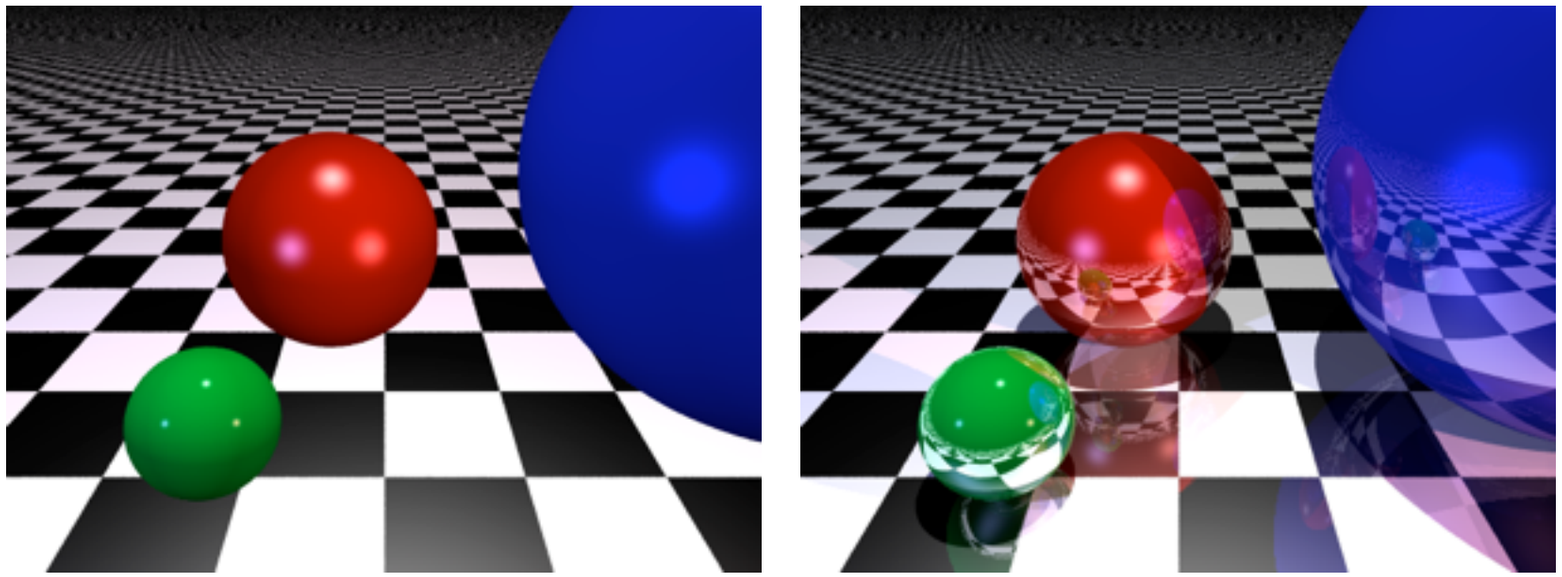
Introduction to Computer Graphics

Ray Tracing Acceleration



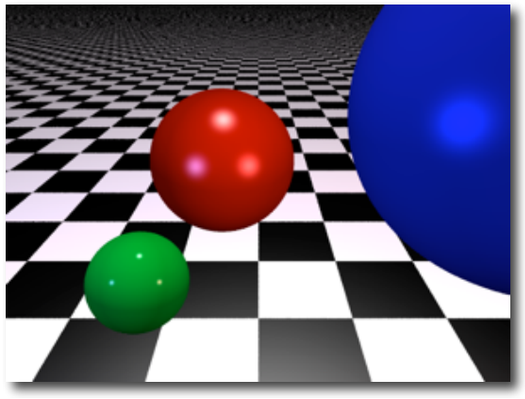
Prof. Dr. Mario Botsch

Computer Graphics & Geometry Processing

Universität Bielefeld

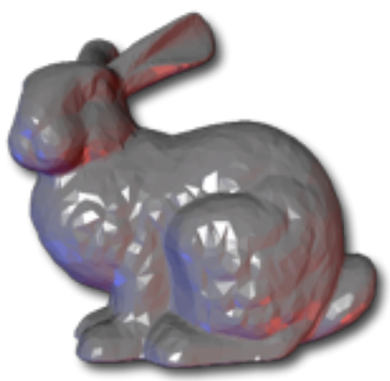
CITEC

Overview

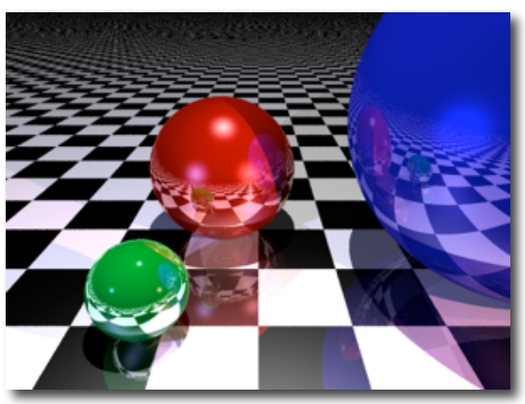


Rendering Pipeline

Geometry

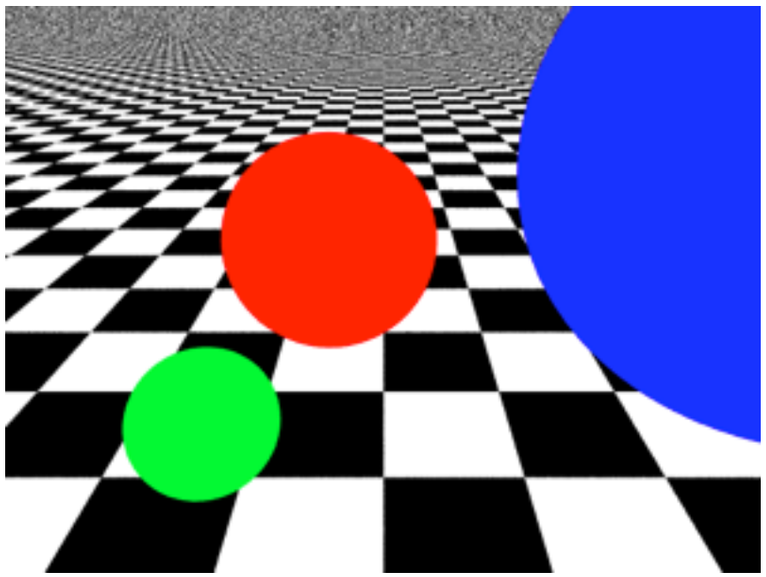


Global Illumination

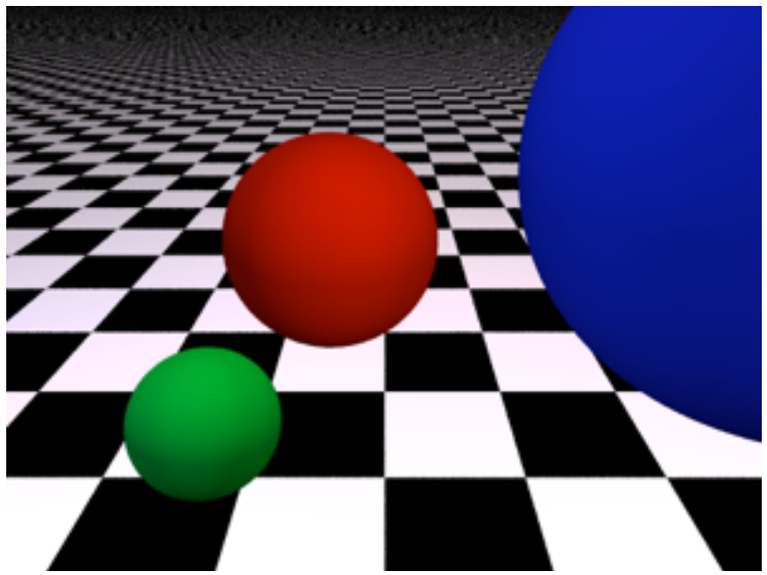


Image

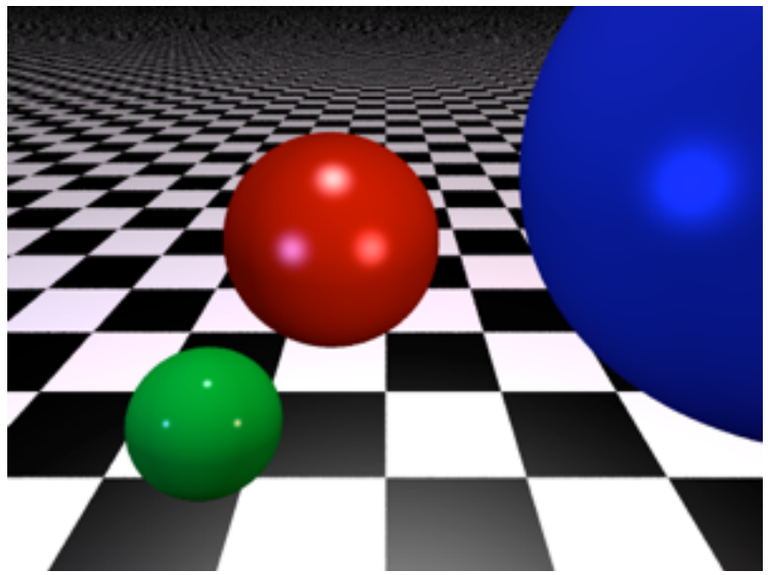
The Quest for Realism



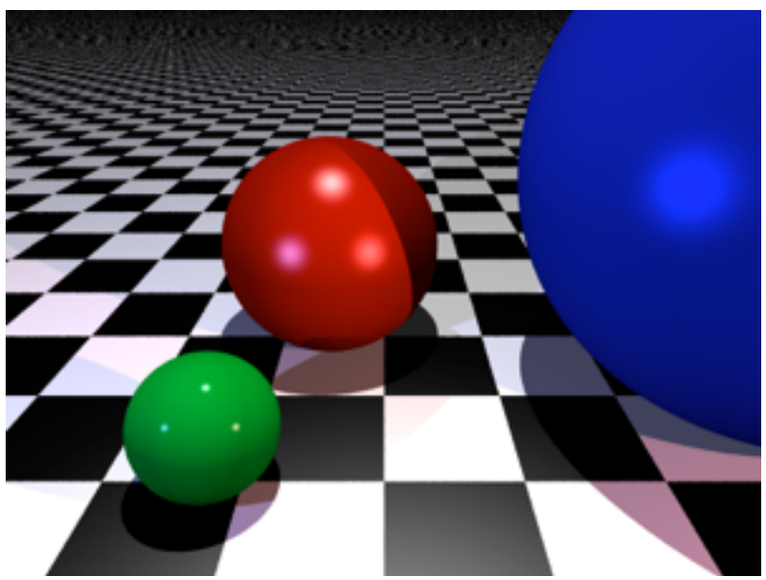
Ambient



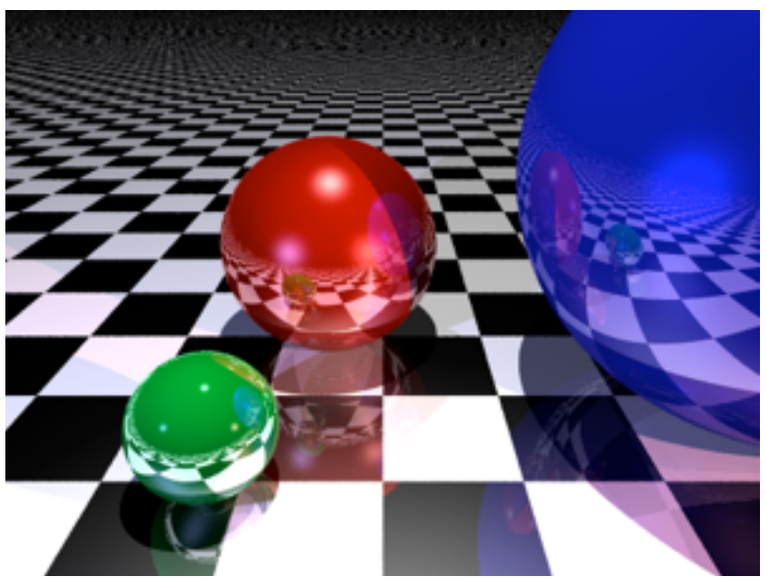
+ Diffuse



+ Specular



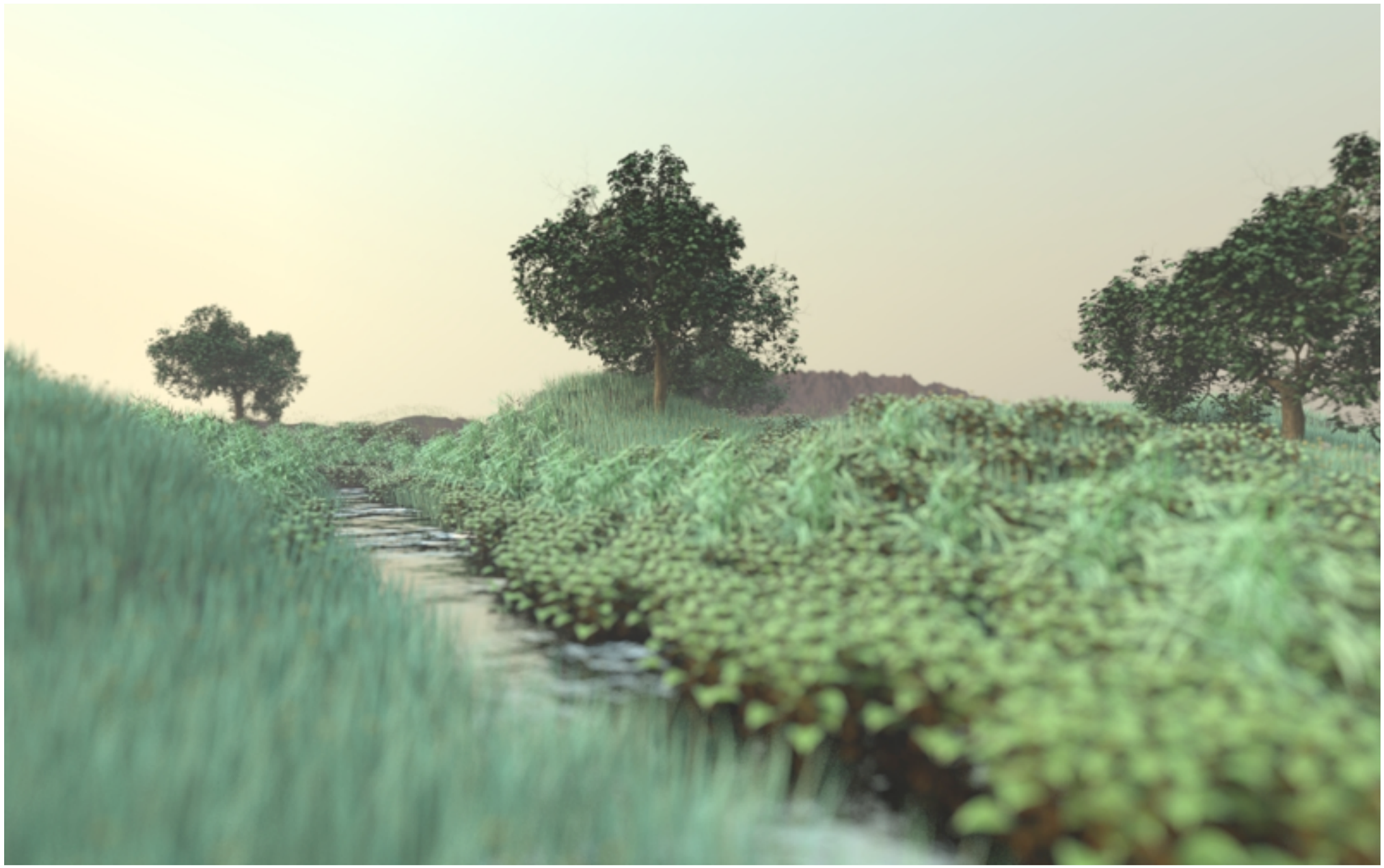
+ Shadows



+ Reflections

The Quest for Realism

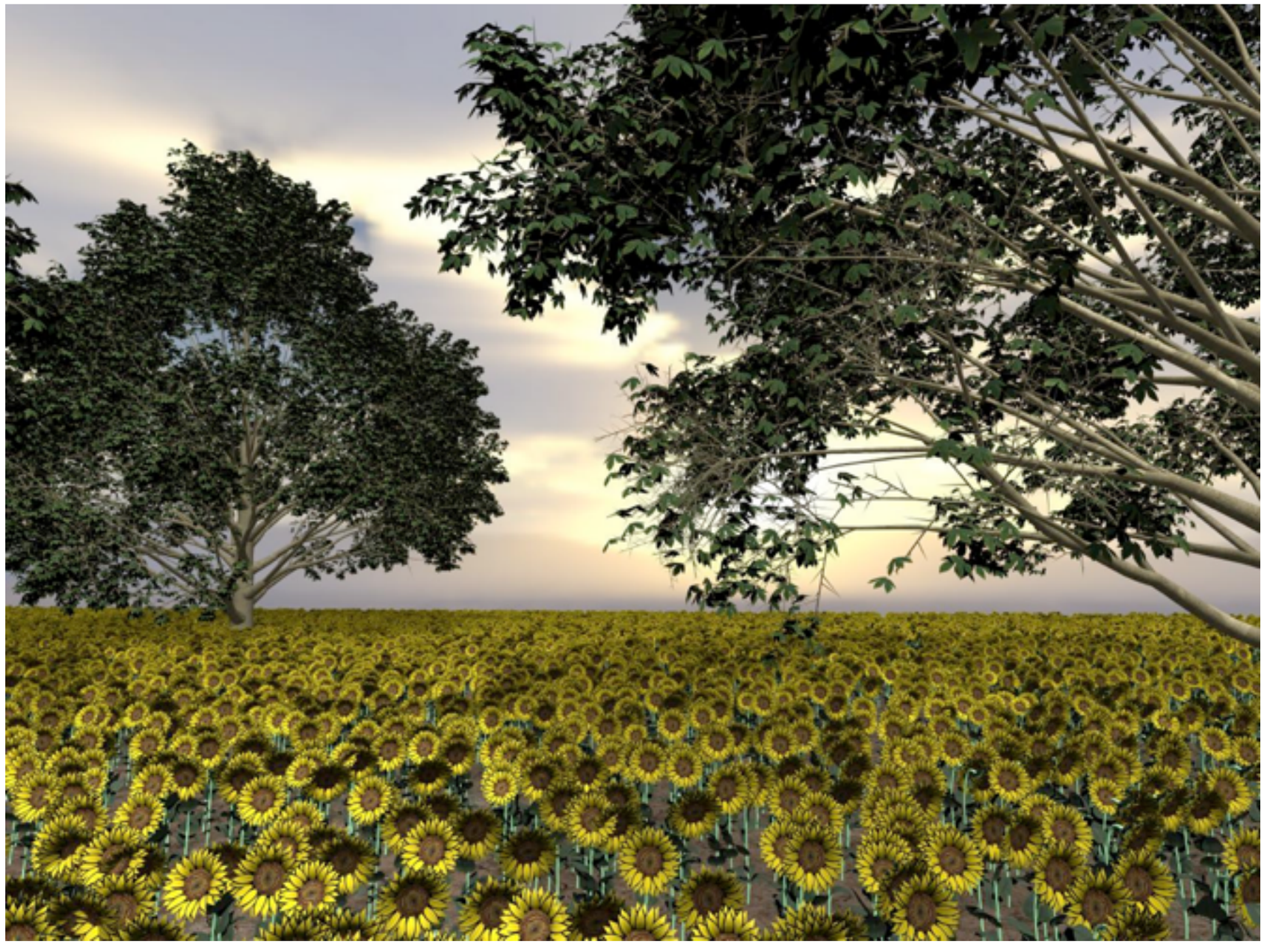
- Realism through geometric complexity



[www.pbrt.org, 20M triangles]

