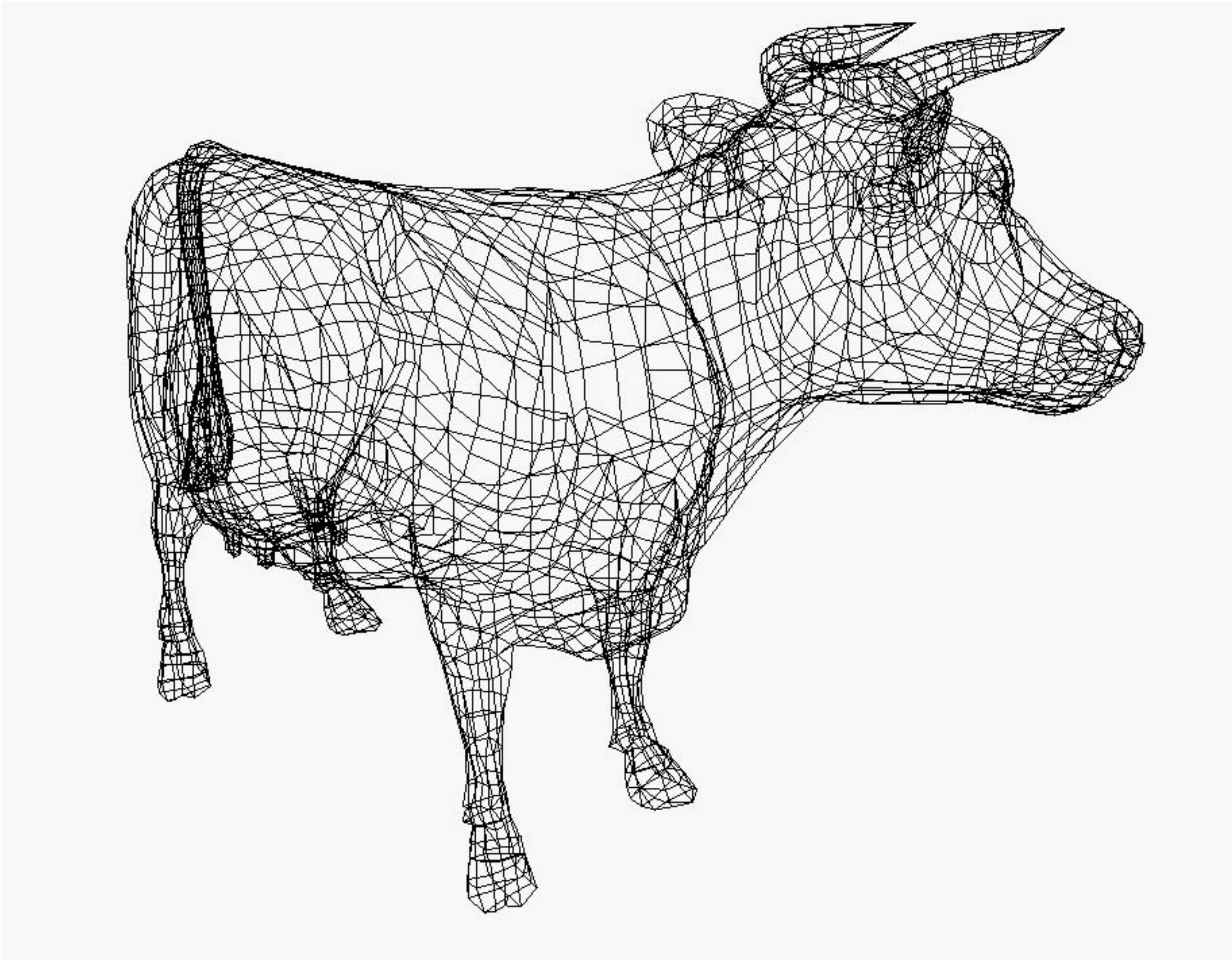
Accelerating Ray Tracing Polygons

UCL

Acceleration

- In ray tracing - > 90% of cost is in ray-polygon intersections
  - As described so far, O(n) where n is number of polygons
- As in general algorithms in CS, optimization can be done by appropriate & efficient representations
- What can we exploit?

