Proximity Problems

Given a set S of points in d-dimensional space:

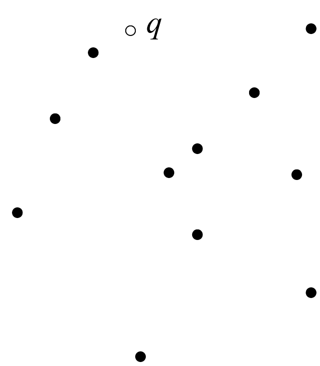
- ***Closest-pair*** (CP): find two points of S that are closest.
- ***Nearest-neighbor*** (NN): find the point in S that is closest to an arbitrary point Q
- ***All nearest-neighbors***: Find NN in S for each point in S
- ***k-Clustering***: Partition S into k clusters to maximize the shortest distance between elements of different clusters
- ***Minimum spanning tree***: construct a tree with vertices S of minimum cost (cost of an edge is the distance between its endpoints)
- ***Triangulation***: construct a maximal set of "fat" triangles whose vertices are the points in S
- ***Largest empty circle***: find the largest circle inside conv(S) containing no points from S
- ***Path planning***: find a path from A to B that stays as far away from S as possible

1

Nearest Neighbor

Given a set P of n points in the plane:

- Store P in a data structure $D(P)$
- Given a query point q, use $D(P)$ to find the point in P that is closest to q



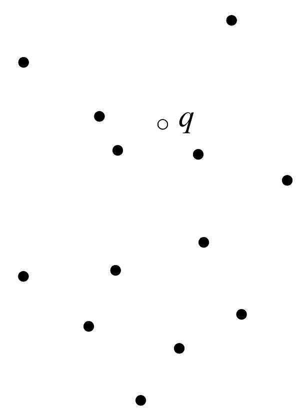
2

k-Nearest Neighbors

Given a set P of n points in the plane:

– Store P in a data structure $D(P)$

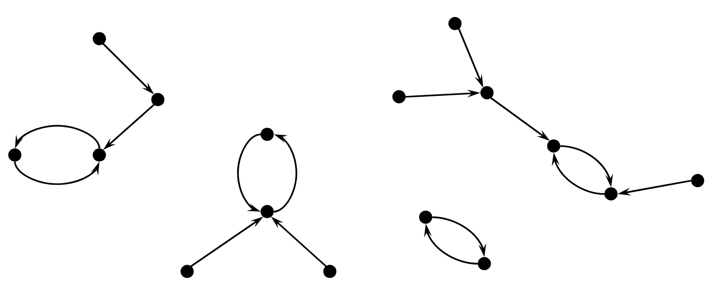
– Given a query point q, use $D(P)$ to report the k points in P that are closest to q



3

All Nearest Neighbors

- Given a set P of n points in the plane, for each point $p \in P$ find all nearest $q \in P$, $q \neq p$
  – This is, in general, a relation, not a function

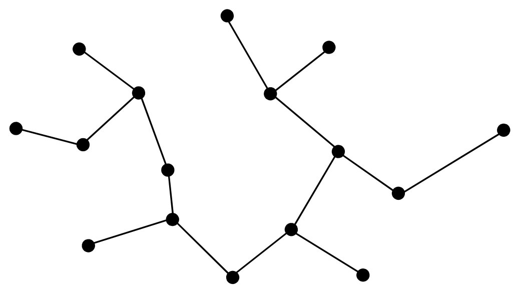


- How about finding, for each point $q \in P$, which points have q as nearest neighbor?

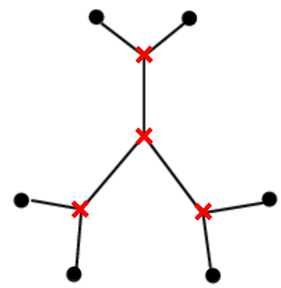
4

Euclidean Minimum Spanning Tree

Given a set P of n points in the plane, construct a tree of minimum total length whose vertices are the points in P



Spanning tree



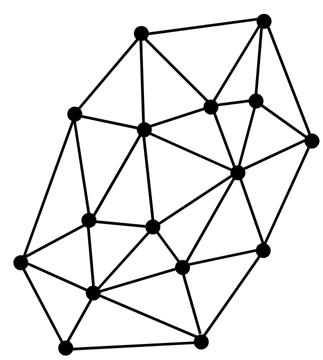
Steiner tree

5

The Triangulation Problem

A ***triangulation*** of a point set $P \subset \mathbb{R}^2$ is a subdivision of the plane using a maximal set of segments joining the endpoints of P

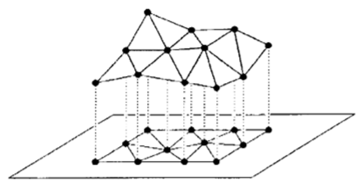
Given a finite set of points $P \subset \mathbb{R}^2$, construct a triangulation that maximizes the smallest angle of the resulting triangles

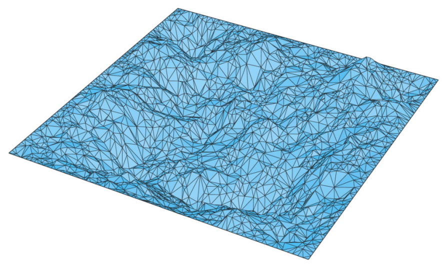


6

An Application: Height Interpolation

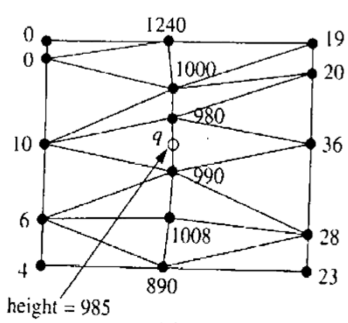
Given a set of points $P \subset R^2$ and a function $f : P \to R$, construct a *polyhedral terrain* for *P*, i.e., a piecewise linear function that approximates the original terrain



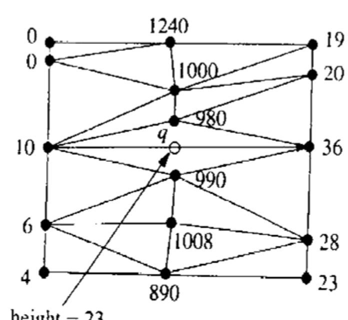


7

Not All Triangulations are Created Equal...



(a)



(b)

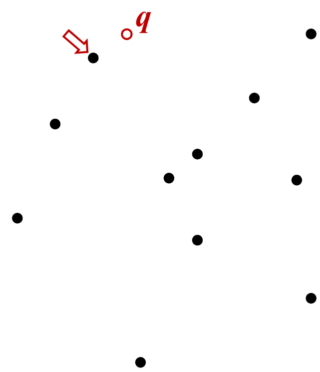
8

Nearest Neighbor

Given a set P of n points in the plane:

– Store P in a data structure $D(P)$

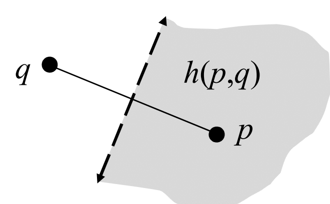
– Given a query point q, use $D(P)$ to find the point in P that is closest to q



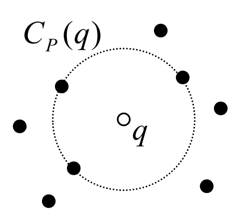
9

Notation

- For sites p and q the perpendicular bisector of segment pq splits the plane into two half-planes. The open halfplane that contains p is denoted by $h(p,q)$



- For a point q the largest *empty* circle centered at q is denoted by $C_P(q)$



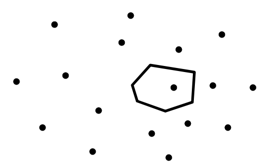
10

Voronoi Diagram

- The Voronoi diagram, Vor(P), is a subdivision of the plane into n cells $V(p_1),\ldots,V(p_n)$, one for each site in P
- The Voronoi cell $V(p_i)$ is the locus of points closer to p_i than to any other site in P:

$$q \in V(p_i) \Leftrightarrow \text{dist}(q, p_i) < \text{dist}(q, p_j), \forall p_j \neq p_i$$

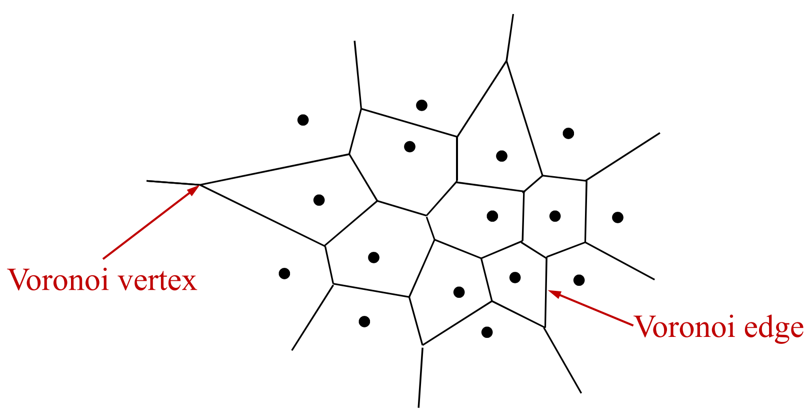
- $V(p_i) = \bigcap_{i \neq j} h(p_i, p_j)$



Note: The Voronoi diagram can be defined for any metric and any dimension, here we concentrate on the planar, Euclidean case

11

Voronoi Diagram: Example

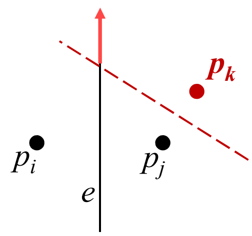


Questions:
- What does it mean for p to lie on a Voronoi edge?
- What does it mean for p to lie on a Voronoi vertex?

