Anti-aliased and accelerated ray tracing

Brian Curless
CSE 557
Fall 2013

Reading

Required:

- Shirley 12.3, 13.4.1

Further reading:

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

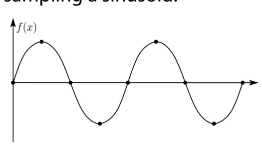
Aliasing

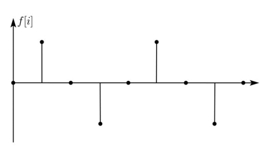
Ray tracing is a form of sampling and can suffer from annoying visual artifacts...

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:

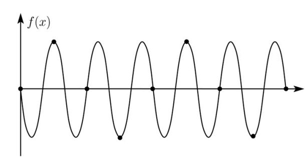


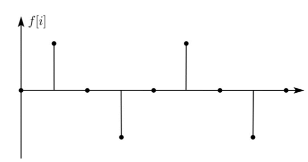


In this case, the sinusoid is reasonably well approximated by the samples.

Aliasing (con't)

Now consider sampling a higher frequency sinusoid



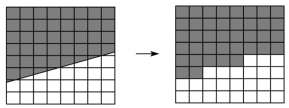


We get the exact same samples, so we seem to be approximating the first lower frequency sinusoid again.

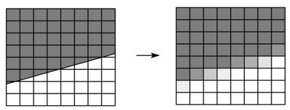
We say that, after sampling, the higher frequency sinusoid has taken on a new "alias", i.e., changed its identity to be a lower frequency sinusoid.

Aliasing 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:

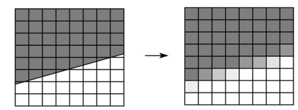


Anti-aliasing

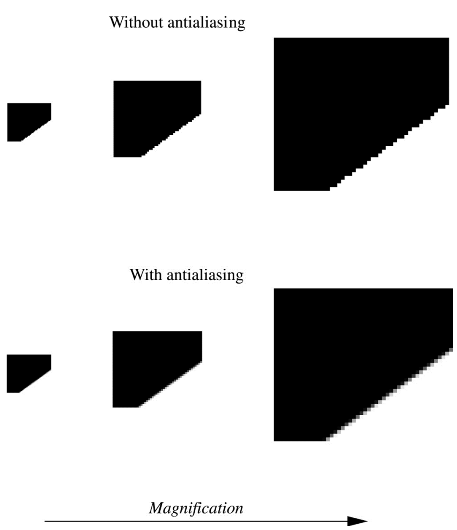
Q: How do we avoid aliasing artifacts?

1. Sampling:
2. Pre-filtering:
3. Combination:

Example - polygon:

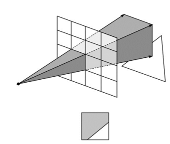


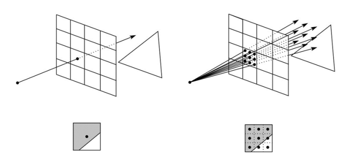
Polygon anti-aliasing



Antialiasing in a ray tracer

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





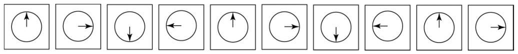
When casting one ray per pixel, we are likely to have aliasing artifacts.

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

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

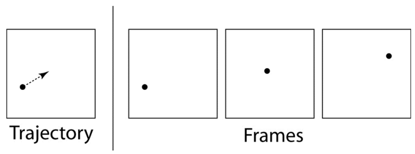
Temporal aliasing

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:



How might we address these temporal aliasing effects?

Speeding it up

Brute force ray tracing is really slow!

Consider rendering a single image with:

- m x m pixels
- k x k supersampling
- n primitives
- average ray path length of d
- ℓ shadow ray per intersection
- 0, 1, or 2 rays cast recursively per intersection

Asymptotic # of intersection tests =

For m=1,000, k = 5, n = 100,000, $\ell = 10$, d=8…very expensive!!

In practice, some acceleration technique is almost always used.

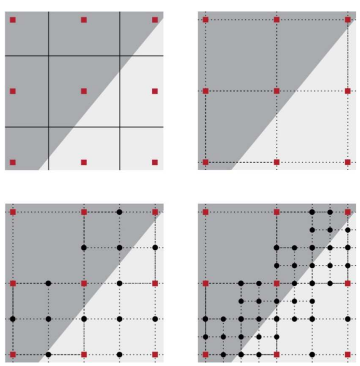
We've already looked at reducing d with adaptive (early) ray termination.

Now we look at reducing the effect of the k and n terms…

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.

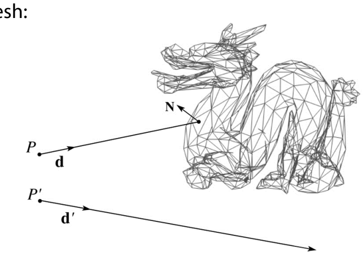
Solution: **adaptive sampling**.



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:



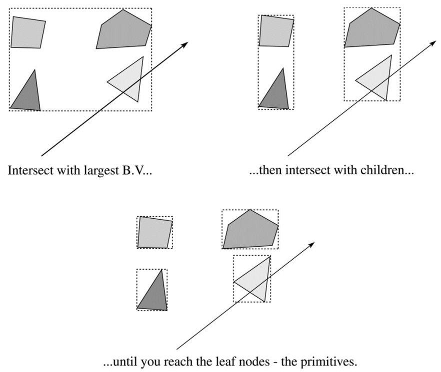
Straightforward method

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

Q: How might you speed this up?

Hierarchical bounding volumes

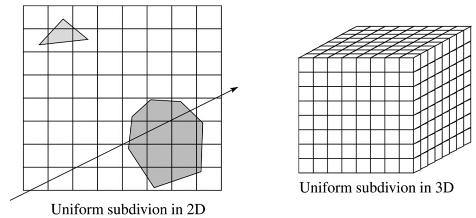
We can generalize the idea of bounding volume acceleration with **hierarchical bounding volumes**.



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

Uniform spatial subdivision

Another approach is **uniform spatial subdivision**.

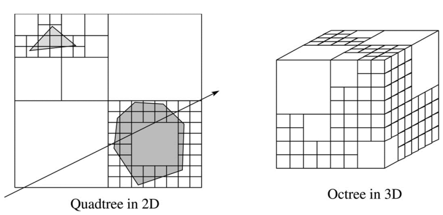


Idea:

- 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**.



Other variants include k-d trees and BSP trees.

Various combinations of these ray intersections techniques are also possible. See Shirley, Glassner, and pointers at bottom of project web page for more.