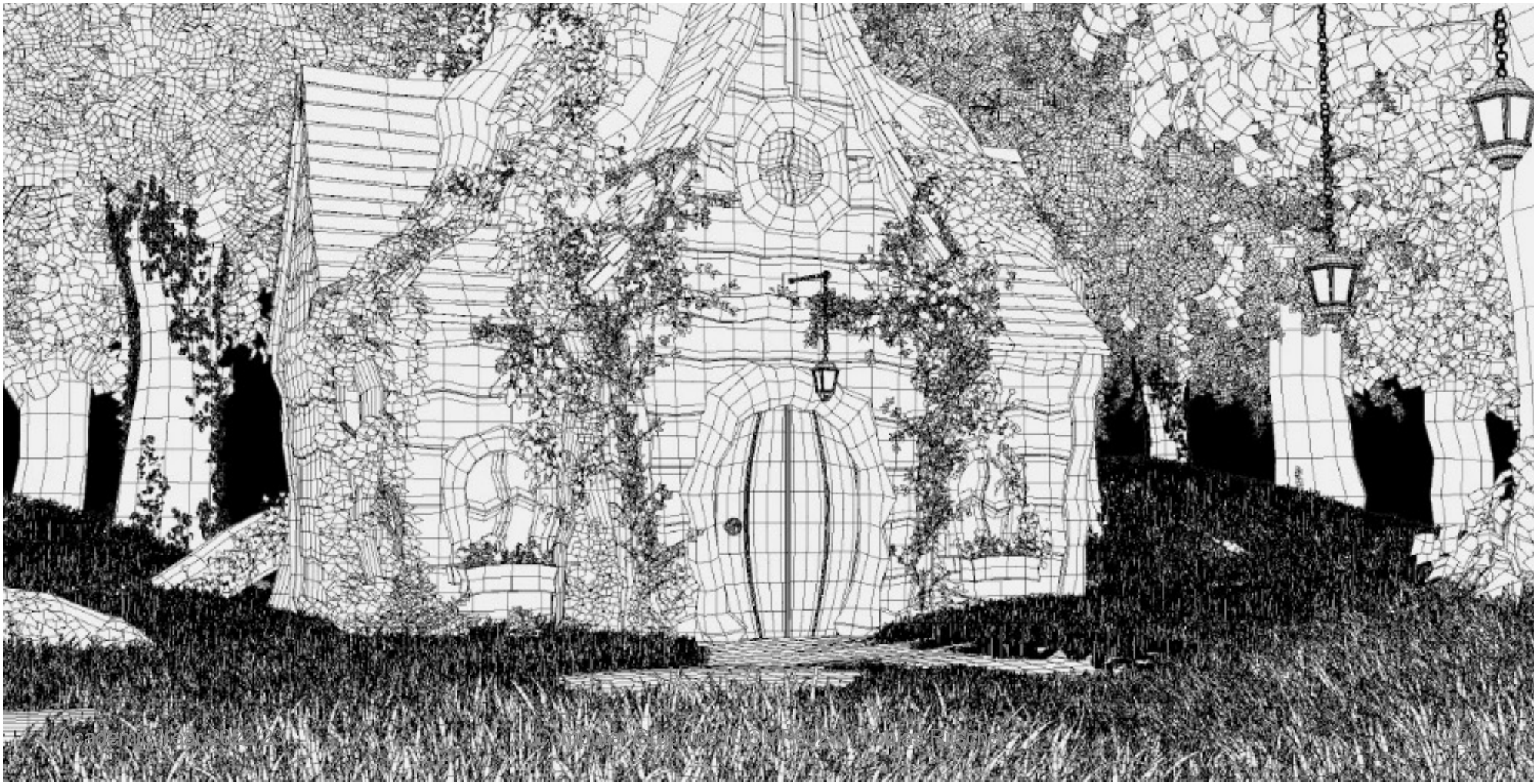
Computer Graphics (COMP0027) 2022/23

Bounding Volumes

Tobias Ritschel

UCL

Polygonal meshes



Trivial

- Simple
  - Loop over all rays
  - Loop over all polygons
  - Intersect
- Complexity
  - Quadratic in ray-poly
  - linear in polys
- Can we do better?

