

Ray Tracing: The Final Chapter

CSG

Pretty easy using the method of Roth (“Ray Casting for Modeling Solids”, Computer Graphics and Image Processing **18**, 1982 – I’ve put a copy in class notebook).

Describe CSG

- Union (+)
- Intersection (&)
- Difference (-)

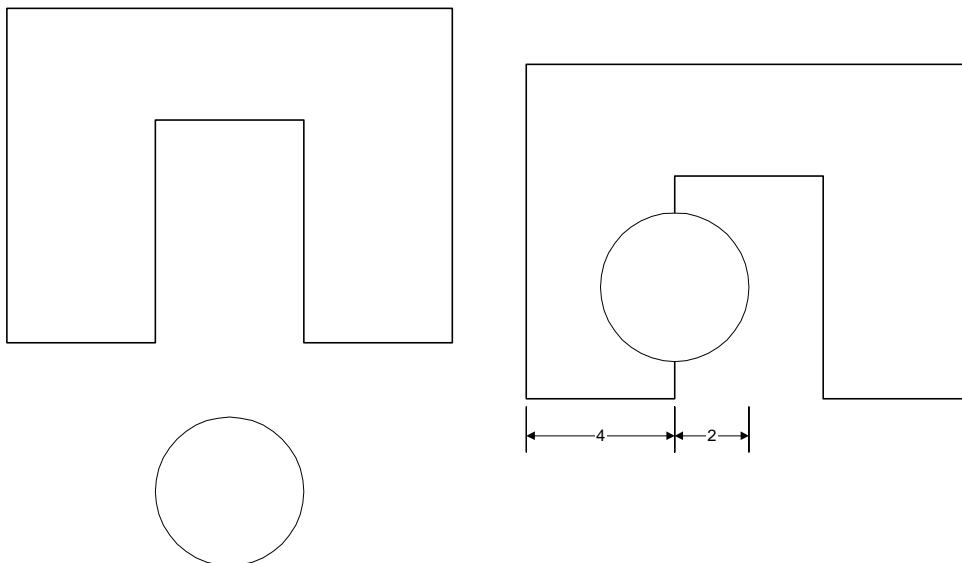
Examples, including Redentore.

Can create boundary model but ray tracing works great.

Roth came up with method.

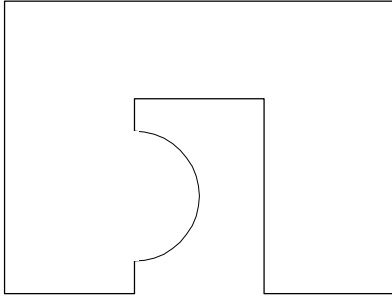
1. Trace ray to each component and log the intersections into in-out lists.
2. Perform the Boolean operations on these lists.
 - a. Merge the sorted lists.
 - b. Classify the merged points as in or out (table).
 - c. Combine adjacent segments.

Example in 2D

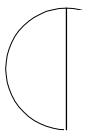


Two objects, an arch (A) and a circle (C) . We’ll position them, overlapping as in the right picture.

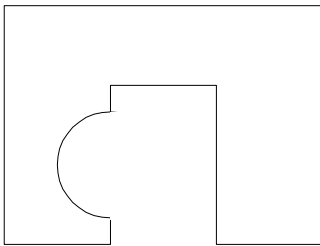
Union (A + C)



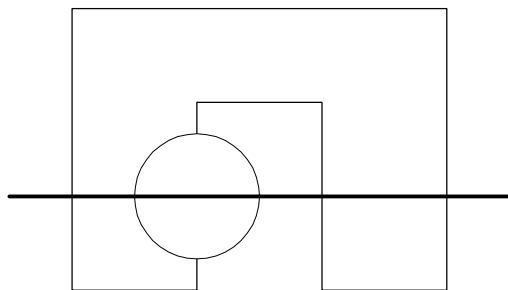
Intersection (A & C)



Difference (A - C)

