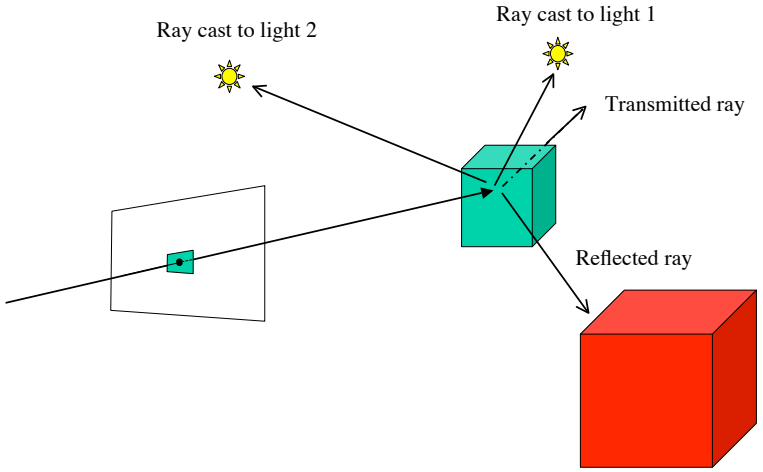
Recursive ray tracing

H&B, page 597



Recursive ray tracing rendering algorithm

- Cast ray from pinhole (projection center) through pixel, determine nearest intersection
- Compute components by casting rays
  - to sources = shadow ray (diffuse and for specular lobe)
  - along reflected direction = reflected ray
  - along transmitted dir = refracted ray
- Determine each component and add them up with contribution from ambient illumination.
- To determine some of the components, the ray tracer must be called **recursively**.

Recursive ray tracing rendering (cont)

- Recursion needs to stop at some point!
- Contributions die down after multiple bounces---there is no such thing as a perfect reflector---so we either set mirror reflections to be less than 100% (even if the user asks for 100%), or simply include an attenuation factor for each new ray.
- Can also model absorption due to light traveling in medium
  - Usually ignored in air, but depends on the application
  - Translucent absorption is exponential in depth

$$I = I_0 e^{-\alpha d}$$

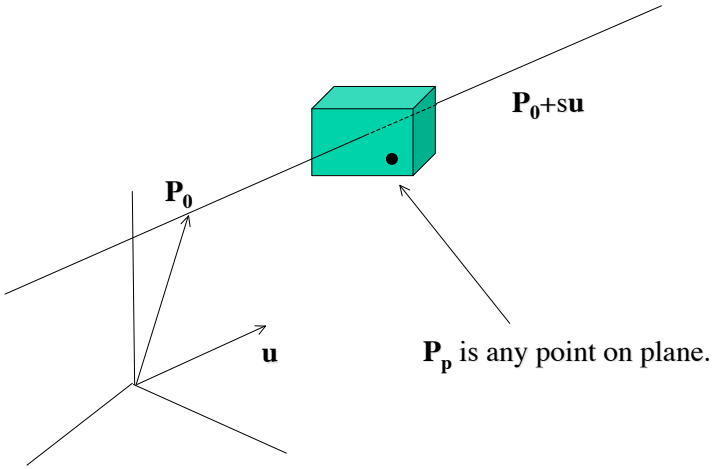
- Recursion is stopped when contributions are too small
  - need to track the cumulative effect
  - common to also limit the depth explicitly

Mechanics

- Primary issue is intersection computations.
  - E.g. sphere, triangle.
- Polygon (should feel familiar!)
- Find point on plane of polygon and then determine if it is inside
  - One way is to make an argument with angles
  - Another way---thinking of the polygon as a surface of a polyhedra---is to check if the point is on the inside side of each of the other planes of the polyhedra.
- Sphere, relatively simple algebra.

Poly details

May be helpful for A6.



Poly details

May be helpful for A6.

