# Basic Ray-Tracing Ideas

D.A. Forsyth, UIUC

# What is rendering?

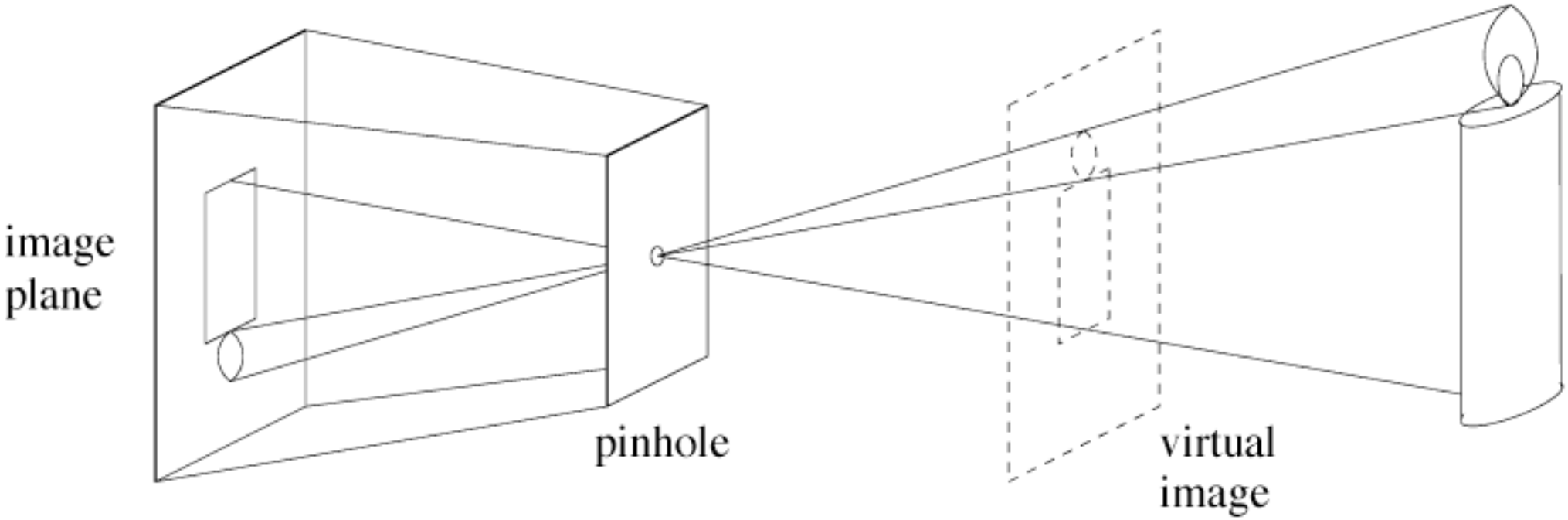
# How cameras work

# Cameras

- First photograph due to Niepce
- First on record shown in the book - 1822
- Basic abstraction is the pinhole camera
  - lenses required to ensure image is not too dark
  - various other abstractions can be applied

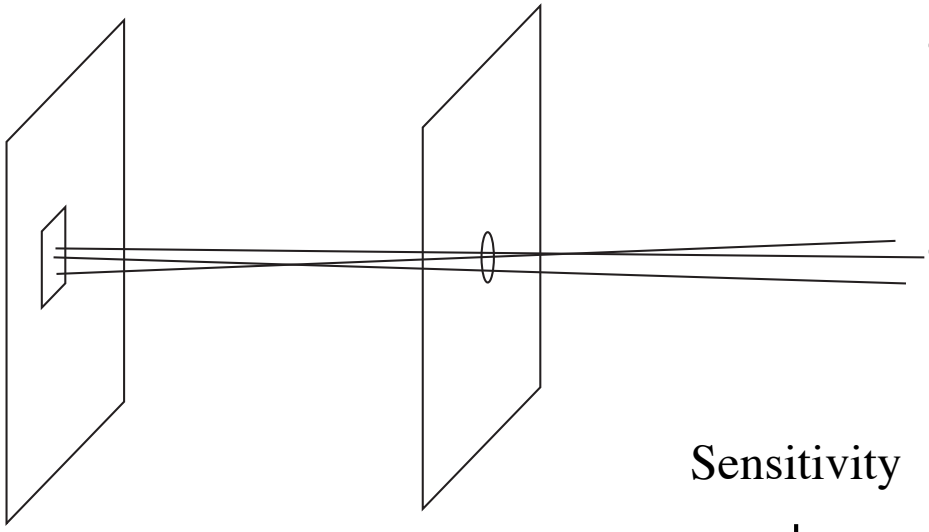
# Pinhole cameras

- Abstract camera model - box with a small hole in it
- Pinhole cameras work in practice




A photo of the Camera Obscura in San Francisco. This Camera Obscura is located at the Cliff House on the Pacific ocean. Credit to Jacob Appelbaum of http://www.appelbaum.net.

# Pixels, time and integration



- For every practical camera
  - pinhole opens, is open for some time, closes
- Each pixel is a light sensor
  - typically, linear
  - typically, average all incoming light
    - over all positions, directions, wavelengths, times at pixel
    - perhaps weighted

$$v = \int_{\Lambda} \int_{D} \int_{\Omega} \int_{T} w(\mathbf{x}, \lambda, \omega, t) L(\mathbf{x}, \omega, t) dt d\omega dx d\lambda$$

Sensitivity

Incoming radiance

wavelength

position

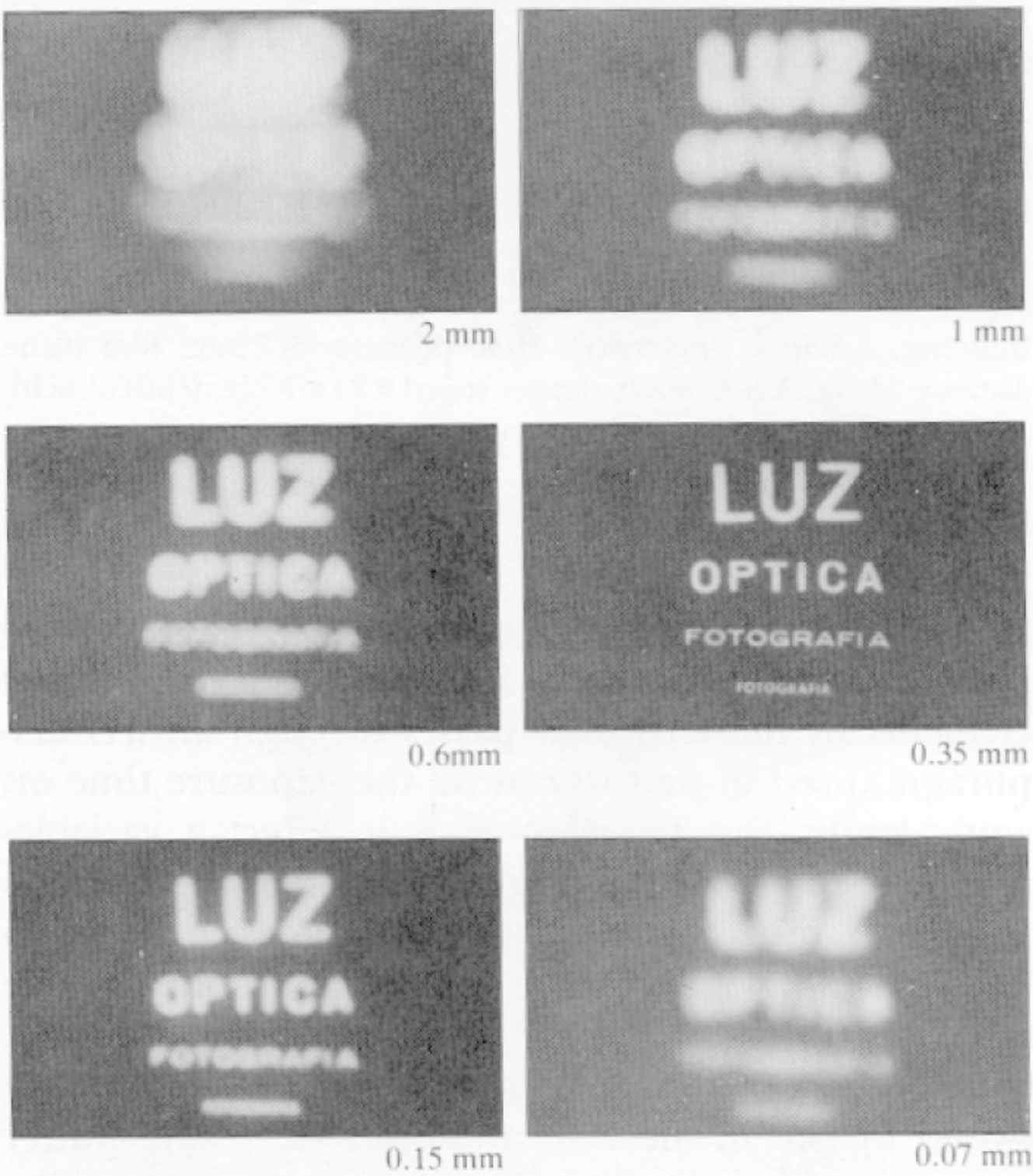
angle

time

Pinhole too big - many directions are averaged, blurring the image

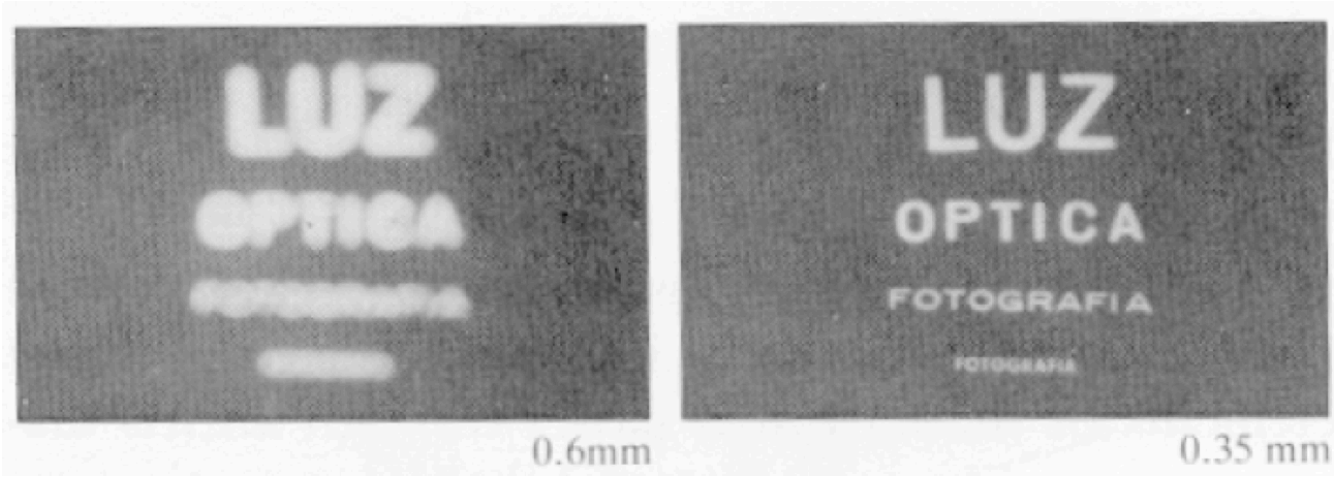
Pinhole too small- diffraction effects blur the image

Generally, pinhole cameras are *dark*, because a very small set of rays from a particular point hits the screen.



Blurring with large pinholes is an integration effect

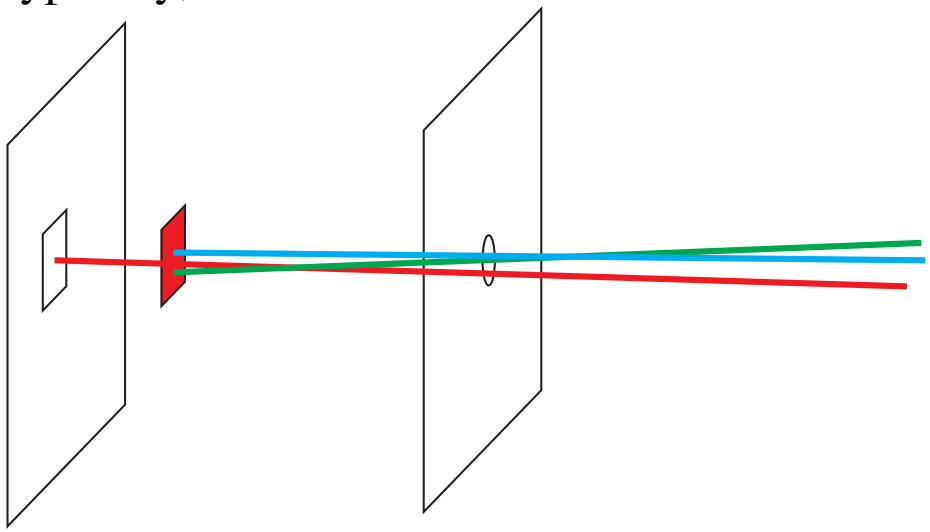
we're averaging over many directions,
some don't hit the letters,
and so average in the background.




$$v = \int_{\Lambda} \int_{D} \int_{\Omega} \int_{T} w(\mathbf{x}, \lambda, \omega, t) L(\mathbf{x}, \omega, t) dt d\omega dx d\lambda$$

# Color cameras

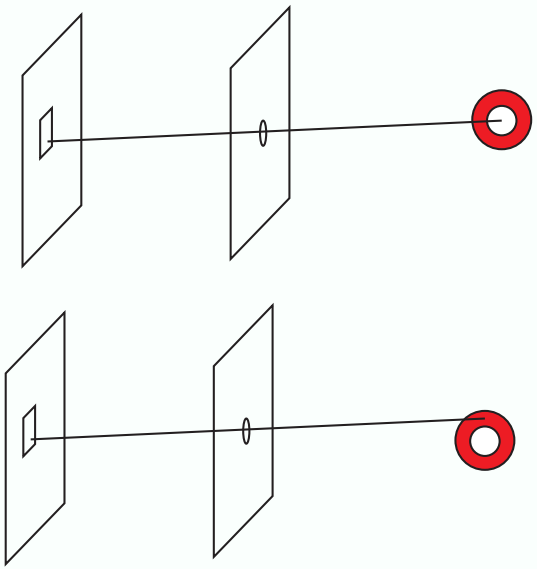
- Red, Green, Blue sensors
  - each has a different wavelength sensitivity
  - typically, some translucent colored material in front of the pixel



$$v = \int_{\Lambda} \int_{D} \int_{\Omega} \int_{T} w(\mathbf{x}, \lambda, \omega, t) L(\mathbf{x}, \omega, t) dt d\omega dx d\lambda$$

# Motion Blur

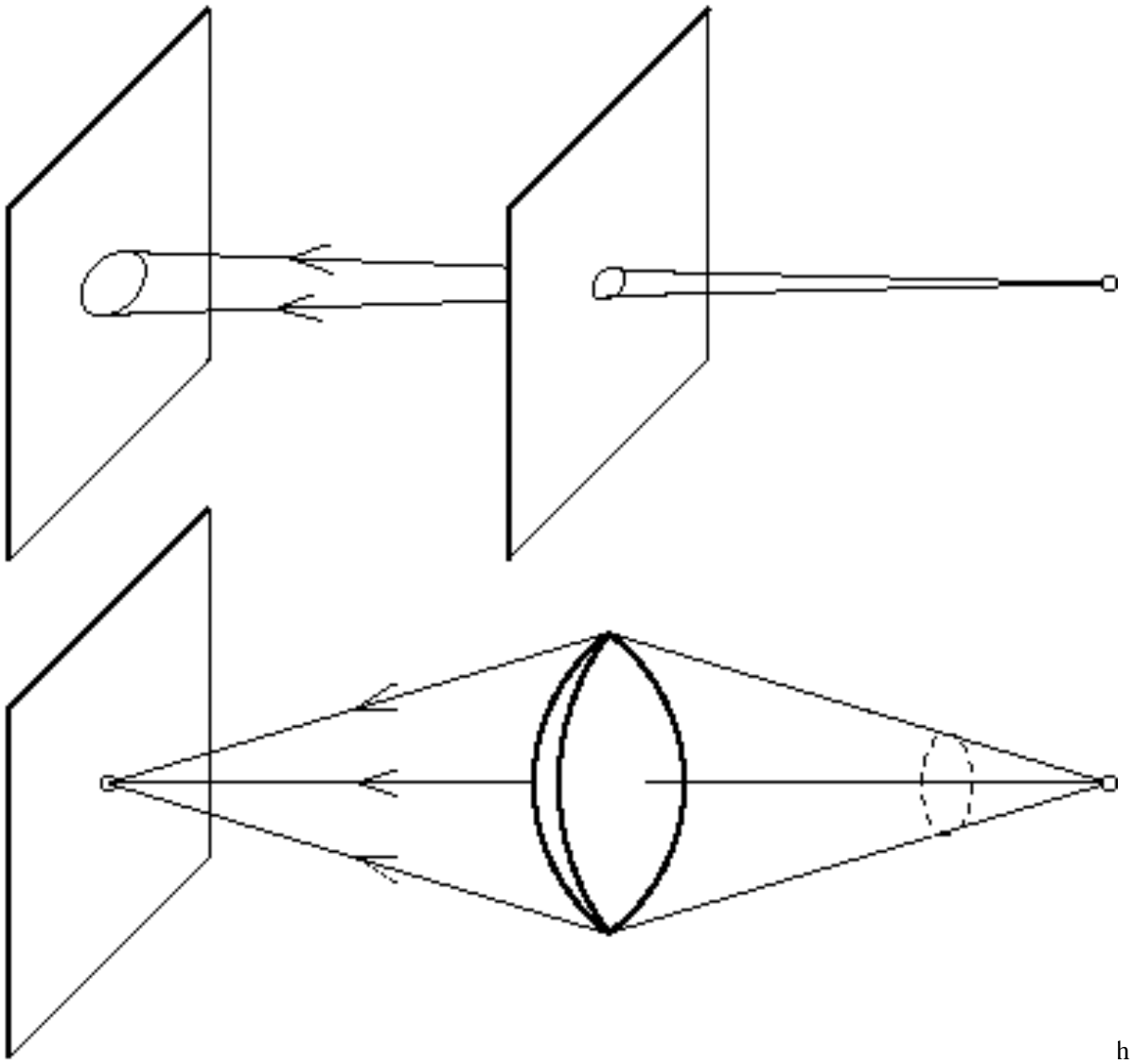
- Another integration effect



T

T+dt

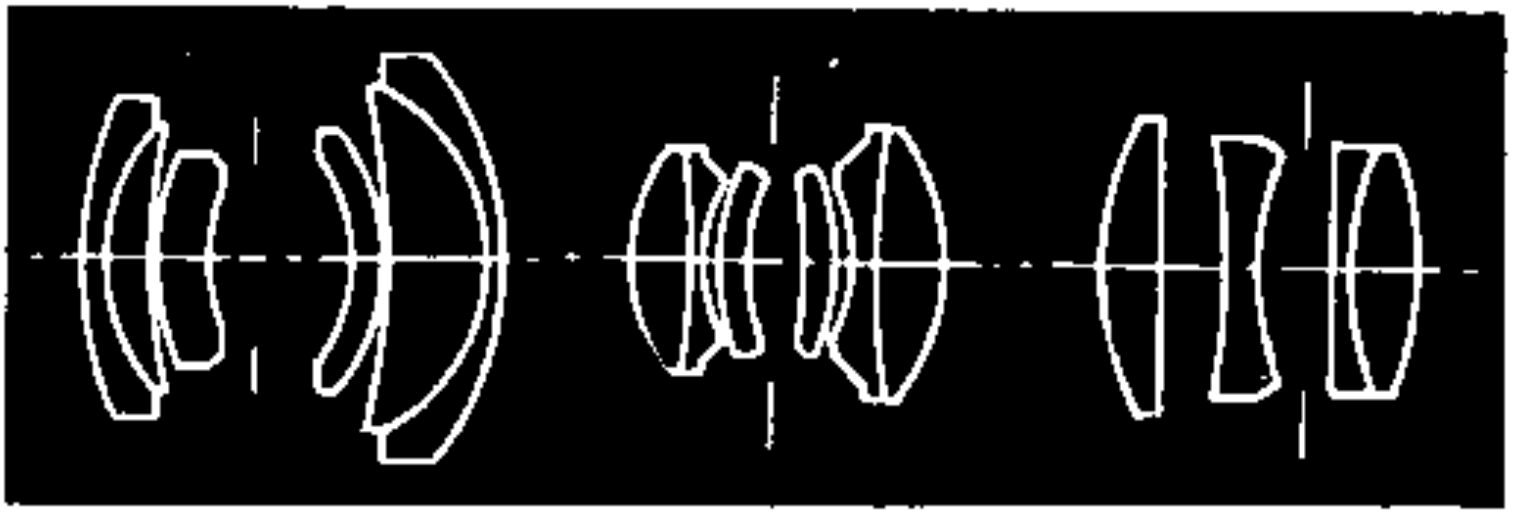
# The reason for lenses



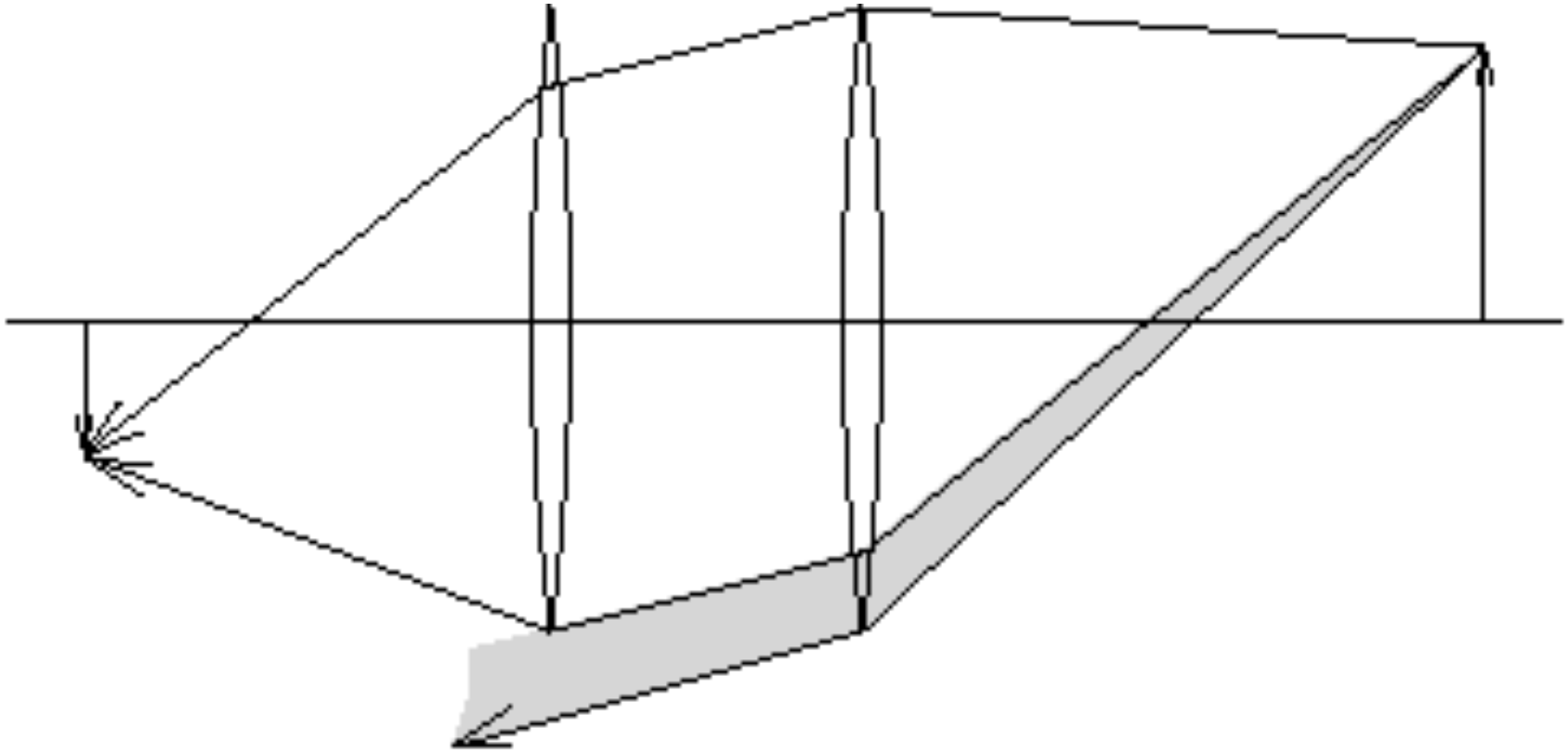
# Lenses come with problems

- Spherical aberration
  - Lens is not a perfect thin lens, and point is defocused

# Lens systems



# Vignetting



# Other (possibly annoying) phenomena

- Chromatic aberration
  - Light at different wavelengths follows different paths; hence, some wavelengths are defocussed
  - Machines: coat the lens
  - Humans: live with it
- Scattering at the lens surface
  - Some light entering the lens system is reflected off each surface it encounters (Fresnel's law gives details)
  - Machines: coat the lens, interior
  - Humans: live with it (various scattering phenomena are visible in the human eye)
- Geometric phenomena (Barrel distortion, etc.)

