Simulation in Computer Graphics
Space Subdivision

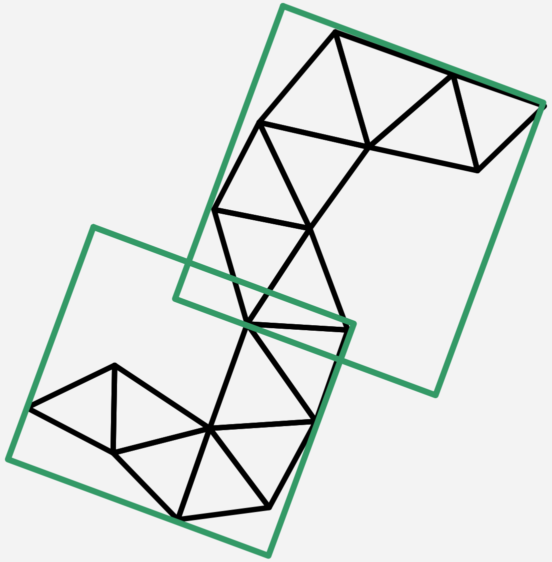
Matthias Teschner

UNI FREIBURG

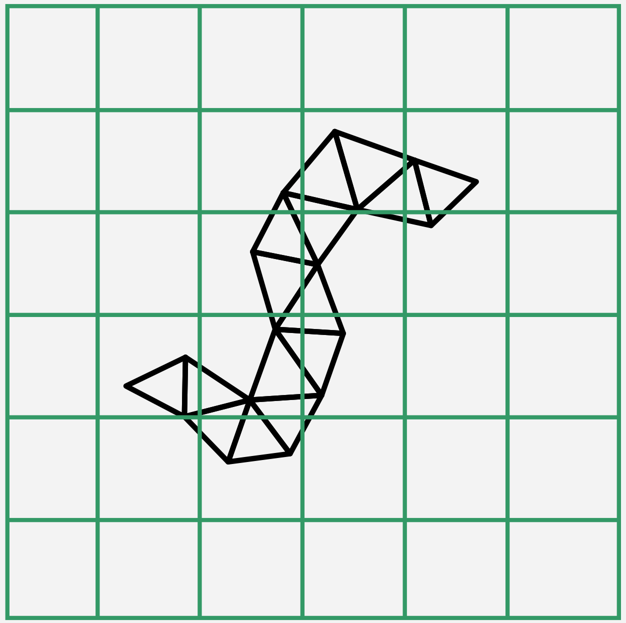
Outline

- Introduction
- Uniform grid
- K-d tree
- BSP tree

Model vs. Space Partitioning



Model partitioning

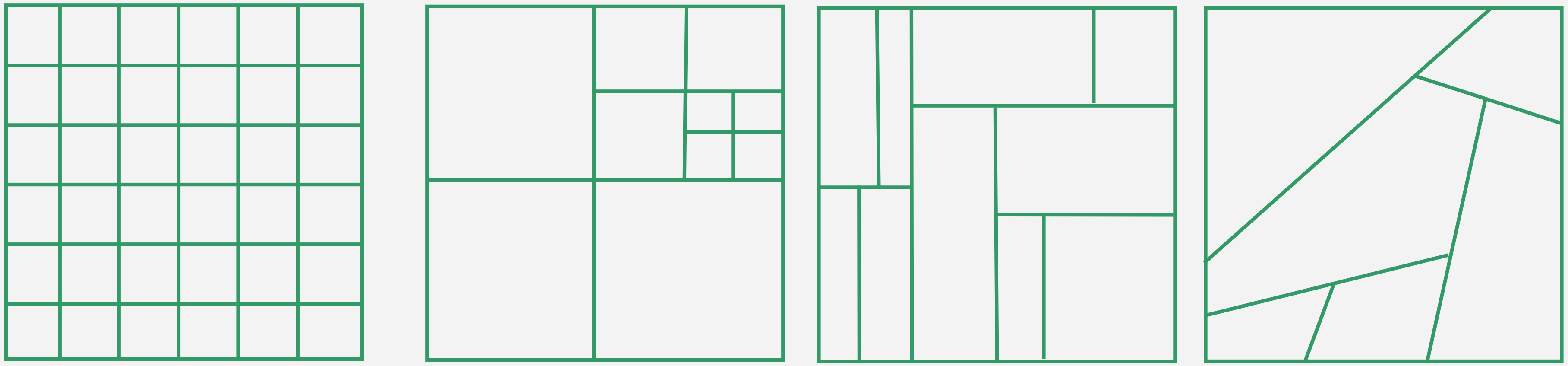


Space partitioning

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

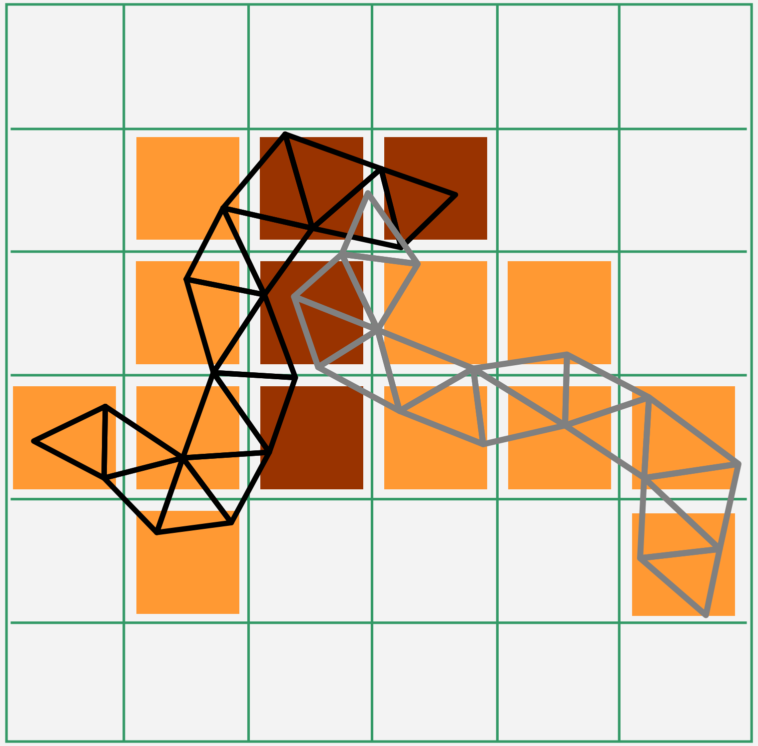
- Space is subdivided into cells
- Cells maintain references to primitives intersecting the cell
- Data structures have different degrees-of-freedom
- Actual space subdivision is adapted to the scene

