Anti-aliased, distribution, accelerated ray tracing

Brian Curless
CSE 457
Spring 2015

Reading

Required:

- Shirley 10.9, 10.11 (online handout)

Further reading:

- A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989.
- 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.

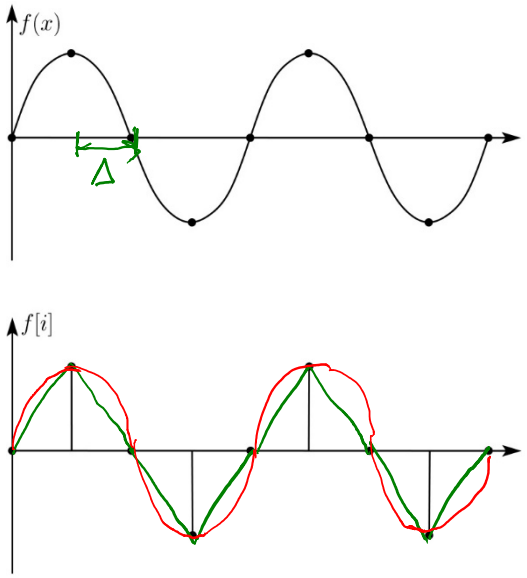
Aliasing

Ray tracing is a form of sampling and can suffer from annoying visual artifacts...

Consider a continuous function $f(x)$. Now sample it at intervals Δ to give $f[i] = \text{quantize}[f(i\Delta)]$.

Q: How well does $f[i]$ approximate $f(x)$?

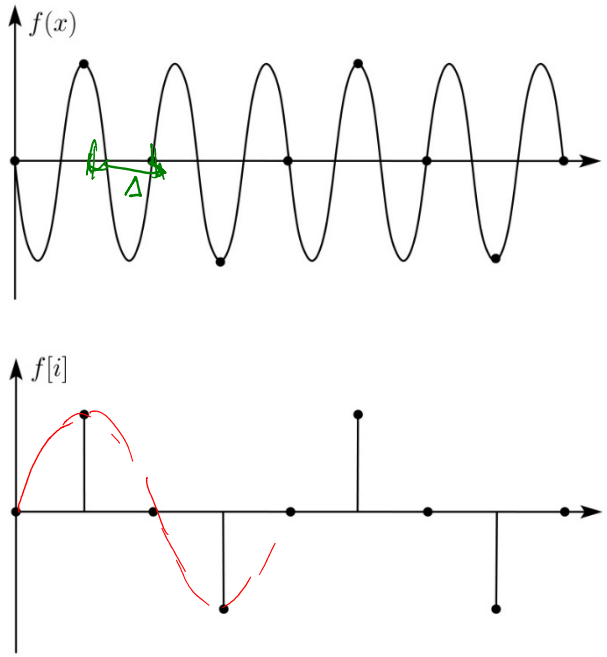
Consider sampling a sinusoid:



In this case, the sinusoid is reasonably well approximated by the samples.

Aliasing (con't)

Now consider sampling a higher frequency sinusoid

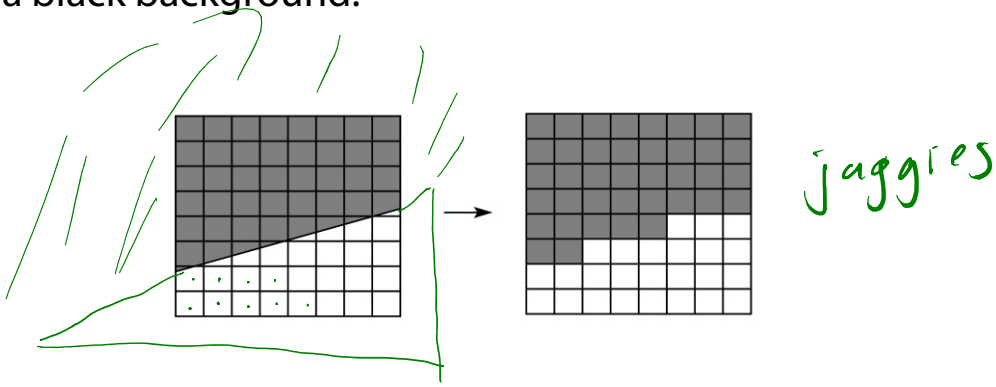


We get the exact same samples, so we seem to be approximating the first lower frequency sinusoid again.

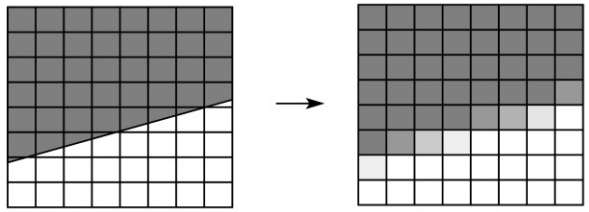
We say that, after sampling, the higher frequency sinusoid has taken on a new "alias", i.e., changed its identity to be a lower frequency sinusoid.

