Anti-aliased, Monte Carlo, accelerated ray tracing

Brian Curless
CSE 457
Spring 2017

Reading

Required:

- Marschner and Shirley, Sections 12.3 and 13.4 (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.

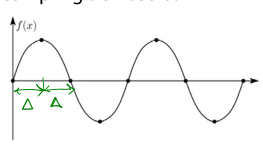
Aliasing

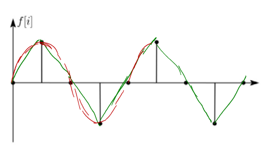
Ray tracing is a form of sampling and can suffer from annoying visual artifacts...

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

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

Consider sampling a sinusoid:

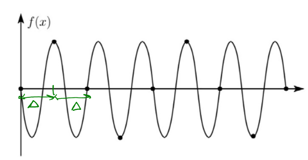


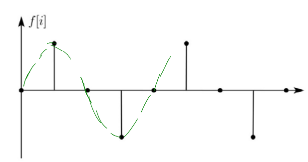


In this case, the sinusoid is reasonably well approximated by the samples.

Aliasing (con't)

Now consider sampling a higher frequency sinusoid



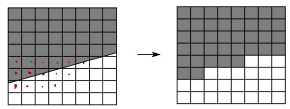


We get the exact same samples, so we seem to be approximating the first lower frequency sinusoid again.

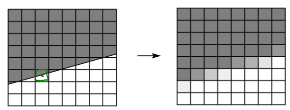
We say that, after sampling, the higher frequency sinusoid has taken on a new "alias", i.e., changed its identity to be a lower frequency sinusoid.

Aliasing and anti-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:

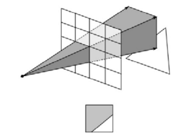


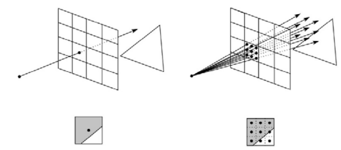
Anti-aliasing is the process of removing high frequencies *before* they cause aliasing.

In a renderer, computing the average color within a pixel is a good way to anti-alias. How exactly do we compute the average color?

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.

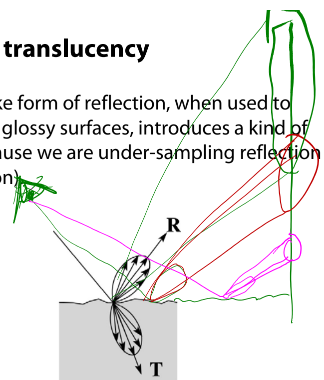
To improve matters, we can cast more than one ray per pixel and average the result.

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

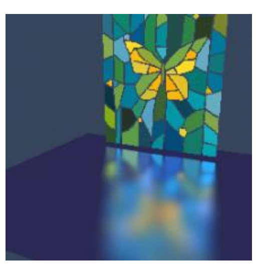
Gloss and translucency

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).

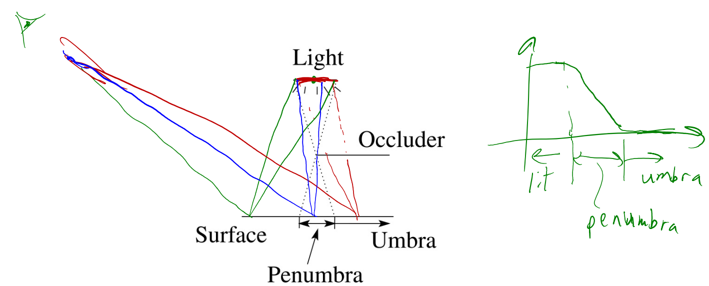
For example:



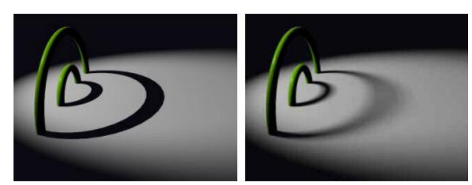
Distributing rays over reflection directions gives:



Soft shadows

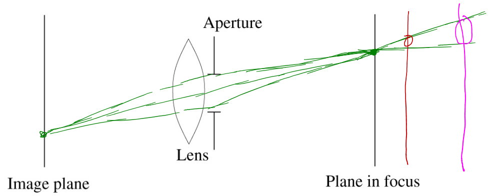


Distributing rays over light source area gives:



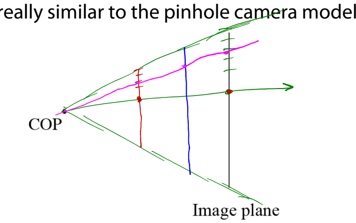
Depth of field

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.



