Computational Geometry

Sunghee Choi

Computational Geometry

- Course Number : CS 504
  - Webpage : gclab.kaist.ac.kr/cs504
  - Board : noah.kaist.ac.kr/course/CS504
- Instructor : Prof. Choi, Sunghee
  - Office 3404 (Tel 3534)
  - Email : sunghee@kaist.edu
  - Office hour : M/W 2-3
- TA : Son, Jeongho
  - Office 3431 (Tel 7834)
  - Email : sonjh@kaist.ac.kr

Textbook

- Computational Geometry: Algorithms and Applications
  by Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars, Springer. 2nd or 3rd edition
- Prof. David Mount's course notes (available in course webpage)

References

- *Computational Geometry: An Introduction Through Randomized Algorithms*, by Mulmuley, Prentice Hall, 1994
- *Computational Geometry In C (Second Edition)*, by O'Rourke, Cambridge University Press, 1998.
- *Handbook on Discrete and Computational Geometry*, by Goodman and O'Rourke (eds), CRC Press LLC, 1997.
- *Algorithms in Combinatorial Geometry*, by Edelsbrunner, Springer-Verlag, 1987.
- *Computational Geometry (An Introduction)*, by Preparata and Shamos, Springer-Verlag, 1985.

Grading

- Homework 20 %
- Midterm 20%
- Attendance, presentation and participation 30 %
- Project 30 %

Course outline

- The first 1/3 of the course are lectures by instructor.
- The rest 2/3 will be presentations by students.
- We will randomly make teams of 2 students.
- Each team presents 1 course topic.
- 2-3 presentations by students.

Project

- Make a team of 2.
- Presentation/report at the end of the semester.
- List of possible topics will be available on the course homepage.
- Can be anything related to computational geometry, e.g.
  - Write a survey paper for a specific topic
  - Implement and present computational geometry research papers
  - Implementation and experimental study of geometric algorithms
  - New ideas for geometric problems of your interest.

Computational Geometry

- Design and analysis of algorithms for geometric problems (involving geometric inputs such as points, line segments, planes, triangles, circles, etc.)
- Developed from the field of algorithms in the late 70's
- Application domains : computer graphics, computer vision, image processing, robotics, computer-aided design and manufacturing, computational fluid-dynamics, computational chemistry, geographic information systems, sensor networks, databases…

Strengths

- Provide basic geometric tools needed from which applications can build their programs.
- Emphasis on “provably correct and efficient” algorithmic solutions to geometric problems.

Limitations

- Mainly deal
  - Discrete objects (vs. continuous)
  - Straight and flat objects (vs. curved/smooth)
  - Problems in low (2 or 3) dimensions.

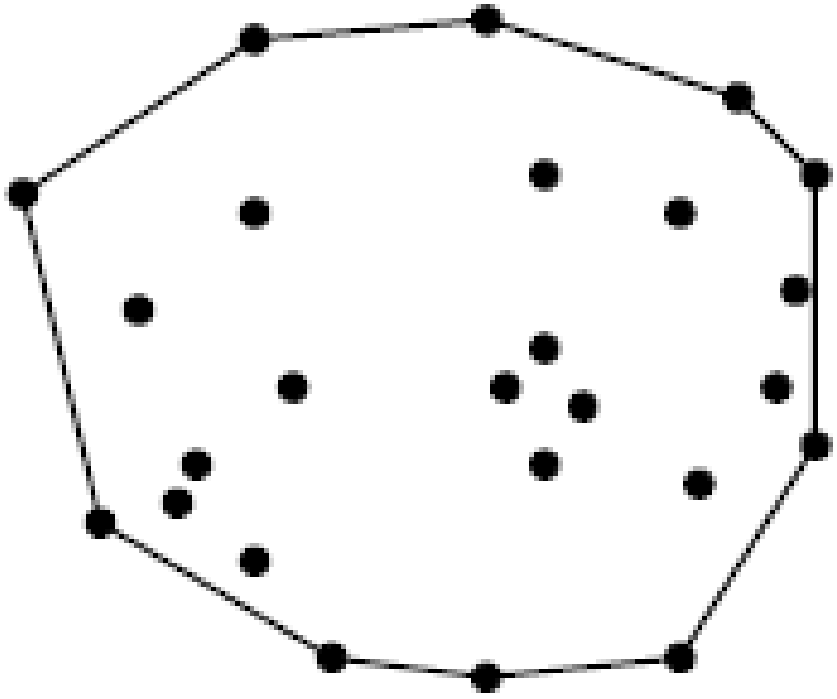
From theory to practice

- Many optimal algorithms in theory are hard to implement in practice.
- Many algorithms assume “general position” - Need to deal with “degeneracies”.
- Robustness – floating point arithmetic vs. exact arithmetic.

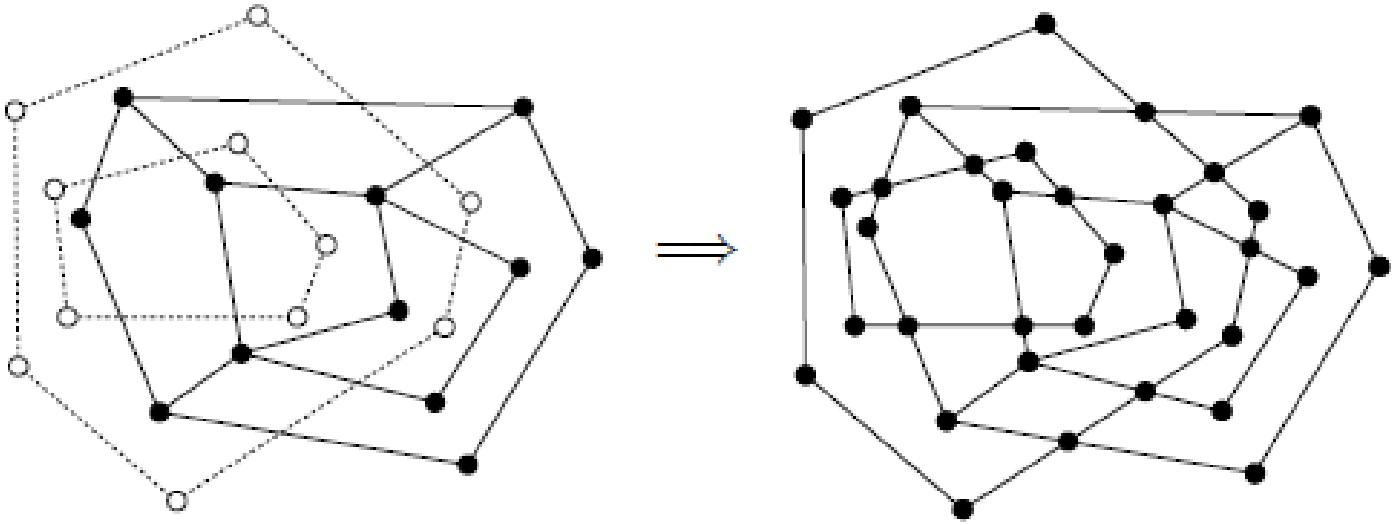
Topics

- Convex hulls
- Intersection
- Low-dimensional linear programming
- Range Searching – Kd-trees, range trees, fractional cascading
- Point location
- Voronoi diagram / Delaunay triangulation
- Power diagram / Regular triangulation
- Arrangements and duality
- Curve / surface reconstruction
- Mesh generation
- Other topics depending on the interests of the class

