Advanced Ray Tracing

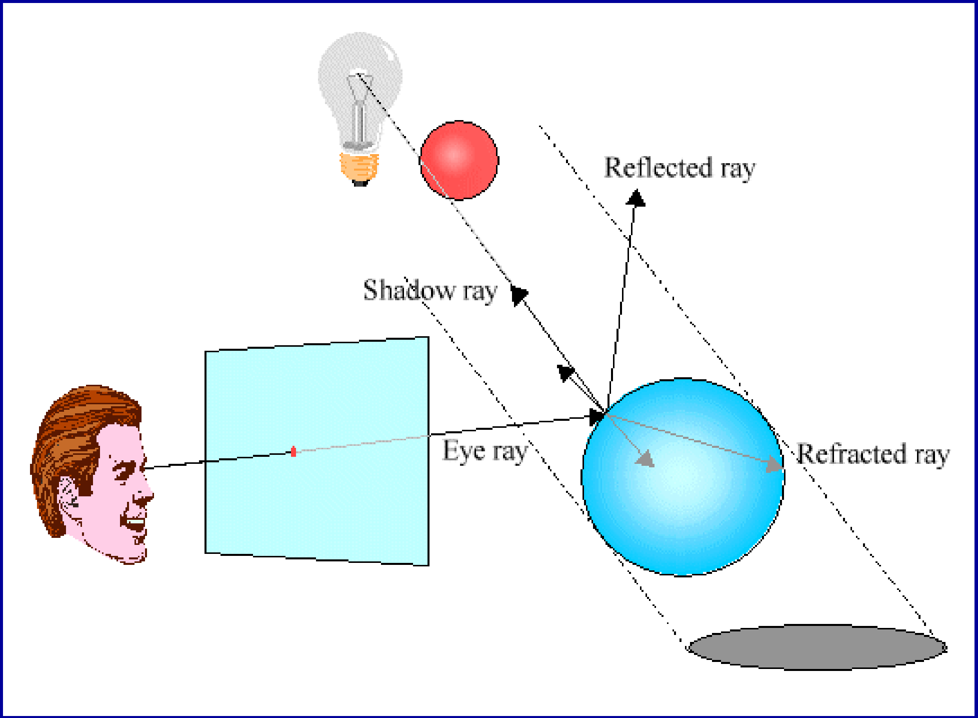
(Recursive) Ray Tracing
Antialiasing
Motion Blur
Distribution Ray Tracing
Ray Tracing and Radiosity

Assumptions

- Simple shading (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 those assumptions, simulating:
  - specular reflection
  - shadows
  - transparent surfaces (transmission with refraction)
  - sometimes indirect illumination (a.k.a. global illumination)
  - sometimes area light sources
  - sometimes fog

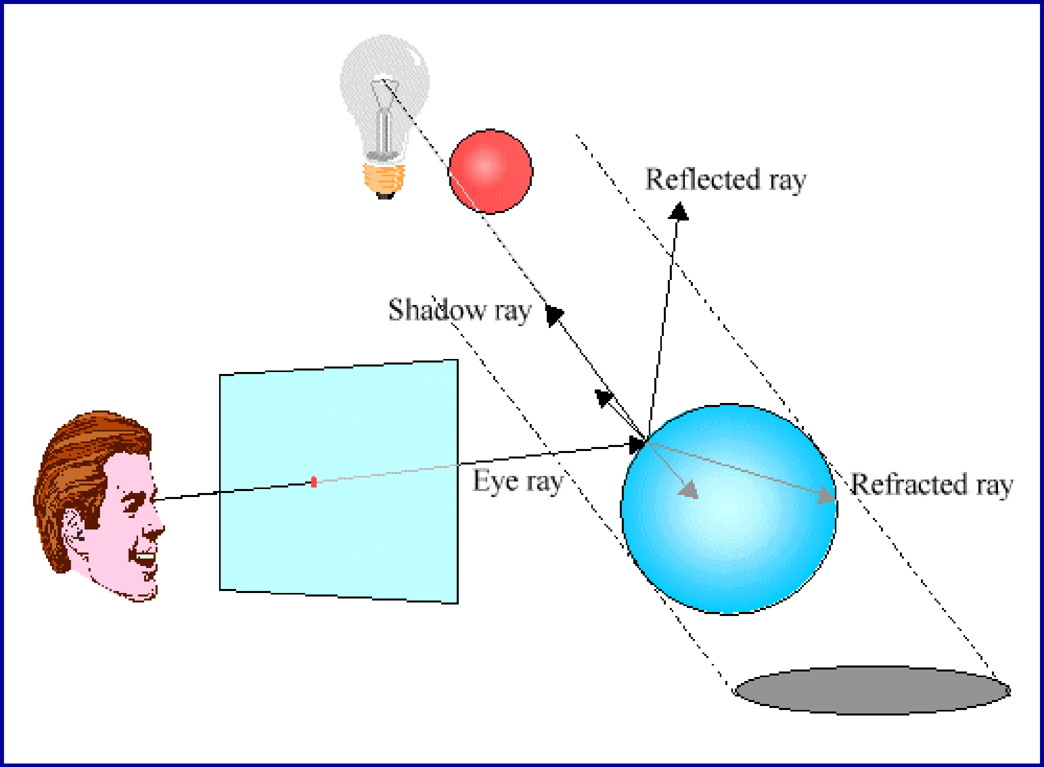
Ray Types for Ray Tracing



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



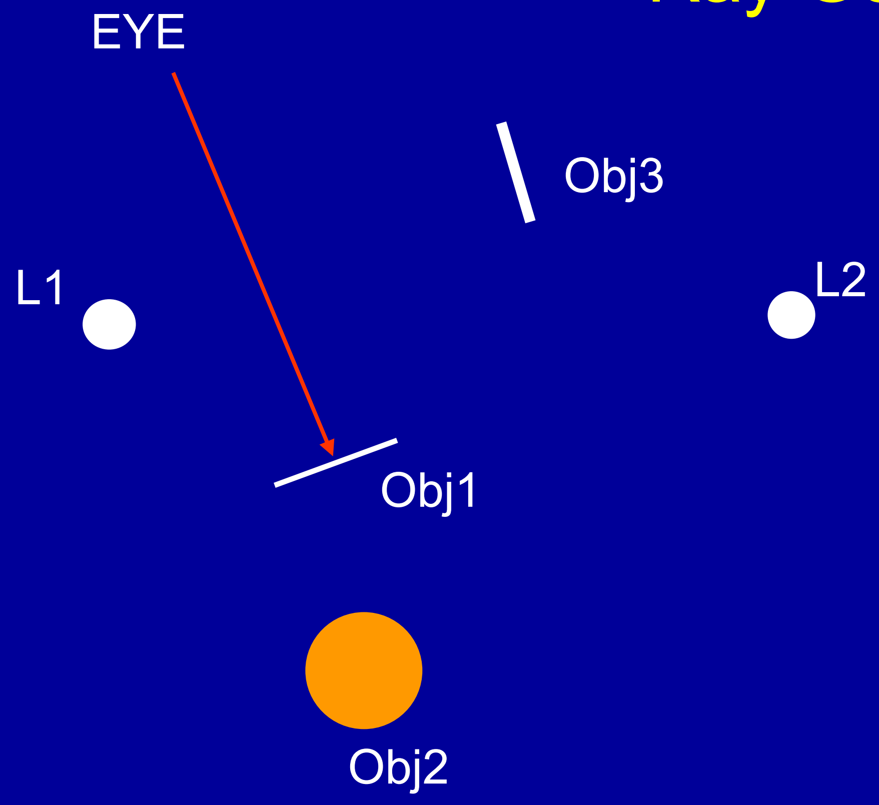
send ray from eye through each pixel

compute point of closest intersection with a scene surface

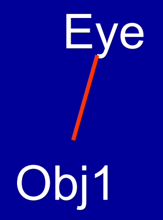
shade that point by computing shadow rays