To find the intersection of the ray and the plane, solve:

$$\left(\mathbf{P_0} + s\mathbf{u} - \mathbf{P_p}\right) \bullet \mathbf{n} = 0$$

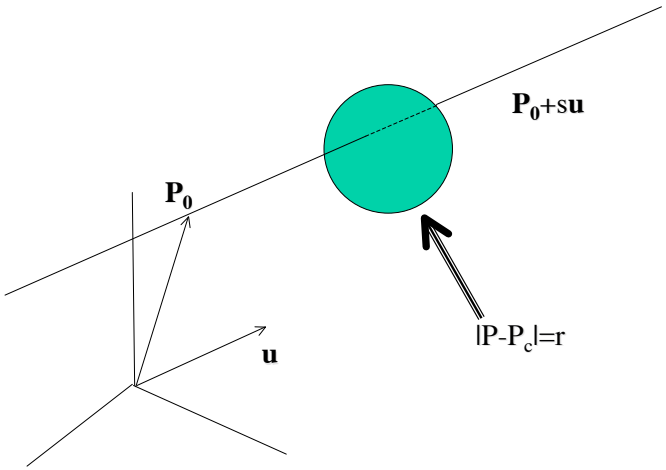
Once you have the point of intersection, $\mathbf{P_i}$, test that it is inside by testing against all other faces.

$$\left(\mathbf{P_i} - \mathbf{P_p}\right) \bullet \mathbf{n} < 0$$

Note that $\mathbf{n}$ and $\mathbf{P_p}$ are now from those *other* faces.

Sphere details (H&B, 602)

May be helpful for grad version of A6.



Sphere details (H&B, 602)

May be helpful for grad version of A6.

$$\left|\mathbf{P_0} + s\mathbf{u} - \mathbf{P_c}\right| = r$$

$$\left|\Delta\mathbf{P} + s\mathbf{u}\right| = r$$

$$\left(\Delta\mathbf{P} + s\mathbf{u}\right) \bullet \left(\Delta\mathbf{P} + s\mathbf{u}\right) = r^2$$

$$\Delta\mathbf{P} \bullet \Delta\mathbf{P} - r^2 + 2s\Delta\mathbf{P} \bullet \mathbf{u} + s^2\mathbf{u} \bullet \mathbf{u} = 0$$

The last expression is easily solved using the quadratic equation. If the discriminant is negative (complex solutions), then the ray does not intersect the sphere.

Sphere details (H&B, 602)

May be helpful for grad version of A6.

Recall that if: $as^2 + bs + c = 0$

The "discriminant" is: $b^2 - 4ac$

The solution is: $s = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

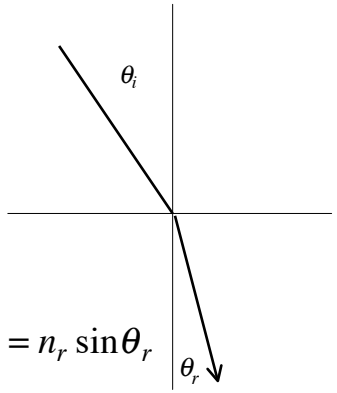
Note that in the book, **u** is a unit vector, so **u•u**=1, thus a=1, and b has a factor of 2 that is removed by dividing by 2a=2, to get equation 10-71.

Refraction Details

Index of refraction, n, is the ratio of speed of light in a vacuum, to speed of light in medium.

Typical values:

air:	1.00 (nearly)
water:	1.33
glass:	1.45-1.6
diamond:	2.2

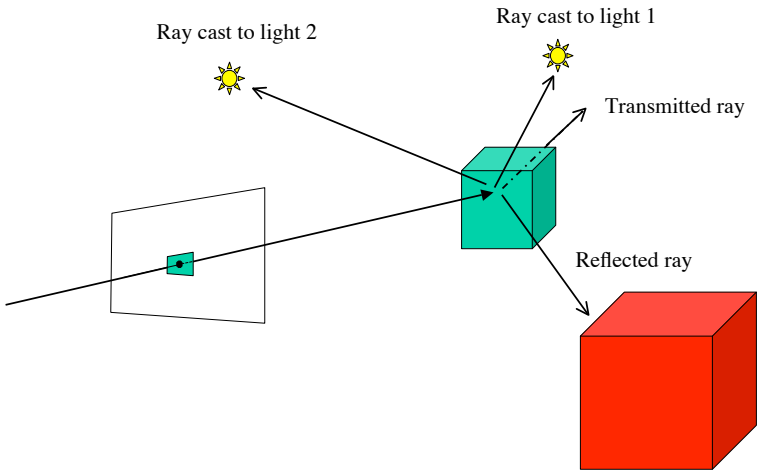


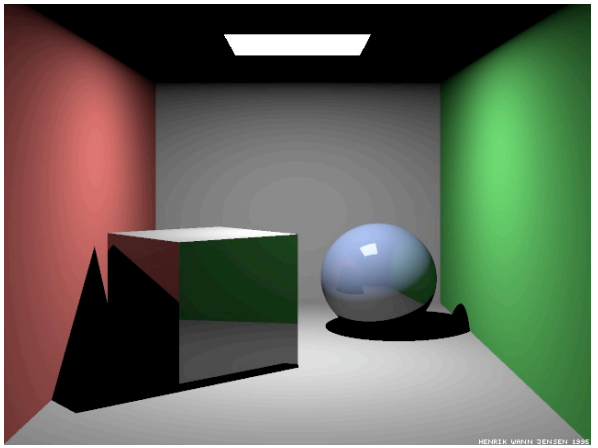
$$n_i \sin\theta_i = n_r \sin\theta_r$$

The indices of refraction for the two media, and the incident angle, θ_i, yield the refracted angle θ_r. (Also need planarity).

Recursive ray tracing

H&B, page 597





Ray-traced Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj

PCKTWTCH by Kevin Odhner, POVRay

6Z4.JPG - A Philco 6Z4 vacuum tube by Steve Anger

Issues

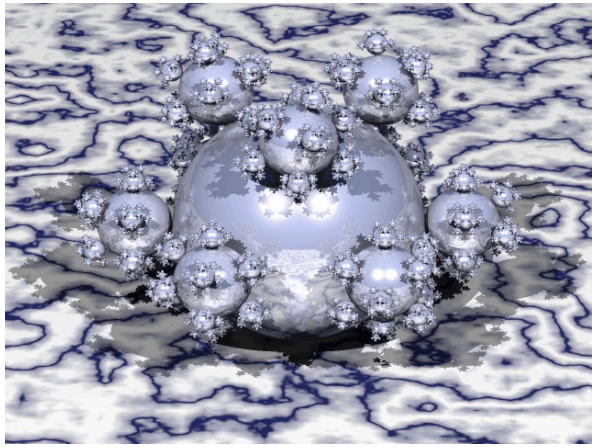
- Sampling (aliasing)
- Very large numbers of objects
  - Need making intersections efficient, exclude as much as possible using clever data structures
- Surface detail
  - bumps, texture, etc.
- Illumination effects
  - Caustics, specular to diffuse transfer
- Camera models

Sampling

- Simplest ray-tracer is one ray per pixel
  - This gives aliasing problems
- Solutions
  - Cast multiple rays per pixel, and use a weighted average
  - Rays can be on a uniform grid
  - It turns out to be better if they are “quite random” in position
    - “hard-core” Poisson model appears to be very good
    - different patterns of rays at each pixel

Efficiency - large numbers of objects

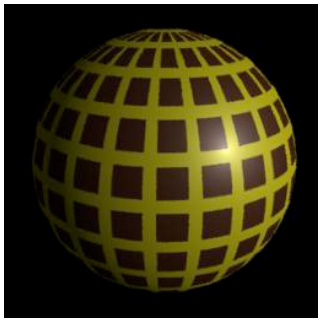
- Construct a space subdivision hierarchy of some form to make it easier to tell which objects a ray might intersect
- Uniform grid
  - easy, but many cells
- Bounding Spheres
  - easy intersections first
- Octtree
  - rather like a grid, but hierarchical
- BSP tree



