Ray Tracing Acceleration

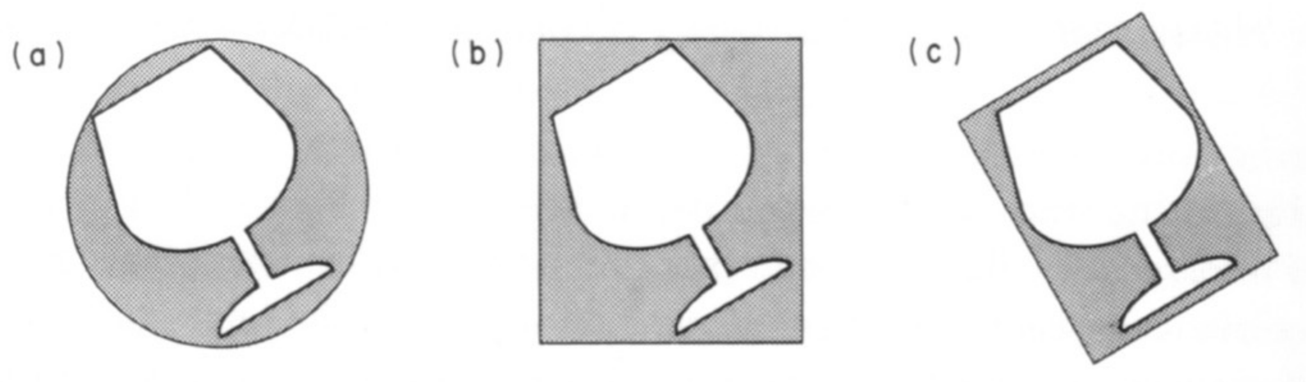
CS 4620 Lecture 9

Ray tracing acceleration

- Ray tracing is slow. This is bad!
  - Ray tracers spend most of their time in ray-surface intersection methods
- Ways to improve speed
  - Make intersection methods more efficient
    - Yes, good idea. But only gets you so far
  - Call intersection methods fewer times
    - Intersecting every ray with every object is wasteful
    - Basic strategy: efficiently find big chunks of geometry that definitely do not intersect a ray

Bounding volumes

- Quick way to avoid intersections: bound object with a simple volume
  - Object is fully contained in the volume
  - If it doesn't hit the volume, it doesn't hit the object
  - So test bvol first, then test object if it hits



[Glassner 89, Fig 4.5]

Bounding volumes

- Cost: more for hits and near misses, less for far misses
- Worth doing? It depends:
  - Cost of bvol intersection test should be small
    - Therefore use simple shapes (spheres, boxes, …)
  - Cost of object intersect test should be large
    - Bvols most useful for complex objects
  - Tightness of fit should be good
    - Loose fit leads to extra object intersections
    - Tradeoff between tightness and bvol intersection cost

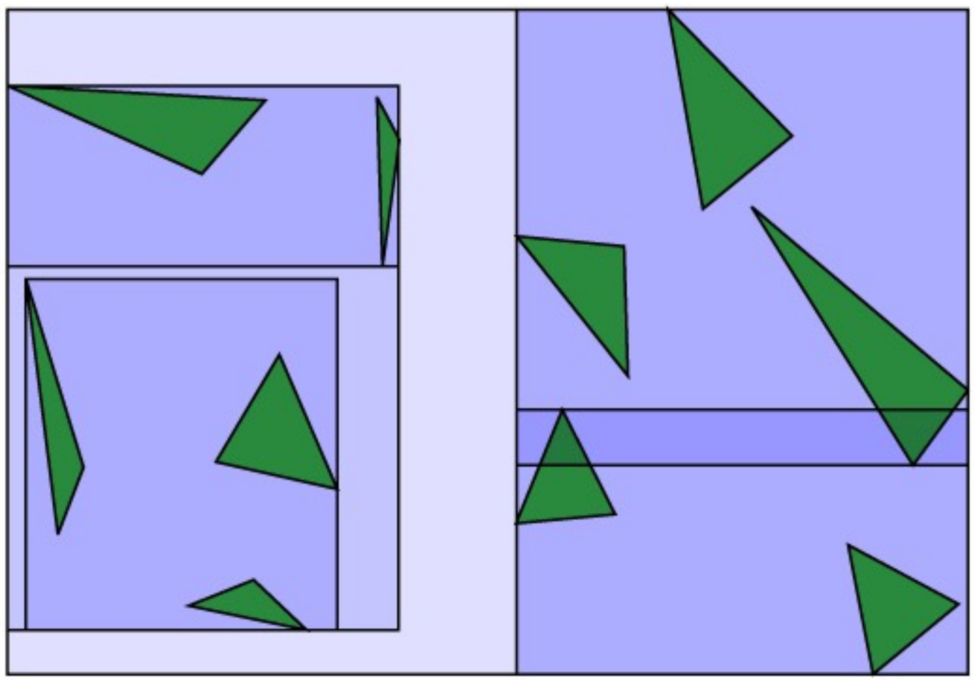
If it’s worth doing, it’s worth doing hierarchically!