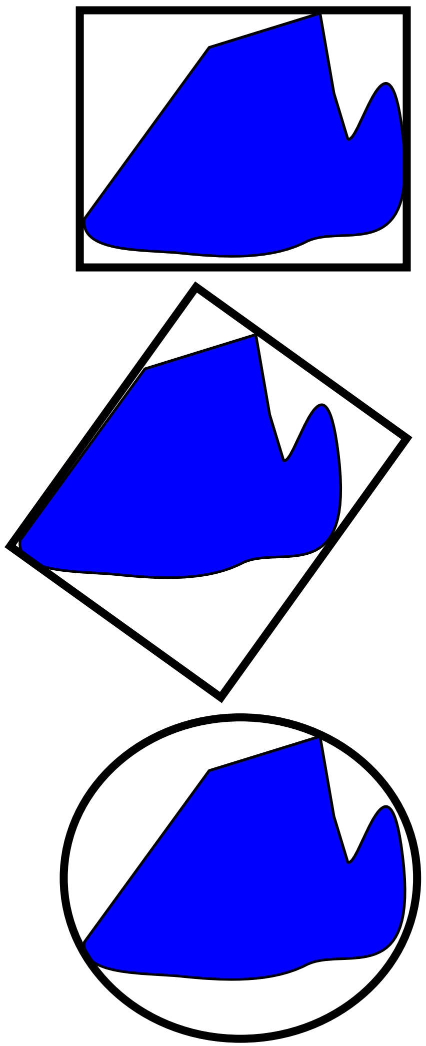


Overview

- Many potential data structures
- Choose three common ones
  - Bounding volumes
  - Hierarchical bounding volumes
  - kD-Tree
- There are other common data structures such as octrees and quadtrees.
- Note that this is strongly related to the issue of hashing & indexing in tables.

Bounding Volume

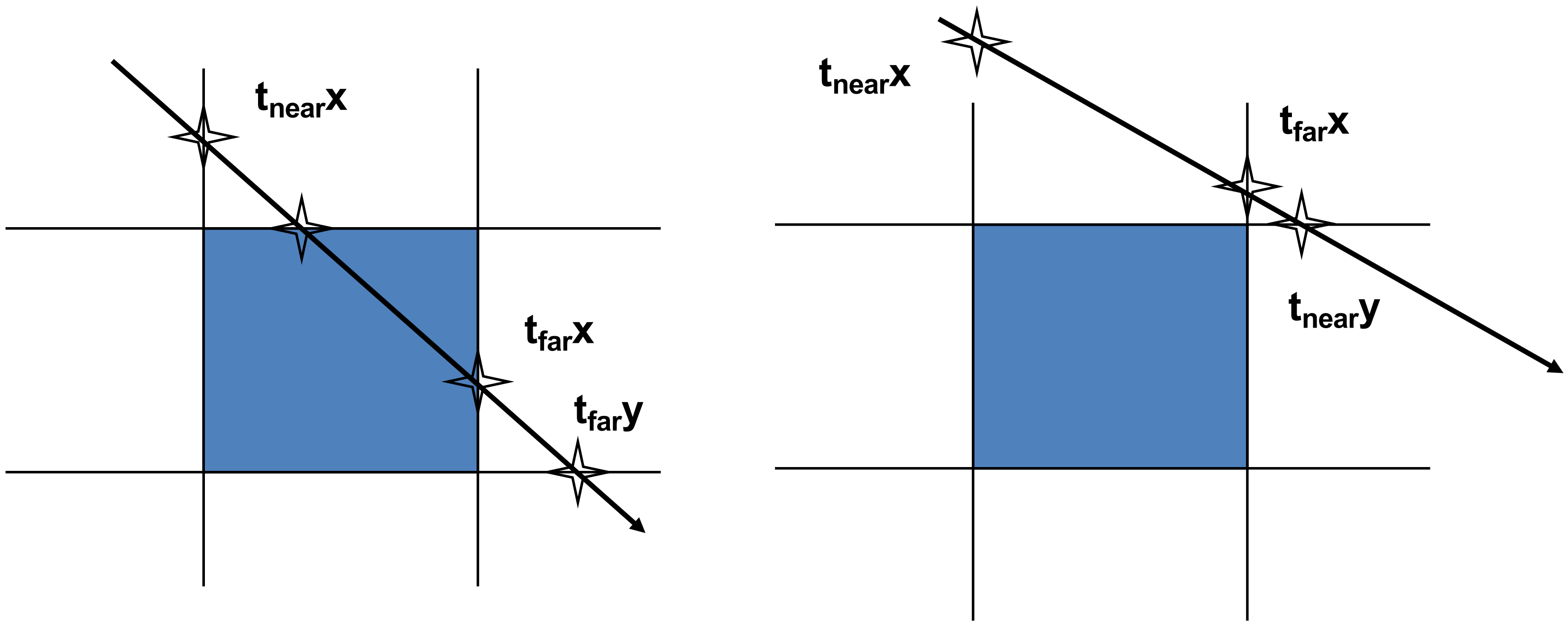
- Find a tight bounding volume and use it for a **reject test**
- If hit volume then test full object