The Quest for Realism

- Realism through geometric complexity



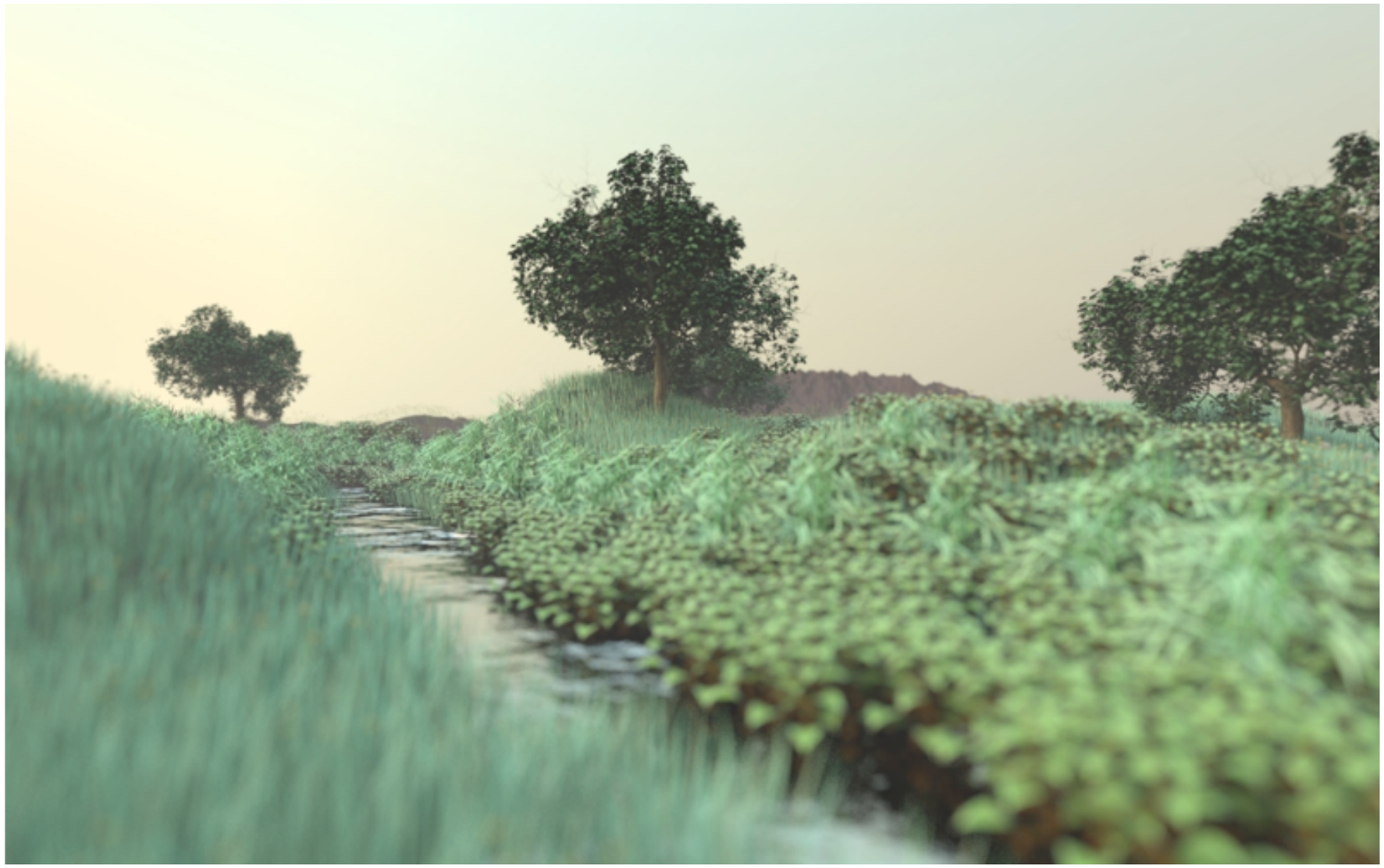
[Oliver Deussen, 1G triangles]

The Quest for Realism

- Realism through geometric complexity
- What challenges do we face when rendering complex models?
  - Memory consumption
  - Computational cost

Memory Consumption

- Geometry instancing
  - 20M triangles, but 19M are instanced versions



Computational Cost

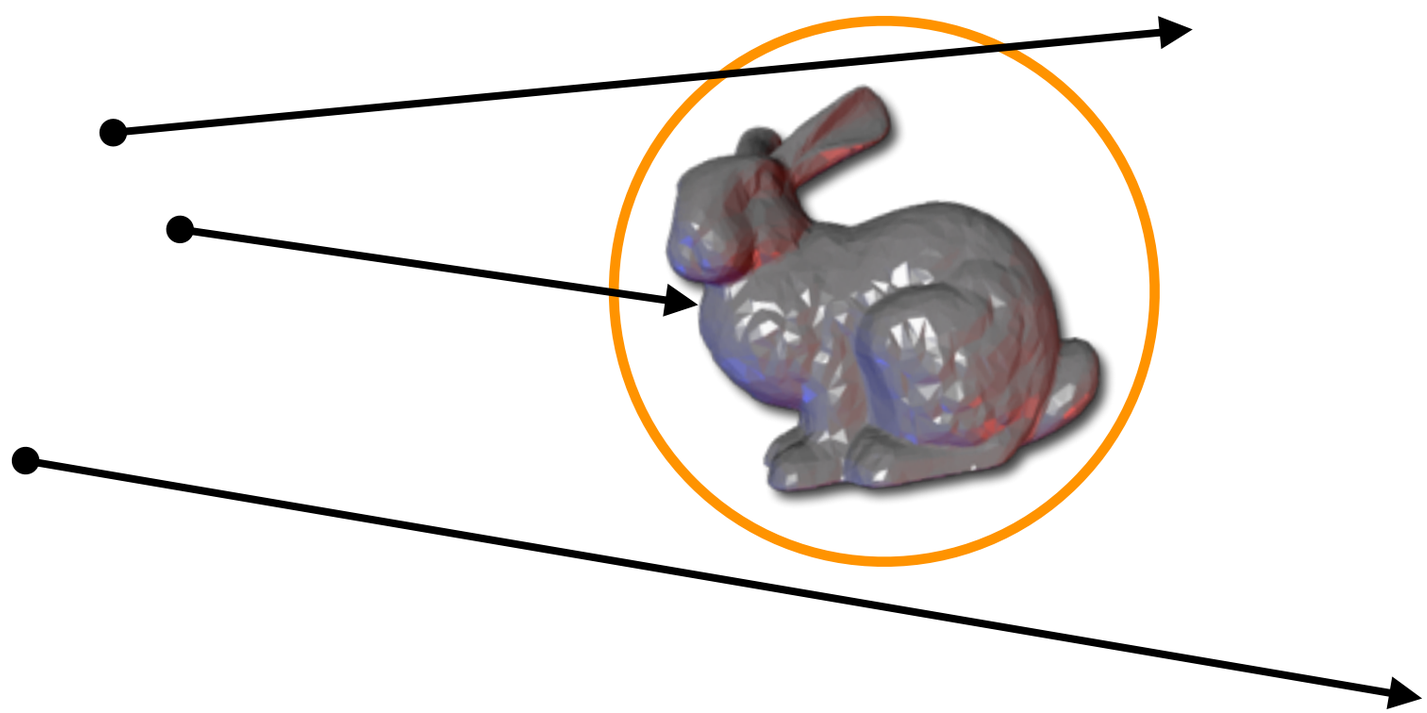
- typically >90% of computing time spent on ray-surface intersection calculations
- Brute force approach
  - Intersect every ray with every primitive: $O(w \cdot h \cdot n)$
  - Recursion leads to exponential #rays
  - Many unnecessary intersection tests

Outline

- **Bounding Volumes**
- Spatial Sorting
- Parallelization

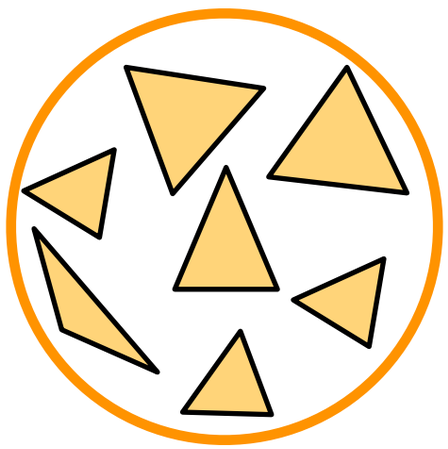
Bounding Volumes

- Preprocessing: Construct bounding volume enclosing an object.
- Fast ray-intersection rejection: If ray does not intersect bounding volume, it does not intersect the enclosed object.



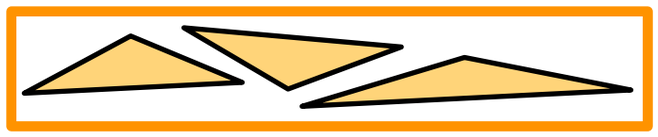
Bounding Spheres

- Construction
  - Center: barycenter of vertices or of bounding box
  - Radius: maximum distance of a vertex from center
- Intersection
  - Simple ray-sphere intersection



Axis-Aligned Bounding Box (ABB)

- Construction:
  - Simple min/max of x/y/z coordinates
- Intersection?



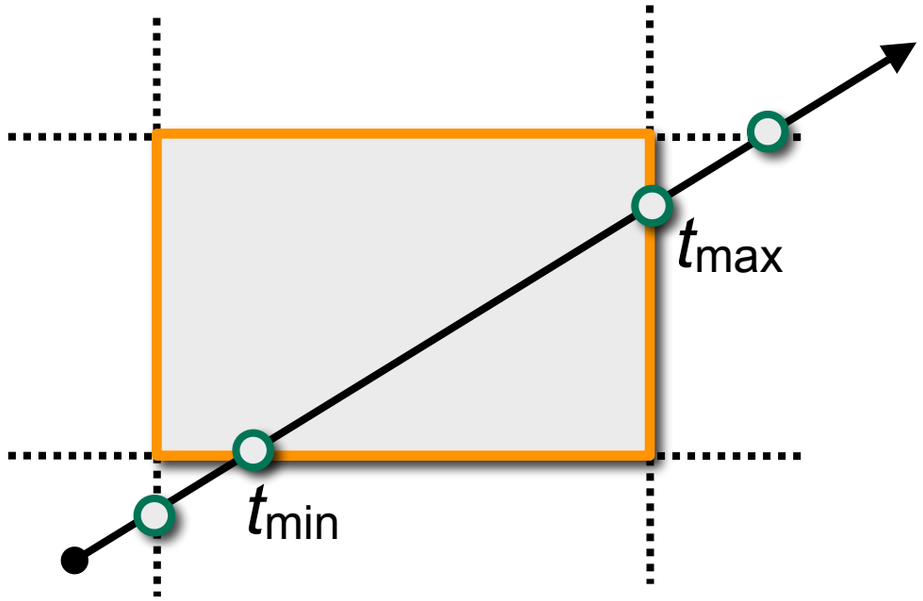
Ray-ABB Intersection

- ABB: $\mathbf{x} \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$
- Ray: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}, \;\; t \in [t_{\min}, t_{\max}], \;\; t_{\min} = 0, \; t_{\max} = \infty$
- Intersect ray with planes $x{=}x_{\min}$, $x{=}x_{\max}$
  - Gives intersection parameters $t_0 < t_1$
  - Update ray interval:

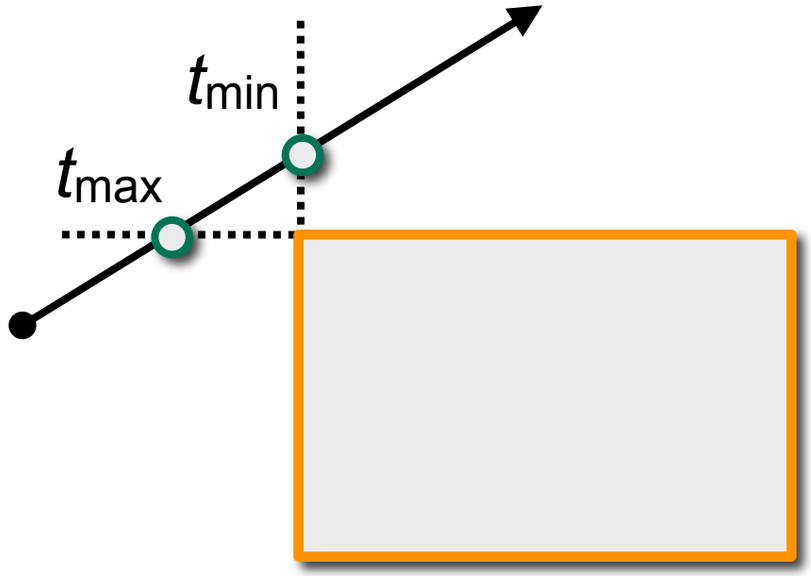
$$t_{\min} = \max\{t_{\min}, t_0\}, \; t_{\max} = \min\{t_{\max}, t_1\}$$

- Repeat for y and z planes

Ray-ABB Intersection



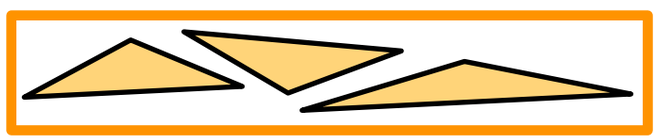
$t_{min} < t_{max} \Rightarrow$ accept

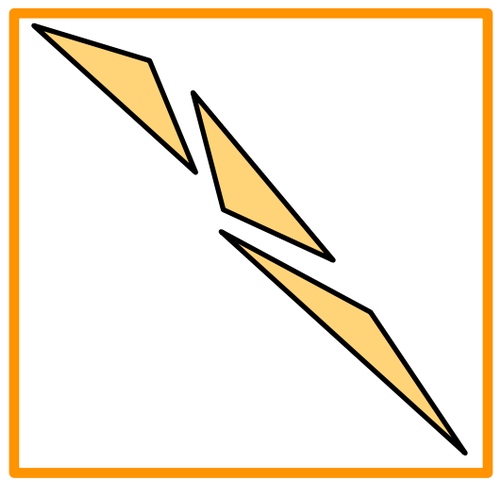


$t_{min} > t_{max} \Rightarrow$ reject

Axis-Aligned Bounding Box

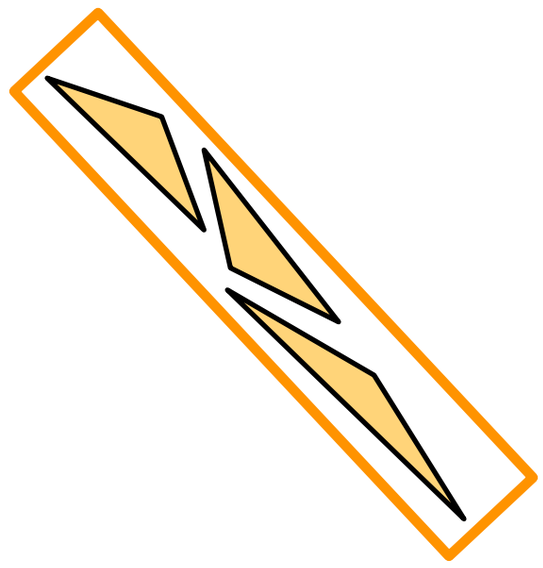
- Construction:
  - Simple min/max of x/y/z coordinates
- Intersection:
  - Intersect with x/y/z slabs of planes





Object-Aligned Bounding Box (OBB)

- Construction:
  - Center: Barycenter of vertices
  - Axes: Principal component analysis of vertices
- Intersection
  - Intersect with three (non-aligned) slabs



Bounding Volumes

- Bounding Volumes
  - Spheres
  - Axis-aligned bounding box
  - Oriented bounding box