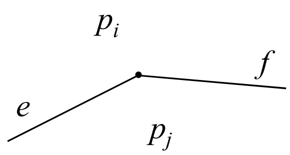
12

Voronoi Diagram: Properties

1. If all sites are collinear then Vor(P) consists of $n - 1$ parallel lines. Otherwise, each edge of Vor(P) is either a line segment or a half-line.



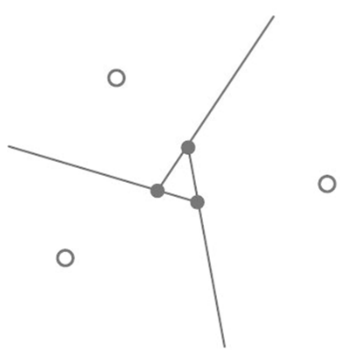
2. A vertex of Vor(P) is the intersection of at least three bisectors



13

Exercise

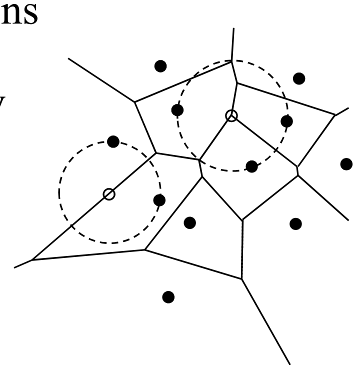
- Explain why this cannot happen



14

Voronoi Properties...

3. A point q is a vertex of Vor(P) iff $C_P(q)$ contains three or more points from P on its boundary

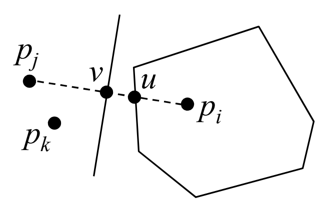


4. The bisector of p_i and p_j defines an edge of Vor(P) iff there is a point q such that $C_P(q)$ contains exactly p_i and p_j on its boundary

15

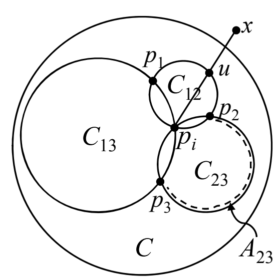
Voronoi Properties…

5. <u>Every</u> nearest neighbor of p_i defines an edge of the cell $V(p_i)$



$$d(p_i p_k) < 2d(up_i) < 2d(vp_i) = d(p_i p_j)$$

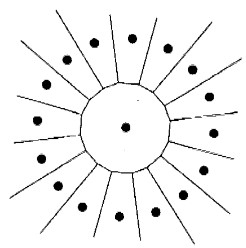
6. Cell $V(p_i)$ is unbounded iff p_i is part of boundary of Conv(P)

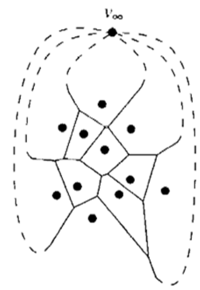


16

Voronoi Diagram: Complexity

Theorem. The number of vertices in the Voronoi diagram of n points in the plane is at most $2n - 5$ and the number of edges is at most $3n - 6$.

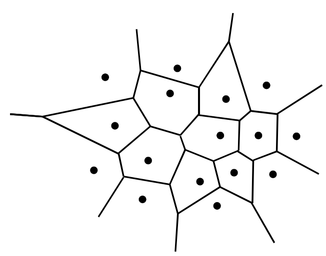




Corollary. The average size of a Voronoi cell is $O(1)$.

17

Computing the Voronoi Diagram



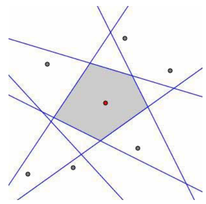
1. Brute force: $O(n^2 \log n)$
2. Plane sweep: $O(n \log n)$
3. Incremental:
   – Naïve: $O(n^2)$
   – Randomized incremental on the dual graph of Vor(P): $O(n \log n)$
4. Divide and Conquer: $O(n \log n)$
5. 3D lift-up transformation: $O(n \log n)$

- Should we look for a faster algorithm?

18

A Brute Force Algorithm

- For each $i \in \{1, \dots, n\}$ compute $V(p_i) = \bigcap_{i \neq j} h(p_i, p_j)$

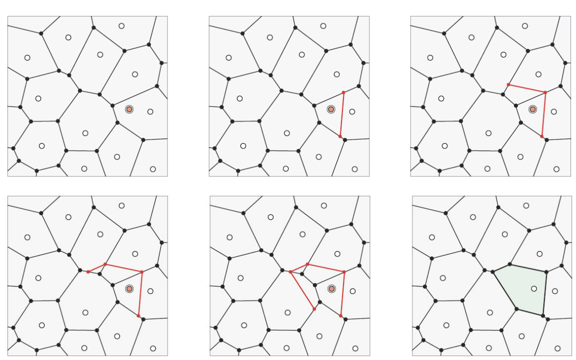


- Total time is $O(n^2 \log n)$
  - *Problem*: while the average size of a cell is $O(1)$, the intersection of $n-1$ halfplanes is computed

19

Naïve Incremental

- Given $\mathrm{Vor}(P_{i-1})$ compute $\mathrm{Vor}(P_i)$:
  1. Find the region $V(q)$ that contains the new site
  2. Draw the perpendicular bisector for qp_i in
  3. Repeat (2) in neighbor cells until closing the loop



20

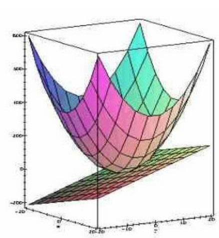
Exercise

- What is the bottleneck in the naïve incremental algorithm?
- Describe how to reduce the complexity of the algorithm and justify the new running time

21

Lift-up Transformation

- Can construct $\mathrm{Vor}(P)$ in R^2 from a polyhedron in R^3
- Each input point (a, b) is mapped to the plane tangent to the paraboloid U: $z = x^2 + y^2$ at point $(a, b, a^2 + b^2)$:
  $h(a, b) \rightarrow z = 2ax + 2by - (a^2 + b^2)$



- For each plane we are interested in the positive half-space $h^+(a, b)$ consisting of all points above $h(a, b)$
- $\mathrm{Vor}(P)$ is now the projection of $\bigcap_{(a,b)\in P} h^+(a,b)$ onto the xy-plane

22

Voronoi Diagrams in 1D

- Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of "points" in 1D. What is $\text{Vor}(A)$?
- Lifting a to the parabola $\mathcal{P}: y = x^2$ yields the point (a, a^2). What is the equation of the line tangent to $\mathcal{P}$ at (a, a^2)?
- What is the relation between the perpendicular bisector of a_i and a_j and the intersection of the corresponding tangent lines?
- Each tangent line $\ell(a)$ induces an upper halfplane $h^+(a)$. What is $\bigcap_{a \in A} h^+(a)$?

23

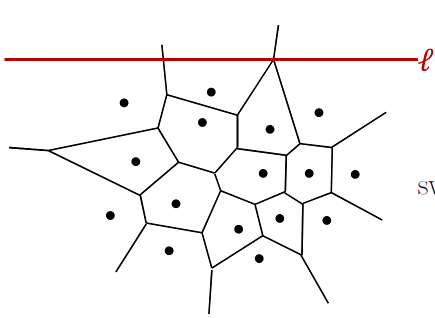
Exercise

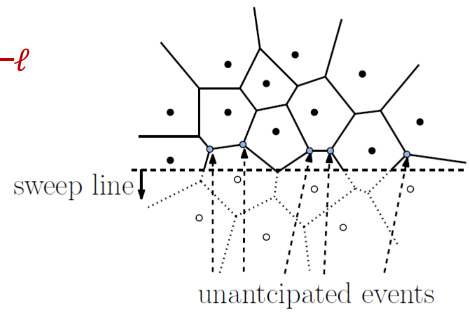
1. Prove that the plane tangent to the paraboloid $z = x^2 + y^2$ at $(a, b, a^2 + b^2)$ has equation $z = 2ax + 2by - (a^2 + b^2)$
2. Find a vector normal to the perpendicular bisector of points (a_1, b_1) and (a_2, b_2)
3. What is the equation of the perpendicular bisector of points (a_1, b_1) and (a_2, b_2)?
4. Consider the intersection of two halfplanes $h^+(a_i, b_i)$ and $h^+(a_j, b_j)$. What is the projection of the intersection onto the xy-plane?

24

A Sweepline Algorithm

- What is the problem with the standard approach of sweeping with a line?
  - $\mathrm{Vor}(P)$ above ℓ depends on sites of P below ℓ
  - When the top vertex of $V(p_i)$ is reached, the sweep line has not yet seen p_i





25

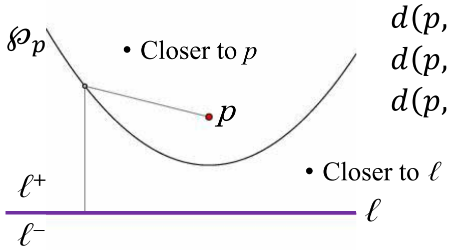
A Modified Sweep

- Maintain the part of $\mathrm{Vor}(P)$ for sites above ℓ that cannot change due to sites below ℓ
- For which points above the sweep line ℓ do we know with certainty their nearest site in P?
  - The distance from a point q above ℓ to a site below ℓ is greater than the distance from q to ℓ itself
  - The nearest site to q cannot lie below ℓ if *some* site above ℓ is as close to q as ℓ is

26

Bisector of a point and a line

- What is the locus of points closer to a point p than to a line ℓ?



