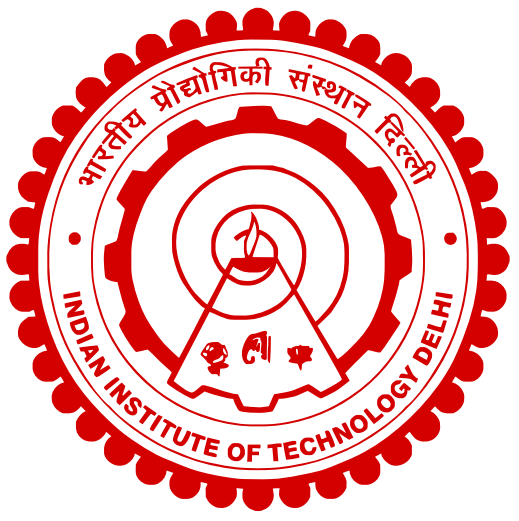


Computer Graphics

Geometric Queries

Subodh Kumar, Dept of Computer Sc. & Engg., IIT Delhi



Common Geometric Queries

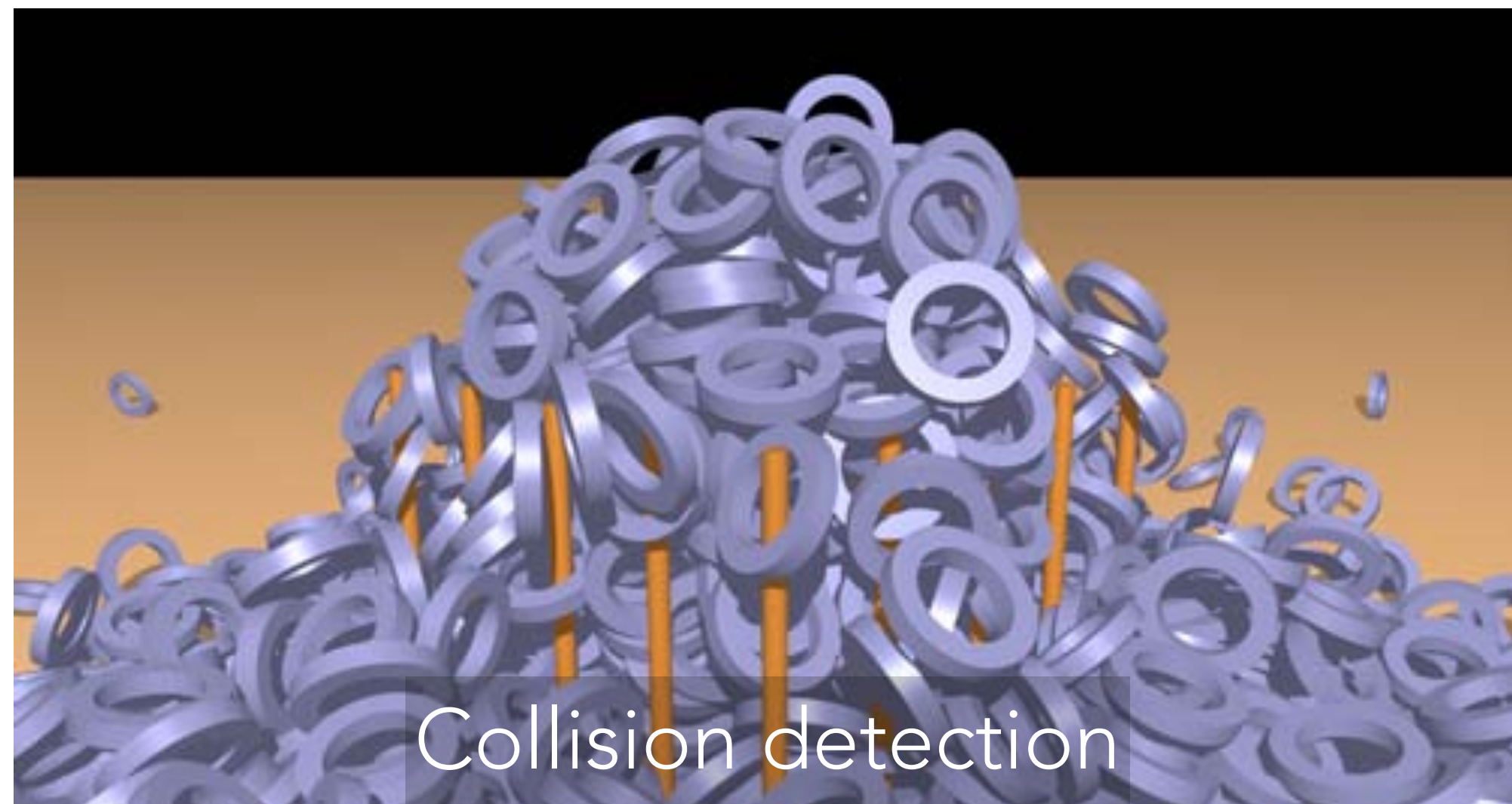
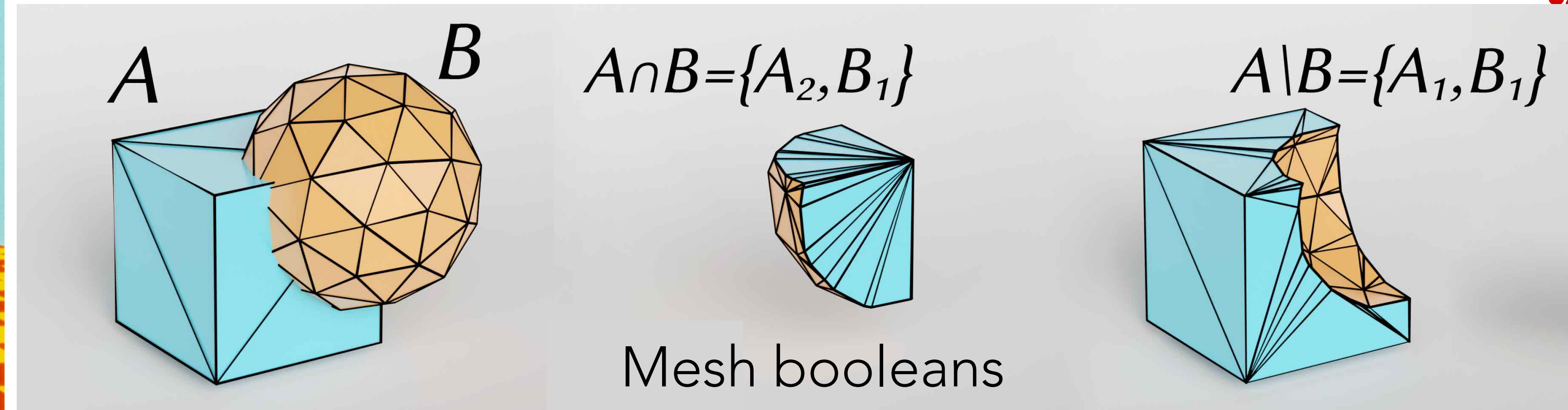
- Containment

- ➔ Is geometry inside frustum
- ➔ Is geometry hidden behind others

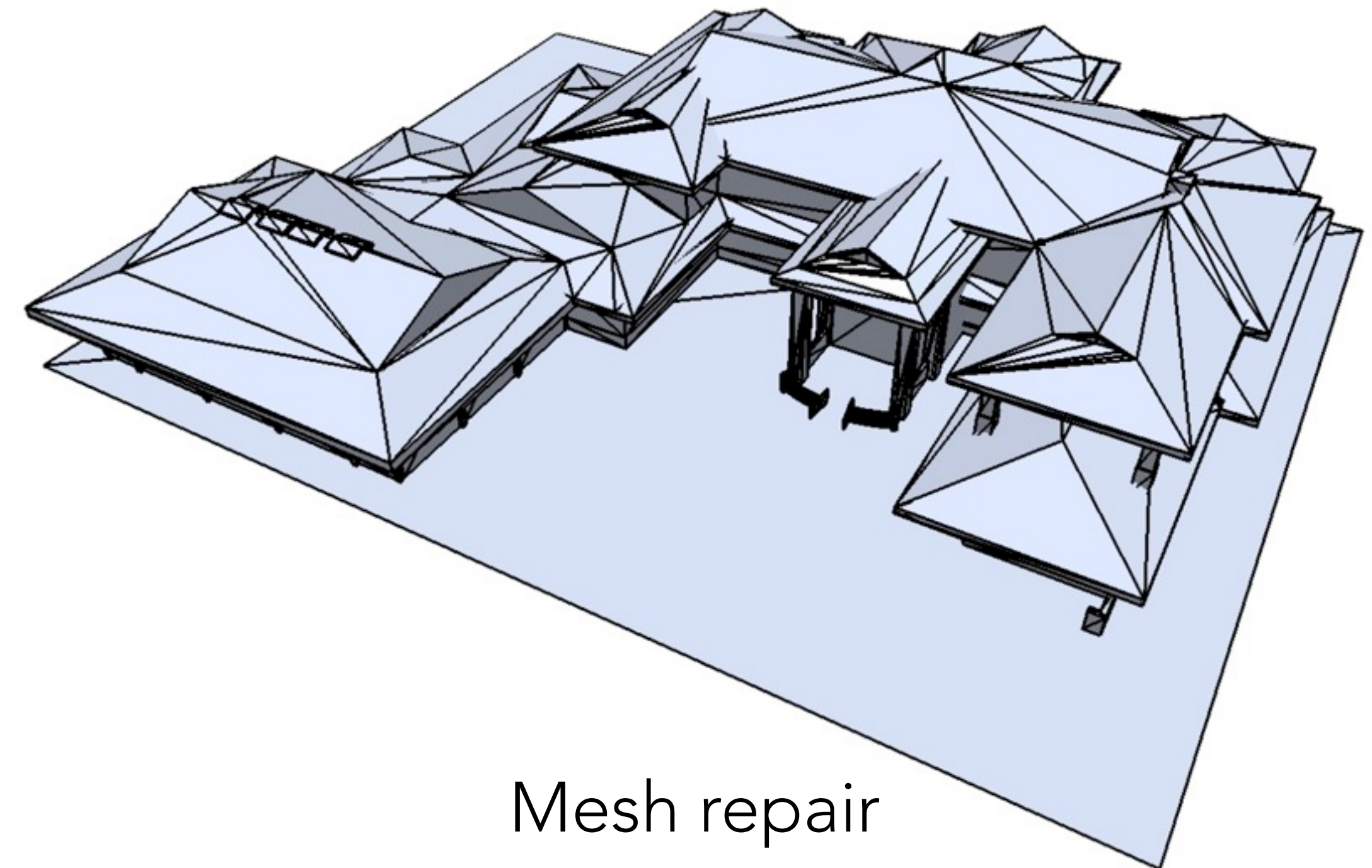
- Proximity

- ➔ Do they intersect
- ➔ Point/Curve of intersection
- ➔ Closest/Largest distance
- ➔ Is distance less than threshold

Applications



[Material borrowed from Rahul N.]



Inside/outside test

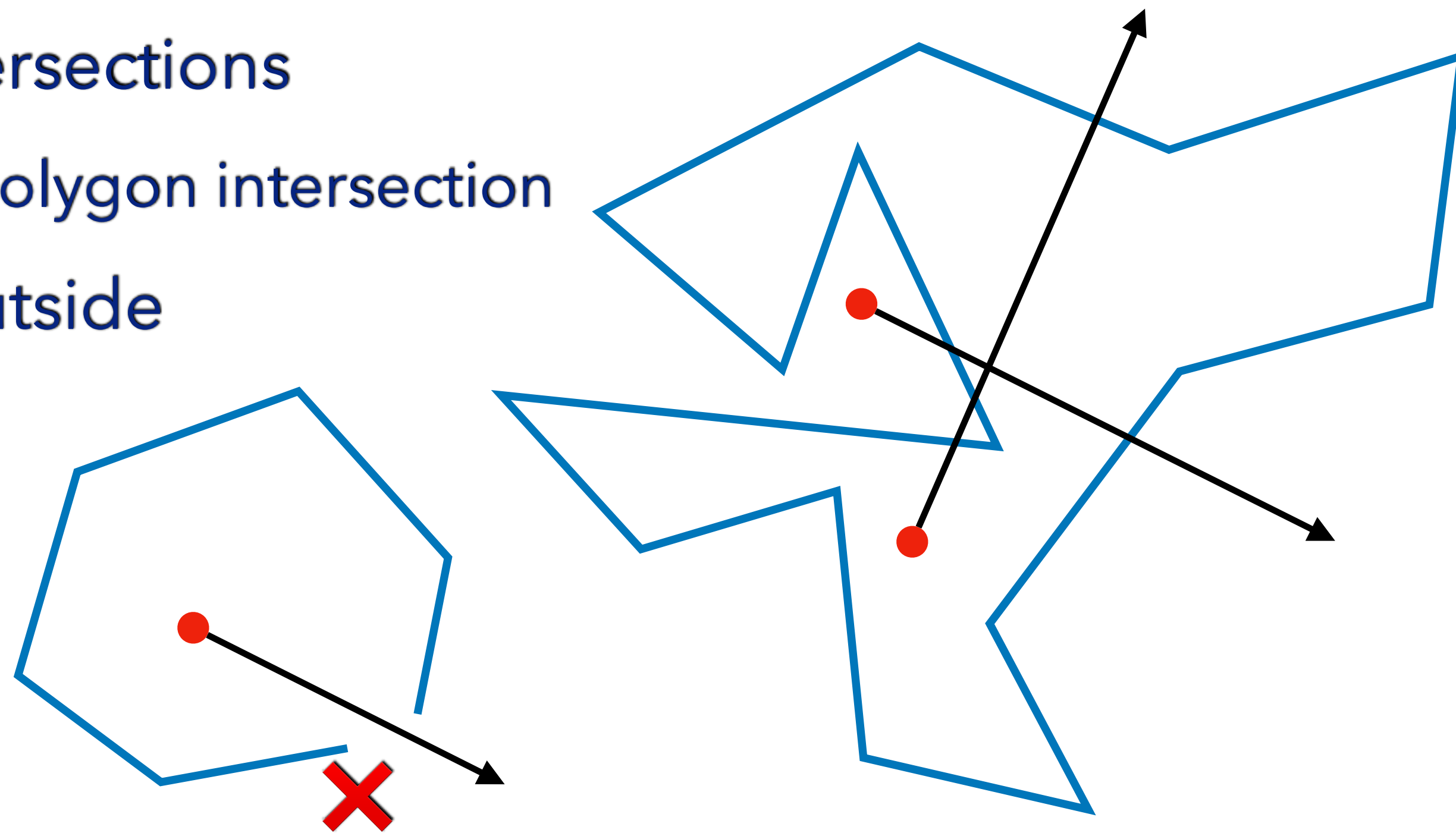


- Given a point $\mathbf{p}$, is it inside a closed mesh?



Inside/outside test

- Given a point p , is it inside a closed mesh?
- Shoot a ray in any direction
 - ➔ Count the number of intersections
 - Recall ray line-segment/polygon intersection
 - ➔ Odd $\Rightarrow$ inside, even $\Rightarrow$ outside

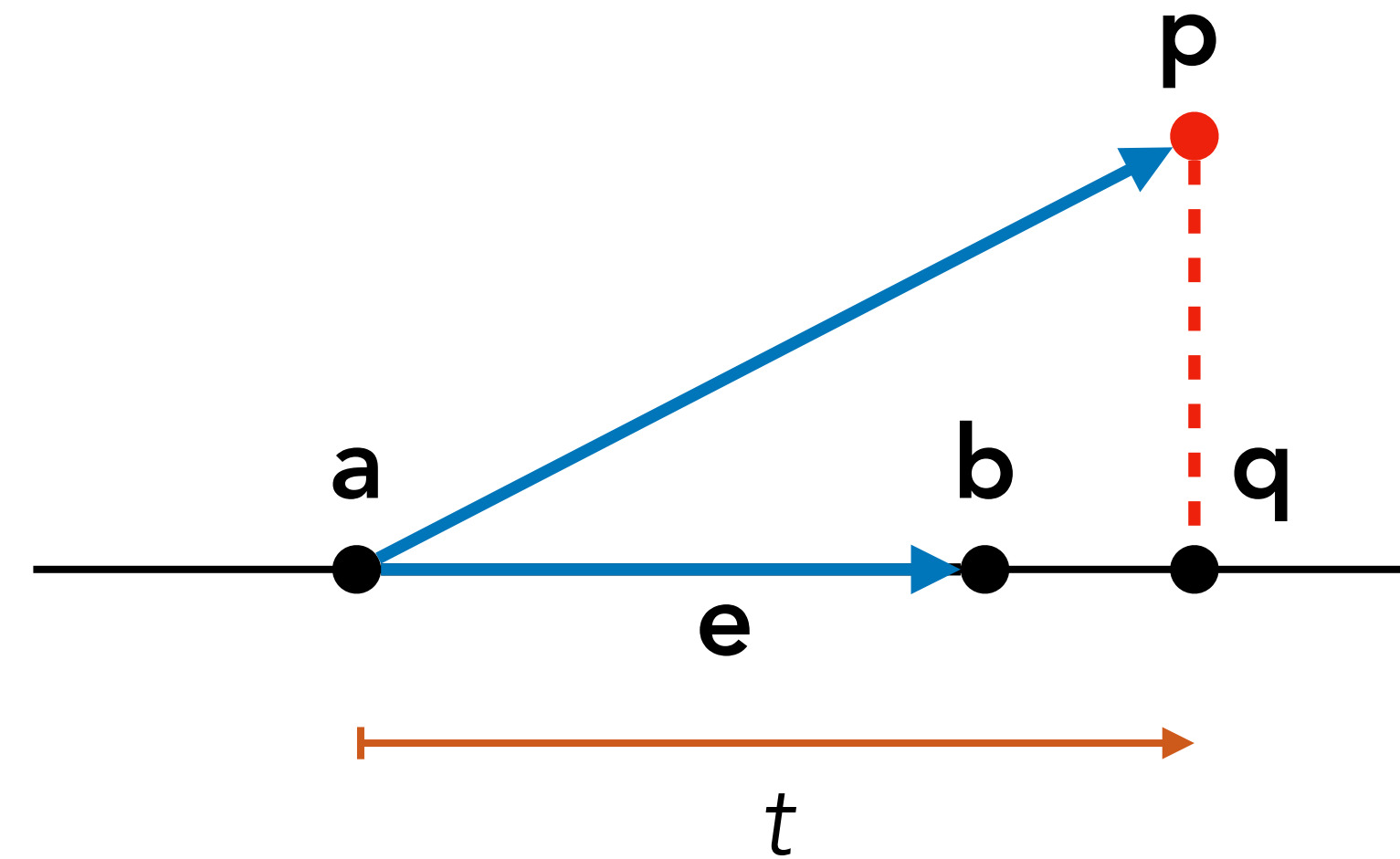




Closest point to p on a line ab

The line between $\mathbf{a}$ and $\mathbf{b}$ is the set of points $\{\mathbf{a} + t(\mathbf{b} - \mathbf{a}) : t \in \mathbb{R}\}$.

$$\mathbf{q} = \mathbf{a} + t\mathbf{e} \text{ where } \mathbf{e} = \mathbf{b} - \mathbf{a}$$



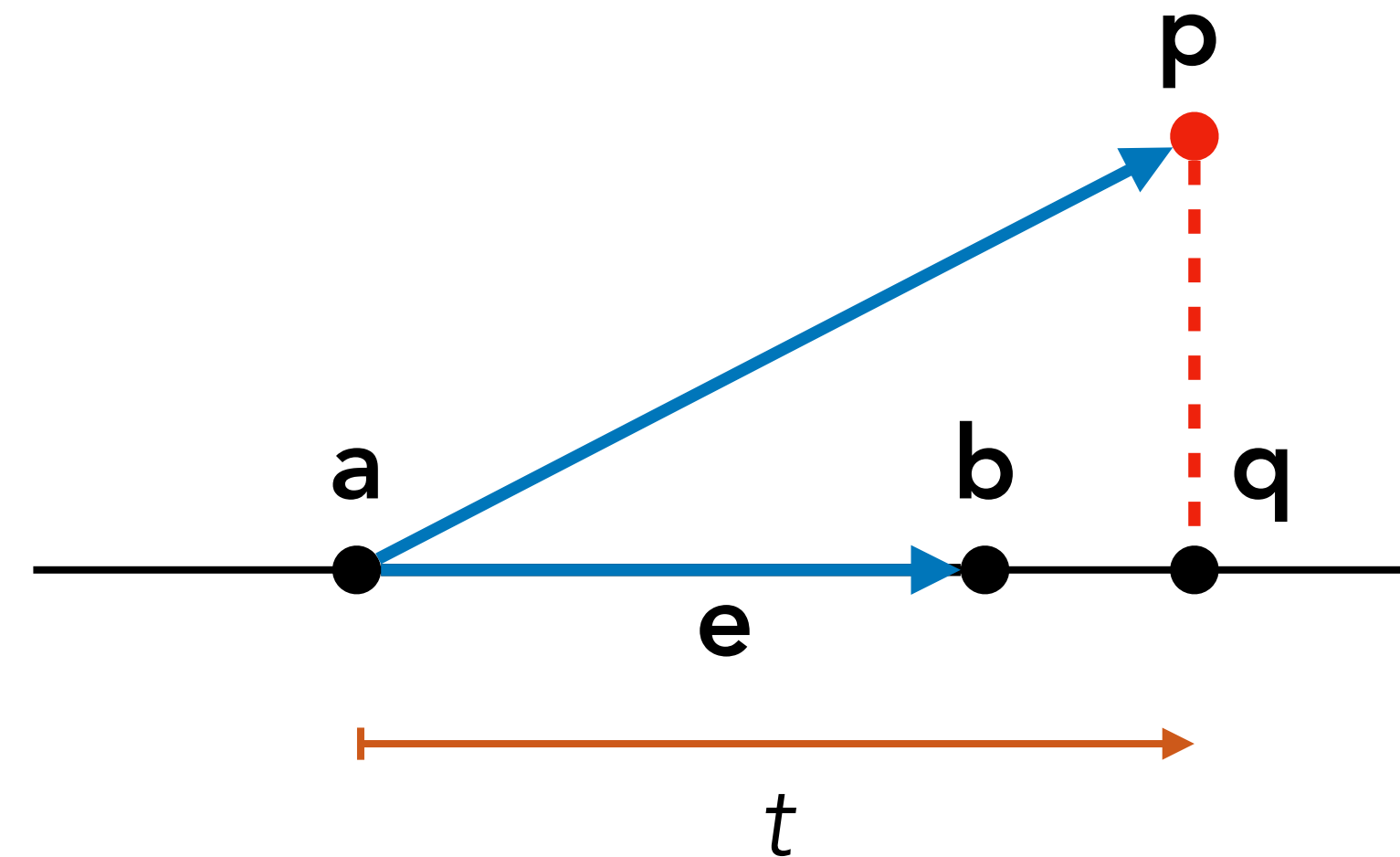


Closest point to p on a line ab

The line between $\mathbf{a}$ and $\mathbf{b}$ is the set of points $\{\mathbf{a} + t(\mathbf{b} - \mathbf{a}) : t \in \mathbb{R}\}$.

$\mathbf{q} = \mathbf{a} + t\mathbf{e}$ where $\mathbf{e} = \mathbf{b} - \mathbf{a}$

$$t = \frac{\mathbf{e} \cdot (\mathbf{p} - \mathbf{a})}{|\mathbf{e}|^2}$$





Closest point to p on a line ab

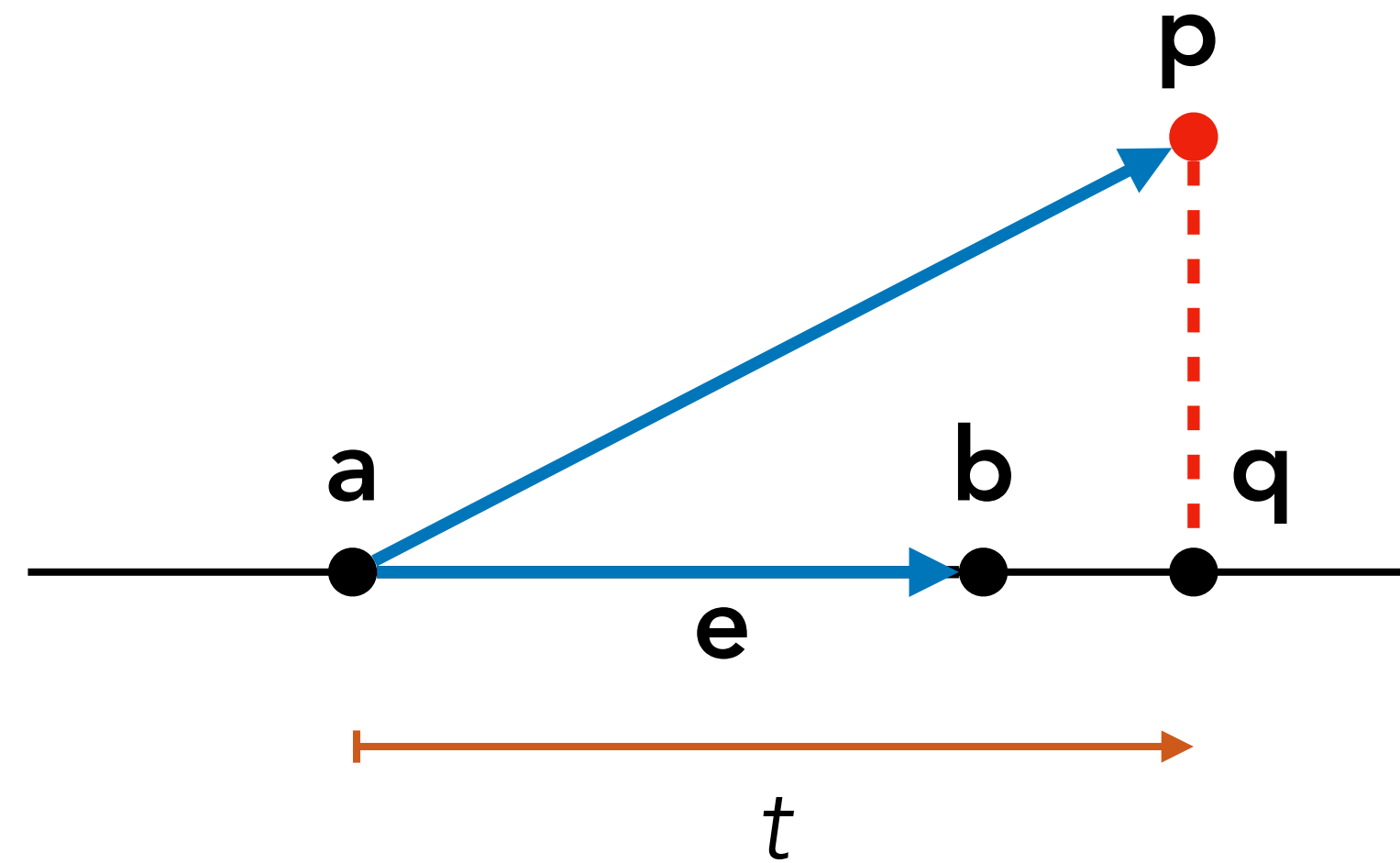
The line between $\mathbf{a}$ and $\mathbf{b}$ is the set of points $\{\mathbf{a} + t(\mathbf{b} - \mathbf{a}) : t \in \mathbb{R}\}$.

$\mathbf{q} = \mathbf{a} + t\mathbf{e}$ where $\mathbf{e} = \mathbf{b} - \mathbf{a}$

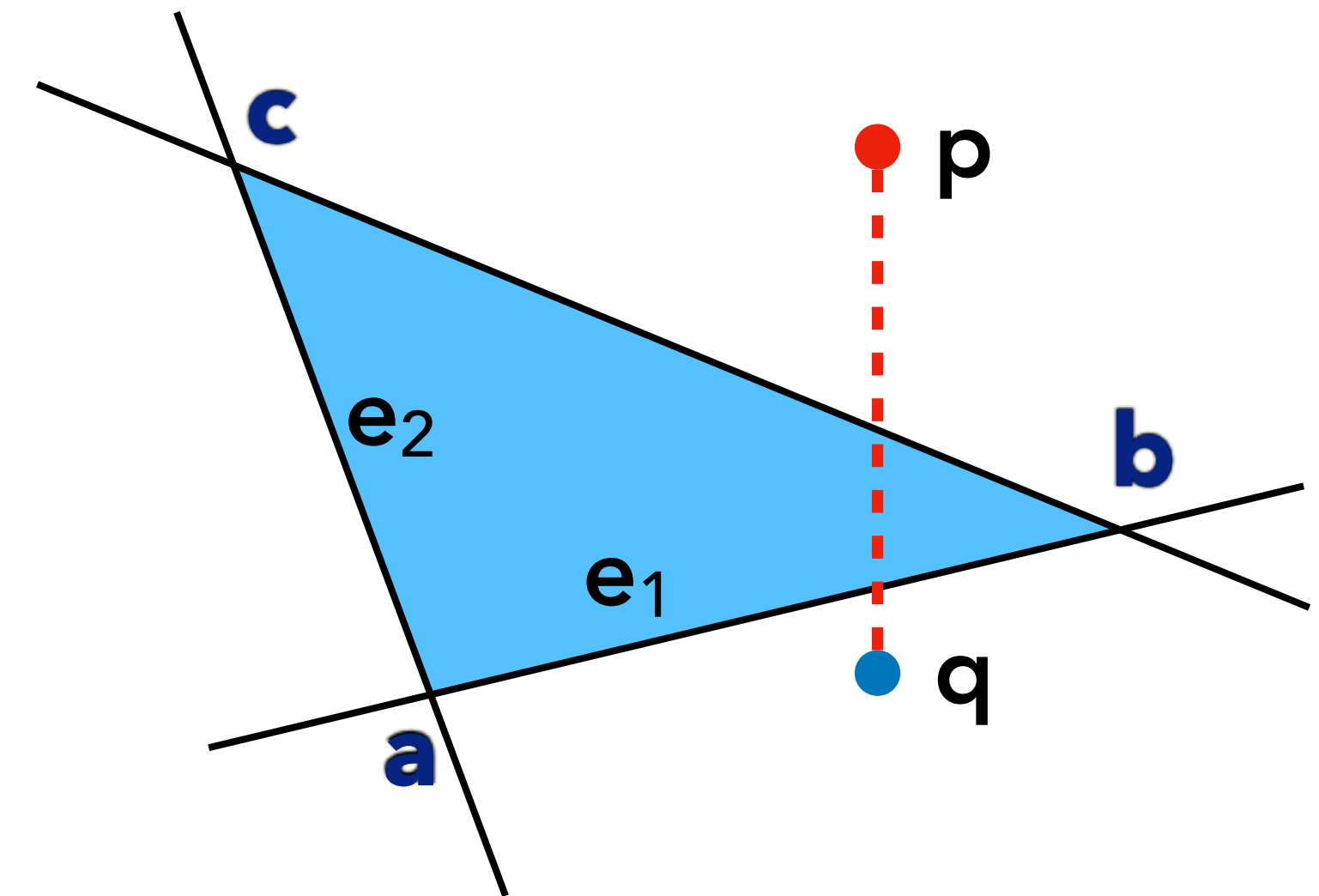
$$t = \frac{\mathbf{e} \cdot (\mathbf{p} - \mathbf{a})}{|\mathbf{e}|^2}$$

Closest to line segment $\mathbf{ab}$?

Clamp t to $[0-1]$



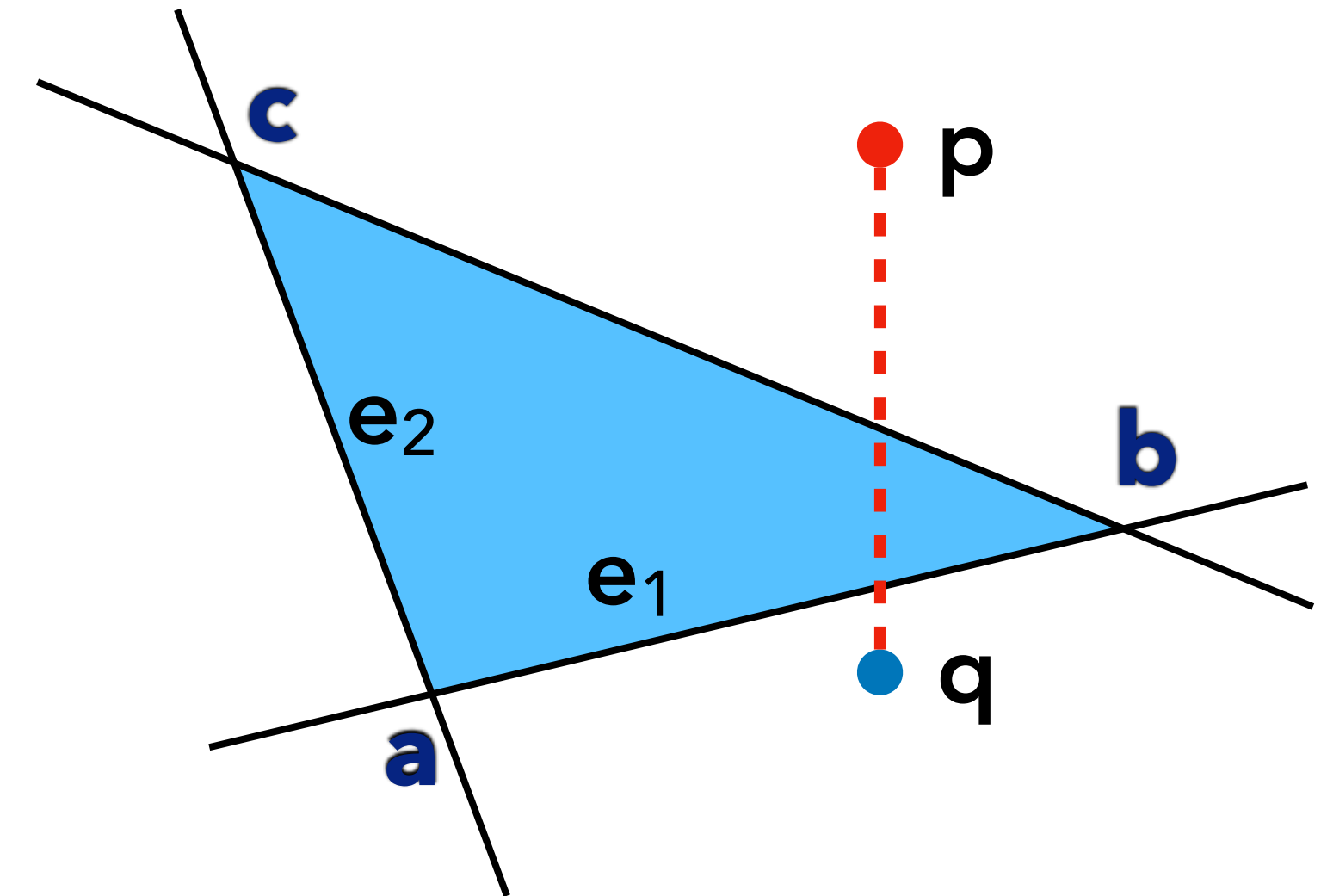
Closest point to p on triangle abc



Closest point to p on triangle abc



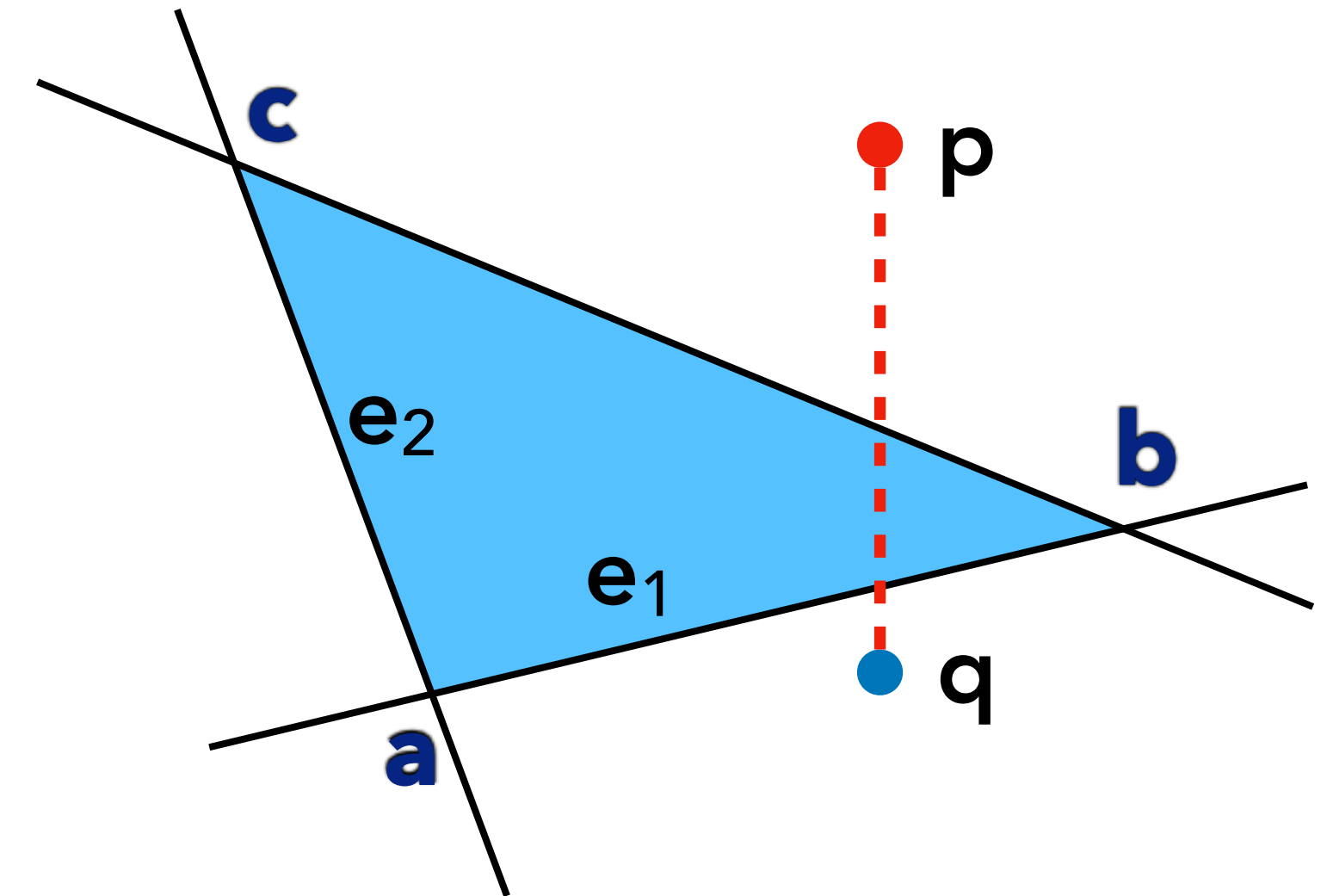
- Closest point on the plane of the triangle?
- Equation of the plane of the triangle with normal $\mathbf{n}$?
 - ⇒ $\mathbf{n}^T \mathbf{q} = \mathbf{n}^T \mathbf{a}$
 - ⇒ Check if $\mathbf{q}$ is within the triangle (barycentric coordinates?)