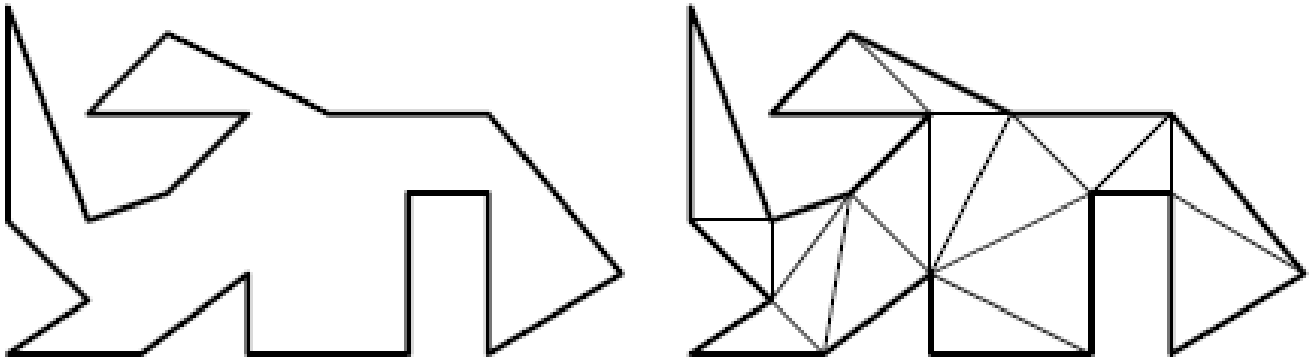
Convex Hull



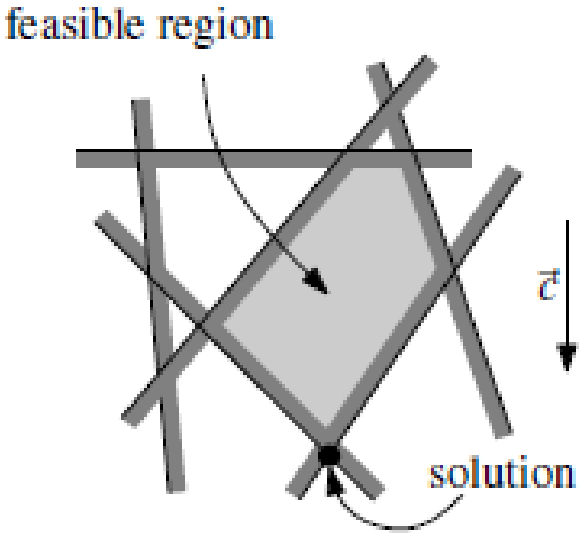
Intersection



Triangulation



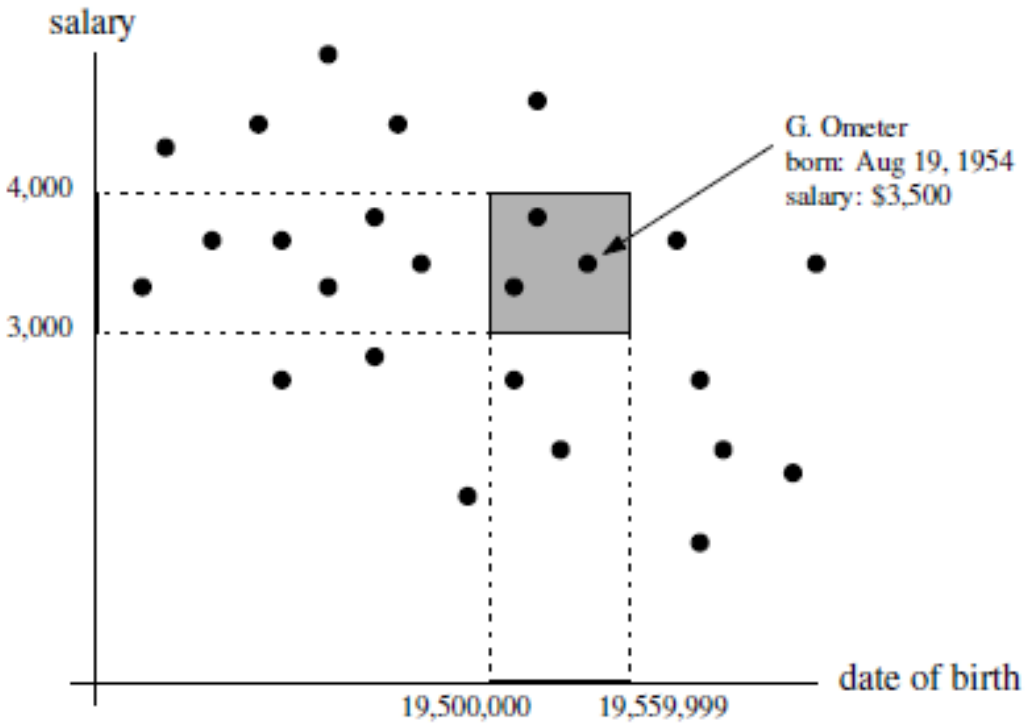
Linear Programming



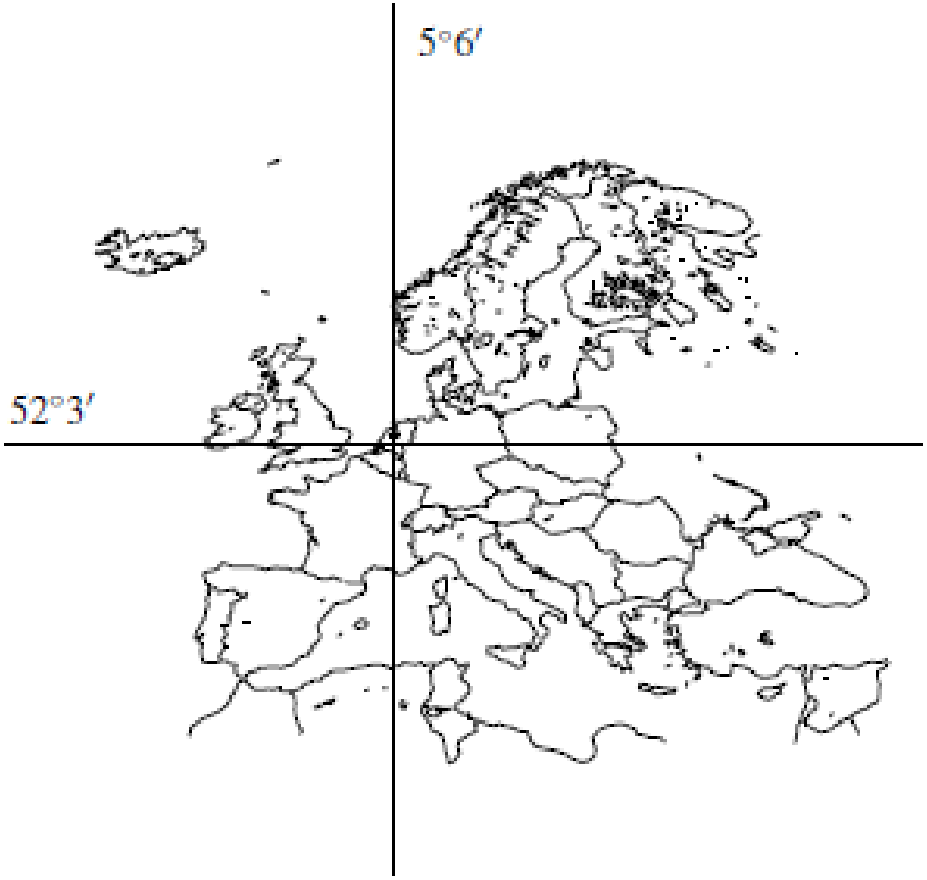
Geometric Search Problems

- Given a data set (e.g., points, lines, polygons) which will not change, preprocess this data set into a data structure so that query can be answered as efficiently as possible.
- Range searching
- Point location
- Nearest neighbor problem