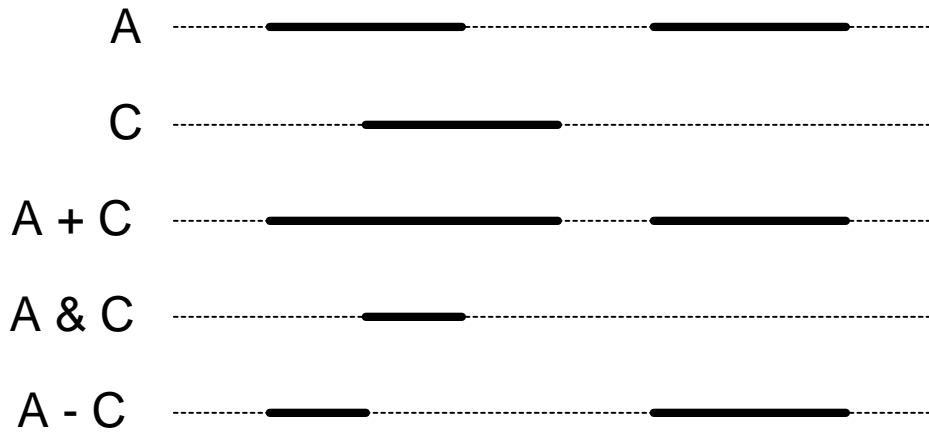


Cast Ray



Log intersections into in/out lists.

Follow up by combining lists according to a table of rules.

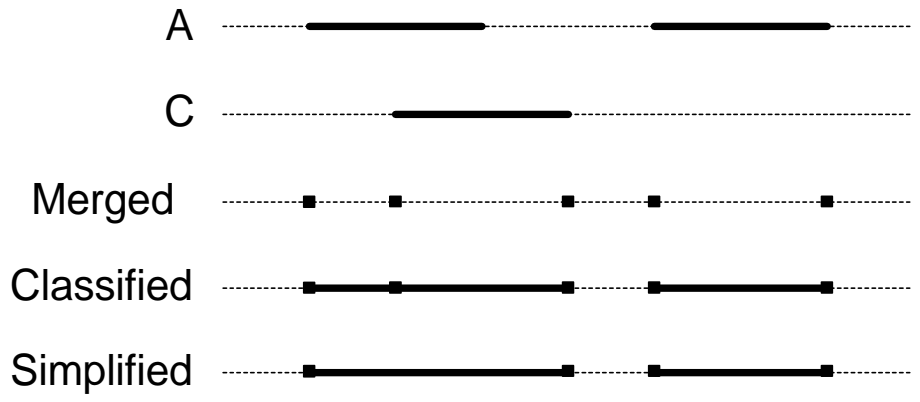


and we have the CSG combinations.
 We can do this step by step up a CSG tree.

Table for Boolean ops.

Operator	Left	Right	Composite
+	IN	IN	IN
	IN	OUT	IN
	OUT	IN	IN
	OUT	OUT	OUT
&	IN	IN	IN
	IN	OUT	OUT
	OUT	IN	OUT
	OUT	OUT	OUT
-	IN	IN	OUT
	IN	OUT	IN
	OUT	IN	
	OUT	OUT	

Illustration with three substeps. Merge, classify, simplify.



Ray Tracing Optimizations

(discussion mostly after Arvo and Kirk, A Survey of Ray Tracing Acceleration Techniques, in Glassner, *ed.*, An Introduction to Ray Tracing).

Whitted, “95% of time spent computing intersections”.

Exhaustive ray tracing – intersect each ray with all objects.

Arvo & Kirk present a taxonomy of acceleration:

1. Faster intersections
 - a. Faster ray/object intersections
 - b. Fewer of them
2. Fewer rays
3. Generalized rays

1. Faster intersections

Bounding volumes for objects of complex primitives.

2. Fewer intersections

Hierarchical bounding volumes. Still a topic of interest mostly for polygonal rendering.

Bound the ray length and a tree of volumes. Only follow branches that the ray intersects and if a ray hits, you can cut the length (don't care about further intersections).

