Simulation in Computer Graphics

Collision Detection based on Spatial Partitioning

Matthias Teschner

Computer Science Department
University of Freiburg

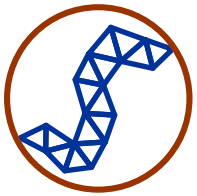
UNI FREIBURG

Outline

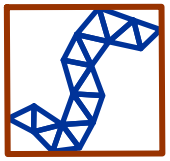
- introduction
- uniform grid
- Octree and k-d tree
- BSP tree

Model Partitioning

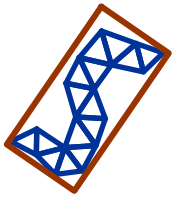
(1) Bounding volumes



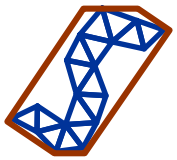
Sphere



Axis-Aligned Bounding Box (AABB)

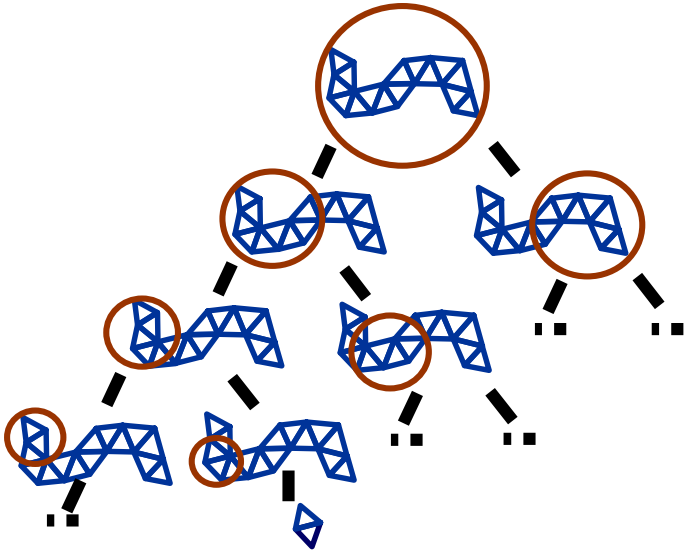


Object-Oriented Bounding Box (OBB)

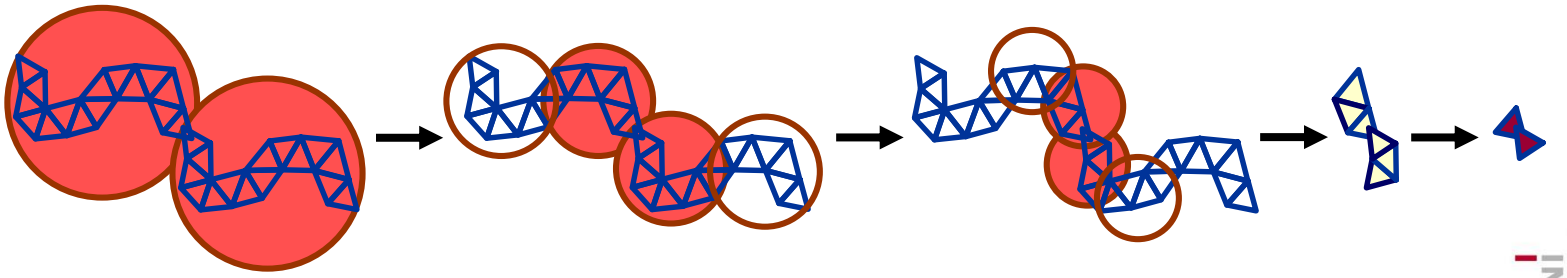


Discrete-Orientation Polytope (k-DOP)

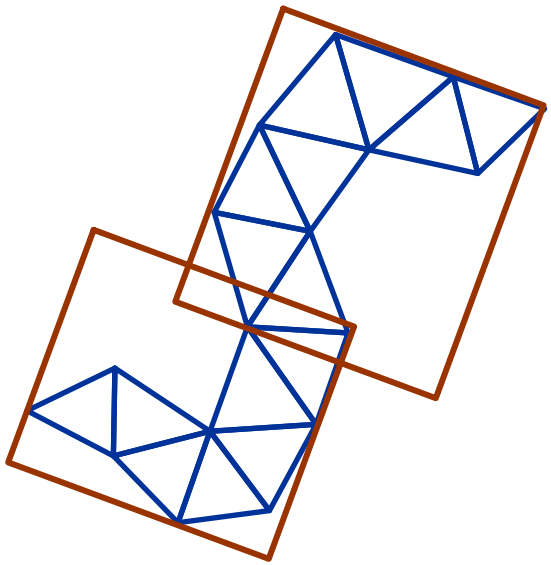
(2) Bounding volume tree



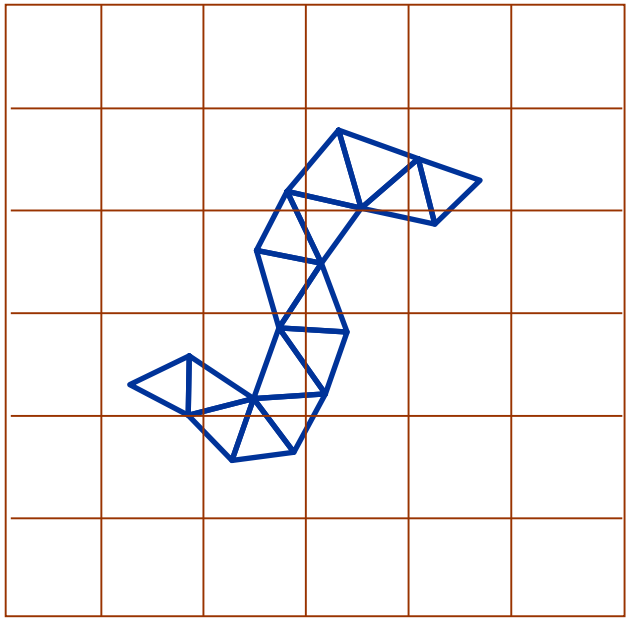
(3) Collision detection test



Model vs. Space Partitioning



model partitioning

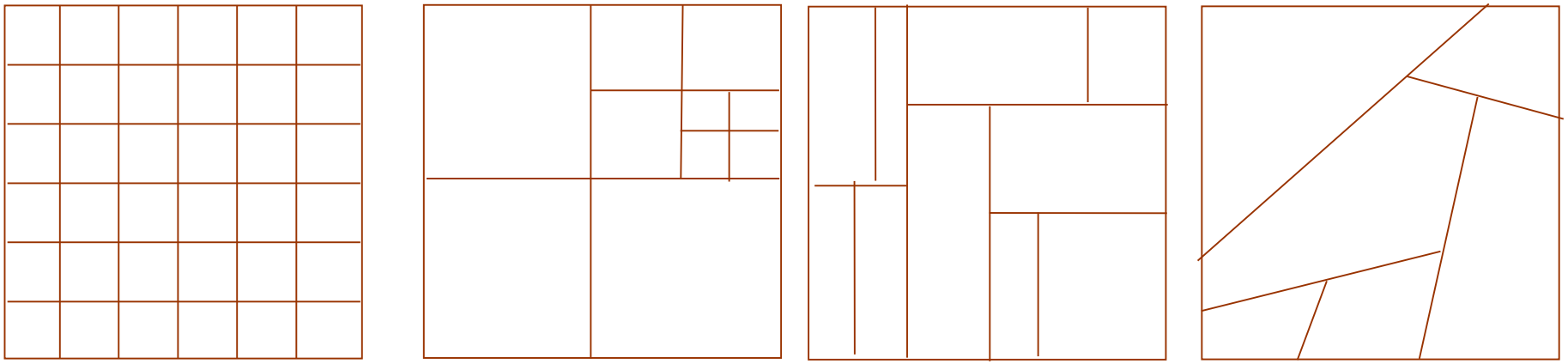


space partitioning

Motivation

- restrict pairwise object tests to objects that are located in the same region of space
- only objects or object primitives in the same region of space can overlap
- efficient broad-phase approach for larger numbers of objects