- Bvols around objects may help
- Bvols around groups of objects will help
- Bvols around parts of complex objects will help
- Leads to the idea of using bounding volumes all the way from the whole scene down to groups of a few objects

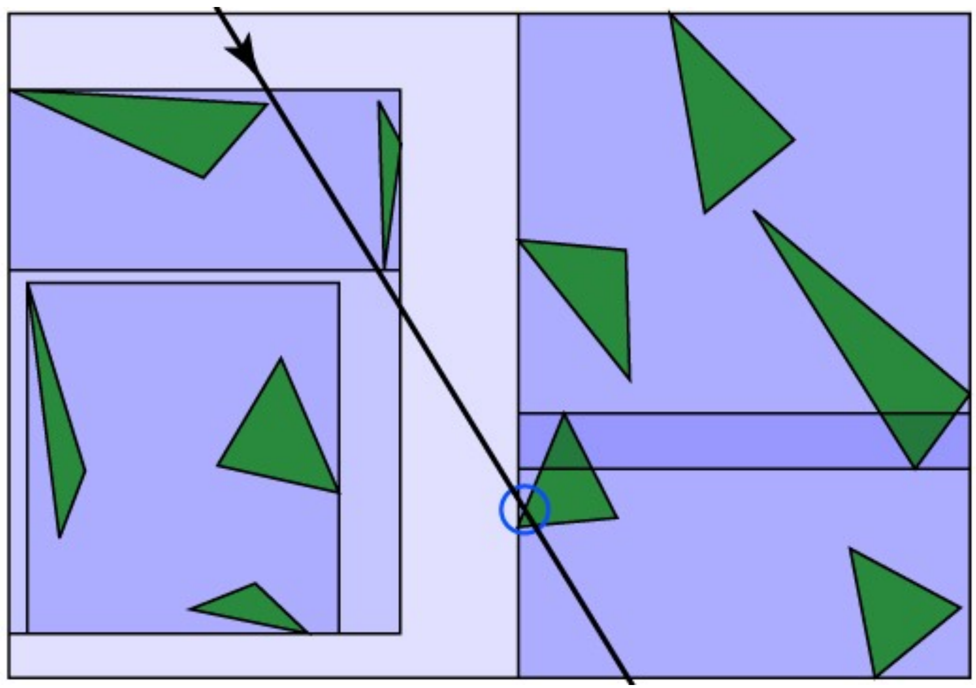
Implementing a bvol hierarchy

- A bounding volume hierarchy is a tree of boxes
  – each bounding box contains all children
  – ray misses parent implies ray misses all children
- Leaf nodes contain surfaces
  – again the bounding box contains all geometry in that node
  – if ray hits leaf node box, then we finally test the surfaces
- Replace the intersection loop over all objects in the scene with a partial tree traversal
  – test node first; test all children only ray hits parent
- Usually we use binary trees (each non-leaf box has exactly two contained boxes)

