Announcements

- Assignment 4 will be out later today
- Problem Set 3 is due today or tomorrow by 9am in my mail box (4th floor NSH)
- How are the machines working out?
  - I have a meeting with Peter Lee and Bob Cosgrove on Wednesday to discuss the future of the cluster

Advanced Ray Tracing

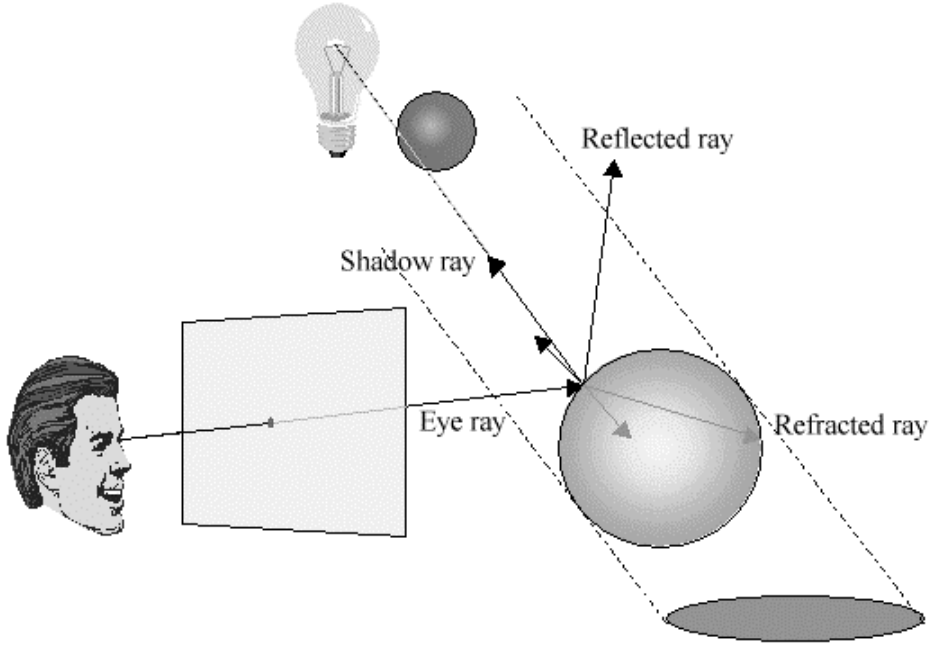
(Recursive) Ray Tracing
Antialiasing
Motion Blur
Distribution Ray Tracing
Other fancy stuff

Assumptions

- Simple shading (typified by OpenGL, z-buffering, and Phong illumination model) assumes:
  - direct illumination (light leaves source, bounces at most once, enters eye)
  - no shadows
  - opaque surfaces
  - point light sources
  - sometimes fog

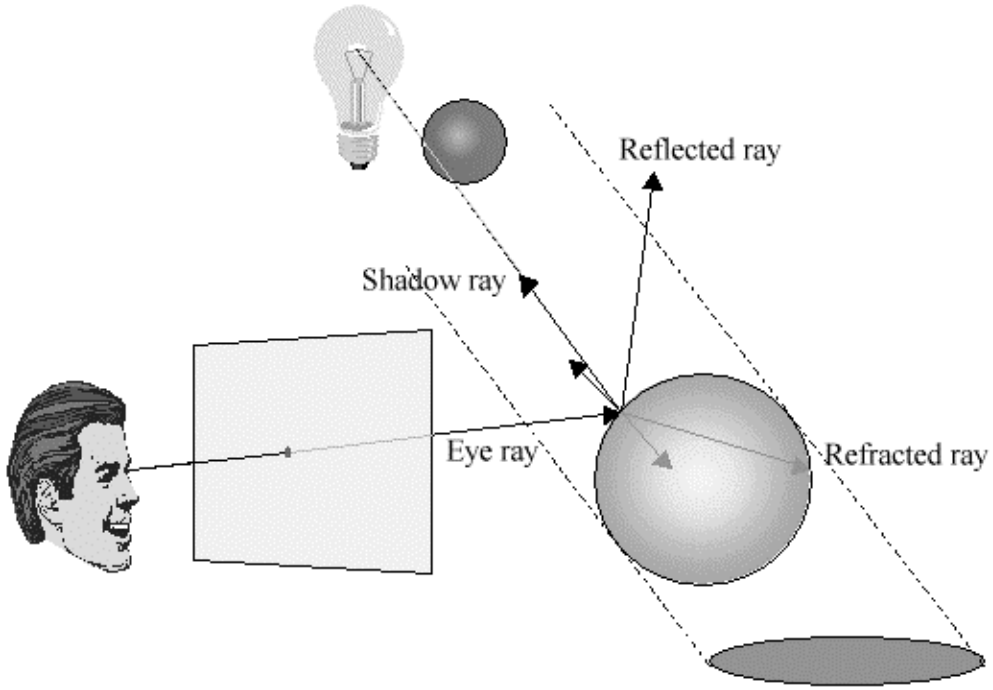
- (Recursive) ray tracing relaxes that, simulating:
  - specular reflection
  - shadows
  - transparent surfaces (transmission with refraction)
  - sometimes indirect illumination (a.k.a. global illumination)
  - sometimes area light sources
  - sometimes fog

Ray Types for Ray Tracing



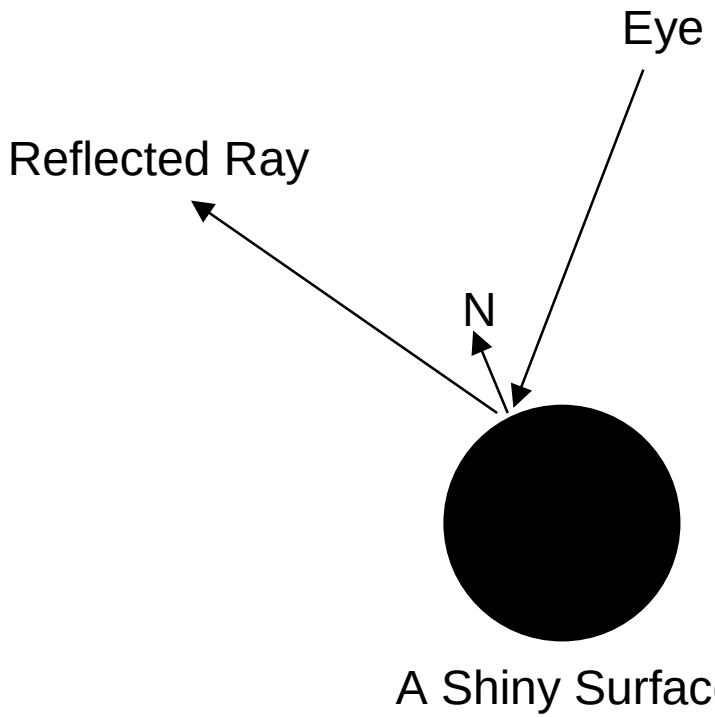
- We'll distinguish four ray types:
  - Eye rays: originate at the eye
  - Shadow rays: from surface point toward light source
  - Reflection rays: from surface point in mirror direction
  - Transmission rays: from surface point in refracted direction

Ray Tracing Algorithm



– send ray from eye through each pixel
– compute point of closest intersection with a scene surface
– shade that point by computing shadow rays
– spawn reflected and refracted rays, repeat

Specular Reflection Rays



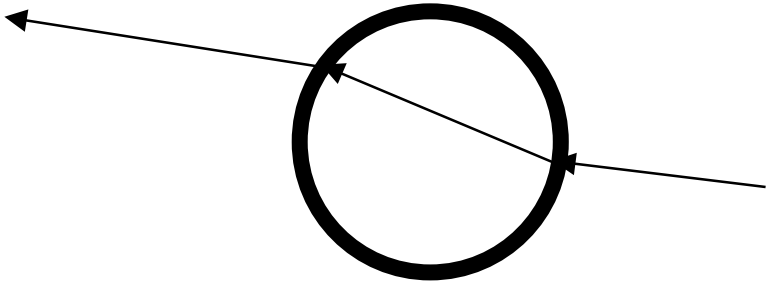
Note: arrowheads show the direction in which we're *tracing the rays,* not the direction the light travels.