Spatial Data Structures



uniform grid | Quadtree / Octree | k-d tree | BSP tree

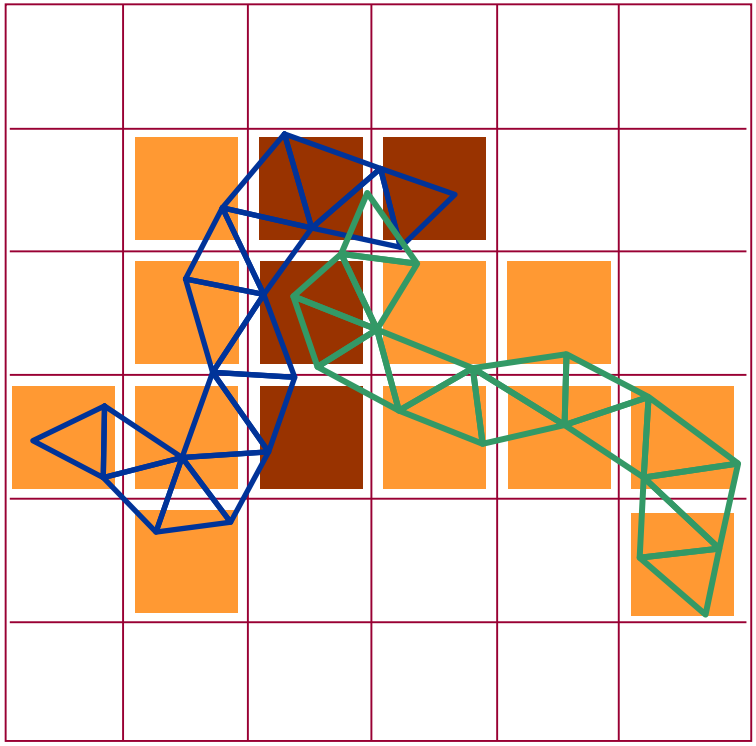
- space is subdivided into cells
- cells maintain references to primitives intersecting the cell
- data structures have different degrees-of-freedom
- actual space subdivision is adapted to the scene

Outline

- introduction
- uniform grid
- Octree and k-d tree
- BSP tree

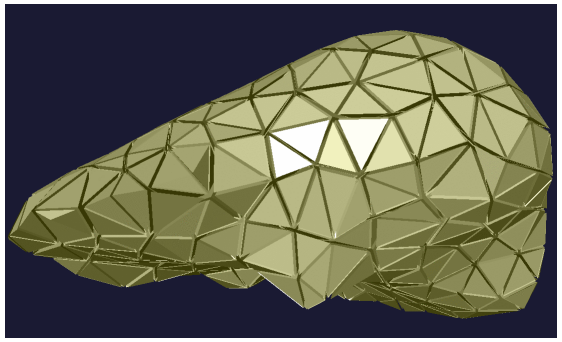
Basic Idea

- space is divided up into cells
- object primitives are placed into cells
- object primitives within the same cell are checked for collision
- pairs of primitives that do not share the same cell are not tested (trivial reject)

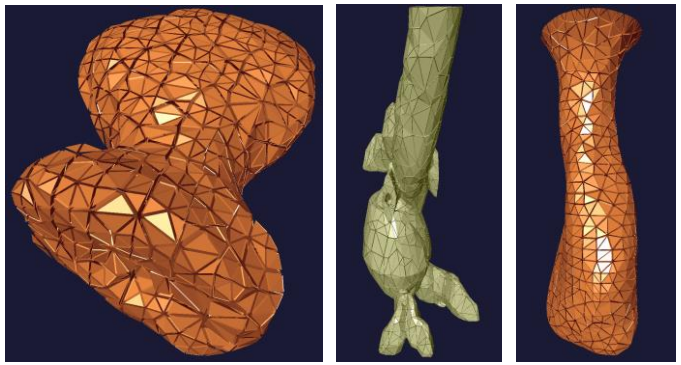


Application

- collision detection for deformable objects
- tetrahedral meshes
- uniform 3D grid
- non-uniform distribution of object primitives and unbounded simulation domain
  - hashed storage
- detection of collisions and self-collisions
- discussion of parameters



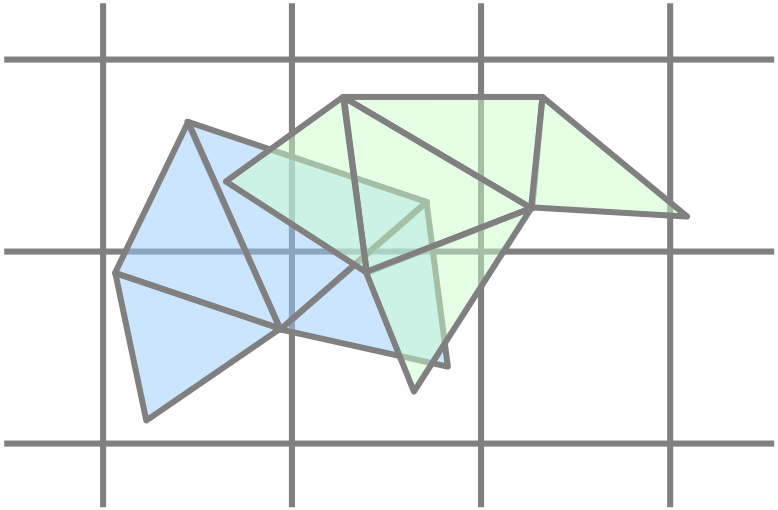
Epidaure, INRIA



NCCR Co-Me

Setup

infinite uniform grid:



hash function:

H(cell) → hash table index

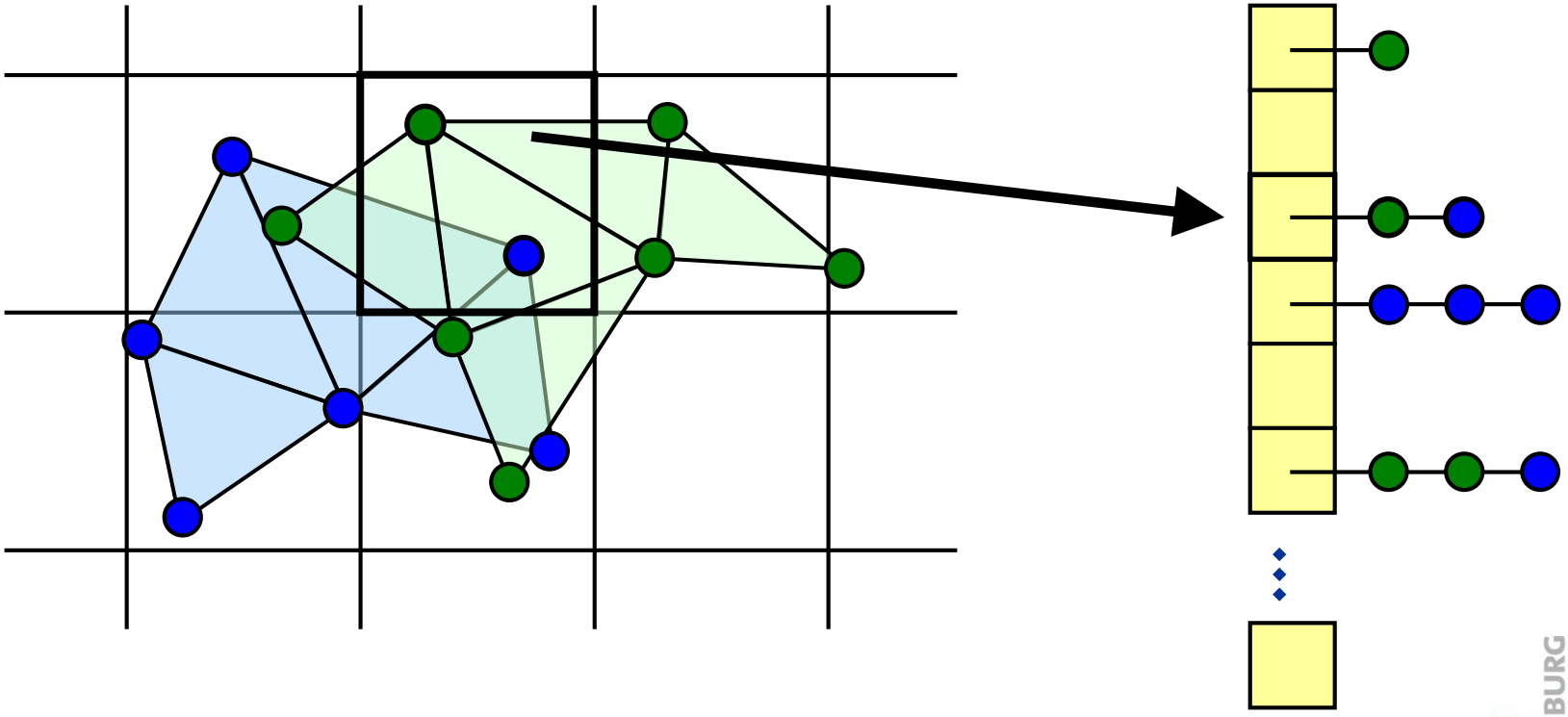
hash table:

(spatial data structure)

(representation)

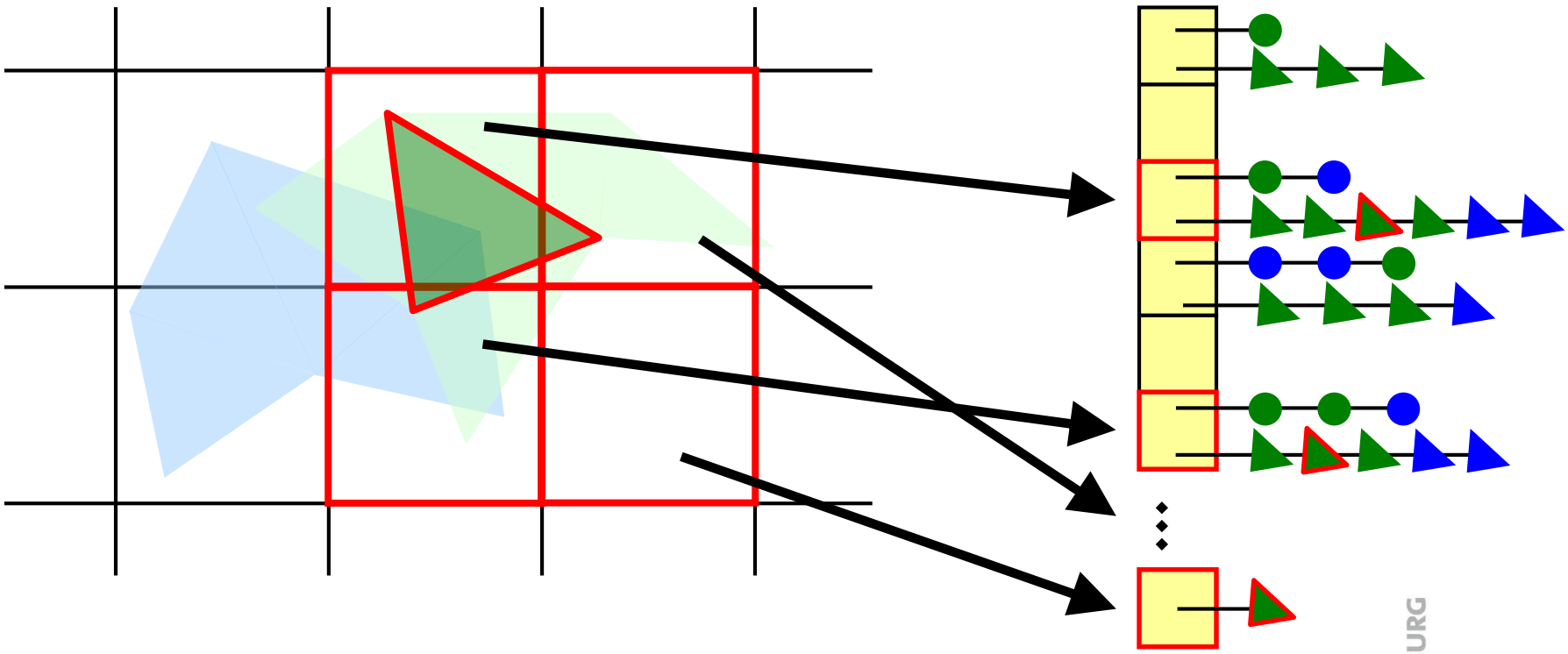
Stage 1

- all vertices are hashed according to their cell



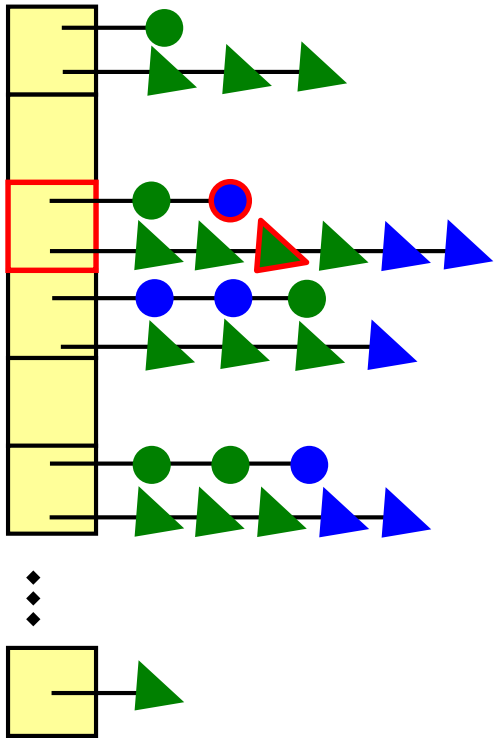
Stage 2

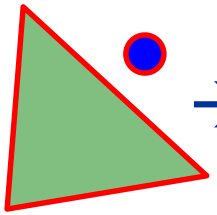
- all tetrahedrons are hashed according to the cells touched by their bounding box

