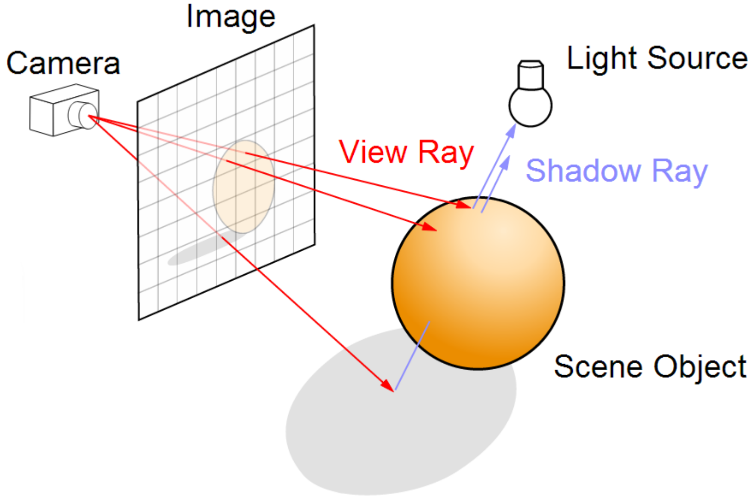
CMSC740
Advanced Computer Graphics

Fall 2025
Matthias Zwicker

Ray tracing pseudocode

```
rayTrace() {
  construct scene representation

  for each pixel
    ray = computePrimaryViewRay( pixel )
    hit = first intersection with scene
    color = shade( hit ) // using shadow ray
    set pixel color
}
```



Cost of naïve approach

- For each ray, the cost is linear in the number of primitives (triangles) in the scene
- Complexity $\Theta(n)$ per ray, n primitives
- Total cost: objects*rays

Example

- 1024x1024 image, 1000 triangles
- 10^9 ray triangle intersections

Ray tracing acceleration techniques

Fast intersections

Faster ray-object intersection tests

- Efficient numerical intersection algorithms
- Code optimization (vector instructions, multi-threading, GPU http://en.wikipedia.org/wiki/OptiX)

Fewer ray-object intersection tests

- „Acceleration structures“
- Bounding volume hierarchies
- Space subdivision

Generalized rays

- Beam tracing
  http://en.wikipedia.org/wiki/Beam_tracing
- Cone tracing
  http://en.wikipedia.org/wiki/Cone_tracing

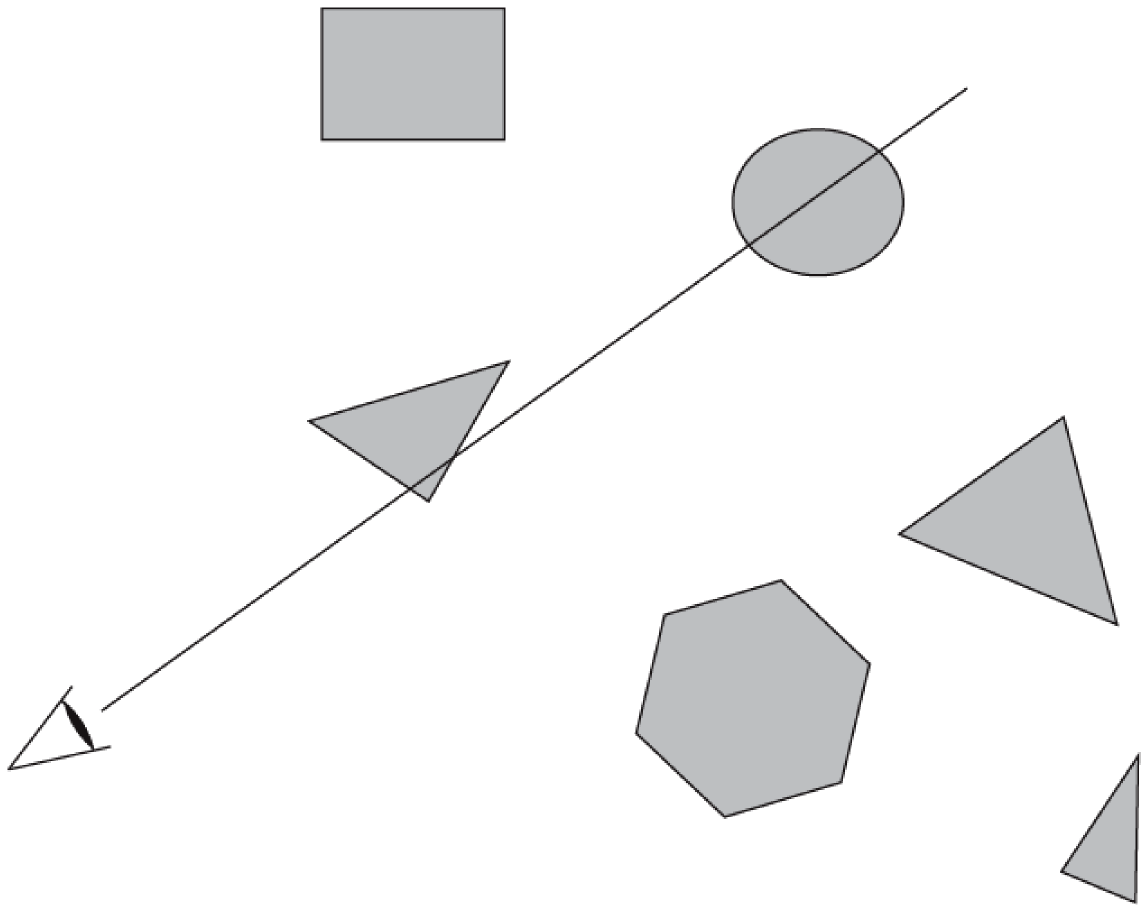
Acceleration structures

- Goal: “sub-linear” complexity
  - Number of intersection tests grows more slowly than proportional to number of primitives
  - Logarithmic complexity: $\Theta(\lg n)$
- Don’t touch every single object (i.e., triangle)

Spatial data structures

- Enable efficient operations on data organized in a metric space
  - „Metric space“: can make distance measurements
  - Operations: intersection tests, search queries based on proximity, etc.
- Applications
  - Ray tracing (rendering), acceleration structure = spatial data structure
  - Collision detection (phyiscs simulations)
  - Chemical simulations
  - Machine learning (nearest neighbor queries)
  - Data analysis
  - …
- Detailed background in ”Foundations of Multidimensional and Metric Data Structures”
  http://books.google.dk/books?id=KrQdmLjTSaQC

Acceleration structures: ideas?



Acceleration structures

- Two types

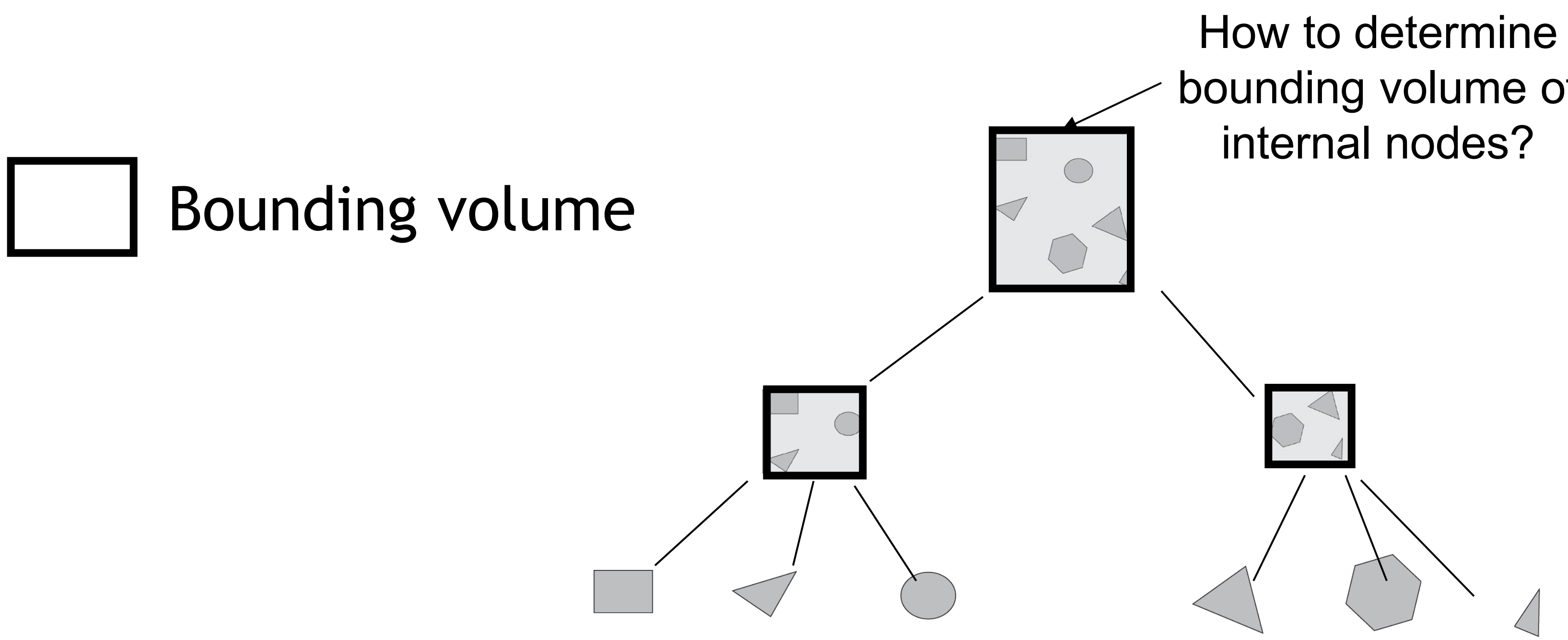
1. Object subdivision
   - Bounding volume hierarchies (BVHs)
2. Spatial subdivision
   - Uniform, hierarchical grids
   - Octrees
   - Binary space partitioning (BSP) trees, kd-trees

Object subdivision

- Hierarchies of groups of objects
- Groups are represented by their bounding volumes
  - Bounding volume: simple geometry that encloses all objects in a group, allows fast ray intersection computation (for example: axis aligned boxes, spheres, etc.)
- "Bounding volume hierarchies", BVH
  http://en.wikipedia.org/wiki/Bounding_volume_hierarchy
- Logarithmic complexity $\Theta(\lg n)$ to find intersection
  - Depth of BVH hierarchy is logarithmic in terms of number of objects n

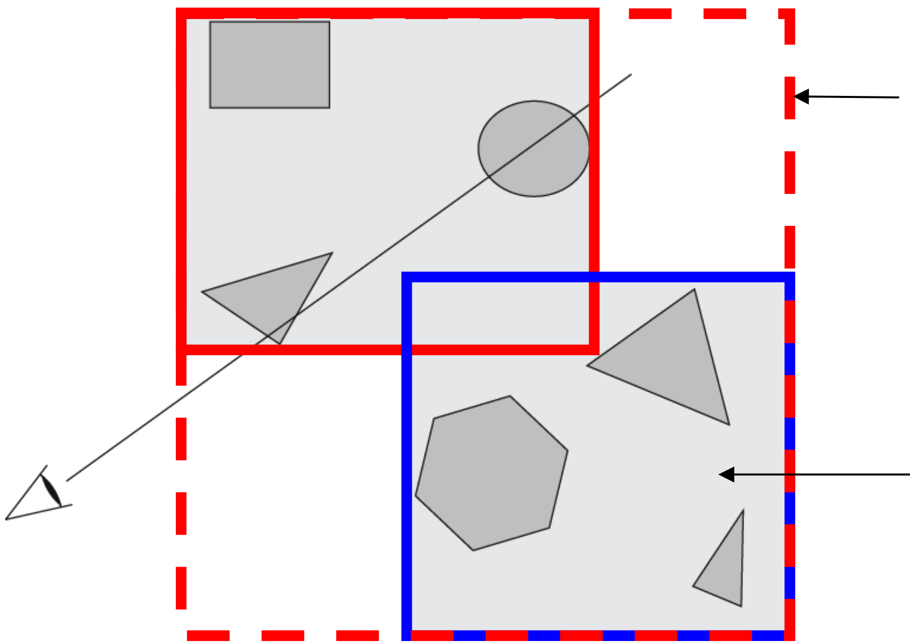
Bounding volume hierarchies

- Tree structure
- Leave nodes contain objects (e.g. triangles)
- Each internal node is bounding volume



Bounding volume hierarchies

- Bounds of each bounding volume contain all objects in its subtree

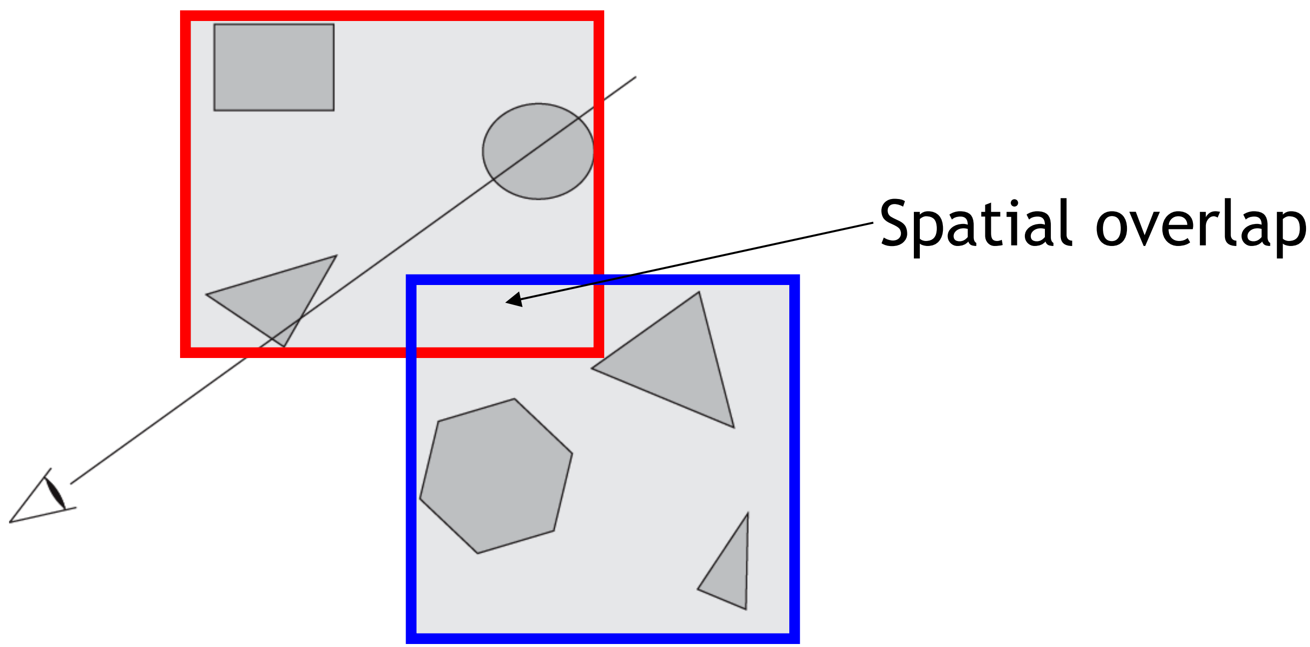


Parent node bounding volume: **required** to enclose children, **should be** as small as possible