Acceleration

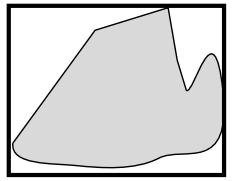
- In ray tracing 90% of cost is in ray-polygon intersections
- As described so far, O(n) where n is number of polygons
- As in general algorithms in CS, optimization can be done by appropriate & efficient representations
- What can we exploit?

Overview

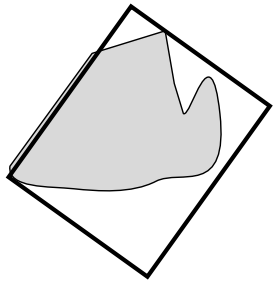
- Many potential data structures
- Choose three common ones
  - Bounding volumes
  - Hierarchical bounding volumes
  - kD-Tree
- There are others such as octrees and quad trees.
- Note, that this is related to issues of hashing & indexing in tables.

Bounding Volume

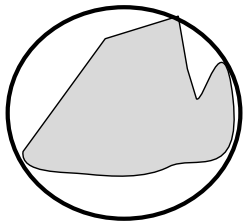
- Find a tight bounding volume and use it for a *reject test*
- If hit volume then test full object



Axis Aligned Bounding Box (AABB)



Bounding Box

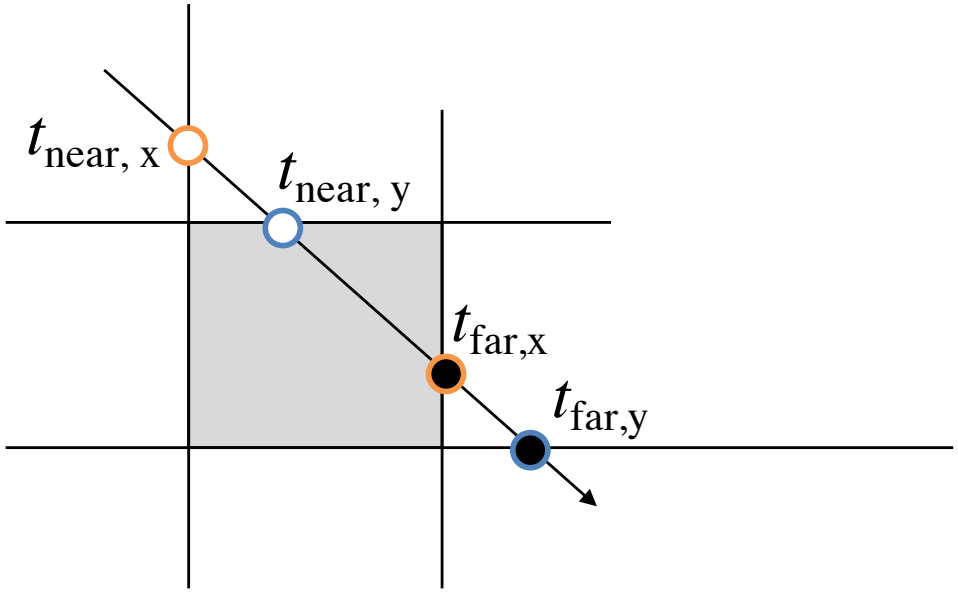


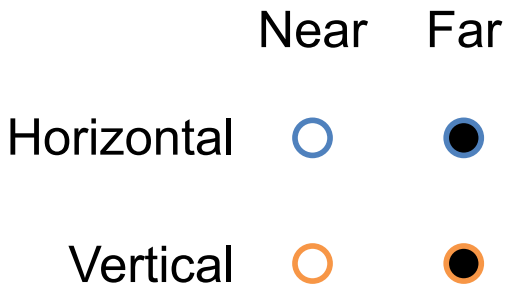
Bounding Sphere

Fast BV Tests (AABB)

