Lecture slides (CT4201/EC4215 – Computer Graphics)

Acceleration Data Structures

Lecturer: Bochang Moon

Ray Tracing

- Procedure for Ray Tracing:

- For each pixel
  - Generate a primary ray (with depth 0)
  - While (depth < d) {
    - Find the closest intersection point between the ray and objects
    - If (there is a hit) then
      - Generate a shadow ray
      - If (there is no hit between the shadow ray and a light) then
        - Perform a shading
      - Generate a secondary ray (reflection or refraction ray) // increase the ray depth +1
    - Else
      - Perform a shading with background color }
  - Return background color

Naïve Ray Tracing

- Problem: find the closest intersection point between the ray $\boldsymbol{p}(t) = \boldsymbol{e} + t\boldsymbol{d}$

- For each triangle
  - Compute the intersection point (i.e., t) between a ray and triangle
  - If (there is a hit and t < stored t)
    - Store shading information and the ray parameter t
  - Return the shading information

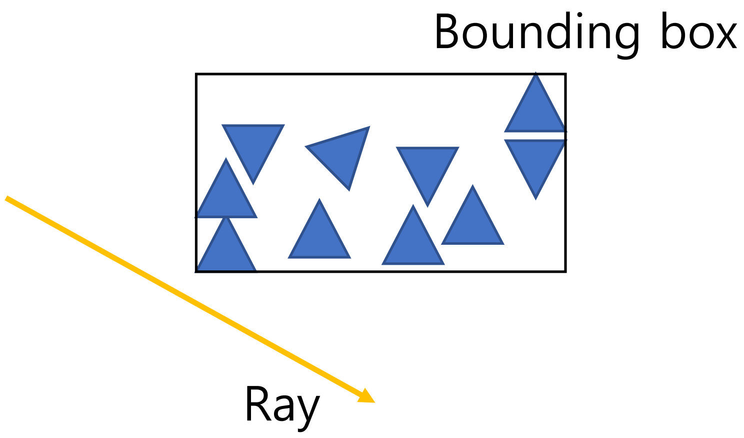
- The complexity of this naïve algorithm is O(N), where N is the number of triangles in the scene

Spatial Data Structures

- Group objects together into a hierarchy to accelerate the geometry processing
- The complexity using the acceleration data structures can be a sub-linear time (e.g., O(logN))
- Object partitioning:
  - Bounding Volume Hierarchy (BVH)
- Space partitioning:
  - Uniform Grids
  - Octree (3D) or QuadTree (2D)
  - Binary space partition tree (BSP)
  - kD-Trees

Bounding Boxes

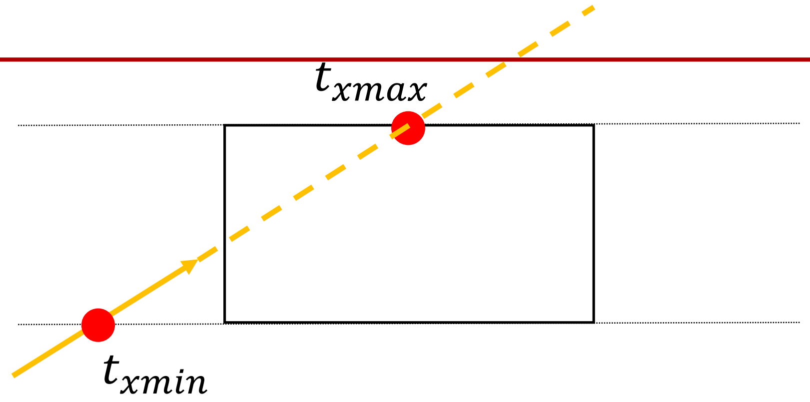
- The key operation is to perform an intersection test between a ray and bounding box
  - Need to know only whether a ray hits the box or not



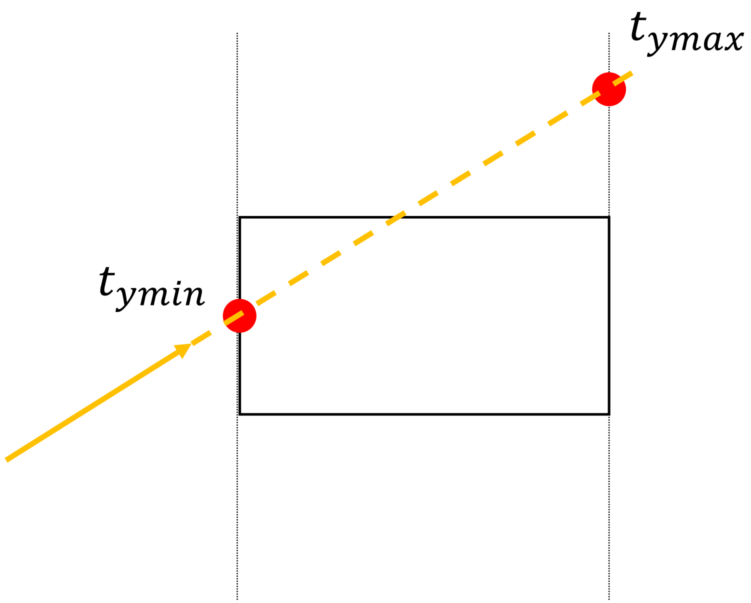
- Ray: $\boldsymbol{p}(t) = \boldsymbol{e} + t\boldsymbol{d}$
- 2D version
  - $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$

Bounding Boxes

- Ray: $\boldsymbol{p}(t) = \boldsymbol{e} + t\boldsymbol{d}$
- 2D version
  - $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$



- $t_{xmin} = \frac{x_{min} - x_e}{x_d}$
- $t_{xmax} = \frac{x_{max} - x_e}{x_d}$



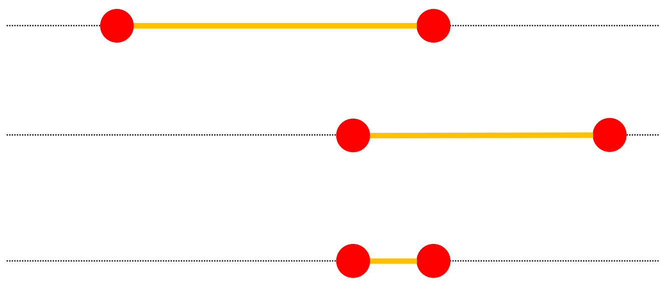
- $t_{ymin} = \frac{y_{min} - y_e}{y_d}$
- $t_{ymax} = \frac{y_{max} - y_e}{y_d}$

Bounding Boxes

- Ray: $\boldsymbol{p}(t) = \boldsymbol{e} + t\boldsymbol{d}$

- $t_{xmin} = \frac{x_{min} - x_e}{x_d}$, $t_{xmax} = \frac{x_{max} - x_e}{x_d}$
- $t_{ymin} = \frac{y_{min} - y_e}{y_d}$, $t_{ymax} = \frac{y_{max} - y_e}{y_d}$

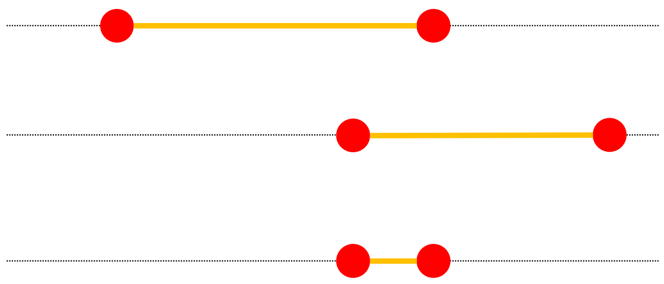
- $t \in [t_{xmin}, t_{xmax}]$
- $t \in [t_{ymin}, t_{ymax}]$
- $t \in [t_{xmin}, t_{xmax}] \cap [t_{ymin}, t_{ymax}]$



- A ray hits the box if and only if the two intervals overlap.