Leave node encloses all objects in it

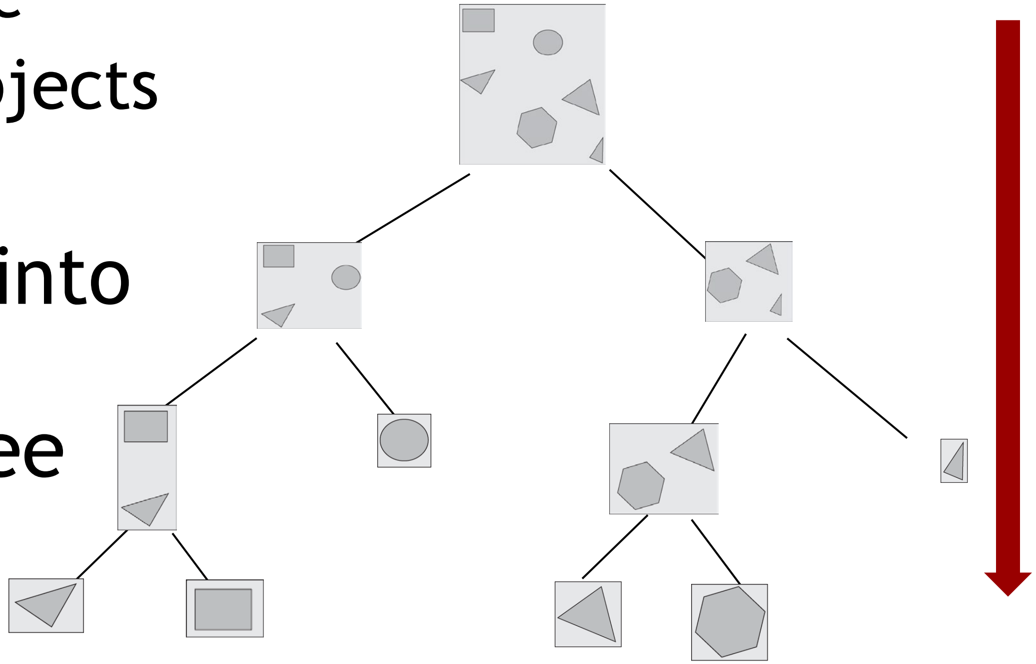
Bounding volume hierarchies

- Subtrees can overlap spatially
  - Not all objects within the bounding volume of a node need to be in its own subtree
- Subtrees are not ordered in any way



BVH construction

- Partitioning objects along coordinate axes in a top-down fashion, starting at root
  - Other strategies (e.g., bottom-up) possible
    http://en.wikipedia.org/wiki/Bounding_volume_hierarchy
- Partitioning strategies, alternatives
  - In geometric middle
  - Into equal nr. of objects
  - Equal surface area
- Partitioning nodes into two children results in binary tree



BVH intersection

- If bounding volume of node is not intersected by a ray, none of the objects in its subtree are
  - Subtree can be pruned (ignored) during intersection testing
- If node is intersected, all children have to be tested for intersections recursively

BVH intersection

- Types of bounding volumes
  http://en.wikipedia.org/wiki/Bounding_volume
  - (Axis aligned) bounding boxes (AABB)
  - Bounding spheres
  - Bounding anything
- BVH with axis aligned bounding boxes (AABB) are popular because of efficient intersection testing

BVHs are always binary trees

A. True

B. False

Leaf nodes in a BVH always contain a single object (triangle)

A. True

B. False

Today: acceleration structures

- Introduction
- Two types

1. Object subdivision
   - Bounding volume hierarchies (BVHs)
2. Spatial subdivision
   - Uniform, hierarchical grids
   - Octrees
   - Binary space partitioning (BSP) trees, kd-trees

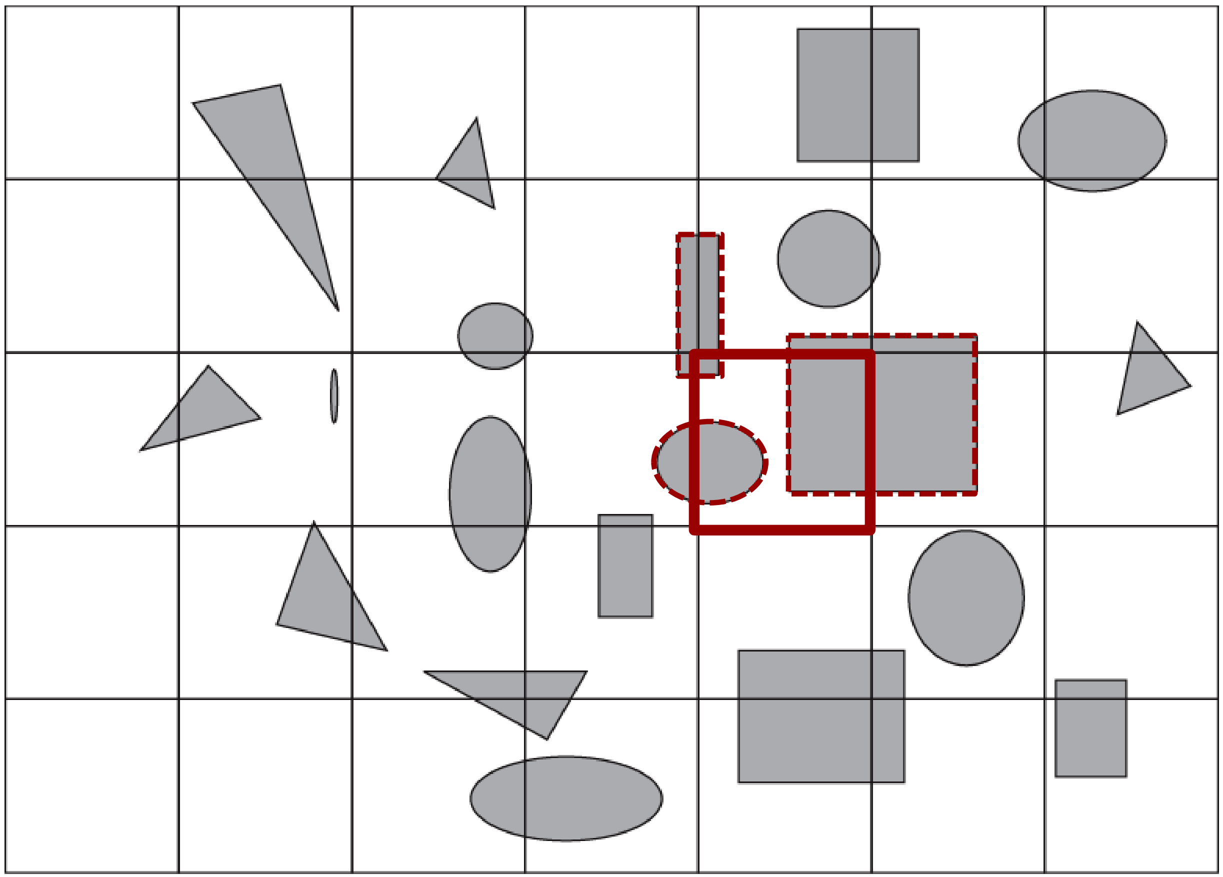
Spatial subdivision

Main idea

- Partition space into non-overlapping cells
- Each cell stores reference to all objects that overlap it