Outline

- Introduction
- Uniform grid
- K-d tree
- BSP tree

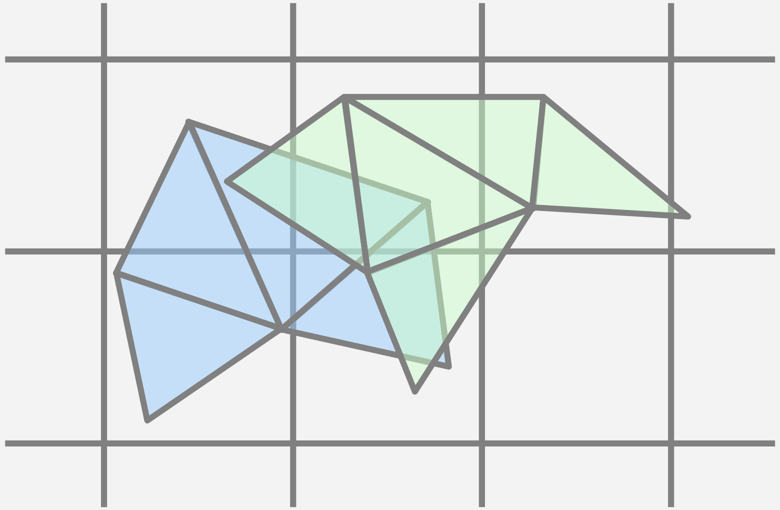
Basic Idea

- Space is divided into cells
- Object primitives are placed into cells
- Object primitives in the same cell are checked for collision
- Pairs of primitives that do not share the same cell are not tested (trivial reject)



Implementation - Setup

Infinite uniform grid

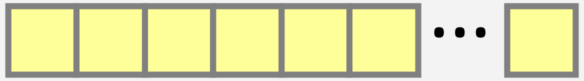


Spatial data structure

Hash function:

H(cell) → hash table index

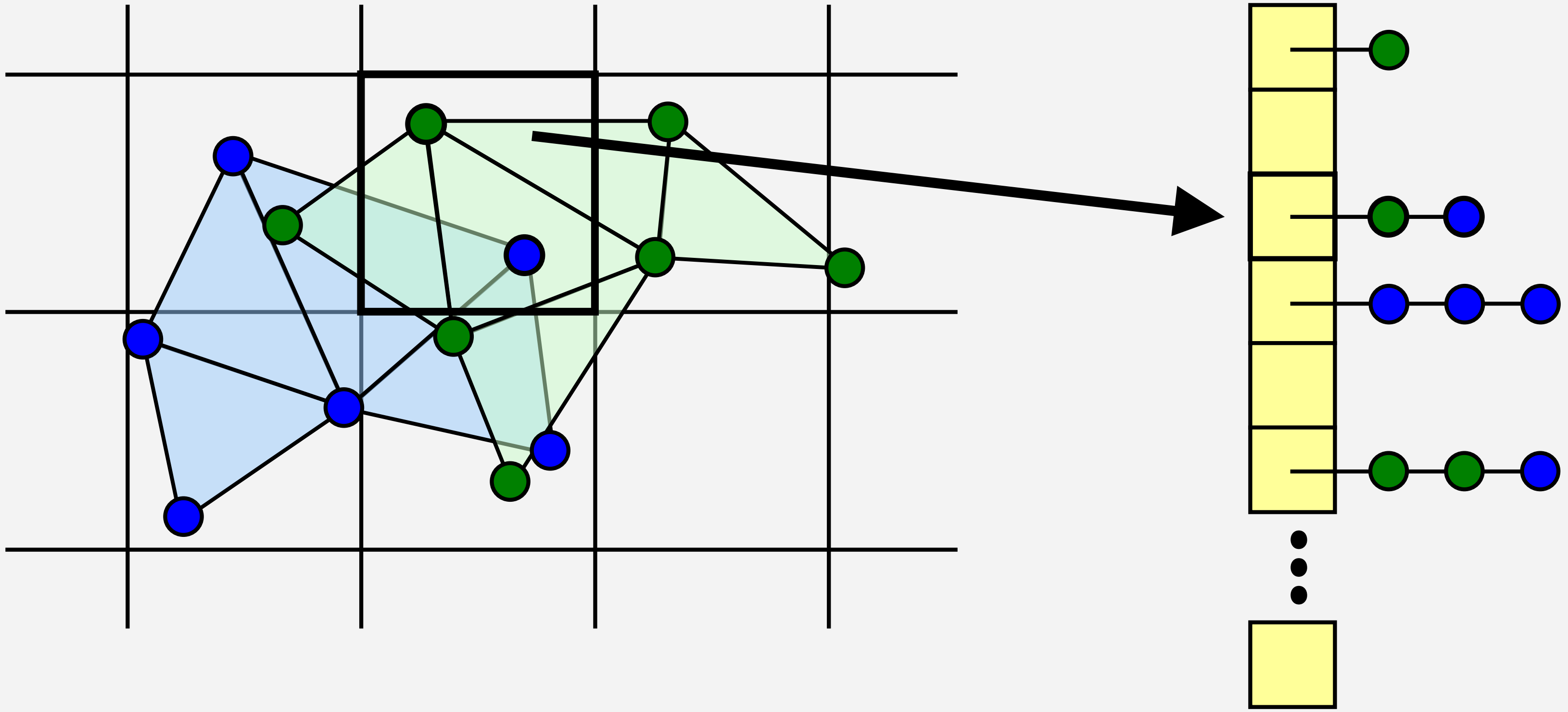
Hash table



Representation / implementation

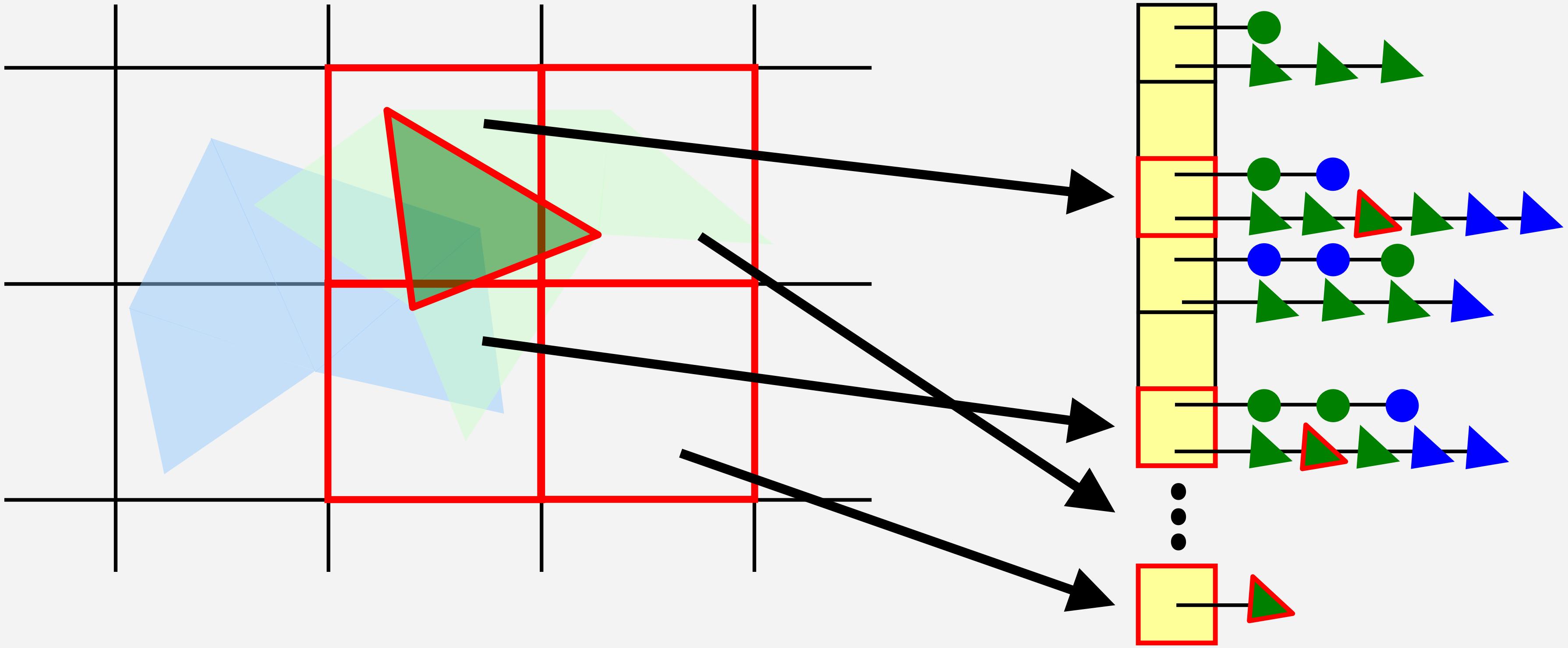
Implementation - Stage 1

– All vertices are hashed according to their cell



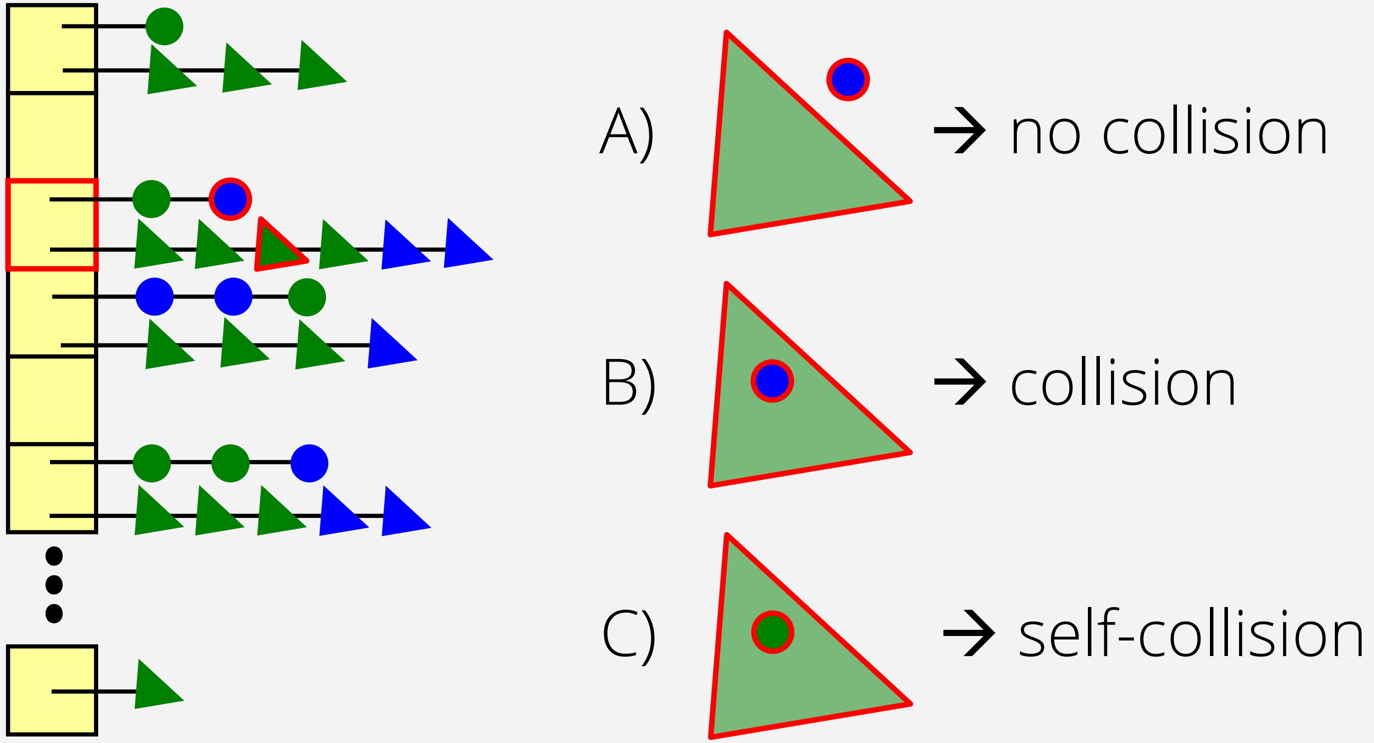
Implementation - Stage 2

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

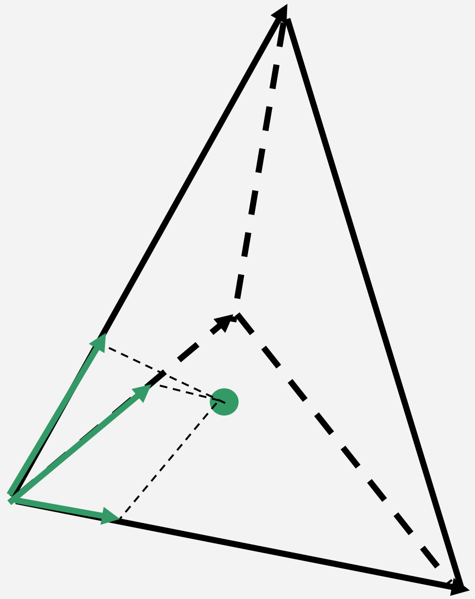


Implementation - Stage 3

