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

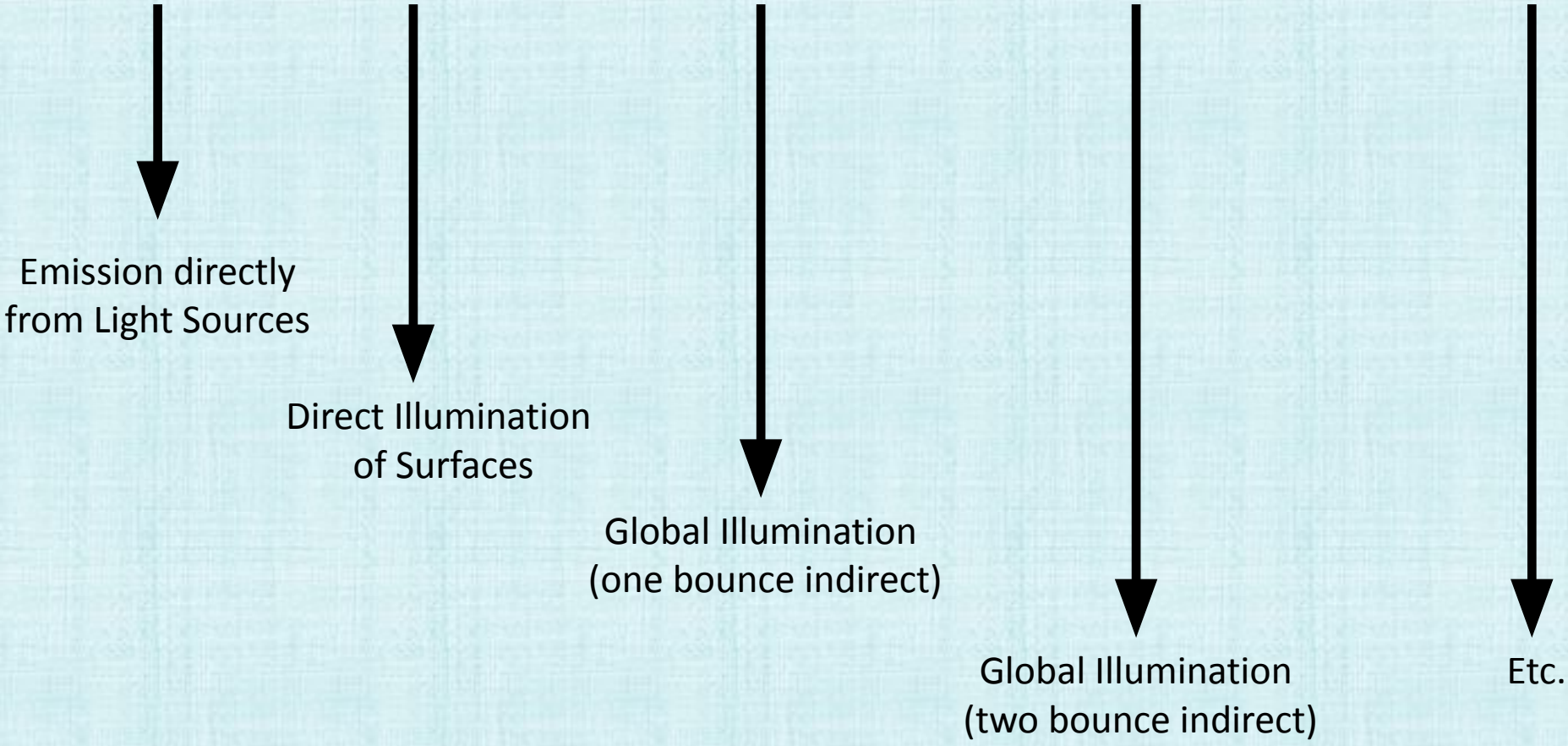
$$l(u) = e(u) + \int l(v) k(u, v) dv$$

Discretization

- Choose p points, each representing a chunk of surface area (or a 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 subset of solid angles of the hemisphere/sphere), which is a 2D discretization
  - q can vary from surface chunk to surface chunk
- Represent the discretized versions of L_o and L_e by vectors L and E, each having length $p * q$ (a 4D or 5D discretization)
- The light transport matrix K has size $(p * q)$ by $(p * q)$
- System of linear equations: $L = E + KL$ or $(I - K)L = E$
- Solution: $L = (I - K)^{-1}E$
- Using the Binomial Theorem: $L = (I + K + K^2 + \cdots)E$
- Typically valid to truncate this series, since higher order terms vanish with each application of K bouncing only a fraction of the light (the rest is absorbed)

