CIT2213: Game Engine Architecture

Lecture: Spatial Partitioning

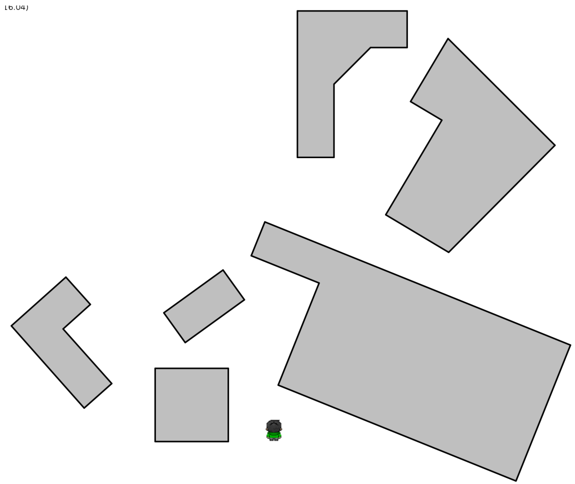
Dr Minsi Chen

Computer Science

Overview

- Spatial partitioning
- Uniform and adaptive spatial partitioning
- Binary spatial partitioning
- k-d tree

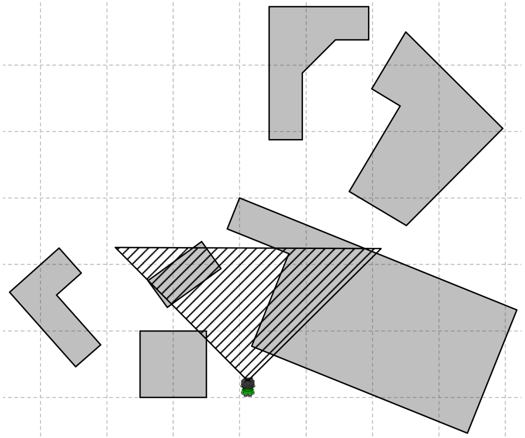
A Scene



Scene Query

Visibility
: Which objects/polygons are visible from the current view position?
    - Ray intersection test
    - View frustum test

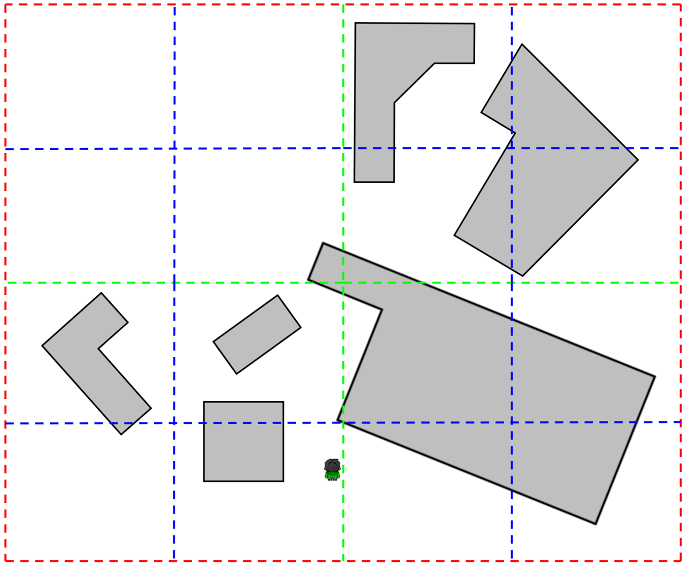
Collision
: Which objects/polygons are the closest to me?
    - Distance calculation
    - Intersection test