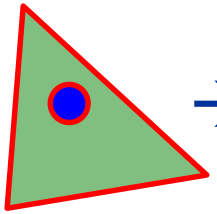


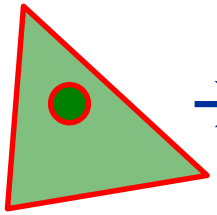
Stage 3

- vertices and tetrahedrons in the same hash table entry are tested for intersection

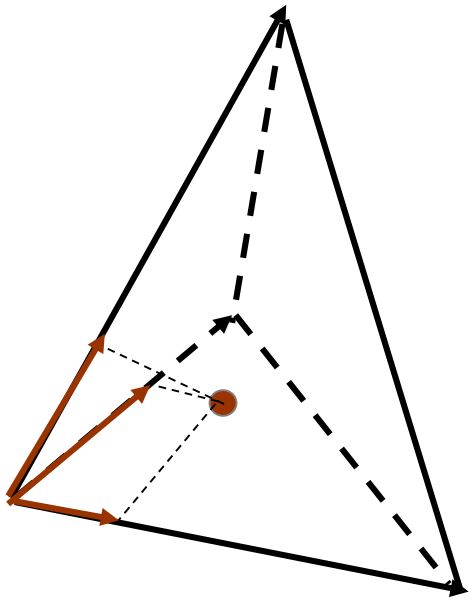


A)  → no collision

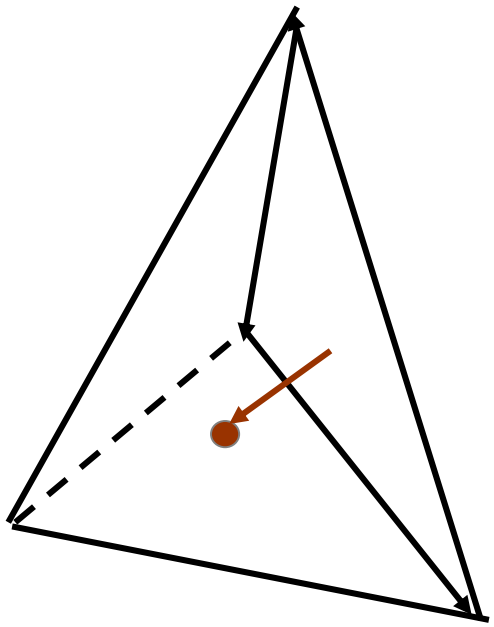
B)  → collision

C)  → self-collision

Vertex-in-Tetrahedron Test



Barycentric coordinates



Oriented faces

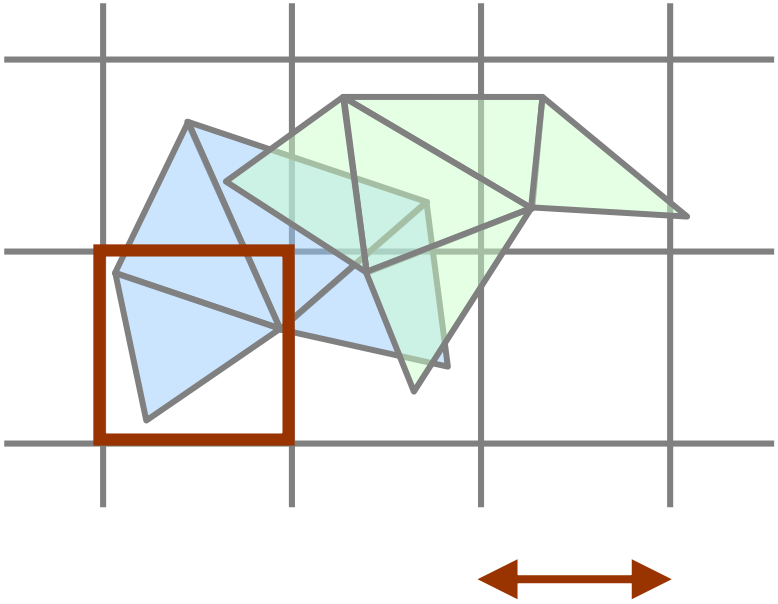
- Barycentric coordinates more efficienct
- they also provide useful collision information

Implementation

- store all vertices in the hash table
- compute hash table indices for the bounding boxes of the tetrahedrons
- do not store the tetrahedrons in the hash table, but check for intersection with all vertices in the respective entry
- parameters
  - grid cell size
  - grid cell shape
  - hash table size
  - hash function

Parameters

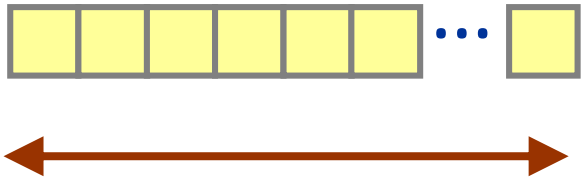
infinite uniform grid:



hash function:

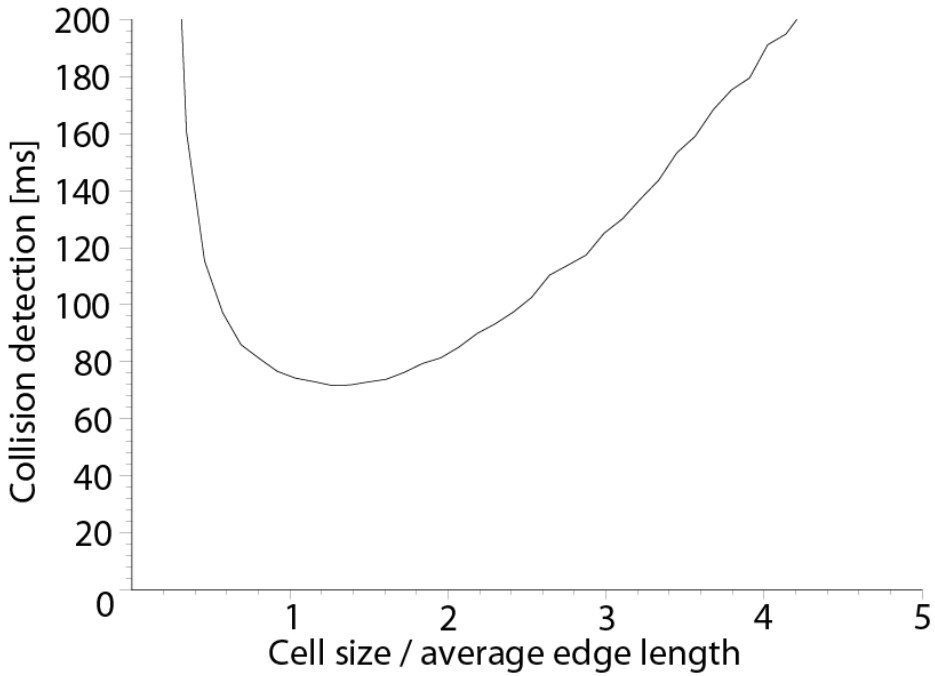
H(cell) → hash table index

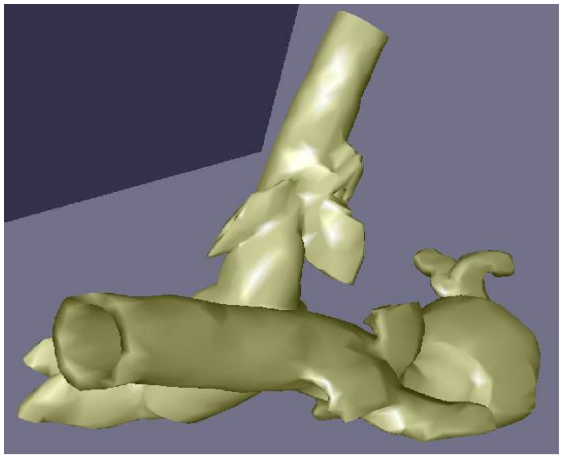
hash table:



Grid Cell Size

- cell size should be equal to the size of the bounding box of an object primitive [Bentley 1977]
- [Teschner, Heidelberger et al. 2003]

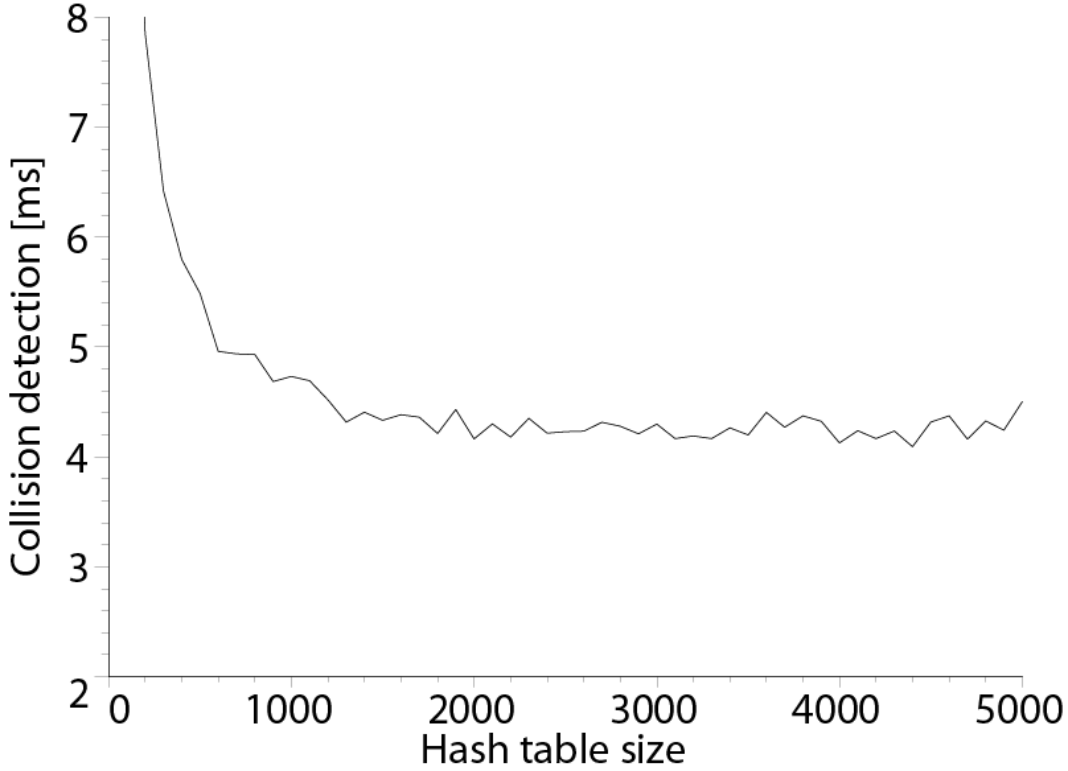


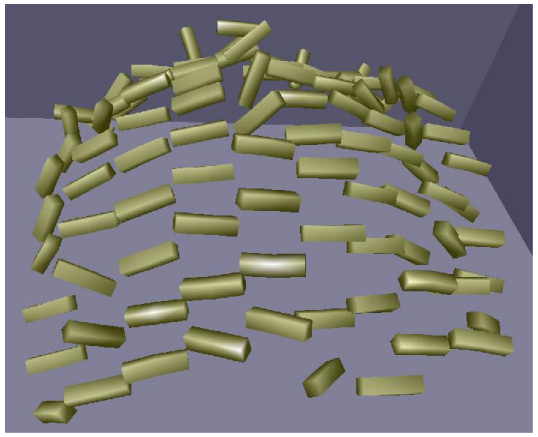


test scenario

Hash Table Size

- hash collisions reduce the performance
- larger hash table can reduce hash collisions





test scenario

Hash Function

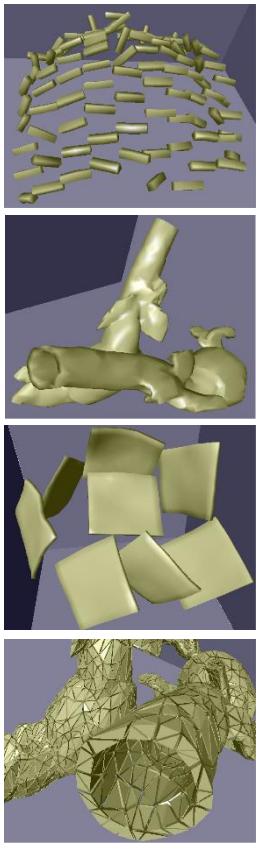
- should avoid hash collisions
- should be efficient (has to be computed for all primitives)

  $$H(x, y, z) = (p_1 x \ \text{ xor } \ p_2 y \ \text{ xor } \ p_3 z) \ \text{ mod } \ n$$
- cell coordinates: x, y, z
- large primes: p_1, p_2, p_3
- hash table size: n

Performance

- linear in the number of primitives
- independent of the number of objects

objects	tetras	vertices	max time [ms]
100	1000	1200	6
8	4000	1936	15
20	10000	4840	34
2	20514	5898	72
100	50000	24200	174



test scenarios

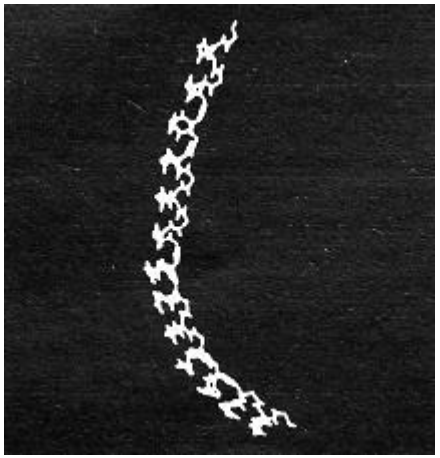
Pentium 4, 1.8GHz

Summary – Uniform Grid

- space uniformly partitioned into axis-aligned space cells
- primitives (or their AABBs) are scan-converted to identify intersected space cells
- hashed storage of cells for non-uniform distribution
- simple and efficient
- particularly interesting for deformable objects, n-body environments and self-collision
- parameters significantly influence the performance
- performance dependent on the number of primitives
- performance independent of the number of objects
- technique works with various types of primitives

… Some History

- [Levinthal 1966]
  - 3D grid ("cubing")
  - analysis of molecular structures
  - neighborhood search
    to compute atom interaction
- [Rabin 1976]
  - 3D grid + hashing
  - finding closest pairs
- [Turk 1989, 1990]
  - rigid collision detection
  - 3D grid + hashing



