CSEP 521
Applied Algorithms
Spring 2005

Computational Geometry

Reading

- Chapter 33

Outline for the Evening

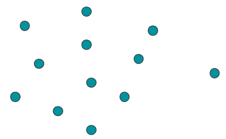
- Convex Hull
- Line Segment Intersection
- Voronoi Diagram

Geometric Algorithms

- Algorithms about points, lines, planes, polygons, triangles, rectangles and other geometric objects.
- Applications in many fields
  - robotics, graphics, CAD/CAM, geographic systems

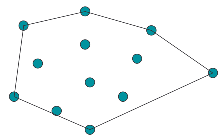
Convex Hull in 2-dimension

- Given n points on the plane find the smallest enclosing curve.



Convex Hull in 2-dimension

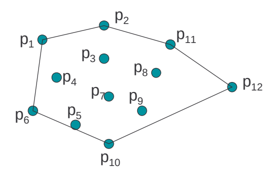
- The convex hull is a polygon whose vertices are some of the points.



Definition of Convex Hull Problem

- Input:
  Set of points $p_1, p_2, \ldots, p_n$ in 2 space. (Each point is an ordered pair p = (x,y) of reals.)
- Output:
  A sequence of points $p_{i1}, p_{i2}, \ldots, p_{ik}$ such that traversing these points in order gives the convex hull.

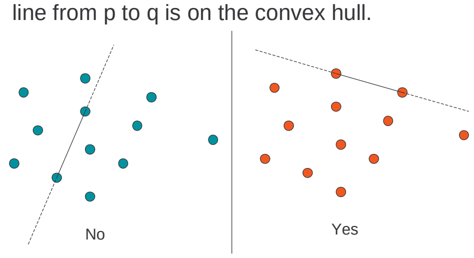
Example



Input: $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_8, p_9, p_{10}, p_{11}, p_{12}$
Output: $p_6, p_1, p_2, p_{11}, p_{12}, p_{10}$

Slow Convex Hull Algorithm

- For each pair of points p, q determine if the line from p to q is on the convex hull.



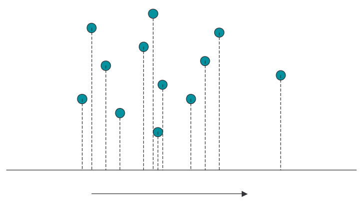
Slow Convex Hull Algorithm

- For each pair of points p, q, form the line that passes through p and q and determine if all the other points are on one side of the line.
  - If so the line from p to q is on the convex hull
  - Otherwise not
- Time Complexity is $O(n^3)$
  - Constant time to test if point is on one side of the line from (p_1,p_2) to (q_1,q_2).

$$0 = (q_2 - p_2)x + (p_1 - q_1)y + p_2q_1 - p_1q_2$$

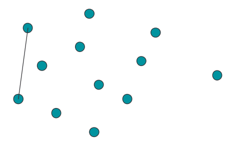
Graham's Scan Convex Hull Algorithm

- Sort the points from left to right (sort on the first coordinate in increasing order)