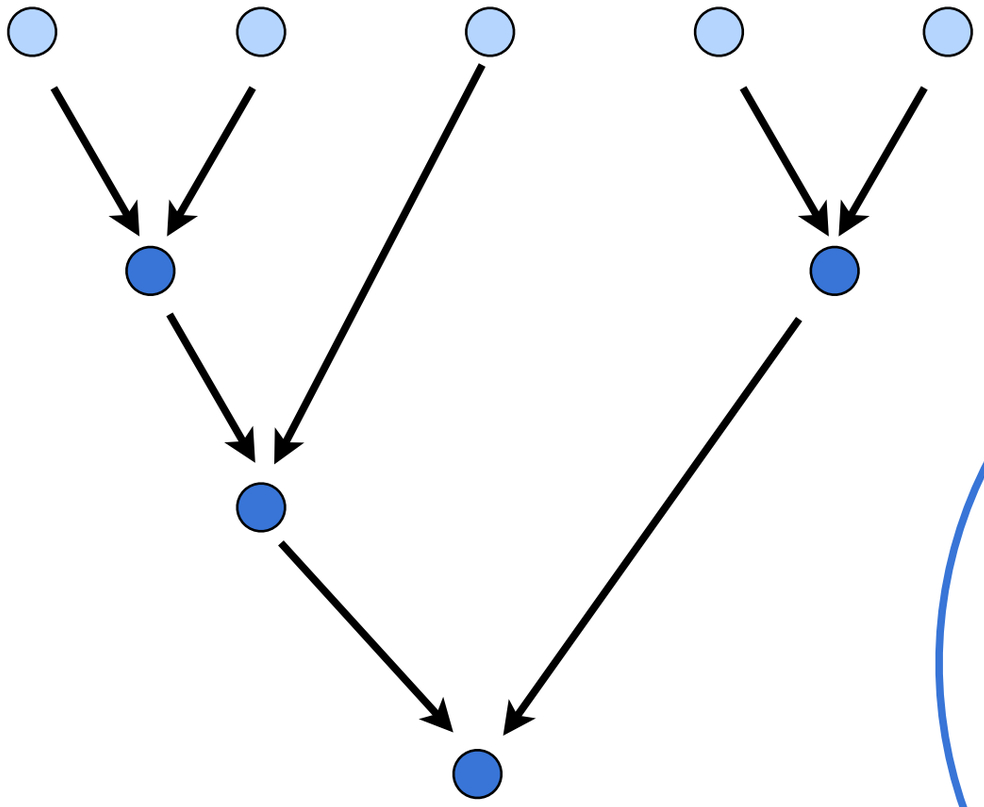
- Tradeoff:
  - complex BV shape
    - expensive intersections, fewer “false positives”
  - simple BV shapes
    - simple intersectinos, more “false positives”

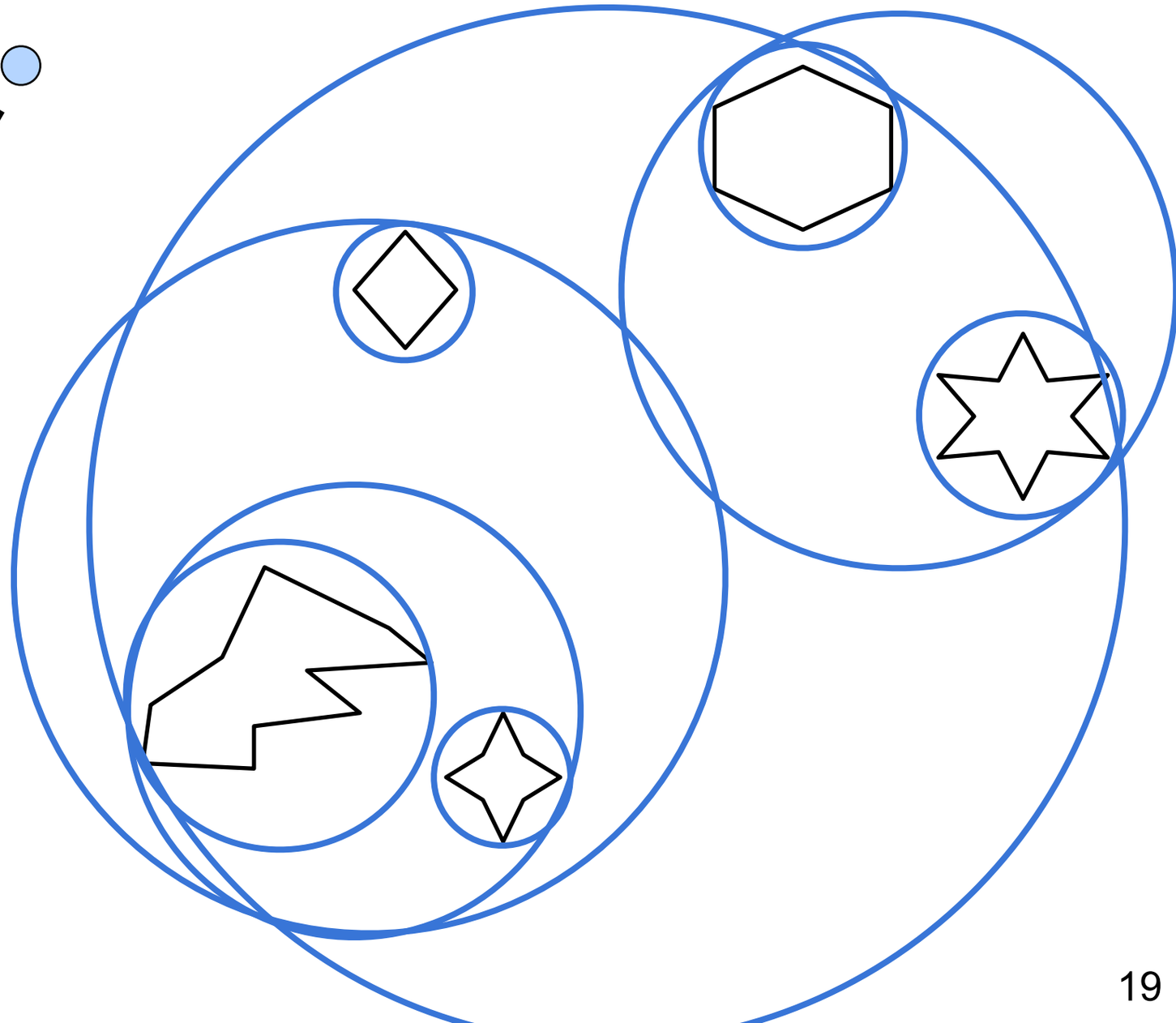
Exercise 1

- With ABBs: 11s
- Without ABBs: 335s

Bounding Volume Hierarchies

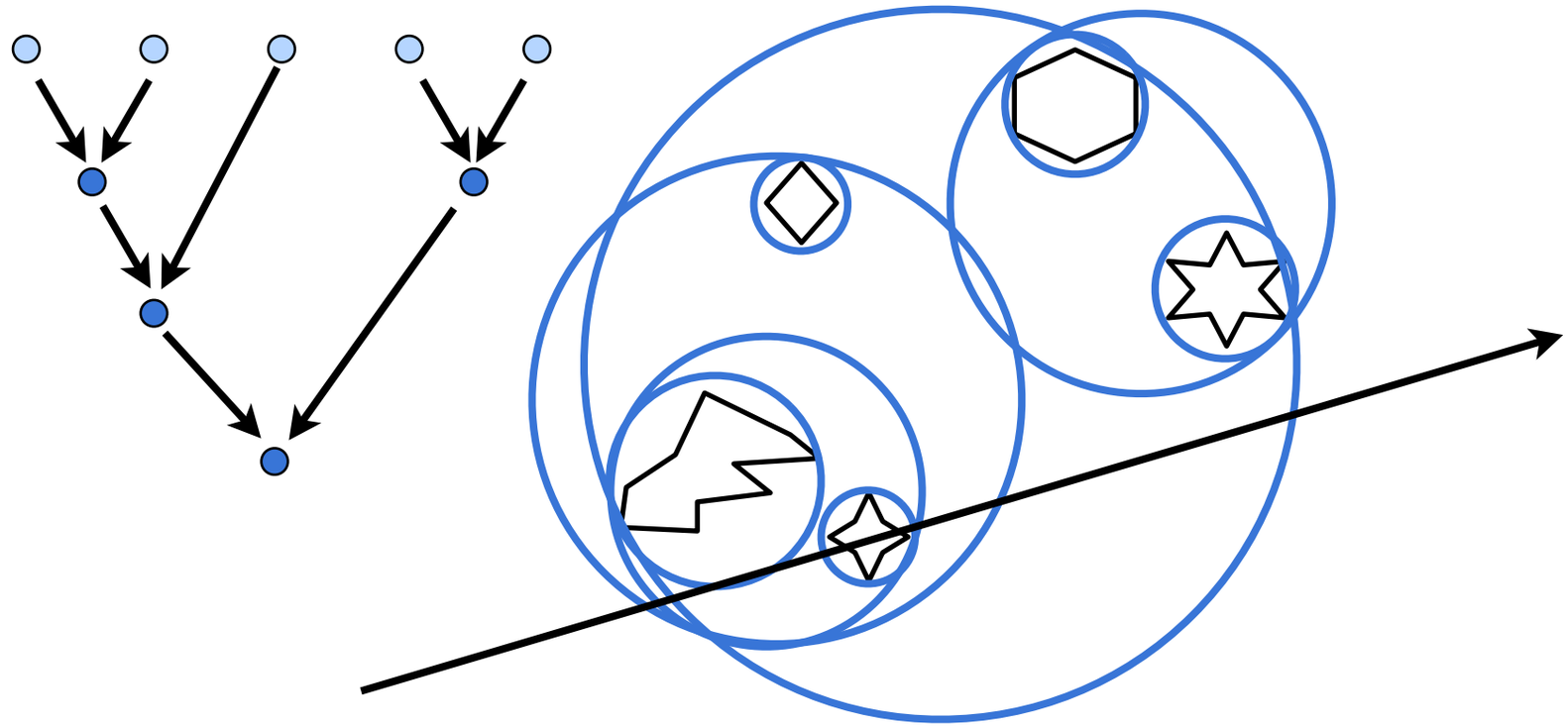
- Construction: bottom-up





Bounding Volume Hierarchies

- Ray intersection
  - Traverse top-down recursively
  - Each geometric primitive intersected only once



Bounding Volume Hierarchies

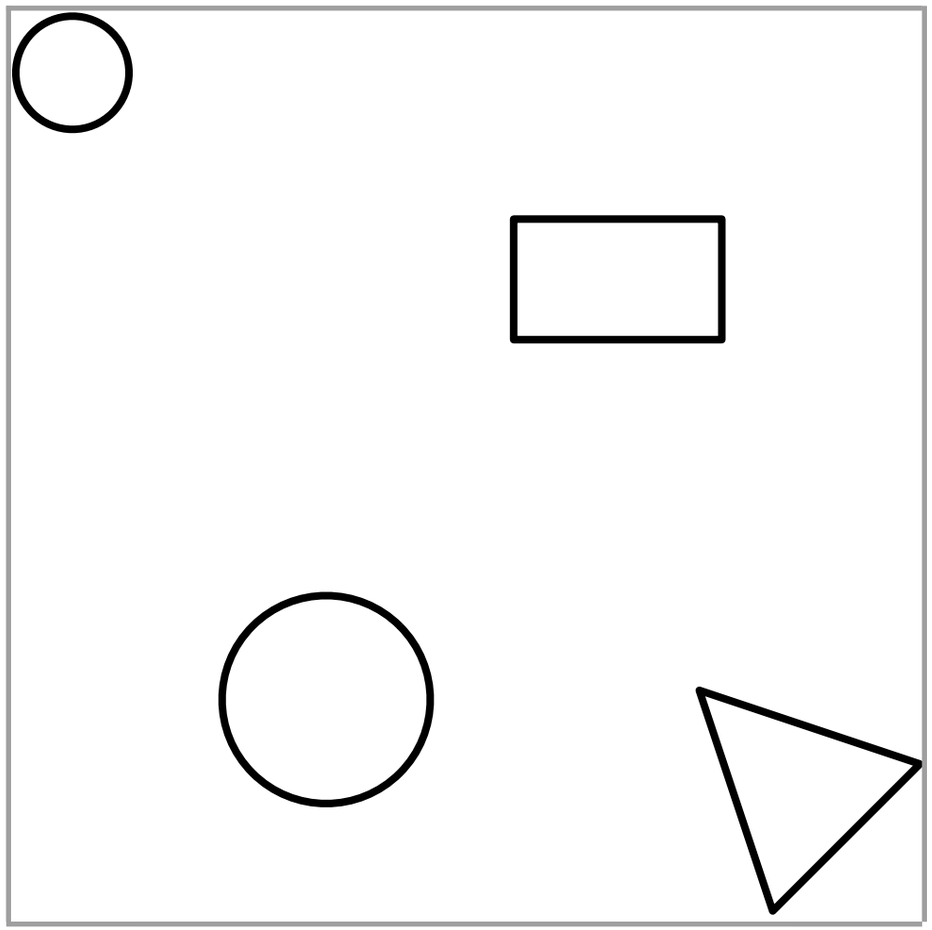
- Idea
  - Organize BVs hierarchically into new BVs
- Advantages
  - Good adaptivity
  - Reduces cost from O(*n*) to O(log*n*)
- Problems:
  - How to arrange BVs?

Outline

- Bounding Volumes
- **Spatial Sorting**
- Parallelization

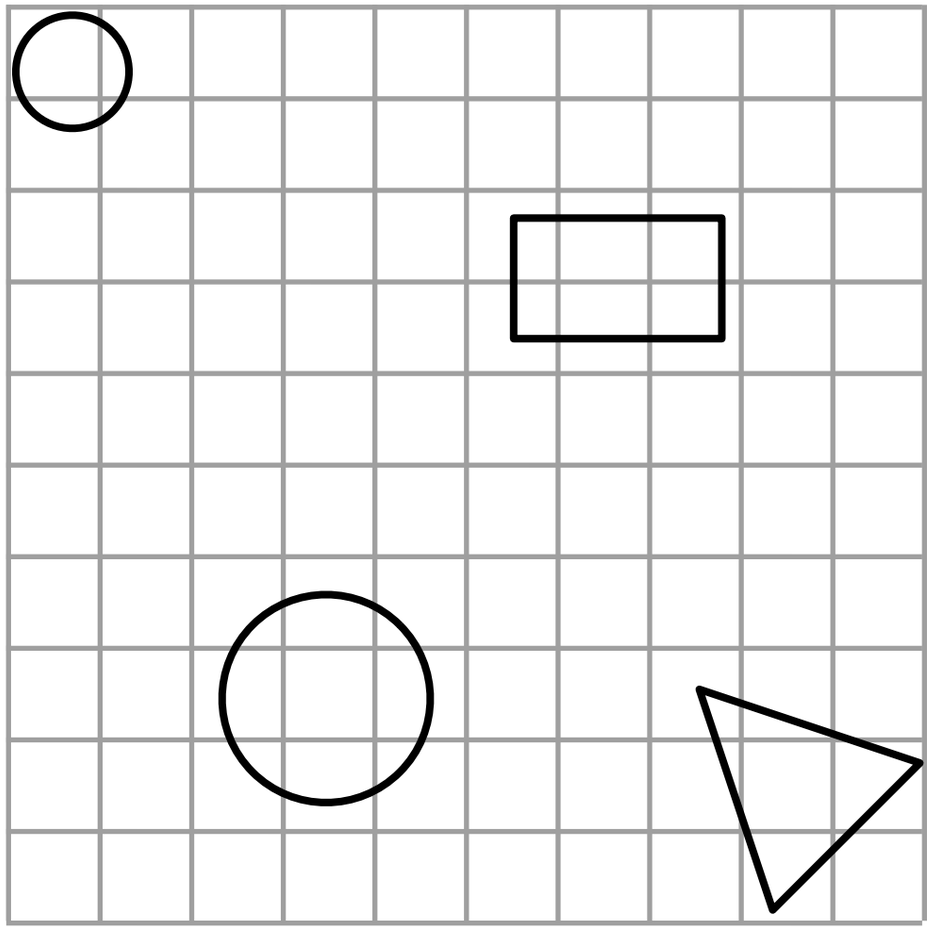
Uniform Grids

- Construction
  - compute bounding box



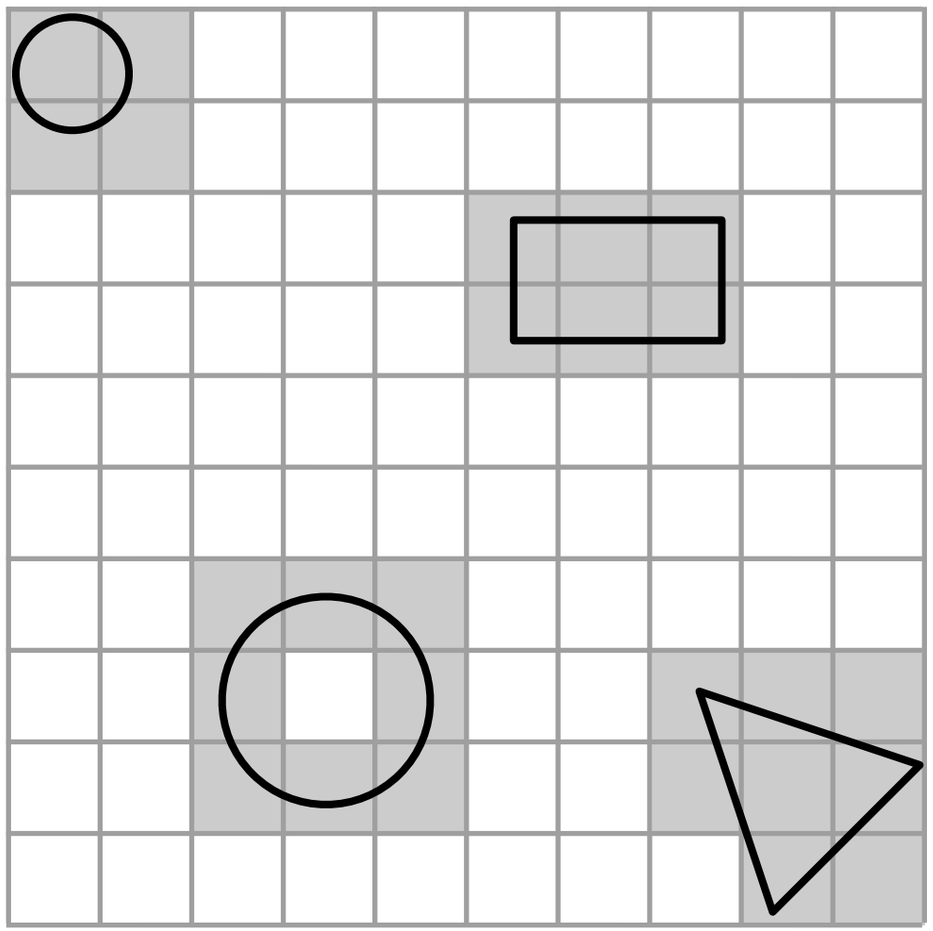
Uniform Grids

- Construction
  - compute bounding box
  - determine grid resolution (often $\sim \sqrt[3]{n}$)