spawn reflected and refracted rays, repeat

Ray Genealogy

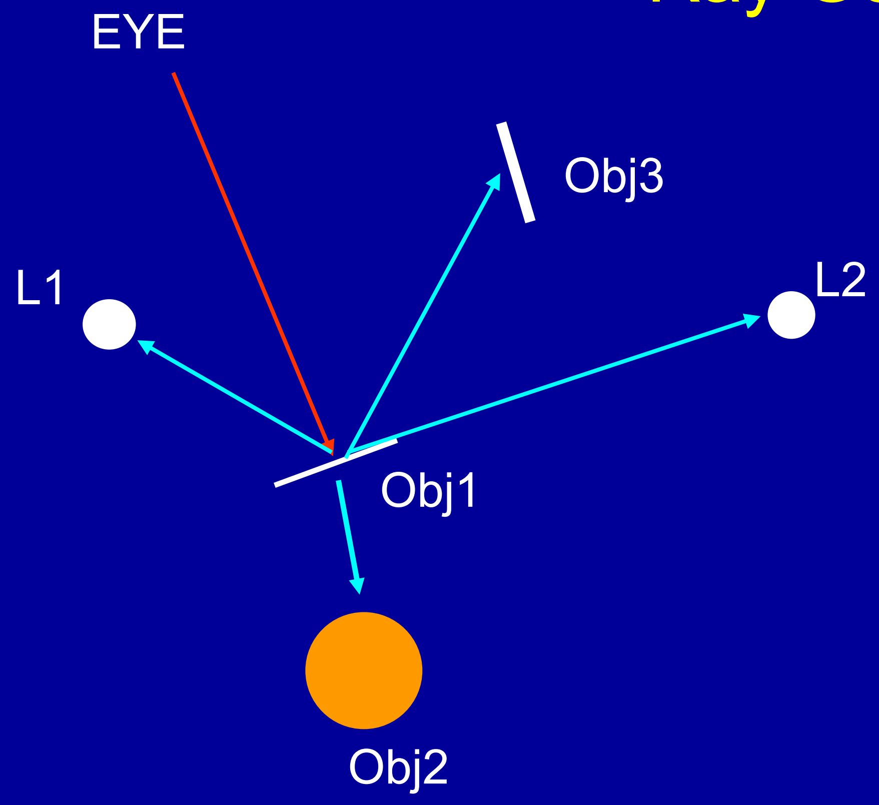


RAY PATHS (BACKWARD)

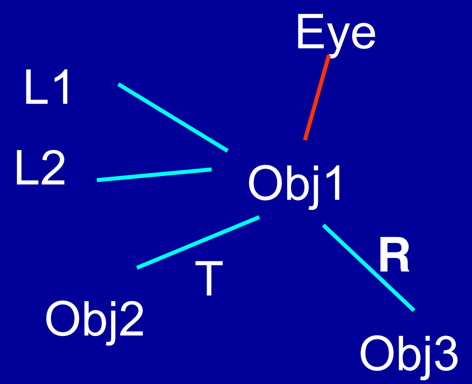


RAY TREE

Ray Genealogy

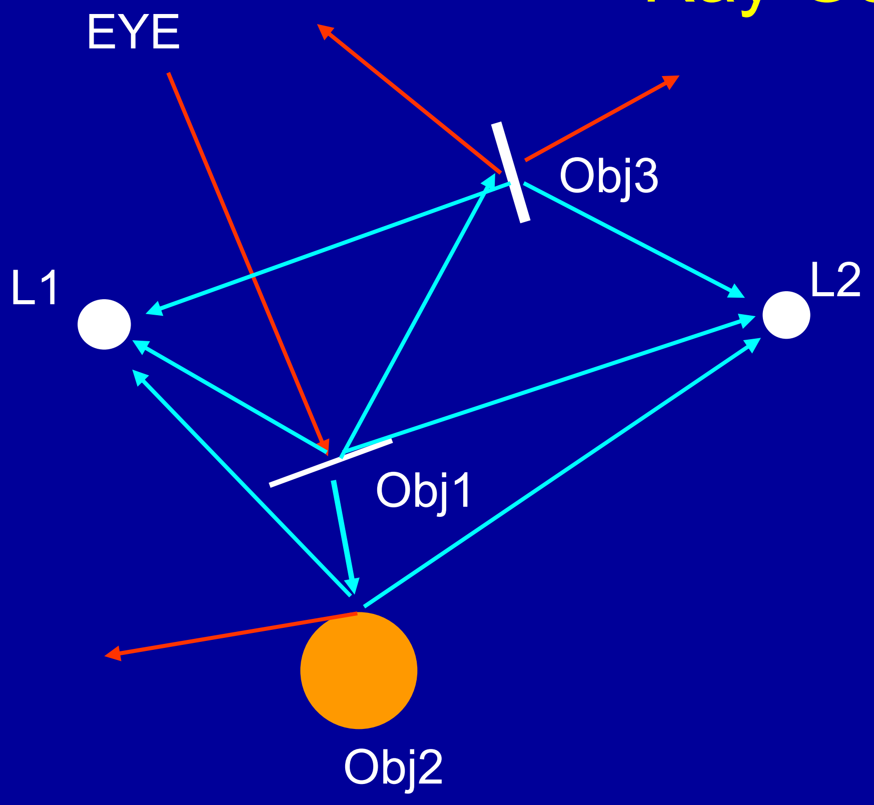


RAY PATHS (BACKWARD)

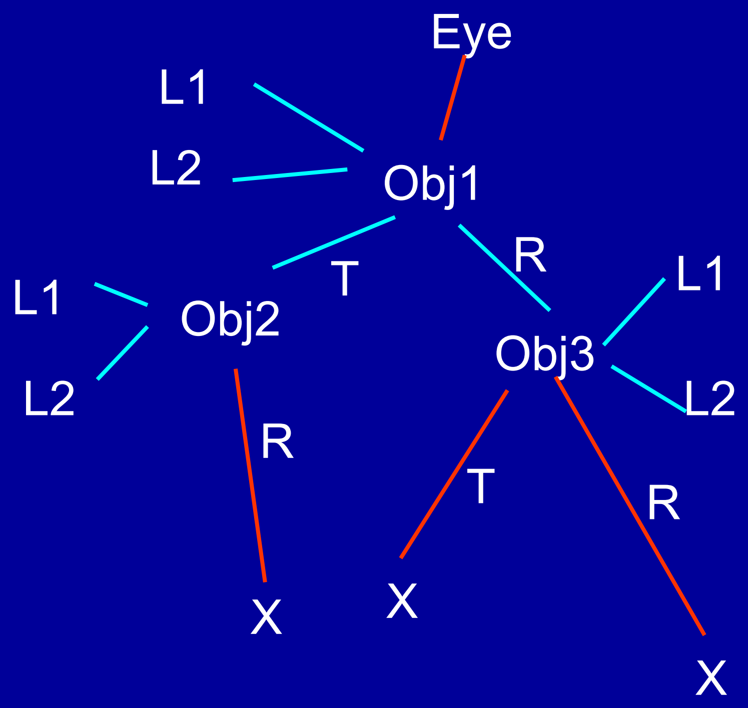


RAY TREE

Ray Genealogy

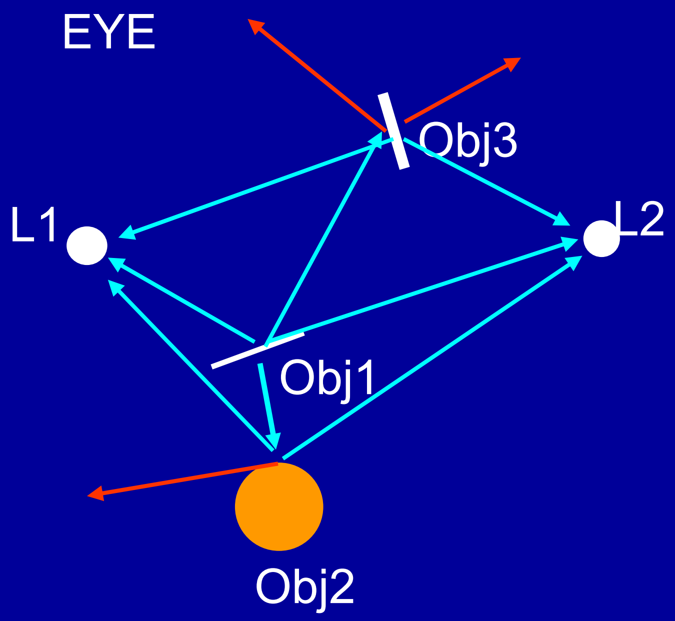


RAY PATHS (BACKWARD)



RAY TREE

When to stop?