500,000 spheres, Henrik Jensen, http://www.gk.dtu.dk/~hwj

Surface detail

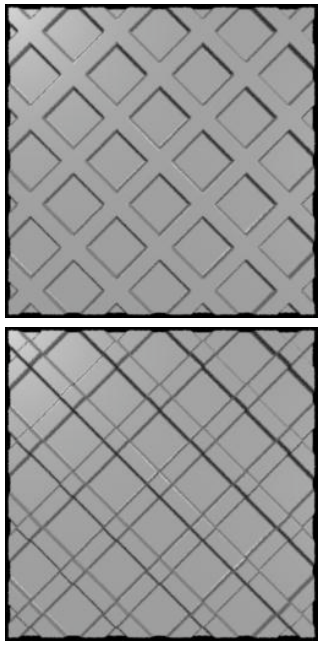
- Knowing the intersection point gives us a position in intrinsic coordinates on the surface
  - e.g. for a triangle, distance from two of three bounding planes
- This is powerful - we could attach some effect at that point
- Texture maps:
  - Make albedo (or color) a function of position in these coordinates
  - Rendering: when intersection is found, compute coordinates and get albedo from a map
  - This is not specific to ray-tracing

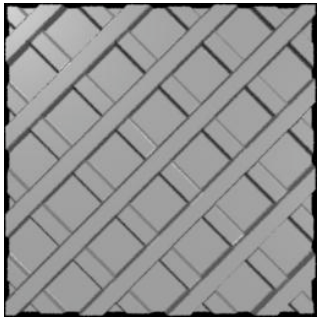


From RmanNotes: http://www.cgrg.ohio-state.edu/~smay/RManNotes/index.html

Surface detail, II

- Bumps
  - we assume that the surface has a set of bumps on it
    - e.g. the pores on an orange
  - these bumps are at a fine scale, so don't really affect the point of intersection, but do affect the normal
  - strategy:
    - obtain normal from "bump function"
    - shade using this modified normal
    - notice that some points on the surface may be entirely dark
    - bump maps might come from pictures (like texture maps)

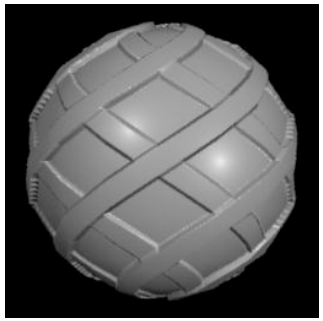




From RmanNotes
http://www.cgrg.ohiostate.edu/~smay/RManNotes/index.html

Surface detail, III

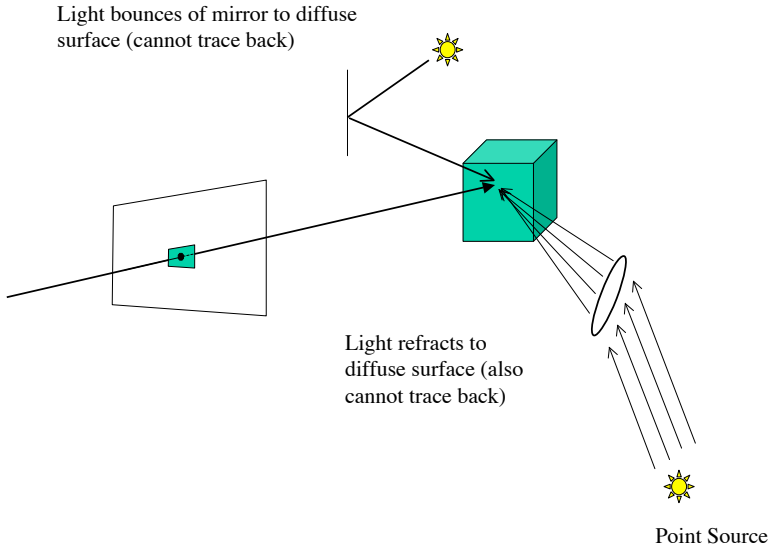
- A more expensive trick is to have a map which includes **displacements** as well
- Must be done **before** visibility