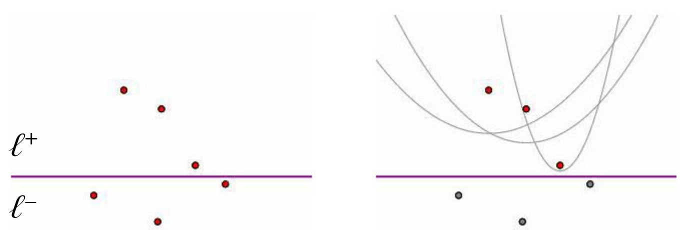
$d(p,q) = q_y - \ell_y$ if q lies on $\wp_p$
$d(p,q) < q_y - \ell_y$ if q lies above $\wp_p$
$d(p,q) > q_y - \ell_y$ if q lies below $\wp_p$

- *Key insight.* $\wp_p$ partitions the points in ℓ^+ into those closer to p and those closer to ℓ

27

Modified Sweep…

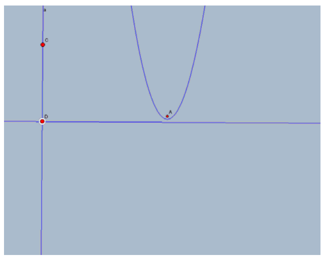
- Keep track of the locus of points closer to some $p_i \in \ell^+$ than to ℓ
  - The distance from a point $q \in \ell^+$ to a site $p_j \in \ell^-$ is $\geq$ dist(q,ℓ)

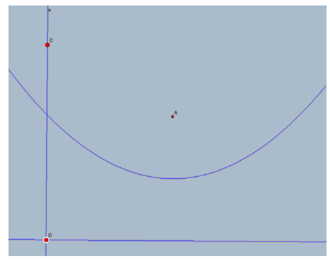


28

Exercise

- For an arbitrary site p what happens to the parabola $\wp_p$ as ℓ sweeps down?



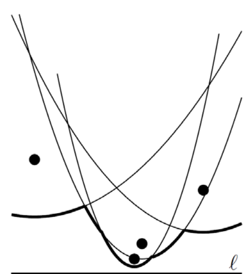


- What is the nature of $\wp_p$ when $p \in \ell$?

29

The Beach Line

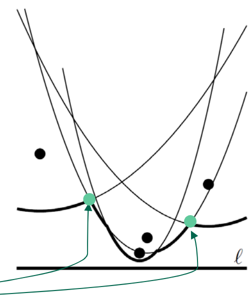
- The locus of points equidistant to their nearest site in ℓ^+ <u>and</u> to the sweep line is called the ***beach line***
- The beach line β consists of a monotone sequence of parabolic arcs that correspond to the lower envelope of the union of all parabolas
- A point above β, is closer to some site in ℓ^+ than to every point in ℓ^-



30

The Beach Line…

- the Voronoi diagram above β is determined by the sites above ℓ
- Sites that induce parabolas above β do not contribute a parabolic arc to β
- Some parabolas may contribute several pieces to β
- Two consecutive arcs define a ***breakpoint***

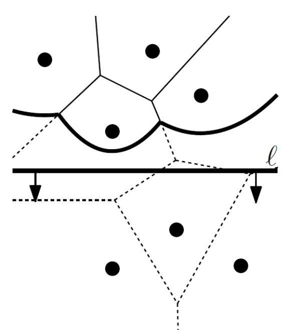


31

Breakpoints

- A breakpoint is equidistant from two sites and from the sweep line
- If the beach line arcs for sites p_i and p_j share a common breakpoint on the beach line, then this breakpoint lies on the Voronoi edge between p_i and p_j

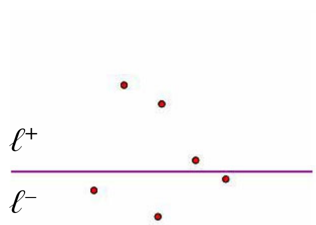
  ⇒ The edges of Vor(P) are traced by the breakpoints of β as ℓ moves down

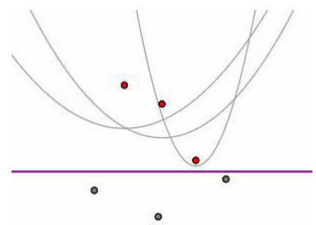


32

Fortune's Approach

- Instead of maintaining the intersection of $\mathrm{Vor}(P)$ with ℓ, maintain β as ℓ sweeps down, as this is the part of $\mathrm{Vor}(P)$ that cannot change due to sites below ℓ
- Points above β are closer to some $p_i \in \ell^+$ than to ℓ and, consequently, cannot belong to the cell of any site $p_j \in \ell^-$





33

Updating β

- How does β change as ℓ sweeps down?
  - Since β changes continuously, Fortune's algorithm does not maintain β explicitly. Instead, it tracks *topological changes* to β
- Two types of changes (events)
  - Insertion of a new parabolic arc (a ***site event***)
  - Removal of an arc as it shrinks to a point and disappears (a ***circle event***)
- Between consecutive events the sequence of sites contributing arcs to β remains the same

34

Computing Nearest Neighbors

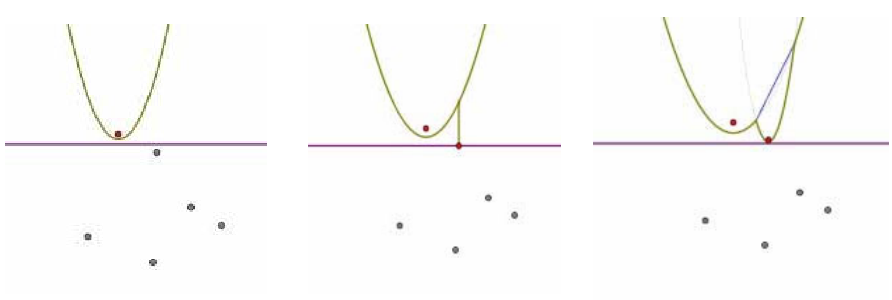
- Fix an arbitrary point q in $\mathbb{R}^2$. When q first appears on β on a parabolic arc $\wp_{p_i}$
  - q is outside every parabola $\wp_{p_j}$, $j \neq i$
  - $d(q, p_j) \geq d(q, p_i) = q_y - \ell_y, j \neq i$
  - If q coincides with a breakpoint, then it is equidistant to two sites

***Lemma**. When a point first appears on the beach line, it is on a parabolic arc associated to its nearest site. The breakpoints lie on the edges of the Voronoi diagram.*

35

Detecting Voronoi Edges…

- Breakpoints are created when a new arc is added to the beach line, i.e., when the sweepline reaches a new site.



36

Detecting Voronoi Vertices

- Breakpoints move outwards along a Voronoi edge until they reach a vertex
- This happens when a parabolic arc α shrinks to a point
- α and its two neighbors correspond to three sites whose cells meet at the vertex
- This happens when the sweep line is tangent to the circumcircle of the three sites

37

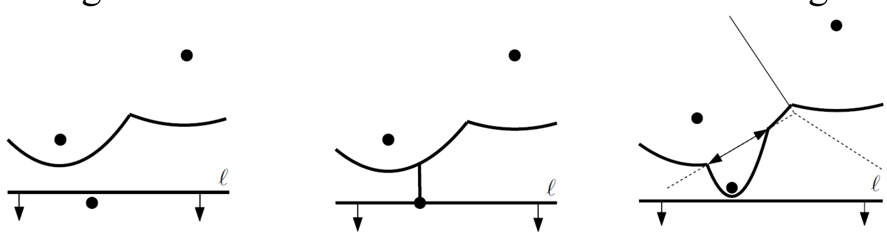
Sweepline Events

- While the beach line β changes continuously, its combinatorial structure changes discretely at two types of events
  1. At a ***site event*** a new parabolic arc appears and a new edge starts to grow
  2. At a ***circle event*** an existing arc α disappears as its two neighbors meet and "consume" α
     - Corresponds to two growing edges meeting at a Voronoi vertex

38

Site Events

- A site event occurs when the sweep line meets a new site, creating a new arc and two breakpoints
- As ℓ moves down, the two breakpoints move in opposite directions, tracing the same Voronoi edge
- Edge is "disconnected" until it meets another edge

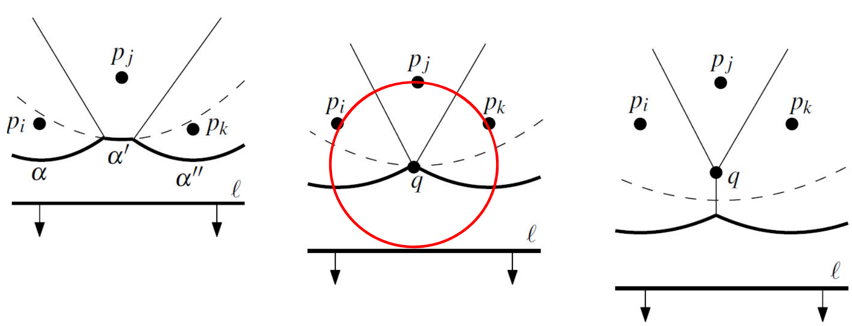


Lemma. The beach line consists of no more than $2n - 1$ arcs.