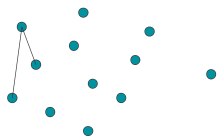
Convex Hull Algorithm

- Process the points in left to right order



Convex Hull Algorithm

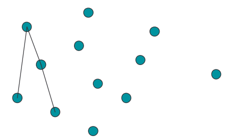
- Right Turn



Lecture 8 - Computational Geometry 13

Convex Hull Algorithm

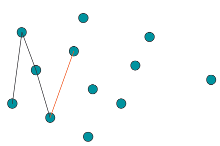
- Right Turn



Lecture 8 - Computational Geometry 14

Convex Hull Algorithm

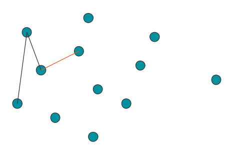
- Left Turn – back up



Lecture 8 - Computational Geometry 15

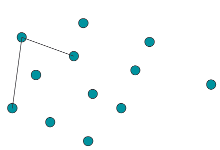
Convex Hull Algorithm

- Left Turn – back up



Lecture 8 - Computational Geometry 16

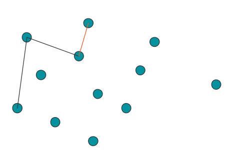
Convex Hull Algorithm



Lecture 8 - Computational Geometry 17

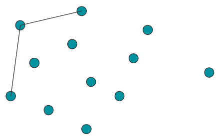
Convex Hull Algorithm

- Left Turn – back up



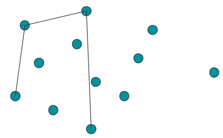
Lecture 8 - Computational Geometry 18

Convex Hull Algorithm



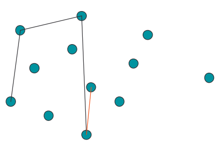
Convex Hull Algorithm

- Right Turn

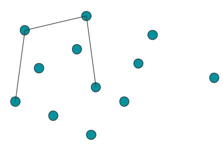


Convex Hull Algorithm

- Left Turn – back up

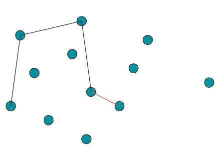


Convex Hull Algorithm

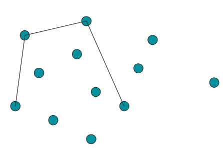


Convex Hull Algorithm

- Left Turn – back up

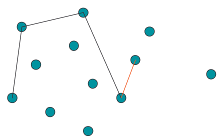


Convex Hull Algorithm

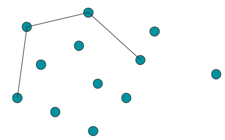


Convex Hull Algorithm

- Left Turn – back up

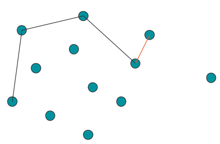


Convex Hull Algorithm

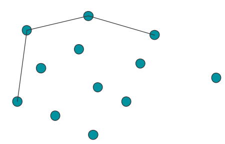


Convex Hull Algorithm

- Left Turn – back up

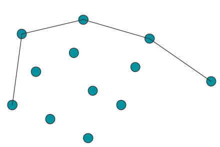


Convex Hull Algorithm



Convex Hull Algorithm

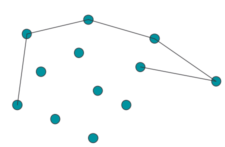
- Upper convex hull is complete



Continue the process in reverse order to get the lower convex hull

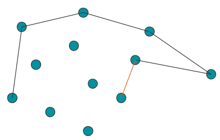
Convex Hull Algorithm

- Right Turn

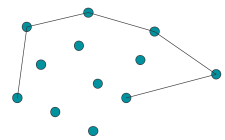


Convex Hull Algorithm

- Left Turn – back up

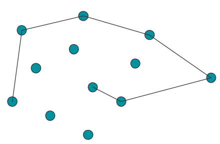


Convex Hull Algorithm



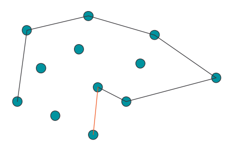
Convex Hull Algorithm

- Right Turn



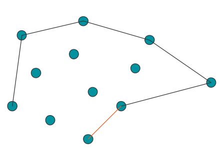
Convex Hull Algorithm

- Left Turn – back up

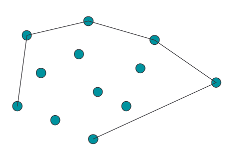


Convex Hull Algorithm

- Left Turn – back up

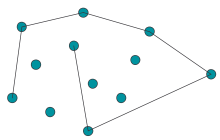


Convex Hull Algorithm



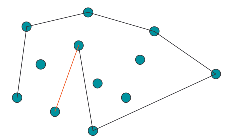
Convex Hull Algorithm

- Right Turn

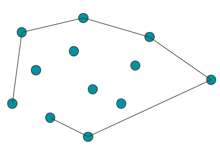