# Randomized estimates of integrals

Weak law of large numbers

if $x_i \sim p(x)$

then $\frac{1}{N}\sum_{i=1}^{N} f(x_i) \rightarrow \int f(x)p(x)dx$

- i.e. we can approximate integrals with sums
  - example: p(x) uniform, stochastic sampling of pixel
- generically, known as Monte Carlo estimates

# Importance weighting

if $x_i \sim p(x)$

then $\frac{1}{N}\sum_{i=1}^{N} f(x_i) \rightarrow \int f(x)p(x)dx$

If $x_i \sim p(x)$

Then $$\int f(x)dx \approx \frac{1}{N}\frac{f(x_i)}{p(x_i)}$$

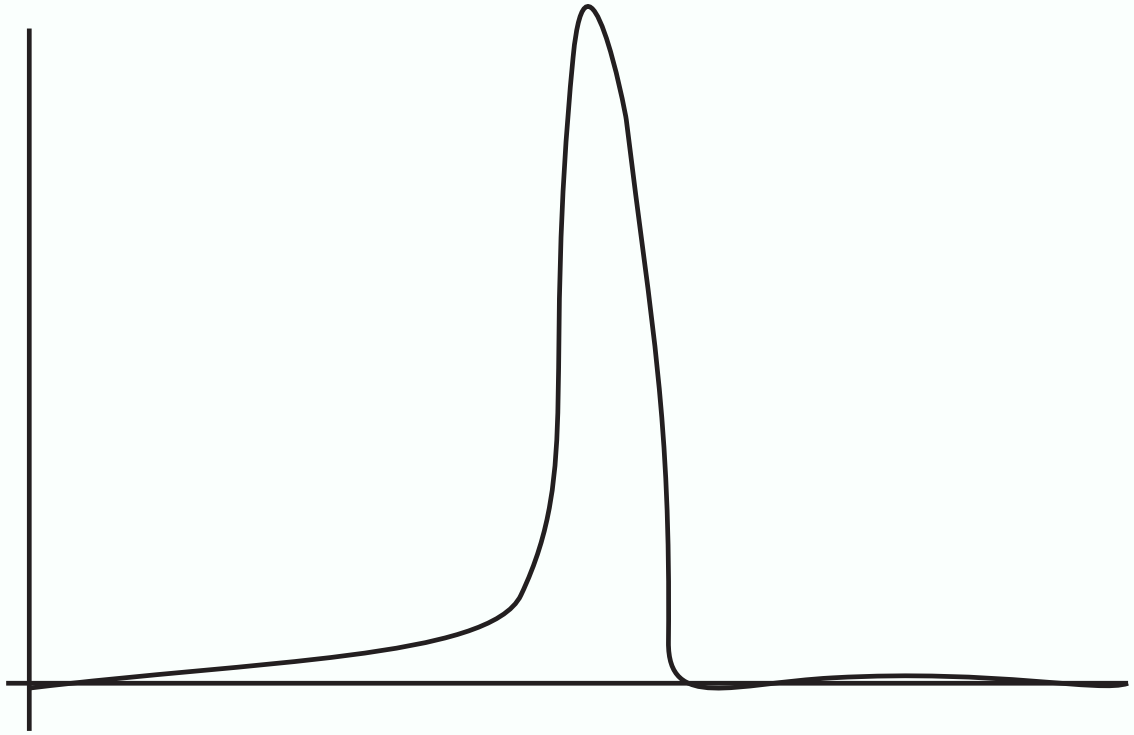
# Randomized estimates and variance

$$\int f(x)dx \approx \frac{1}{N}\frac{f(x_i)}{p(x_i)}$$

- The estimate is the value of a random variable
  - (different random samples -> different estimates)
  - whose expected value is the value of the integral
  - but whose variance might be very big
    - and is usually very hard to know

- Simple reasoning suggests that
  - p should be big when f is big, etc.

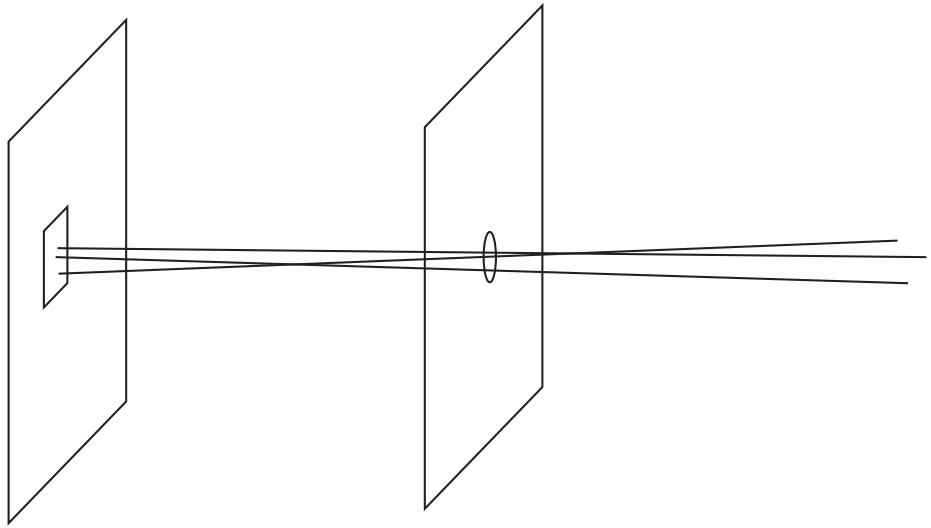
# Variance example



Drawing uniform samples will get a poor estimate
-> draw samples mostly at peak and downweight

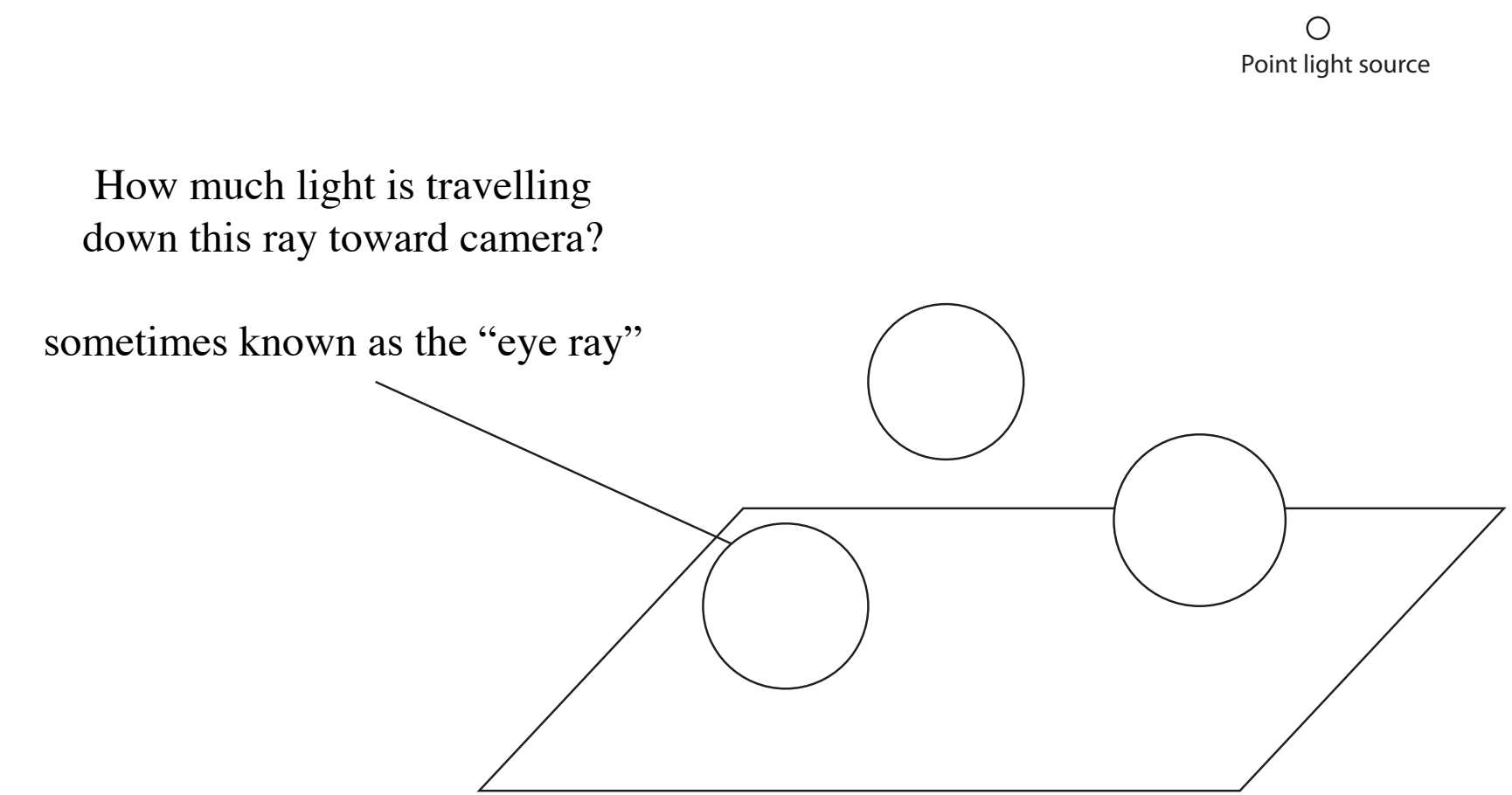
# Algorithmic framework

$$v = \int_\Lambda \int_D \int_\Omega \int_T w(\mathbf{x}, \lambda, \omega, t) L(\mathbf{x}, \omega, t) dt d\omega dx d\lambda \approx \sum_{i \in \text{rays}} g(\text{ray}) L(\text{ray})$$



Computational problem - what is L(ray)?

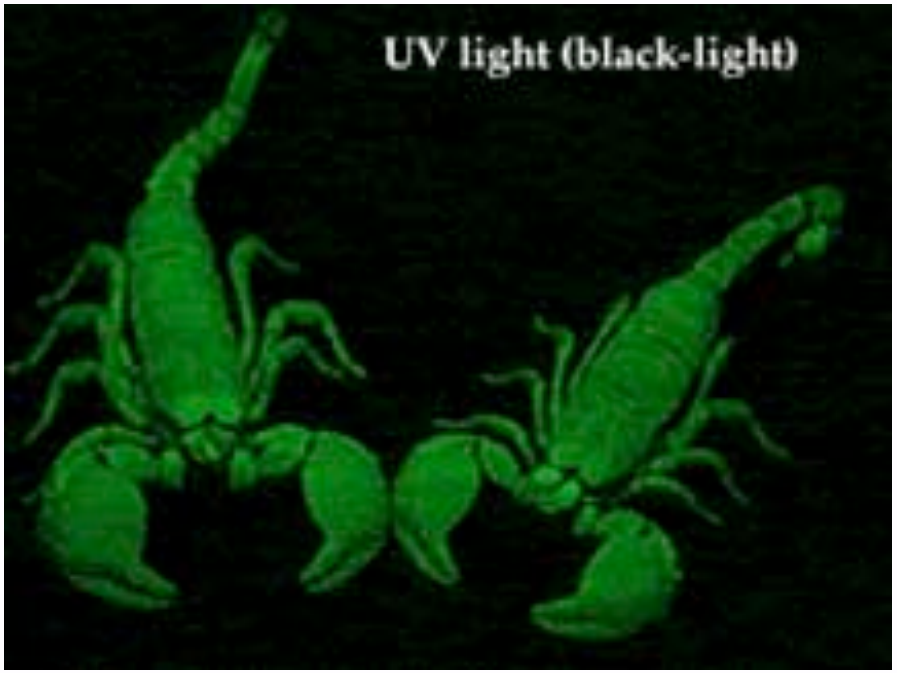
# Very simple ray-tracing



# Light at surfaces

- Many effects when light strikes a surface -- could be:
  - absorbed; transmitted; reflected; scattered
  - Simplify
    - Assume that
      - surfaces don’t fluoresce
      - surfaces don’t emit light (i.e. are cool)
      - all the light leaving a point is due to that arriving at that point
- Two main abstractions
  - Diffuse surface
  - Specular surface


White light (indoor)

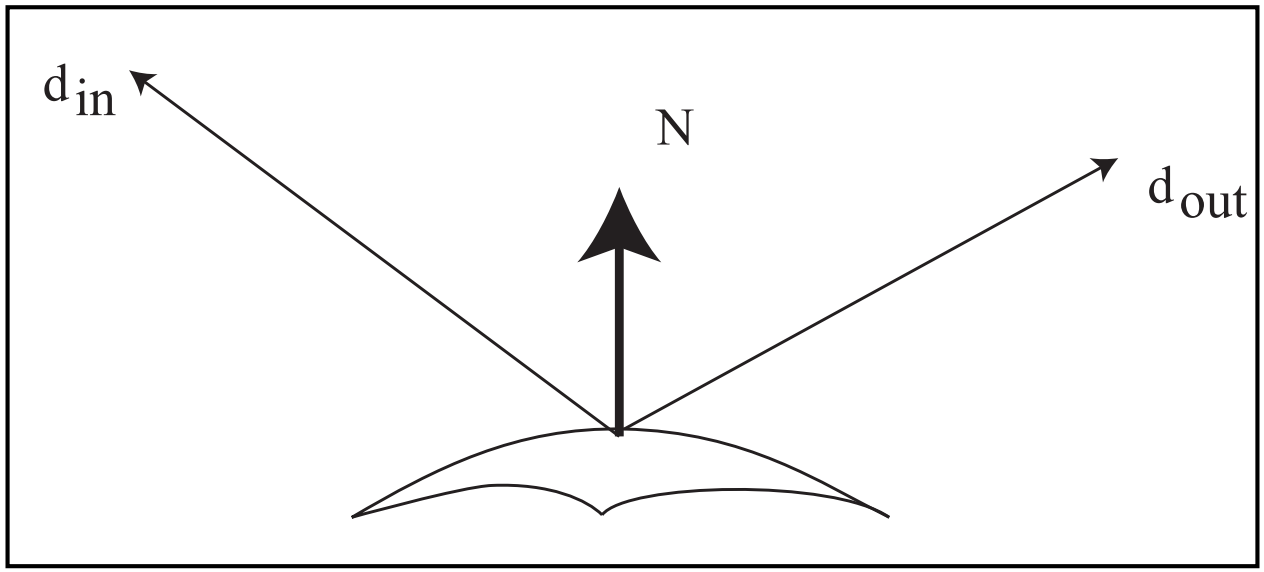

UV light (black-light)

# Diffuse reflection

- Light leaves the surface evenly in all directions
  - cotton cloth, carpets, matte paper, matte paints, etc.
  - most “rough” surfaces
  - Parameter: Albedo
    - percentage of light arriving that leaves
    - range 0-1
      - practical range is smaller
- Test:
  - surface has same apparent brightness when viewed from different dir’ns

# Specular surface

- For some surfaces, reflection depends strongly on angle
  - mirrors (special case)
    - incoming direction, normal and outgoing direction are coplanar
    - angle din, normal and angle dout, normal are the same
  - more general cases later
  - rules:
    - din, dout, N coplanar
    - angle(din, N)=angle(dout, N)

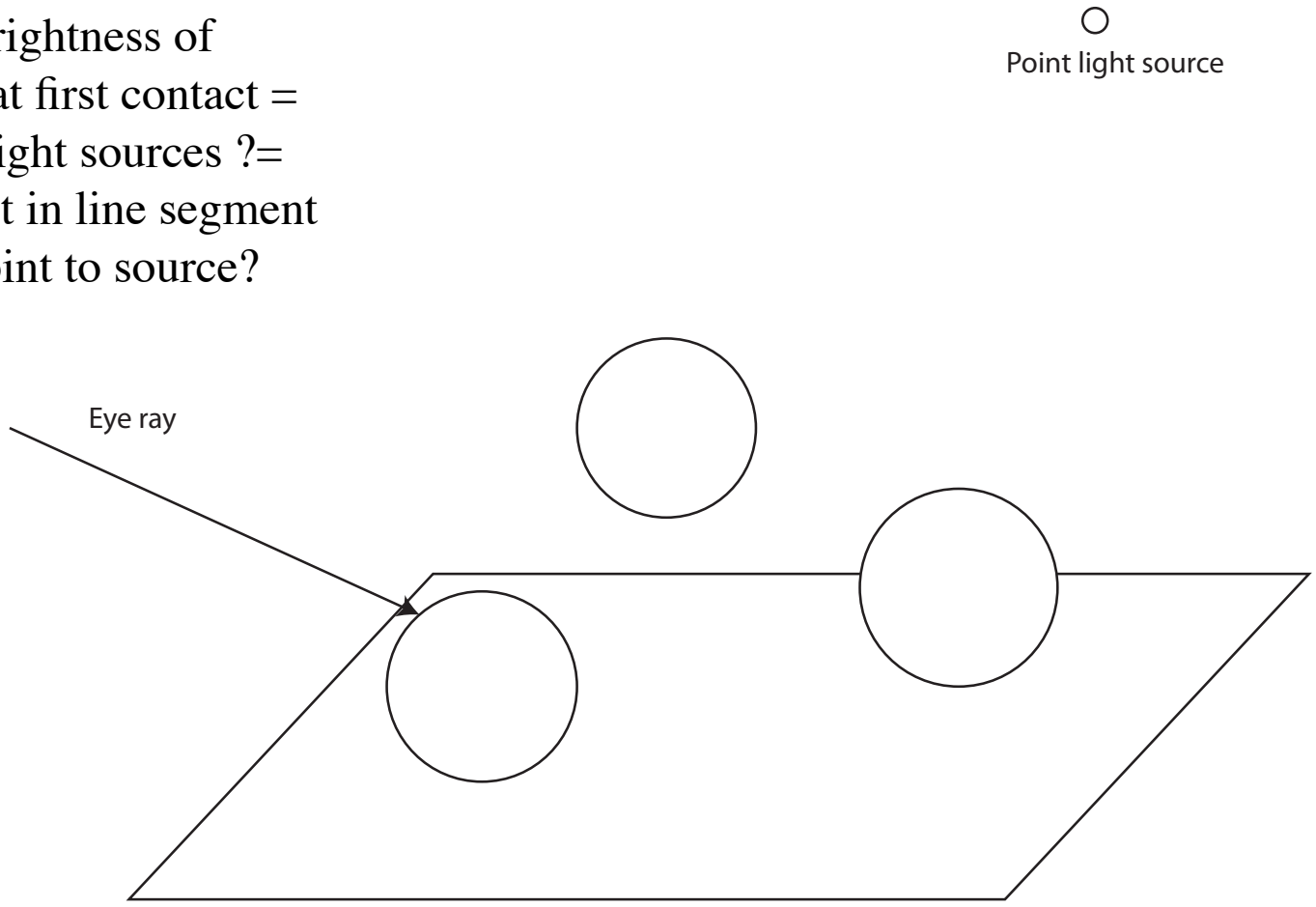


# Lighting model

- Light arrives at a surface ONLY from a luminaire
  - this is an object that “makes light”
    - through chemical, mechanical, etc means
- Wild oversimplification, good for us right now
  - wait a few slides and it’ll get more complicated

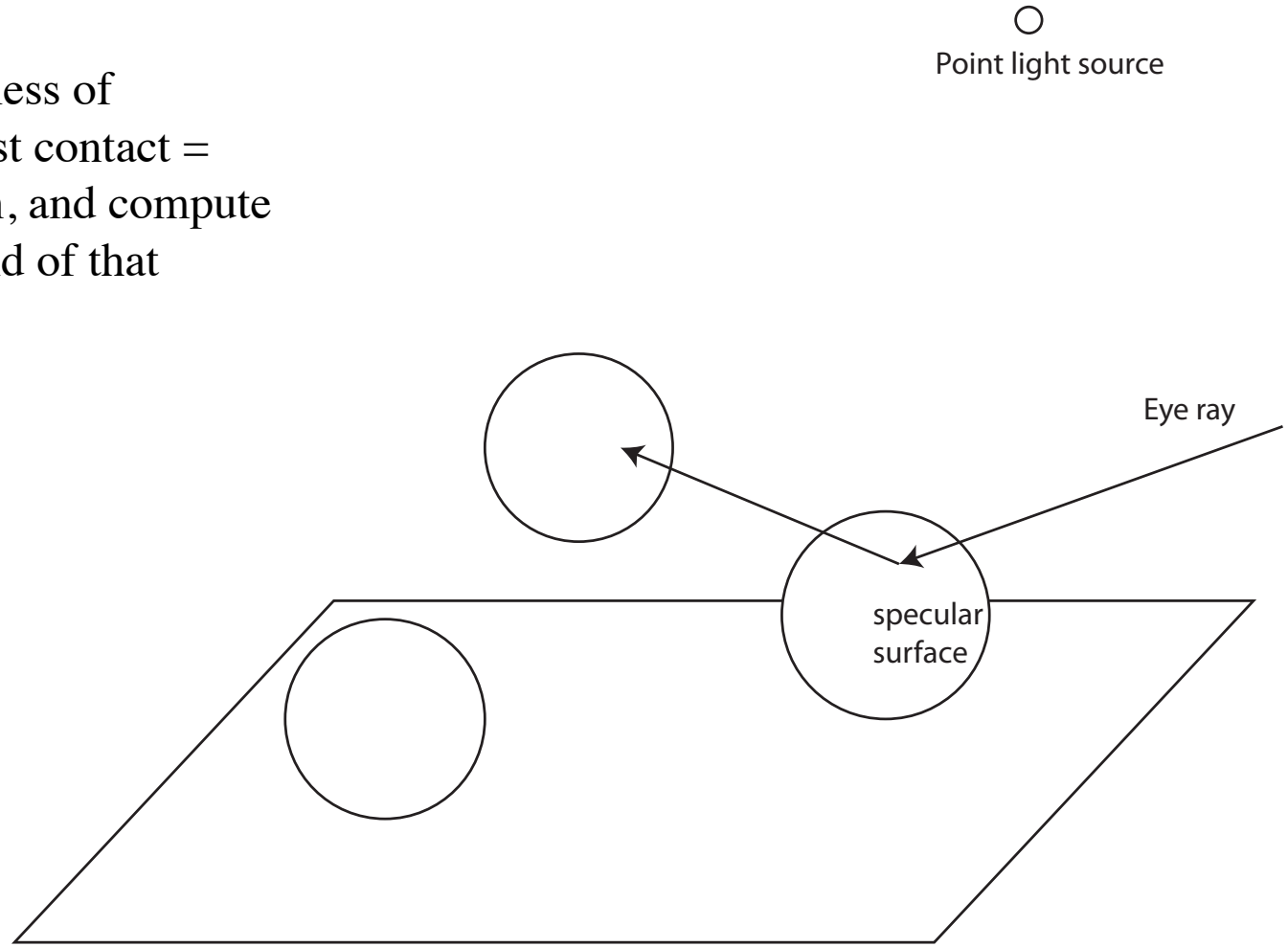
# Eye ray strikes diffuse surface

Compute brightness of diffuse surface at first contact = Can it see the light sources ?= Is there an object in line segment connecting point to source?



