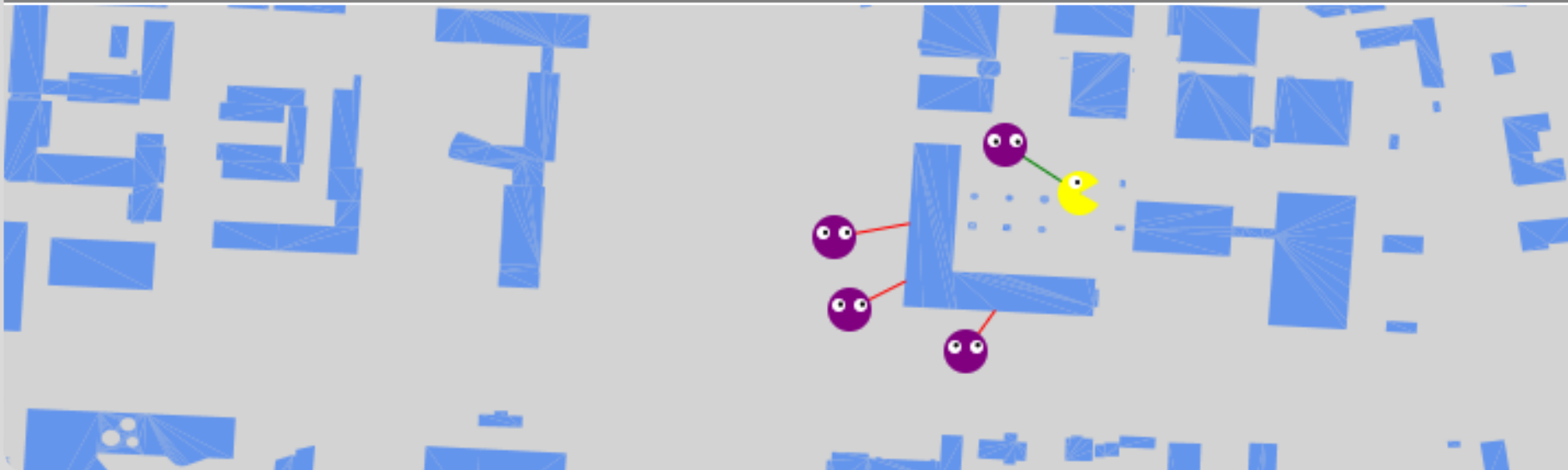


Pacman – Ghosts activate on sight

Mao L., Wang C., Sebastian G.



Problem specification

■ Given

- A graph $G = (V, E)$ describing a landscape with obstacles modeled through
 - simple polygons with n vertices in total
- A straight-line drawing of the landscape $\Gamma: V \rightarrow R^2$
- The position $P \in R^2$ (Pacman)
- The positions $(Q_i \in R^2)_{i \in \{1, \dots, m\}}$ of m ghosts

■ Problem

- Find an efficient way to determine if P is *visible* from Q_i for each $i \in \{1, \dots, m\}$
- For positions $A, B \in R^2$, A is *visible* from B iff the segment \overline{AB} has no intersection with any polygon in Γ

Our approach

- OpenStreetMap data for realistic input geometry
- Java
- Outline:
 1. Parse an OSM file for polygons of buildings
 2. Triangulate the input polygons
 3. Build a *kd*-tree on the triangle soup
 4. Perform the m visibility checks with the help of the *kd*-tree in $O(m \log n)$ time

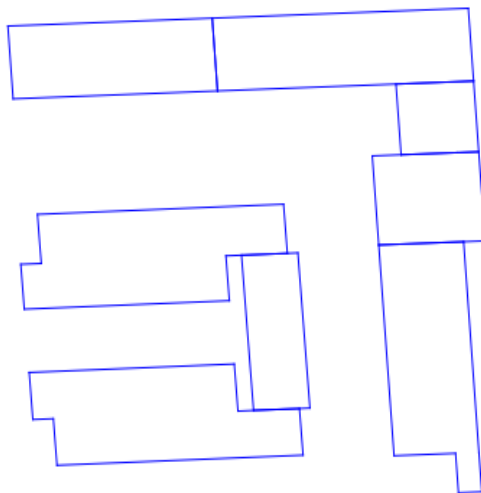
DEMO

Polygon Triangulation

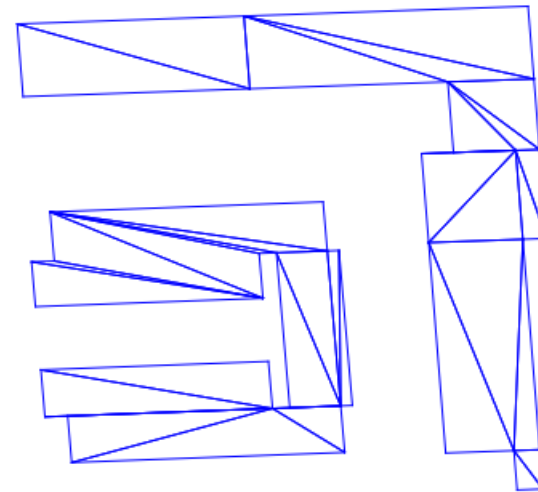
EAR CLIPPING

Task

- Decompose polygons into a collection of triangle



untriangulated



triangulated

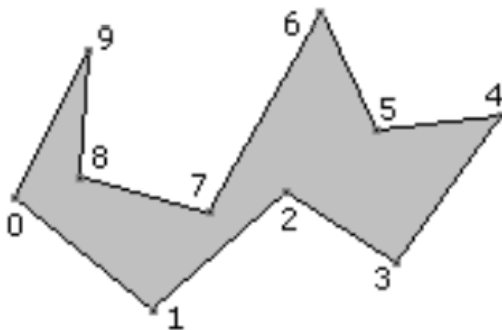
Types of polygons

Simple polygon :

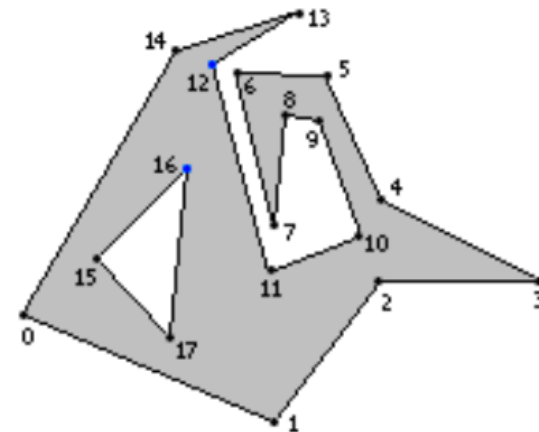
- Traverse the edges the interior bounded region is always to one side

Polygon with holes :

- Consists of one outer polygon and at least one inner polygon



Simple polygon



Polygon with holes

Ear Clipping for the simple polygon

Definition:

- Polygon an ordered sequence of n vertices, V_0 through V_{n-1}
- Ear V_i

Three consecutive vertices V_{i-1} , V_i , V_{i+1} . V_i is a convex vertex

No vertices of the polygon are contained in Triangle (V_{i-1}, V_i, V_{i+1})

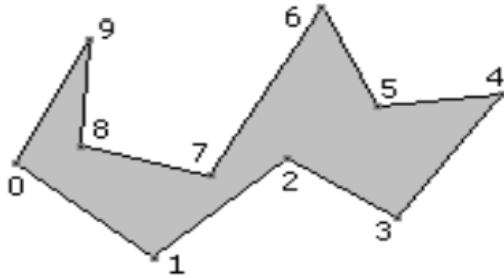
Theorem :

- Any triangulation of n vertices polygon has $n-2$ triangles
- A polygon has at least two nonoverlapping ears

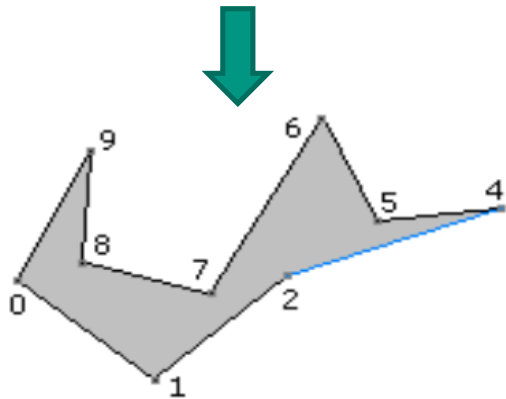
Algorithm Ear Clipping

- Step1 Initiate the Earlist
- Step2 Get ear V_i from ear list, if V_i is ear, go Step3,
else repeat Step2
- Step3 Create triangle (V_{i-1} , V_i , V_{i+1})
- Step4 Judge, whether V_{i-1} and V_{i+1} are ears. If so, add
them into Earlist
- Step5 Remove ear
- Step6 If the vertices less than four
end
- else
go Step2

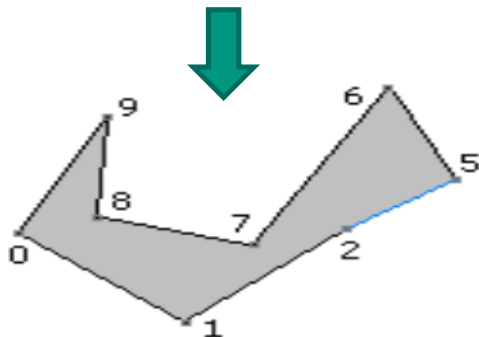
Example



The initial list of ears is $E = \{ \textcolor{red}{3}, 4, 6, 9 \}$



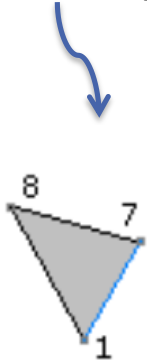
The ear at vertex $\textcolor{red}{3}$ is removed, $E = \{ \textcolor{red}{4}, 6, 9 \}$



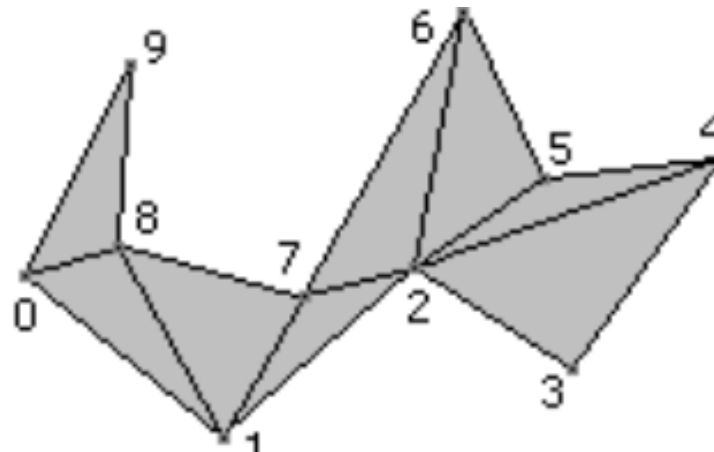
Added vertex $\textcolor{red}{5}$, removed vertex 4, $E = \{ \textcolor{red}{6}, 9 \}$

Example

Repeat the process



The number of vertices less than **four**, then end



The full triangulation of the original polygon

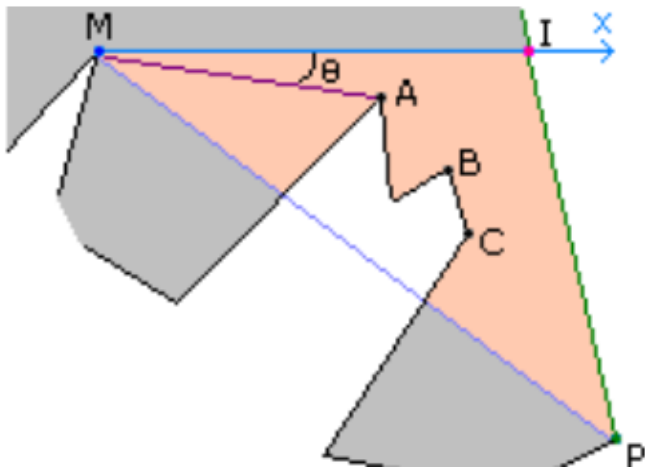
Time Complexity

- There are $O(n)$ ears. Each update of an adjacent vertex involves an earonestest, a process that is $O(n)$ per update. Thus, the total removal process is $O(n^2)$.

