Spatial data structures

Charalambos Poullis

Department of Computer Science & Software Engineering
Faculty of Engineering & Computer Science

March 14, 2019

Lecture Overview

Ray tracing acceleration

- **faster intersections**
  - faster ray-object intersections e.g. object bounding volume, efficient computation of intersections
- **fewer ray-object intersections**
  - hierarchical bounding volumes (boxes, spheres)
  - spatial data structures
  - directional techniques
- **fewer rays**
  - adaptive tree-depth control
  - stochastic sampling
- **generalized rays** (beams, cones)

Spatial data structures

Data structures to store geometric information
Sample applications

- collision detections
- location queries
- simulations
- rendering

Spatial data structures for ray tracing

- object-centric data structures (bounding volumes)
- space subdivision (grids, octrees, BSP trees)

$\rightarrow$ speed-up of 10x, 100x, or more

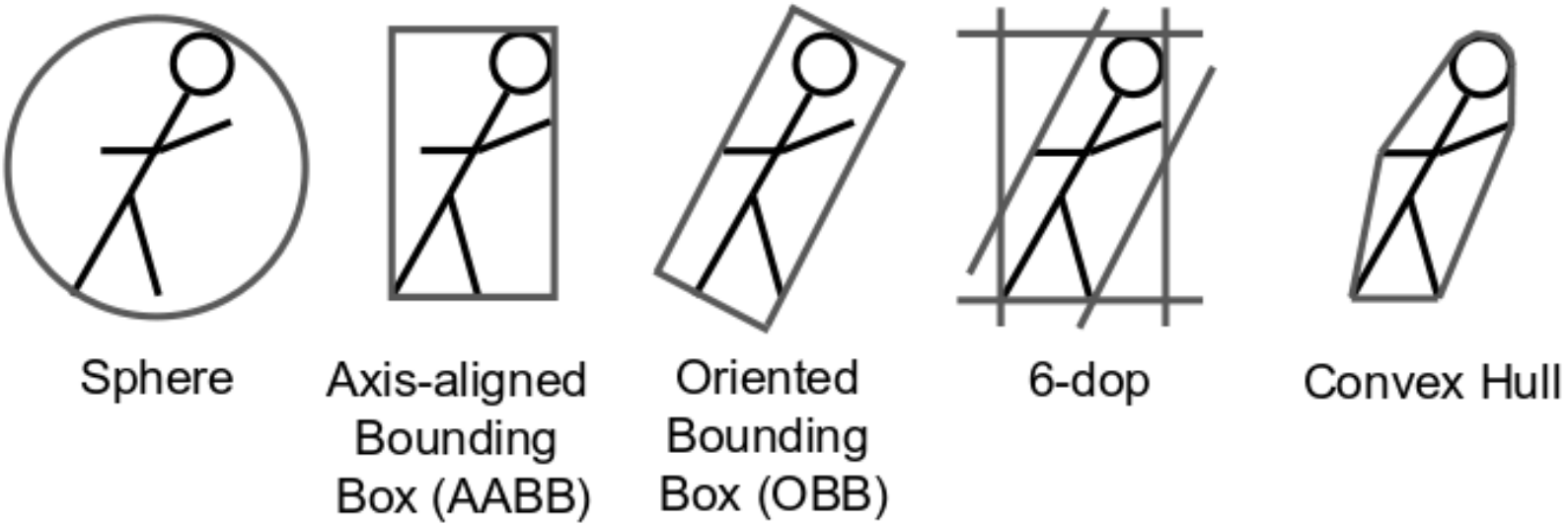
Bounding volumes

Wrap complex objects in simple ones

Does ray intersect bounding box?

