+

INFS4205/7205 Advanced Techniques for High Dimensional Data

Spatial Query Processing 1

Semester 1, 2021

University of Queensland

+ Advanced Techniques for High Dimensional Data

- Course Introduction
- Introduction to Spatial Databases
- Spatial Data Organization
- Spatial Query Processing
- Managing Spatiotemporal Data
- Managing High-dimensional Data
- Other High-dimensional Data Applications
- When Spatial Temporal Data Meets AI
- Route Planning
- Trends and Course Review

+ Learning Objectives

- What we will cover
  - The filter-and-refine approach
  - Some basic computational geometry algorithms
  - Intersection join algorithms using various spatial indexes
  - Efficient processing of some advanced spatial queries
- Goals
  - Understand efficient processing of spatial queries using spatial indexes and the filter-and-refine approach
  - Enhance the understanding of spatial databases from in-depth knowledge of advanced spatial processing
  - Provide a brief view of the frontier of spatial database research

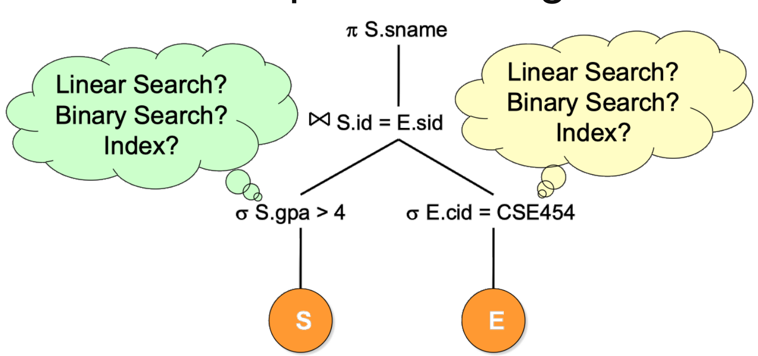
+ Readings

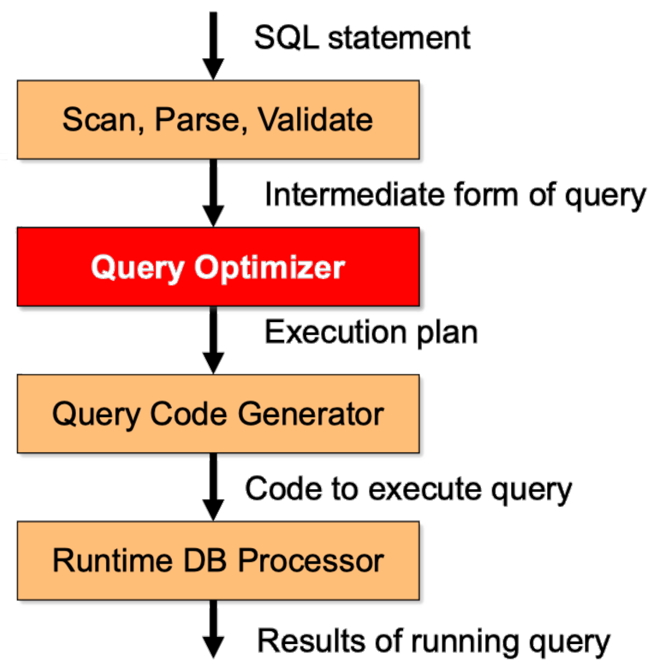
- R. Güting, An Introduction to Spatial Database Systems, The VLDB Journal, 3:4, 1994
- Oracle Business and Technical White Papers
- V. Gaede and O Günther, Multidimensional Access Methods, ACM Computing Surveys, 30:2, 1998
- J. Orenstein and F. Manola, PROBE Spatial Data Modeling and Query Processing in an Image Database Application, IEEE Transactions on Software Engineering, 14:5,1988
- T. Brinkhoff, H.-P. Kriegel and B. Seeger, Efficient Processing of Spatial Joins Using R-Trees, SIGMOD'93

+ Query Processing

- Process a query accurately and in the minimum amount of time possible
  1. Design and fine-tune algorithms for each of the basic operators
  2. Map high-level queries into a composition of these basic operators and optimize with their information

- RDBMS Query Processing
  - Projection / Selection / Join
    - Linear Scan / Binary Search / Index
    - Nested Loop / Sort Merge / Hash Join



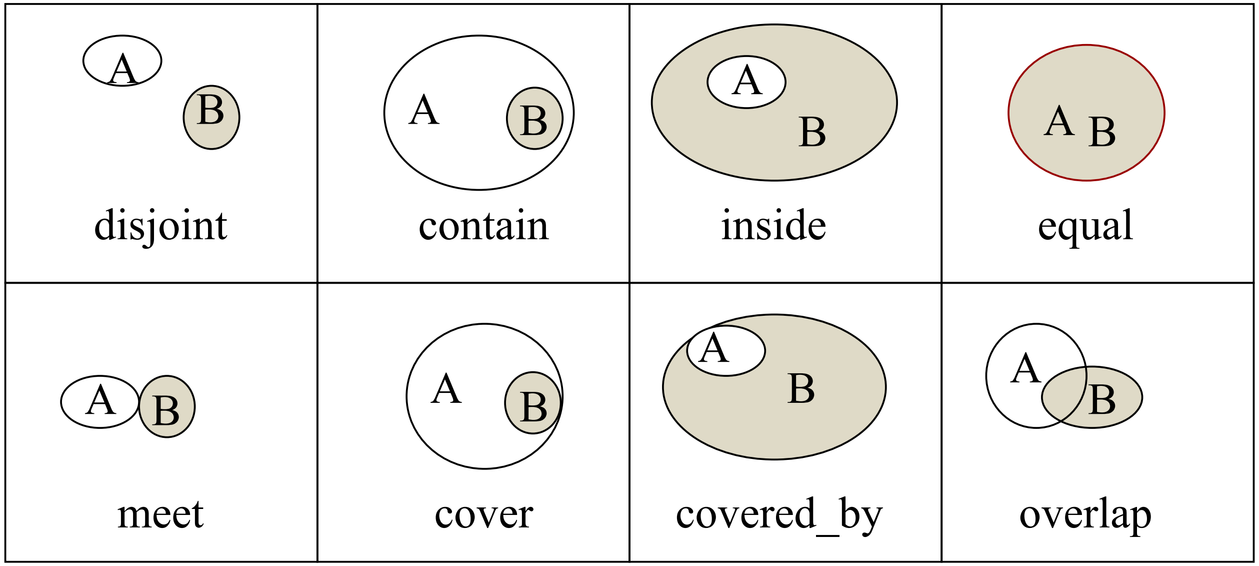


+ Spatial Operations

- Spatial operations are intrinsically more expensive than relational operations
  - Large amount of complex data
    - → higher I/O cost
  - Computational geometry algorithms
    - → higher CPU cost
- Spatial query processing is different from RDBMS query processing
  - Different types of indexes
  - CPU costs typically not considered in RDBMS query optimization

+ Spatial Relations

- Topological



- Directional
  - Above, Left, North of,…
- Metric
  - Distance, Length, Area,…

+ Spatial Operations

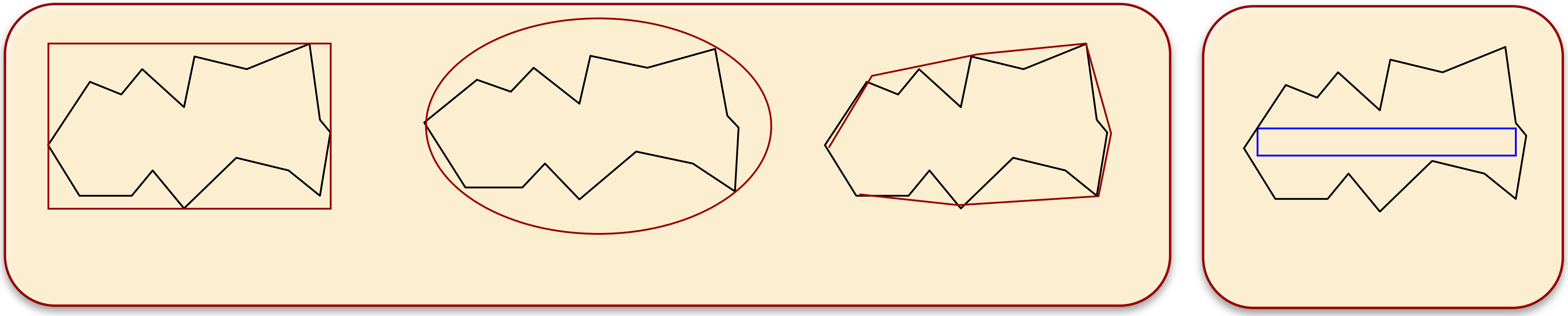
- Update
  - Create, Delete, Modify…
- Selection
  - Point Query
    - Given a point p
      - Find all the spatial objects O that *contains* it
      - Find all the spatial objects O that within 10km in Euclidean/Network distance
  - Range Query
    - Given a query polygon Q, find all spatial objects O that *intersects* it
    - Window query: when Q is a rectangle
- Spatial Join
  - R and S join on a spatial predicate θ