# Eye ray strikes specular surface

Compute brightness of specular surface at first contact = eye ray changes direction, and compute brightness at the end of that

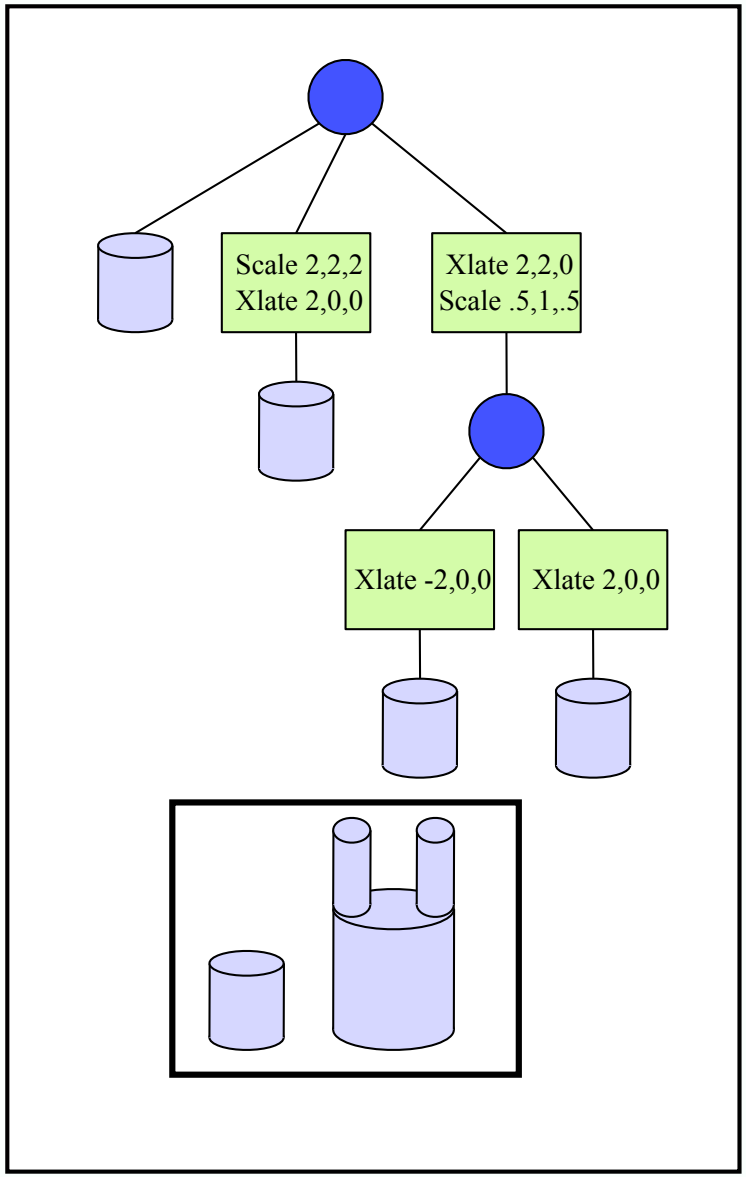


# Implied computational problems

- Fast, accurate intersection with complicated models
  - Improved rendering
    - anti aliasing (= more rays)
    - motion blur (= more rays)
    - more complex illumination phenomena (= more rays, caching)

# Reminder: Scene Graphs

- Hierarchical representation of all objects in scene
  - familiar from raster graphics, etc

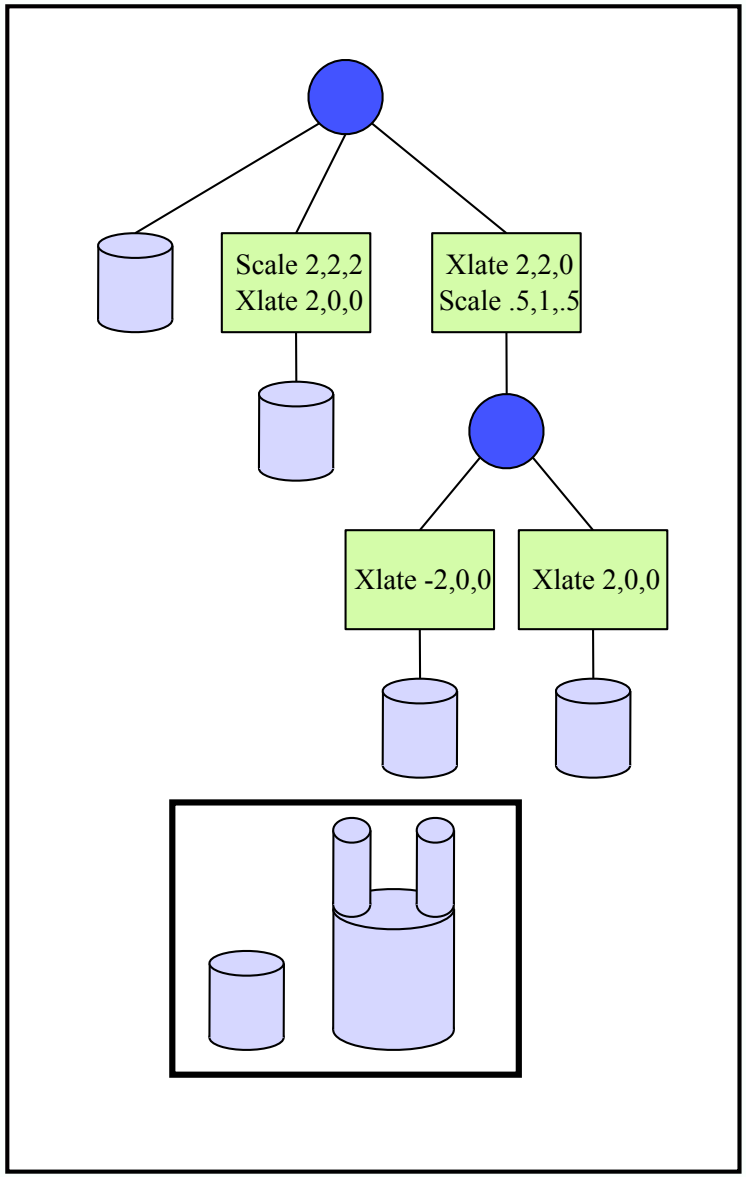


# Geometric Primitives

- Primitives we can deal with
  - half-space (because we can do plane intersection)
  - sphere (because we can do sphere intersection)
  - cylinder (easy generalization of sphere)
  - convex polyhedron (easy generalization of half-space)
- Others will come as we learn more intersection techniques

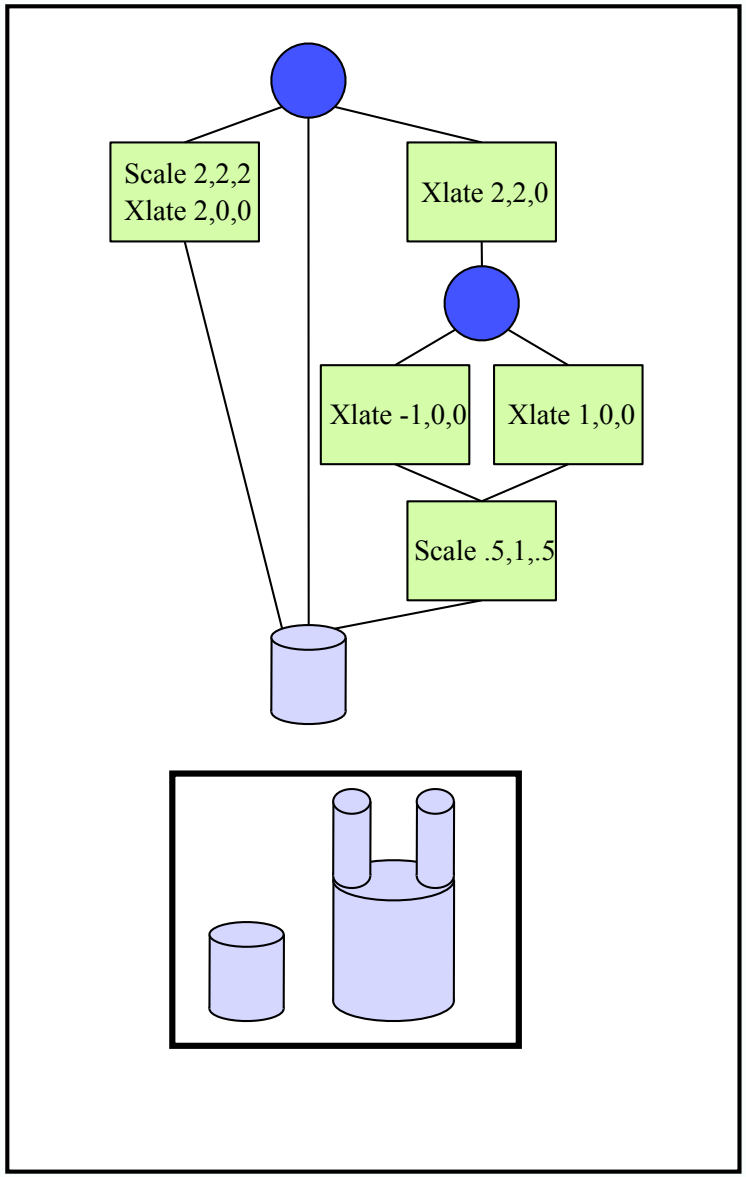
# Reminder: Scene Graphs

- Hierarchical representation of all objects in scene
  - familiar from raster graphics, etc
- Transformation nodes now:
  - Intersect children with ray
    - transform ray to child's frame
    - i.e. inverted from usual
  - Returned normal must be in world frame
    - i.e. transpose(inverse(T))
  - Maintain inverse(T)

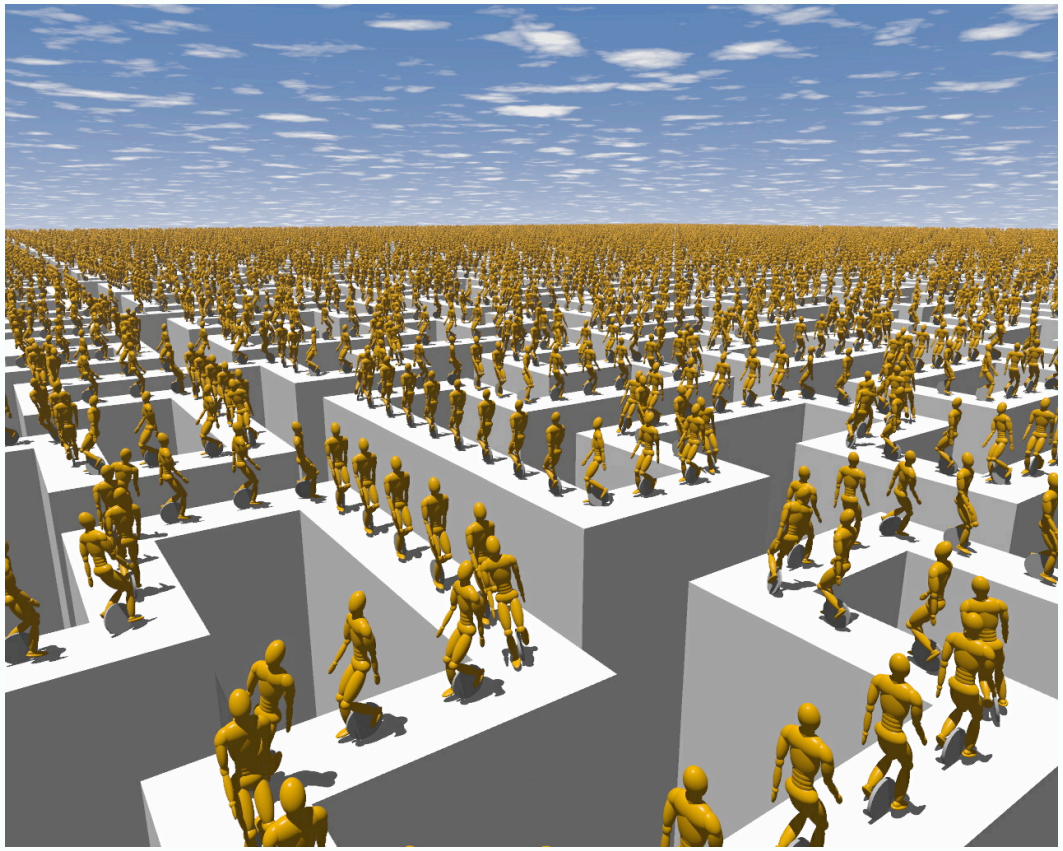


# Reminder: Instancing

- Scene graph is a hierarchy
- Not necessarily a tree
- Directed acyclic graph (DAG)
- Nodes may have multiple parents
- Instance
  - Appearance of each node's geometry in scene

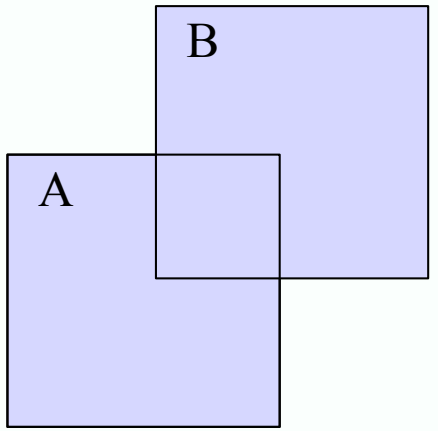


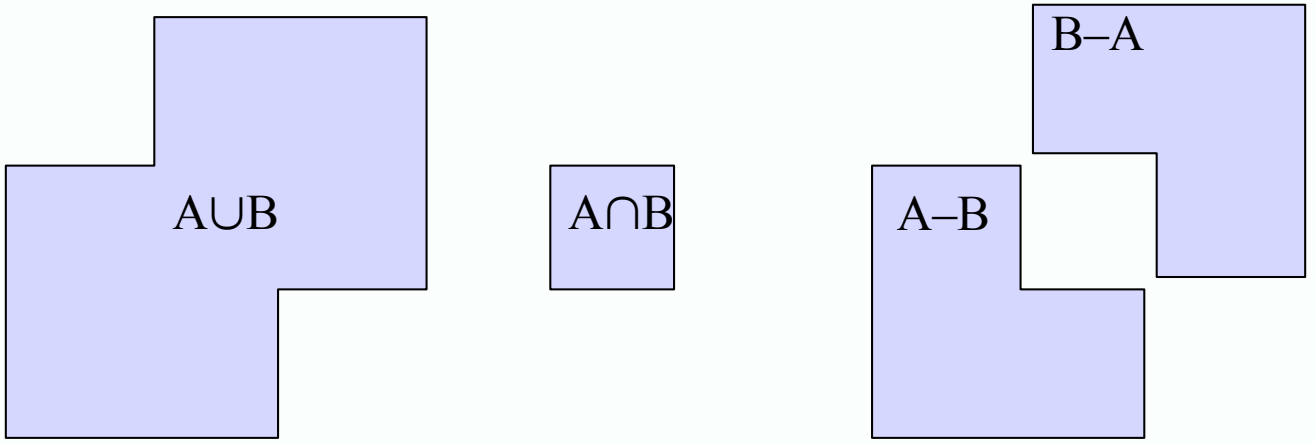
# Fun with instancing



# CSG

- Constructive Solid Geometry
  - objects are boolean combinations of primitive volumes
  - union, intersection, difference
    - usually regularized



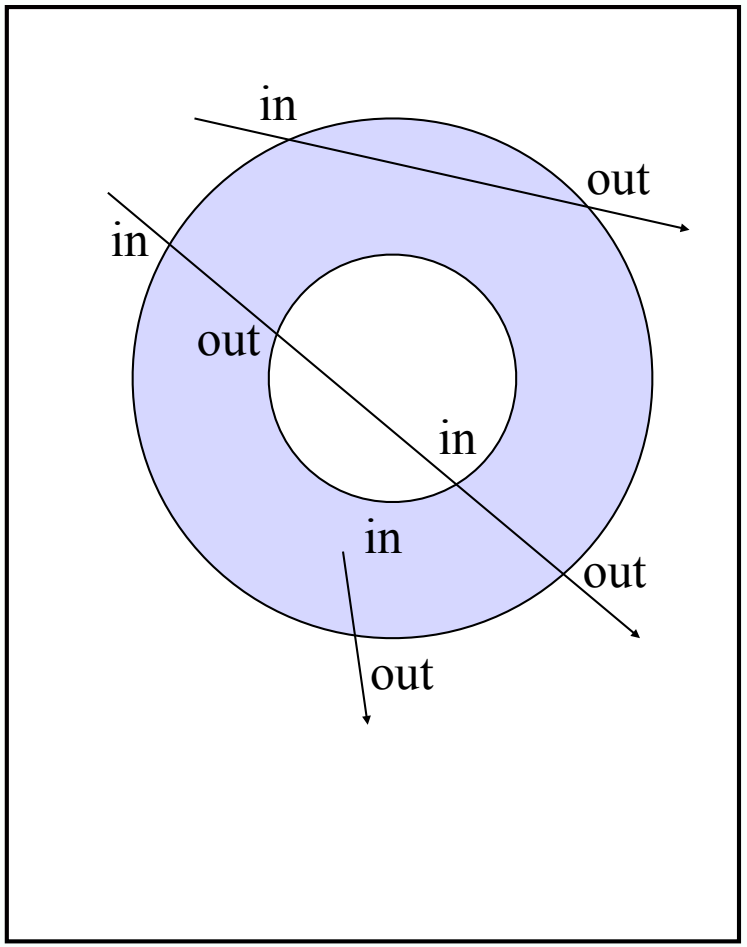



# Geometric Primitives

- Primitives we can deal with
  - half-space (because we can do plane intersection)
  - sphere (because we can do sphere intersection)
  - cylinder (easy generalization of sphere)
  - convex polyhedron (easy generalization of half-space)
- Others will come as we learn more intersection techniques

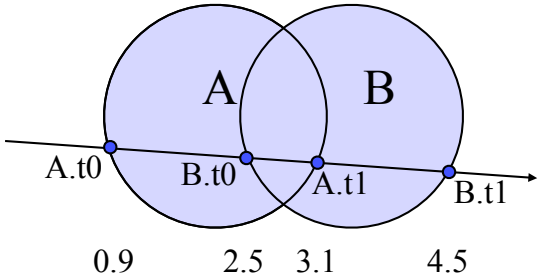
# Raytracing CSG

- Represent all intersections in a hit record
  - list
- If we know where focal point is (in/out), parity classifies all others



# Raytracing CSG

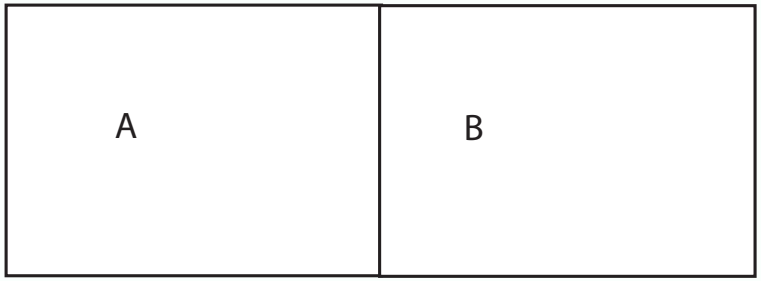
- List of t-values for A, B w/in-out classification
  - A.t_list = {0.9, 3.1} = {0.9in, 3.1out}
  - B.t_list = {2.5, 4.5} = {2.5in, 4.5out}
    - Use dot(r.d,n) to determine in,out
- Merge both lists into a single t-ordered list
  - { 0.9 Ain Bout,
    2.5 Ain Bin,
    3.1 Aout Bin,
    4.5 Aout Bout }
    - Keep track of A and B in/out classification
- Use Roth table to classify t-values




Roth Table

| Op | A | B | Res |
|---|---|---|---|
| + | in | in | in |
| | in | out | in |
| | out | in | in |
| | out | out | out |
| * | in | in | in |
| | in | out | out |
| | out | in | out |
| | out | out | out |
| – | in | in | out |
| | in | out | in |
| | out | in | out |
| | out | out | out |

# Regularizing CSG




- Primitives can produce non-volumes
  - e.g. A intersect B in pic gives line

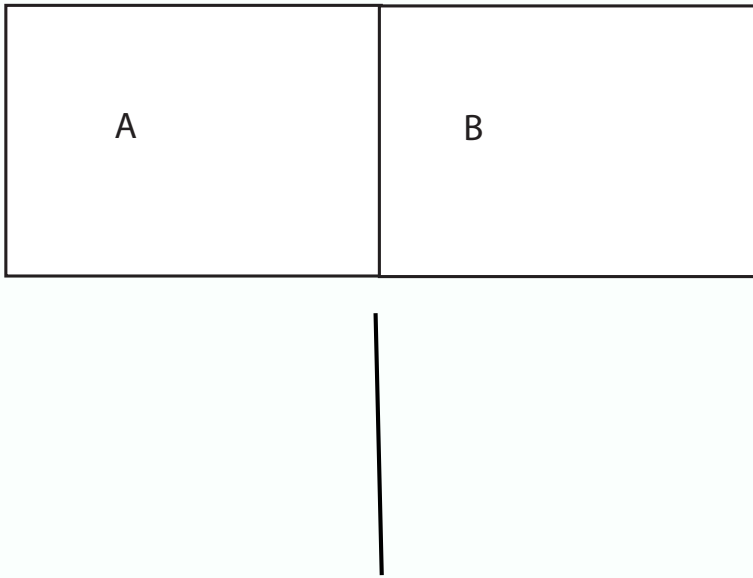


$$A \cap^{*} B = \text{closure}\left(\text{interior}(A) \cap \text{interior}(B)\right)$$

# There's a general phenomenon here

- Points that lie on top of one another
  - but we may not be able to tell
- Our t-values aren't precisely correct
  - numerical representations aren't precise
    - could be for polynomial surfaces, but this is not worth the effort
- This means
  - intersections aren't precisely where we think they are
  - eg  shadow ray eczema
- Tolerable solution
  - regard points that are "very close" as the same point
  - cures shadow ray eczema by ignoring surface as blocker
  - can be used to cure previous problem

# Regularizing CSG




- Primitives can produce non-volumes
  - e.g. A intersect B in pic gives line
- Regularize
  - eg

$$A \cap^* B = \text{closure}\left(\text{interior}(A) \cap \text{interior}(B)\right)$$

  - equivalently
    - require Bin to occur some small distance before Aout to get hit

This makes the line go away. (ex: how do you regularize union, difference?)