Fast BV Tests (AABB)

- Box-Ray test (when box planes parallel to axes)
  - a box is three sets of parallel planes, each set orthogonal to the other two,
  - ray defined by $q(t) = q_0 + t.dq$
    - Calculate t_{near} for each of the three plane pairs
    - find max of the 3 t_{near}
    - Calculate t_{far} for each of the three plane pairs
    - find min of the 3 t_{far}
  - If max t_{near} is greater that min t_{far}, then the box is not intersected

Fast BV Tests (AABB, 2D case)



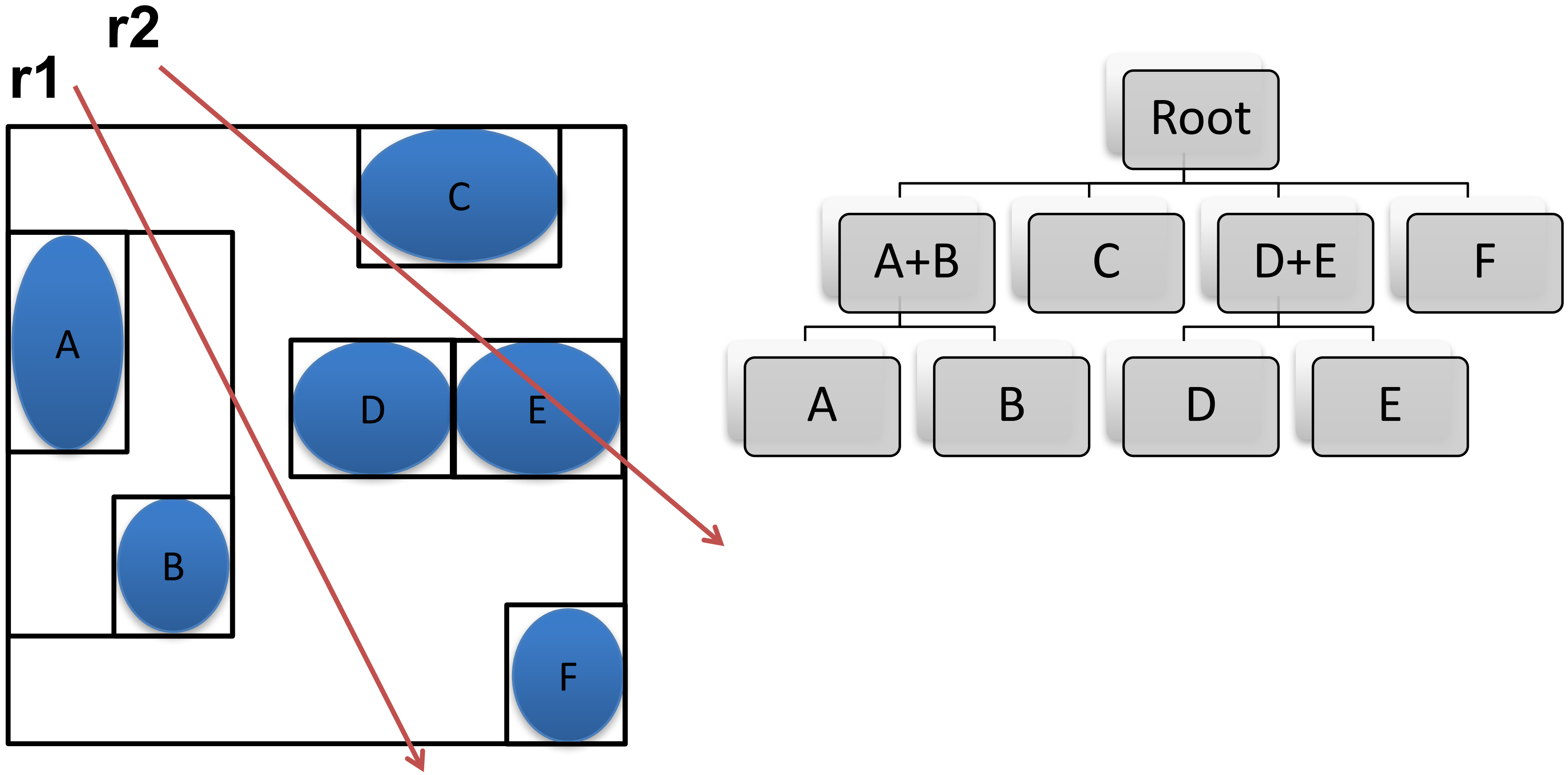
Pros and Cons