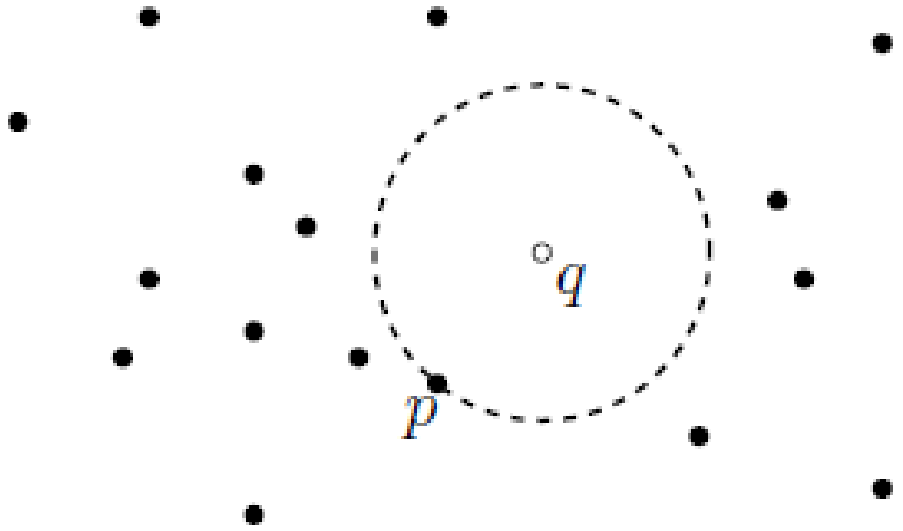
Range Searching



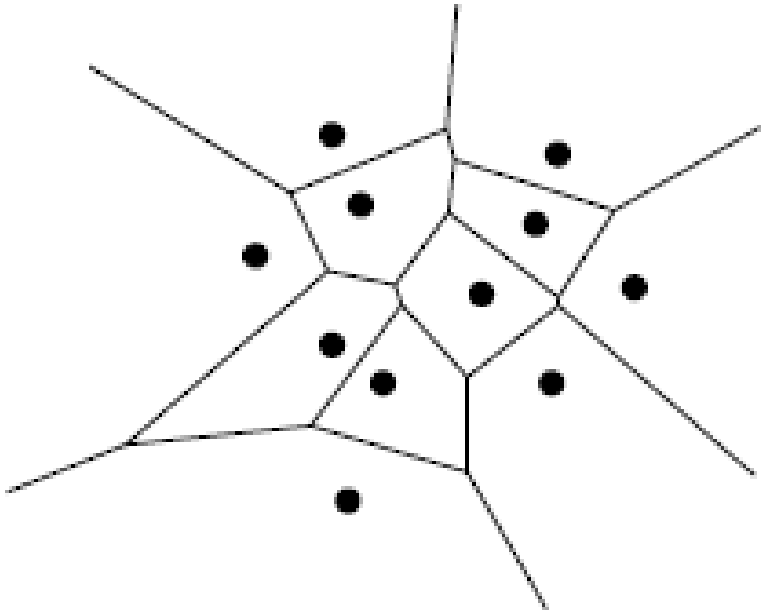
Point Location



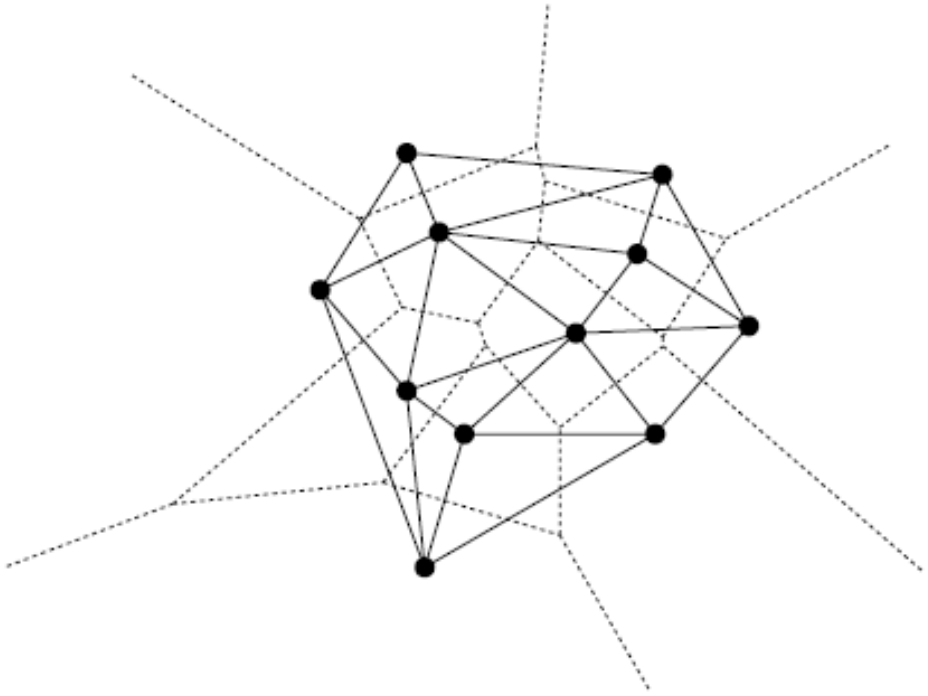
Nearest neighbor search



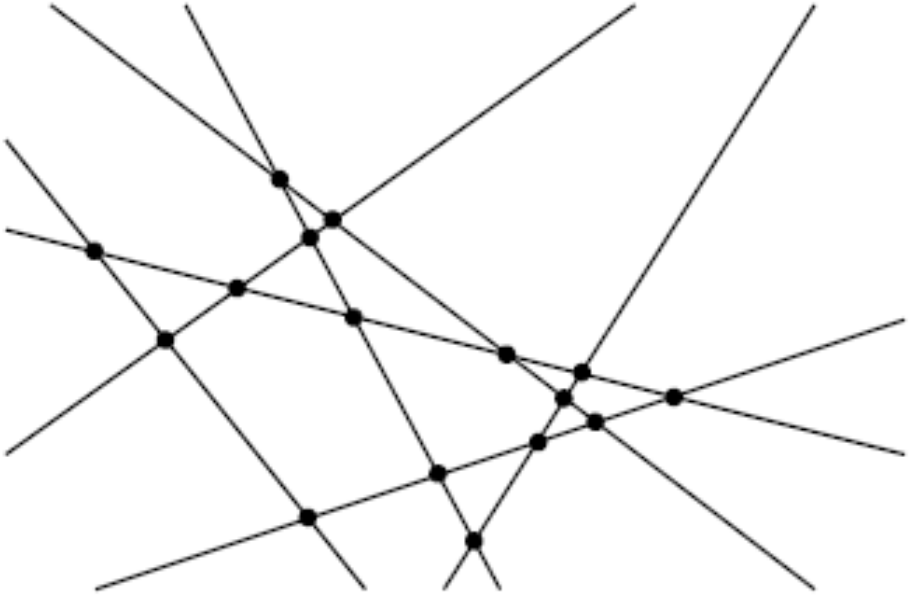