+ Filter-and-Refine

- A most commonly used processing strategy
- Motivation
  - Avoid expensive spatial processing as much as possible
- Basic Idea
  - A filter step, followed by a refinement step
  - Filter step: applying *simple* operations on *approximations* of spatial objects
  - Refinement step: applying the *actual* spatial operations on the *full geometry* of spatial objects

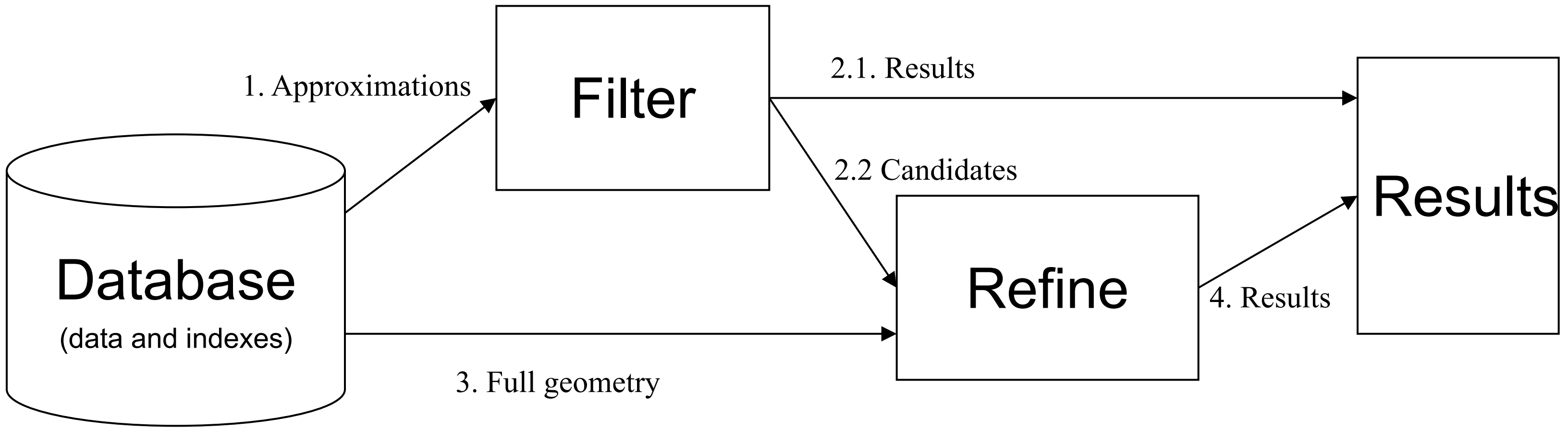
+ Object Approximation

- Different types of approximation



- Conservative approximation: containing the object (min)
  - Approximation don't overlap ➔ impossible for objects to overlap
  - Approximation do overlap ➔ objects may overlap (need to refine!)
- Progressive approximation: contained by the object (max)
  - Approximation do overlap ➔ objects overlap too!
  - Approximation don't overlap ➔ objects may overlap (need to refine!)

+ The Filter-and-Refine Workflow



+ An Algorithm Using MBR Filter

- The Filter step

```
C = ϕ;
for each d in D
   if d.mbr contains p
      add d.pid to C;
```

D:

pid	mbr	geometry	attributes

Query: find all polygons containing point p.

Note: "contains" in the two steps are different.

- The Refine step

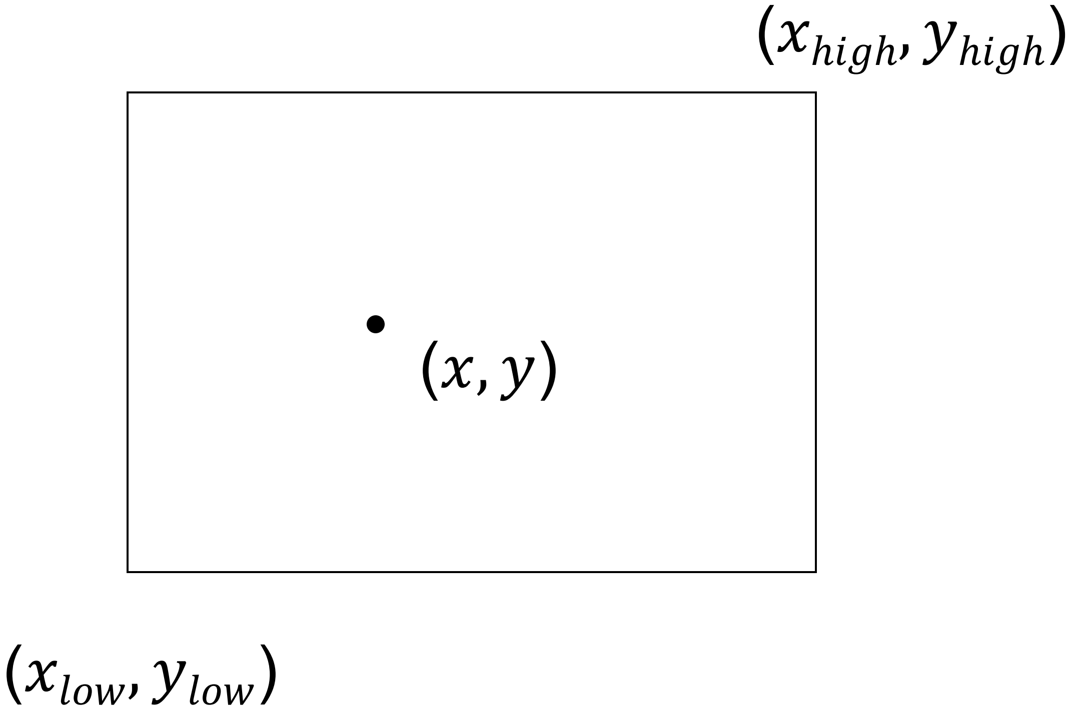
```
for each c in C
   select geometry from D where pid = c into geo;
   if geo contains p
      output c;
```

+ Computational Geometry Algorithms

- Point in Rectangle
- Rectangle intersection
- Point in Polygon
- Polyline intersection
- Polygon intersection

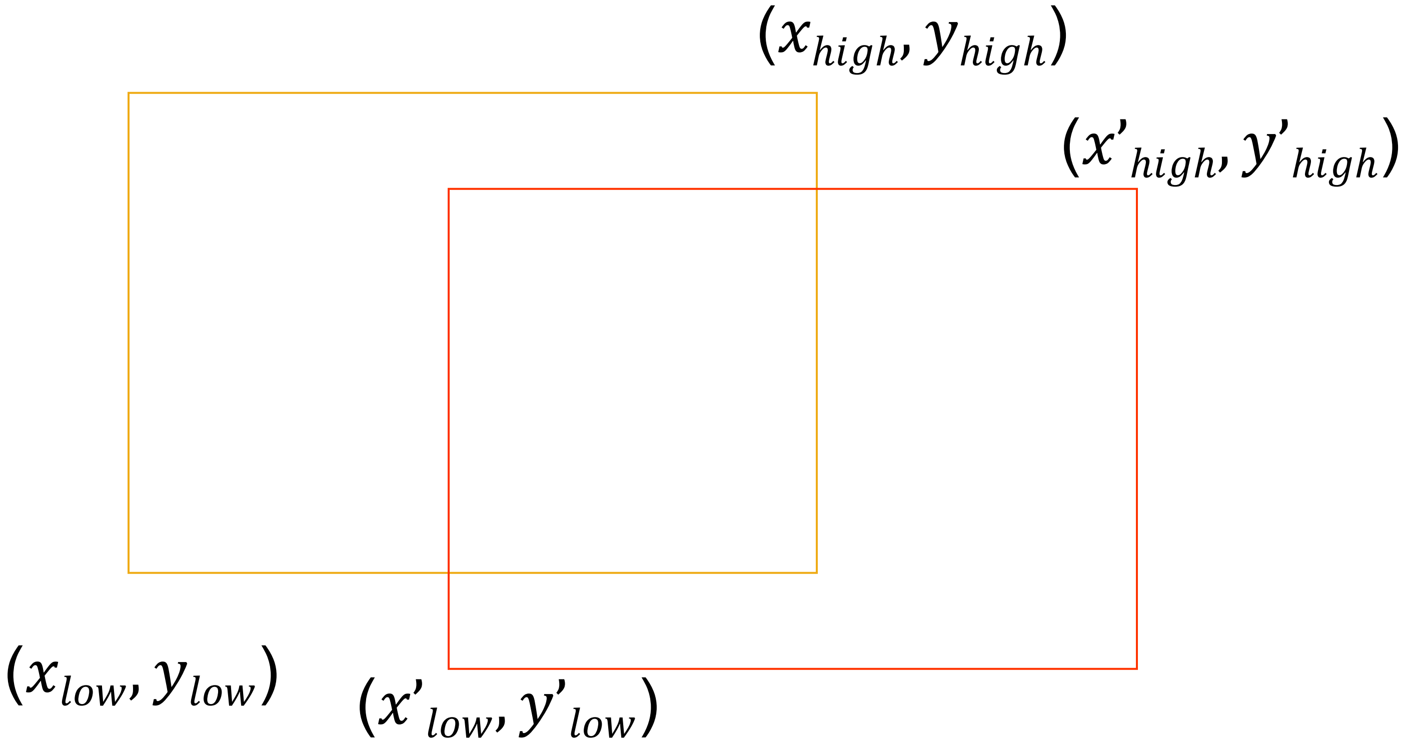
Reading: Preparata and Shamos,
Computation Geometry: An Introduction
Springer-Verlag 1985.