Convex Hull Algorithm

- Left Turn – back up

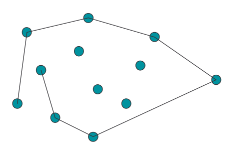


Convex Hull Algorithm



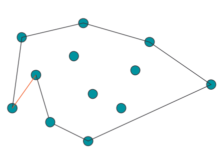
Convex Hull Algorithm

- Right Turn



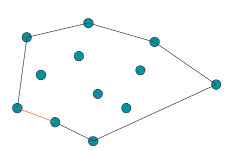
Convex Hull Algorithm

- Left Turn – back up



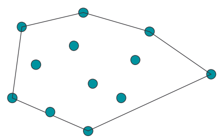
Convex Hull Algorithm

- Left Turn – back up



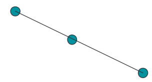
Convex Hull Algorithm

- Done!



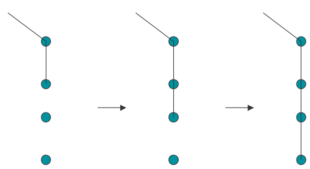
Co-linear Points

- Not a left turn
  - Middle point is included in the convex hull



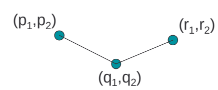
Vertical Points

- Sort
  - First increasing in x
  - Second decreasing in y



Testing For Left Turn

- Slope increases from one segment to next



left turn $\frac{q_2 - p_2}{q_1 - p_1} < \frac{r_2 - q_2}{r_1 - q_1}$

$(q_2 - p_2)(r_1 - q_1) < (r_2 - q_2)(q_1 - p_1)$ to avoid dividing by zero

Time Complexity of Graham's Scan

- Sorting – O(n log n)
- During the scan each point is "visited" at most twice
  - Initial visit
  - back up visit (happens at most once)
- Scan - O(n)
- Total time O(n log n)
- This is best possible because sorting is reducible to finding convex hull.

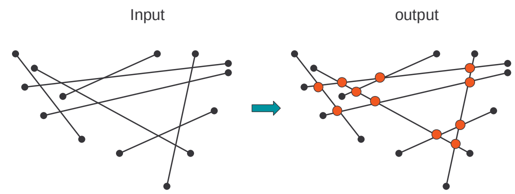
Exercise

- Find an algorithm that, given two sets of points A and B on the plane, determines if there is a line that separates the two sets.

Notes on Convex Hull

- O(n log n)
  – Graham (1972)
- O(n h) algorithm where h is the size of hull
  – Jarvis' March, "Gift wrapping" (1973)
  – Output sensitive algorithm
- O(n log h) algorithm where h is size of hull
  – Kirkpatrick and Seidel (1986)
- d-dimensional Convex Hull
  – $\Omega(n^{d/2})$ in the worst case because the output can be this large.

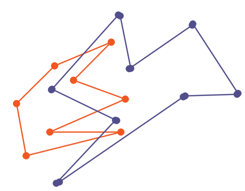
Line Segment Intersection Problem



Special cases

Polygon Intersection

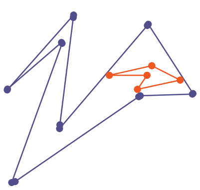
- Polygons have no self intersections



Use line segment intersection to solve polygon intersection

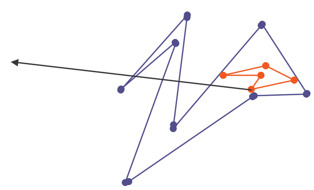
Polygon Intersection

- What if no line segment intersections?



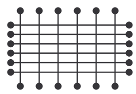
Polygon Intersection

- Intersect a ray from each polygon with the other
  – Inside, if ray has an odd number of intersections, otherwise outside. Jordan curve theorem (1887).



Issues

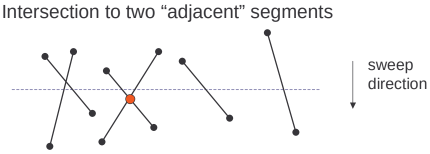
- With n line segments there may be $O(n^2)$ intersections.