Closest point to p on triangle abc

- Closest point on the plane of the triangle?
- Equation of the plane of the triangle with normal $\mathbf{n}$?
 - ⇒ $\mathbf{n}^T \mathbf{q} = \mathbf{n}^T \mathbf{a}$
 - ⇒ Check if $\mathbf{q}$ is within the triangle (barycentric coordinates?)
- OR
 - ⇒ $\mathbf{a} + \beta \mathbf{e}_1 + \gamma \mathbf{e}_2$, where $\mathbf{e}_1 = \mathbf{b} - \mathbf{a}$, $\mathbf{e}_2 = \mathbf{c} - \mathbf{a}$
 - ⇒ $\beta, \gamma, 1 - \beta - \gamma$ (aka α) $\in [0, 1]$ within the triangle, At least one is -ve outside
 - ⇒ Solve $(\mathbf{a} + \beta \mathbf{e}_1 + \gamma \mathbf{e}_2) + h\mathbf{n} = \mathbf{p}$ for β, γ, h . $\mathbf{q} = \mathbf{a} + \beta \mathbf{e}_1 + \gamma \mathbf{e}_2$ is the closest point on the plane.
 - If $0 \leq \alpha, \beta, \gamma \leq 1$: $\mathbf{q}$ is the closest point
 - Otherwise, pick any $\alpha, \beta, \gamma < 0$, and find the closest point on corresponding opposite edge.



Line segment–triangle intersection

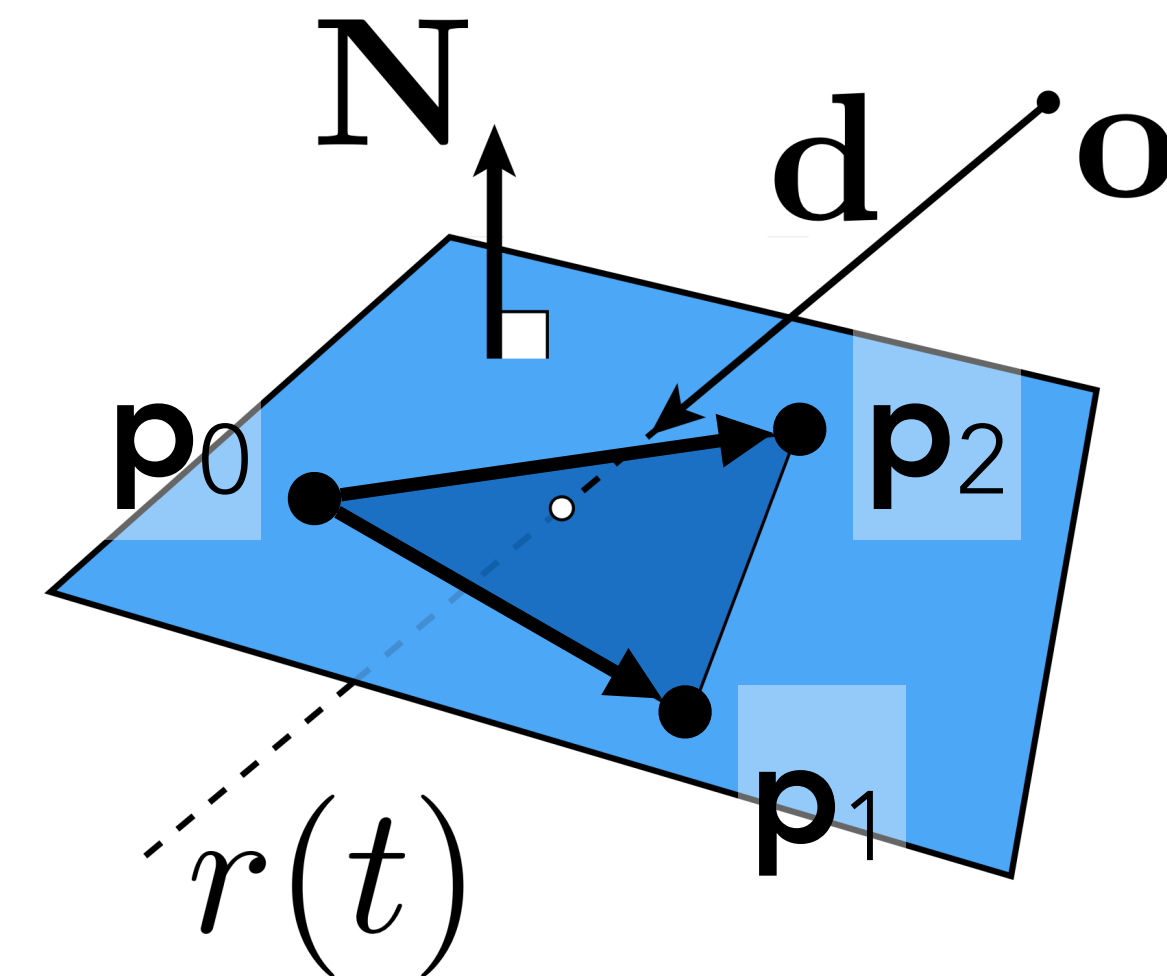


Same as intersecting a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with triangle

- ray origin $\mathbf{o} = \mathbf{a}$

- ray direction $\mathbf{d} = (\mathbf{b} - \mathbf{a})/|\mathbf{b} - \mathbf{a}|$

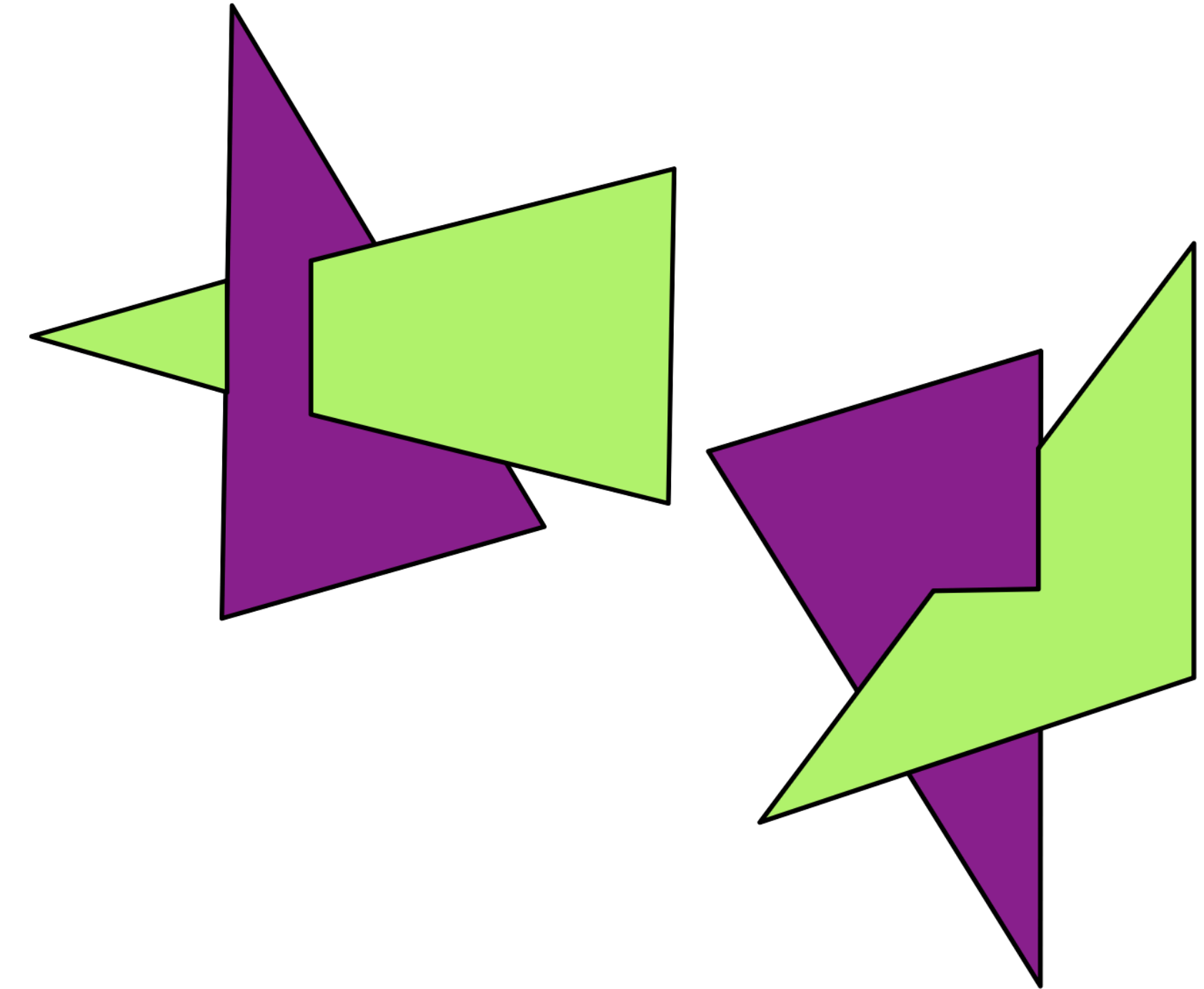
Check $0 \leq \alpha, \beta, \gamma \leq 1$ and $0 \leq t \leq 1$

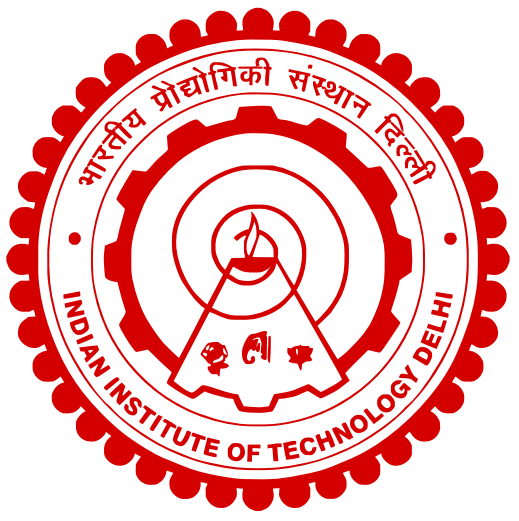


Triangle-triangle intersection



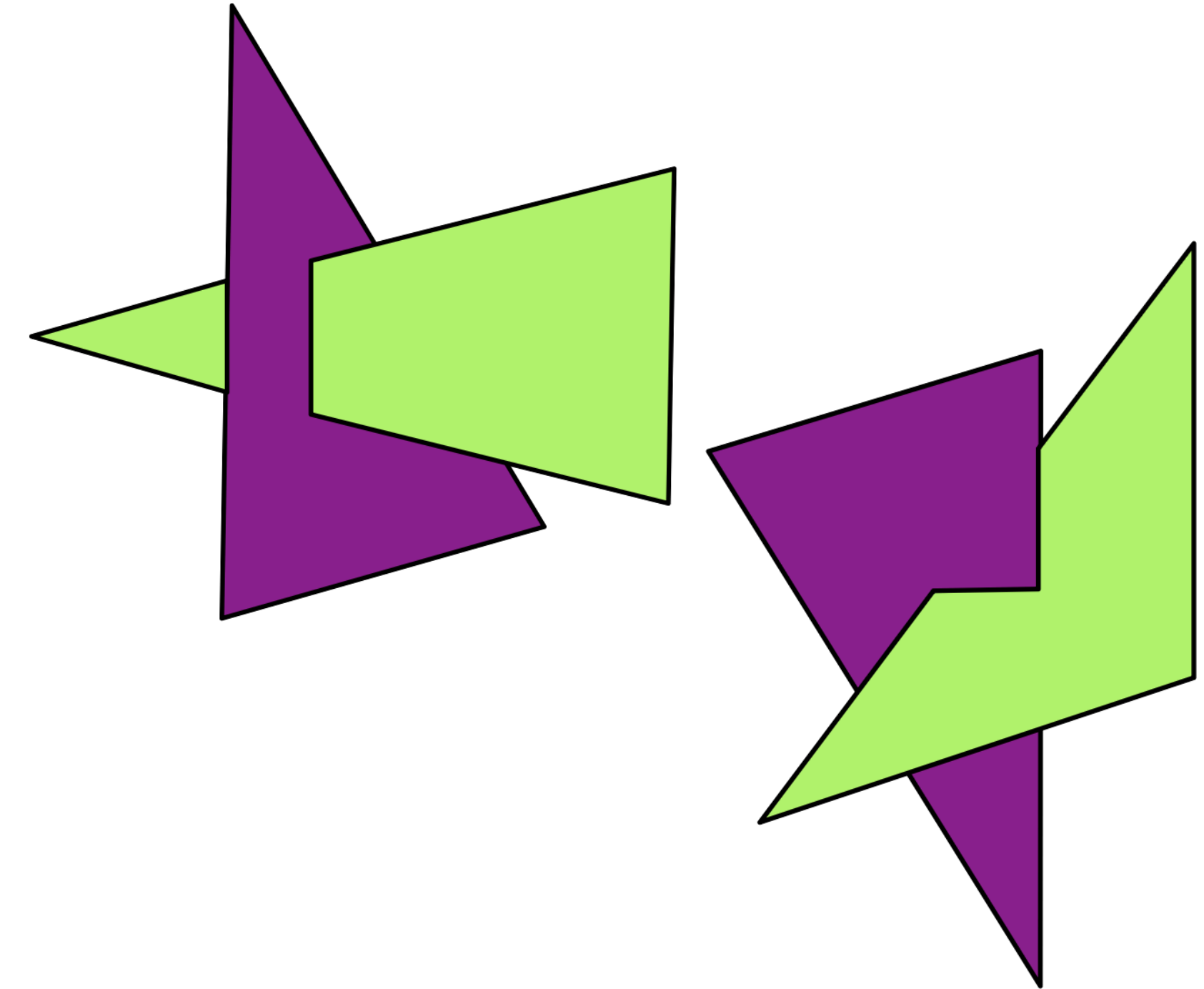
- Early culling
 - ➔ If all vertices of one triangle are on same side of other triangle plane: no intersection

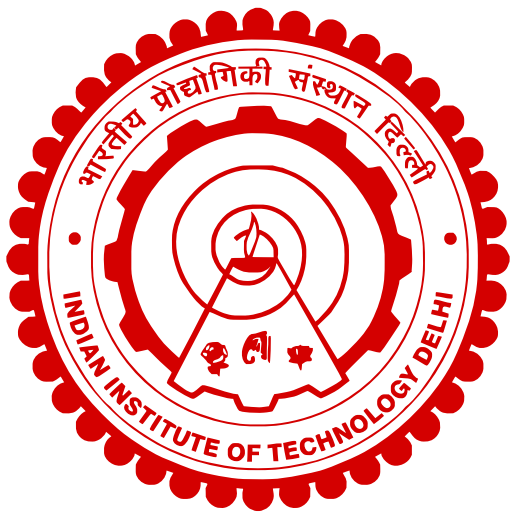




Triangle-triangle intersection

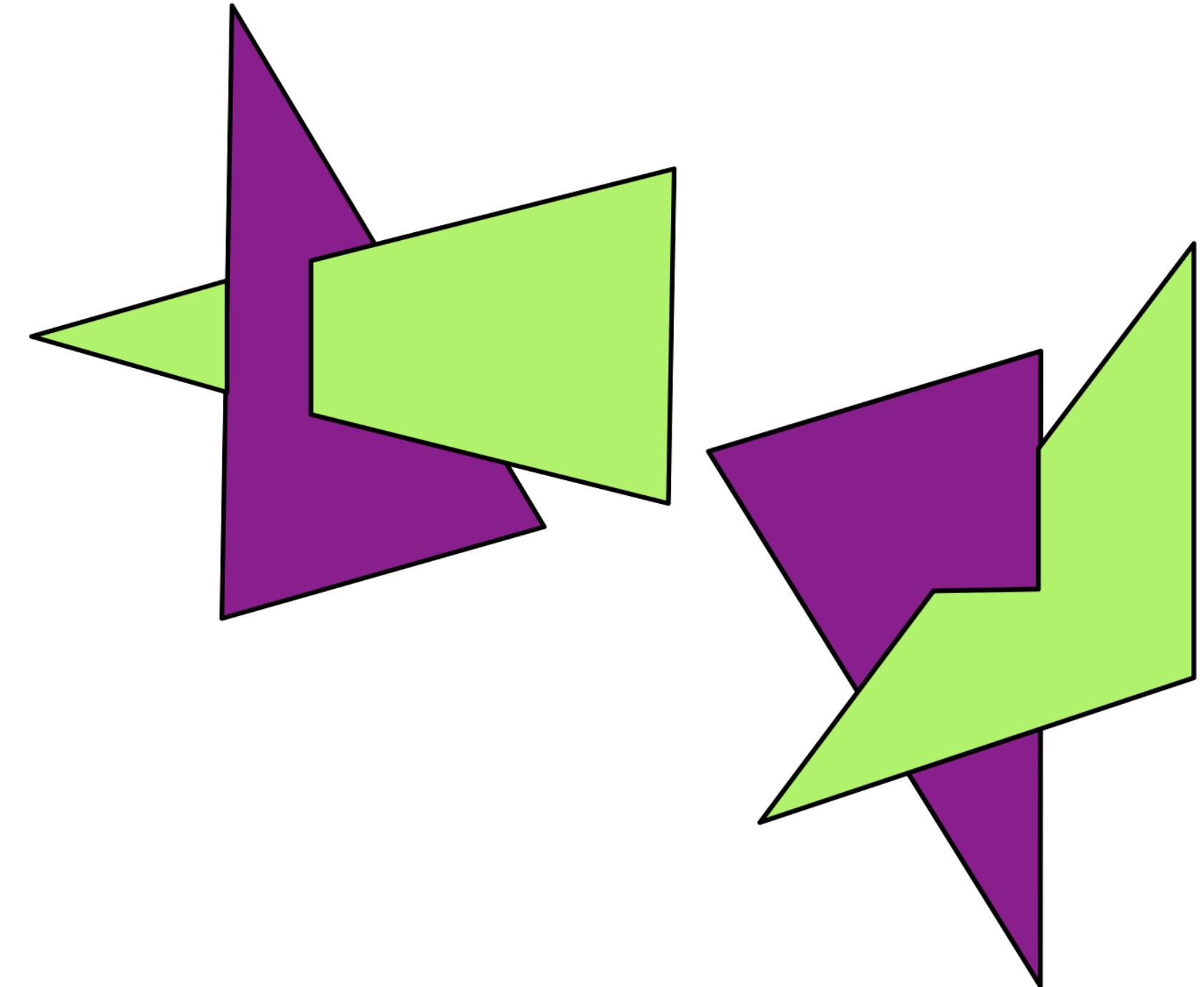
- Early culling
 - ➔ If all vertices of one triangle are on same side of other triangle plane: no intersection
- Otherwise,





Triangle-triangle intersection

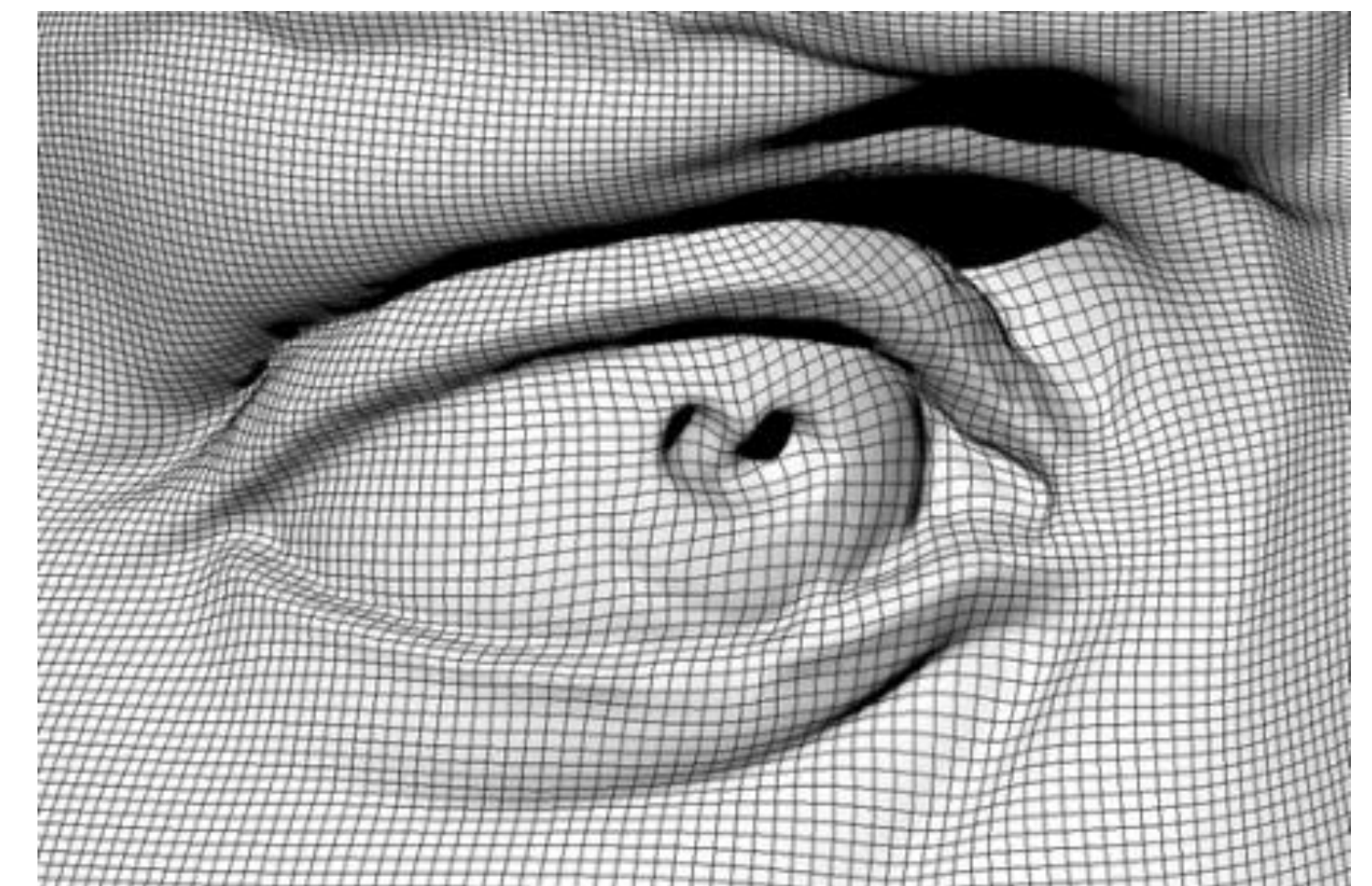
- Early culling
 - ➔ If all vertices of one triangle are on same side of other triangle plane: no intersection
- Otherwise,
 - ➔ Check if any edge of triangle T_1 intersects triangle T_2 , or vice versa

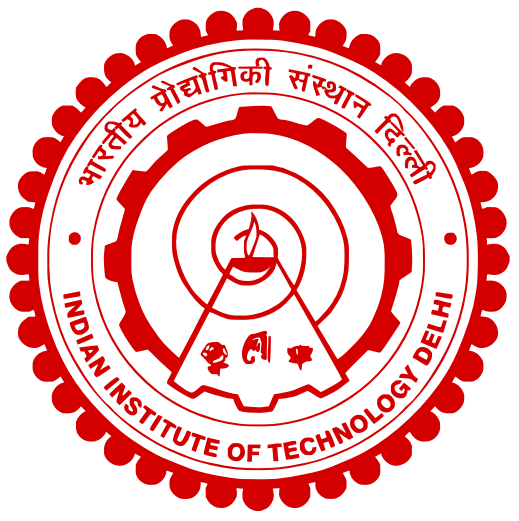


Closest point on a triangle mesh



- Naïve algorithm:
 - ➔ For each triangle, find the closest point $\mathbf{q}_i$ and its distance $|\mathbf{p} - \mathbf{q}_i|$
 - ➔ Return the $\mathbf{q}_i$ with the lowest distance
 - ➔ Inefficient for high-resolution meshes
- Spatial data structures can speed up
 - ➔ Including computing an upper bound on the closest distance, testing intersections or containment



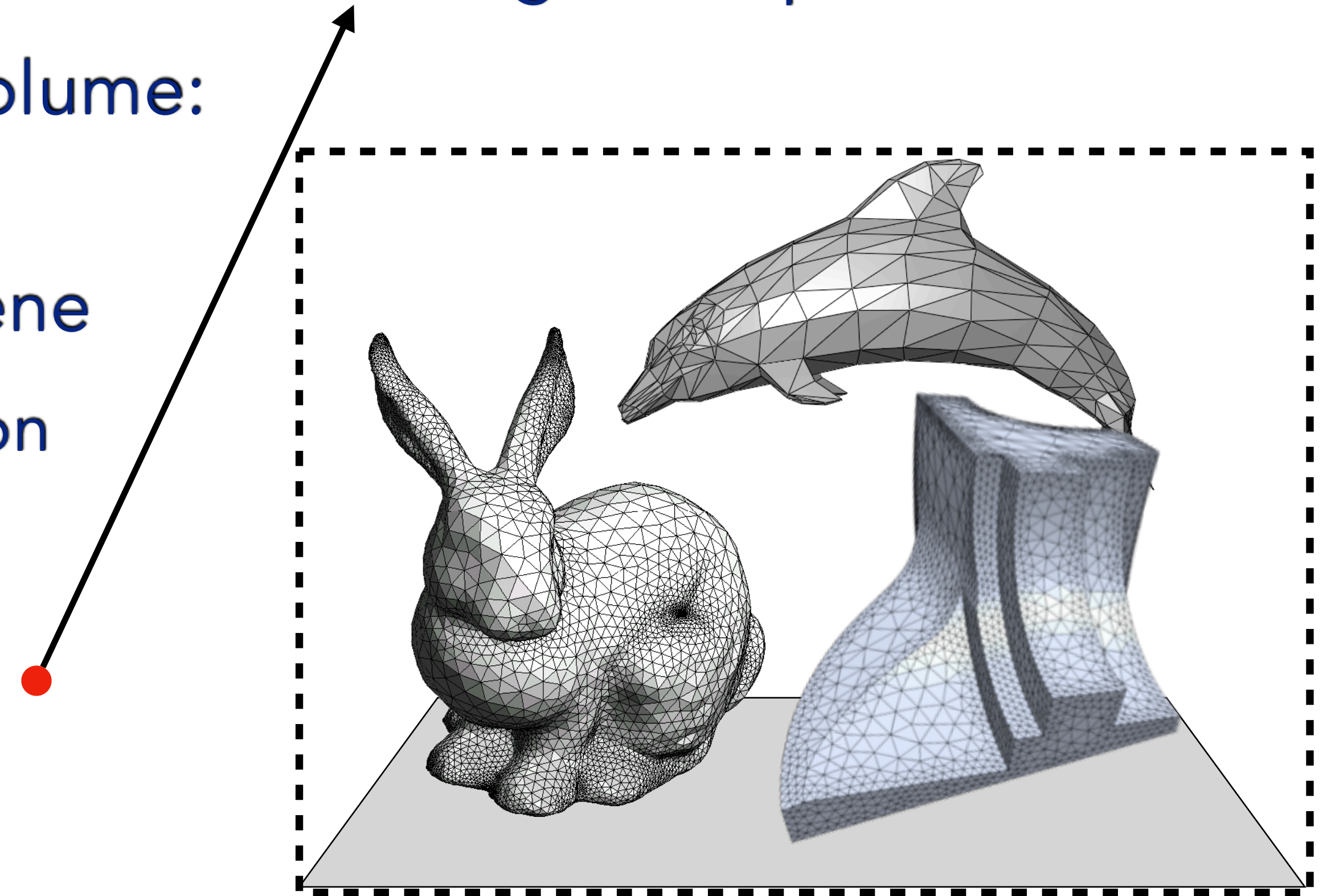


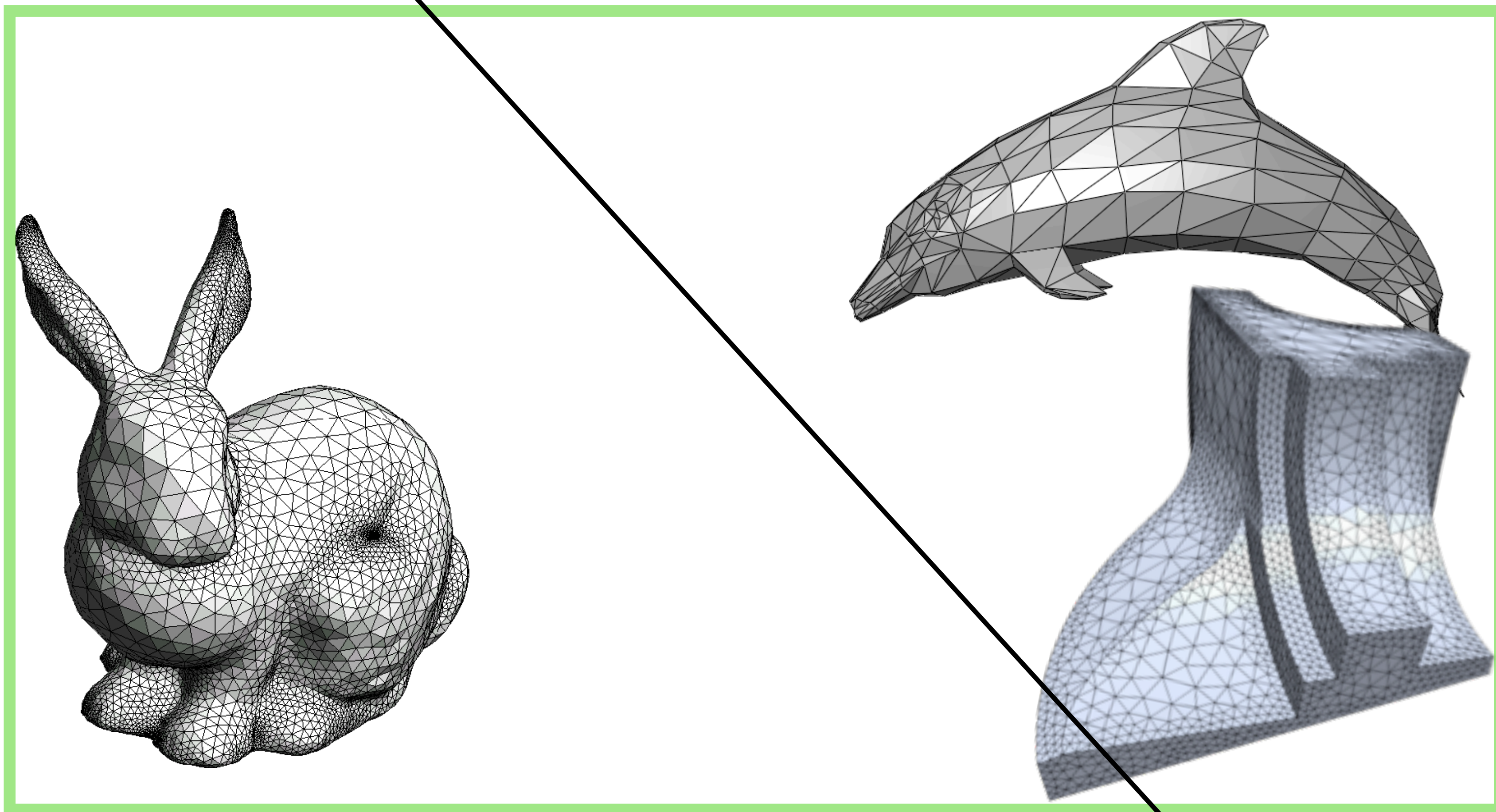
Spatial Data Structures

- Bounding Volumes
 - ➔ Axis-aligned bounding boxes
 - ➔ Oriented bounding boxes
 - ➔ k-DOPs
 - ➔ Spherical, ellipsoidal bounding boxes
- Spatial partitioning
 - ➔ Octree
 - ➔ k-D tree

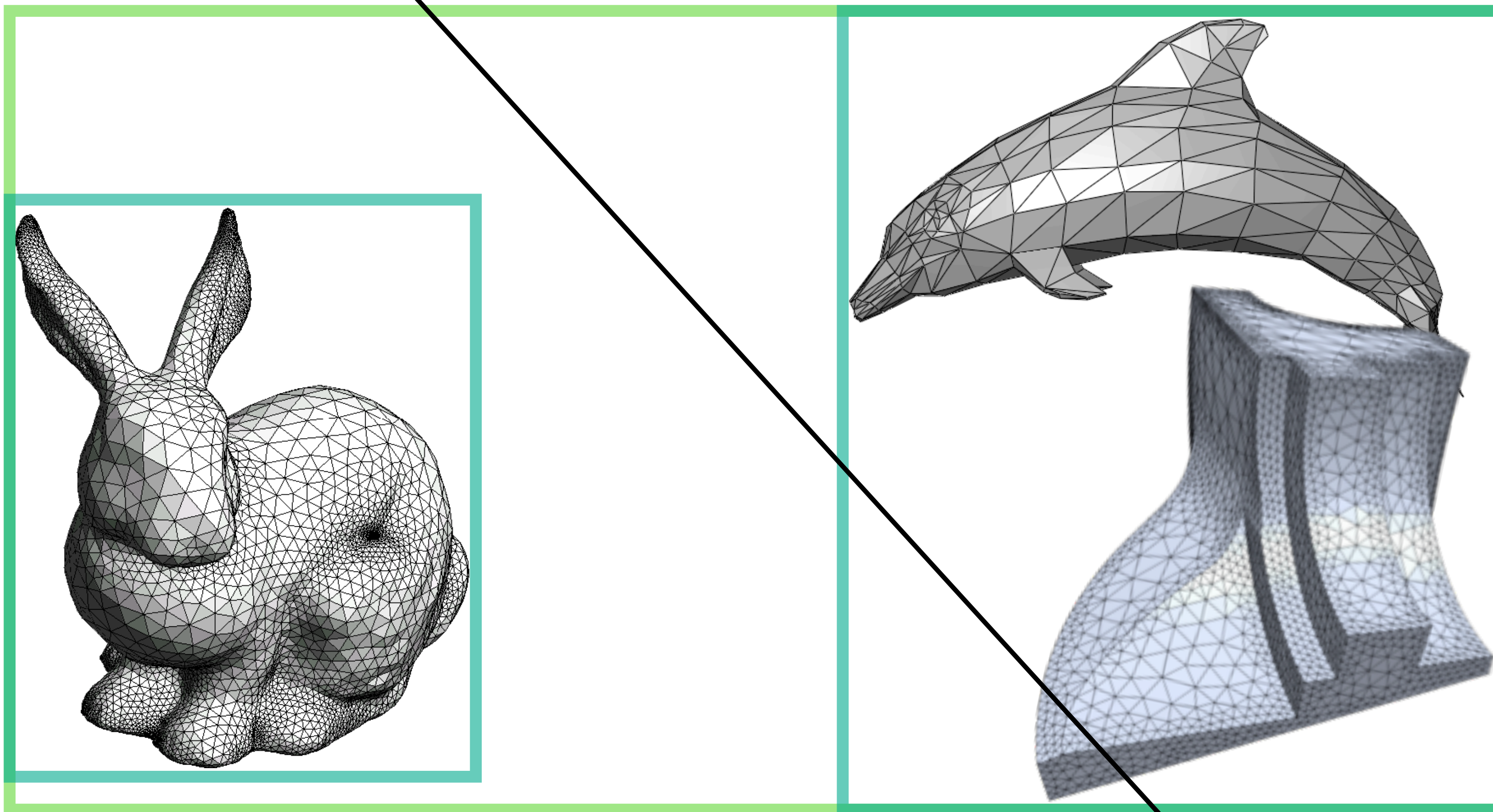
Bounding volumes

- How can we speed up, e.g., ray intersection with a large, complex scene?
- Construct a conservative **bounding** volume: all scene geometry lies inside it
- Reject rays that don't intersect the scene
 - ➔ Recall ray-box or ray-sphere intersection
- But most rays do hit the scene...