Power Series

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



Power Series

L_e

$K \circ L_e$

$K \circ K \circ L_e$

$K \circ K \circ K \circ L_e$

L_e

$L_e + K \circ L_e$

$L_e + \cdots K^2 \circ L_e$

$L_e + \cdots K^3 \circ L_e$

Tractability Issues

- One easily might care about thousands or tens of thousands of area chunks, so p could be 1e3, 1e4, 1e5, 1e6, etc.
- There could be a large variance in the light in different directions on the hemisphere, so q could be 1e2, 1e3, 1e4, etc.
- This means L and E can range in length from 1e5 to 1e10
- Then, the matrix K would range in size from 1e5 by 1e5 up to 1e10 by 1e10

- That is, K would have between 1e10 and 1e20 entries

- These tractability numbers for the 4D problem (5D is even worse!) are typical, and are often referred to as the curse of dimensionality

Separating Diffuse and Specular

- First, assume all materials are diffuse
- Then, compute global illumination for the entire scene in a pre-processing step
- This global illumination is view independent, and doesn't change as the camera moves

- Later, compute specular illumination using Phong or another model
- Since specular is view dependent, it has to be updated "on-the-fly" as the camera moves

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. diffuse), the integral reduces to:

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

- 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 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 function of a 2D input variable now, whereas L was a function of a 4D input varible

Albedo

- A "reflection coefficient" relating the incoming light that hits a surface patch (irradiance E_i) to the outgoing light emitted from the surface patch 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 angles (diffuse), the integral reduces to:

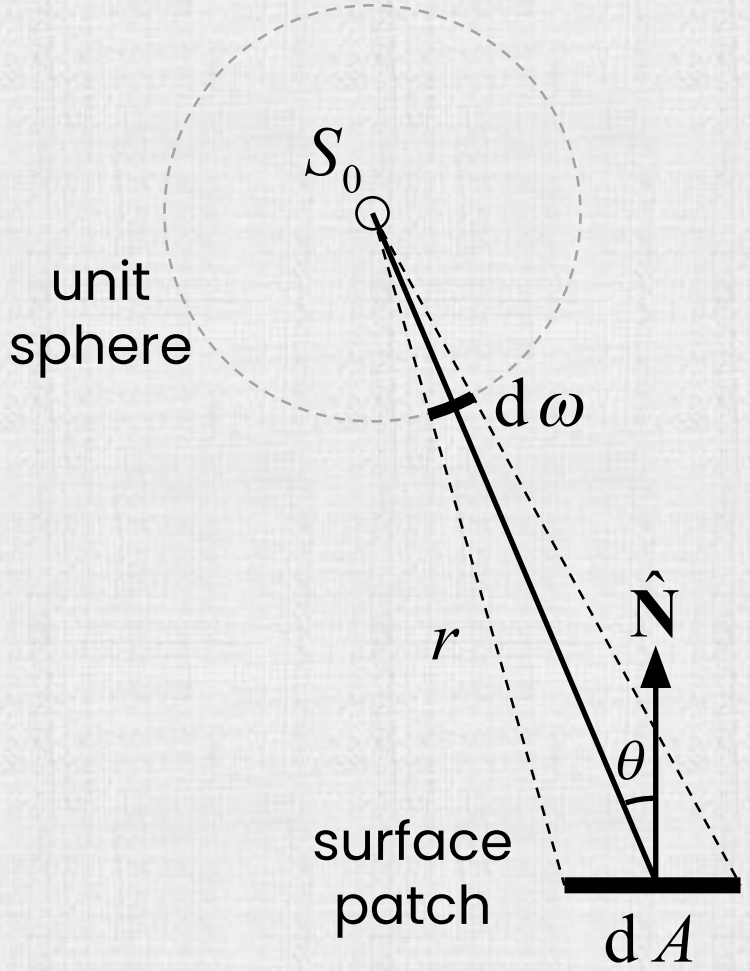
$$\rho(x) = BRDF(x) \int_{hemi} \cos\theta_o \, d\omega_o = \pi \, BRDF(x)$$