# Implicit Surfaces

- Surface is:
- points in vector form:
- ray is:
- intersections are:
  - and are obtained by root finding

$$f(x, y, z) = 0$$

$$f(\mathbf{x}) = 0$$

$$\mathbf{a} + t\mathbf{v}$$

$$f(\mathbf{a} + t\mathbf{v}) = 0$$

# Accurate Intersection: Computing roots

- Options: numerical root finding
  - Interval halving
  - Newton's method with deflation
  - Bracketing with Sturm sequence

# Interval halving

- Assume we have two points on ray
  - perhaps generated by some form of spatial subdivision scheme
  - one on positive side, one on negative side of intersection
- Split the interval in half
  - One half has the root (+-)
  - Other doesn't (++, --)
- Keep the one that does, and go again if it is too big

# Newton's method

- Estimate is: $t_n$
- Observe that: $f(t_n + \Delta t) = f(t_n) + \Delta t \frac{df}{dt} = 0$
- so update is: $\Delta t = -\frac{f(t_n)}{\left(\frac{df}{dt}\right)}$
-

# Practicalities

- Deflation: if you have found a root, divide the polynomial by (t-root) to reduce degree
- Newton's method can behave badly
  - start in a good place
  - e.g. root from previous ray with this object
- Newton's method not efficient for shadow rays
- Newton's method doesn't guarantee closest root

# Sturm sequences

- Build a sequence of polynomials

$$
\begin{aligned}
p_0(t) &= f(t) \\
p_1(t) &= \frac{df}{dt} \\
&\cdots \\
p_k(t) &= -\mathrm{rem}(p_{k-2}, p_{k-1}) \\
&\cdots \\
&p_m \\
&0
\end{aligned}
$$

  - (where rem stands for remainder; f should not have repeated roots)

# Sturm sequences

- write $\sigma(\xi)$ for the number of sign changes in

$$(p_0(\xi), p_1(\xi), p_2(\xi), \ldots, p_m(\xi))$$

- then for a<b, number of real roots in (a, b] is

$$\sigma(a) - \sigma(b)$$

Can bracket root using interval halving, use for shadow rays

# Sturm sequences: example

$$p_0 = t^3 + 3t^2 - 1$$
$$p_1 = 3t^2 + 6t \qquad \text{so } p_0 = (t/3)p_1 + (1/3)p_1 - 2t - 1$$
$$p_2 = 2t + 1$$
$$p_3 = \text{constant}$$

-9/4

Ex: how many roots in 0-1 interval?
how many roots in 0 - infinity interval?
find root in 0-1 interval

# Making Ray Tracing Faster

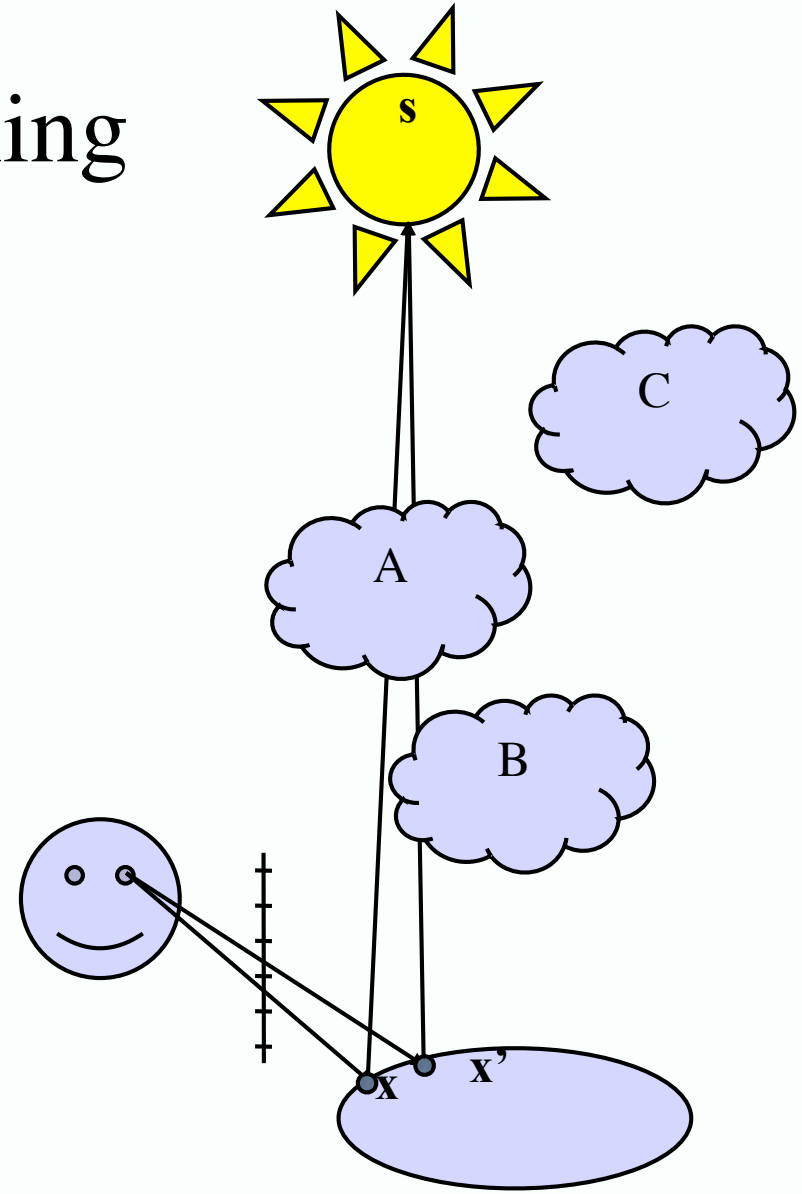
- Coherence
  - Image coherence:  rays through nearby pixels go through nearby things
  - Spatial coherence: similar rays go through similar things
  - Temporal coherence: the same ray at the next time goes through similar things

# Item buffer

- Use conventional z-buffer renderer to render surfaces
  - shade with pointer, not illumination
  - this gives pointer to closest surface
  - not much used now (ex: why?)

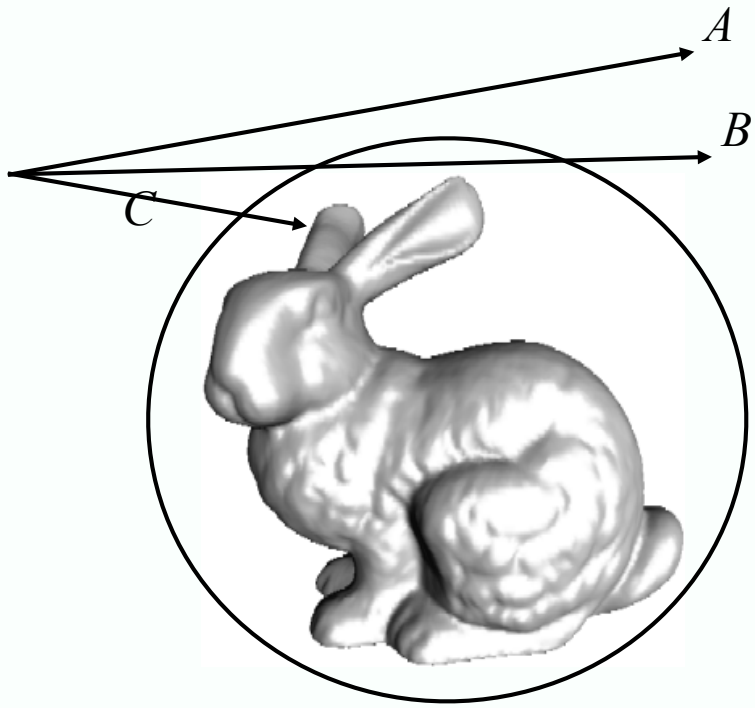
# Shadow Caching

- Any interloper between surface point **x** and the light source **s** will cast a shadow
  - Doesn't matter how many
  - Doesn't matter which is closest
  - Stop ray intersections once *any* intersection found
- Neighboring shadowed surface points **x** and **x**' probably shadowed by the same object
  - Start shadow ray intersection search with object intersected in last shadow search



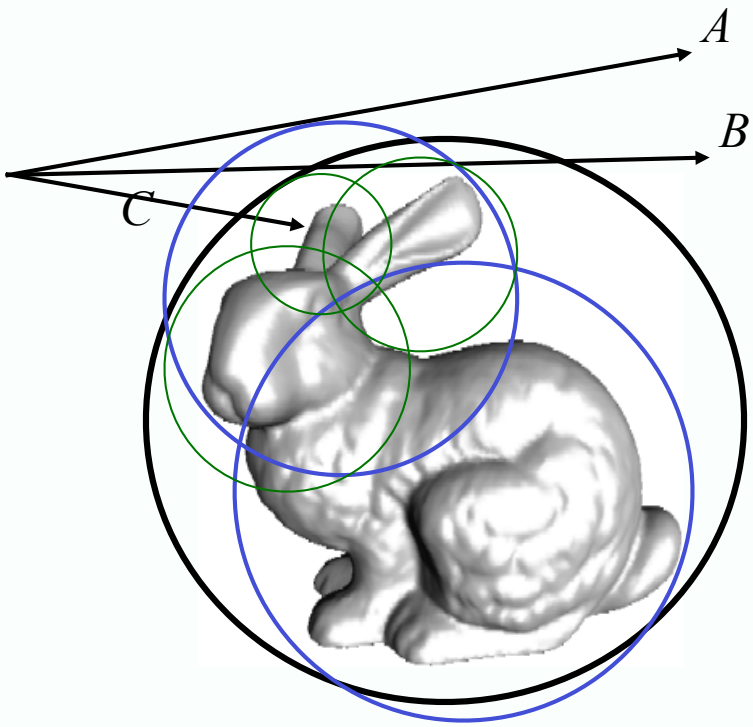
# Bounding Volume

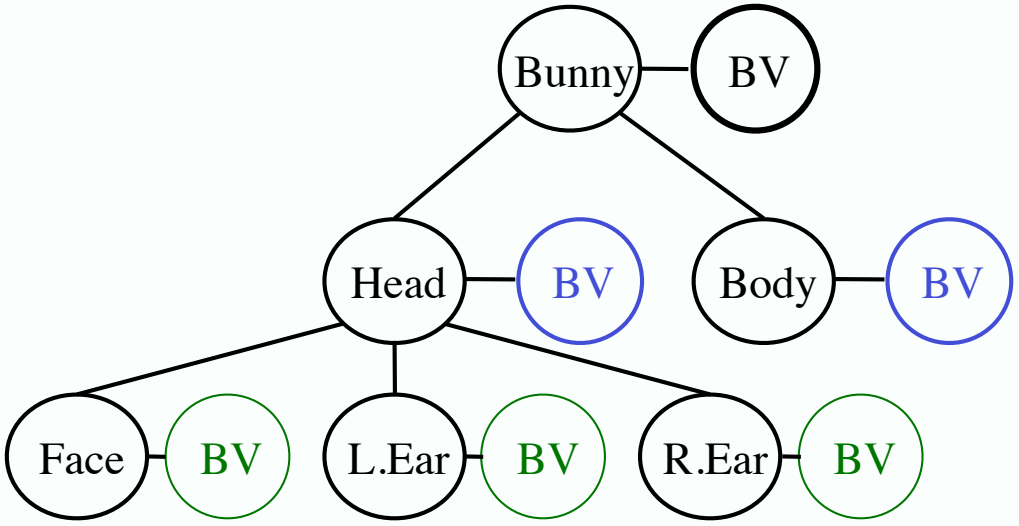
- Ray-bunny intersection takes 70K ray-triangle intersections even if ray misses the bunny
- Place a sphere around bunny
  - Ray *A* misses sphere so ray *A* misses bunny without checking 70K ray-triangle intersections
  - Ray *B* intersects sphere but still misses bunny after checking 70K intersections
  - Ray *C* intersects sphere and intersects bunny
- Can also use axis-aligned bounding box
  - Easier to create for triangle mesh



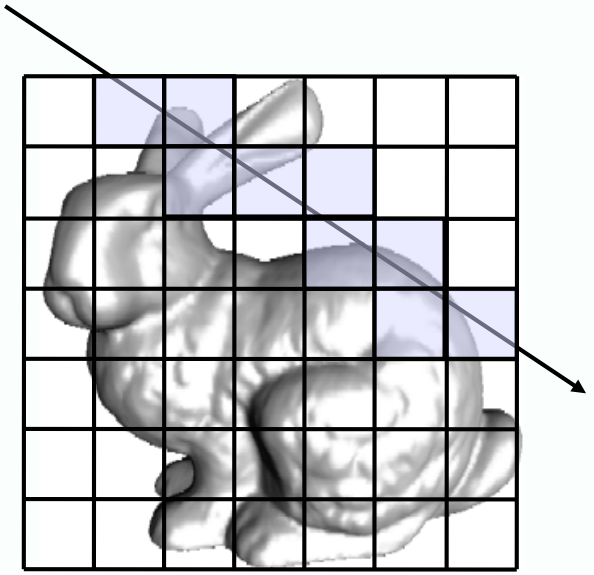
# Bounding Volume Hierarchy

- Associate bounding volume with each node of scene graph
- If ray misses a node's bounding volume, then no need to check any node beneath it
- If ray hits a node's BV, then replace it with its children's BV's (or geometry)
- Breadth first search of tree
  - Maintain heap ordered by ray-BV intersection *t*-values
  - Explore children of node w/least pos. ray-BV *t*-value


A
B
C


Bunny
BV
Head
BV
Body
BV
Face
BV
L.Ear
BV
R.Ear
BV

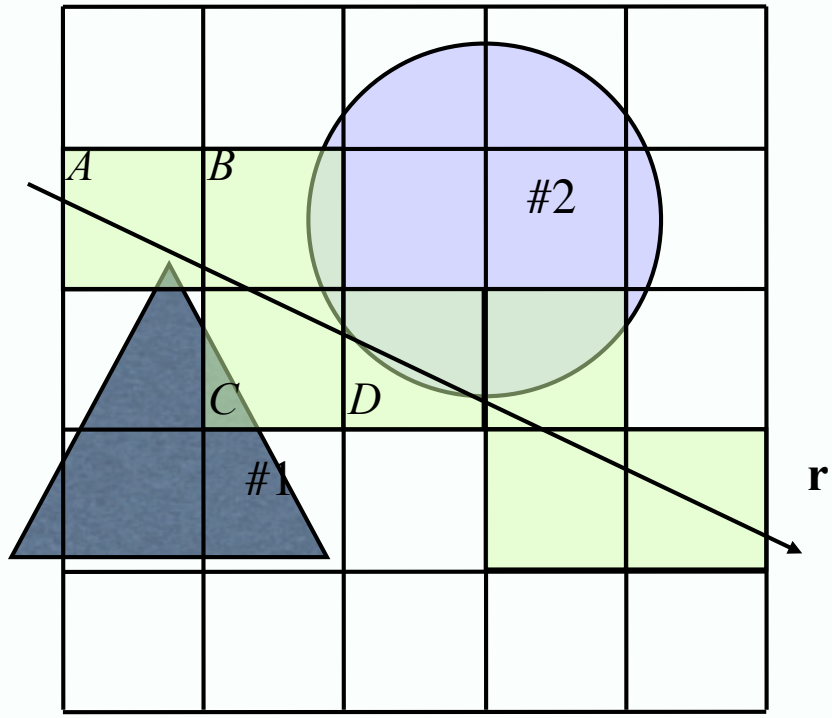
# Grids



- Encase object in a 3-D grid of cubes
  - each has list of all triangles it intersects
- Rasterize ray to find which cells it intersects
  - 3D Bresenham algorithm
  - All cells that contain any part of ray
- Working from first ray-cell to last…
  - Find least positive intersect of ray with triangles in cell's list
  - If no intersection, move on to next cell

# Tagging

- Ray-object intersection test valid for ray with entire object
  - not just portion of object inside current cell
  - Need only intersect object once for each ray
- Tags
  - does not intersect
  - intersection at ...

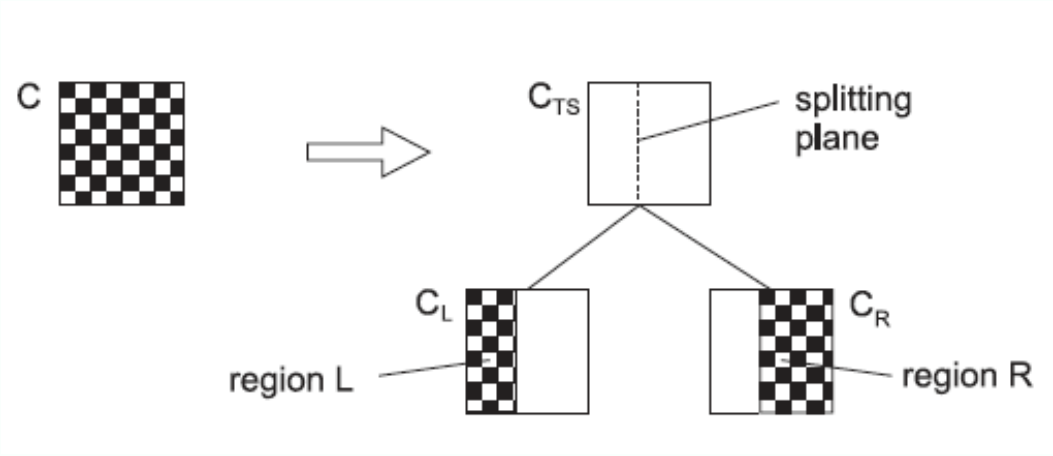


# K-D trees

- Put bounding box around all objects
  - split with coordinate plane (x, y, or z) into two boxes
  - distribute objects into boxes
    - split each child box recursively until stop
- Questions:
  - how do we compute intersections?
    - easy
      - pass ray into children it intersects
      - intersect with objects in leaf nodes
  - what is a good split?
  - how should we stop splitting?

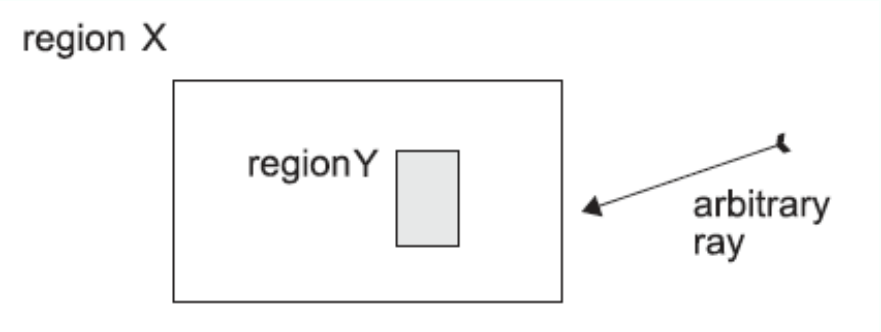
# K-D trees - what is a good split?