- no: does not intersect enclosed objects
- yes: calculate intersection with closest objects

Common types:



Selection of bounding volumes

Effectiveness depends on:

- probability that ray hits bounding volume, but not enclosed objects (tight fit is better)
- cost of calculating intersections with bounding volume and enclosed objects

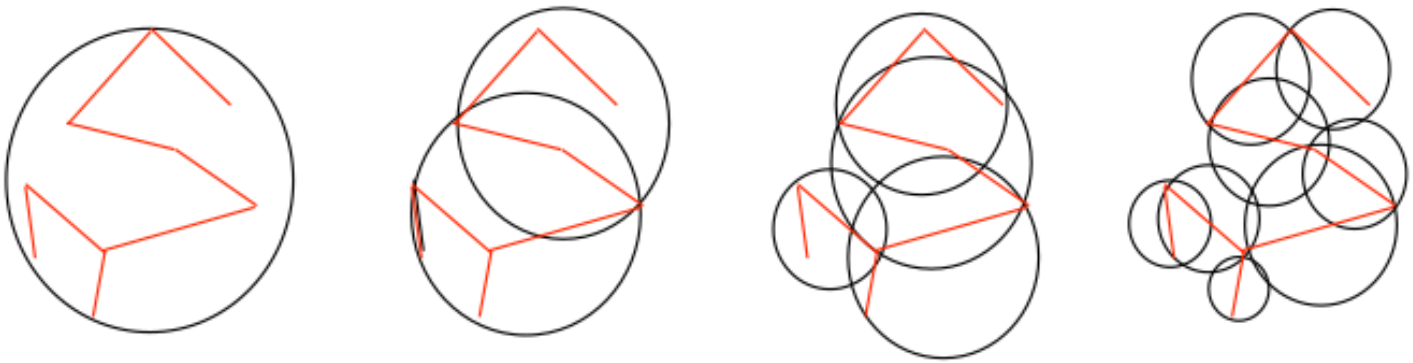
Use heuristics

Hierarchical bounding volumes

With simple bounding volumes, ray casting still requires $O(n)$ intersection tests

Idea: use tree data structure

- larger bounding volumes contain smaller ones etc
- sometimes naturally available (e.g. human figure)
- sometimes difficult to compute
- often reduces complexity to $O(log(n))$



Ray intersection algorithm

Recursively descend down the tree

- if ray misses bounding volume, no intersection
- if ray intersects bounding volume, recurse with the enclosed volumes and objects

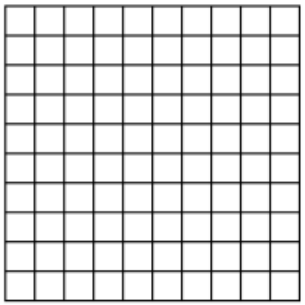
Maintain near and far bounds to prune further

Overall effectiveness depends on model and constructed hierarchy

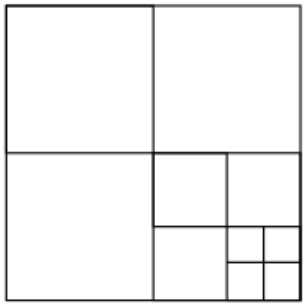
Spatial subdivision

Bounding volumes enclose objects, recursively i.e.object-centric

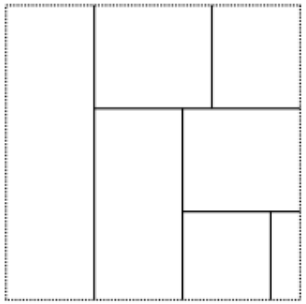
Alternatively, divide space (as opposed to objects) $\rightarrow$ for each segment of space, keep a list of intersecting surfaces or objects
Basic techniques:



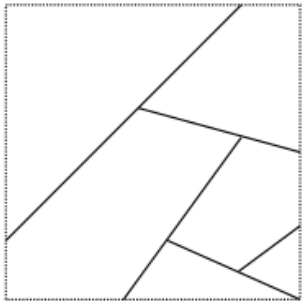
Uniform Spatial Sub



Quadtree/Octree



kd-tree



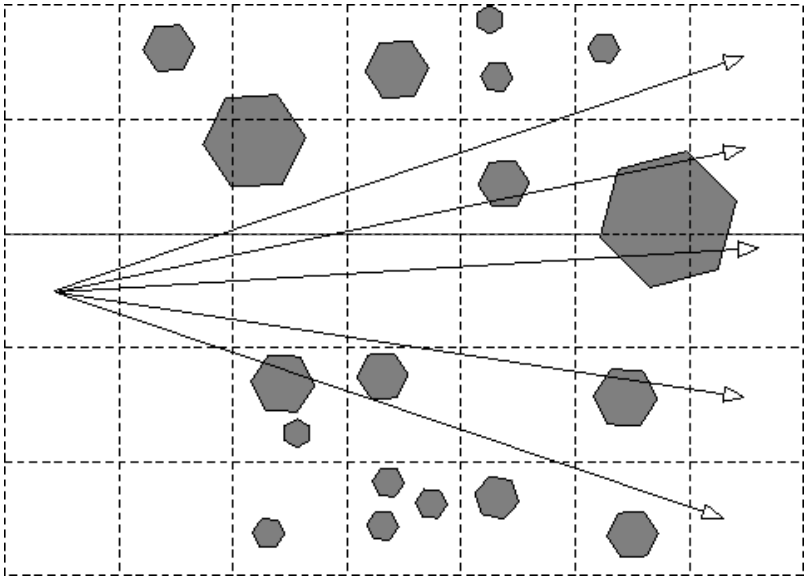
BSP-tree

Grids

3D array of cells (voxels) that tile space

Each cell keeps a list of all intersecting surfaces

Intersection algorithm steps from cell to cell

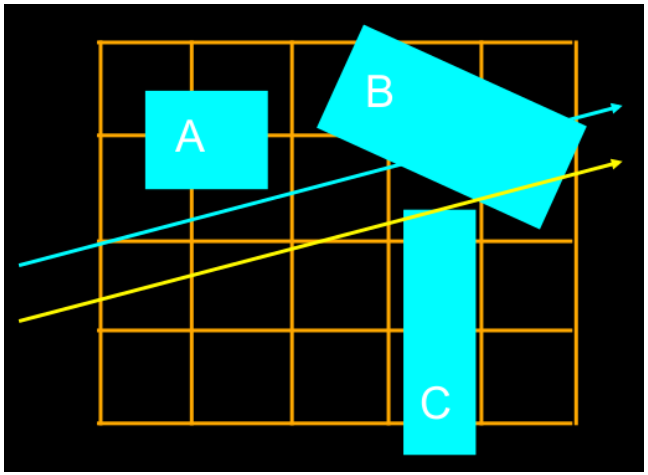


Caching intersection points

Objects can span multiple cells

For A need to test intersection only once

For B need to cache intersection and check next cell for any closer intersection with other objects. If not, C could be missed (yellow ray)



Assessment of grids

Poor choice when world is non-homogeneous
Grid resolution:

- too large: too many surfaces per cell
- too small: too many empty cells to traverse

Can use algorithms like Bresenham's for efficient traversal

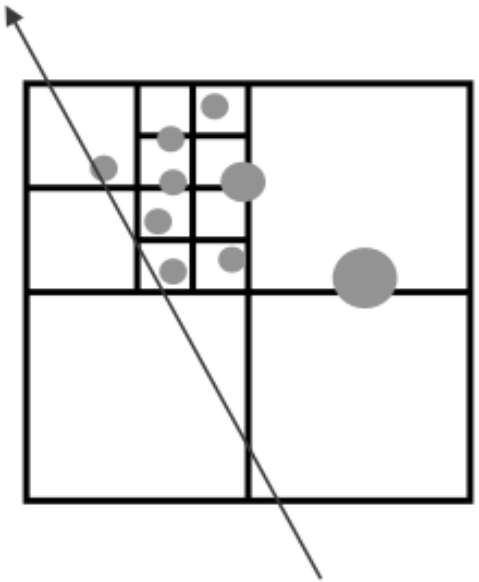
Non-uniform spatial subdivision more flexible. Can adjust to objects that are present

Quadtrees

Generalization of binary trees in 2D

- node (cell) is a square
- recursively split into 4 equal sub-squares
- stop subdivision based on number of objects

Ray intersection has to traverse quadtree. More difficult to step to next cell.



