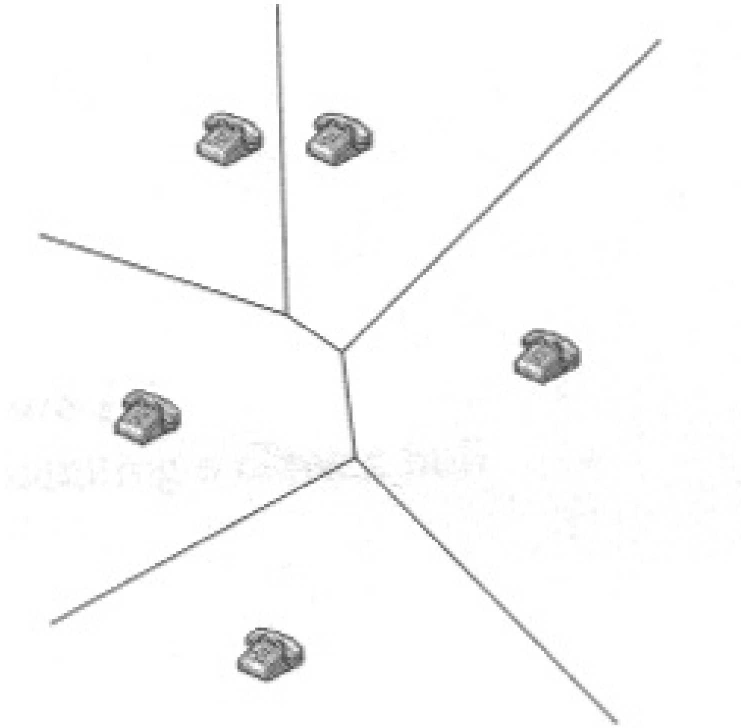
Computational Geometry

7. Voronoi-Diagrams and Delaunay-Triangulation

7.1 Voronoi-Diagrams

- **Input:** Set of phone boxes or post offices and current query position.
- **Goal:** Find closest phone box or post office
- Distance problems of this kind (nearest-neighbor-search) can be solved with Voronoi-Diagrams.

Georgi Feodosjewitsch Woronoi, 1868-1908



7.1 Voronoi-Diagrams

Definition 1
Let $P = \{p_1, \ldots, p_n\}$ be a set of n distinct points (data sites) and d a metric.

A tessellation of the plane (or space) in n *Voronoi-cells* $V(p_i)$ with

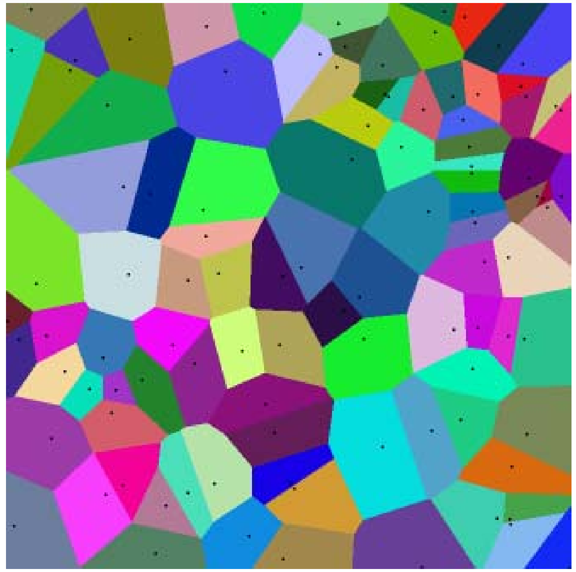
$$q \in V(p_i) \Leftrightarrow d(q, p_i) < d\left(q, p_j\right) \forall j \neq i.$$

is called *Voronoi-Diagram* $\mathrm{Vor}(P)$ of P.