BVH construction example



BVH ray-tracing example



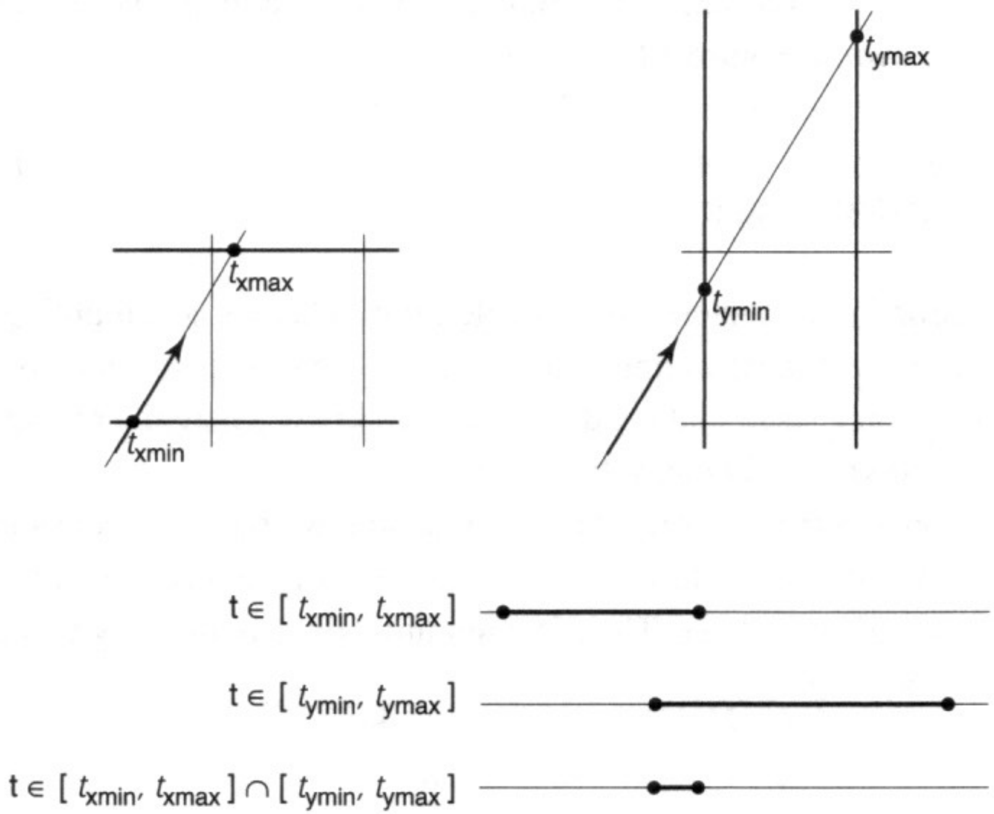
Choice of bounding volumes

- Spheres -- easy to intersect, not always so tight
- Axis-aligned bounding boxes (AABBs) -- easy to intersect, often tighter (esp. for axis-aligned models)
- Oriented bounding boxes (OBBs) -- easy to intersect (but cost of transformation), tighter for arbitrary objects
- Computing the bvols
  - For primitives -- generally pretty easy
  - For groups -- not so easy for OBBs (to do well)
  - For transformed surfaces -- not so easy for spheres

Axis aligned bounding boxes

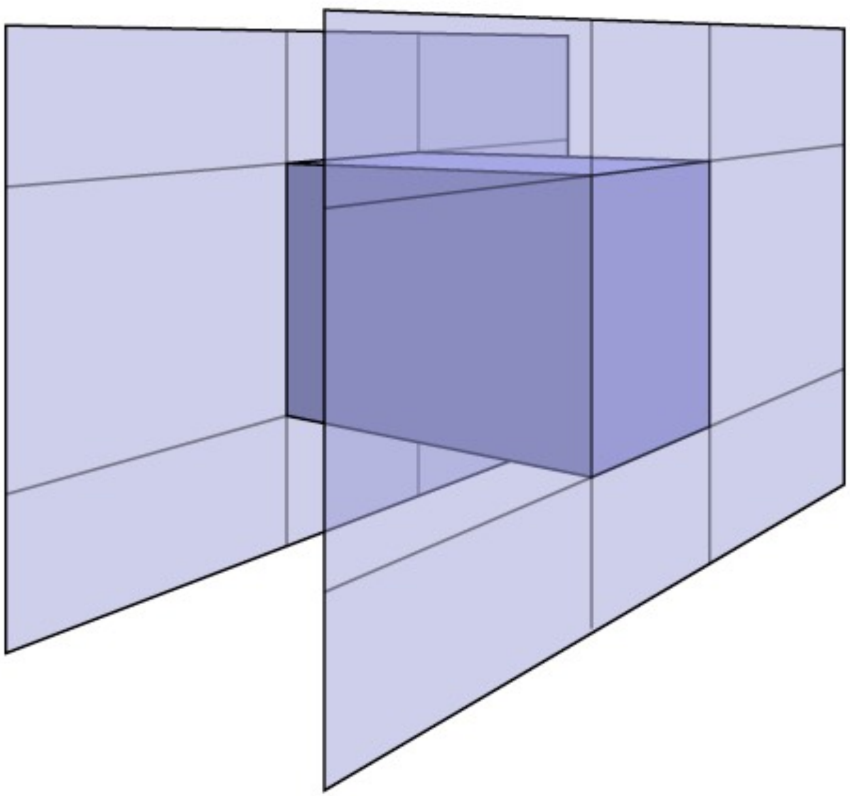
- Probably easiest to implement
- Computing for (axis-aligned) primitives
  - Cube: duh!
  - Sphere, cylinder, etc.: pretty obvious
  - Groups or meshes: min/max of component parts
- AABBs for transformed surface
  - Easy to do conservatively: bbox of the 8 corners of the bbox of the untransformed surface
- How to intersect them
  - Treat them as an intersection of slabs (see textbook)