An eye ray hits a shiny surface

- We know the direction from which a specular reflection would come, based on the surface normal
- Fire a ray in this reflected direction
- The reflected ray is treated just like an eye ray: it hits surfaces and spawns new rays
- Light flows in the direction opposite to the rays (towards the eye), is used to calculate shading
- It's easy to calculate the reflected ray direction

Specular Transmission Rays

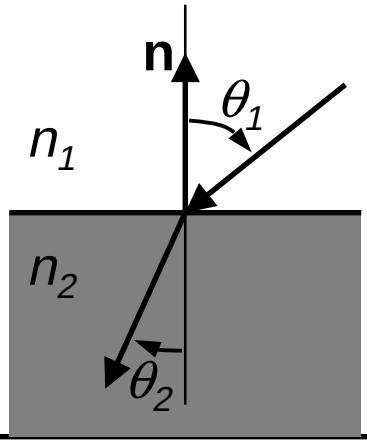
- To add transparency:
  - Add a term for light that's coming from within the object
  - These rays are refracted (bent) when passing through a boundary between two media with different refractive indices
  - When a ray hits a transparent surface fire a *transmission ray* into the object at the proper refracted angle
  - If the ray passes through the other side of the object then it bends again (the other way)



Refraction

- Refraction:
  - The bending of light due to its different velocities through different materials
  - rays bend toward the normal when going from sparser to denser materials (e.g. air to water), away from normal in opposite case
- Refractive index:
  - Light travels at speed c/n in a material of refractive index n
    - » c is the speed of light in a vacuum
    - » c varies with wavelength, hence rainbows and prisms
  - Use Snell's law $n_1 \sin \theta_1 = n_2 \sin \theta_2$ to derive refracted ray direction

MATERIAL	INDEX OF REFRACTION
air/vacuum	1
water	1.33
glass	about 1.5
diamond	2.4

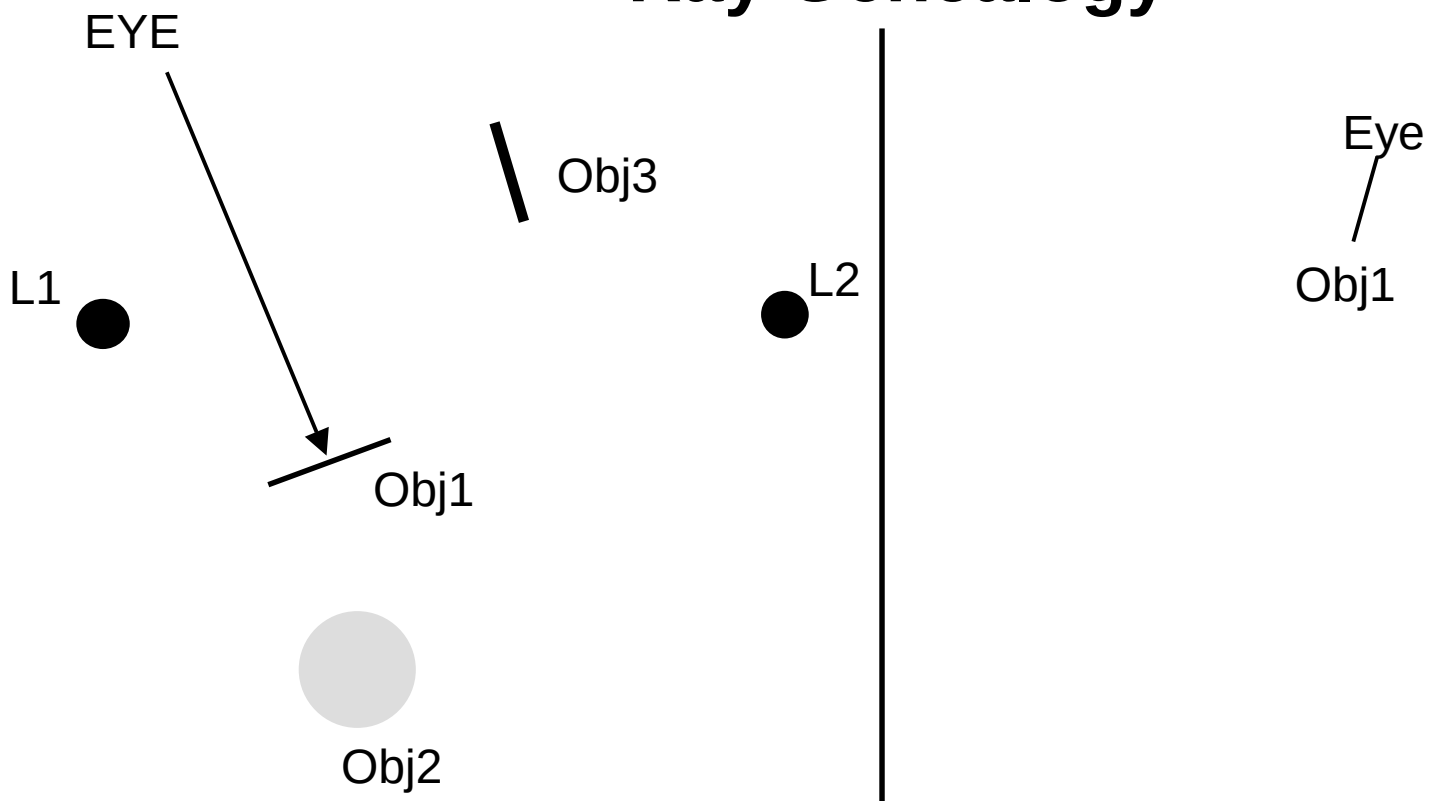


Refraction—Demo program

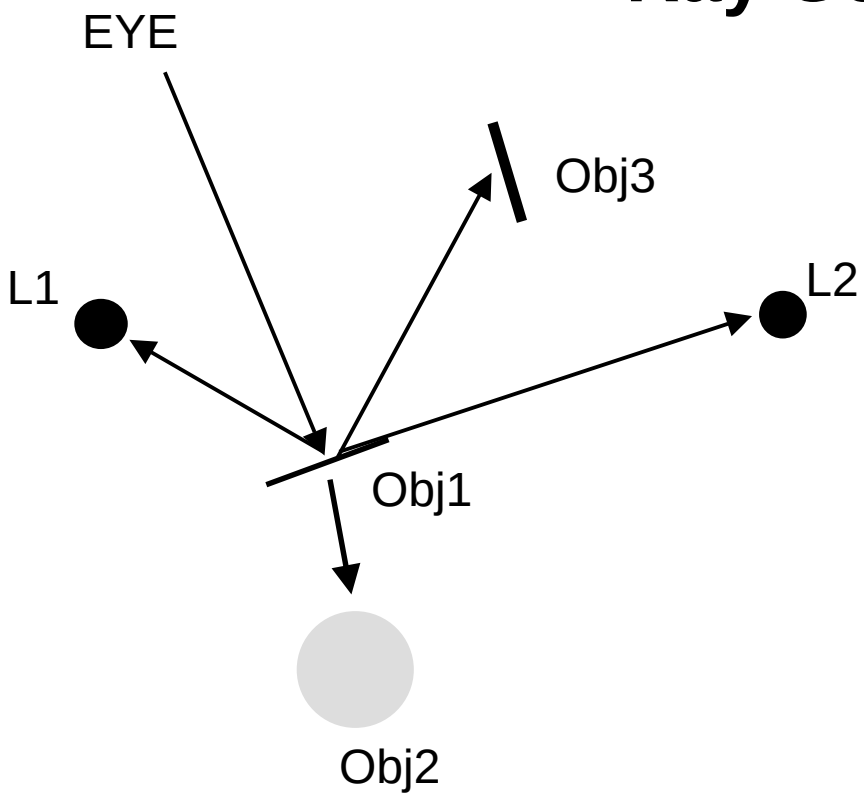
http://micro.magnet.fsu.edu/primer/java/refraction/index.html