Aliasing in rendering

One of the most common rendering artifacts is the "jaggies". Consider rendering a white polygon against a black background:



We would instead like to get a smoother transition:

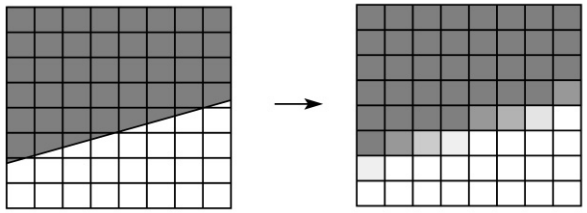


Anti-aliasing

Q: How do we avoid aliasing artifacts?

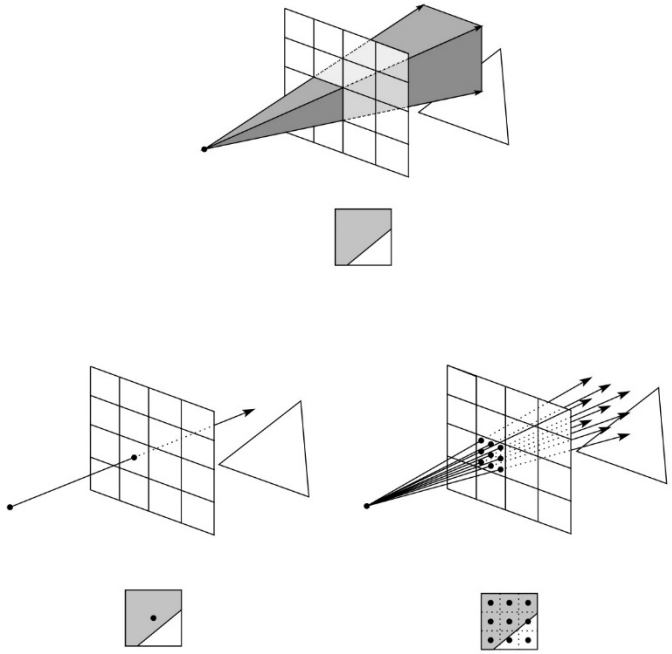
1. Sampling: increase sampling ... need higher res display
2. Pre-filtering: analytic integration ... too hard for real problems
3. Combination: super-sampling + averaging down

Example - polygon:



Antialiasing in a ray tracer

We would like to compute the average intensity in the neighborhood of each pixel.



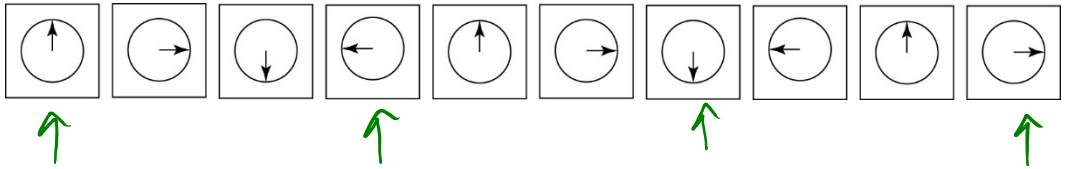
When casting one ray per pixel, we are likely to have aliasing artifacts.

To improve matters, we can cast more than one ray per pixel and average the result.

A.k.a., **super-sampling and averaging down**.

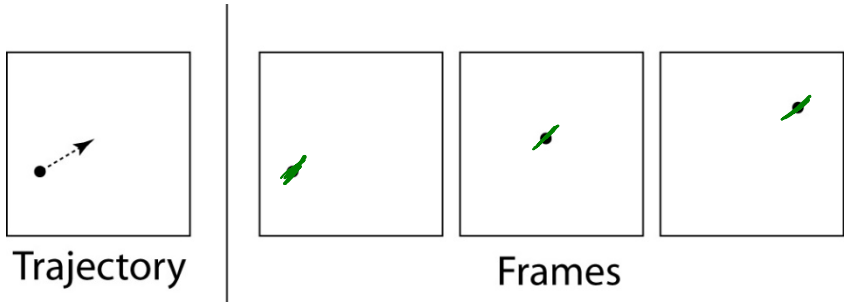
Temporal aliasing

Suppose we are rendering a "clock" with a fast turning hand:



What happens if we sample too infrequently? (This is sometimes called the "wagon wheel" effect.)