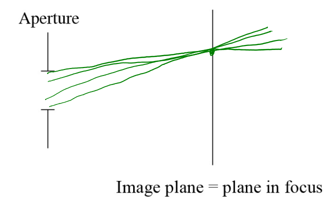
Depth of field (cont'd)

This is really similar to the pinhole camera model:



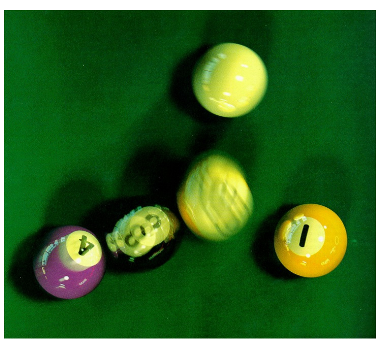
But now:

- 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.



Motion blur

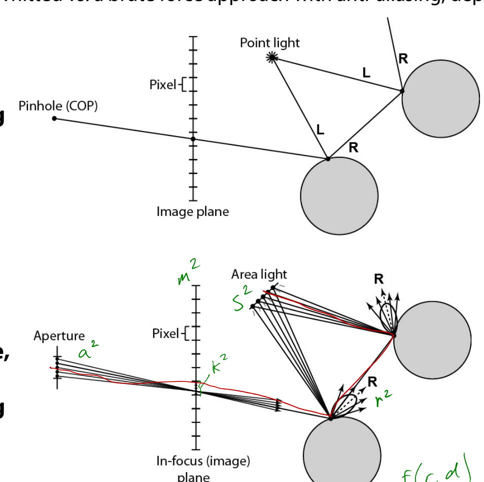
Distributing rays over time gives:



How can we use super-sampling and averaging down to get motion blur?

Naively improving Whitted ray tracing

Consider Whitted vs. a brute force approach with anti-aliasing, depth of field, area lights, gloss…



Advanced ray tracing has:

- 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

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

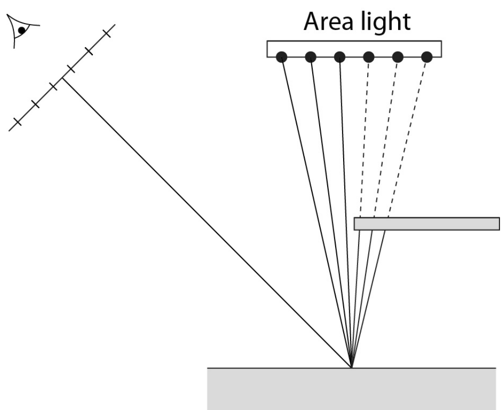
$10^6\, 2^6\, 2^6\, 2^6\, 2^6\, 2^3 \ldots$

For m =1,000, $k = a = s = r = 8$, n = 1,000,000, $\ell = 4$, $d = 8$ … very expensive!!

$10^6\, 2^{26} \approx 10^6 \cdot 2^{20} \cdot 2^6 \approx 10^6 \cdot 10^6 \cdot 2^6 \approx 6 \times 10^{13} \ldots \approx 10^{20}$

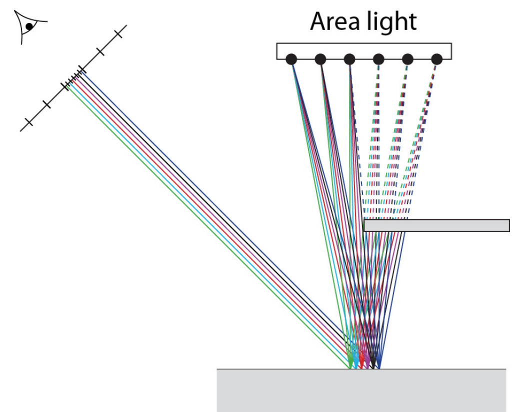
Penumbra revisited

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.