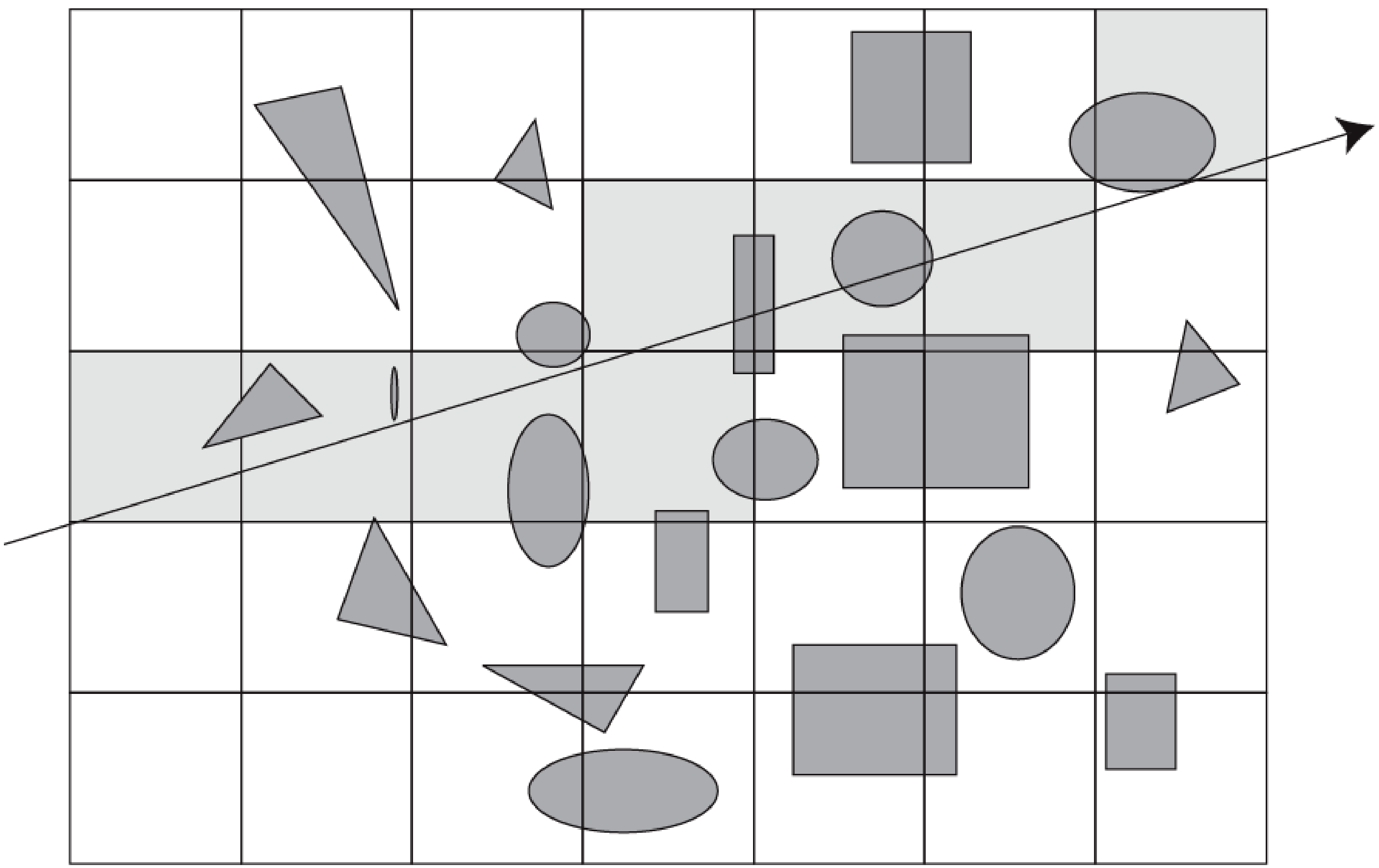
Uniform grid

- Each cell stores ref. to all objects in it



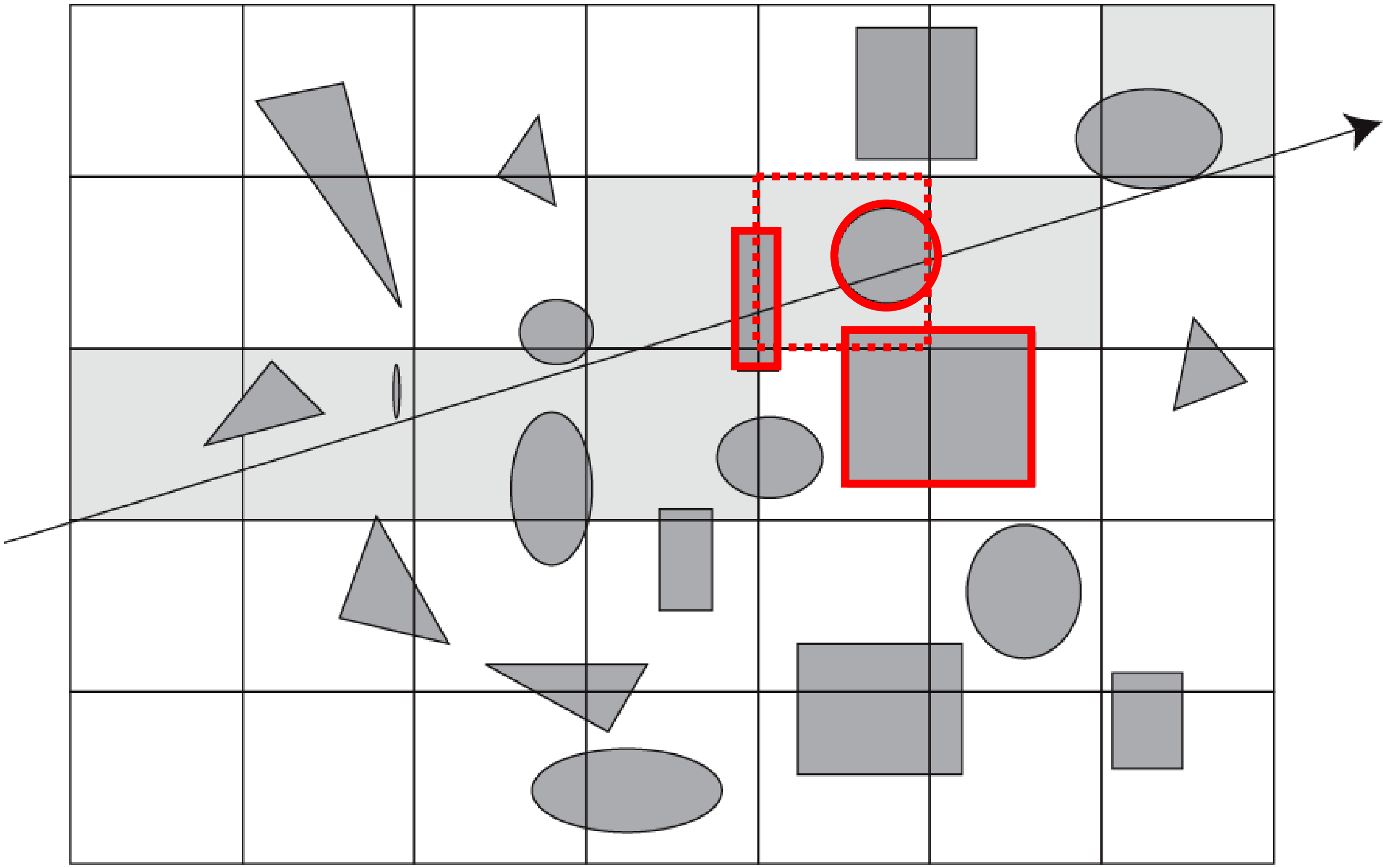
Uniform grid

- Traverse grid along ray

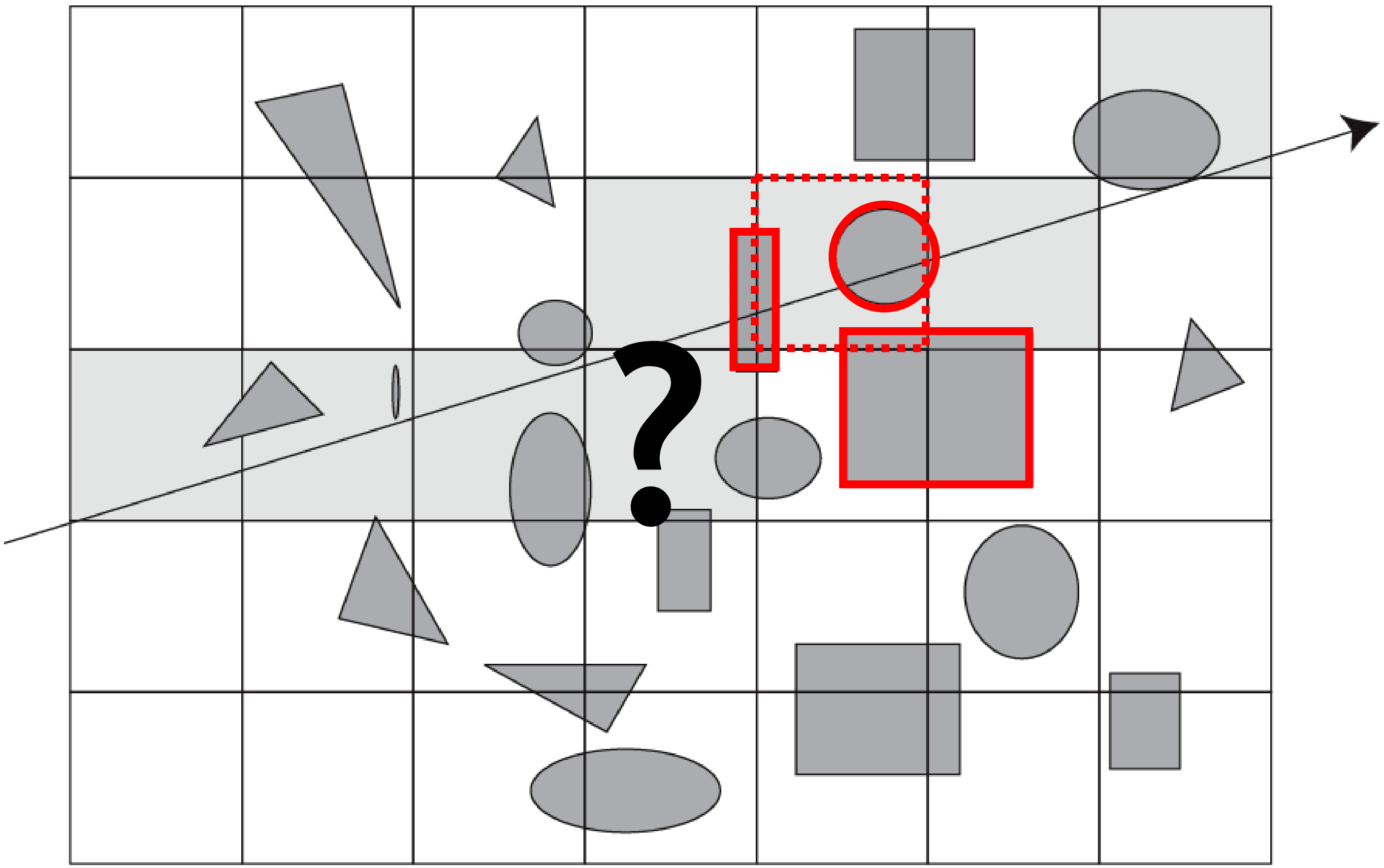


Uniform grid

- For each cell, intersect all objects in cell

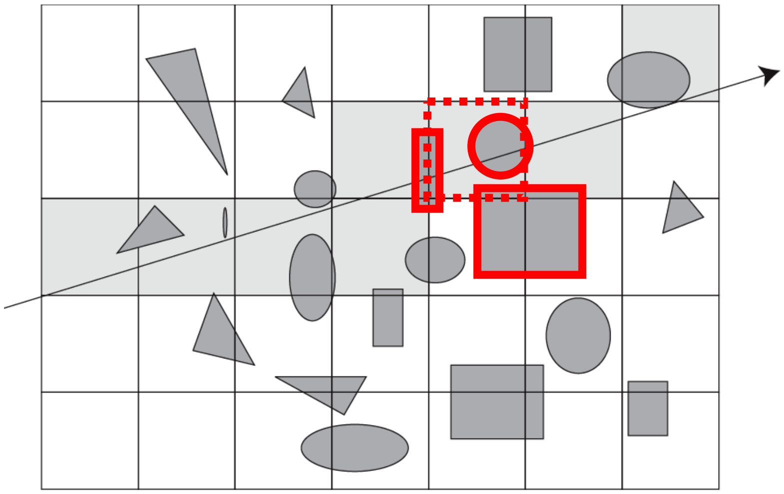


Uniform grid: disadvantages?

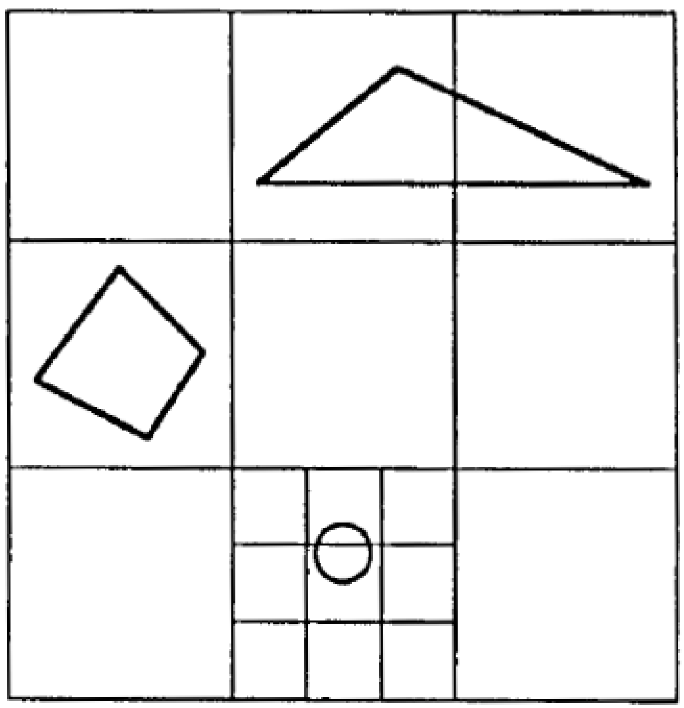


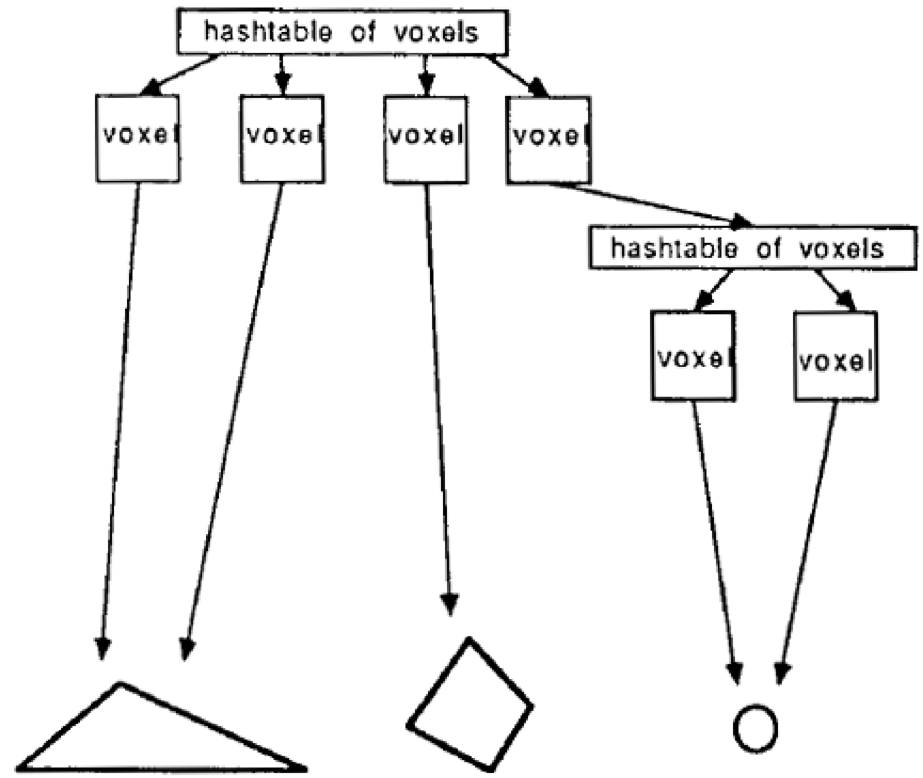
Uniform grid

- Advantages
  - Can traverse ray along grid (front to back)
  - Can stop as soon as a hit is found
- Disadvantages
  - "Teapot in a stadium" problem: no good uniform grid size
  - Potentially intersect same object multiple times



Hierarchical grid



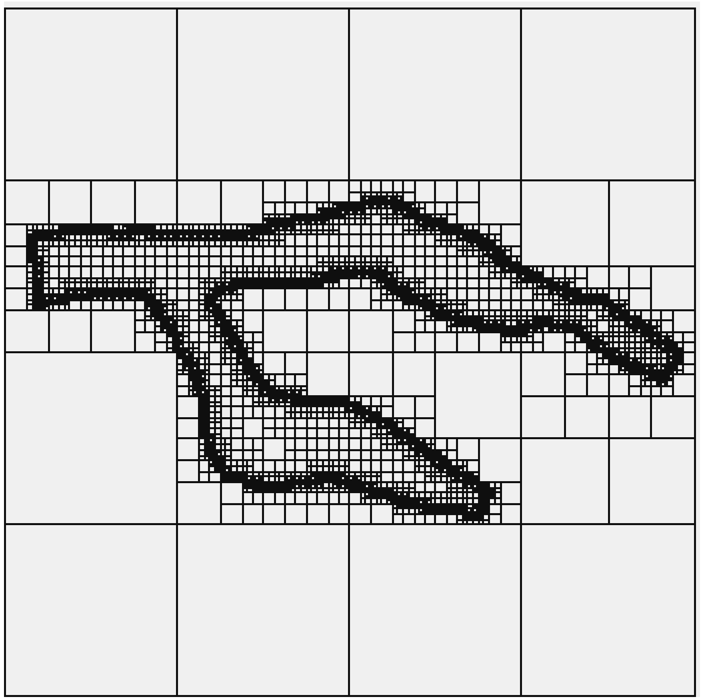


"Adaptive voxel subdivision for ray tracing", 1989

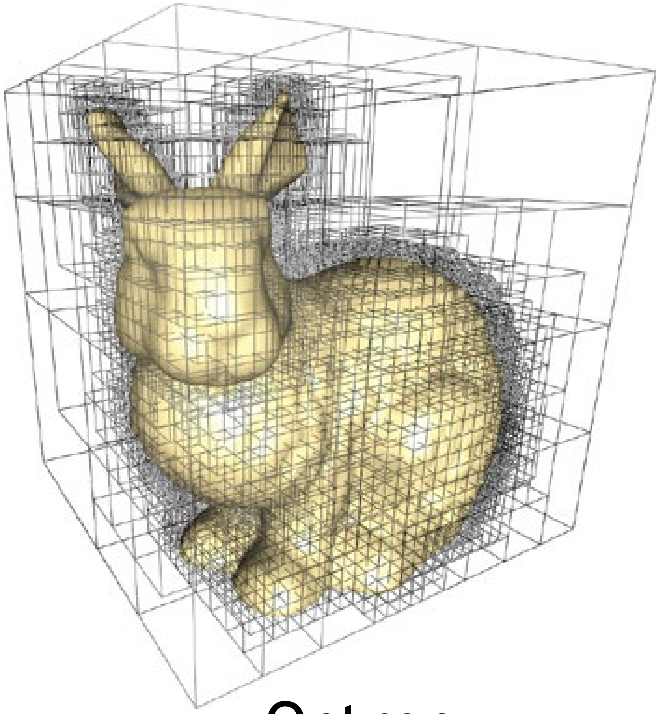
Octree

http://en.wikipedia.org/wiki/Octree

- Special case of hierarchical grid
- Analogous to quadtree in 2D
- Recursively split each cubic cell (at its center) into 8 equally sized cubic cells



Quadtree



Octree

Octrees are binary trees

A. True

B. False

Octrees implement the best possible space partitioning because space is divided into equally sized child nodes

A. True

B. False

Binary space partitioning (BSP) trees

http://en.wikipedia.org/wiki/Binary_space_partitioning

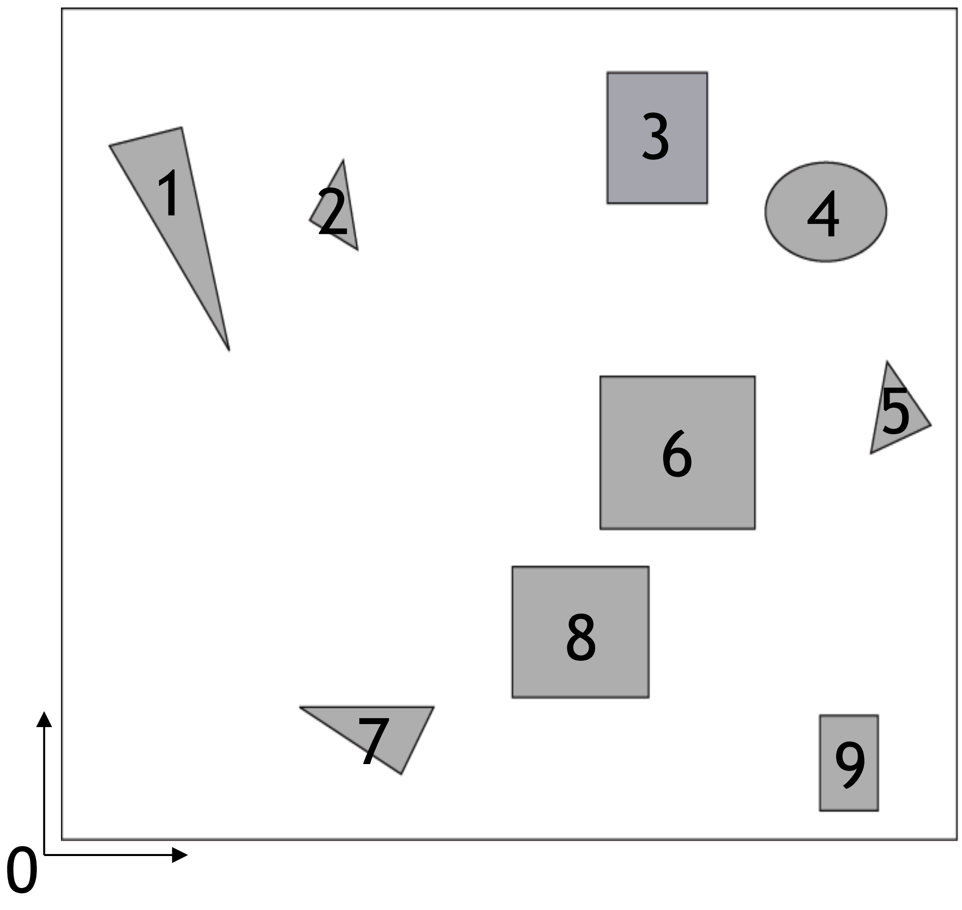
- Main idea
  - Recursively divide space into two parts using dividing planes (with arbitrary position, orientation)
- Special case: k-d-trees
  - Dividing planes are axis aligned

  http://en.wikipedia.org/wiki/Kd-tree

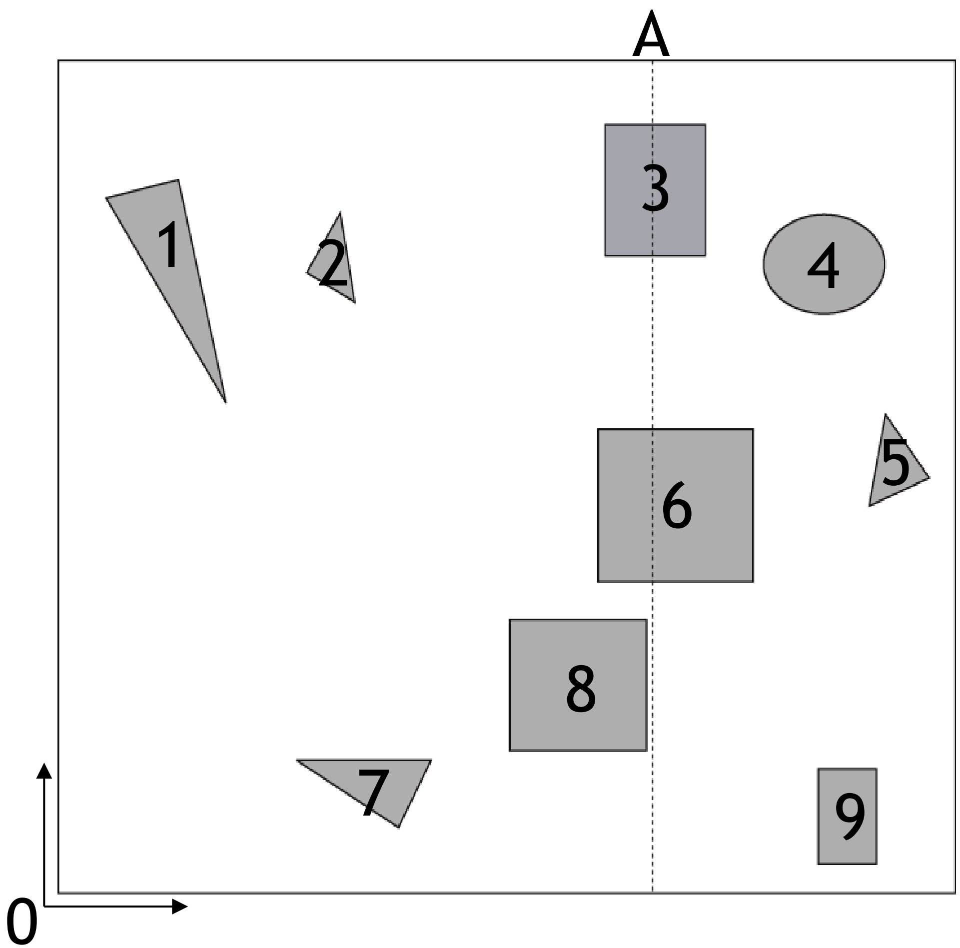
k-d tree example

- Stopping criterion: subdivide until fewer than 3 objects in node
- Convention for children in binary tree
  - Left child “below” split plane (smaller coordinates along split axis)
  - Right child “above” split plane (larger coordinates along split axis)
- Typically, cycle through splitting axis from one hierarchy level to next