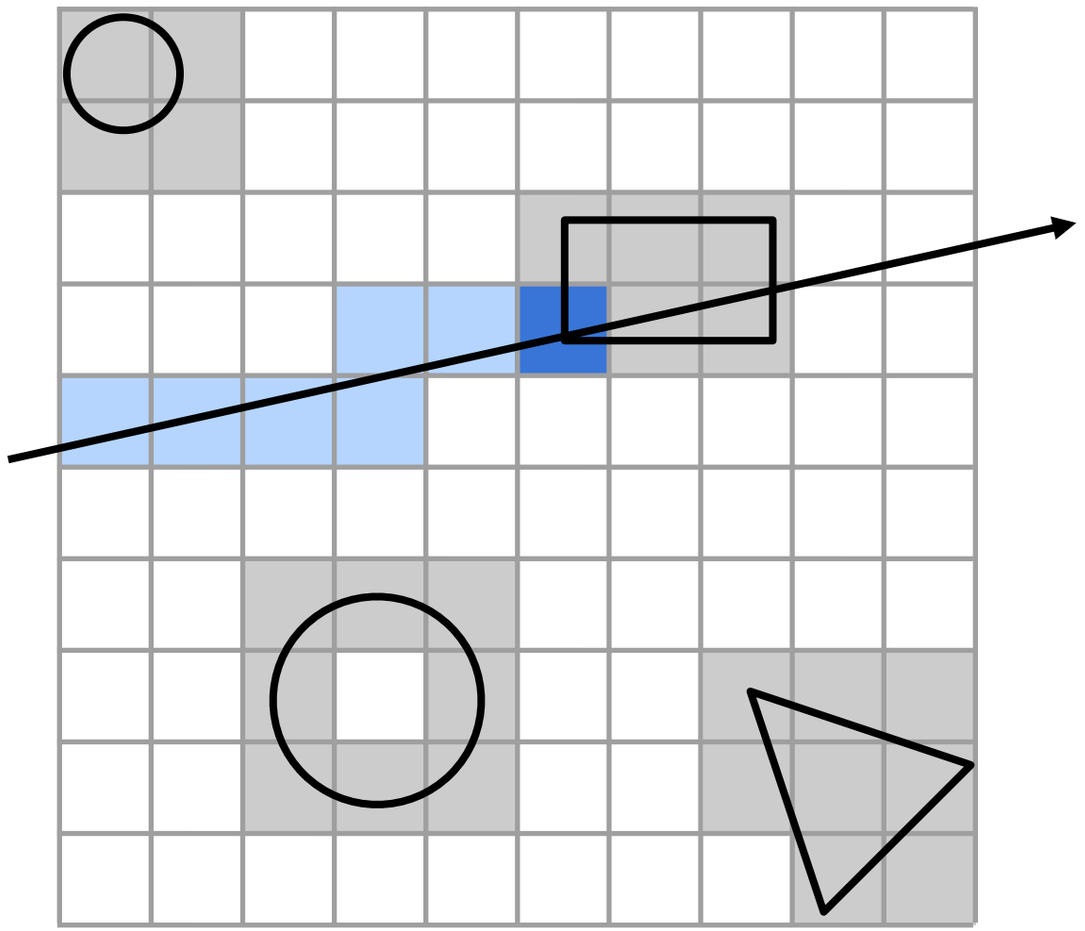
Uniform Grids

- Construction
  - compute bounding box
  - determine grid resolution (often $\sim \sqrt[3]{n}$)
  - insert objects into cells
    - rasterize bounding box
    - prune empty cells
    - store reference for each object in cell



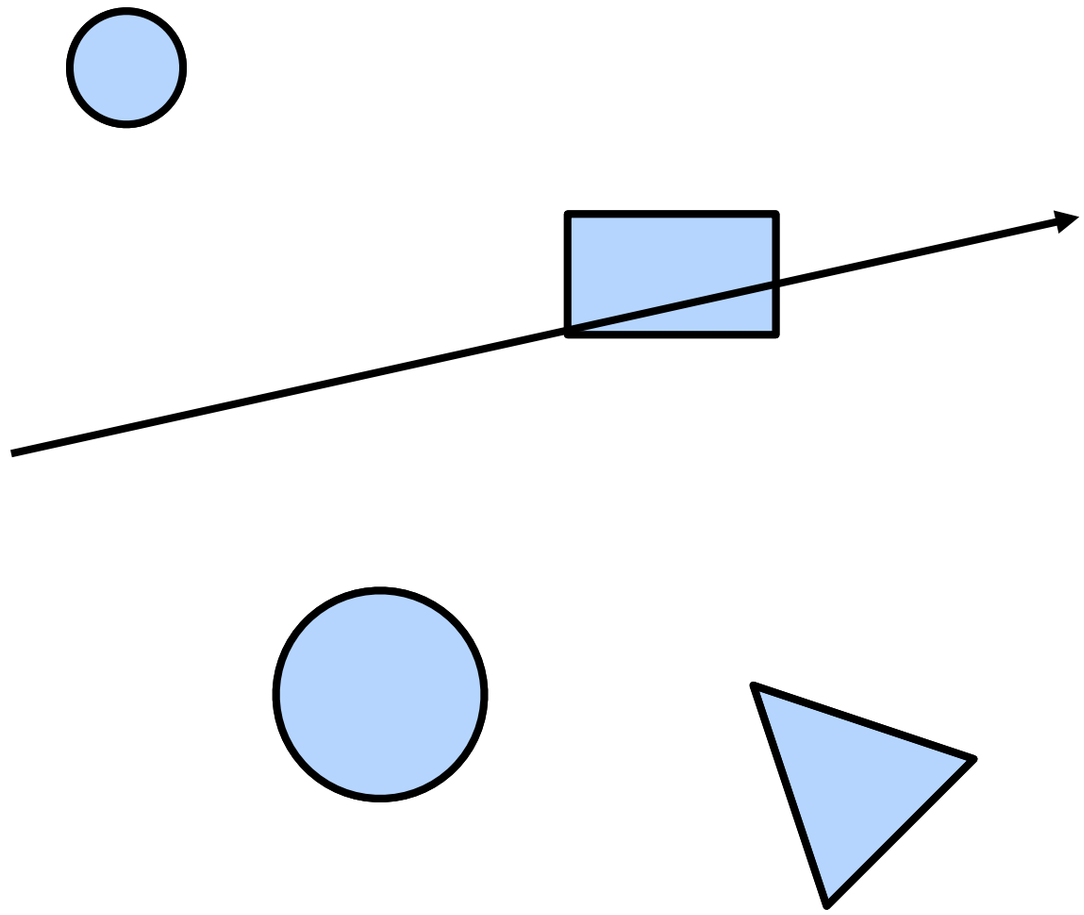
Uniform Grids

- Intersection
  - incrementally walk along ray through grid
  - compute intersection with objects in each cell
  - stop when intersection found in current voxel



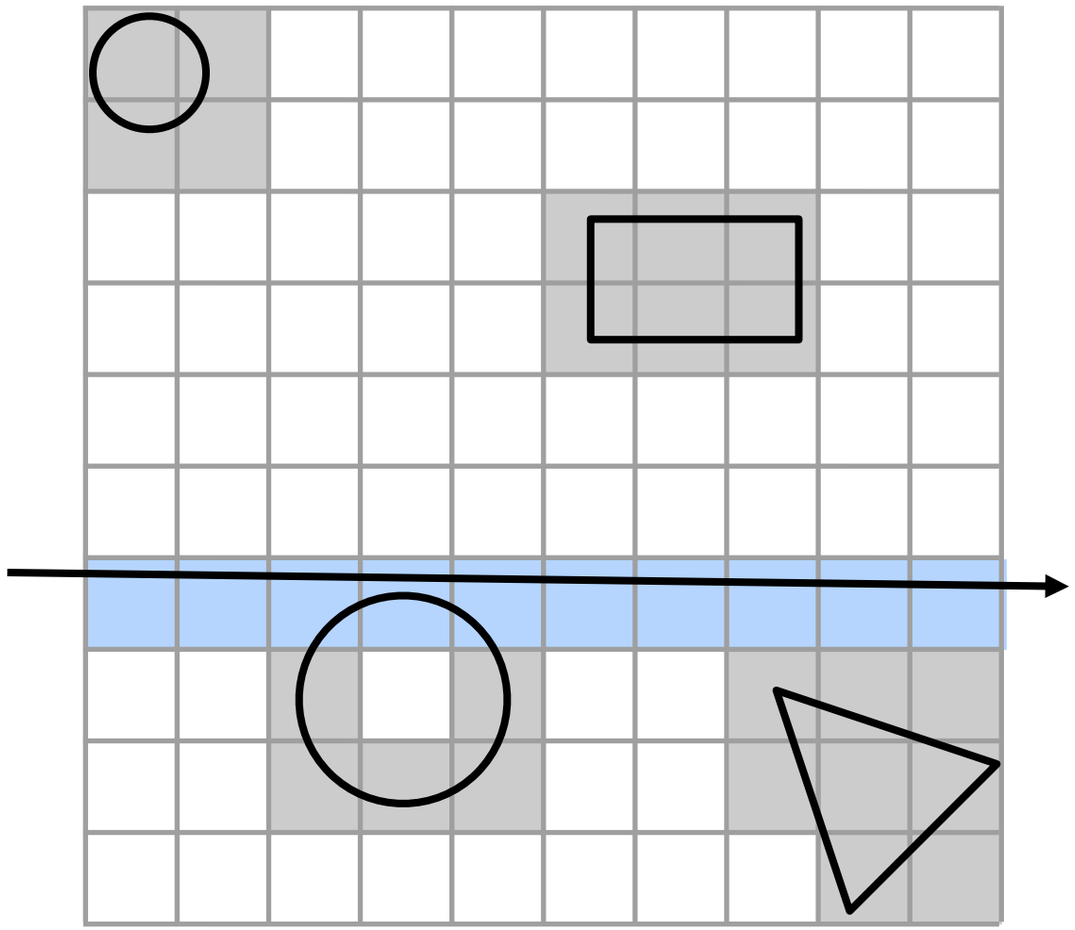
Uniform Grids

- Comparison: brute-force
  - intersect ray with every primitive
  - take closest intersection



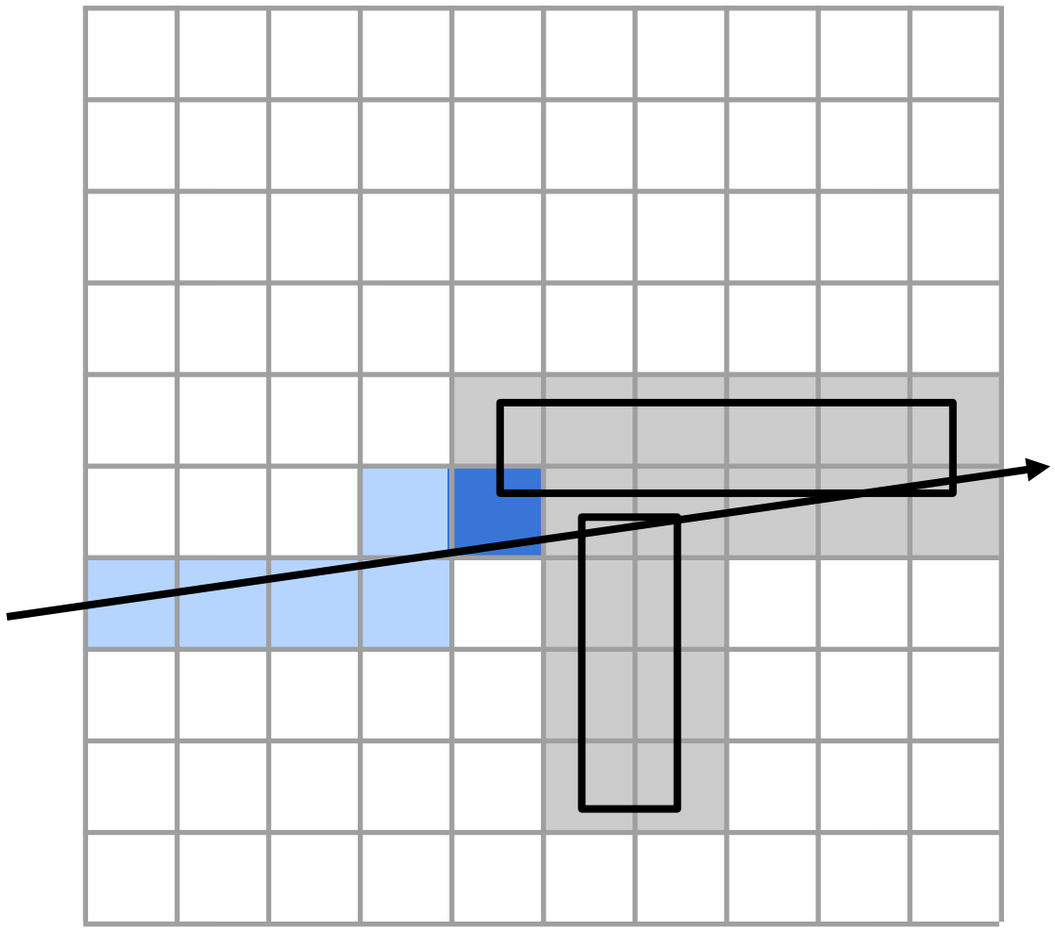
Uniform Grids

- Intersection
  - incrementally walk along ray through grid
  - compute intersection with objects in each cell
  - stop when intersection found in current voxel
  - use “mailboxing” to avoid multiple intersection tests



Uniform Grids

- Intersection
  - incrementally walk along ray through grid
  - compute intersection with objects in each cell
  - stop when intersection found in current voxel
  - use “mailboxing” to avoid multiple intersection tests
  - only intersection points within current grid cell

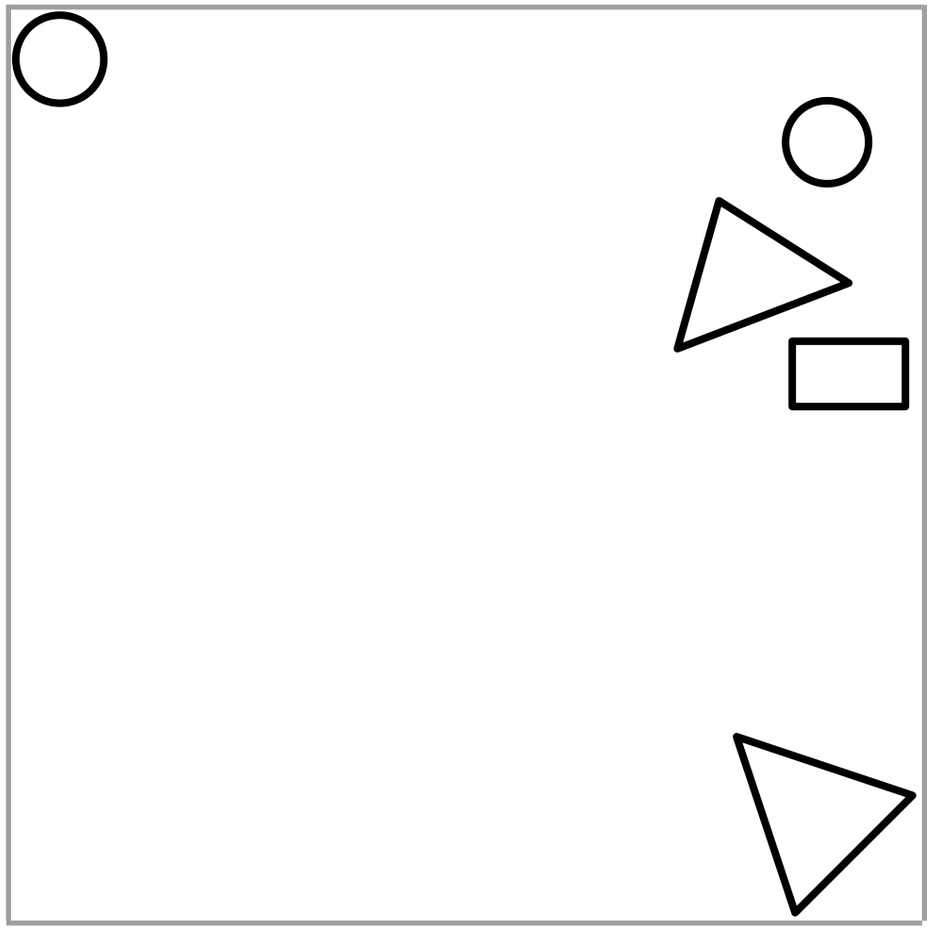


Uniform Grids

- Advantages
  - Simple and efficient construction
  - Simple and efficient traversal
- Problems
  - How to choose resolution?
  - Memory consumption
  - **Cannot adapt to local object density**