A big topic is what the bounding primitives should be (boxes, spheres?) and how they should be aligned, axis aligned or oriented to the objects.

DRAW bounding sphere, axis-aligned box, and oriented box.

Construct manually, or automatically. See Stefan Gottschalk publications and thesis on oriented bounding boxes and

Spatial Subdivision.

Instead of building a hierarchy bottom-up, divide space top-down.

The former *selects volumes based on sets of objects.*

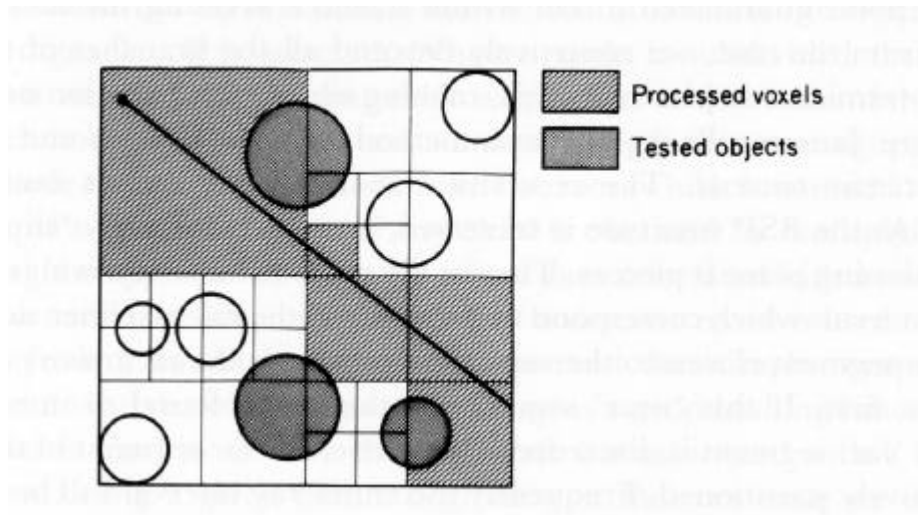
This one *selects objects based on volumes.*

Octrees

Built so that a list of objects intersecting (or contained in) a node is kept at each node.

Test for intersection with faces and containment.

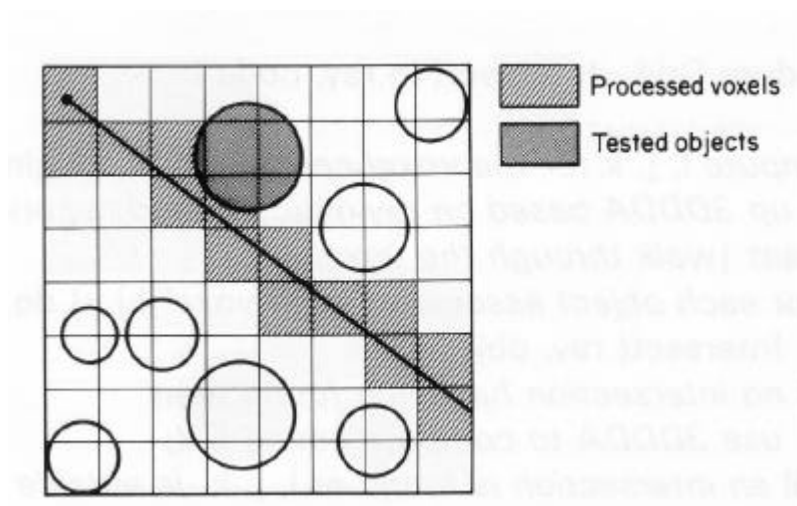
Follow ray through octrees, node by node, and only test with objects which are in nodes which ray traverses.



Non-uniform subdivision.

BSP trees. **ASK.** Axis aligned planes instead of Henry's arbitrarily planes for visibility determination. Rays are cut into segments by crossings of partitioning planes and if there's a hit the farther ones are not searched.