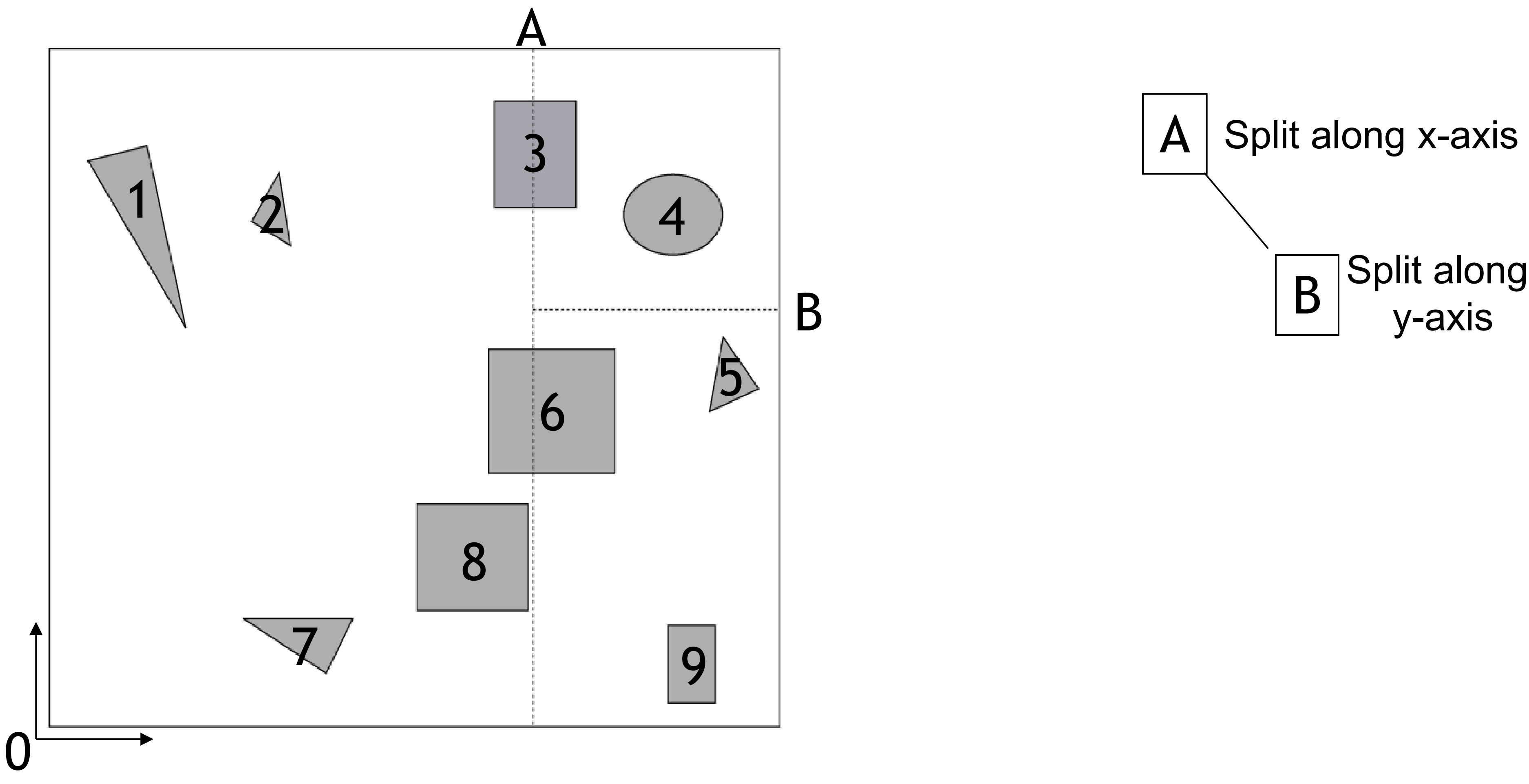
k-d tree example



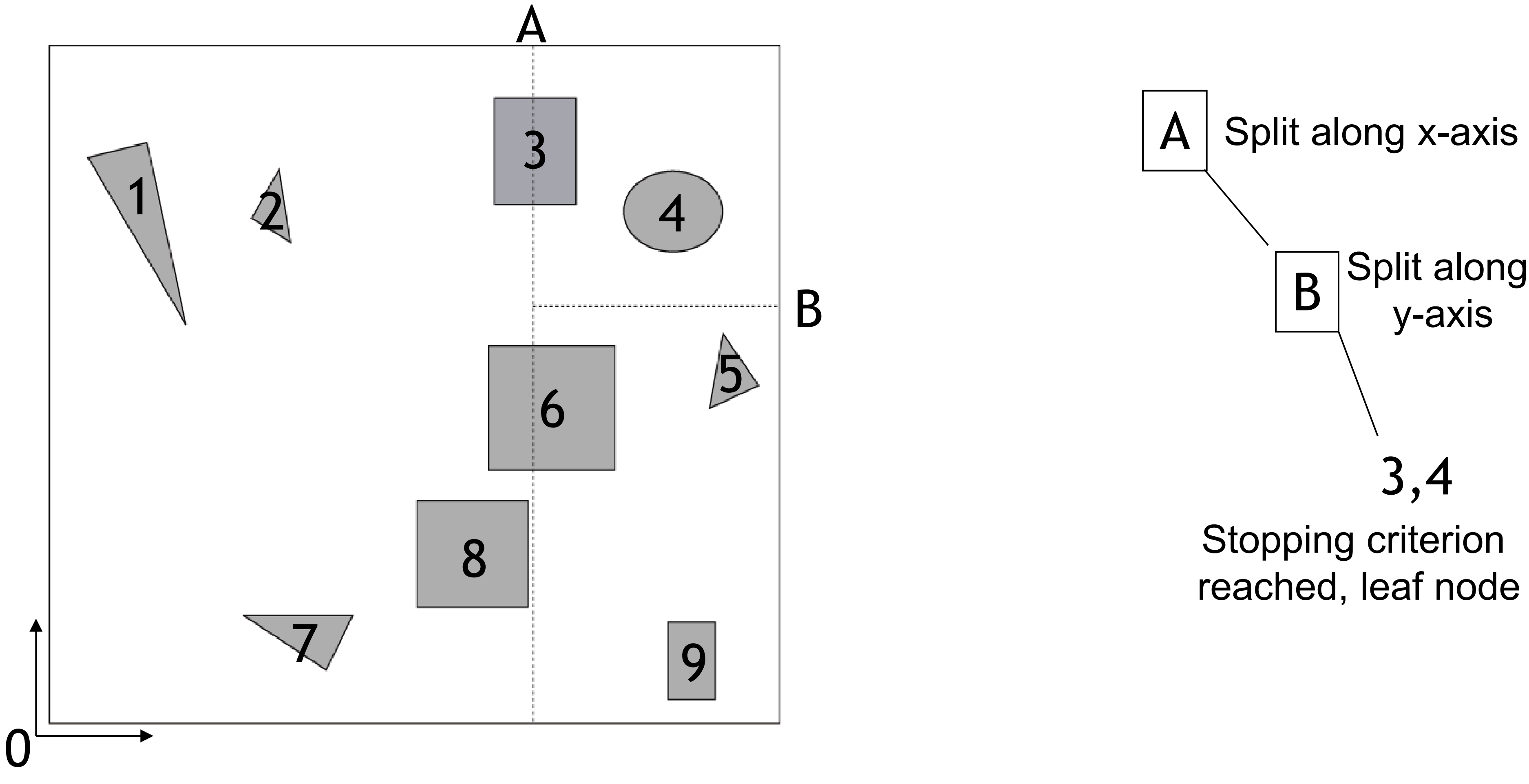
k-d tree example



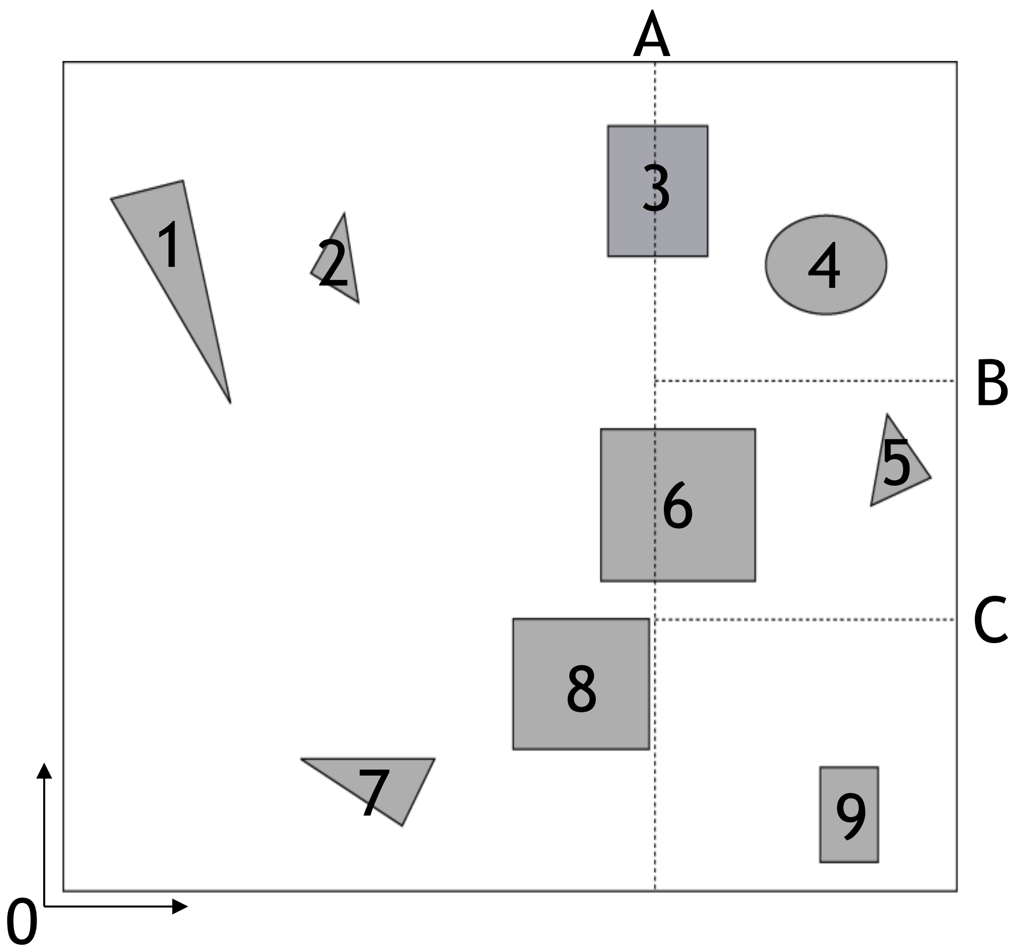
k-d tree example

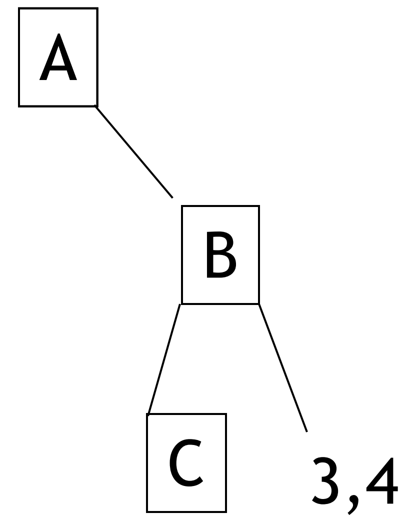


k-d tree example

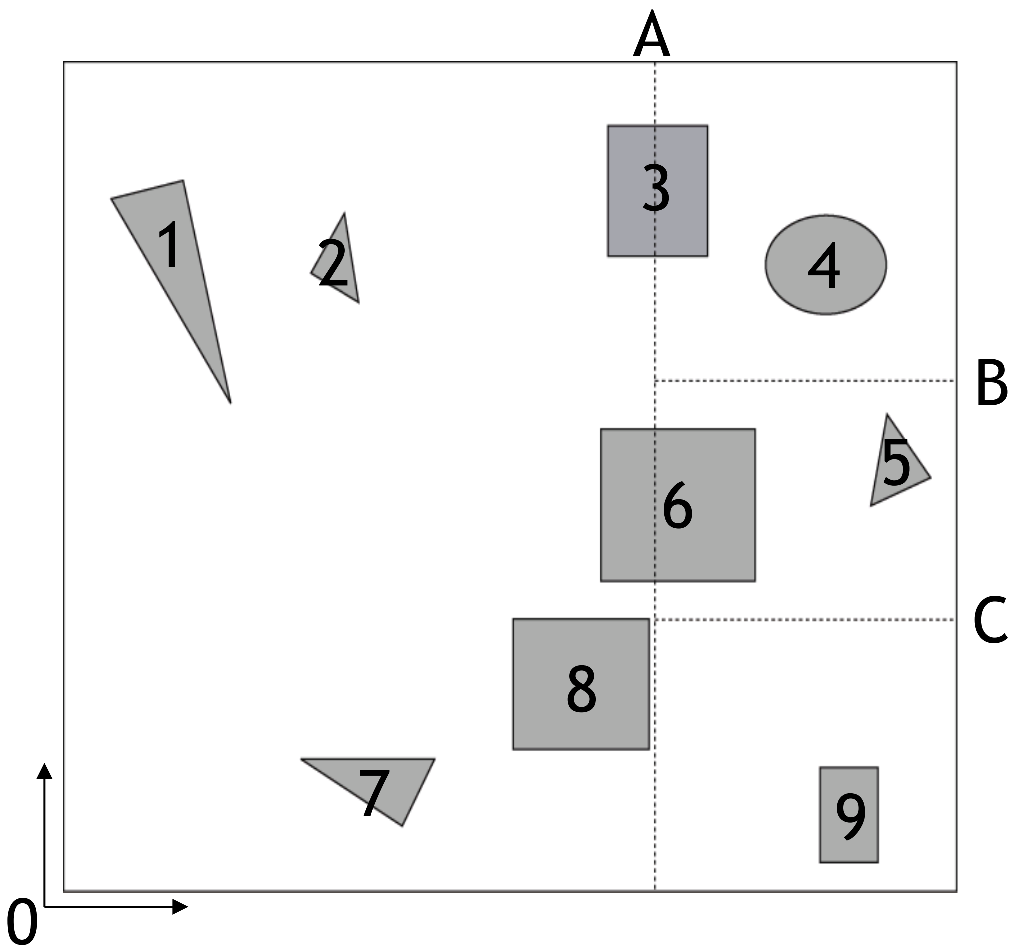


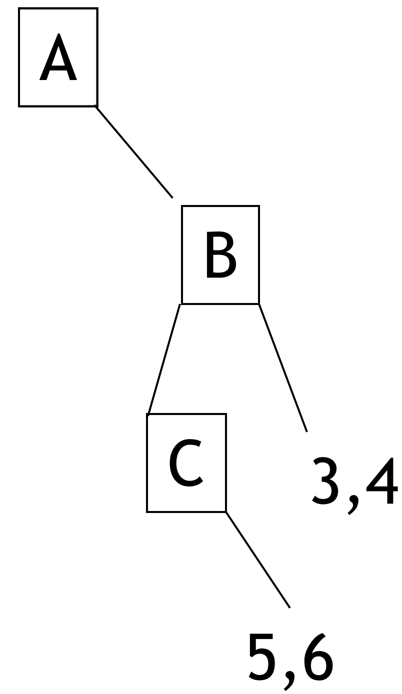
k-d tree example



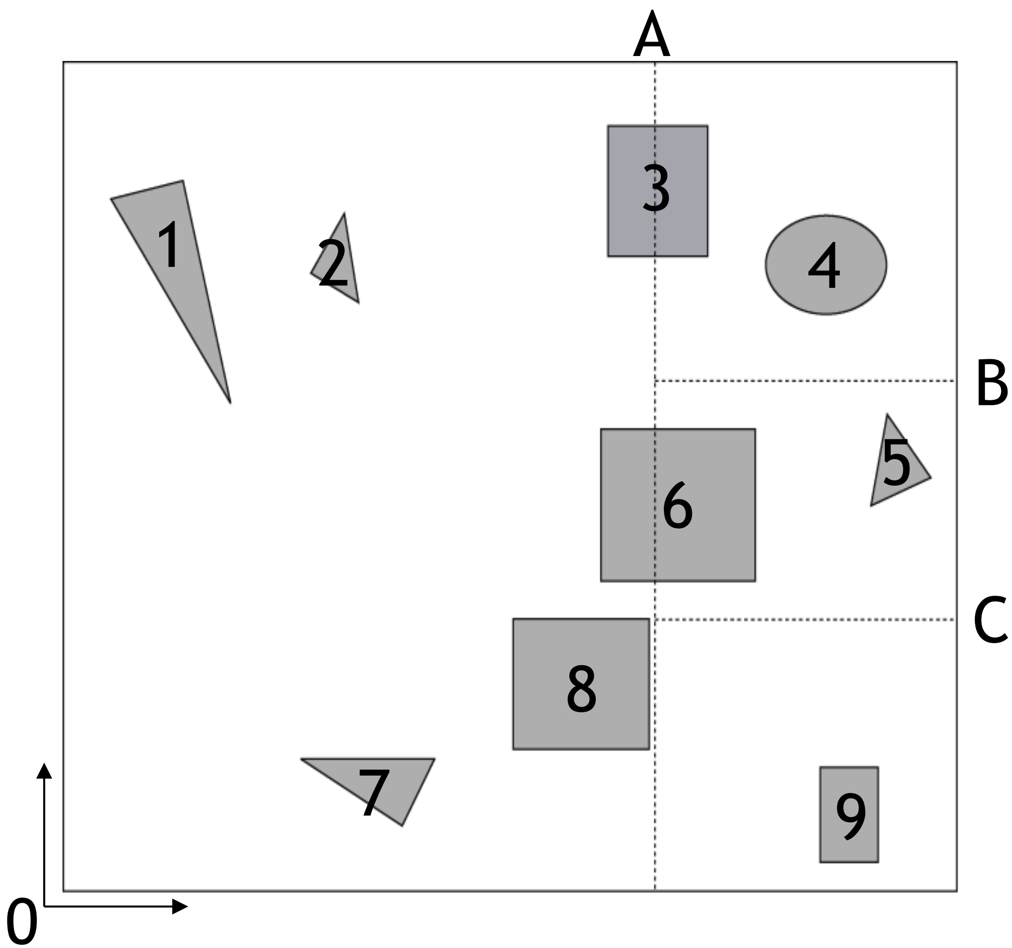


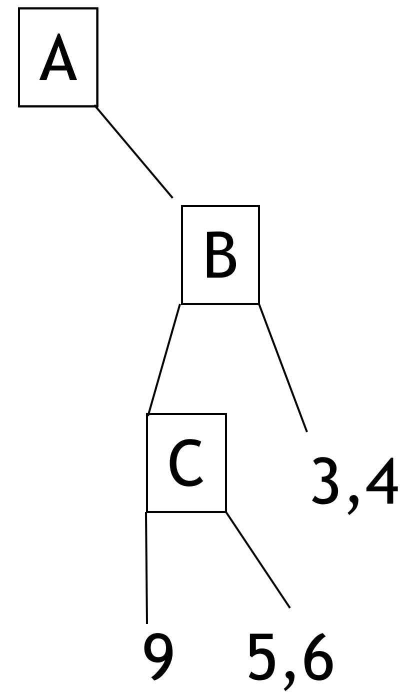
k-d tree example



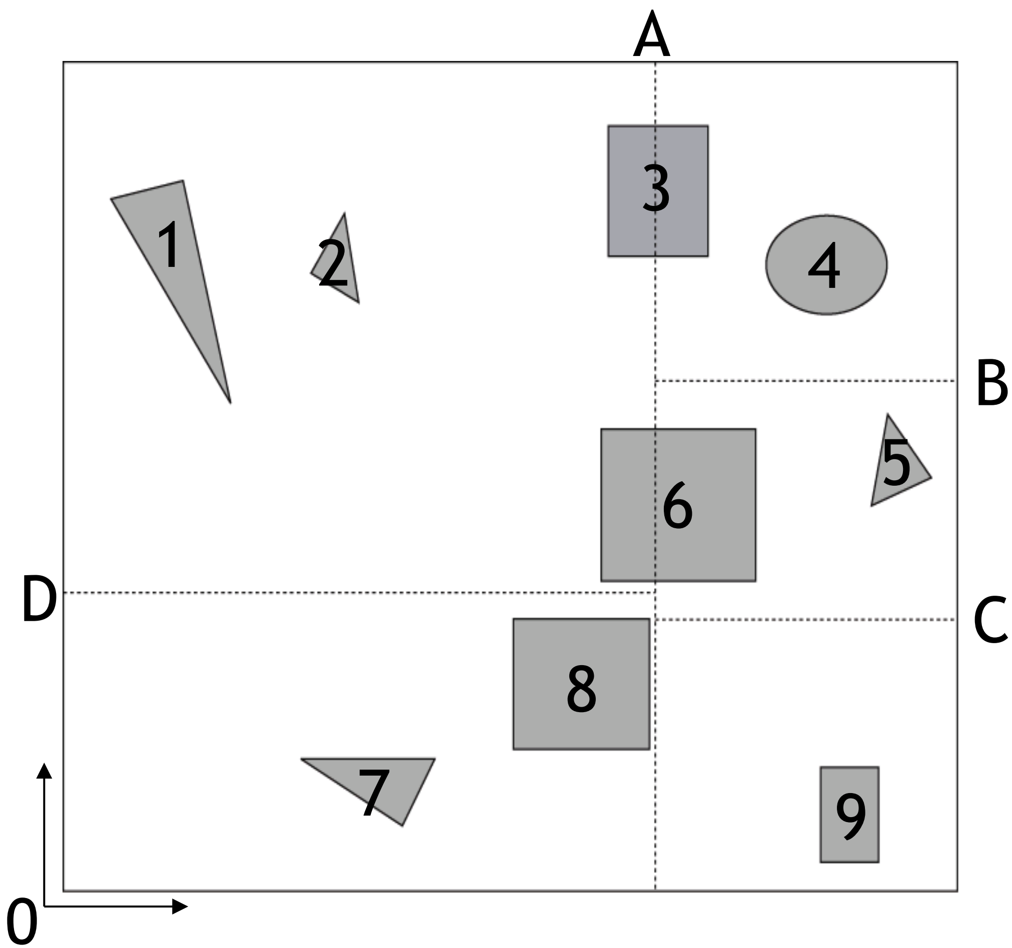


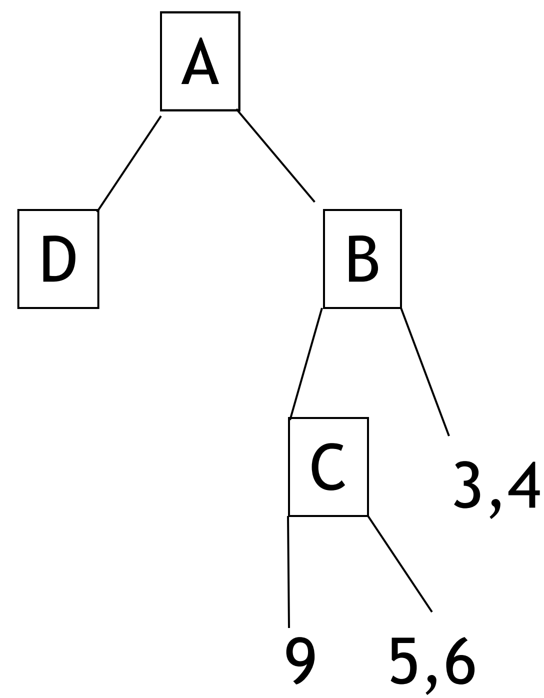
k-d tree example



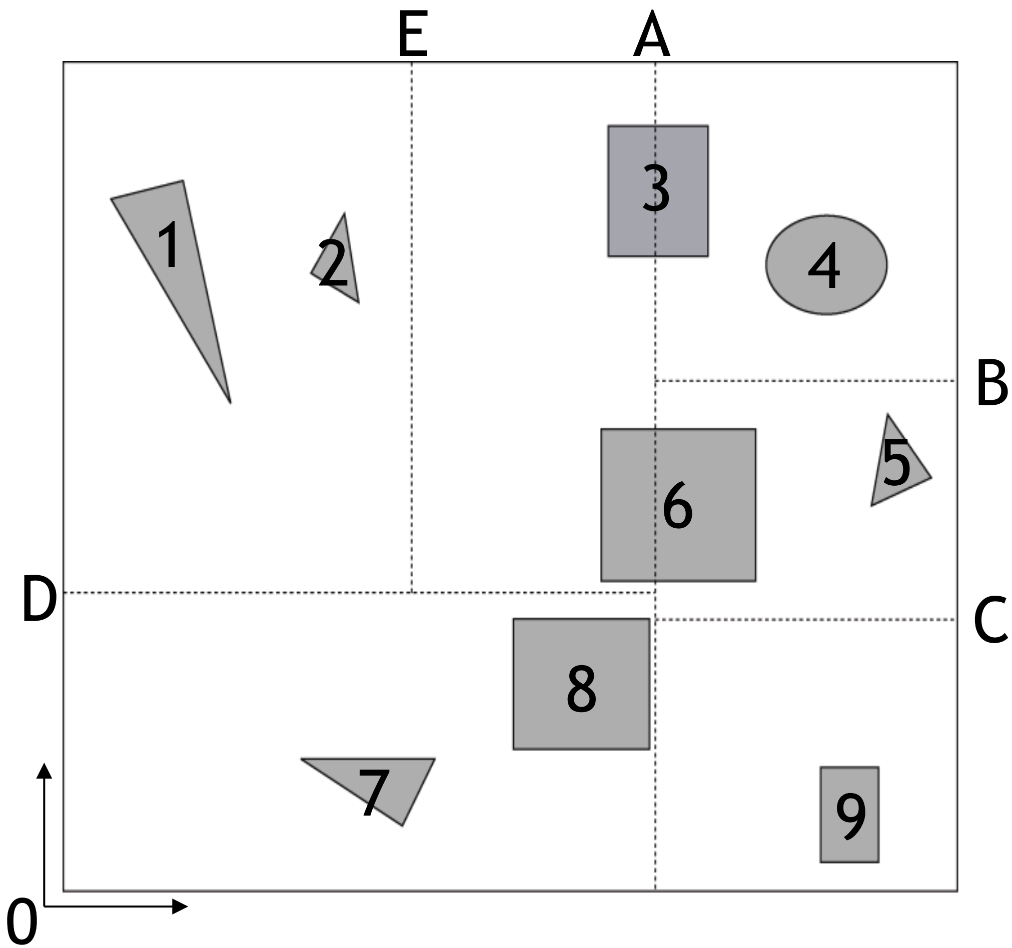