- Given $B(x) = E(x) + \int_{i \in hemi} B(x') BRDF(x, \omega_i, \omega_o) \cos\theta_i \, d\omega_i$, assume all surfaces have a diffuse BRDF independent of angle to obtain:

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

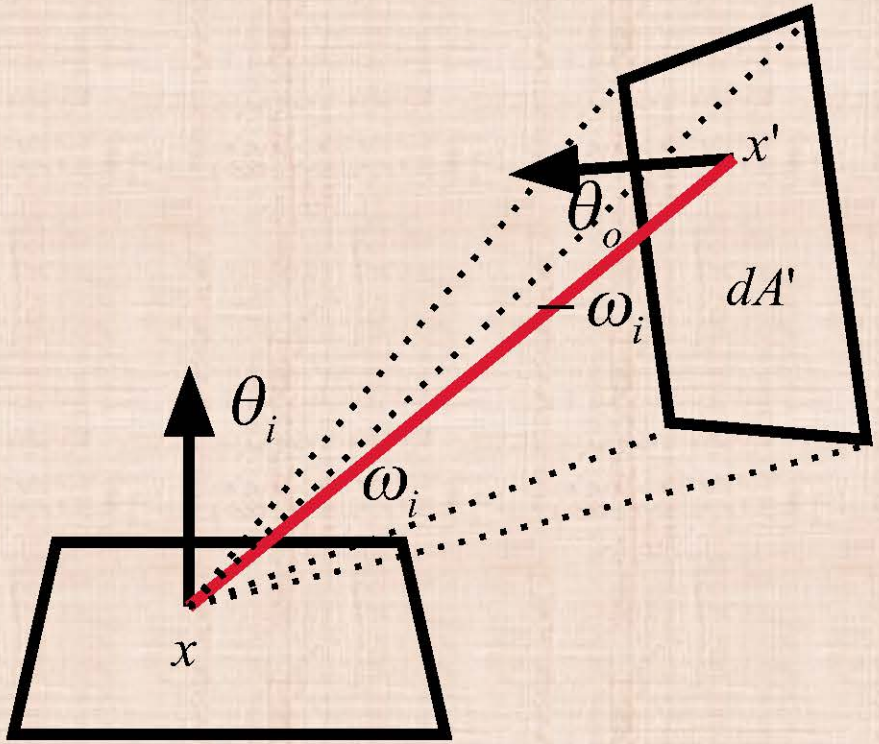
Recall: Solid Angle vs. Area

- The relationship between solid angle and cross-sectional area is $d\omega = \frac{dA_{sphere}}{r^2} = \frac{dA\,cos\theta}{r^2}$, since the area of the orthogonal cross section is $dA\,cos\theta$ (see previous slide)



Interchange Solid Angle and Surface Area

- Consider $d\omega = \frac{dA\cos\theta}{r^2}$ and the figure (to the right) in order to obtain $d\omega_i = \frac{dA'\cos\theta_o}{\|x-x'\|_2^2}$



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

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

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

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

System of Linear Equations

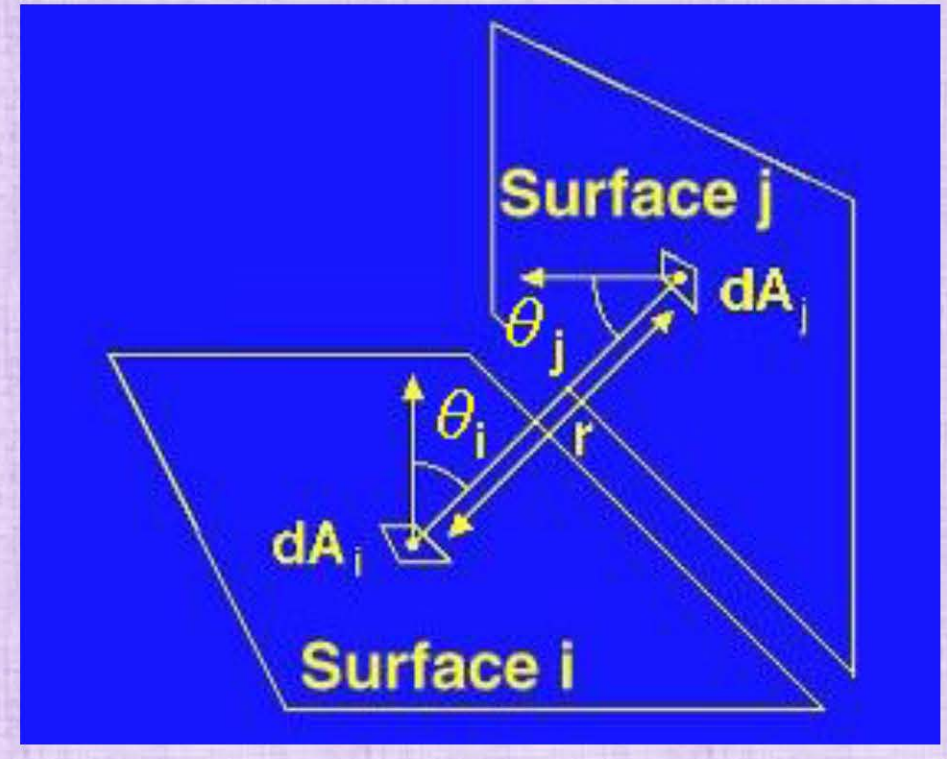
- Choose p points, each representing a chunk of surface area (a tractable 2D discretization)
- Then $B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{i,j}$ with a geometric term $F_{i,j} = V(x_i, x_j) \frac{\cos\theta_i \cos\theta_j}{\pi \|x_i - x_j\|_2^2} A_j$
- Rearranging to $B_i - \rho_i \sum_{j \neq i}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