Bounding Boxes

- Procedure for testing the intersection
  - Compute $t_{xmin}, t_{xmax}, t_{ymin}, t_{ymax}$
  - If $(t_{xmin} > t_{ymax}\ or\ \ t_{xmax} < t_{ymin})$
    - No hit
  - else
    - Hit



Bounding Boxes

- Negative x_d or y_d:
  - A ray will hit x_{max} $(or\ y_{\max})$ before it hits x_{min} $(or\ y_{min})$
  - If $(x_d \geq 0)$ then
    - $t_{min} = (x_{min} - x_e)/x_d$
    - $t_{max} = (x_{max} - x_e)/x_d$
  - else
    - $t_{min} = (x_{max} - x_e)/x_d$
    - $t_{max} = (x_{min} - x_e)/x_d$
  - If $(y_d \geq 0)$ then
    - $t_{min} = (y_{min} - y_e)/y_d$
    - $t_{max} = (y_{max} - y_e)/y_d$
  - else
    - $t_{min} = (y_{max} - y_e)/y_d$
    - $t_{max} = (y_{min} - y_e)/y_d$

Bounding Boxes

- Zero x_d or y_d:
  - Divide-by-zero issue

- Given a number $a \in \mathbb{R}^+$, IEEE floating point rules provide the following:
  - $\frac{+a}{+0} = \infty$
  - $\frac{-a}{+0} = -\infty$
  - $[t_{xmin}, t_{xmax}] = [-\infty, -\infty], [\infty, \infty]$: no hit
  - $[t_{xmin}, t_{xmax}] = [-\infty, \infty]$: hit
  - The precious code works for +0 denominator

- How about -0 denominator?
  - We can test a reciprocal of the ray direction (e.g., $1/x_d$)

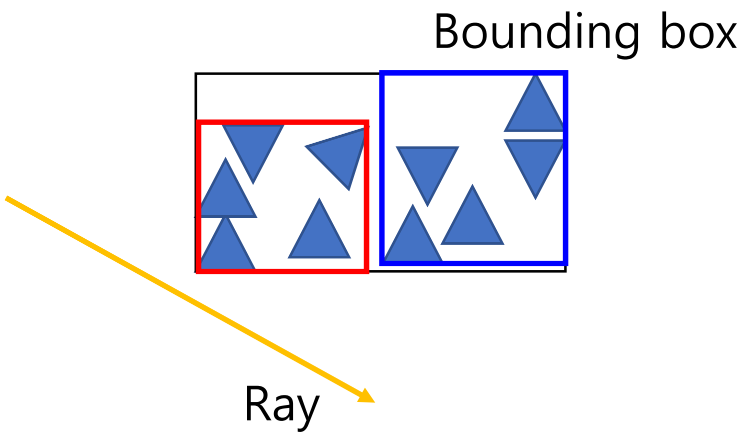
Bounding Boxes

- -0 denominator?
  - If $(x_d \geq 0)$ then
    - $t_{min} = (x_{min} - x_e)/x_d$
    - $t_{max} = (x_{max} - x_e)/x_d$
  - else
    - $t_{min} = (x_{max} - x_e)/x_d$
    - $t_{max} = (x_{min} - x_e)/x_d$

- Problem: the first if statements will be true because $-0 == 0$ is true (IEEE floating point standard), so we can miss valid hits.
  - A remedy is test a reciprocal of the ray direction (e.g., $1/x_d$) instead of x_d
  - More detail:
    - An Efficient and Robust Ray–Box Intersection Algorithm, Williams et al. 2005

Hierarchical Bounding Boxes

- Motivation: expensive as we need to test all primitives within a bounding box that a ray hits
- Solution: the bounding boxes can be built in a hierarchical way
- Two popular hierarchical methods:
  - Bounding volume hierarchy (BVH)
  - Kd-tree

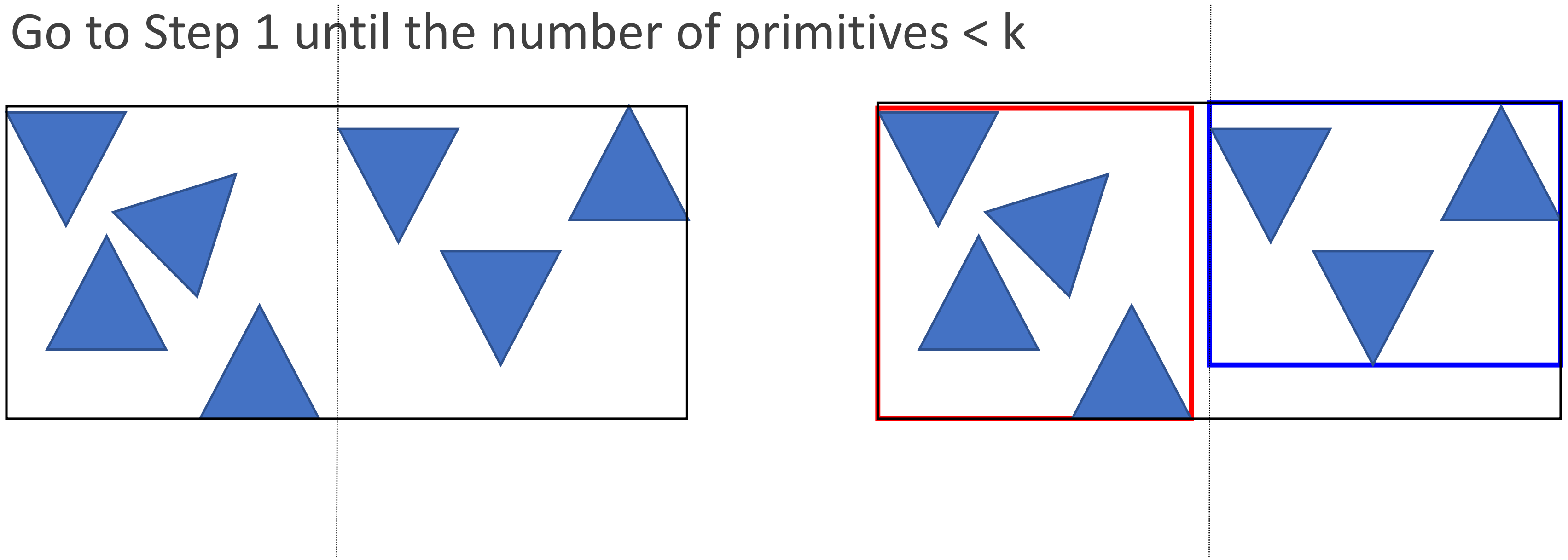


Bounding Volume Hierarchy

- Step 1. Compute a bounding box of primitives
  - e.g., Axis-Aligned Bounding Box (AABB) $[x_{min}, y_{min}, z_{min}] \times [x_{max}, y_{max}, z_{max}]$

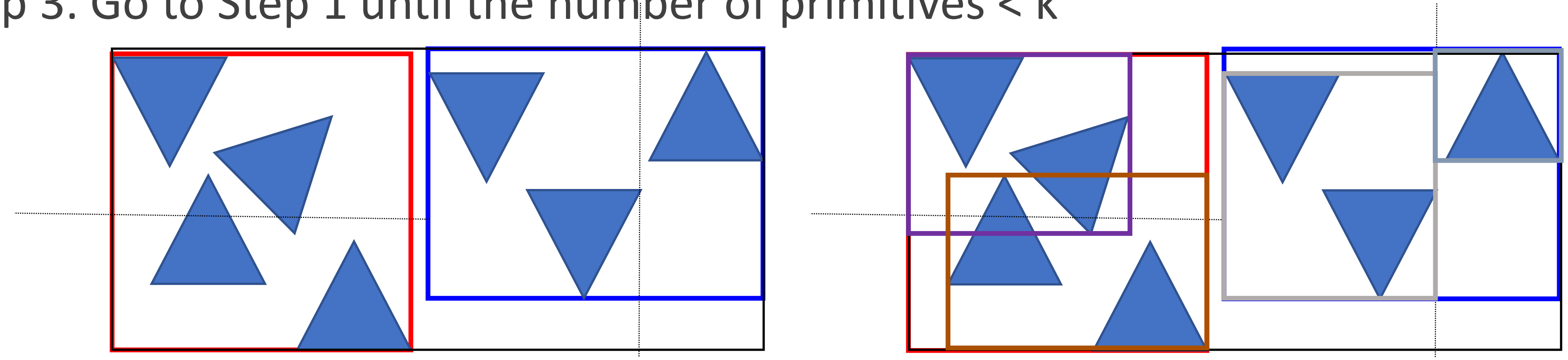
- Step 2. Split the primitives into two groups and compute the child BVs

- Step 3. Go to Step 1 until the number of primitives < k



Bounding Volume Hierarchy

- Step 1. Compute a bounding box of primitives
  - e.g., Axis-Aligned Bounding Box (AABB) $[x_{min}, y_{min}, z_{min}] \times [x_{max}, y_{max}, z_{max}]$
- Step 2. Split the primitives into two groups and compute the child BVs
- Step 3. Go to Step 1 until the number of primitives < k

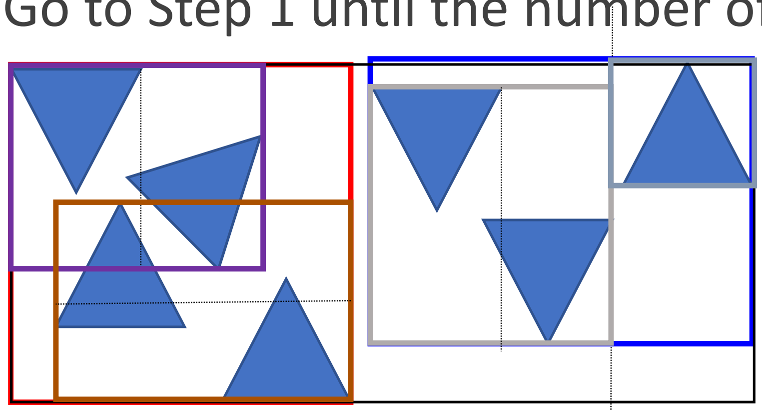


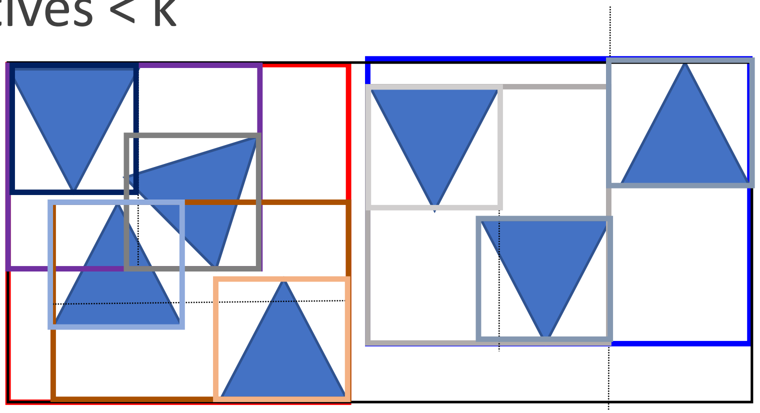
Bounding Volume Hierarchy

- Step 1. Compute a bounding box of primitives
  - e.g., Axis-Aligned Bounding Box (AABB) $[x_{min}, y_{min}, z_{min}] \times [x_{max}, y_{max}, z_{max}]$