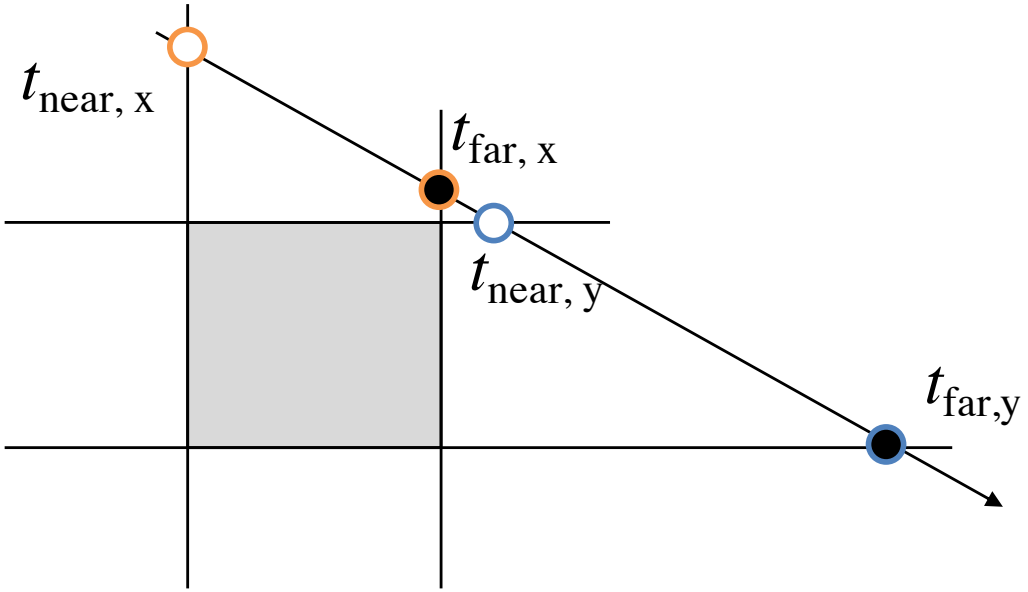
- Box-Ray test (when box planes parallel to axes)
  - A box is three pairs of parallel planes, each pair orthogonal to the other two
  - Ray defined by $\mathbf{r}(t) = \mathbf{p} + t\,\mathbf{d}$
    - Calculate t_{near} for each of the three axis
    - Find max of the three t_{near}
    - Calculate t_{far} for each of the three axis
    - Find min of the three t_{far}
  - If max t_{near} is greater that min t_{far}, then the box is not intersected

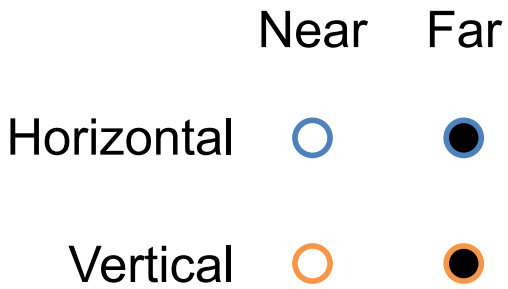
Fast BV Tests (AABB, 2D case)





Fast BV Tests (AABB, 2D case)