http://stwww.weizmann.ac.il/Lasers/laserweb/Java/Twoangles2.htm

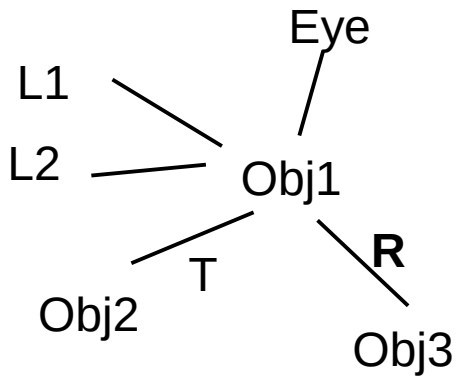
Ray Genealogy



Ray Genealogy

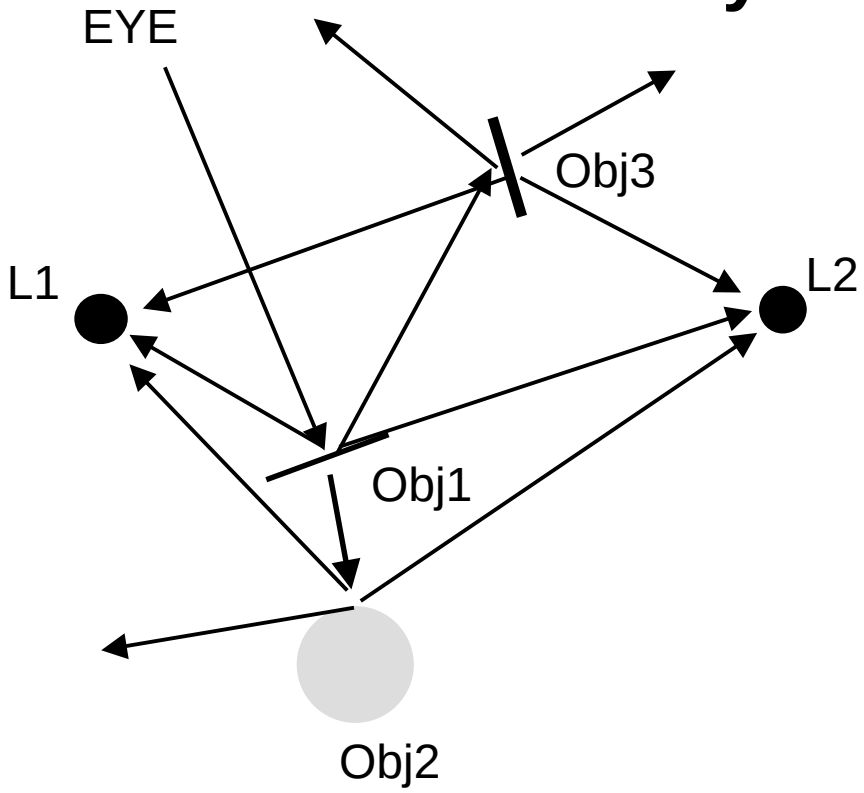


RAY PATHS (BACKWARD)

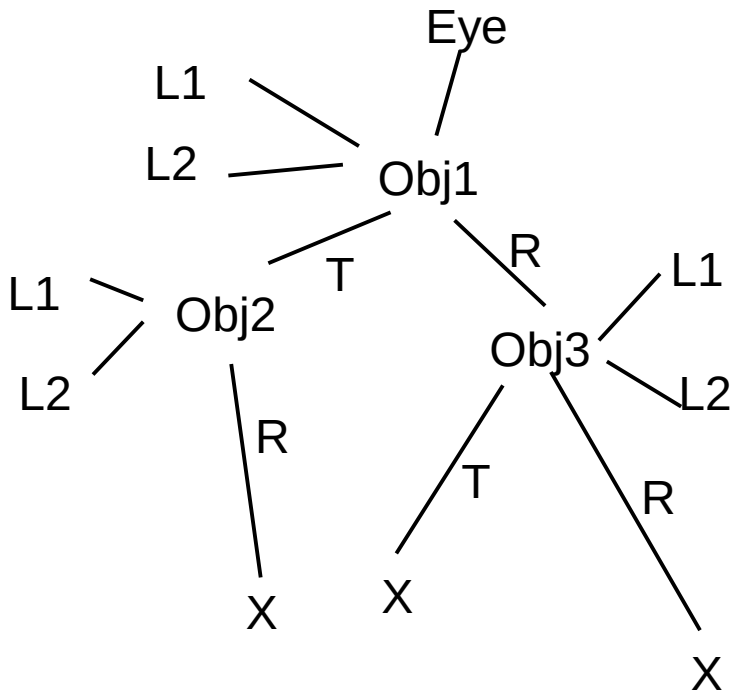


RAY TREE

Ray Genealogy

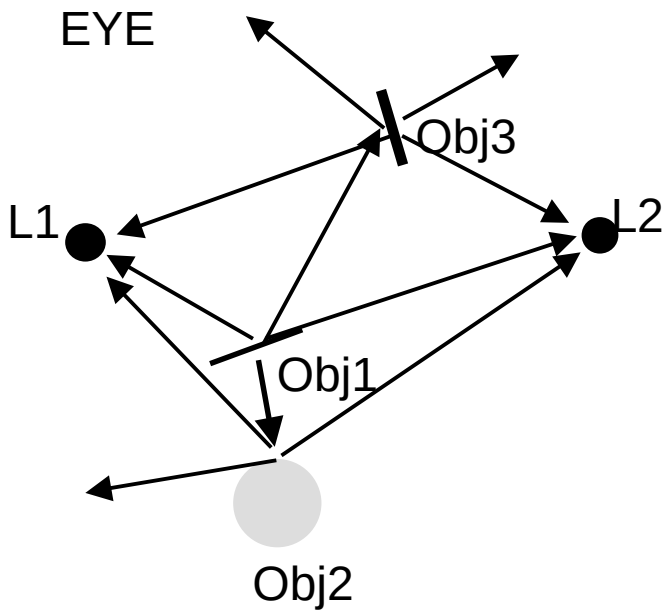


RAY PATHS (BACKWARD)



RAY TREE

When to stop?

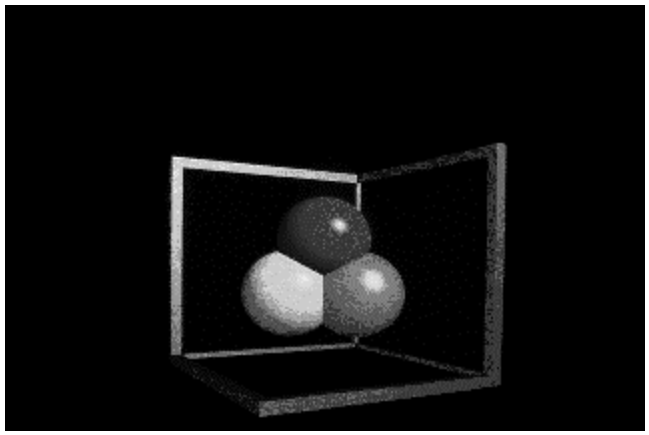


When a ray leaves the scene

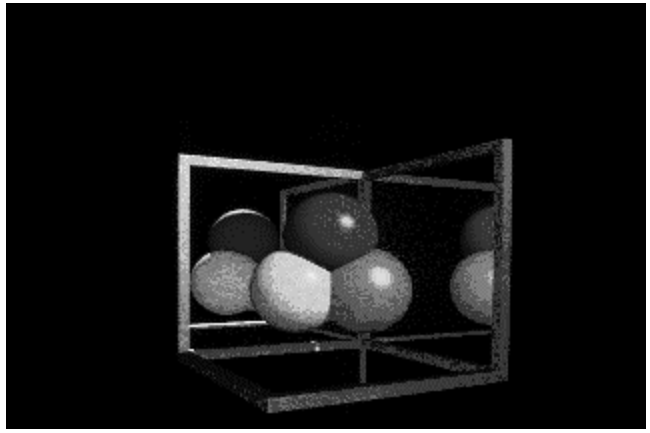
When its contribution becomes small—at each step the contribution is attenuated by the K's in the illumination model.