Intersecting boxes



Ray-box intersection

- Could intersect with 6 faces individually
- Better way: box is the intersection of 3 slabs

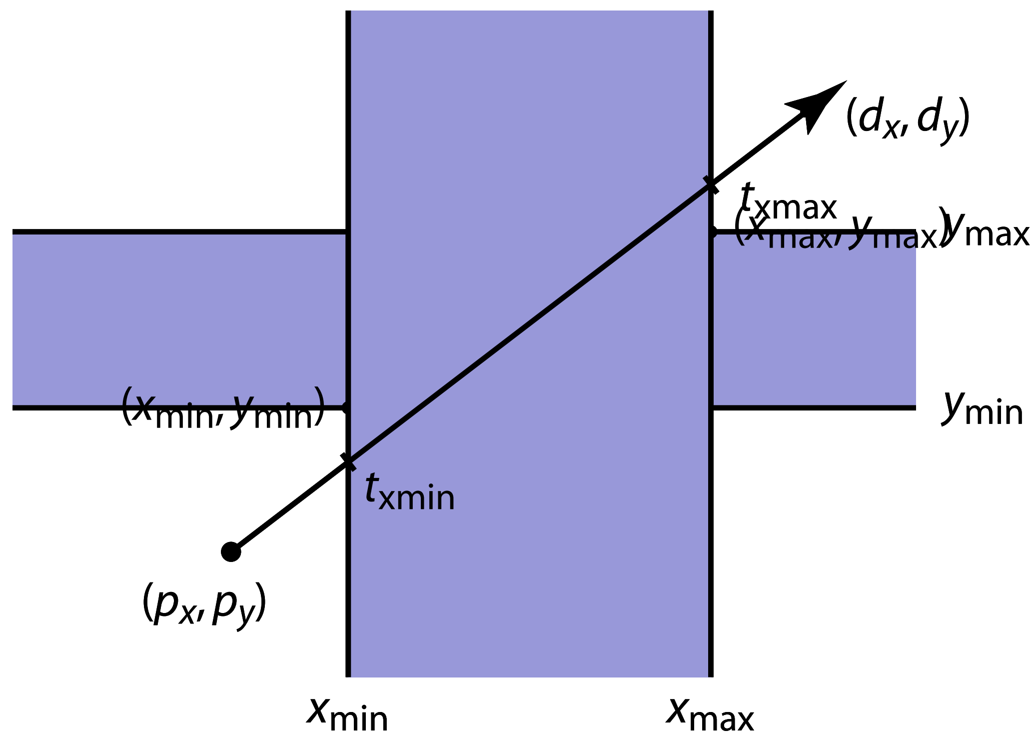


Ray-slab intersection

- 2D example
- 3D is the same!

$$p_x + t_{x\mathrm{min}}\, d_x = x_{\mathrm{min}}$$
$$t_{x\mathrm{min}} = (x_{\mathrm{min}} - p_x)/d_x$$

$$p_y + t_{y\mathrm{min}}\, d_y = y_{\mathrm{min}}$$
$$t_{y\mathrm{min}} = (y_{\mathrm{min}} - p_y)/d_y$$