Voronoi diagram



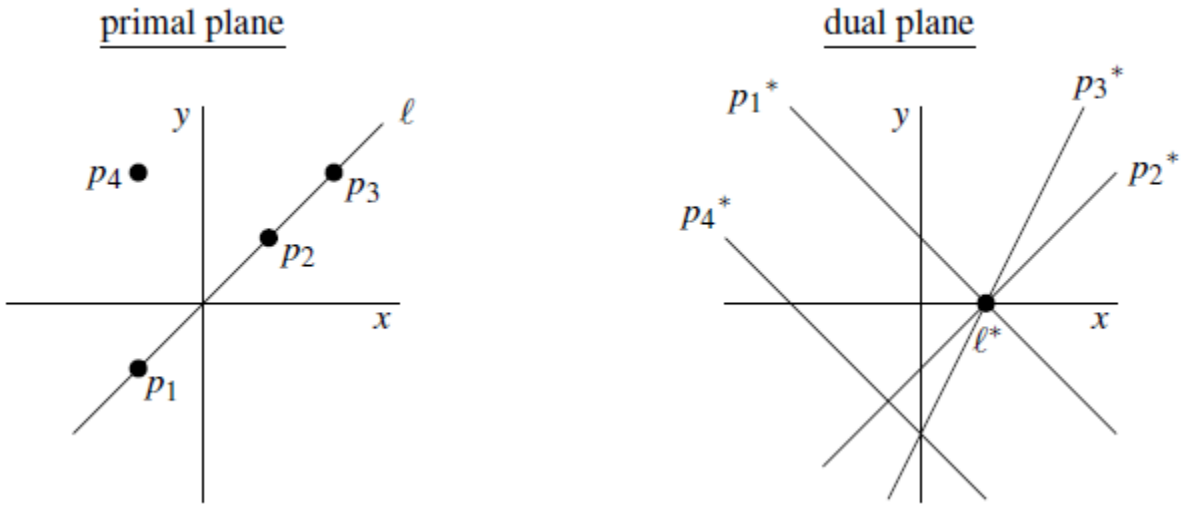
Delaunay triangulation



Arrangement

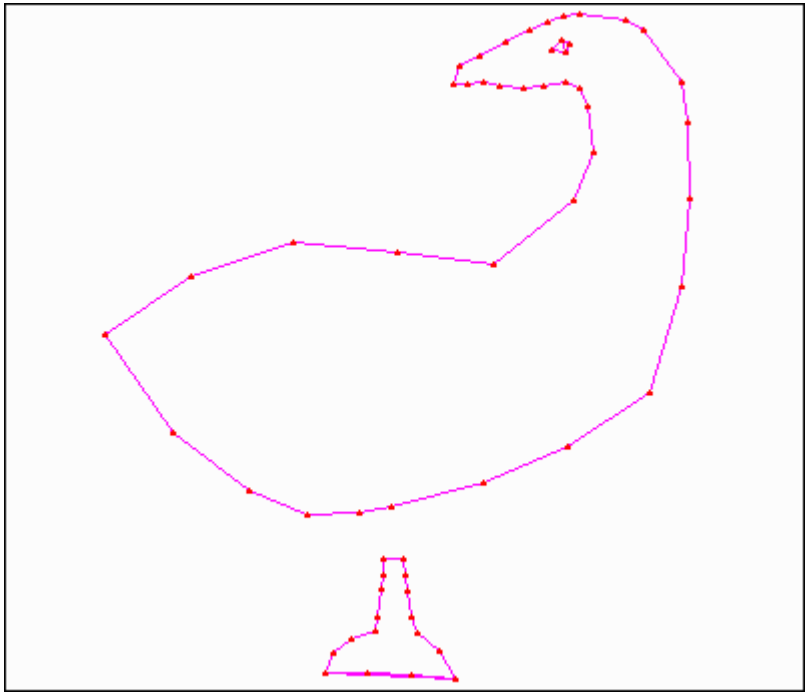


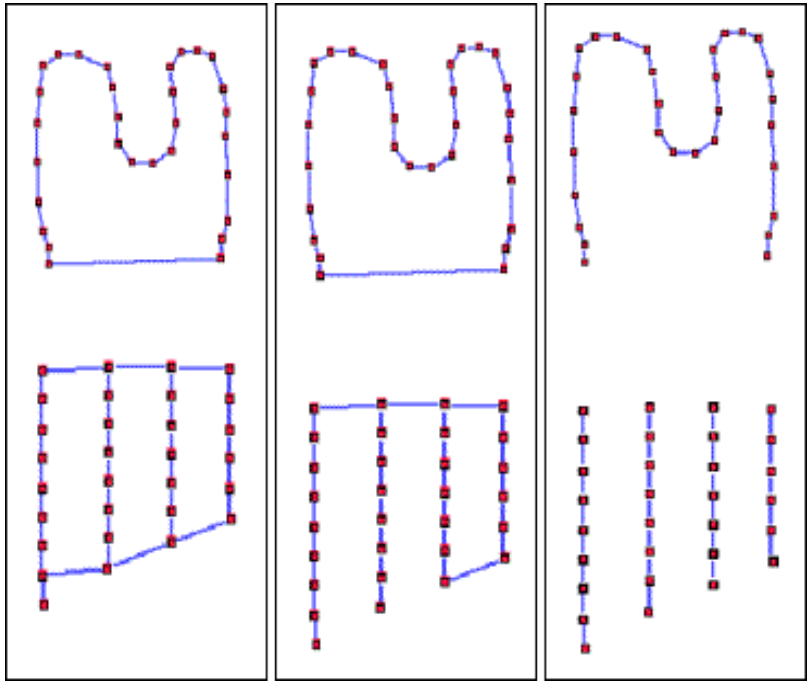
Duality



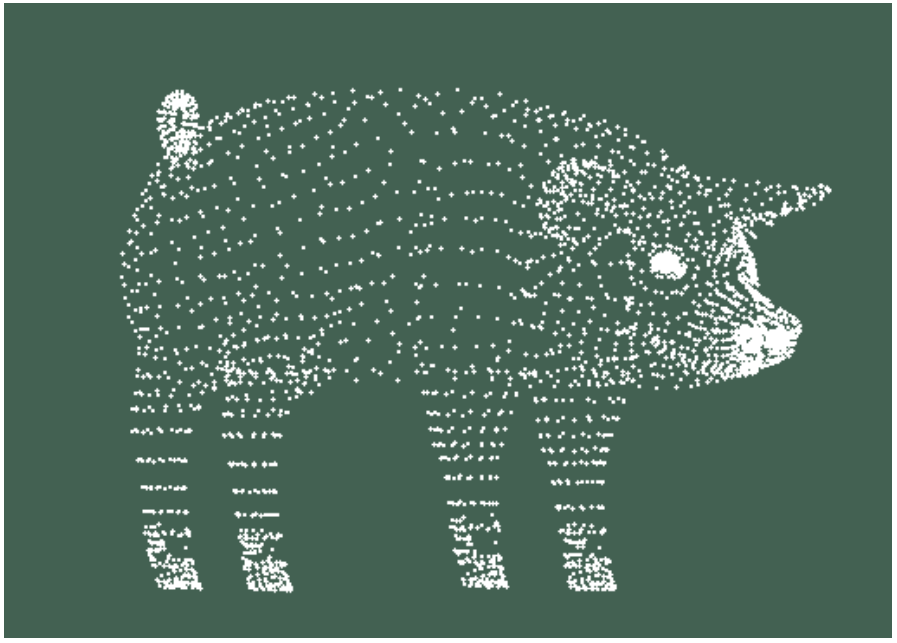