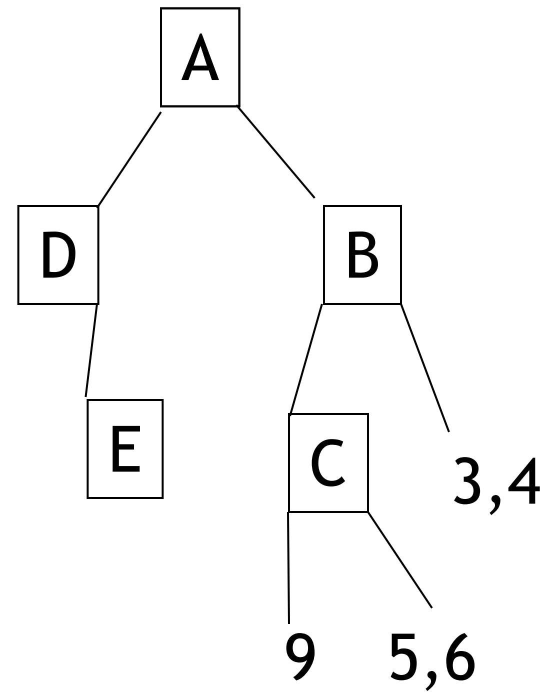
k-d tree example



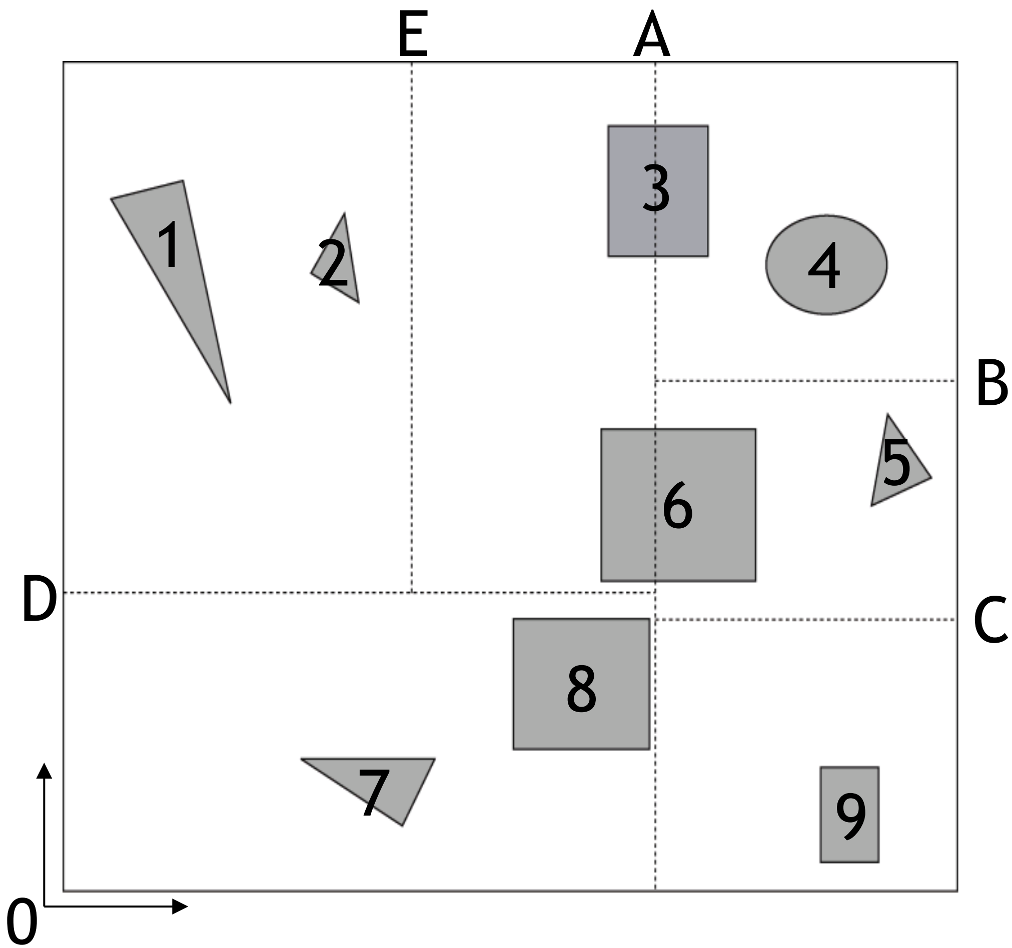


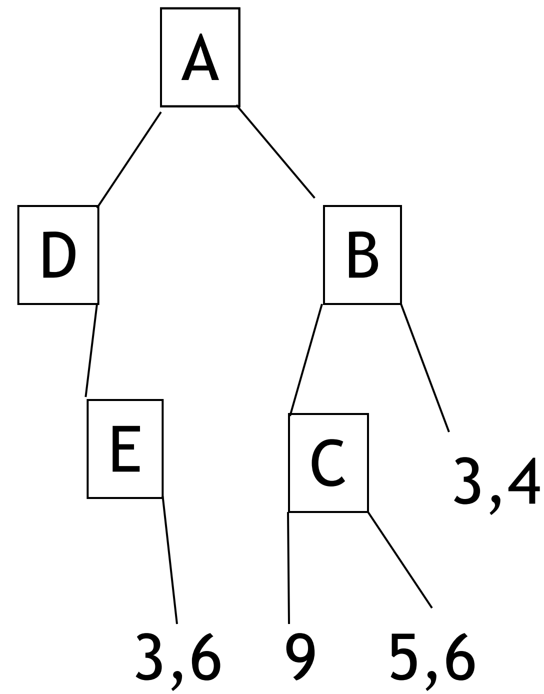
k-d tree example



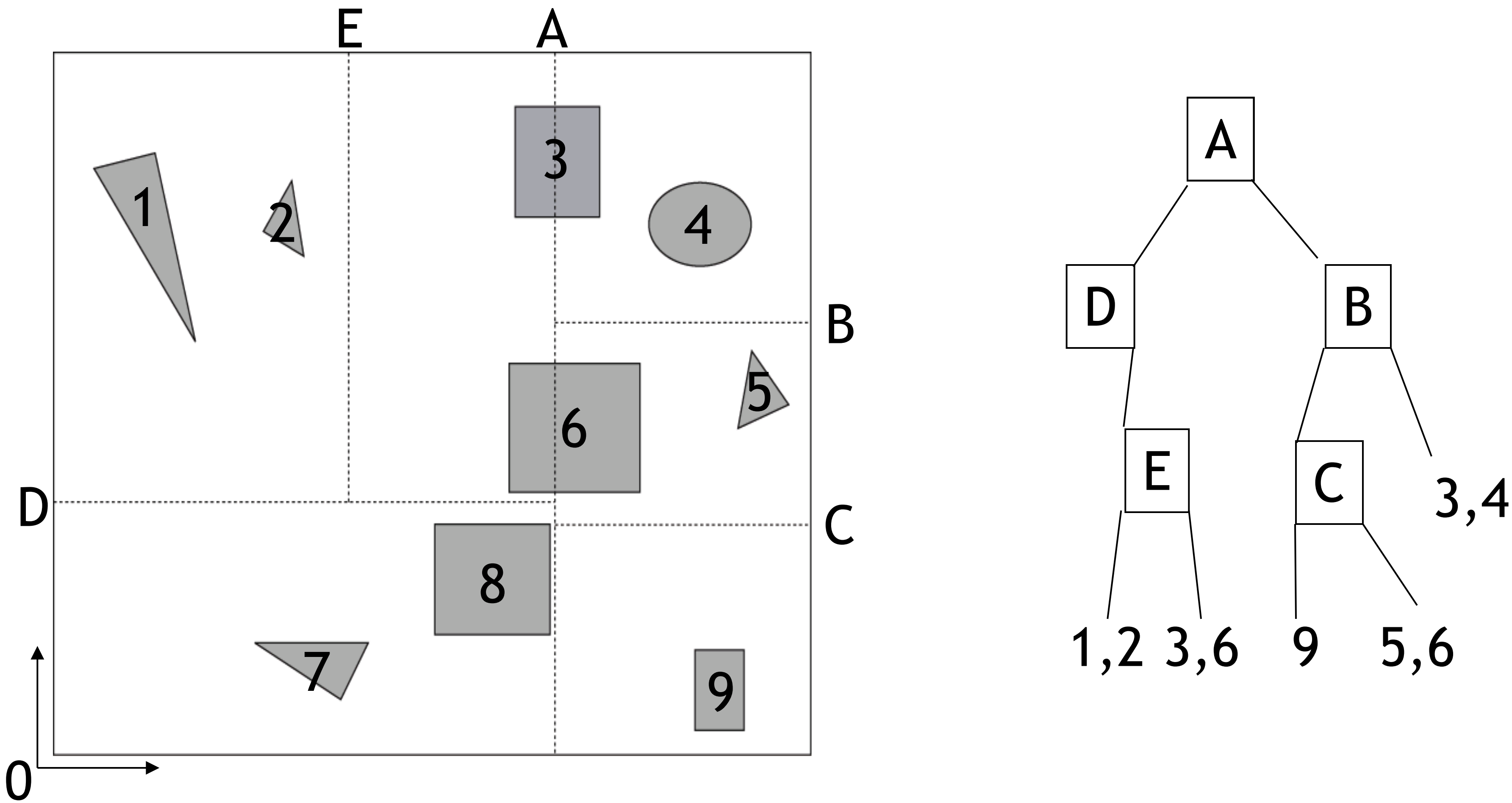


k-d tree example

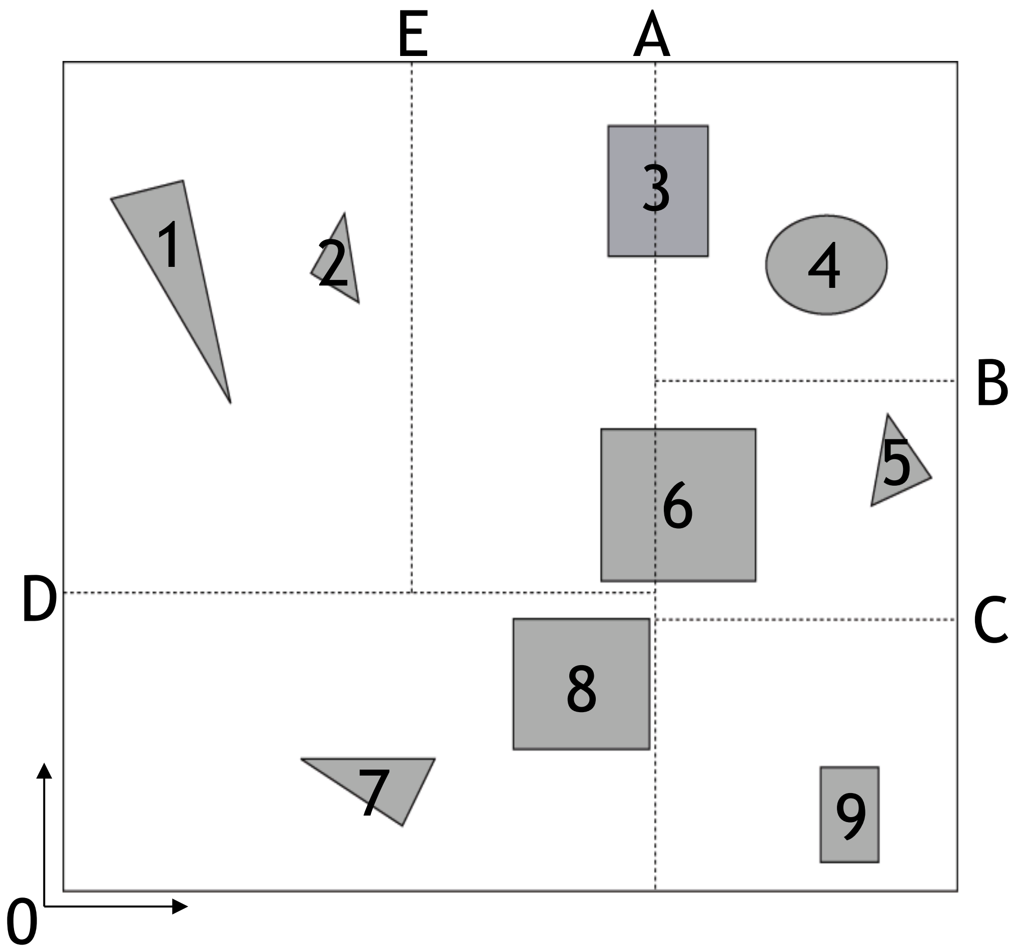


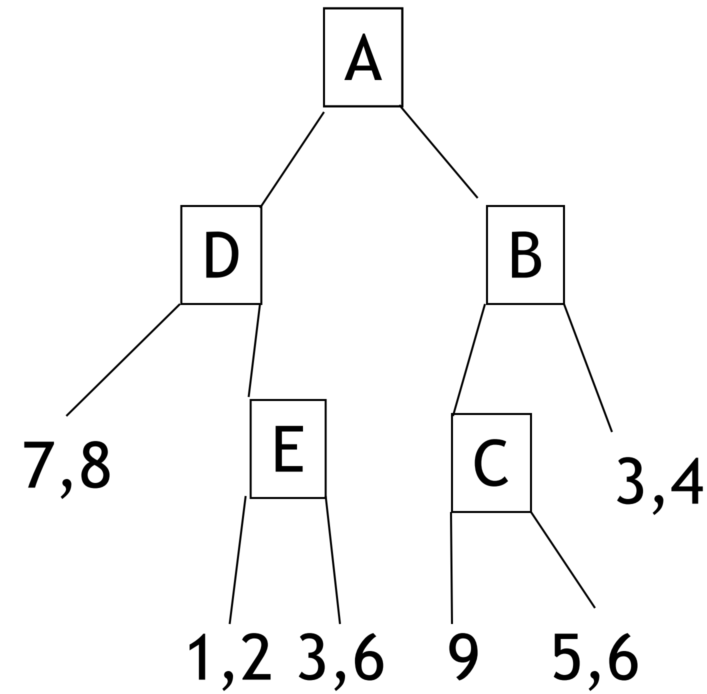


k-d tree example



k-d tree example





k-d tree construction

- Goal: construct tree that minimizes rendering cost
  - Minimize expected number of intersection tests
- Parameters to optimize?
- Details see PBRT book, Section 4.4
  http://pbrt.org/index.html