Polygon with holes

Idea

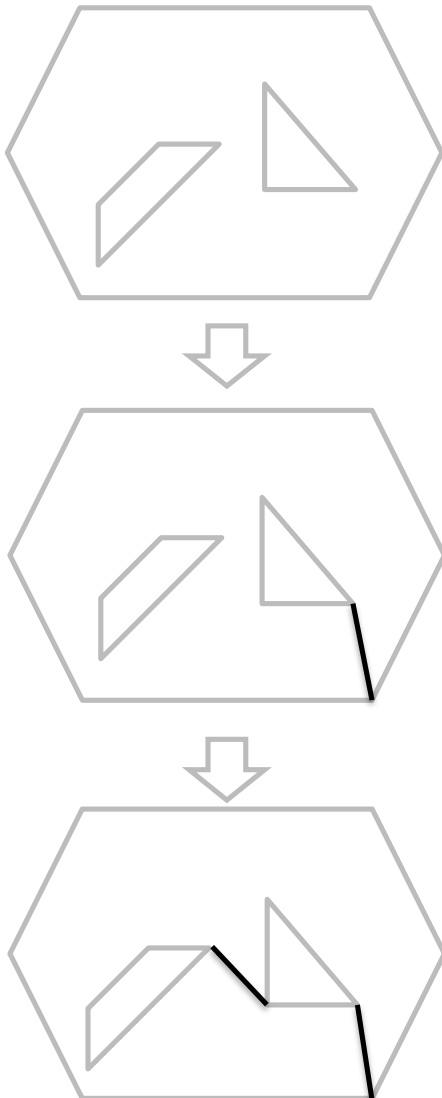
Connect two vertex, one vertex from the outer polygon and one vertex from the inner polygon



Algorithm

- Find the vertex in inner polygon M of maximum x-value
- Find the vertex P with most left segment of the outer polygon
- Find the vertex R that minimizes the angle
- Connect M and R

Polygons with Multiple Holes

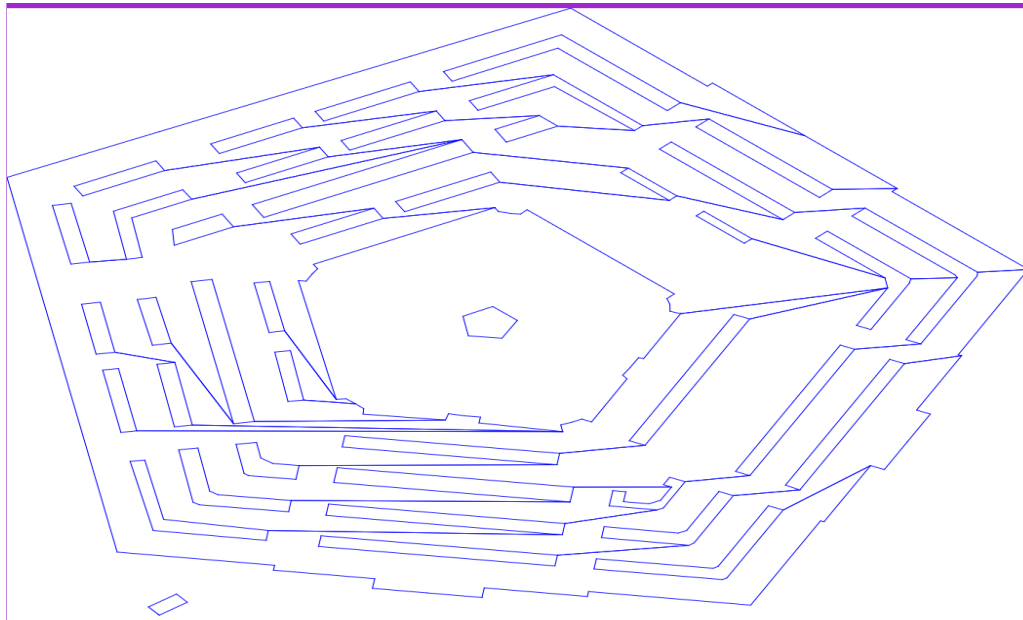


Algorithm

- Find the inner polygon with the vertex of maximum x-value
- Use the previously mentioned algorithm to combine the outer polygon and the select inner polygon
- Repeated with the new outer polygon and the remaining inner polygons

Time Complexity

- There are $O(m)$ inner polygons. Each connection building needs $O(n)$. Thus, the total process is $O(m \cdot n)$.

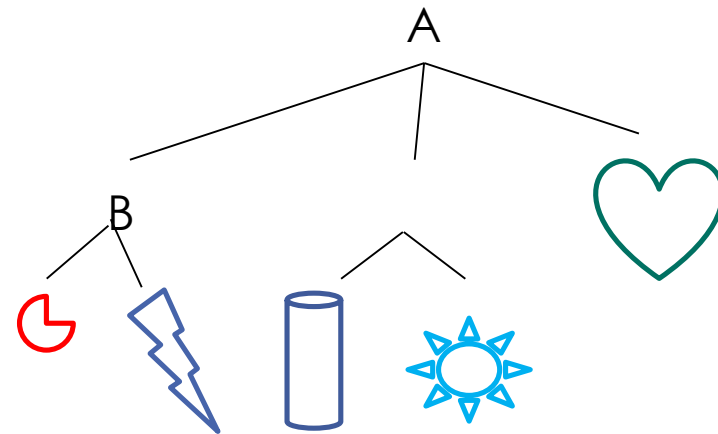
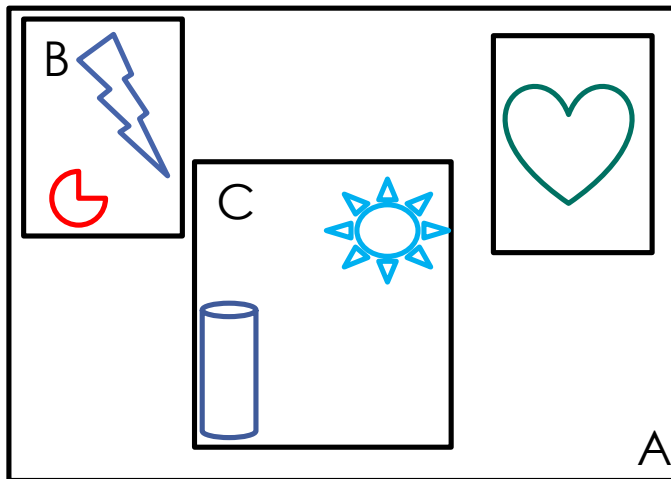


Example polygon with holes to simple polygon

Bounding Volume Hierarchy

BVH

BVH Structure



BVH Introduction

- Tree structure on a set of geometric objects

BVH Introduction

- Tree structure on a set of geometric objects
- Leaf nodes of the tree:
Geometric objects (wrapped in bound volumes)

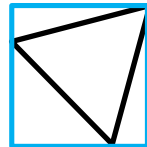
BVH Introduction

- Tree structure on a set of geometric objects
- Leaf nodes of the tree:
Geometric objects (wrapped in bound volumes)
- Child Nodes → Small sets
→ Father node (with larger bounding volumes)

BVH in the project

Leaves of the tree

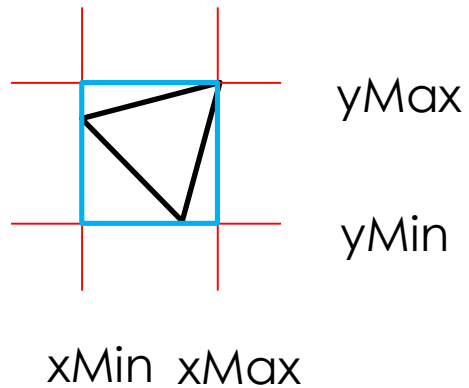
- Triangles (triangulated form the polygons)



BVH in the project

Leafs of the tree

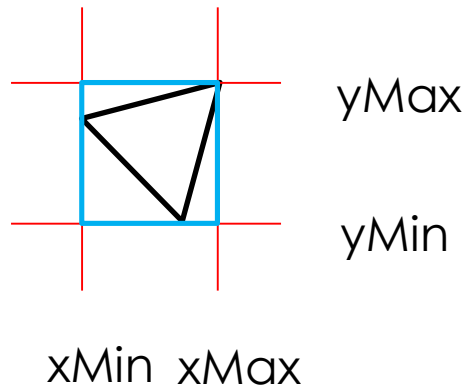
- Triangles (triangulated form the polygons)



BVH in the project

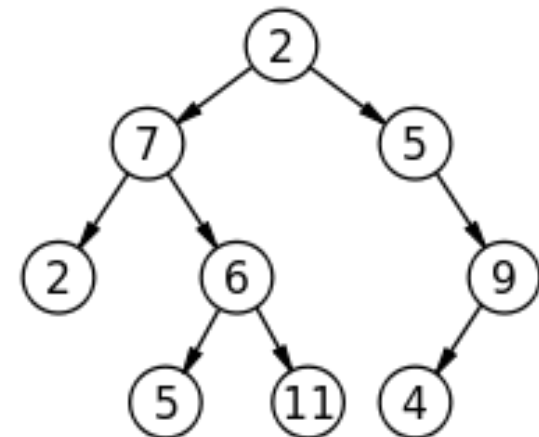
Leafs of the tree

- Triangles (triangulated form the polygons)



Structure of the tree

- Binary tree



BVH build up

Method :

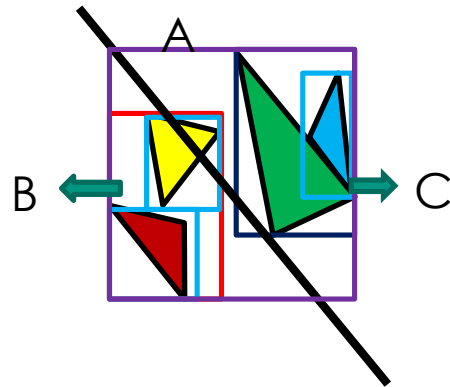
- Find the nearest bounding volumes
- Combination → new BVH node generated
-
- When only one activated node exists
→ BVH completed

BVH search

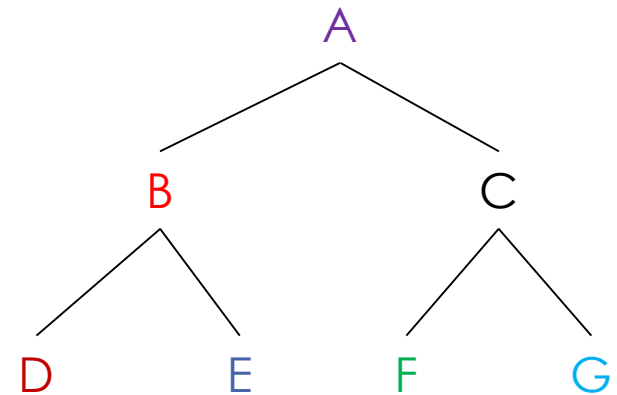
Method :

- When node A need to be searched:
 - Segment intersects A's bounding volume?
 - Yes
 - A is a leaf node → report A
 - A is not a leaf node → search A's children

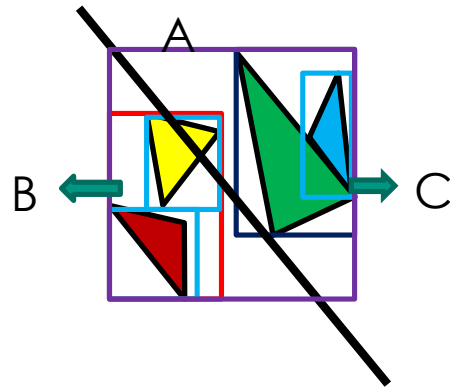
BVH search example



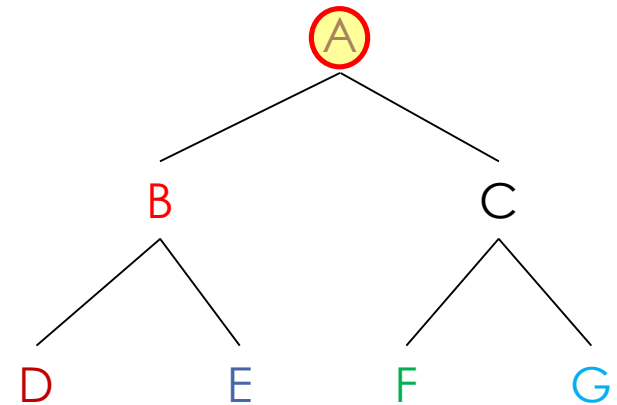
Intersect ?
Leaf node?



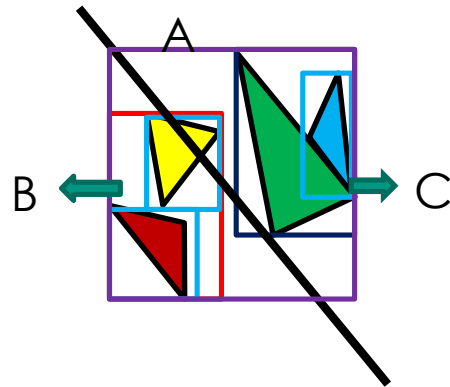
BVH search example



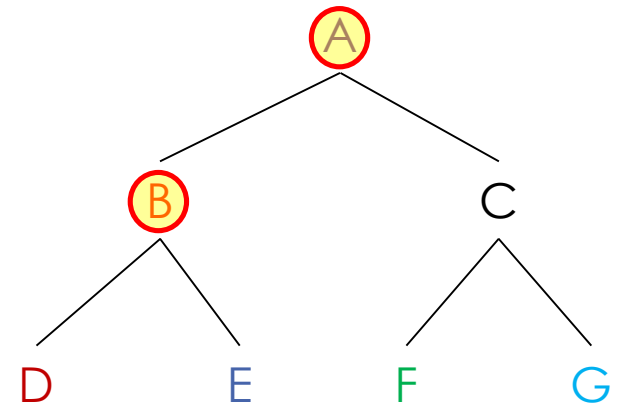
Intersect ?
Leaf node?



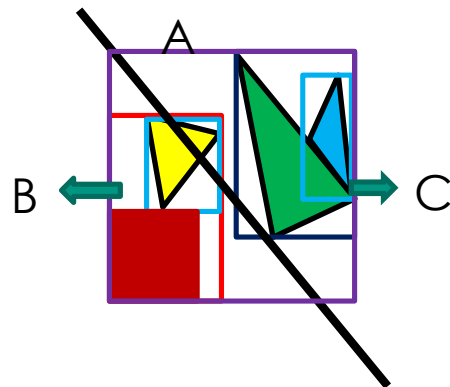
BVH search example



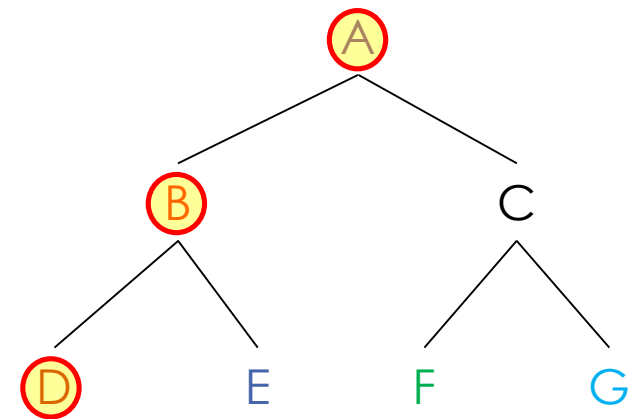
Intersect ?
Leaf node?