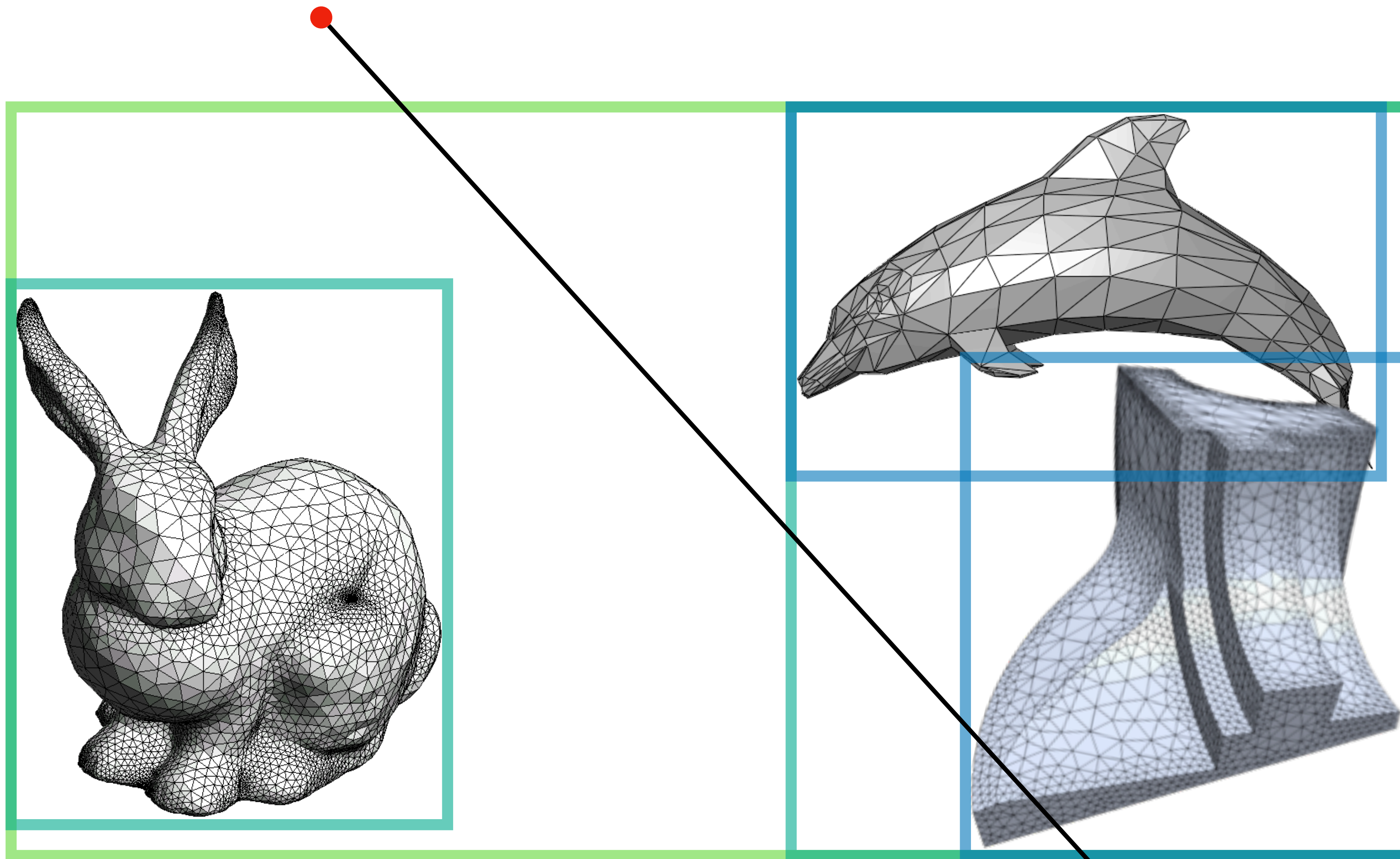




But not all separate bounding volumes



But not all separate bounding volumes

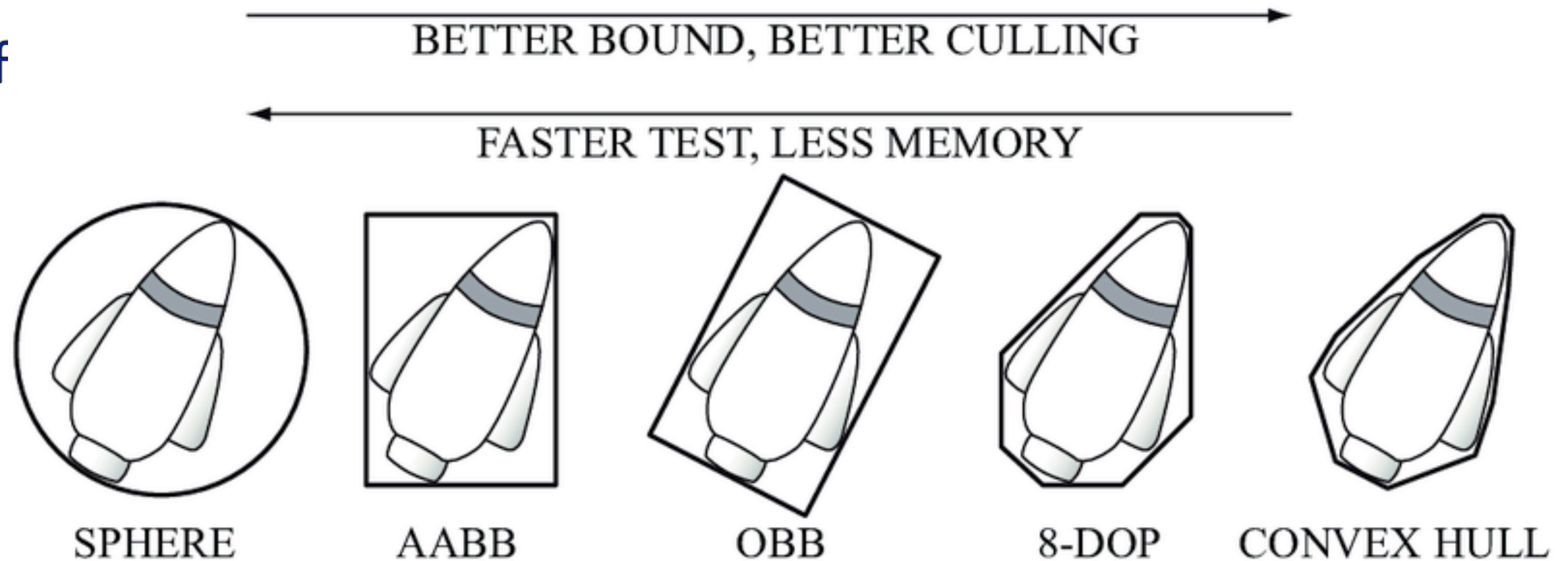


But not all separate bounding volumes

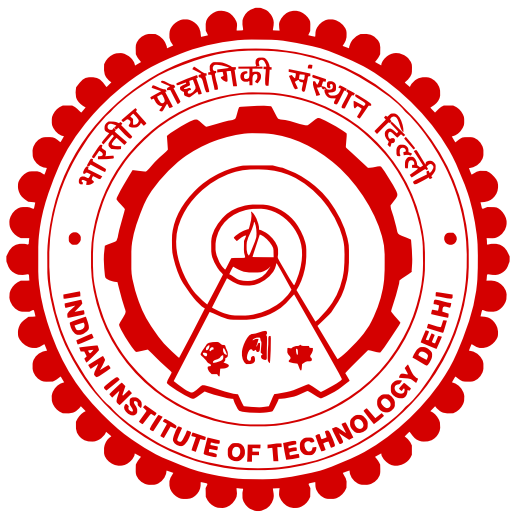
Good Bounding Volume?

- Tight (minimize # of false positives)
- Fast to intersect

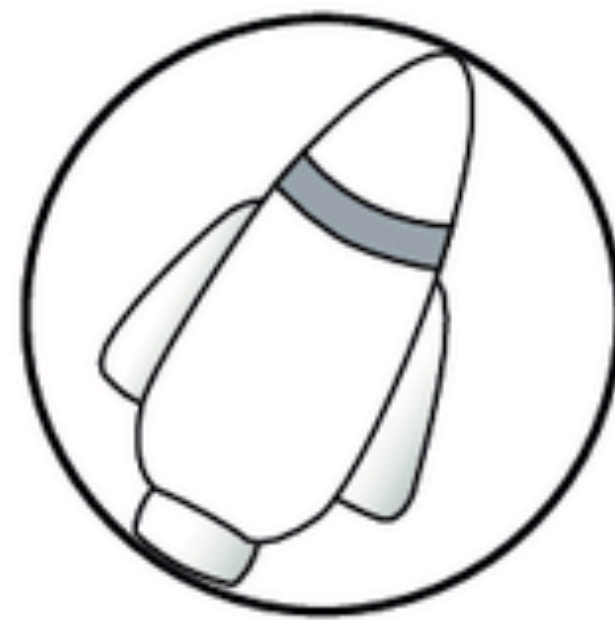
This is a tradeoff



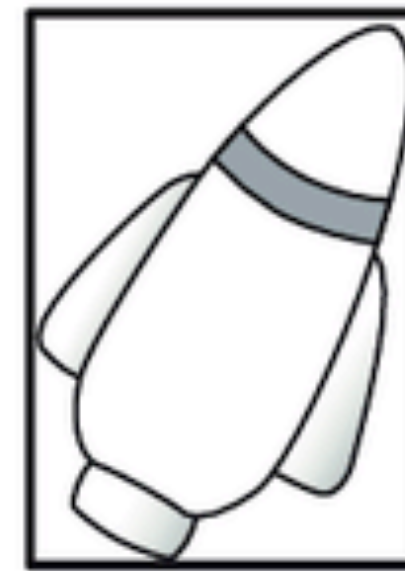
Representation of BV



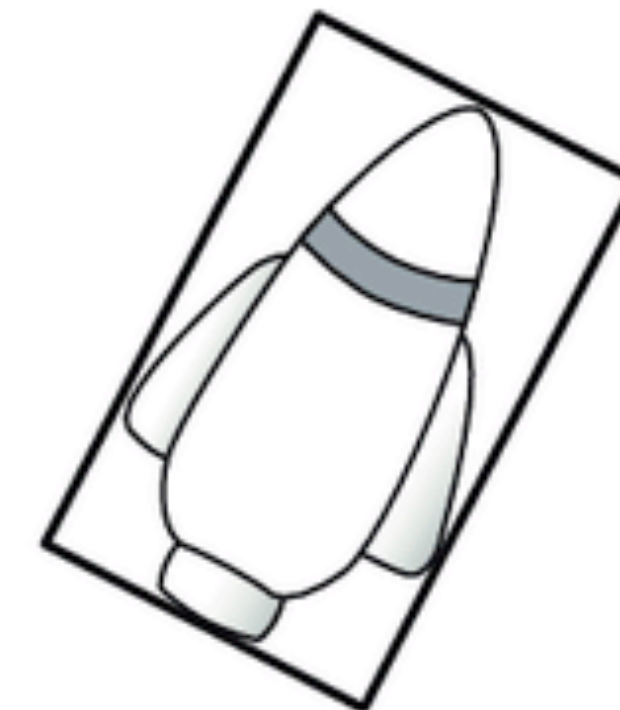
- Sphere: center and radius



SPHERE



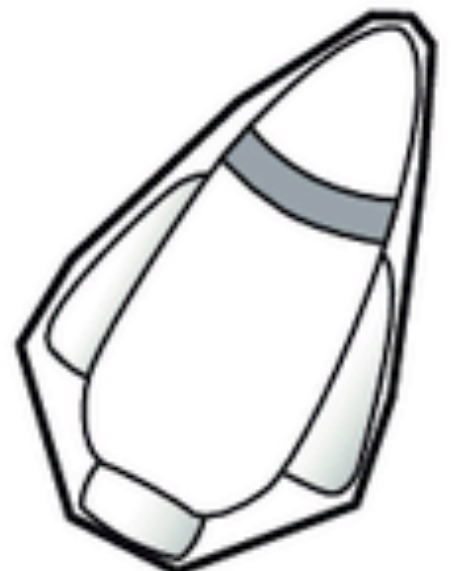
AABB



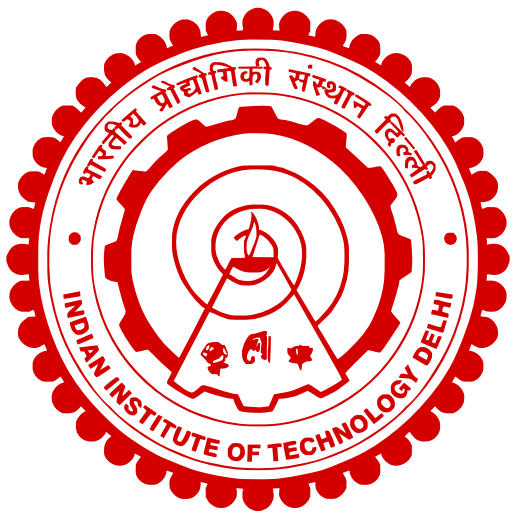
OBB



8-DOP

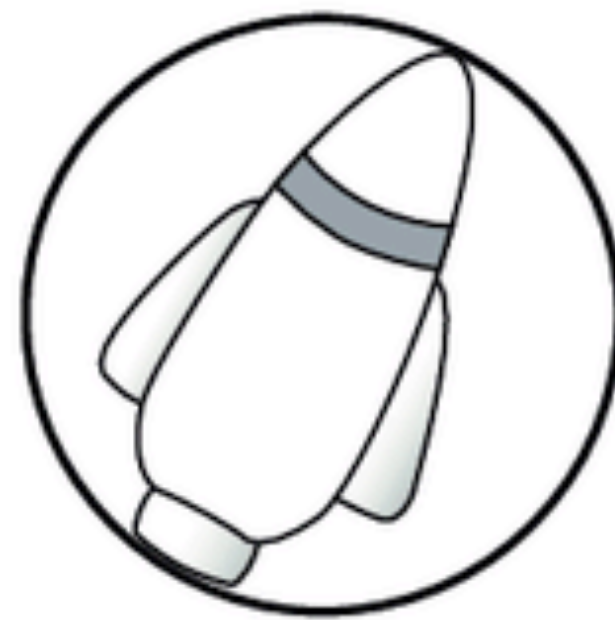


CONVEX HULL

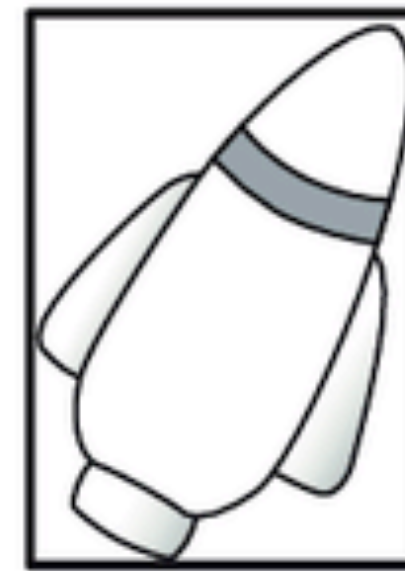


Representation of BV

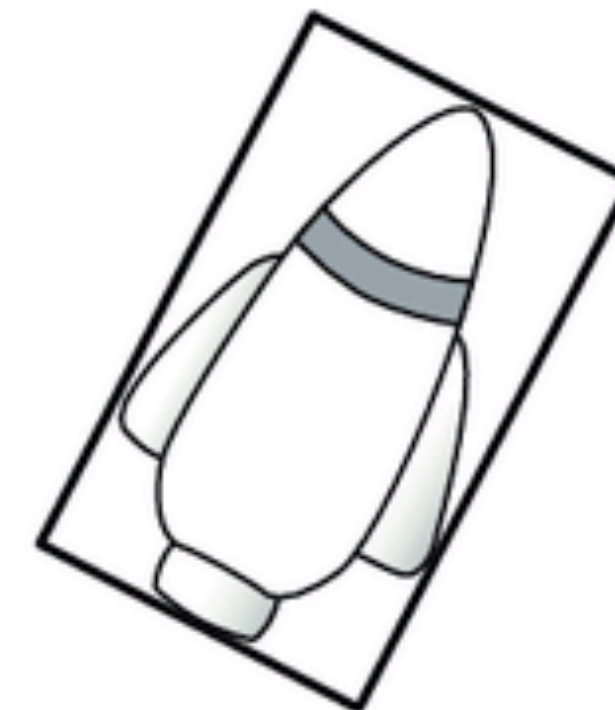
- Sphere: center and radius
- Axis-aligned bounding box: min & max along x, y, z



SPHERE



AABB



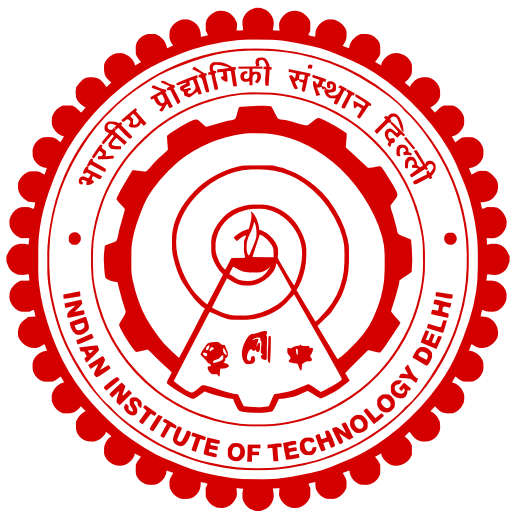
OBB



8-DOP

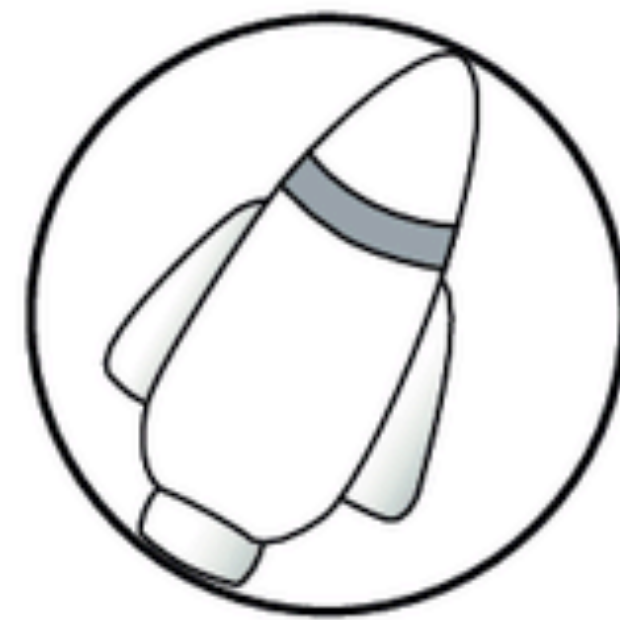


CONVEX HULL

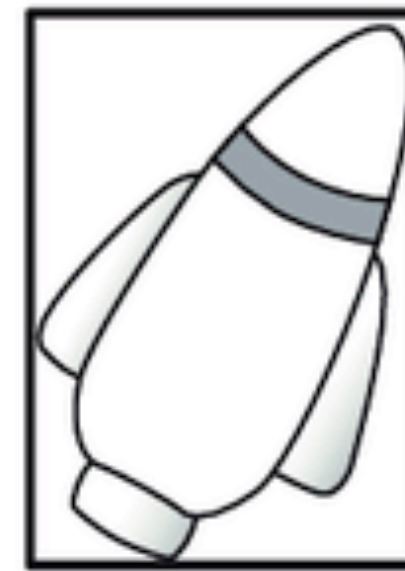


Representation of BV

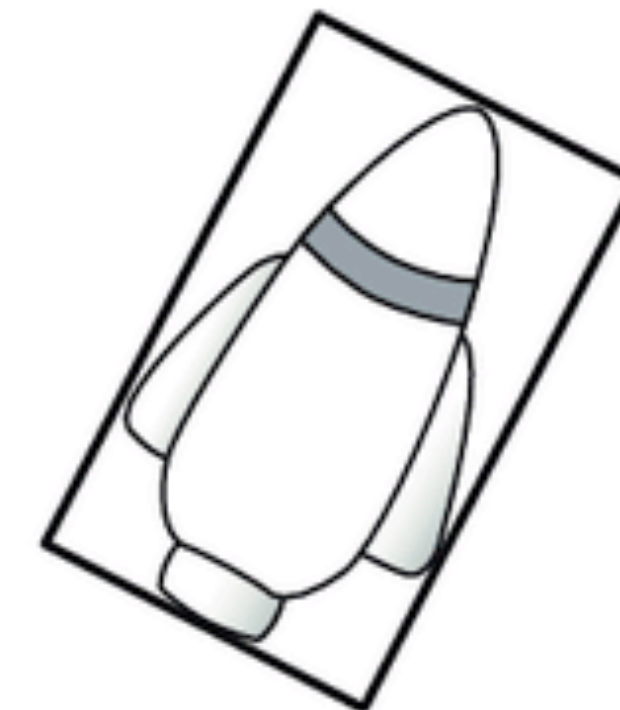
- Sphere: center and radius
- Axis-aligned bounding box: min & max along x, y, z
- Oriented bounding box: AABB + rotation



SPHERE



AABB



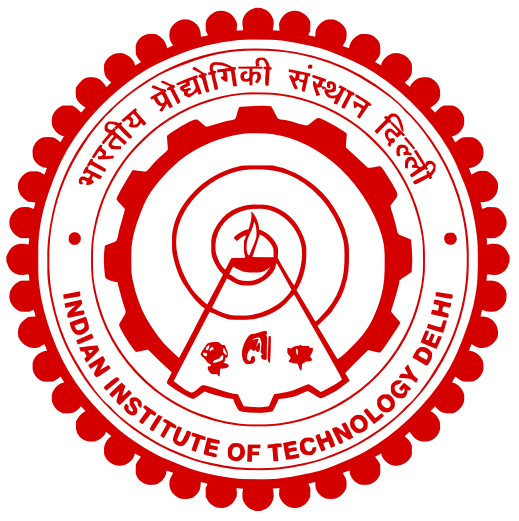
OBB



8-DOP

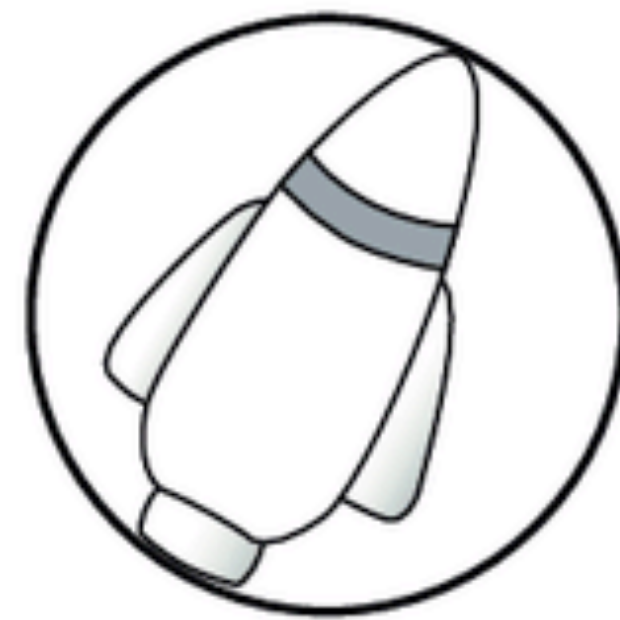


CONVEX HULL

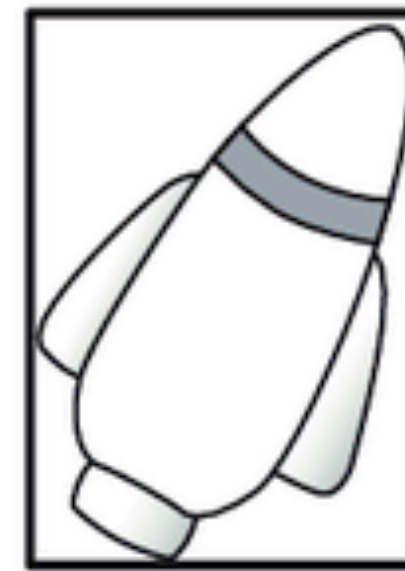


Representation of BV

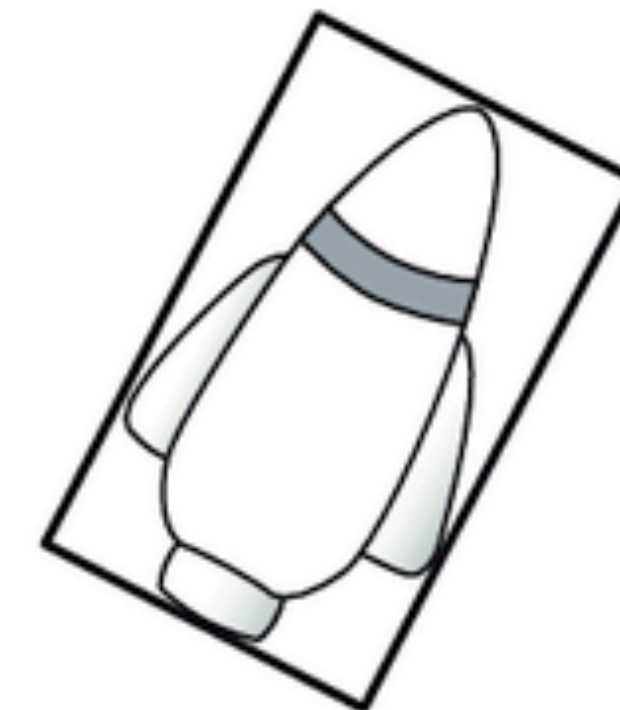
- Sphere: center and radius
- Axis-aligned bounding box: min & max along x, y, z
- Oriented bounding box: AABB + rotation
 - ➔ More storage, more compute



SPHERE



AABB



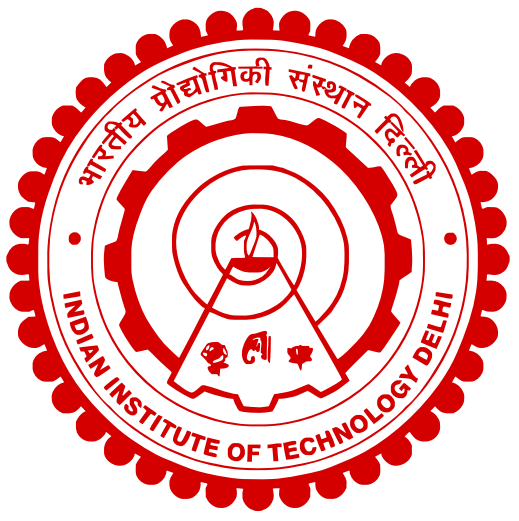
OBB



8-DOP

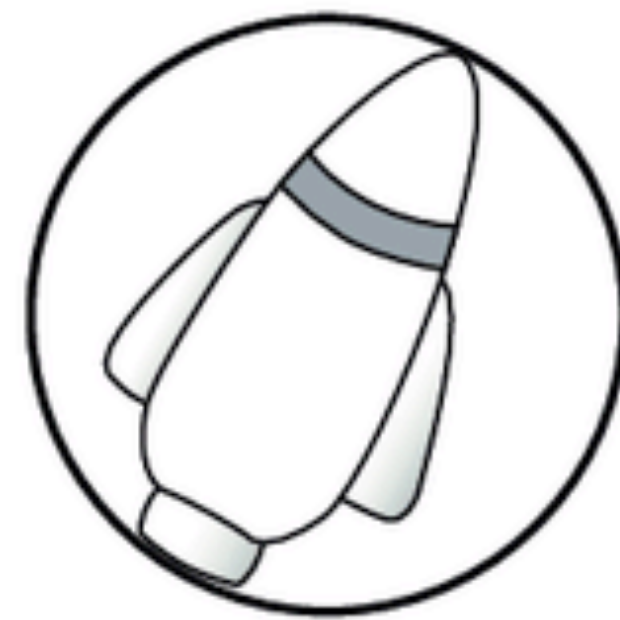


CONVEX HULL

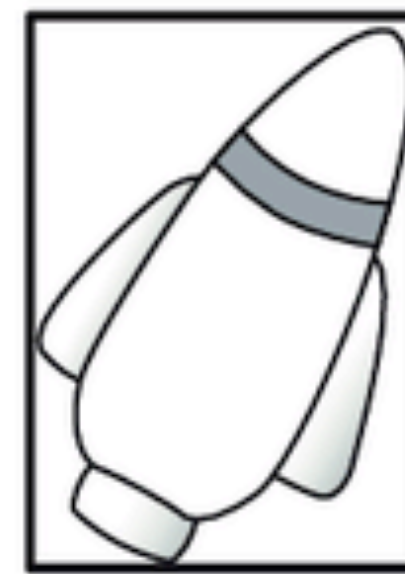


Representation of BV

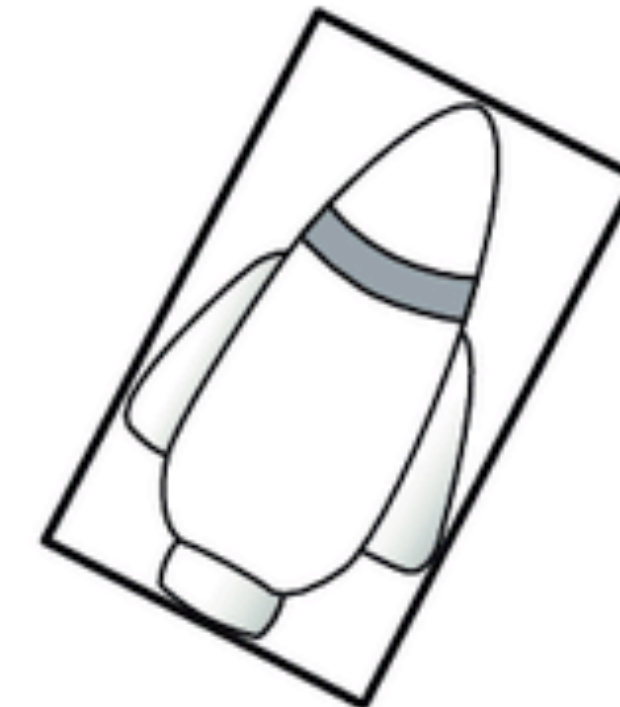
- Sphere: center and radius
- Axis-aligned bounding box: min & max along x, y, z
- Oriented bounding box: AABB + rotation
 - ➔ More storage, more compute
- k-DOP: max along k fixed directions e.g. $(\pm 1, 0, 0)$, $(\pm 1, \pm 1, 0)$, ...