Traversing a kd-tree is faster than traversing an octree, because each node has fewer children (2 vs. 8)

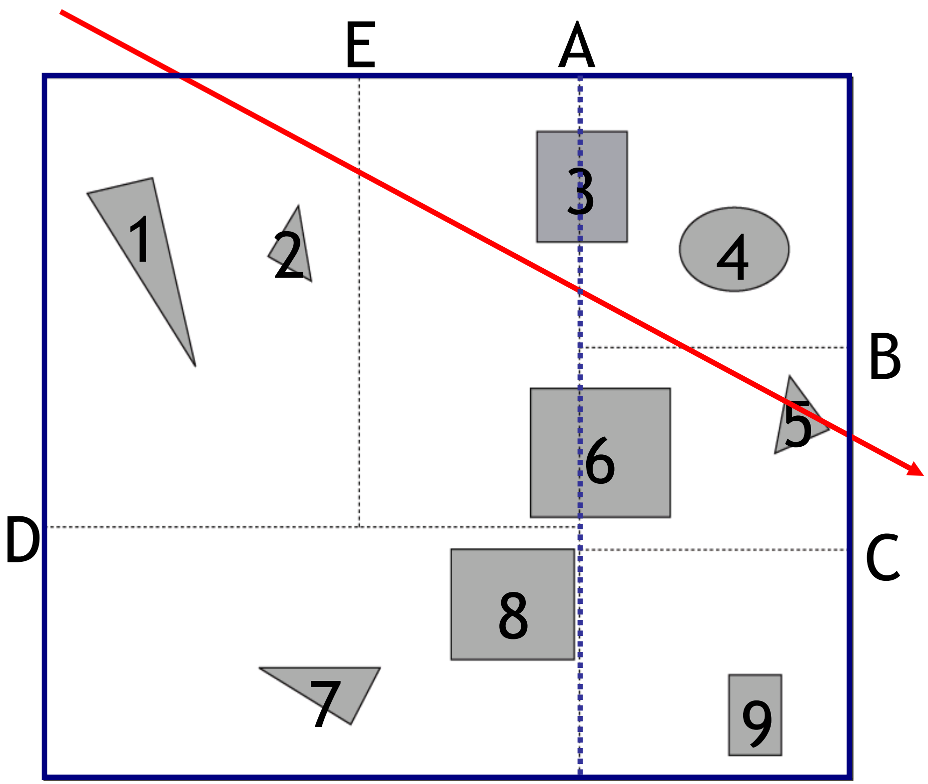
A. True

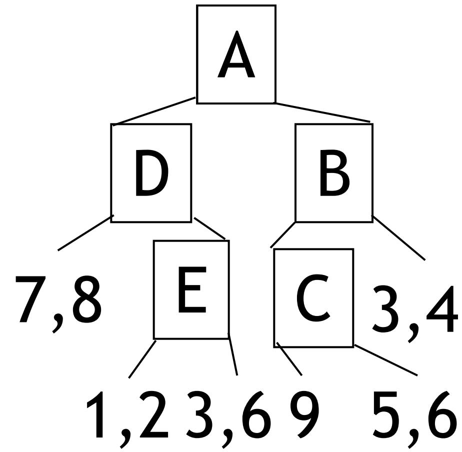
B. False

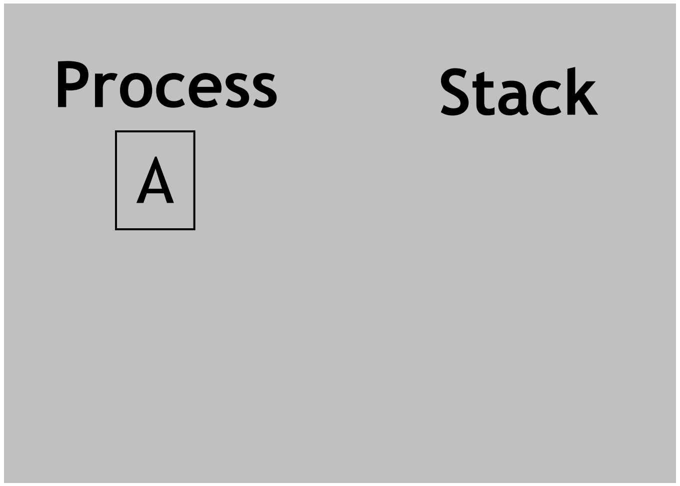
Tree traversal & intersection testing

- “Front-to-back” traversal
  - Traverse child nodes in order (front to back) along rays
- Stop traversing as soon as first surface intersection is found
  - Advantage over BVHs, where this is not possible
- Maintain own stack of subtrees to traverse
  - More efficient than recursive function calls