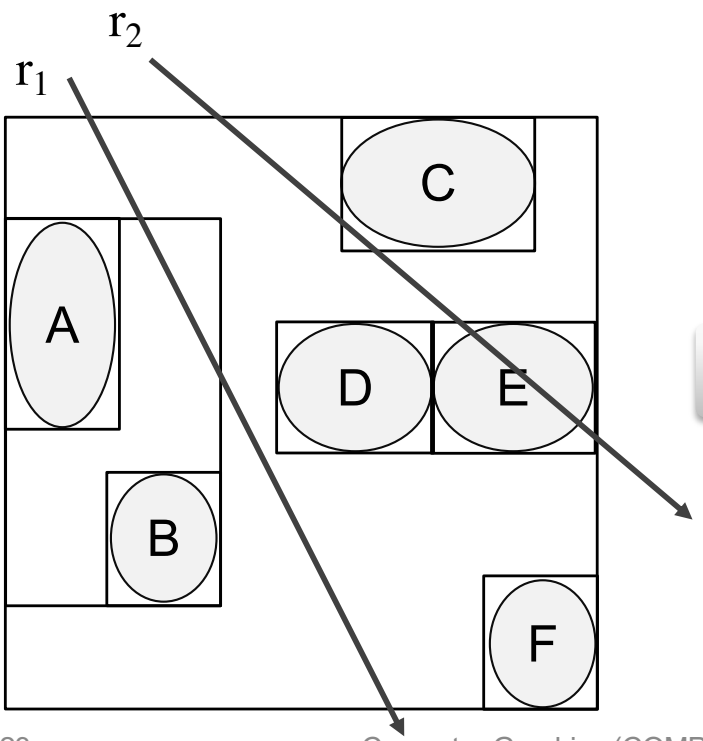
Pros and Cons

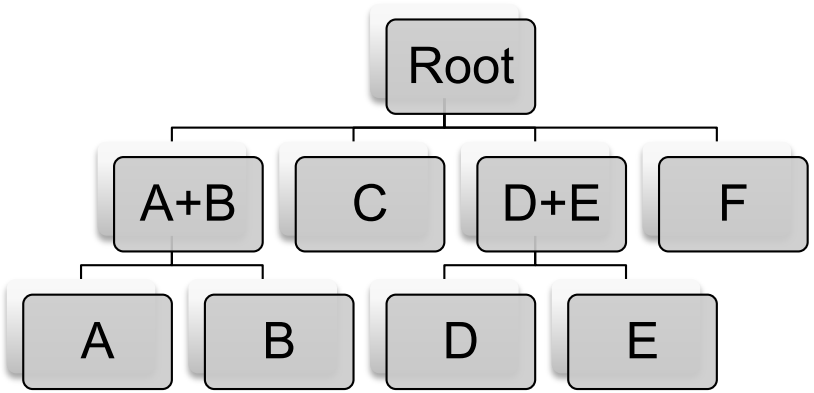
- Utility is a trade-off between the simplicity of the bounding volume and the “void” space
- Pros: a bounding volume can be extremely efficient when it is unlikely that the volume will be “hit” (for ray tracing, visibility, etc.)
- Cons: the “void space” may be very large, leading to many redundant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

- Solution: better intermediate representations.

Bounding Volume Hierarchy

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





r_1 tests with A+B, then A and B

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

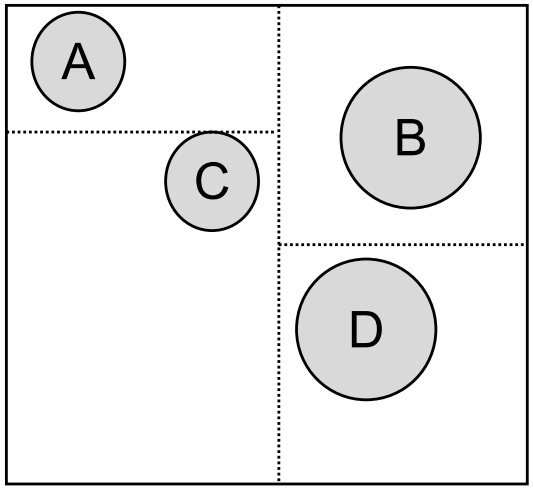
Bounding Volume Hierarchy

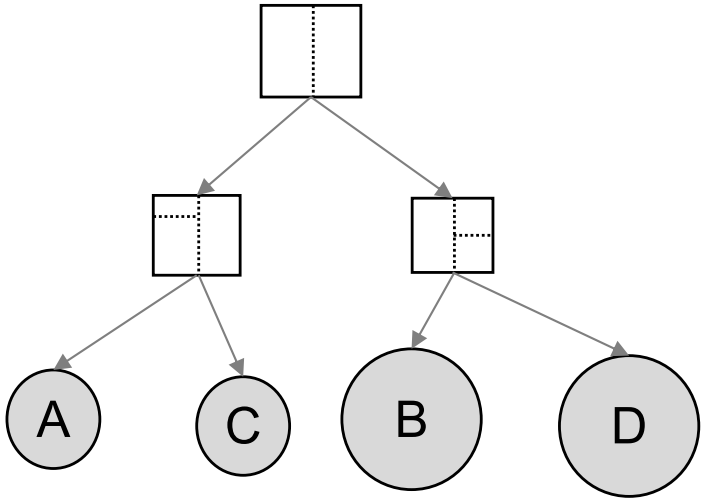
- Pros:
  - Very good adaptivity
  - Efficient traversal $\mathrm{O}(\log n)$

- Cons
  - How to arrange BVs?