+ Point in Rectangle



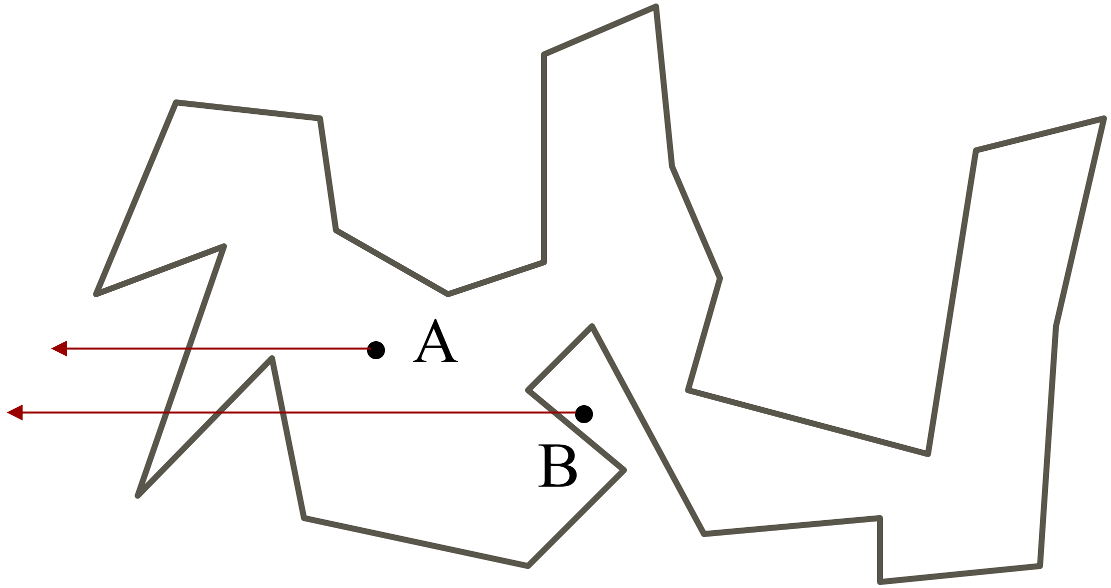
...the condition?

+ Rectangle Intersection



- Four *point-in-rectangle* check?
- When do not overlap?
  - One is on top: $y'_{low} > y_{high}$ or $y_{low} > y'_{high}$
  - One is left/right: $x_{low} > x'_{high}$ or $x'_{low} > x_{high}$

+ Point in Polygon



The ray-cutting algorithm:

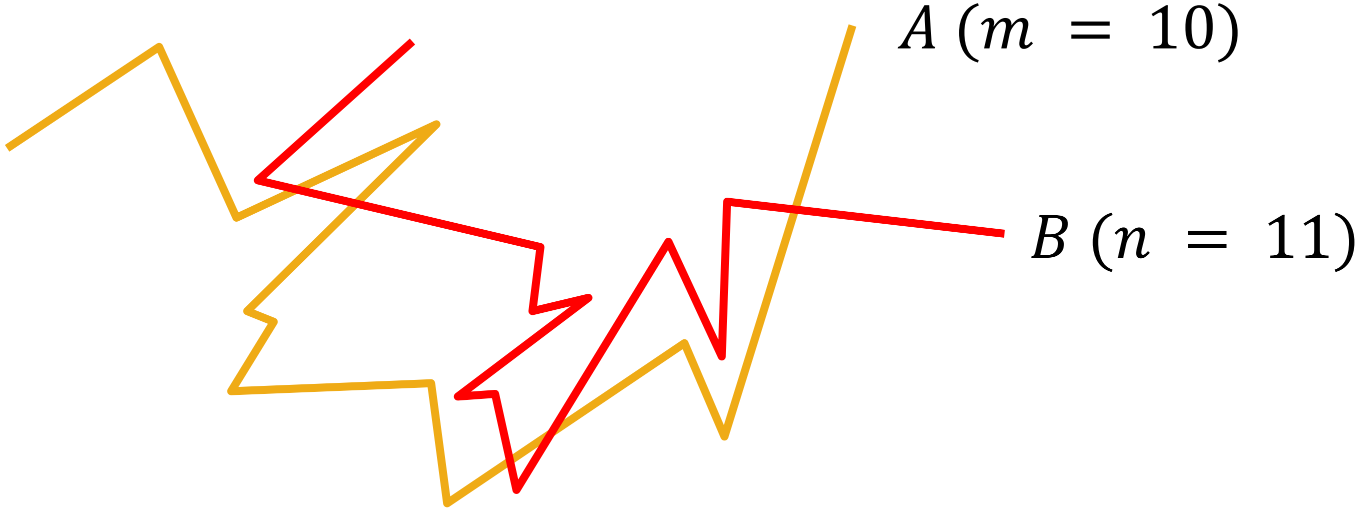
#Intersection: odd = inside, even = outside

...how to represent a line using equations?

...how to check if two lines interest each other?

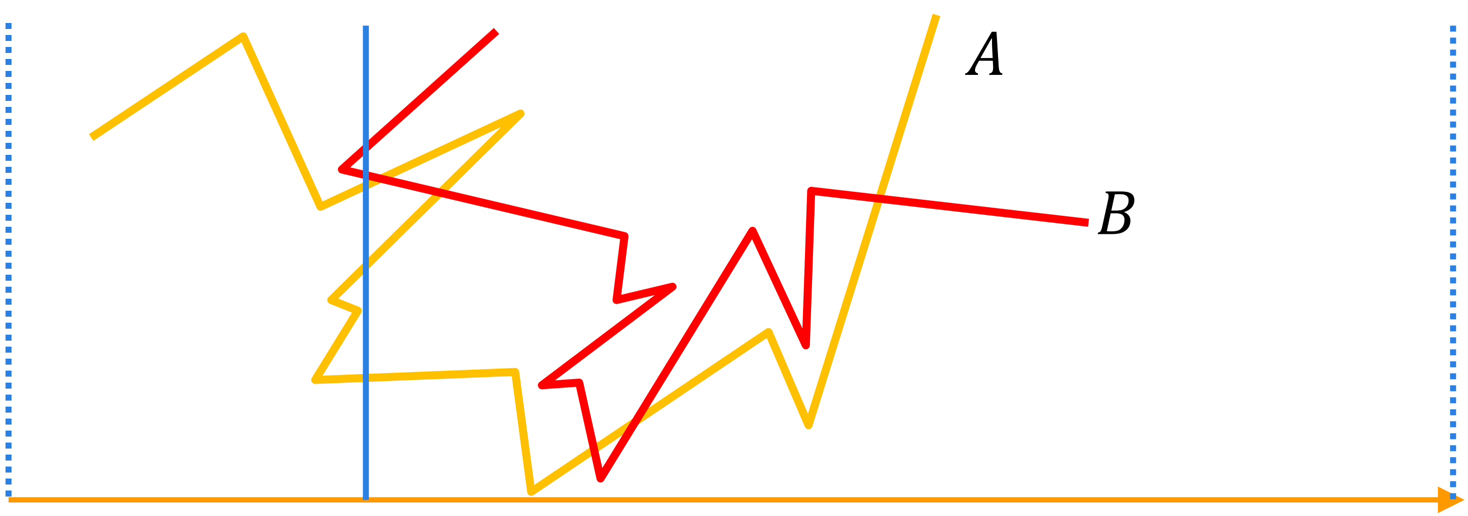
+ Polyline Intersection

- Naïve: for each line segment in A, test against each and every line segment in B
  - $O(m \times n)$, where A and B have *m* and n line segments respectively
    - When each segment intersects with all the others, $\Omega(m \times n)$ is inevitable
    - But in practice, the total number of intersections is much smaller
    - Output / Intersection sensitive?