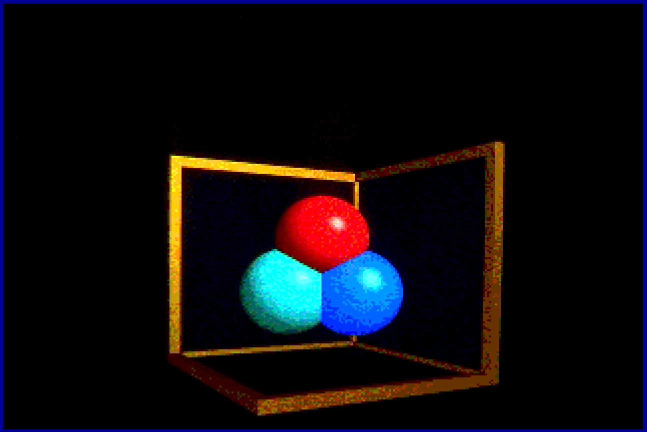


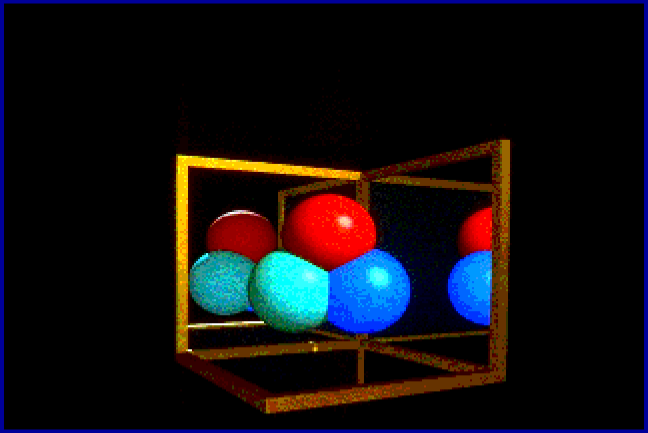
When a ray leaves the scene

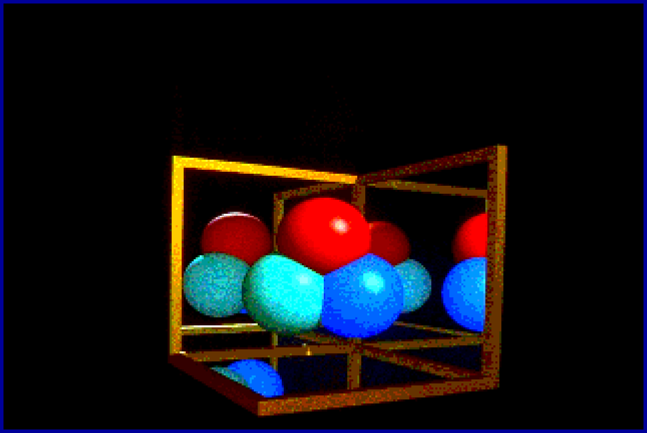
When its contribution becomes small—at each step the contribution is attenuated by the K's in the illumination model.

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

Ray Casting vs. Ray Tracing


Ray Casting -- 1 bounce


Ray Tracing -- 2 bounces


Ray Tracing -- 3 bounces

Ray Tracing—Demo program

http://www.siggraph.org/education/materials/HyperGraph/raytrace/rt_java/raytrace.html

