Distribution Ray Tracing

Brian Curless
CSE 557
Fall 2015

Reading

Required:

- Shirley, 13.11, 14.1-14.3

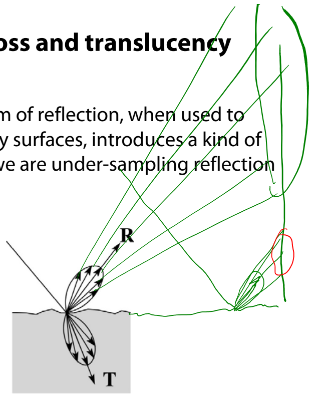
Further reading:

- A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989. [In the lab.]
- Robert L. Cook, Thomas Porter, Loren Carpenter. "Distributed Ray Tracing." Computer Graphics (Proceedings of SIGGRAPH 84). *18 (3)*. pp. 137-145. 1984.
- James T. Kajiya. "The Rendering Equation." Computer Graphics (Proceedings of SIGGRAPH 86). *20 (4)*. pp. 143-150. 1986.

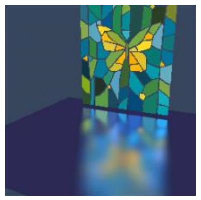
Simulating gloss and translucency

The mirror-like form of reflection, when used to approximate glossy surfaces, introduces a kind of aliasing, because we are under-sampling reflection (and refraction).

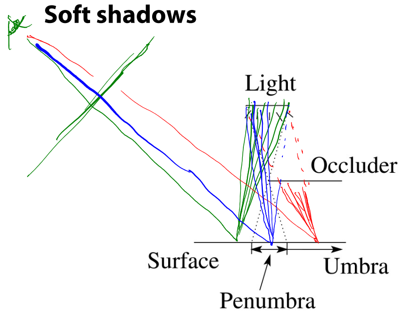
For example:

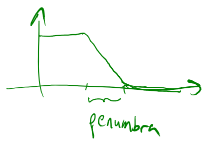


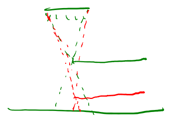
Distributing rays over reflection directions gives:



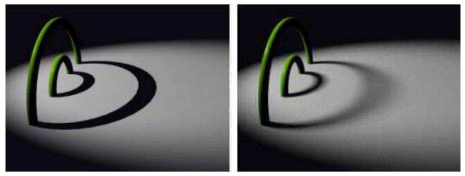
Soft shadows





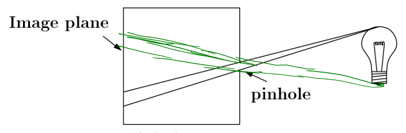


Distributing rays over light source area gives:



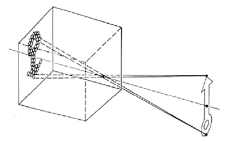
The pinhole camera

The first camera - “camera obscura” - known to Aristotle.



Pinhole camera

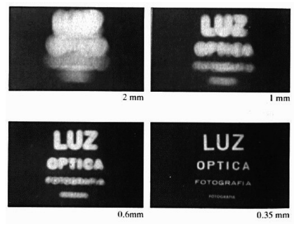
In 3D, we can visualize the blur induced by the pinhole (a.k.a., **aperture**):



Q: How would we reduce blur?

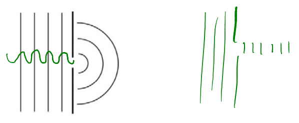
Shrink aperture

Shrinking the pinhole

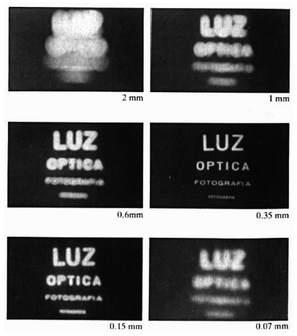


Q: What happens as we continue to shrink the aperture?

Shrinking the pinhole, cont’d

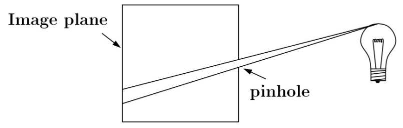


Diffraction



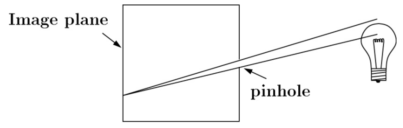
The pinhole camera, revisited

We can think in terms of light heading toward the image plane:



Pinhole camera

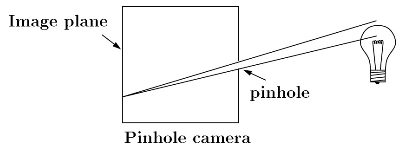
We can equivalently turn this around by following rays from the viewer:



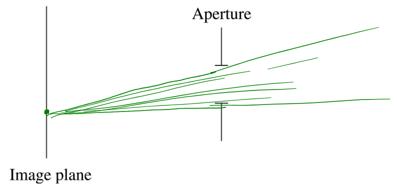
Pinhole camera

The pinhole camera, revisited

Given this flipped version:



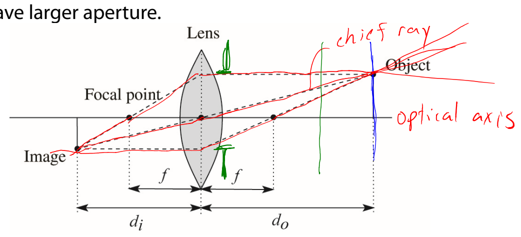
how can we simulate a pinhole camera more accurately?



Lenses

Pinhole cameras in the real world require small apertures to keep the image in focus.

Lenses focus a bundle of rays to one point => can have larger aperture.



For a "thin" lens, we can approximately calculate where an object point will be in focus using the the Gaussian lens formula:

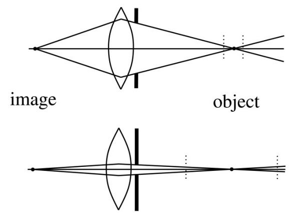
$$\frac{1}{d_i} + \frac{1}{d_o} = \frac{1}{f}$$

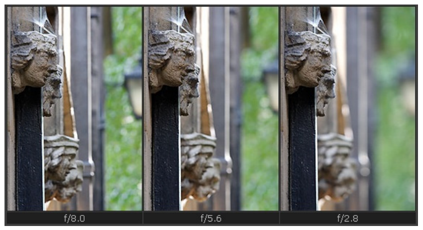
where f is the **focal length** of the lens.

Depth of field

Lenses do have some limitations. The most noticeable is the fact that points that are not in the object plane will appear out of focus.

The **depth of field** is a measure of how far from the object plane points can be before appearing "too blurry."

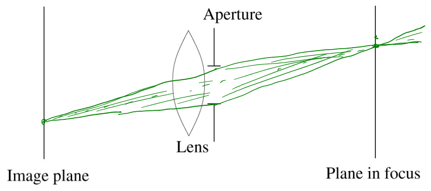




http://www.cambridgeincolour.com/tutorials/depth-of-field.htm

Depth of field (cont'd)

To simulate depth of field, we can model the refraction of light through a lens. Objects close to the in-focus plane are sharp, and the rest is blurry.



Depth of field (cont'd)

This is really similar to the pinhole camera model, but now:

- Put the image plane at the depth you want to be in focus.
- Treat the aperture as multiple COPs (samples across the aperture).
- For each pixel, trace multiple viewing/primary rays for each COP and average the results.

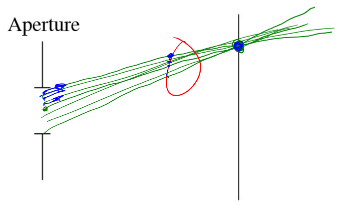


Image plane = plane in focus

Speeding it up, revisited

Sampling over all these effects makes ray tracing even slower!

Now consider rendering a single image with:

- *m* x *m* pixels
- *k* x *k* supersampling
- *a* x *a* sampling of camera aperture
- *n* primitives
- ℓ area light sources
- *s* x *s* sampling of each area light source
- *r* x *r* rays cast recursively per intersection (gloss/translucency)
- *d* is average ray path length

Without any acceleration we'd get:

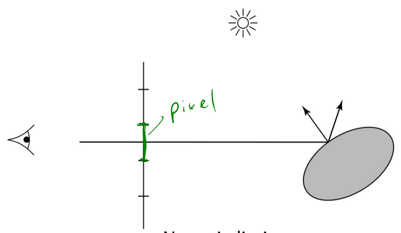
Asymptotic # of intersection tests = $O\left(m^2 \cdot k^2 \cdot a^2 \cdot n \cdot (r^2)^d \, \ell \cdot s^2\right)$

We've looked at reducing the cost of *d* (early ray termination), *n* (acceleration data structure), and *k* (adaptive super-sampling).

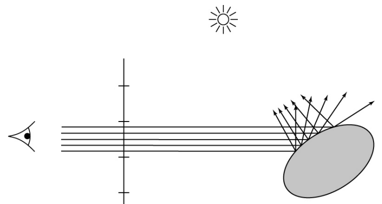
Now we look at reducing the effect of the *a*, *s*, and *r* terms.