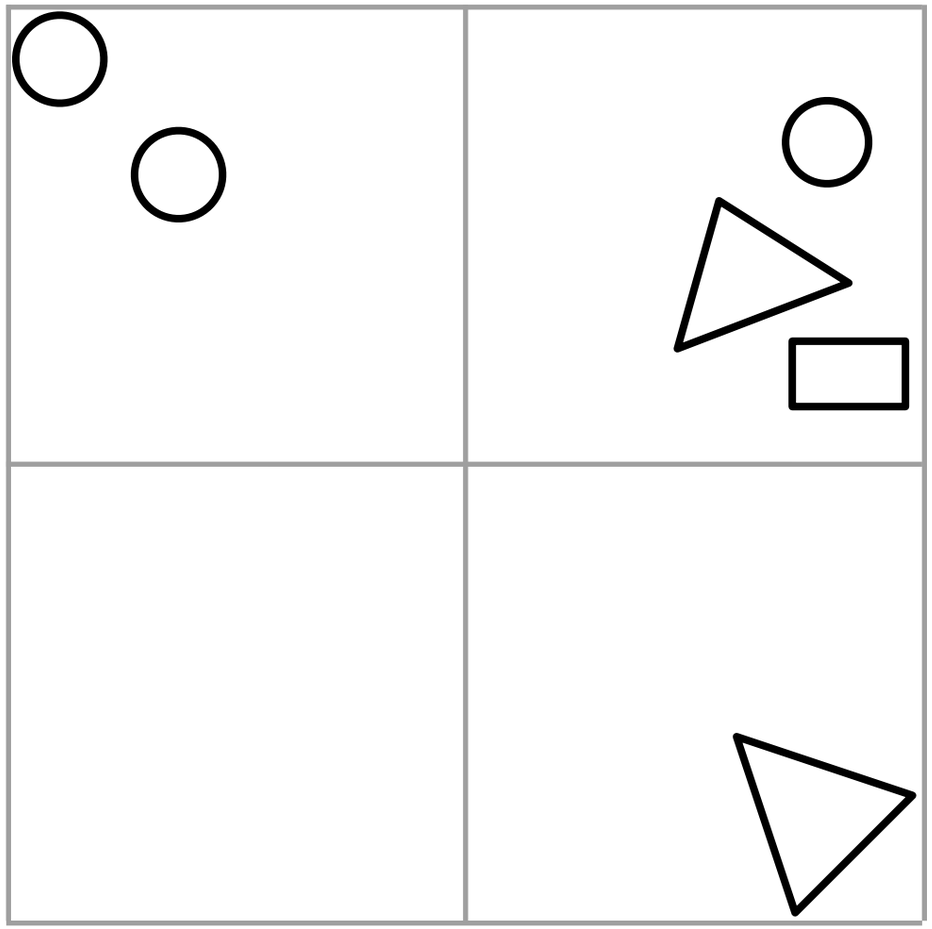
Octrees

- Construction
  - compute bounding box



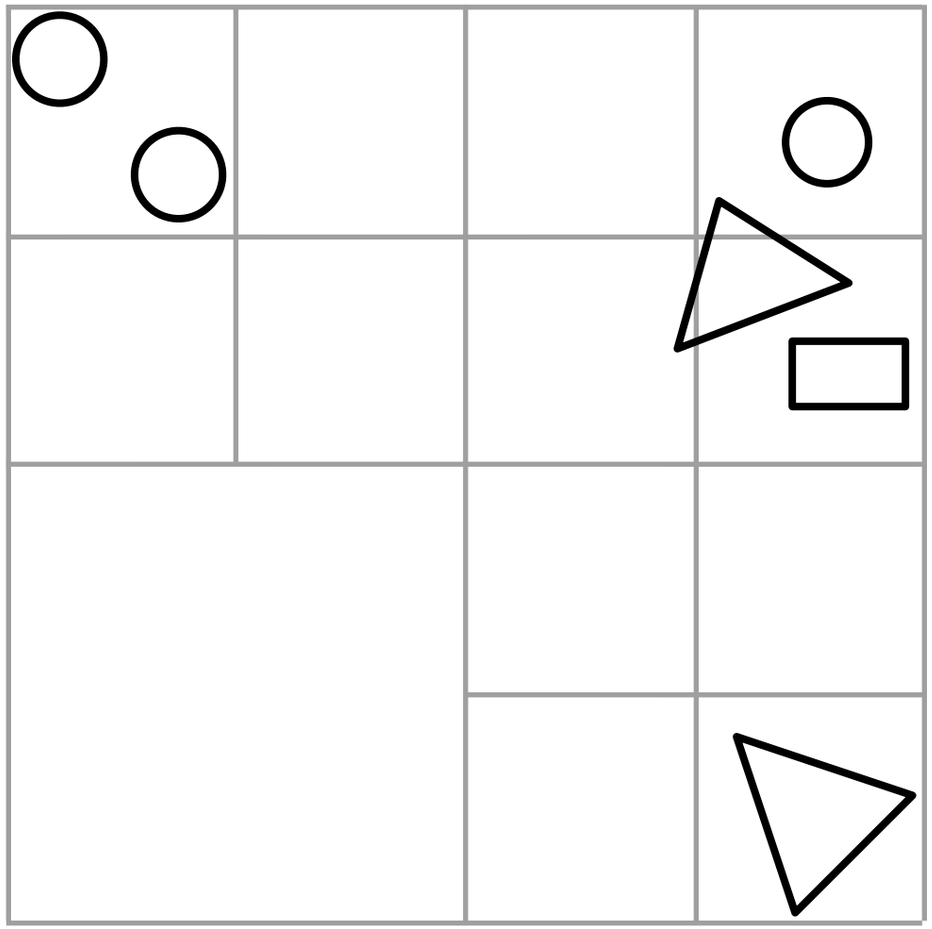
Octrees

- Construction
  - compute bounding box
  - recursively subdivide cells into 8 equal sub-cells



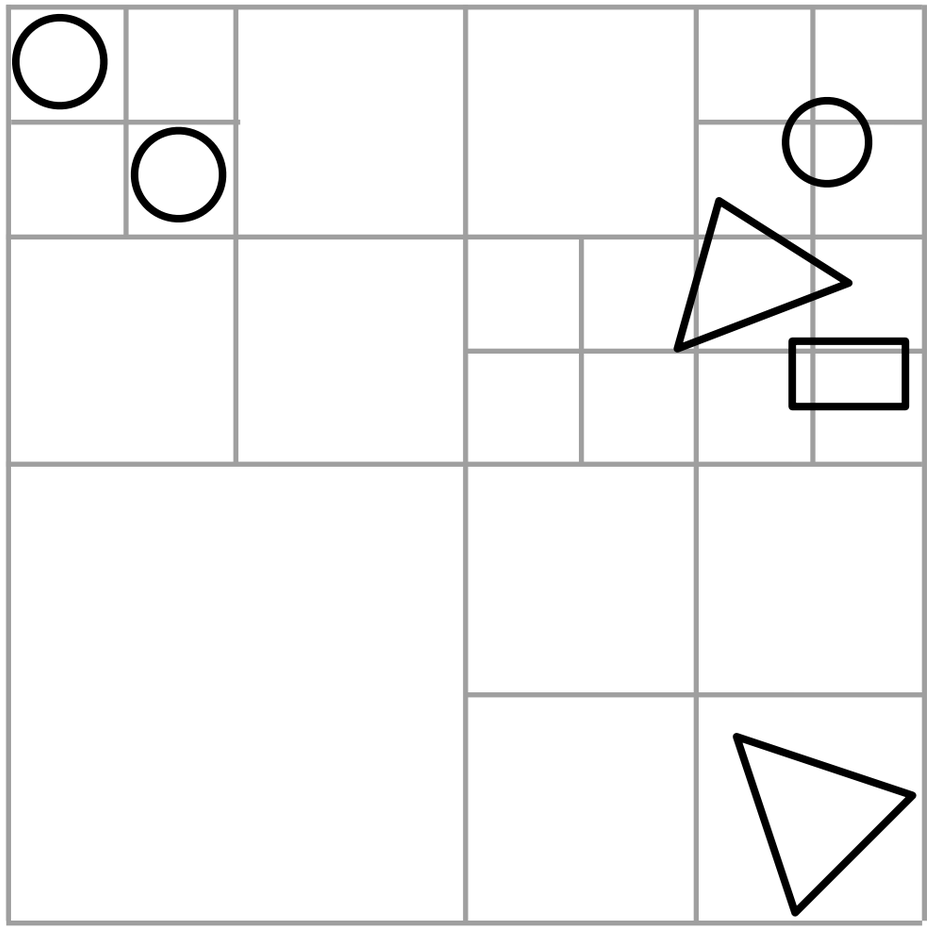
Octrees

- Construction
  - compute bounding box
  - recursively subdivide cells into 8 equal sub-cells



Octrees

- Construction
  - compute bounding box
  - recursively subdivide cells into 8 equal sub-cells
  - until maximum depth or minimum number of remaining objects is reached

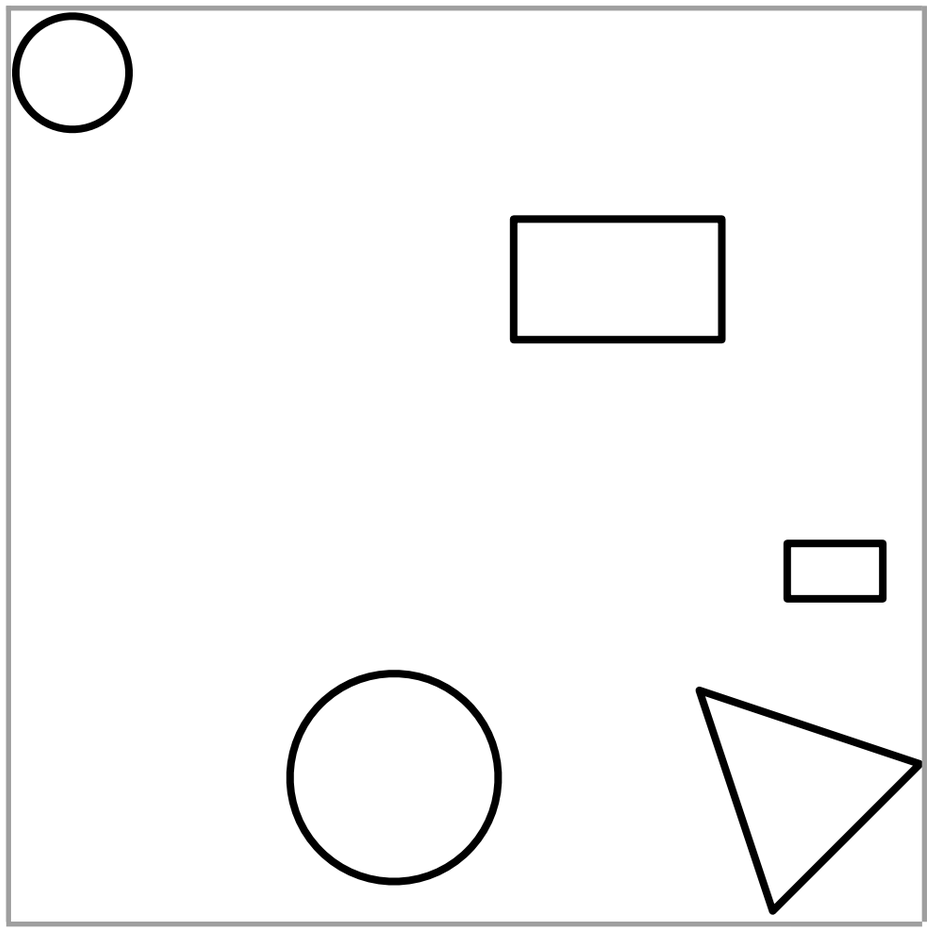


Octrees

- Advantages
  - Can adapt to local object density/distribution
  - Still quite simple to construct & intersect
- Problems
  - Fixed intersection pattern creates too many cells

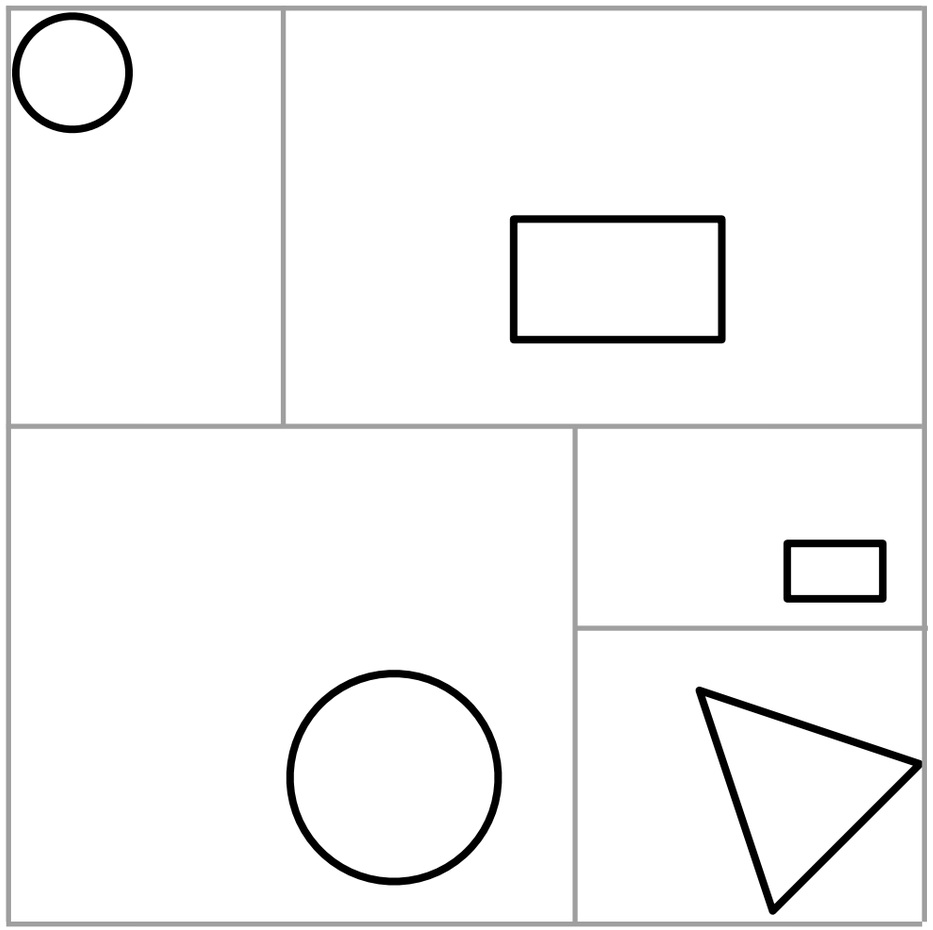
kD-Trees

- Construction
  - compute bounding box



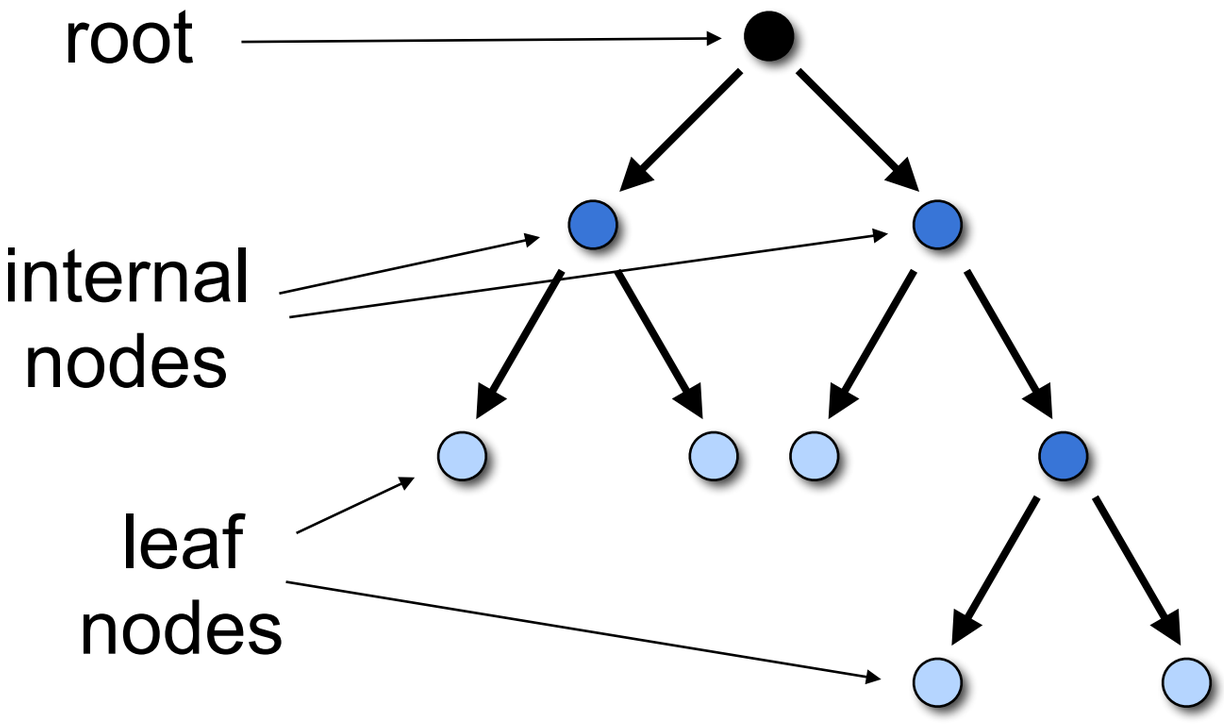
kD-Trees

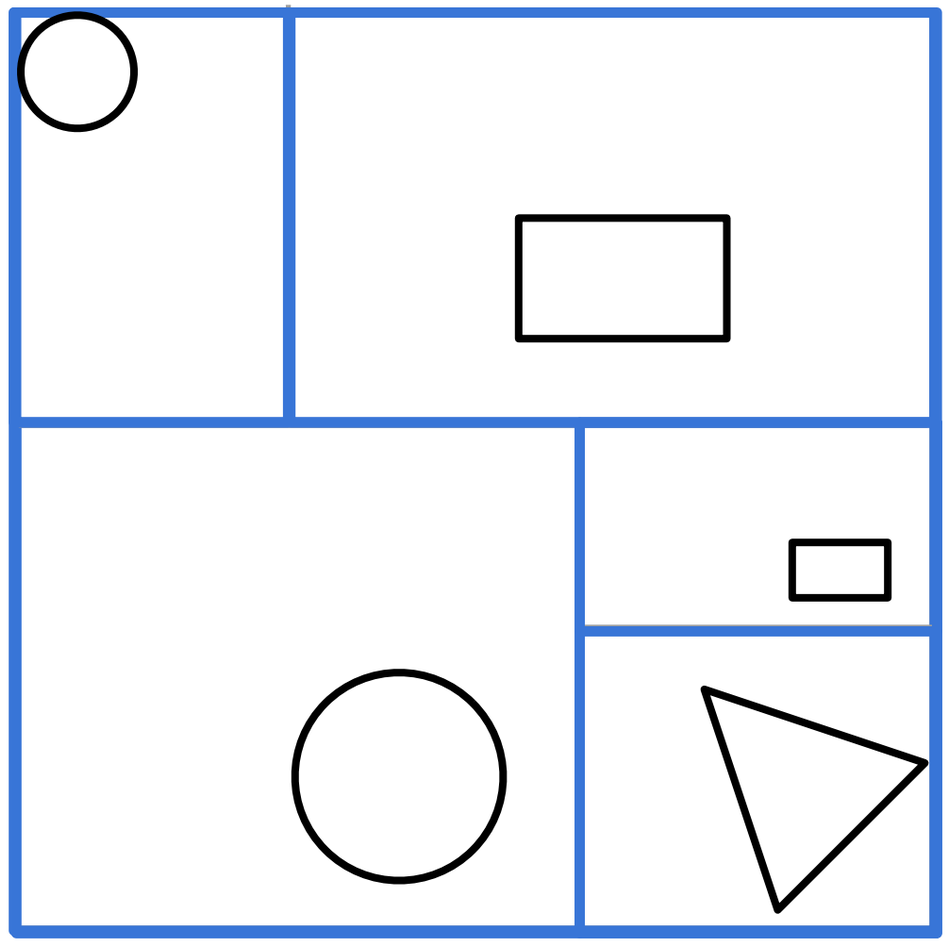
- Construction
  - compute bounding box
  - recursively split cell using axis-aligned plane
  - until maximum depth or minimum number of remaining objects is reached