From RmanNotes: http://www.cgrg.ohio-state.edu/~smay/RManNotes/index.html

Illumination effects

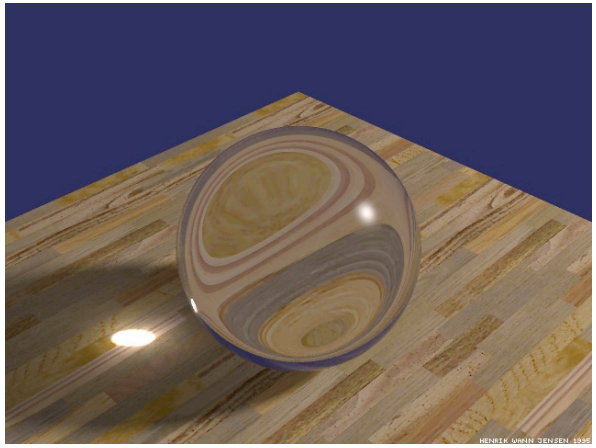
- Caustics:
  - refraction or reflection causes light to be “collected” in some regions.
- Specular-> diffuse transfer
  - source reflected in a mirror
- Can’t render this by tracing rays from the eye - how do they know how to get back to the source?



Illumination effects (cont)

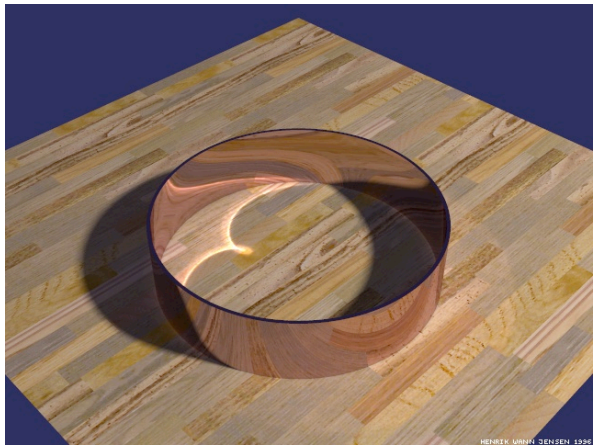
- To get the effect of light reflected and refracted from sources onto diffuse surfaces, we can trace rays **from** the light **to** the first diffuse surface
  - leave a note that illumination has arrived - an illumination map, or photon map
  - sometimes referred to as the forward ray
  - now retrieve this note by tracing eye rays
- Issues
  - efficiency (why trace rays to things that might be invisible?)
  - aliasing (rays are spread out by, say, curved mirrors)

Refraction caustic



Henrik Jensen, http://www.gk.dtu.dk/~hwj

Reflection caustic



Henrik Jensen, http://www.gk.dtu.dk/~hwj

Refraction caustics

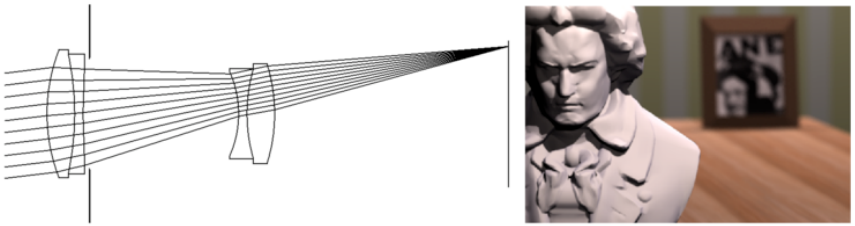
Henrik Jensen, http://www.gk.dtu.dk/~hwj