- Goal: Good output sensitive algorithm
  - O(n log n + s) would be ideal where s is the number of intersections.

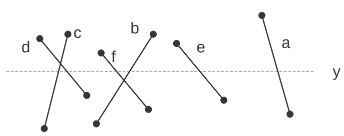
Plane Sweep Algorithm

- Sweep a plane vertically from top to bottom maintaining the set of known future events.
- Events
  - Beginning of a segment
  - End of a segment
  - Intersection to two "adjacent" segments



Segment List

- We maintain ordered list of segments



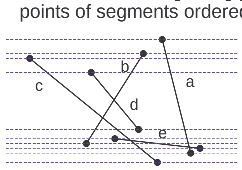
segment ordering at y = c, d, f, b, e, a

Key Idea in the Algorithm

- Just before an intersection event the two line segments must be adjacent in the segment order.
- When a new adjacency occurs between two lines we must check for a possible new intersection event.

Initialization

- Event Queue
  - contains all the beginning points and all the end points of segments ordered by decreasing y value.



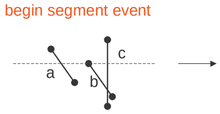
Event Queue
$b_a, b_b, b_c, b_d, e_d, b_e, e_b, e_e, e_a, e_c$

- Segment List
  - Empty

Algorithm

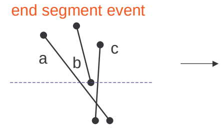
- Remove the next event from the event queue

begin segment event



1. Insert b into the segment list between a and c
2. Check for intersections with adjacent segments (a,b) and (b,c), and add any to event queue

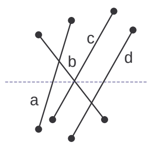
end segment event



1. Delete b from the segment list
2. Check for intersections with adjacent segments (a,c), and add any to event queue

Algorithm

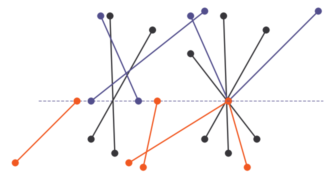
intersection event event



1. Reverse the order of b and c on the segment list
2. Check for intersections with adjacent segments (a,c) and (b,d) and add any to event queue

Complications

- Several events can coincide.