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

Form Factor

- Calculating the $F_{i,j} = V(x_i, x_j) \frac{\cos\theta_i \cos\theta_j}{\pi \|x_i - x_j\|_2^2} A_j$ requires checking the visibility between x_i and x_j for $V(x_i, x_j)$ as well as computing a symmetric form factor:

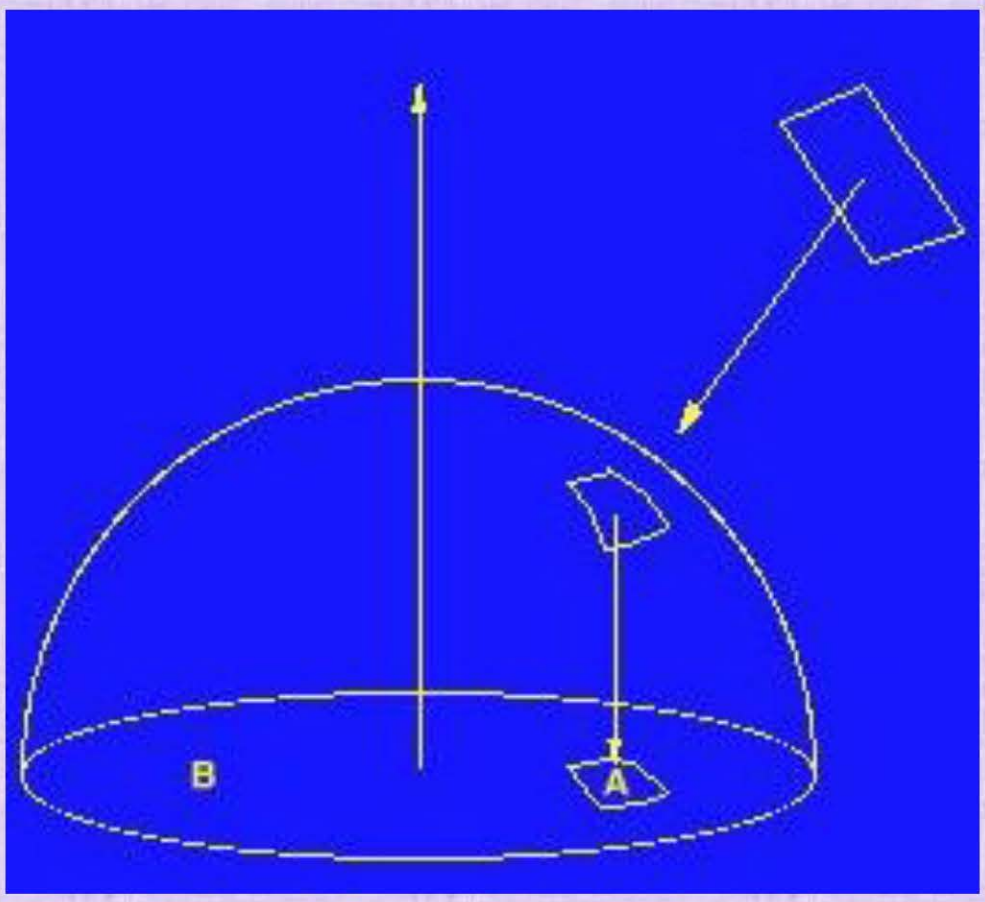
$$\hat{F}_{i,j} = \frac{\cos\theta_i \cos\theta_j}{\pi \|x_i - x_j\|_2^2} A_i A_j$$