Curve reconstruction

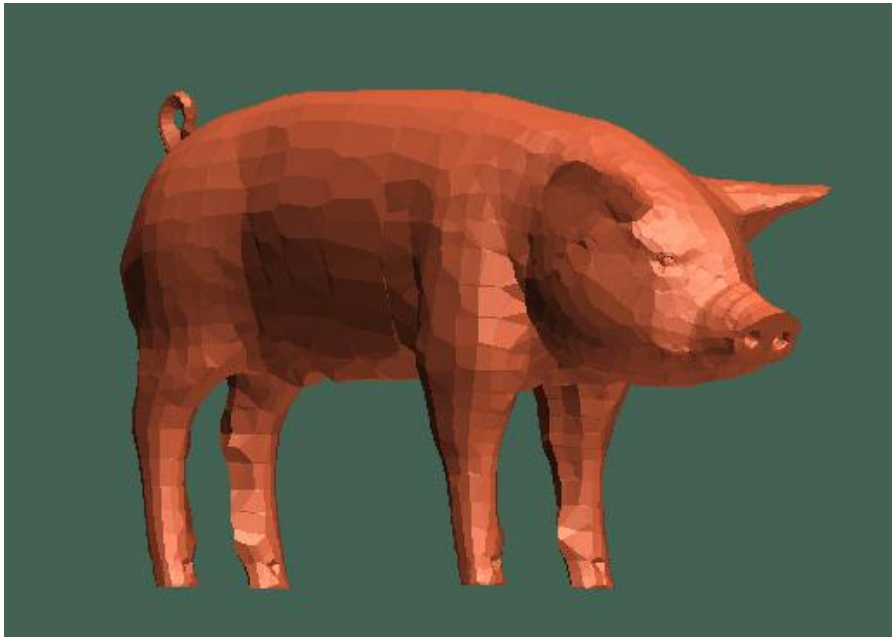
- Given a set of samples from a curve, we want to compute a polygonal reconstruction of the curve, i.e., points should be joined by edges in the order they appear on the curve