- Keep track of intersection costs
  - cheap to intersect with nearly empty boxes
  - expensive to intersect with a box with lots of stuff
  - expensive to look at many small boxes
- Cost of split=
  - Cost of traversal+Cost Left Intersect +Cost Right Intersect
- Need a model for intersect costs



# K-D trees - what is a good split?

- Intersect cost model:
  - Each box contains voxels on some fine grid
  - Filled voxels might be convex
  - If they were, probability of intersection would be ratio of surface areas




$$\text{Expected cost of ray entering box} = \frac{S_y}{S_x} \text{Base cost of intersection}$$

# K-D trees - what is a good split?

- Expected cost of split =
  - expected cost of LHS box+
  - expected cost of RHS box+
  - cost of traversal
- Notice expression does not depend on probability ray visits parent

# K-D trees

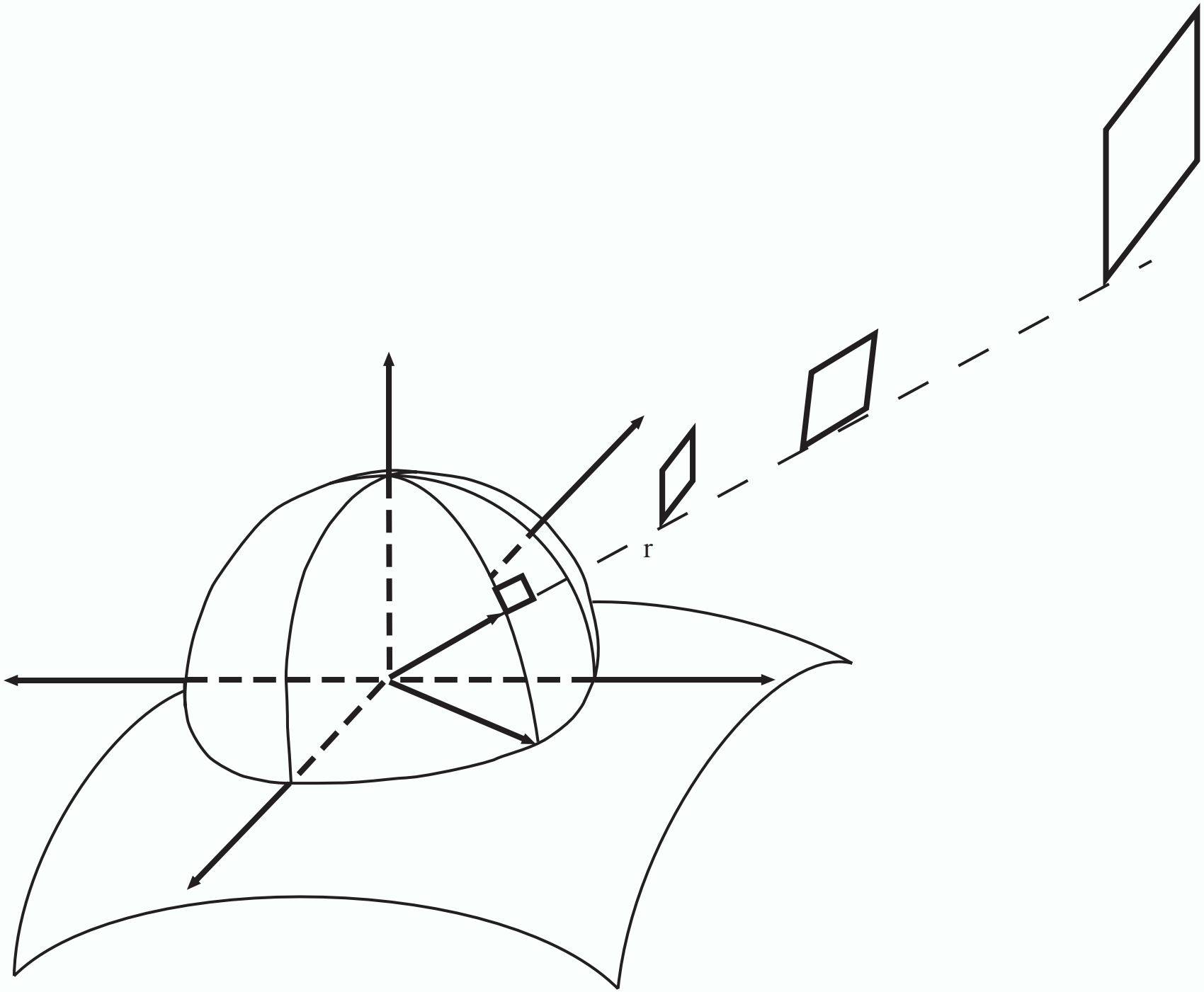
- Splits occur only on planes that bound filled voxels
- Search all splits for lowest cost, using model
- Stopping
  - fixed depth
  - threshold number of objects per voxel
  - both
  - adaptive (i.e. make cost estimate for each leaf, split of each leaf)

http://www.flipcode.com/archives/Raytracing_Topics_Techniques-Part_7_Kd-Trees_and_More_Speed.shtml

# Key idea - how bright is this point?

# Radiometry

- Questions:
  - how “bright” will surfaces be?
  - what is “brightness”?
    - measuring light
    - interactions between light and surfaces
- Core idea - think about light arriving at a surface
  - around any point is a hemisphere of directions
  - what is important is what a source “looks like” to a receiver
    - receiver can’t know anything else about source


r

# Solid angle

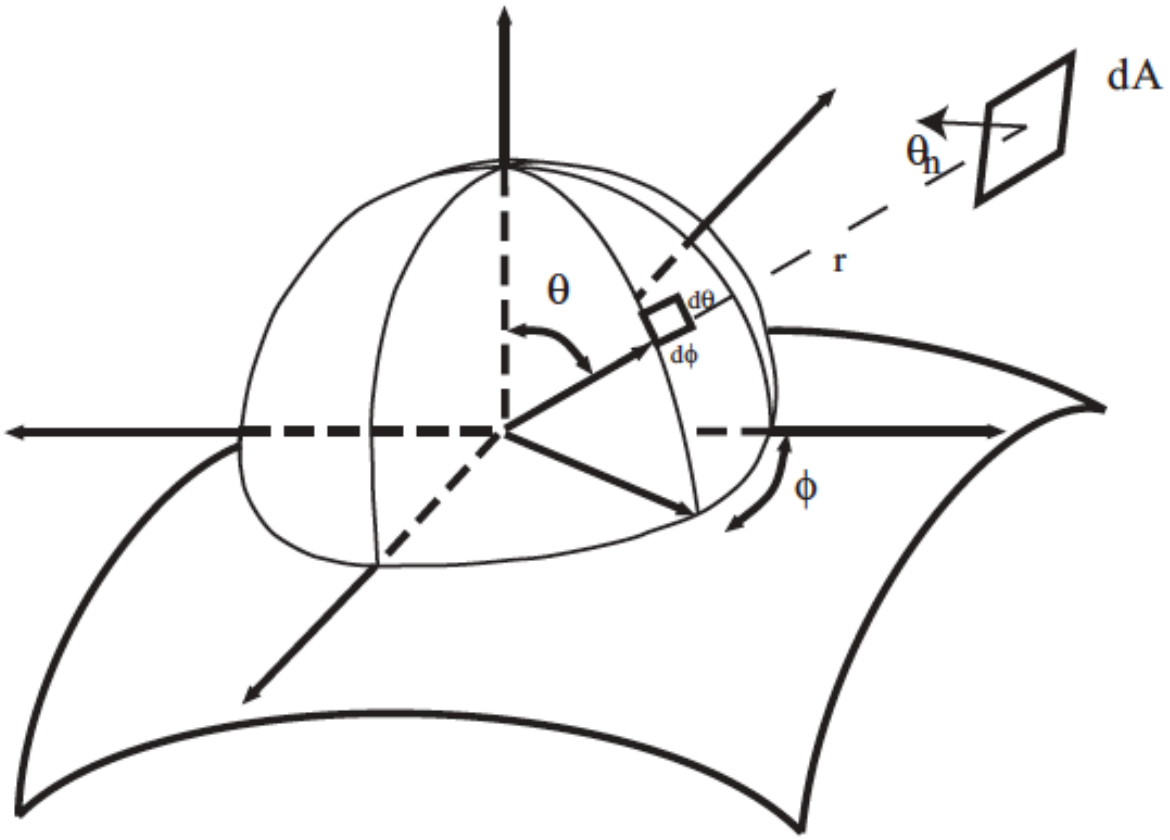



FIGURE 2.15: A hemisphere on a patch of surface, to show our angular coordinates for computing radiometric quantities. The coordinate axes are there to help you see the drawing as a 3D surface. An infinitesimal patch of surface with area $dA$ which is distance $r$ away is projected onto the unit hemisphere centered at the relevant point; the resulting area is the solid angle of the patch, marked as $d\theta d\phi$. In this case, the patch is small so that the area and hence the solid angle is $(1/r^2)dA\cos\theta_n$, where $\theta_n$ is the angle of inclination of the patch.

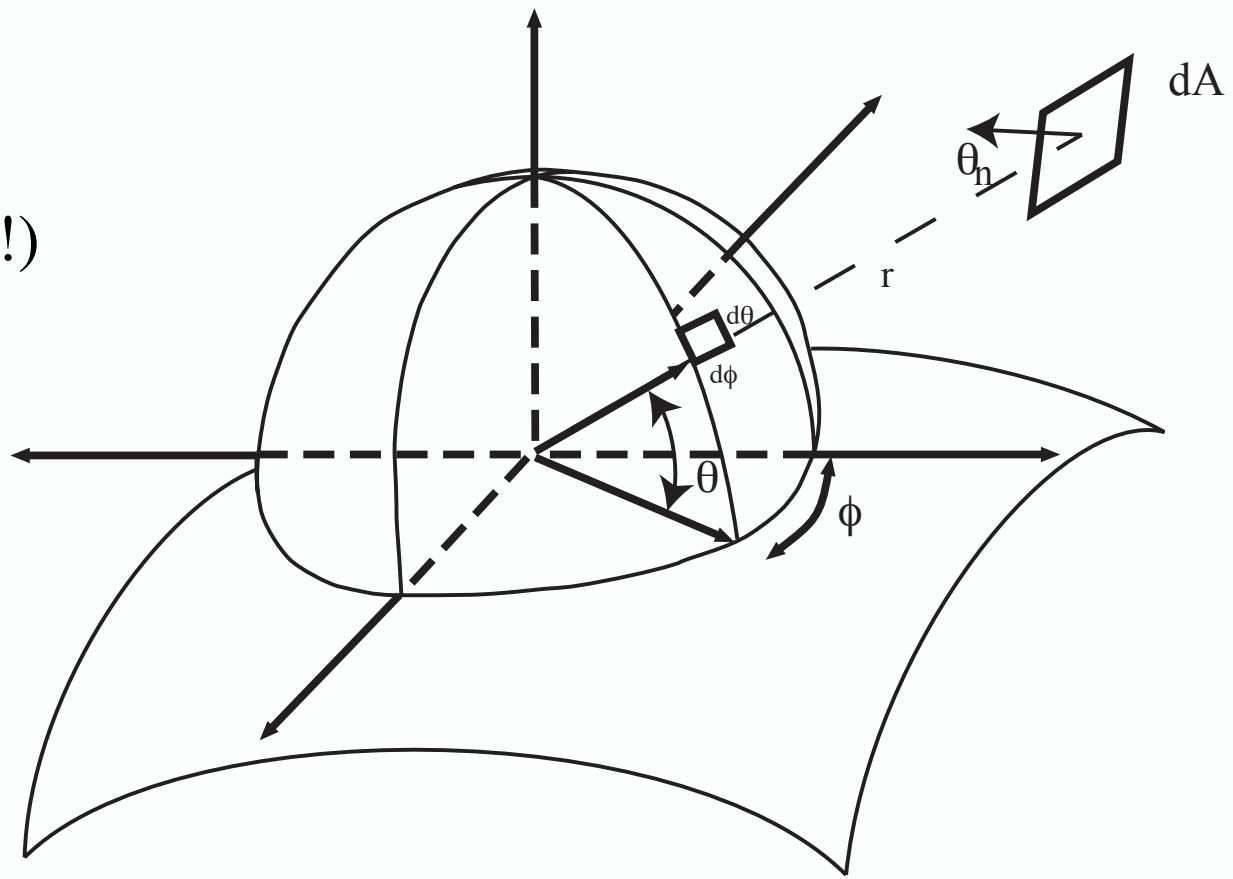
# Solid Angle

- By analogy with angle (in radians)
- The solid angle subtended by a patch area dA is given by

$$d\omega = \frac{\cos\theta_n}{r^2} dA$$

- and (in right coords!)

$$d\omega = \cos\theta d\theta d\phi$$



# Radiance

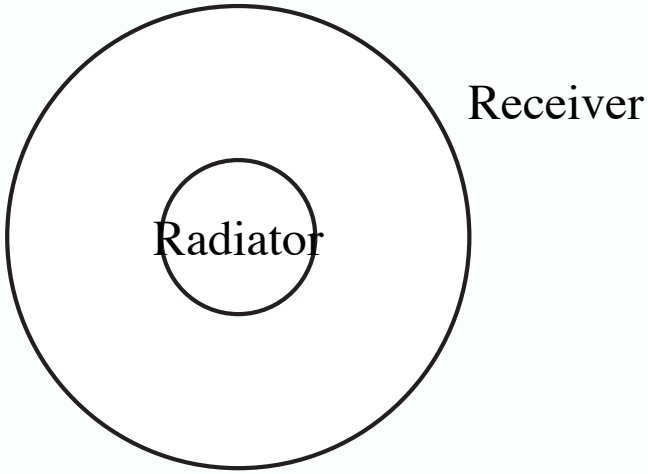
- Measure the "amount of light" at a point, in a direction

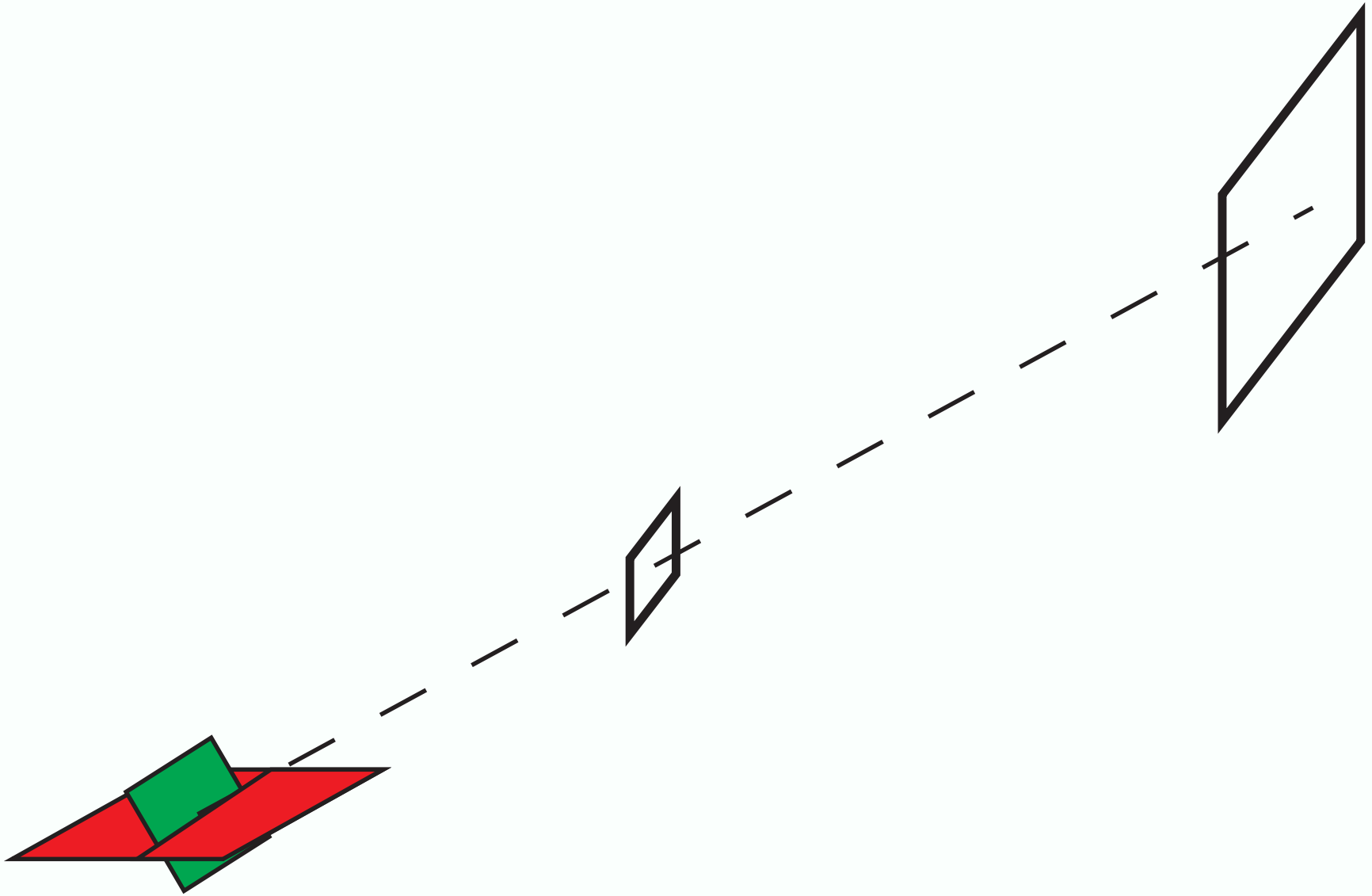
the power (amount of energy per unit time) traveling at some point in a specified direction, per unit area *perpendicular to the direction of travel*, per unit solid angle.

- Units: watts per square meter per steradian (wm-2sr-1)
- Crucial property:
  - In a vacuum, radiance leaving p in the direction of q is the same as radiance arriving at q from p
  - hence the units

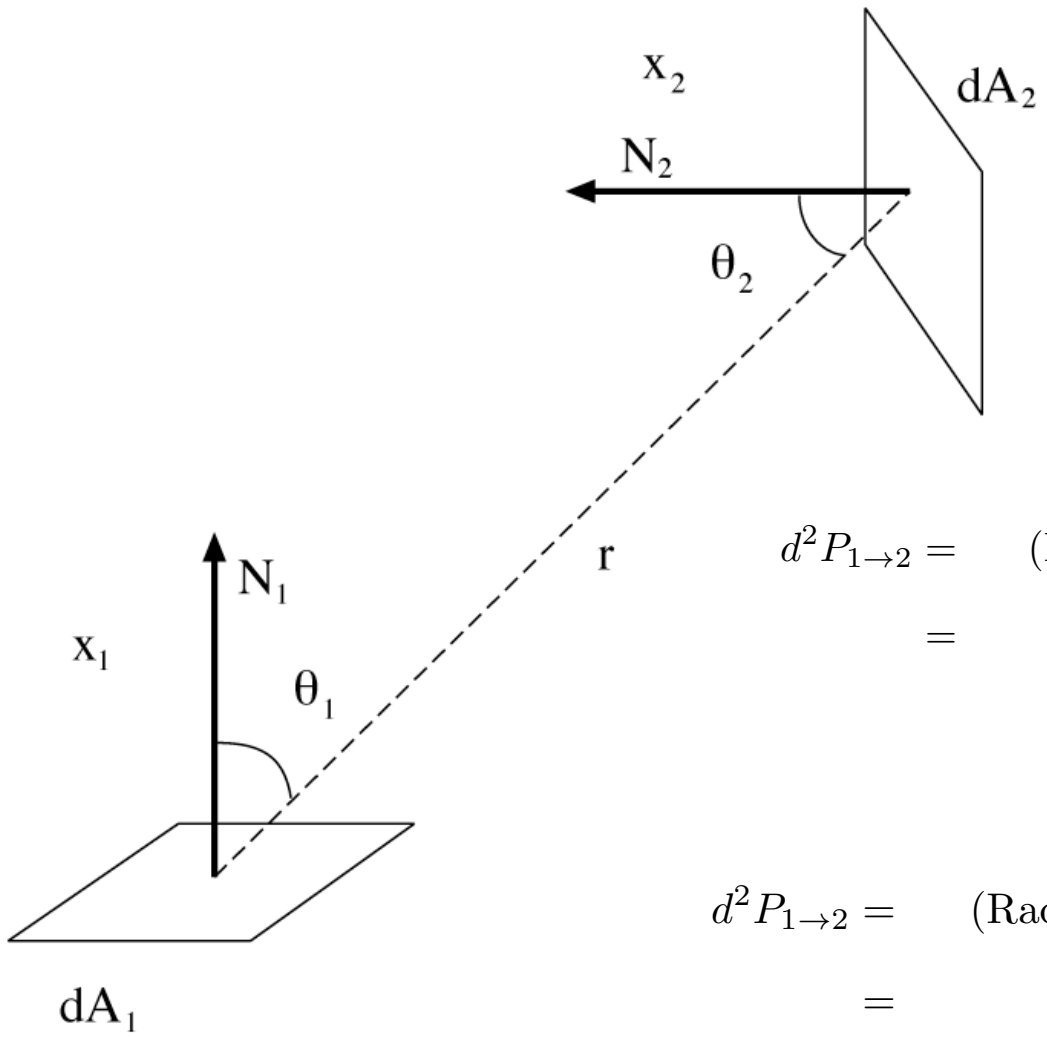
# Why not watts/square meter?