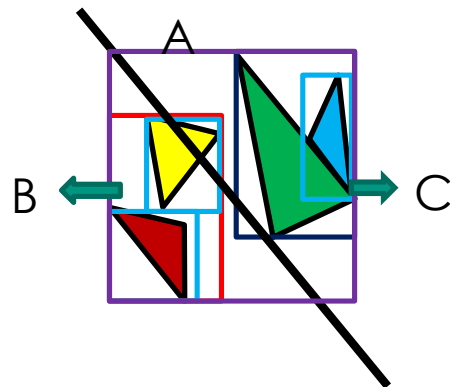
BVH search example



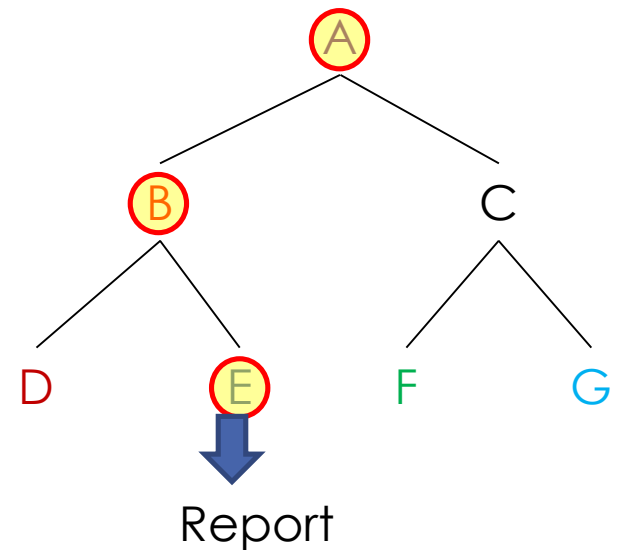
Intersect ?
Leaf node?



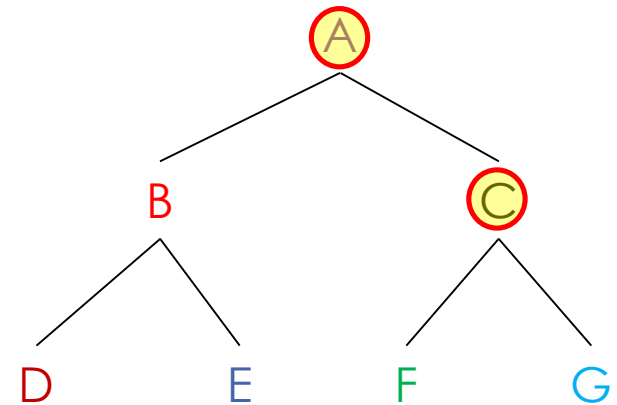
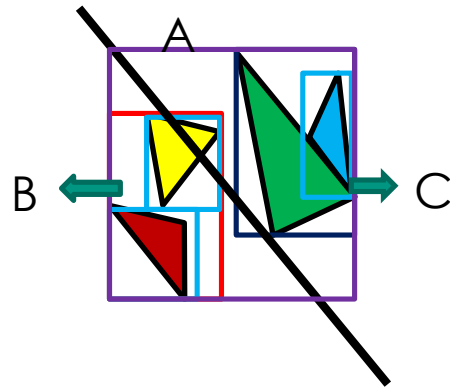
BVH search example



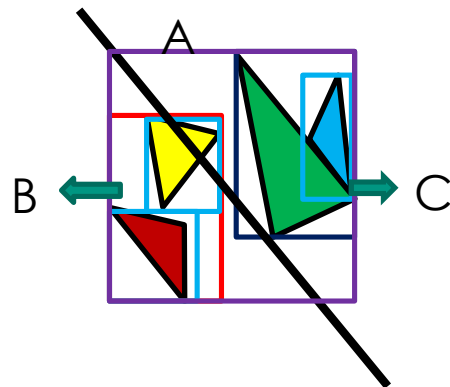
Intersect ?
Leaf node?



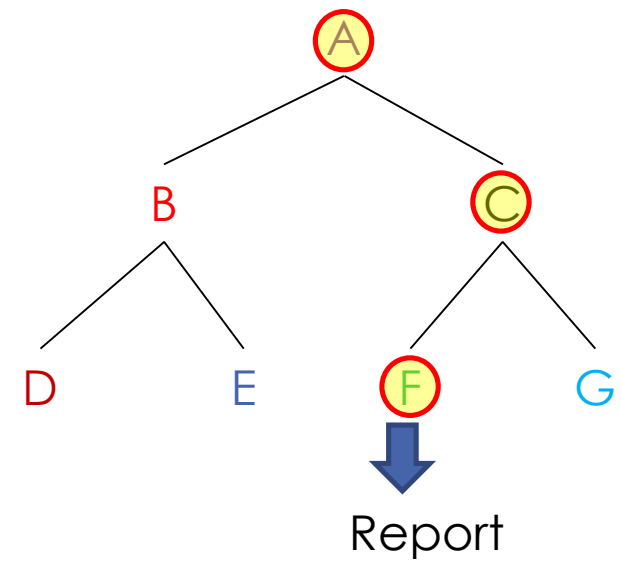
BVH search example



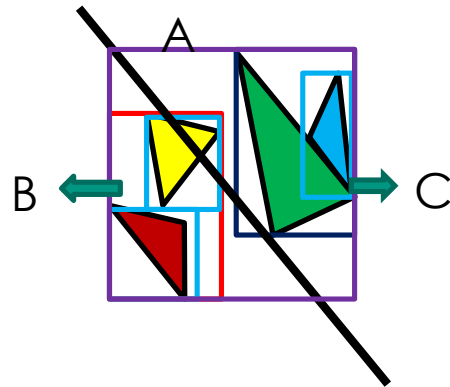
BVH search example



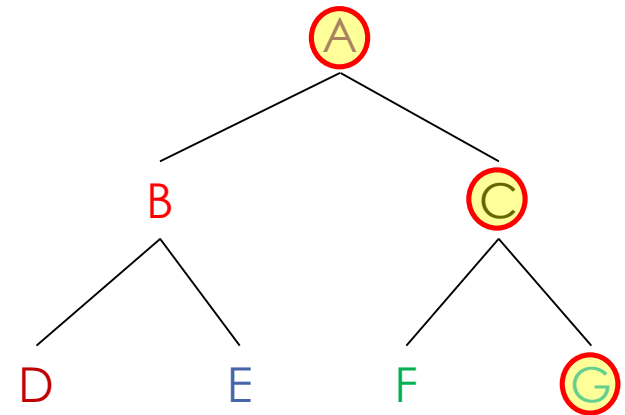
Intersect ?
Leaf node?



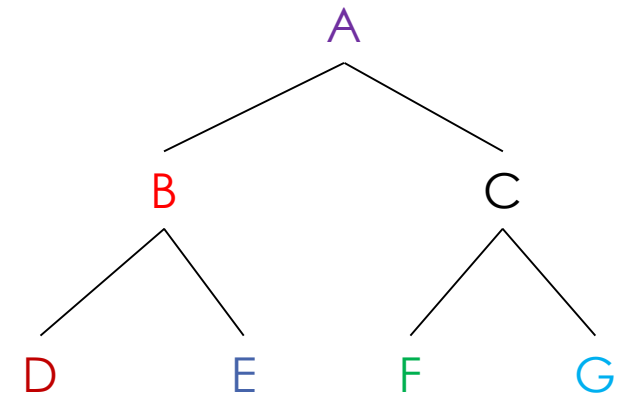
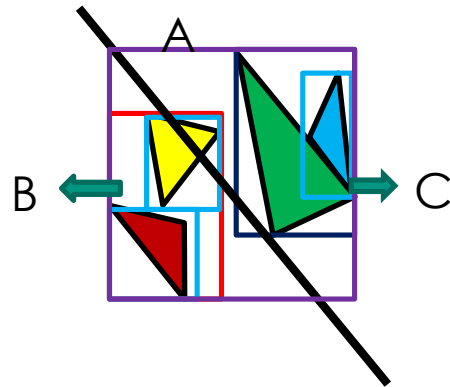
BVH search example



Intersect ?
Leaf node?



BVH search example



Result : E , F
reported

Advantage

- Simple structure
- Allows overlapped triangles
- Fast binary search

Disadvantage

- Unexpected output (sometimes)
<reason : segment intersects the bounding volumes but not intersects the triangles>
- Long construction time ($O(n^3)$)
<reason : find the nearest bounding volumes costs too much time $O(n^2)$ >

kd-tree

INTERSECTION TEST

Intersection test

■ Given

- The line-of-sight segment \overline{AB} to check for intersection with the triangles
- The currently visited node n in the kd -tree

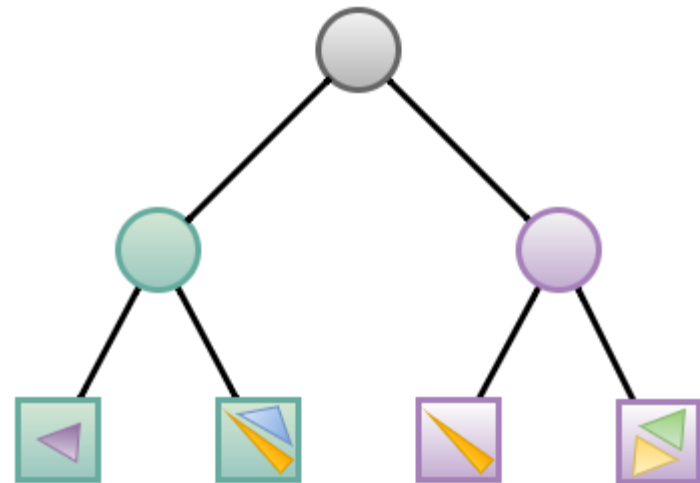
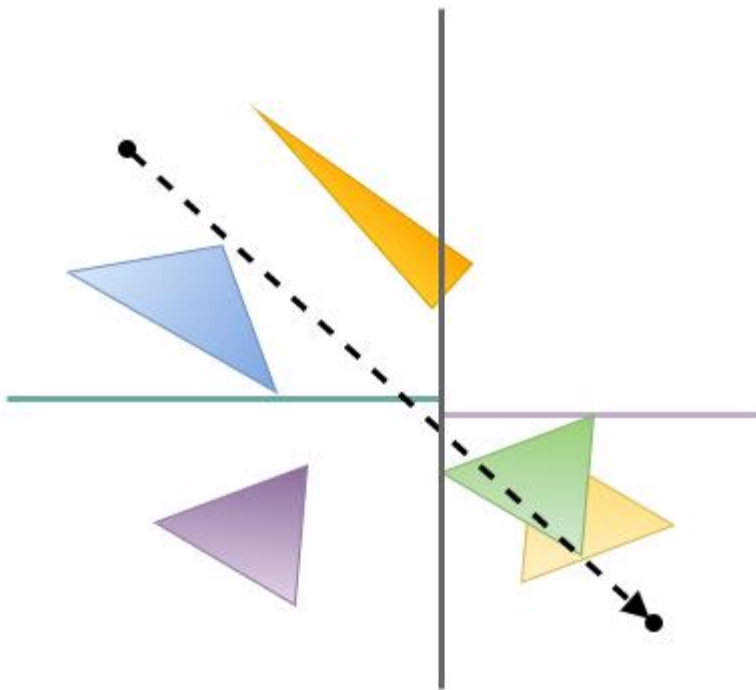
■ If n is not a leaf

1. Split \overline{AB} into \overline{AC} and \overline{CB} , where C is the intersection of the line through \overline{AB} with the splitting plane of n
2. Perform the intersection test with the child of n containing A and \overline{AC} as line-of-sight segment
3. Perform the intersection test with the child of n containing B and \overline{CB} as line-of-sight segment

■ If n is a leaf

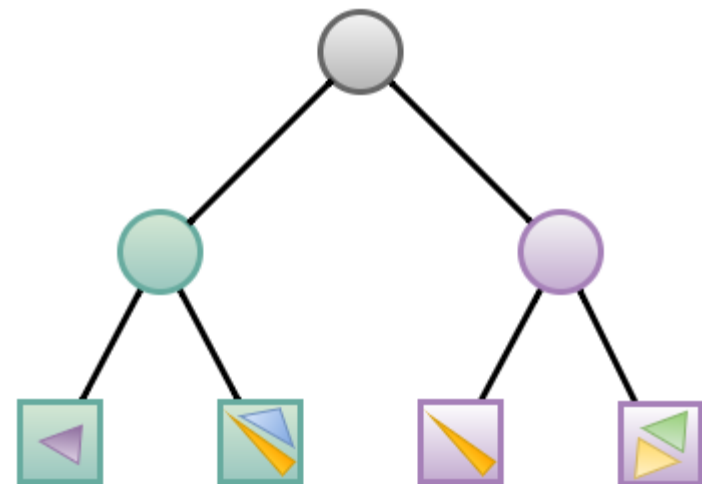
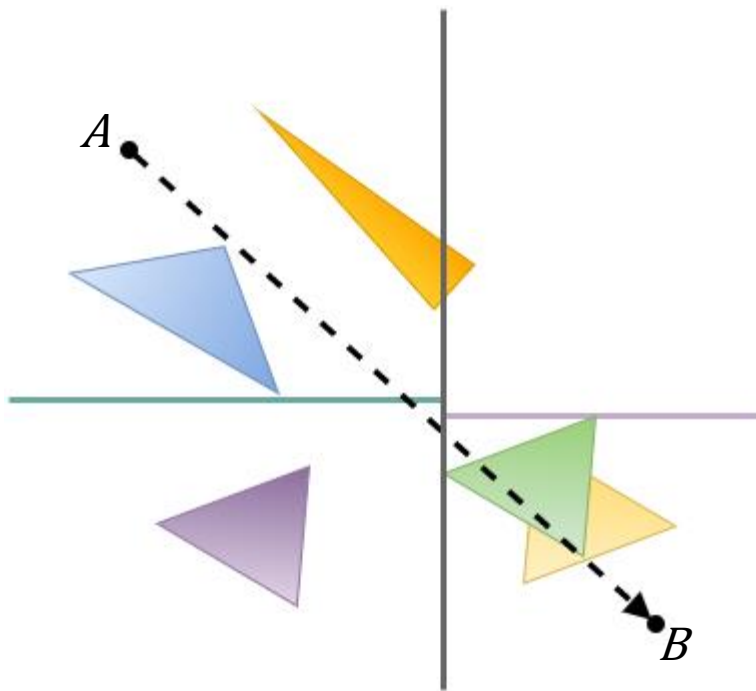
1. Return the *nearest* intersection of \overline{AB} with any of the associated triangles of n , if any