But first…

Pixel anti-aliasing



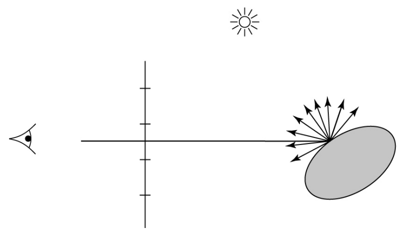
No anti-aliasing



Pixel anti-aliasing

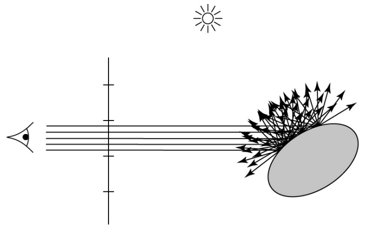
All of this assumes that inter-reflection behaves in a mirror-like fashion…

Reflection anti-aliasing



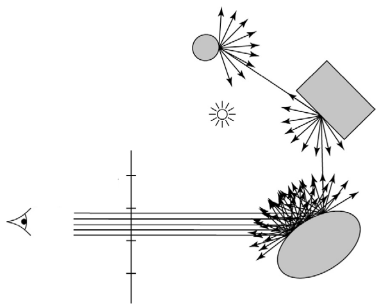
Reflection anti-aliasing

Pixel and reflection anti-aliasing



Pixel and reflection anti-aliasing

Full anti-aliasing of reflections

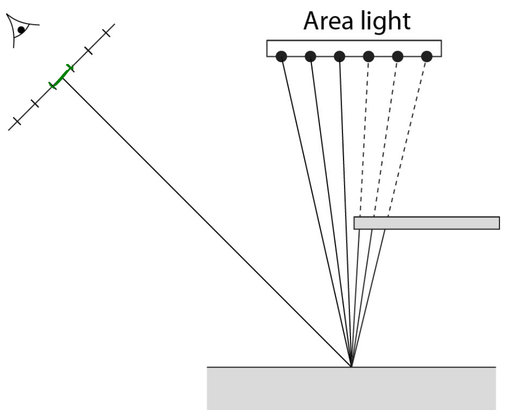


Full anti-aliasing over pixel and reflections

EEEK!!

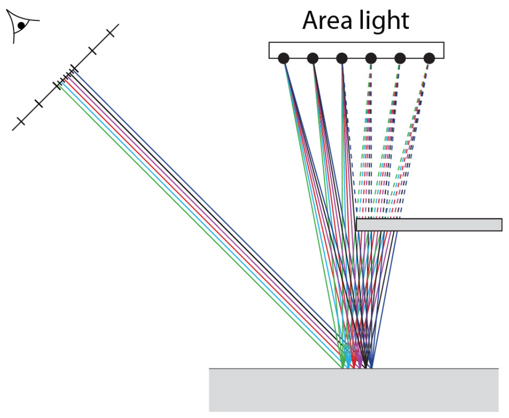
Penumbra revisited

Let's revisit the area light source…



We can trace a ray from the viewer through a pixel, but now when we hit a surface, we cast rays to samples on the area light source.

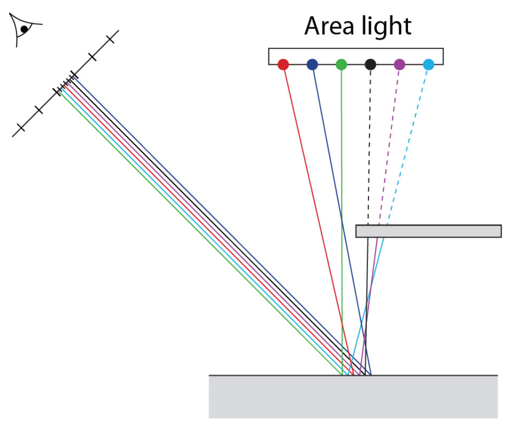
Penumbra: full antialaising



We should anti-alias to get best looking results.

Whoa, this is a lot of rays…just for one pixel!!

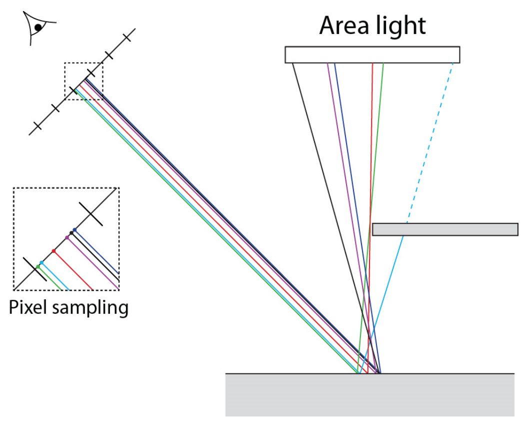
Penumbra: fewer rays



We can get a similar result with much less computation:

- Break up the light source into points with ID's.
- Similarly, give an ID to each sub-pixel ray.
- Only send shadow ray to point with same ID.