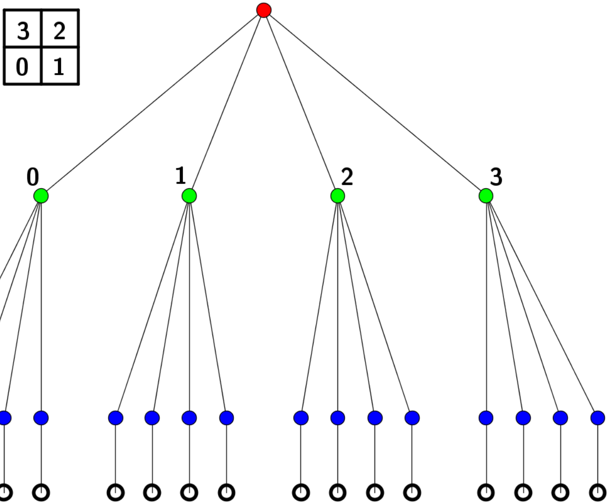
Spatial Partitioning

Concept	• A space is divided in to small non-overlapping regions. • Objects are then assigned into each region depending on their occupancy. • Each region is defined using simple geometries, e.g. an AABB or a plane.
Motivation	• Allows the use of much simpler tests to calculate visibility and intersection • Minimises the amount of geometries to be tested

Subdividing the Space



Representing Space Subdivision As A Tree



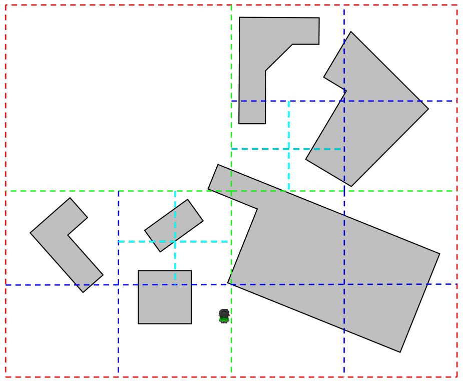
The Anatomy of a Search Tree

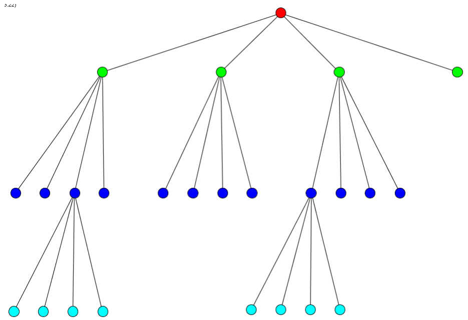
Non-leaf Node	Each non-leaf node contains a comparable “key”; incoming data are compared with the value of the key in order to determine the branch to descend; e.g the key for an octree node is the bounding box.
Leaf Node	A leaf node commonly contains data, e.g. polygons and objects
Branch	Each branch satisfies a condition related to the key in the parent node; e.g. each branch in an octree lies entirely *inside* its parent.
Partitioning	Each branch can be further partitioned if certain criteria is not met, i.e. the number of objects is greater than a threshold.
Descent	Perform a comparison with the key to determine which branch to descend; e.g. inside the bounding box.

Quadtrees/Octrees

Node	A node stores the bounding box of a region as the key, e.g. an AABB.
Partitioning	If the number of objects inside the region is greater than a set threshold, the region is further subdivided.
Descent	An incoming object is tested against the bounding box and we only descend nodes containing the object.

Alternative Partition

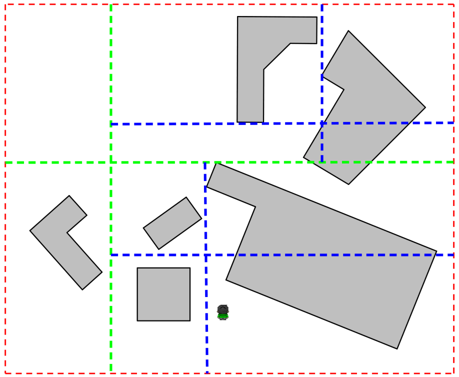




Uniform versus Adaptive

Uniform Each cell is subdivided into 4 or 8 sub-cells of equal dimension. It is simple to construct but lacks control on the number of objects in leaf nodes.

Adaptive The subdivision of each cell is dependent on the object distribution within the cell.