$$I = k_a I_a + f_{att} I_{light} \left[k_d \cos\theta + k_s (\cos\phi)^{n_{shiny}} \right]$$

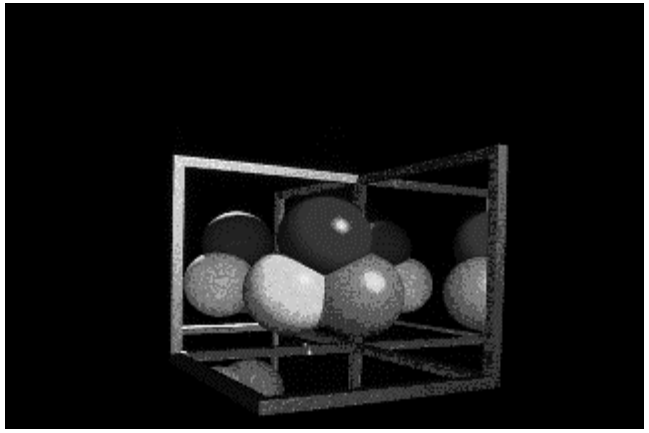
Ray Casting vs. Ray Tracing



Ray Casting -- 1 bounce



Ray Tracing -- 2 bounce



Ray Tracing -- 3 bounce

Writing a Simple Ray Tracer

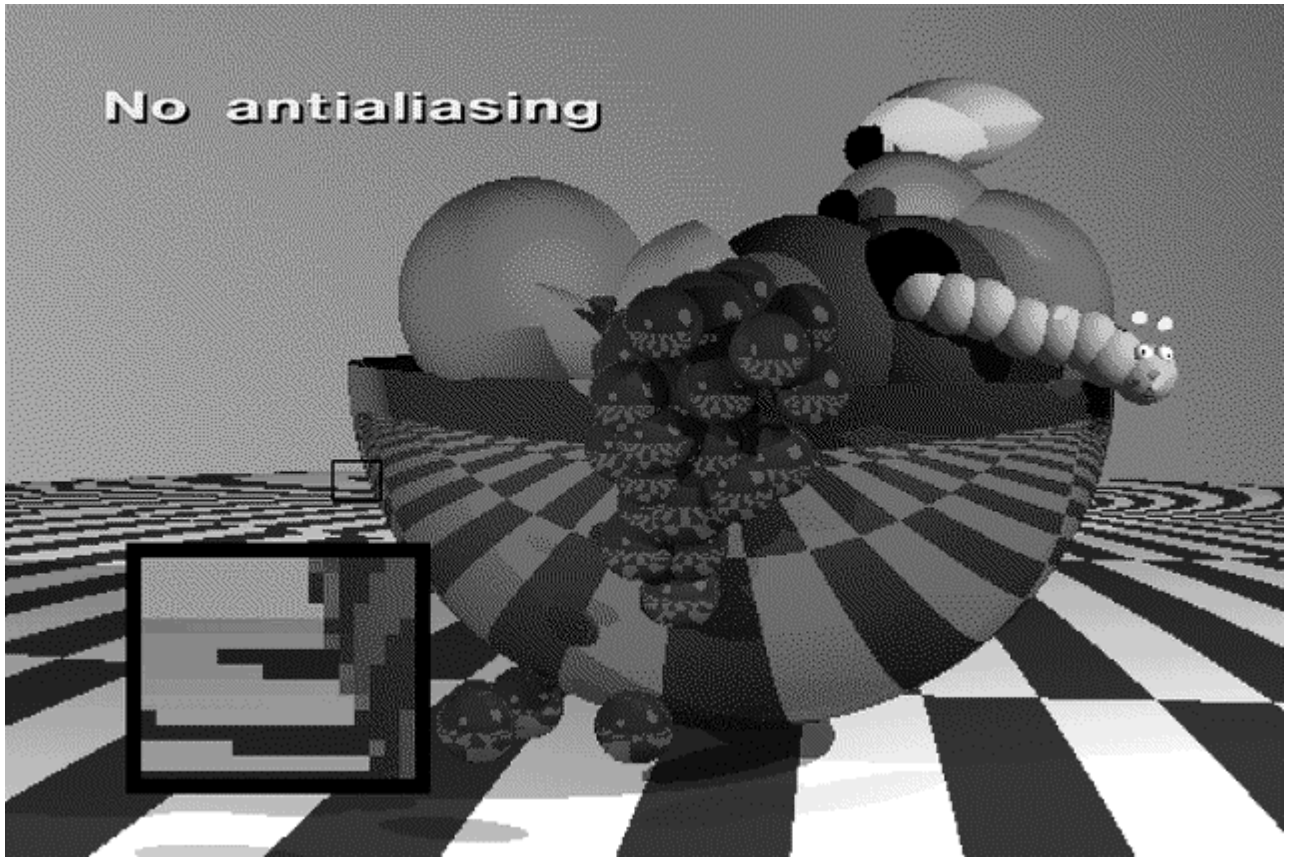
```
Raytrace()          // top level function
   for each pixel x,y
      color(pixel) = Trace(ray_through_pixel(x,y))


Trace(ray)          // fire a ray, return RGB radiance
   object_point = closest_intersection(ray)
   if object_point return Shade(object_point, ray)
   else return Background_Color
```

Writing a Simple Ray Tracer (Cont.)

```
Shade(point, ray)              /* return radiance along ray */
   radiance = black;           /* initialize color vector */
   for each light source
      shadow_ray = calc_shadow_ray(point,light)
      if !in_shadow(shadow_ray,light)
         radiance += phong_illumination(point,ray,light)
   if material is specularly reflective
      radiance += spec_reflectance *
        Trace(reflected_ray(point,ray)))
   if material is specularly transmissive
      radiance += spec_transmittance *
        Trace(refracted_ray(point,ray)))
   return radiance
```

```
Closest_intersection(ray)
   for each surface in scene
      calc_intersection(ray,surface)
   return the closest point of intersection to viewer
   (also return other info about that point, e.g., surface
    normal, material properties, etc.)
```

Problem with Simple Ray Tracing: Aliasing



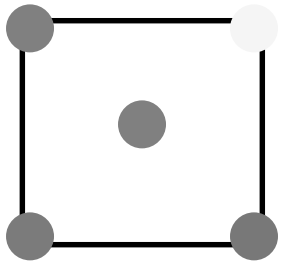
Aliasing

- Ray tracing gives a color for every possible point in the image
- But a square pixel contains an *infinite* number of points
  - These points may not all have the same color
  - Sampling: choose the color of one point (center of pixel)
  - This leads to *aliasing*
    - » jaggies
    - » moire patterns
  - aliasing means one frequency (high) masquerading as another (low)
    - » e.g. wagon wheel effect
- How do we fix this problem?