Surface Reconstruction



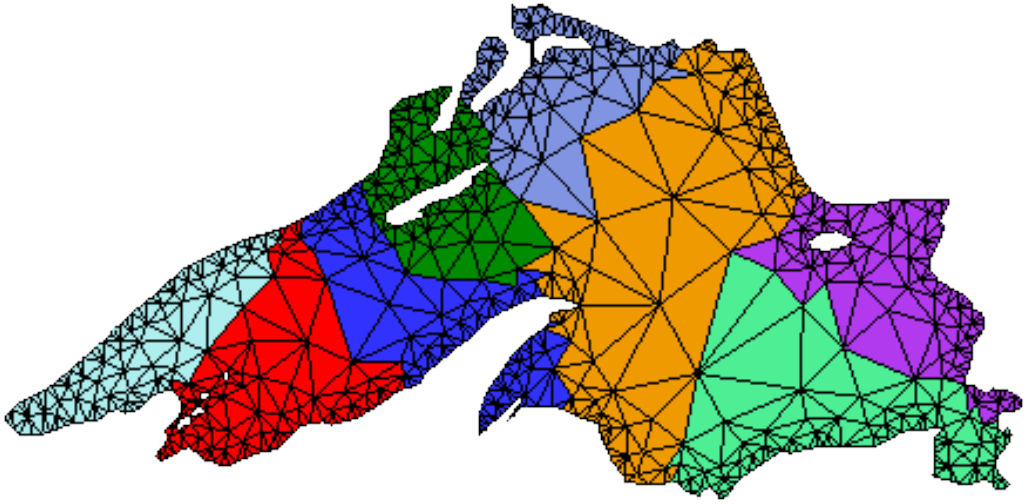


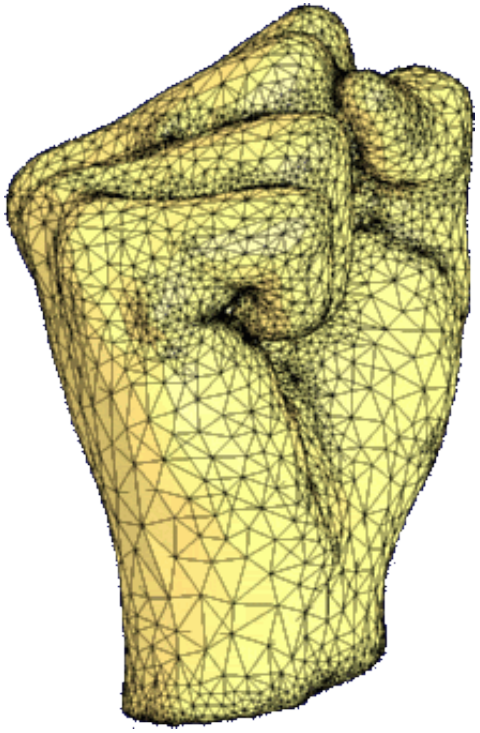
Input: *Samples* from an object surface.

Output: Polygonal model.

Mesh generation

- Given a domain (e.g., a polygon or polyhedron), partition it into simple “elements” meeting in well-defined ways.
- A key step of the finite element method for numerical computation.





Range Searching

Range Searching

Given a set P of geometric objects in E^d
and a query range(a connected region) q,
report (or count) all objects in P intersecting the range q.

- Objects : points, line segments, …
- Range : interval, rectangle, simplex,…

Point Enclosure

- Ser R of ranges in E^d
- Point enclosure query : given a query point q, report all ranges in R containing q.
- Dual of the range searching problem (also called *inverse orthogonal range searching*).
- For d=1, R : set of intervals.
- For d=2, R : set of rectangles.

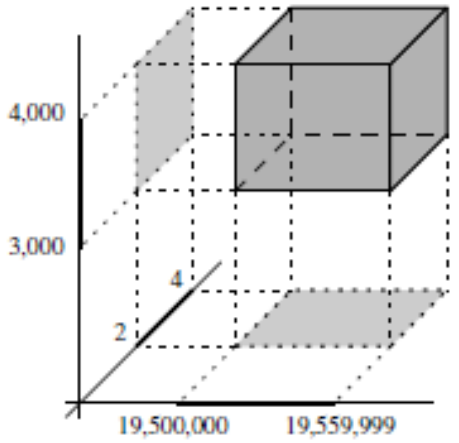
Point Location

- Let S be a planar subdivision with n edges.
- Planar point location problem : given a query point q, report the face f of S that contains q.

Orthogonal range searching

- Set P of points in d-dimensional space E^d.
- Range query : report the points of P contained in a query range r.
- Rectangular or orthogonal range query :

  $r = (a_1, b_1) \times (a_2, b_2) \times \ldots \times (a_d, b_d)$
- Collection of 1-d searches



Canonical subset

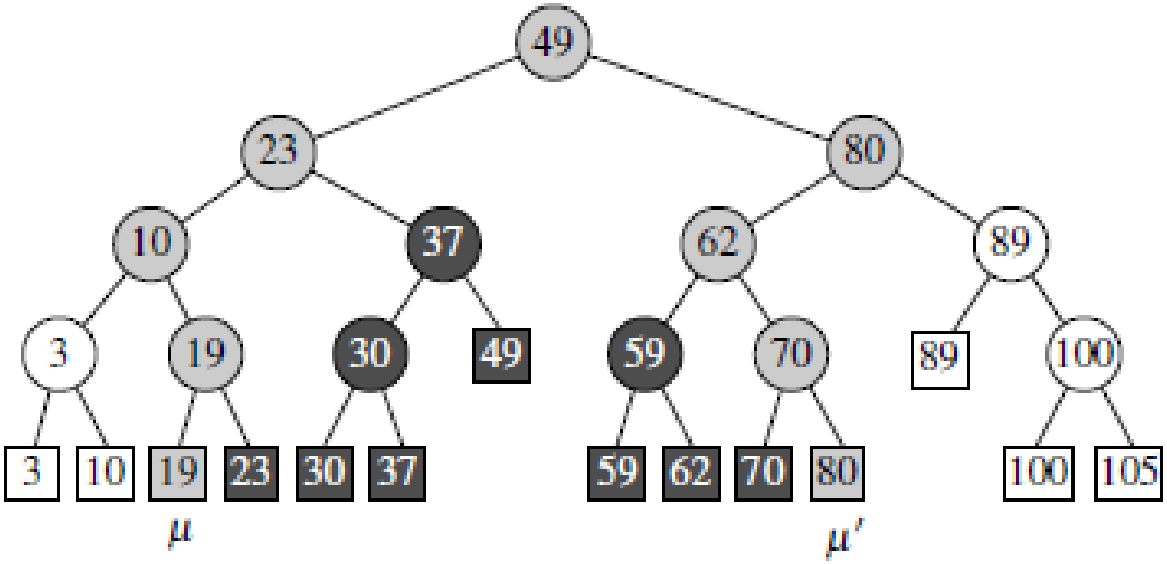
- A common approach to solve range queries is to represent P as a collection of *canonical subsets* $\{P_1, P_2 \ldots, P_k\}$, each $P_i \subseteq P$ s.t. any set can be formed as the disjoint union of canonical subsets.
- n singleton sets : O(n) space needed, k sets needed to answer a query with k objects.
- The power set of P : any query can be answered with a single canonical subset but we need 2^n subsets.
- Balance total number of canonical subsets (space) and number of canonical subsets needed to answer a query (time).