Adaptive subdivision

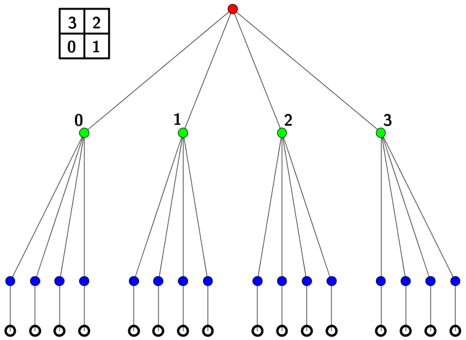
Traversing A Search Tree

Breadth First Traversal

```
void tree_traverse_BF ( Node* node, Object* o )
{
   Queue q;
   q.push_back( node );

   while queue NOT empty  {
      Node* curr_node = queue.dequeue();

      if compare o with curr_node key is true
        and node has children
          q.push_back ( curr_node->getChildren() );
   }
}
```



Binary Spatial Partitioning Tree - BSP Tree

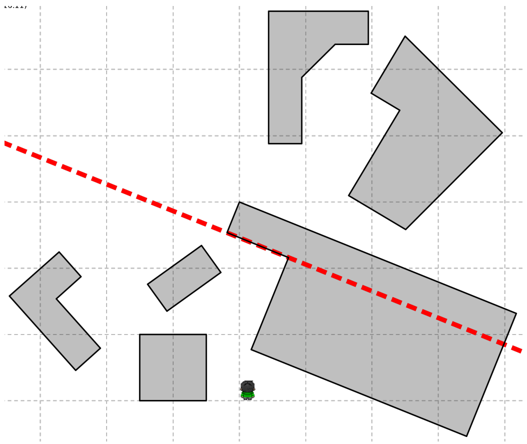
The space is partitioned by an arbitrarily oriented plane.

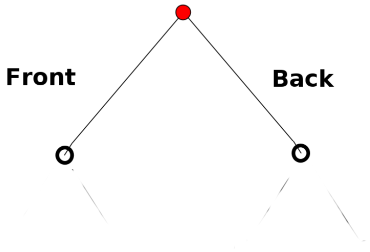
Node
: A BSP node contains an arbitrarily oriented splitting plane as the key. A plane is given by its normal $\bar{\mathbf{n}}$ and the displacement d along the normal; i.e. algebraically $ax + by + cz + d = 0$, where $\bar{\mathbf{n}} = (a, b, c)$.

Partitioning
: Objects are partitioned into *front* and *back* groups. If $ax + by + cz + d \geq 0$ the object is in front, otherwise behind.

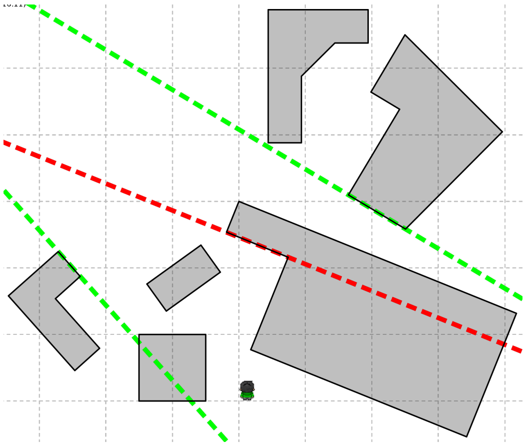
Descent
: Incoming objects are compared against the splitting plane to determine if the descent is along the front or the back branch.