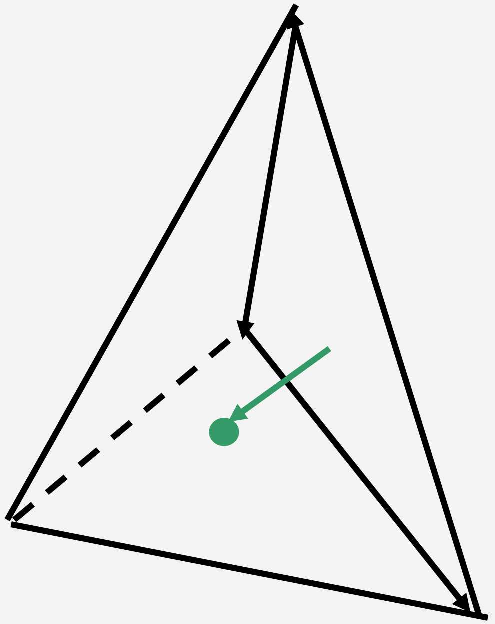
– Vertices and tetrahedrons in the same hash table entry are tested for intersection



Vertex-in-Tetrahedron Test



Barycentric coordinates



Oriented faces

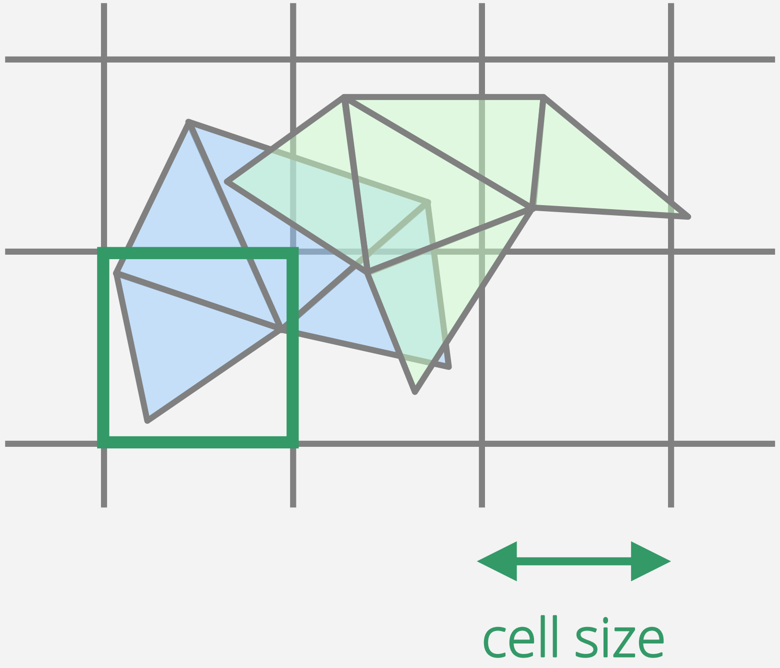
- Barycentric coordinates more efficient
- They also provide useful collision information

Implementation - Summary

- 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, hash table size, hash function