SPHERE



AABB



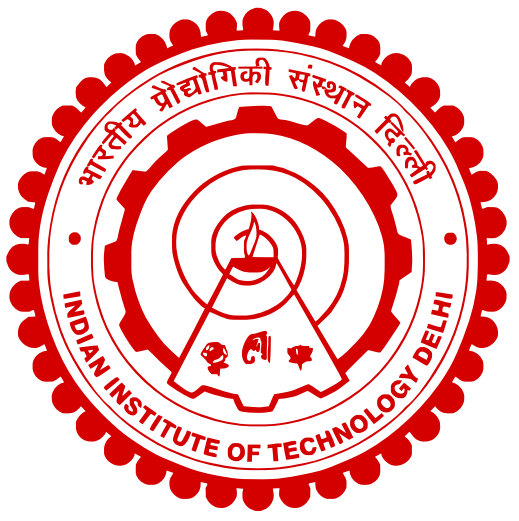
OBB



8-DOP

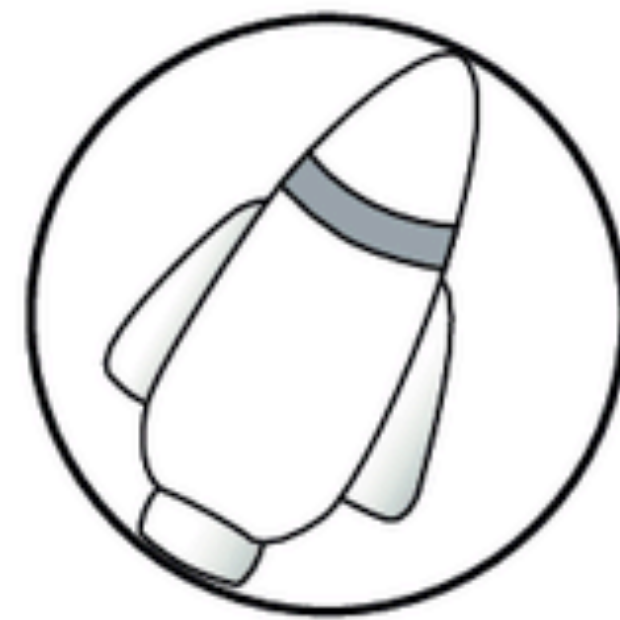


CONVEX HULL

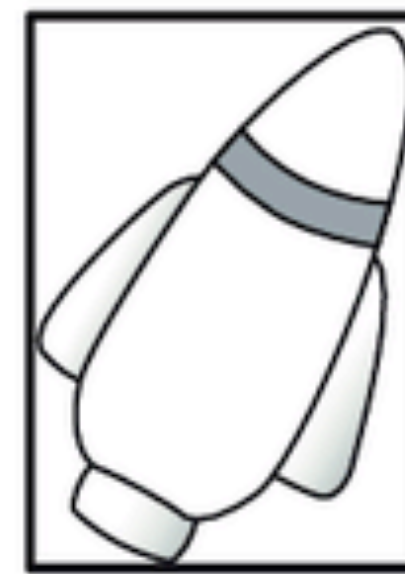


Representation of BV

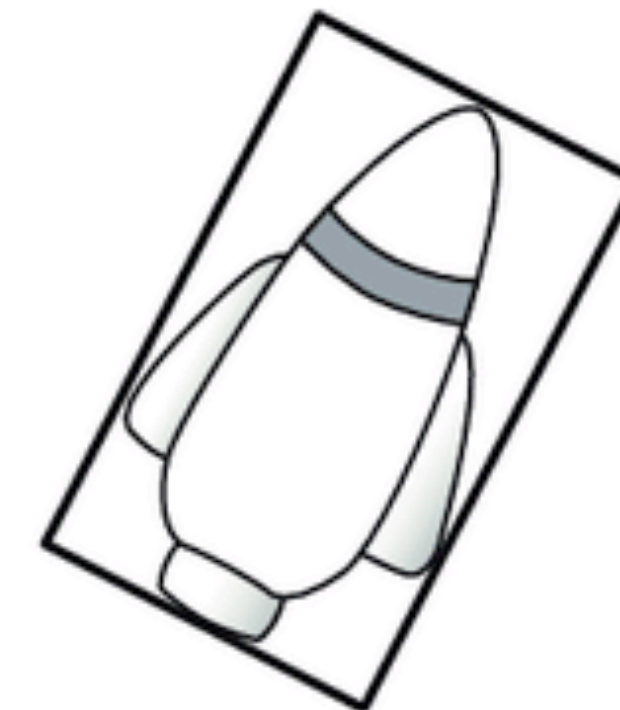
- Sphere: center and radius
- Axis-aligned bounding box: min & max along x, y, z
- Oriented bounding box: AABB + rotation
 - ➔ More storage, more compute
- k-DOP: max along k fixed directions e.g. $(\pm 1, 0, 0)$, $(\pm 1, \pm 1, 0)$, ...
- Convex hull: polygon list



SPHERE



AABB



OBB



8-DOP

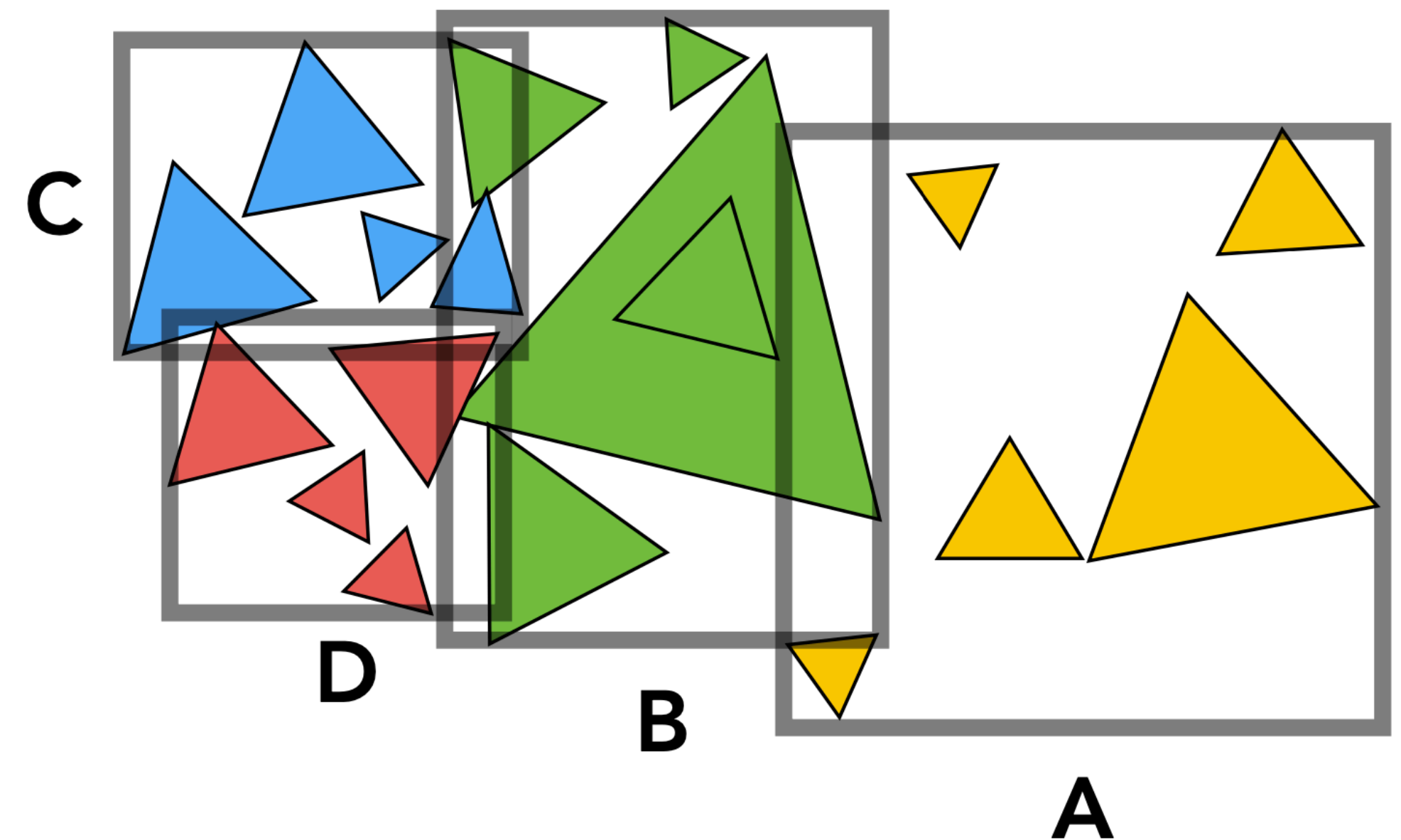
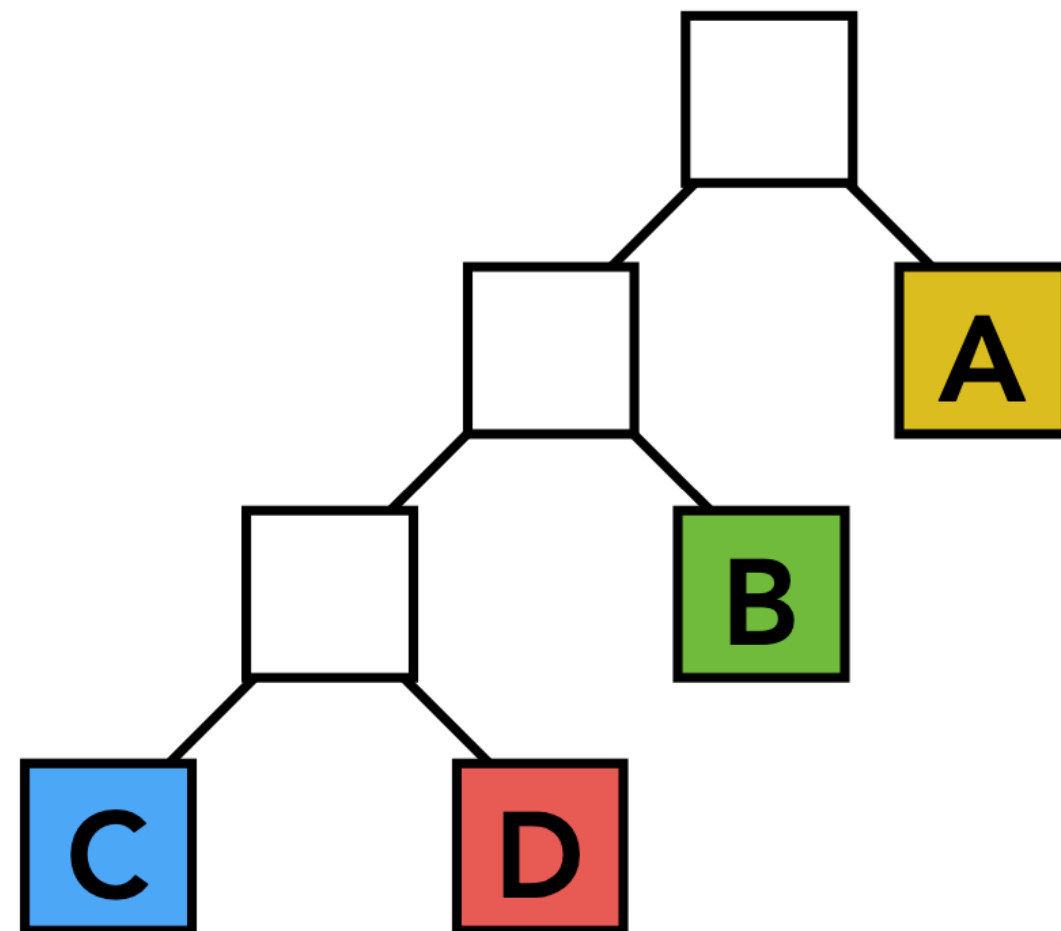


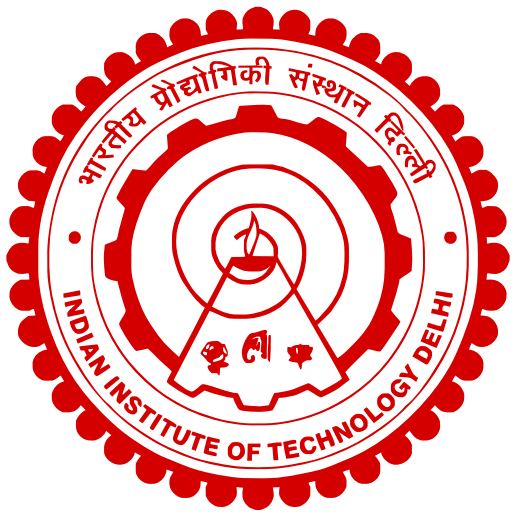
CONVEX HULL

Bounding volume hierarchy (BVH)



- Leaf nodes store a small set of objects and their bounding volume
- Internal nodes store the bounding volume of the union of their children
- Note: Bounding volumes of nodes can overlap
- Build top-down (bottom-up is simpler)





BVH traversal

- Intersect Ray (node)

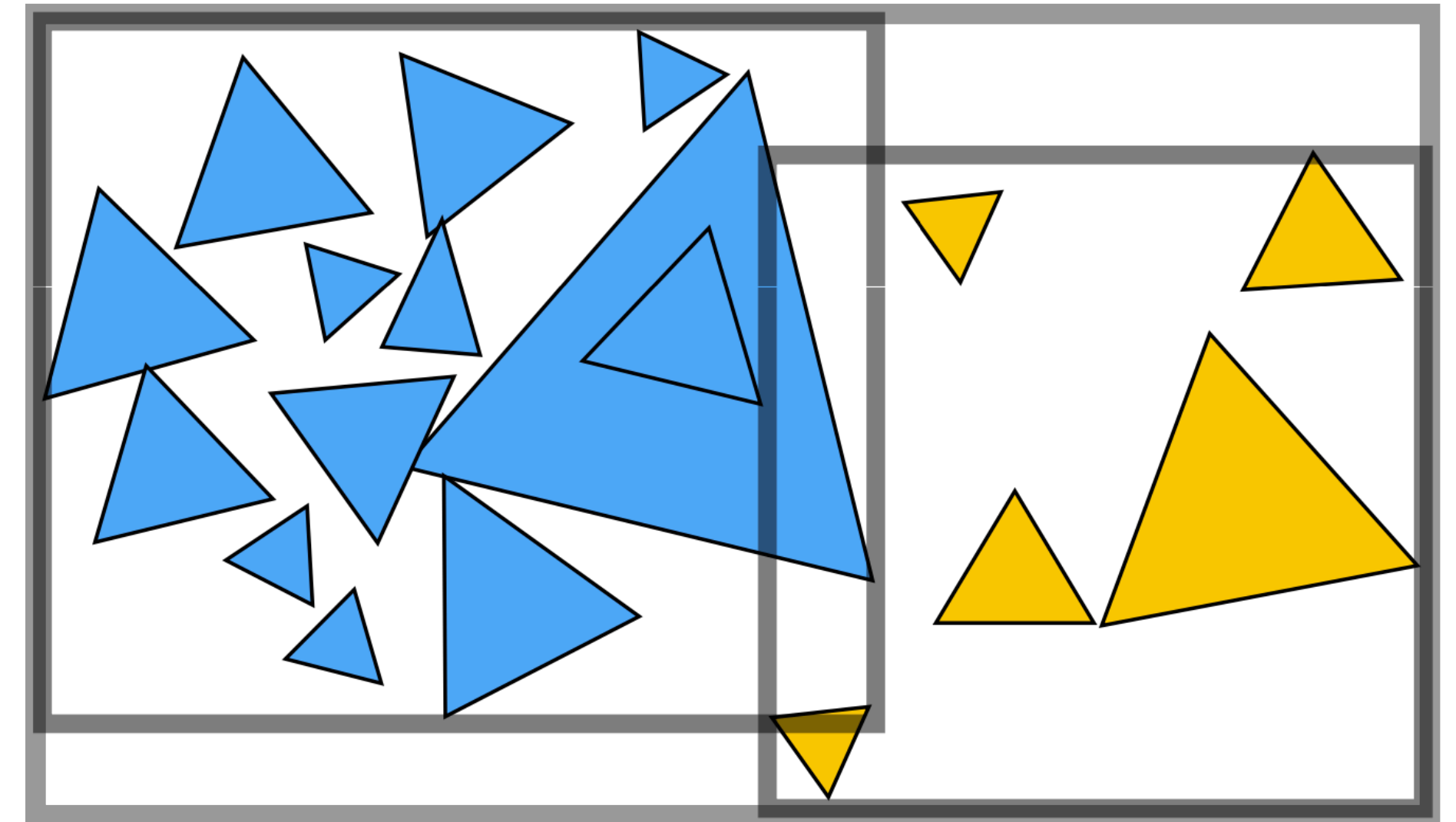
Test ray against bounding volume
if hit:

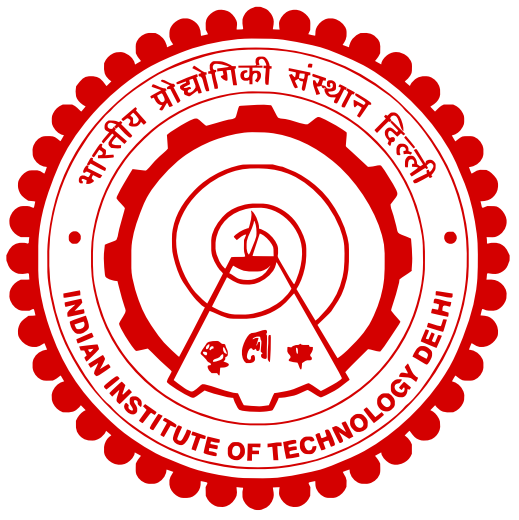
if leaf:

intersect ray with objects
return earliest hit

else:

intersect ray with child 1
intersect ray with child 2
return earlier hit





BVH traversal

- Intersect Ray (node)

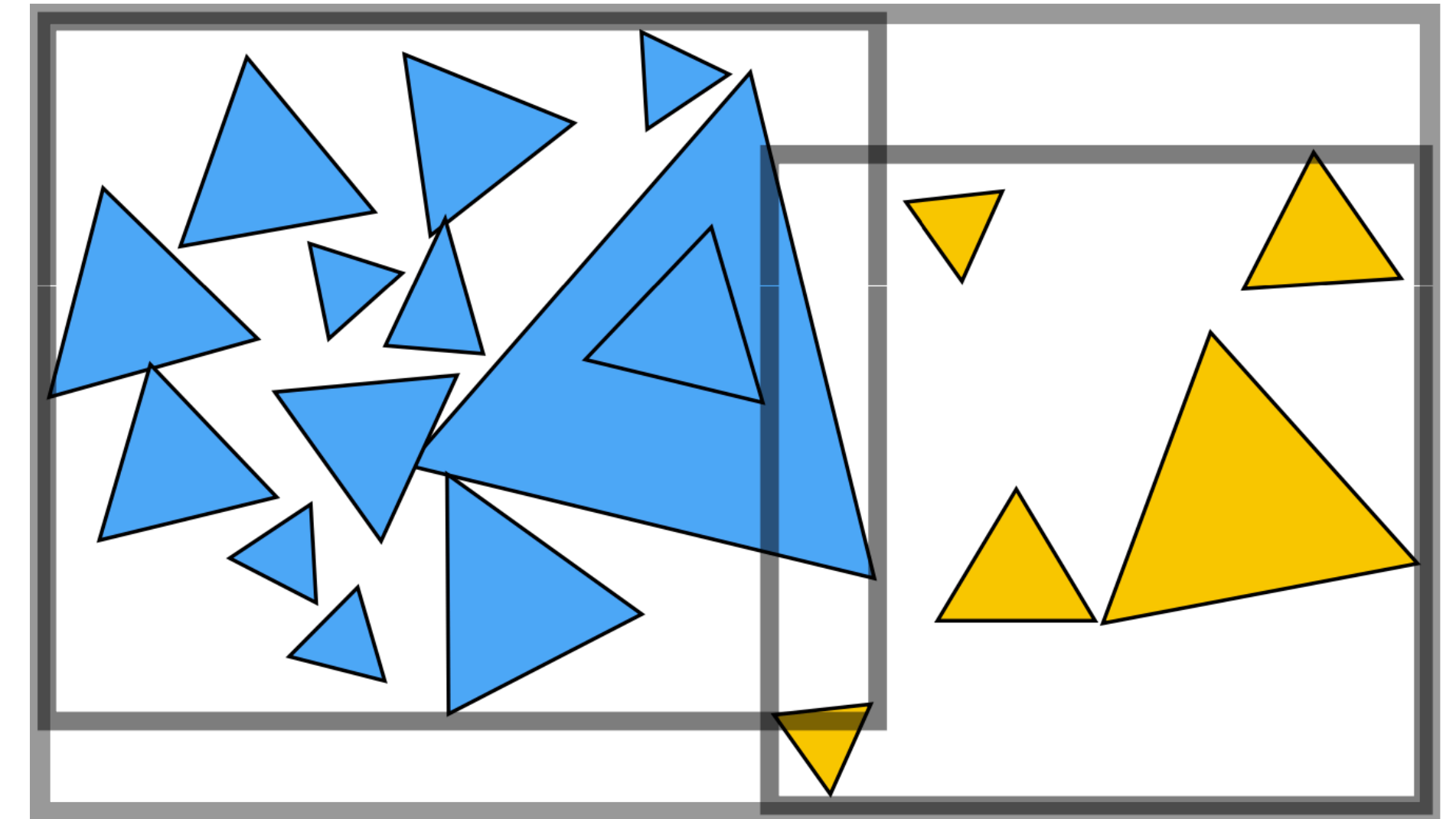
Test ray against bounding volume
if hit:

if leaf:

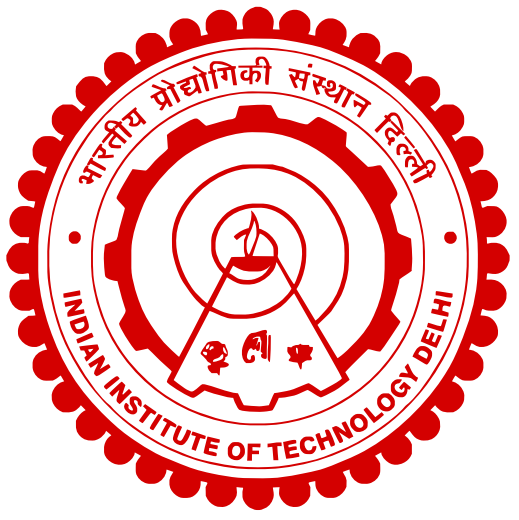
intersect ray with objects
return earliest hit

else:

intersect ray with child 1
intersect ray with child 2
return earlier hit



How many children?



Ray-BVH intersection

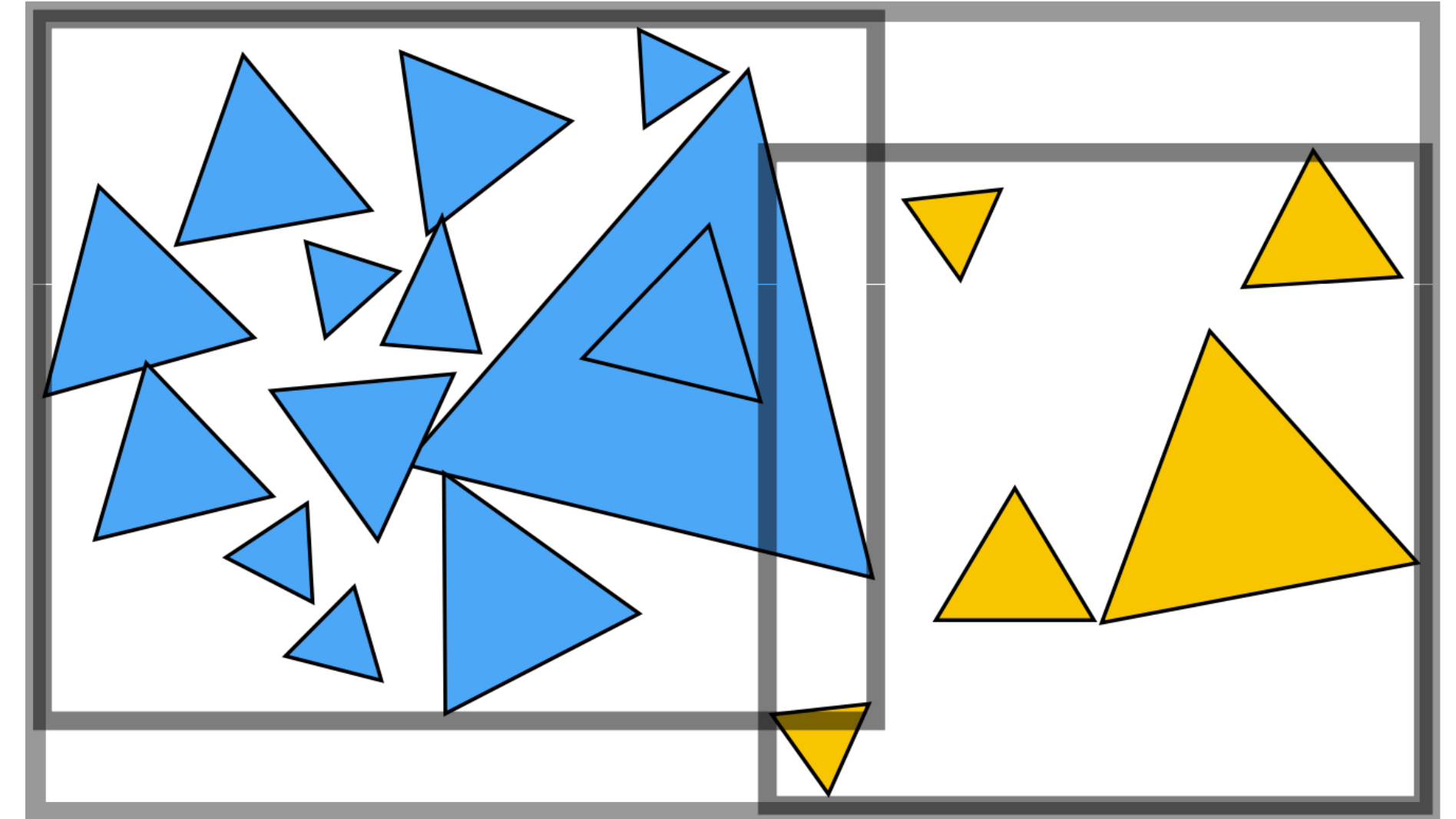
...

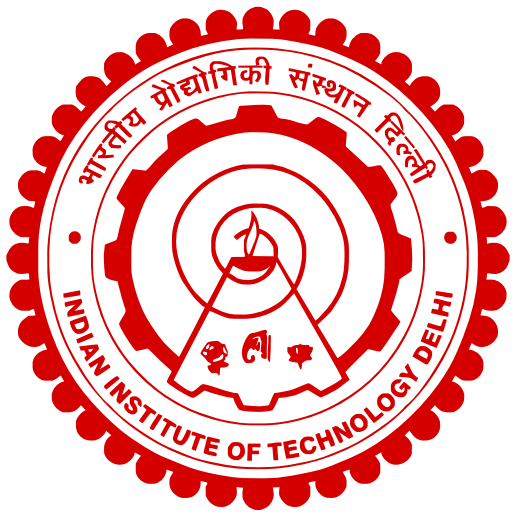
...

intersect ray with child 1
intersect ray with child 2
return earliest hit

Smarter: find which child's BV is hit earlier,
intersect ray with it first

- If no object hit: intersect with other child
- If hit: **can you skip recursing down the other child?**





Ray-BVH intersection

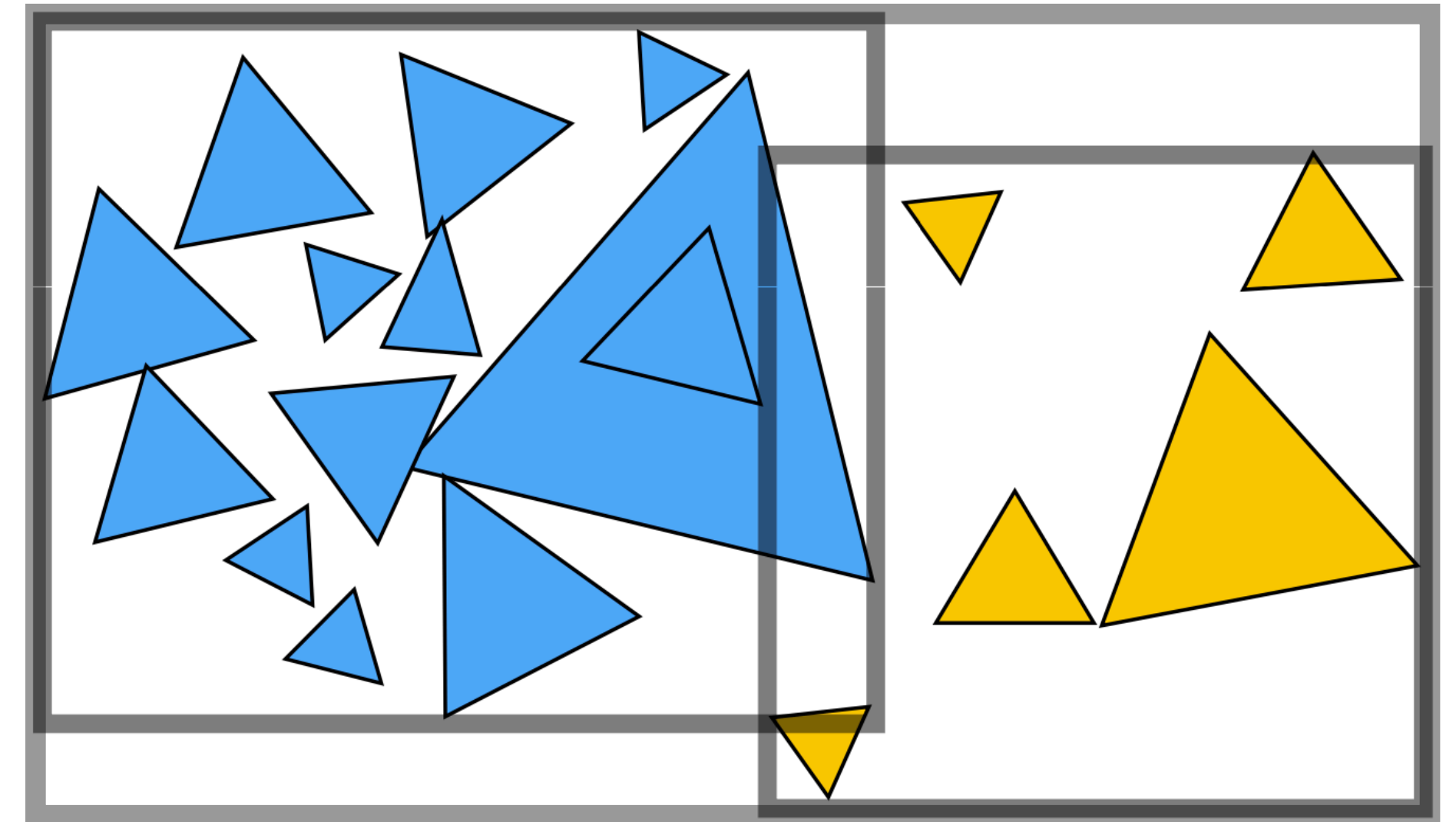
...

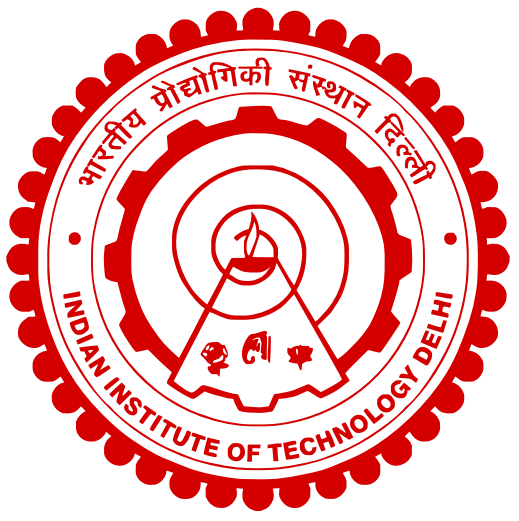
...

intersect ray with child 1
intersect ray with child 2
return earliest hit

Smarter: find which child's BV is hit earlier,
intersect ray with it first

- If no object hit: intersect with other child
- If hit: **can you skip recursing down the other child?**
 - ➔ Only if the hit occurs before reaching the other child's BV!

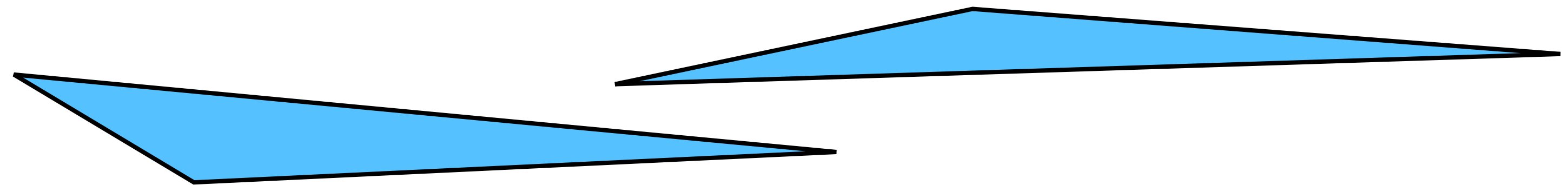
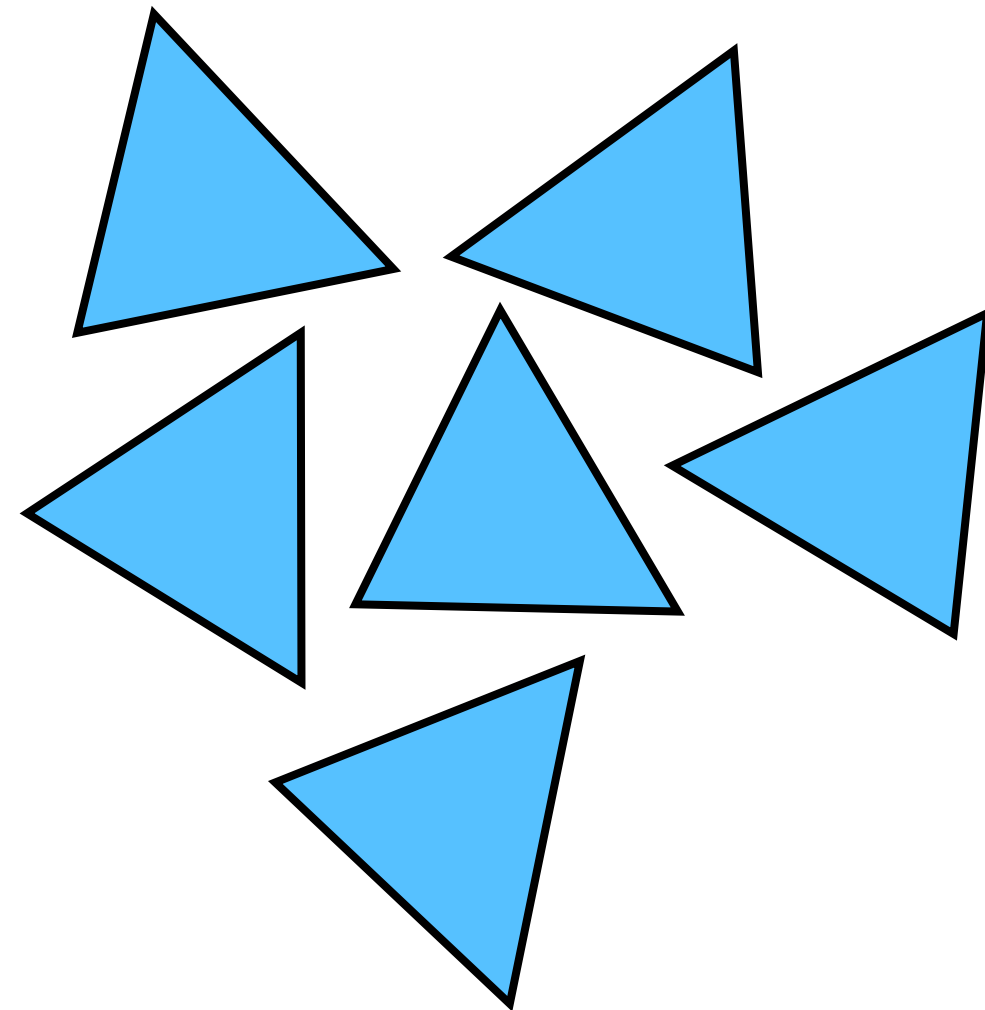




BVH construction

- Set root node = BV of all objects in scene
- Recursively create child nodes by splitting objects into two subsets

What's a good way to split? And when should we stop?

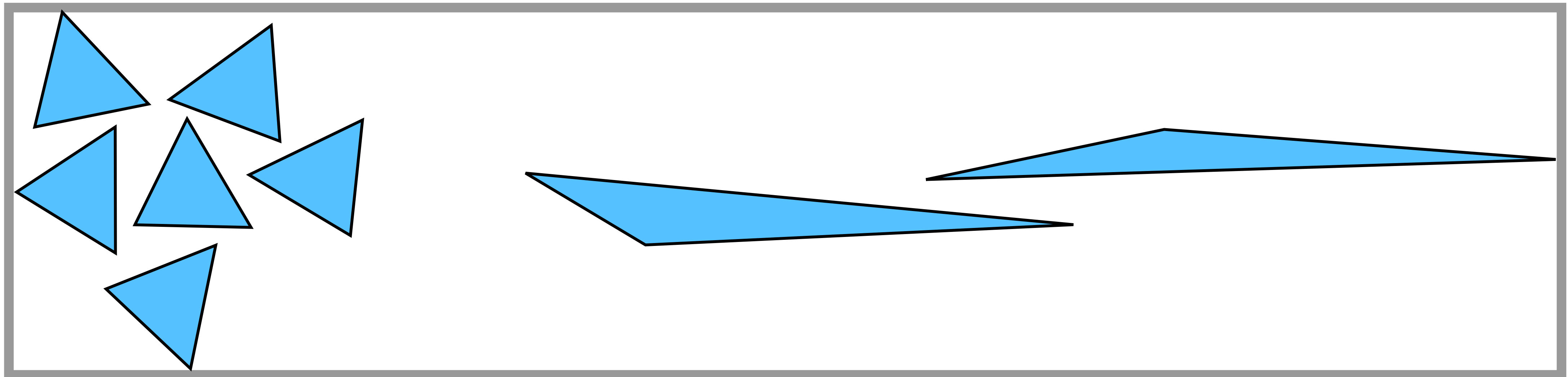




BVH construction

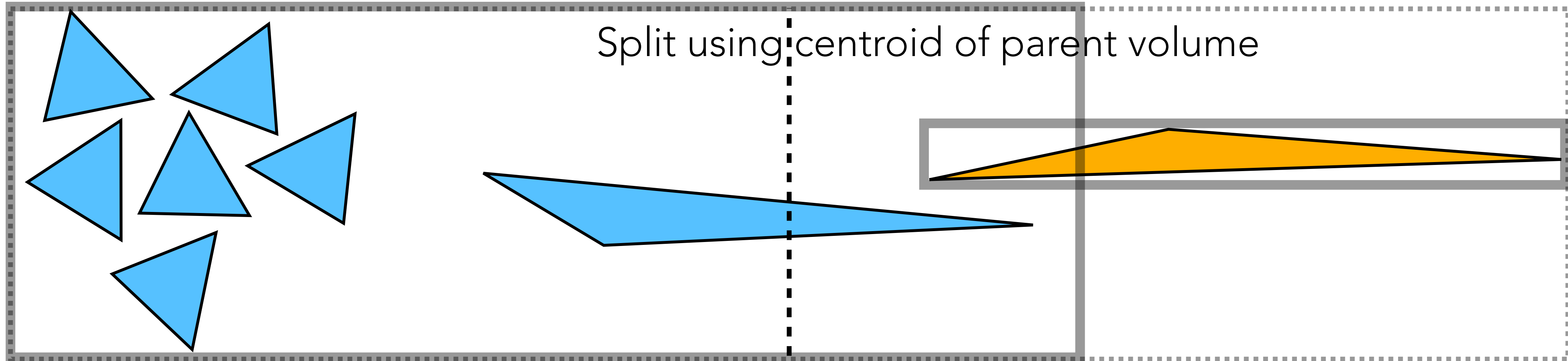
- Set root node = BV of all objects in scene
- Recursively create child nodes by splitting objects into two subsets

What's a good way to split? And when should we stop?