- In the sequel we will use the Euclidian metric in the plane,

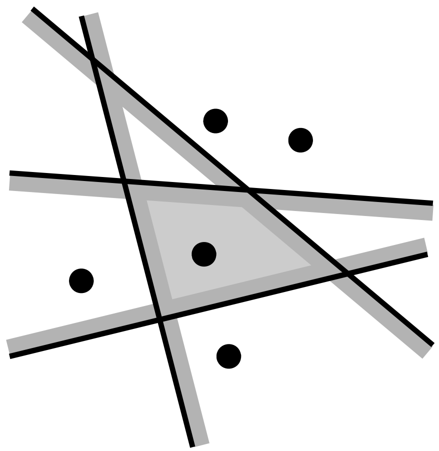
$$d(p, q) = \sqrt{(p_x - q_x)^2 + \left(p_y - q_y\right)^2}.$$

7.1 Voronoi-Diagrams



- Because the bisector m_{ij} of p_i and p_j is a line, the Voronoi-cells for the Euclidian metric are intersections of open half-spaces respectively half-planes $h(p_i, p_j)$.
- For
  $h(p_i, p_j) = \{q | d(q, p_i) < d(q, p_j)\}$
  we get

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



7.1 Voronoi-Diagrams

Proposition 1 (Global shape and topology of a Voronoi-diagram)

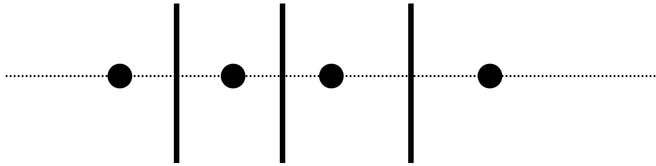
Let P be a set of $n \geq 3$ distinct points in the plane.

1. If all points are collinear, $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines.
2. Otherwise, $\mathrm{Vor}(P)$ is a
   a) connected graph whose
   b) edges are line segments or half-lines.

Proof

- Because the Voronoi-cells are intersections of half-planes, they are convex.

7.1 Voronoi-Diagrams



1. The first case is easy.
2. Assume p_i, p_j, p_k are not collinear.

   b) Because all edges in $\mathrm{Vor}(P)$ are pieces of bisectors, they can only be line segments, half-lines or full lines.

   - Because the bisectors m_{ij} and m_{ik} intersect, both cannot belong completely to $\mathrm{Vor}(P)$, i.e. full lines are not possible.

   a) Assume $\mathrm{Vor}(P)$ is not connected.

   - Then there must be a cell $V(p_i)$, separating the two components.
   - This cell is bounded by two parallel lines due to convexity.
   - These lines are full lines, contradicting b). $\square$

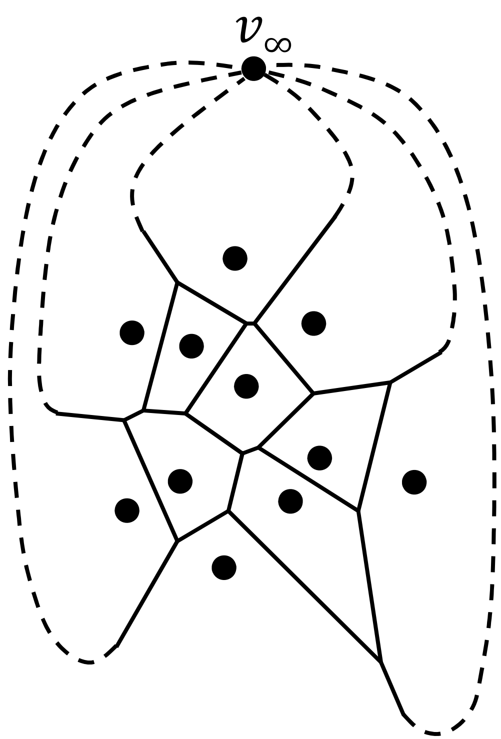
7.1 Voronoi-Diagrams

Proposition 2 (Global combinatorics of a Voronoi-diagram)

A planar Voronoi-diagram of $n \geq 3$ points has at most $n_v = 2n - 5$ Voronoi-vertices and $n_e = 3n - 6$ Voronoi-edges.

Proof

- Not all points are collinear, otherwise we had $n - 1$ edges.
- Connect all half-lines to an auxiliary vertex "at infinity" v_∞.