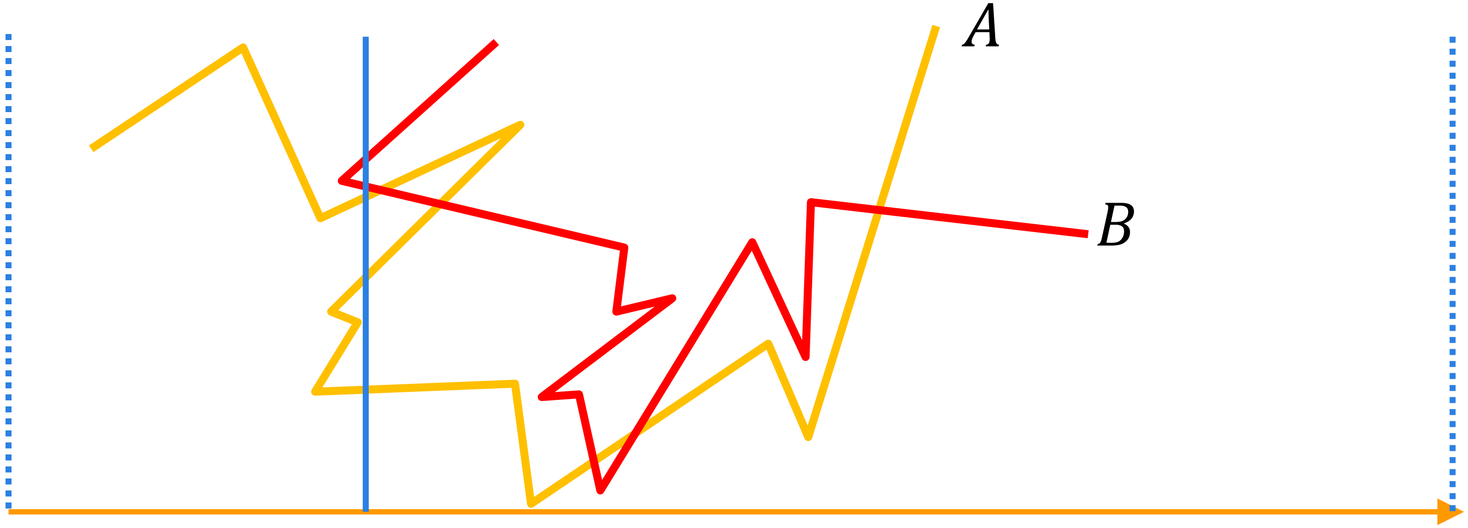
+ The Plane-Sweep Algorithm

- Extensively used in computational geometry to compute intersections of geometric objects
  - Move a vertical line (*the sweep line*) from left to right
    - Only test those whose x-interval overlaps
    - End points are the *event* point: Only test at these points
    - Only those A and B lines which intersect with the sweep line at the same time can possibly intersect



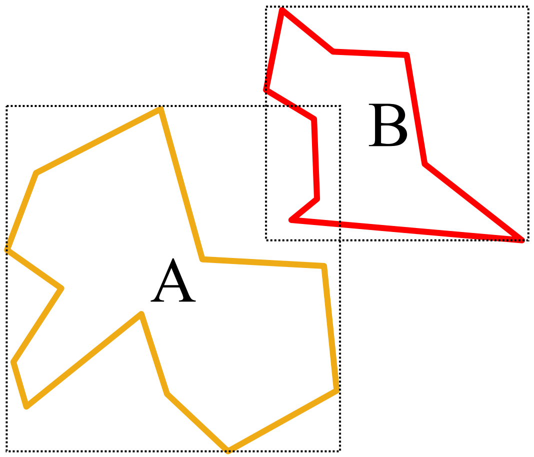
+ The Plane-Sweep Algorithm

- Only those A and B lines which intersect with the sweep line at the same time can possibly intersect
  - Add into test buffer when the sweep line is at the left end
  - Delete when the sweep leaves at the right end
    - But the two segments that intersect the sweep line can still be far apart in the vertical direction
    - Order the segments from top to bottom as they intersect the sweep line
    - $O((m+n)\log(m+n))$ ...*much better than $O(m*n)$!*



+ Polygon Intersection

- Step1: MBR Intersection
  - Filter the impossible
- Step2: Point-in-Polygon testing
  - "$A[i]$ in B" or "$B[j]$ in A"
    - i, j: arbitrary point
- Step3: Polyline Intersection

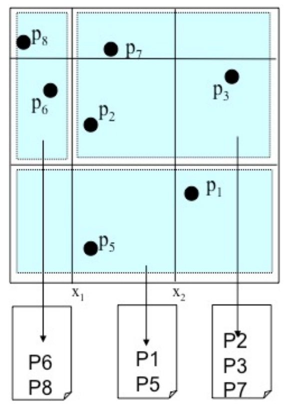


+ Computational Algorithms vs Spatial Database Algorithms

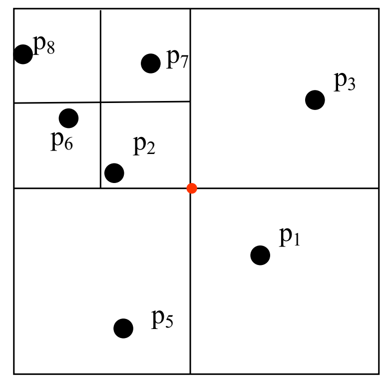
- Computational algorithms
  - Concerning two complex objects
  - Goal: (in memory) computational complexity
- Spatial database algorithms
  - Concerning one or two very large sets of object, which can be simple or complex
  - Goal: *the total cost* (I/O and CPU)
- A spatial database algorithm uses some sort of computational algorithms, but focuses on making data retrieval more efficient (e.g., using spatial indexes)

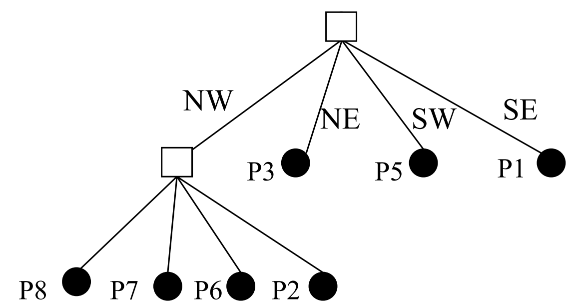
+ Spatial Selection Query

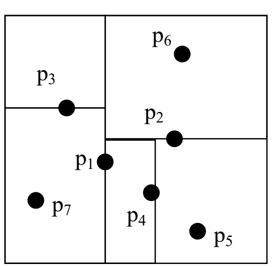
- Point / Range / Within Distance Query
  - Grid Files
  - Quadtree
  - Kd-Tree
  - Z-Value
  - R-Tree