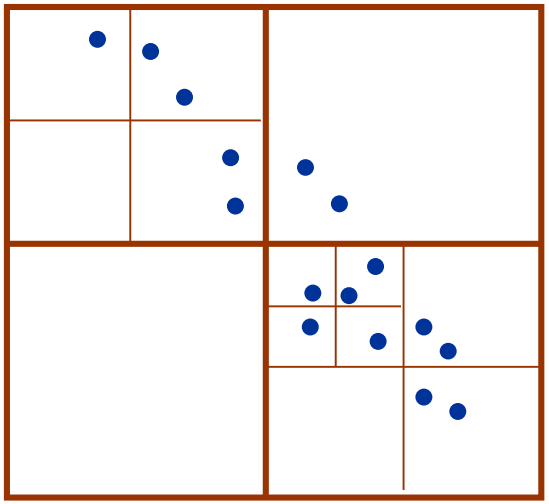
Cyrus Levinthal, MIT

Outline

- introduction
- uniform grid
- Octree and k-d tree
- BSP tree

Octree

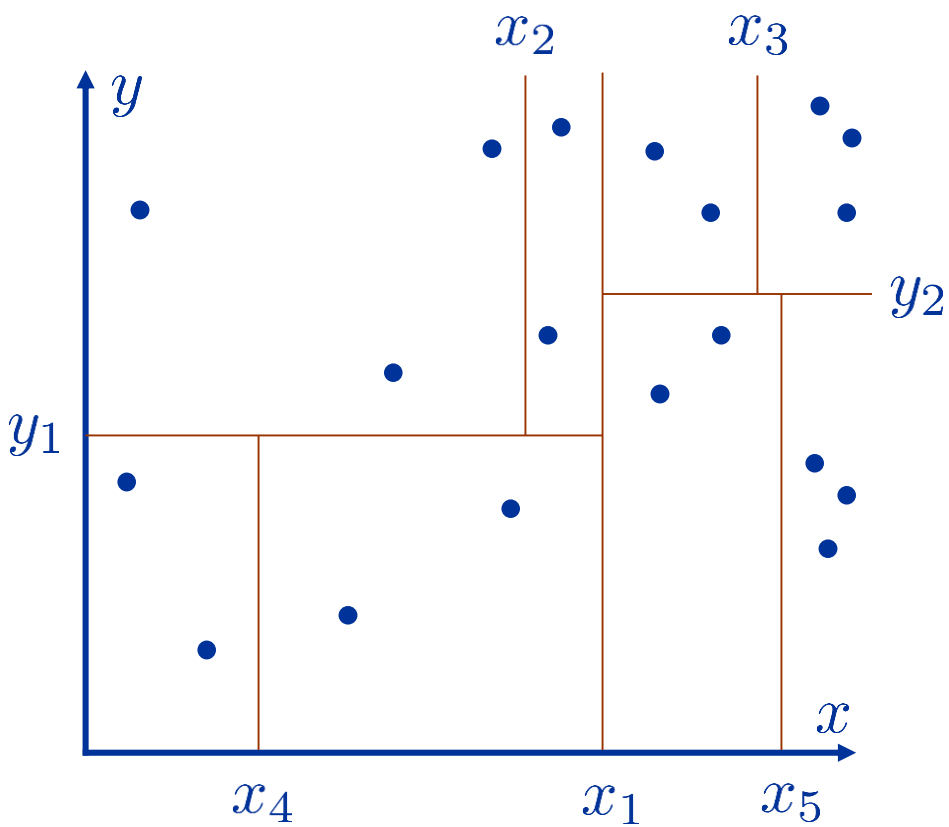
- hierarchical structure
- space partitioning into rectangular, axis-aligned cells
- Octree root node corresponds to AABB of an object
- internal nodes represent subdivisions of the AABB
- leaves represent cells which maintain primitive lists

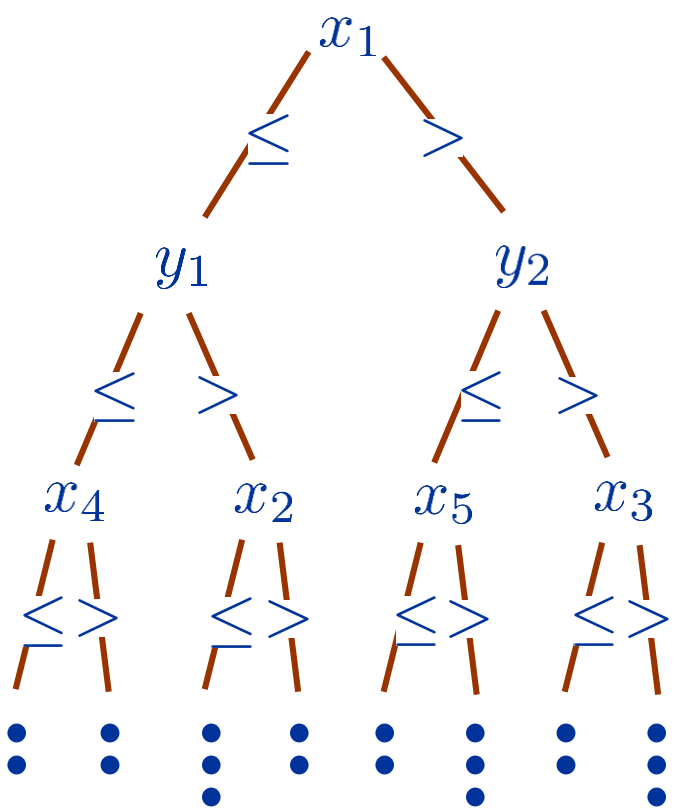


Octree

- uniform or non-uniform subdivision
- adaptive to local distribution of primitives
  - large cells in case of low density of primitives
  - small cells in case of high density
- dynamic update
  - cells with many primitives can be subdivided
  - cells with less primitives can be merged

