

## COS 226 Lecture 16: Geometric search

Extend search ADT to geometric data

### PROBLEMS

- Range search
- Intersections among geometric objects
- Near neighbor search
- Point location

TWO-DIMENSIONAL  
MULTIDIMENSIONAL

### APPROACHES

- trees
- divide-and-conquer
- discretized algorithms

## Range searching

Possible addition to symbol-table ADT:

```
. void STinit();  
. void STinsert(Item x);  
. Item STsearch(Key v);  
. int STempty();  
. int STrange(Key v1, Key v2);
```

Options to actually process the records

- pass a procedure to call for each record in the range
- return list of records (possibly sorted)

Depends on how many records expected (count them first)

**ARRAY** implementation: do binary search on both keys

**HASH TABLE** implementation: no easy algorithm

**BST, TRIE** implementations: recursive traversal works



## 2D Range searching

Same basic method works in higher dimensions (!!)

- discovered by an undergraduate

INTERVAL in 1D is RECTANGLE in 2D

**2D TREE:** alternate x and y

Recursively search subtrees that COULD HAVE keys in interval

- root may or may not be in rectangle
- search BOTH subtrees if it is

Corresponds to recursive subdivision of the plane

- alternating horizontal and vertical lines

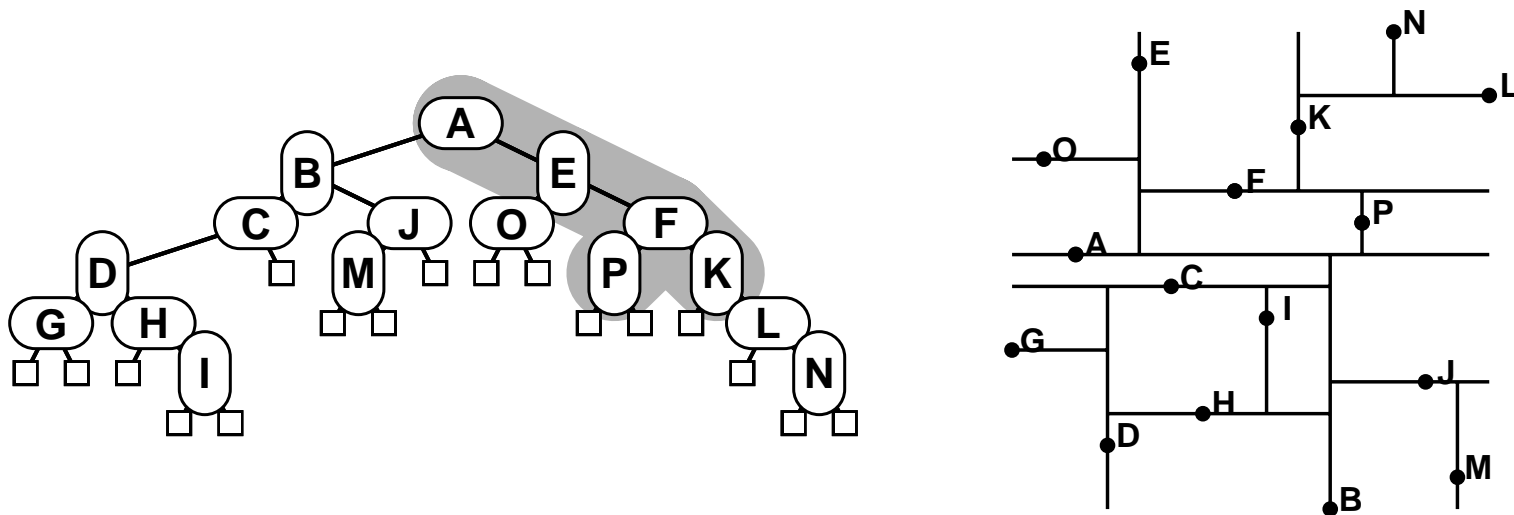
**kD tree:** trivial generalization

## 2D tree example

Each EXTERNAL node corresponds to an area in the plane

Each INTERNAL node divides its area into two subdivisions

Switch between horizontal and vertical dividing lines



### Quad tree

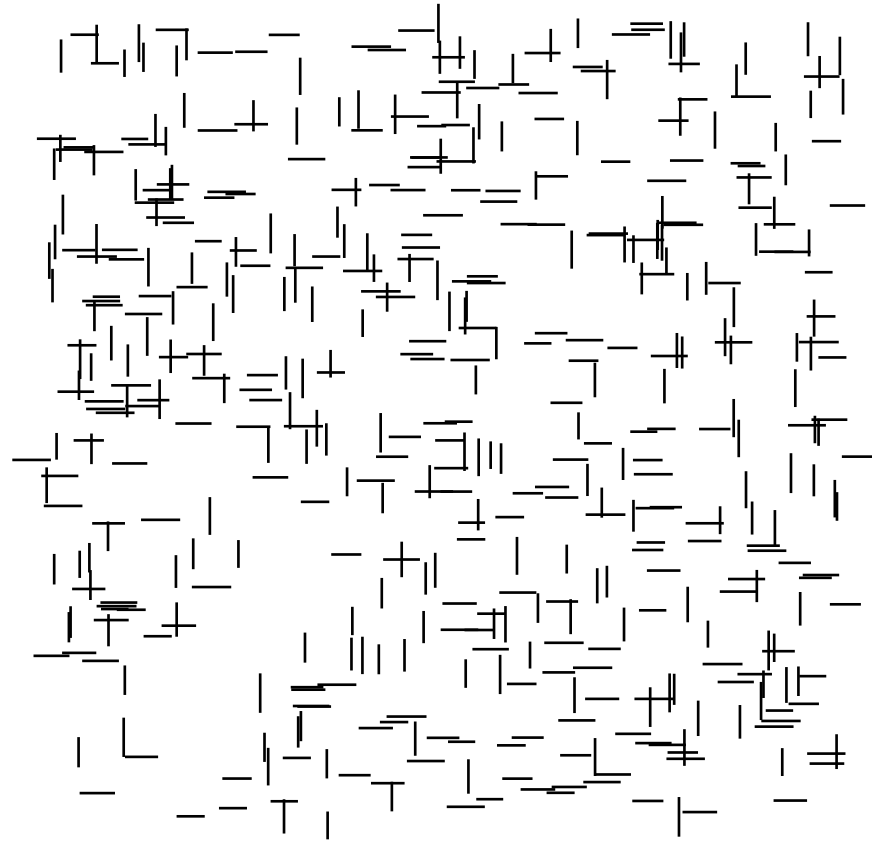
- use 4-way tree (divide on both coordinates at once)



## Manhattan line intersection problem

**N lines, all either horizontal or vertical**

**How many pairs intersect?**



**As with other search problems**

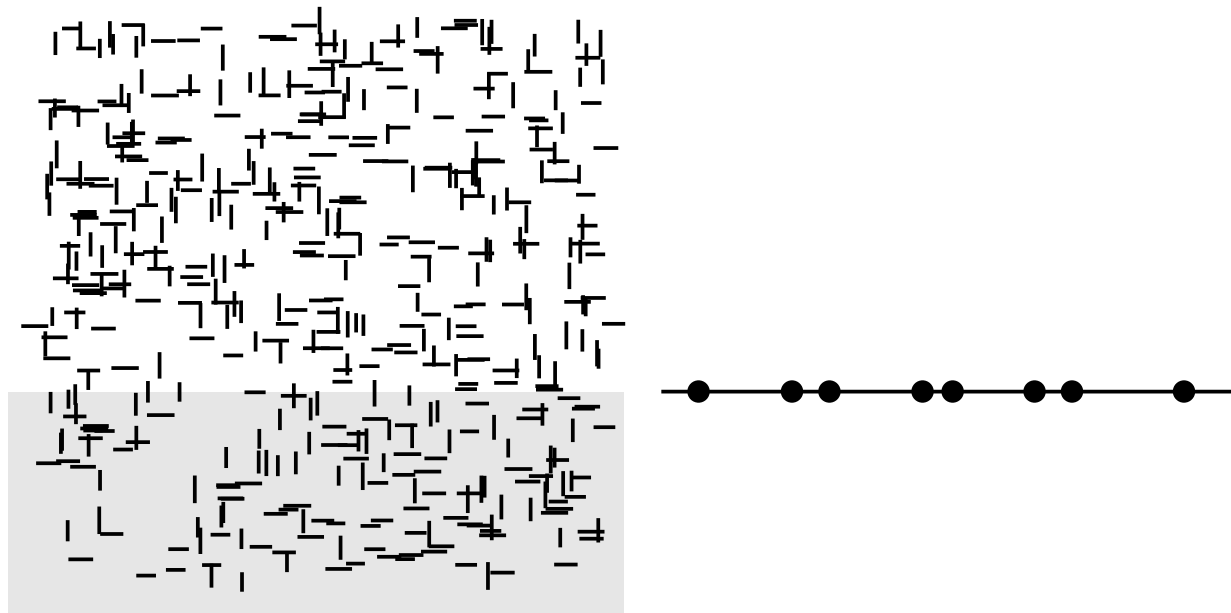
- usually no harder to **REPORT** all intersections
- (call a given function for each)