- Utility is a trade-off between the simplicity of the bounding volume and the “void” space
- Pros: a bounding volume can be extremely efficient when it is unlikely that the volume will be “hit” (for ray tracing, visibility, etc.)
- Cons: the “void space” may be very large, leading to many redundant expensive tests.

- Solution: better intermediate representations.

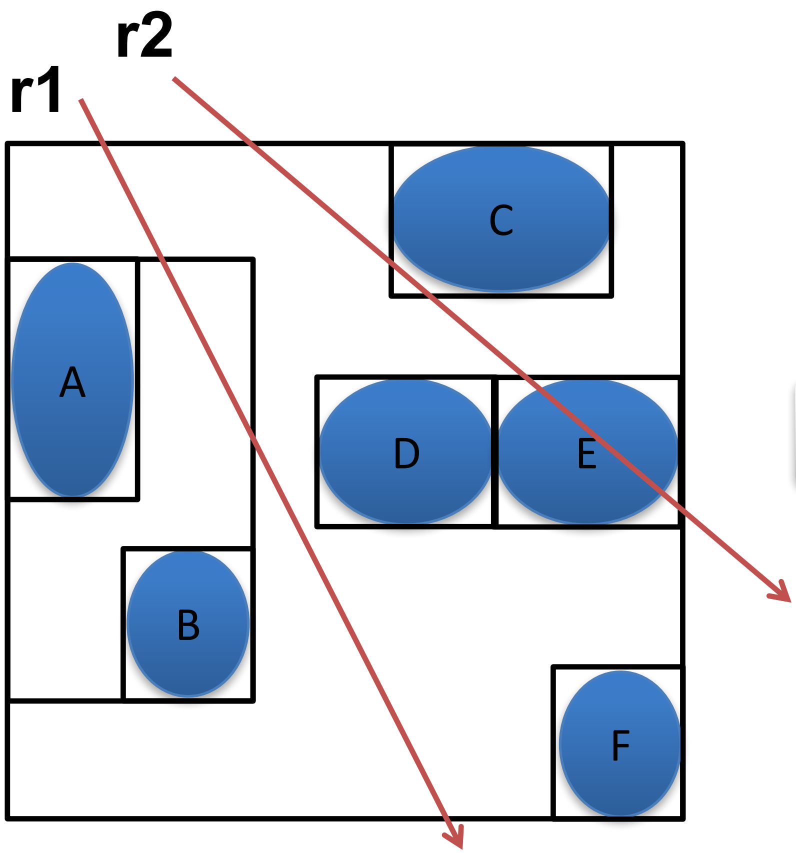
Bounding Volume Hierarchy

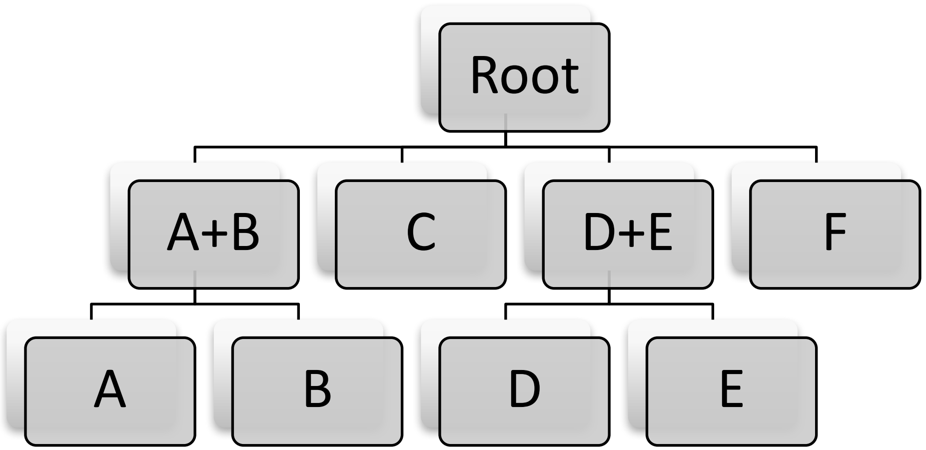
- Organise a hierarchy of bounding volumes
  - Bounding volumes of bounding volumes