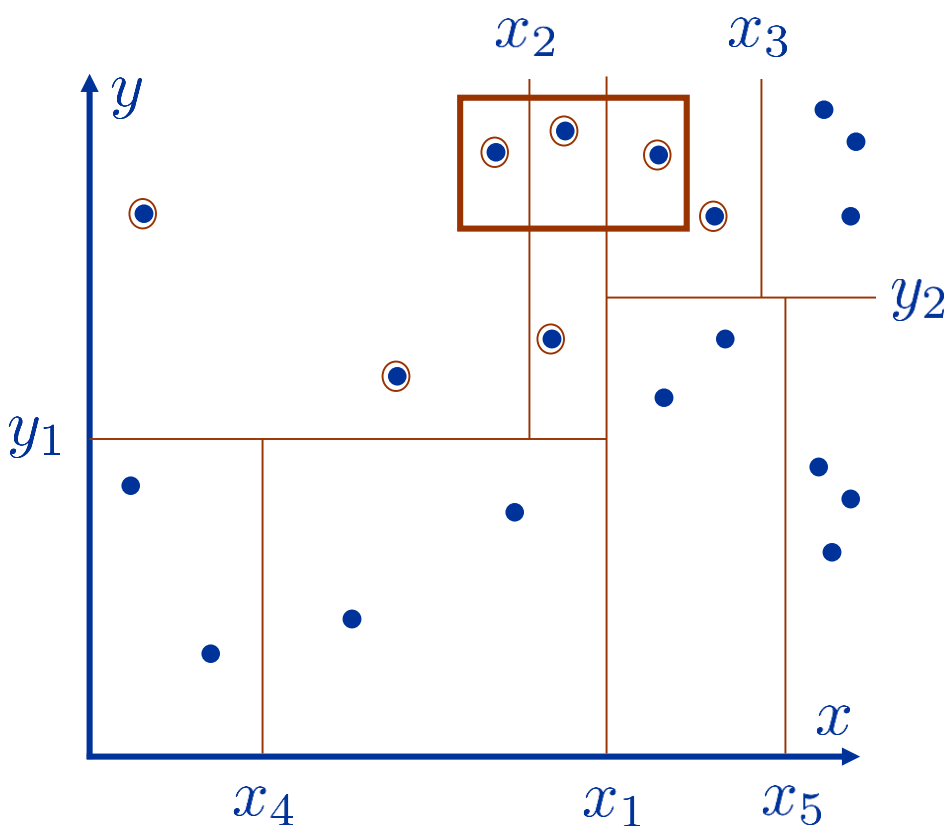
k-d Tree – 2-d Example

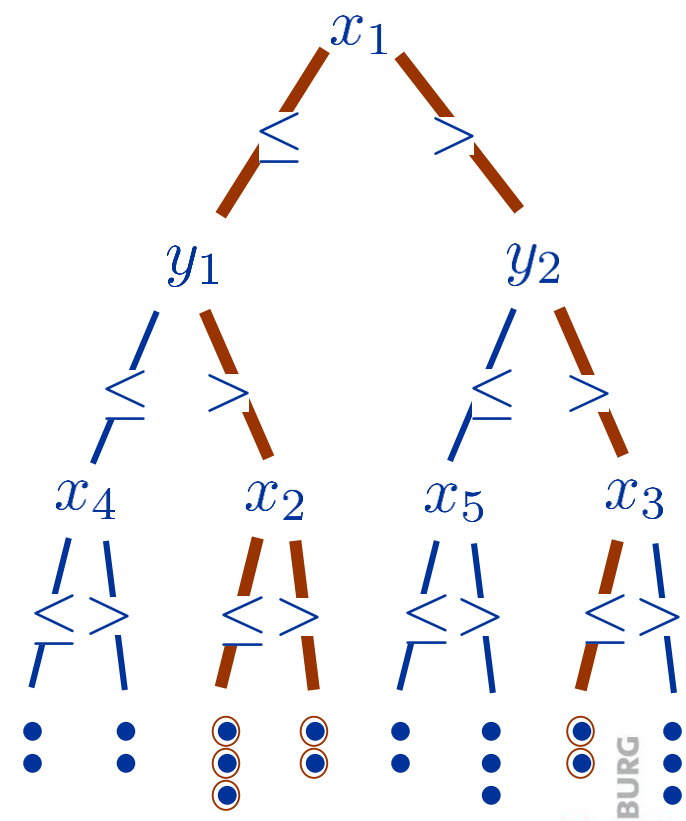




Collision Query (Range Query)

- traverse all nodes affected by the intervals of an AABB
- check all primitives ⊙ in the leaves for intersection



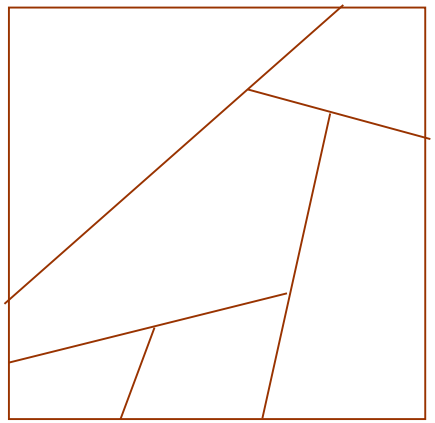


Outline

- introduction
- uniform grid
- Octree and k-d tree
- BSP tree

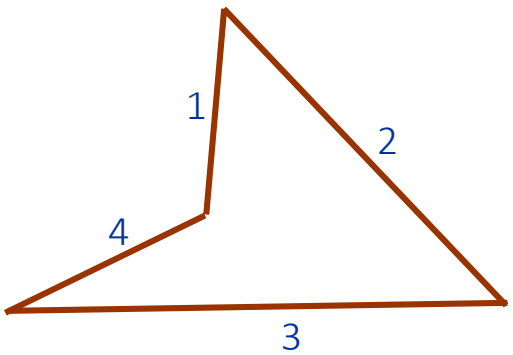
Binary Space Partitioning Tree BSP

- hierarchical structure
- space is subdivided by means of arbitrarily oriented planes
- generalized k-d tree
- space partitioning into convex cells
- discrete-orientation BSP trees DOBSP (finite set of plane orientations)
- proposed by [Henry Fuchs et al. 1980] to solve the visible surface problem

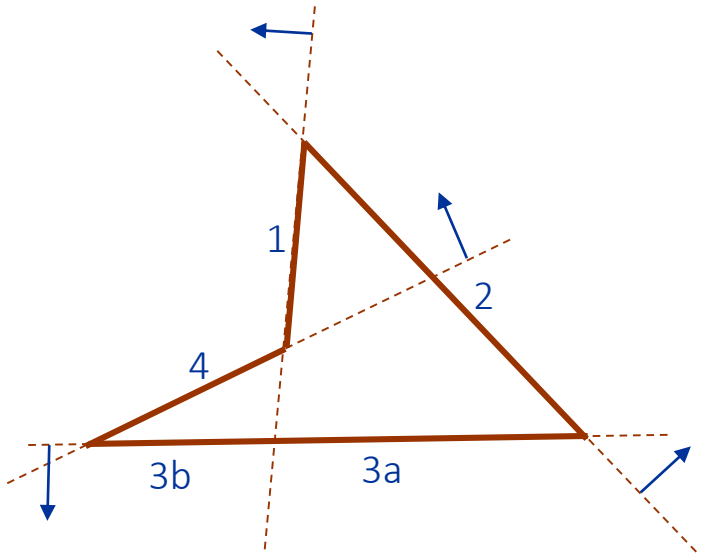


Collision Detection Example

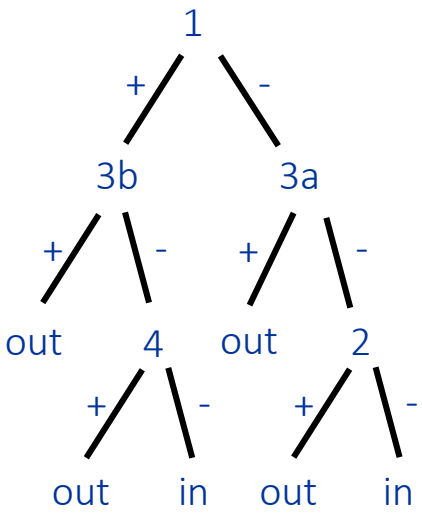
- BSP trees can be used for the inside / outside classification of closed polygons