Antialiasing

Supersampling

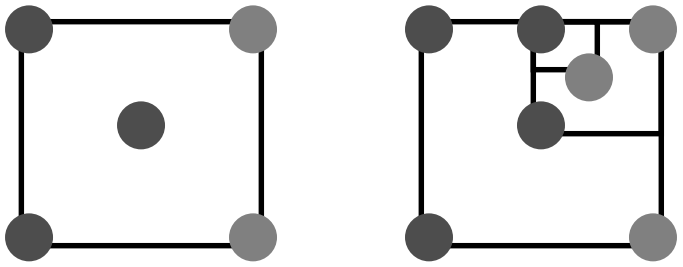
- Fire more than one ray for each pixel (e.g., a 3x3 grid of rays)
- Average the results using a filter



Antialiasing

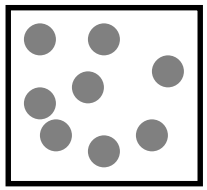
Supersampling

- Can be done *adaptively*
  - divide pixel into 2x2 grid, trace 5 rays (4 at corners, 1 at center)
  - if the colors are similar then just use their average
  - otherwise recursively subdivide each cell of grid
  - keep going until each 2x2 grid is close to uniform or limit is reached
  - filter the result

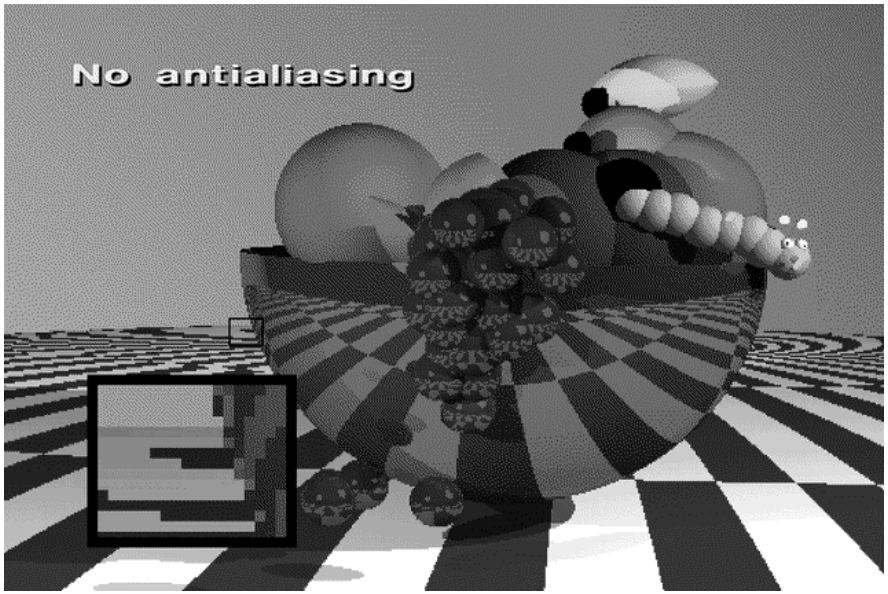


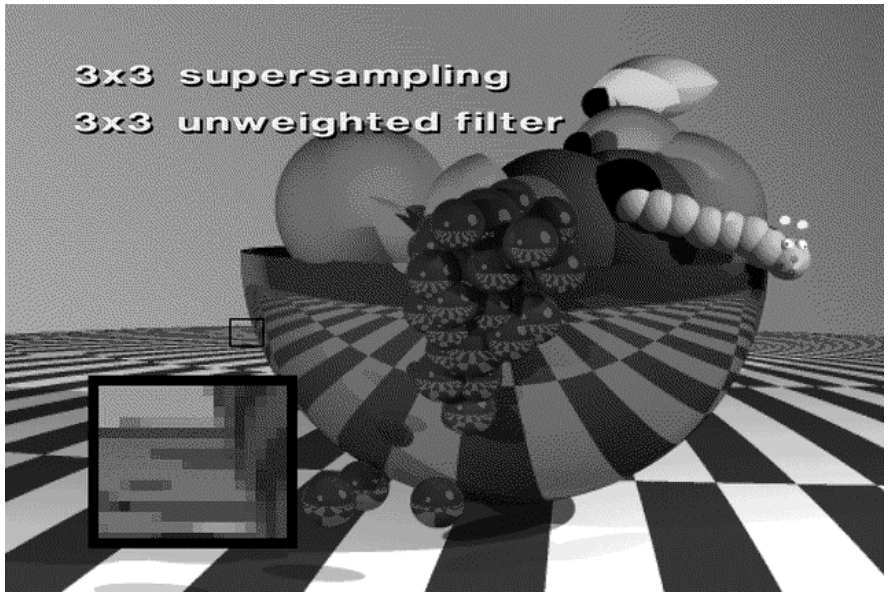
Adaptive Supersampling: Making the World a Better Place

- Is adaptive supersampling the answer?
  - Areas with fairly constant appearance are sparsely sampled (good)
  - Areas with lots of variability are heavily sampled (good)
- But alas...
  - even with massive supersampling visible aliasing is possible when the sampling grid interacts with regular structures
  - problem is, objects tend to be almost aligned with sampling grid
  - noticeable beating, moire patterns, etc… are possible
- So use *stochastic sampling*
  - instead of a regular grid, subsample randomly (or pseudo)
  - adaptively sample *statistically*
  - keep taking samples until the color estimates converge
  - jittering: perturb a regular grid



Supersampling



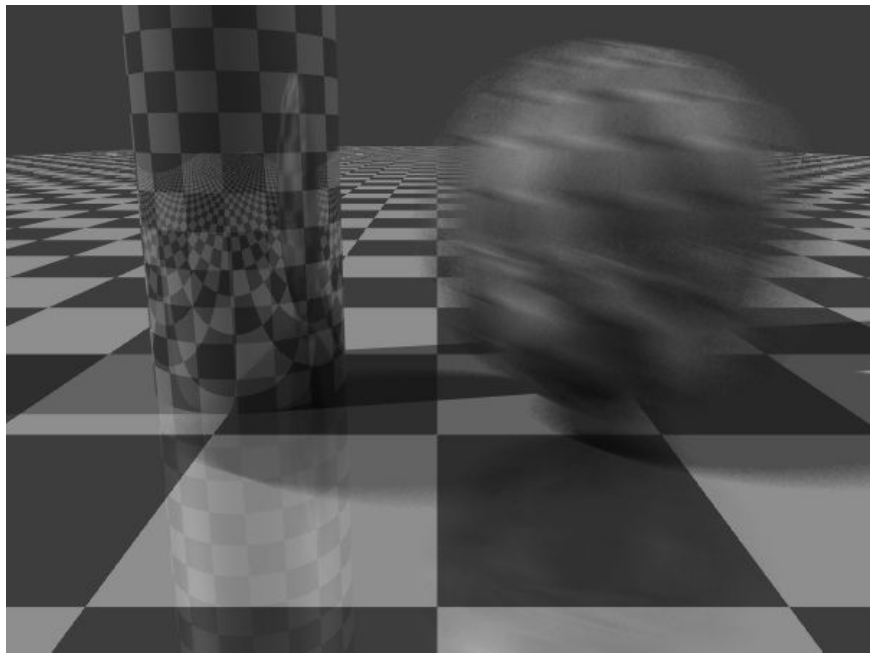


Temporal Aliasing

- Aliasing happens in time as well as space
  - the sampling rate is the frame rate, 30Hz for NTSC video, 24Hz for film
  - fast moving objects move large distances between frames
  - if we point-sample time, objects have a jerky, strobed look
- To avoid temporal aliasing we need to filter in time too
  - so compute frames at 120Hz and average them together (with appropriate weights)?
  - fast-moving objects become blurred streaks
- Real media (film and video) automatically do temporal anti-aliasing
  - photographic film integrates over the exposure time
  - video cameras have persistence (memory)
  - this shows up as *motion blur* in the photographs

