Computational Geometry

Complexity Notions

- algorithm
- time, space
- complexity bounds
  - $O(n^2)$, … (upper bounds)
  - $\Omega(n \log n)$, … (lower bounds)
  - $\Theta(n)$, … (precise bounds)
  - common bounds:
    - log n (logarithmic),
    - n (linear),
    - n log n,
    - n^2 (quadratic),
    - c^n (exponential)

Examples:
- linear search
- binary search
- shortest path
- traveling salesman
- indexing
- quantifier elimination

Sorting

- Can sort n numbers in time O(n log n)
  - which algorithms do this? Deterministic?
  - average versus worst case complexity
- Need $\Omega(n \log n)$
  - decision tree argument
- lower bounds are rare and typically apply to restricted models
- searching: preprocessing vs processing time

Output Lower Bounds

Example: *line segment intersection*

- how many intersections can n line segments have?
- would that make a fair lower bound in all cases?
- can be done in
  O(n log n + k) time and O(n) space,
  where k = # intersections
- output sensitive complexity

Basic Data Structures

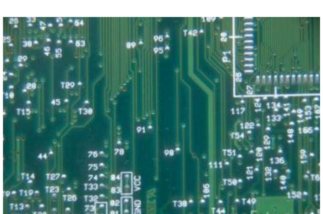
- binary search tree
  - build in time O(n log n), storage O(n)
  - search in O(log n)

  (http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm)

- dynamic binary search trees (red/black, AVL)
  - build in time O(n log n), storage O(n)
  - search, insert, delete in O(log n)

Sample Problem:
Windowing a Circuit

- simplify problem: only straight horizontal/vertical lines (orthogonal layout)
- report all points/line segments with parts in the window



- naïve algorithm ?
- better solution ?
- what is needed ?

1d Range Search

Input: $\{x_0, \ldots, x_n\}$, points on the line,
interval x, x'
Output: $\{x_0, \ldots, x_n\} \cap [x, x']$

Data structure: binary search tree
- O(n log n) construction
- O(n) space, O(k + log n) time for query
- can be made more efficient by storing "canonical sets" of leaves

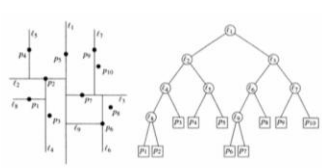
2d Range Search

Input: $\{x_0, \ldots, x_n\}$, points in the plane,
rectangle R := [x, x'] x [y, y']
Output: $\{x_0, \ldots, x_n\} \cap R$

How can we solve this ?

kd-Tree

- alternate 1d-strategy for x/y
- split point-set at median value (divide & conquer)
- e.g. http://homes.ieu.edu.tr/~hakcan/projects/kdtree/kdTree.html
- what's construction time/storage?

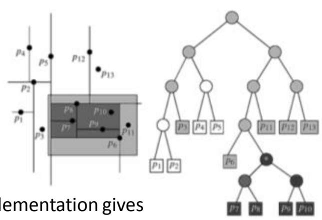


- how long to determine whether a point belongs to the set ?

kd-Tree

- simple implementation gives
  - O(n log n) construction
  - O(log n) point query
  - O(n) storage
- what about region (rectangle) query ?

kd-Tree range search



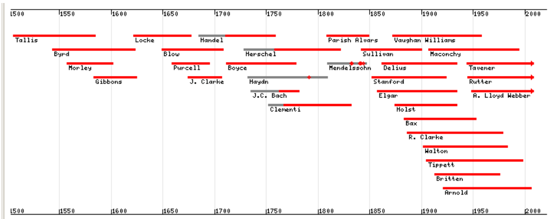
- simple implementation gives
  - O(sqrt(n) + k) query
    - look at horizontal/vertical lines, how many regions can they intersect
    - animations: http://www.cs.cmu.edu/~awm/animations/kdtree/
- can be improved to O($\log^2$ n + k) using range trees and O(log n + k) using fractional cascading, storage increased to O(n log n)

Circuit Windowing, 1st step

- can reports all points in window
- what else ?
- what's left ?

Interval Tree

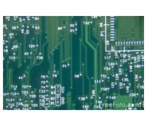
- answers stabbing queries for axis-parallel line (segments)
- imagine intervals on a line:



Interval Tree

- find median
  - store intervals containing the median in root
  - store twice: 1) ordered by left end-point, 2) ordered by right endpoint
  - store intervals to left/right in left/right subtree with same recursive structure
- stabbing query in time O(log n + k)
- construction: O(n log n), storage O(n)

Circuit Windowing, 2nd step



- line segments intersecting window:
  - inside
  - overlap using stabbing

    problem: interval trees for infinite lines, not line segments, replace lists in roots with ?
- overall analysis:
  - query
  - storage
  - construction

Interval Intersection

Input: a set of n intervals, an interval [x,x’]
Output: list all intervals intersecting [x,x’]

E.g. which English composers could Wagner (1813-1883) have met?

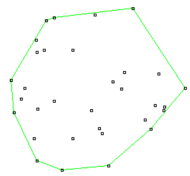
Interval Intersection

- use interval tree
- intersection query for [x,x’]:
  - find node f with f.median in [x,x’], let P be path from root to f
  - from f continue as if running two stabbing queries for x and x’, let paths be Q and Q’
  - for all nodes in P, Q, Q’ report intervals containing x or x’
  - for Q: report all intervals in right subtrees
  - for Q’: report all intervals in left subtrees

General Strategies

- Incremental
- Divide & Conquer
- Line sweep
- Randomization

(I) Convex Hull



- naïve algorithm
  assume:
  - test whether two line segments intersect
  - test whether a point is to the right of a line segment

  how to improve this approach ?