Another more common scenario is something moving quickly across the frame, e.g., a fast-moving particle:



motion blur

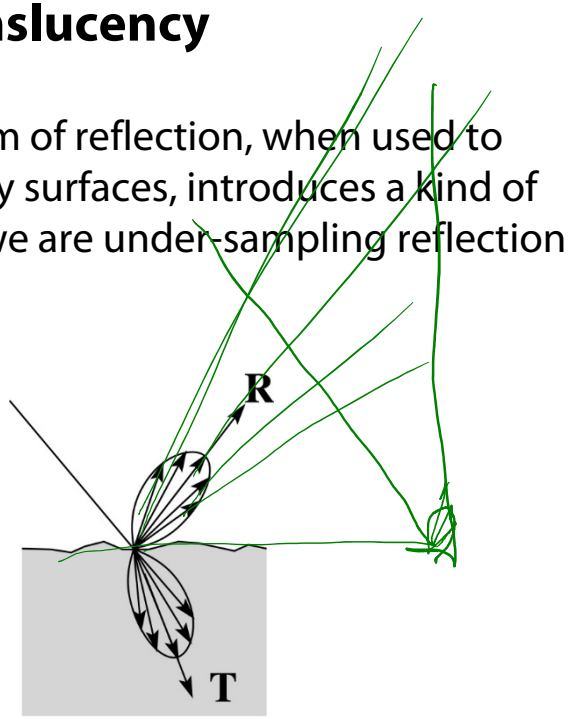
How might we address these temporal aliasing effects?

super-sampling (in time) and averaging down

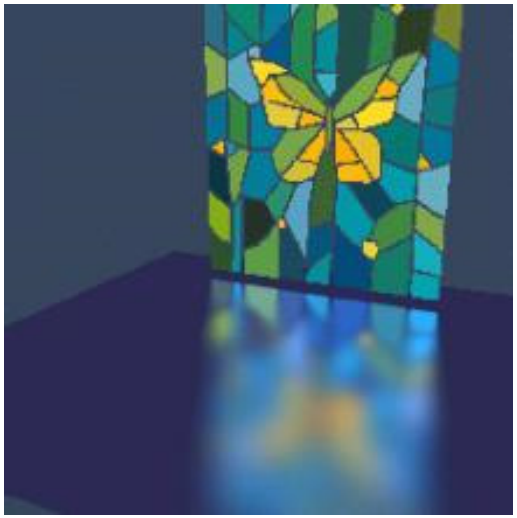
Gloss and translucency

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).

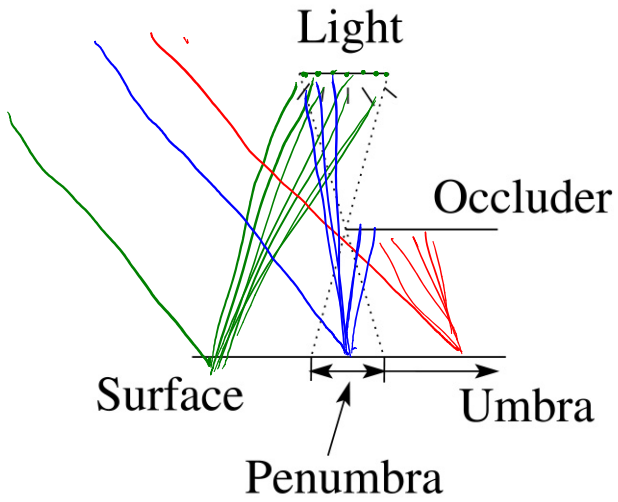
For example:

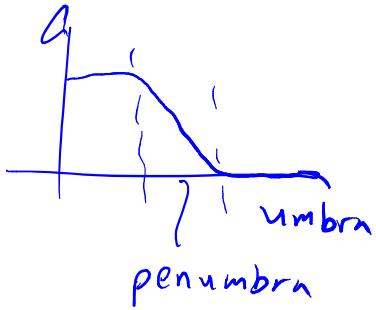


Distributing rays over reflection directions gives:

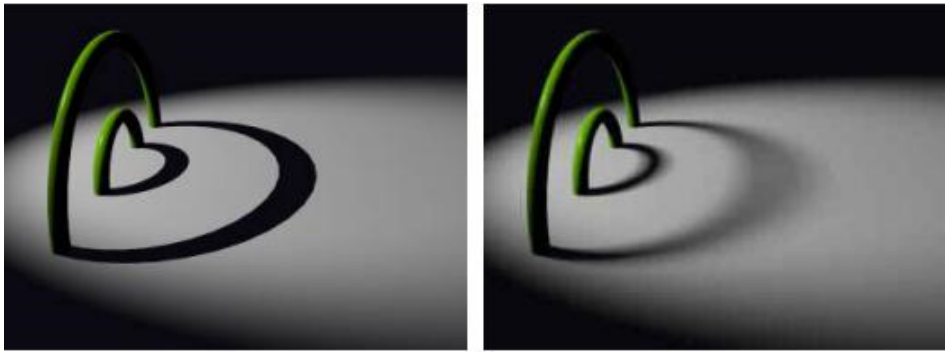


Soft shadows



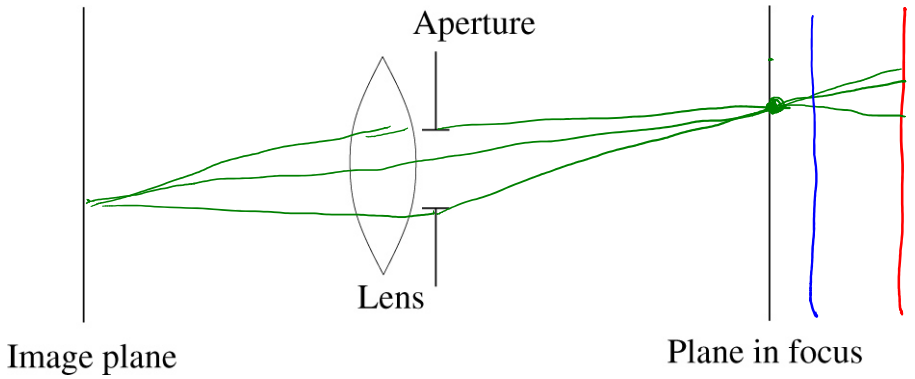


Distributing rays over light source area gives:



Depth of field

To simulate a camera, we can model the refraction of light through a lens. This will give us a "depth of field" effect: 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:

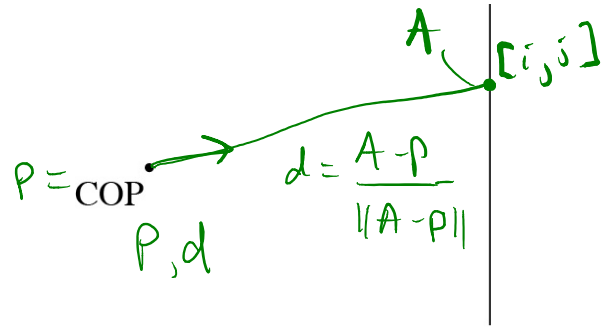


Image plane

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.

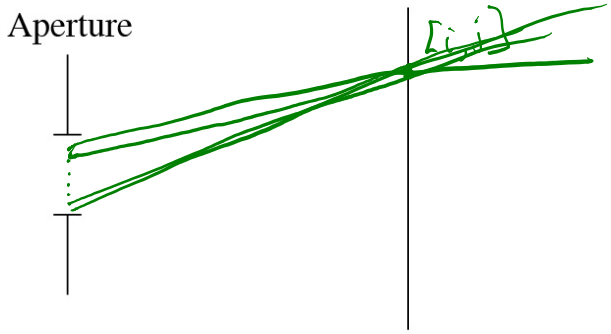


Image plane = plane in focus

Motion blur

Distributing rays over time gives:

Speeding it up

Brute force ray tracing is really slow!

Consider rendering a single image with:

- m xm pixels
- k xk supersampling
- a xa 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 of d

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

For m =1,000, $k = a = s = r = 8$, n = 1,000,000, $\ell = 10$, $d = 8$…very expensive!!

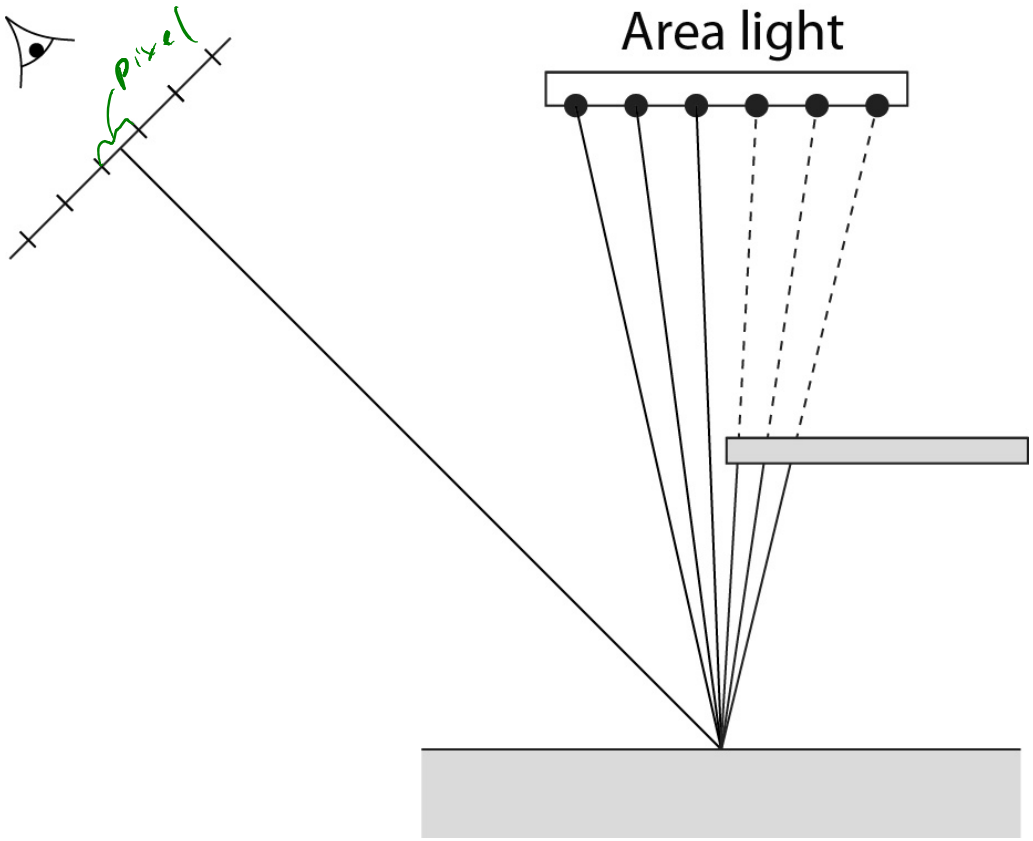
In practice, some acceleration technique is almost always used.