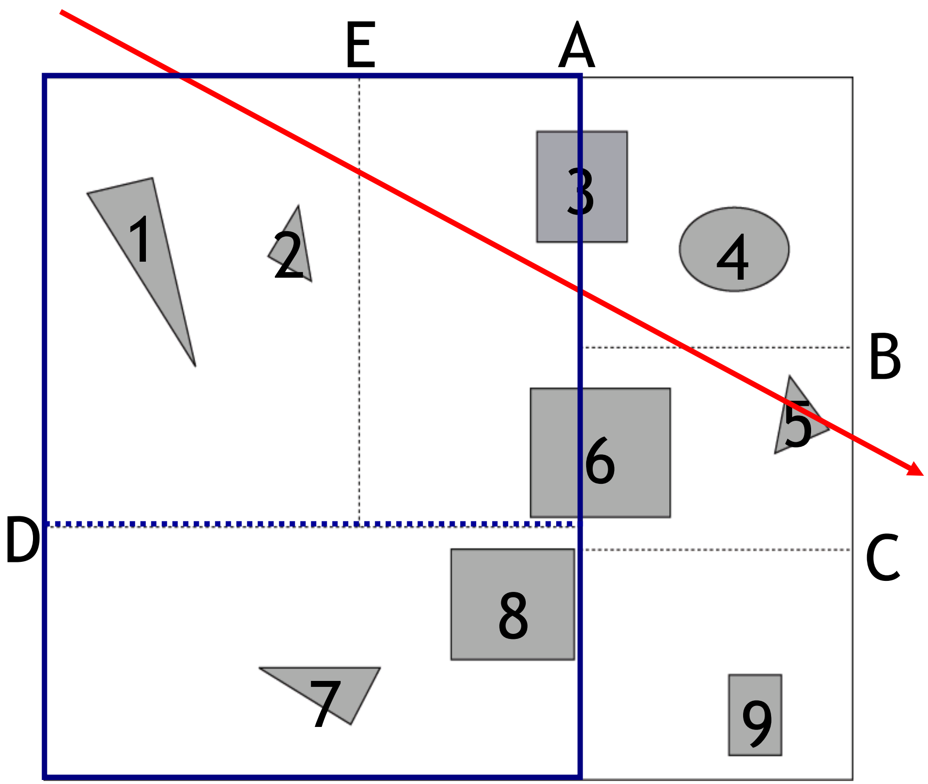
Tree traversal

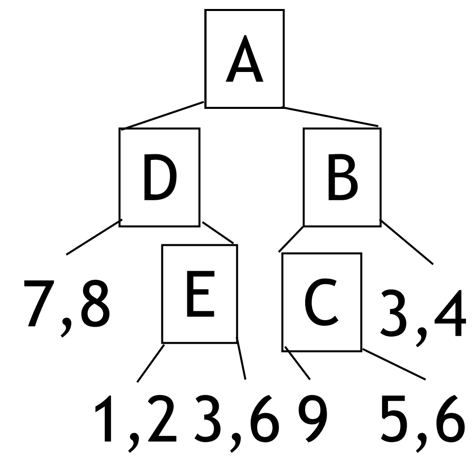


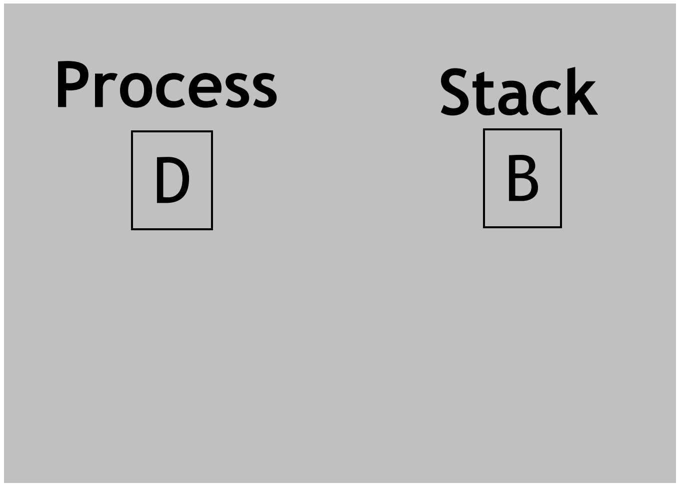




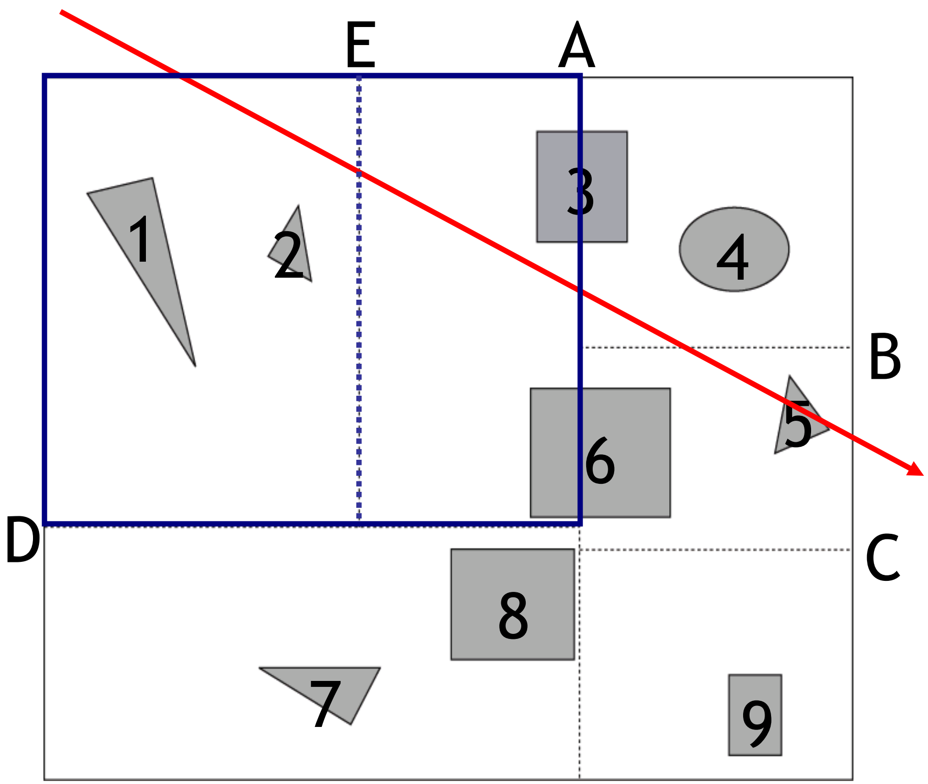
Tree traversal

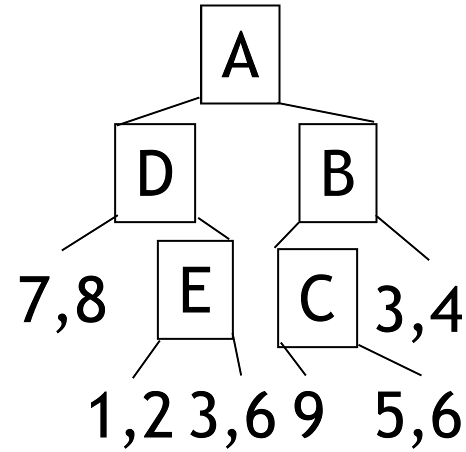






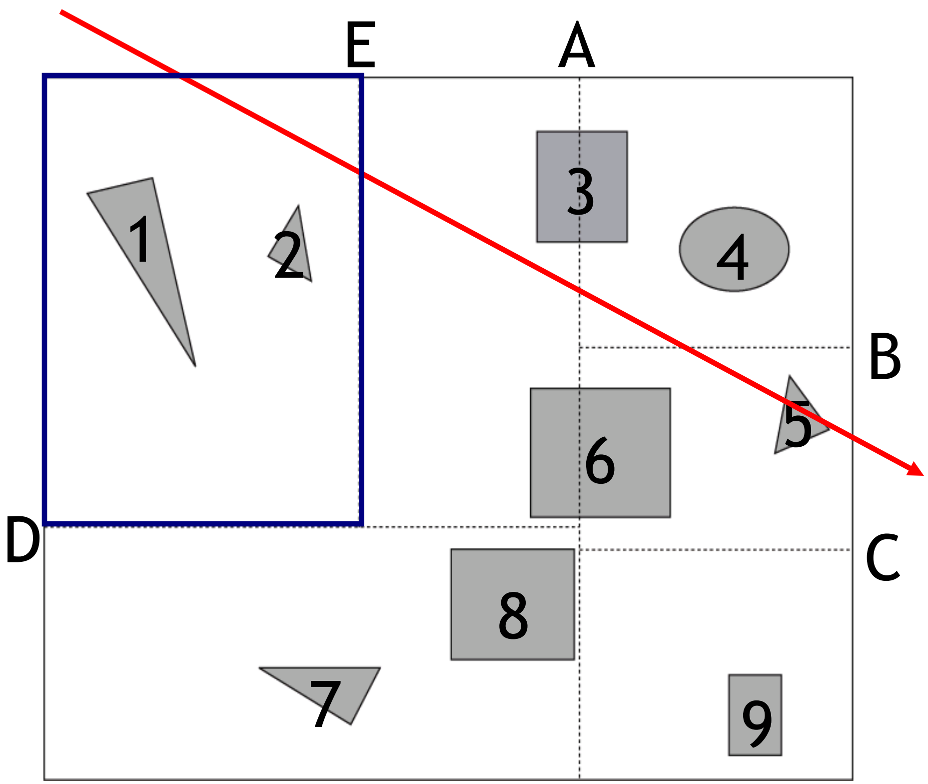
Tree traversal

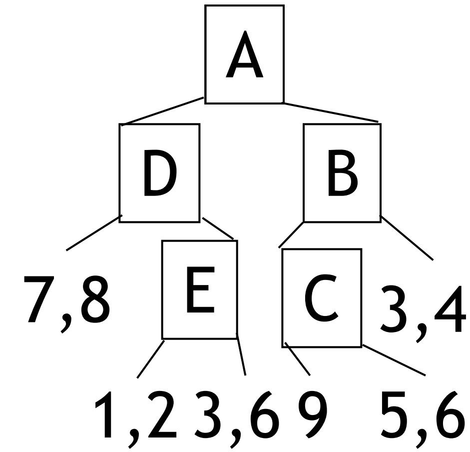




Process	Stack
E	B

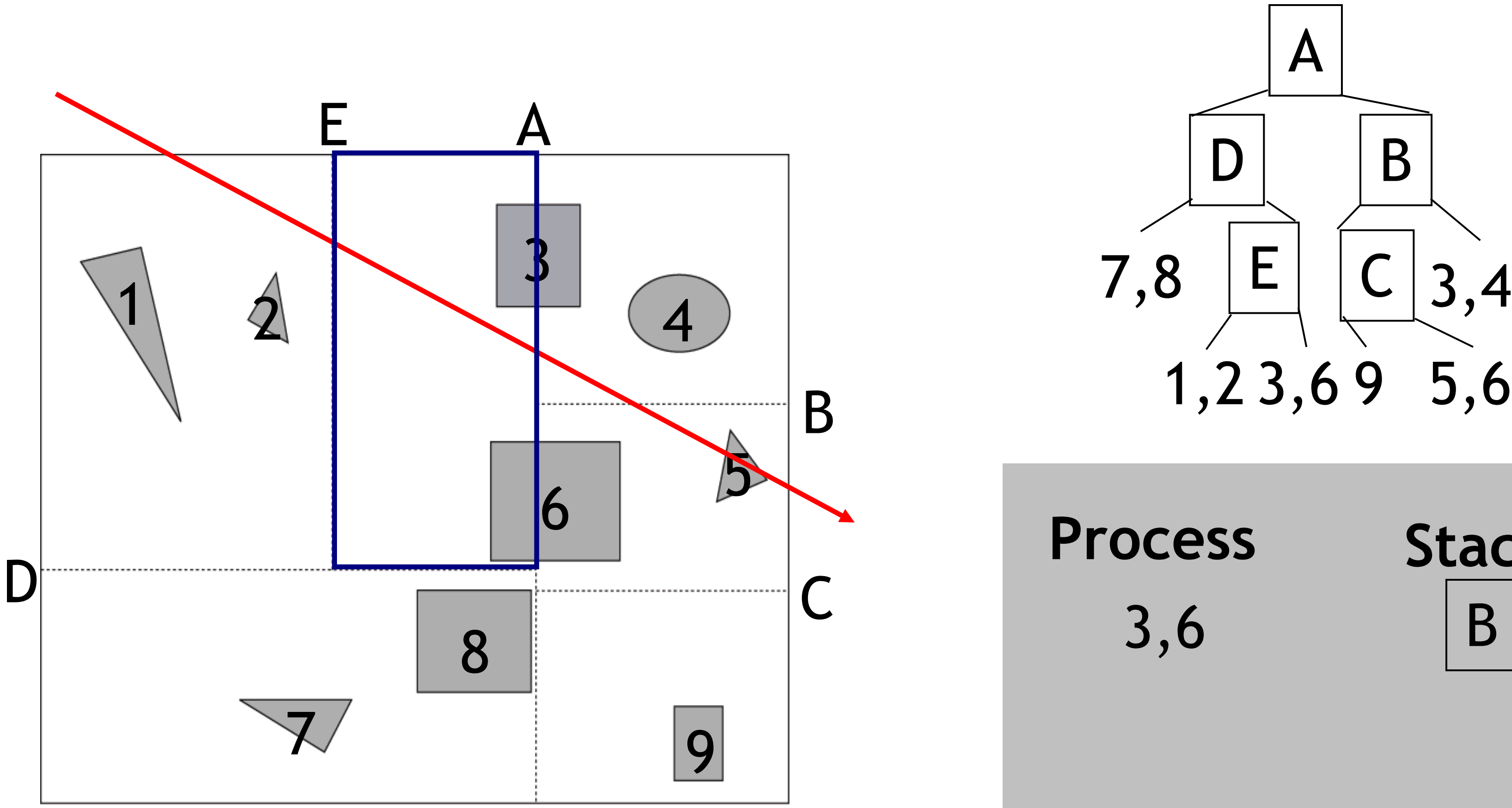
Tree traversal





Process	Stack
1,2	$\boxed{B}$
	3,6

Tree traversal



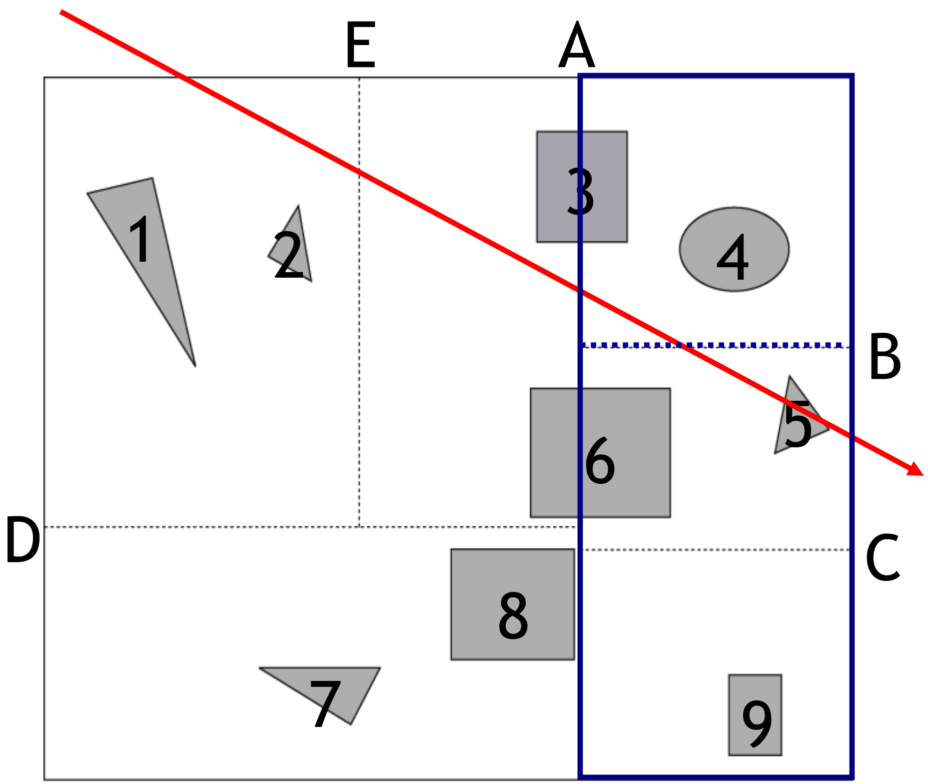
Process

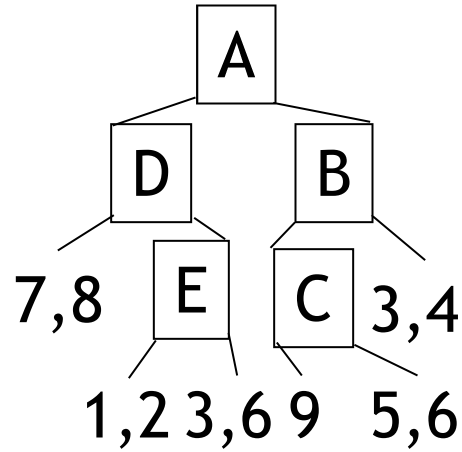
3,6

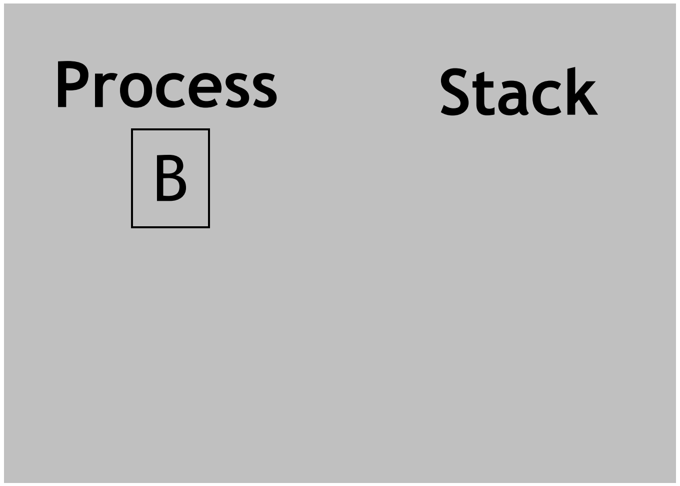
Stack

B

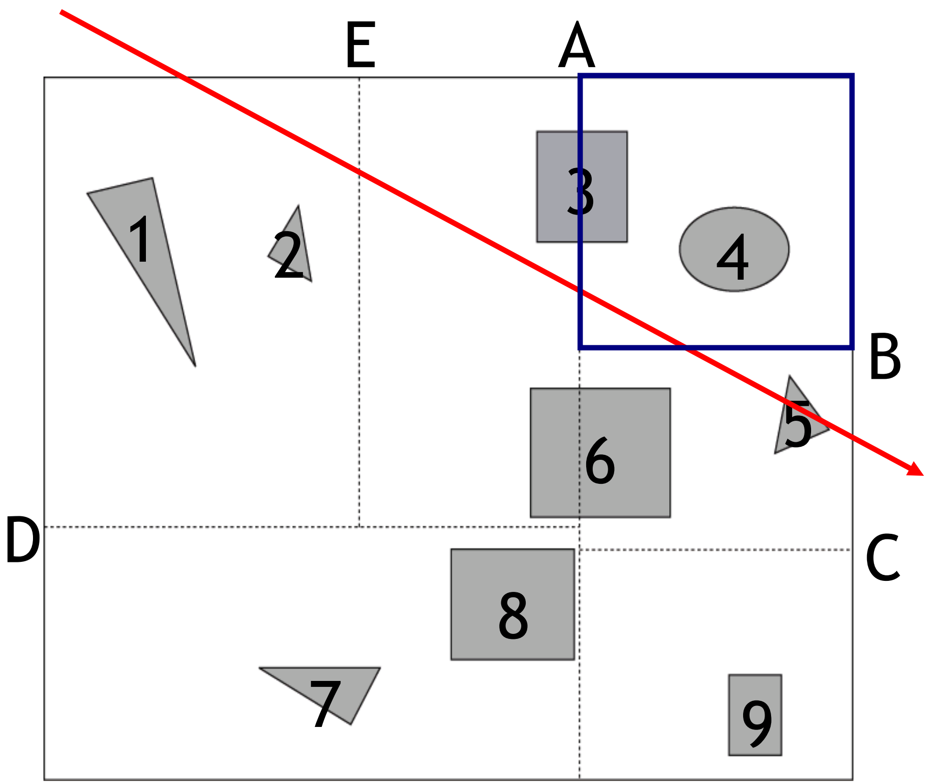