## Manhattan line intersection

### Dynamic SWEEP LINE algorithm

Horizontal line sweeps from bottom to top

- vertical data line represents "point"
- horizontal data line represents "interval"



There is an h-v intersection if "point" is in "interval"  
Reduces 2D line intersection problem to 1D range searching!

## Sweep line implementation

Uses both PQ and ST (with range search) ADT

- PQ: get  $y$  coordinates in increasing order
- ST: range search on  $x$  coordinates for intersection

Three types of "events"

**B:** bottom of vertical line [INSERT  $x$ ]

**T:** top of vertical line [DELETE  $x$ ]

**H:** horizontal line [RANGE ( $x_1, x_2$ )]

Generalizes to give fast algorithms for

- rectangles, general lines, circles, convex polygons

Generalizes to higher dimensions

- "sweep hyperplane"

## Near neighbor searching

Another possible addition to Search ADT:

- `void STinit();`
- `void STinsert(Item x);`
- `Item STsearch(Key v);`
- `int STempty();`
- `Item STnearest(Key v);`

Find the record with key value closest to  $v$

Need a concept of "distance", not just "less"

- easy if keys are numbers, or points in space

**ARRAY** implementation: scan both ways after binary search

**HASH TABLE** implementation: no easy algorithm

**BST, TRIE** implementations: recursive traversal works

## 1D BST near neighbor searching

Recursively search subtrees that COULD HAVE near neighbor

- may search BOTH subtrees

```
void BSTnear(link h)
{
    if (h == z) return;
    if (dist(v, h->key) < min)
        { best = h; min = dist(v, best->key); }
    if (v < h->key || (v - h->key) < min)
        BSTnear(h->l);
    if (v > h->key || (h->key - v) < min)
        BSTnear(h->r);
}
```

Multidimensional near neighbor searching:

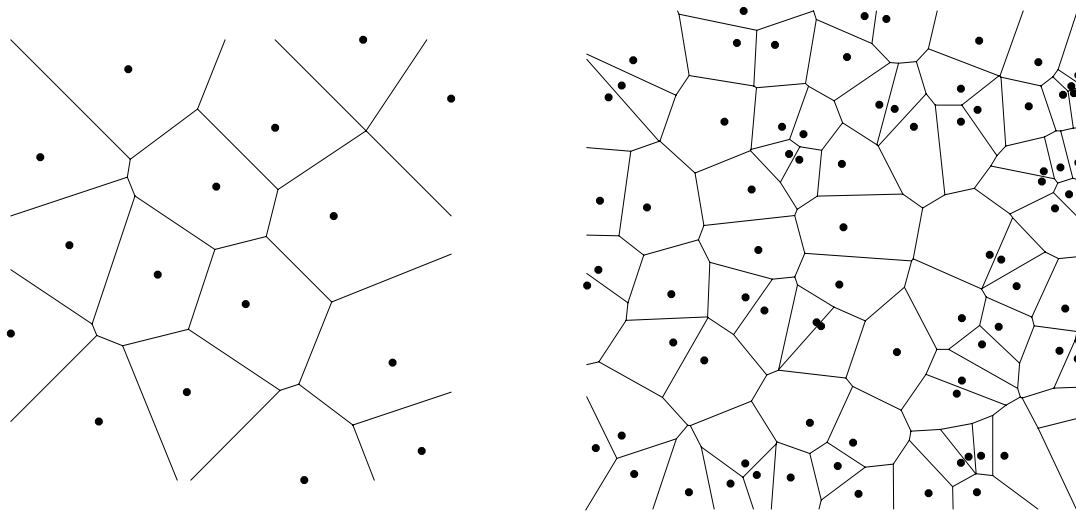
- same algorithm on kD tree

# Voronoi diagram

Given: set  $S$  of  $N$  points

point  $x$ 's Voronoi REGION:

- set of points closer to  $x$  than to any other  $y$  in  $S$



Voronoi EDGES: perpendicular bisectors of point pairs

- intersect at centroids of point triple triangles

**Voronoi DIAGRAM:** union of Voronoi edges

Challenge to compute

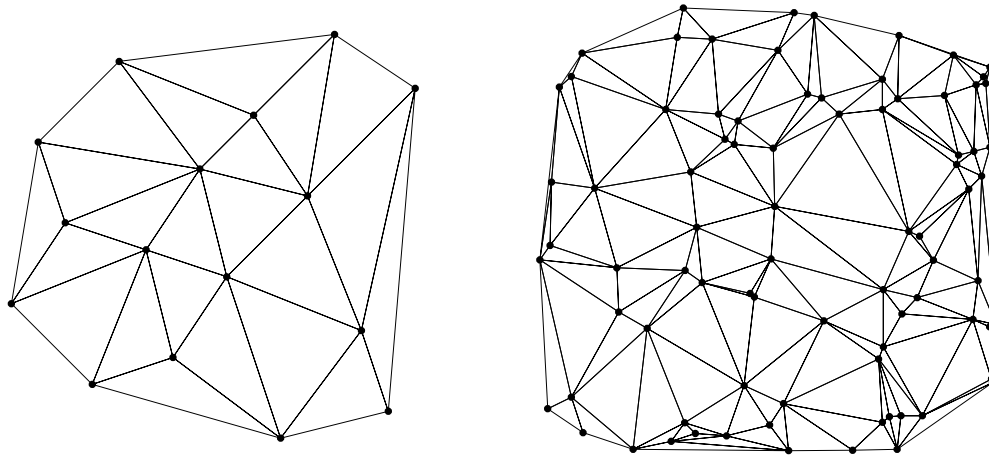
- Representation?
- Degenerate cases?

## Delaunay triangulation

Given: set  $S$  of  $N$  points

**DELAUNAY TRIANGULATION**

- edge  $x-y$  iff Voronoi edge separates  $x$  and  $y$



Outer boundary is convex hull

Representation easier: no extra points

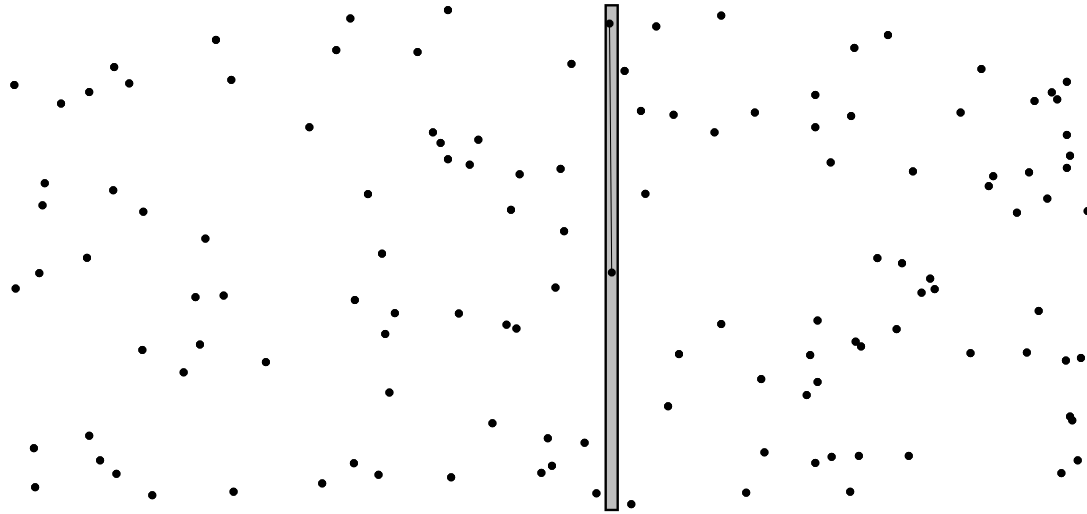
**THM:** Voronoi diagram and Delaunay triangulation can be computed in  $N \log N$  steps (!!)