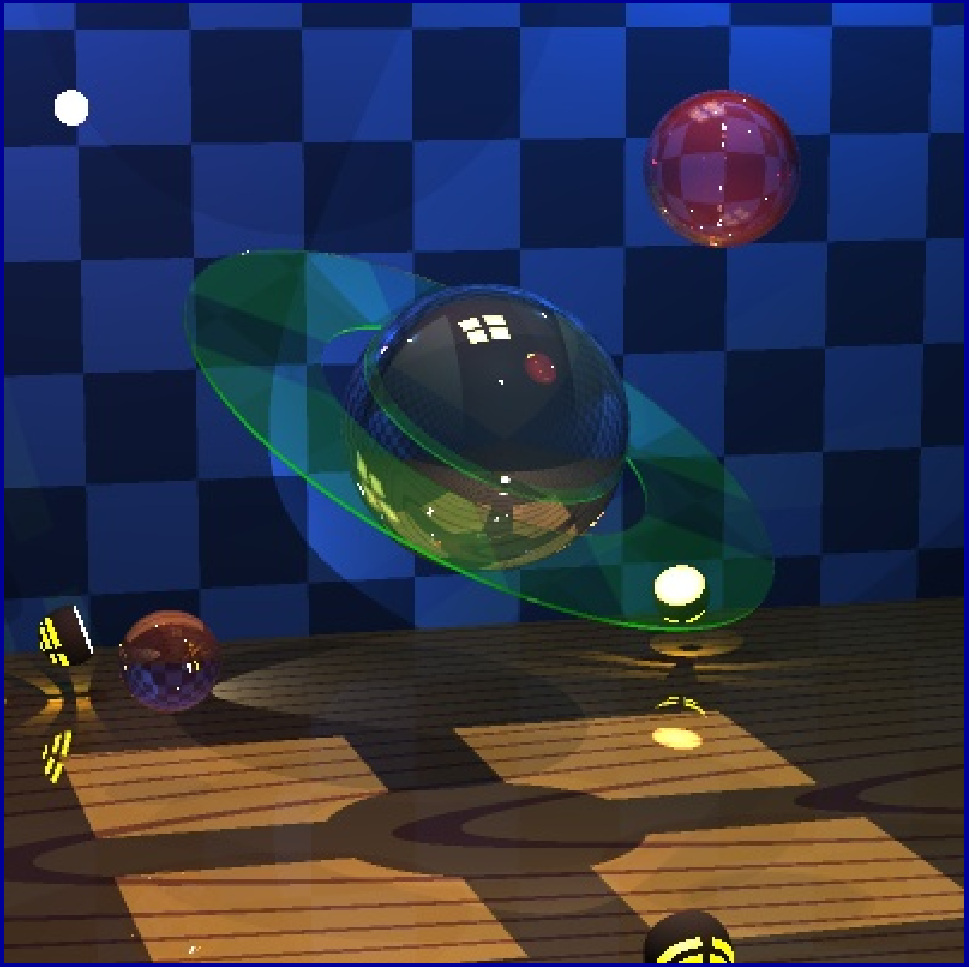
Writing a Simple Ray Tracer

```
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.)

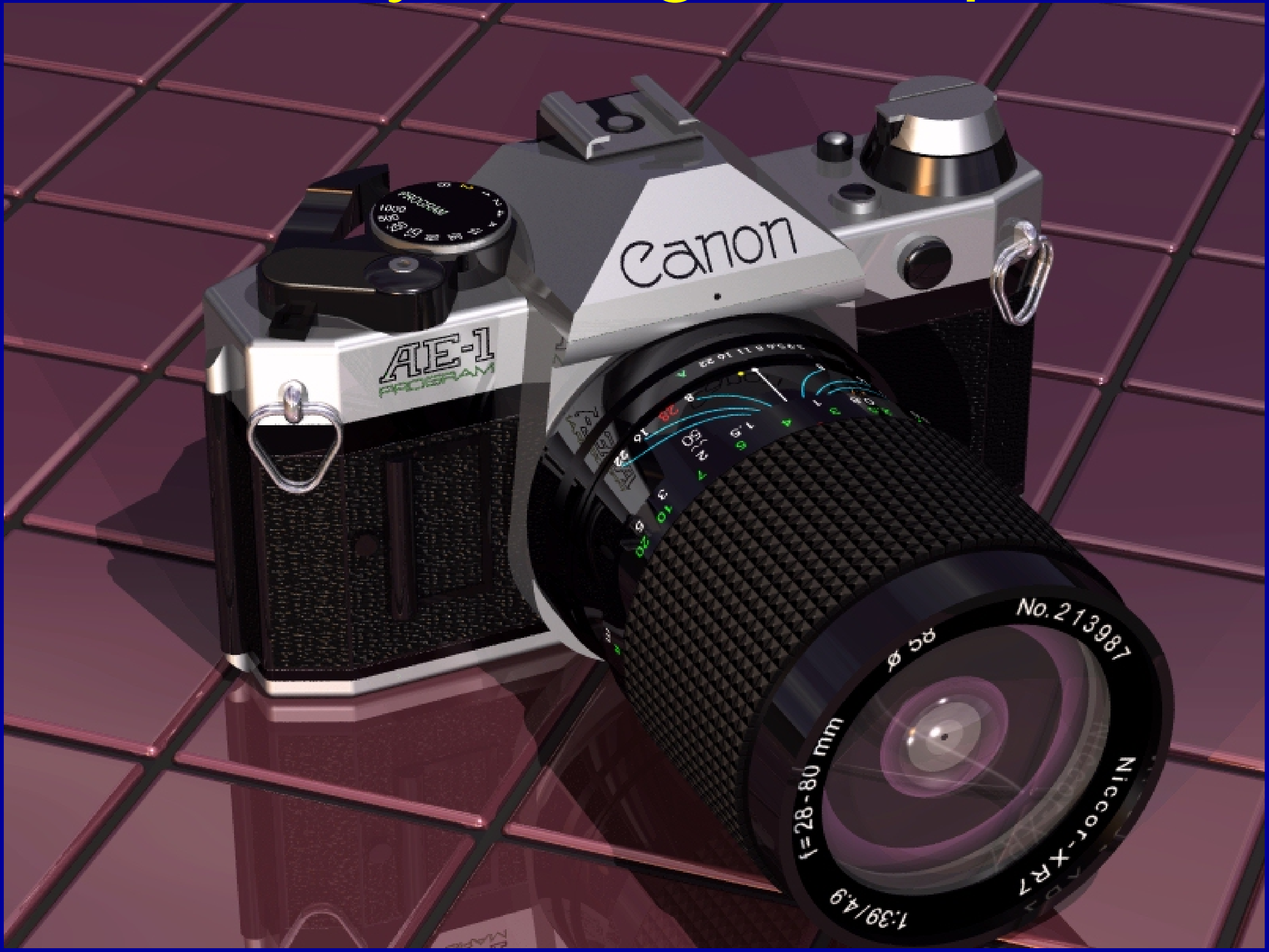
```
Shade(point, ray)                /* return radiance along ray */
   radiance = black;             /* initialize color vector */
   for each light source
      shadow_ray = calc_shadow_ray(point,light)
      if !in_shadow(shadow_ray,light)
         radiance += phong_illumination(point,ray,light)
   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.)
```

Raytracing Example



http://www.med.osaka-u.ac.jp/pub/cl-comp/saito/raytr/saturn-img.html

Raytracing Example



http://www.povray.org/

What problems do you see?

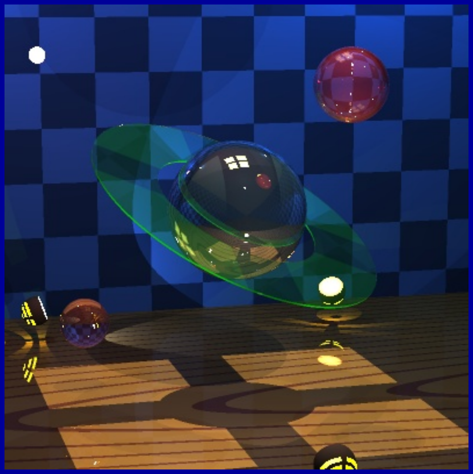
Images are VERY clean

Alignment of objects and sampling can lead to unintended patterns

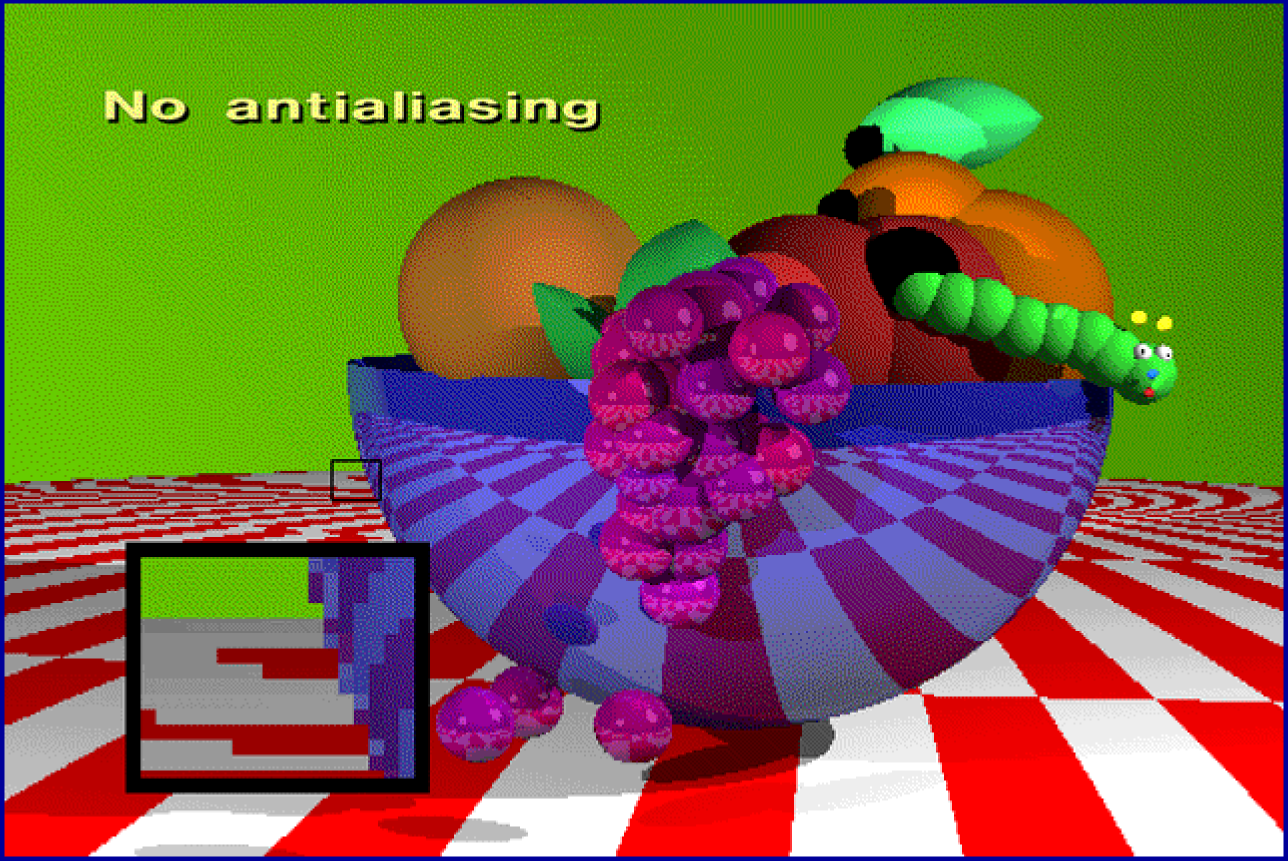
Solution:

more rays

more randomness



Problem with Simple Ray Tracing: Aliasing



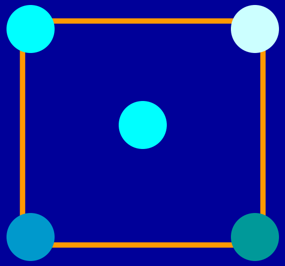
Aliasing

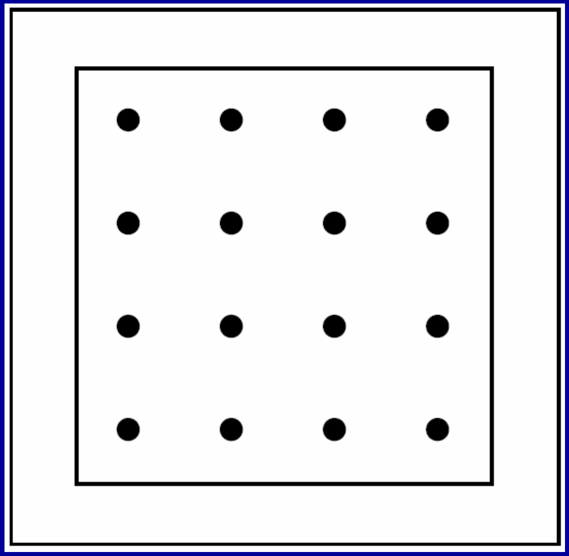
- Ray tracing gives a color for every possible point in the image