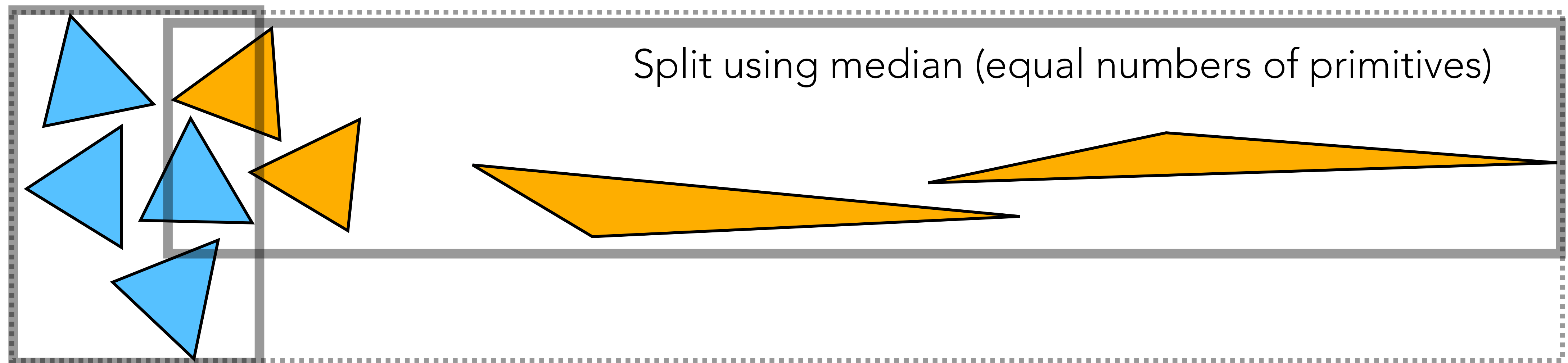
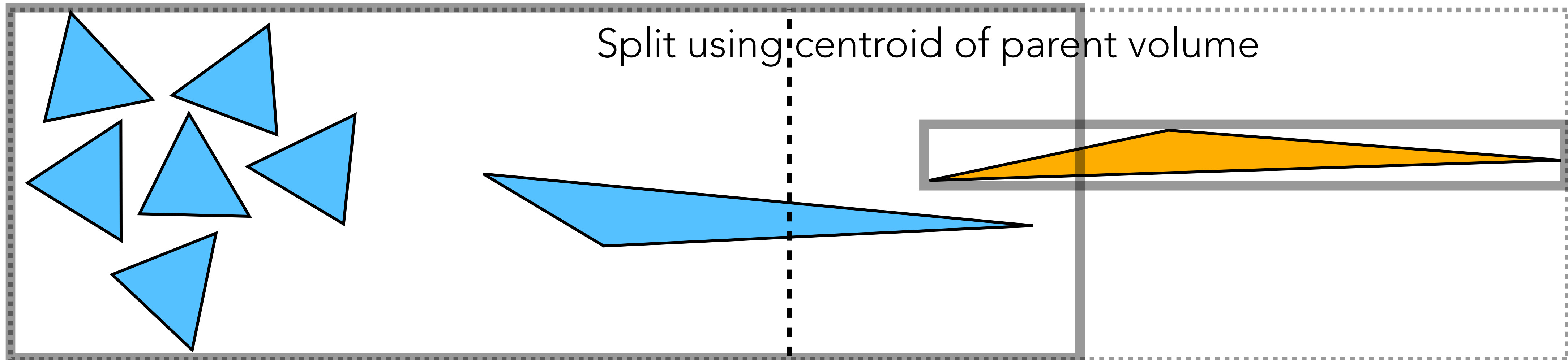
BVH construction

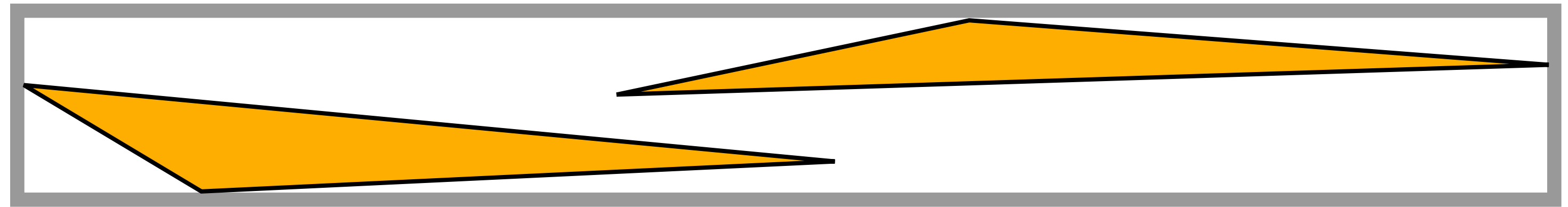
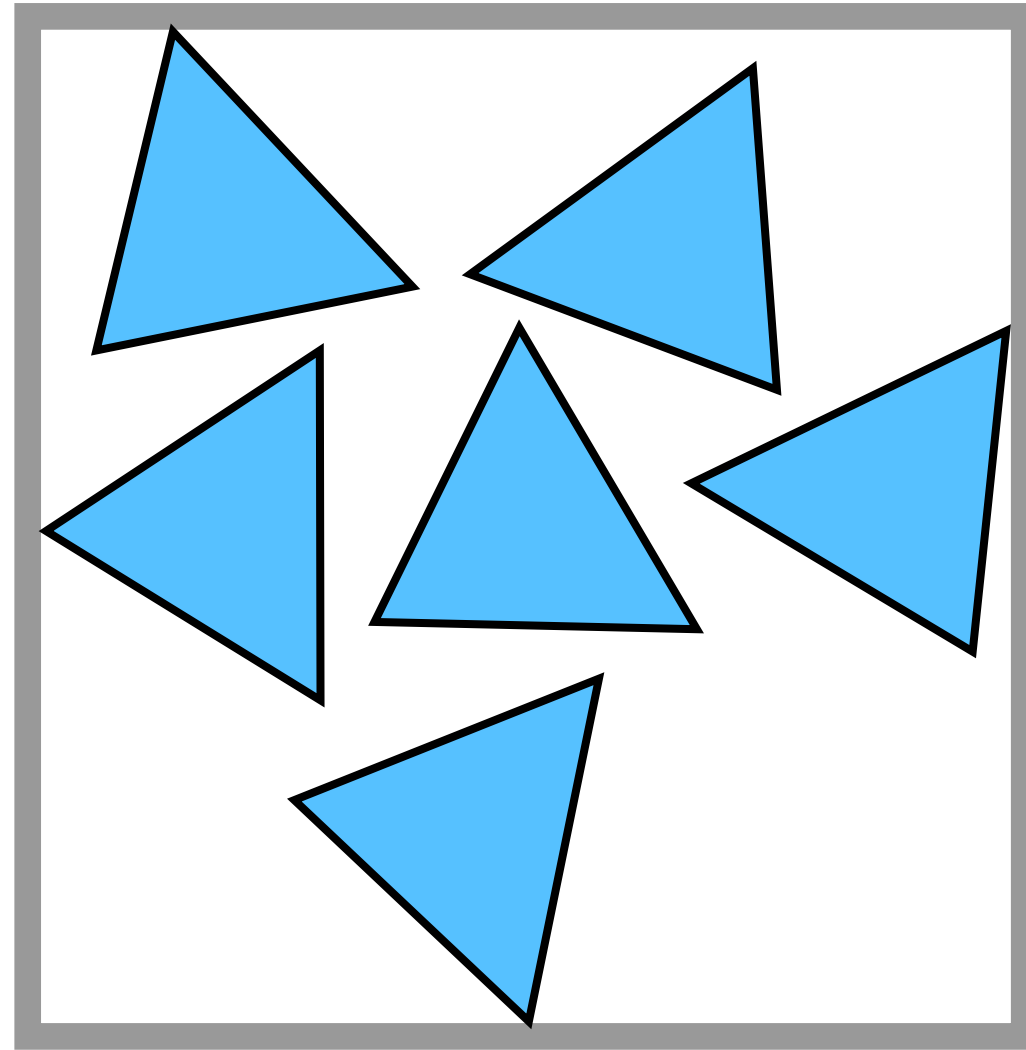
- How to partition primitives into groups



BVH construction

- How to partition primitives into groups





- Intuitively, a good partition has
 - ➔ Minimal overlap between children
 - ➔ Minimal empty space within bounding volumes
- How to formalize this?

Splitting Bounding Volume

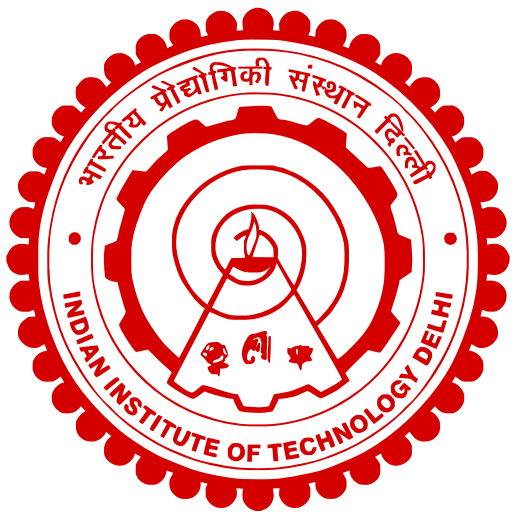


- Minimize the **cost of intersecting a ray** with the BVH

Splitting Bounding Volume

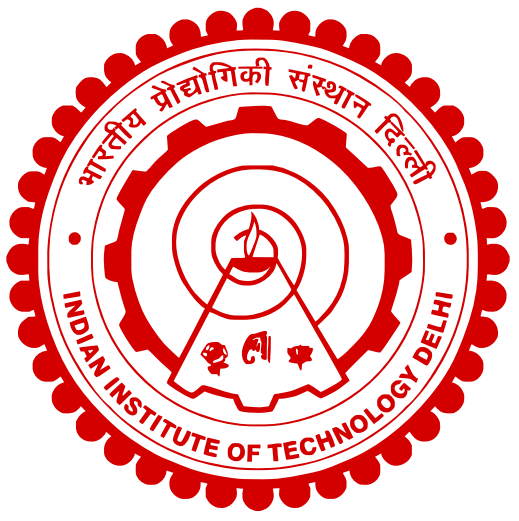


- Minimize the **cost of intersecting a ray** with the BVH
- Cost of leaf = **$N C_{\text{isect}}$**
 - N = number of primitives
 - C_{isect} = cost of intersecting one primitive



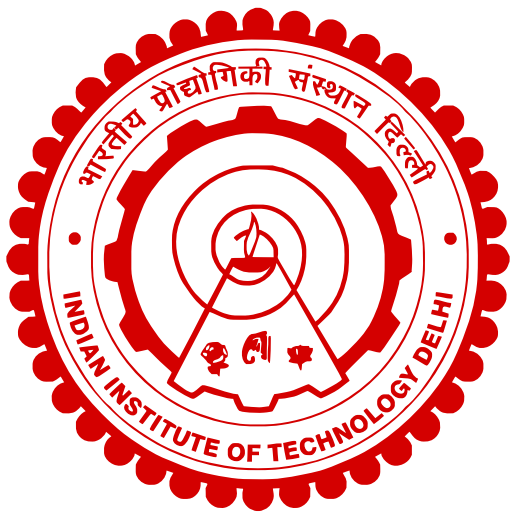
Splitting Bounding Volume

- Minimize the **cost of intersecting a ray** with the BVH
- Cost of leaf = **$N C_{\text{isect}}$**
 - N = number of primitives
 - C_{isect} = cost of intersecting one primitive
- Cost of internal node = **$C_{\text{trav}} + p_L C_L + p_R C_R$**
 - C_{trav} = cost of traversing an internal node (e.g. ray-BV intersection)
 - p_L, p_R = probabilities of hitting child BVs
 - C_L, C_R = costs of intersecting children



Splitting Bounding Volume

- Minimize the **cost of intersecting a ray** with the BVH
- Cost of leaf = **$N C_{\text{isect}}$**
 N = number of primitives
 C_{isect} = cost of intersecting one primitive
- Cost of internal node = **$C_{\text{trav}} + p_L C_L + p_R C_R$**
 C_{trav} = cost of traversing an internal node (e.g. ray-BV intersection)
 p_L, p_R = probabilities of hitting child BVs
 C_L, C_R = costs of intersecting children
- For C_L, C_R , take children as leaves: **$C_L = N_L C_{\text{isect}}, C_R = N_R C_{\text{isect}}$**



Splitting Bounding Volume

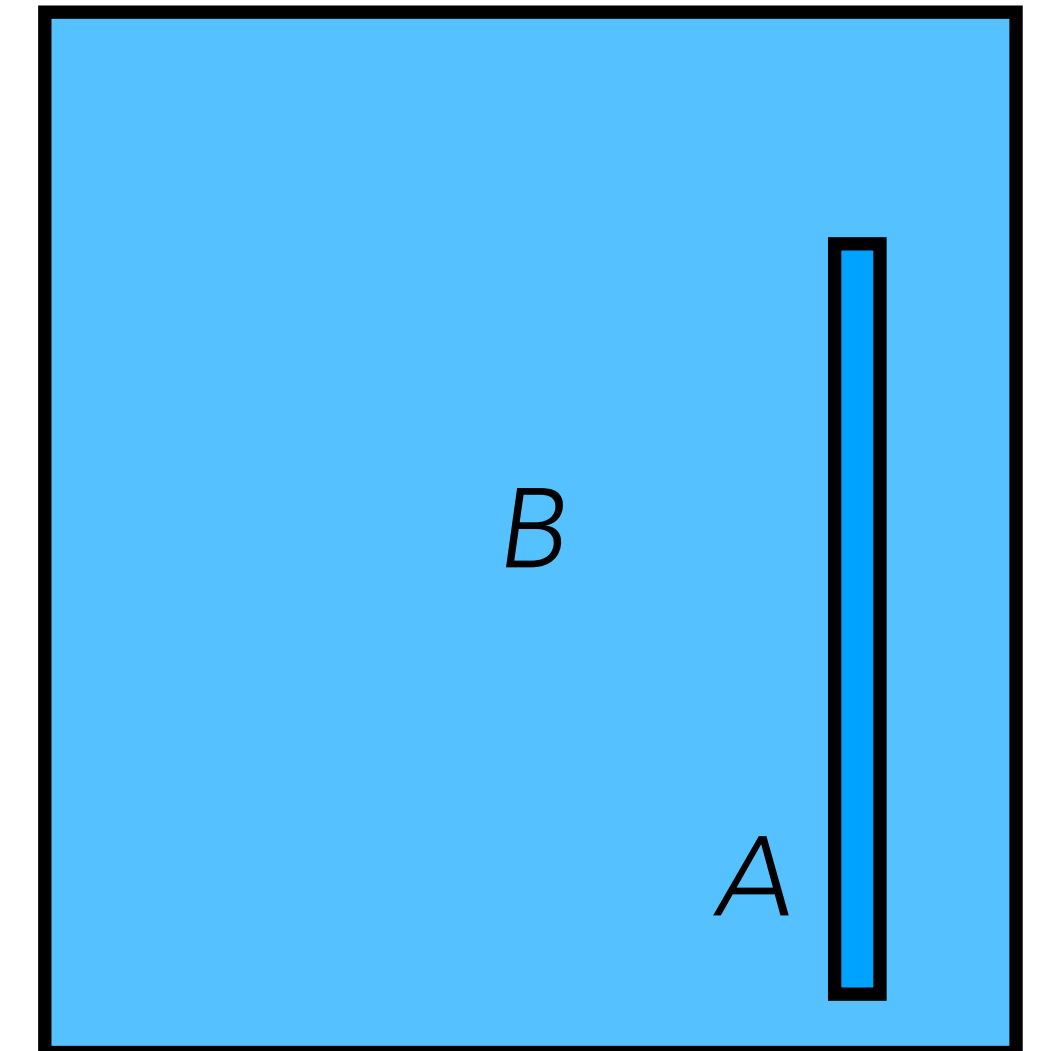
- Minimize the **cost of intersecting a ray** with the BVH
- Cost of leaf = **$N C_{\text{isect}}$**
 N = number of primitives
 C_{isect} = cost of intersecting one primitive
- Cost of internal node = **$C_{\text{trav}} + p_L C_L + p_R C_R$**
 C_{trav} = cost of traversing an internal node (e.g. ray-BV intersection)
 p_L, p_R = probabilities of hitting child BVs
 C_L, C_R = costs of intersecting children
- For C_L, C_R , take children as leaves: **$C_L = N_L C_{\text{isect}}, C_R = N_R C_{\text{isect}}$**
- How to estimate p_L, p_R ?

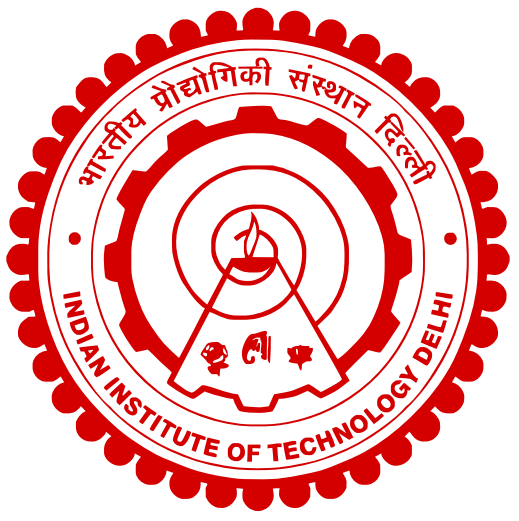


Surface Area Heuristic

Fact: For two **convex** shapes $A \subseteq B$, the probability that a **random** ray which hits B also hits A is equal to the ratio of their surface areas.

$$p(\text{hit } A \mid \text{hit } B) = \frac{S_A}{S_B}$$



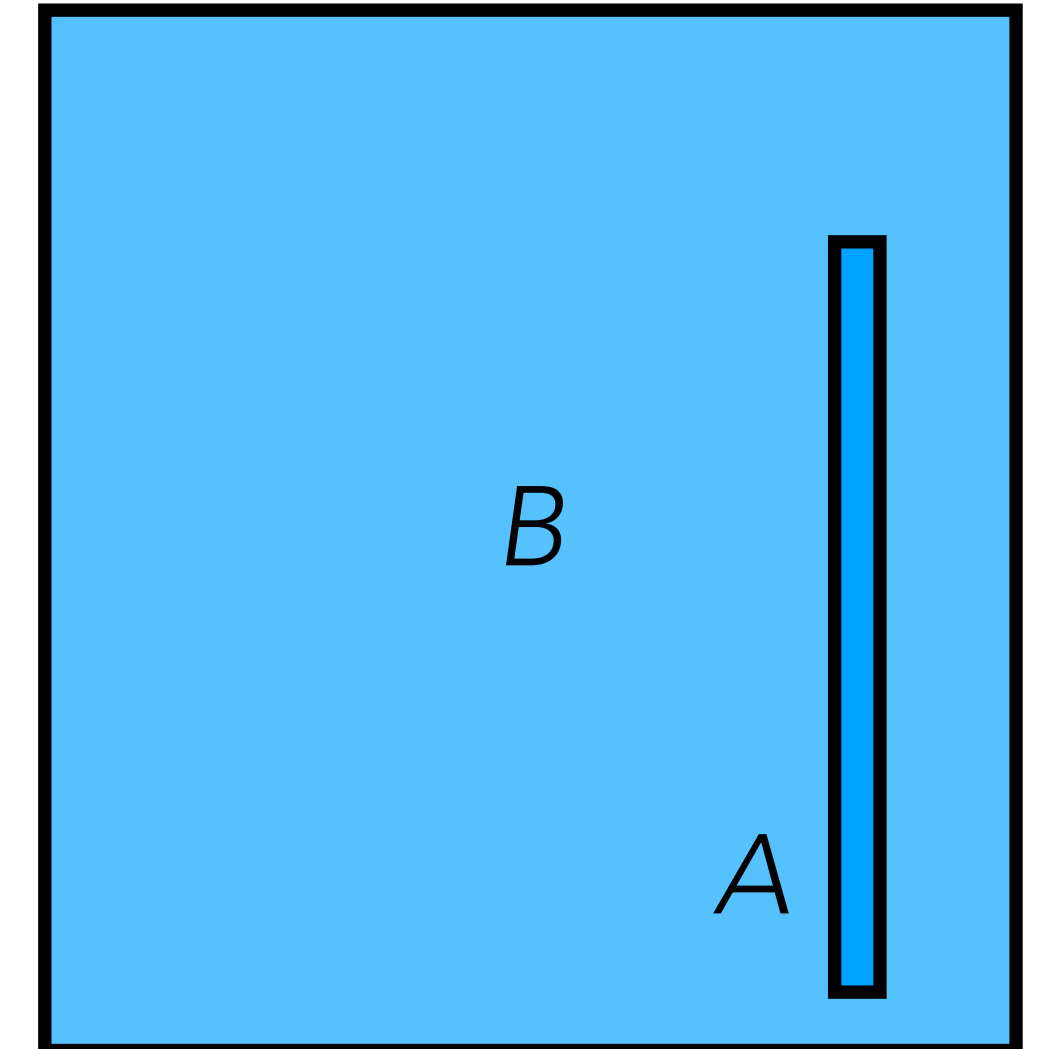


Surface Area Heuristic

Fact: For two **convex** shapes $A \subseteq B$, the probability that a **random** ray which hits B also hits A is equal to the ratio of their surface areas.

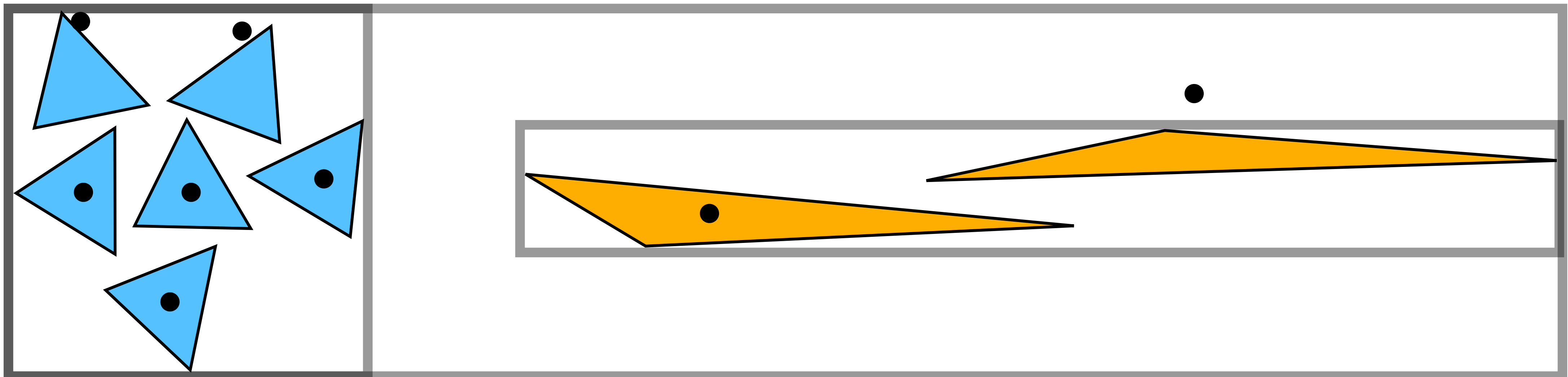
$$p(\text{hit } A \mid \text{hit } B) = \frac{S_A}{S_B}$$

So, cost of internal node $\approx C_{\text{trav}} + \frac{S_L}{S_P} N_L C_{\text{isect}} + \frac{S_R}{S_P} N_R C_{\text{isect}}$



Split Node

- Choose a splitting axis (e.g., longest axis or try all three)
- Sort primitives by their centroid (along the axis)
- For candidate choices of partition, evaluate SAH cost
- Choose candidate with the lowest cost

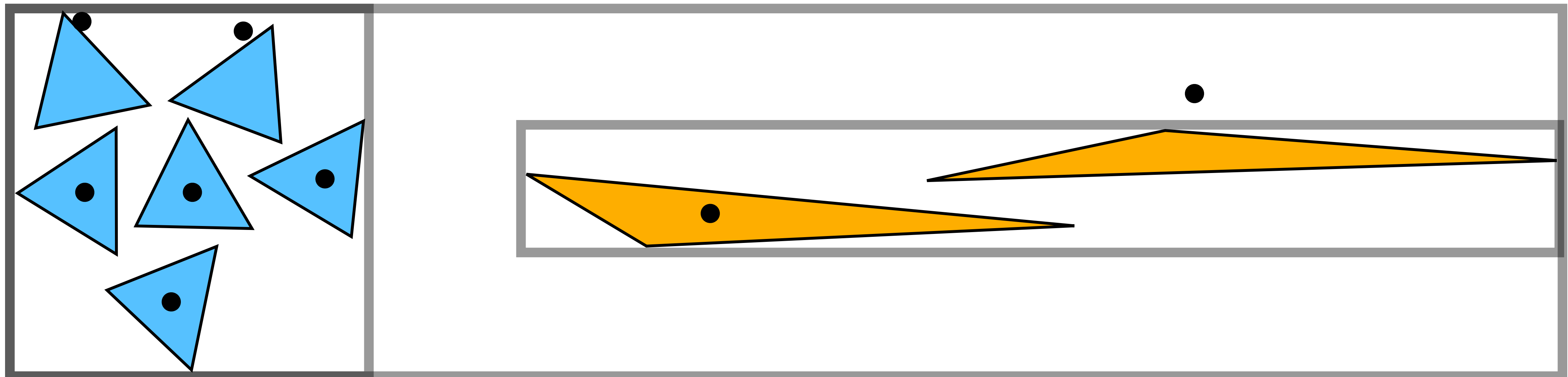




Split Node

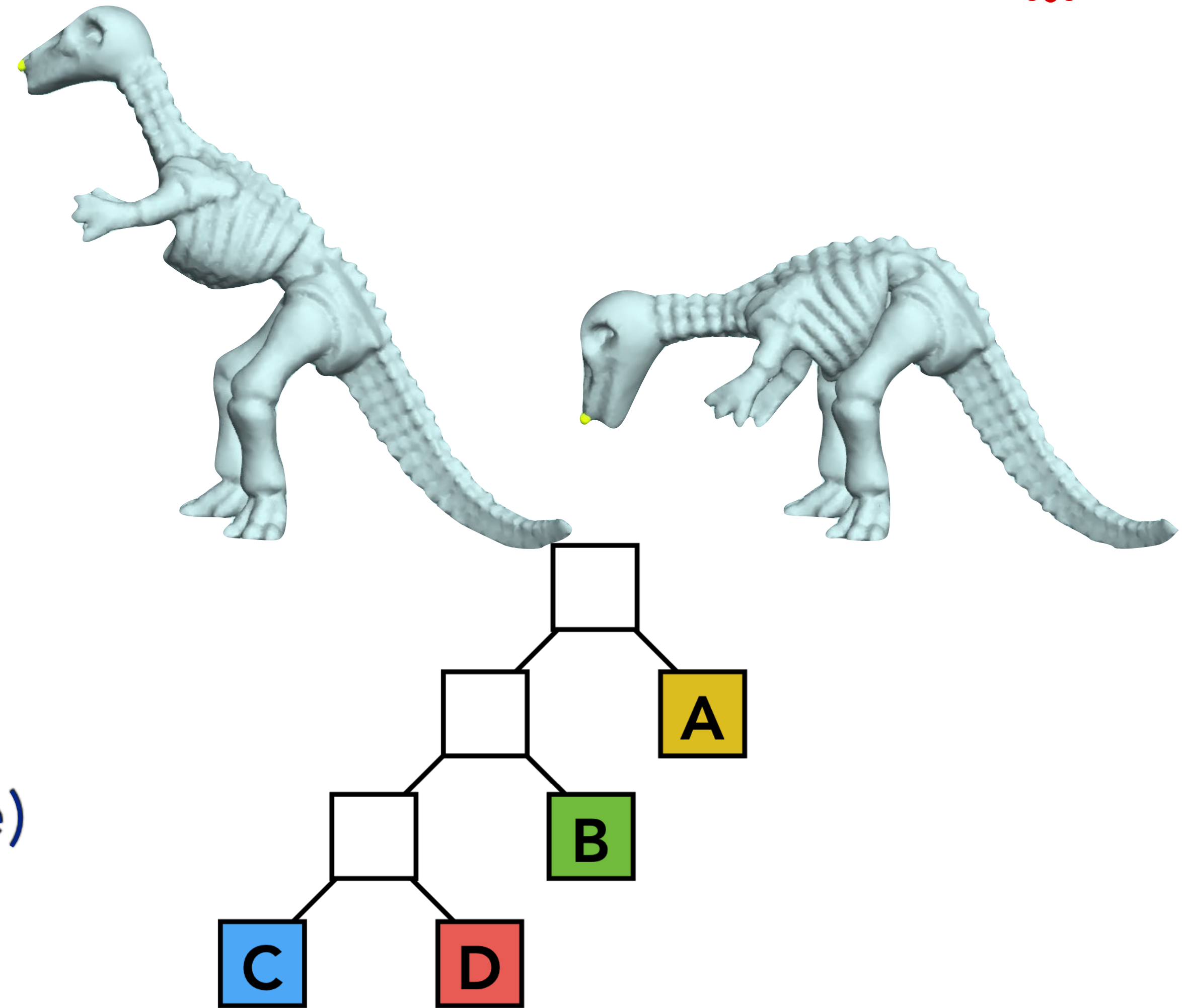
(Faster approximation: just collect into B buckets)

- Choose a splitting axis (e.g., longest axis or try all three)
- Sort primitives by their centroid (along the axis)
- For candidate choices of partition, evaluate SAH cost
- Choose candidate with the lowest cost



BVH Refitting

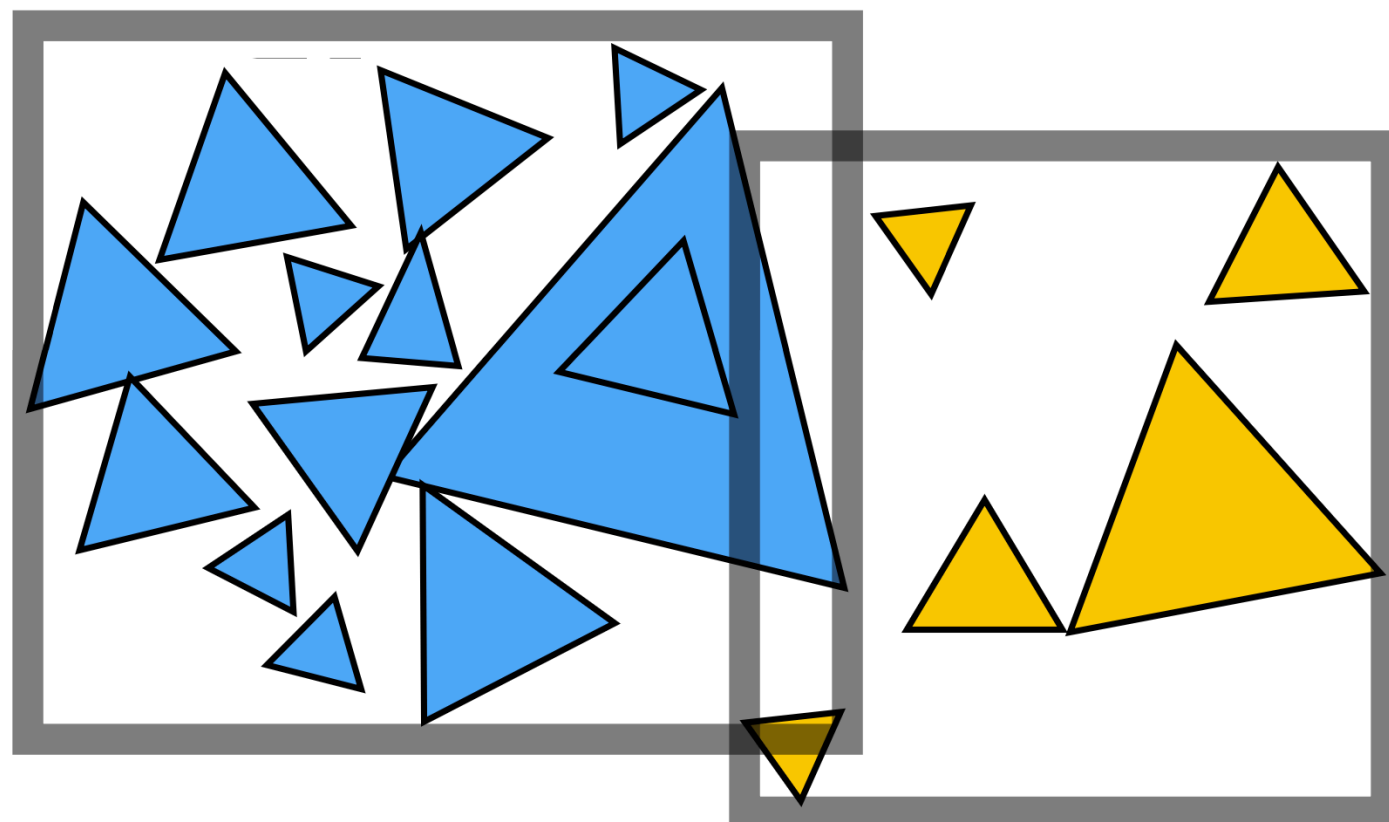
- Given a pre-built BVH and Object transformation
 - ➔ Update BVH
- What if only a few primitive moved?
- What if all primitives have moved?
- Keep structure, update boxes
- Local updates to structures (split, merge)
- Re-build (sub-trees or entire tree)