kD-Tree

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

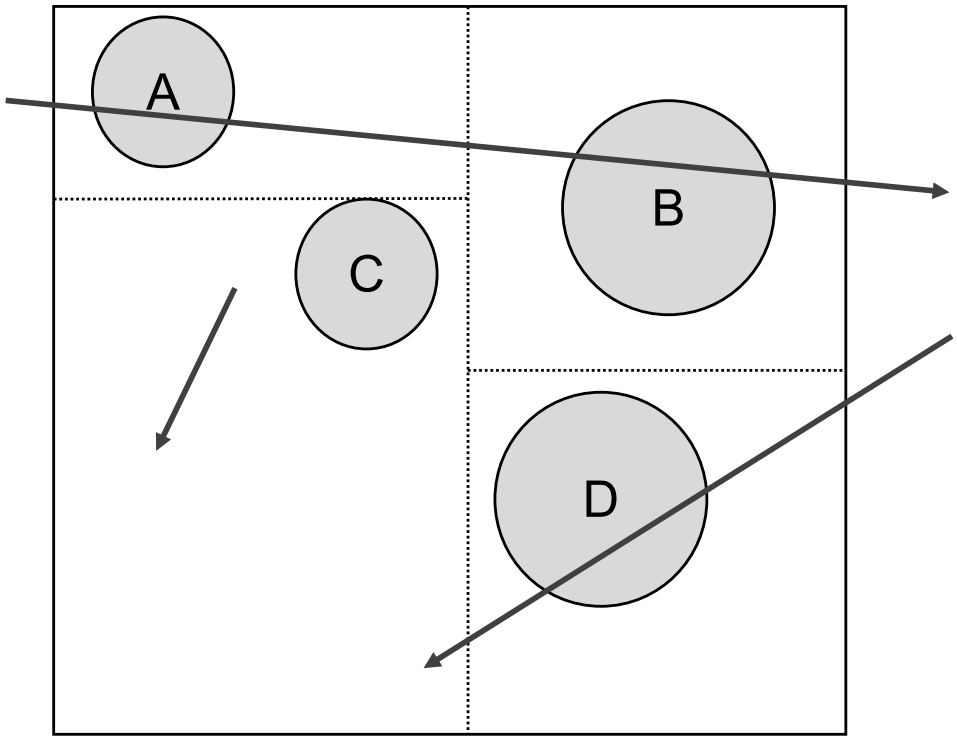


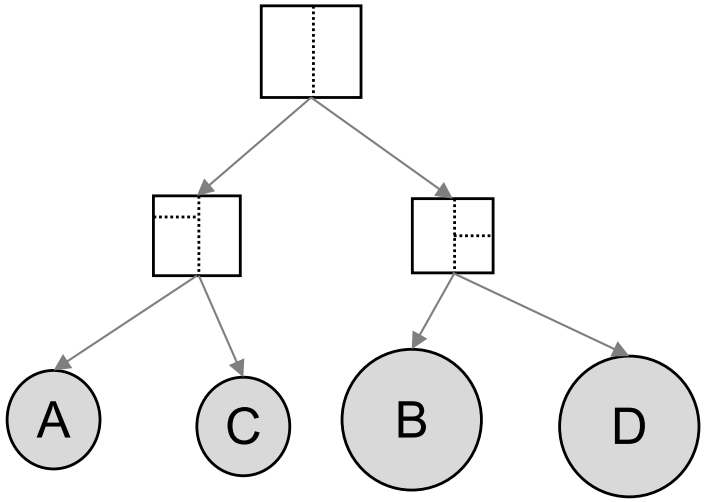


Traversing a kD-tree

```
intersectKDTree(ray, node)
   If empty(ray) return
   If isLeaf(node) intersectObjects(node.objects, ray)
   else
      rayNear = Clip ray to near side of the split
      intersectKDTree(rayNear, nearChild(node, ray))
      If(no intersection)
         rayFar = Clip ray to far side of the plane
         intersectKDTree(rayFar, farChild(node, ray))
```