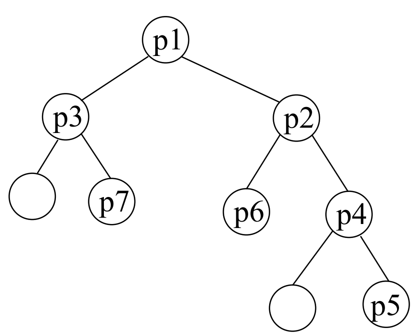


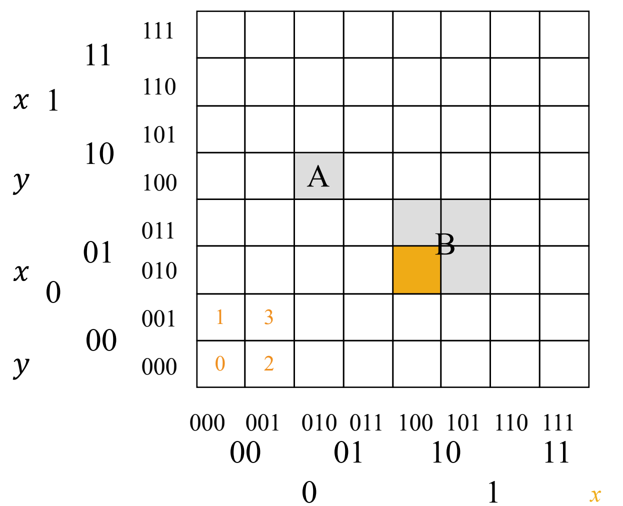
Data buckets
Capacity = 3

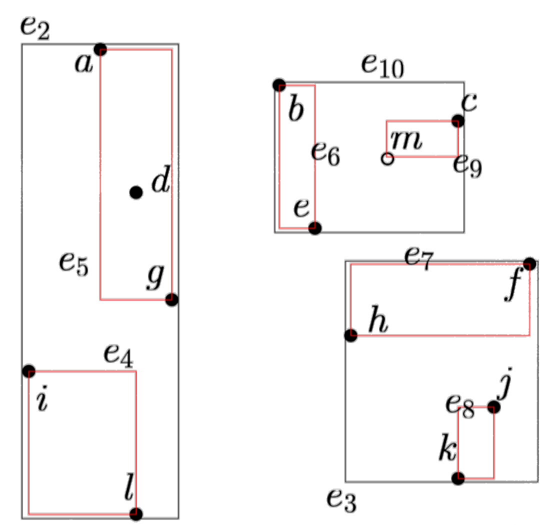










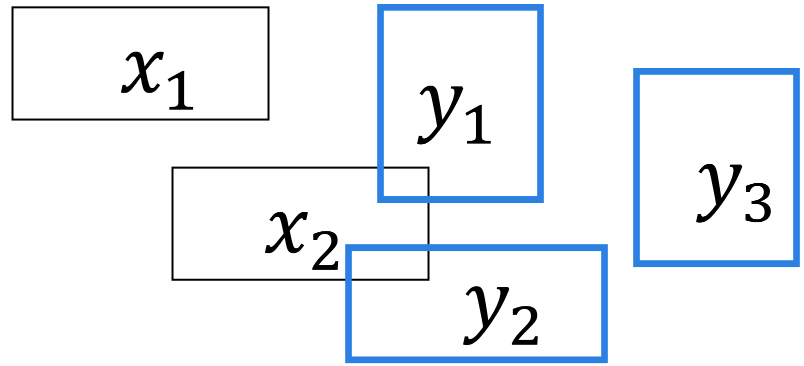


+ Spatial Join Algorithms

- One of the most important spatial operations
- Can be very time-consuming
  - Complex data, complex operations
  - It's not an equijoin!
- Three types of join algorithms:
  - Nested-loops
  - Sort-merge
  - Hash-based

+ Spatial Join Example

- Intersection join



- Join results: (x_2, y_1), (x_2, y_2)
- Other spatial join operations
  - Topological: intersection, adjacent, contains…
  - Metrical or directional: within_distance…
  - More advanced: nearest....

+ Processing Framework

- Filter step
  - Find a set of candidates $C = \{(p, s)\colon p \in R\ and\ s \in S\}$ *quickly*
    - Using approximations (e.g., MBR) and indexes
    - Other filter steps possible (eg, using *progressive* approximation)
- Housekeeping step
  - Process C such that the IO cost for the refinement step can be further minimized
  - E.g., Removing duplicates; Performing refinement in optimal order
- Refine step
  - Fetch full geometry for the objects in each candidate, and apply a full test to drop “false hits”

+ Simple Nested-Loops Join

for each *r* in *R*

    for each *s* in *S*

        if (*r,MBR* **intersect** *s.MBR*)

            put (*r*, *s*) to the candidate set;

+ Properties with R/R+-Trees

- All tree nodes contain a set of (MBR, pointer) pairs
  - For each entry in a leaf node:
    - The pointer points to a disk page
    - The MBR covers all the polygons on the disk page
  - For each entry in an internal node:
    - The pointer points to the node of a sub-tree
    - The MBR covers all the polygons (or their MBRs) in that sub-tree
- Therefore
  - If the MBRs of two entries are disjoint, so must be all their children
  - If the MBRs of two entries intersect, then some children pairs *might* intersect somewhere down the tree/on the page

+ Indexed Nested-Loops Join

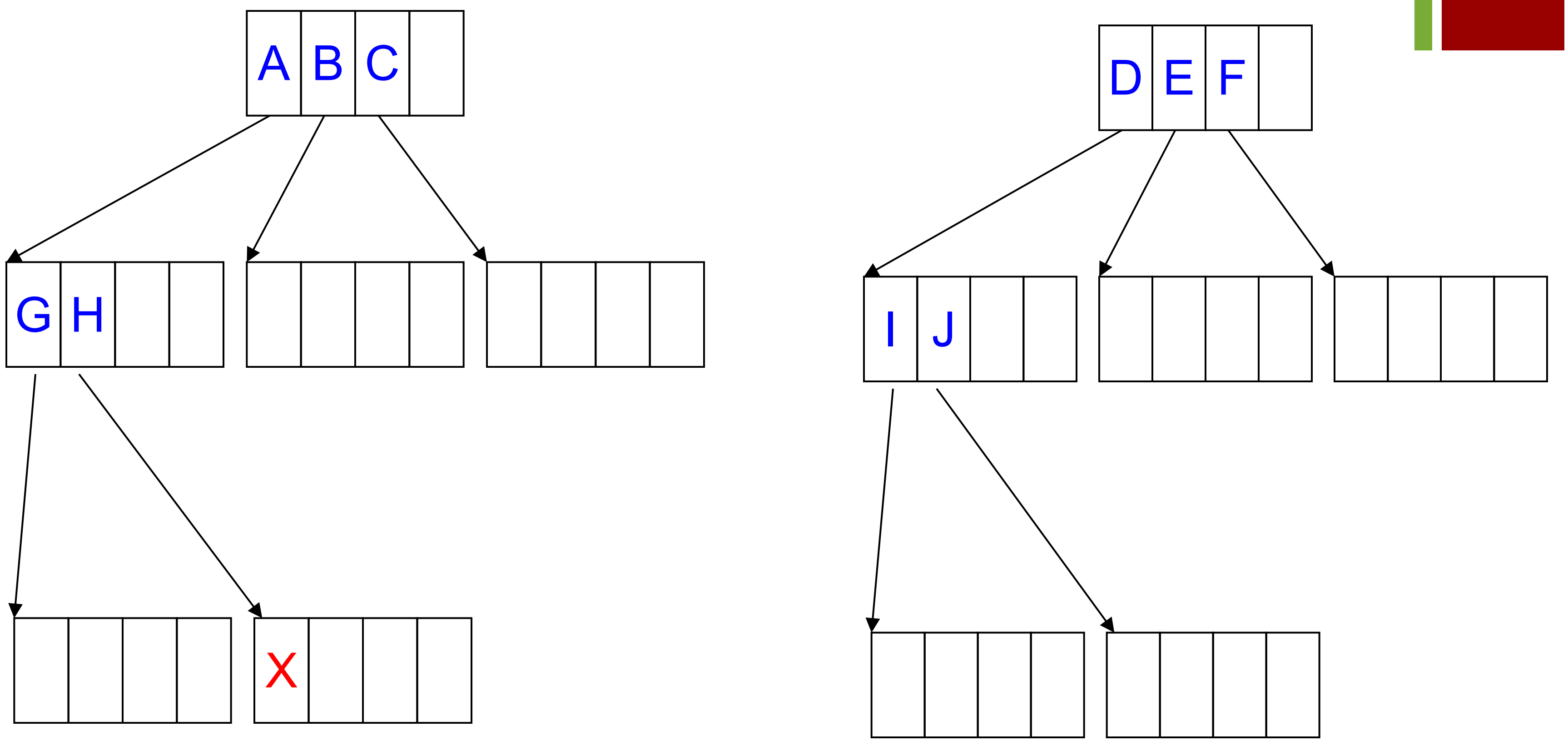
- Using a window query against S

for each r in R

Find all s in S such that $s.MBR$ **intersect** $r.MBR$

put (r, s) to the candidate set;