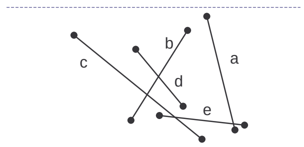


begin segment event
end segment event
intersection event

do in left to right order

- Horizontal lines

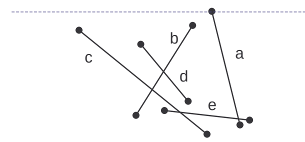
Example



Segment List

Event Queue
$b_a, b_b, b_c, b_d, e_d, b_e, e_b, e_e, e_a, e_c$

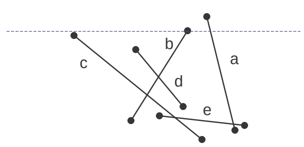
Example



Segment List
a

Event Queue
$b_b, b_c, b_d, e_d, b_e, e_b, e_e, e_a, e_c$

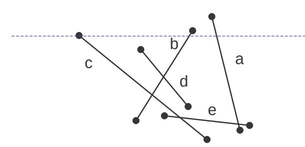
Example



Segment List
b, a

Event Queue
$b_c, b_d, e_d, b_e, e_b, e_e, e_a, e_c$

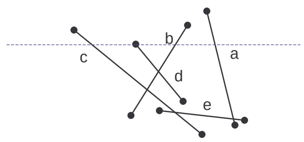
Example



Segment List
c, b, a

Event Queue
$b_d, i_{(c,b)}, e_d, b_e, e_b, e_e, e_a, e_c$

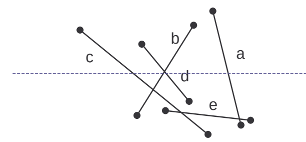
Example



Segment List
c, d, b, a

Event Queue
$i_{(d,b)}$, $i_{(c,b)}$, e_d, b_e, e_b, e_e, e_a, e_c

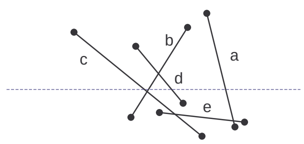
Example



Segment List
c, b, d, a

Event Queue
$i_{(c,b)}$, e_d, b_e, e_b, e_e, e_a, e_c

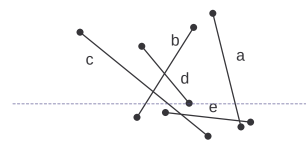
Example



Segment List
b, c, d, a

Event Queue
e_d, b_e, e_b, e_e, e_a, e_c

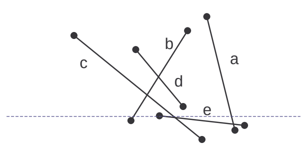
Example



Segment List
b, c, a

Event Queue
b_e, e_b, e_e, e_a, e_c

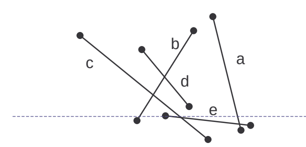
Example



Segment List
b, e, c, a

Event Queue
$i_{(e,c)}$, e_b, e_e, e_a, e_c

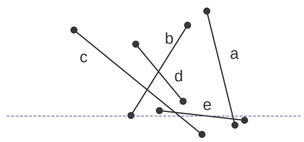
Example



Segment List
b, c, e, a

Event Queue
e_b, $i_{(e,a)}$ e_e, e_a, e_c

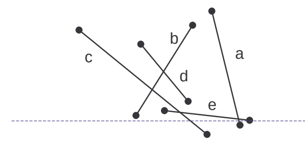
Example



Segment List
c, e, a

Event Queue
$i_{(e,a)}, e_e, e_a, e_c$

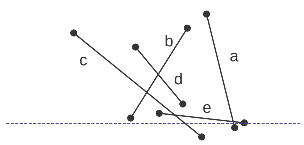
Example



Segment List
c, a, e

Event Queue
e_e, e_a, e_c

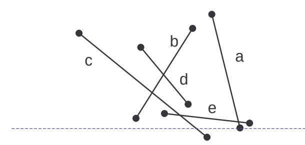
Example



Segment List
c, a

Event Queue
e_a, e_c

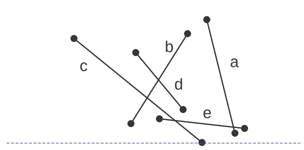
Example



Segment List
c

Event Queue
e_c

Example



Segment List

Event Queue

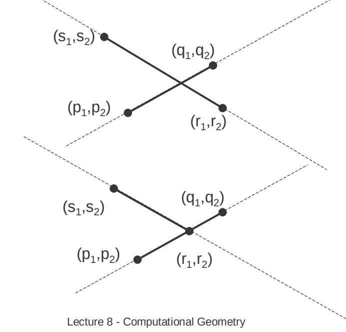
Data Structures

- Event List
  - Priority queue ordered by decreasing y, then by increasing x
  - Delete minimum, Insertion
- Segment List
  - Balanced binary tree search tree
  - Insertion, Deletion
  - Reversal can be done by deletions and insertions
- Time per event is O(log n)

Finding Line Segment Intersections

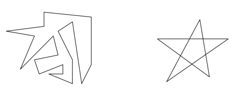
- Given line segments $(p_1,p_2),(q_1,q_2)$ and $(r_1,r_2),(s_1,s_2)$ do they intersect, and if so where.
- Where? Solve
  - $0 = (q_2 - p_2)x + (p_1 - q_1)y + p_2q_1 - p_1q_2$
  - $0 = (s_2 - r_2)x + (r_1 - s_1)y + r_2s_1 - r_1s_2$
- If?
  - (p_1,p_2) and (q_1,q_2) on opposite sides of line $(r_1,r_2),(s_1,s_2)$ and
  - (r_1,r_2) and (s_1,s_2) on opposite sides of line $(p_1,p_2),(q_1,q_2)$