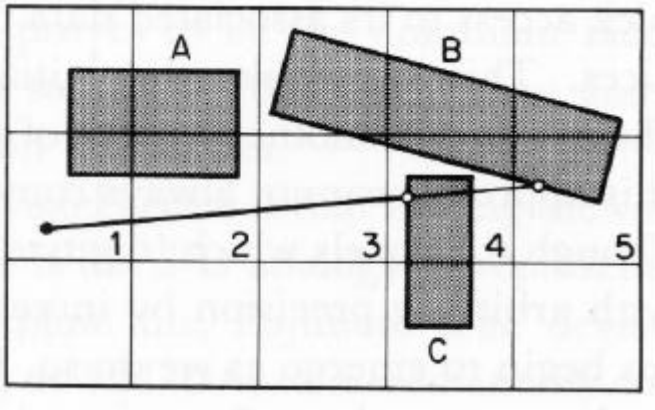
Also uniform subdivision. Not adapted to objects, but may be easier. Problem is if primitives are sparsely distributed.



Watch out for

Repeated tests with same object. Can keep intersection info with object.

Missing an object



Directional Techniques (do this?)

Idea is to use a cube around coordinate origin as a parameter space for direction of rays.

Light buffer – do this for each point light to object point. Keeps list of objects/primitives in boxes. It's a shadow test. Can do preprocessing, such as backface culling.

3. Fewer rays

We've talked some about sampling.

Adaptive tree depth control – maybe you want to use.

4. Generalized rays

This traces volumes instead of rays: cones, beams, pencils. Seems awfully complicated.

Beams are polygonal and intersect polygonal objects only. As a polygon is intersected, parts of the end of the beam are clipped away.

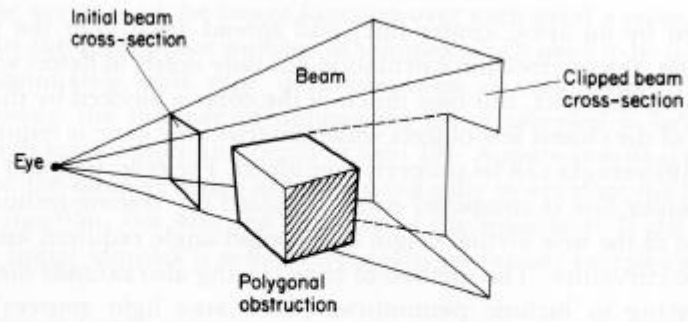


Fig. 26. A polygonal obstruction is clipped out of the cross section of a beam. This operation can quickly lead to cross sections which are non-simple polygons (e.g. disconnected with holes).

Parallelization-

Very parallelizable, if you can replicate database.

Recent papers from Utah. Parallel on SGI.

Fairly brute force. Groups of eye rays given to processors as tasks.

Since you can't predict the amount of work required for each group, they use a work queue and each processor takes items from queue. To reduce load imbalance and idle time at end of frame, they have different task granularities and start out with larger ones.