Parameters

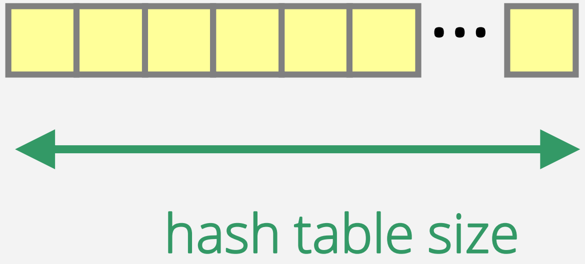
Infinite uniform grid



Hash function:

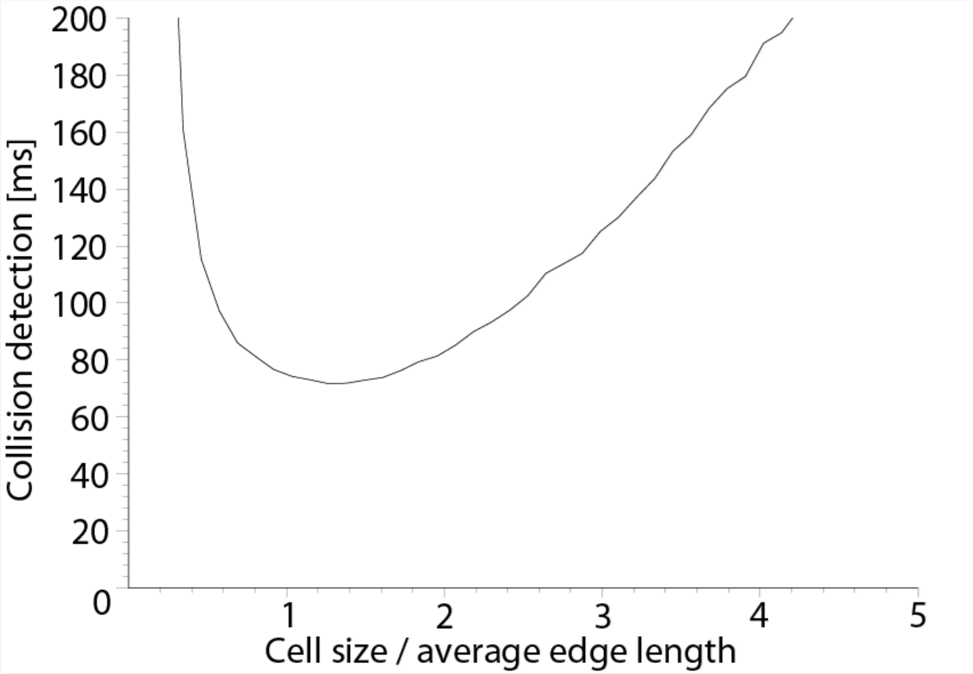
H(cell) → hash table index

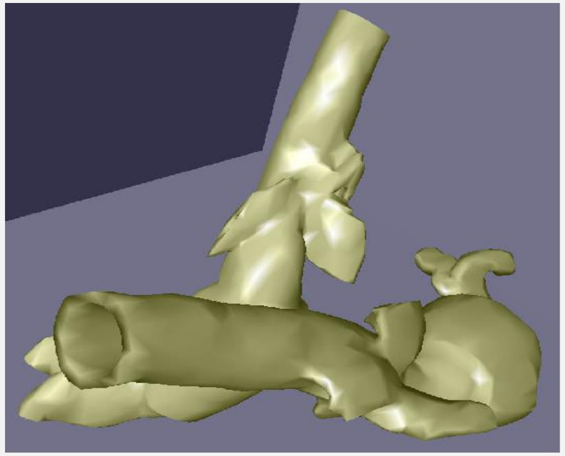
Hash table



Grid Cell Size

– Cell size should be equal to the size of the bounding box of an object primitive [Bentley 1977]