- Step 2. Split the primitives into two groups and compute the child BVs

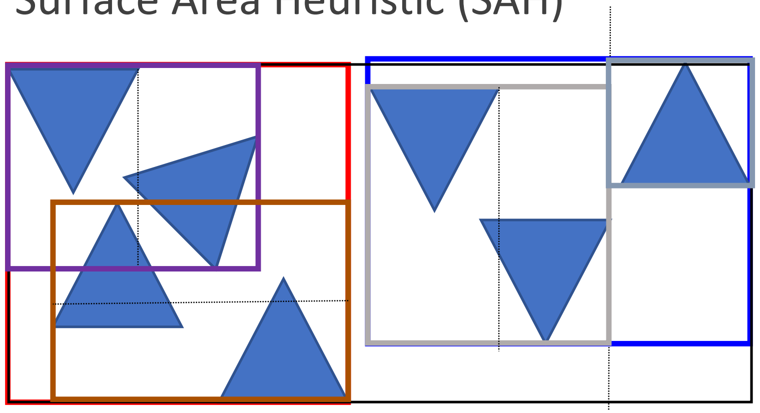
- Step 3. Go to Step 1 until the number of primitives < k

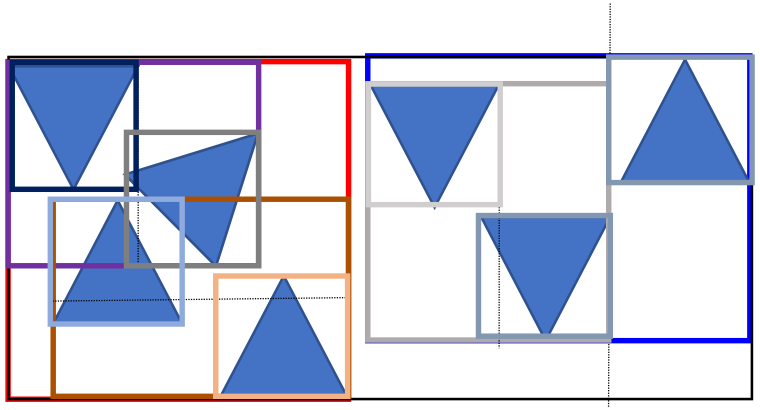




Bounding Volume Hierarchy

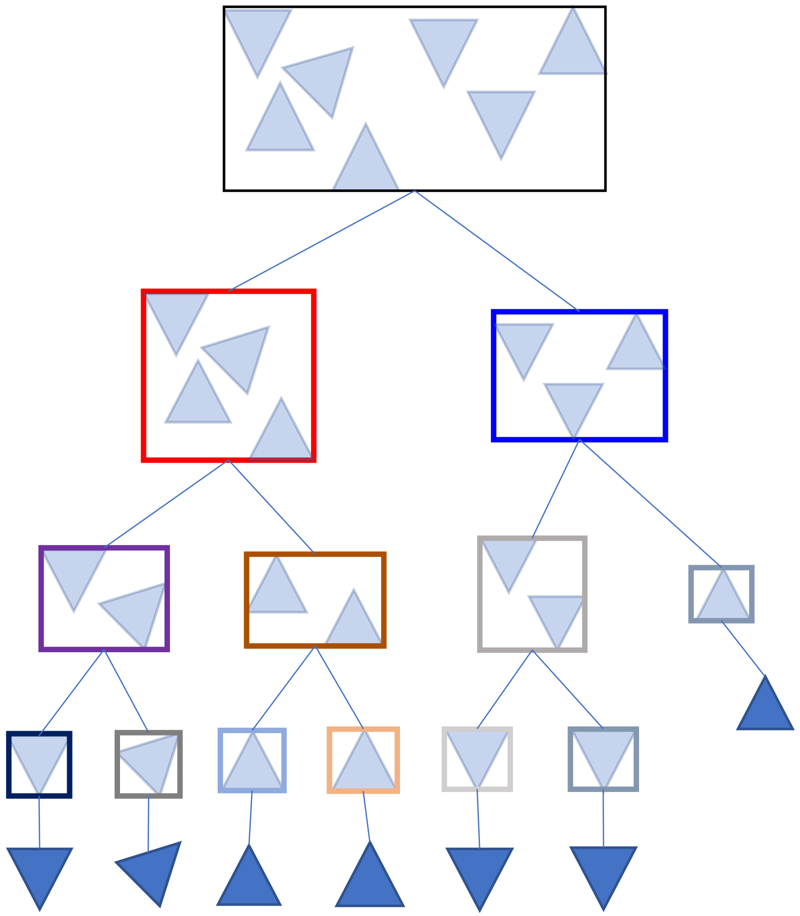
- Where should we split the primitives?
  - Midpoint of a volume
  - Sort the primitives, and select the median
  - Other approaches?
    - Surface Area Heuristic (SAH)





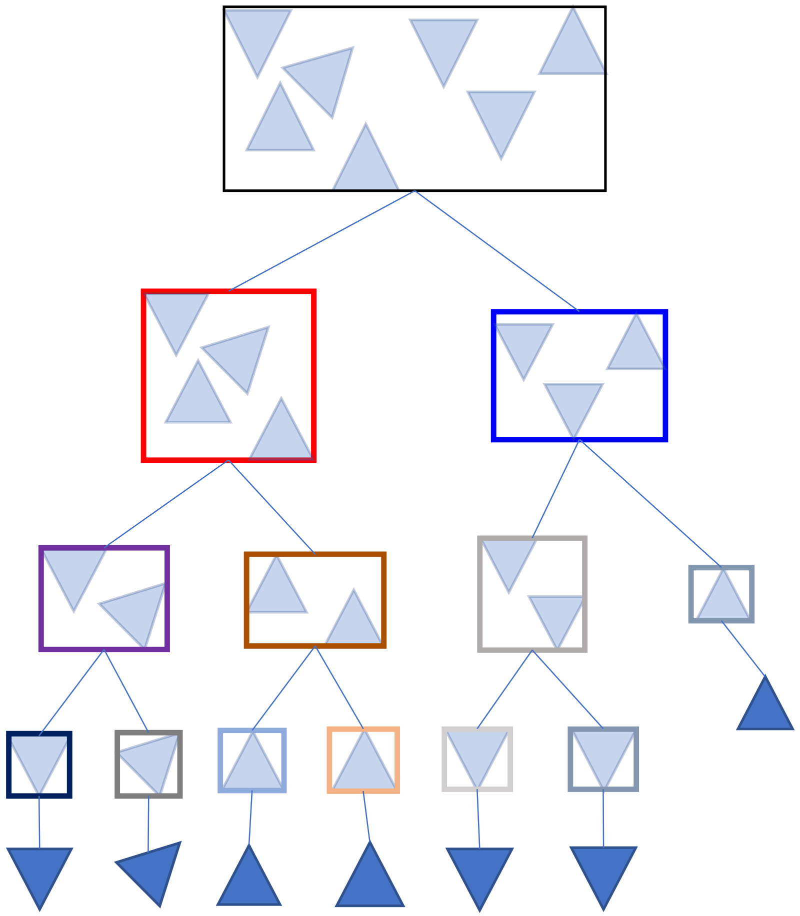
Bounding Volume Hierarchy

- Traversal procedure:
  - Check whether the intersection occurs
  - If (hit and t < ray.t) then
    - If (the BV is a leaf node)
      - Find the closest intersection point between the ray and triangle
      - If (the ray hits triangles) then
        - ray.t = t (from the closest intersection)
        - Store some shading info.
    - else
      - Check an intersection using its child BVs