Object vs. Space Partitioning



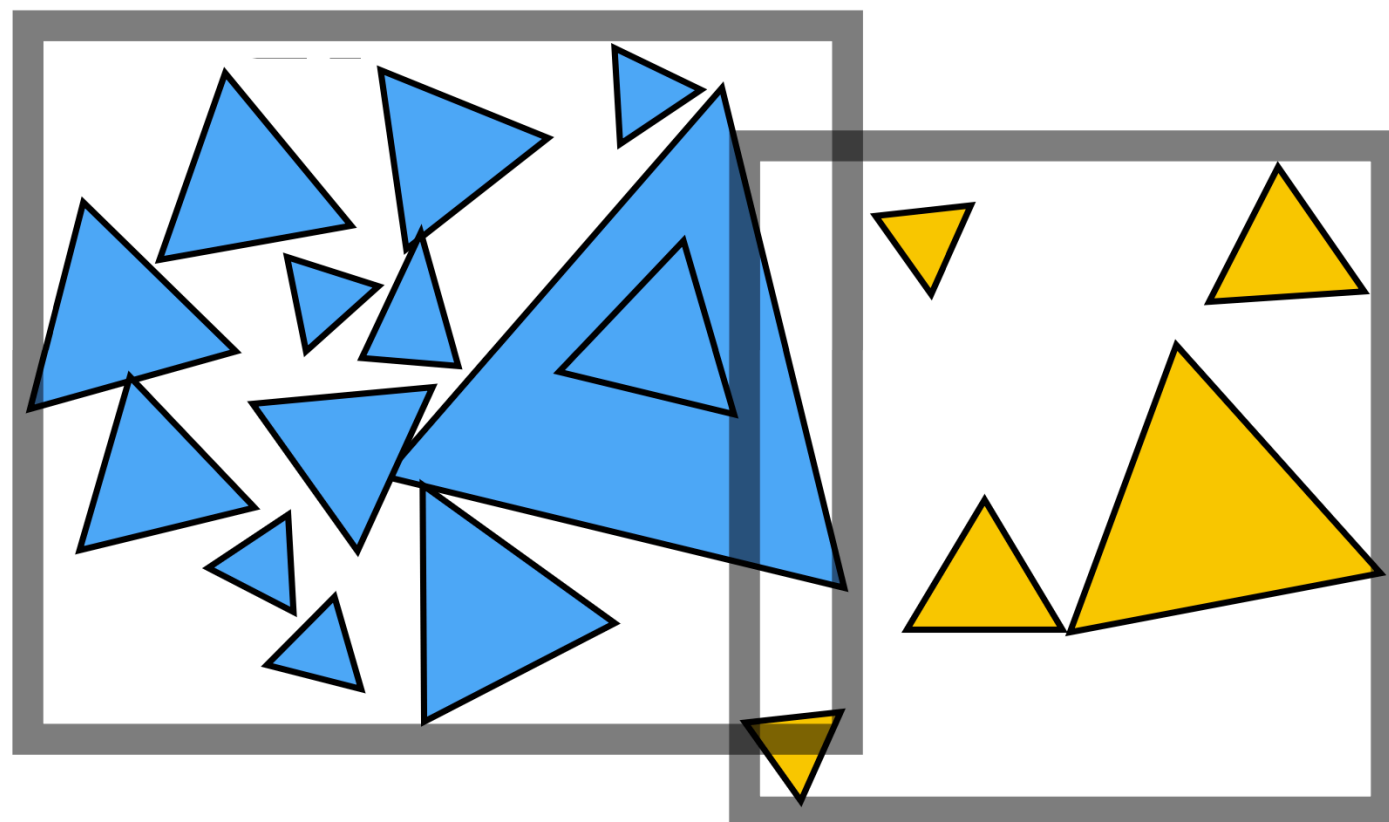
Object partitioning: split primitives into disjoint subsets



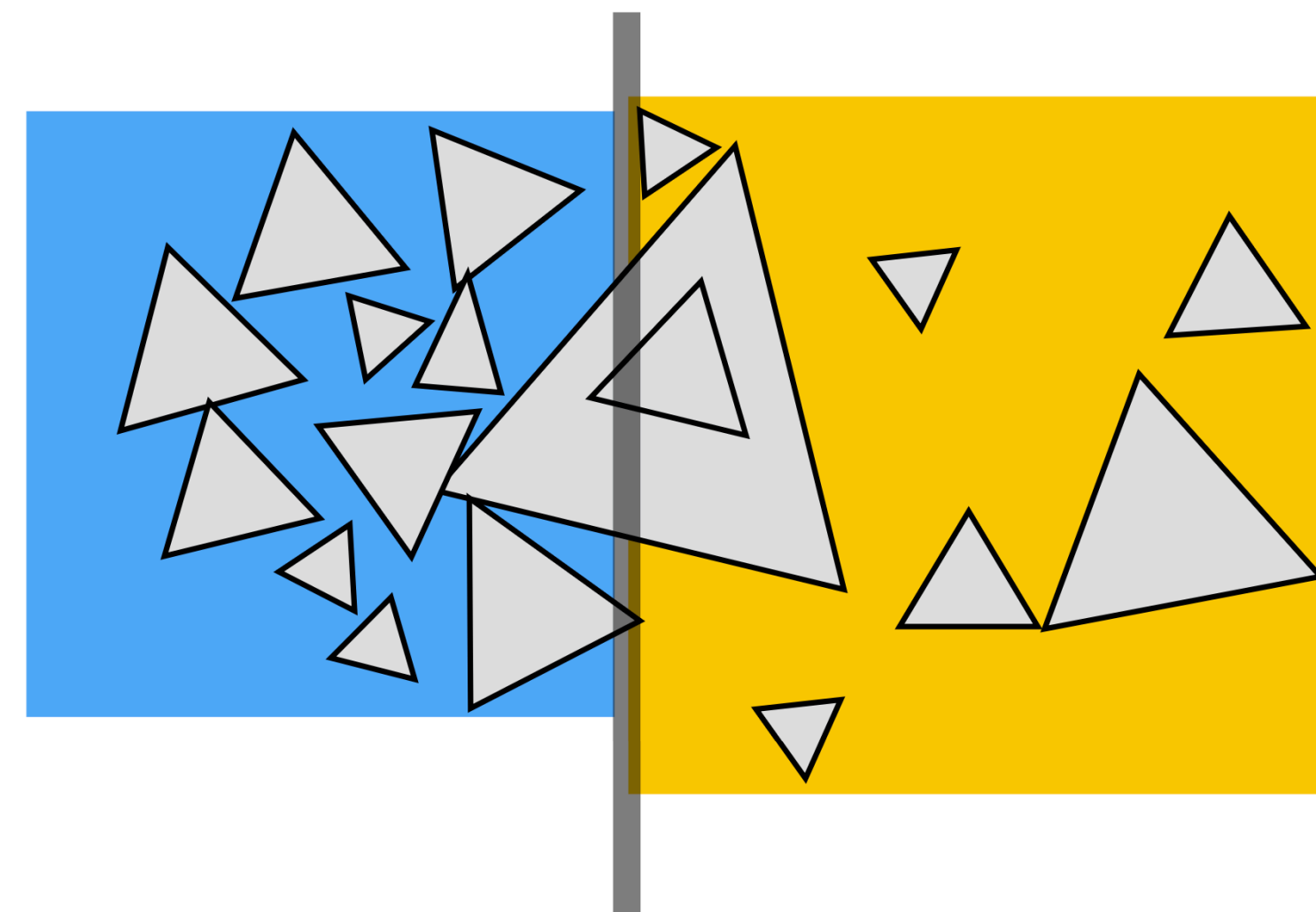
Object vs. Space Partitioning



Object partitioning: split primitives into disjoint subsets



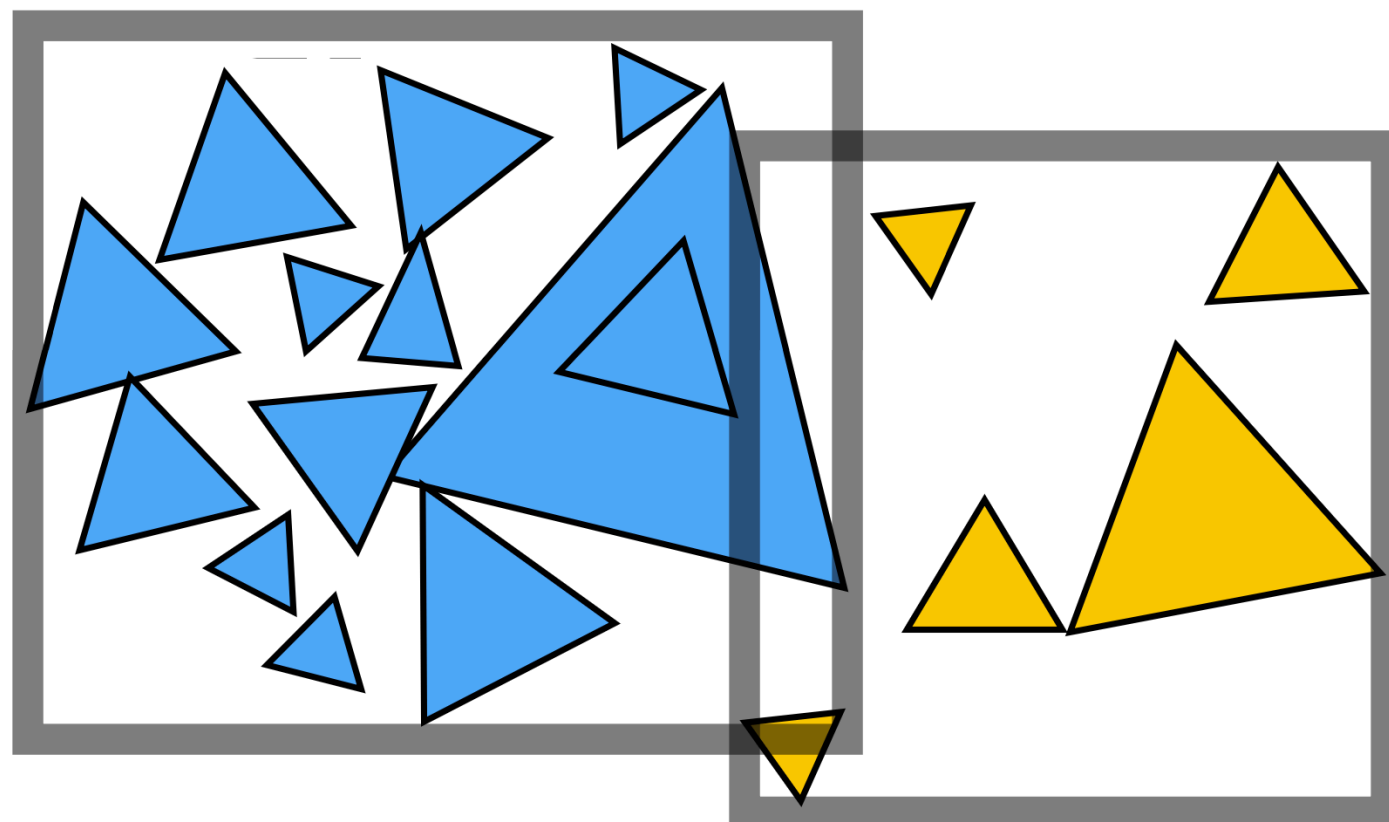
Space partitioning: split space into regions (k-d trees, grids, octrees, ...)



Object vs. Space Partitioning

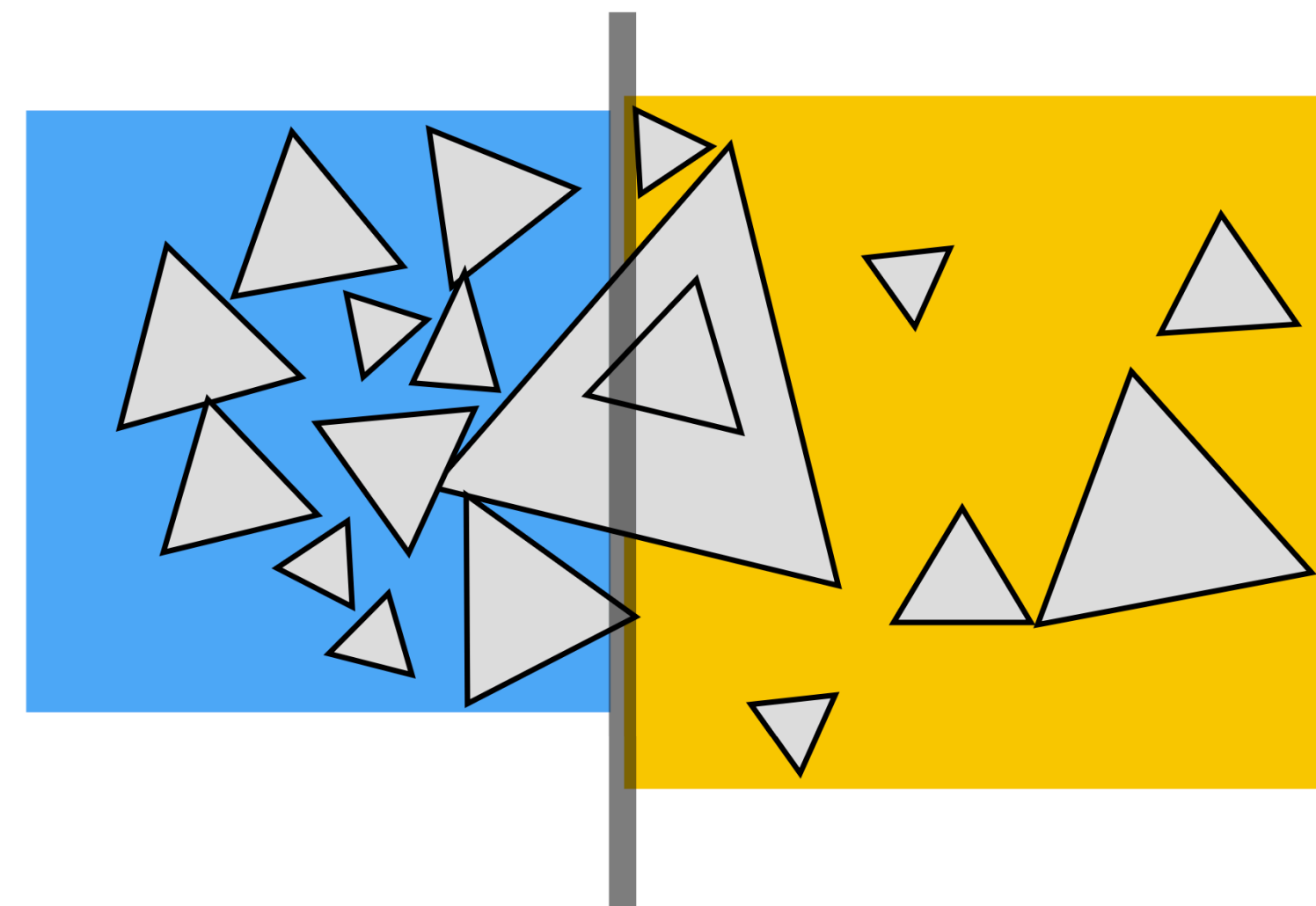
Object partitioning: split primitives into disjoint subsets

- Bounding volumes may overlap



Space partitioning: split space into regions (k-d trees, grids, octrees, ...)

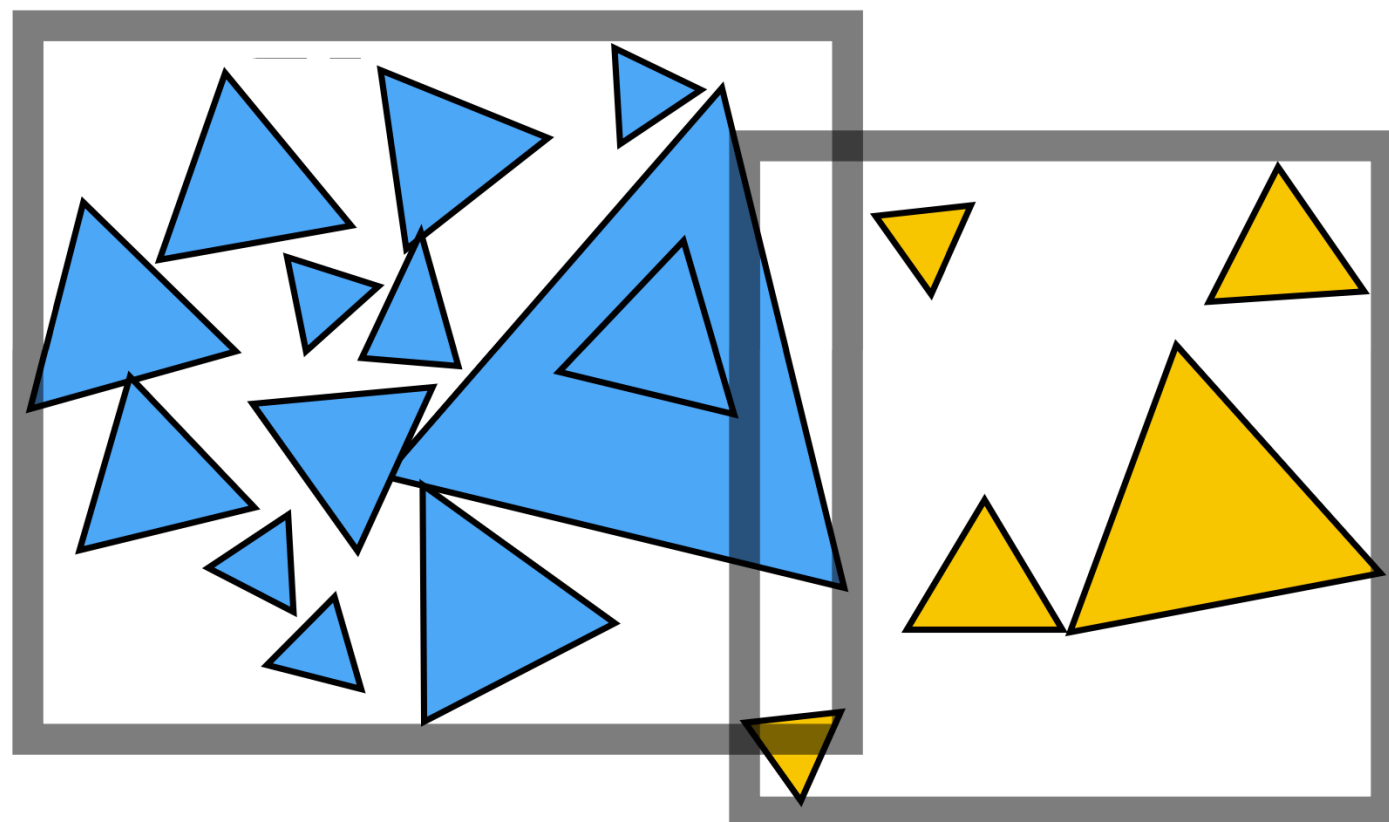
- Regions are non-overlapping



Object vs. Space Partitioning

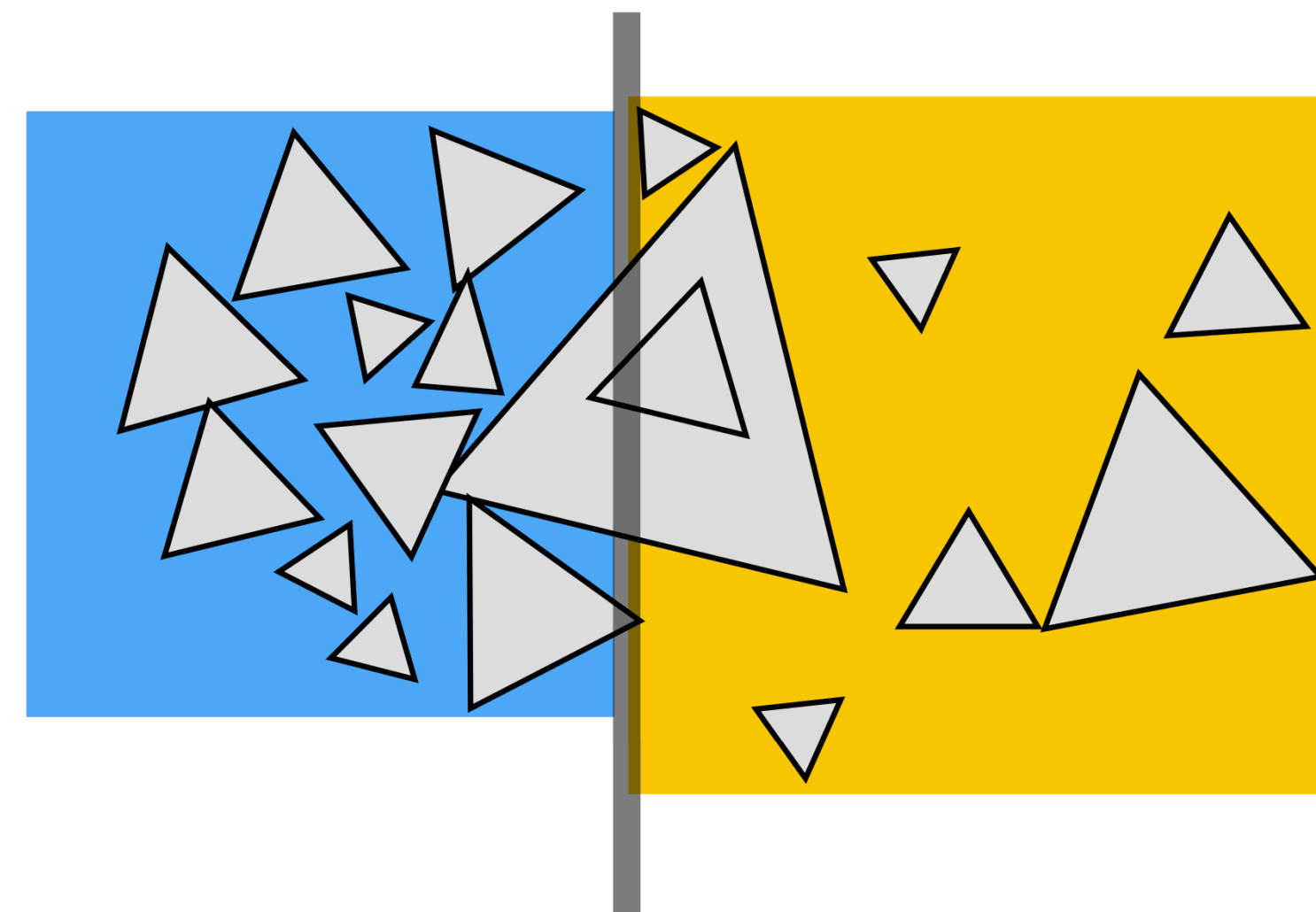
Object partitioning: split primitives into disjoint subsets

- Bounding volumes may overlap
- Each primitive lies in one leaf node



Space partitioning: split space into regions (k-d trees, grids, octrees, ...)

- Regions are non-overlapping
- Primitives may lie in multiple regions

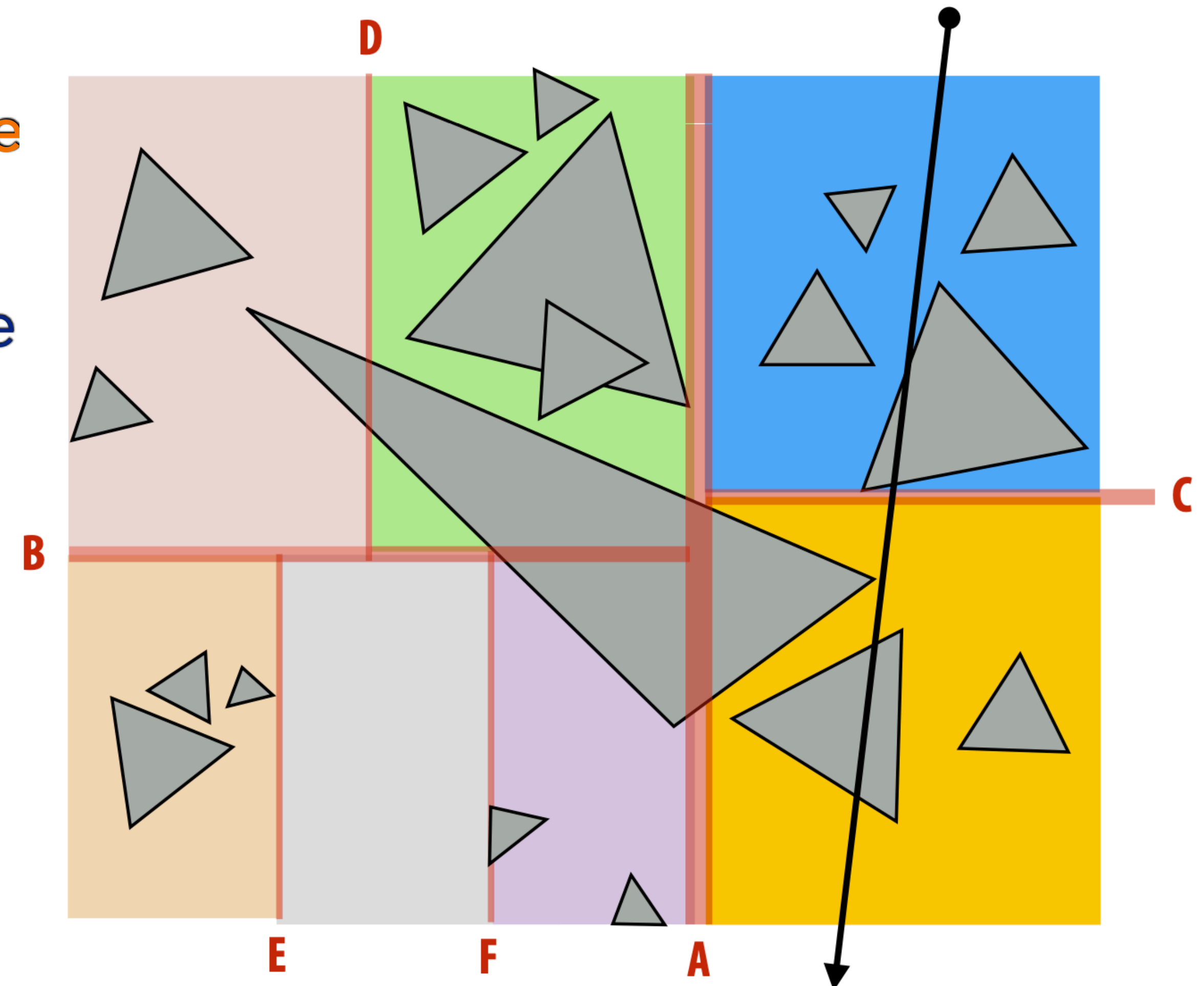
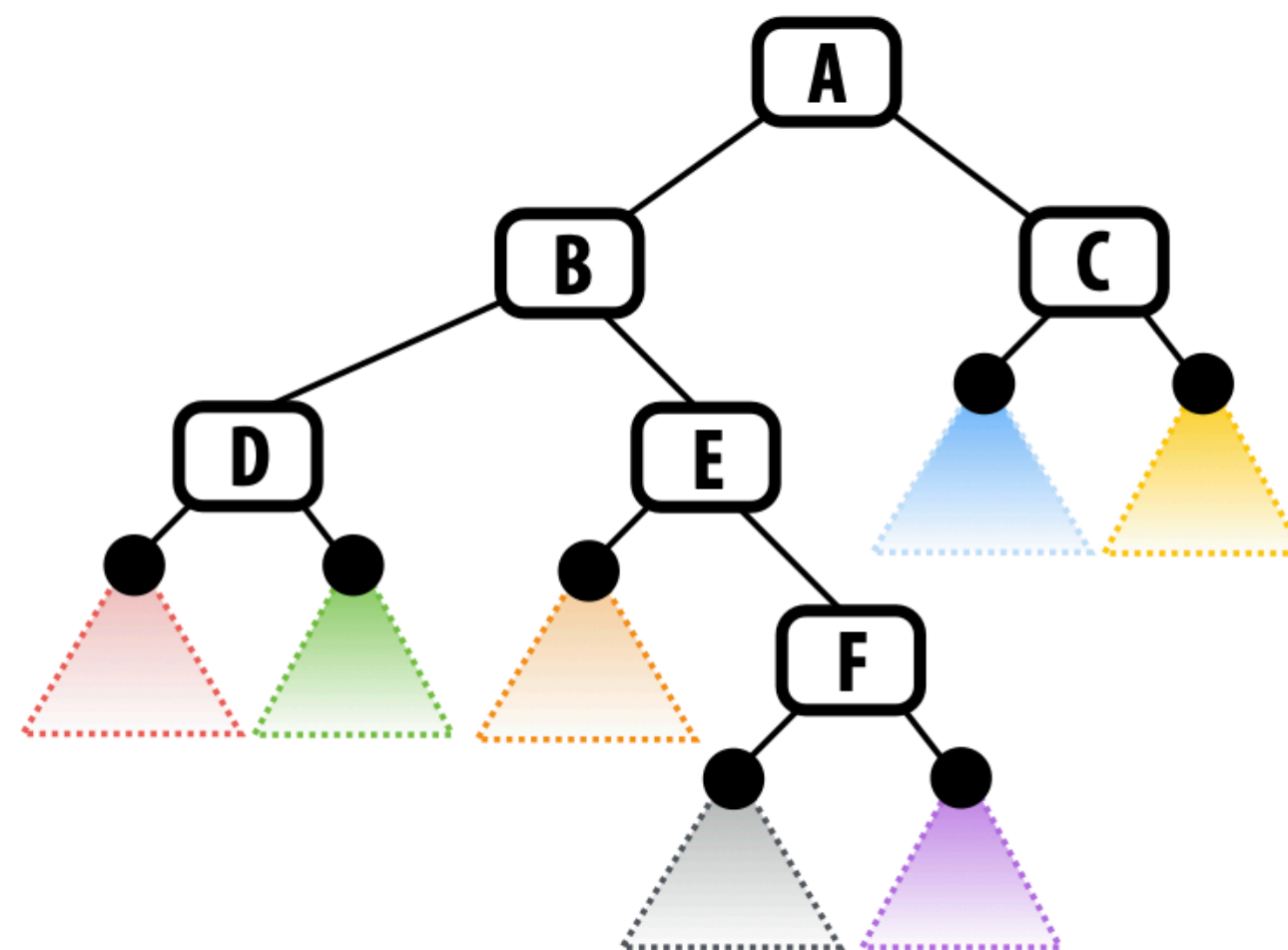


K-d Tree

Divide space via axis-aligned **split plane**

Leaf nodes store list of primitives.

Internal nodes store only the split plane





K-d Tree Traversal

- Keep track of interval $[t_{\min}, t_{\max}]$ covered by current node.=

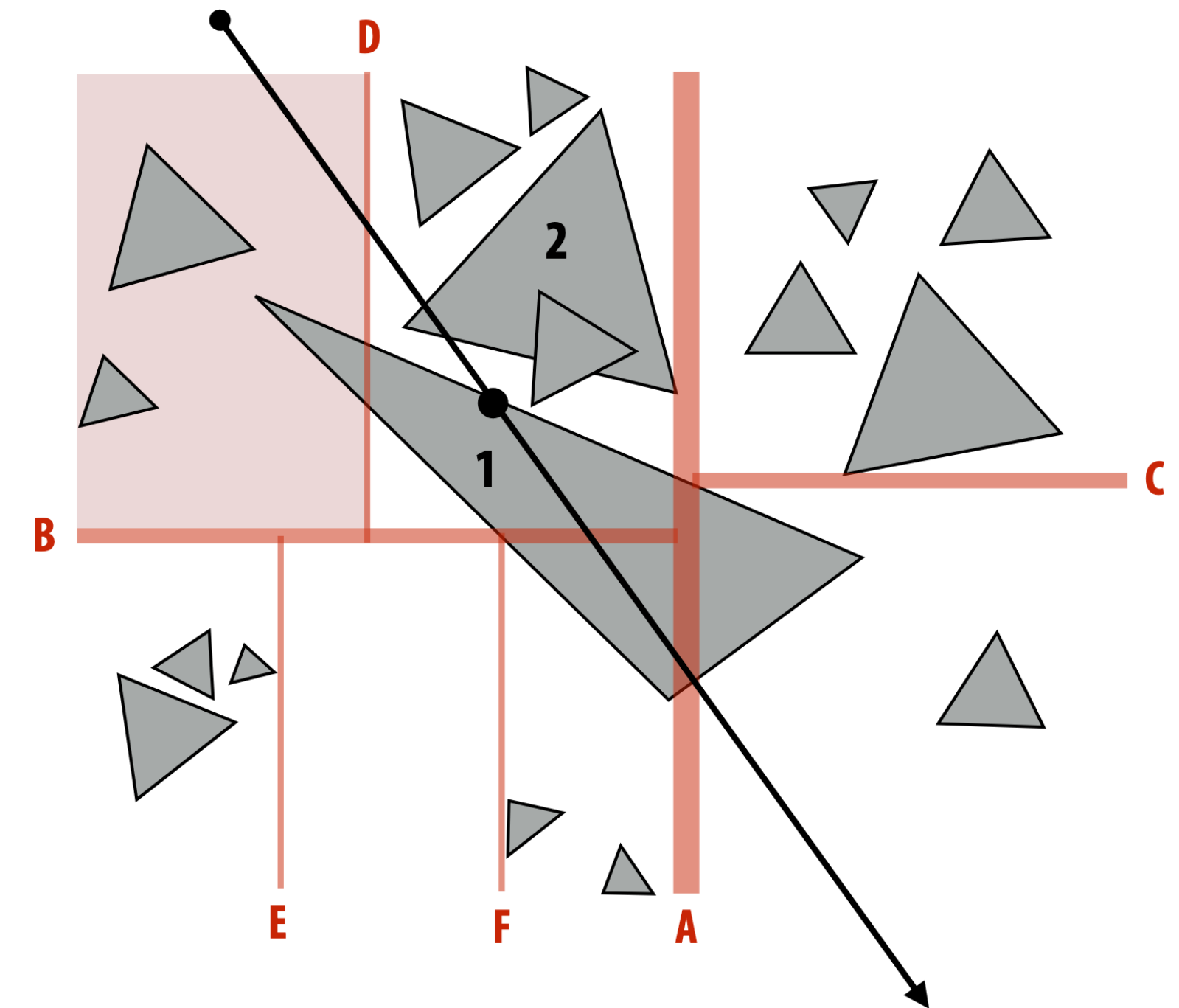
At internal node:

find t_{split} where ray hits split plane

intersect ray with front child

if not hit:

intersect ray with back child





K-d Tree Traversal

- Keep track of interval $[t_{\min}, t_{\max}]$ covered by current node.=

At internal node:

find t_{split} where ray hits split plane

intersect ray with front child

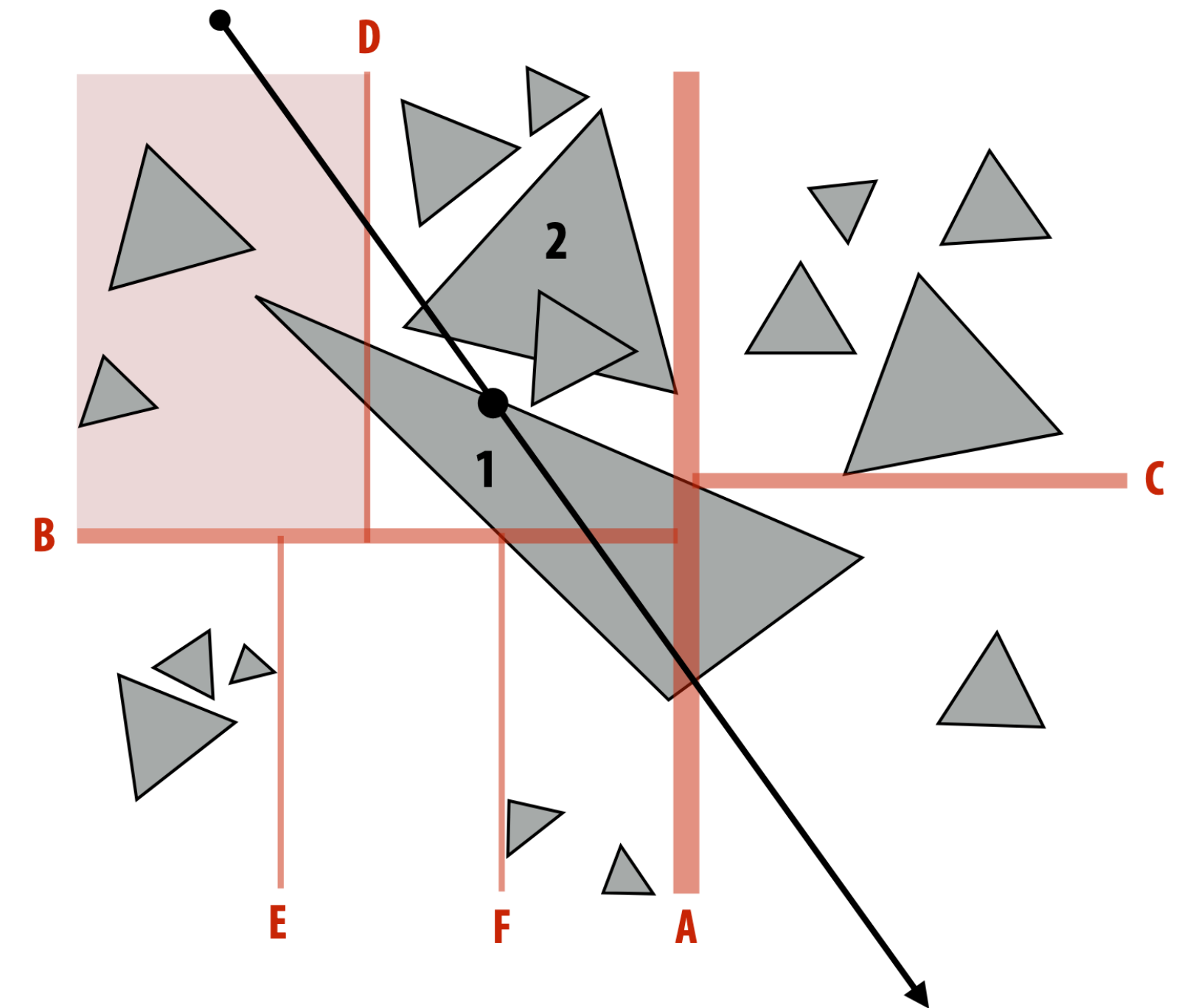
if not hit:

intersect ray with back child

At leaf:

intersect ray with all primitives

return the earliest hit that occurs in $[t_{\min}, t_{\max}]$





K-d Tree Traversal

- Keep track of interval $[t_{\min}, t_{\max}]$ covered by current node.=

At internal node:

find t_{split} where ray hits split plane

intersect ray with front child

if not hit:

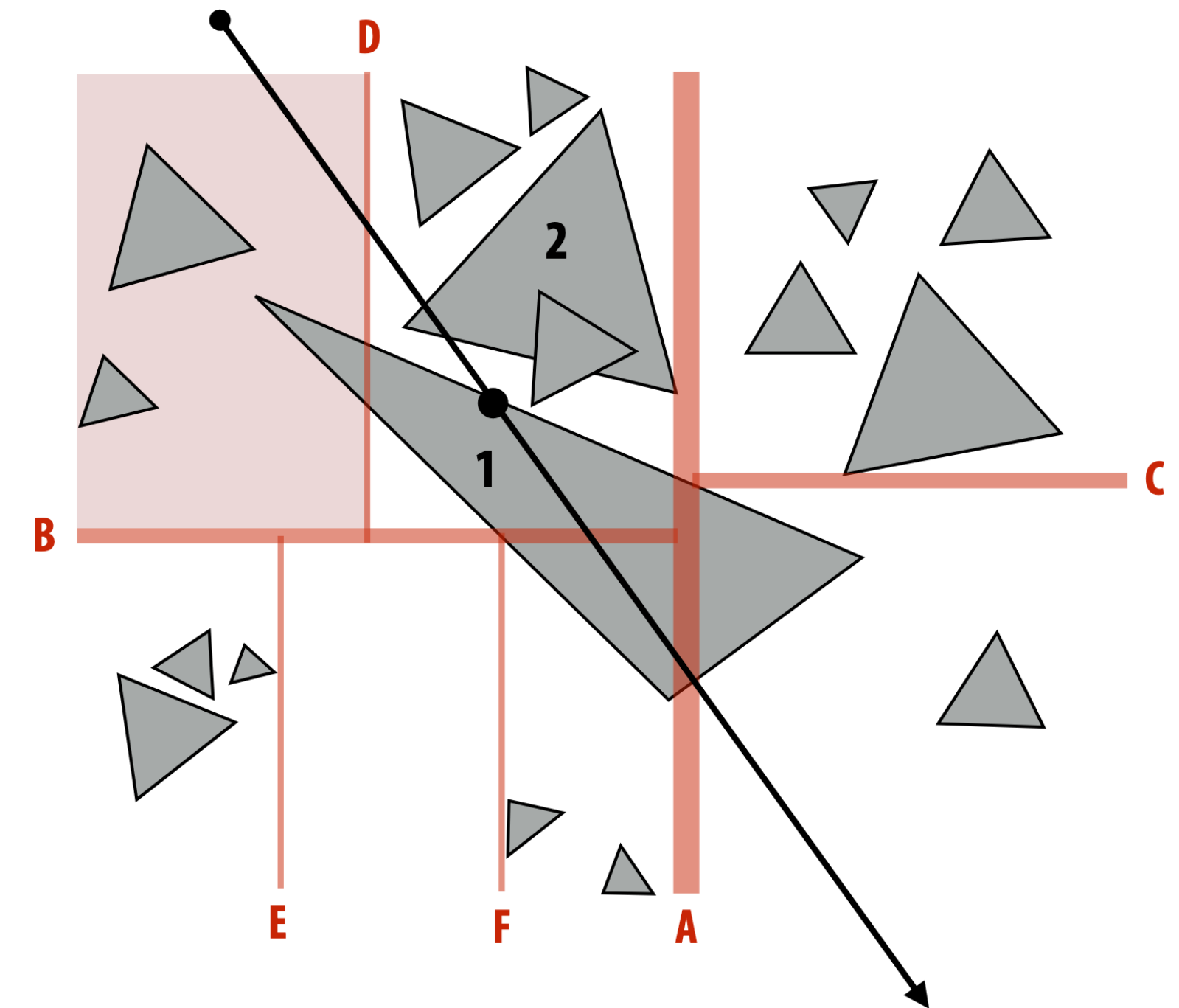
intersect ray with back child

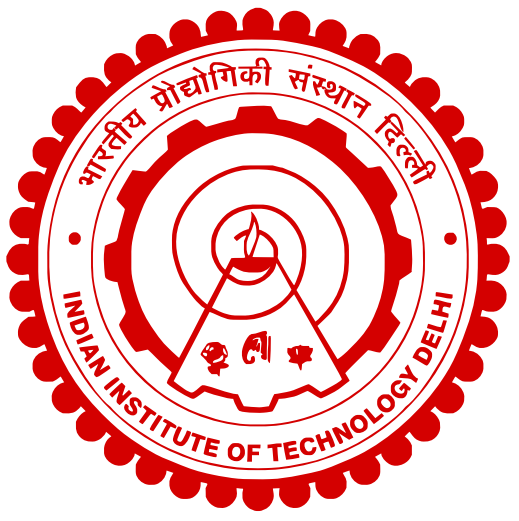
At leaf:

intersect ray with all primitives

return the earliest hit **that occurs in $[t_{\min}, t_{\max}]$**

(Can cache hit info per primitive to avoid repeated intersection computation)

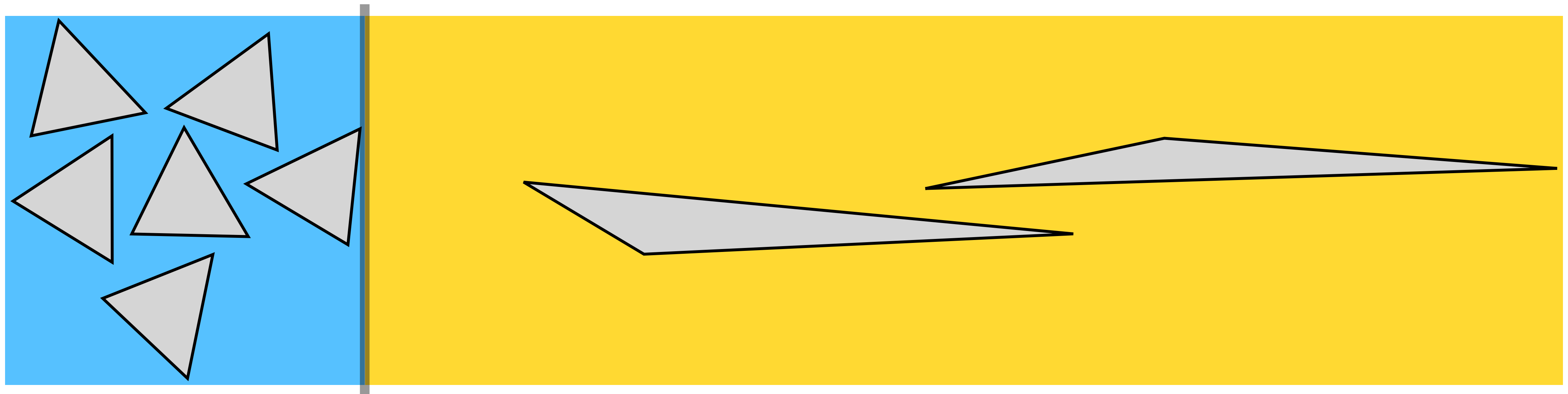




K-d Tree Construction

- Set root node = bounding box of all primitives in scene
- Recursively create child nodes by choosing split planes

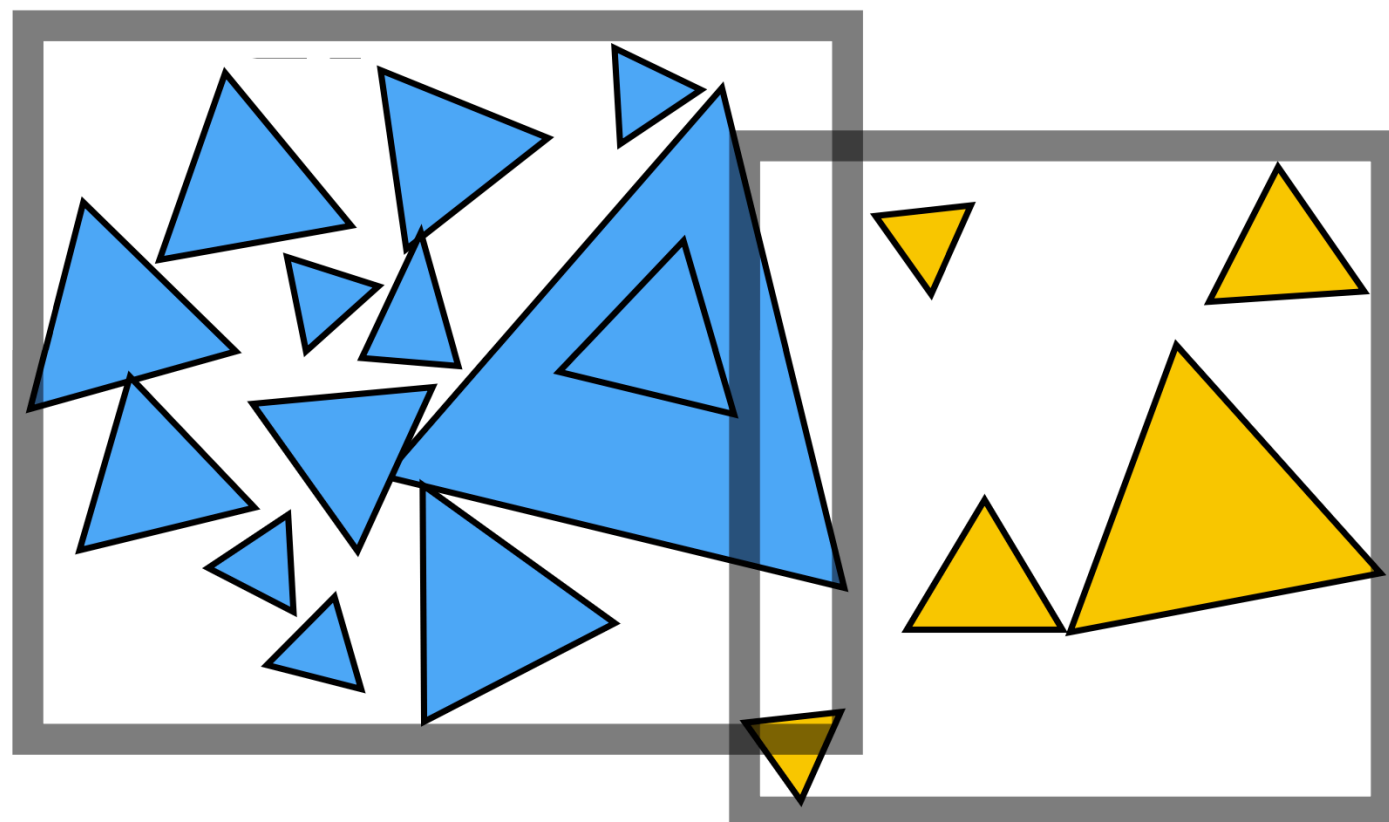
How to choose a split plane? Can reuse same surface area heuristic as in BVH



Pros and cons

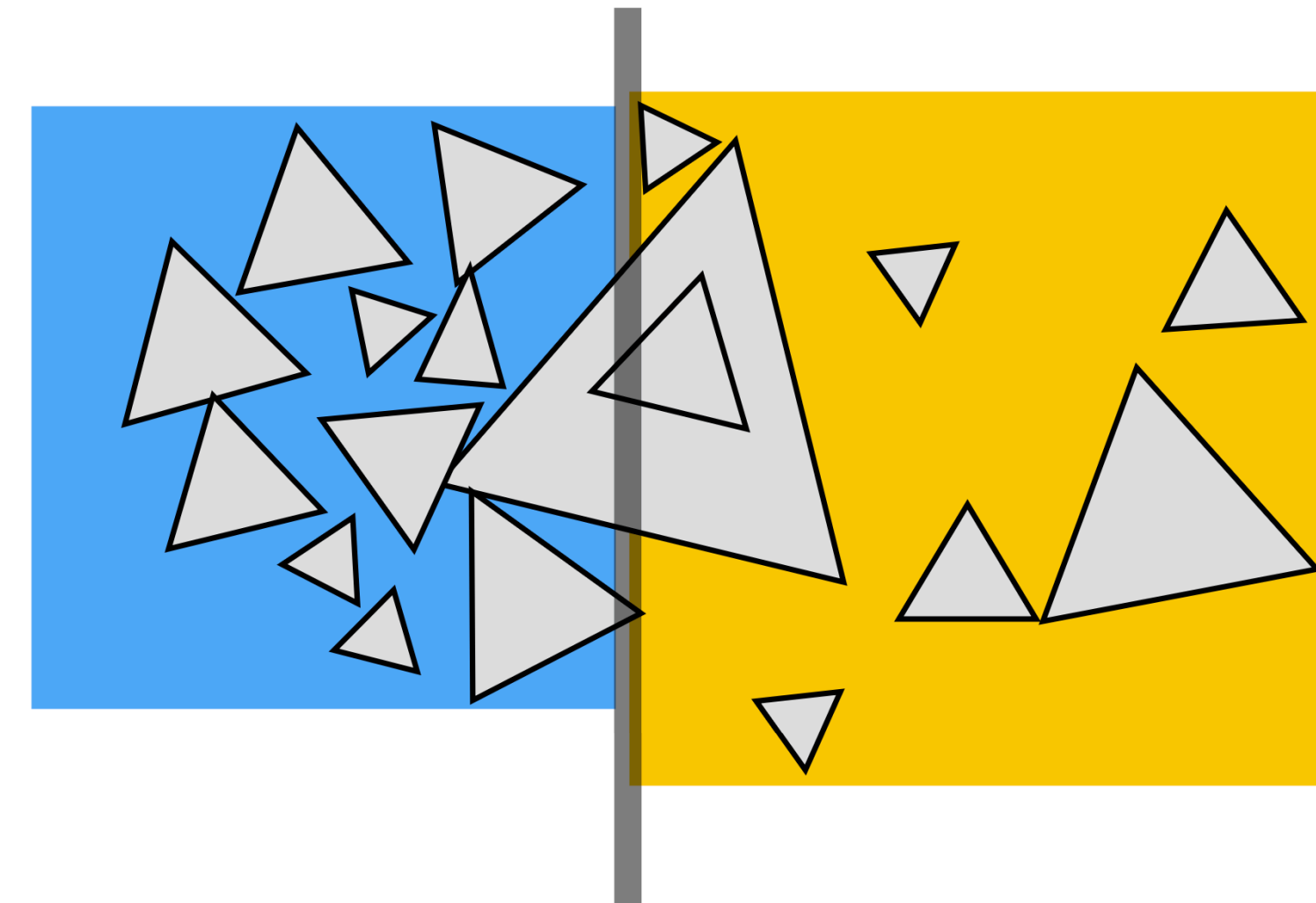
BVHs

- + Easier to update
- + More flexible (can include OBBs, k-DOPs, ...)

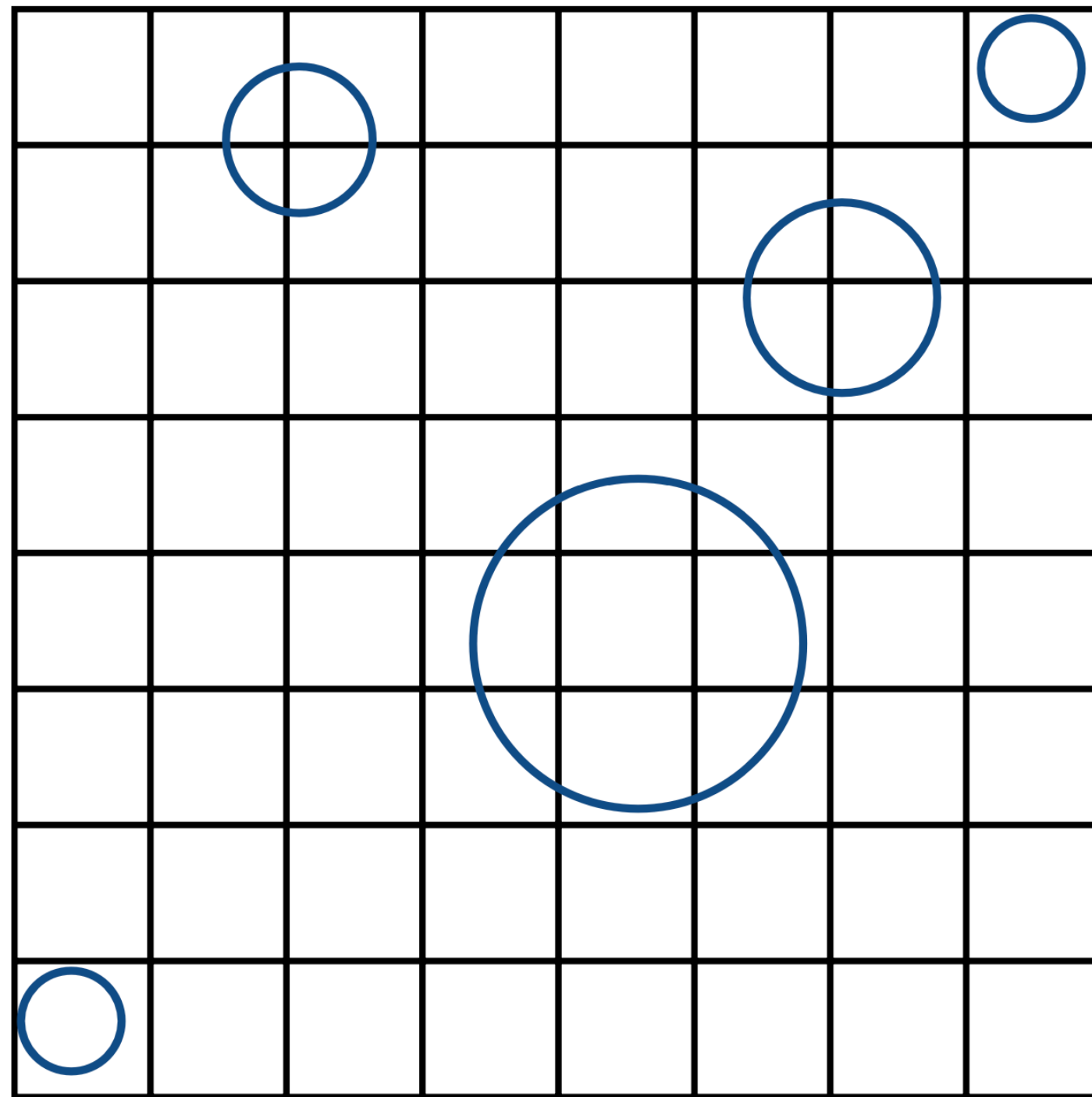


k-d trees

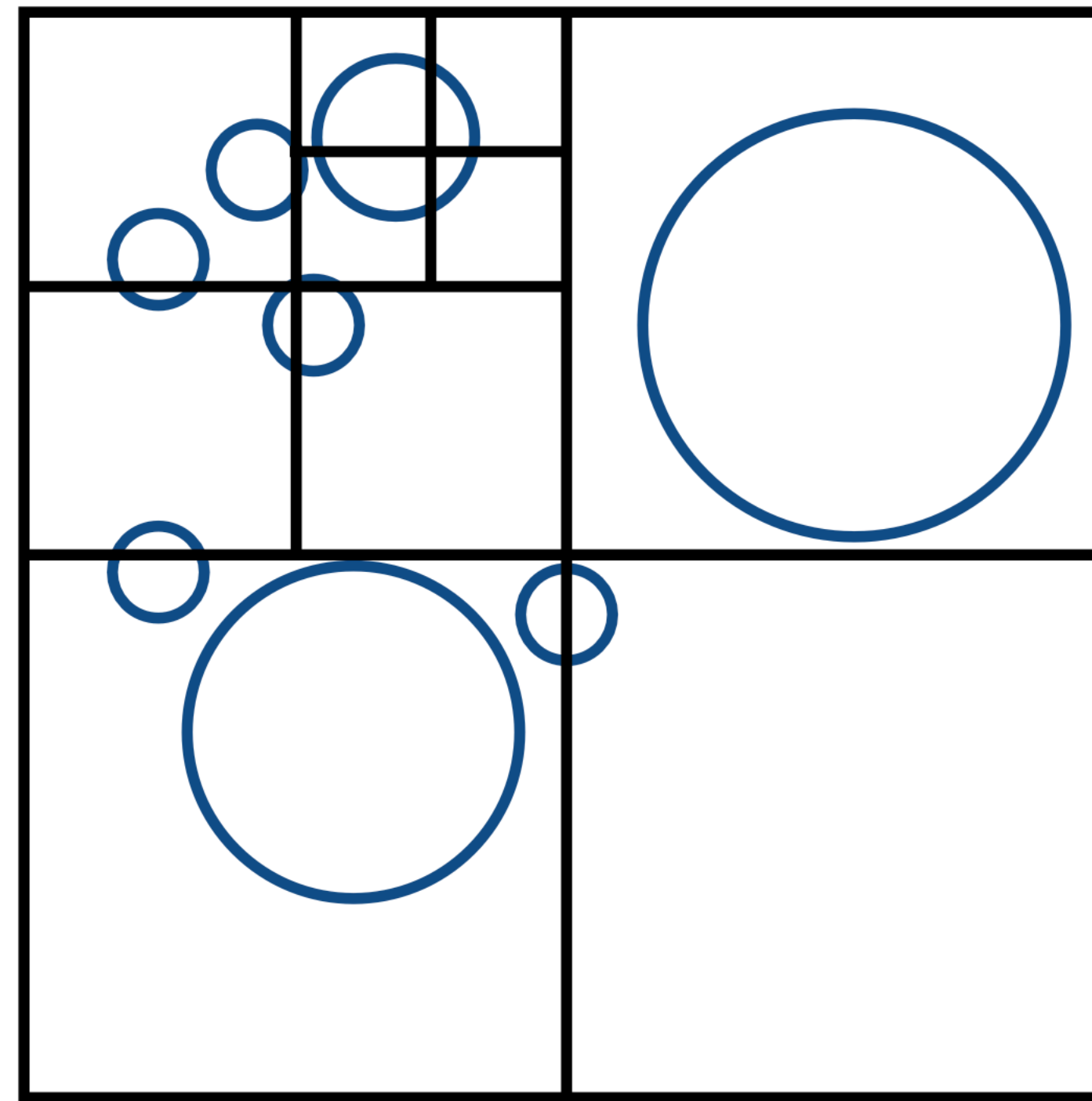
- + No overlap between regions
- + Can terminate on first hit
- More empty space
- May hit primitive multiple times



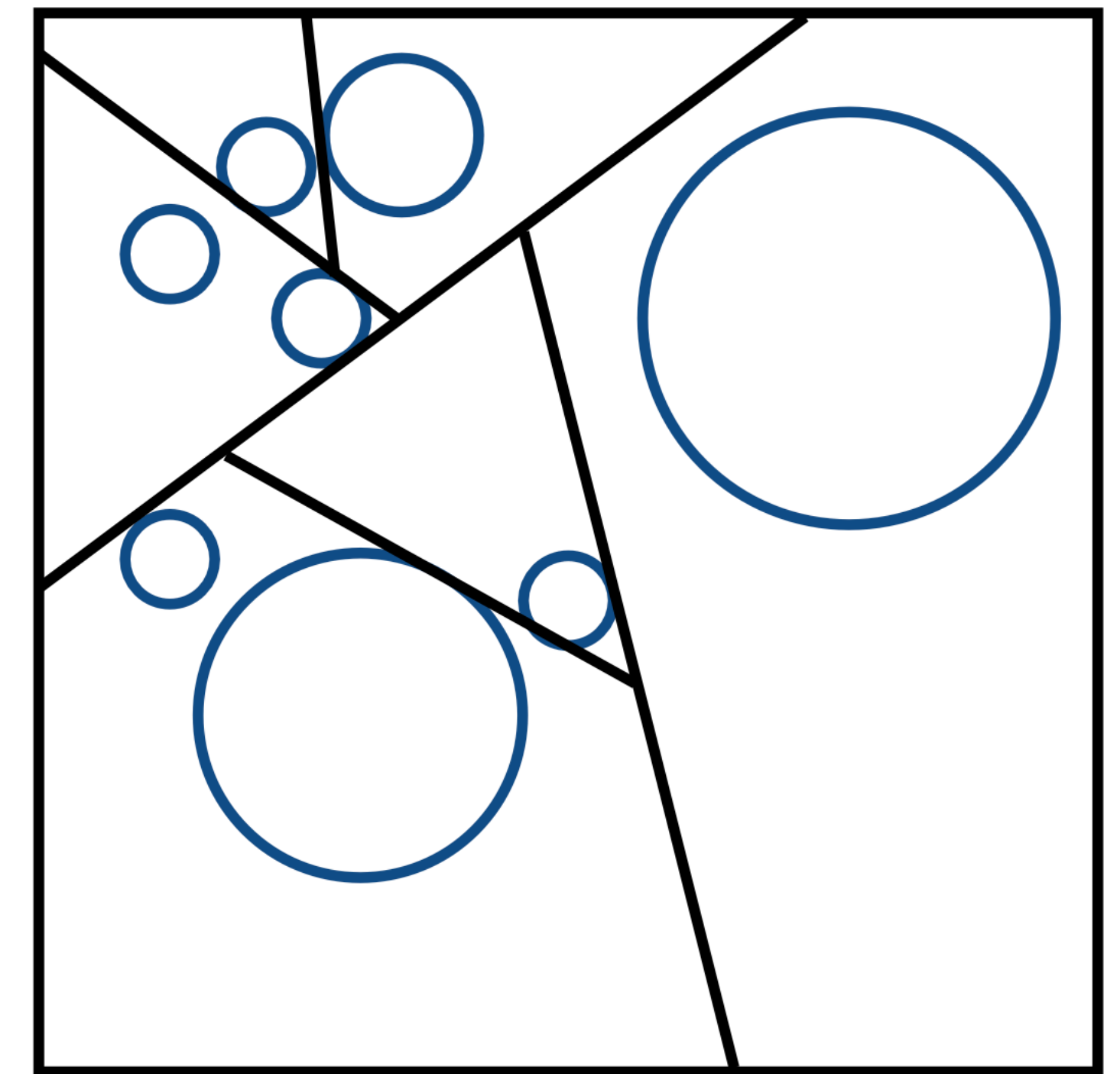
More Space Partitioning Techniques



Uniform grids



Quadtrees / octrees

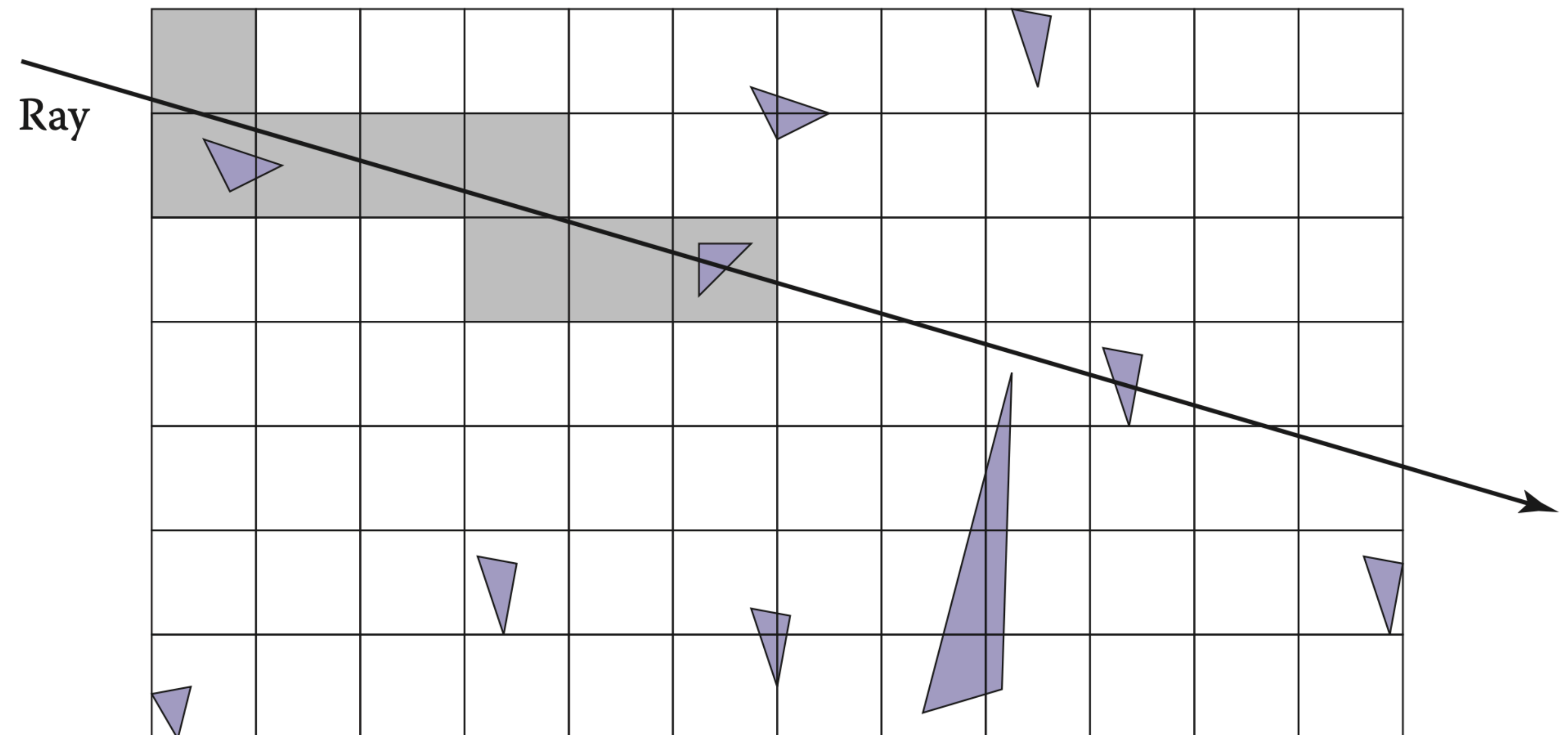


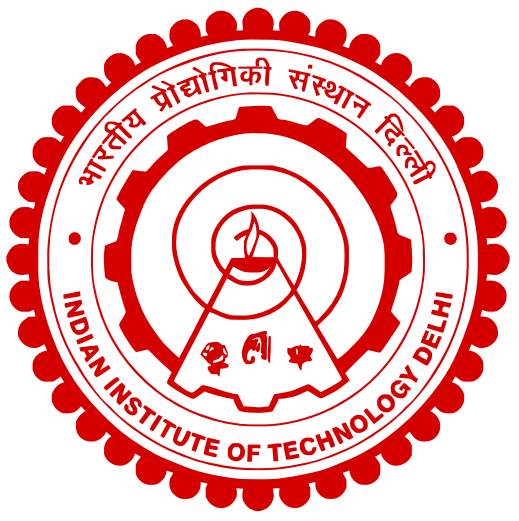
Binary space partitioning
(BSP trees)



Uniform Grids

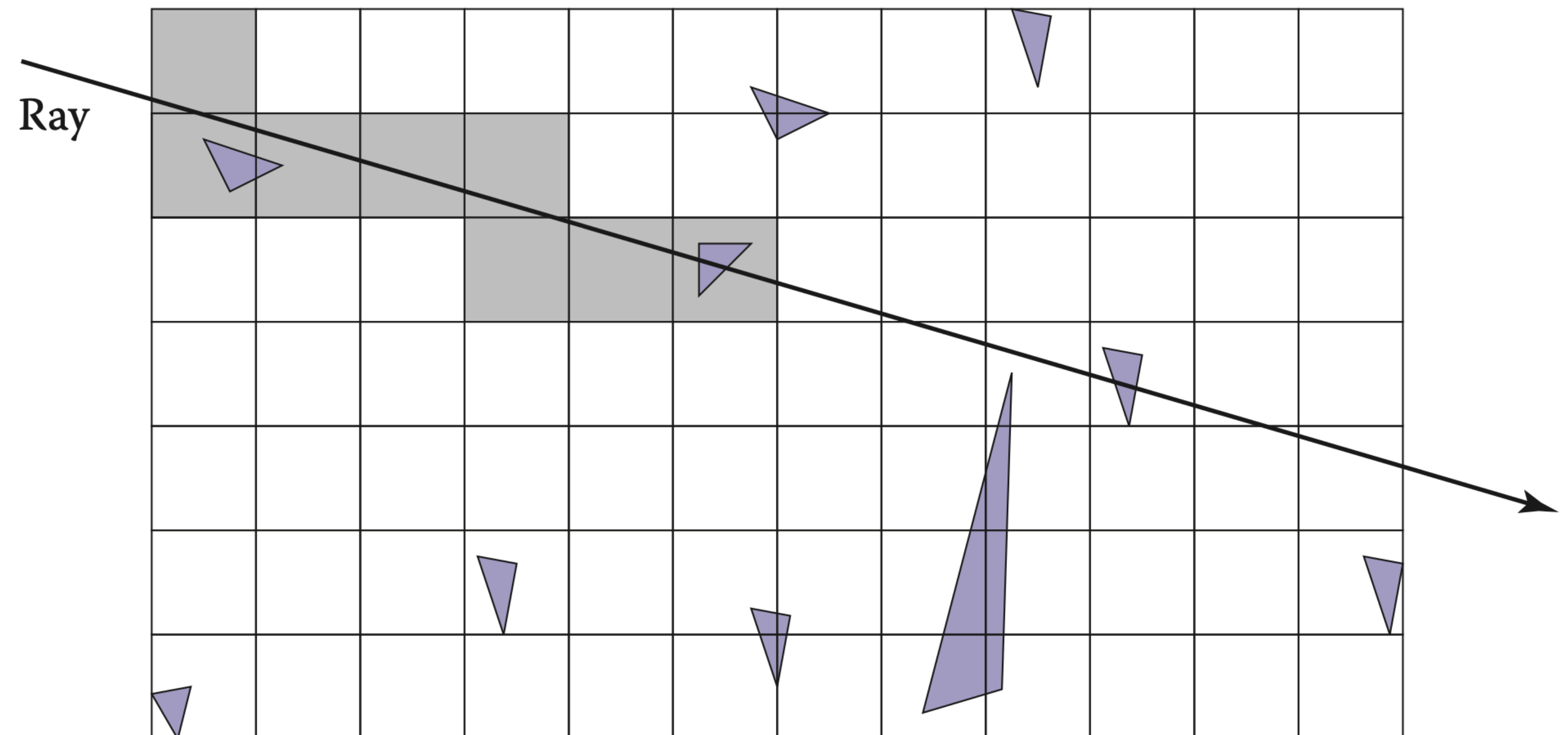
- Very simple: divide bounding box into equal sized cells.
- Each cell stores list of overlapping primitives.