- But a square pixel contains an *infinite* number of points
  - These points may not all have the same color
  - Sampling: choose the color of one point (center of pixel)
  - Regular sampling leads to *aliasing*
    - jaggies
    - moire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

Anti-aliasing

Supersampling

- Fire more than one ray for each pixel (e.g., a 4x4 grid of rays)
- Average the results (perhaps using a filter)



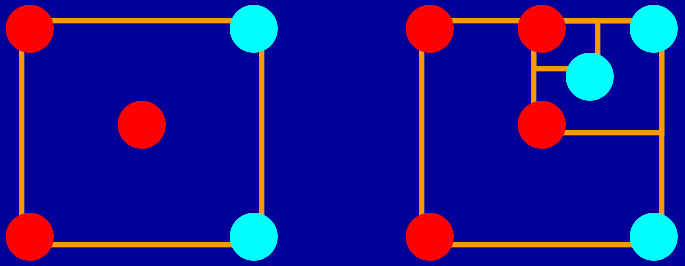


blackboard

Anti-aliasing: Supersampling

Can be done *adaptively*

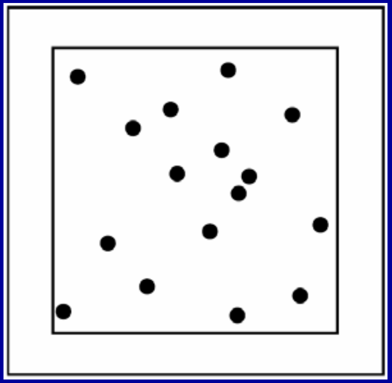
- divide pixel into 2x2 grid, trace 5 rays (4 at corners, 1 at center)
- if the colors are similar then just use their average
- otherwise recursively subdivide each cell of grid
- keep going until each 2x2 grid is close to uniform or limit is reached
- filter the result



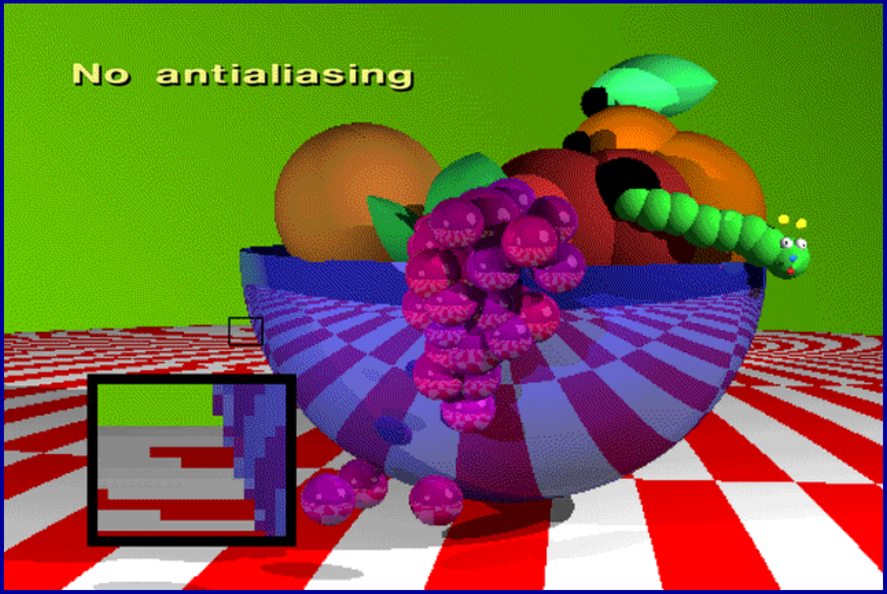
Adaptive Supersampling

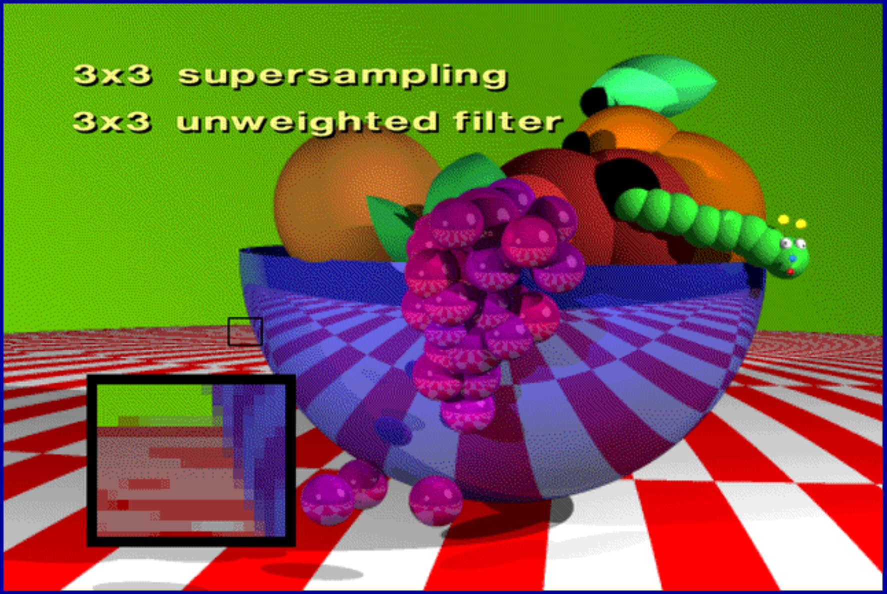
- Is adaptive supersampling the answer?
  - Areas with fairly constant appearance are sparsely sampled (good)