+ Nested-Loops With R/R+-Trees



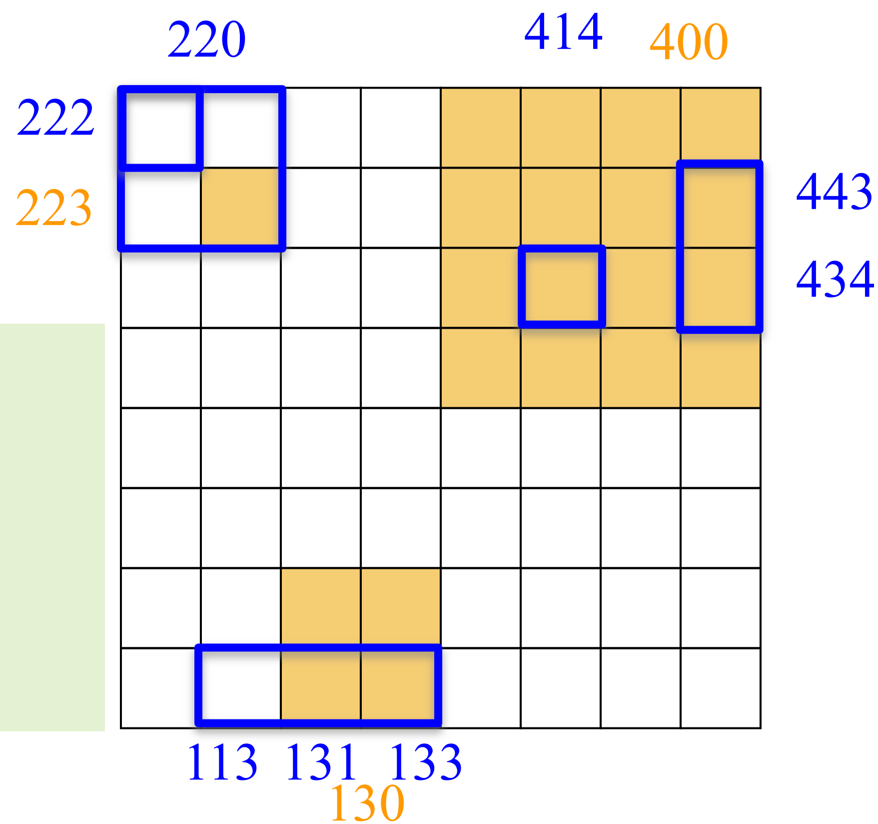
Is it possible to produce the same candidate (p_1, p_2) multiple times?

Reading: the paper by Brinkhoff, Kriegel and Seeger 1993.

+ Sort-Merge Join using Z-values

r		s
113		130
131		223
133		400
220		
222		
414		
434		
443		

- Differences
  1. $131 \neq 130$
  2. Cannot move from 130 to 223 immediately

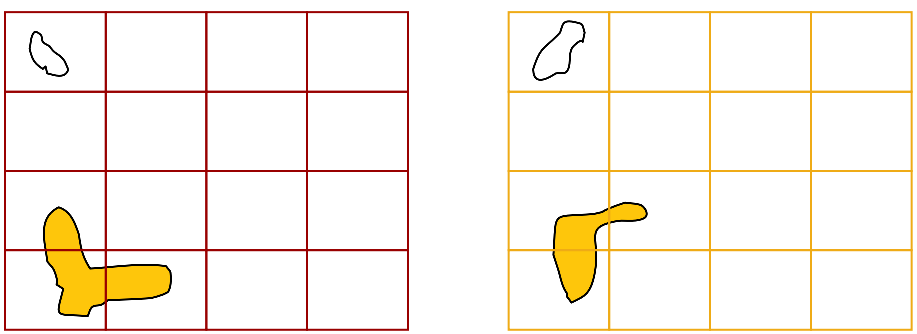


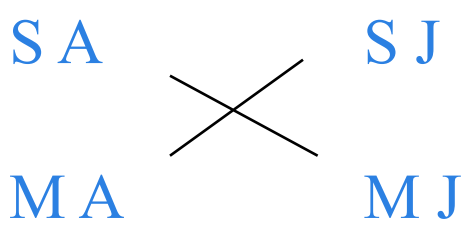
Algorithm Sketch:
1) Two sorted lists and two pointers
2) Synchronized traversal
   - overlap(*r*, *s*)?
   - increase min(*r*, *s*)
3) Some values in stack

Reading: the paper by Orenstein and Manola 1988.

+ Spatial Hash Join

- Bucket Assignment
  - SA: a polygon is mapped to a cell by its centroid
  - MA: a polygon is mapped to all cells it overlaps with
- Bucket Join
  - SJ: one R bucket joins with one corresponding S bucket
  - MJ: one R bucket joins with many S buckets
    - Question: which buckets to join with?

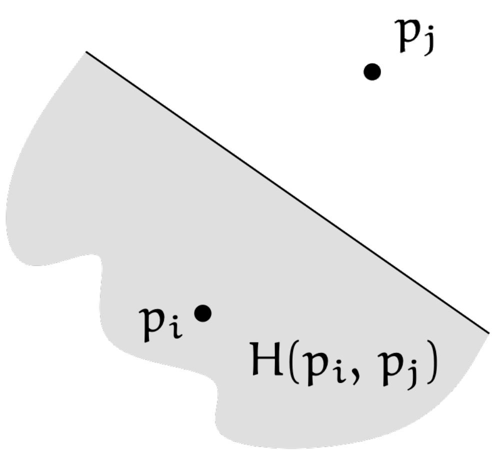


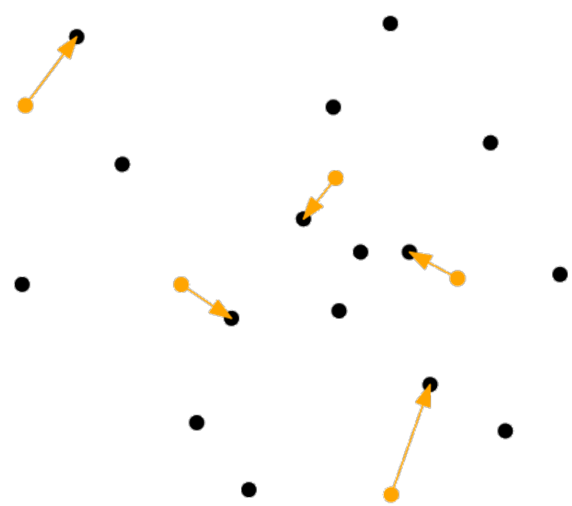


[1] Data Partitioning for Parallel Spatial Join Processing, SSD 1997

+ The Post Office Problem

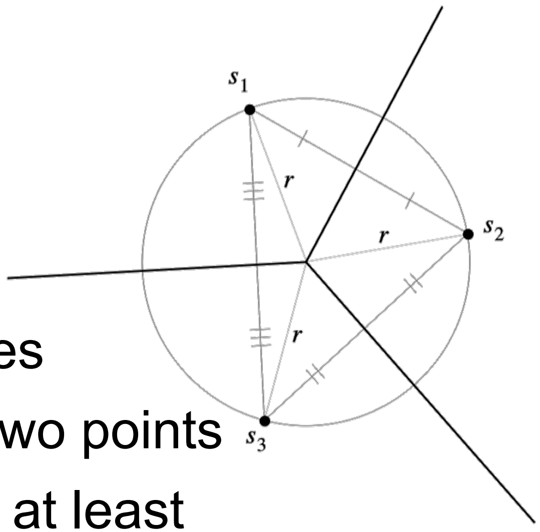
- Given n post offices $\mathrm{P} = \{p_1, \ldots, p_n\}$ in a city, find the nearest post office of a given location q
  - The locations of post offices are known and do not change frequently
  - How to answer the query efficiently?