- Then $F_{i,j} = V(x_i, x_j) \frac{\hat{F}_{i,j}}{A_i}$ and $F_{j,i} = V(x_i, x_j) \frac{\hat{F}_{i,j}}{A_j}$
- The form factor $\hat{F}_{i,j}$ is the fraction of energy leaving one surface that reaches another, based purely on the geometry between the two surfaces
- note: $V(x_i, x_j)$ can be included in $\hat{F}_{i,j}$ (if desired)

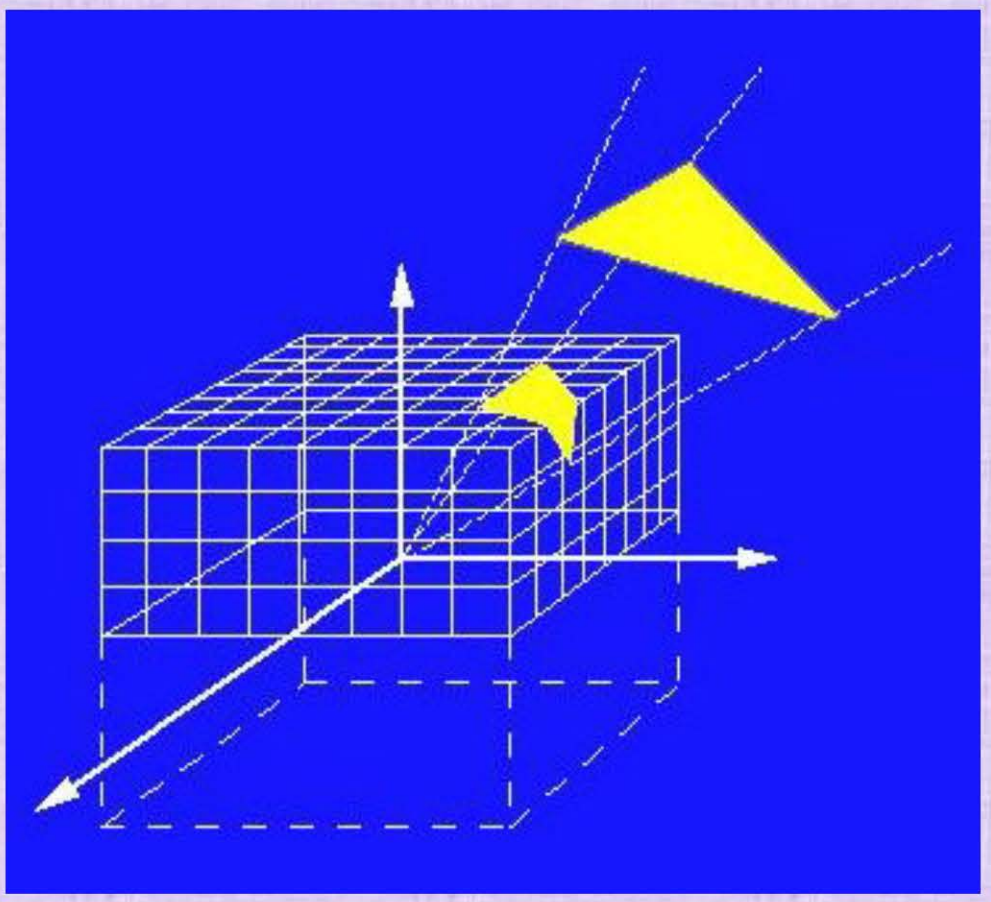
Nusselt Analog

- Place a unit hemisphere at the surface point x_i
- Project the other surface onto the hemisphere, noting that $d\omega = \frac{dA\cos\theta}{r^2}$ gives $\frac{A_j \cos\theta_j}{\|x_i - x_j\|_2^2}$
- Then, cylindrically project the result downwards onto the circular base of the hemisphere, accounting for $\cos\theta_i$ (recall $\int_{i\in hemi} \cos\theta_i \, d\omega_i = \pi$, which is the area of the unit circle)
- The form factor is the fraction of the circle occupied, so divide the result by the total area π
- In summary, this gives $F_{ij} = \frac{\cos\theta_i \cos\theta_j}{\pi\|x_i - x_j\|_2^2} A_j$
- The visibility term can be added by pruning the occluded area when projecting onto the hemisphere

