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

Rectangular query

Circular query
Nearest Neighbor Search

Nearest neighbor is e.
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)
k-d Tree Construction (7)
k-d Tree Construction (8)
k-d Tree Construction (9)
k-d Tree Construction (10)
k-d Tree Construction (11)
k-d Tree Construction (12)
k-d Tree Construction (13)
k-d Tree Construction (14)
k-d Tree Construction (15)
k-d Tree Construction (16)
k-d Tree Construction (17)
k-d Tree Construction (18)

k-d tree cell

12/6/02 K-D Trees - Lecture 22 24
2-d Tree Decomposition
k-d Tree Splitting

sorted points in each dimension

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>a</td>
<td>d</td>
<td>g</td>
<td>b</td>
<td>e</td>
<td>i</td>
<td>c</td>
<td>h</td>
</tr>
<tr>
<td>y</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>d</td>
<td>f</td>
<td>e</td>
<td>h</td>
<td>g</td>
</tr>
</tbody>
</table>

• max spread is the max of \( f_x - a_x \) and \( i_y - a_y \).

• 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 Splitting

sorted points in each dimension

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>d</td>
<td>g</td>
<td>b</td>
<td>e</td>
<td>i</td>
<td>c</td>
<td>h</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>d</td>
<td>f</td>
<td>e</td>
<td>h</td>
<td>g</td>
<td>i</td>
</tr>
</tbody>
</table>

indicator for each set

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

scan sorted points in y dimension and add to correct set

<table>
<thead>
<tr>
<th>y</th>
<th>a</th>
<th>b</th>
<th>d</th>
<th>e</th>
<th>g</th>
<th>c</th>
<th>f</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
</table>
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.
k-d Tree NNS (1)

query point

Diagram of a k-d tree with points and bounding boxes.
k-d Tree NNS (2)

query point
k-d Tree NNS (3)

query point

12/6/02
33
k-d Tree NNS (4)
k-d Tree NNS (6)
k-d Tree NNS (7)
k-d Tree NNS (8)

query point
k-d Tree NNS (9)

query point

```
x
s1

y
s2

z
s3

w
s4

d
s5

b
s6

c
s7

a
s8

query point

```

```
x
s1

y
s2

z
s3

w
s4

d
s5

b
s6

c
s7

a
s8

query point

```
**k-d Tree NNS (10)**

![k-d Tree Diagram]

- **query point**

The diagram illustrates a k-d tree with points labeled from a to i, and the corresponding nodes in the tree structure. The tree is traversed based on the query point, highlighting the path and nodes involved in the nearest neighbor search.
k-d Tree NNS (11)
k-d Tree NNS (12)
k-d Tree NNS (13)
k-d Tree NNS (14)
k-d Tree NNS (15)

query point

12/6/02  K-D Trees - Lecture 22  45
k-d Tree NNS (16)
k-d Tree NNS (17)

query point

12/6/02  K-D Trees - Lecture 22
k-d Tree NNS (18)
k-d Tree NNS (19)
k-d Tree NNS (20)
k-d Tree NNS (21)
Nearest Neighbor Search

NNS(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 := NNS(q, n.left, p, w);
                w := distance(p, q);
                if q(n.axis) + w > n.value then p := NNS(q, n.right, p, w);
            else
                p := NNS(q, n.right, p, w);
                w := distance(p, q);
                if q(n.axis) - w ≤ n.value then p := NNS(q, n.left, p, w);
        else /*w is finite*/
            if q(n.axis) - w ≤ n.value then
                p := NNS(q, n.left, p, w);
                w := distance(p, q);
                if q(n.axis) + w > n.value then p := NNS(q, n.right, p, w);
            return p
}
The Conditional

\[ q(n.\text{axis}) + w > n.\text{value} \]
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