Bounding Volume Hierarchy

- Organise a hierarchy of bounding volumes
  - Bounding volumes of bounding volumes





r_1 tests with A+B, then A and B

r_2 tests with C and D+E, then D and E

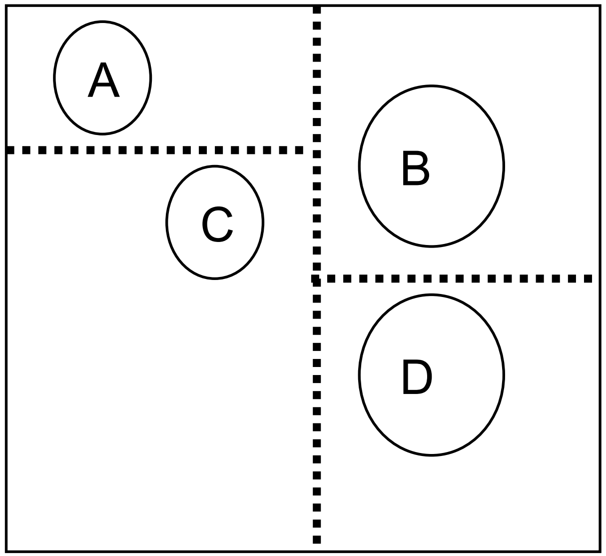
Bounding Volume Hierarchy

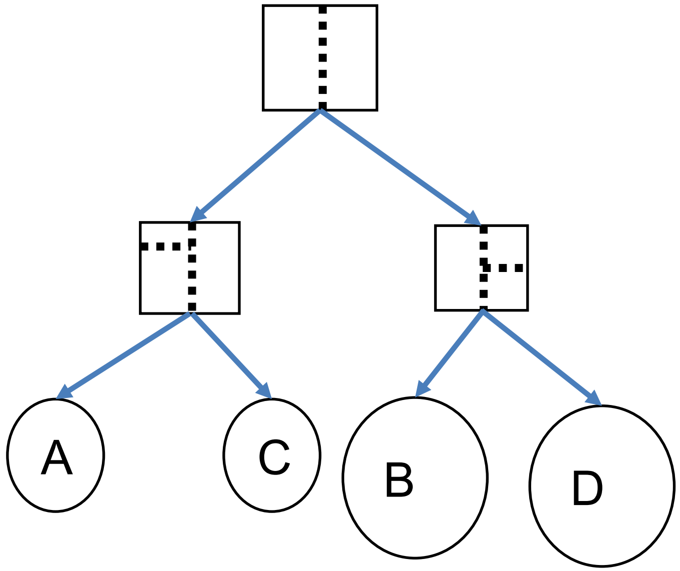
- Pros:
  - Very good adaptivity
  - Efficient traversal O(log N)

- Cons
  - How to arrange BVs?

kD-Tree

- One of a class of spatial data structure that partition space
- Uses horizontal and vertical splits



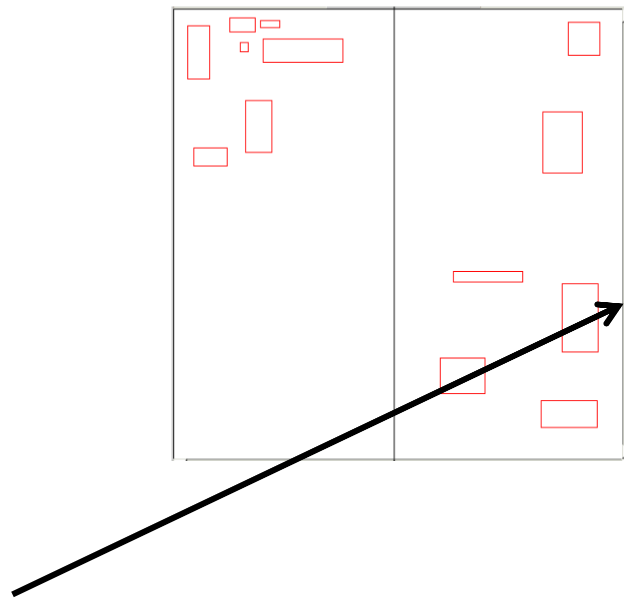


Traversing a kD-tree

- kdTree-RayIntersect(Ray, Node)
  - If node is a leaf, intersect ray with objects
  - Else
    - Clip ray to near side of the split (RayNear)
    - kdTree-RayIntersect(RayNear, Node->near)
    - If (no intersection)
      - Clip ray to far side of the plane (RayFar)
      - kdTree-RayIntersect(RayFar, Node->far)