Intersection test - example



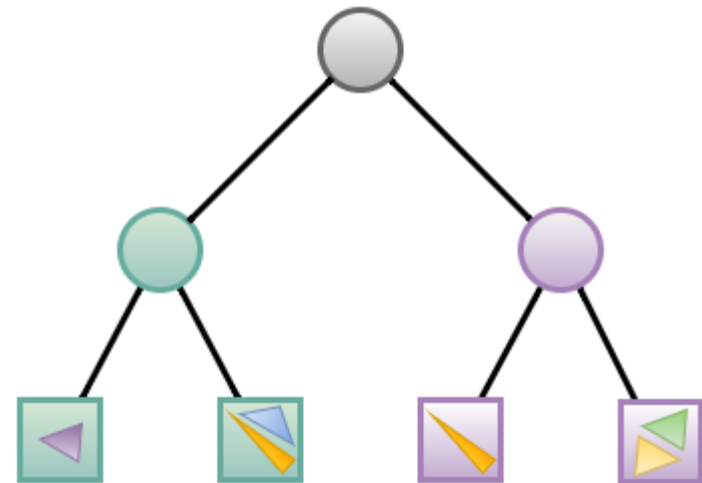
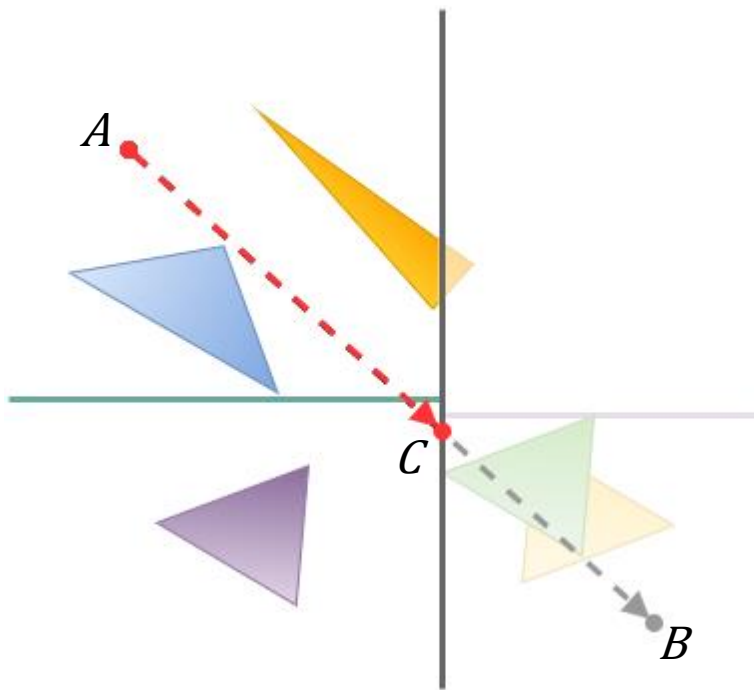
Intersection test - example

- Split \overline{AB} into \overline{AC} and \overline{CB} , where C is the intersection of the line through \overline{AB} with the splitting plane of n



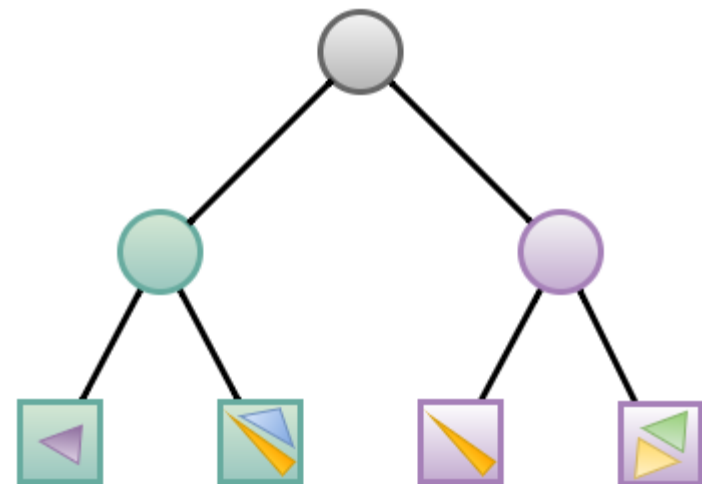
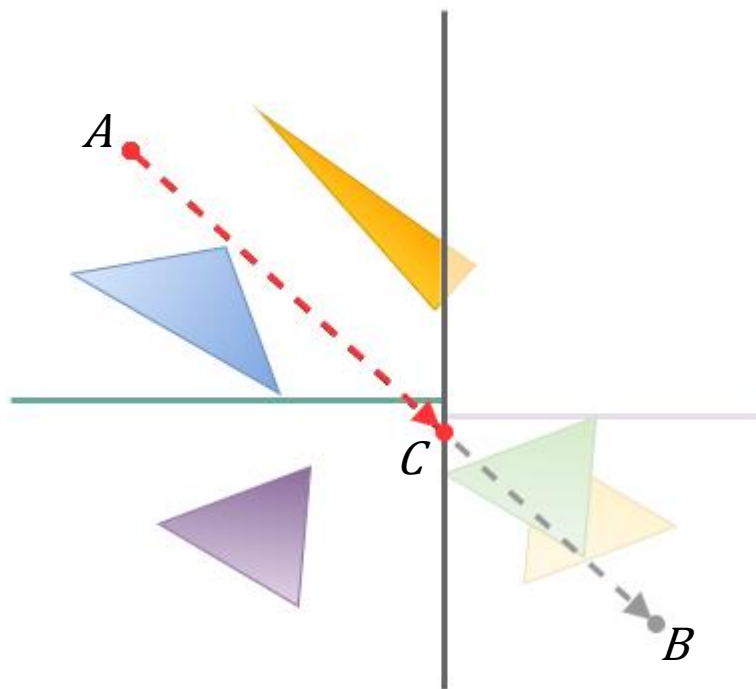
Intersection test - example

- Split \overline{AB} into \overline{AC} and \overline{CB} , where C is the intersection of the line through \overline{AB} with the splitting plane of n



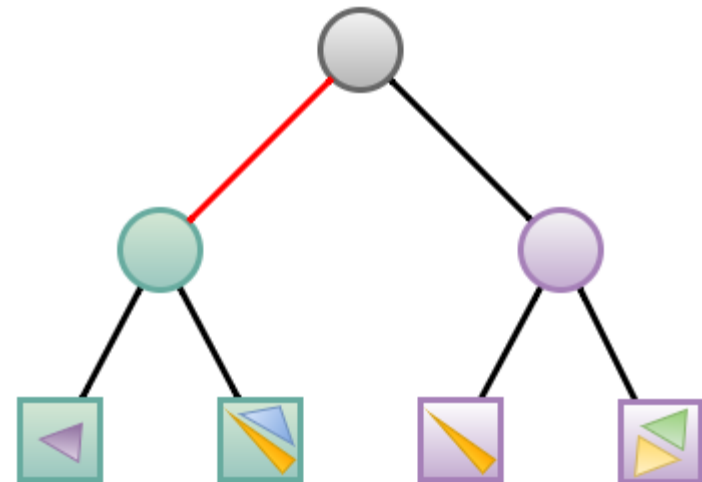
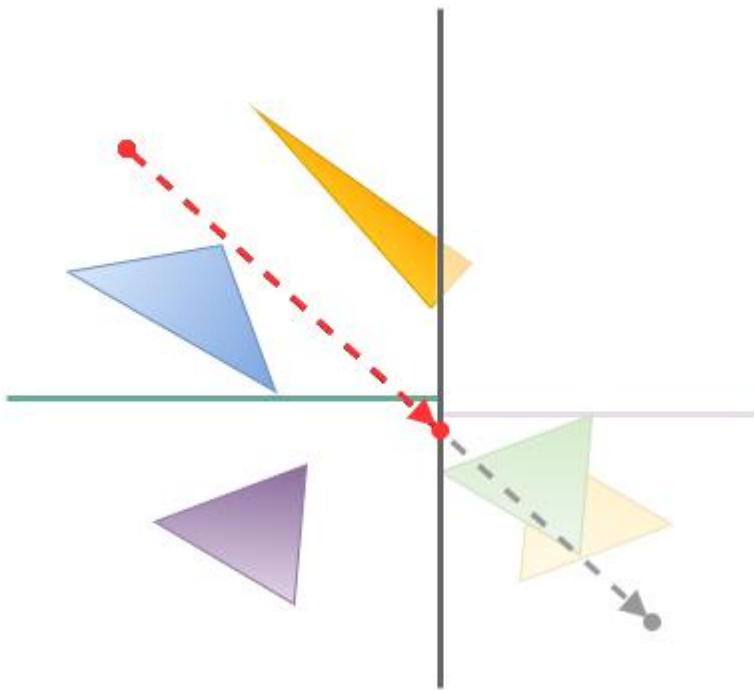
Intersection test - example

- Perform the intersection test with the child of n containing A and \overline{AC} as line-of-sight segment



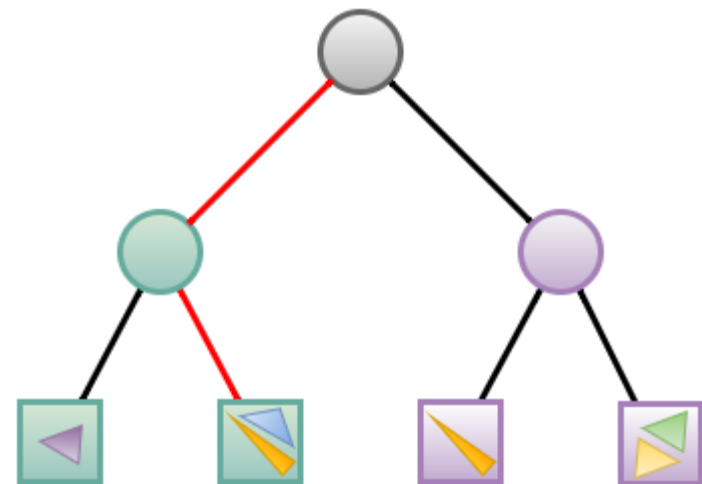
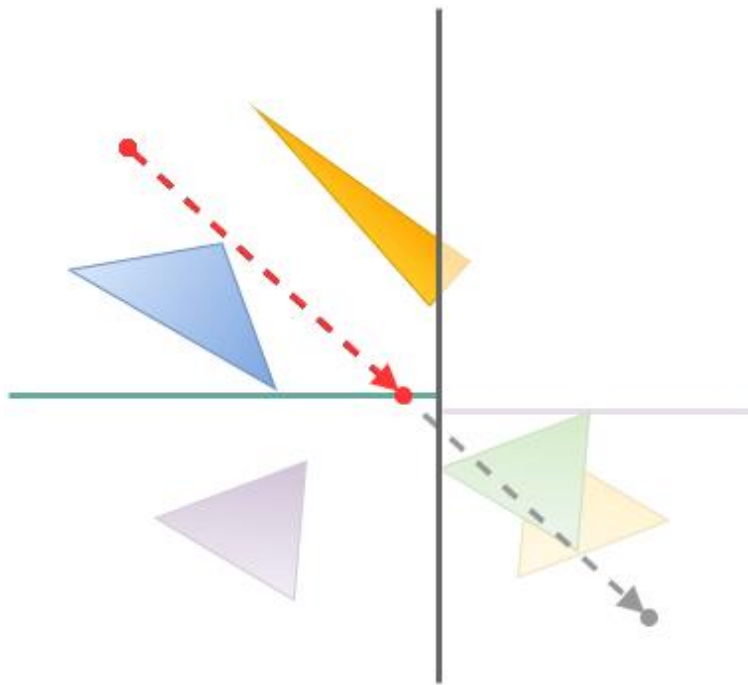
Intersection test - example

- Perform the intersection test with the child of n containing A and \overline{AC} as line-of-sight segment



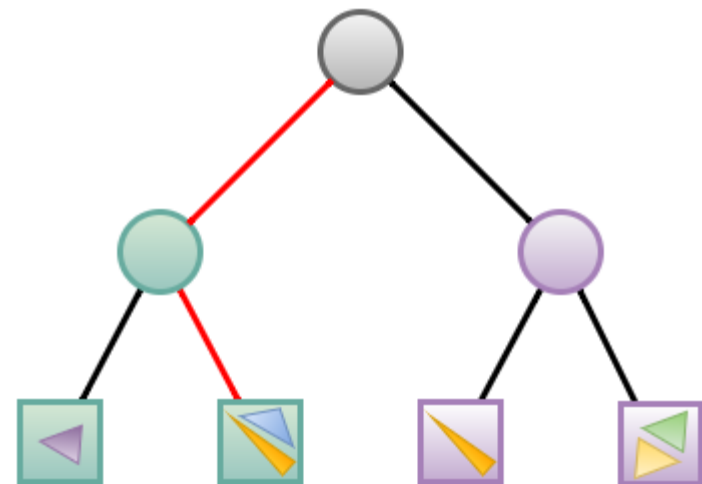
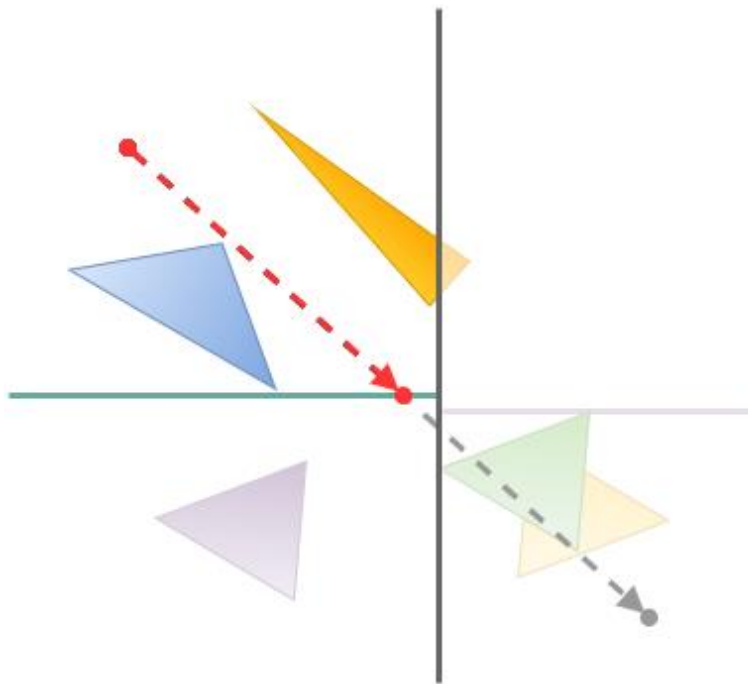
Intersection test - example

- Perform the intersection test with the child of n containing A and \overline{AC} as line-of-sight segment