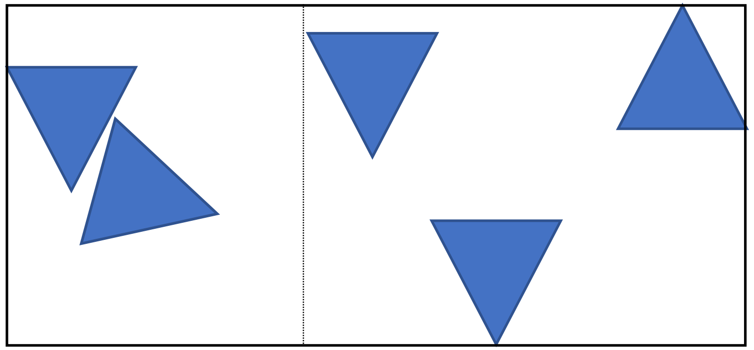
Bounding Volume Hierarchy

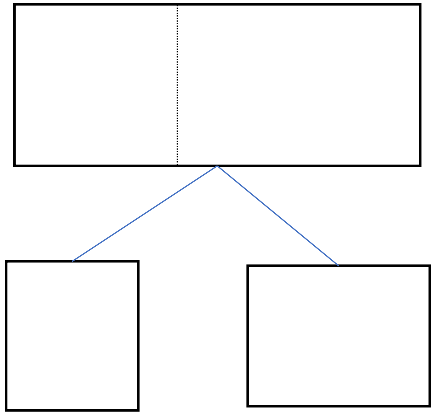
- Properties of BVH
  - Object partitioning: split primitives
  - Some BVs can overlap each other



Kd-trees

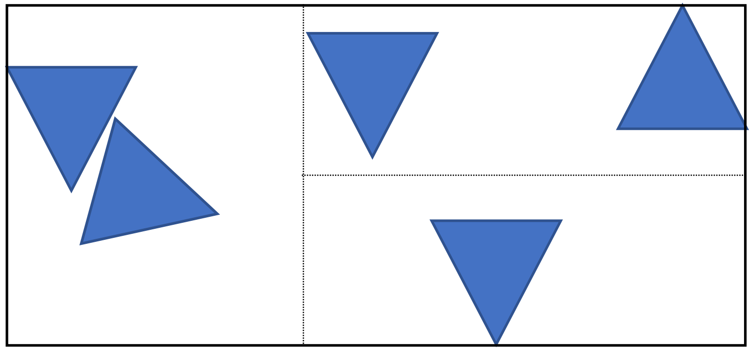
- Recursively split space with axis-aligned planes

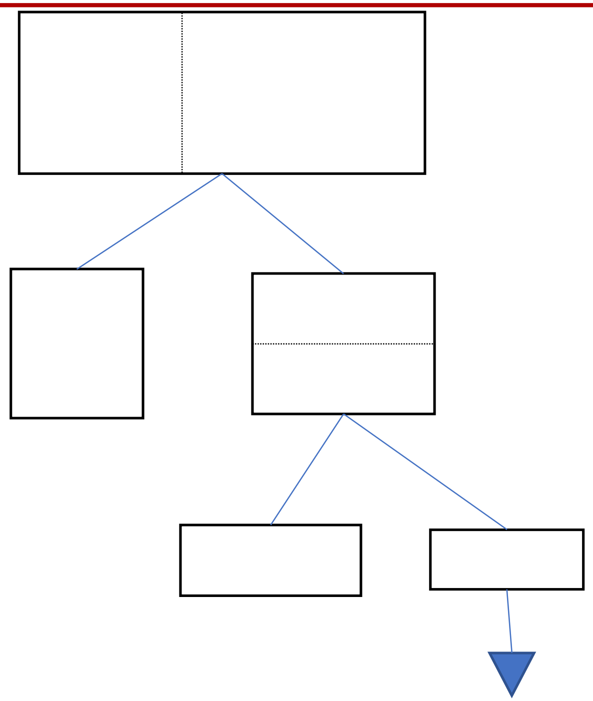




Kd-trees

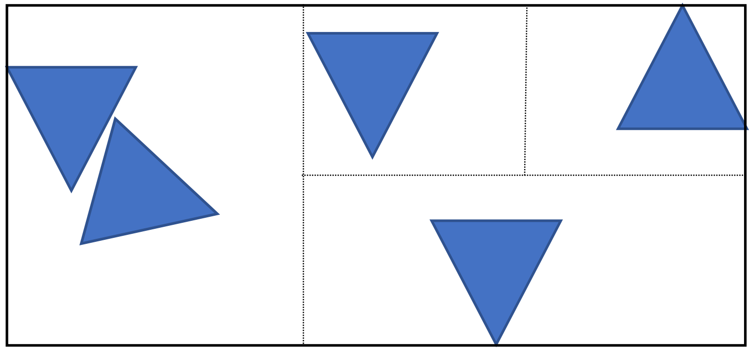
- Recursively split space with axis-aligned planes

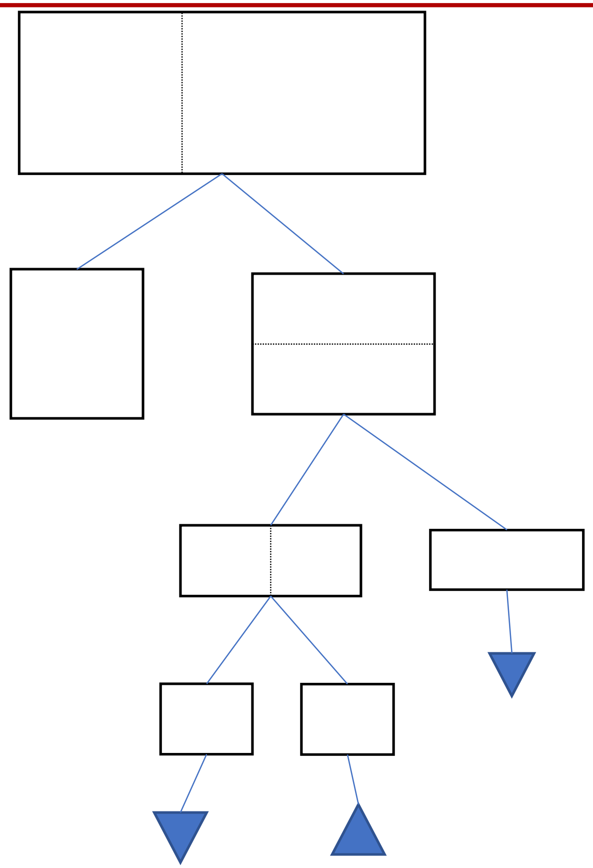




Kd-trees

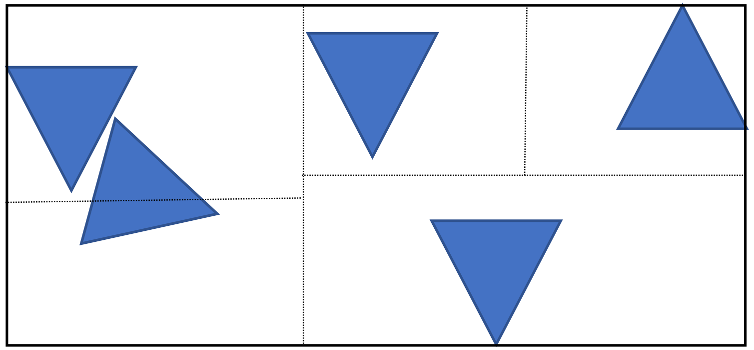
- Recursively split space with axis-aligned planes