We've already looked at reducing d with adaptive (early) ray termination.

Now we look at reducing the effect of the a, s, r, k and n terms…

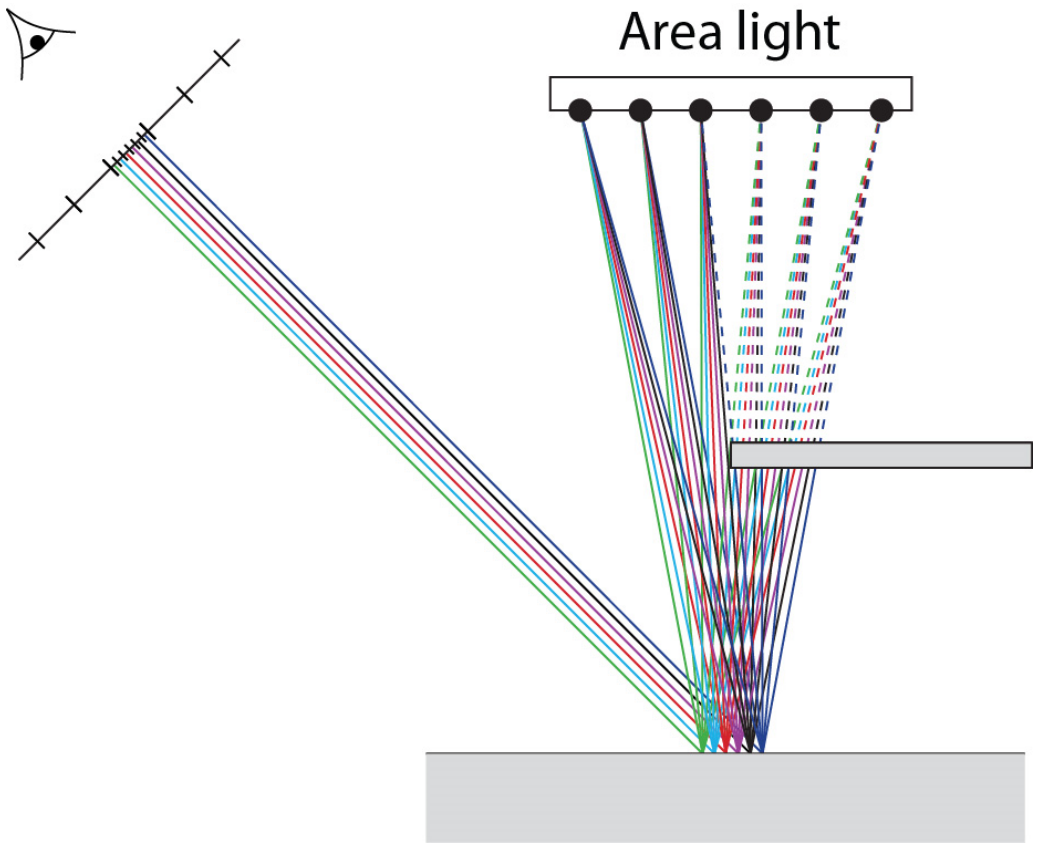
Penumbra revisited

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.