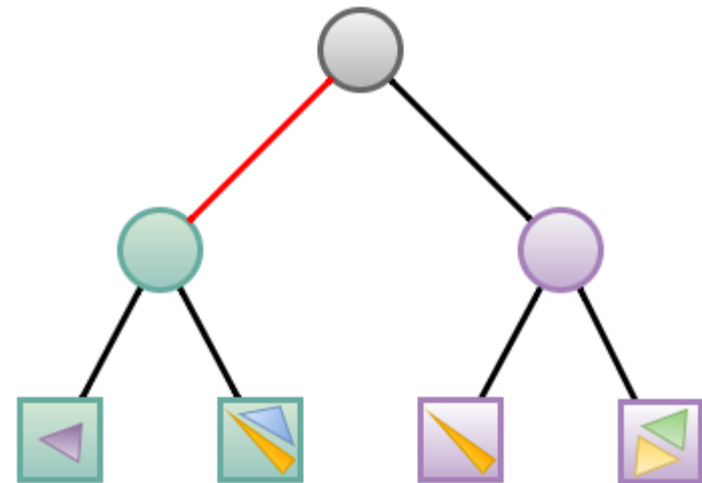
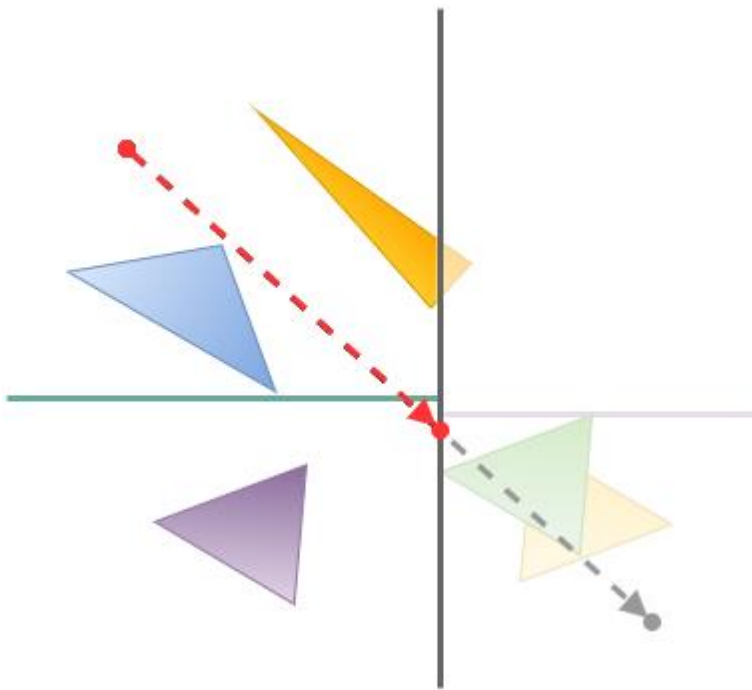
Intersection test - example

- Return the *nearest* intersection of \overline{AB} with any of the associated triangles of n , if any



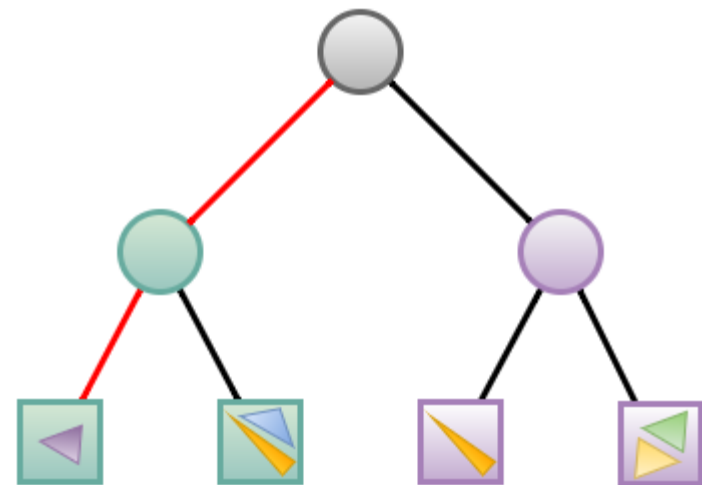
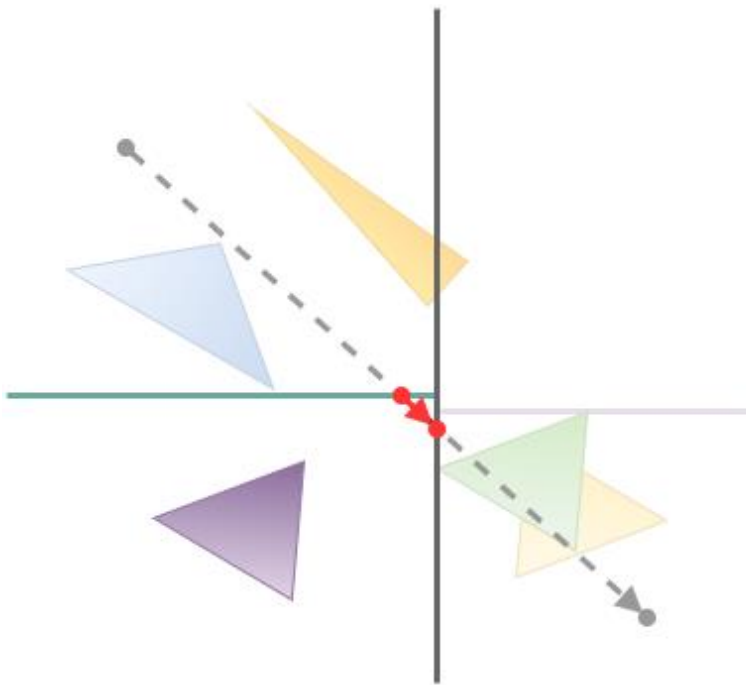
Intersection test - example

- Perform the intersection test with the child of n containing B and \overline{CB} as line-of-sight segment

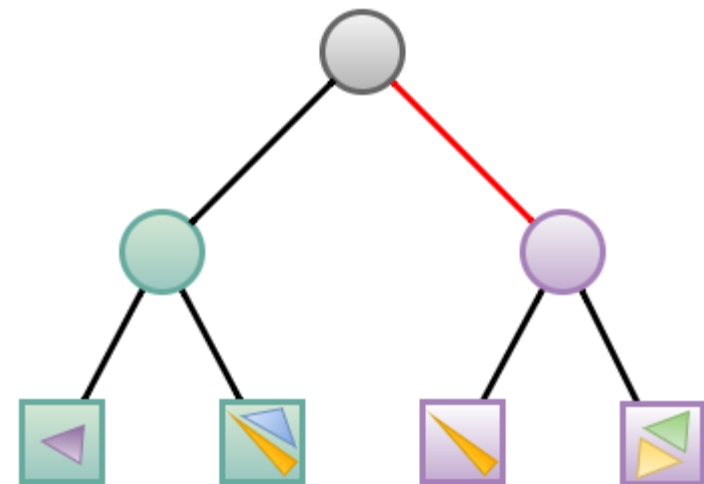
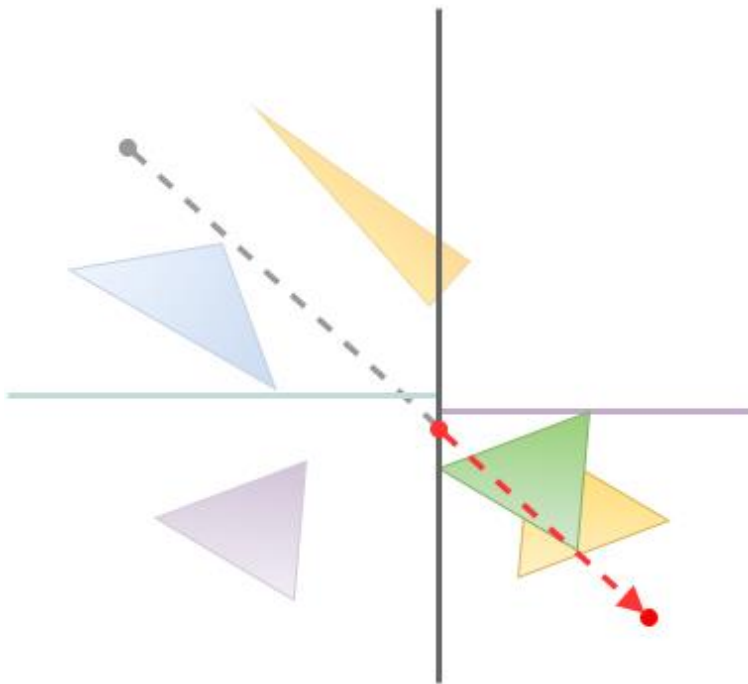


Intersection test - example

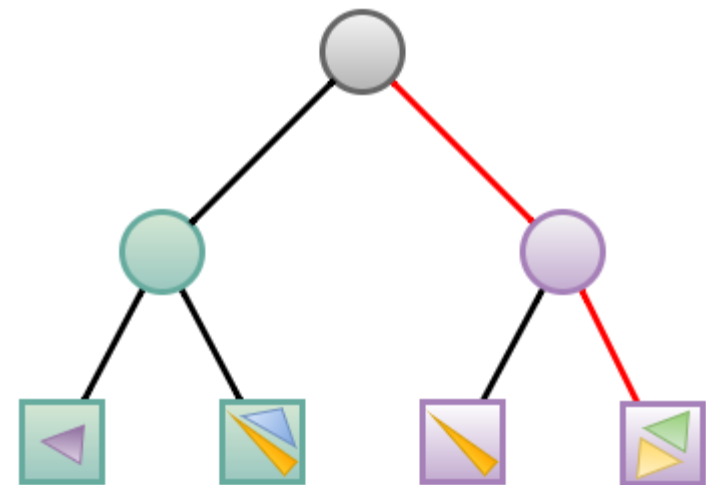
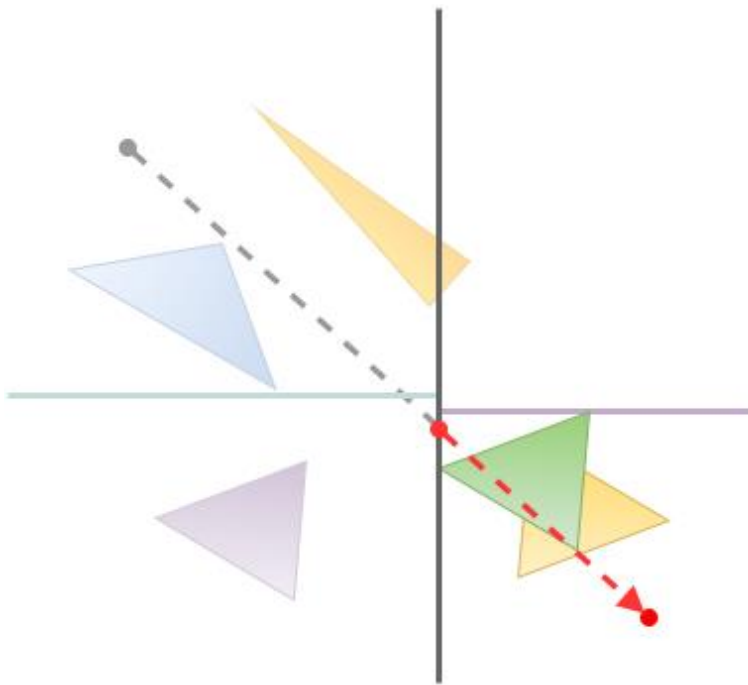
- Perform the intersection test with the child of n containing B and \overline{CB} as line-of-sight segment



Intersection test - example

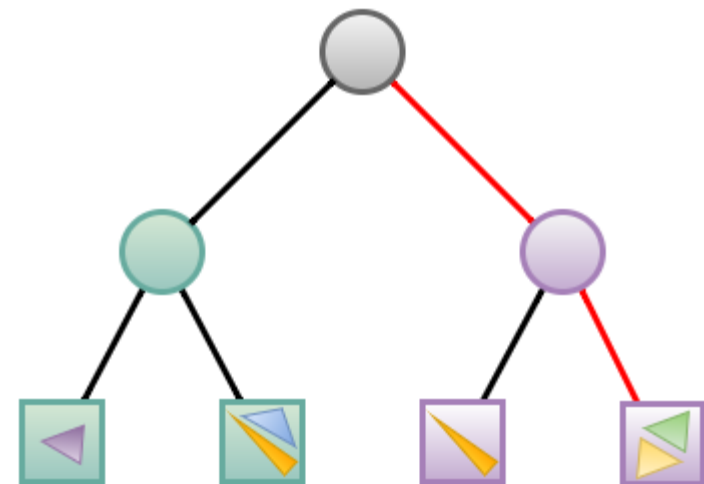
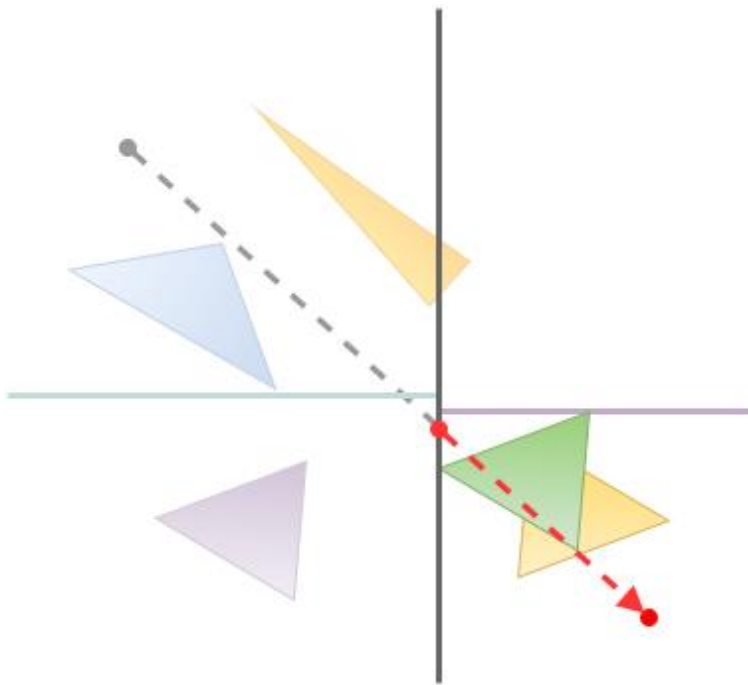