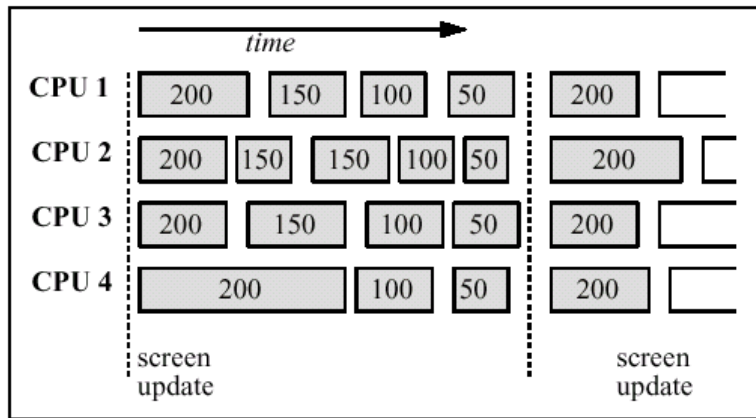
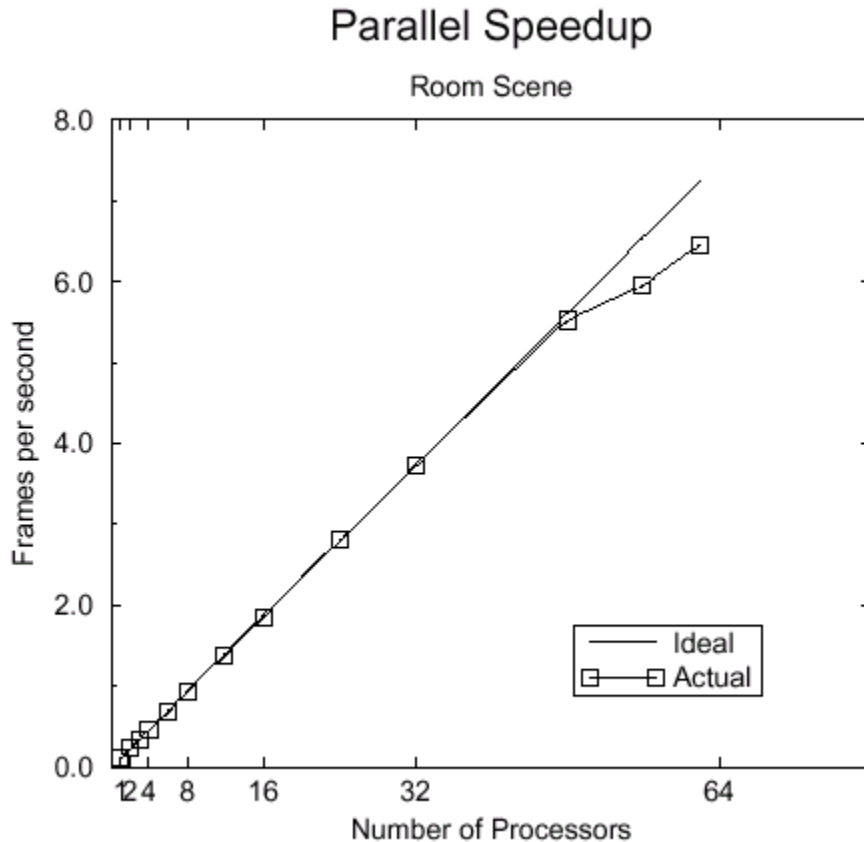


Figure 3: *Operation of ray tracer in synchronous mode. Numbers in boxes represent number of pixels in a block being processed. All pixels are traced before the screen swaps buffers.*

Note that size of tasks (granularity) directly affects load balancing, but small tasks incur too much overhead.

Reasonable speedup.

Load balancing and synchronization time limits final speedup.



Frameless Rendering

Don't update all pixels at once. They update a pseudo-random set every frame. Screen updated constantly. New pixels are blended. They can use 4 jittered samples and update only some of the samples.

Frameless rendering not as useful for conventional rendering because coherence is what makes it fast!

Explain progressive refinement.

Ambient approximation

Instead of a constant ambient term, they use one that is directional. This helps make objects lit by ambient appear less flat.

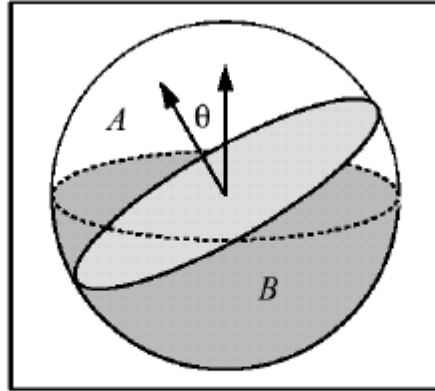


Figure 8: *A surface is illuminated by a hemisphere with colors A and B.*

This is supposed to represent indirect, so sphere away from light is brighter than that toward light. Heuristic.

They use other hacks, like softening the edge of shadows, to avoid casting more rays.

Special-purpose machines-

Ray casting machine from Duke.

ARC machine from England.