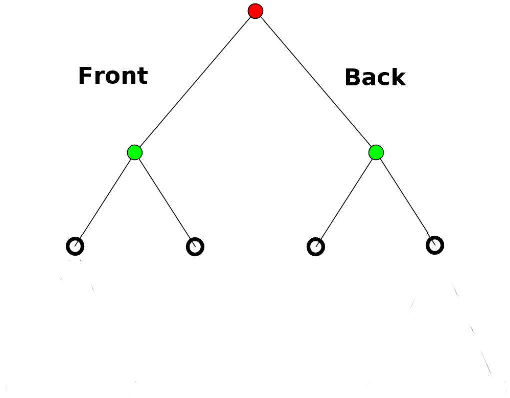
BSP - Level 1



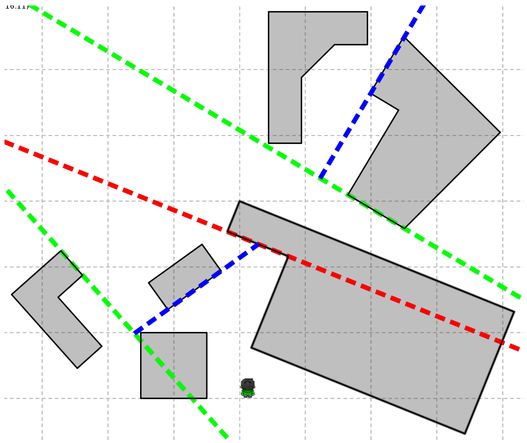


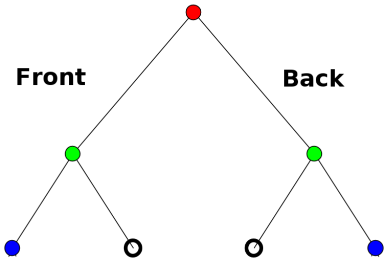
BSP - Level 2



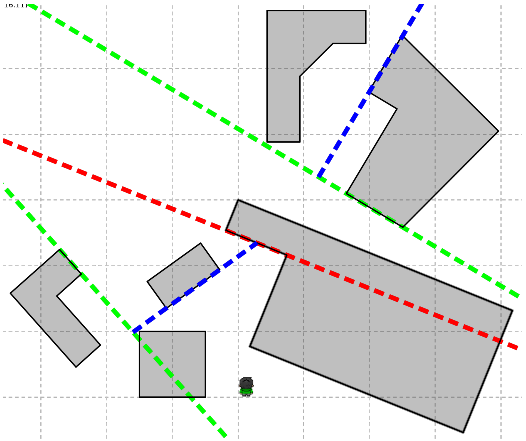


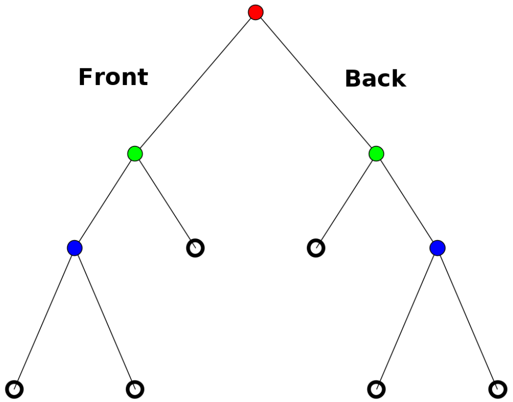
BSP - Level 3





BSP - Final





k-d Trees (Jon Bently 1975)

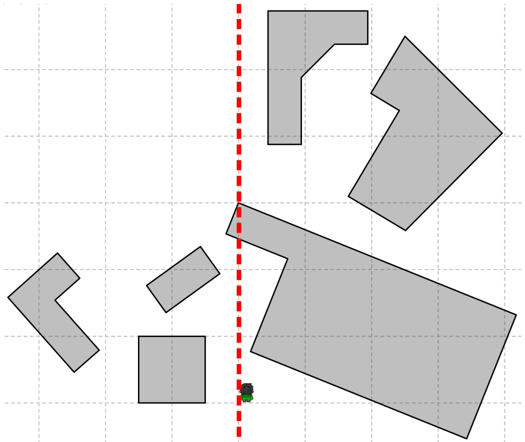
The space is partition along each principal dimension in an alternating manner. Generalised for n-dimensional data.

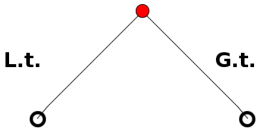
Node
: A k-d tree node contains a splitting plane perpendicular to a principal axis. The key is a pair DIM, d, i.e DIM is either x, y or z. e.g. a plane perpendicular to x-axis is $x = d$ where d is its distance from the origin.

Partitioning
: Objects are partitioned into the less than and greater than groups based on the key.

