Simulation in Computer Graphics
Space Subdivision

Matthias Teschner
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

- 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

Hash function:
\[ H(\text{cell}) \rightarrow \text{hash table index} \]

Hash table

Spatial data structure

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

- A) ➞ no collision
- B) ➞ collision
- C) ➞ self-collision
Vertex-in-Tetrahedron Test

- 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:

\[ H(\text{cell}) \rightarrow \text{hash table index} \]

Hash table

- cell size
- hash table size
Grid Cell Size

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

[Graph: Collision detection time vs. cell size / average edge length]

[Teschner, Heidelberger et al. 2003]
Hash Table Size

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

[Teschner, Heidelberger et al. 2003]
# Hash Function

- Should avoid hash collisions
- Should be efficient (has to be computed for all primitives)
  \[ H(x, y, z) = (p_1 \cdot x \text{ xor } p_2 \cdot y \text{ xor } p_3 \cdot z) \mod n \]
- 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

<table>
<thead>
<tr>
<th>Objects</th>
<th>Tetras</th>
<th>Vertices</th>
<th>Max time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1000</td>
<td>1200</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>4000</td>
<td>1936</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>10000</td>
<td>4840</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>20514</td>
<td>5898</td>
<td>72</td>
</tr>
<tr>
<td>100</td>
<td>50000</td>
<td>24200</td>
<td>174</td>
</tr>
</tbody>
</table>

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
Outline

- Introduction
- Uniform grid
- K-d tree
- BSP tree
k-d Tree – 2-d Example

- The diagram illustrates a 2-dimensional k-d tree, where the tree splits the space into regions based on the values of the dimensions.
- The tree structure is depicted with nodes labeled by the coordinates, and the regions are divided by comparing the coordinates with the split values.
- The left side of the diagram shows the set of points plotted on the 2-dimensional plane, with axes for x and y.
- The right side of the diagram represents the tree structure with nodes and split conditions.

University of Freiburg – Computer Science Department – 22
Collision Query (Range Query)

- Traverse all nodes affected by the intervals of an AABB
- Check all primitives \( \bullet \) 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
Collision Detection Example

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

- Query point is inside

- Query point is outside
Construction

– Keep the number of nodes small
– Keep the number of levels small