Let's traverse some





Quiz

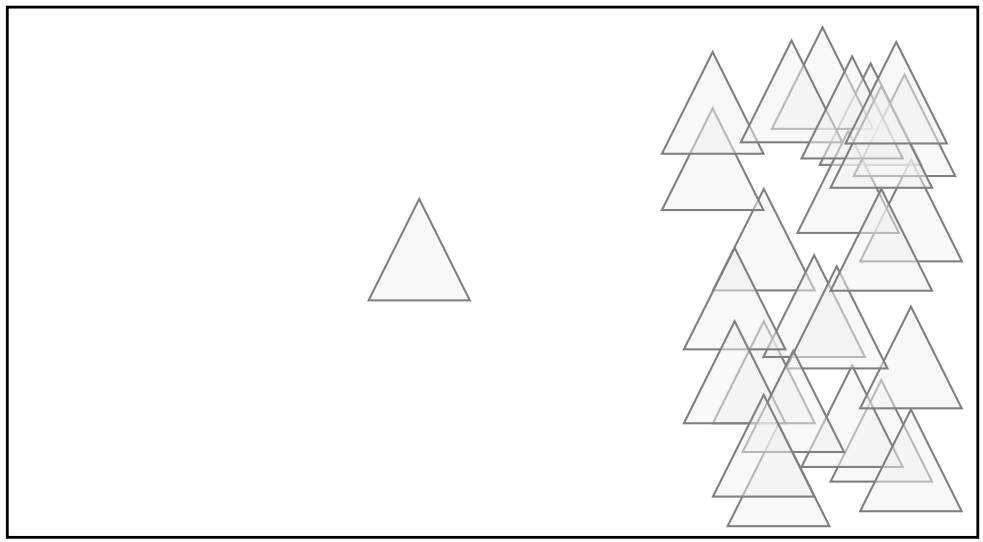
- Why first do near side?
- What happens if we just traverse in file order (the random order in which nods happen to be in memory)?

Building good kD-trees

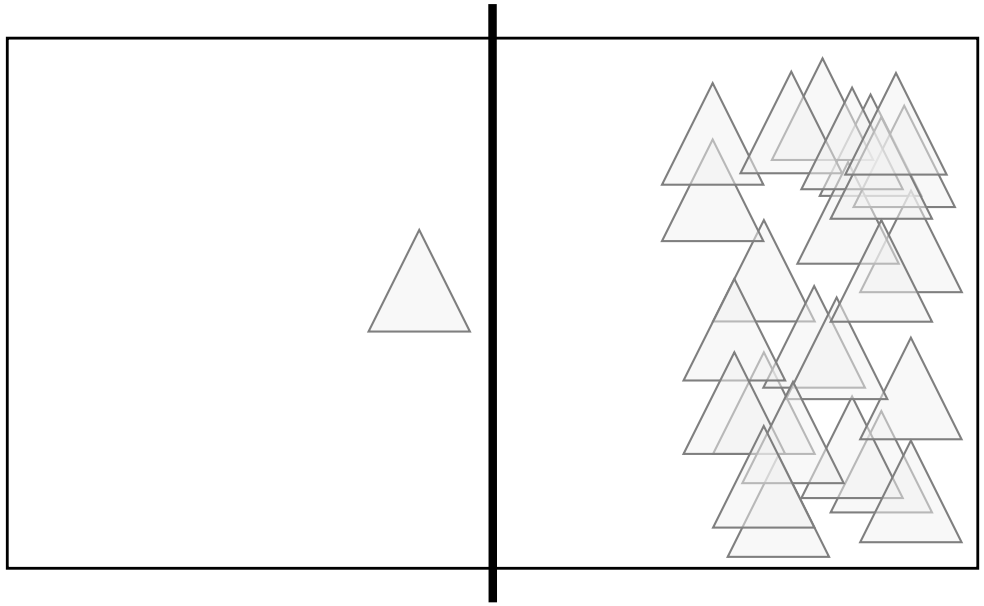
- What split do we really want?
  - Main Idea: The one that makes ray tracing cheap
  - Write down an expression of cost and minimize it
  - Cost Optimization
- What is the cost of tracing a ray through a cell?

$$\text{cost(cell)} = \text{cost}_{\text{trav}} + \text{prob(hit L) cost(L)} + \text{prob(hit R) cost(R)}$$