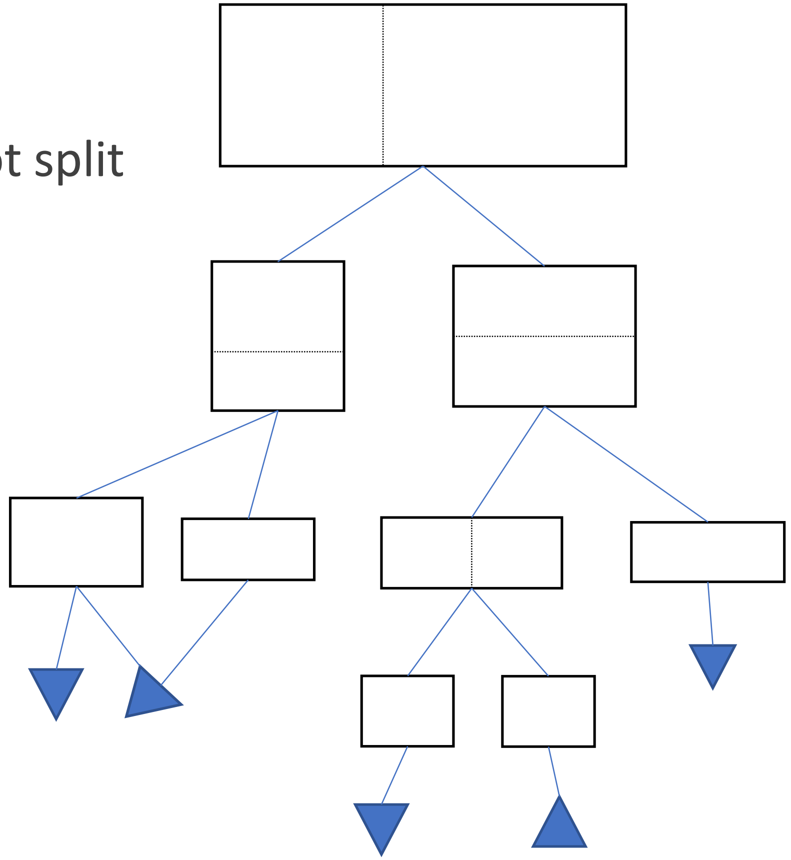




Kd-trees

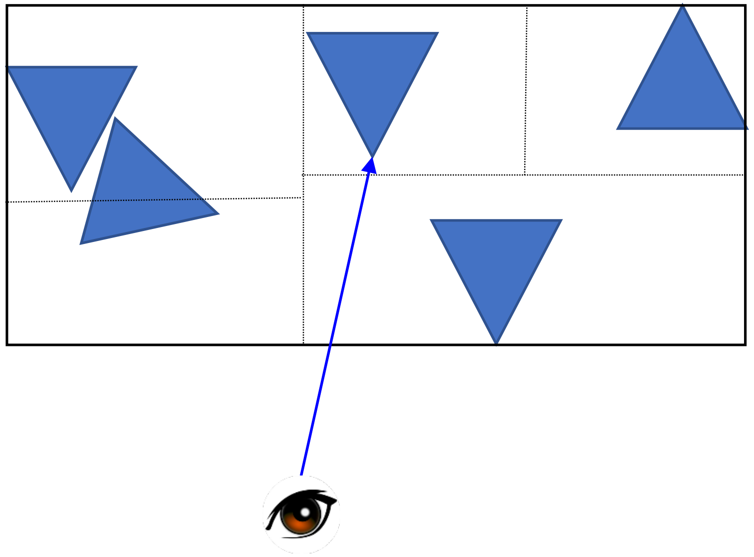
- Recursively split space with axis-aligned planes
  - Some nodes can point same triangles if we cannot split them

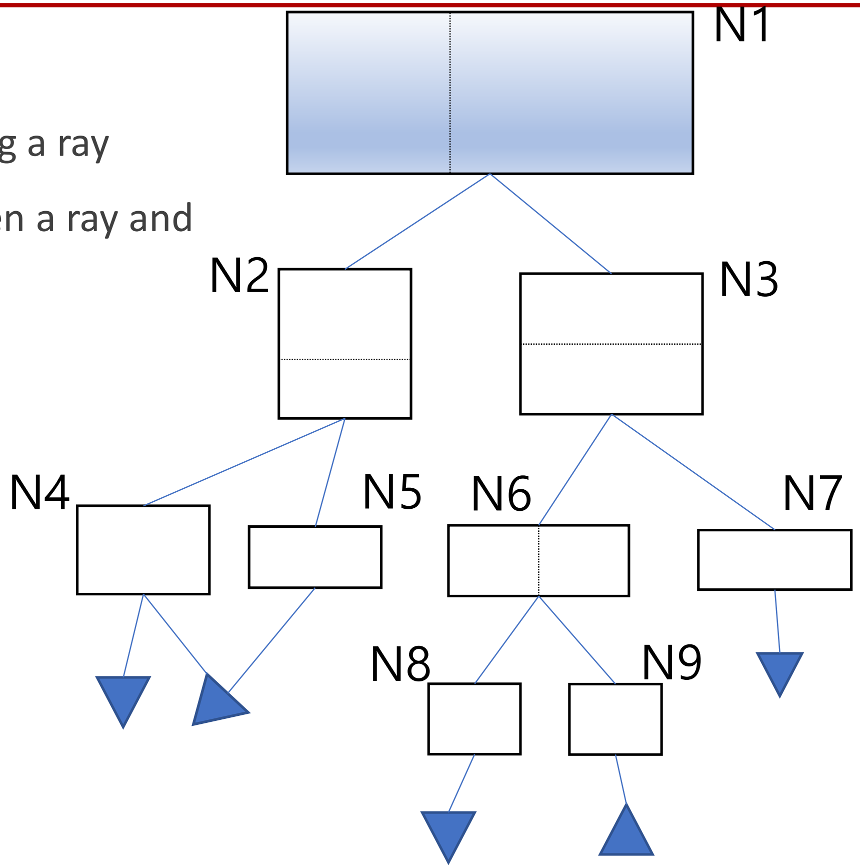




Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found



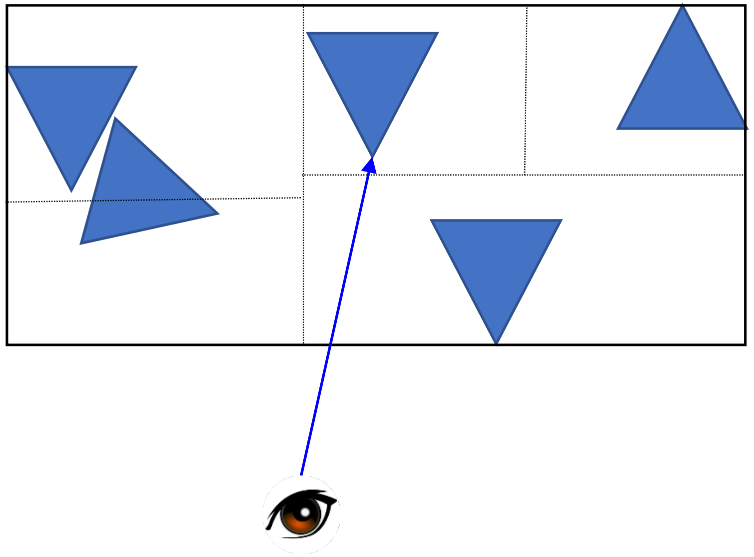


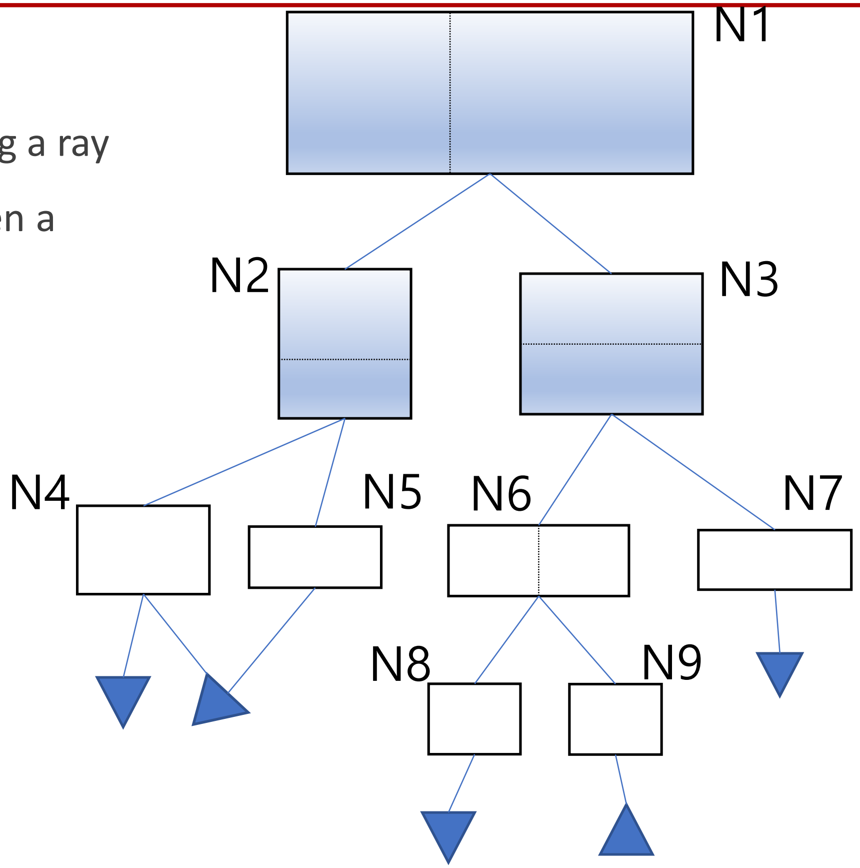
Current node: N1

Stack:

Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found



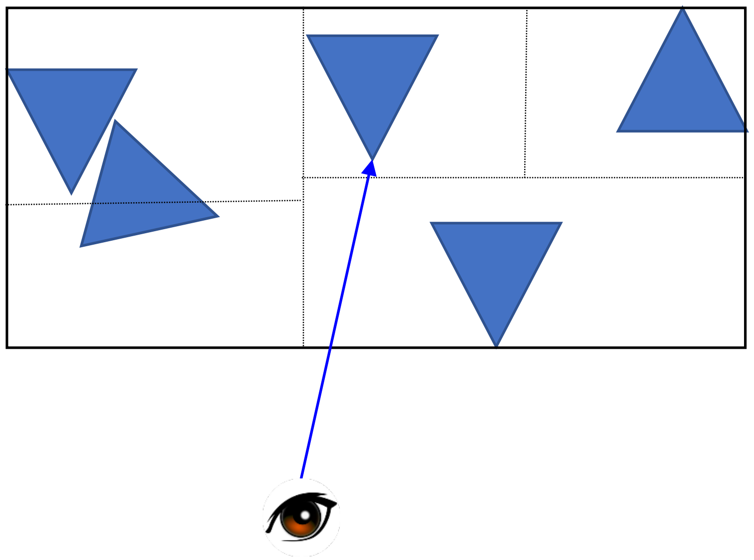


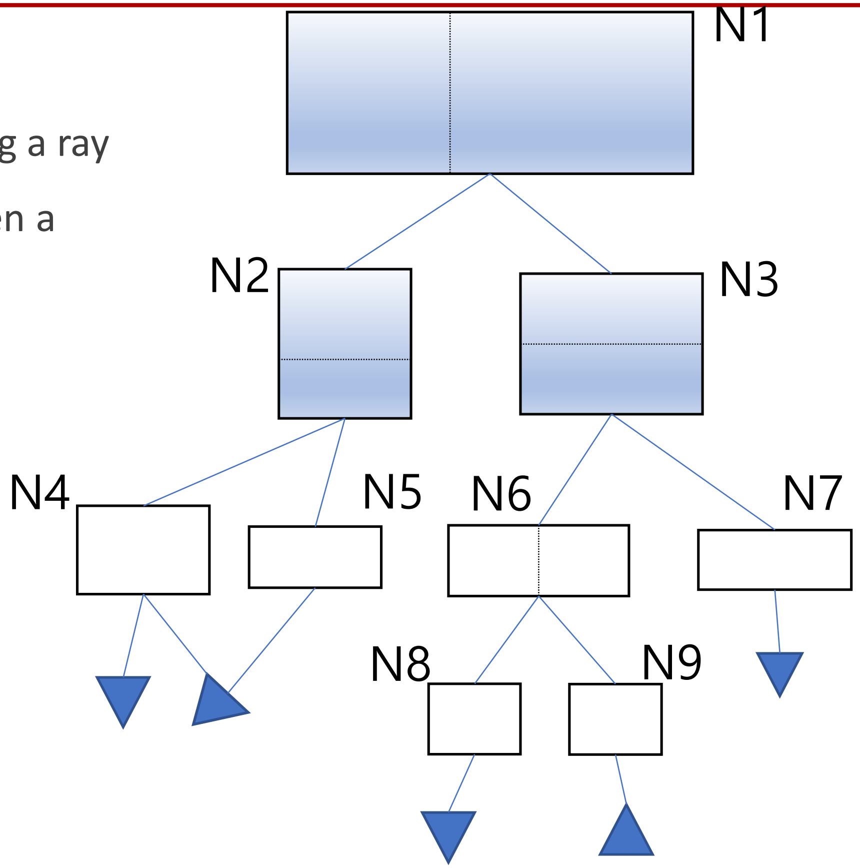
Current node: N2

Stack: N3

Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found



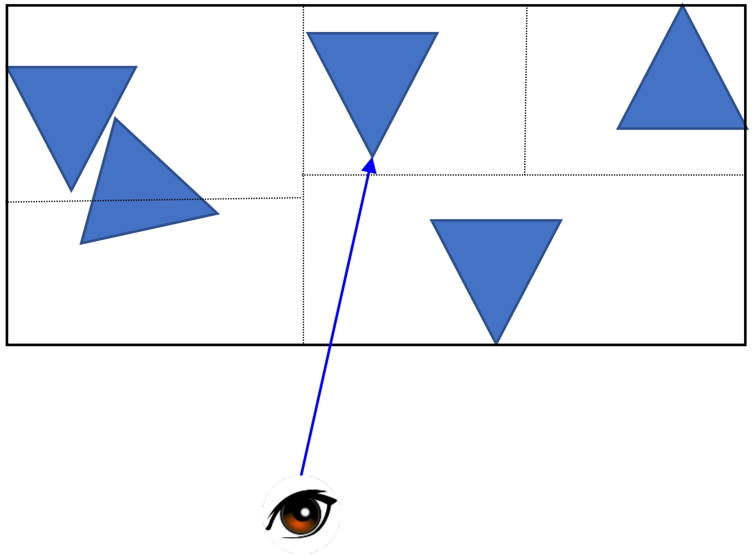


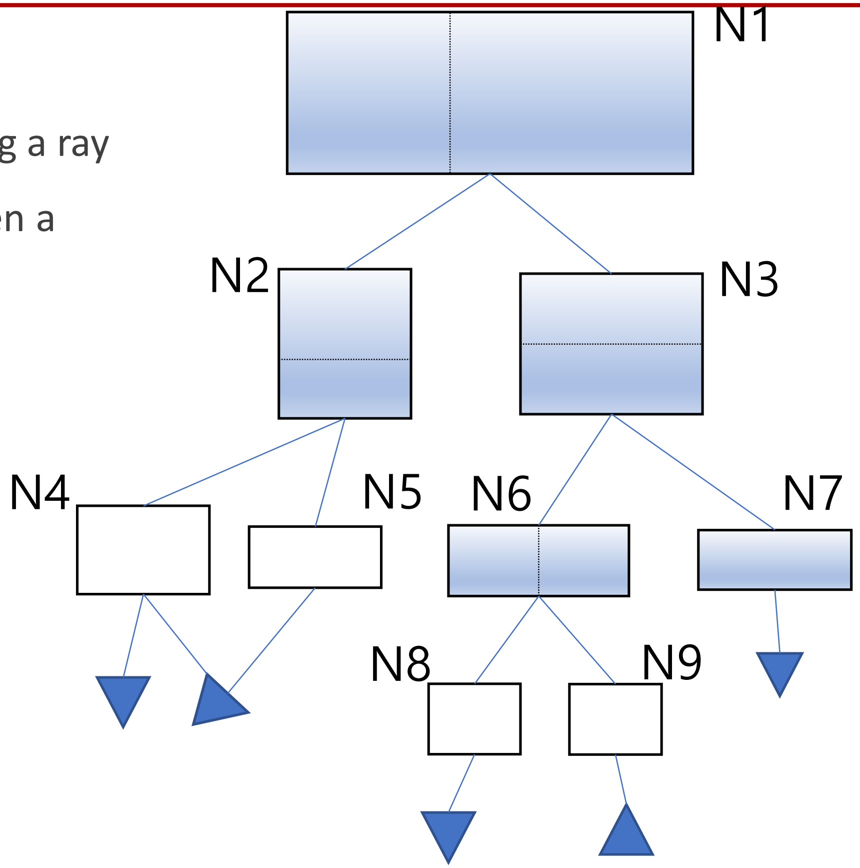
Current node: N3

Stack:

Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found



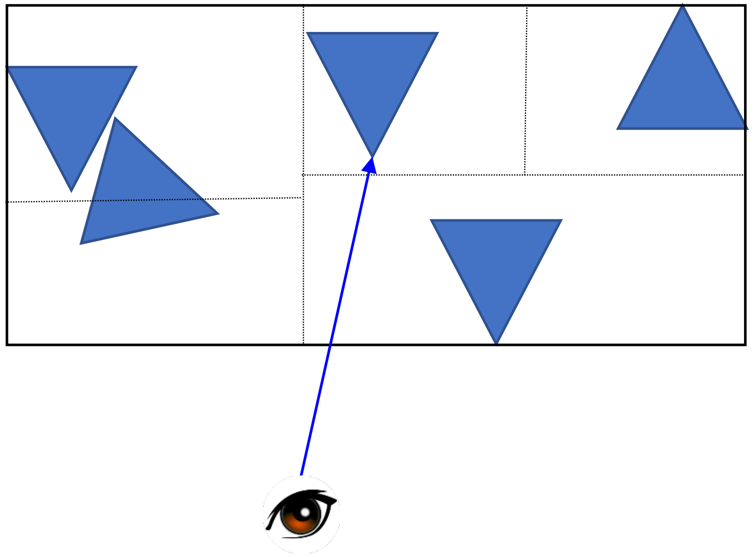


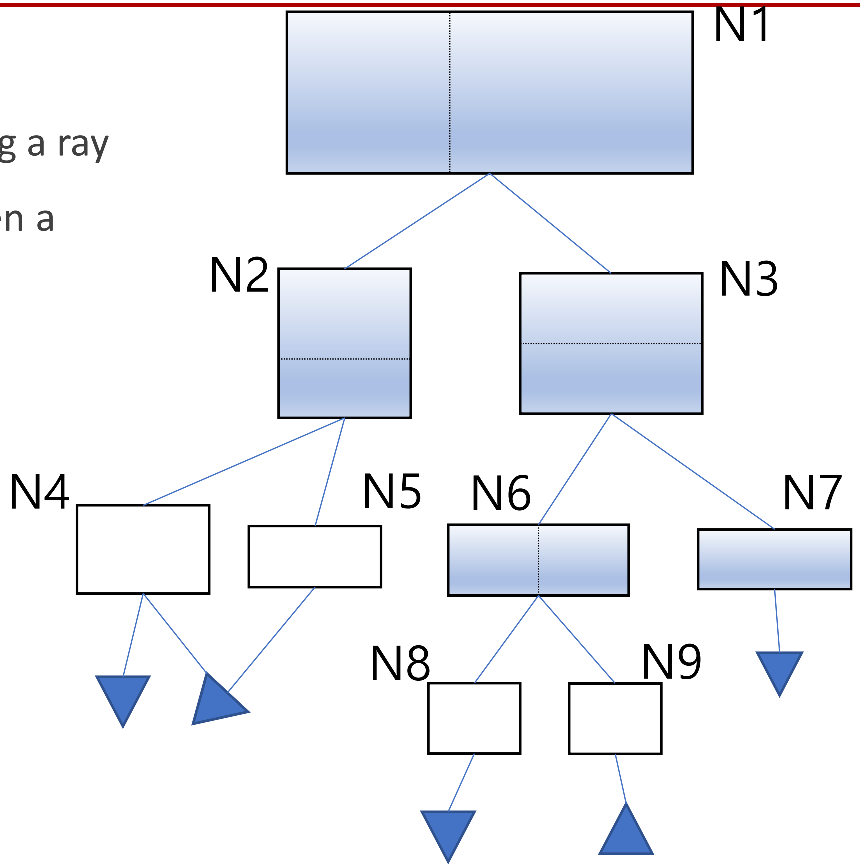
Current node: N7

Stack: N6

Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found



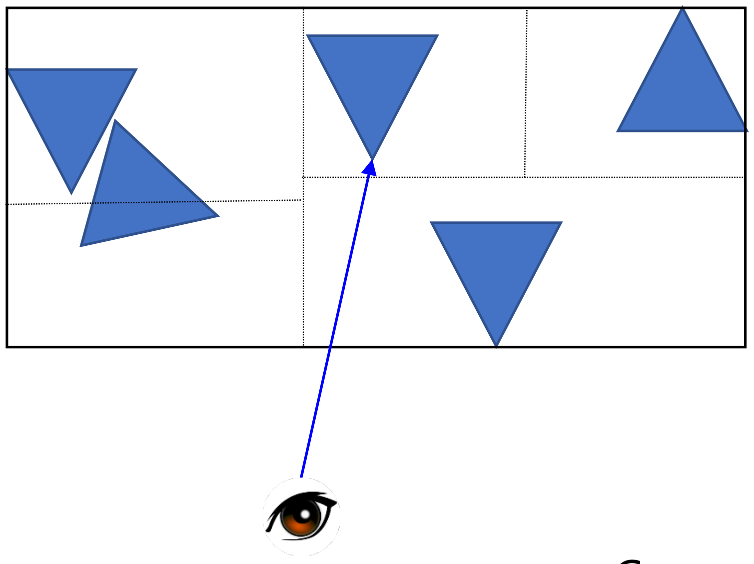


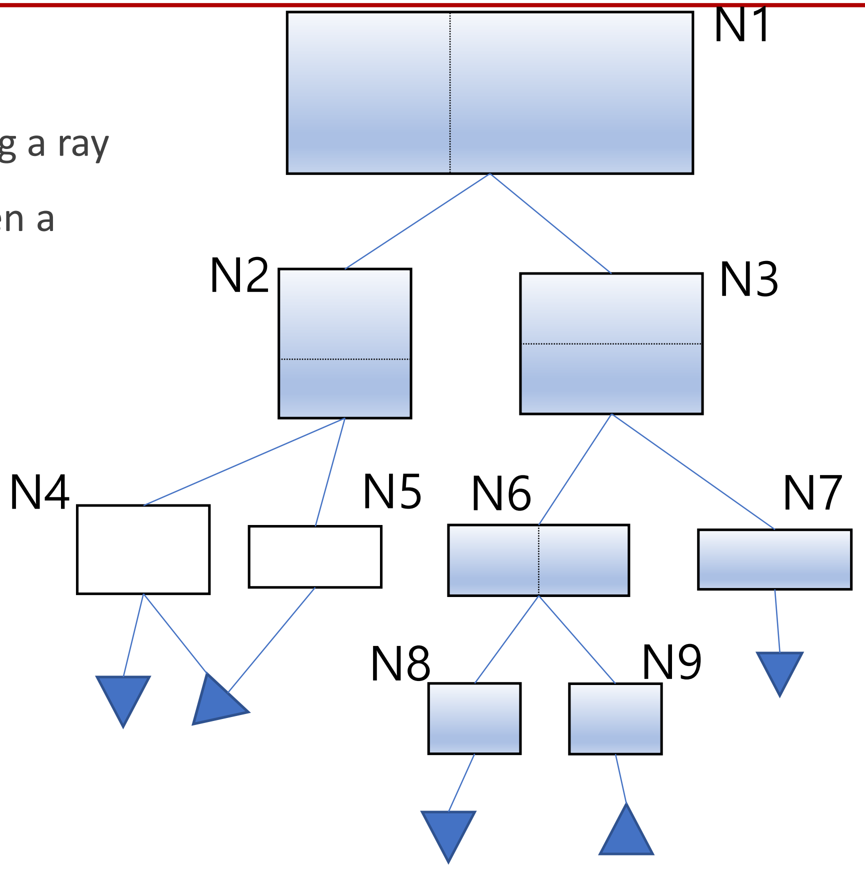
Current node:

Stack: N6

Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found



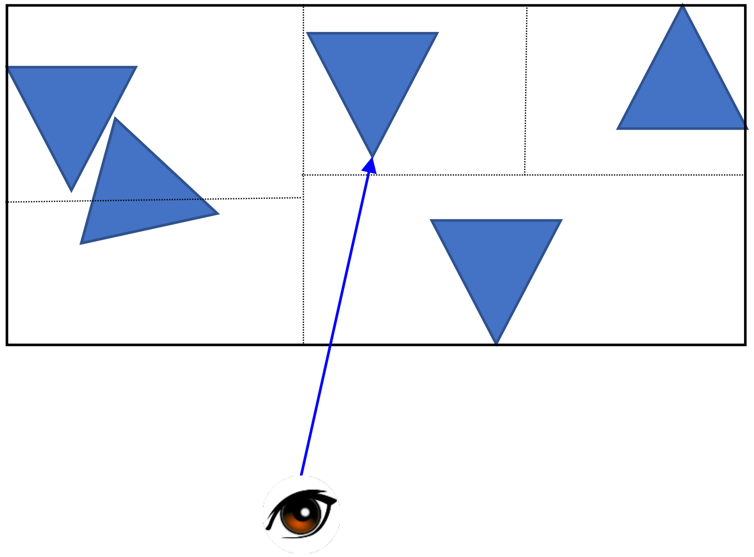


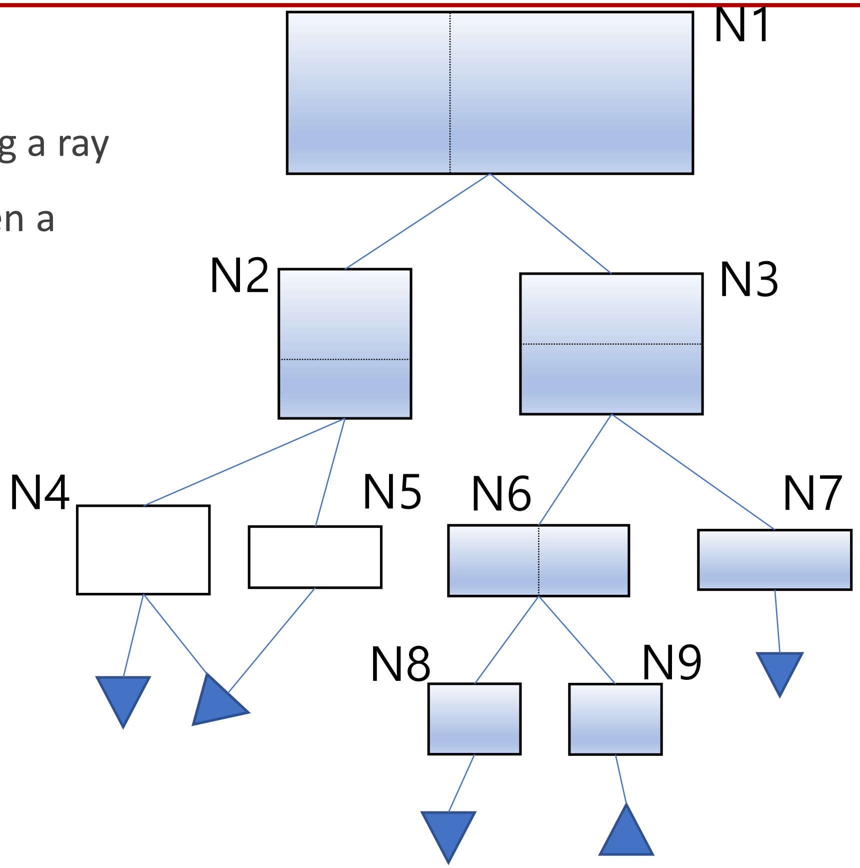
Current node: N6

Stack:

Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found



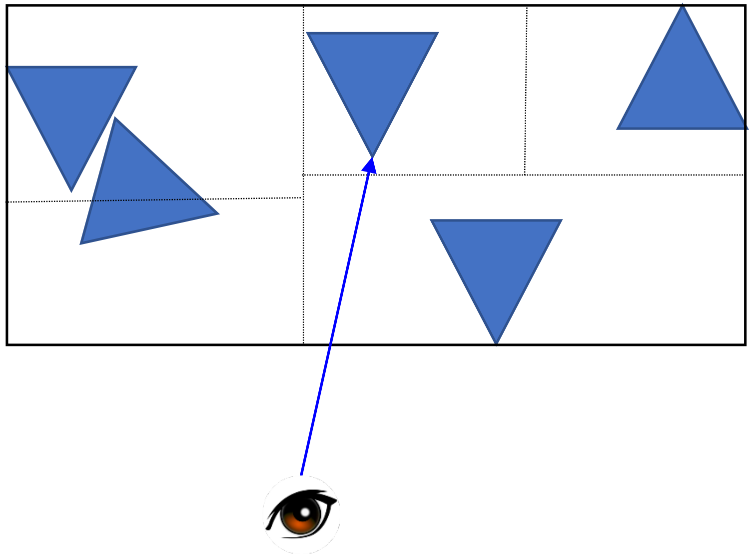


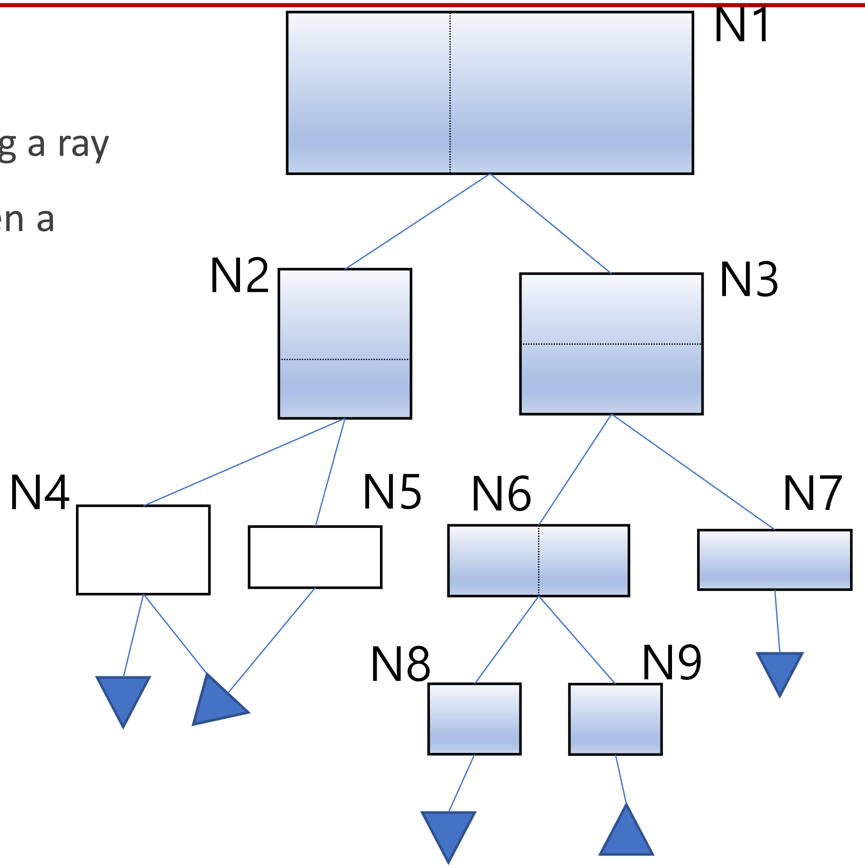
Current node: N8

Stack: N9

Kd-trees

- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found





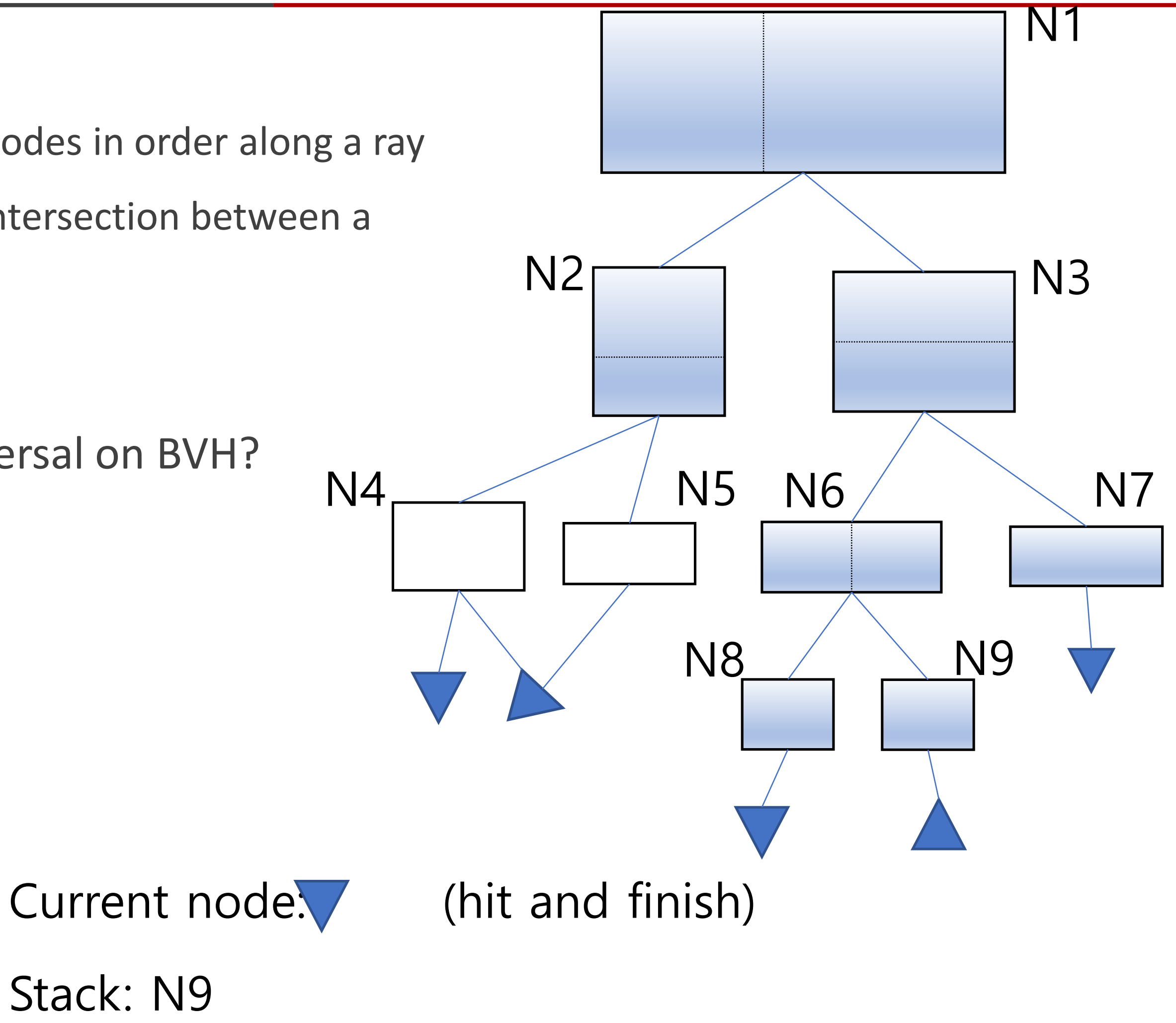
Current node: (hit and finish)

Stack: N9

Kd-trees

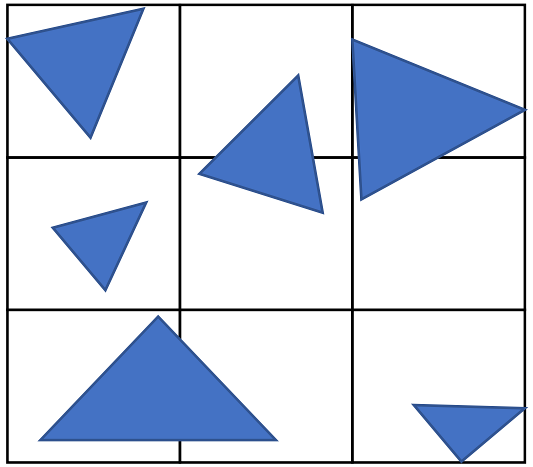
- Traversal
  - Front-to-back traversal: traverse child nodes in order along a ray
  - Can terminate traversal as soon as an intersection between a ray and triangle is found

- What's difference compared to the traversal on BVH?



Other Structures

- Uniform grids
  - Partition the whole space into equal-size cells
- Binary space partition (BSP) tree
  - Recursively split space with planes (arbitrary orientations)
  - Kd-tree is a special case of BSP tree: it uses an axis-aligned plane for partitioning
- Octree
  - Recursively split space but each inner node has 8 equal-size voxels



Discussion Points

- Axis-aligned bounding box (AABB)?
  - Cheap to compute the intersection
  - Bounding box may be too loose
  - Oriented bound box (OBB) can be better to fit objects, but this requires more complex computations
  - Other shapes (e.g., sphere) can be utilized
  - What’s the ideal bounding volume?

Discussion Points

- What's the best hierarchy?
  - Usually need to consider the following:
    - Pre-processing time (construction)
    - Run-time (rendering)
    - Memory to save all the nodes
  - Deformable objects can require run-time constructions
  - Hybrid?
    - Maintain two-level hierarchy
    - e.g., top-level: grids, low-level: kd-tree

Further Readings

- Chapter 12