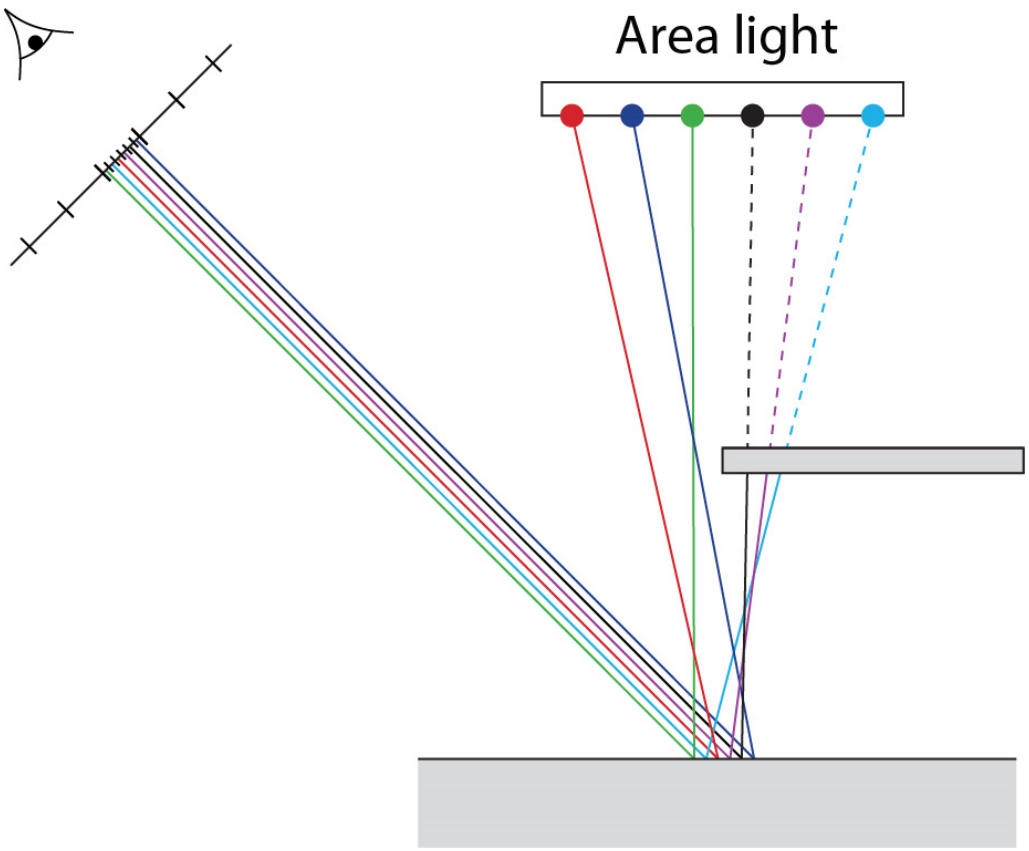
Penumbra revisited



We should anti-alias to get best looking results.

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

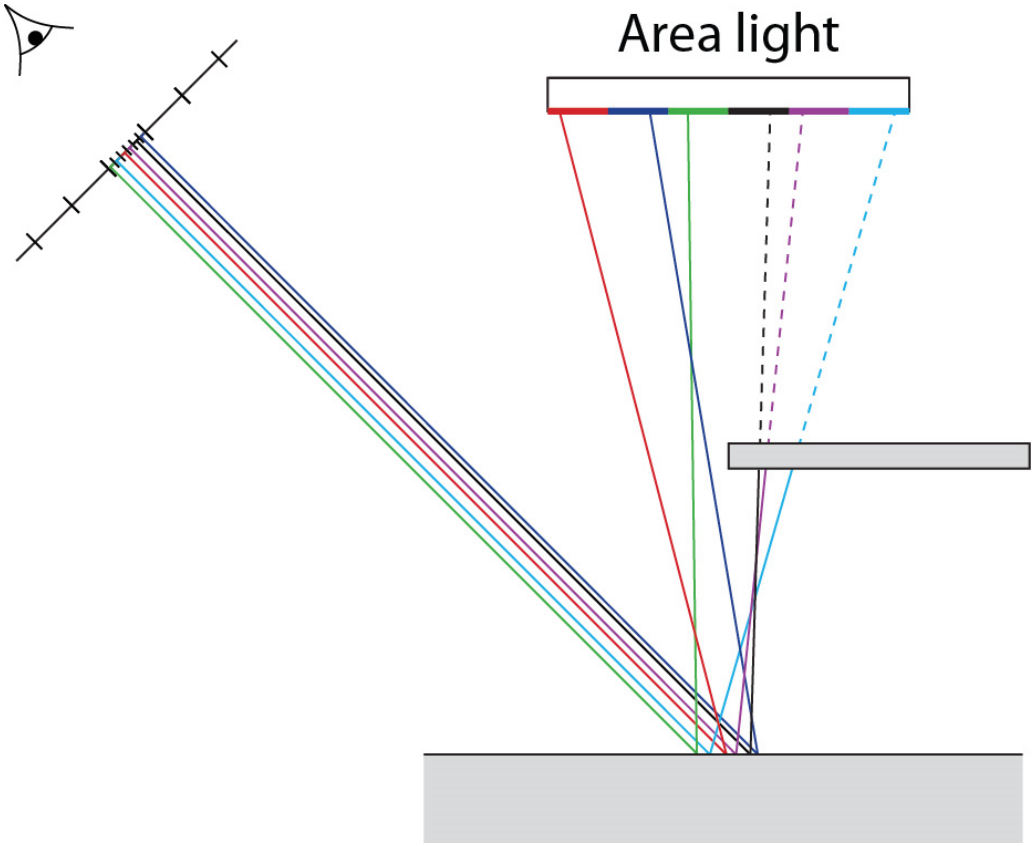
Penumbra revisited



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 rays.
- Only send shadow ray to point with same ID.