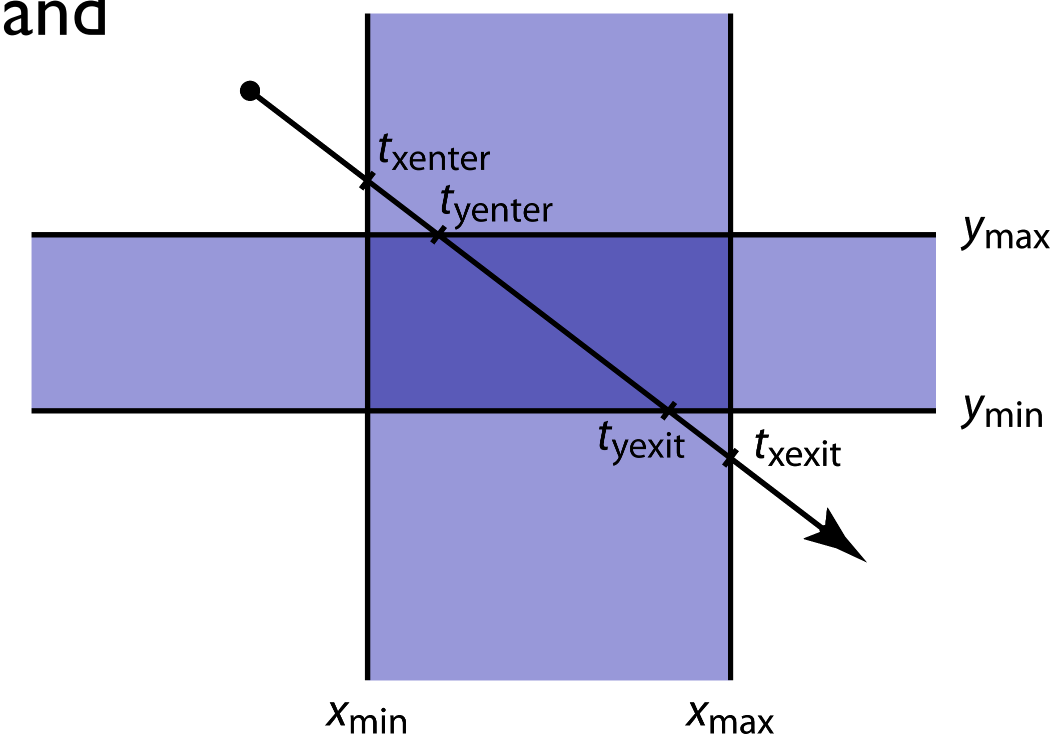
Intersecting intersections

- Each intersection is an interval
- Want last entry point and first exit point

$$t_{x\mathrm{enter}} = \min(t_{x\mathrm{min}}, t_{x\mathrm{max}})$$
$$t_{x\mathrm{exit}} = \max(t_{x\mathrm{min}}, t_{x\mathrm{max}})$$

$$t_{y\mathrm{enter}} = \min(t_{y\mathrm{min}}, t_{y\mathrm{max}})$$
$$t_{y\mathrm{exit}} = \max(t_{y\mathrm{min}}, t_{y\mathrm{max}})$$

$$t_{\mathrm{enter}} = \max(t_{x\mathrm{enter}}, t_{y\mathrm{enter}})$$
$$t_{\mathrm{exit}} = \min(t_{x\mathrm{exit}}, t_{y\mathrm{exit}})$$