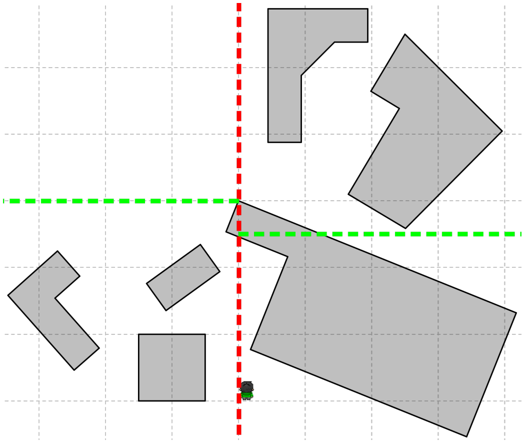
Descent
: Incoming objects are compared against the splitting plane to determine if the descent is along the less than or the greater than branch.

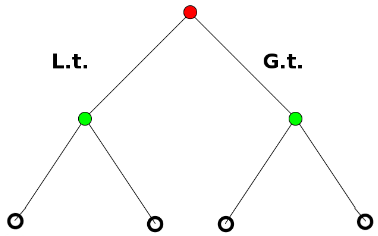
k-d Tree - Level 1



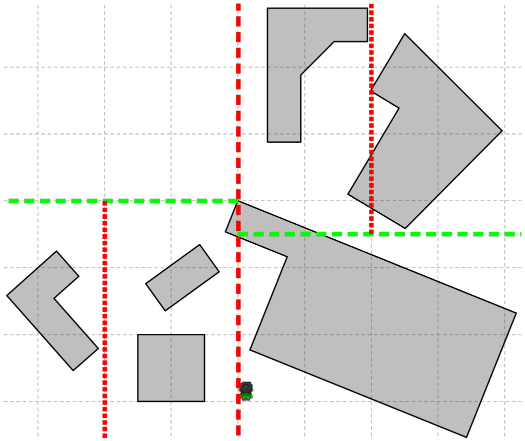


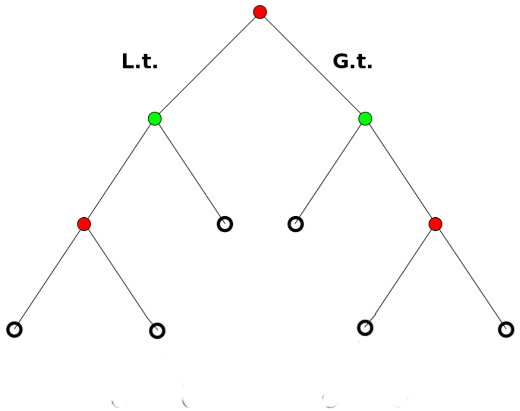
k-d Tree - Level 2



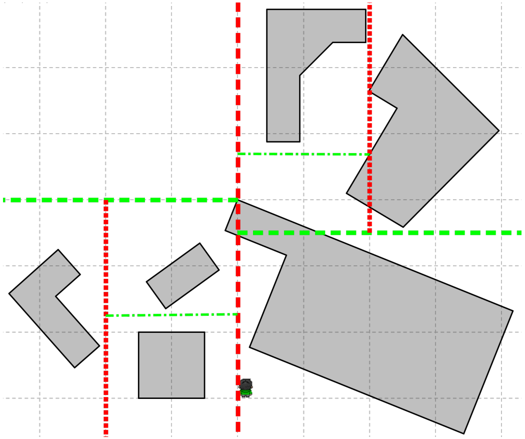


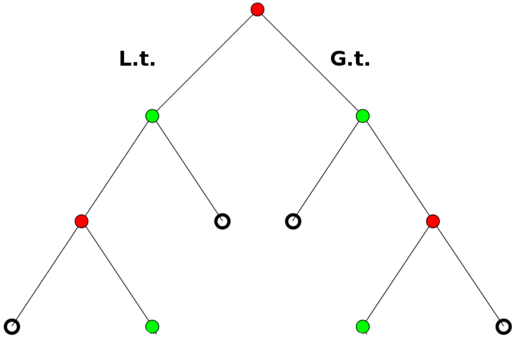
k-d Tree - Level 3



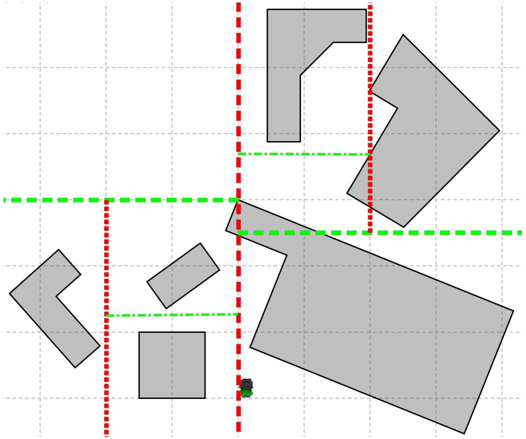


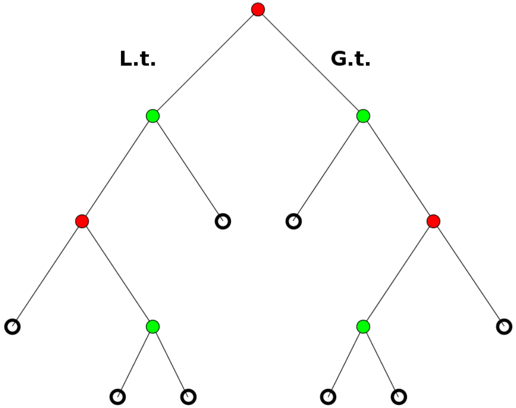
k-d Tree - Level 4





k-d Tree - Final





Applications

Spatial partitioning is applied to support the efficient run-time of various alogrithms, for examples:

- Can be applied to partition a single mesh, useful for high resolution meshes
- Fast object and/or polygon culling
- Collision query
- Support algorithms that are dependent on spatial data, e.g. ray tracing, radiosity

Some Caveats

- Memory overhead
- Requires run-time update for scene with lots of moving objects
- The choice of partitioning plane influences the depth of the tree, we should avoid hitting the worst run-time $O(N^2)$ for some algorithms.

Related Papers:

- Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. Commun. ACM 18, 9 (September 1975), 509-517. DOI=`http://dx.doi.org/10.1145/361002.361007`
- Jon Louis Bentley. 1990. K-d trees for semidynamic point sets. In Proceedings of the sixth annual symposium on Computational geometry (SCG '90). ACM, New York, NY, USA, 187-197. DOI=`http://dx.doi.org/10.1145/98524.98564`