Tree traversal

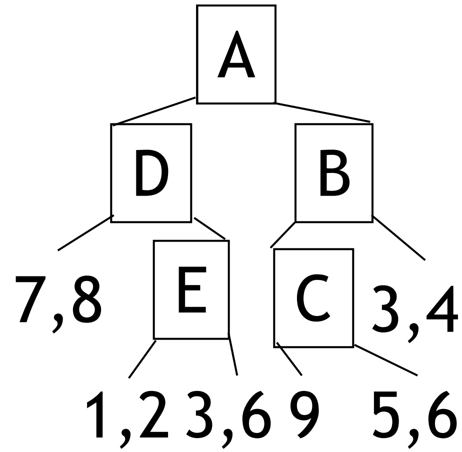






Tree traversal





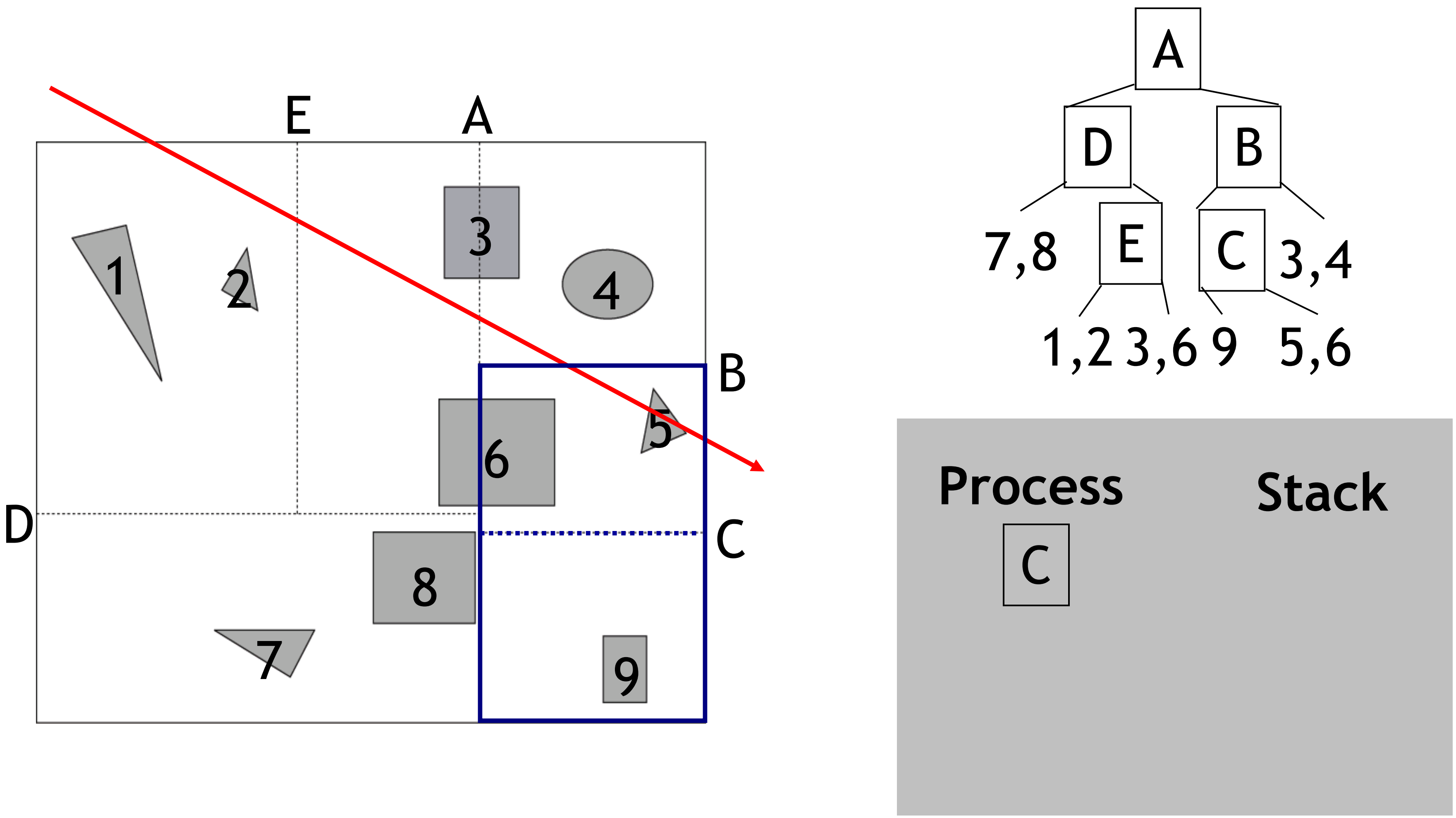
Process

3,4

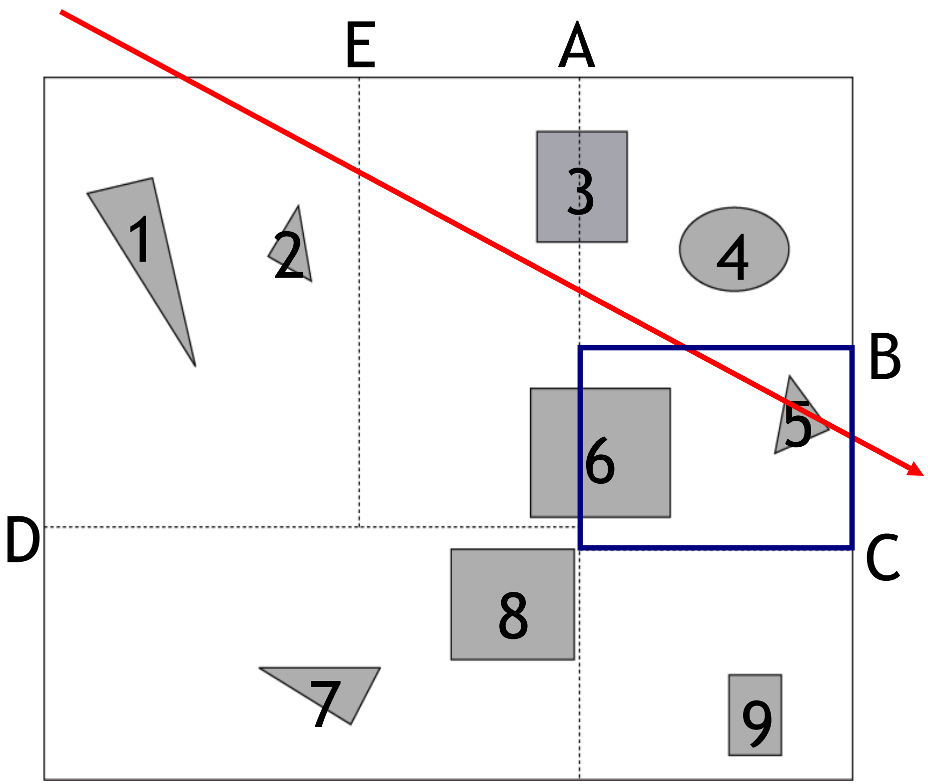
Stack

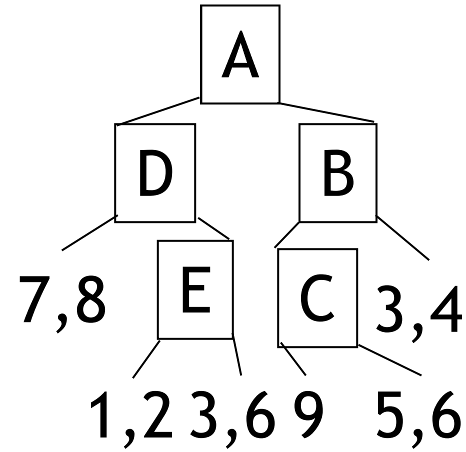
C

Tree traversal



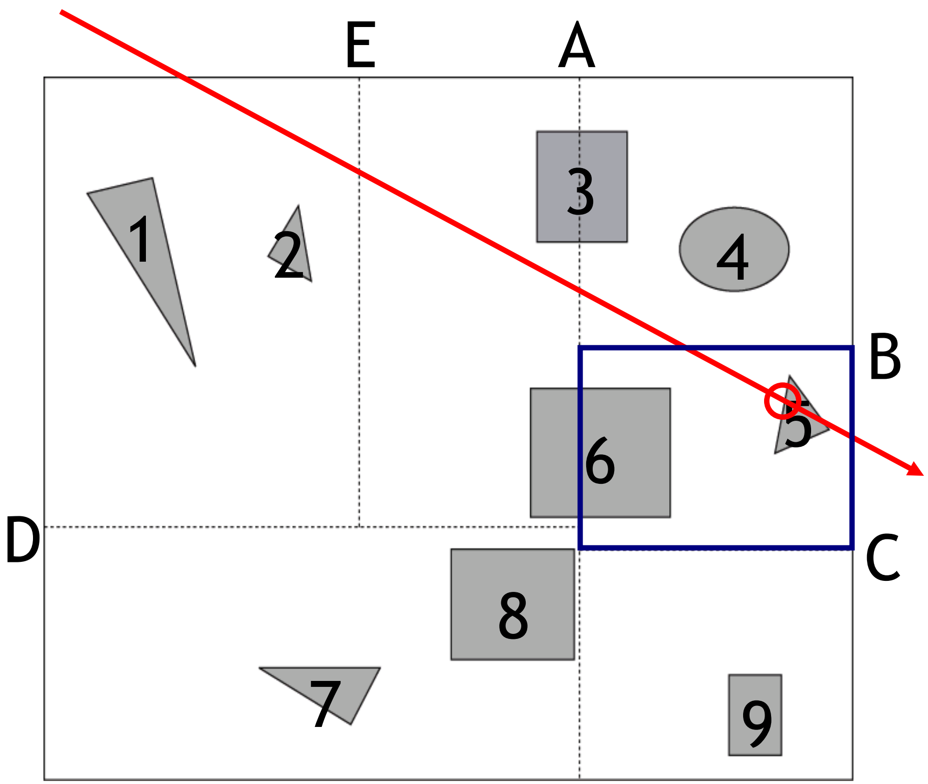
Tree traversal

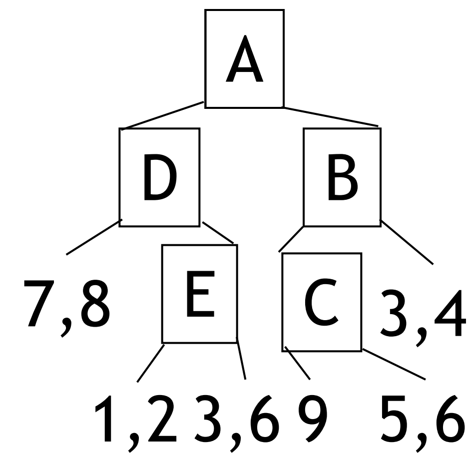




Process	Stack
5,6	

Tree traversal





Process **Stack**

5,6

Note

- Python renderer relies on Open3D for acceleration ray-triangle mesh intersections
  http://www.open3d.org/docs/latest/tutorial/geometry/ray_casting.html

Next time

- Physical models for light and light transport