39

Circle Events

- An arc α' of β shrinks to a point and disappears
- Arc α' and neighbors α and α'' correspond to sites that are co-circular when α' disappears
- $C_P(q)$ is tangent on ℓ



40

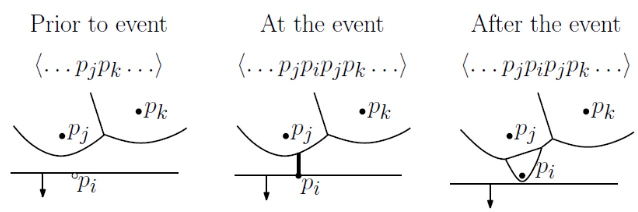
Data Structures

1. The *schedule* is stored as a priority queue that contains all *site events* and known *circle events*
   – Events are stored by y-coordinate
2. The algorithm maintains the current location (y-coordinate) of the sweep line
3. The *status* is a binary search tree $\mathfrak{I}$ that stores at the leaves, in left to right order, the sites that define β. Internal nodes correspond to break points (i.e., edges of $\mathrm{Vor}(P)$ being traced)
   – *Note*: parabolic arcs are not stored explicitly
4. A doubly connected edge list stores $\mathrm{Vor}(P)$

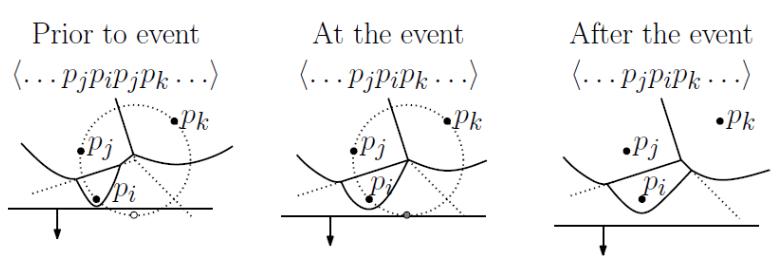
41

Event Generation and Handling

- Site events are generated up front



- Circle events are generated on the fly



42

Fortune's Algorithm

Algorithm VORONOIDIAGRAM(P)
Input. A set $P := \{p_1, \dots, p_n\}$ of point sites in the plane.
Output. The Voronoi diagram $\mathrm{Vor}(P)$ given inside a bounding box in a doubly-connected edge list $\mathcal{D}$.

1. Initialize the event queue $\mathcal{Q}$ with all site events, initialize an empty status structure $\mathcal{T}$ and an empty doubly-connected edge list $\mathcal{D}$.
2. **while** $\mathcal{Q}$ is not empty
3.   **do** Remove the event with largest y-coordinate from $\mathcal{Q}$.
4.   **if** the event is a site event, occurring at site p_i
5.   **then** HANDLESITEEVENT(p_i)
6.   **else** HANDLECIRCLEEVENT(γ), where γ is the leaf of $\mathcal{T}$ representing the arc that will disappear
7. The internal nodes still present in $\mathcal{T}$ correspond to the half-infinite edges of the Voronoi diagram. Compute a bounding box that contains all vertices of the Voronoi diagram in its interior, and attach the half-infinite edges to the bounding box by updating the doubly-connected edge list appropriately.

43

Handling Site Events

HANDLESITEEVENT(p_i)

1. If $\mathcal{T}$ is empty, insert p_i into it (so that $\mathcal{T}$ consists of a single leaf storing p_i) and return. Otherwise, continue with steps 2– 5.
2. Search in $\mathcal{T}$ for the arc α vertically above p_i. If the leaf representing α has a pointer to a circle event in $\mathcal{Q}$, then this circle event is a false alarm and it must be deleted from $\mathcal{Q}$.
3. Replace the leaf of $\mathcal{T}$ that represents α with a subtree having three leaves. The middle leaf stores the new site p_i and the other two leaves store the site p_j that was originally stored with α. Store the tuples $\langle p_j, p_i \rangle$ and $\langle p_i, p_j \rangle$ representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary.
4. Create new half-edge records in the Voronoi diagram structure for the edge separating $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$, which will be traced out by the two new breakpoints.
5. Check the triple of consecutive arcs where the new arc for p_i is the left arc to see if the breakpoints converge. If so, insert the circle event into $\mathcal{Q}$ and add pointers between the node in $\mathcal{T}$ and the node in $\mathcal{Q}$. Do the same for the triple where the new arc is the right arc.

44

Handling Circle Events

HANDLECIRCLEEVENT(γ)

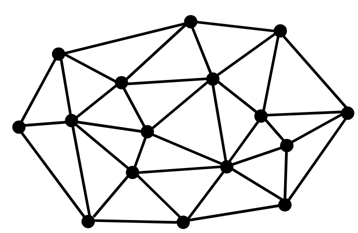
1. Delete the leaf γ that represents the disappearing arc α from $\mathcal{T}$. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary. Delete all circle events involving α from $\mathcal{Q}$; these can be found using the pointers from the predecessor and the successor of γ in $\mathcal{T}$. (The circle event where α is the middle arc is currently being handled, and has already been deleted from $\mathcal{Q}$.)
2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list $\mathcal{D}$ storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoint of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end at the vertex.
3. Check the new triple of consecutive arcs that has the former left neighbor of α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into $\mathcal{Q}$. and set pointers between the new circle event in $\mathcal{Q}$ and the corresponding leaf of $\mathcal{T}$. Do the same for the triple where the former right neighbor is the middle arc.

Theorem. Fortune's algorithm runs in $O(n \log n)$ time and uses $O(n)$ space

45

Triangulations of Point Sets

- A *triangulation T* of *P* is a *maximal straight line planar subdivision* whose vertex set is *P*

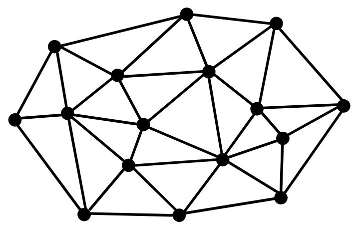


- Basic properties:
  - Every edge of the unbounded face belongs to the boundary of convex hull of *P*
  - Each bounded face is a triangle

46

Exercise

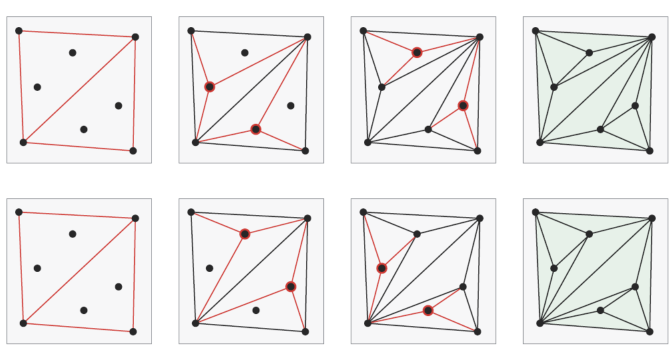
- Let P denote a set of points in the plane. Show that the edges of $\text{conv}(P)$ must appear in *any* triangulation of P



47

Triangle Splitting Algorithm

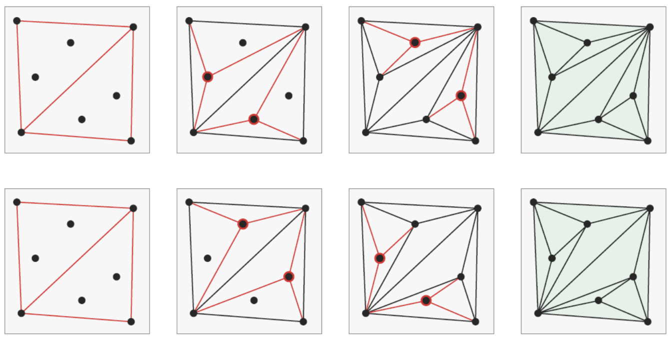
1. Find conv(P) and triangulate it as a polygon
2. For each interior point q do
   a) Find the triangle t that contains q
   b) Add edges from q to the three vertices of t



48

Triangle Splitting…

1. Make sure you know how to implement the algorithm using a DCEL
2. How many triangles and edges do you get?



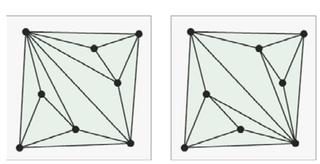
$t = 10$
$e = 17$

49

Exercise

- Let P be a set of n points in the plane with h extreme vertices. Consider a triangulation with t triangles and e edges produced by the triangle splitting algorithm
- Express t as a function of n and h
- Express e as a function of n and h
- Can different insertion orders produce different values of t and e?