Octrees

Generalization of quadtree in 3D

Each cell may be split into 8 equal sub-cells

Internal nodes store pointers to children

Leaf nodes store list of surfaces

Adapts well to non-homogeneous scenes

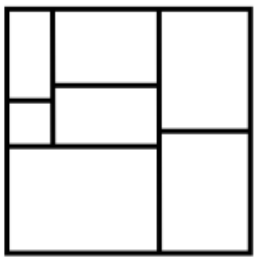
Assessment for ray tracing

- Grids
  - easy to implement
  - require a lot of memory
  - poor results for non-homogeneous scenes
- Octrees
  - better on most scenes (more adaptive)
- Alternative: nested grids
- Spatial subdivision expensive for animations
- Hierarchical bounding volumes
  - natural for hierarchical objects
  - better for dynamic scenes

Other spatial subdivision techniques

Relax rules for quadtrees and octrees

- k-dimensional tree (k-d tree)
  - split at arbitrary interior point
  - split one dimension at a time
- binary space partitioning tree (BSP tree)
  - in 2 dimensions, split with any line
  - in k dims. split with k-1 dimensional hyperplane
  - particularly useful for painter's algorithm
  - can also be used for ray tracing



BSP trees

Split space with any line (2D) or plane (3D)

Applications

- Painter's algorithm for hidden surface removal
- Ray casting

Inherent spatial ordering given viewpoint

- left subtree: in front, right subtree: behind

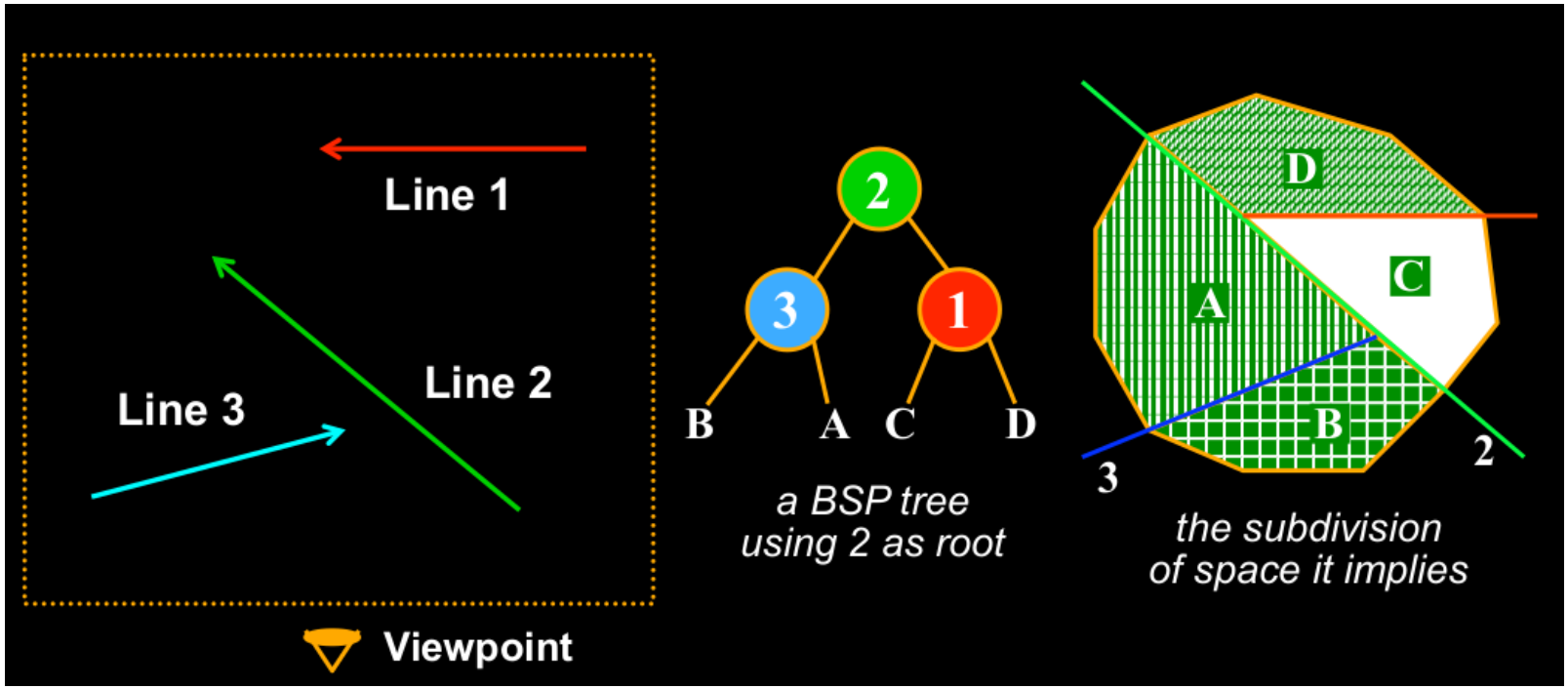
Problem: finding good space partitions

- proper ordering for any viewpoint
- how to balance the tree

Building a BSP tree

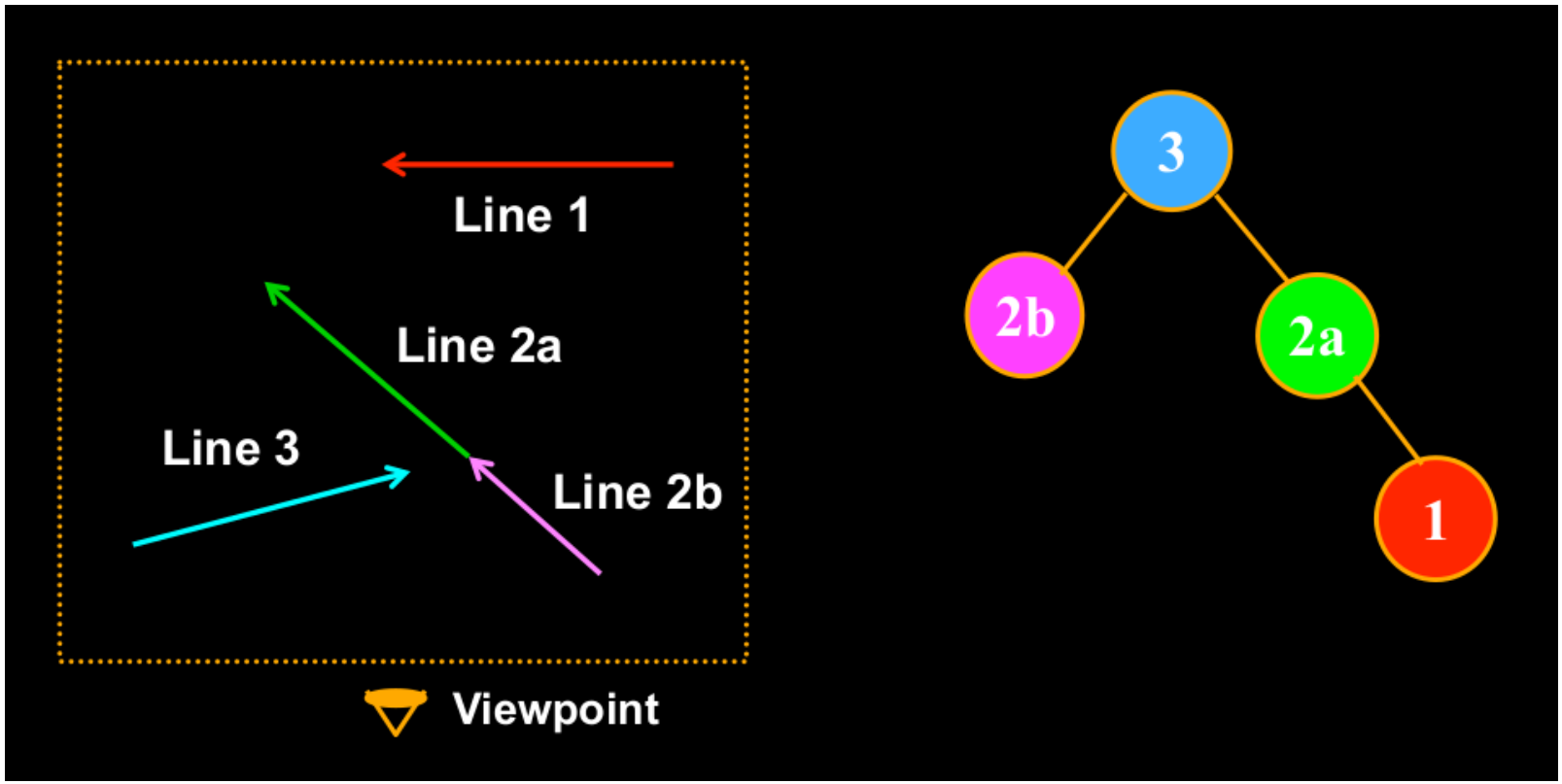
Use hidden surface removal as intuition

Using line 1 or line 2 as root is easy



Splitting of surfaces

Using line 3 as root requires splitting



Building a good tree

- naive partitioning of n polygons yields $O(n^3)$ polygons (in 3D)
- algorithms with $O(n^2)$ increase exist
  - try all, use polygon with fewest splits
  - do not need to split exactly along polygon planes
- should balance tree
  - more splits allow easier balancing
  - rebalancing?

