Computer Graphics Lecture 10: Spatial Data Structures

DR. ELVIS S. LIU

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

SPRING 2018

Spatial Decomposition

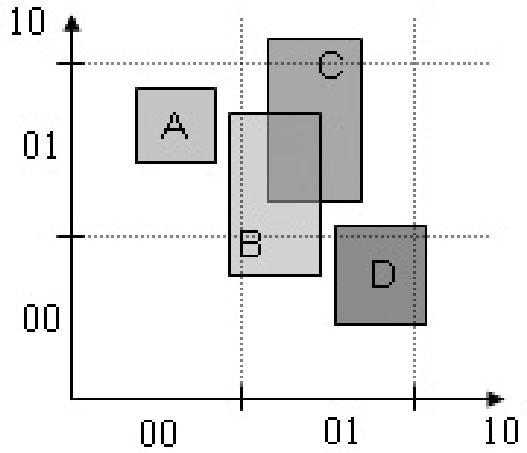
LECTURE 10: SPATIAL DATA STRUCTURES

Spatial Decomposition Methods

- Decompose the space into regions (or “cells”)
- Pairwise comparisons for each cell
- First Phase Complexity: dependent on the method, e.g. O(N) for spatial hashing
- Second Phase Complexity: $O(N_r^2)$ for N_r pairs in each cell
- Ideally, most of the cells contain only one object

Example – Spatial Hashing in 2D

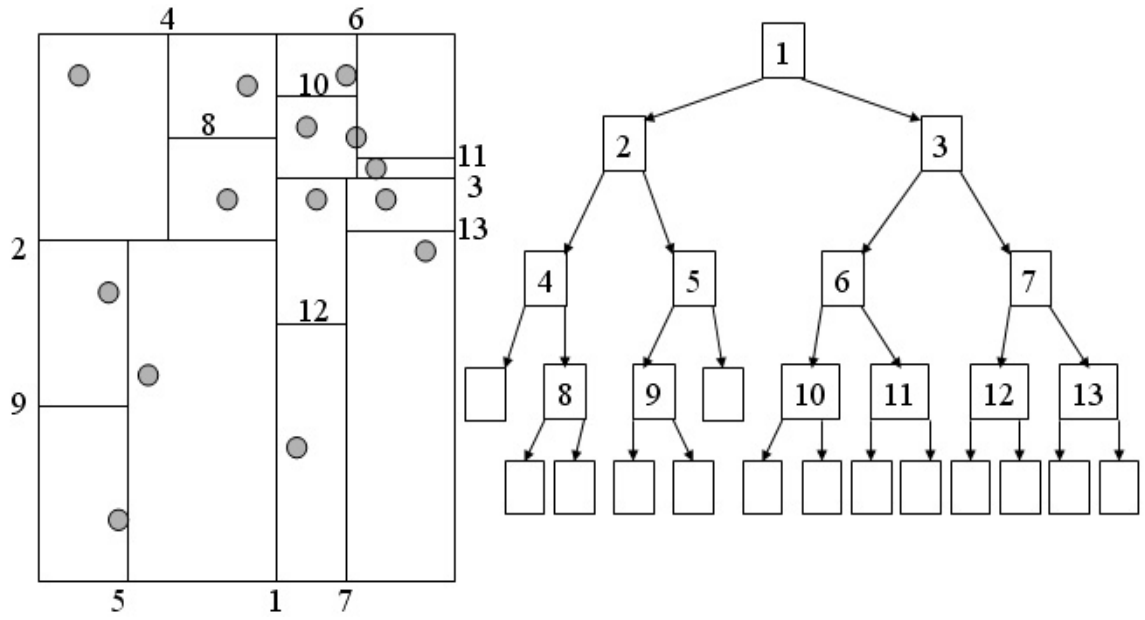
- Object A is hashed into 0100
- Object B is hashed into 0100, 0000, 0101 and 0001
- Object C is hashed into 1001, 0101
- Object D is hashed into 0101, 0110, 0001, 0010
- A-B, B-C, B-D, C-D are all potential collisions



Properties of Spatial Hashing

- Collision query for one object can be processed in O(1)
- Crowded subdivisions => number of potentially collided pairs becomes large
- What is the worst case?
  - All objects reside in a single subdivision
- What is the time complexity for worst case?
  - O (N^2)

Example: Hierarchical Structure (K-D Tree) in 2D



Properties of Hierarchical Structures

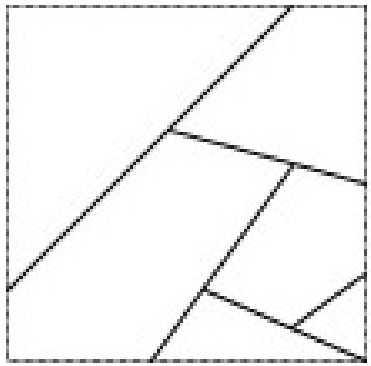
- A collision query is O(log N) in the average case
- What is the average case?
  - A balanced tree
- What is the time complexity of collision queries for N objects in the average case
  - O (N log N)
- In the worst case, collision queries for N objects could become O (N^2)
- What is the worst case?
  - The height of the tree = the number of objects