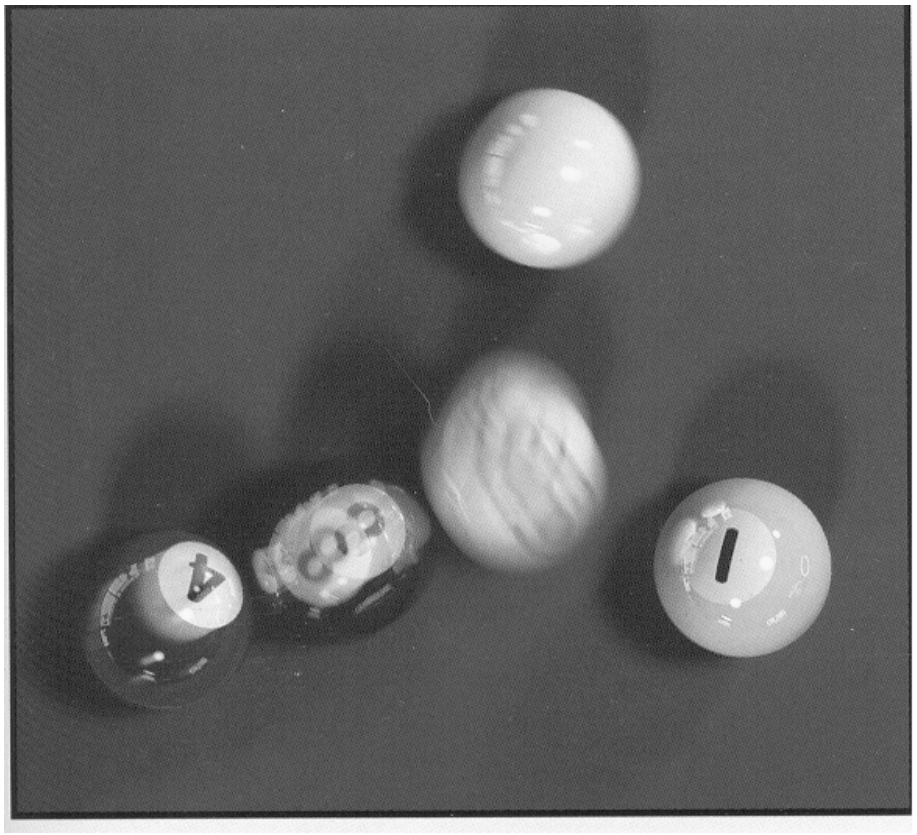
Motion Blur

- Apply stochastic sampling to time as well as space
- Assign a time as well as an image position to each ray
- The result is still-frame motion blur and smooth animation
- This is an example of distribution ray tracing



The Classic Example of Motion Blur

- From Foley et al. Plate III.16
- Rendered using distribution ray tracing at 4096x3550 pixels, 16 samples per pixel.
- Note motion-blurred reflections and shadows with penumbrae cast by extended light sources.



Distribution Ray Tracing

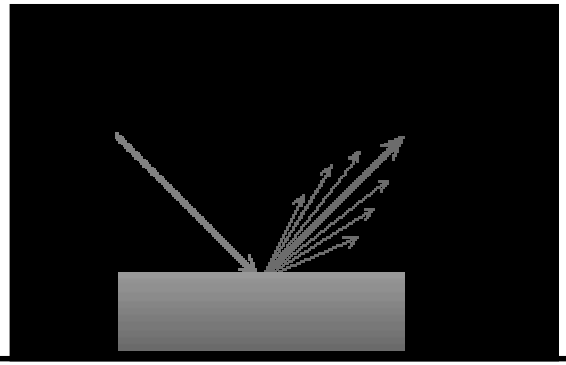
- distribute rays throughout a pixel to get spatial antialiasing
- distribute rays in time to get temporal antialiasing (motion blur)
- distribute rays in reflected ray direction to simulate gloss
- distribute rays across area light source to simulate penumbras (soft shadows)
- distribute rays across hemisphere to simulate diffuse interreflection
- a.k.a. “distributed ray tracing” or stochastic ray tracing
- a form of numerical integration

aliasing is replaced by less visually annoying noise!

powerful idea! (but can get slow)

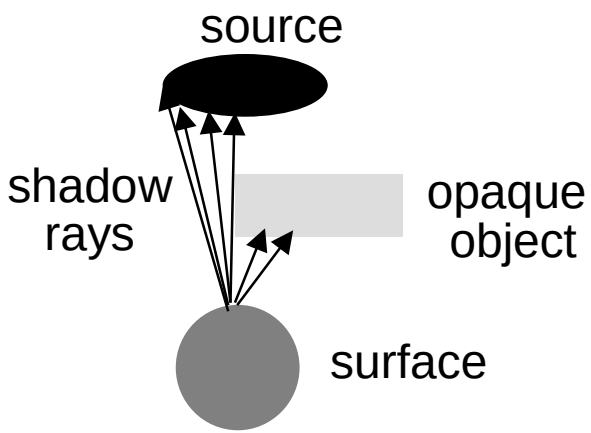
Gloss and Highlights

- Simple ray tracing spawns only one reflected ray
- But Phong illumination models a cone of rays
  - Produces fuzzy highlights
  - Change fuzziness (cone width) by varying the shininess parameter
- Can we generate fuzzy highlights?
  - Yes
  - But there's a catch
    - » we can't do light reflected from the fuzzy highlight onto other objects
- A more accurate model is possible using stochastic sampling
  - Stochastically sample rays within the cone
  - Sampling probability drops off sharply away from the specular angle
  - Highlights can be soft, blurred reflections of other objects

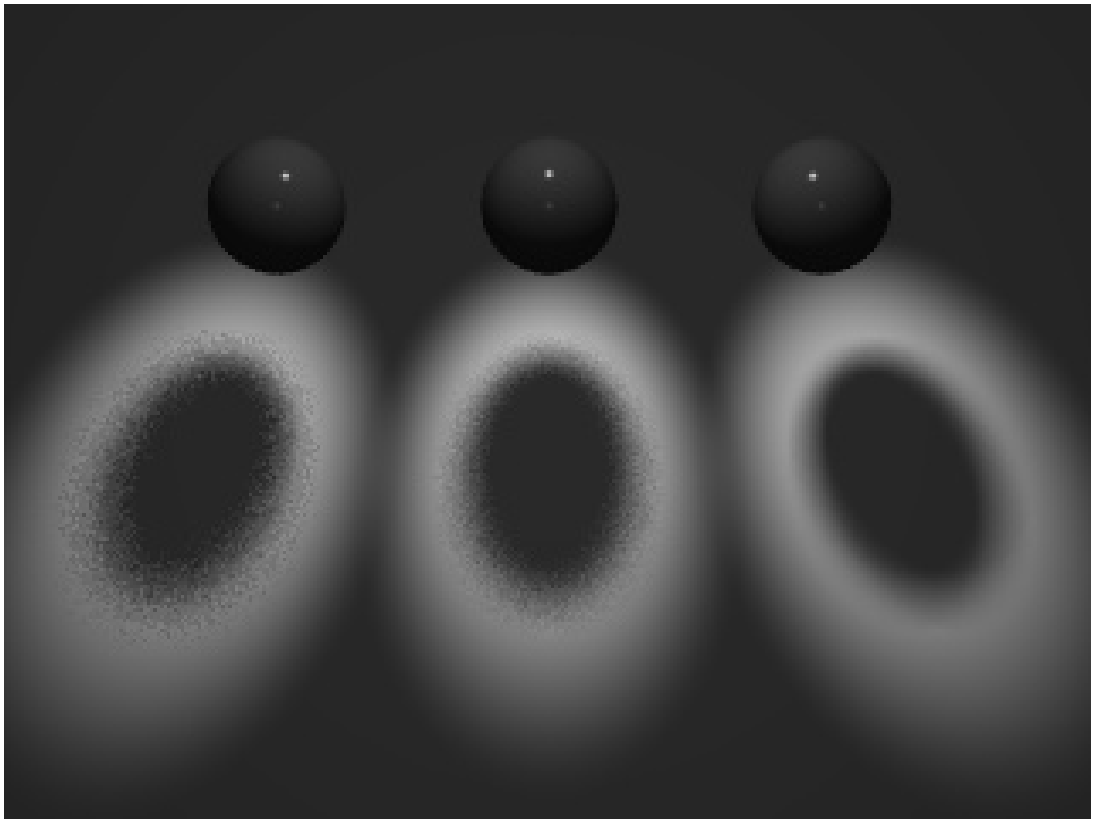


Soft Shadows

- Point light sources produce sharp shadow edges
  - the point is either shadowed or not
  - only one ray is required
- With an extended light source the surface point may be partially visible to it (*partial eclipse*)
  - only part of the light from the sources reaches the point
  - the shadow edges are softer
  - the transition region is the *penumbra*
- Distribution ray tracing can simulate this:
  - fire shadow rays from random points on the source
  - weight them by the brightness
  - the resulting shading depends on the fraction of the obstructed shadow rays