scene



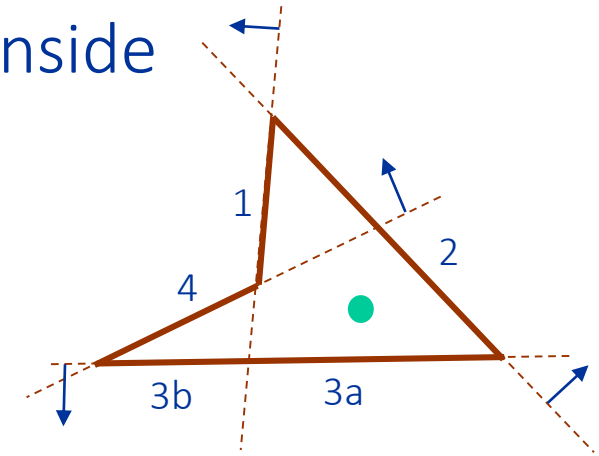
scene partitioning

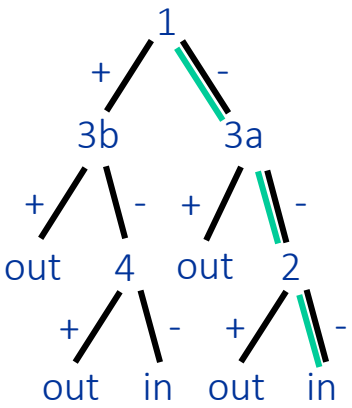


solid-leaf BSP tree

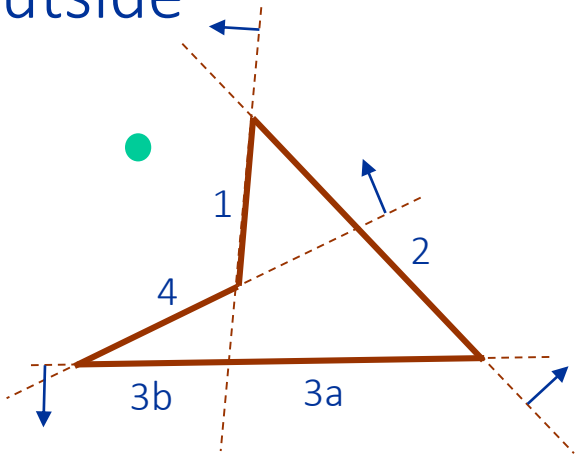
Collision Query

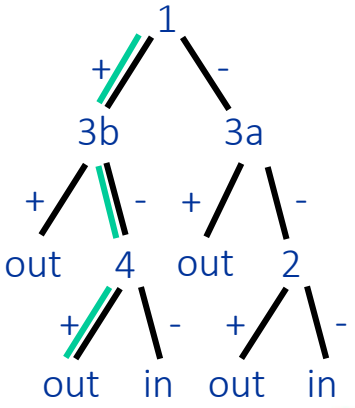
- query point is inside



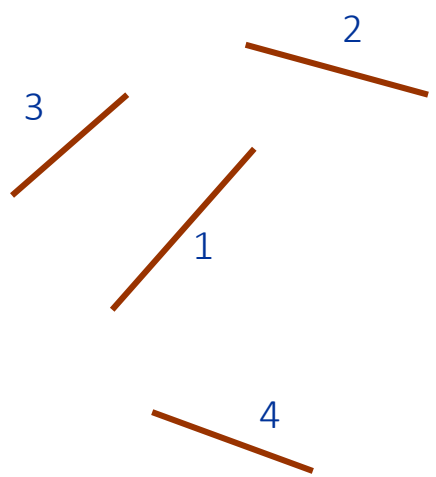


- query point is outside

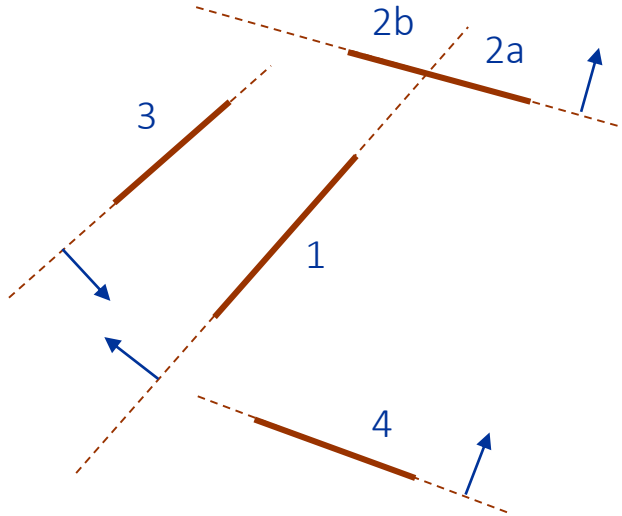




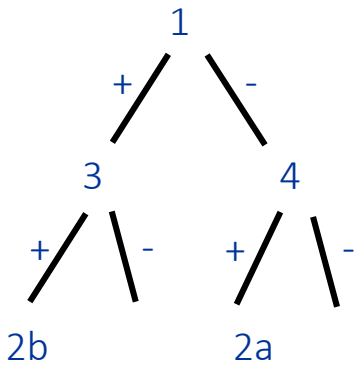
Rendering Example



scene



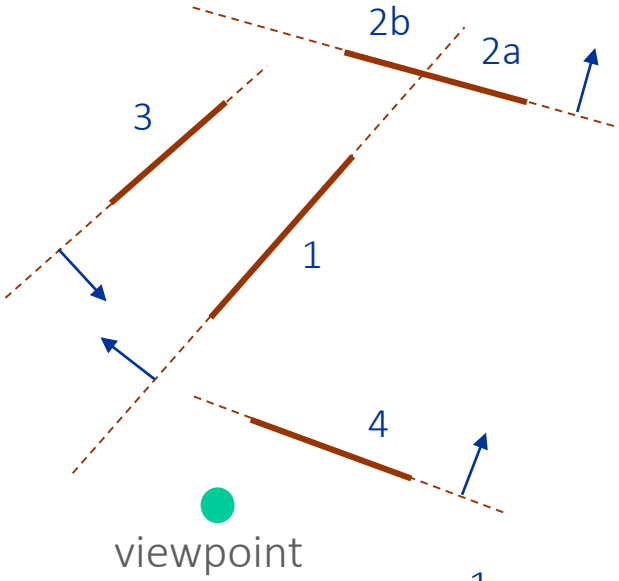
scene partitioning

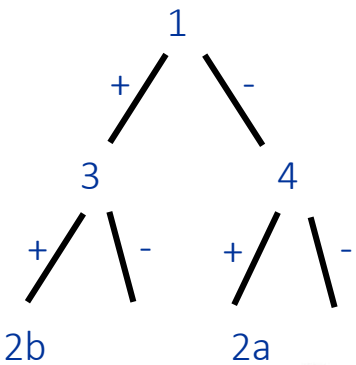


BSP tree

BSP Tree for Rendering

- for a given viewpoint
  - render far branch
  - render root (node) polygon
  - render near branch
- recursively applied to sub-trees
- back to front rendering
- example: viewpoint is in 1-
- rendering of 1+, 1, 1-
- rule recursively applied to 1+ and 1-
- viewpoint is in 3+ ⟶ rendering of 3, 2b
- viewpoint is in 4- ⟶ rendering of 2a, 4





BSP tree

Construction

- keep the number of nodes small
- keep the number of levels small
- introduce arbitrary support planes (especially in case of convex objects, where all polygon faces are in the same half-space with respect to a given face)

Summary

- uniform grid
- Octree and k-d tree
- BSP tree

References

- **C. Levinthal**, "Molecular model-building by computer," Scientific American, pp. 42-52, June 1966.
- **J. L. Bentley**, D. F. Stanat, E. H. Williams, "The complexity of fixed-radius near neighbor searching," *Inf. Process. Letters*, vol. 6, 209-212, 1977.
- **G. Turk**, "Interactive collision detection for molecular graphics," TR90-014, University of North Carolina at Chapel Hill, 1990.
- **S. Bandi**, D. Thalmann, "An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies," *Proc. of Eurographics*, pp. 259-270, 1995.
- **A. Gregory**, M. Lin, S. Gottschalk, R. Taylor, "*H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction*," TR98-032, University of North Carolina at Chapel Hill, 1998.
- **S. Melax**, "Dynamic plane shifting BSP traversal," *Proc. Graphics Interface*, pp. 213-220, 2000.
- **M. Teschner**, B. Heidelberger, M. Mueller, D. Pomeranets, M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," *Proc. Vision, Modeling, Visualization VMV'03*, pp. 47-54, Nov 2003.