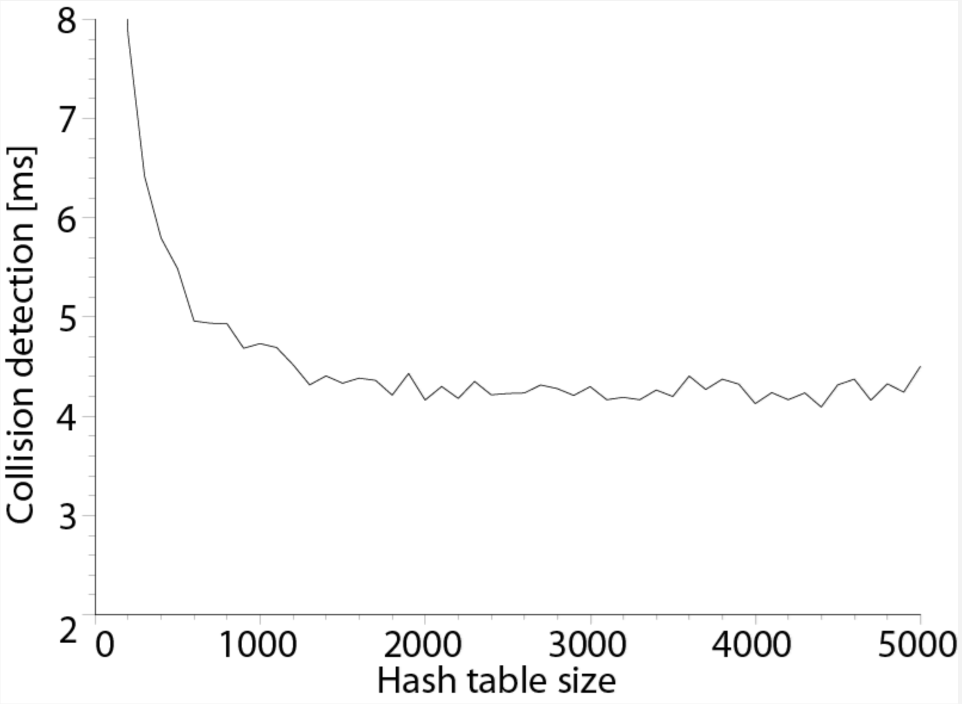


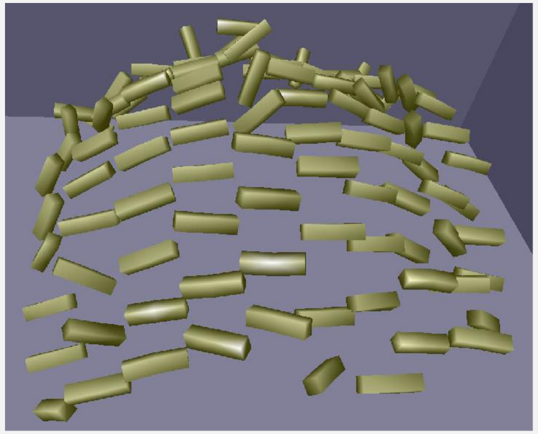
test scenario

[Teschner, Heidelberger et al. 2003]

Hash Table Size

– Hash collisions reduce the performance
– Larger hash table can reduce hash collisions





test scenario

[Teschner, Heidelberger et al. 2003]

Hash Function

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

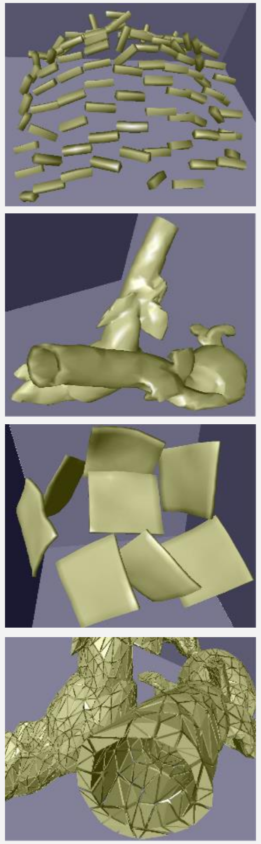
$$H(x, y, z) = (p_1 \cdot x \;\; \text{xor} \;\; p_2 \cdot y \;\; \text{xor} \;\; p_3 \cdot z) \;\; \text{mod} \;\; n$$

- Cell identifier: 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 memory-efficient