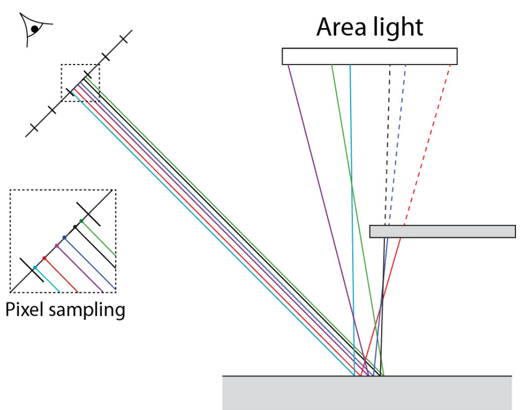
Penumbra revisited



We should anti-alias to get best looking results.

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

Penumbra revisited



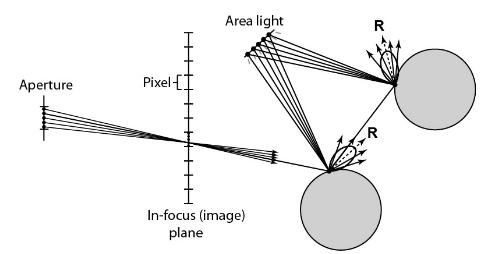
We can get a similar result with **much** less computation:

- Choose random location within a pixel, trace ray.
- At first intersection, choose random location on area light source and trace shadow ray.
- Continue recursion as with Whitted, but always choose random location on area light for shadow ray.

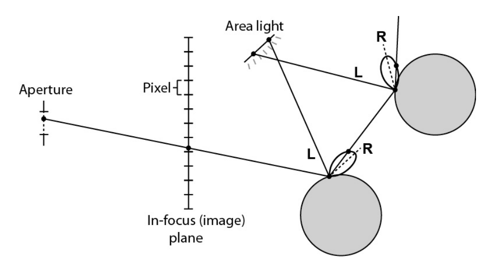
Monte Carlo Path Tracing vs. Brute Force

We can generalize this idea to do random sampling for each viewing ray, shadow ray, reflected ray, etc. This approach is called **Monte Carlo Path Tracing** (MCPT).

Brute force, advanced ray tracing



Monte Carlo path tracing

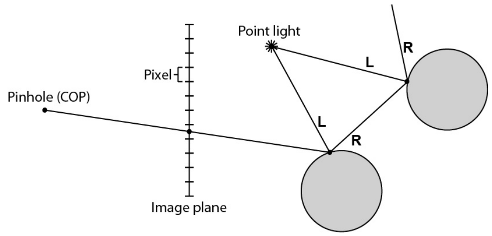


MCPT vs. Whitted

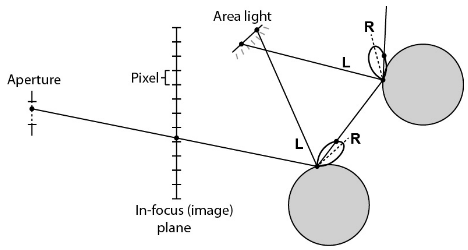
Q: For a fixed number of rays per pixel, does MCPT trace more total rays than Whitted? No.

Q: Does MCPT give the same answer every time?

Whitted ray tracing

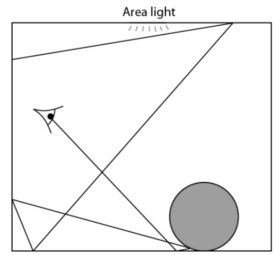


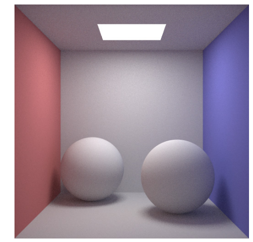
Monte Carlo path tracing



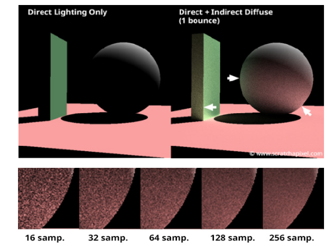
Noise and MCPT

You can also model diffuse interreflection by reflecting rays in completely random directions (and weighting the result of each bounce by **N•d**).