- Consider sphere radiating 1 W into vacuum
  - Radius 1, center at origin
  - Vacuum neither creates nor consumes power
- There's another sphere around it
  - Radius R, center at origin
  - Area - 4 pi R^2
  - It can't collect more power than first sphere radiates so
    - watts/square meter must go down with distance….!!! (ew)





# Radiance is constant along straight lines




Power 1-> 2, leaving 1

$$d^2P_{1\to 2} = \text{(Radiance)(foreshortened area of 1)(solid angle of 2 at 1)}$$

$$= L(\mathbf{x}_1, \mathbf{x}_1 \to \mathbf{x}_2)(\cos\theta_1 dA_1)(\frac{\cos\theta_2}{r^2} dA_2)$$

Power 1-> 2, arriving at 2

$$d^2P_{1\to 2} = \text{(Radiance)(foreshortened area of 2)(solid angle of 1 at 2)}$$

$$= L(\mathbf{x}_2, \mathbf{x}_1 \to \mathbf{x}_2)(\cos\theta_2 dA_2)(\frac{\cos\theta_1}{r^2} dA_1)$$

# Irradiance

- How much light is arriving at a surface?
- Sensible unit is Irradiance
  - Incident power per unit area not foreshortened
  - This is a function of incoming angle.
- A surface experiencing radiance L(x,θ,ϕ) coming in from dω experiences irradiance

$$L(\mathbf{x}, \theta, \phi) \cos\theta d\omega$$

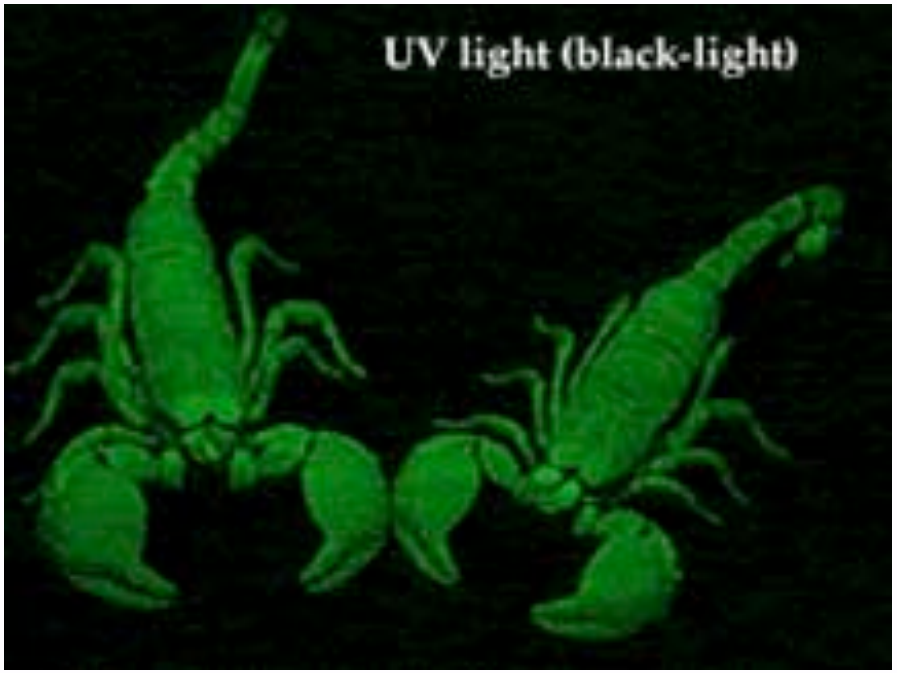
• Crucial property:
Total power arriving at the surface is given by adding irradiance over all incoming angles --- this is why it's a natural unit

# Surfaces and the BRDF

- Many effects when light strikes a surface -- could be:
  - absorbed; transmitted. reflected; scattered
- Assume that
  - surfaces don't fluoresce
  - surfaces don't emit light (i.e. are cool)
  - all the light leaving a point is due to that arriving at that point
- Can model this situation with the Bidirectional Reflectance Distribution Function (BRDF)
- the ratio of the radiance in the outgoing direction to the incident irradiance

$$\rho_{bd}\left(\underline{x},\vartheta_o,\varphi_o,\vartheta_i,\varphi_i,\right) = \frac{L_o\left(\underline{x},\vartheta_o,\varphi_o\right)}{L_i\left(\underline{x},\vartheta_i,\varphi_i\right)\cos\vartheta_i d\omega}$$


White light (indoor)


UV light (black-light)

# BRDF

- Units:  inverse steradians  (sr-1)
- Symmetric in incoming and outgoing directions
- Radiance leaving in a particular direction:
  - add contributions from every incoming direction

$$\int_{\Omega} \rho_{bd}\left(\underline{x},\vartheta_o,\varphi_o,\vartheta_i,\varphi_i,\right) L_i\left(\underline{x},\vartheta_i,\varphi_i\right)\cos\vartheta_i d\omega_i$$

# Suppressing Angles - Radiosity

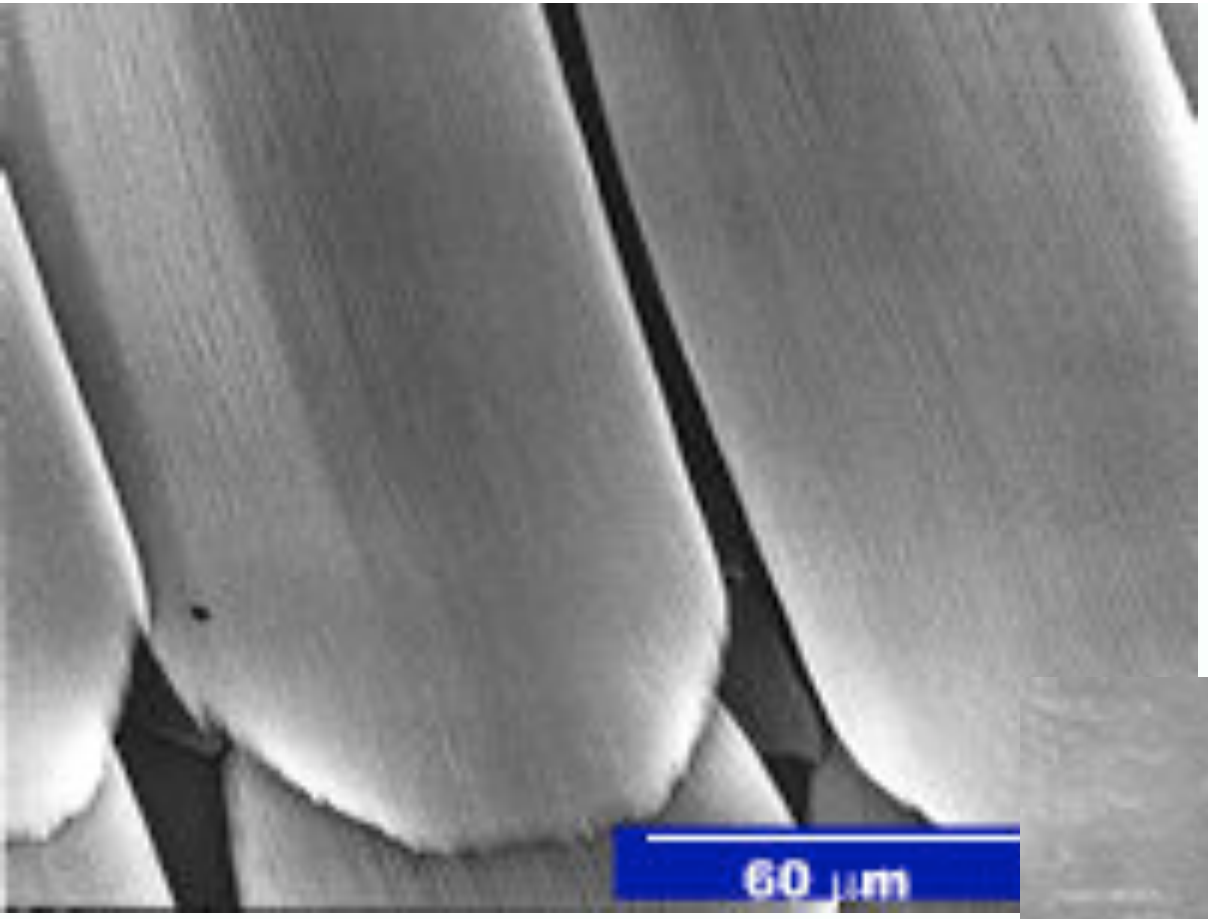
- In many situations, we do not really need angle coordinates
  - e.g. cotton cloth, where the reflected light is not dependent on angle
- Appropriate radiometric unit is radiosity
  - total power leaving a point on the surface, per unit area on the surface (Wm-2)
- Radiosity from radiance?
  - sum radiance leaving surface over all exit directions

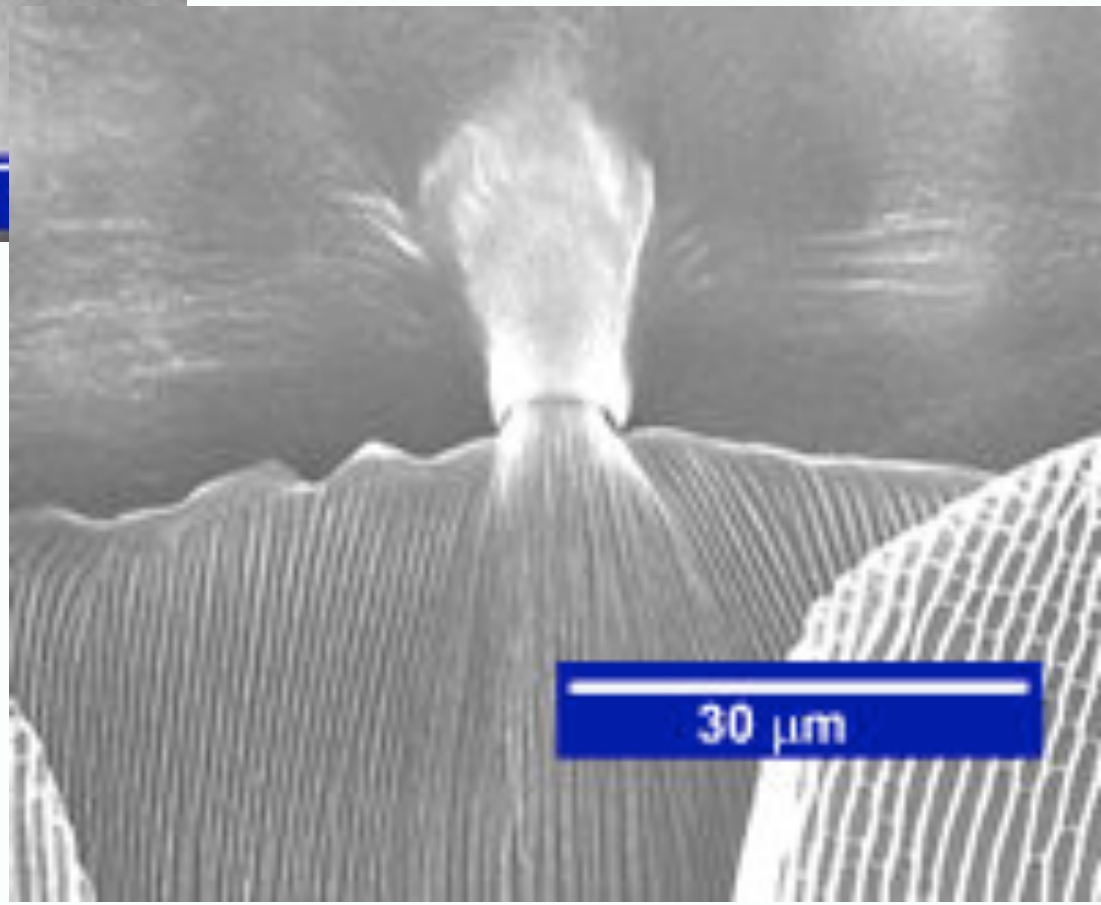
$$B(\underline{x}) = \int_{\Omega} L_o(\underline{x}, \vartheta, \varphi) \cos\vartheta d\omega$$

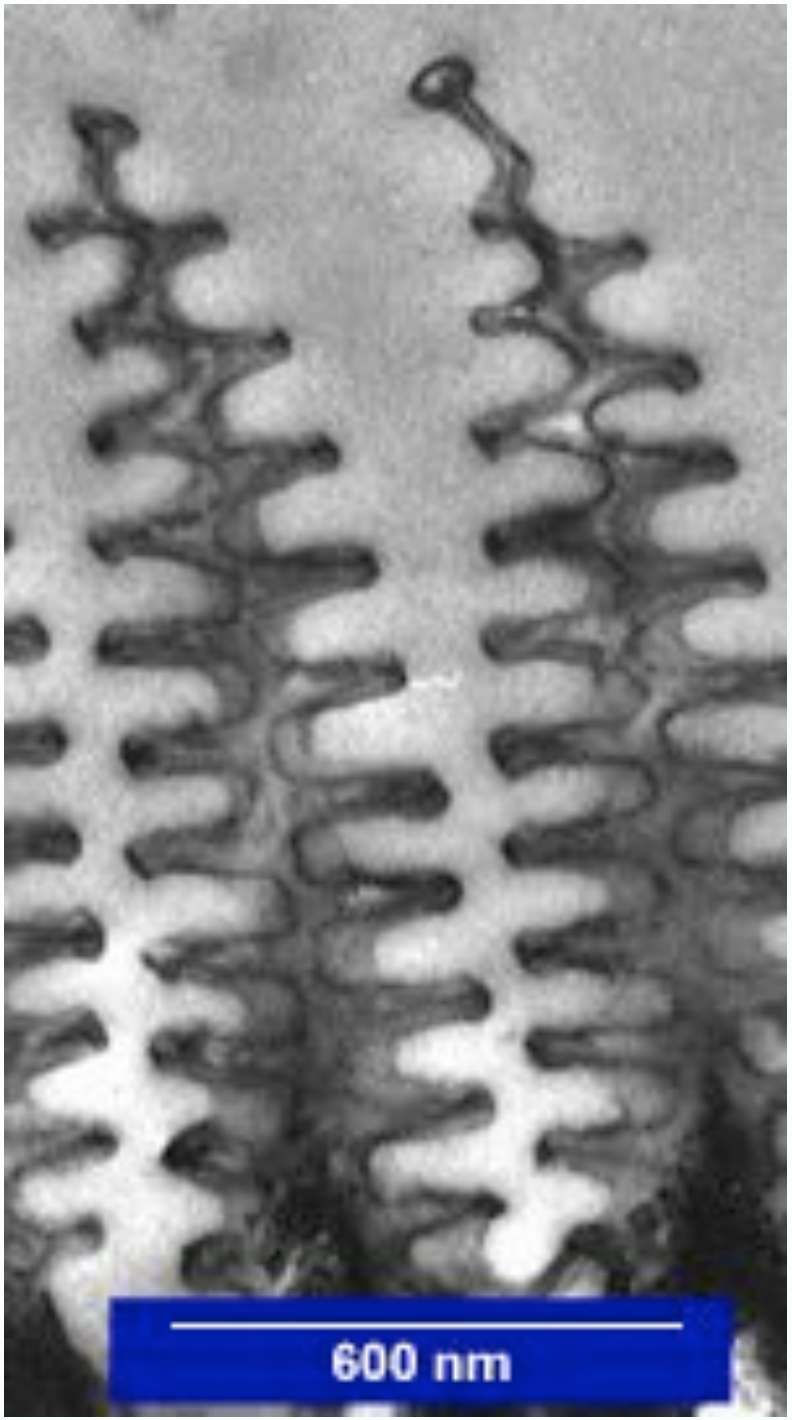
# Radiosity

- Important relationship:
  - radiosity of a surface whose radiance is independent of angle (e.g. that cotton cloth)

$$
\begin{aligned}
B(\underline{x}) &= \int_{\Omega} L_o(\underline{x}, \vartheta, \varphi) \cos\vartheta d\omega \\
&= L_o(\underline{x}) \int_{\Omega} \cos\vartheta d\omega \\
&= L_o(\underline{x}) \int_0^{\pi/2} \int_0^{2\pi} \cos\vartheta \sin\vartheta d\varphi d\vartheta \\
&= \pi L_o(\underline{x})
\end{aligned}
$$


60 µm


30 µm


600 nm

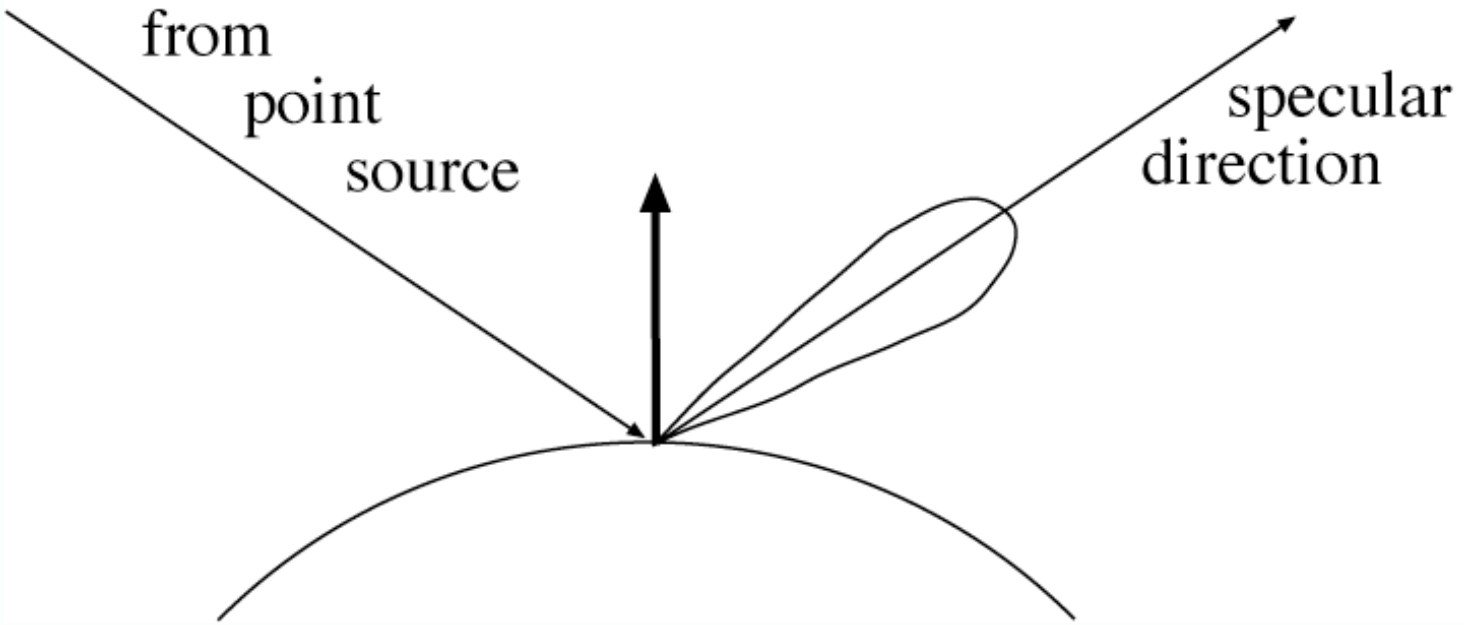
# Lambertian surfaces and albedo

- For some surfaces, the BRDF is independent of direction
  - cotton cloth, carpets, matte paper, matte paints, etc.
  - radiance leaving the surface is independent of angle
  - Lambertian surfaces (same Lambert) or ideal diffuse surfaces
  - Use radiosity as a unit to describe light leaving the surface
  - percentage of incident light reflected is diffuse reflectance or albedo
- Useful fact:

$$\rho_{brdf} = \frac{\rho_d}{\pi}$$

# Specular surfaces

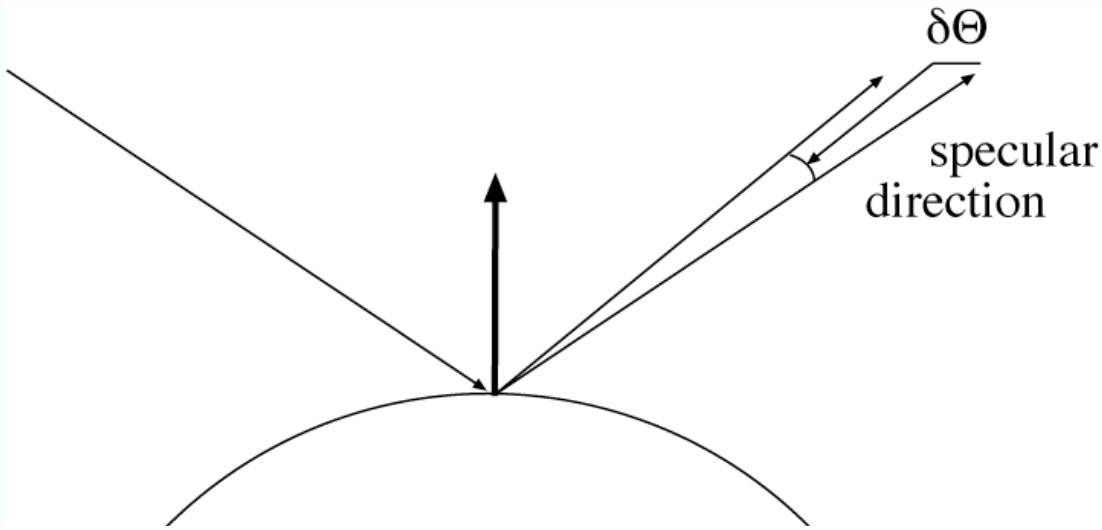
- Another important class of surfaces is specular, or mirror-like.
  - radiation arriving along a direction leaves along the specular direction
  - reflect about normal
  - some fraction is absorbed, some reflected
  - on real surfaces, energy usually goes into a lobe of directions
  - can write a BRDF, but requires the use of funny functions



# Phong's model

- There are very few cases where the exact shape of the specular lobe matters.
- Typically:
  - very, very small --- mirror
  - small -- blurry mirror
  - bigger -- see only light sources as "specularities"
  - very big -- faint specularities
- Phong's model
  - reflected energy falls off with

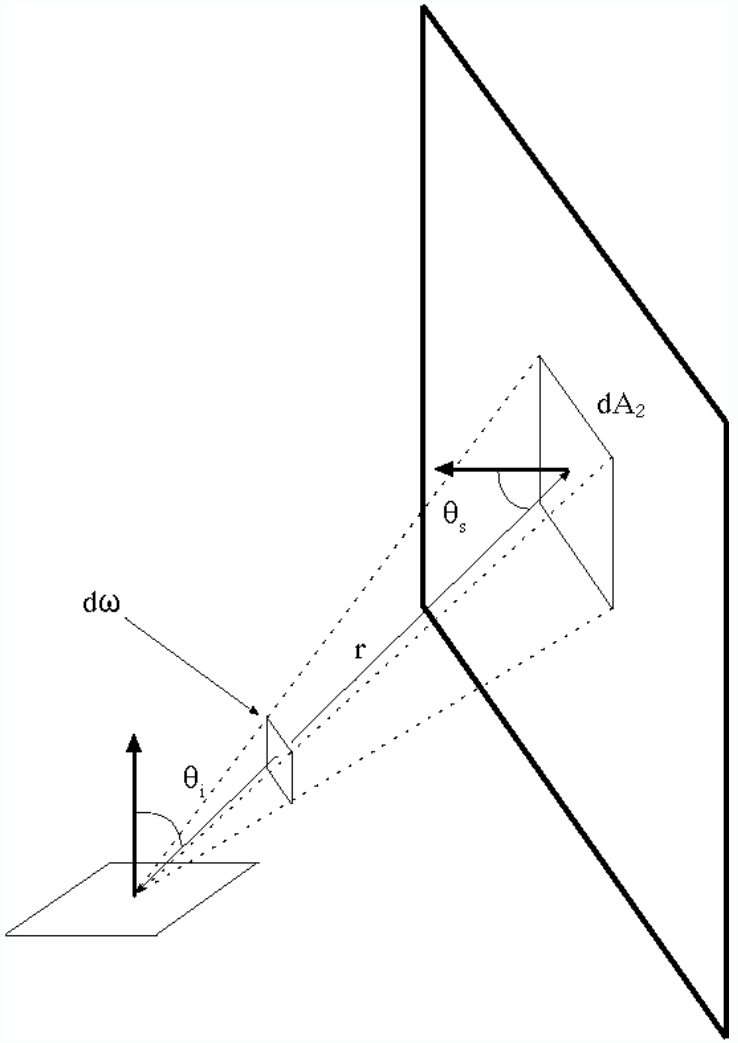
$$\cos^n(\delta\vartheta)$$



# Lambertian + specular

- Widespread model
  - all surfaces are Lambertian plus specular component
- Advantages
  - easy to manipulate
  - very often quite close true
- Disadvantages
  - some surfaces are not
    - e.g. underside of CD's, feathers of many birds, blue spots on many marine crustaceans and fish, most rough surfaces, oil films (skin!), wet surfaces
  - Generally, very little advantage in modelling behaviour of light at a surface in more detail -- it is quite difficult to understand behaviour of L+S surfaces