7.1 Voronoi-Diagrams

- Use Euler's formula $n_v - n_e + n_f = 2$, which relates the number of vertices n_v, edges n_e and faces n_f of a connected, planar embedded graph:

$$2 = (n_v + 1) - n_e + n.$$

- Because every edge has two endpoints and every Voronoi-vertex has at least three edges, we get

$$3(n_v + 1) \leq 2n_e\,,$$

$$6 = 3(n_v + 1) - 3n_e + 3n \leq -n_e + 3n \qquad \Leftrightarrow n_e \leq 3n - 6$$

$$4 = 2(n_v + 1) - 2n_e + 2n \leq -n_v - 1 + 2n \Leftrightarrow n_v \leq 2n - 5.$$

$\square$

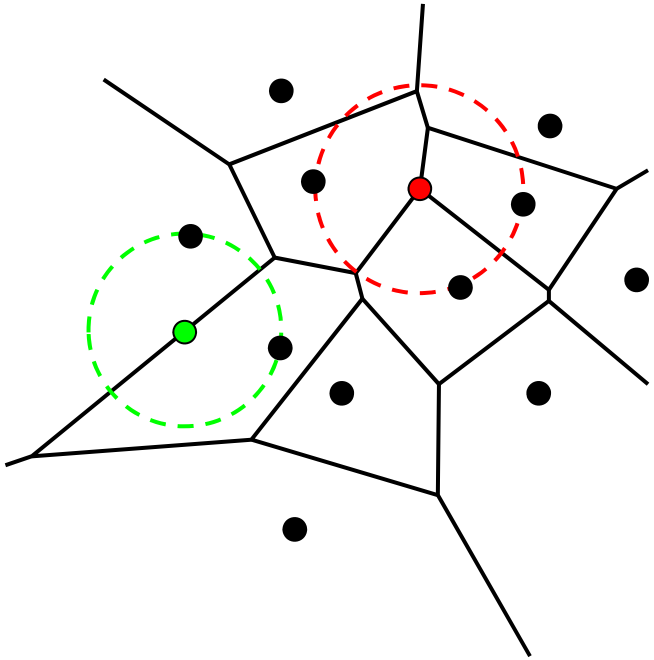
7.1 Voronoi-Diagrams

Proposition 3 (Characterization of Voronoi-vertices and -edges)

1. A point q in the plane is a Voronoi-vertex of $\mathrm{Vor}(P)$, if and only if the largest empty circle $c_P(q)$ at q contains at least three points of P on its boundary.

No point of P lies in the interior of c_P.

2. A bisector m_{ij} defines an edge of $\mathrm{Vor}(P)$, if and only if there is a point $q \in m_{ij}$, whose largest empty circle $c_P(q)$ at q contains only p_i and p_j on its boundary.



7.1 Voronoi-Diagrams

Proof

1. $\Leftarrow$ Let q be a point with empty circle $c_P(q)$ through p_i, p_j, p_k.

   - Thus, at least three cells $V(p_i), V(p_j), V(p_k)$ meet in q, i.e. q is a Voronoi-vertex.

   $\Rightarrow$ If q is a Voronoi-vertex, then there are at least three points p_i, p_j, p_k on the empty circle around q.

2. $\Leftarrow$ Let q be a point with empty circle $c_P(q)$ through two points p_i, p_j.

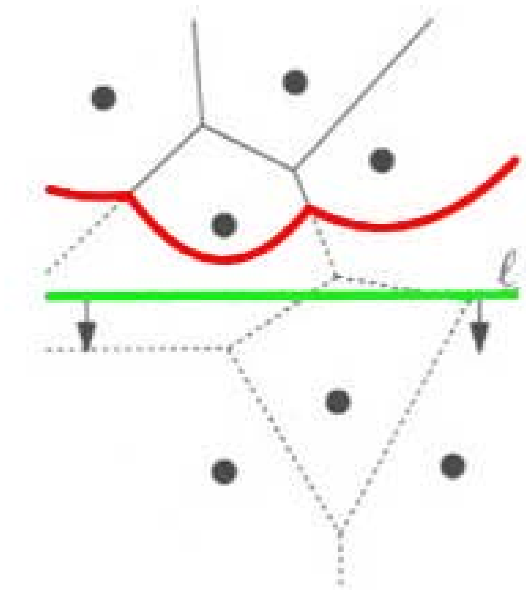
   - Then q belongs to the edge between $V(p_i), V(p_j)$ and from 1) it is not a Voronoi-vertex.

   $\Rightarrow$ If m_{ij} contains an edge of $\mathrm{Vor}(P)$, for every inner point q its largest empty circle $c_P(q)$ contains only p_i and p_j.

$\square$

7.2 Voronoi-Algorithm

- The classic algorithm to compute a Voronoi-Diagram is a difficult to implement divide-and-conquer-method with complexity $O(n \log n)$, see [2].
- A simpler sweep-line-algorithm with run time $O(n \log n)$ is due to S. Fortune [3]:
  - A sweep-line ℓ scans from top to bottom and completes the Voronoi-diagram above.
  - **But:** Points below the sweep-line can change the diagram above in an area below a piecewise quadratic curve $\beta(x)$. This curve is the so-called *beach*.