kD-Trees

- Construction
  - binary tree structure





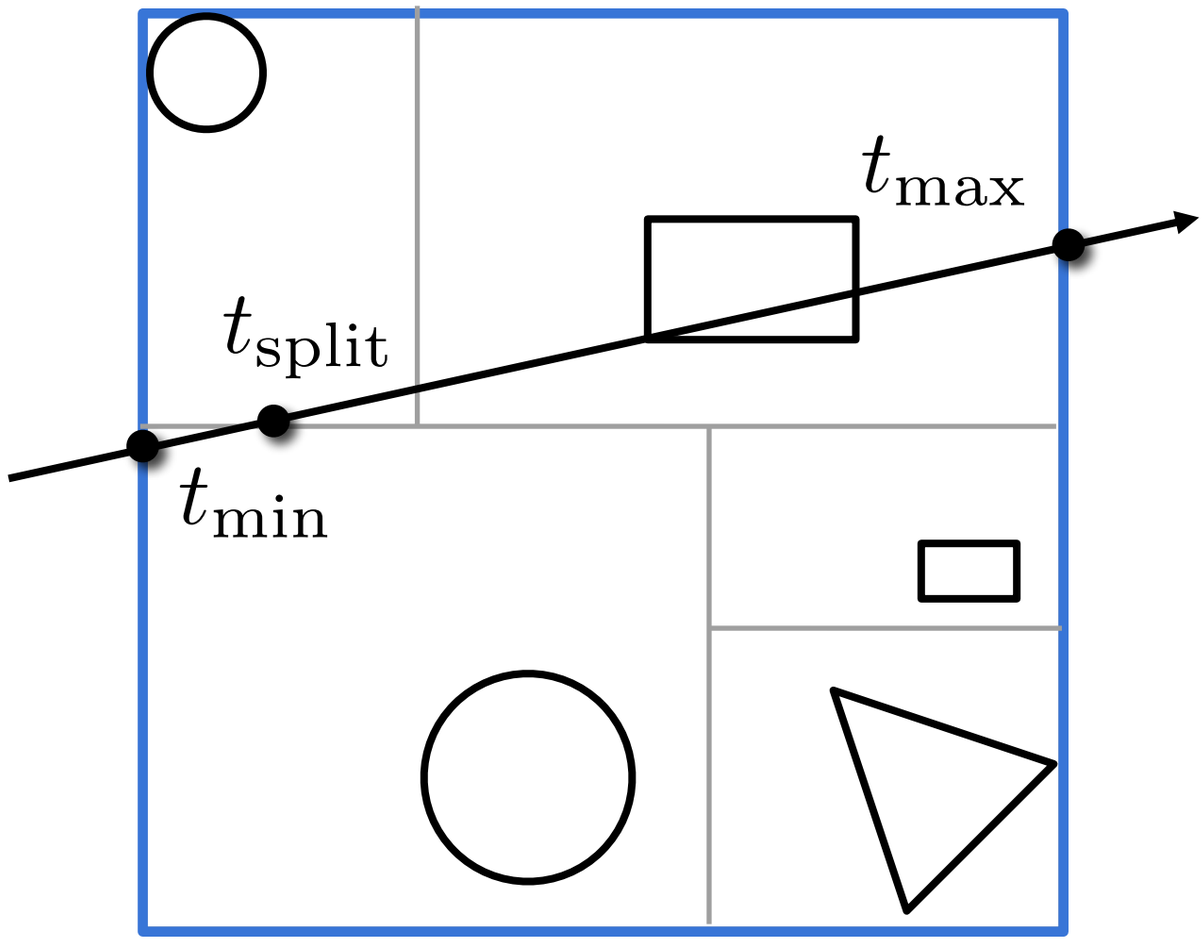
only leaf nodes store reference to geometry!

kD-Trees

- Internal nodes store
  - split axis: x-, y-, or z-axis
  - split position: coordinate of split plane along axis
  - children: reference to child nodes
- Leaf nodes store
  - list of primitives
  - mailboxing information