Splitting with Cost in Mind

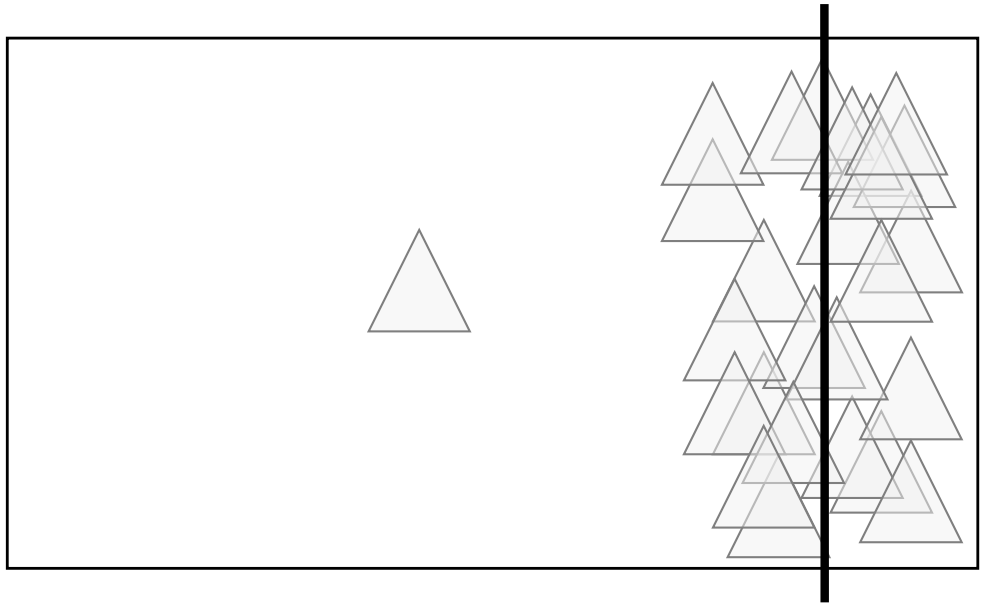


Split in the middle



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

Split at the Median

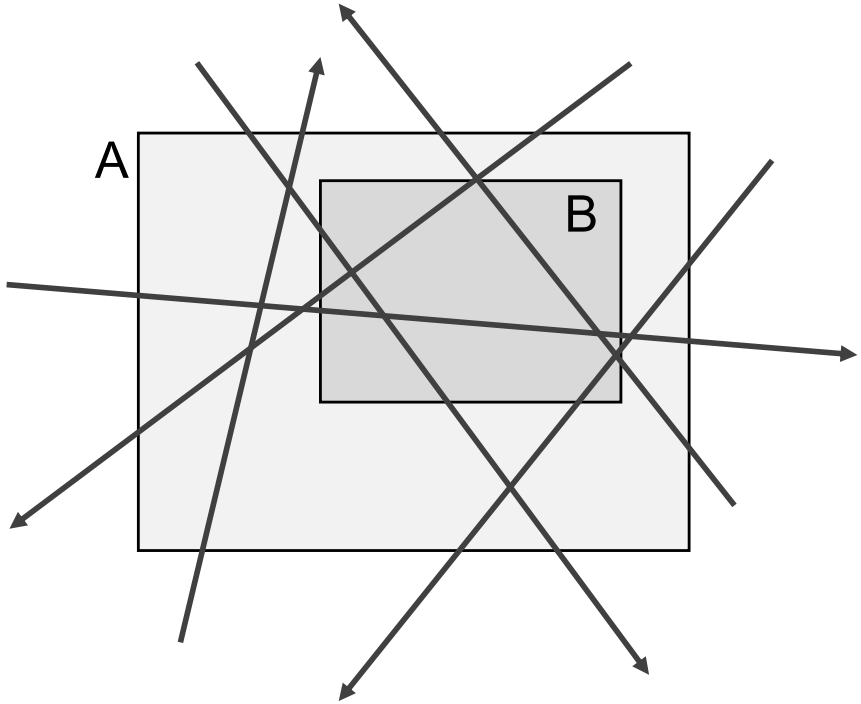


- Makes the L & R costs equal
- Pays no attention to the L & R probabilities

Surface Area and Rays

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

$$\text{prob}(\text{hit}_B \mid \text{hit}_A) = S_b/S_a$$

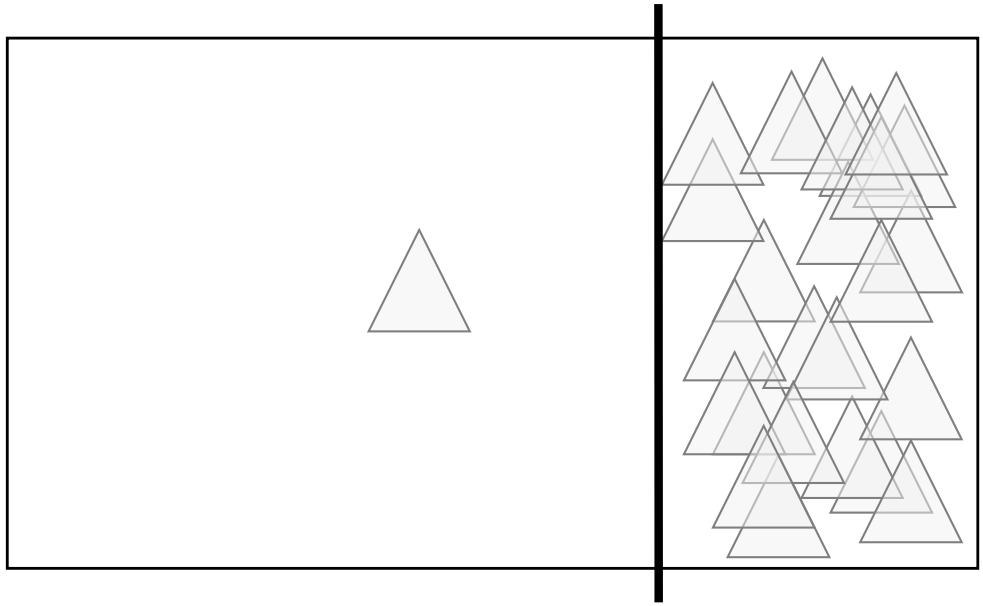


Surface Area Heuristic

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

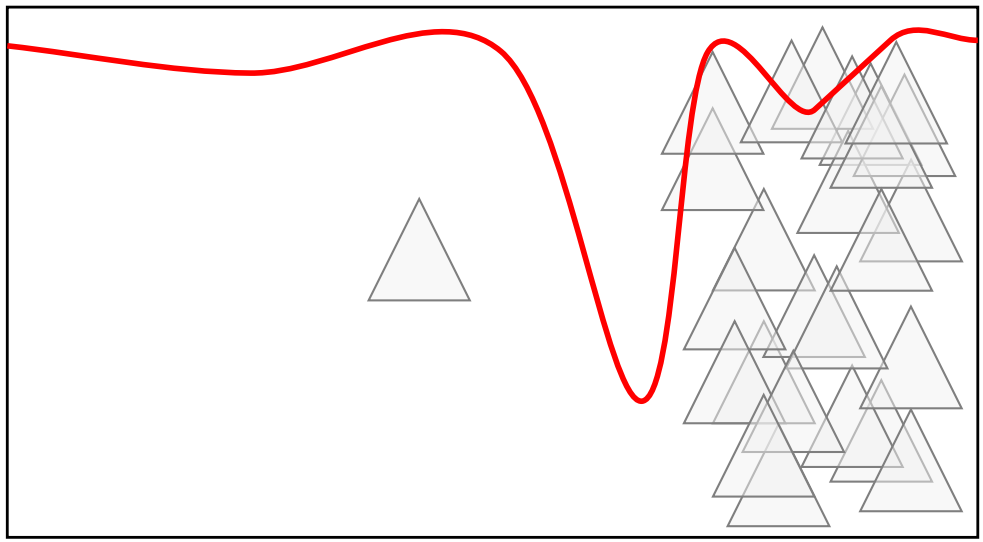
$$\begin{aligned} \text{cost(cell)} &= C_{\text{trav}} + \text{prob(hit } L\text{) cost}(L) + \quad \text{prob(hit } R\text{)} \\ \text{cost}(R) &= C_{\text{trav}} + \text{SA}(L)\ \text{triCount}(L) + \text{SA}(R)\ \text{triCount}(R) \end{aligned}$$

Cost-Optimized Split



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

Splitting is optimization



- Split is optimum of some cost function
- Typical strategies apply
- Primitives need to be sorted to do this reasonably

Two ways to build, one without sort

```
build(nodes)
 xS=findSplit(sortX(nodes))
 yS=findSplit(sortY(nodes))
 zS=findSplit(sortZ(nodes))
 S = best(xS, yS, ZS)
 build(left(S, nodes))
 build(right(S, nodes))
```

```
xNodes = sortX(nodes)
yNodes = sortY(nodes)
zNodes = sortZ (nodes)

build(xNodes, yNodes, zNodes)
 xS=findSplit(xNodes)
 yS=findSplit(yNodes)
 zS=findSplit(zNodes)
 S = best(xS, yS, ZS)
 build(left(S, xNodes, yNodes, zNodes))
 build(right(S, xNodes, yNodes, zNodes))
```

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