Lens Effects

Note that a ray tracer very elegantly deals with the projection geometry that we struggled with in earlier lectures which was based on a very simple and “ideal” camera model

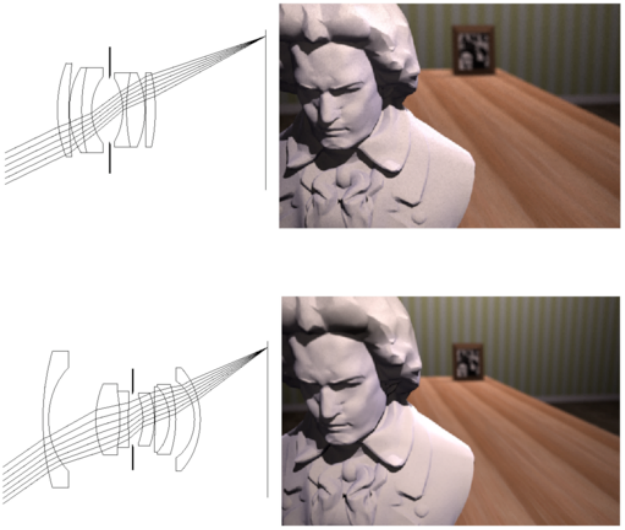
We can go further and introduce a more interesting or realistic camera model

from
A Realistic Camera Model for Computer Graphics
Craig Kolb, Don Mitchell, and Pat Hanrahan
Computer Graphics (Proceedings of SIGGRAPH '95), ACM SIGGRAPH, 1995, pp. 317-324



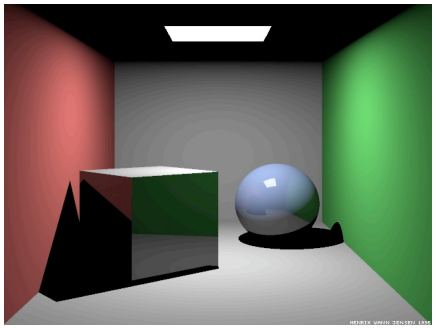
Note limited depth of field, just like a real lens

from
A Realistic Camera Model for Computer Graphics
Craig Kolb, Don Mitchell, and Pat Hanrahan
Computer Graphics (Proceedings of SIGGRAPH '95), ACM SIGGRAPH, 1995, pp. 317-324

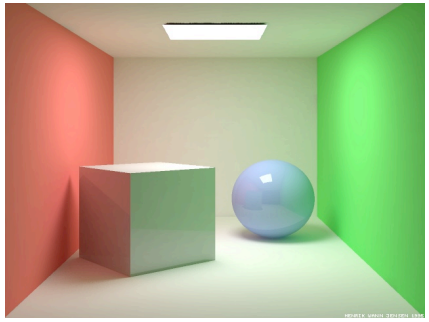


from
A Realistic Camera Model for Computer Graphics
Craig Kolb, Don Mitchell, and Pat Hanrahan
Computer Graphics (Proceedings of SIGGRAPH '95), ACM SIGGRAPH, 1995, pp. 317-324

Radiosity



Ray-traced Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj



Radiosity Cornell box, due to Henrik Jensen, http://www.gk.dtu.dk/~hwj, rendered with ray tracer

Radiosity

Want to capture the basic effect that surfaces illuminate each other

Again, following every piece of light from a diffuse reflector is impractical--but combinations of brute force and clever hacks can be done