kD-Trees

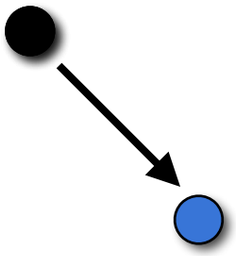
- Intersection
  - top-down recursion

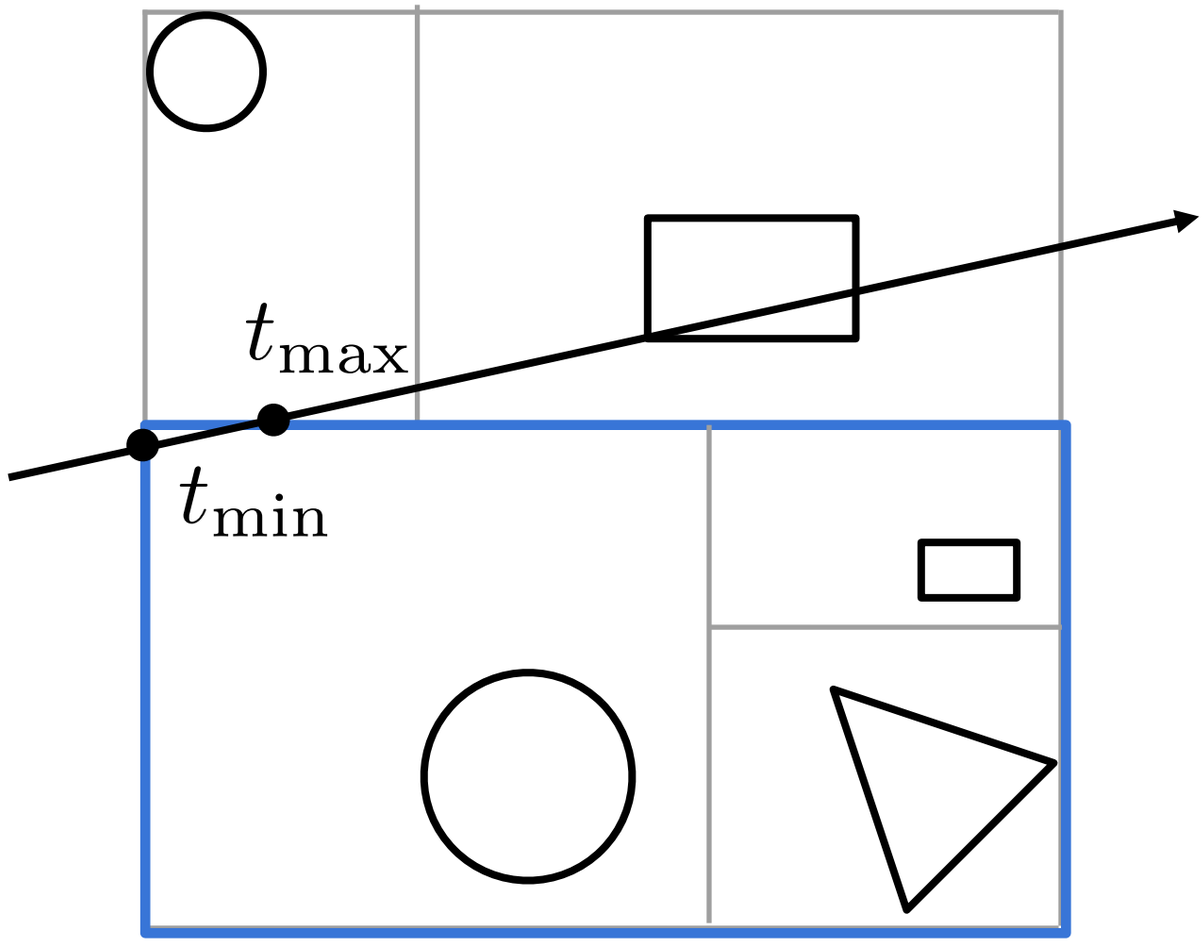


internal node → split

kD-Trees

- Intersection
  - top-down recursion

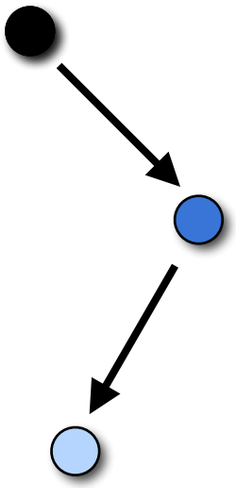


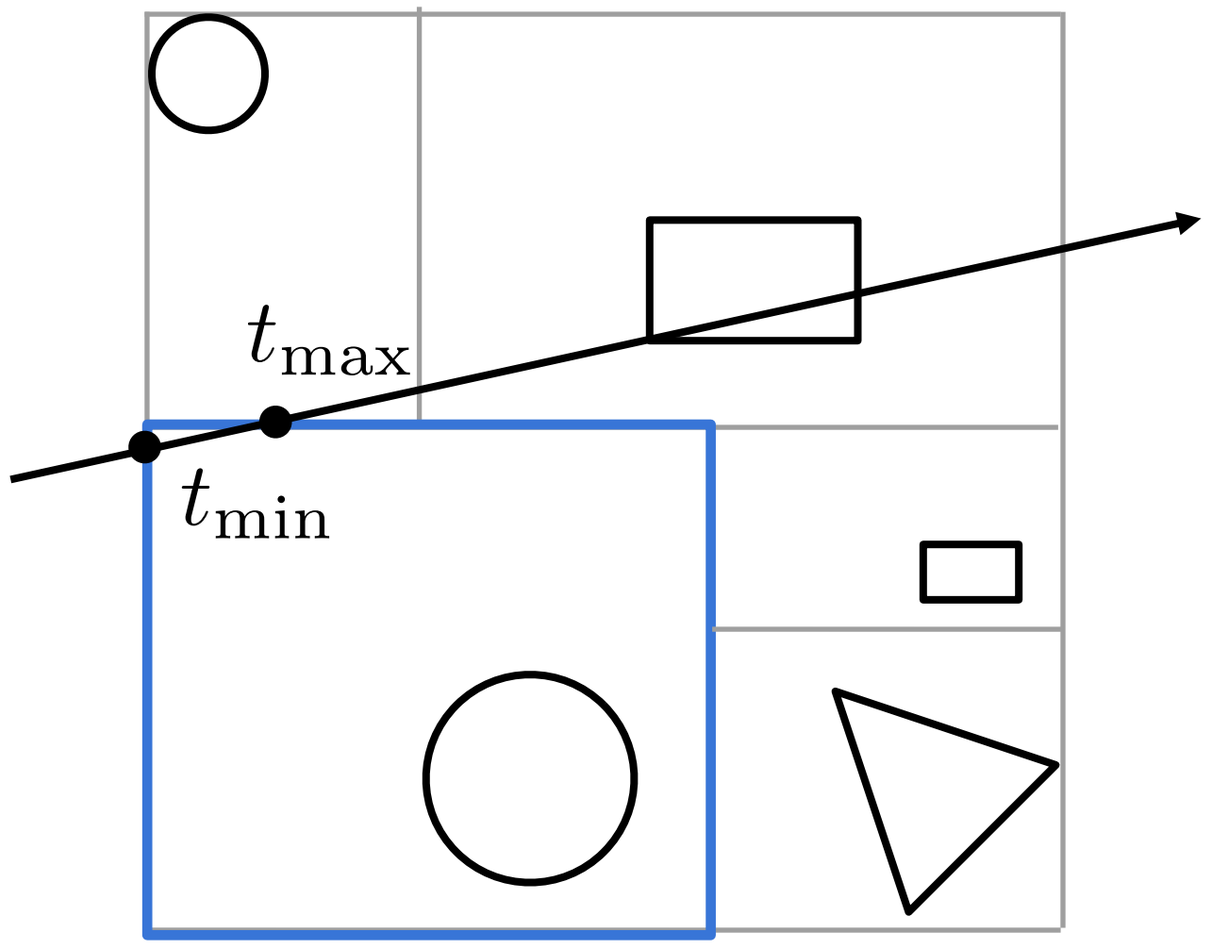


internal node → split

kD-Trees

- Intersection
  - top-down recursion

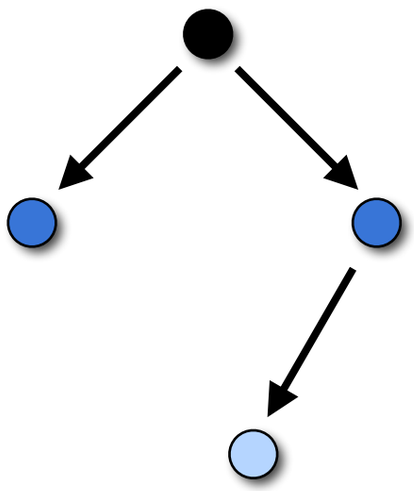


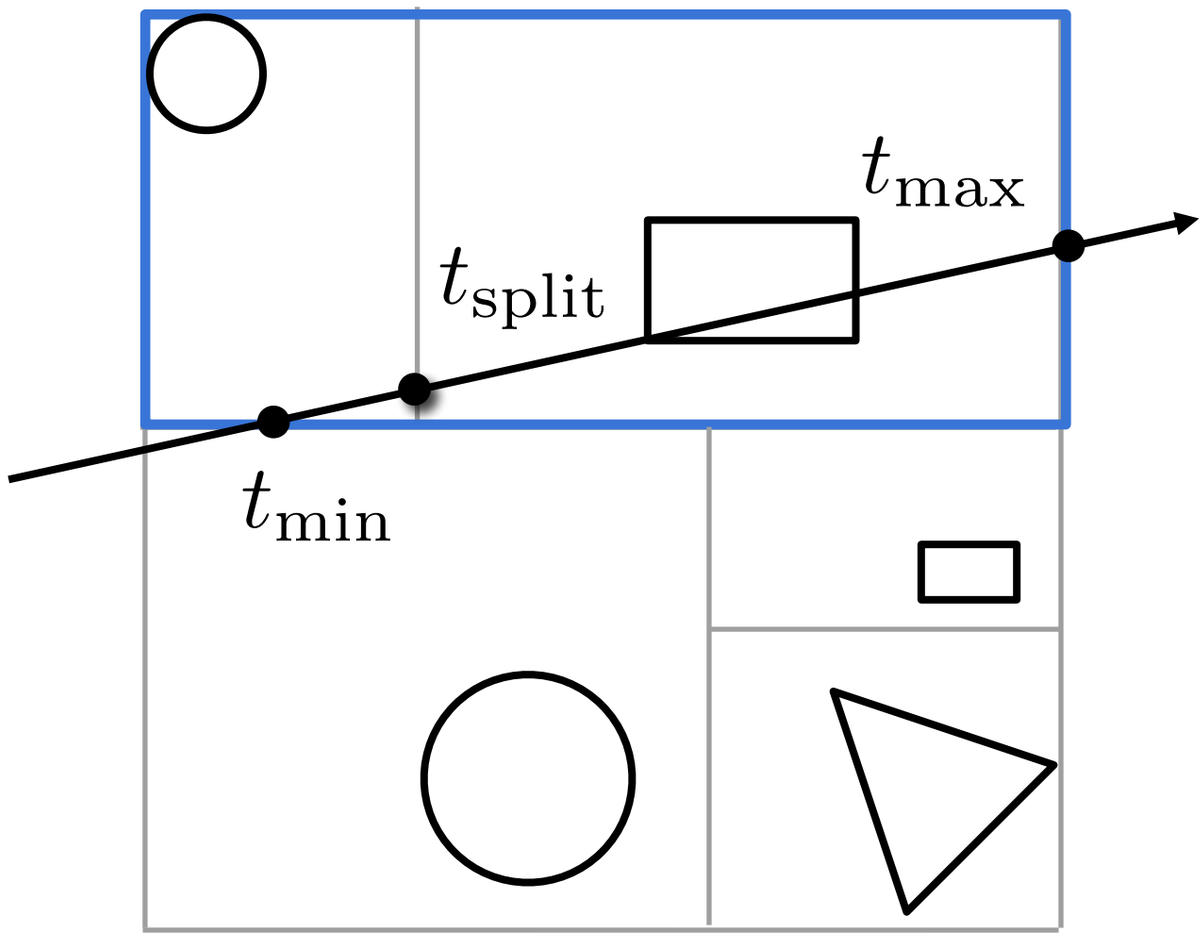


leaf node → intersect

kD-Trees

- Intersection
  - top-down recursion

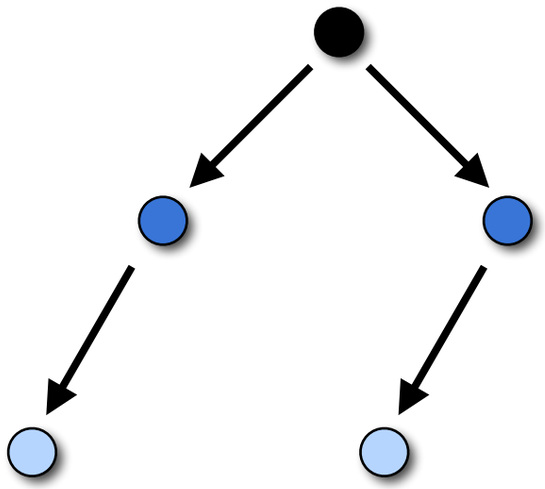


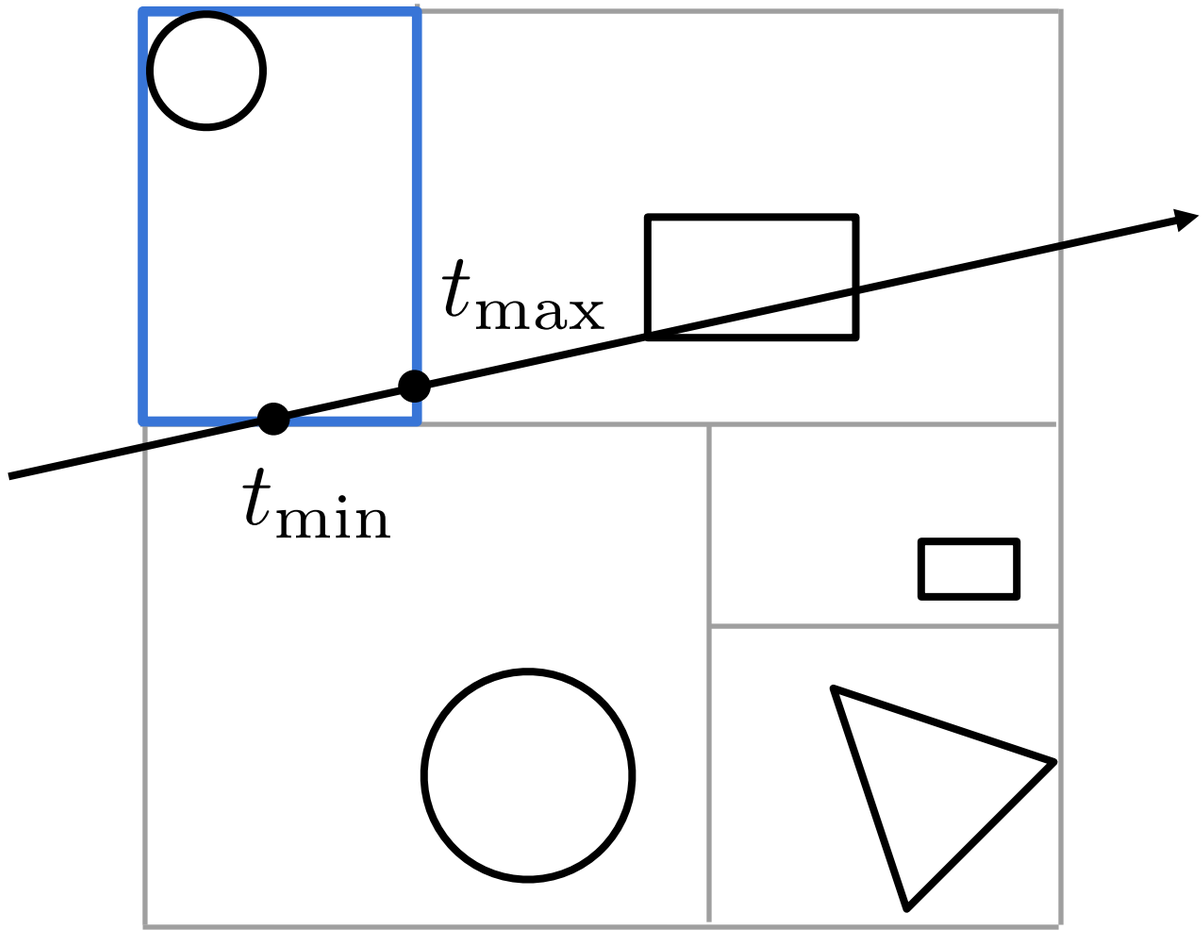


internal node → split

kD-Trees

- Intersection
  - top-down recursion

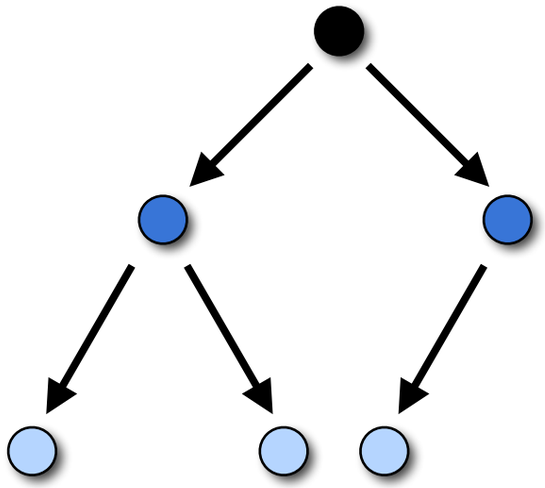


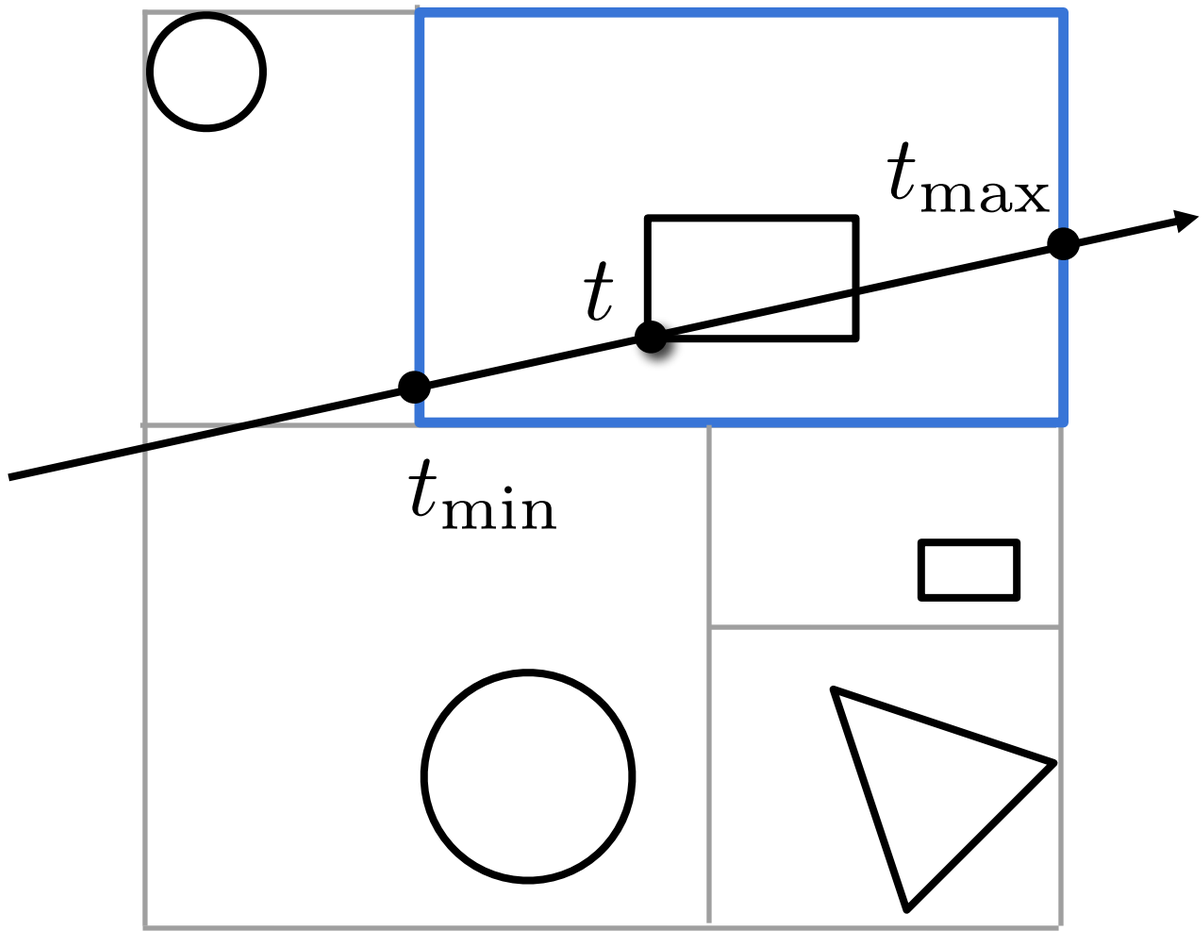


leaf node → intersect

kD-Trees

- Intersection
  - top-down recursion





leaf node → intersect

kD-Tree Construction

- Construction details
  - Prescribe maximum tree depth (empirical $8+1.3\log n$)
  - Need good termination criteria
  - How to choose the split plane?
    - midpoint
    - median cut
    - surface area heuristic (see PH 4.4.2 for details)

Where to split?

- Goal: Make ray-tracing fast!
  - Write down (estimated) cost of ray tracing
  - Choose split plane to minimize cost
- What is the cost of traversing a kD-tree cell?

$$\mathrm{C}(\mathrm{cell}) \;=\; \mathrm{C}(T) \;+\; \mathrm{P}(L)\cdot\mathrm{C}(L) \;+\; \mathrm{P}(R)\cdot\mathrm{C}(R)$$

  - $\mathrm{C}(T)$: cost of traversing cell (small)
  - $\mathrm{P}(L)$, $\mathrm{P}(R)$: probability to hit left/right child
  - $\mathrm{C}(L)$, $\mathrm{C}(R)$: cost of intersecting with l/r child

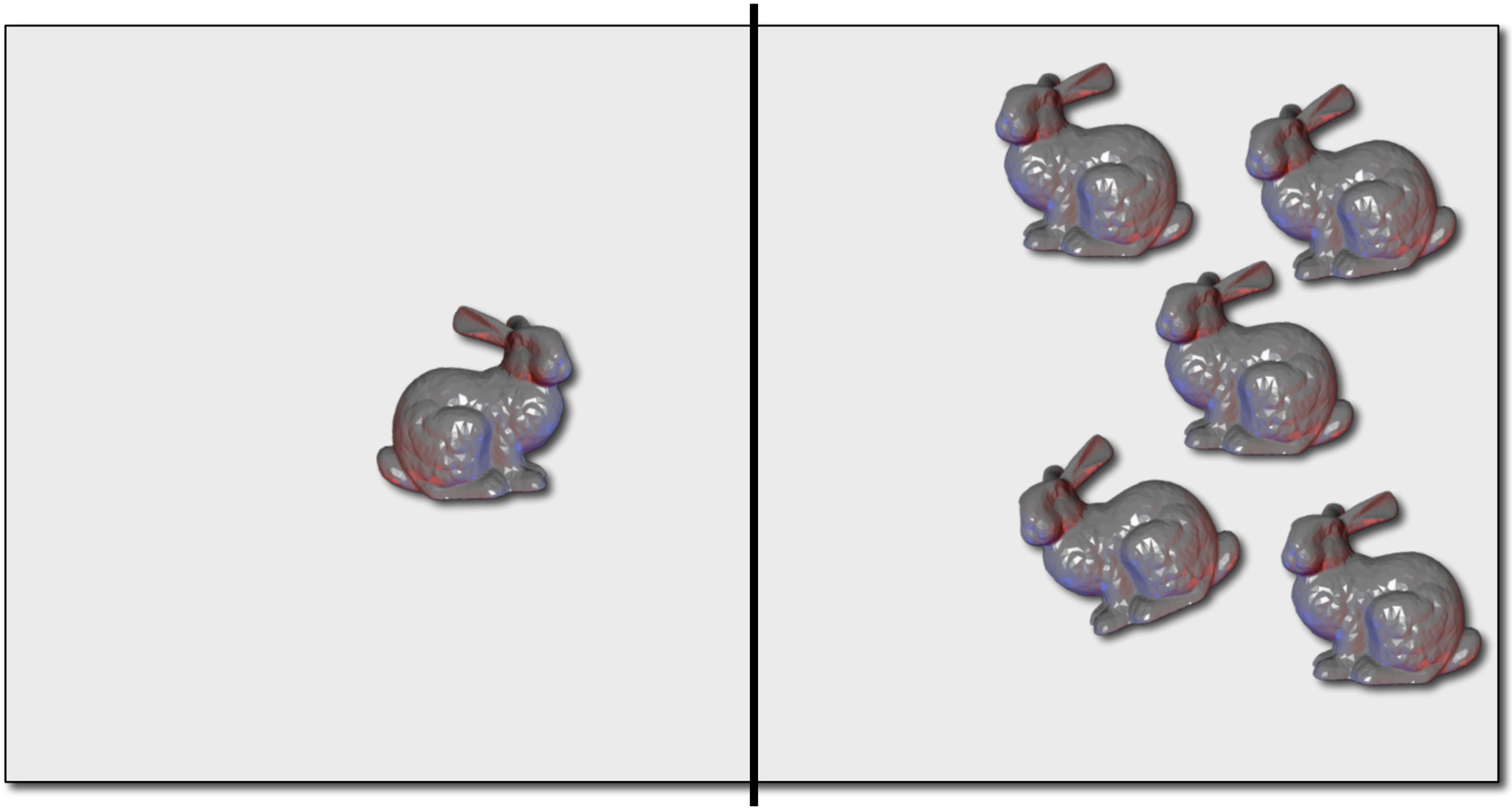
Where to split?

- Goal: Make ray-tracing fast!
  - Write down (estimated) cost of ray tracing
  - Choose split plane to minimize cost
- What is the cost of traversing a kD-tree cell?

$$\begin{aligned} \mathrm{C}(\mathrm{cell}) &= \mathrm{C}(T) + \mathrm{P}(L) \cdot \mathrm{C}(L) + \mathrm{P}(R) \cdot \mathrm{C}(R) \\ &= \mathrm{C}(T) + \mathrm{A}(L) \cdot \mathrm{Tri}(L) + \mathrm{A}(R) \cdot \mathrm{Tri}(R) \end{aligned}$$

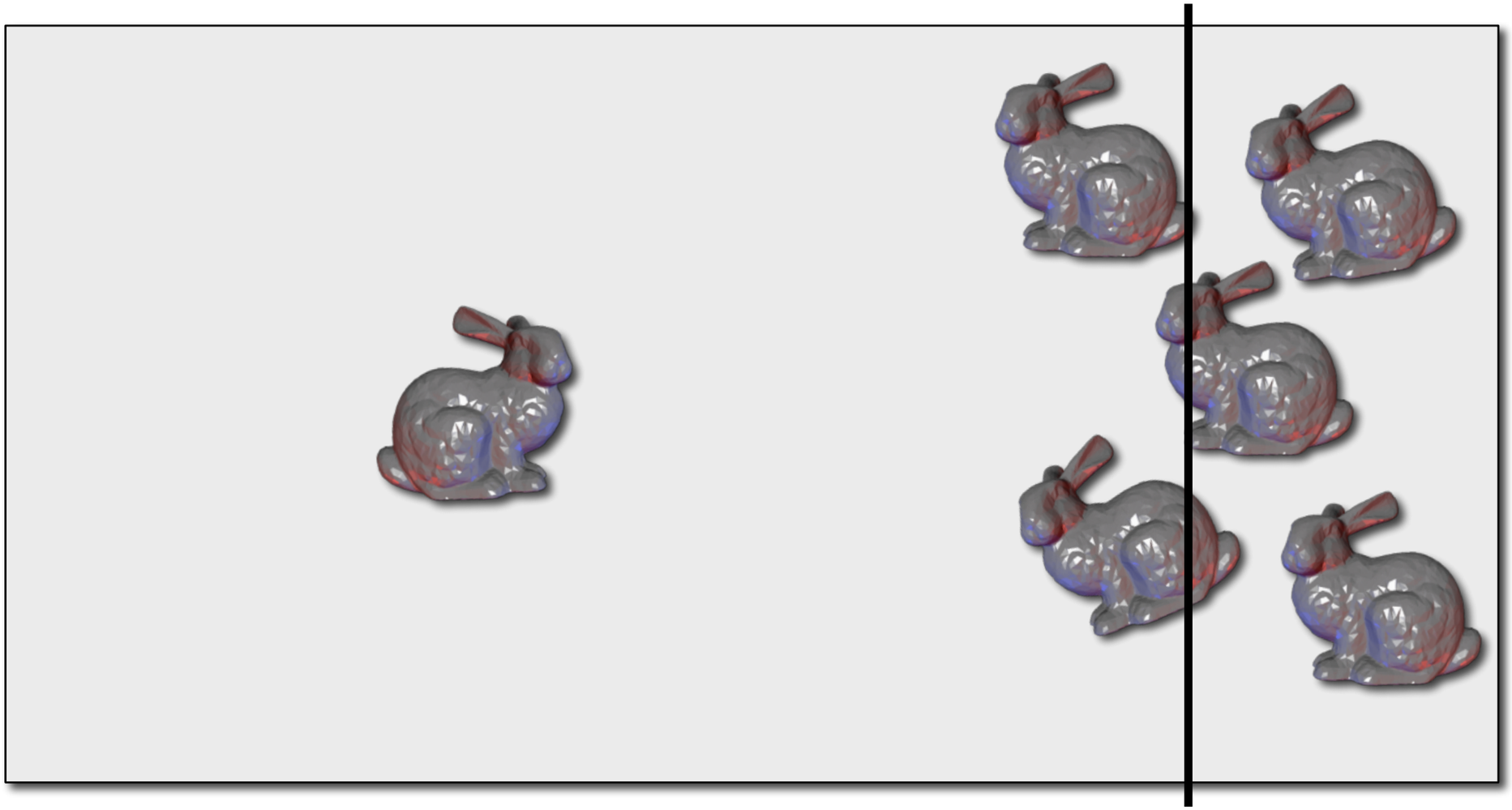
  - $\mathrm{C}(T)$: cost of traversing cell (small)
  - $\mathrm{A}(L)$, $\mathrm{A}(R)$: surface area of l/r child cell
  - $\mathrm{Tri}(L)$, $\mathrm{Tri}(R)$: number of triangles in cells