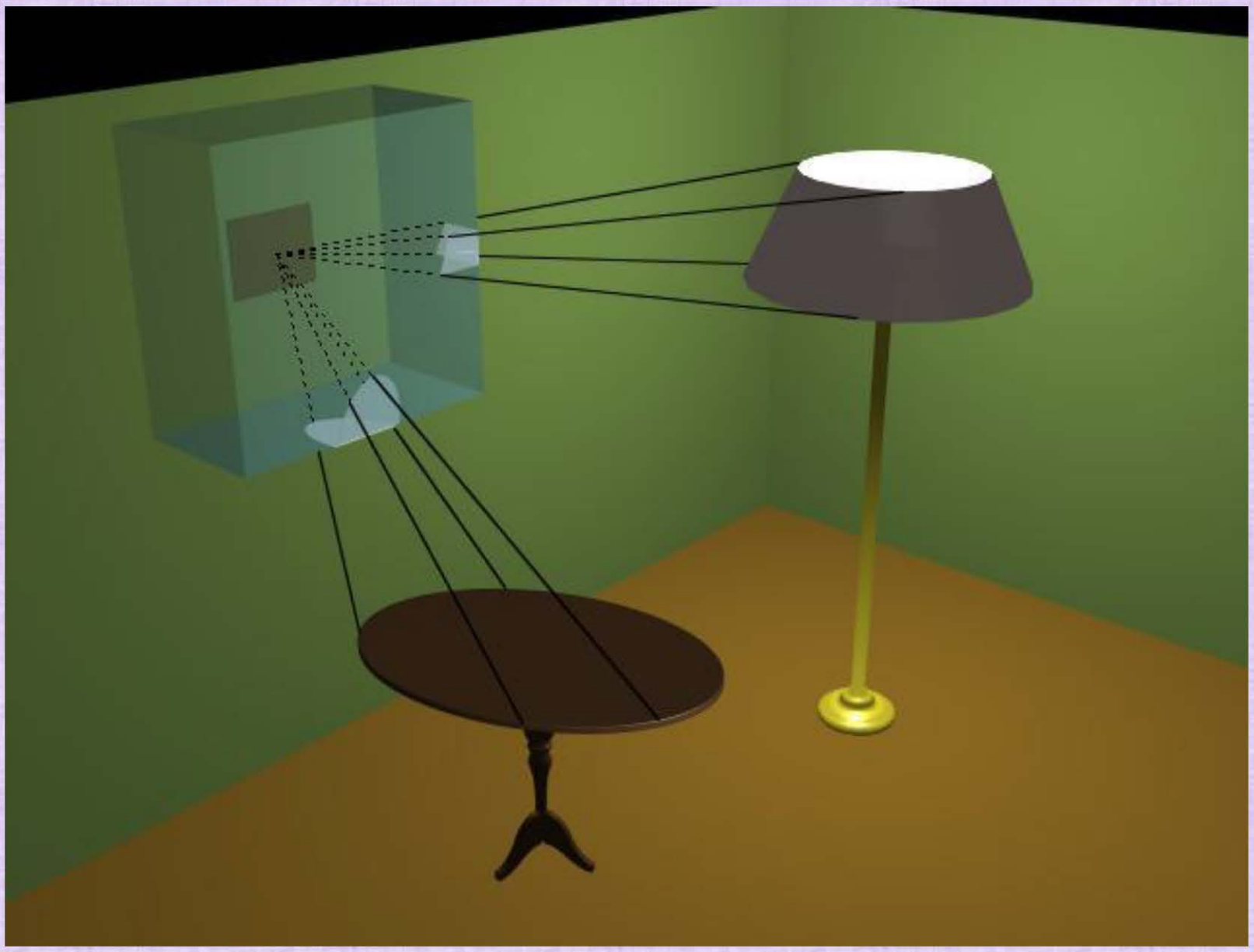


Incorporating Visibility

- Given a point x_i, create a hemicube and divide each face into a number of sub-squares (as small as desired)
- For each sub-square, use the Nusslet analog (hemisphere projection) to pre-compute its contribution to the form factor
- A surface patch (from another object) can be projected onto the hemicube, and its form factor can be approximated using the pre-computed values for the sub-squares
- The hemicube faces can be treated as image planes and the sub-squares as pixels, converting hemicube projection into a scanline rasterization
- The depth buffer can be used to detect occlusions making the visibility term readily computable