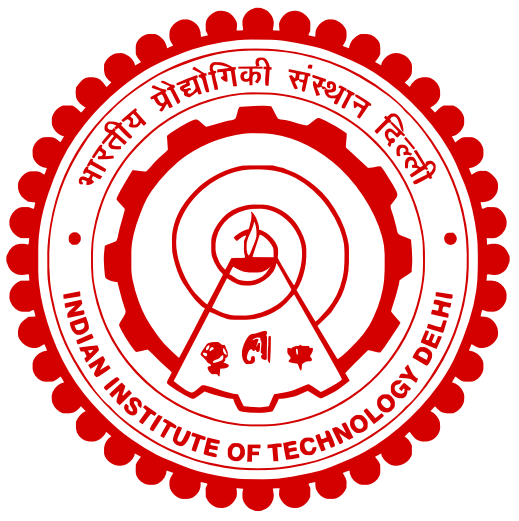




Uniform Grids

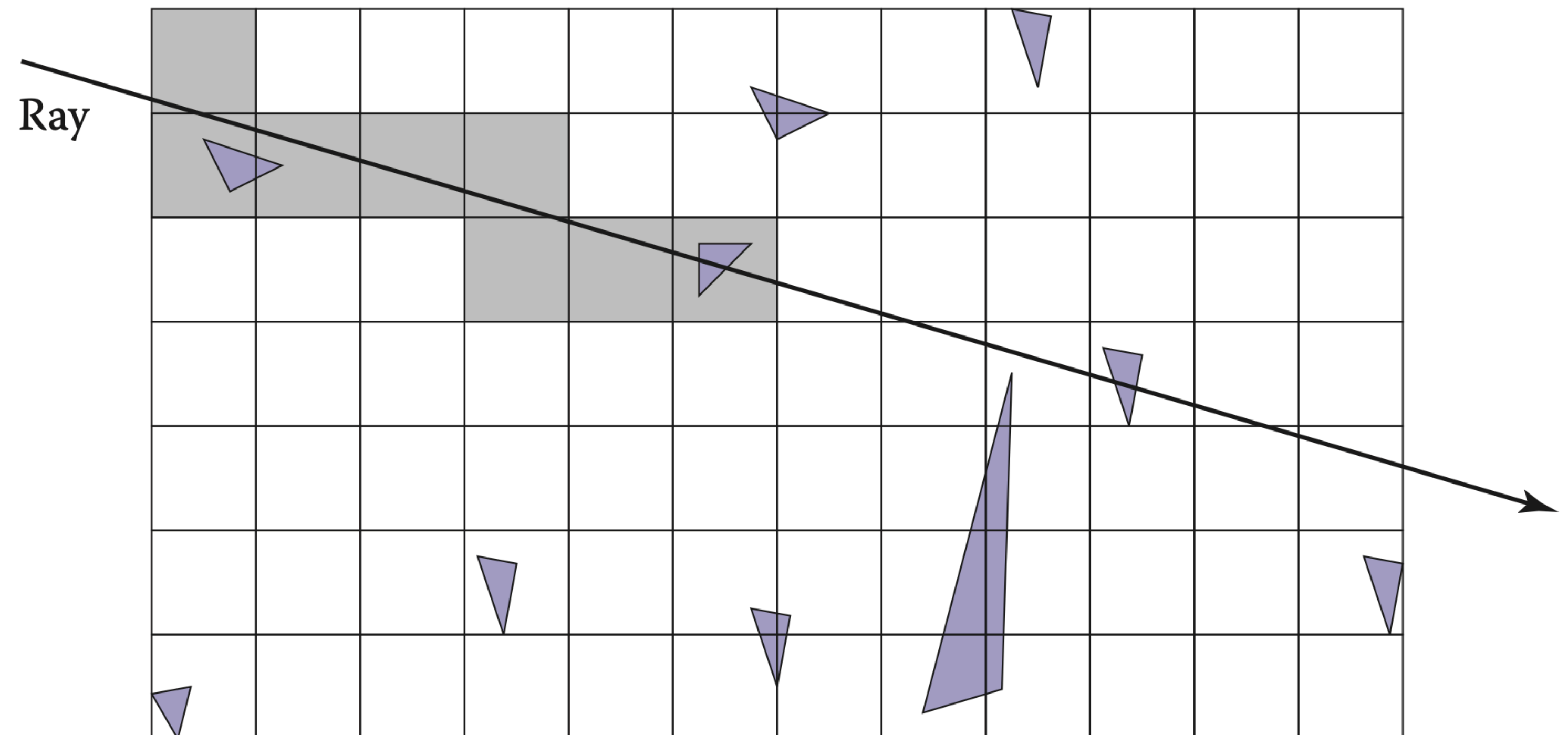
- Very simple: divide bounding box into equal sized cells.
- Each cell stores list of overlapping primitives.
- Easy to traverse (like rasterization)





Uniform Grids

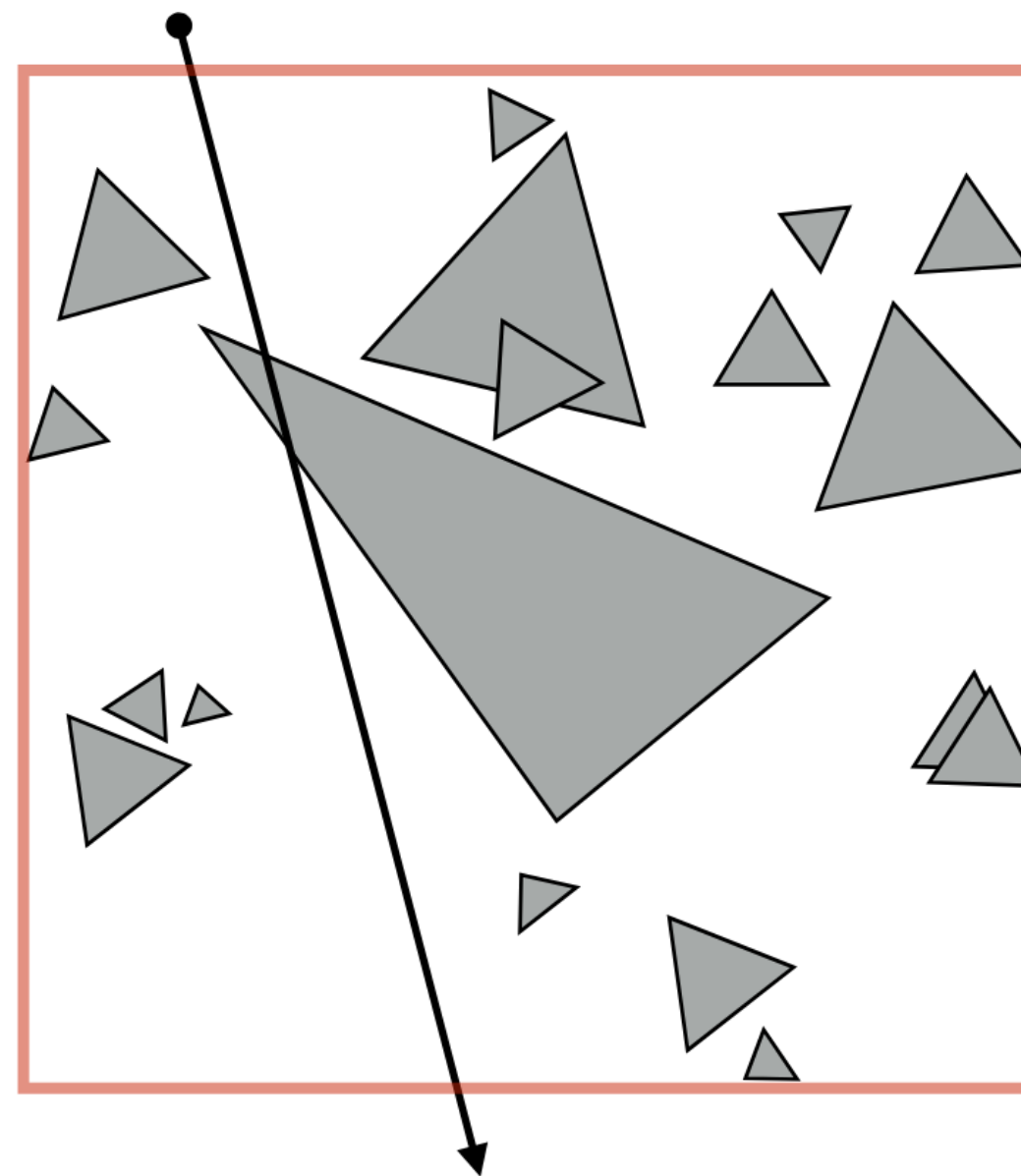
- Very simple: divide bounding box into equal sized cells.
- Each cell stores list of overlapping primitives.
- Easy to traverse
(like rasterization)
- Parallelization works well



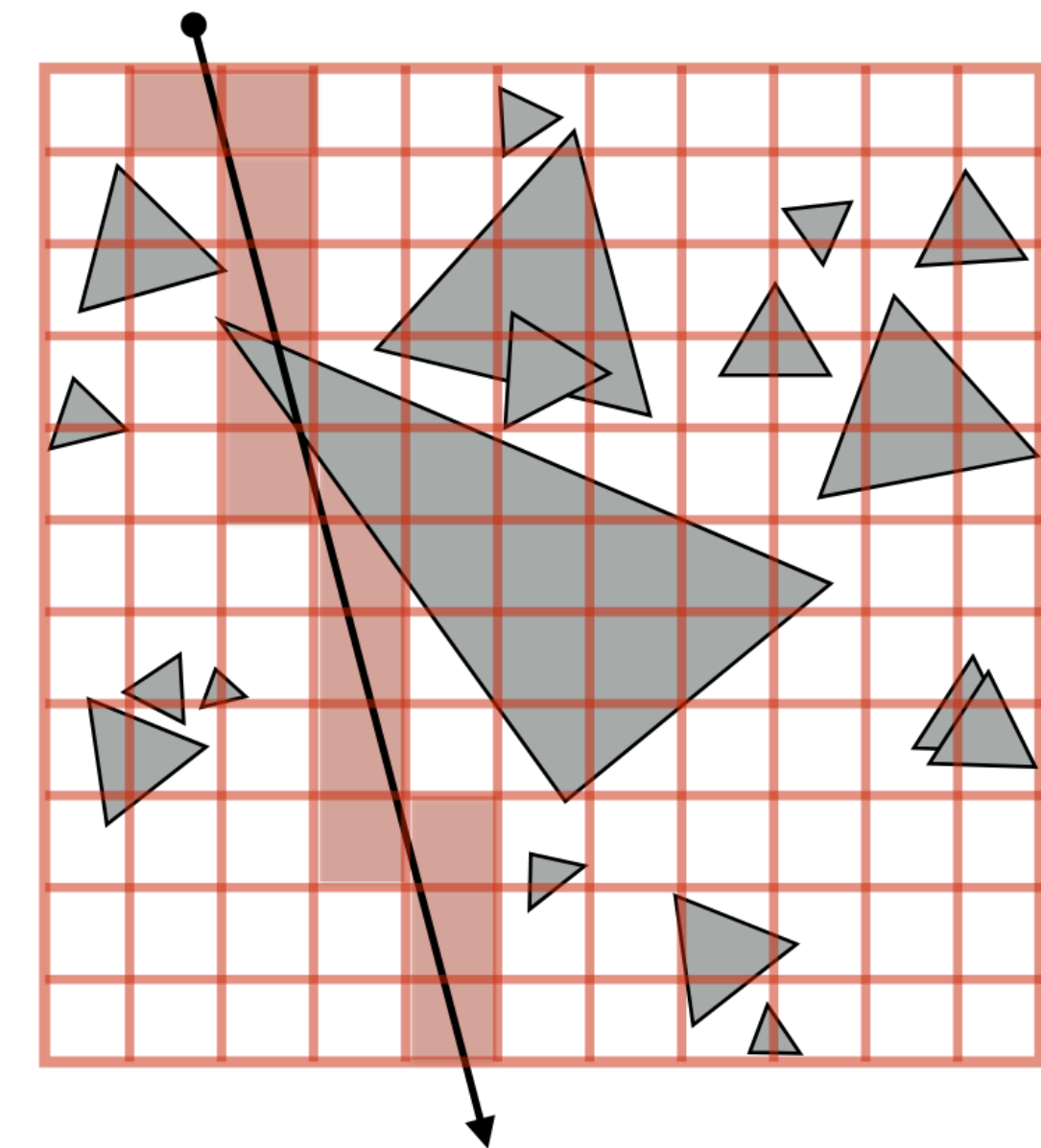
Grid Design



- What should be the grid resolution?



Too few cells

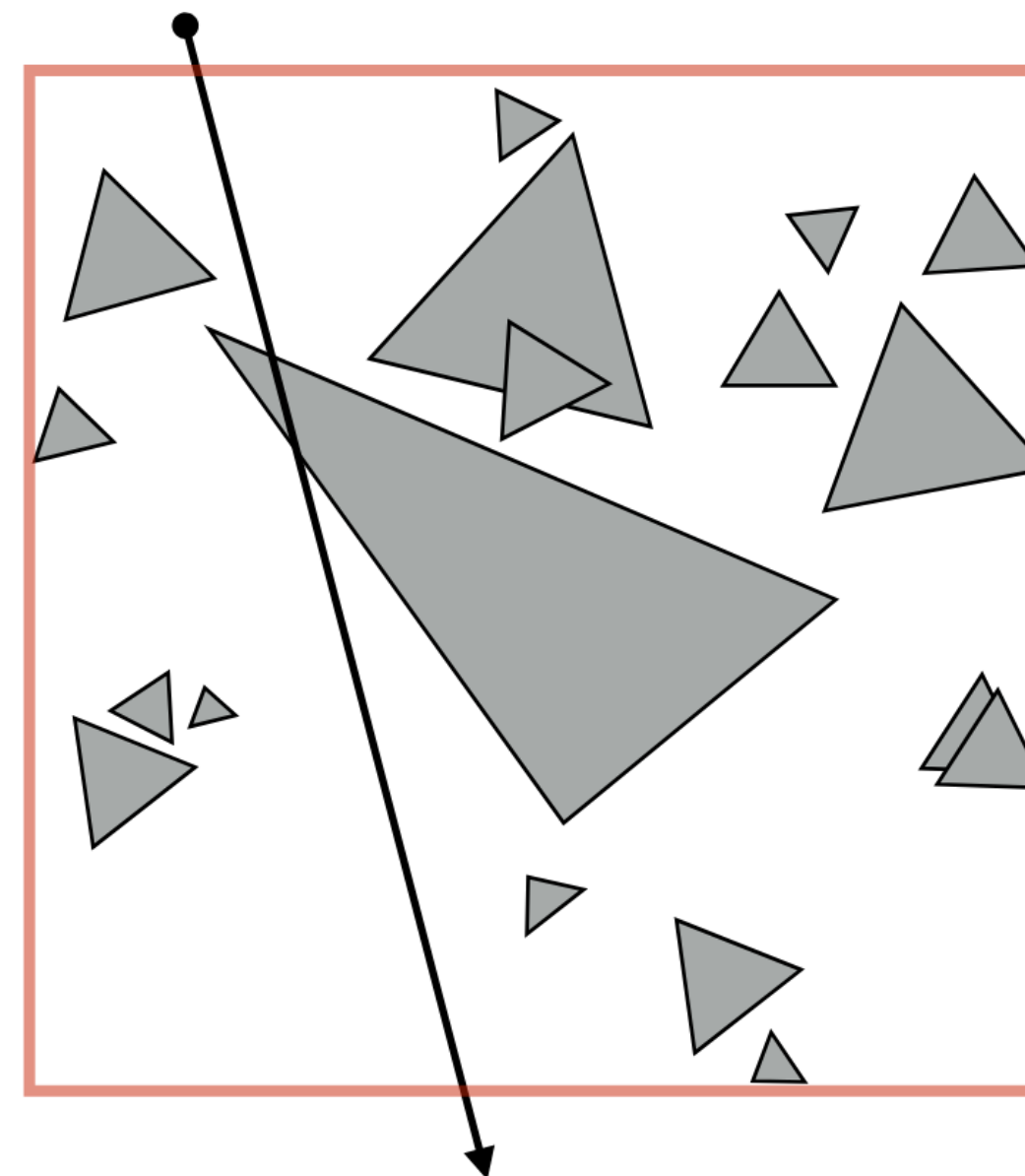


Too many cells

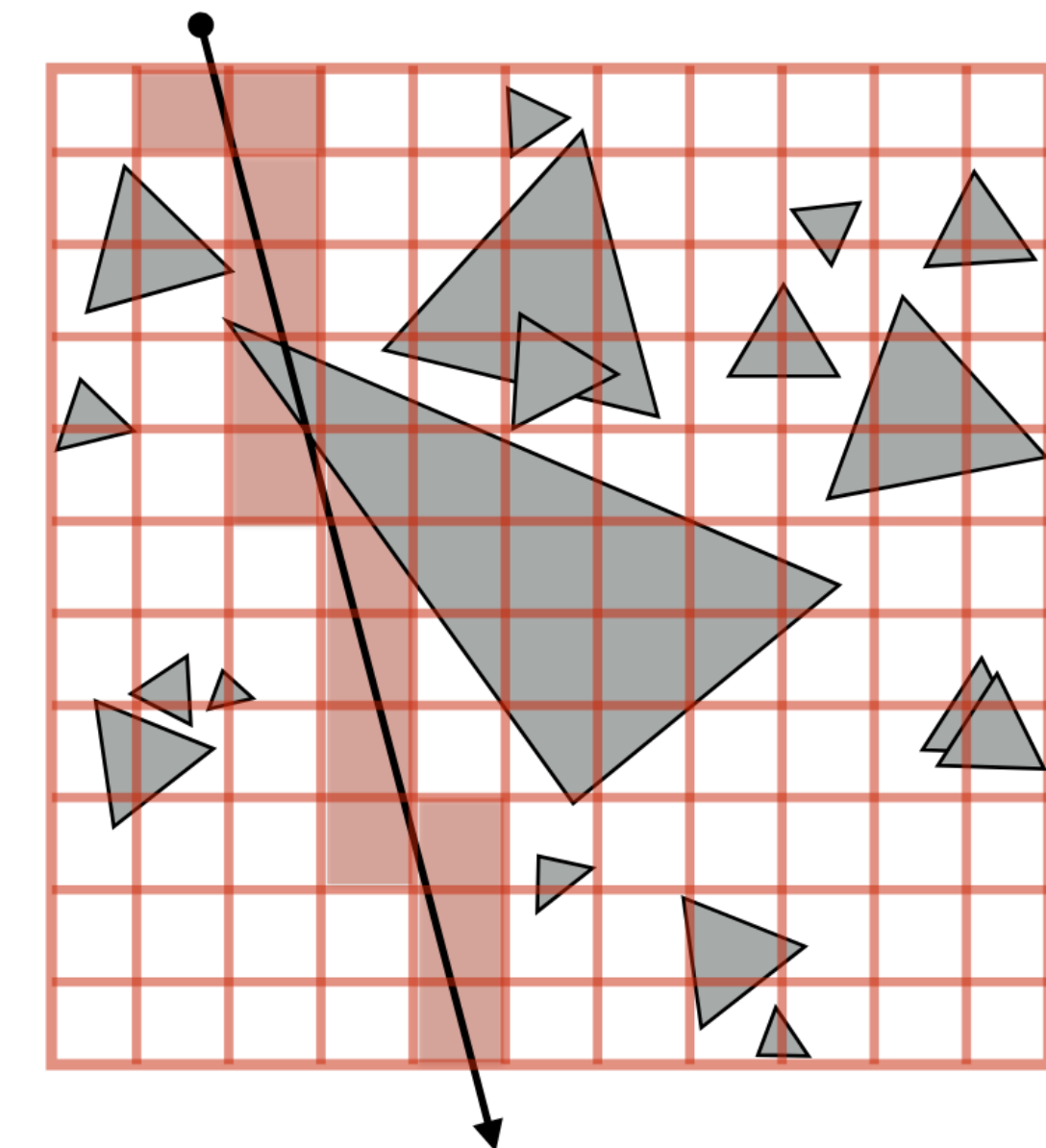
Kayvon Fatahalian

Grid Design

- What should be the grid resolution?
- Good rule of thumb: $\#cells \approx \#primitives$



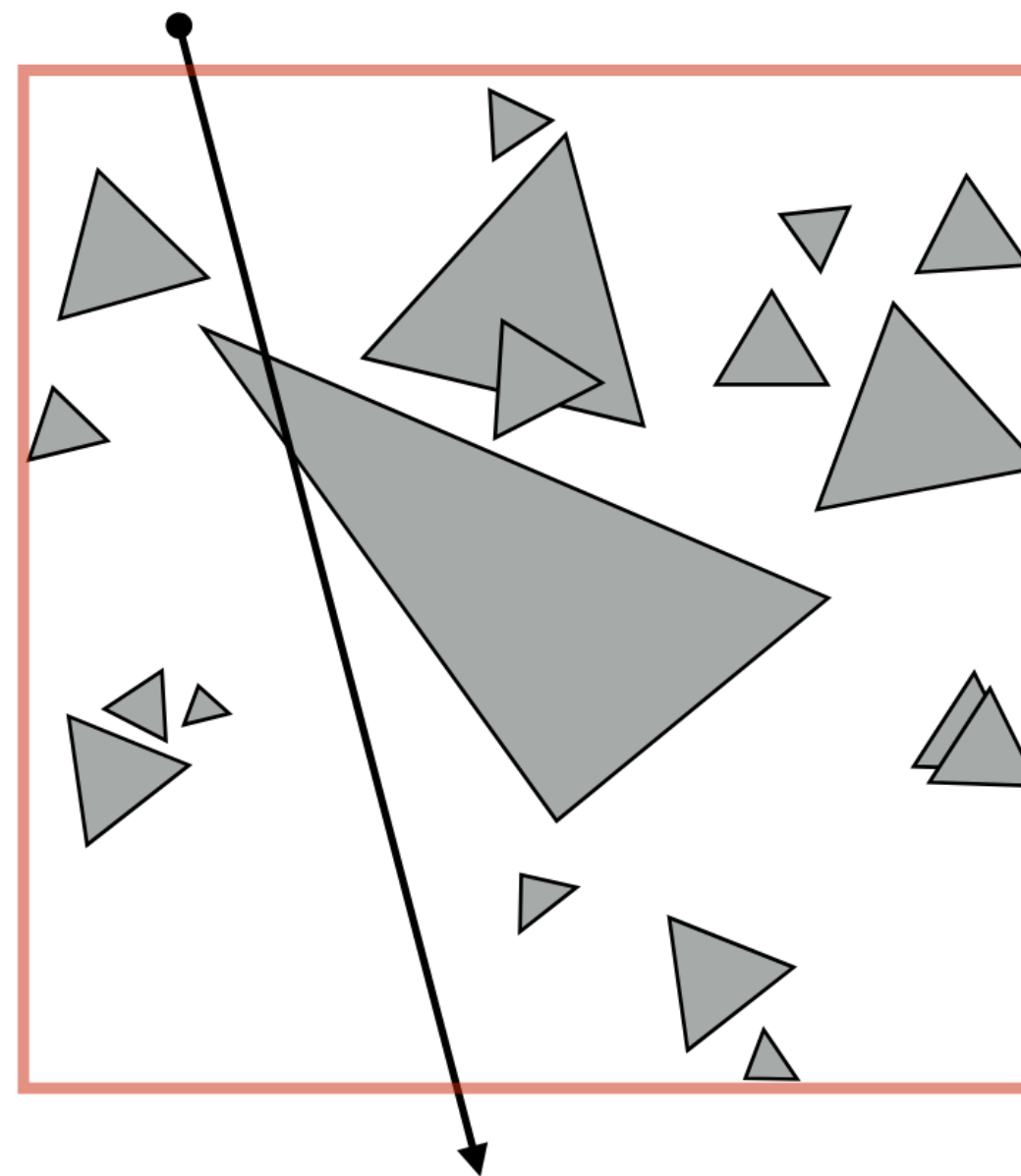
Too few cells



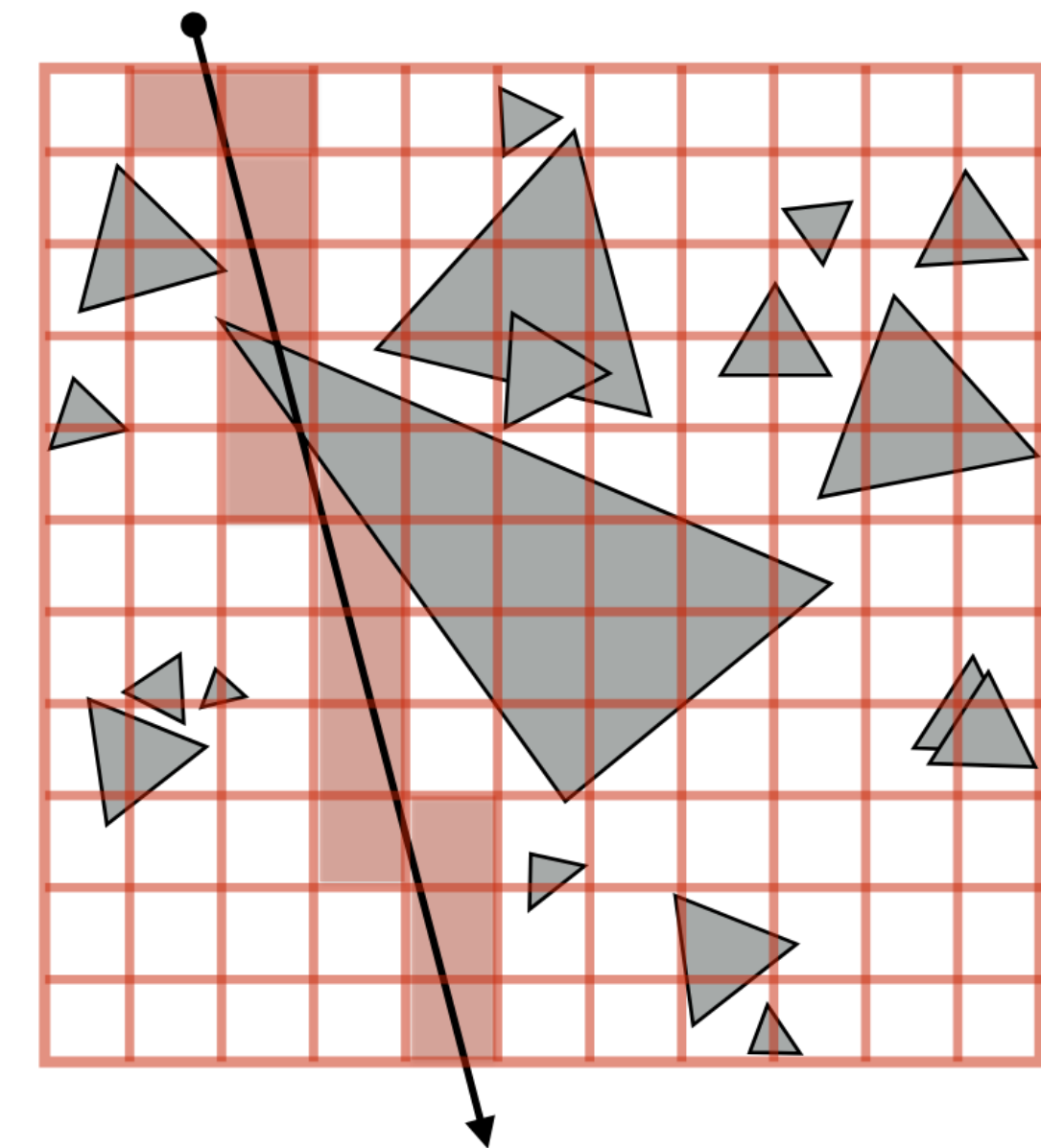
Too many cells

Grid Design

- What should be the grid resolution?
- Good rule of thumb: $\#cells \approx \#primitives$
- Ray hits $O(\sqrt[3]{N})$ cells



Too few cells



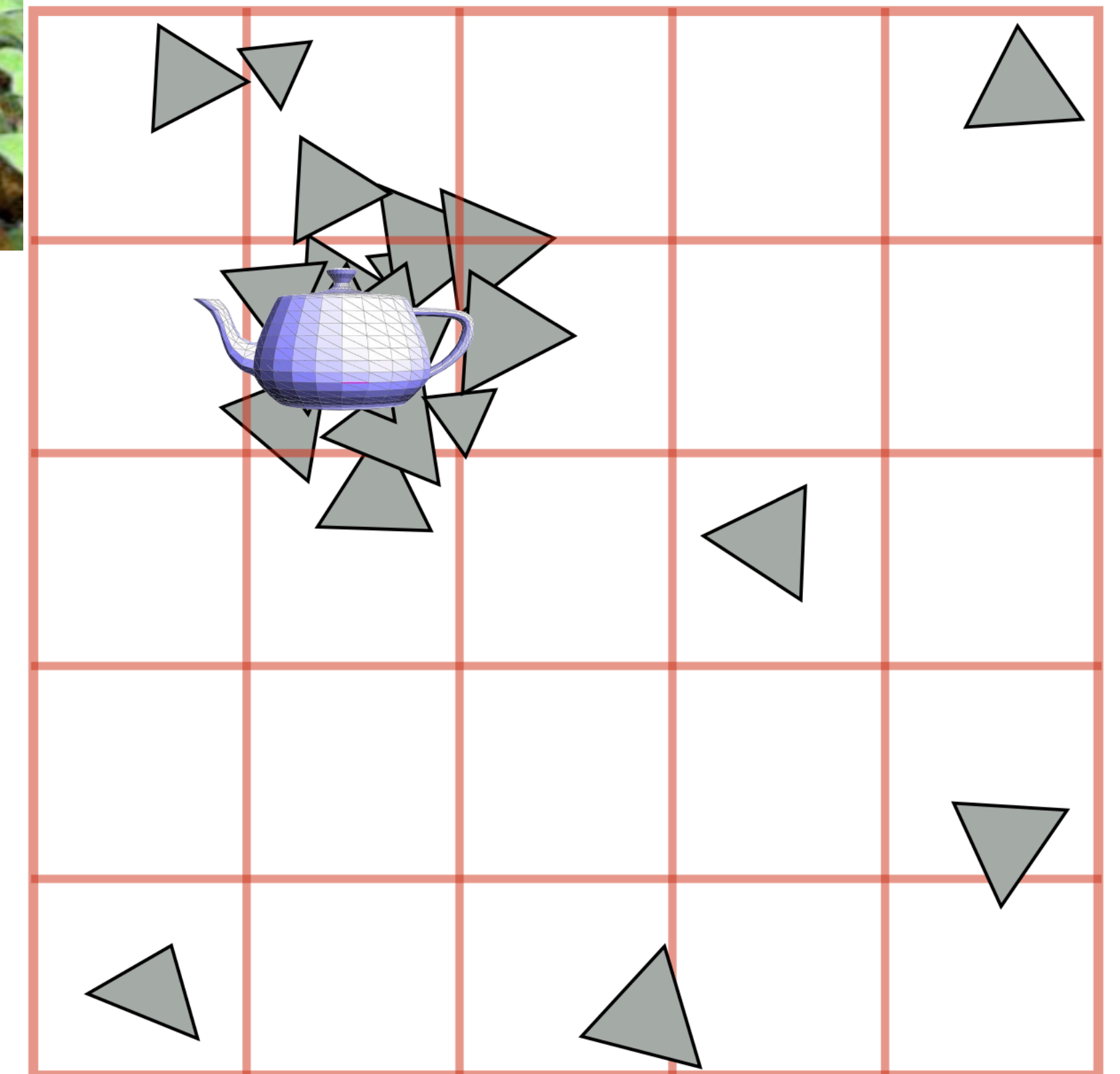
Too many cells



Good case for grids: 
roughly uniform distribution of primitives

Bad case for grids: 

"Teapot in a stadium" problem



Dynamic scenes with many moving objects and millions of total triangles..

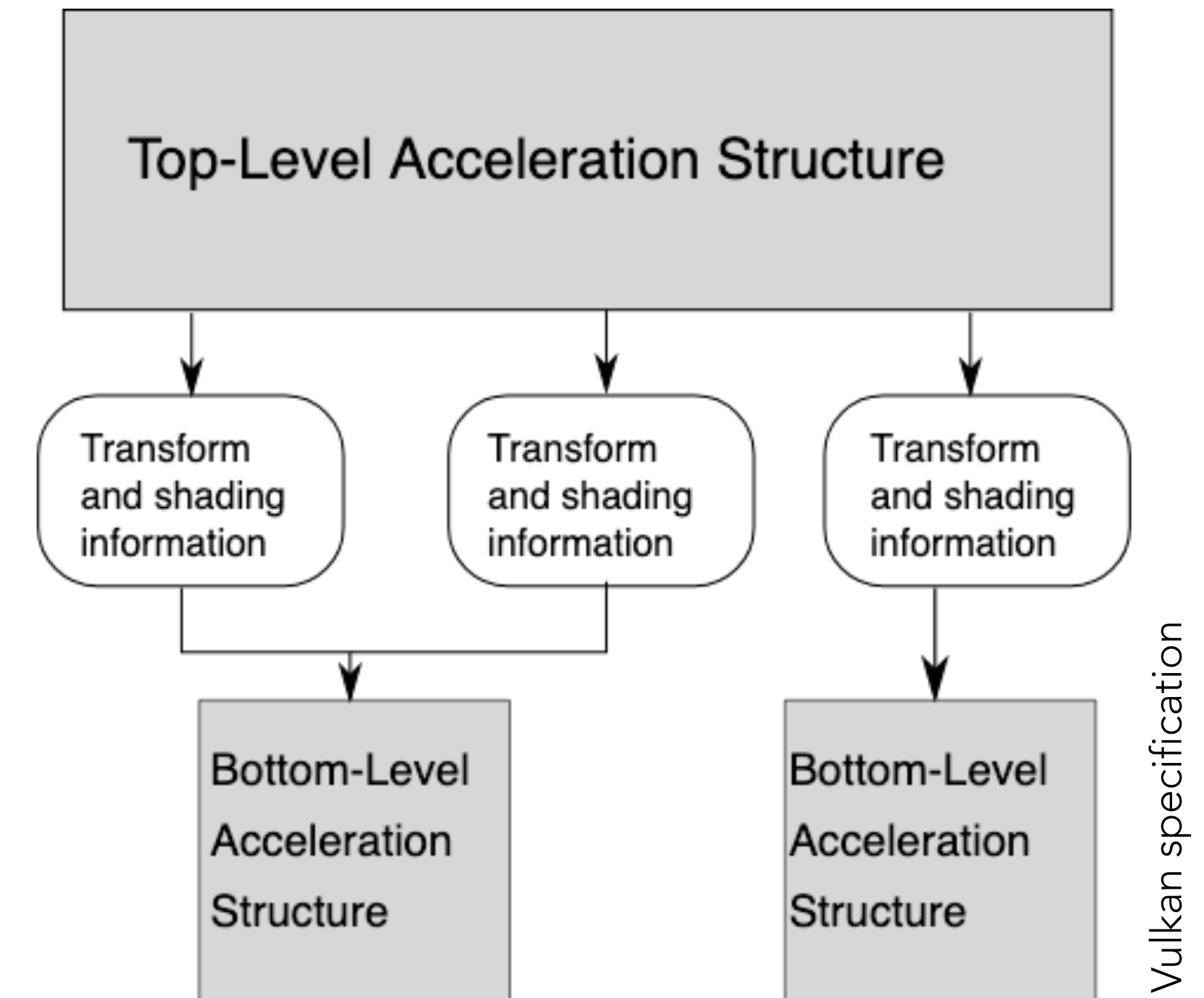


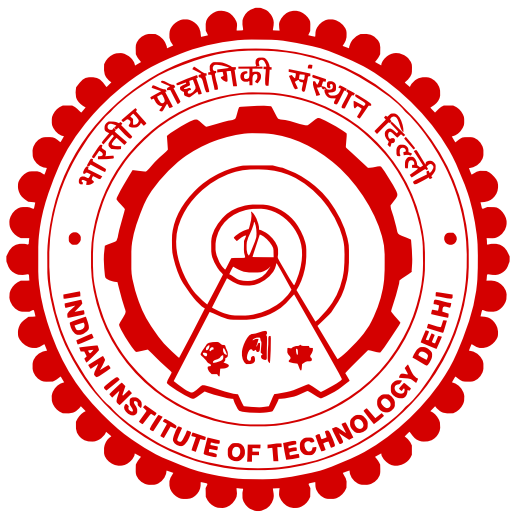
Two-level acceleration structures



Per object: **bottom-level acceleration structure**

- Built on scene initialization
- (Refit per frame if non-rigidly deformed)





Two-level acceleration structures

Per object: **bottom-level acceleration structure**

- Built on scene initialization
- (Refit per frame if non-rigidly deformed)

Top-level acceleration structure for the whole scene

- Leaves point to object BLASes
- Rebuilt per frame based on current object positions