Painter's algorithm with BSP trees

- building the tree
  - may need to split some polygons
  - slow, but done only once
- traverse back-to-front or front-to-back
  - order is viewer-direction dependent
  - what is front and what is back of each line changes
  - determine order on the fly

Details of Painter's algorithm

Each plane has form $Ax + By + Cz + D = 0$

Plug in coordinates and determine

- positive: front side
- zero: on plane
- negative: back side

Back-to-front: inorder traversal, farther child first

Front-to-back: inorder traversal, near child first

Do backface culling with same sign test

Clip against visible portion of space (portals)

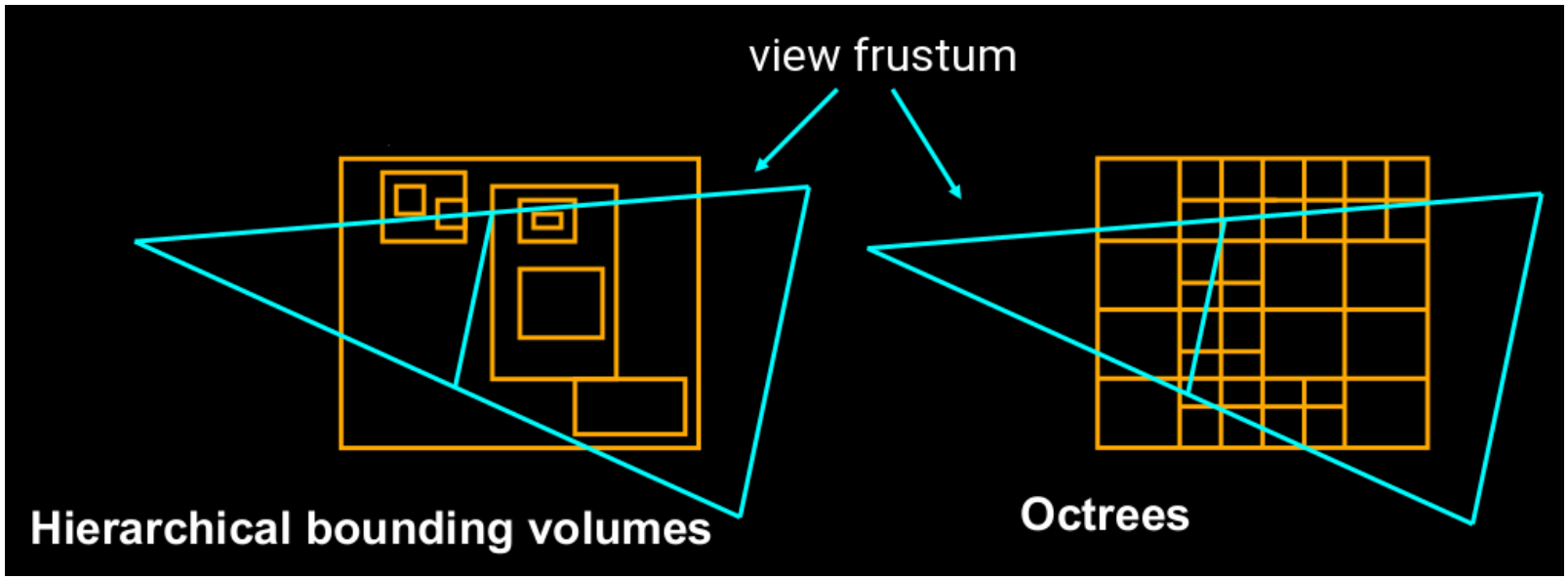
Culling with spatial data structures

Accelerate culling

- goal accept or reject whole sets of objects
- can use any spatial data structures

Scene should be mostly fixed

- terrain fly-through
- gaming



Data structures demos

BSP Tree construction

KD Tree construction

Real-time and interactive ray tracing

Interactive ray tracing via space subdivision

State of the art in interactive ray tracing