Example
$n = 8,\ h = 4$
$t = 10,$
$e = 17$

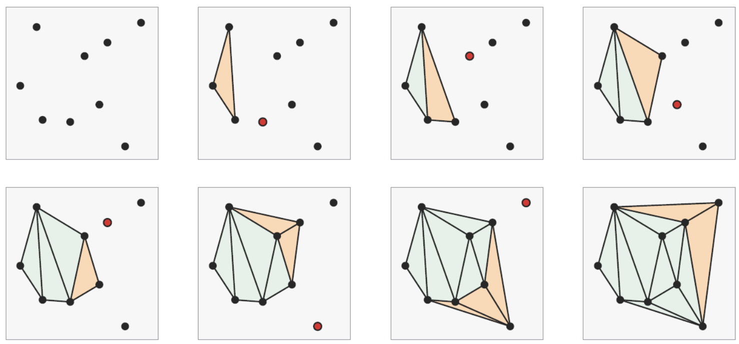


50

A Simpler Incremental Algorithm

1. Sort the points of S by x-coordinate. The first three points $\langle p_1, p_2, p_3 \rangle$ determine a triangle T_3
2. **for** $i \leftarrow 4$ **to** n **do** // compute T_i from T_{i-1}

   Connect p_i with all points $\{p_{i_1}, \ldots, p_{i_k}\}$ of current triangulation T_{i-1} which are visible to p



51

Triangulation Complexity

Theorem. Let P be a set of n points in the plane, not all collinear, and let h denote the number of points in P that lie on the boundary of conv(P). Then, *any* triangulation for P consists of

$t = 2n - h - 2$ triangles, and

$e = 3n - h - 3$ edges

Proof. If t is the number of triangles then $f = t + 1$ and $e = (3t + h)/2$.

Using Euler's formula $n - e + f = n - (3t + h)/2 + (t + 1) = 2$ and the claim follows.

52

How Many Triangulations are there?

- Let $t(P)$ denote the number of triangulations of a point set P and $t(n) = \max_{|P|=n} t(P)$

Theorem. $C_{n-2} \le t(n) \le 30^n$ where

$$C_k = \frac{1}{k+1}\binom{2k}{k} = \frac{(2k)!}{(k+1)!k!} = \prod_{i=2}^{k} \frac{k+i}{i}, k \ge 1$$

The first few values of C_k for $k = 1,2,3,\ldots$:

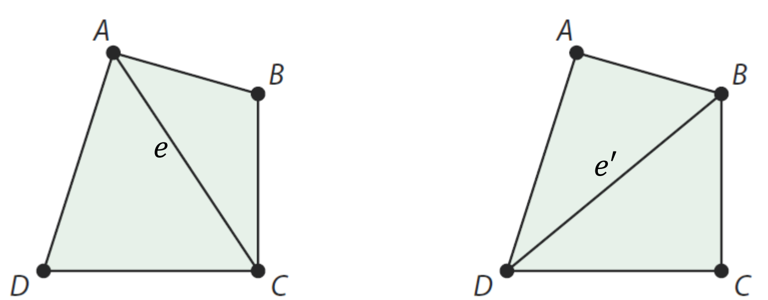
1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, …

Open problem. Design a polynomial time algorithm to compute $t(P)$ for a set of points P

53

Edge Flips

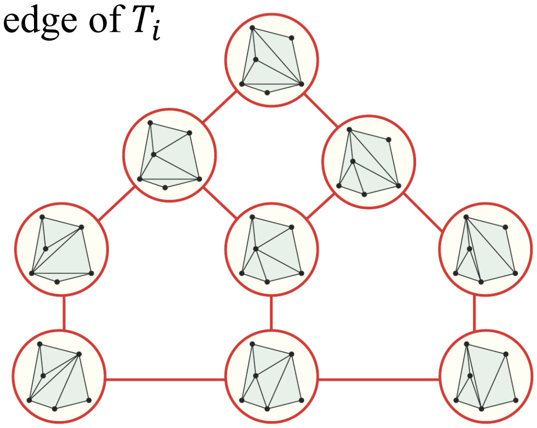
- Let $e = AC$ denote an edge of T and $Q = ABCD$ the quadrilateral consisting of the two triangles incident on e. If Q is convex then AC can be replaced by BD to produce a different triangulation
- This is called a ***flip*** of e



54

The Flip Graph

- The ***flip graph*** of P is a graph G whose nodes are the triangulations of P. Nodes T_i and T_j are connected by an edge if T_j can be produced by flipping an edge of T_i



55

Exercise

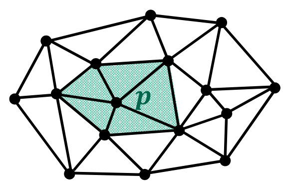
- For every $n > 3$, construct a point set of size n whose flip graph consists of a single node
- For every $n > 3$, construct a point set of size n whose flip graph consists of two nodes connected by an edge
- Can you construct a set with two nodes and no edges?

56

Flip Graph Properties

1. The flip graph of P is connected
2. Any triangulation can be turned into the incremental one using $\leq \binom{n-2}{2}$ flips
3. If P has n points, then the diameter of its flip graph is at most $(n-2)(n-3)$

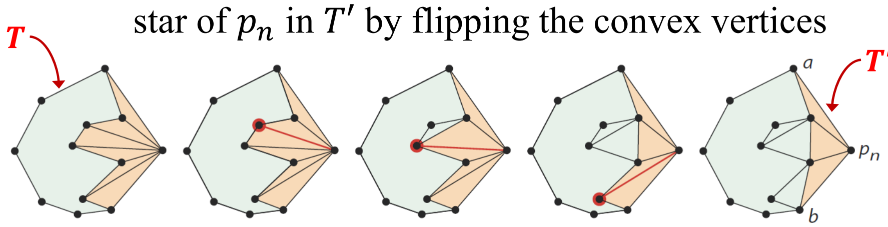
Note. The ***diameter*** is the longest shortest path between two nodes. The ***star*** of a point p in a triangulation of P is the set of triangles incident with p.



57

Proof Sketch

1. Any triangulation T can be converted into the triangulation T' produced by the x-incremental algorithm by using edge flips
    a. Assume this can be done for $|S| < n$ points
    b. Take the star of p_n in T and change it to the star of p_n in T' by flipping the convex vertices

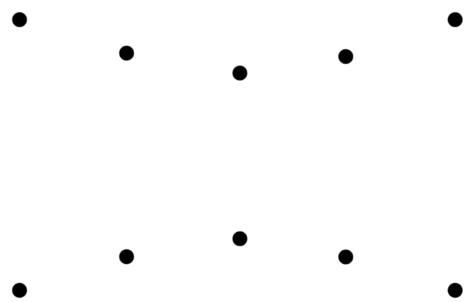


2. By induction on n, number of flips to get T' is at most $\binom{n-2}{2} \Rightarrow$ diameter is $\leq (n-2)(n-1)$

58

Exercise

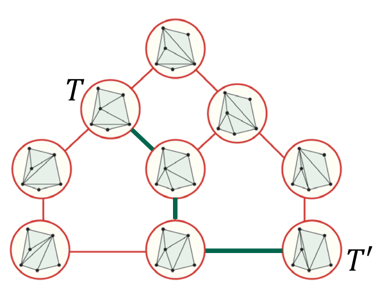
- Find the diameter of the flip graph for the point set below



59

Open Problem

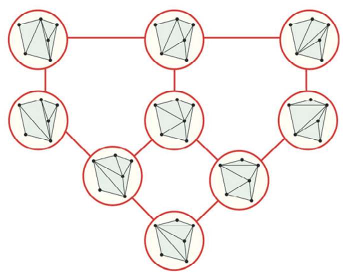
- Let P be a set of n points in the plane with flip graph $\mathcal{G}$. Design a polynomial time algorithm that finds a shortest path between two arbitrary nodes T and T' of $\mathcal{G}$, i.e., a smallest number of flips that transforms T into T'



60

Exercise

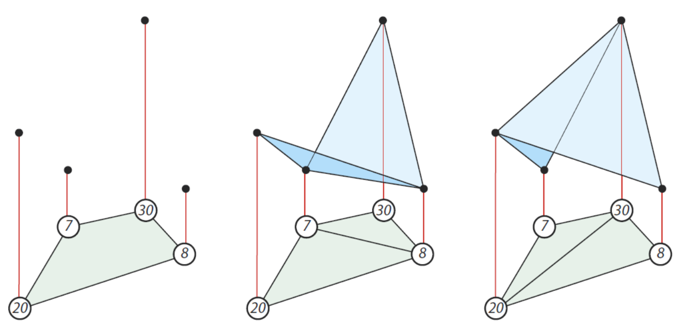
- Let G be the flip graph of a set of points in the plane
- Is it possible to have C_3 (a cycle of length 3) as a subgraph of G?



61

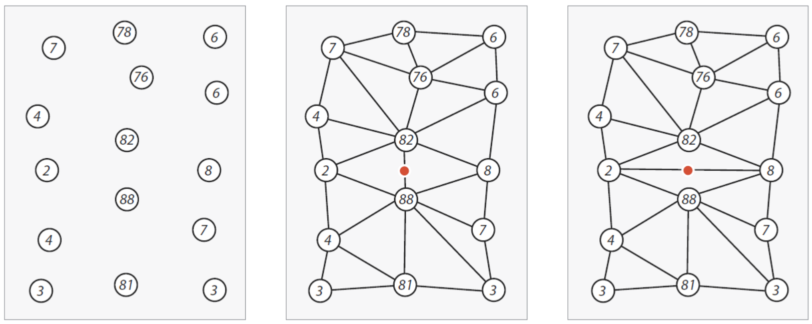
Choosing a Triangulation

- The choice of triangulation has a big impact on the appearance of a terrain
- "True terrain" is only known at sample points



62

Which Triangulation is Better?