Summary – Uniform Grid

- 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

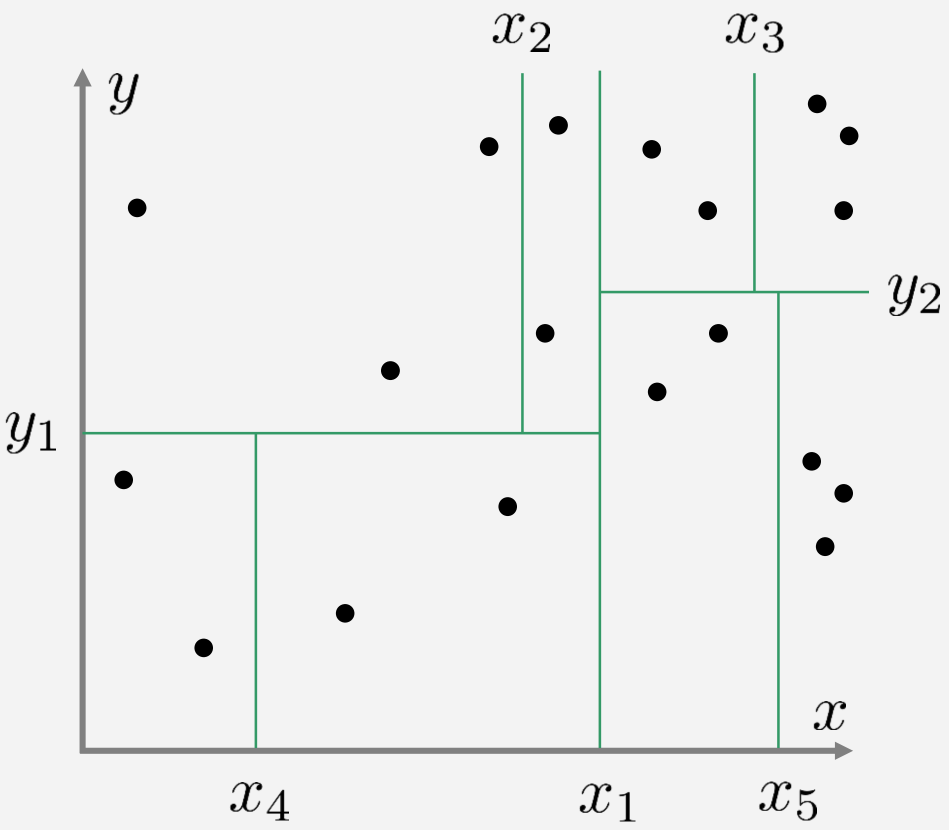
UNI FREIBURG

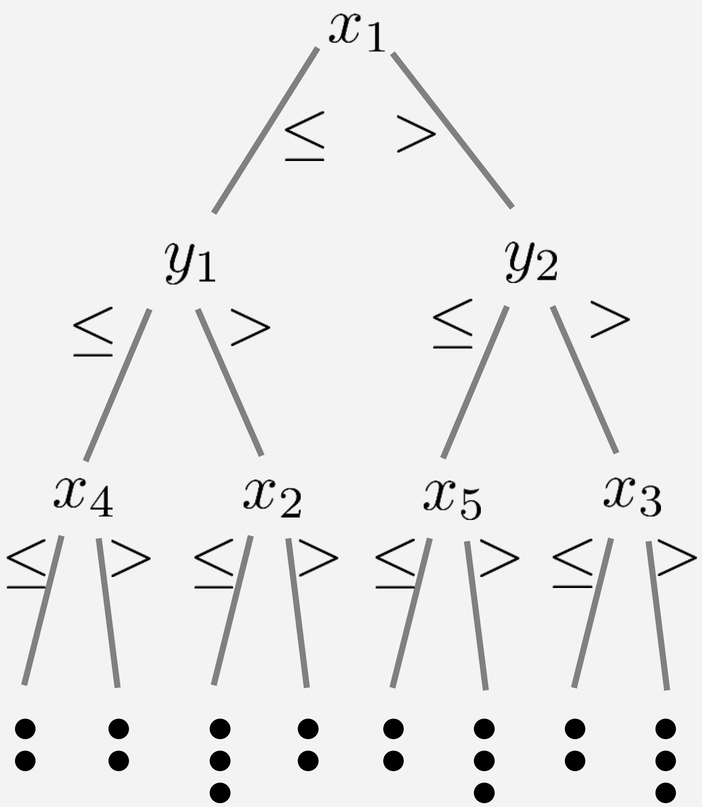
Outline

- Introduction
- Uniform grid
- K-d tree
- BSP tree

UNI FREIBURG

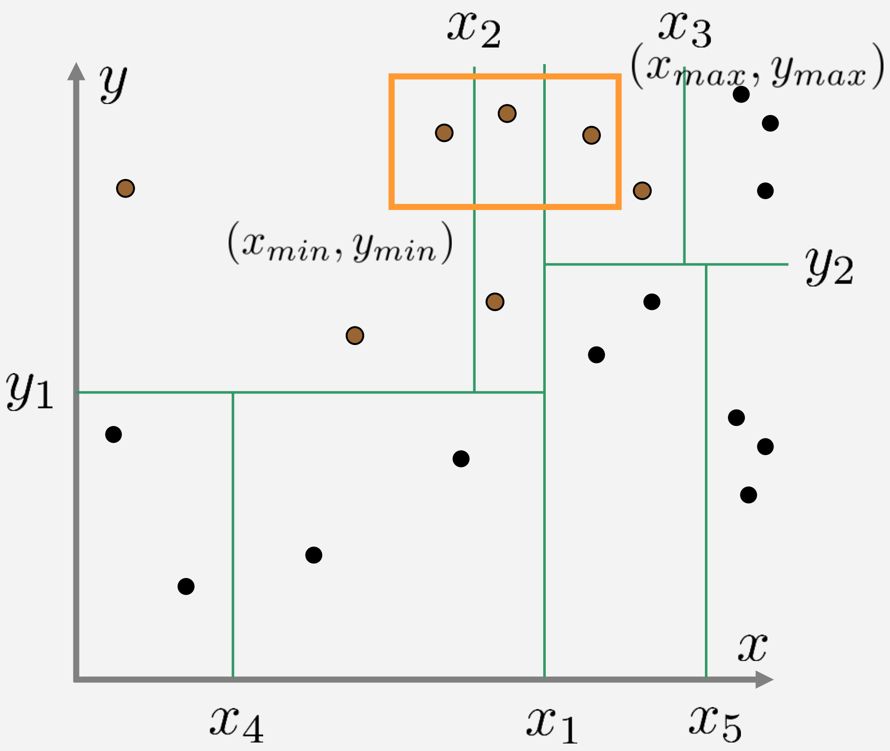
k-d Tree – 2-d Example

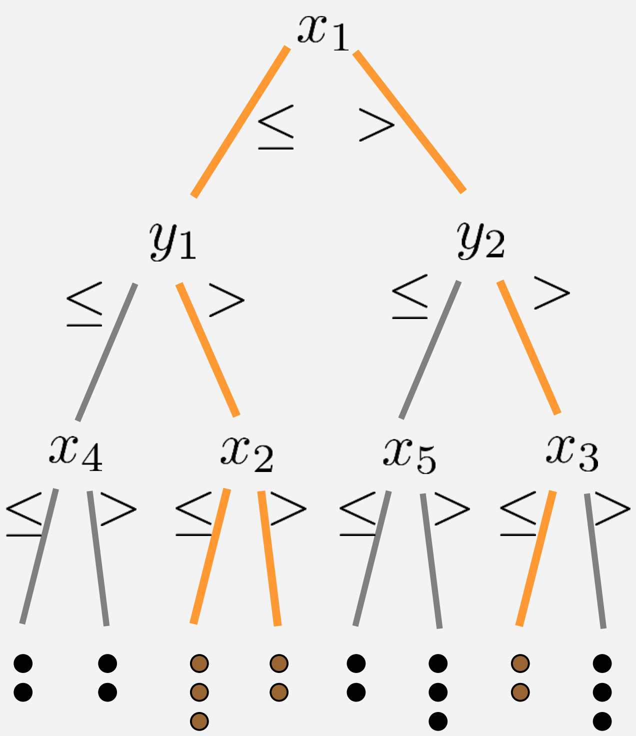




Collision Query (Range Query)

– Traverse all nodes affected by the intervals of an AABB
– Check all primitives ● in the leaves for intersection





Outline

- Introduction
- Uniform grid
- K-d tree
- BSP tree

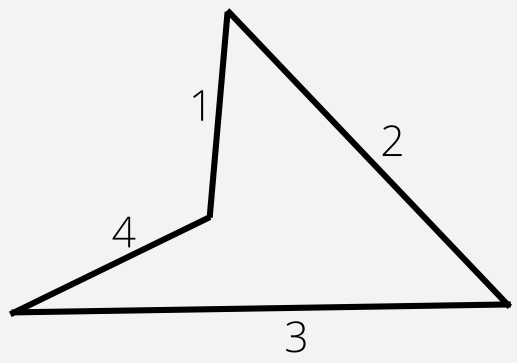
Binary Space Partitioning Tree BSP

- Generalized k-d tree
- Space is recursively subdivided by means of arbitrarily oriented planes
- Space partitioning into convex cells
- Proposed by [Henry Fuchs et al. 1980] to solve the visible surface problem