# Area sources

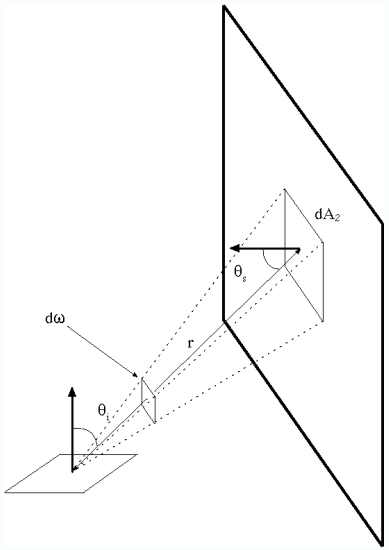



- Examples: diffuser boxes, white walls.
- The radiosity at a point due to an area source is obtained by adding up the contribution over the section of view hemisphere subtended by the source
  - change variables and add up over the source

# Radiosity due to an area source

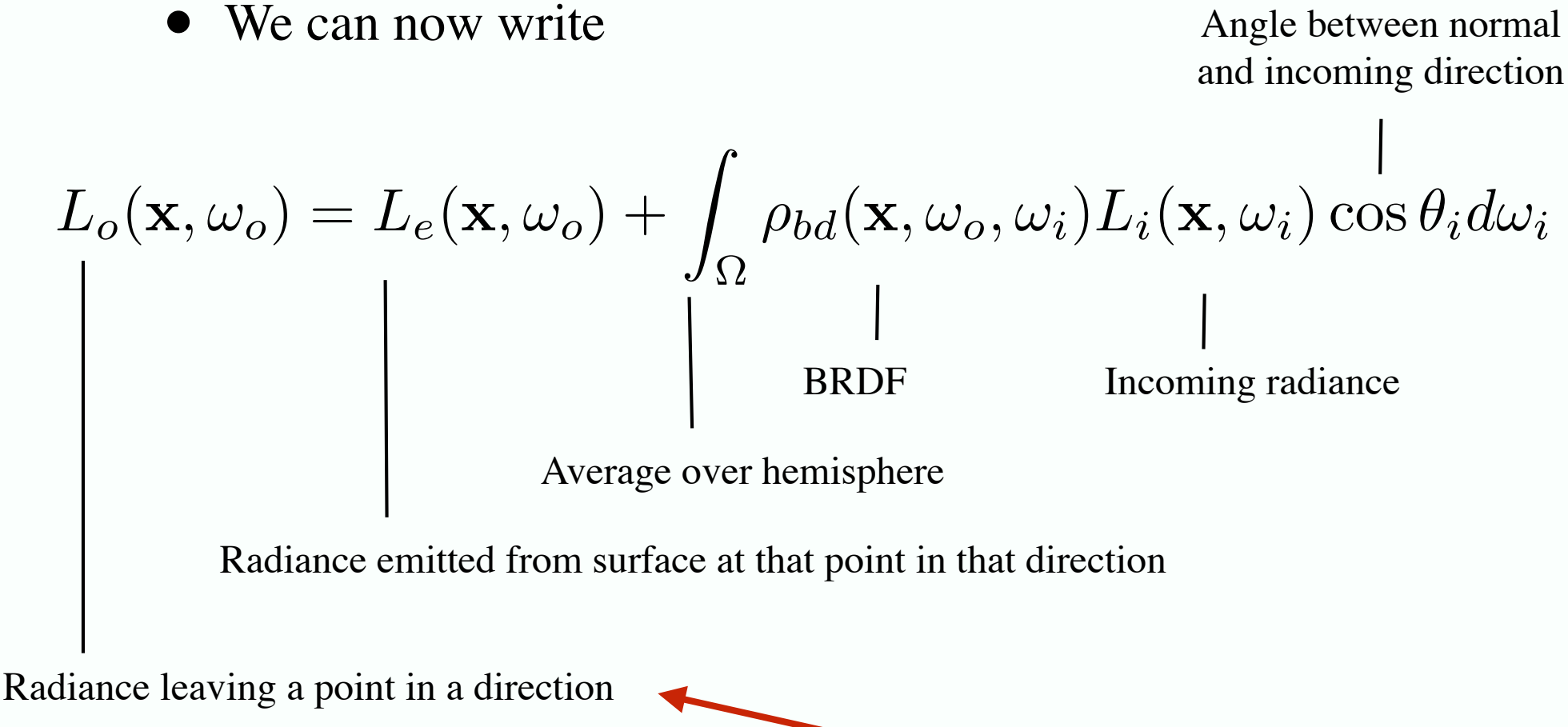
- rho is albedo
- E is exitance
- r(x, u) is distance between points
- u is a coordinate on the source

$$
\begin{aligned}
B(x) &= \rho_d(x)\int_{\Omega} L_i(x, u \rightarrow x)\cos\theta_i d\omega \\
&= \rho_d(x)\int_{\Omega} L_e(x, u \rightarrow x)\cos\theta_i d\omega \\
&= \rho_d(x)\int_{\Omega}\left(\frac{E(u)}{\pi}\right)\cos\theta_i d\omega \\
&= \rho_d(x)\int_{source}\left(\frac{E(u)}{\pi}\right)\cos\theta_i\left(\cos\theta_s\frac{dA_u}{r(x,u)^2}\right) \\
&= \rho_d(x)\int_{source} E(u)\frac{\cos\theta_i\cos\theta_s}{\pi r(x,u)^2}dA_u
\end{aligned}
$$



# The Rendering Equation- 1

- We can now write




Radiance is constant along straight lines, so this is what we want to know

# The Rendering Equation - II

- This balance works for
  - each wavelength,
  - at any time, so
- So

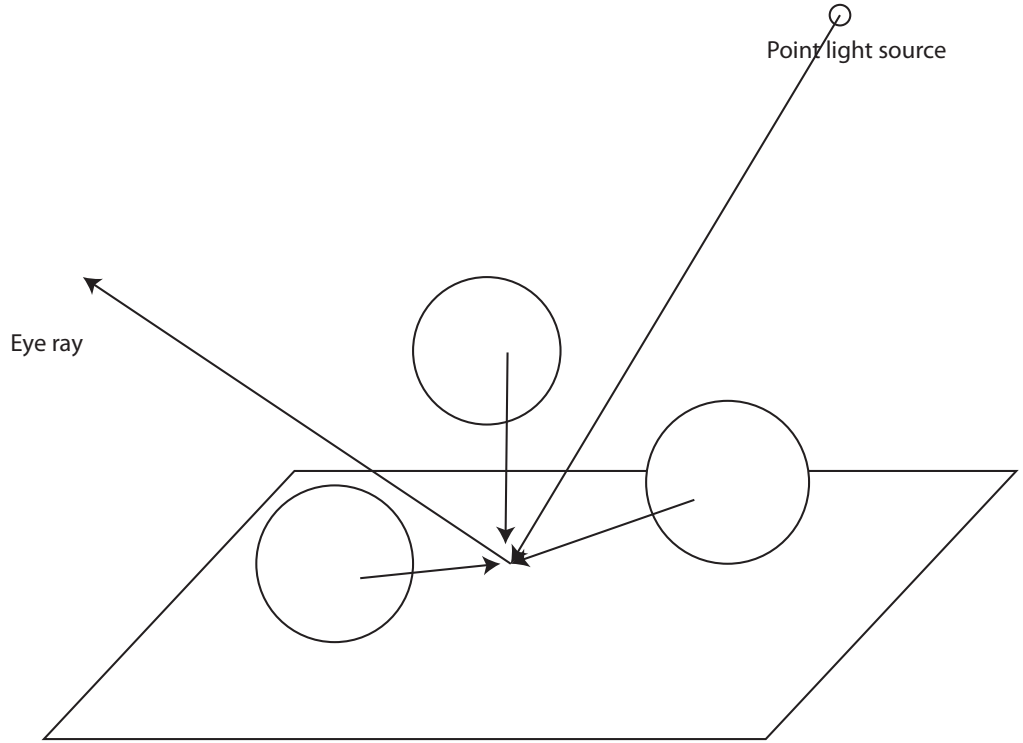
$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega \rho_{bd}(\mathbf{x}, \omega_o, \omega_i, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t) \cos\theta_i d\omega_i$$

# Global illumination

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega \rho_{bd}(\mathbf{x}, \omega_o, \omega_i) L_i(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i$$

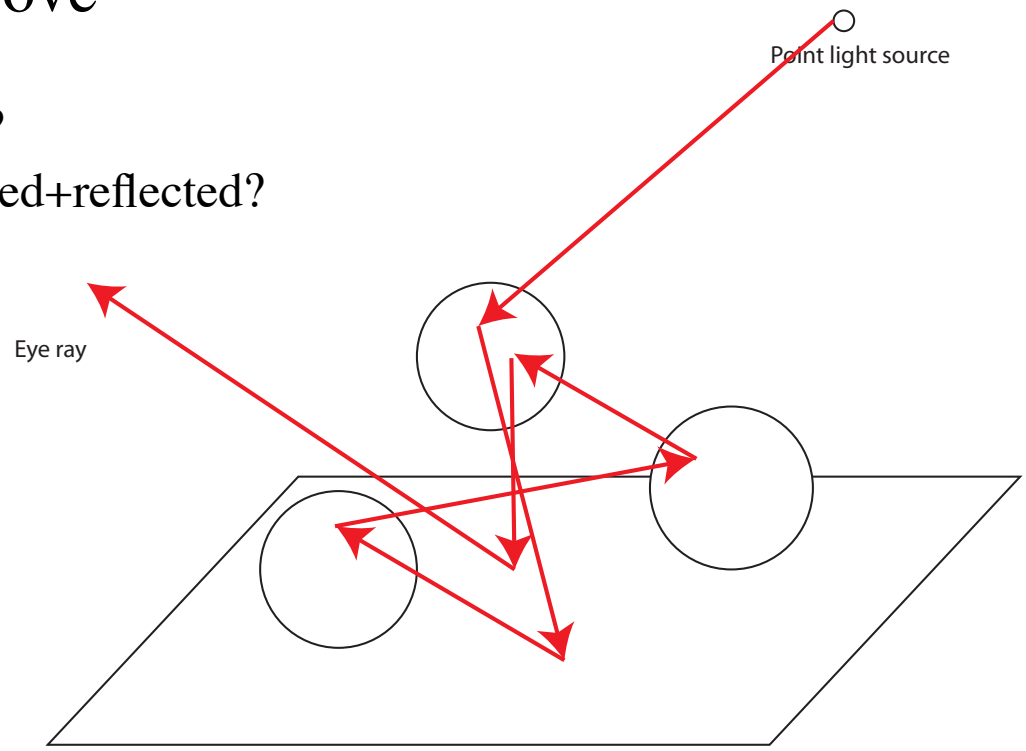
Incoming radiance

- Incoming radiance isn't just from luminaires
  - the reason you can see surfaces is they reflect light
  - other surfaces don't distinguish between reflected light and generated light



# Light paths

- Recursively expand, as above
  - sample the incoming directions
    - what radiance is coming in?
    - go to far end - what is emitted+reflected?
    - recur
-



# Light paths - II

$$v = \int_\Lambda \int_D \int_\Omega \int_T w(\mathbf{x}, \lambda, \omega, t) L(\mathbf{x}, \omega, t) dt d\omega dx d\lambda \approx \sum_{i \in \text{rays}} g(\text{ray}) L(\text{ray}$$

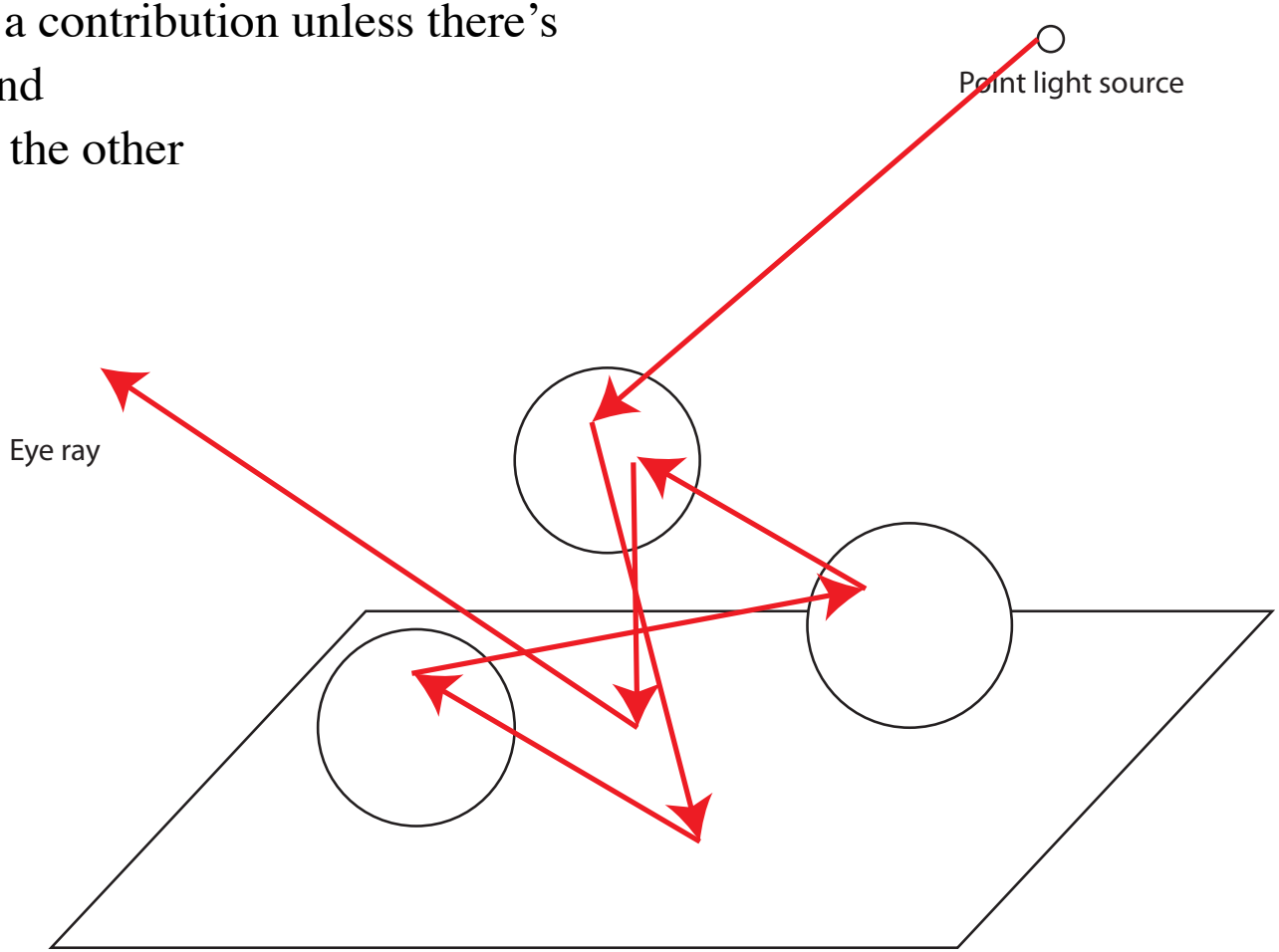
- But this is really (suppressing wavelength and time)

$$\int_D \int_\Omega L(\mathbf{x}, \omega) w(\mathbf{x}, \omega) dx d\omega \approx \frac{1}{N} \sum_{paths} (\text{contribution of path})$$

# Light paths - III

- Now consider contribution of path
  - it doesn't make a contribution unless there's
    - eye at one end
    - luminaire at the other
- We can write

L (Something) E

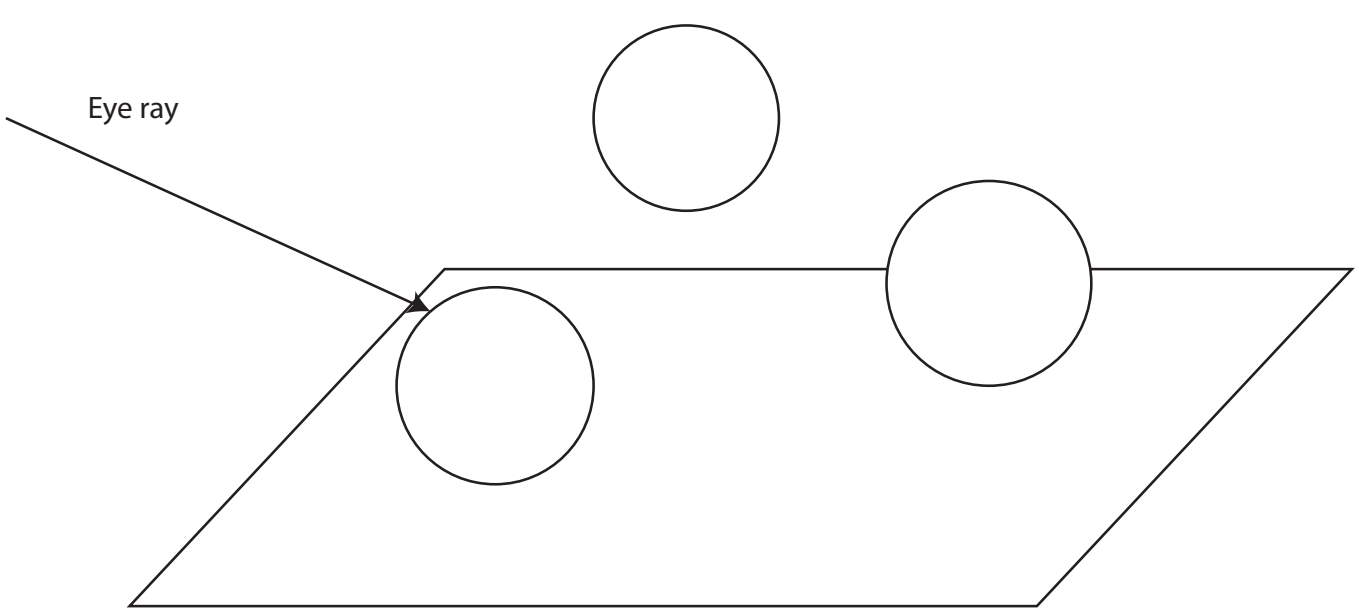


# Some light paths are harder than others

- We have already seen how to render
  - LDE - (light diffuse eye)
    - eye ray to diffuse surface, can it see light?
  - LSE - (light specular eye)
    - eye ray to specular surface, reflect and hit diffuse, can it see light?
  - Actually, can do:
    - LDS*E - (light diffuse 0 or more specular bounces eye)
- How about
  - LDDE - (light diffuse diffuse eye)
    - easy geometry likely high variance
  - LS+DE - (light diffuse at least one specular eye)
    - rather harder

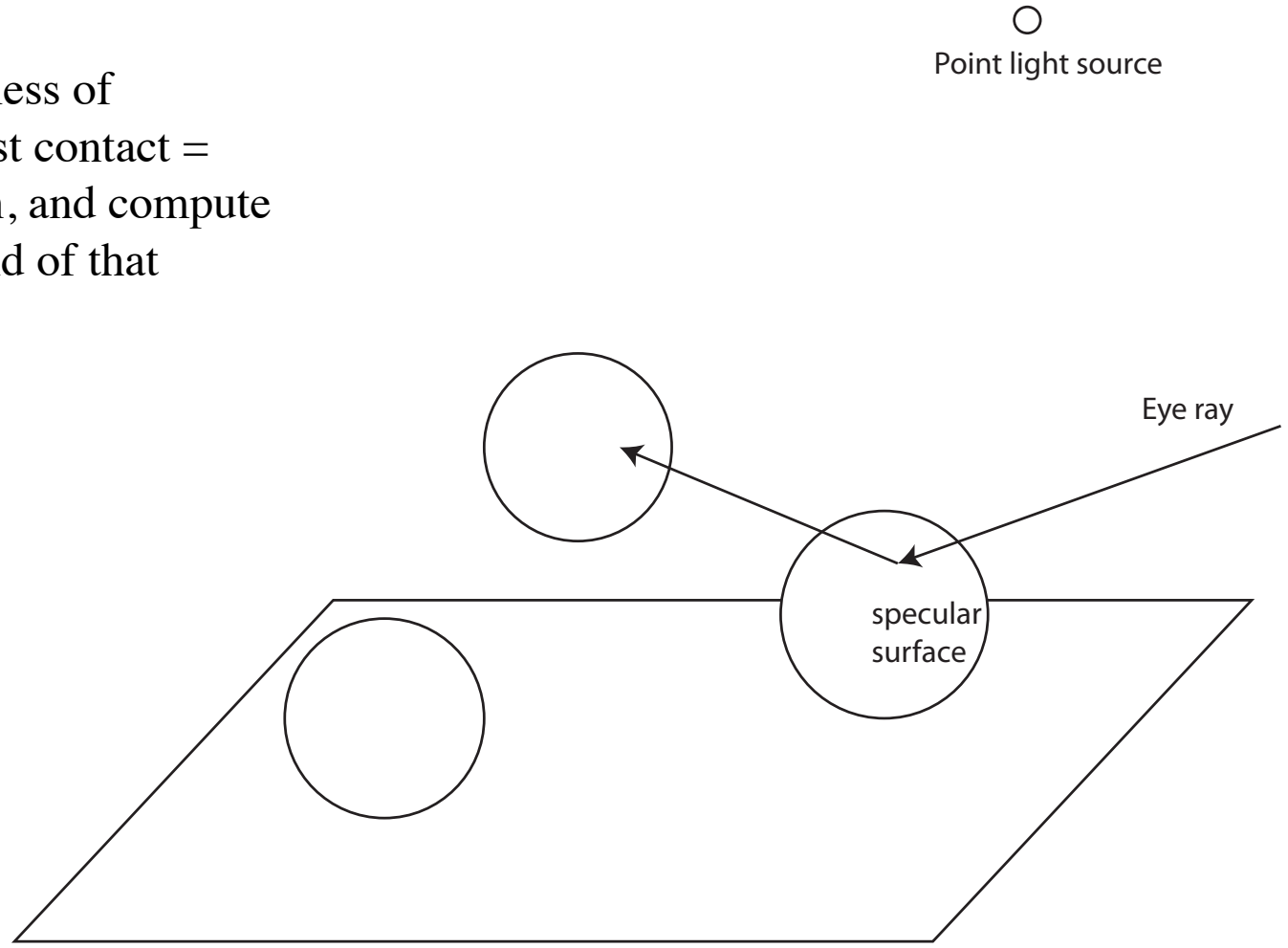
# Eye ray strikes diffuse surface - LDE

Compute brightness of diffuse surface at first contact = Can it see the light sources ?= Is there an object in line segment connecting point to source?