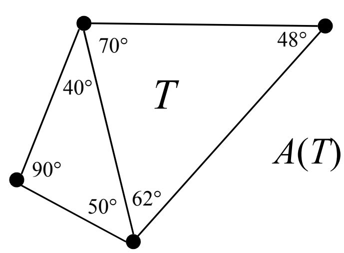


- Prefer big over small angles
- Try to maximize the smallest angle

63

Angle Vectors

- The ***angle vector*** of a triangulation *T* is the *sorted* list of internal angles of the *t* triangles of *T*: $A(T) = \langle \alpha_1, \alpha_2, \ldots, \alpha_{3t} \rangle, \alpha_i \leq \alpha_{i+1}$



$A(T) = \langle 40°,48°,50°,62°,70°,90° \rangle$

64

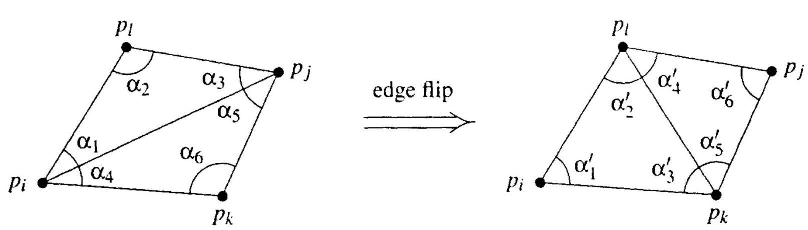
Can Triangulations be Ordered?

- Yes, lexicographically! Define $A(T) < A(T')$ iff there is $1 \leq i \leq 3t$ such that $\alpha_j = \alpha'_j$ for $j < i$ and $\alpha_i < \alpha'_i$
- T' is *fatter* than T if $A(T') > A(T)$
- Other relations (≤, >, ≥, =) defined similarly
- Triangulation T is ***angle-optimal*** if it is fattest, i.e., $A(T) \geq A(T')$ for *all* triangulations T' of P
- Angle-optimal triangulations are desirable for polyhedral terrain approximation

65

Edge Flips

- Recall that an edge flip produces a new triangulation
- When is this triangulation fatter?



66

Illegal Edges

- An edge e of T is ***illegal*** if we can locally increase the smallest angle by flipping e, i.e.,

$$\min_{1\le i\le 6}\alpha_i < \min_{1\le i\le 6}\alpha_i'$$

- Flipping an illegal edge of T results in a triangulation T' with $A(T) < A(T')$
- A triangulation is ***legal*** if it does not contain any illegal edges

67

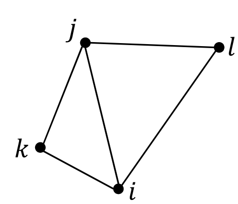
Constructing a Legal Triangulation

Algorithm LegalTriangulation(T)

Input: Some triangulation T of a point set P

Output: A legal triangulation of P

1. **while** T contains an illegal edge p_ip_j **do**
2. let $p_ip_jp_k$ and $p_ip_jp_l$ be the two triangles incident to edge p_ip_j
3. remove p_ip_j from T and add p_kp_l
4. **return** T



68

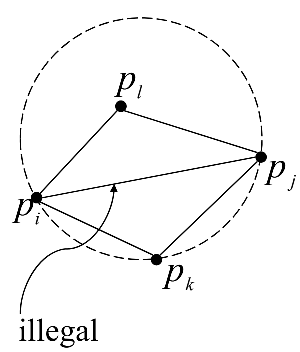
Some Issues

- Is the algorithm guaranteed to terminate?
  - If so, what is its running time?
- How do you determine in practice if an edge is legal?
- Is a legal triangulation angle-optimal?

69

Determining if an Edge is Legal

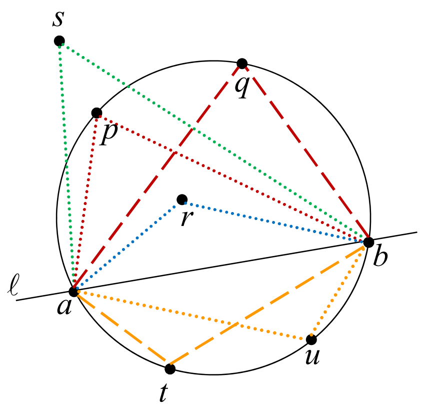
Theorem. let $p_i p_j p_k$ and $p_i p_j p_\ell$ be the two triangles adjacent to edge $p_i p_j$. Edge $p_i p_j$ is illegal **iff** the point p_ℓ lies in the interior of the circle through $p_i p_j p_k$. Also, if the points p_i, p_j, p_k, p_ℓ form a convex quadrilateral and do not lie on a common circle then exactly one of $p_i p_j$ and $p_k p_\ell$ p is illegal



70

Thales Theorem

- Attributed to Thales of Miletus (c.624–c.546 BCE)

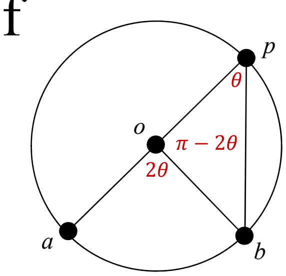


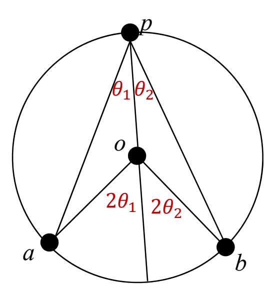
- $\angle apb = \angle aqb$
- $\angle atb = \angle aub$
- $\angle asb < \angle apb < \angle arb$
- $\angle atb = \pi - \angle apb$

71

Proof

- Let $\angle apb$ and $\angle aob$ have the same arc base ab. We have two cases:
  1. Special case in which one leg of $\angle apb$ is a diameter.
  2. In the general case we draw a diameter from p which splits $\angle apb$ into two instances of case 1.





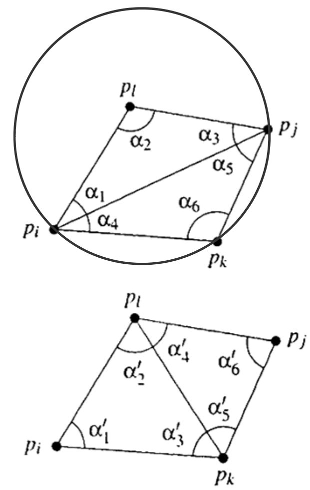
*Exercise***.** What is $\angle apb$ when ab is a diameter?

*Exercise***.** What is $\angle apb$ when $\angle apb$ and $\angle aob$ have opposite arc bases?

72

Proof of Legality Test

- Assume p_l is ***inside*** circle through p_i, p_j, p_k.
- For every angle of T' there is a smaller angle in T:
  - $\alpha_4 < \alpha_4'$
  - $\alpha_4 < \alpha_1'$
  - $\alpha_5 < \alpha_2'$
  - $\alpha_5 < \alpha_6'$
- Similarly, get $\alpha_3 < \alpha_3'$ and $\alpha_1 < \alpha_5'$ by using circle through p_i, p_j, p_l

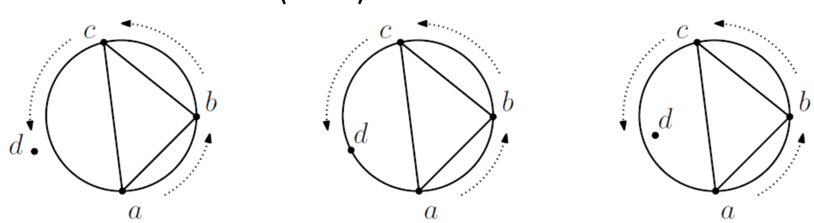


73

A Practical Legality Test

$$\text{inCircle}(a,b,c,d) = \det \begin{pmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{pmatrix} = \boldsymbol{D}$$

Precondition: $\langle abc \rangle$ must be counterclockwise



$\boldsymbol{D} < 0$ $\quad$ $\boldsymbol{D} = 0$ $\quad$ $\boldsymbol{D} > 0$

Exercise. Prove the correctness of the InCircle test

74

Delaunay Graphs

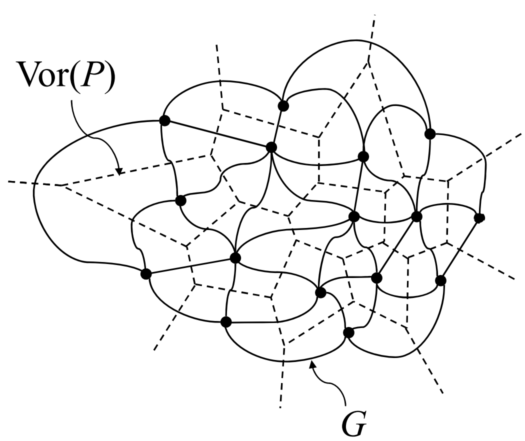
Consider the dual graph G of Vor(P):

- Each face of Vor(P) corresponds to a node of G
- (p_i, p_j) is an arc (i.e., edge) of G iff $V(p_i)$ and $V(p_j)$ share an edge of Vor(P)
- Each vertex of Vor(P) corresponds to a bounded face of G

The *Delaunay graph* of P, denoted Del(P), is the embedding of G that uses the sites of P for nodes and straight line segments for arcs