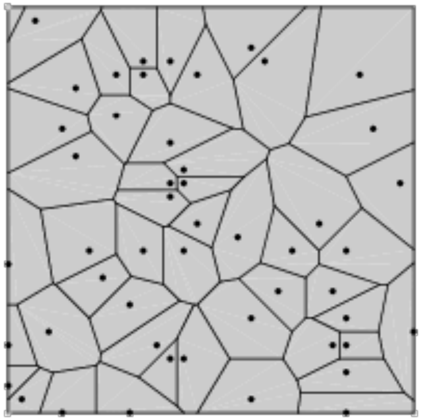




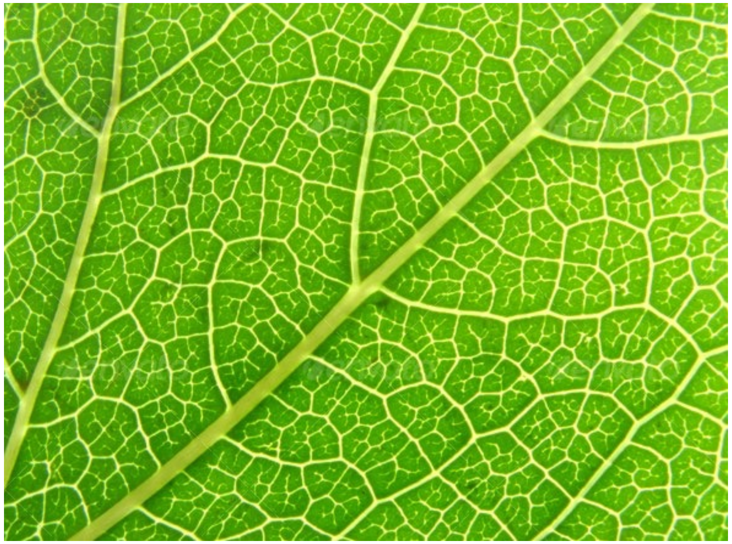
+ Voronoi Diagram

- $\forall p_i, p_j \in S$, p_i $dominates$ p_j is the sub-plane being at least as close to p_i as to p_j
  - $dom(p_i, p_j) = \{q \in R^2 | dist(q, p_i) \leq dist(q, p_j)\}$
  - Region $reg(p_i) = \bigcap_{p_j \in S - \{p_i\}} dom(p_i, p_j)$
    - Intersection of $n-1$ half-planes
    - The boundary has at most $n-1$ edges / vertices
      - Each point on an edge is equidistance from two points
      - Each vertex on an edge is equidistance from at least three points
  - This polygonal partition is Voronoi diagram $Vor(S)$
    - Contains exactly n regions