Building good kD-trees

- What split do we really want?
  - Main Idea: The one that makes ray tracing cheap
  - Write down an expression of cost and minimize it
  - *Cost Optimization*

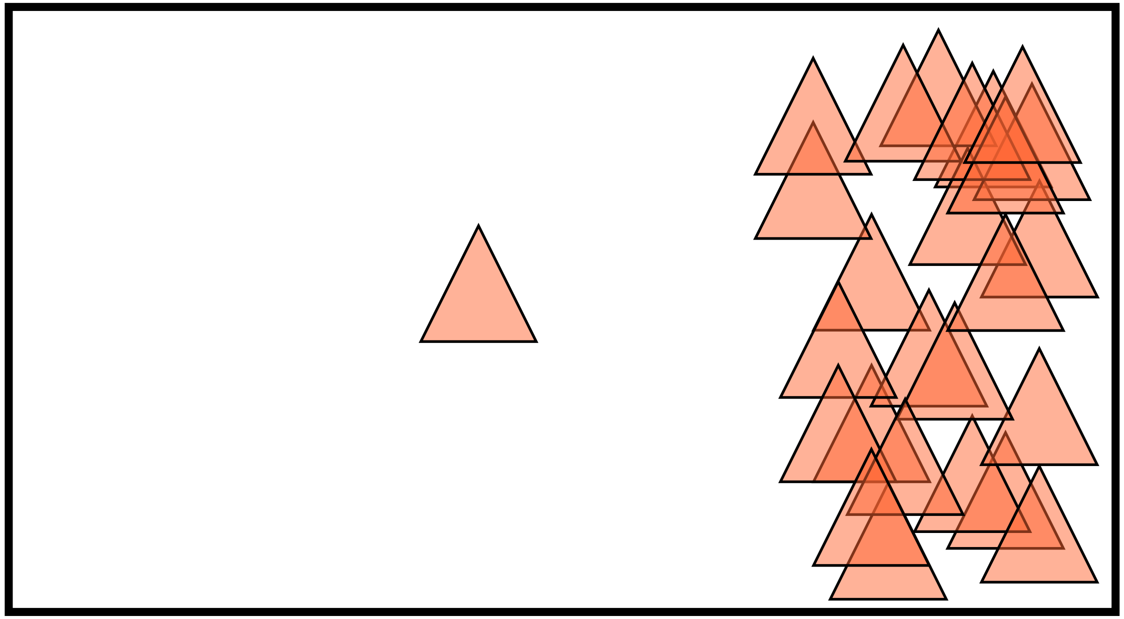


- What is the cost of tracing a ray through a cell?