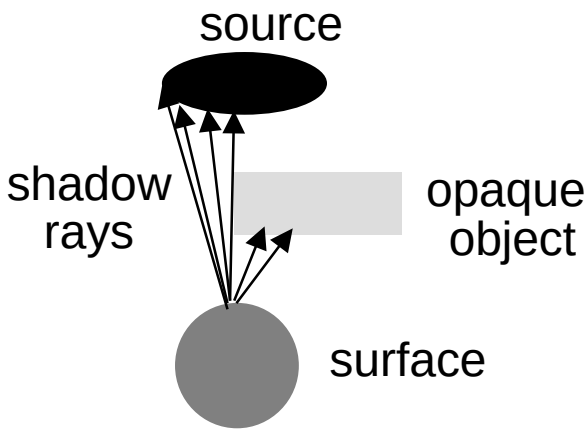


Soft Shadows



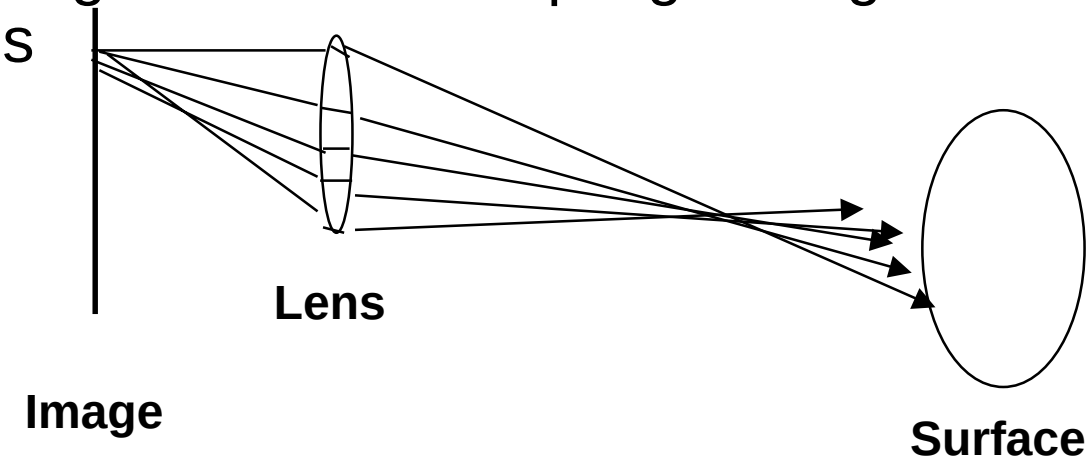
fewer rays,
more noise

more rays,
less noise

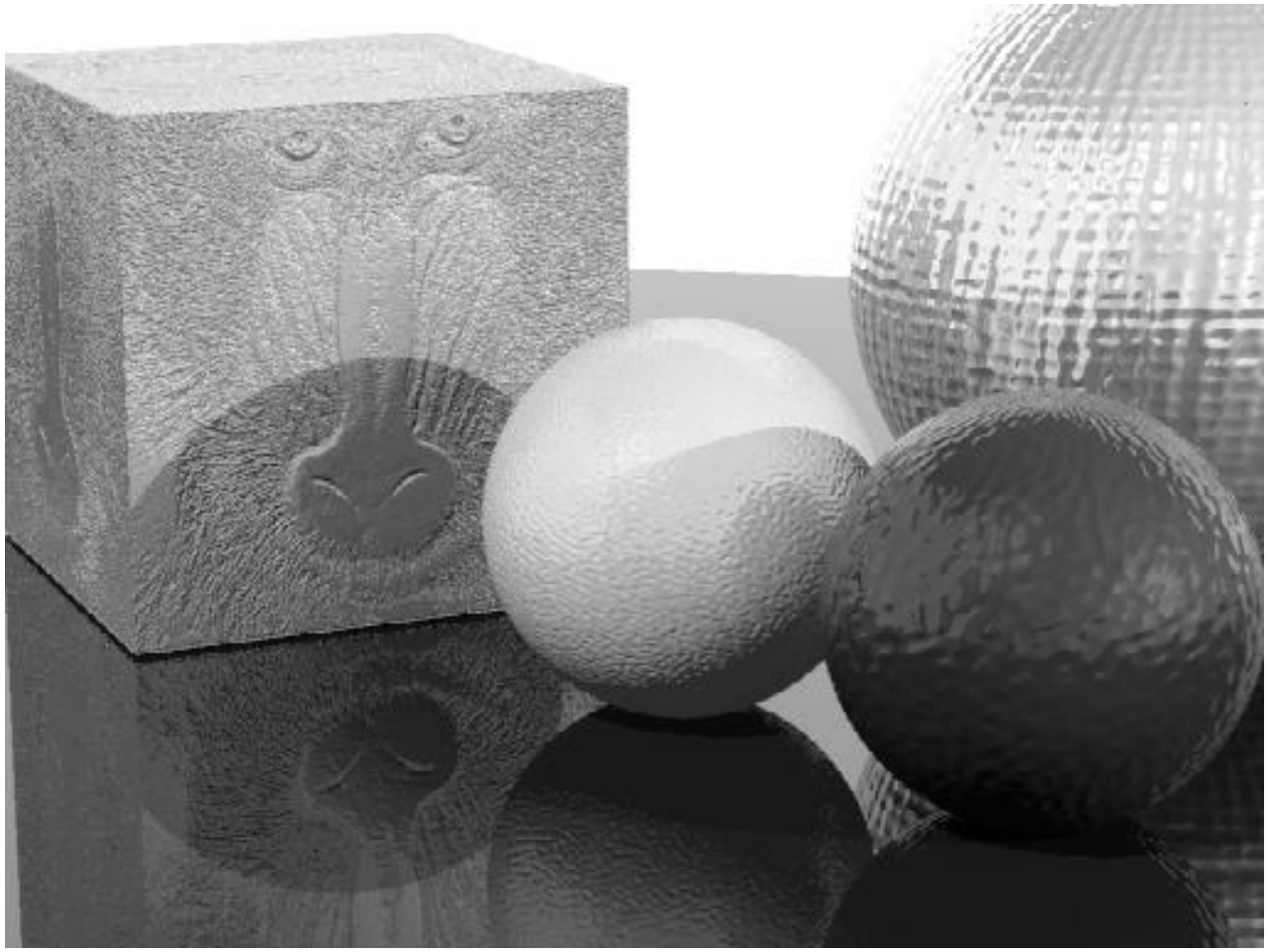


Depth of Field

- The pinhole camera model only approximates real optics
  - real cameras have lenses with focal lengths
  - only one plane is truly in focus
  - points away from the focus project as disks
  - the further away from the focus the larger the disk
- the range of distance that appear in focus is the *depth of field*
- simulate this using stochastic sampling through different parts of the lens