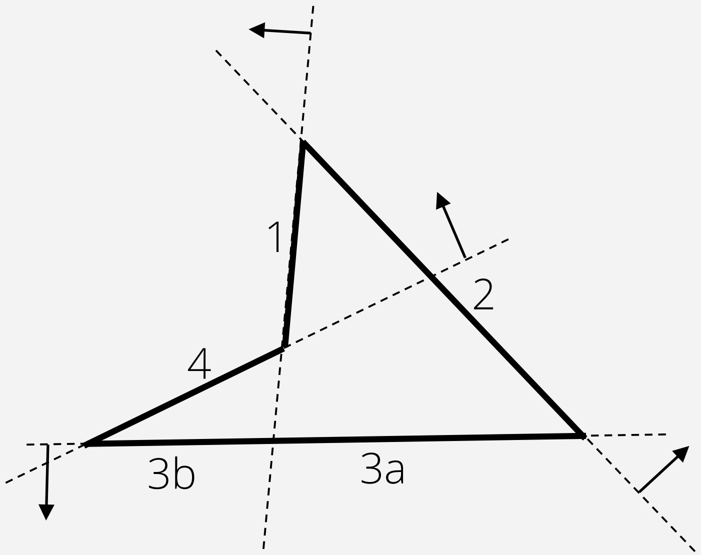
BSP tree

Collision Detection Example

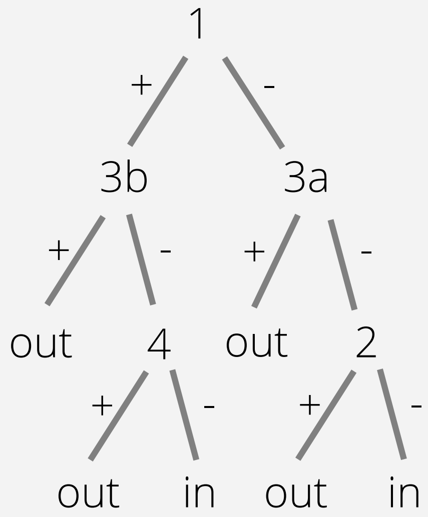
– BSP trees can be used for the inside / outside classification of closed polygons



Scene



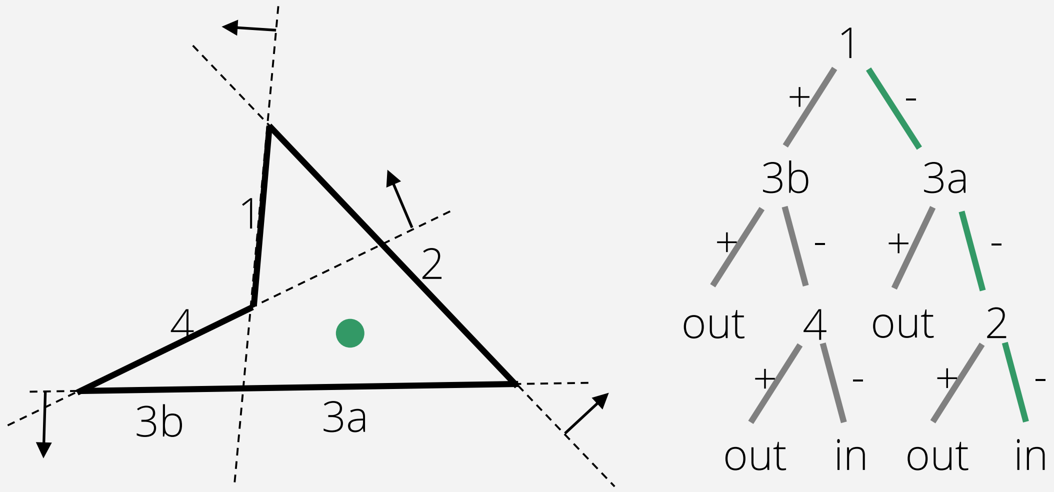
Scene partitioning



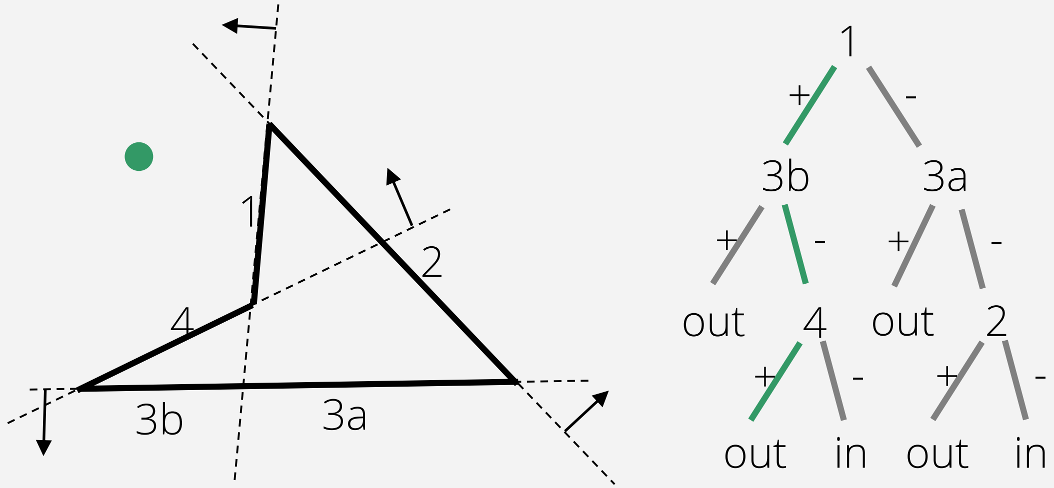
Solid-leaf BSP tree

Collision Query

– Query point is inside



– Query point is outside



Construction

- Keep the number of nodes small
- Keep the number of levels small