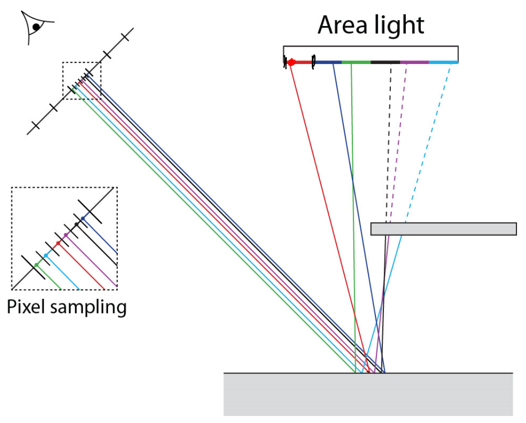
Penumbra: random sampling



Regular sampling of pixels and lights can introduce bias into the result.

An unbiased approach would be to choose subpixel locations and area light samples randomly.

Penumbra: jittered (stratified) sampling



A hybrid approach, which gives a better distribution of samples while being unbiased, is to "jitter" (stratify) the rays:

- Break pixel and light source into *regions*.
- Choose random locations within each region.
- Trace rays through/to those jittered locations.

Distribution ray tracing

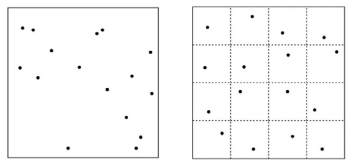
These ideas can be combined to give a particular method called **distribution ray tracing** [Cook84]:

- uses non-uniform (jittered) samples.
- replaces aliasing artifacts with noise.
- provides additional effects by distributing rays to sample:
  - Reflections and refractions
  - Light source area
  - Camera lens area

[This approach was originally called "distributed ray tracing," but we will call it distribution ray tracing (as in probability distributions) so as not to confuse it with a parallel computing approach.]

Stratified sampling of a 2D pixel

Here we see pure uniform vs. stratified sampling over a 2D pixel (here 16 rays/pixel):

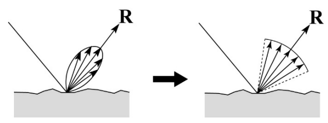


The stratified pattern on the right is also sometimes called a **jittered** sampling pattern.

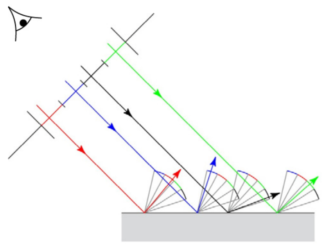
One interesting side effect of these stochastic sampling patterns is that they actually injects noise into the solution (slightly grainier images). This noise tends to be less objectionable than aliasing artifacts.

Glossy reflection, revisited

For illustration (and "ok" simulation), we can approximate specular reflection as:



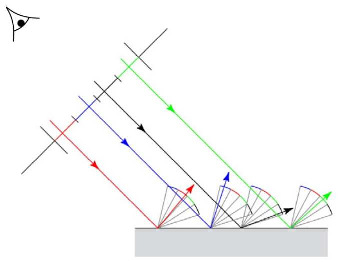
For distribution ray tracing, we break the reflection directions into bins with IDs and distribute rays accordingly:



DRT pseudocode

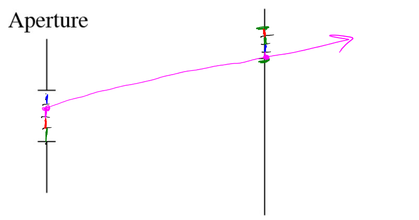
Now consider *traceRay*(), modified to handle opaque glossy surfaces:

```
function traceRay(scene, P, d, ID):
   (t, N, mtrl) ← intersect(scene, P, d)
   Q ← ray (P, d) evaluated at t
   I = shade(scene, mtrl, Q, N, -d, ID)
   R ← jitteredReflectDirection(N, -d, mtrl, ID)
   I ← I + material.k_r * traceRay(scene, Q, R, ID)
   return I
end function
```



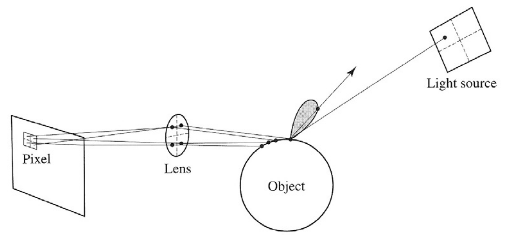
Depth of field revisited

We can also perform distribution ray tracing across a camera aperture:



Chaining the ray ID's

In general, you can trace rays through a scene and keep track of their id's to handle *all* of these effects:



Q: Do you end up tracing any more rays than you would with a standard, anti-aliased Whitted ray tracer?