  - Areas with lots of variability are heavily sampled (good)
- But...
  - even with massive supersampling visible aliasing is possible when the sampling grid interacts with regular structures
  - problem is, objects tend to be almost aligned with sampling grid
  - noticeable beating, moire patterns are possible
- So use *stochastic sampling*
  - instead of a regular grid, subsample randomly
  - then adaptively subsample

blackboard



Supersampling





Temporal Aliasing: Motion Blur

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

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

To avoid temporal aliasing we need to filter in time too

- so compute frames at 120Hz and average them together (with appropriate weights)?
- a bit expensive

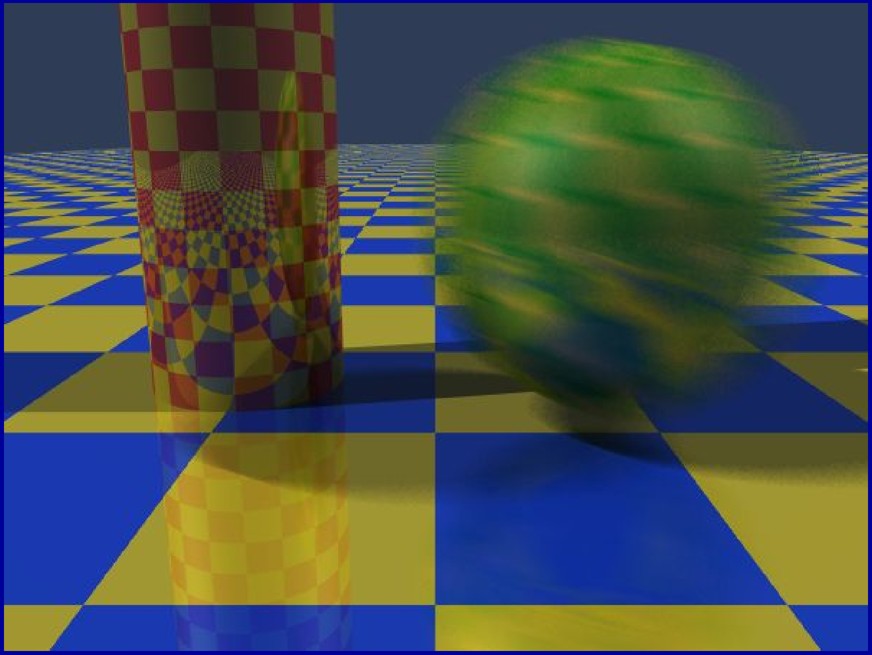
Motion Blur

Apply stochastic sampling to time as well as space

Assign a time as well as an image position to each ray

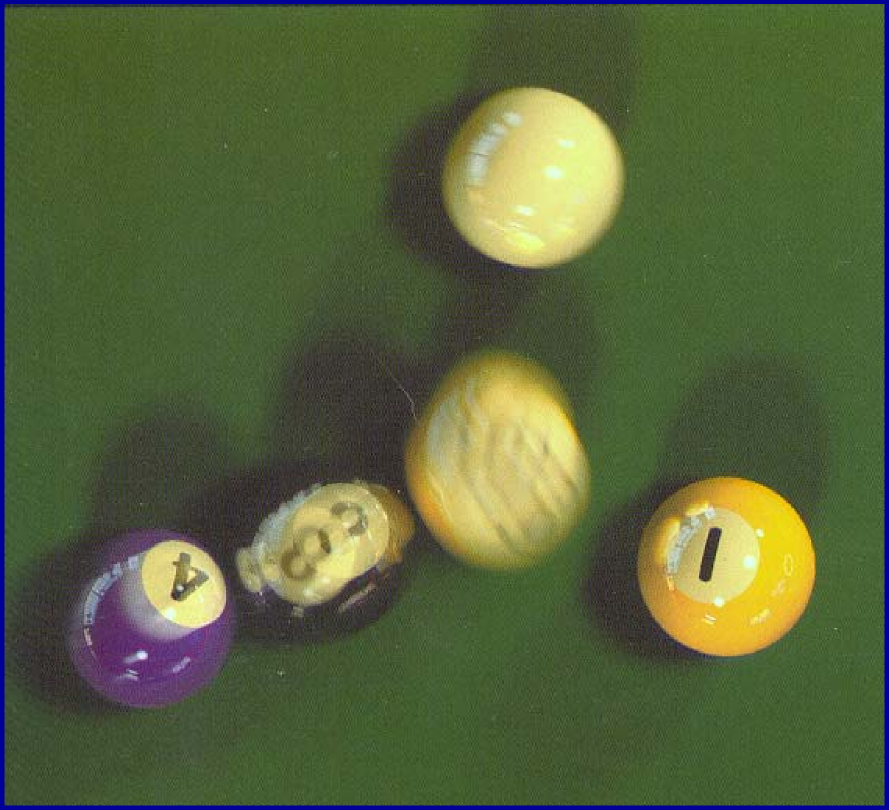
The result is still-frame motion blur and smooth animation

Jitter time: $T = T0 + \delta(T1 - T0)$ for each ray