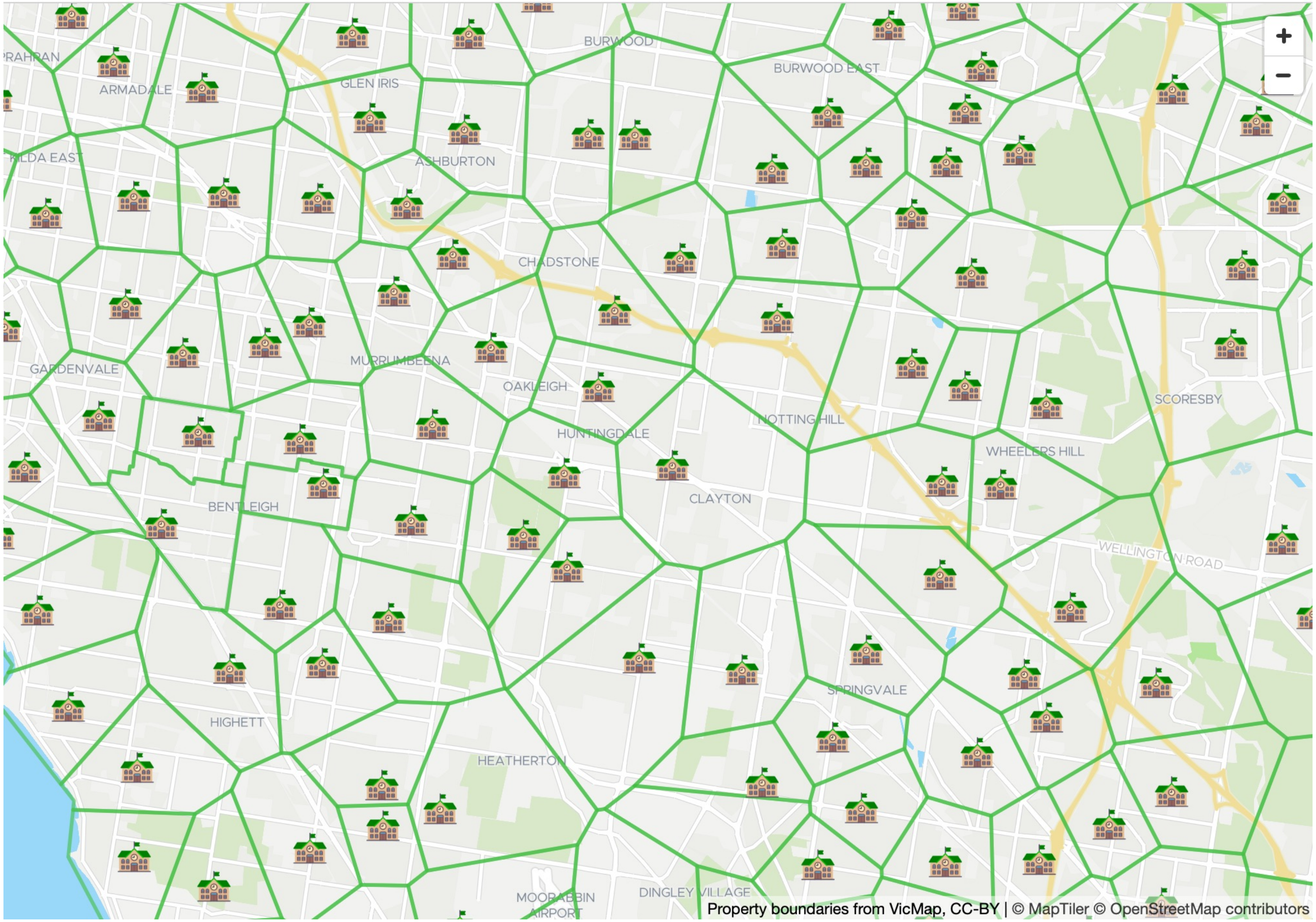




+ Voronoi Diagram Everywhere

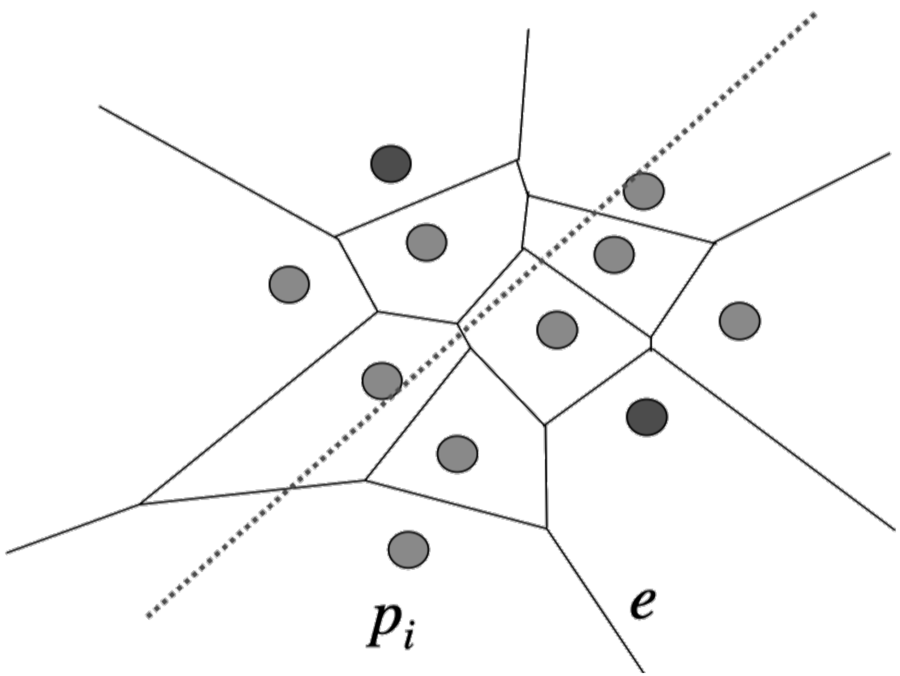


+ Voronoi Diagram Everywhere



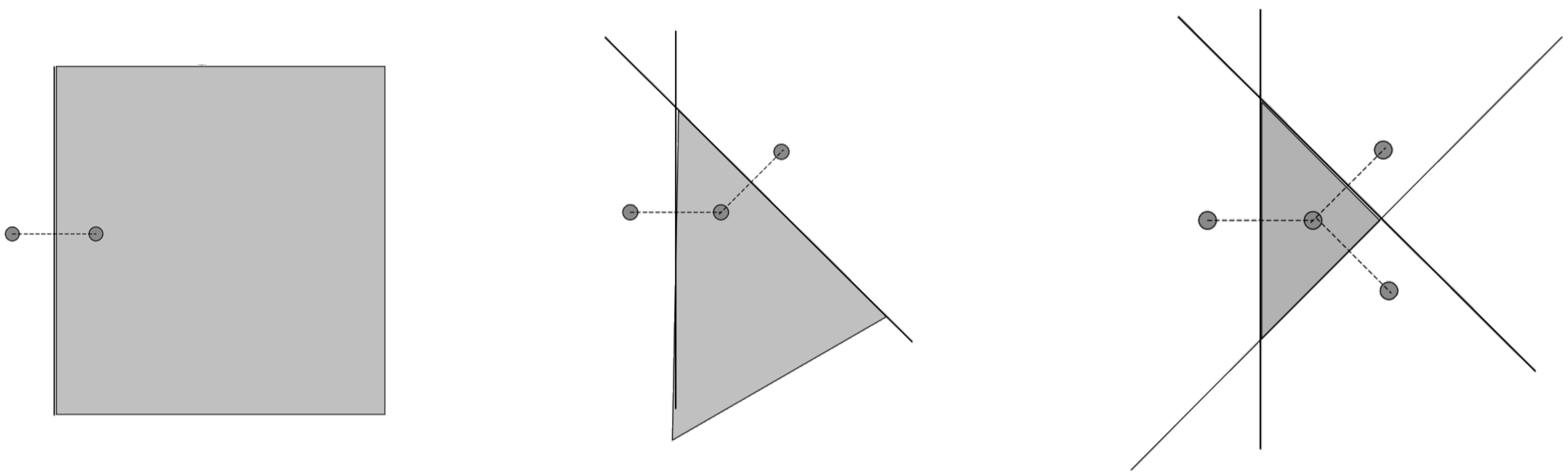
+ Voronoi Diagram

- Linear Size $|\mathrm{V}|, |\mathrm{E}| = O(n)$
  - Intuition: Not all bisectors are Voronoi edges
  - Euler's Formula: $|V| - |E| + f = 2$
    - $|V| \leq 2n - 5$
    - $|E| \leq 3n - 6$



+ Voronoi Diagram Construction

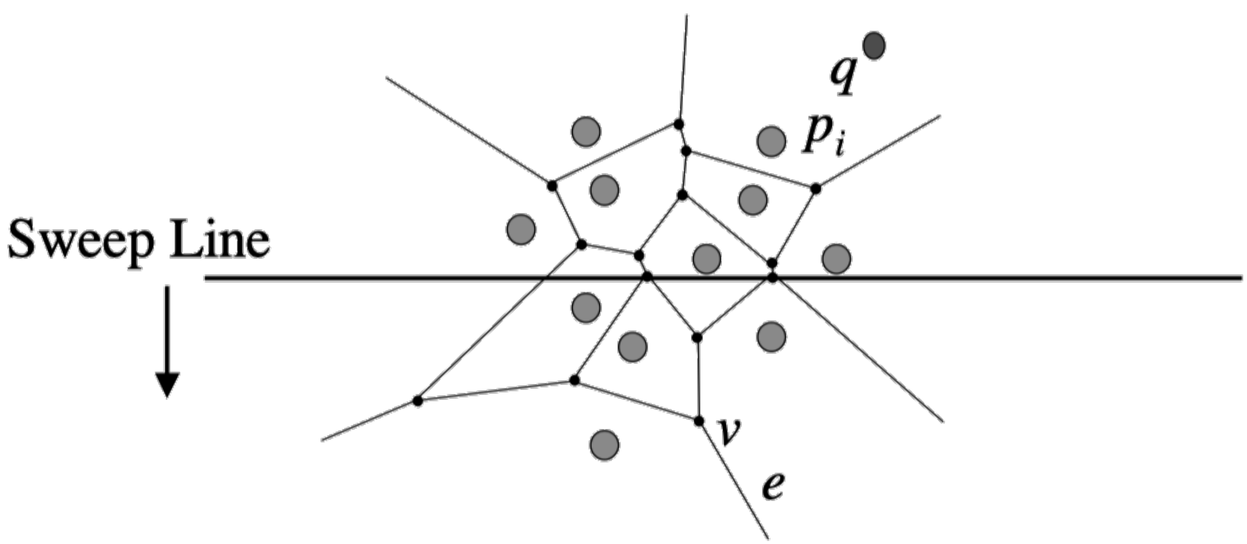
- Half Plane Intersection



- $O(n^2 log n)$

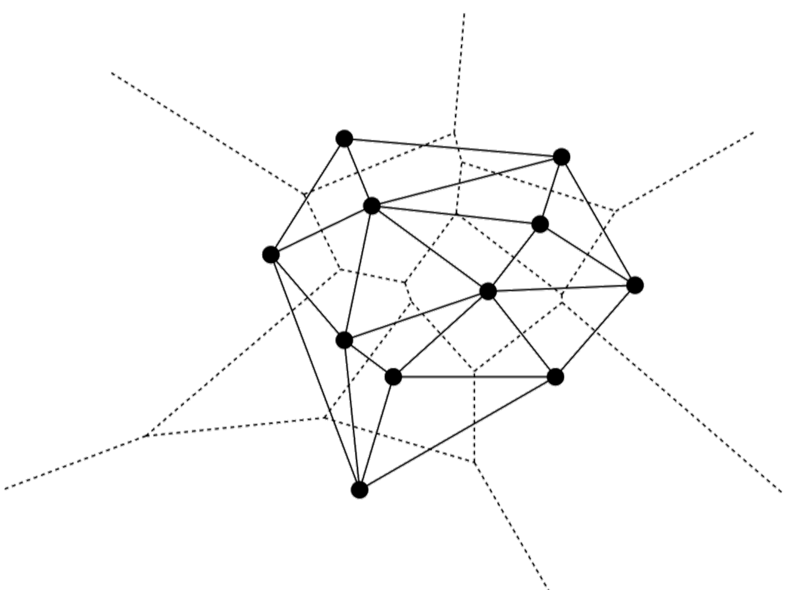
+ Voronoi Diagram Construction

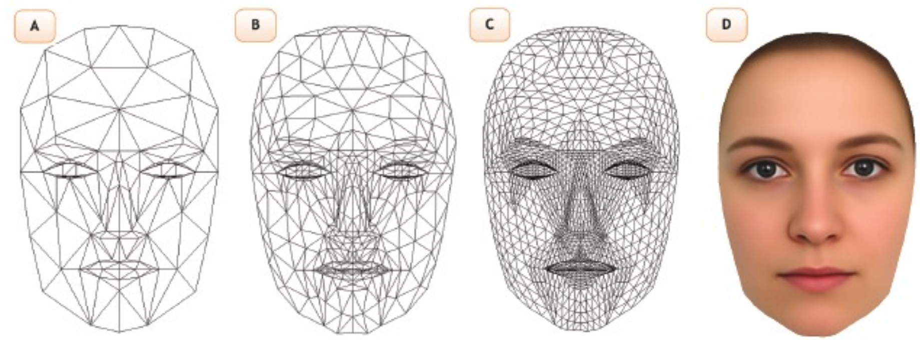
- Fortune's Algorithm
  - Sweep line
    - Horizontal line from top to bottom
    - Incremental construction
    - $O(nlogn)$

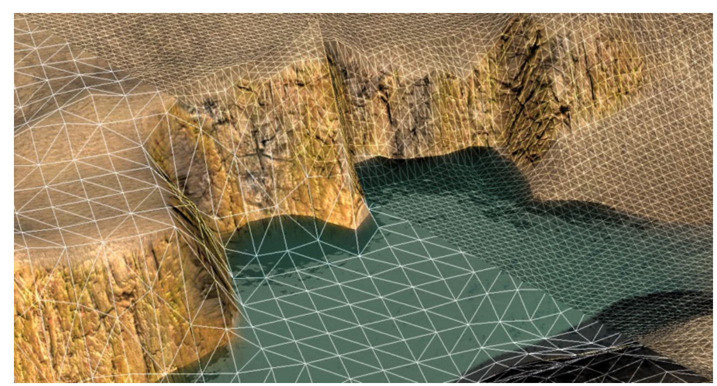


+ Voronoi Diagram

- Dual Problem: Delaunay Triangulation
  - Given a set P of discrete points in a plane, no point in P is inside of any triangle in $DT(P)$.







+ Advanced Spatial Queries

- Other types of spaces
  - Euclidean space – simple but not always realistic
  - Network space and surface space
    - Computing the distance between two points becomes very time consuming
- Other types of relationships
  - Point query and Range query
  - Selection query vs Join query
  - Nearest neighbor and skyline queries
    - Very different query processing strategies
    - With applications beyond spatial databases