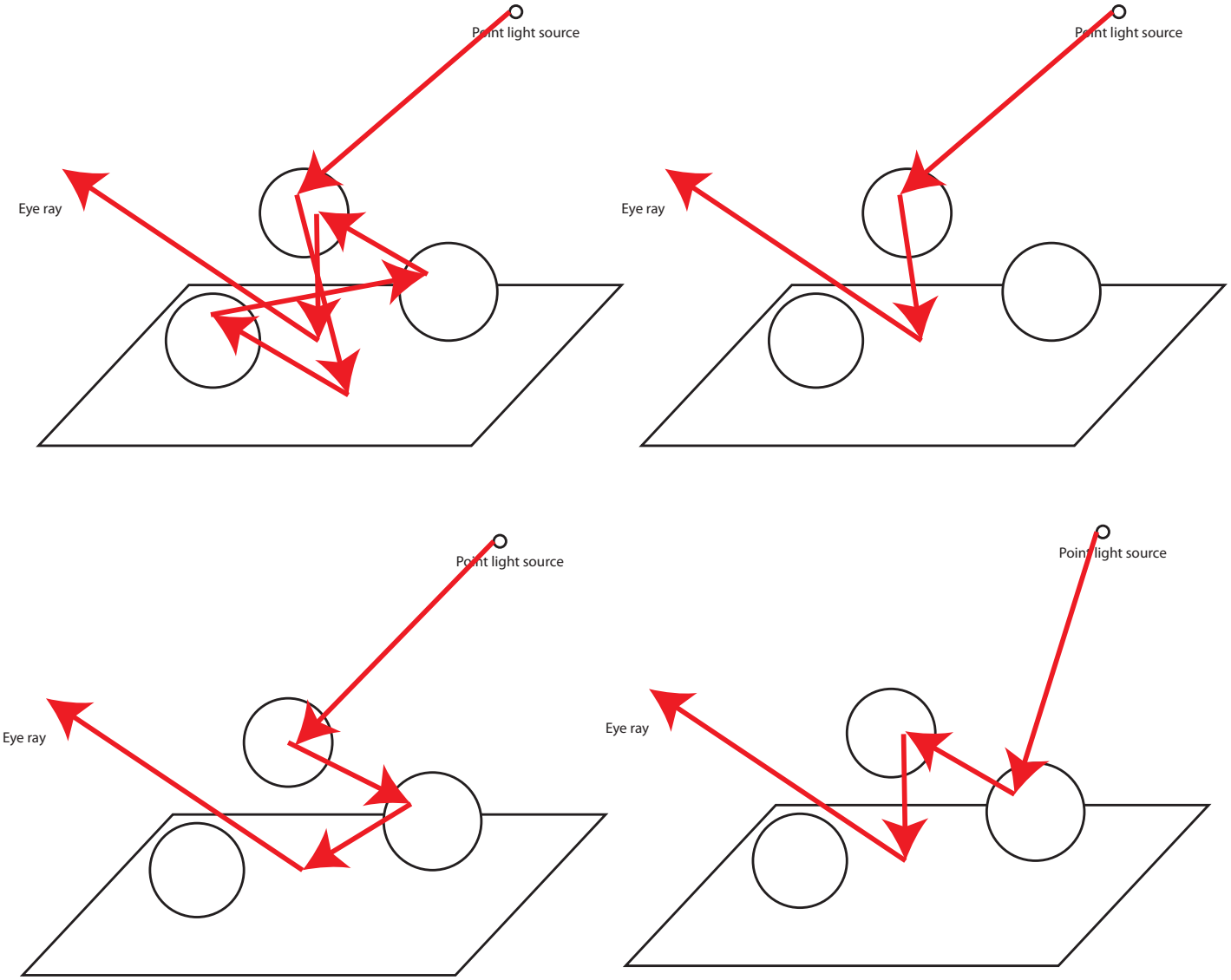


# Eye ray strikes specular surface - LDSE

Compute brightness of specular surface at first contact = eye ray changes direction, and compute brightness at the end of that



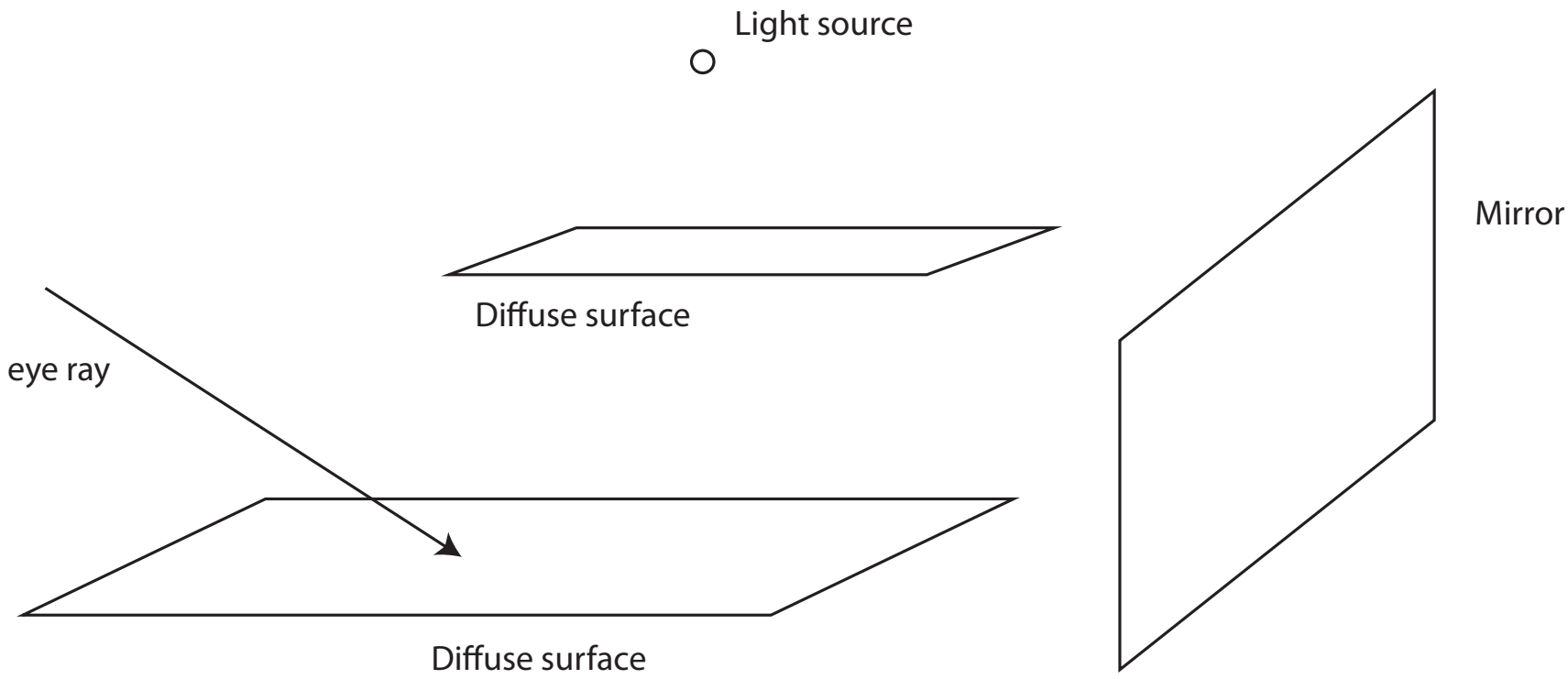
# LDD+E - easy geometry, but variance



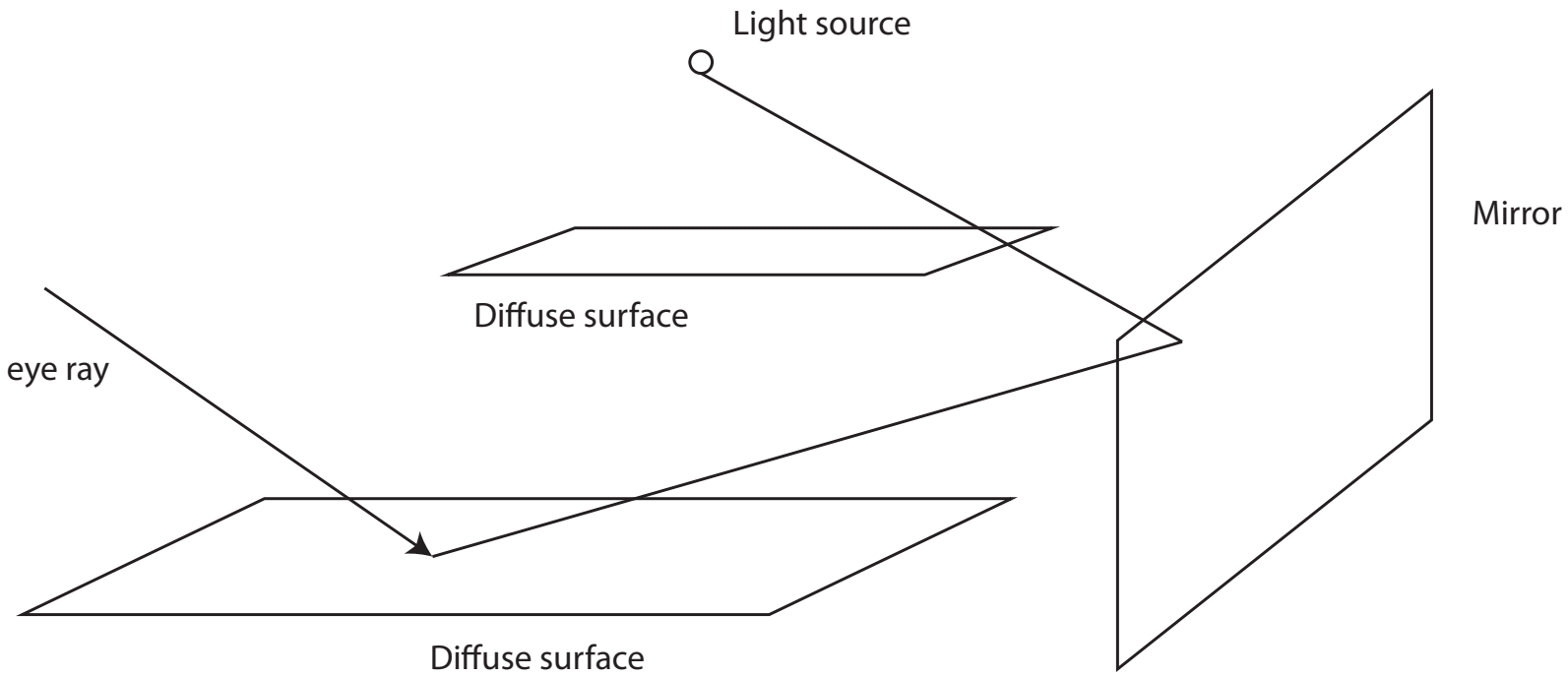
# LDD+E - variance control (sketch!)

- In principle, easily sampled recursively
- Preferentially sample paths that make large contributions
  - these are paths that connect light, eye, via high albedo surfaces
    - “Russian roulette”
      - continue path with probability = albedo
      - weight by (1/albedo)
- Cache results
  - propagating a path:
    - check: is there something in the cache?
      - yes: use it
      - no: propagate path and cache results

# LS+DE - harder - where is the light?



# LS+DE - harder - where is the light?




Problem: which direction leaving diffuse surface will hit the light?

# Strategies

- Bidirectional ray tracing
  - Trace a lot of rays from light through specular surfaces to first diffuse
  - Trace a lot of eye rays to first diffuse
  - Join paths
  - Variance control
    - weight paths as if they'd been found in different ways (Veach +Guibas)
- Markov chain monte carlo
  - take a path and mutate it; reweight contribution of mutated path
- Caching
  - Photon cache - trace many rays from light through specular to diffuse
    - cache at diffuse
    - query with eye ray

# Biased vs unbiased rendering

- Unbiased renderer
  - pixel value is value of random variable (different paths=different values)
  - E(estimate)=True value
  - sometimes essential
    - eg estimate the amount of light in a museaum hall
- Biased renderer
  - E(estimate)=True value + Bias
  - eg Photon map (as above)
    - Bias because we do not know how many photons to stick in cache
  - Often more realistic
- Crucial point
  - Very few people can tell if a render is nearly physically correct
    - and no-one can reliably spot exactness

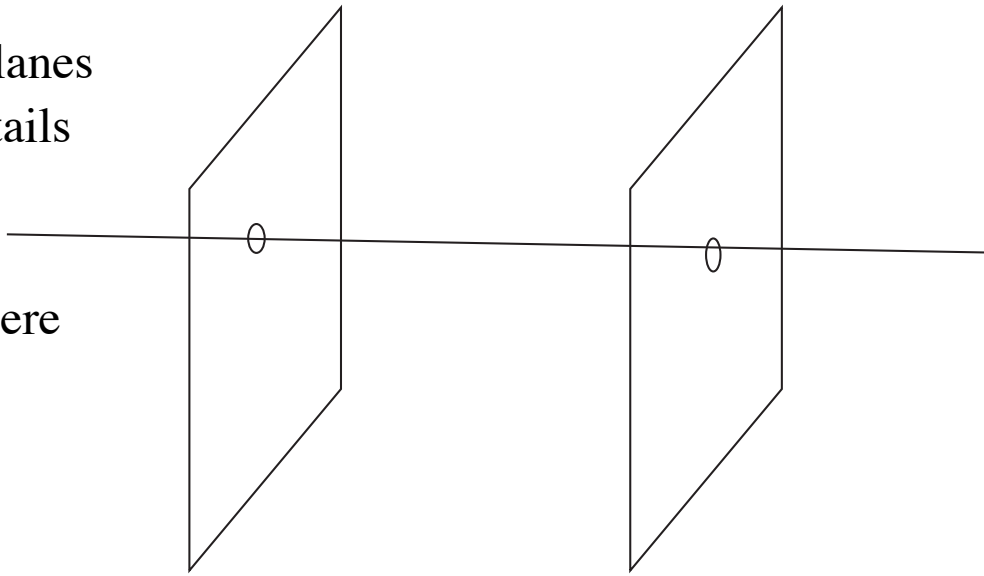
# The plenoptic function

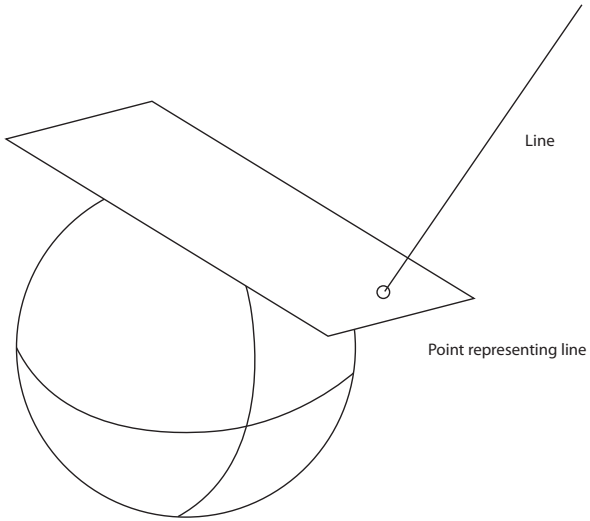
- We are repeatedly sampling radiance
  - as function of point, direction
- What if we had a function that could report that?

- Plenoptic function
  - radiance along a directed line
  - space of lines is rather nastier than you might think

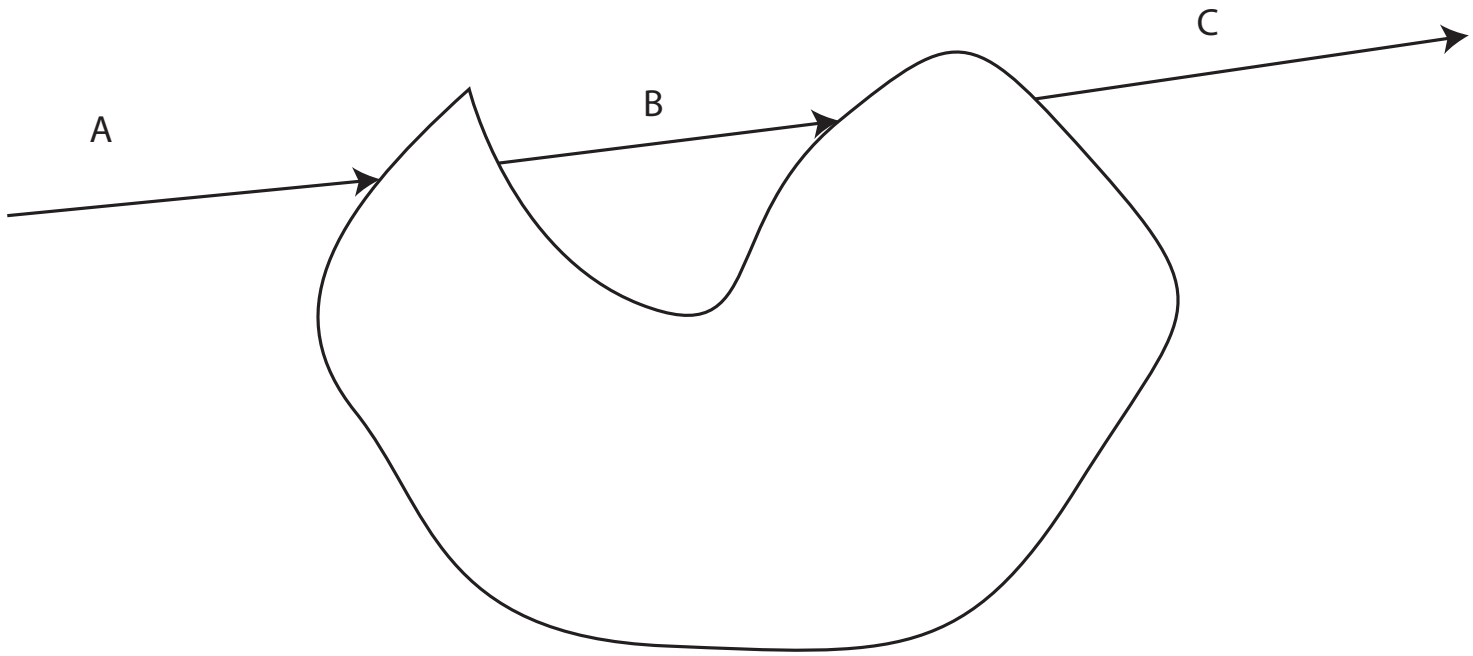
# Lines in 3D (if it's empty!)

- Space of lines is 4 dimensional
  - can specify a line by:
    - where it intersects each of two planes
      - some missing lines, some details
  - alternative
    - directed line
    - point on the tangent plane of sphere



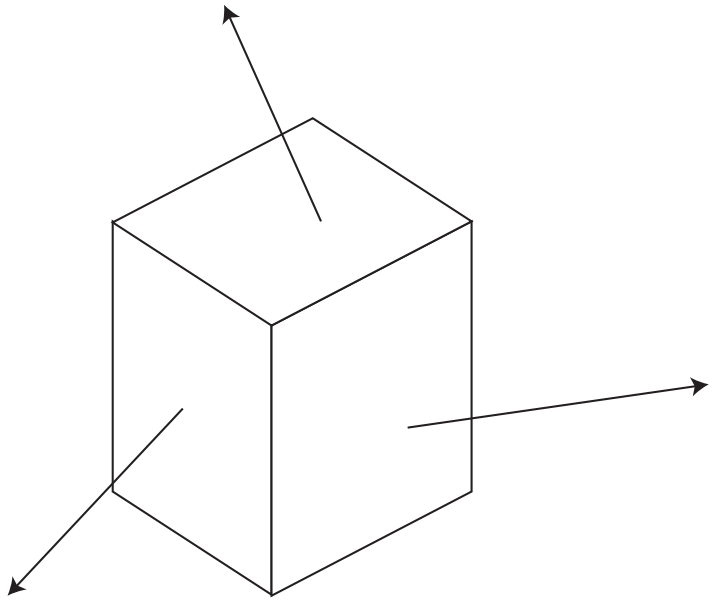


# Lines in 3D with object can be nasty



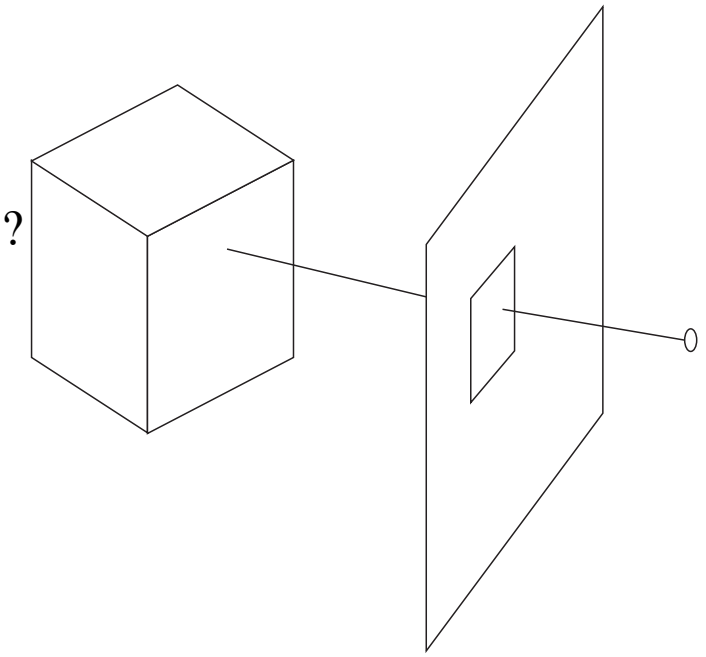
# Simplify

- Place an object in a box
  - record radiance for each ray leaving box
- Easy to ray trace
  - look up eye ray in rays leaving box
    - report that value
- Capture is relatively easy



# Capturing this representation

- Obtain an awful lot of images from calibrated cameras
  - each image is a set of rays leaving the box
  - calibration
    - so you can put all rays into one coordinate system
  - construct some form of index
    - or organize rays
- Issue:
  - we don't sample every ray
  - what if eye ray is between samples?
    - multilinear interpolate



# Light field object representations