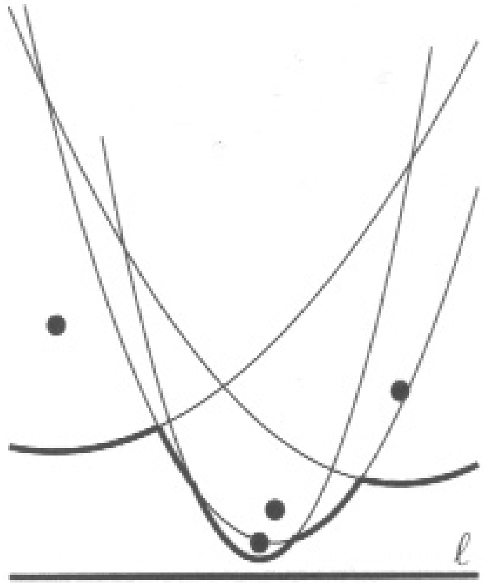
7.2 Voronoi-Algorithm

- The beach consists of all points that have the same distance to the sweep-line ℓ and to the nearest point p above the sweep-line.
- These parabola segments (*arcs*) satisfy

$$\beta(x) - \ell_y = \left\| p - \left(x, \beta(x)\right)^t \right\|$$

- Squaring and solving for β yields

$$\beta(x) = \frac{x^2 - 2p_x x + p_x^2 + p_y^2 - \ell_y^2}{2(p_y - \ell_y)}.$$

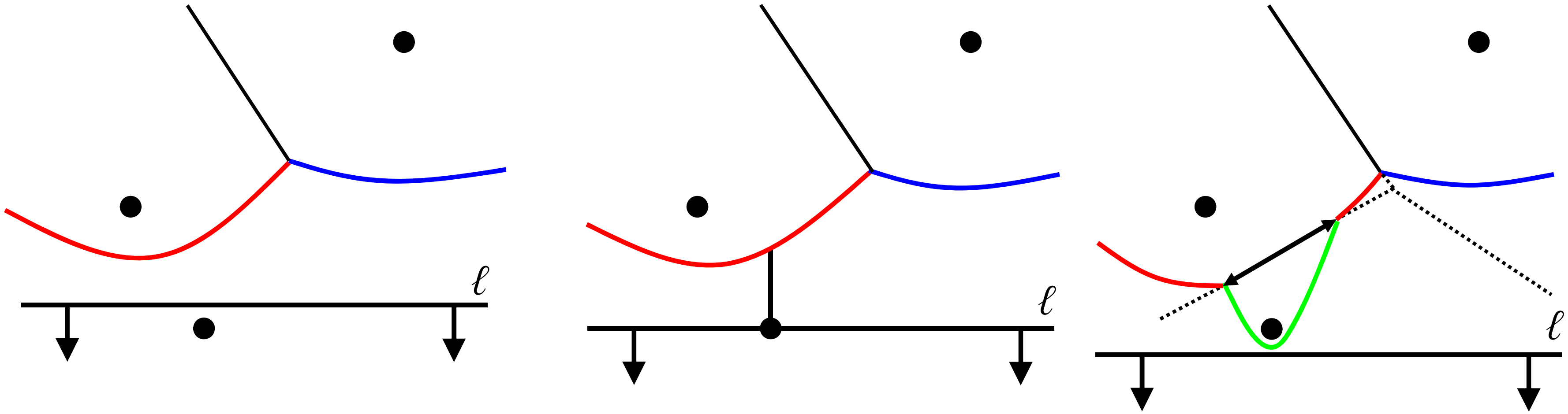


7.2 Voronoi-Algorithm

- Thus, the transitions between arcs lie on edges of the Voronoi-diagram pointing downwards.

➡ Useful to detect Voronoi-edges.

- A new arc occurs, when the sweep-line reaches a new point from P (*site-event*).
- It is possible that one parabola contributes to several arcs.



7.2 Voronoi-Algorithm

Lemma 4

The only way in which an arc can appear on the beach is through a site-event.