Penumbra revisited



For even (statistically) better results, “jitter” the rays:

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

Distribution ray tracing

This idea is called **distribution ray tracing** [Cook84]:

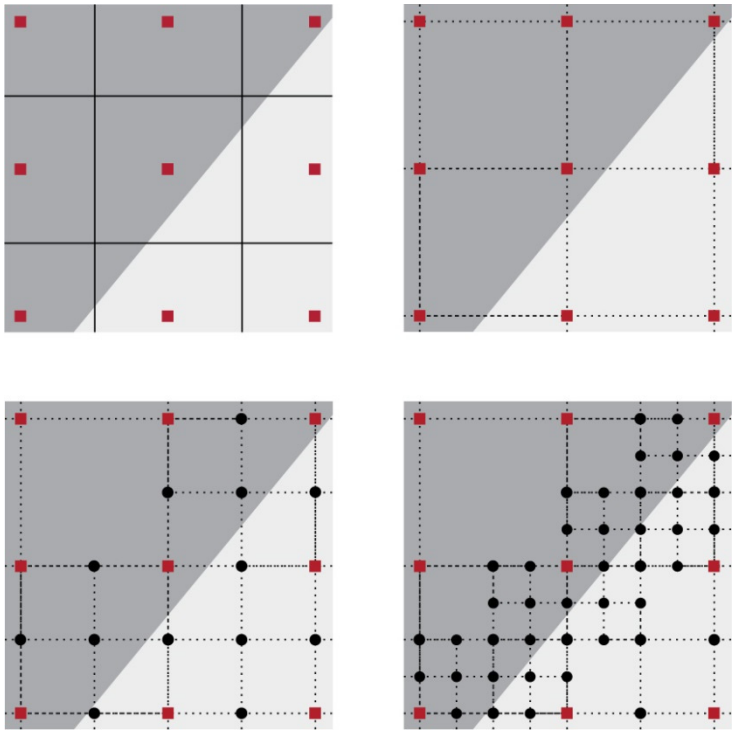
- uses non-uniform (jittered) samples.
- replaces aliasing artifacts with noise.
- provides additional effects by distributing rays to sample:
  - Reflections and refractions
  - Light source area
  - Camera lens area
  - Time

[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.]

Antialiasing by adaptive sampling

Casting many rays per pixel can be unnecessarily costly. If there are no rapid changes in intensity at the pixel, maybe only a few samples are needed.

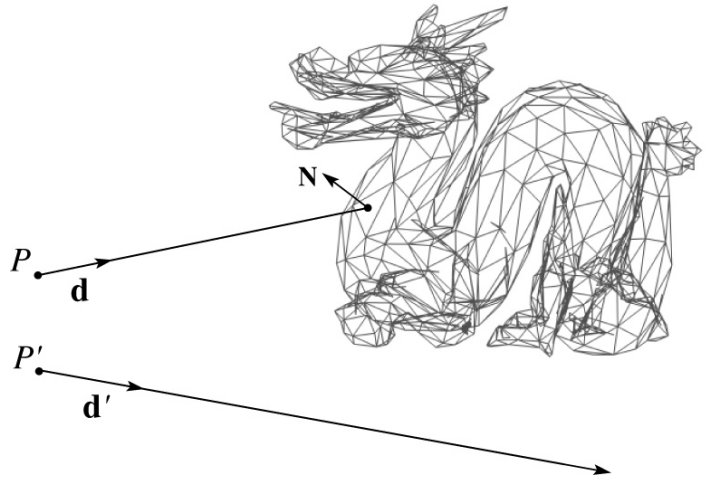
Solution: **adaptive sampling**.



Q: When do we decide to cast more rays in a particular area?

Faster ray-polyhedron intersection

Let's say you were intersecting a ray with a triangle mesh:



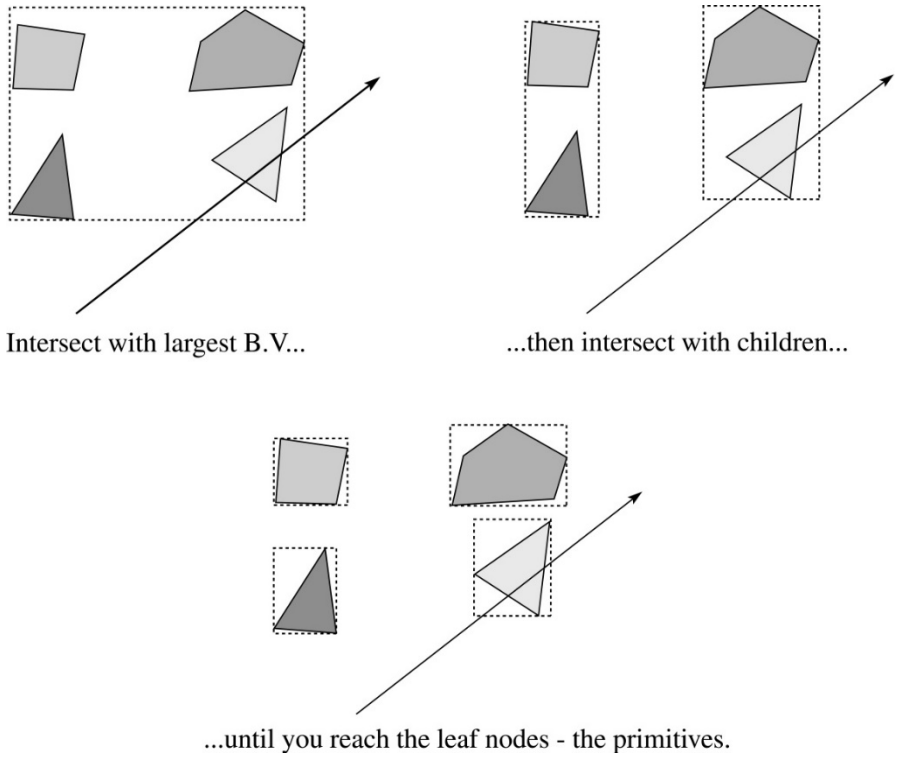
Straightforward method

- intersect the ray with each triangle
- return the intersection with the smallest *t*-value.

Q: How might you speed this up?

Hierarchical bounding volumes

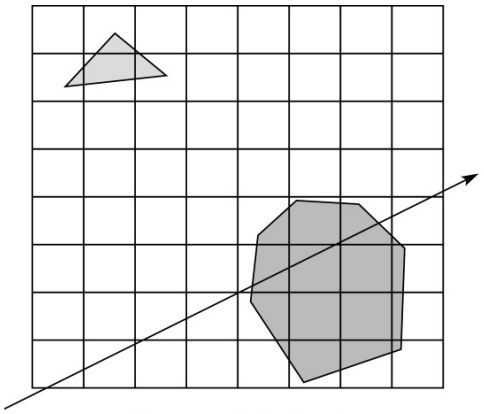
We can generalize the idea of bounding volume acceleration with **hierarchical bounding volumes**.



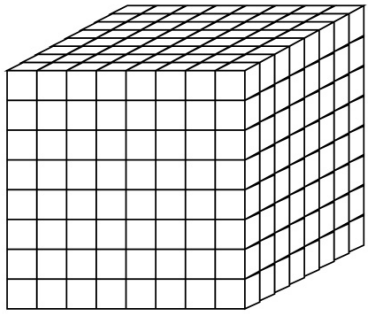
Key: build balanced trees with *tight bounding volumes*.

Uniform spatial subdivision

Another approach is **uniform spatial subdivision**.



Uniform subdivion in 2D



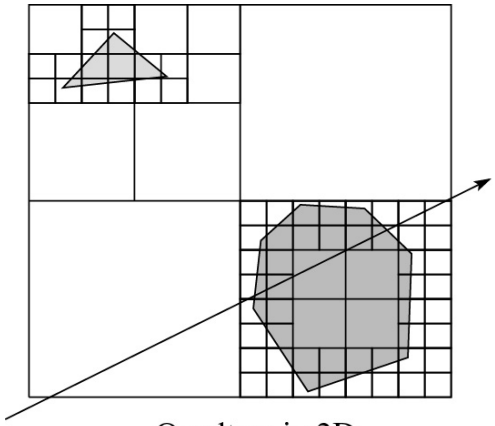
Uniform subdivion in 3D

Idea:

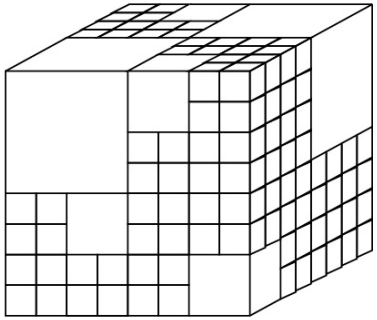
- Partition space into cells (voxels)
- Associate each primitive with the cells it overlaps
- Trace ray through voxel array *using fast incremental arithmetic* to step from cell to cell

Non-uniform spatial subdivision

Still another approach is **non-uniform spatial subdivision**.



Quadtree in 2D



Octree in 3D

Other variants include k-d trees and BSP trees.

Various combinations of these ray intersection techniques are also possible.

Summary

What to take home from this lecture:

- The meanings of all the boldfaced terms.
- An intuition for what aliasing is.
- How to reduce aliasing artifacts in a ray tracer
- The limitations of Whitted ray tracing (no glossy surfaces, etc.)
- The main idea behind distribution ray tracing and what effects it can simulate (glossy surfaces, etc.)
- An intuition for how ray tracers can be accelerated.