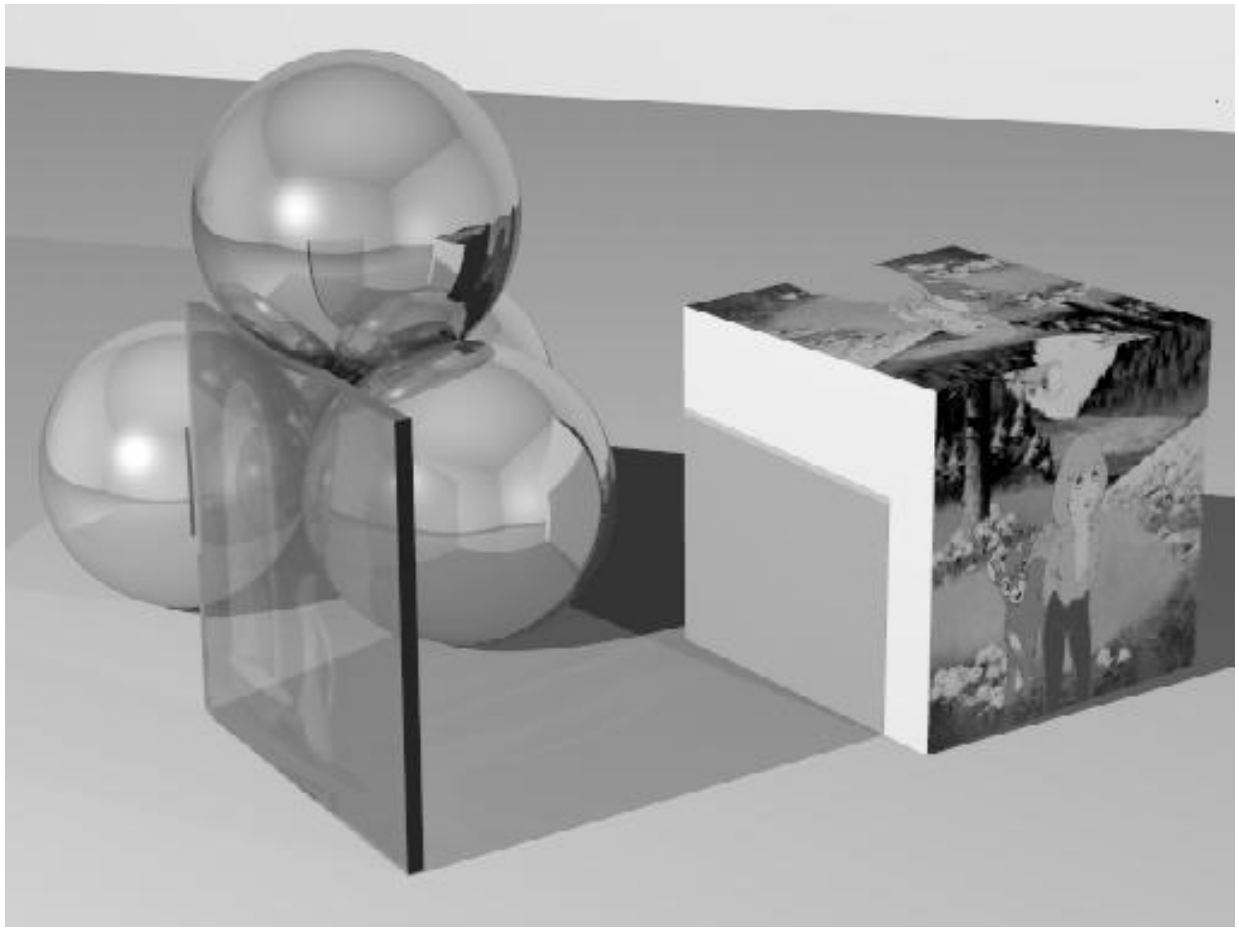
Examples



Including texture map and bump map

http://www.graphics.cornell.edu/online/tutorial/raytrace/

Examples



Semi-transparent glass with etched image.

http://www.graphics.cornell.edu/online/tutorial/raytrace/

Beyond Ray Tracing

- Ray tracing ignores the diffuse component of incident illumination
  - to achieve this component requires sending out rays from each surface point for the whole visible hemisphere
  - this is the *branching factor* of the recursive ray tree
- Even if you could compute such a massive problem there is a conceptual problem:
  - you will create loops:
    - » point A gets light from point B
    - » point B also gets light from point A

Doing it *Really* Right (or trying)

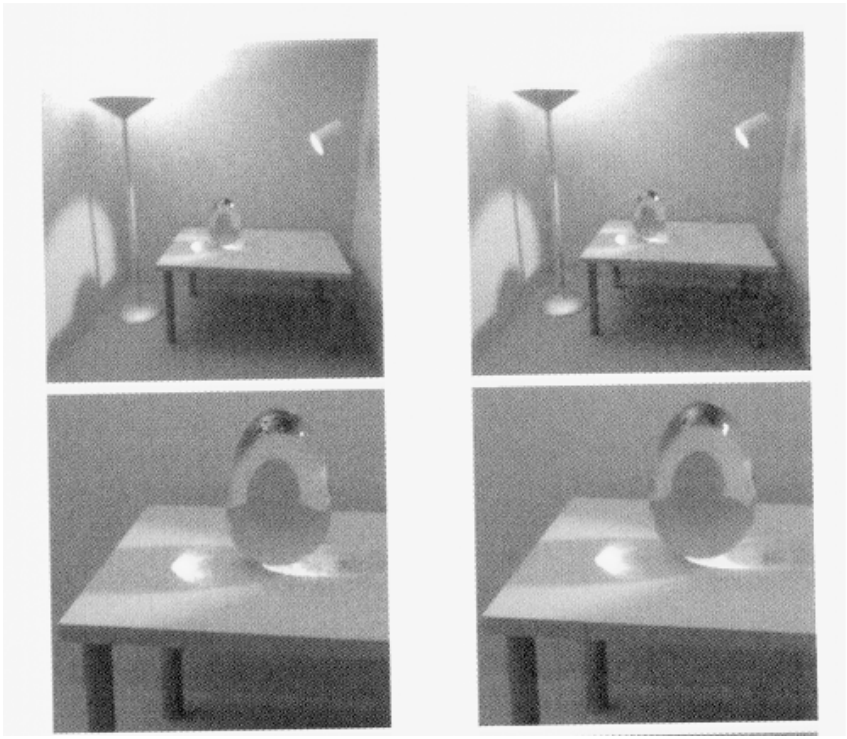
- The real solution is to solve simultaneously for incoming and outgoing light at all surface points
  - this is a massive integral equation
- *Radiosity* deals with the easy case of purely diffuse scenes
- Or, you can sample many, many complete paths from light source to camera
  - Metropolis Light Transport (Veach and Guibas, Siggraph 1997)

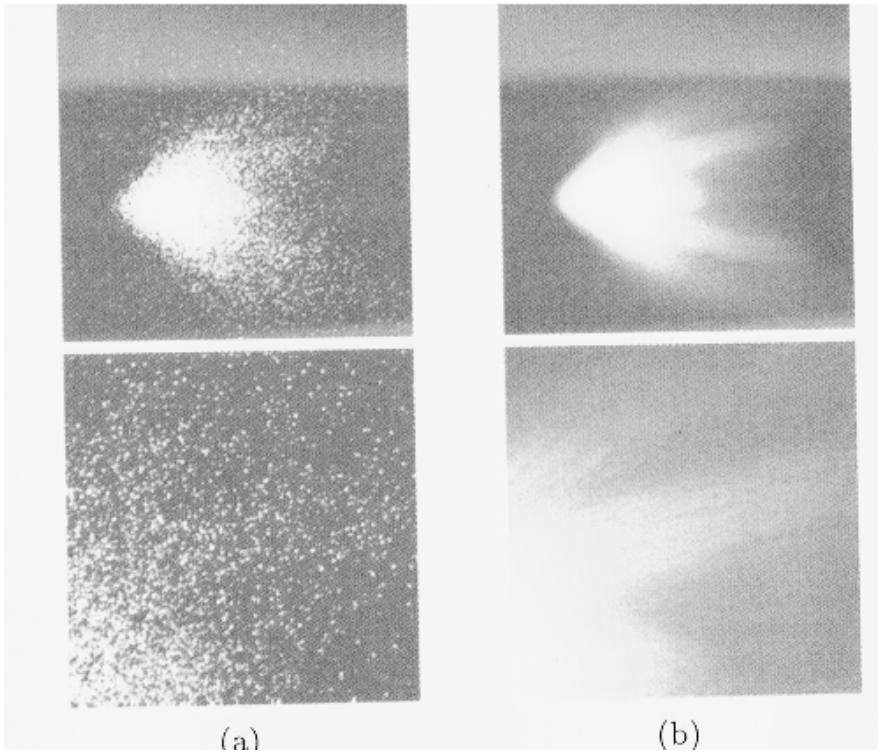
Diffuse Illumination

(b) Metropolis light transport with an average of 250 mutations per pixel [the same computation time as (a)].

From Veach and Guibas, Siggraph ‘97

Caustics





From Veach and Guibas, Siggraph ‘97

Announcements

- Assignment 4 will be out later today
- Problem Set 3 is due today or tomorrow by 9am in my mail box (4th floor NSH)
- How are the machines working out?
  - I have a meeting with Peter Lee and Bob Cosgrove on Wednesday to discuss the future of the cluster