Reconstruction of the Tree

- Objects move from time to time
- Reconstruction of the Tree is required
- Insert an object into the tree takes O(log N) time in the average case
- Total reconstruction might happen, which takes O(N log N) time in the average case
- Hierarchical structures are efficient for largely static scenes, but inefficient for scenes with multiple fast moving objects

Binary Space Partitioning (BSP) Tree

- BSP tree recursively partitions a space into two subdivisions using a partition plane
- Partition plane can be free chosen
  - Easy to keep the tree balanced
- Partition query is relatively complicated



BSP Tree Construction

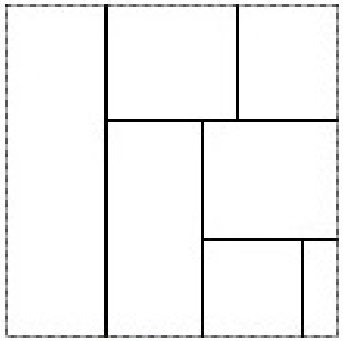
- Step 1: Choose a plane to partition the virtual space into two subdivision, each contains equal number of objects
- Step 2: Recursively partition each of the subdivisions into two, until the subdivision contains only one object
- There are other ways to construct the BSP tree, counting the number of objects is just one of the approaches.

BSP Tree collision query for one object

- Step 1: Traverse the BSP tree from its root node
- Step 2: Check whether the space of left subtree contains this object. If this is the case, traverse recursively the left subtree
- Step 3: Check whether the space of right subtree contains this object. If this is the case, traverse recursively the right subtree
- Step 4: When a leaf node is reached, check whether the objects it contains collide with the object in question

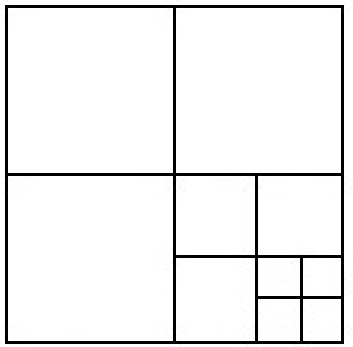
K-Dimensional (K-D) Tree

- 2-D Tree in this case
- K-d tree recursively partitions a space into two subdivisions using a partition plane
- Partition planes must be axis-aligned
  - More difficult to keep the tree balanced than the BSP tree
- Collision queries are simpler than BSP tree



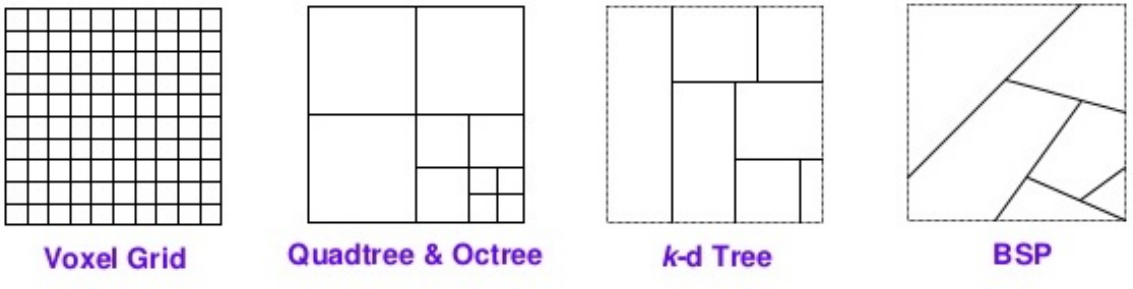
Quadtree (2D) or Octree (3D)

- Quadtree in this case
- A quadtree recursively partitions a space into four uniform subdivisions using two axis-aligned partition planes
- An octree recursively partitions a space into eight uniform subdivisions using three axis-aligned partition planes
- More difficult to keep the tree balanced than BSP tree and k-d tree
- Collision queries are simpler than BSP tree and k-d tree



Hierarchical Structures

- K-D tree is a subset of BSP tree
- Quadtree (or octree) is a subset of K-D tree
- Uniform grid (a special case) is a subset of quadtree (or octree)



Granularity Problem

- Difficult to choose optimal cell size (or tree height)
- Large cell size (or small tree height) =>
  - Many objects reside in a cell
  - More pairwise comparisons
- Small cell size (or large tree height) =>
  - More cells require collision detection
- Essentially, this becomes a trade-off problem

Ray Tracing Acceleration

LECTURE 10: SPATIAL DATA STRUCTURES

Motivations

- Lots of things can slow down the rendering process
  - Many objects in the scene
  - Larger viewport
  - Complicated rendering techniques
- In non-trivial applications we can have very crowded scenes, so how can we quickly scan every object for rendering as our scenes increase in size?

Example - Octree

- Begin at the root node
- If node is a leaf node, perform intersection on all of its primitives
- Otherwise the node contains child voxels, so iterate through the node's children and compute intersection between ray and each child's bounding box
  - If it intersects, recur on child again