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.



Glossy Reflection

Simple ray tracing spawns only one reflected ray—perfect reflection

But Phong illumination models a cone of rays

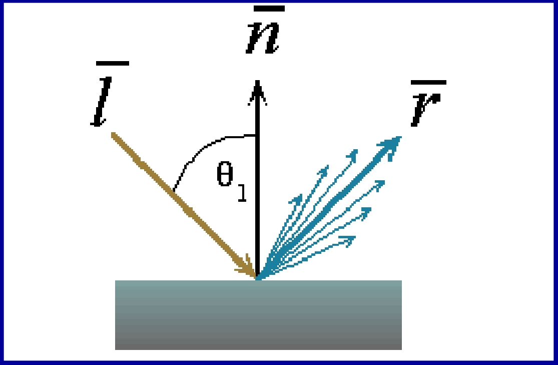
- Produces fuzzy highlights
- Change fuzziness (cone width) by varying the shininess parameter

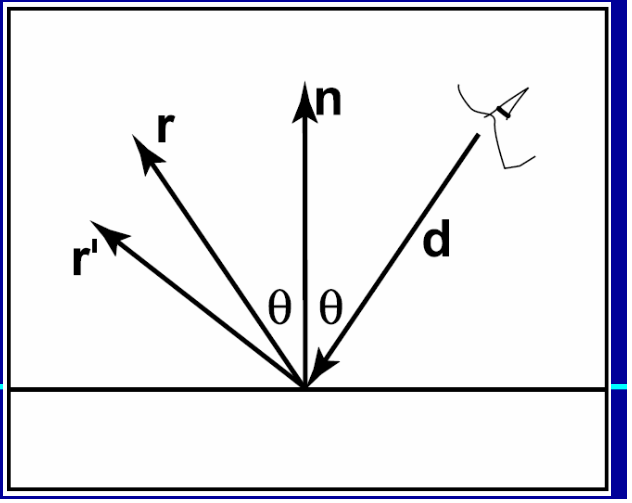
Can we generate fuzzy highlights?

- 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

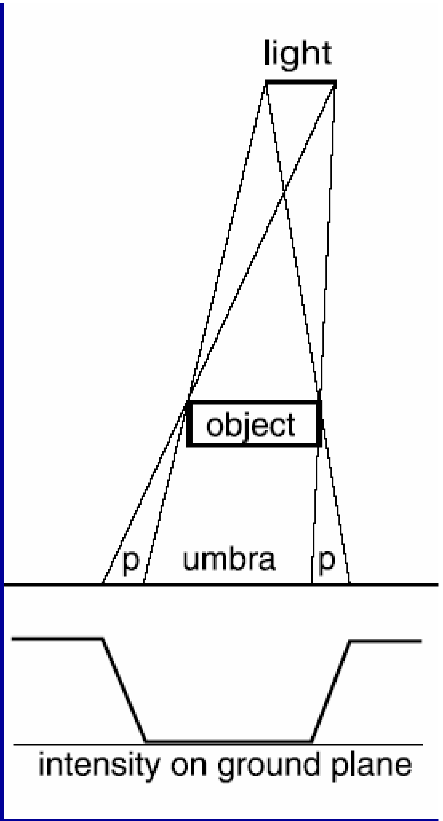
- the point is either shadowed or not
- only one ray is required

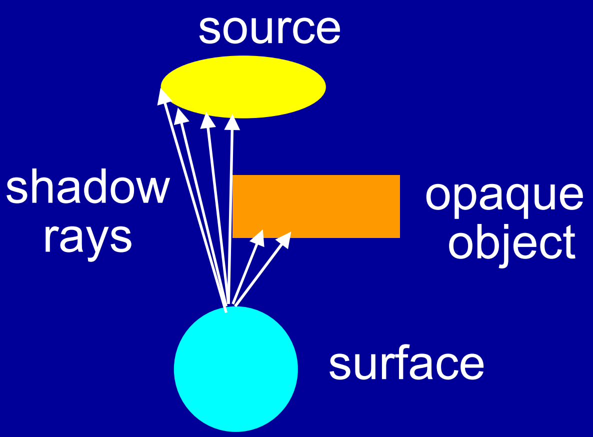
With an extended light source the surface point may be partially visible to it

- only part of the light from the sources reaches the point
- the shadow edges are softer
- the transition region is the *penumbra*

Accomplish this by

- firing shadow rays to random points on the light source
- weighting them by the brightness
- the resulting shading depends on the fraction of the obstructed shadow rays



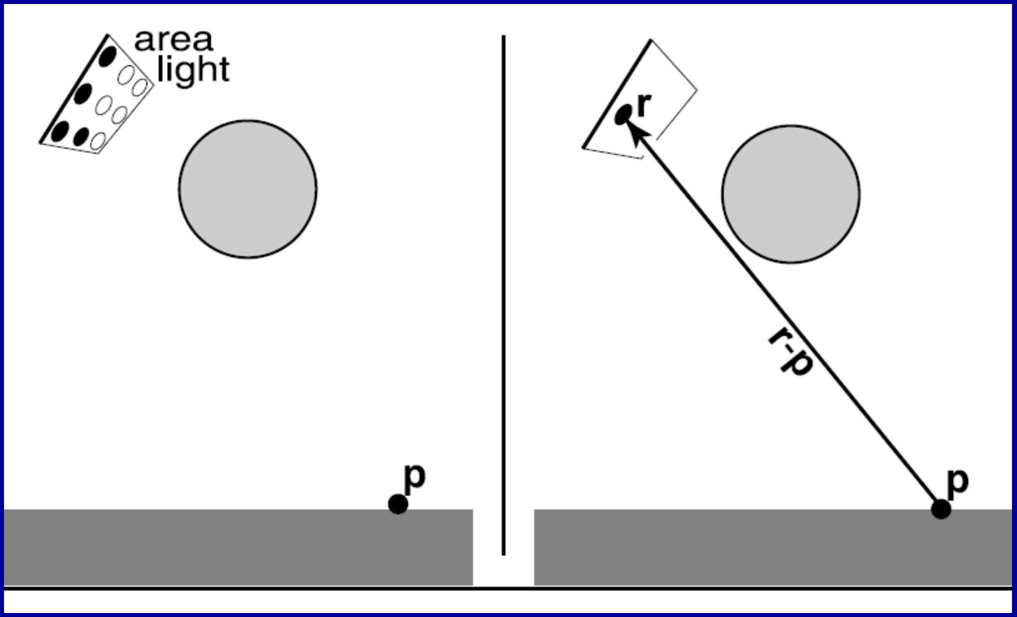


Soft Shadows