Intersection test - example



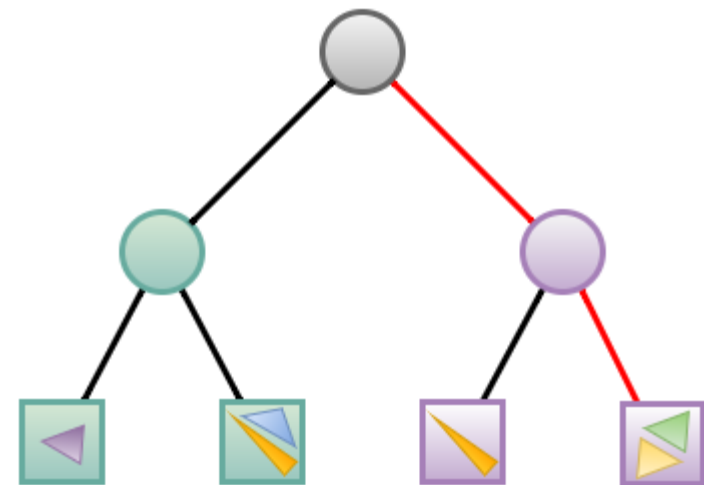
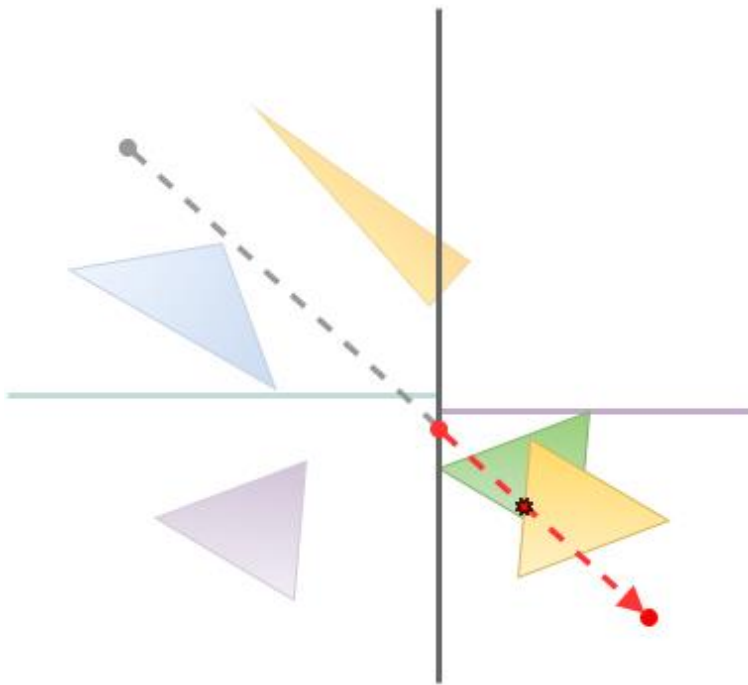
Intersection test - example

- Return the *nearest* intersection of \overline{AB} with any of the associated triangles of n , if any



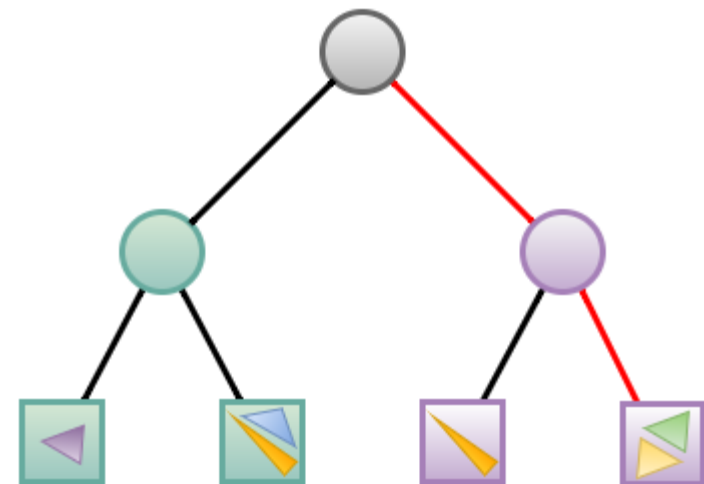
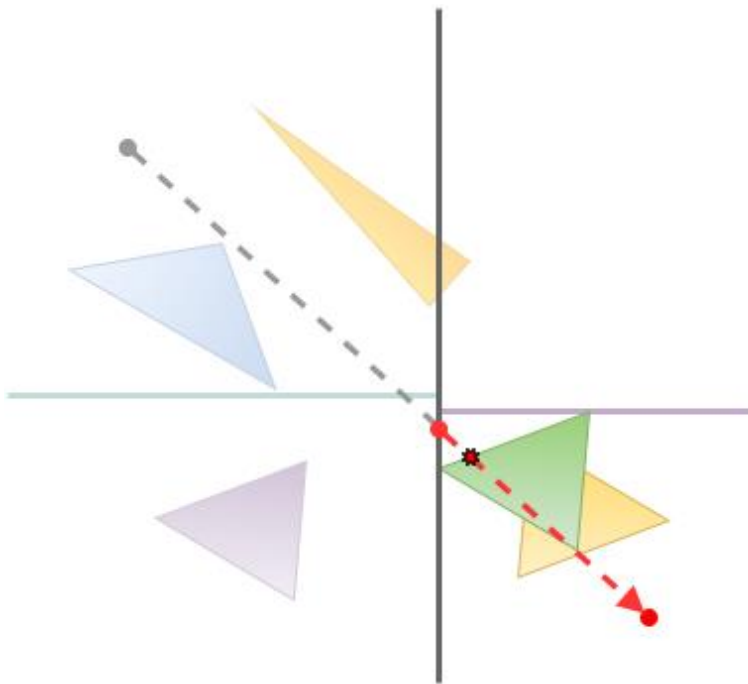
Intersection test - example

- Return the *nearest* intersection of \overline{AB} with any of the associated triangles of n , if any



Intersection test - example

- Return the *nearest* intersection of \overline{AB} with any of the associated triangles of n , if any

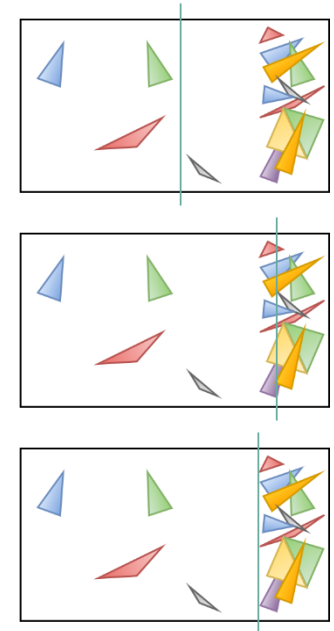


kd-tree

CONSTRUCTION

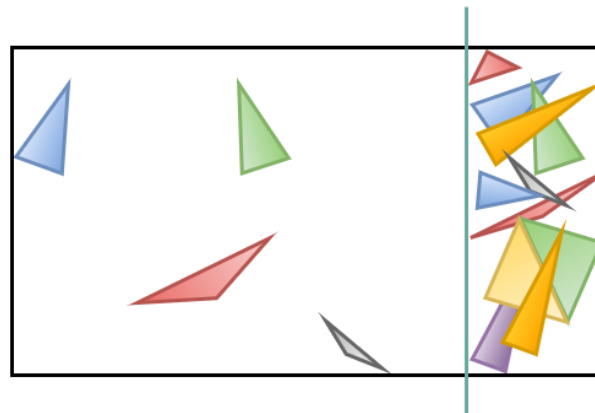
Construction

- Given a list of polygons, build a *kd*-tree such that average query time is low
- Different heuristics
 - Spatial median
 - Object median
 - Cost function (Surface Area Heuristic)
- Implemented *kd*-tree construction based on Surface Area Heuristic
- Based on the $O(n \log^2 n)$ approach in [1]
 - Paper also describes $O(n \log n)$ algorithm



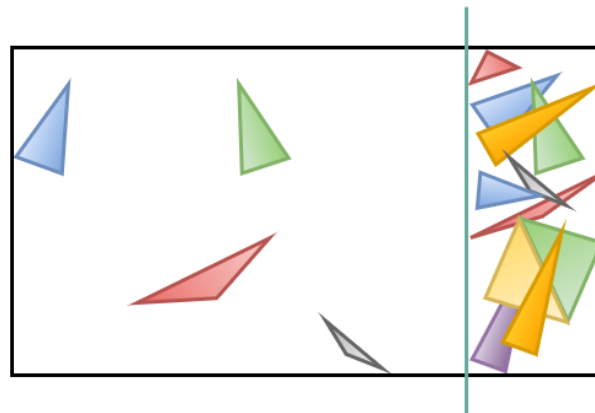
Surface Area Heuristic

- Assigns costs to splits
- Lowest scoring splits (*perfect* splits) are optimal under the following assumptions:
 - Assuming uniformly distributed rays penetrating the bounding box
 - kd-tree traversal costs and triangle intersection costs are known



Surface Area Heuristic

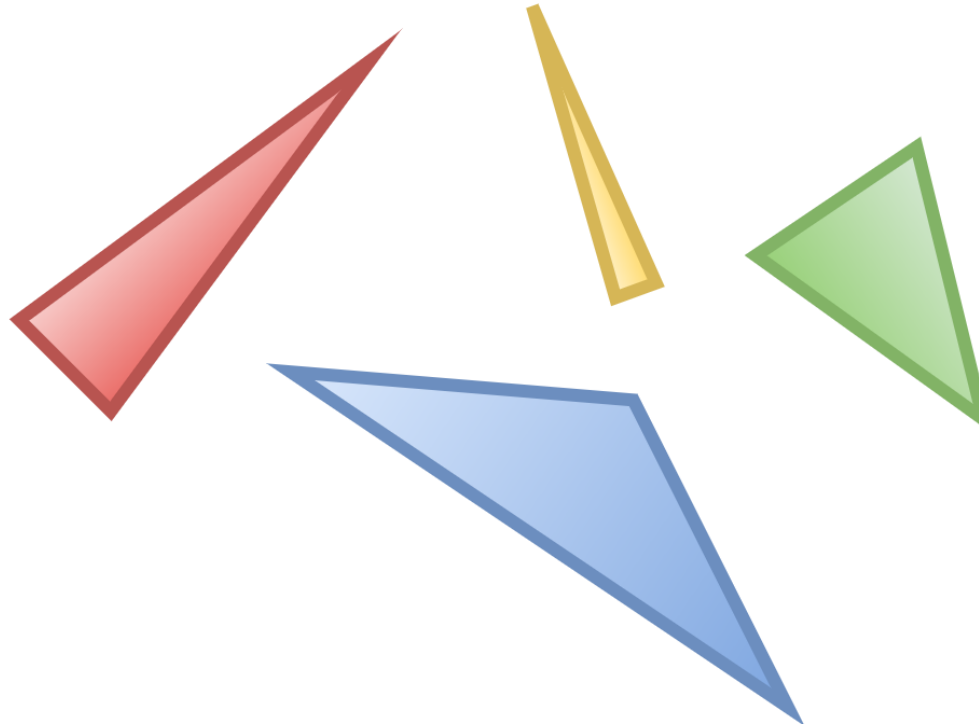
- The cost function depends on
 - The surface area of the current node's bounding box
 - The surface areas of the split bounding boxes
 - The number of triangles intersecting the right or left split bounding box



Construction

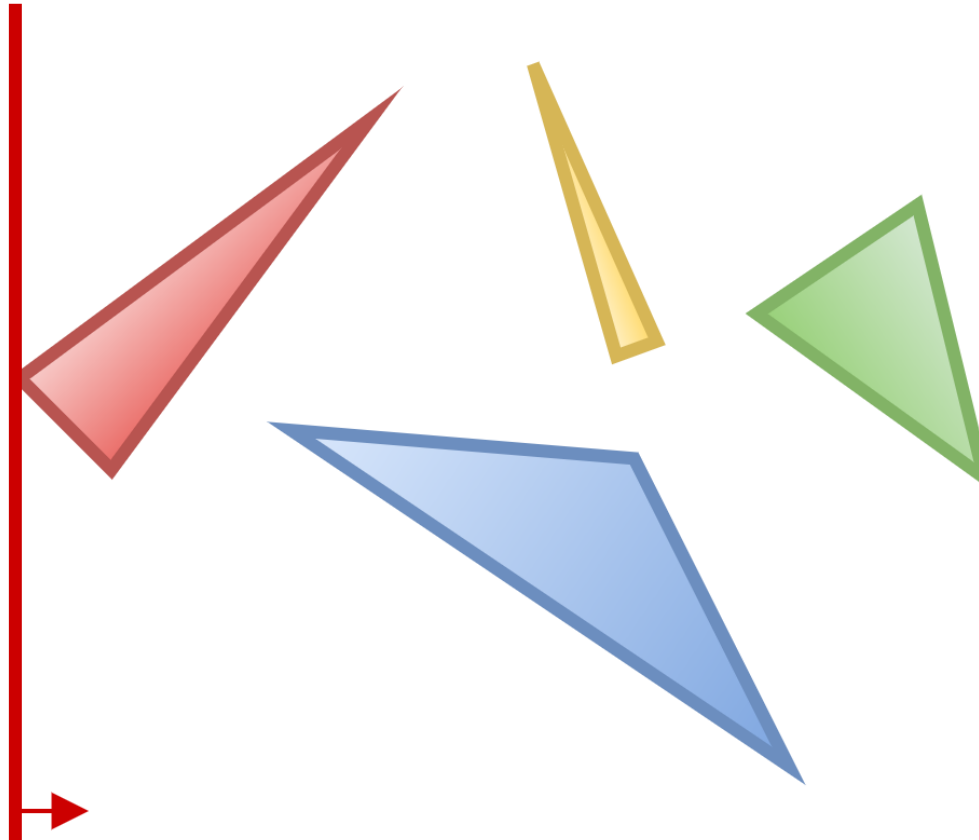
- SAH needs to evaluate the cost for each possible split
 - Perfect split will be at the begin or end of a triangle
 - $O(n)$ split candidates to evaluate per recursion step
 - Naive: Compute number of triangles in right and left child in $O(n)$, resulting in $O(n^2)$ overall
- Sweep-line based algorithm for $O(n \log^2 n)$
 - Sweep along dimension d
 - Regard begin and end of polygons as events ($O(n \log n)$ for sorting)
 - Compute the numbers of triangles and the associated minimum SAH cost incrementally in $O(n)$