Opposite Sides



Exercise

- A simple polygon is one that does not intersect itself. A polygon is given as a sequence of points (x_1,y_1), (x_2,y_2),… (x_n,y_n),

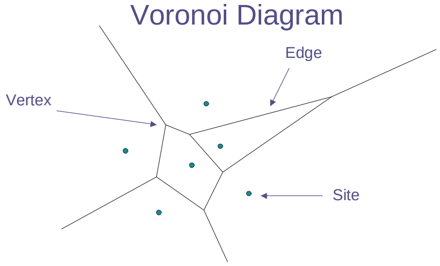


- Design an algorithm for determining if a polygon is simple or not.

Notes on Line Segment Intersection

- Total time for plane sweep algorithm is O(n log n + s log n) where s is the number of intersections.
  - n log n for the initial sorting
  - log n per event
- Plane sweep algorithms were pioneered by Shamos and Hoey (1975).
- Intersection Reporting - Bentley and Ottmann (1979)

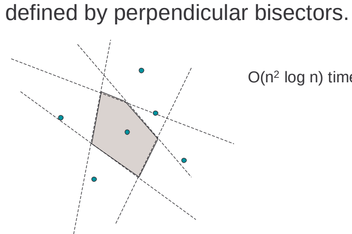
Voronoi Diagram



Each site defines an area of points nearest to it.
Boundaries are perpendicular bisectors.
http://www.cs.cornell.edu/Info/People/chew/Delaunay.

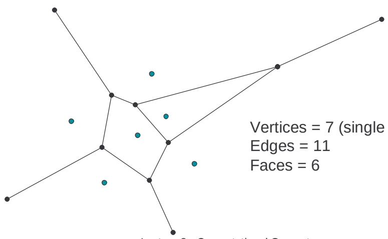
Brute Force

- Each Voronoi area is the intersection of half spaces defined by perpendicular bisectors.



Linear Size of Voronoi Diagram

- The Voronoi Diagram is a planar embedding so it obeys Euler's equation $V - E + F = 2$



Vertices = 7 (single vertex at infinity)
Edges = 11
Faces = 6

Linear Size of Voronoi Diagram

- F = E - V + 2 (Euler's equation)
- n = F (one site per face)
- 2E $\geq$ 3V because each vertex is of degree at least 3 and each edge has 2 vertices.
- n $\geq$ 3V/2 – V + 2 = V/2 + 2
- 2n – 2 $\geq$ V
- n > E – (2n – 2) + 2
- 3n – 4 $\geq$ E

Properties Voronoi Diagram

1. A vertex is the center of a circle through at least three sites

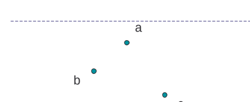
Properties Voronoi Diagram

2. A point on a perpendicular bisector of sites p and q is on an edge if the circle centered at the point through p and q contains no other sites.

Fortune's Sweep

- We maintain a "beach line," a sequence of parabolic segments that is the set of points equidistant from a site and the sweep line.
- Events
  - Site event – new site is encountered by the sweep line
  - Circle event – new vertex is inserted into the Voronoi diagram

Example

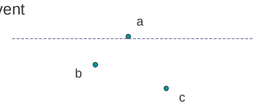


beach line

event queue
a, b, c

Example

site point event

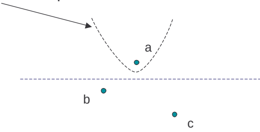


beach line
a

event queue
b, c

Example

points equidistant from point and line

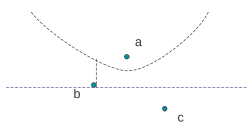


beach line
a

event queue
b, c

Example

site event



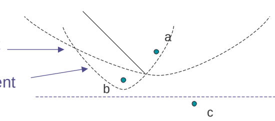
beach line
a, b, a

event queue
c

Example

breakpoint

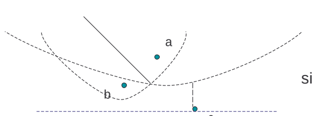
segment



beach line
a, b, a

event queue
c

Example

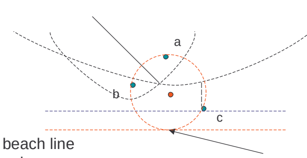


site event

beach line
a, b, a, c, a

event queue
?