Hemicube Scanline Rasterization



Iterative Solvers

- For large matrices, iterative solvers often produce better solutions than direct methods that compute an inverse (and they typically do it more efficiently)
- 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 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 best 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 possible
  - $x^{new} = \frac{8 - y^{current}}{2}$ and $y^{new} = \frac{10 - x^{current}}{2}$

Jacobi & Gauss-Seidal

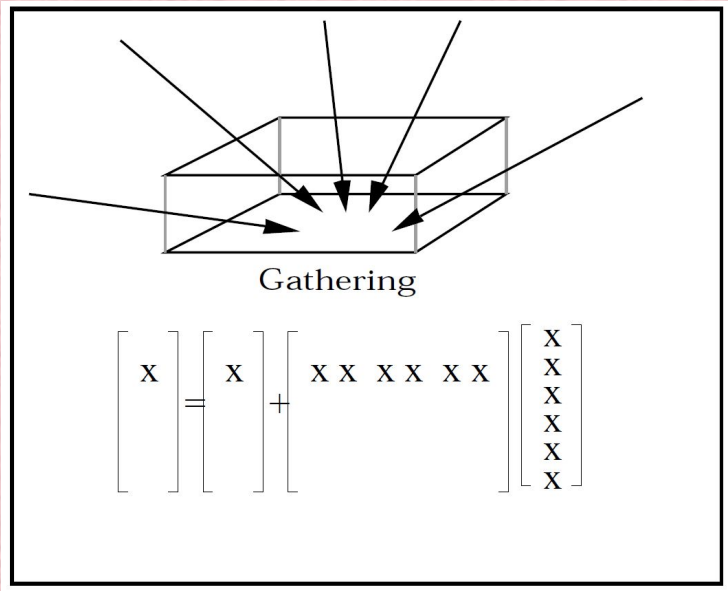
Iteration	Jacobi		Gauss Seidel	
	x	y	x	y
1	0	0	0	0
2	4	5	4	3
3	1.5	3	2.5	3.75
4	2.5	4.25	2.125	3.9375
5	1.875	3.75	2.03125	3.984375
6	2.125	4.0625	2.007813	3.996094
7	1.96875	3.9375	2.001953	3.999023
8	2.03125	4.015625	2.000488	3.999756
9	1.9921875	3.984375	2.000122	3.999939
10	2.0078125	4.00390625	2.000031	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

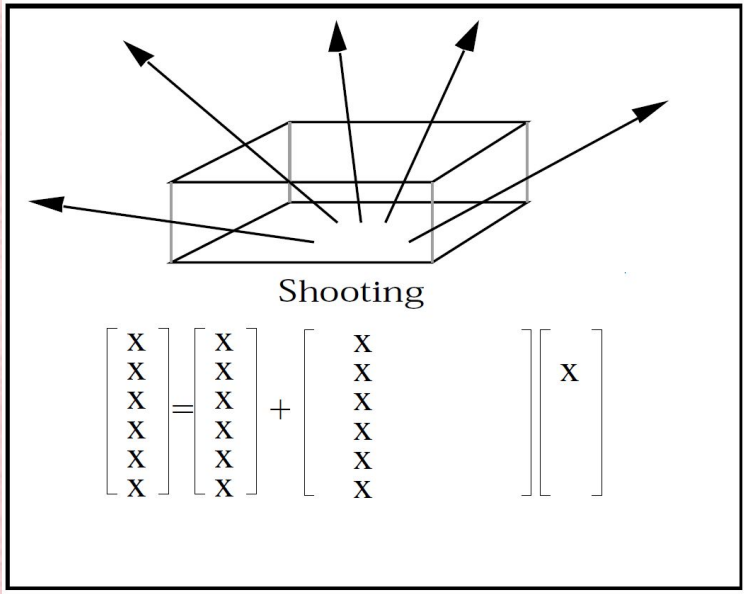
Better Initial Guess

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