- Naylor, B. 1993. Constructing good partitioning trees. Proceedings - Graphics Interface, 181-191.
- Hanan Samet. 1984. The Quadtree and Related Hierarchical Data Structures. ACM Comput. Surv. 16, 2 (June 1984), 187-260. DOI: `https://doi.org/10.1145/356924.356930`

Use Cases

- $NlogN$ NBody Simulation
  - J. Barnes and P. Hut 1986. "A hierarchical O(N log N) force-calculation algorithm". Nature. 324 (4): 446–449
  - Wikipedia entry: `https://en.wikipedia.org/wiki/Barnes\OT1\textendashHut_simulation`
- Procedural Generation
  - Shaker N., Liapis A., Togelius J., Lopes R., Bidarra R. (2016) Constructive generation methods for dungeons and levels. In: Procedural Content Generation in Games. Computational Synthesis and Creative Systems. Springer, Cham. (Draft: `https://graphics.tudelft.nl/Publications-new/2016/SLTLB16/chapter03.Online.pdf`
- Early commercial games:
  - DOOM - `https://github.com/id-Software/DOOM`
  - Quake - `https://github.com/id-Software/Quake`

Guidance Week - Homework

Write a console application to search the nearest point to a given point from an existing set of points, e.g.

```
Vector3 query_point(30, 40, 10);
Vector3 nearest = searchClosest(query_point);
```

- Requirements:
  - One method should use linear search based on data structure such as linked lists, array, e.g. std::vector
  - One method should use non-linear search based on spatial partitioning, e.g. BSP, KD-tree
  - You can randomly generate some existing points within a given bound, e.g. $-100 < x < 100, -100 < y < 100, -100 < z < 100$
  - Compare the timing of linear and non-linear search
  - Preferably write it in C/C++