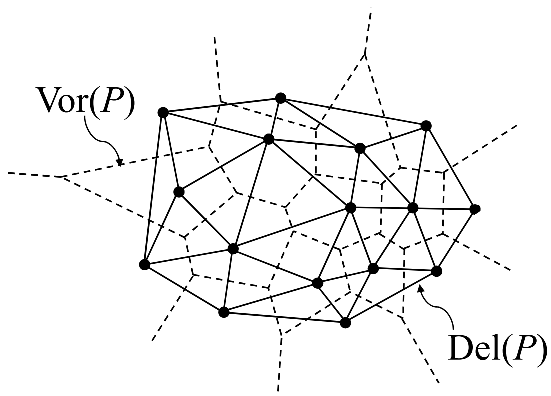
75

Dual Graph: Example



76

Delaunay Graph: Example



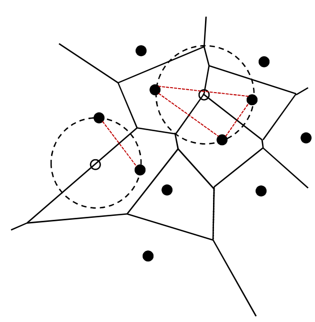
• A *Delaunay triangulation* is any triangulation obtained by adding non-crossing edges to the Delaunay graph

77

Delaunay Graph: Properties

1. Three points q, r, $s \in P$ are vertices of the same face of Del(P) iff the circle through p, q, r contains no point of P in its interior

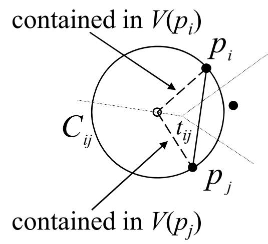
2. Two points q, $r \in P$ form an edge of Del(P) iff there is a closed disc that contains q and r on its boundary and contains no other point of P



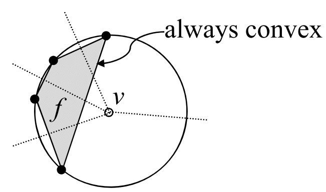
78

Delaunay Properties…

3. Del(P) is a planar graph



4. If the points of P are in general position (no four are co-circular) then Del(P) is a triangulation of P

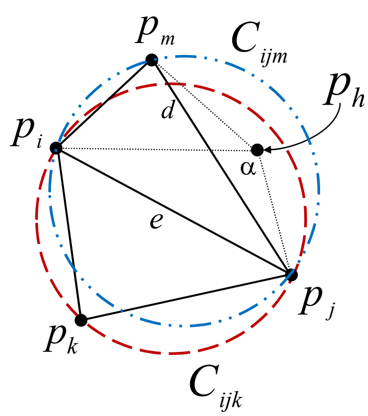


Delaunay Triangulation

- A *Delaunay triangulation* of P is *any* triangulation obtained by adding edges to the Delaunay graph of P
- The Delaunay graph of P is unique. However, if P is not in general position the Delaunay triangulation of P may not be unique
- T is a Delaunay triangulation of P iff the circumcircle of every triangle of T contains no points of P in its ***strict interior***

Delaunay Properties...

5. A triangulation of P is legal iff it is a Delaunay triangulation
6. An angle-optimal triangulation of P is a Delaunay triangulation
7. A Delaunay triangulation of P maximizes the minimum angle over all triangulations of P



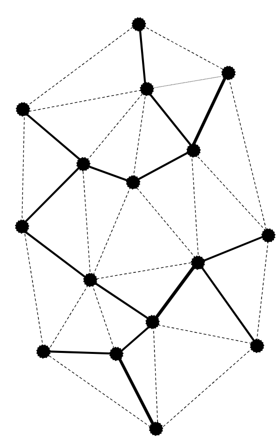
$\angle mhj > \alpha = \angle ihj$

81

Delaunay Properties…

8. Every nearest neighbor q of p defines an edge (p, q) of a Delaunay triangulation of P

9. A minimum spanning tree (*MST*) of a Delaunay triangulation of P is a *Euclidean minimum spanning tree* (*EMST*) of P



82

What does this program do?

```
int mistery(set p of n points){
  int i, j, k, m, flag, xn, yn, zn, numTrian=0;
    for ( i = 0; i < n ; i++ ) p[i].z =p[i].x*p[i].x + p[i].y*p[i].y;
    for ( i = 0; i < n − 2; i++ )
    for ( j = i + 1; j < n; j++ )
    for ( k = i + 1; k < n; k++ )
      if ( j != k ) {
        xn = (p[j].y−p[i].y)*(p[k].z−p[i].z) − (p[k].y−p[i].y)*(p[j].z−p[i].z);
        yn = (p[k].x−p[i].x)*(p[j].z−p[i].z) −  (p[j].x−p[i].x)*(p[k].z−p[i].z);
        zn = (p[j].x−p[i].x)*(p[k].y−p[i].y) −  (p[k].x−p[i].x)*(p[j].y−p[i].y);
         if ( flag = (zn < 0) )
                 for (m = 0; m < n; m++)
                     flag = flag &&
                         (((p[m].x-p[i].x)*xn +
                          (p[m].y-p[i].y)*yn +
                          (p[m].z-p[i].z)*zn) <= 0);
             if (flag) {
                add triangle (i,j,k) to output
                numTrian++;
            }
        }
        return numTrian;
}
```

83

Delaunay Properties...

10. The Delaunay triangulation of a set of *n* points is the projection onto the *x*-*y* plane of the lower hull of a set of *n* points in 3D :
    - Each input 2D point (a, b) is projected to the 3D point $(a, b, a^2 + b^2)$
    - Compute the 3D lower hull of the *n* projected points
    - The projection of the lower hull onto the *x*-*y* plane is the Delaunay triangulation of *P*.

84

A Simple Implementation

```
int DelaunayByProjection(set p of n points){
  int i, j, k, m, flag, xn, yn, zn, numTrian=0; List triangles;
  for ( i = 0; i < p.n ; i++ ) p[i].z =p[i].x*p[i].x + p[i].y*p[i].y;
  for ( i = 0; i < n − 2; i++ )
  for ( j = i + 1; j < n; j++ )
  for ( k = i + 1; k < n; k++ )
     if ( j != k ) {
        xn = (p[j].y−p[i].y)*(p[k].z−p[i].z) − (p[k].y−p[i].y)*(p[j].z−p[i].z);
        yn = (p[k].x−p[i].x)*(p[j].z−p[i].z) −  (p[j].x−p[i].x)*(p[k].z−p[i].z);
        zn = (p[j].x−p[i].x)*(p[k].y−p[i].y) −  (p[k].x−p[i].x)*(p[j].y−p[i].y);
        if ( flag = (zn < 0) )
                  for (m = 0; m < n; m++)
                       flag = flag &&
                              (((p[m].x-p[i].x)*xn +
                               (p[m].y-p[i].y)*yn +
                               (p[m].z-p[i].z)*zn) <= 0);
             if (flag) {
                add triangle (ij,k) to triangles
                numTrian++;
             }
        }
        return [numTrian, triangles];
}
```

85

Relation to Other Proximity Problems

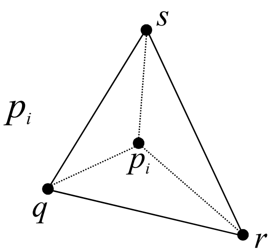
Theorem. Let T be a Delaunay triangulation for a set P of n points on the plane:

- The *convex hull* of P can be computed from T in $O(n)$ time
- The *Voronoi diagram* of P can be computed from T in $O(n)$ time
- *All nearest neighbors* of P can be computed from T in $O(n)$ time
- A *Euclidean minimum spanning tree* for P can be computed from T in $O(n \log n)$ time

86

Computing a Delaunay Triangulation Efficiently

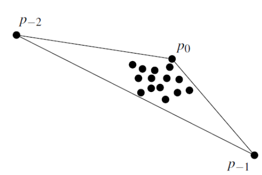
- A randomized incremental algorithm
- Start with a triangulation Π that contains P
- To insert a point p_i:
  - – locate triangle that contains p_i
  - – triangulate locally
  - – reestablish legality
- Which edges are illegal after the local re-triangulation?

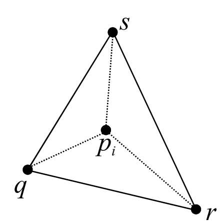


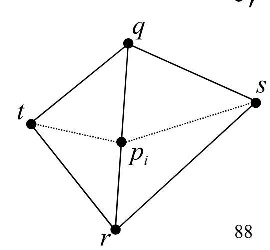
87

Algorithm DelaunayTriangulation(P)

1. Initialize T with enclosing triangle $p_0 p_{-1} p_{-2}$
2. Compute a random permutation of P
3. **for** $i \leftarrow 1$ **to** $n-1$ **do**
4.   find a triangle qrs of T that contains p_i
5.   **if** p_i lies in the interior of qrs **then**
6.   add edges from p_i to vertices q, r, s
7.   LegalizeEdge(p_i, qr, T)
8.   LegalizeEdge(p_i, rs, T)
9.   LegalizeEdge(p_i, sq, T)
10.   **else** (p_i lies on an edge qr of qrs and qrt)
11.   add edges from p_i to s and t
12.   LegalizeEdge(p_i, qt, T)
13.   LegalizeEdge(p_i, tr, T)
14.   LegalizeEdge(p_i, rs, T)
15.   LegalizeEdge(p_i, sq, T)
16. discard p_{-1}, p_{-2} and all incident edges
17. **return** T







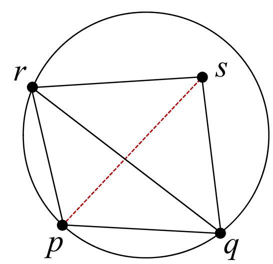
88

Reestablishing Legality

Algorithm LegalizeEdge(p, qr, T)

Purpose: check qr (shared by pqr and sqr) for legality and flip if necessary. New point is p

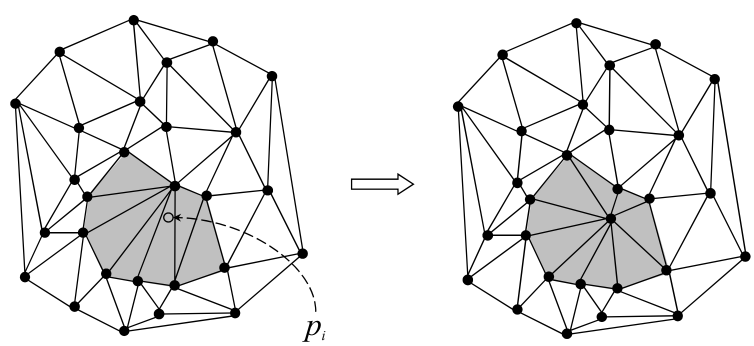
1. **if** qr is illegal **then**
2. replace qr with ps
3. LegalizeEdge(p, qs, T)
4. LegalizeEdge(p, rs, T)



89

Properties

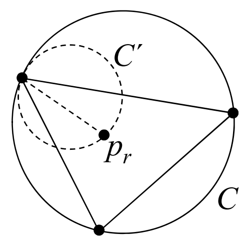
- Every new edge created due to the insertion of p_i is incident to p_i

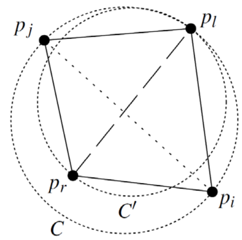


90

Properties...