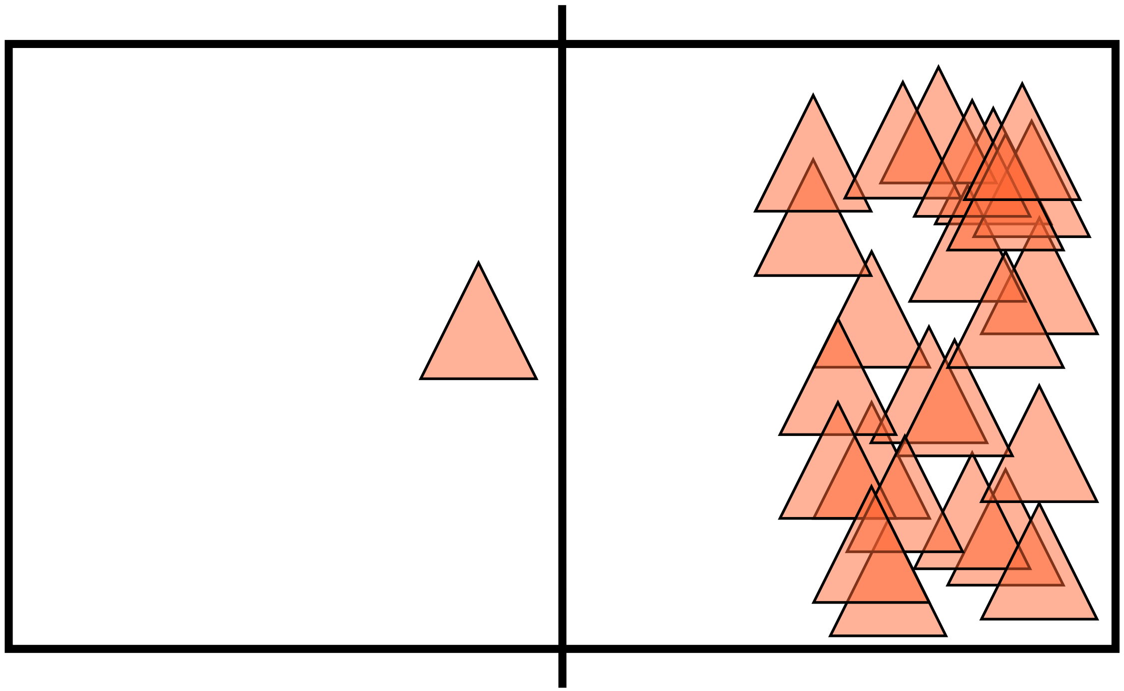
Cost(cell) = C_trav + Prob(hit L) * Cost(L)

+ Prob(hit R) * Cost(R)