Example

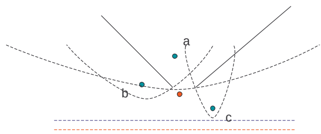


beach line
a, b, a, c, a

circle event must be
added to the event queue

event queue
$c_{(b,a,c)}$

Example

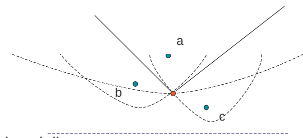


beach line
a, b, a, c, a

event queue
$c_{(b,a,c)}$

Example

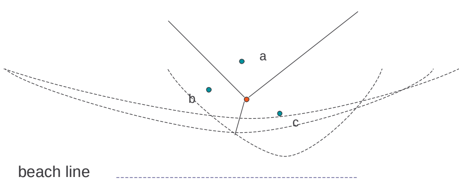
circle event



beach line
a, b, c, a

event queue

Example

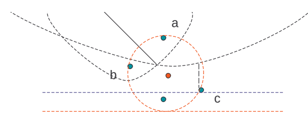


beach line
a, b, c, a

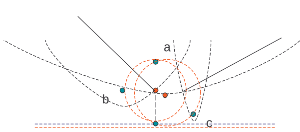
event queue

Event Queue

- Contains site events and circle events sorted by y in decreasing order, then by x in increasing order
- Circle events can be both inserted and deleted.

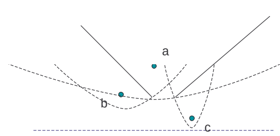


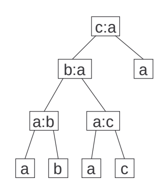
Two New Circle Events



Beach Line

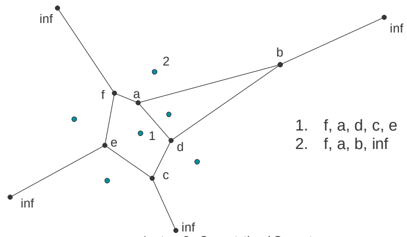
- Implemented as a balanced binary search tree.
  - sites at leaves
  - breakpoints at internal nodes





Output

- For each site output the vertices in clockwise order. When a circle event occurs add to the vertex list of the three (or more) sites.



1. f, a, d, c, e
2. f, a, b, inf

Complexity

- Number of segments in the beach line $\leq 2n$
  - Each site event adds at most 2 segments.
- Number of circle event insertions $\leq 2n$
  - Each site event creates at most 2 circle events.
- Time per event is O(log n)
  - Insert new segments into the segment tree.
  - Insert new circle events into the event queue
  - Delete circle events from the event queue
- Total time is O(n log n)

Voronoi Diagram Notes

- Voronoi diagram
  - Dirichlet (1850), Voronoi (1907)
- O(n log n) algorithm
  - Divide and conquer - Shamos and Hoey (1975)
  - Plane sweep – Fortune (1987)

Exercise

- Give an O(n log n) algorithm which given a set of n points on the plane, for each point finds its nearest neighbor.

Numerics

- Computational geometry algorithms need exact arithmetic over rational numbers or algebraic numbers (solutions to polynomial equations over rationals).
  - In most cases there are predicates P(x,y) that need to be checked.
  - Example of predicates are $x < y$ and $x = y$
- Checking such predicates is very time consuming.
  - There are techniques like interval arithmetic to avoid these exact computations.

More Computational Geometry Problems

- Nearest neighbor search
- Closest pair
- Union of objects
- Silhouette