Sweep-line-based Perfect Split



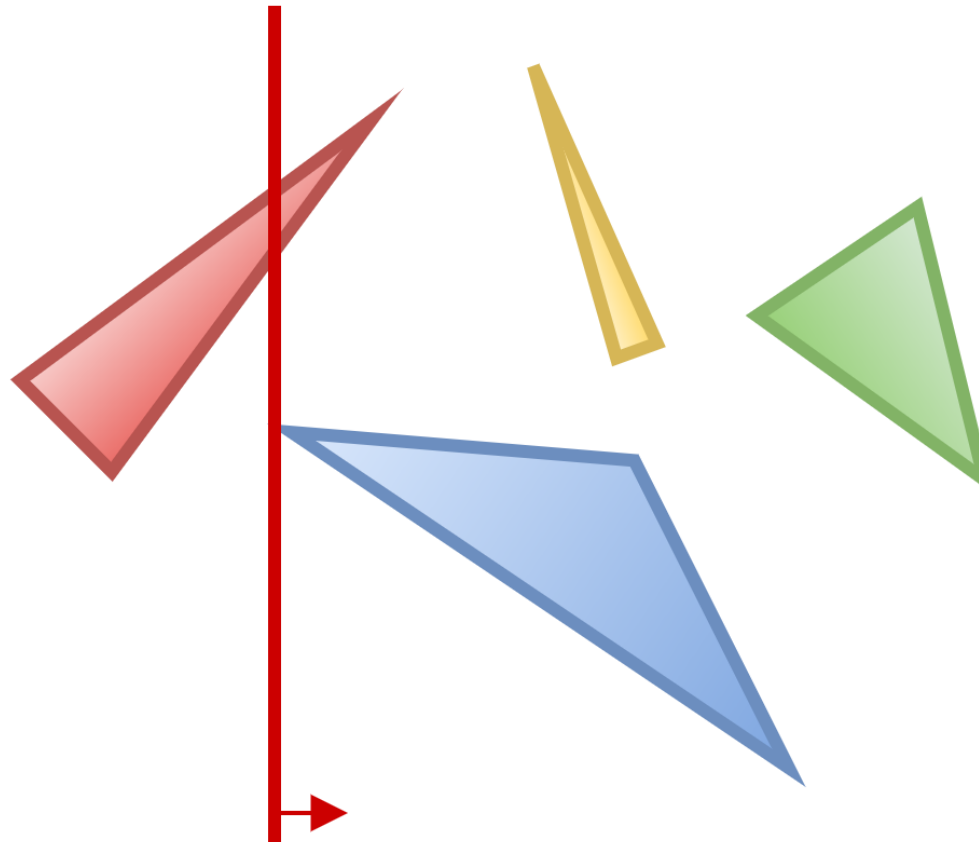
Triangles left of sweep line:	0
Triangles right of sweep line:	4

Sweep-line-based Perfect Split



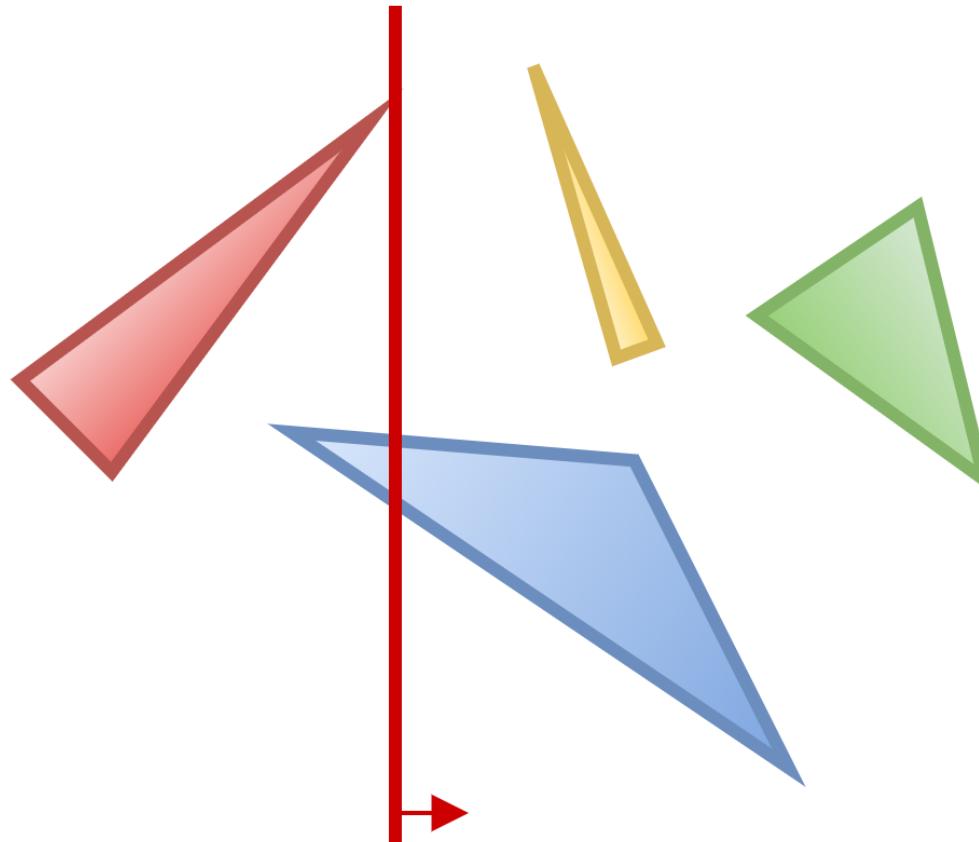
Triangles left of sweep line:	1
Triangles right of sweep line:	4

Sweep-line-based Perfect Split



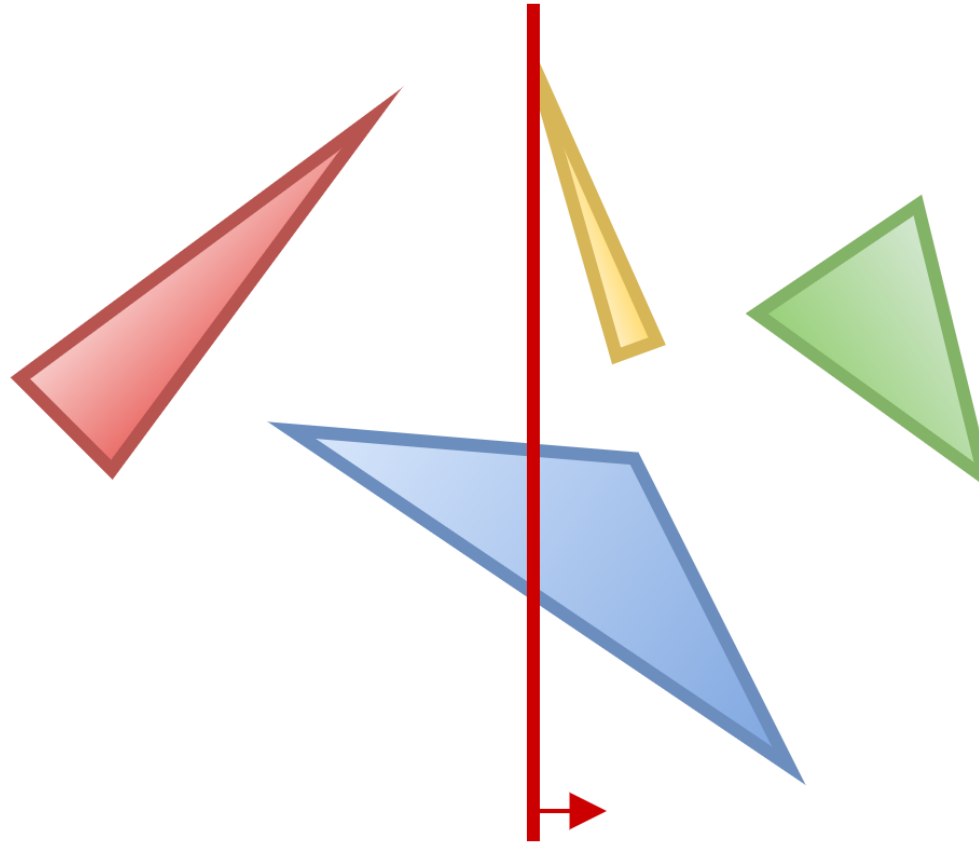
Triangles left of sweep line: 2
Triangles right of sweep line: 4