1-dimensional range searching

Given a set of points $P = \{ p_1, p_2, \ldots, p_n \}$ on the real line, and a query interval [x:x'], report all the points in P in [x:x'].

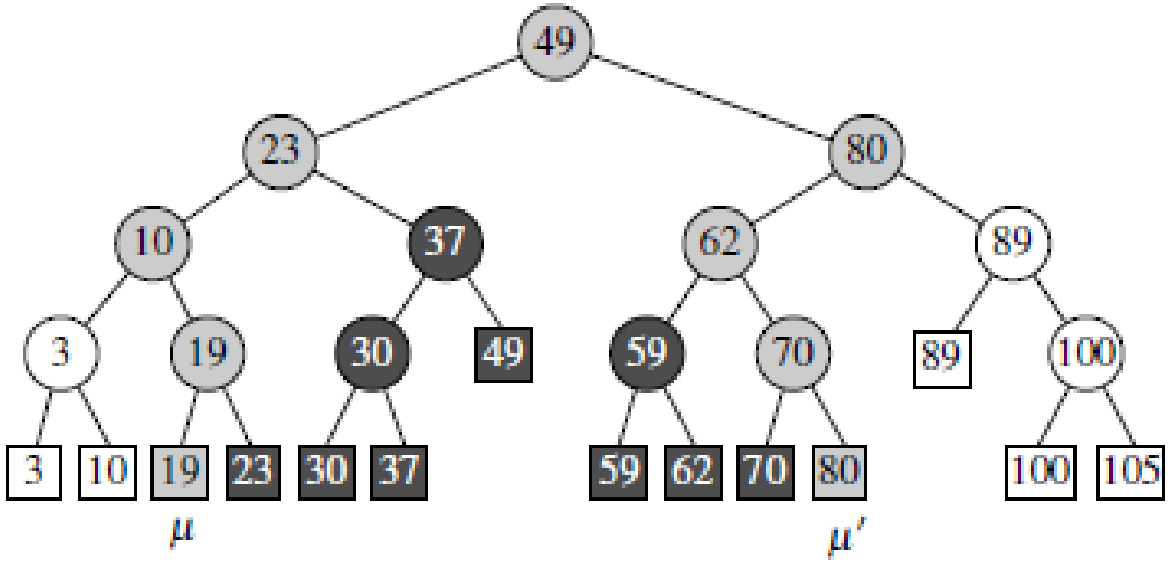
1-dimensional range searching

- Use a balanced binary search tree (internal node labeled with the largest key in its left subtree)
- Search for x and x'. (e.g. [18:77])
- How to report points in between?



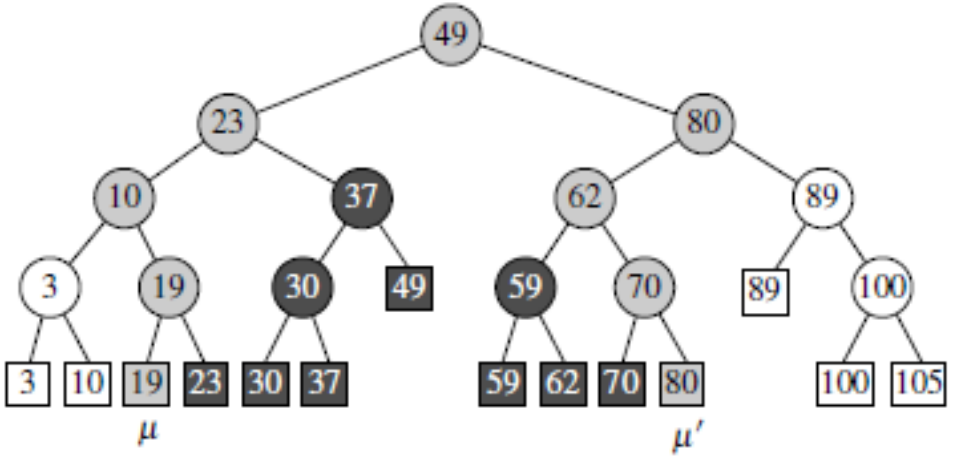
1-dimensional range searching

- What is the canonical subset of the balanced binary search tree?



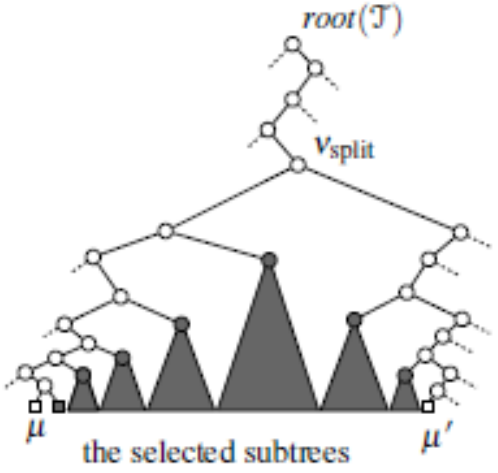
1-dimensional range searching

- Associate each node with the subset of points in its subtree → O(n) canonical subsets.
- The canonical subset for any range can be identified in O(log n) time.



1-dimensional range searching

- Search the tree to find the leftmost leaf μ and rightmost leaf μ'.
- The search paths to μ and μ' may share some common subpath.
- Once the paths diverge, as we follow the left path to μ, whenever the path goes to the left child, add the right child.
- Similarly, as we follow the right path to μ', whenever the path goes to the right child, add the left child.