[http://web.stanford.edu/~dritchie/path]



[http://scratchapixel.com]

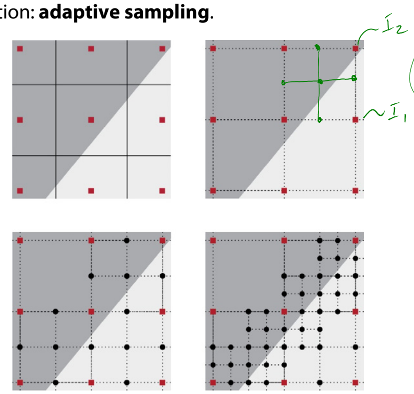
MCPT images tend to be noisy, especially with depth of field or diffuse interreflection. Reduce noise by:

- Casting many rays per pixel (lots of anti-aliasing)
- Importance sampling (choose rays that are likely to collect the most energy)
- Stratified sampling (distribute rays "evenly" to avoid accidentally casting rays that are too close together – see "distribution ray tracing," section 13.4 of handout)
- Filtering the final result (e.g., fancy bilateral filtering)

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**.



$\|\nabla I\| > \text{thresh}$

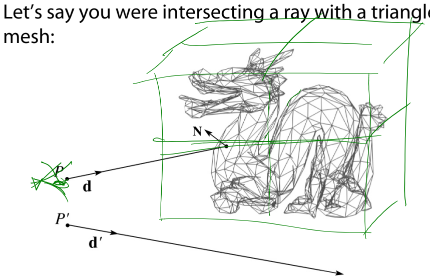
$\|I_1 - I_2\| > \text{thresh}$

$$\frac{\|I_1 - I_2\|}{\|\frac{1}{2}(I_1 + I_2)\|} > \text{thresh}$$

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

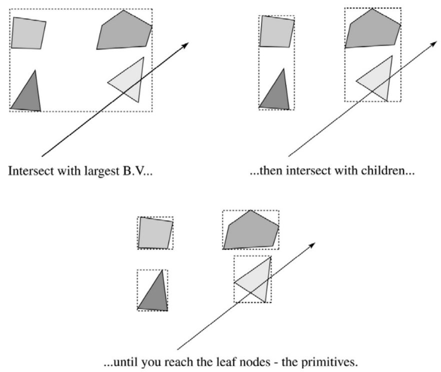
- intersect the ray with each triangle
- return the intersection with the smallest *t*-value.

Q: How might you speed this up?

intersect w/ bbox

Bounding Volume Hierarchies (BVHs)

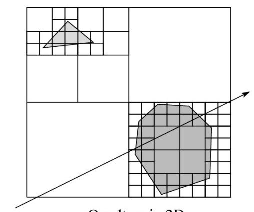
We can generalize the idea of bounding volume acceleration with **bounding volume hierarchies (BVHs)**.



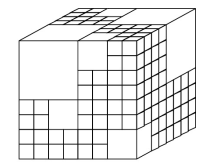
Key: build balanced trees with *tight bounding volumes*.

Non-uniform spatial subdivision: octrees

Another approach is **non-uniform spatial subdivision**. One version of this is octrees:



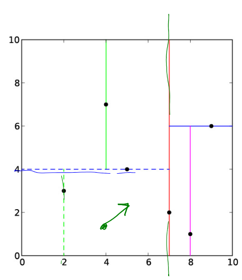
Quadtree in 2D



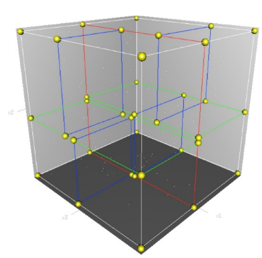
Octree in 3D

Non-uniform spatial subdivision: *k*-d trees

Another non-uniform subdivision is *k*-d (*k*–dimensional) trees:



k-d tree ($d = 2$)



k-d tree ($d = 3$)

If the planes can be non-axis aligned, then you get BSP (binary space partitioning) trees.

Various combinations of these ray intersections techniques are also possible.

[Image credits: Wikipedia.]

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 Monte Carlo path tracing and what effects it can simulate (glossy surfaces, etc.)
- An intuition for how ray tracers can be accelerated.