Another approach: Radiosity methods

Radiosity

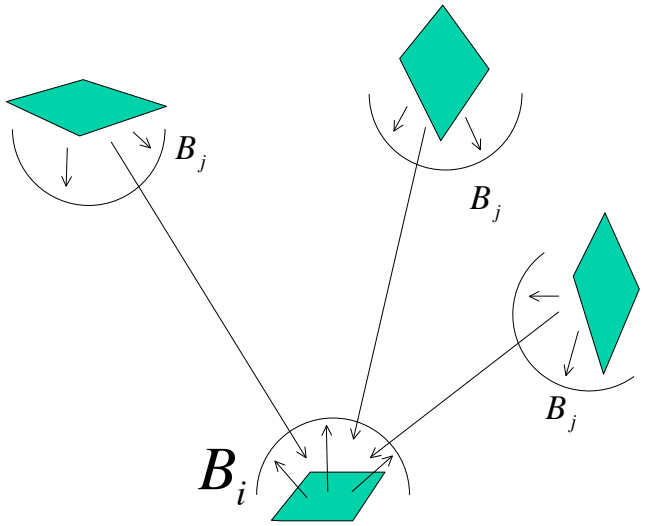
Think of the "world" as a bunch of patches. Some are sources, (and reflect), some just reflect. Each sends light towards all the others.

Consider one color band at a time (some of the computation is shared among bands).

Each surface, i, *radiates* reflected light, B_i

Each surface, *emits* light E_i (if it is not a source, this is 0).

Denote the albedo of surface i as ρ_i



Radiosity equation

$$B_i = E_i + \rho_i \sum_j F_{j \to i} B_j \frac{A_j}{A_i}$$

The form factor $F_{j \to i}$

is the fraction of light leaving dA_j arriving at dA_i taking into account orientation and obstructions

Useful relation

$$A_i F_{i \to j} = A_j F_{j \to i}$$

The equation now becomes

$$B_i = E_i + \rho_i \sum_j F_{i \to j} B_j$$

Rearrange to get

$$B_i - \rho_i \sum_j F_{i \to j} B_j = E_i$$

In matrix form

$$\begin{bmatrix} 1-\rho_1 F_{1\to 1} & -\rho_1 F_{1\to 2} & \dots & -\rho_1 F_{1\to n} \\ -\rho_2 F_{2\to 1} & 1-\rho_2 F_{2\to 2} & & -\rho_2 F_{2\to n} \\ & & & \\ -\rho_n F_{n\to 1} & -\rho_n F_{n\to 2} & & 1-\rho_n F_{n\to n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \\ E_n \end{bmatrix}$$

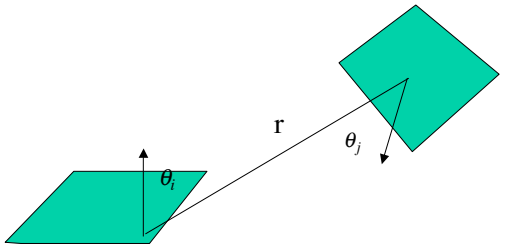
So, in theory, we just compute the Bi's by solving this (large!) matrix equation.

Optional

Optional

The fun part: Computing the $F_{i\to j}$

Without obstruction $dF_{dj\to di} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} dA_j$



Fancy methods exist for of computing and/or approximating storing form factors (e.g. hemisphere and hemicube methods)

Iterative Solution (gather)

The matrix equation can be solved iteratively (Gauss-Siedel method) starting with a crude estimate of the solution.

More intuitively, consider an estimate $\hat{B}_j$ for the B_j and plug them into our equation to re-estimated them.

$$B_i = E_i + \rho_i \sum_j F_{i\to j} \hat{B}_j$$

This is sometimes referred to as "gather", as we update the brightness of each patch in turn by gathering light from the other patches.

Iterative Solution (cast)

Iteration has the additional benefit that we can provide intermediate, approximate, solutions while the user waits.

But, in the previous version, each patch gets better in turn. Better to have all patches get a little better on each iteration.

This leads to the alternative approach where energy is cast from each patch in turn to update the others with:

$$B_j \quad due \ to \ B_i \ is \ p_j B_i F_{i\to j} \frac{A_i}{A_j} \qquad (\text{do } \forall \ B_j)$$

A second advantage is that the casting can be done in order of brightness, which is obviously helpful.