Split at Midpoint



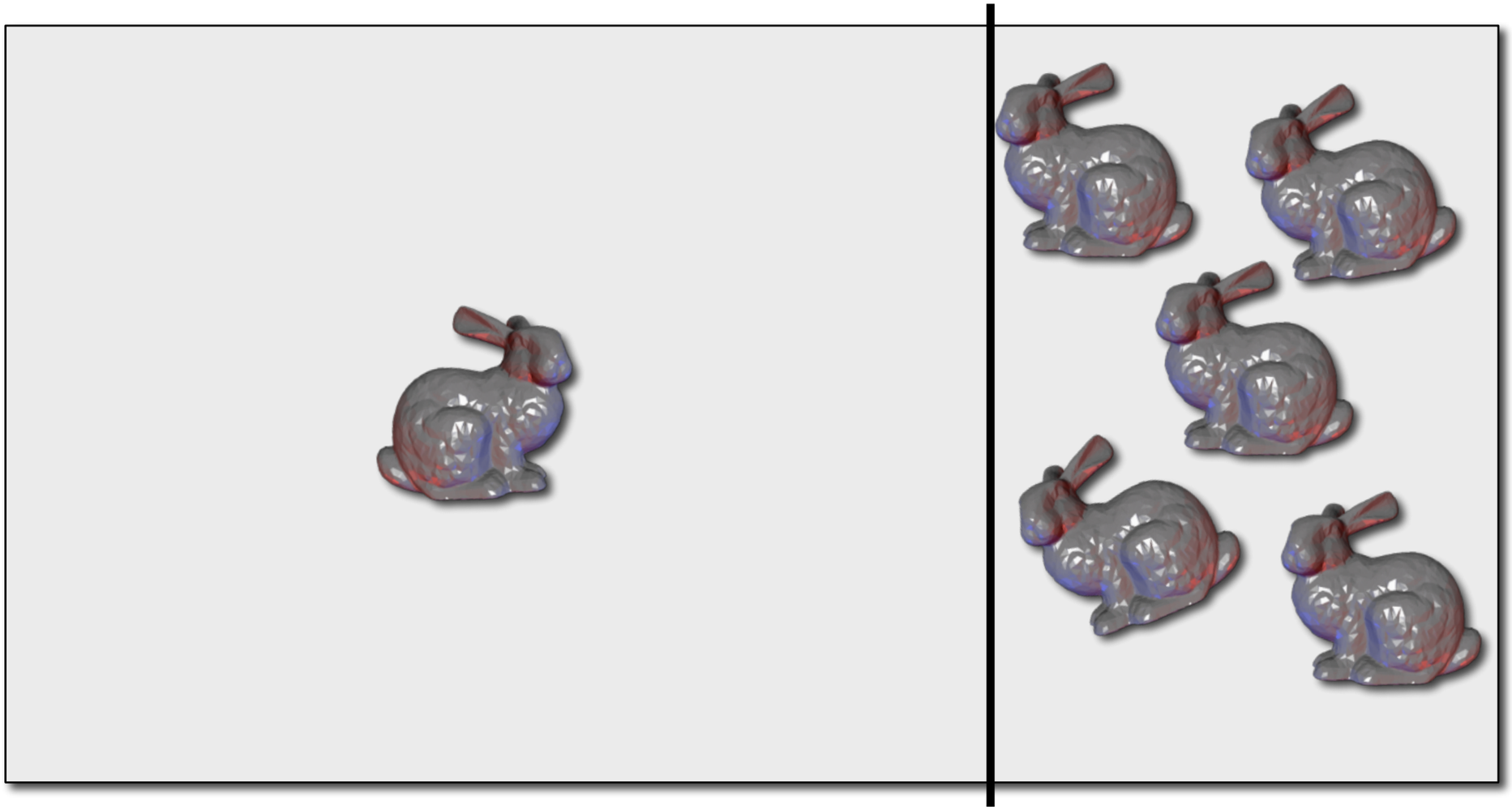
- Makes P(R) and P(L) equal
- Pays not attention to C(L) and C(R)

Split at Median



- Pays no attention to P(R) and P(L)
- Makes C(L) and C(R) equal

Surface Area Heuristic

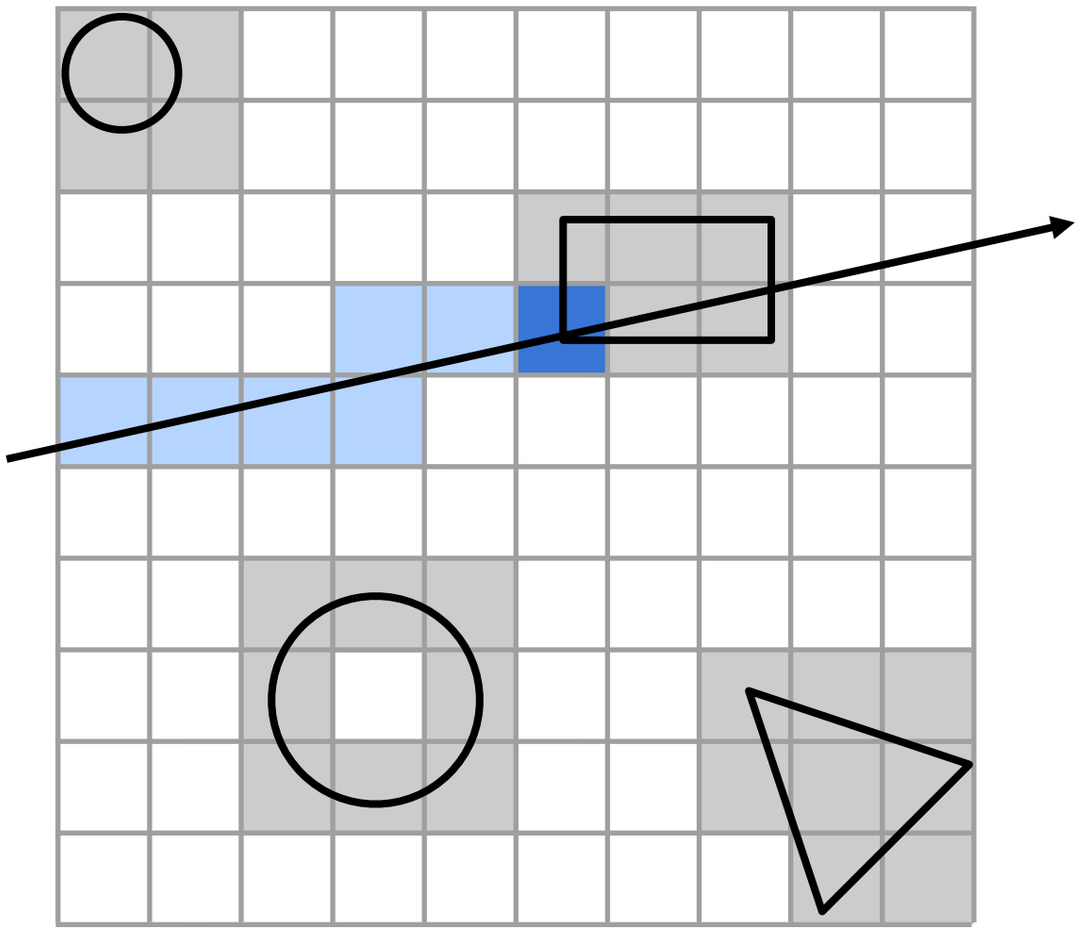


- Takes P’s and C’s into account
- Can be up to twice as fast

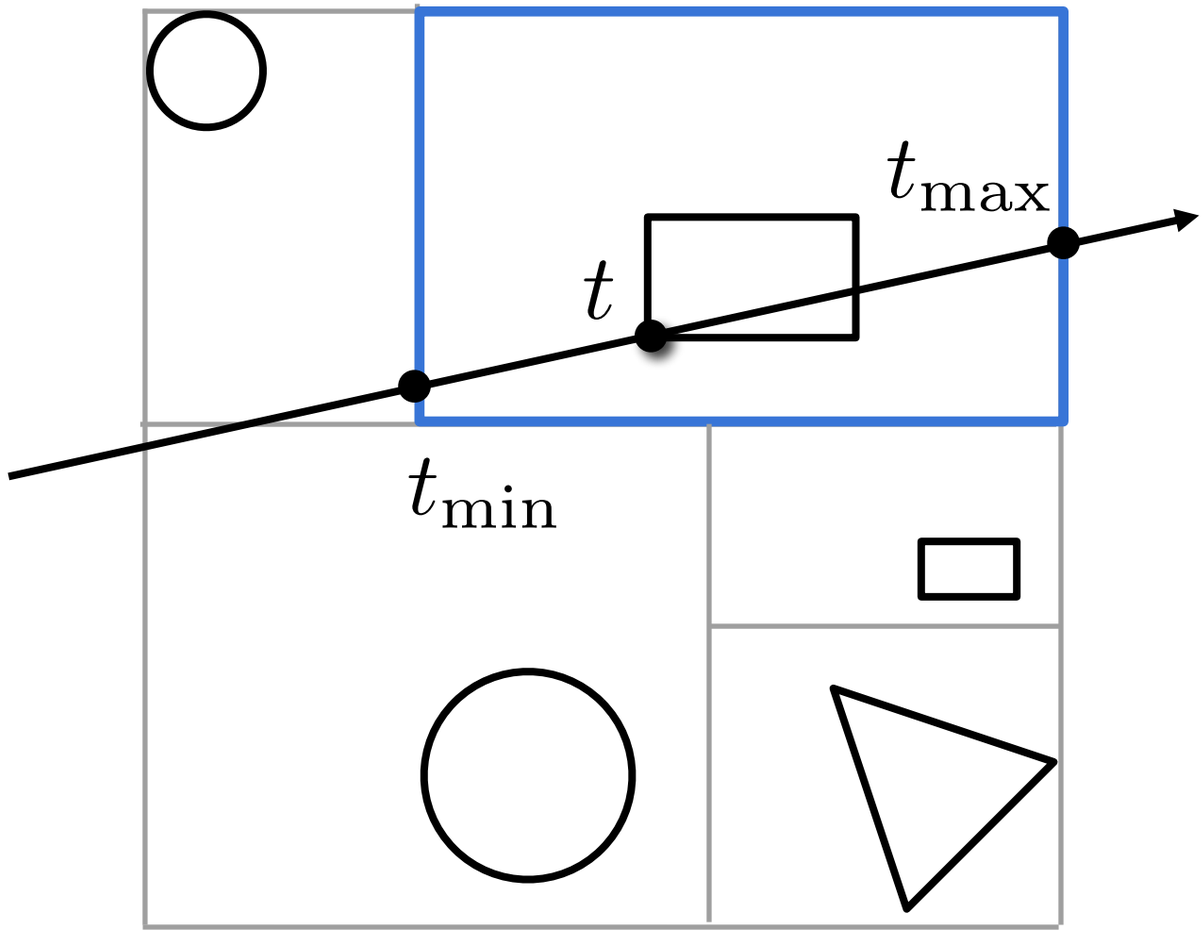
kD-Trees

- Advantages
  - Can adapt to object distribution
  - Construction not too complicated
  - Low memory consumption (a node needs 8 bytes)
  - Efficient traversal
  - Reduces cost to O(log*n*)
- Disadvantages
  - Not as easy to implement as uniform grids

Comparison



uniform grid



kD-tree

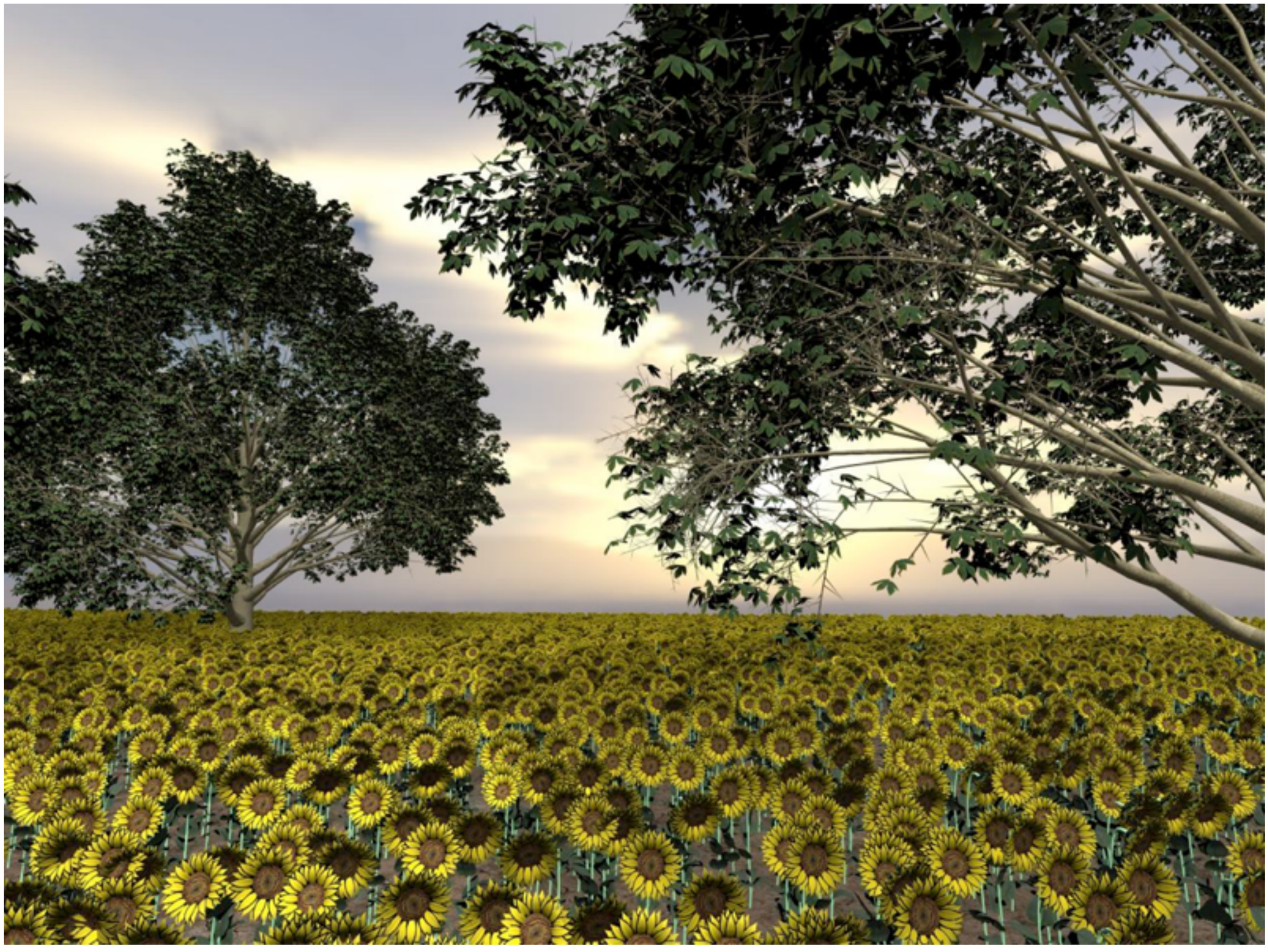
Outline

- Bounding Volumes
- Spatial Sorting
- **Parallelization**

Exploiting Hardware

- Caching
- SIMD extensions (SSE)
- Multi-Core parallelism (OpenMP)
- Programmable GPUs (CUDA, OpenCL, Optix)
- Distributed cluster computing (OpenRT)

1 Billion Triangles



Oliver Deussen

OpenRT

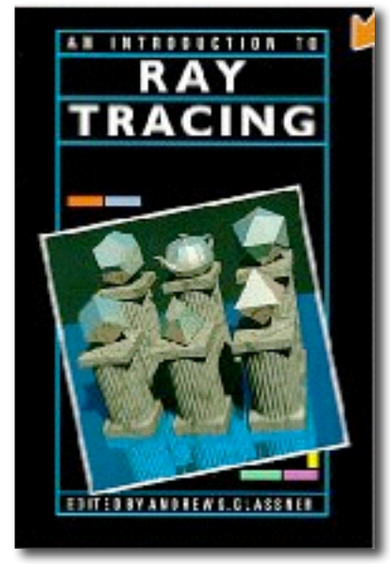

The OpenRT
Interactive Ray Tracing
Project
Computer Graphics Group,
Saarland University, Germany
http://www.openrt.de
http://graphics.cs.uni-sb.de

Summary

- Ray-surface intersections dominate computation effort in ray-tracing
- Spatial pre-sorting significantly reduces ray-surface intersection calculations
  - Divide and conquer: $O(n) \rightarrow O(\log n)$
- How to decide which is best?
  - Uniform grids, kd-trees, bounding volume hierarchies, parallelization, ...

Literature

- Glassner: ***An Introduction to Ray Tracing***, Academic Press, 1989.
  - Chapter 6



- Pharr, Humphreys: ***Physically Based Rendering***, Morgan Kaufmann, 2004.
  - Chapter 4