K-D Trees

CSE 373
Data Structures
Lecture 22

Geometric Data Structures
• Organization of points, lines, planes, ... to support faster processing
• Applications
  – Astrophysical simulation – evolution of galaxies
  – Graphics – computing object intersections
  – Data compression
    • Points are representatives of 2x2 blocks in an image
    • Nearest neighbor search

k-d Trees
• Jon Bentley, 1975
• Tree used to store spatial data.
  – Nearest neighbor search.
  – Range queries.
  – Fast look-up
• k-d tree are guaranteed \( \log_2 n \) depth where \( n \) is the number of points in the set.
  – Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

Range Queries

Nearest Neighbor Search

k-d Tree Construction
• If there is just one point, form a leaf with that point.
• Otherwise, divide the points in half by a line perpendicular to one of the axes.
• Recursively construct k-d trees for the two sets of points.
• Division strategies
  – divide points perpendicular to the axis with widest spread.
  – divide in a round-robin fashion (book does it this way)
k-d Tree Construction (1)

divide perpendicular to the widest spread.

k-d Tree Construction (2)

k-d Tree Construction (3)

k-d Tree Construction (4)

k-d Tree Construction (5)

k-d Tree Construction (6)
2-d Tree Decomposition

k-d Tree Splitting

sorted points in each dimension
x 1 2 3 4 5 6 7 8 9
y 6 5 4 3 2 1 0

• max spread is the max of
- \( f \cdot a \) and \( i \cdot a \)

- In the selected dimension the middle point in the list splits the data.
- To build the sorted lists for the other dimensions scan the sorted list adding each point to one of two sorted lists.

k-d Tree Construction Complexity

- First sort the points in each dimension.
  - \( O(dn \log n) \) time and \( dn \) storage.
  - These are stored in \( A[1..d,1..n] \)
- Finding the widest spread and equally divide into two subsets can be done in \( O(dn) \) time.
- We have the recurrence
  - \( T(n,d) \leq 2T(n/2,d) + O(dn) \)
- Constructing the k-d tree can be done in \( O(dn \log n) \) and \( dn \) storage

Node Structure for k-d Trees

- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)

k-d Tree Nearest Neighbor Search

- Search recursively to find the point in the same cell as the query.
- On the return search each subtree where a closer point than the one you already know about might be found.
Nearest Neighbor Search

```
function Nearest Neighbor Search (q: point, n: node, p: point, w: distance) : point |
  if n.left = null then { leaf case }
    if distance(q,n.point) < w then return n.point else return p;
  else
    if w = infinity then
      if q(n.axis) < n.value then
        p := Nearest Neighbor Search(q,n.left,p,w);
        w := distance(p,q);
      else
        p := Nearest Neighbor Search(q,n.right,p,w);
        w := distance(p,q);
    else //w is finite//
      if q(n.axis) + w > n.value then
        p := Nearest Neighbor Search(q,n.left,p,w);
        w := distance(p,q);
      else
        p := Nearest Neighbor Search(q,n.right,p,w);
        w := distance(p,q);
      end if
    end if
  end if
  return p
```

Main is Nearest Neighbor Search

Notes on k-d NNS

- Has been shown to run in $O(\log n)$ average time per search in a reasonable model. (Assume $d$ a constant)
- Storage for the k-d tree is $O(n)$.
- Preprocessing time is $O(n \log n)$ assuming $d$ is a constant.
Geometric Data Structures

• Geometric data structures are common.
• The k-d tree is one of the simplest.
  – Nearest neighbor search
  – Range queries
• Other data structures used for
  – 3-d graphics models
  – Physical simulations