firing shadow rays to random points on the light source

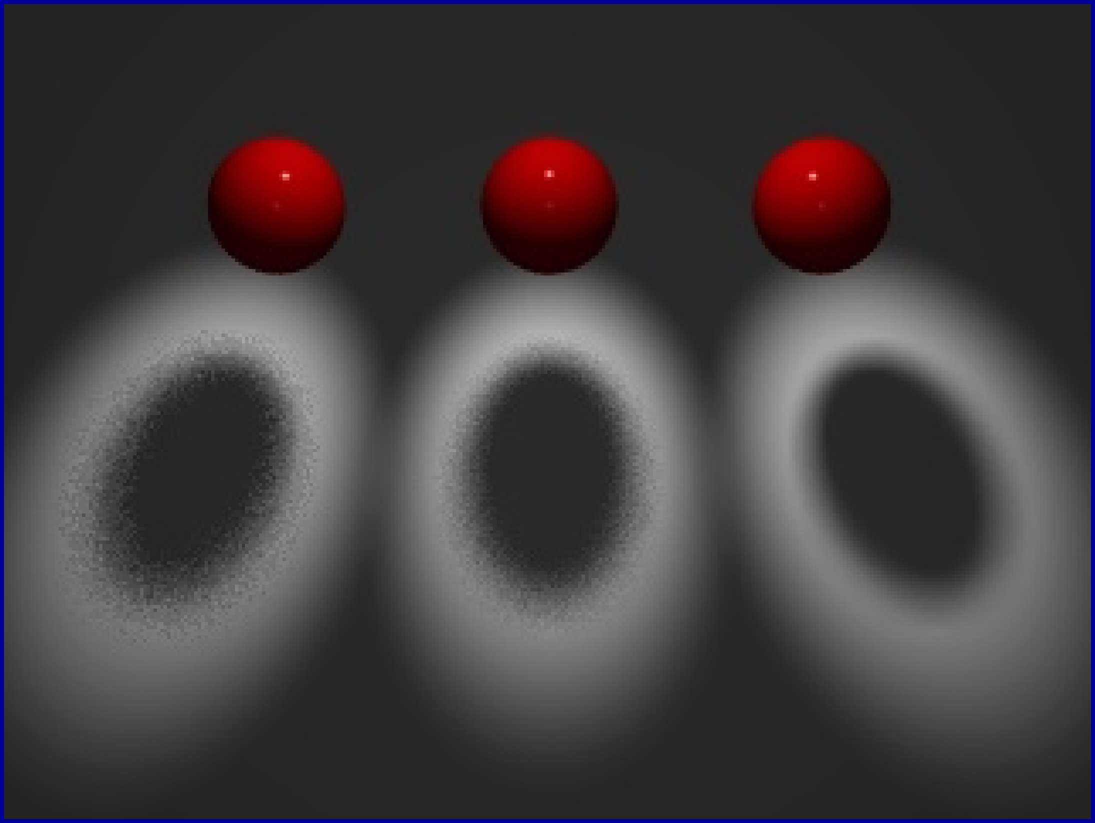
weighting them by the brightness

the resulting shading depends onthe fraction of the obstructed shadow rays



blackboard

Soft Shadows



fewer rays,
more noise

more rays,
less noise

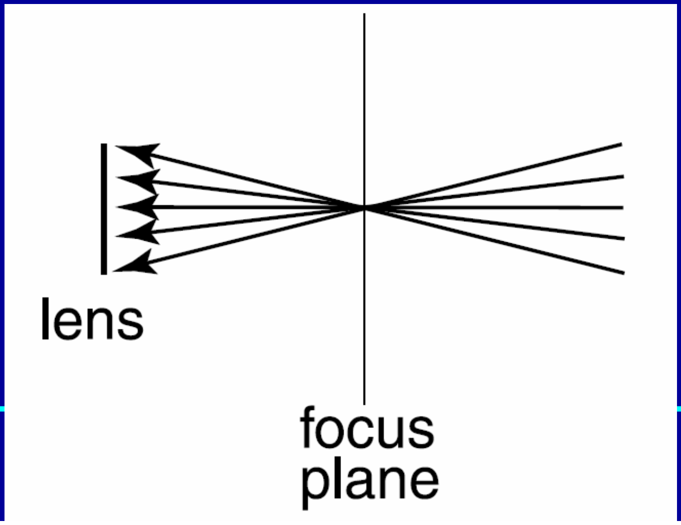
Depth of Field

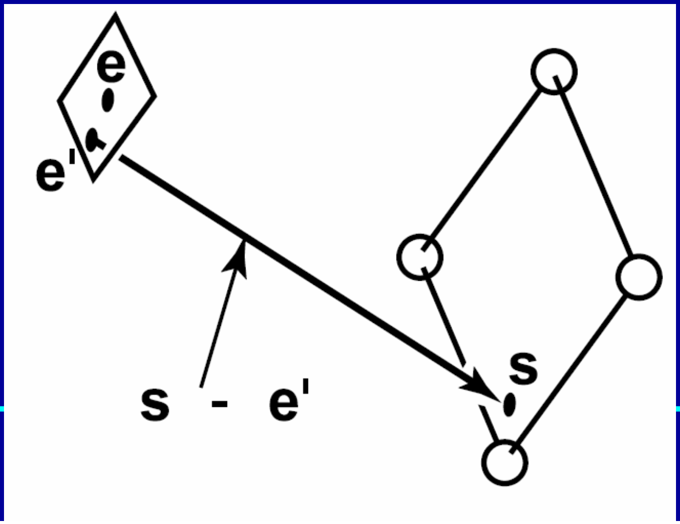
The camera model only approximates real optics

- 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 distances that appear in focus is the *depth of field*

Simulate this using stochastic sampling through different parts of the lens





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 eye to simulate depth of field

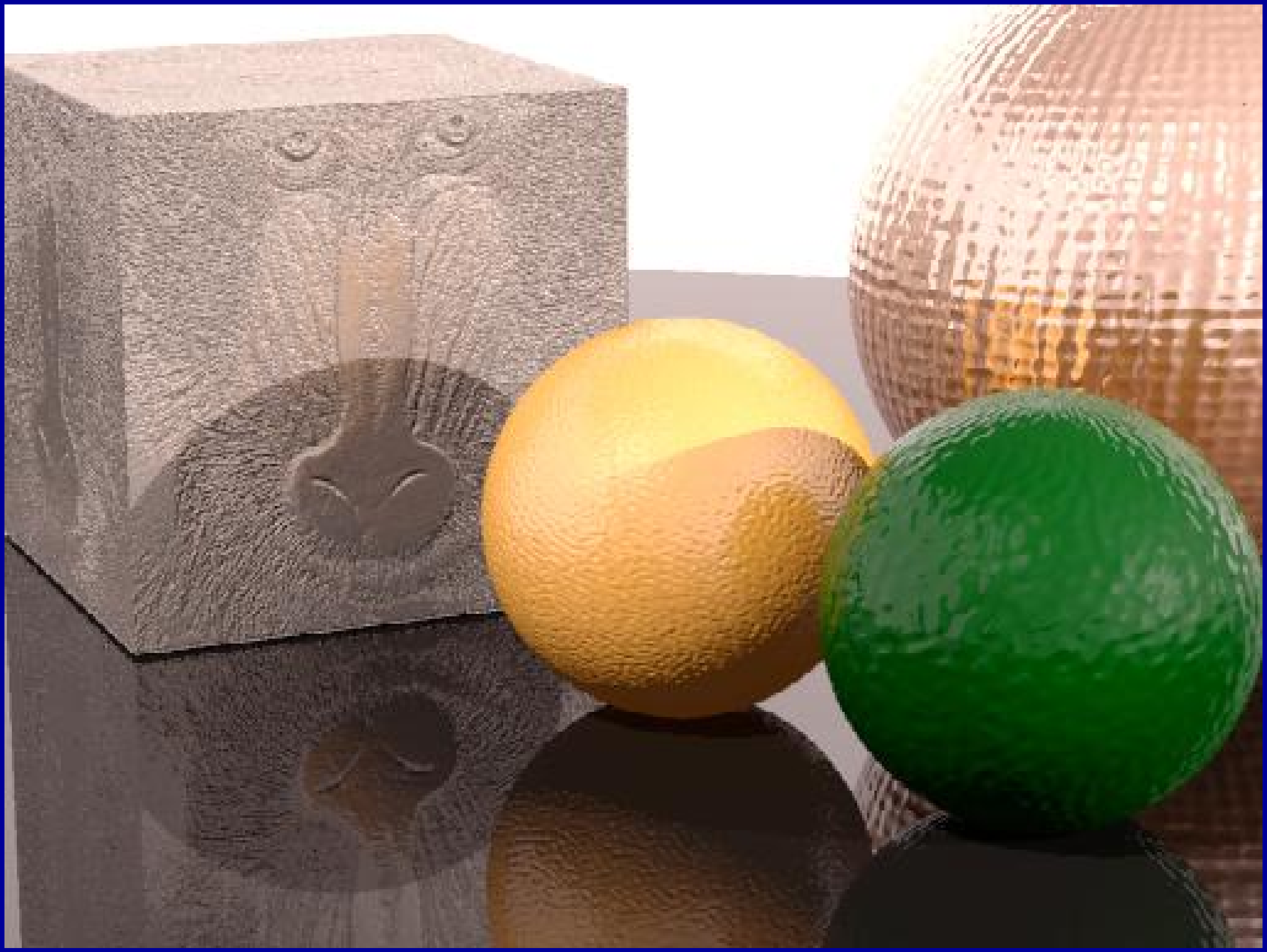
distribute rays across hemisphere to simulate diffuse interreflection

also called: “distributed ray tracing” or stochastic ray tracing

aliasing is replaced by less visually annoying noise.

powerful idea! (but requires significantly more computation)

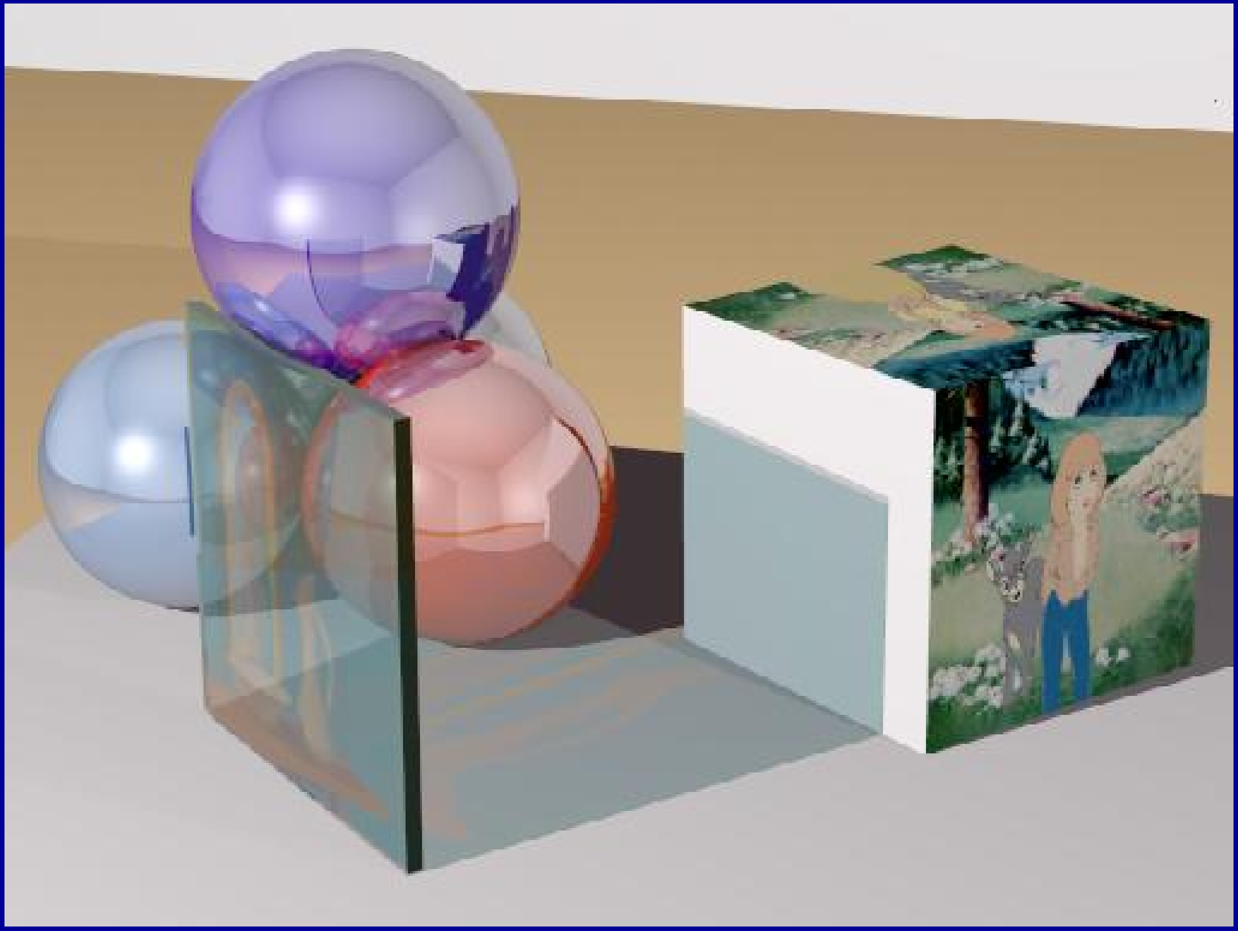
Examples



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/