Building a hierarchy

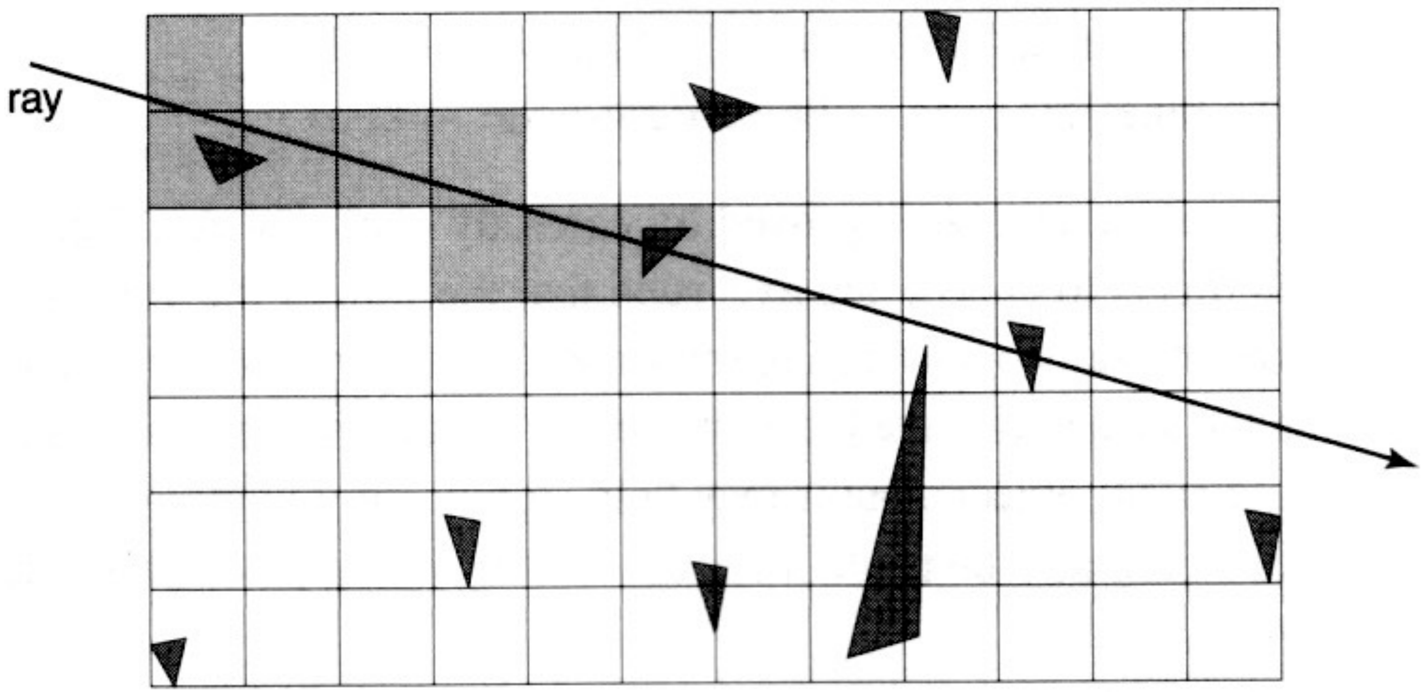
- Usually do it top-down
- Make bbox for whole scene, then split into (maybe 2) parts
  – Recurse on parts
  – Stop when there are just a few objects in your box

Building a hierarchy

- How to partition?
  - Ideal: clusters
  - Practical: partition along axis
    - Center partition
      - Less expensive, simpler
      - Unbalanced tree (but may sometimes be better)
    - Median partition
      - More expensive
      - More balanced tree
    - Surface area heuristic
      - Model expected cost of ray intersection
      - Generally produces best-performing trees