Sweep-line-based Perfect Split



Triangles left of sweep line:	2
Triangles right of sweep line:	3

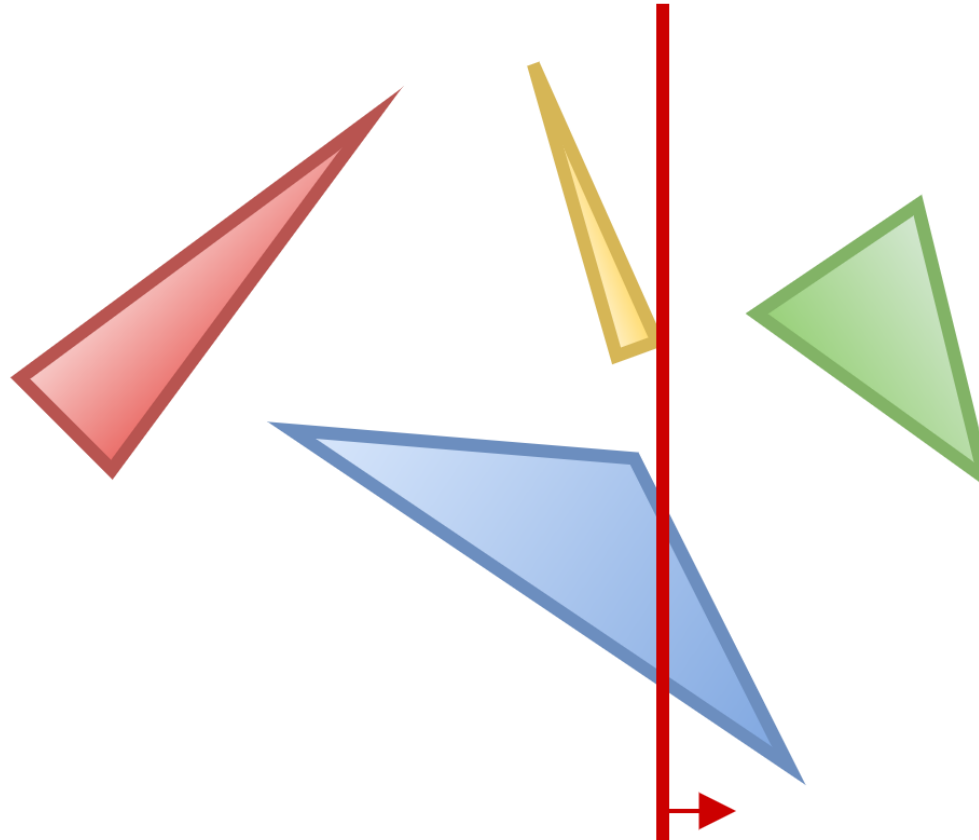
Sweep-line-based Perfect Split



Triangles left of sweep line: 3

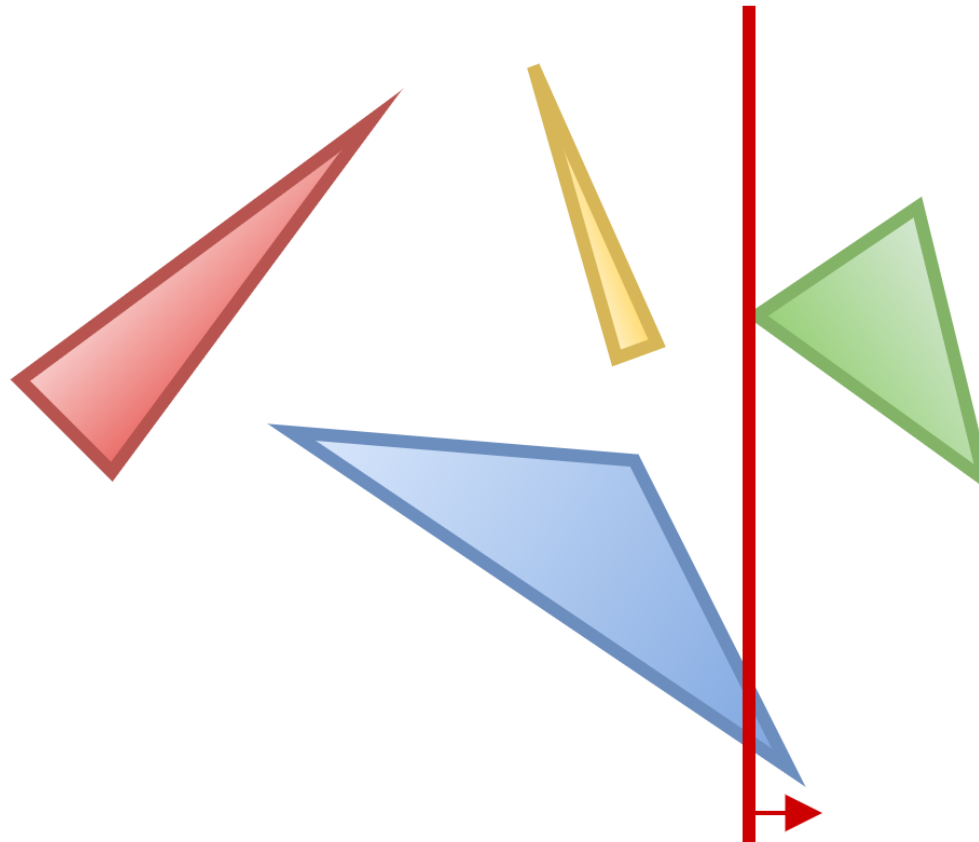
Triangles right of sweep line: 3

Sweep-line-based Perfect Split



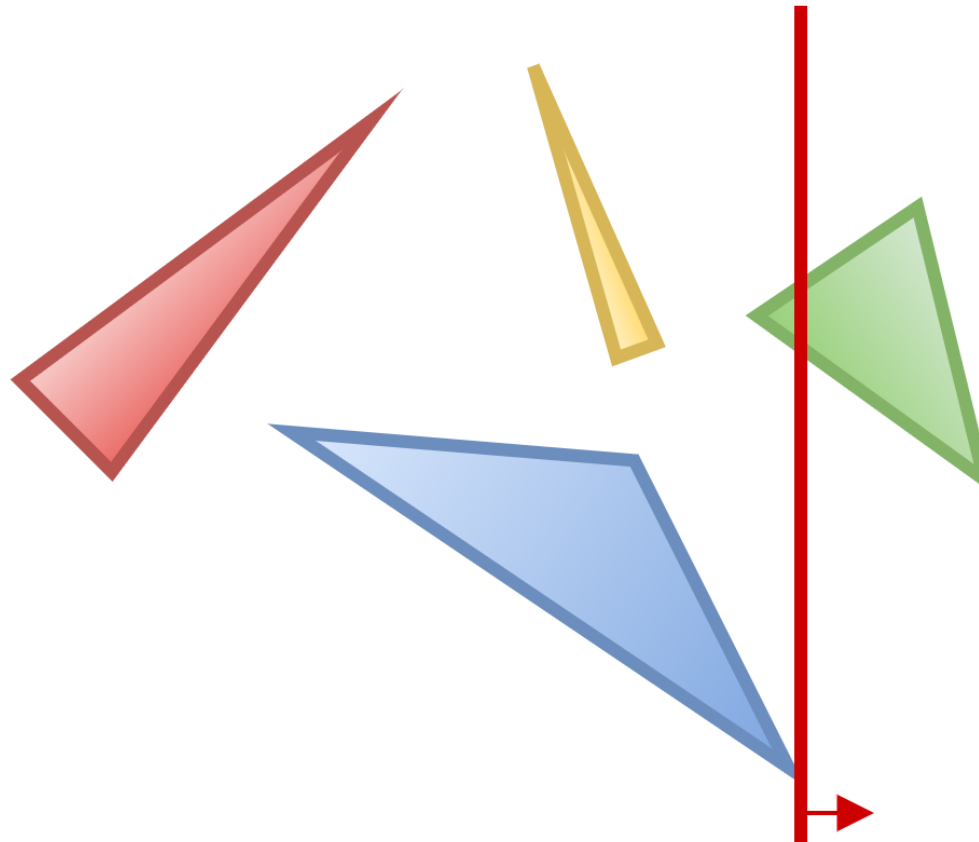
Triangles left of sweep line:	3
Triangles right of sweep line:	2

Sweep-line-based Perfect Split



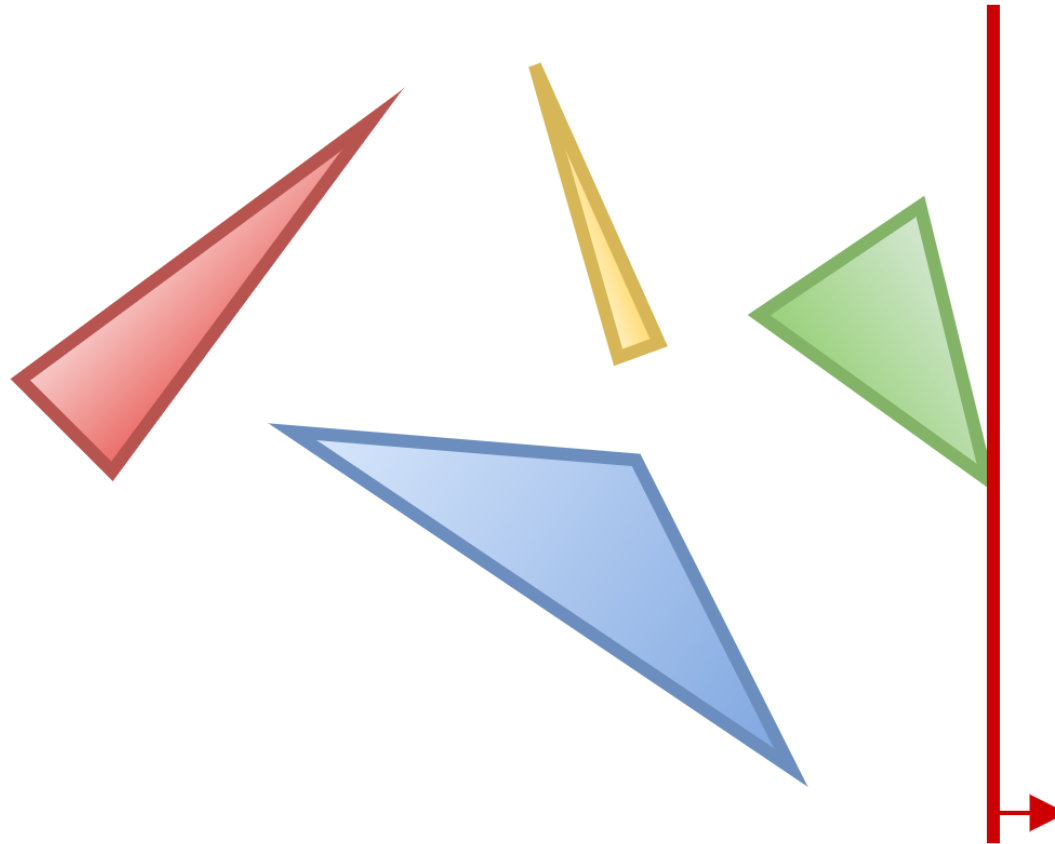
Triangles left of sweep line:	4
Triangles right of sweep line:	2

Sweep-line-based Perfect Split



Triangles left of sweep line:	4
Triangles right of sweep line:	1

Sweep-line-based Perfect Split



Triangles left of sweep line: 4
Triangles right of sweep line: 0

Construction

- Branch or leaf it?
- Branch if and only if the perfect split costs less than intersecting every triangle
- Since tree depth is in $O(\log n)$, we construct the *kd*-tree in $O(n \log^2 n)$
- By sorting only once and maintaining the sort order, construction could happen in $O(n \log n)$ [1]

Performance

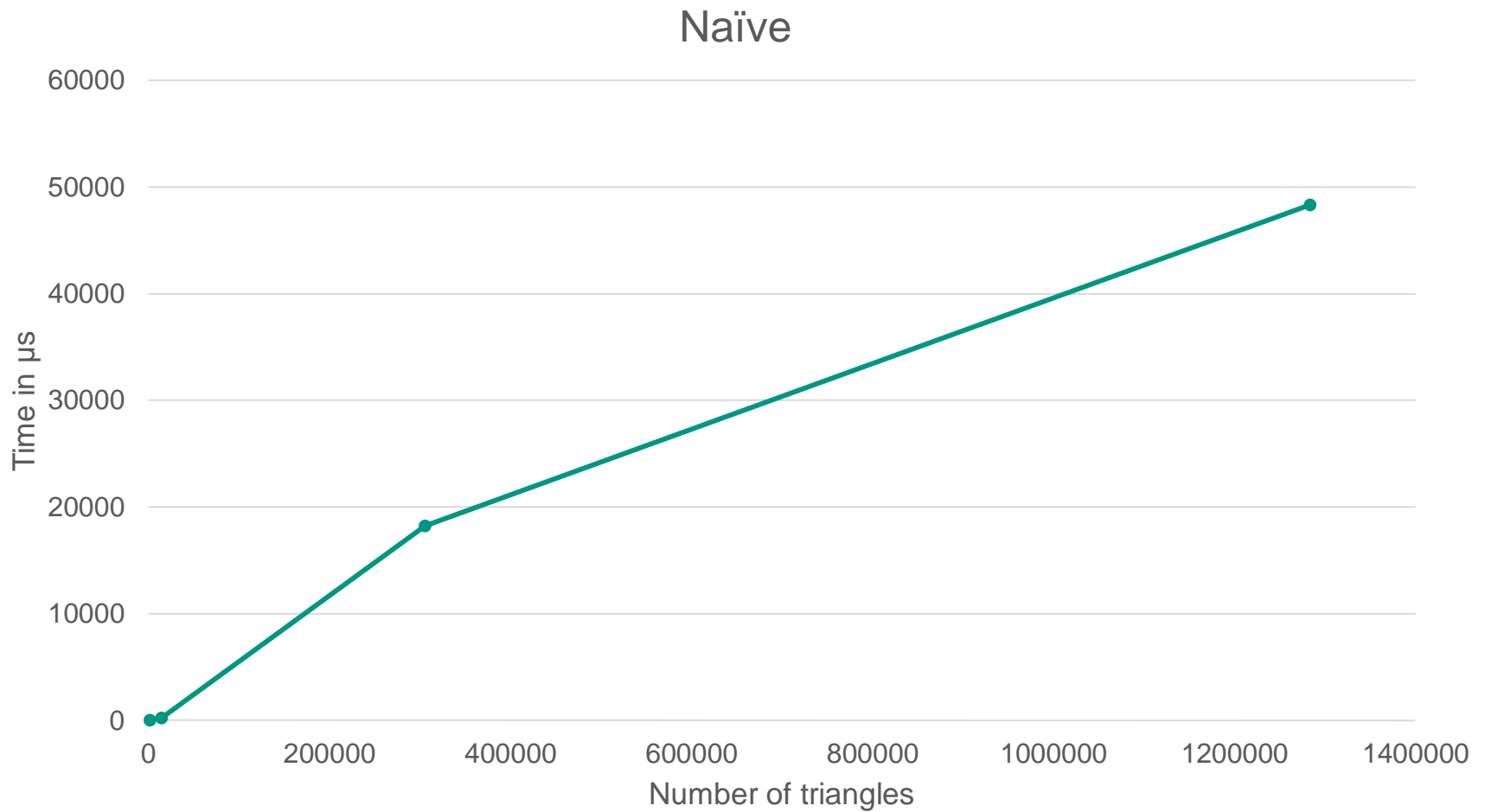
Performance

■ Benchmark system:

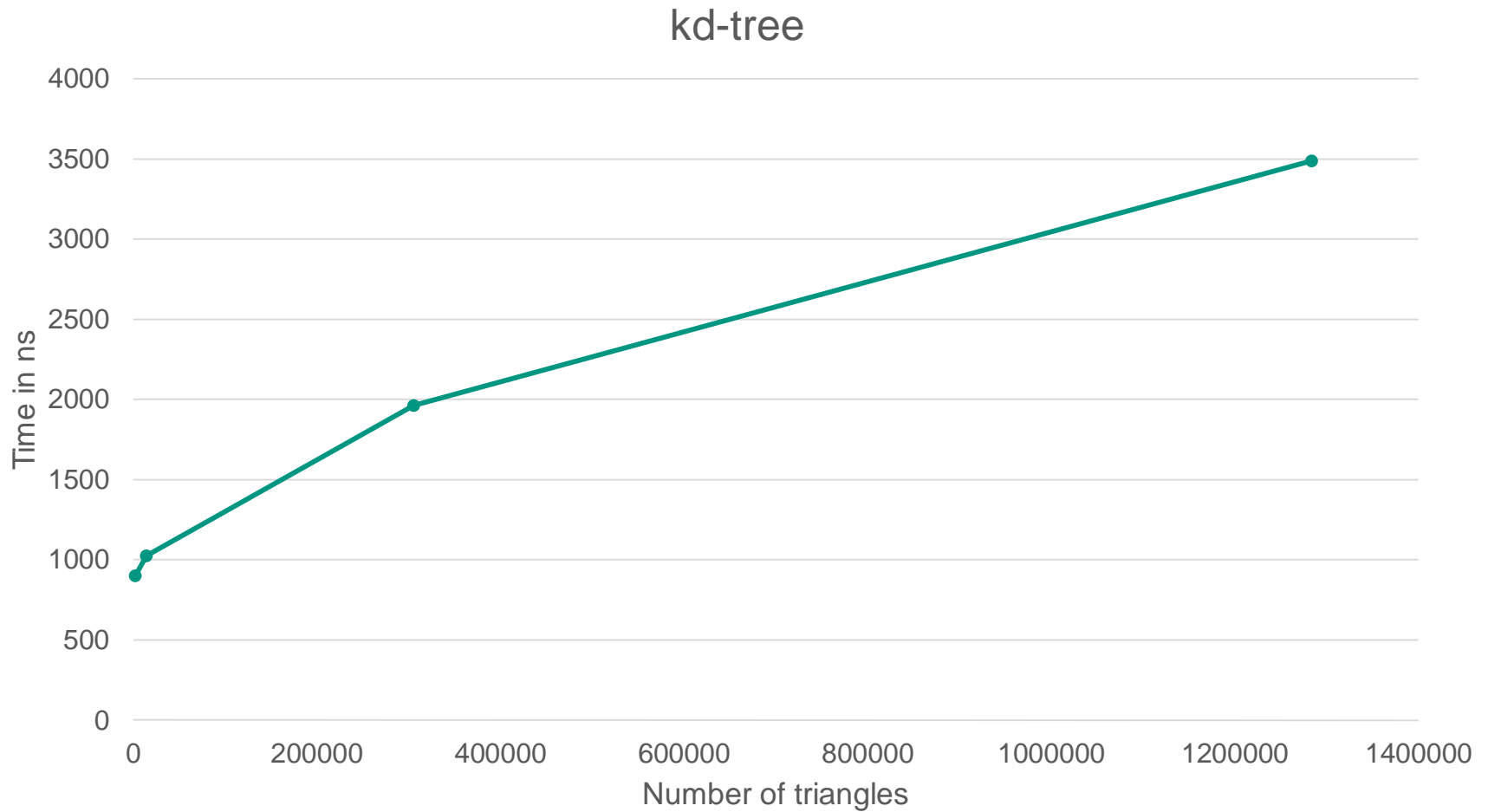
- Intel Xeon E3-1231v3 4x 3.40GHz
- Windows
- Java HotSpot 64-Bit Server VM (build 25.66-b18)
- Max. JVM heap size: 4 GB

number of triangles	naïve (μ s per random query)	kd-tree (ns per random query)
2,117	29.9	902.2
14,485	242.4	1025.0
305,662	18244.5	1961.5
1,283,858	48345.6	3489.4

Performance



Performance



References

- [1] On building fast kd-Trees for Ray Tracing, and on doing that in $O(N \log N)$ (2006), Ingo Wald, Vlastimil Havran