1-dimensional range searching

- Space for n points : O(n)
- Preprocessing : O(n log n) time
- Reporting query : O(log n + k) where k is the number of points reported.
- Counting query : store total number of points in each subtree (preprocessing) and sum all these over canonical subsets ⇒ O(log n) time.
- Time for insertion/deletion of a point : O(log n)

1-dimensional range searching

- Space for n points : O(n)
- Preprocessing : O(n log n) time
- Reporting query : O(log n + k) where k is the number of points reported.
- Counting query : store total number of points in each subtree (preprocessing) and sum all these over canonical subsets ⇒ O(log n) time.
- Time for insertion/deletion of a point : O(log n)

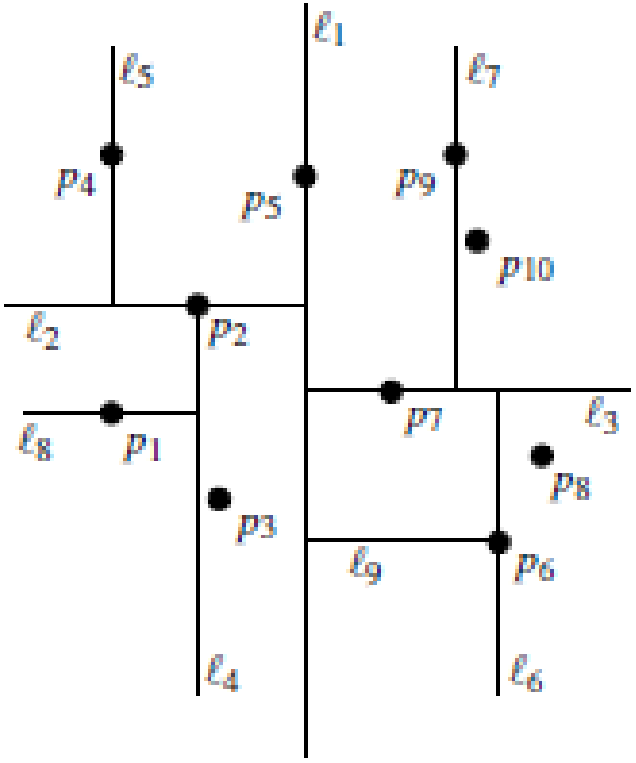
2-dimensional range searching

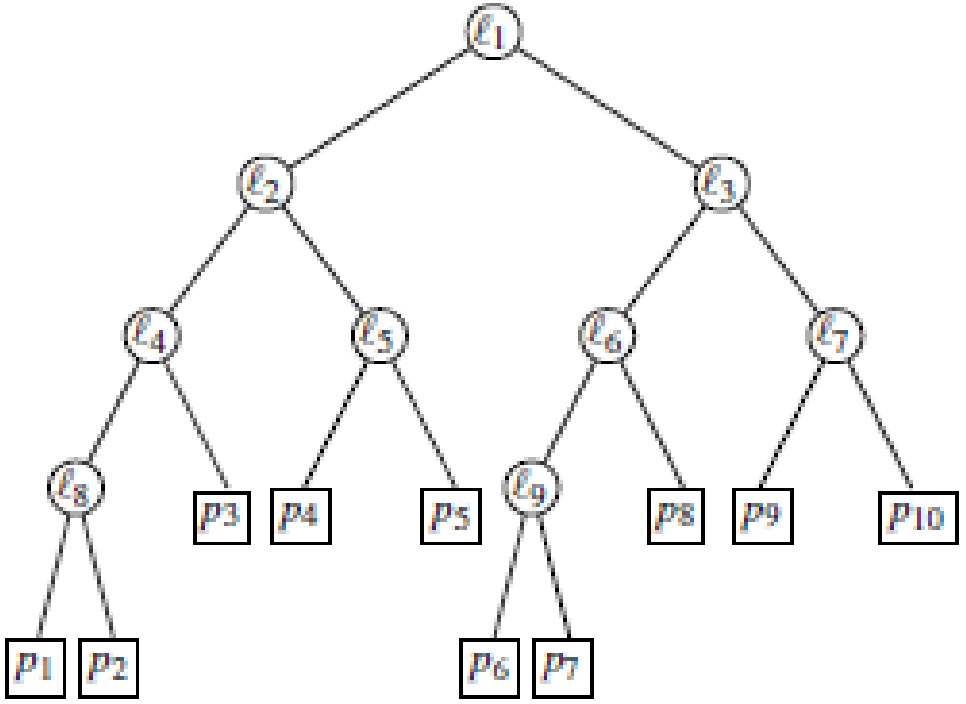
- Query range : rectangles
- How to extend binary search tree?

Kd-trees

- Split on x-coordinate, next on y-coordinate, then again on x-coordinate, and so on.
- Easy to implement and practical (but not asymptotically optimal).

Kd-tree


ℓ_1
ℓ_5
ℓ_7
p_4
p_5
p_9
p_{10}
ℓ_2
p_2
ℓ_8
p_1
p_7
ℓ_3
p_8
p_3
ℓ_9
p_6
ℓ_4
ℓ_6


ℓ_1
ℓ_2
ℓ_3
ℓ_4
ℓ_5
ℓ_6
ℓ_7
ℓ_8
ℓ_9
p_3
p_4
p_5
p_8
p_9
p_{10}
p_1
p_2
p_6
p_7

Kd-trees

- How to build?
- How to query?

Constructing kd-tree

- Top-down recursive procedure
- Maintain 2 lists (sorted by x and y).
- Find median (O(1)).
- Split list (O(n))
- $T(n) = 2\,T(n/2) + n = O(n \log n)$ time
- O(n) space.

Range searching in kd-trees

- Let Q denote the desired range.
- Let t denote the current node.
- Let C denote the rectangular cell associated with t.
- Traverse the tree recursively.
- At a leaf, check whether its point lies in the range Q in O(1) time.
- At an internal node, if C is disjoint with Q, do nothing. If C is contained within Q, report every point in its subtree. If C partially overlaps Q, recurse on t's 2 children.

Analysis of query time

- Time to traverse a subtree and report the points in its leaves is O(k).
- Need to bound the number of nodes visited that are not in one of the traversed subtrees.
- For each such node, its cell overlaps the range but is not contained.
- We say that such a cell is *stabbed* by the query.

Analysis of query time

- Lemma : Given a balanced kd-tree with n points using alternating splitting rule, any vertical or horizontal line stabs $O(\sqrt{n})$ cells of the tree.
- Extend 4 sides of Q into lines $\Rightarrow$ total number of cells stabbed by all 4 lines is at most $O(\sqrt{n})$
- Since we only make recursive calls when a cell is stabbed, total number of nodes visited is $O(\sqrt{n})$
- Given a balanced kd-tree with n points, reporting query takes $O(\sqrt{n} + k)$ time where k is the number of reported points and counting query takes $O(\sqrt{n})$.
- d-dimensional kd-tree?

Can we make query time faster?