- Every edge created in DelaunayTriangulation or LegalizeEdge during the insertion of p_r is an edge of the Delaunay graph of $\{p_{-2}, p_{-1}, p_0, \dots, p_r\}$





91

Properties...

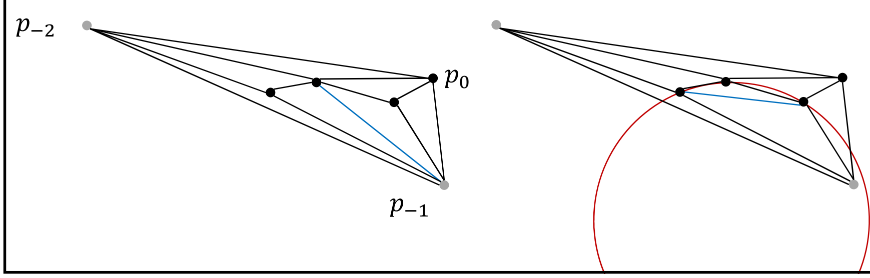
- A previously legal edge can only become illegal if one of its incident triangles changes
- Every edge flip increases the angle-vector of $T \Rightarrow$ LegalizeEdge always terminates

Summary: The proposed algorithm correctly computes a Delaunay triangulation of P.

92

Implementation

- How do we find efficiently the triangle of T containing the new point p_i?
- How do we compute the initial triangle $p_0\, p_{-1} p_{-2}$ that encloses all the points in P?
- How do we deal correctly with the vertices p_{-1} and p_{-2} when testing for legal edges?



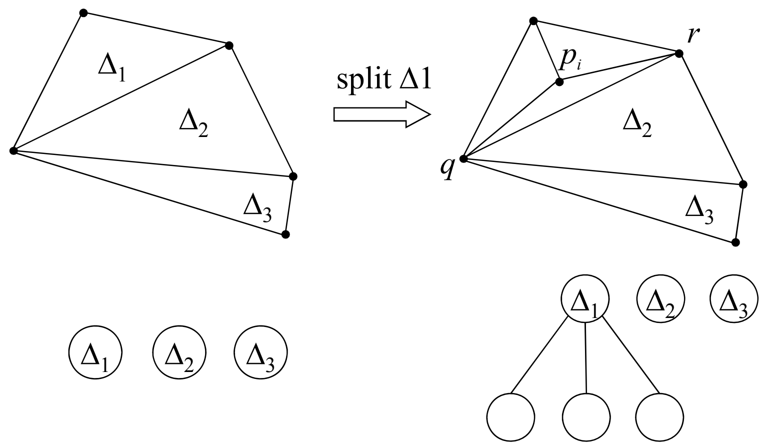
93

Triangular Point Location

- Incrementally build T and search structure D
- Properties of D:
  - directed acyclic graph
  - leaves of D correspond to current triangles of T (keep cross-pointers to go back and forth)
  - internal nodes correspond to deleted triangles
  - path visits all triangles (old and new) that contain p_i
  - D and T are both initialized to triangle $p_0 p_{-1} p_{-2}$

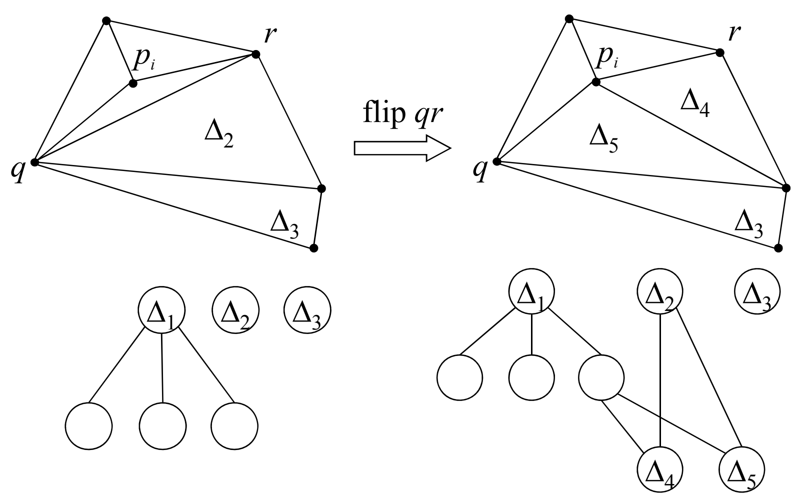
94

Updating the Search Structure



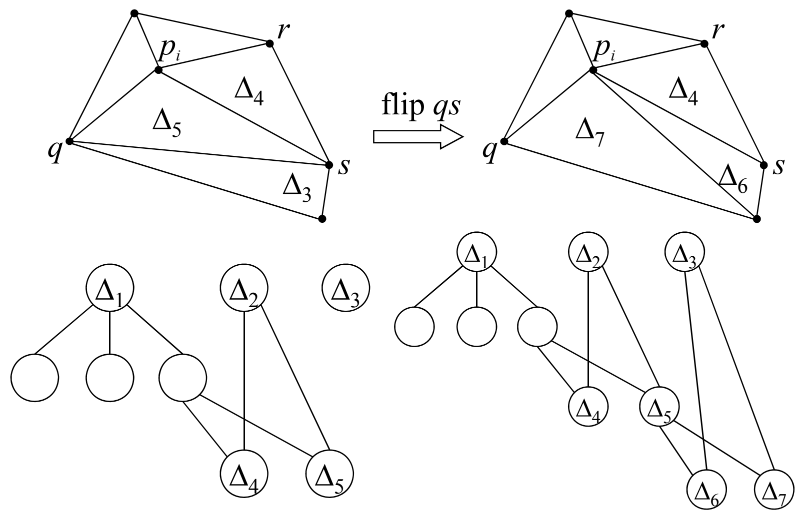
95

Update by Flipping



96

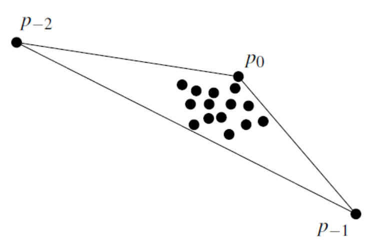
Update by Flipping...



97

An Enclosing Triangle

- Start with a large enough triangle that encloses P
  - p_h, $h < 0$, is outside circ(p_i,p_j,p_k)
- Two sentinel vertices:
  - p_{-1} is below and to the right of P
  - p_{-2} is above and to the left of P
  - p_0 is the highest vertex of P
- When flipping favor edges with only input vertices over edges with sentinel vertices



Goal: DT of $P \cup \{p_{-1}, p_{-2}\}$ consists of DT of P plus edges joining p_{-1} to right hull of P and edges joining p_{-2} to left hull of P

98

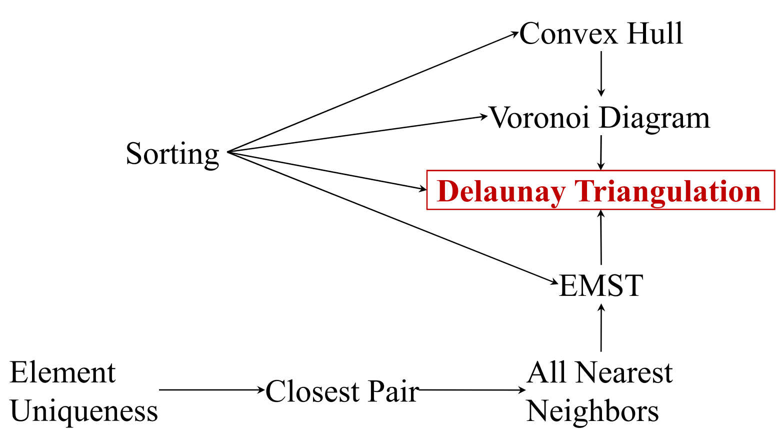
Complexity

Lemma. The expected number of triangles created by algorithm DelaunayTriangulation is at most $9n+1$.

Theorem. A Delaunay triangulation of a set P of n points in the plane can be computed in $O(n \log n)$ expected time using $O(n)$ expected storage.

99

Summary of Proximity Problems



100