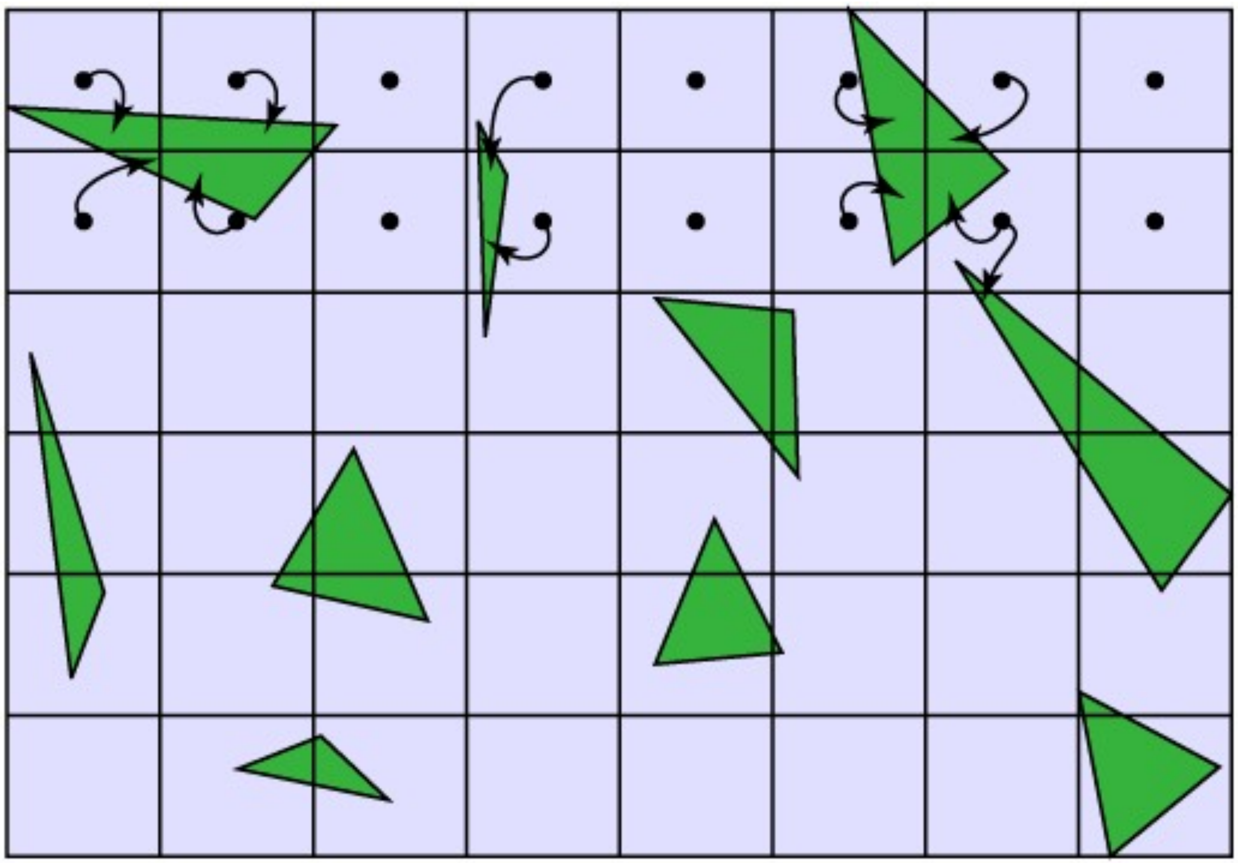
Regular space subdivision

- An entirely different approach: uniform grid of cells

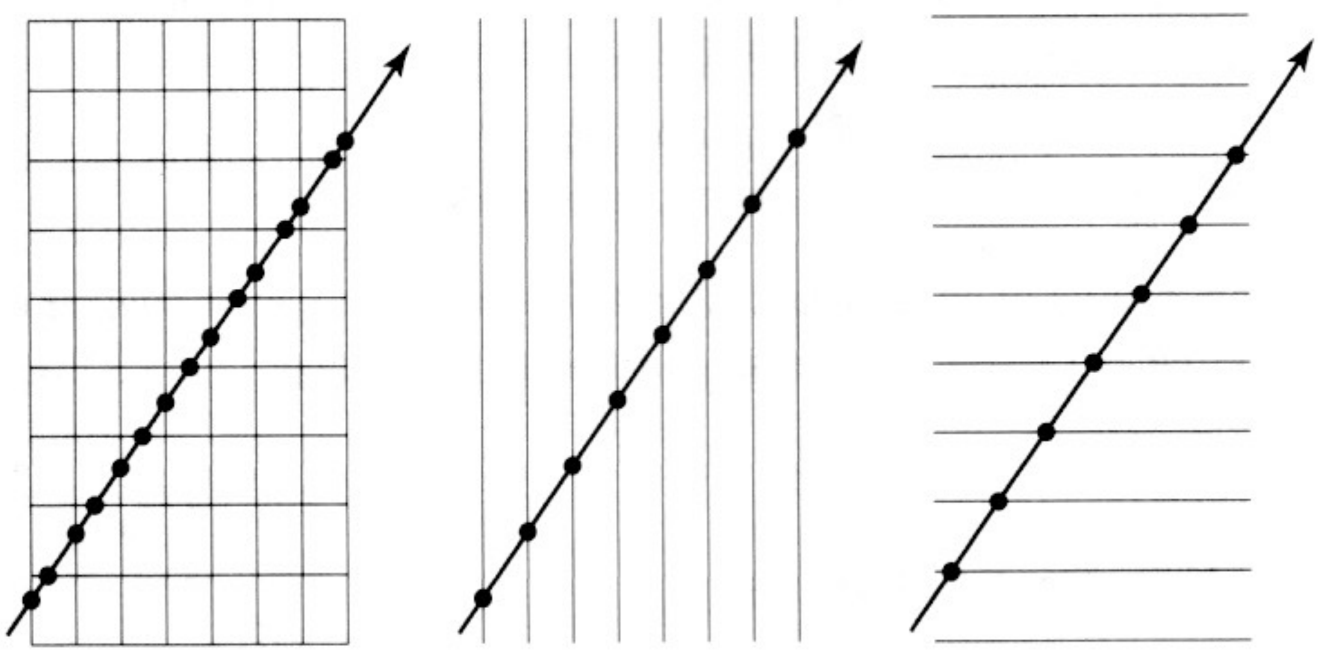


Regular grid example

- Grid divides space, not objects

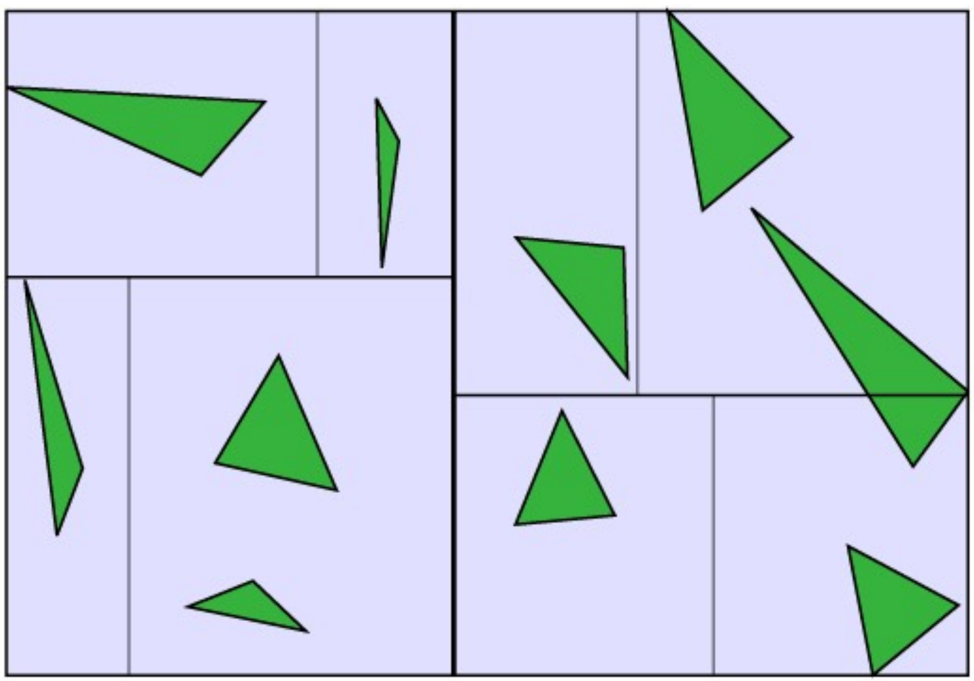


Traversing a regular grid



Non-regular space subdivision

- *k*-d Tree
  - subdivides space, like grid
  - adaptive, like BVH



Implementing acceleration structures

- Conceptually, structure traversal replaces main ray intersection loop
  - could literally replace code in Scene
- Better engineering decision to separate the acceleration structure
  - plug and play different acceleration structures
  - keep representation of scene itself simple
- Acceleration engine has two fundamental methods:
  - build from list of surfaces
  - intersect with ray

Ray tracing acceleration in practice

- High RT performance is a major engineering task
  - becoming more common to rely on external libraries
  - Intel Embree: CPU library optimized for Intel processors
  - Nvidia RTX: hardware accelerated ray tracing for latest generation GPUs
- Fastest current systems:
  - CPU: tens to hundreds of megarays / sec
  - GPU: a few gigarays / sec
  - 1 gigaray / 60 frames / 1M pixels $\approx$ 16 rays/pixel/frame
  - not a ton of rays but with judicious use many nice reflection/ shadow effects can be rendered in real time


tineKhome by Jakub Cech| www.jakubcech.net
rendered with Corona Renderer using the Embree Ray Tracing Kernels

Minecraft RTX tech demo