Splitting with Cost in Mind

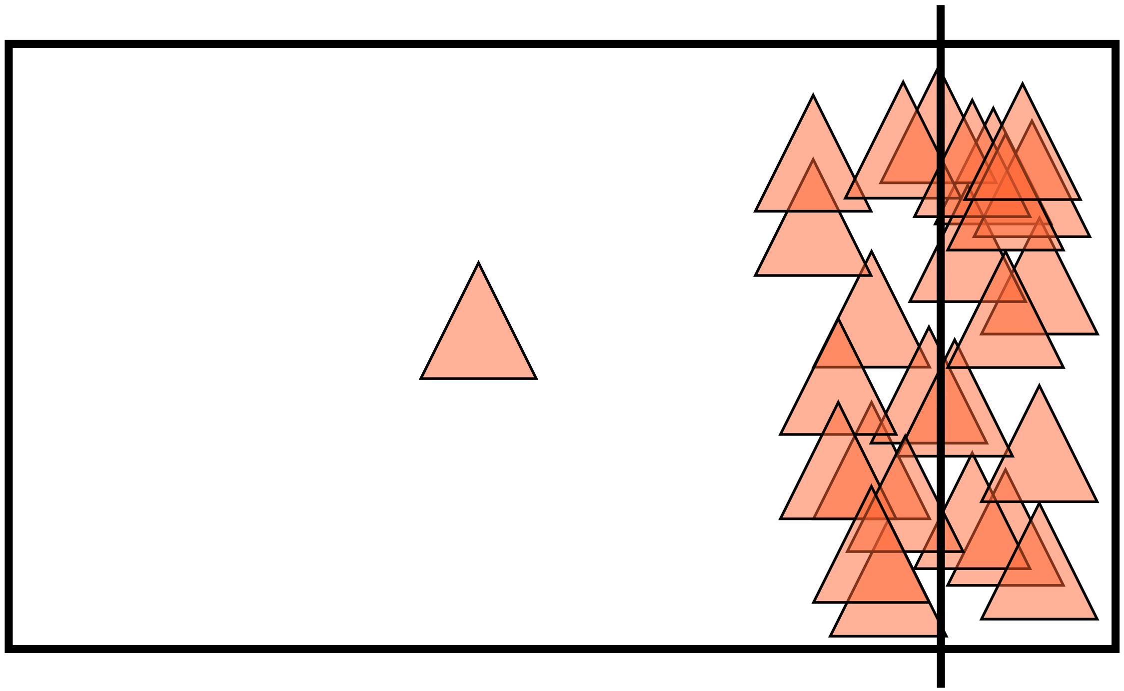


Split in the middle



- Makes the L & R probabilities equal
- Pays no attention to the L & R costs

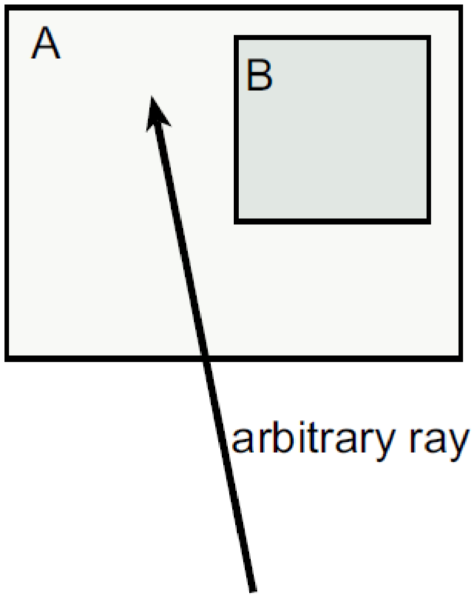
Split at the Median



- Makes the L & R costs equal
- Pays no attention to the L & R probabilities

Surface Area and Rays

- Probability of a ray hitting an object that is completely inside a cell is:



$$Prob[hitB|hitA] = \frac{S_B}{S_A}$$

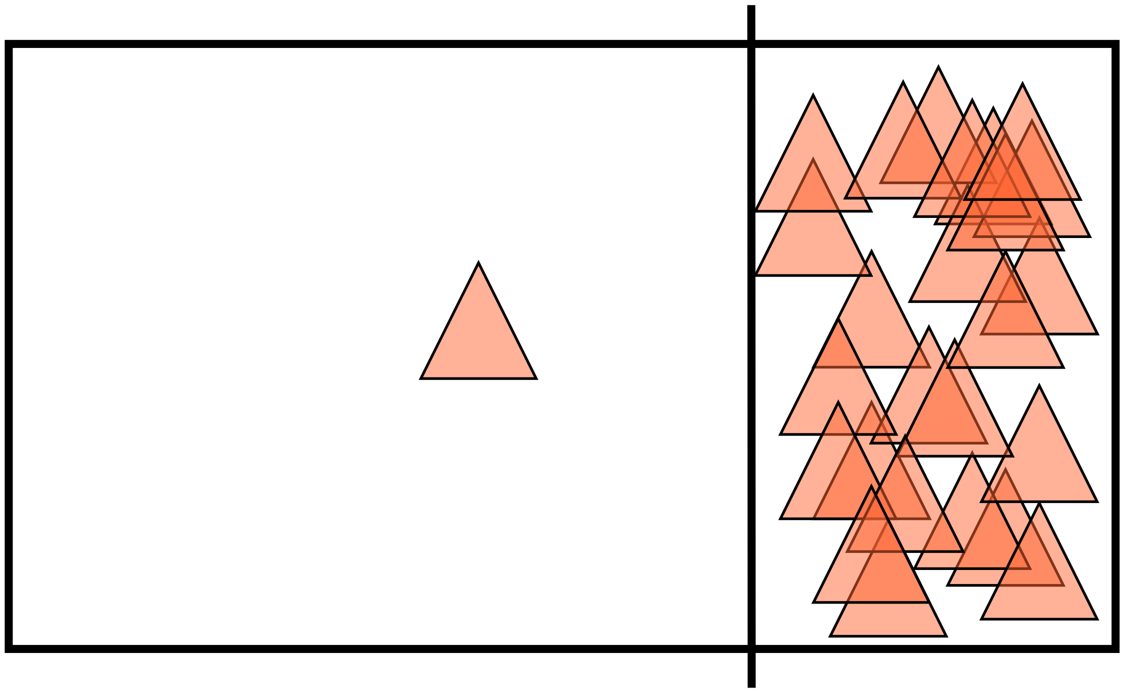
Surface Area Heuristic

- Need the probabilities
  - Turns out to be proportional to surface area
- Need the child cell costs
  - Simple triangle count works great (very rough approx.)

Cost(cell) = C_trav + Prob(hit L) * Cost(L) +
Prob(hit R) * Cost(R)

= C_trav + SA(L) * TriCount(L) +
SA(R) * TriCount(R)

Cost-Optimized Split



- Automatically and rapidly isolates complexity
- Produces large chunks of empty space

Recap

- Several techniques can be applied for accelerating the ray intersection tests with the scene
- Bounding volumes are a very obvious acceleration and almost always a good idea
- Bounding volume hierarchies are useful, but geometry is often irregularly spaced around the environment, so BVH is inefficient in empty space
- kD-Trees are one of a class of spatial data structure that balance precision in implementation with general utility. Very commonly used in practice