Convex Hull (Incremental)

- incremental convex hull
  - $O(n^2)$
  - http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html

  how can we improve this ?

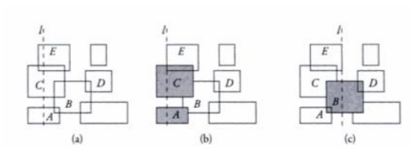
- used presorted input
  - convex hull algorithm in O(n log n)
  - update points of tangency on upper/lower chain

(II) Intersection of Half-planes

- assume we can intersect two convex polygons with p and p' vertices in time O(p+p')
- how do we compute the intersection of n half-planes? analyze:
  - naïve
  - divide & conquer

(III) Rectangle Intersection

plane sweep approach
- event list (vertical line sweeping across plane)
- active elements list

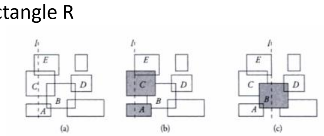


what do we need to implement this plane sweep ?

Rectangle Intersection

L := {}
for each event e in E (ordered)
  if e is start of rectangle R
    L := L u {R}
    report intersections of [R.y1, R.y2] with
    y-ranges of active intervals
  if e is end of rectangle R
    L := L – {R}

Analysis



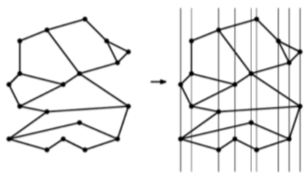
(iv) Point Location

Given a planar map, find the face you are in.

Point Location

partition into slabs

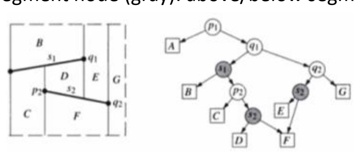
- time: O(log n), but space ?
- how can we improve storage, keeping querying time low ?



Trapezoidation

build search structure adding segments incrementally

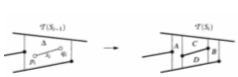
- x-node (white): left/right of point
- segment node (gray): above/below segment

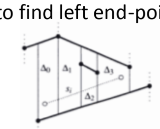


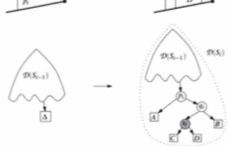
how do we add a segment ?

Point Location: adding a segment

- find trapezoids that intersect segment
  - use search data structure to find left end-point, follow segments to right







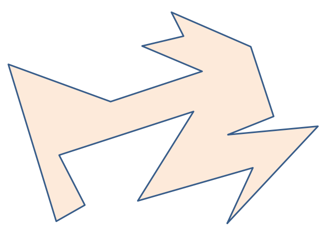
extend data structure and to include new trapezoids

Point Location via Trapezoidation

- based on order of added line segments, storage and query time can vary significantly
- determining optimal order hard
- however, random order will do with high probability:
  - expected storage: O(n)
  - expected construction time: O(n log n)
  - expected query time: O(log n)

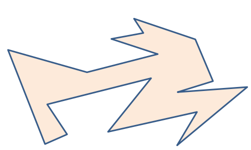
Excursion: Art Gallery

How many guards do you need to guard a museum?



Art Gallery Theorem (Chvatal, Fisk)

Every art gallery (simple polygon) on *n* vertices can be guarded by *n*/3 guards. This bound is optimal.



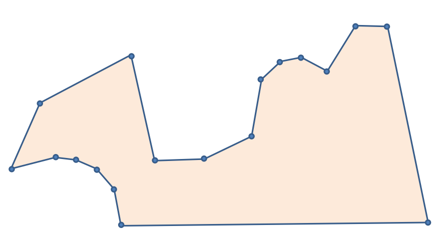
Proof of Art Gallery Theorem

- triangulate (how ?)
- show that triangulated graph can be 3-colored (no two adjacent vertices have the same color)
  hint: dual graph (the graph connected the triangles) is a tree
- select smallest color class

Triangulation

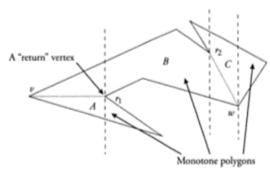
- easy: convex polygons
- how about monotone polygons ?
- strategy:
  - split polygon into monotone polygons
  - triangulate monotone polygons

Triangulate Monotone Polygon



Splitting into Monotone Polygons

- construct trapezoidation
- “turn” vertex: both neighbors on same side (with respect to x) and angle > 180º.
- remove turn vertex by connecting it to other vertex in its trapezoid



Bibliography

de Berg, van Krefeld, Overmars, Schwarzkopf, Computational Geometry, Algorithms and Applications, Springer 1997.
http://books.google.com/books?id=C8zaAWuOIOcC

Preparata, Shamos, Computational Geometry, An Introduction, Springer, 1985
http://books.google.com/books?id=gFtvRdUY09UC

Goodman, O’Rourke, Handbook of Discrete and Computational Geometry, CRC Press, 2004
http://books.google.com/books?id=X1gBshCclnsC