- divide and conquer
- sweep line
- randomize
- discretize

## 2D divide-and-conquer

**Ex:** CLOSEST PAIR algorithm

- sort on  $x$
- divide into two sets of  $N/2$  points
- find closest pair in each half
- find closest pair crossing boundary



Boundary check **MUST** be efficient (terminates recursion)

- sort on  $y$  to make boundary check easy
- $y$  sort comes for free (!!)

Implementation: tricky exercise in recursion (see text)

## Grid methods

Grids : geometric search :: tries : search ADT

### Grid method

- define uniform grid of fixed-size squares
- put points in lists associated with squares
- ignore points in faraway grid squares

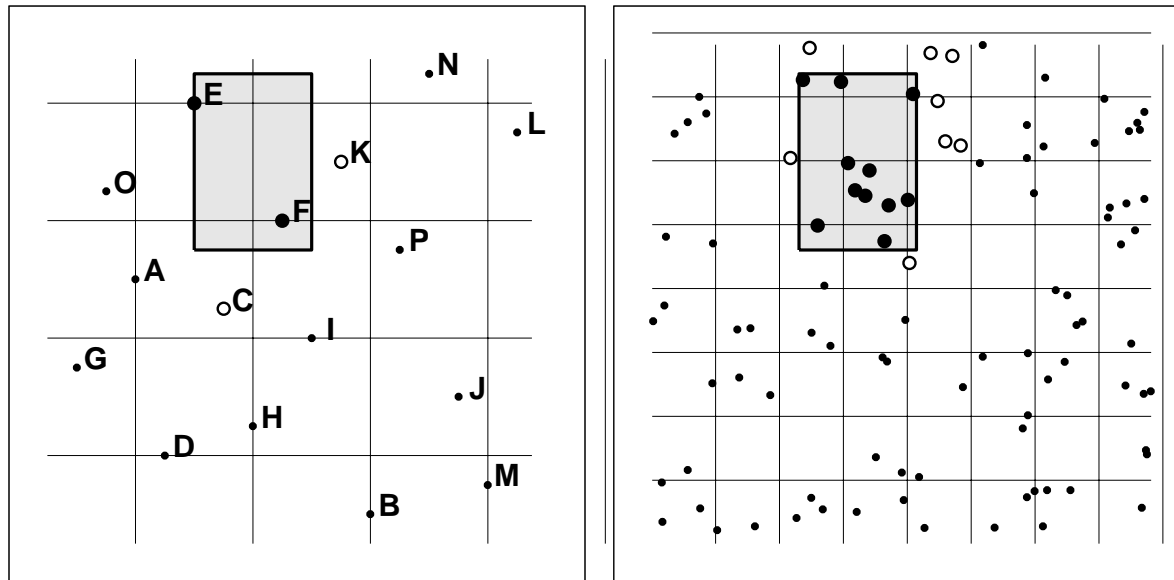
Time-space tradeoff like MSD sort

- grid too fine: empty cells
- grid too coarse: lists too long

Use 2- or 3-level grids, or recurse ala quad trees

## Grid methods (continued)

Ex: range searching



For graphics applications

- ultimate grid is PIXEL ARRAY
- leads to "discretized algorithms"

## Point location problem

**Ex:** find state corresponding to point on map

### Planar subdivision

- 2D tree planar decomposition
- N lines
- Voronoi diagram
- grid
- pixel array

Which division contains the given point?

Difficult in general

if only because of difficulty of  
representing planar subdivisions

## Discretized line intersection

$p$ -by- $p$  bit raster,  $p^2$  pixels

$N$  lines

Draw rasterized version of line

- report intersection if pixel already 1

Cost:

- $p^2$  to initialize pixels to 0
- number of pixels on lines

Cost dominated by  $p^2$

Line intersection same cost as drawing blank picture!

## Discretized Voronoi diagram

put 1 pixels on a priority queue

priority: distance to closest point

### ALGORITHM

- remove pixel from priority queue
- check all neighbor pixels
  - if closer or same: ignore
  - if farther: check pixel value
    - if 0, set to 1 and put back on pq
    - if 1, must be on a voronoi edge!

Time proportional to initialize plus product of

- number of pixels on diagram
- diameter of largest cell

**Idea:** refine discretized diagram to compute real diagram