Proof

- Assume the opposite, i.e. an already existing parabola breaks through another parabola from above.
- There are two ways how this can happen:

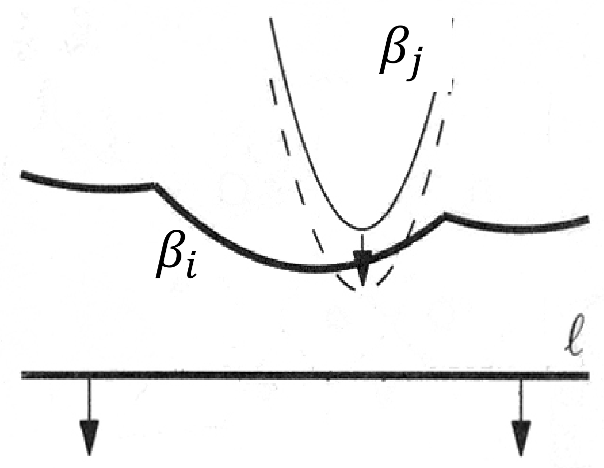
7.2 Voronoi-Algorithm

1. Assume an arc β_j breaks in the middle through another arc β_i from above.

- Then there is a ℓ_y with $\ell_y < p_{i,y}$ and $\ell_y < p_{j,y}$, where β_j and β_i touch in a single point with a common tangent, i.e.

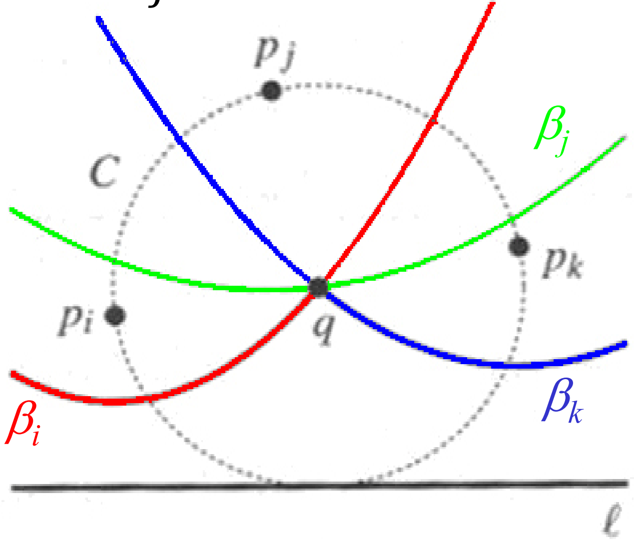
$$\beta_i'(x) = \frac{x-p_{i,x}}{p_{i,y}-\ell_y} = \frac{x-p_{j,x}}{p_{j,y}-\ell_y} = \beta_j'(x)$$

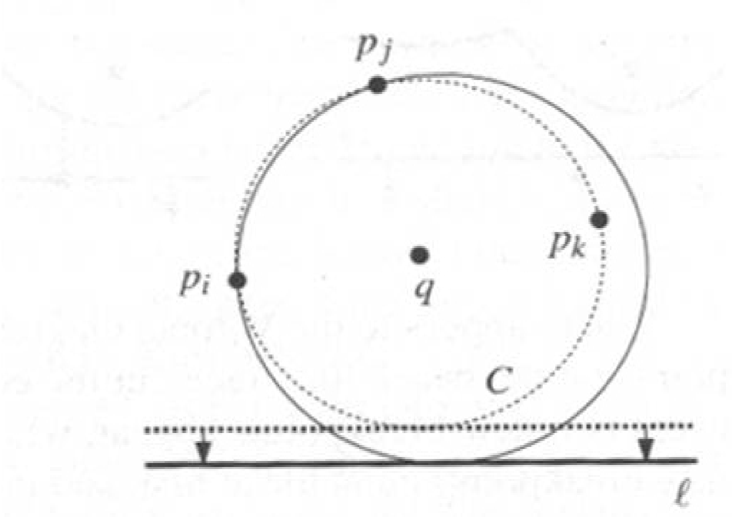
- This yields $\beta_i(x) = \beta_j(x)$, which contradicts that β_j and β_i touch in a single point.



7.2 Voronoi-Algorithm

2. Assume the arc β_j appears at the transition of β_i and β_k.
   - Then there is a circle C through p_i, p_j, p_k, which touches the sweep-line for a certain ℓ_y.
   - Moving ℓ downwards while C remains tangential to ℓ, either p_i or p_k will penetrate the interior of the circle through p_j tangential to ℓ.
   - Thus, p_j cannot add a new arc. □

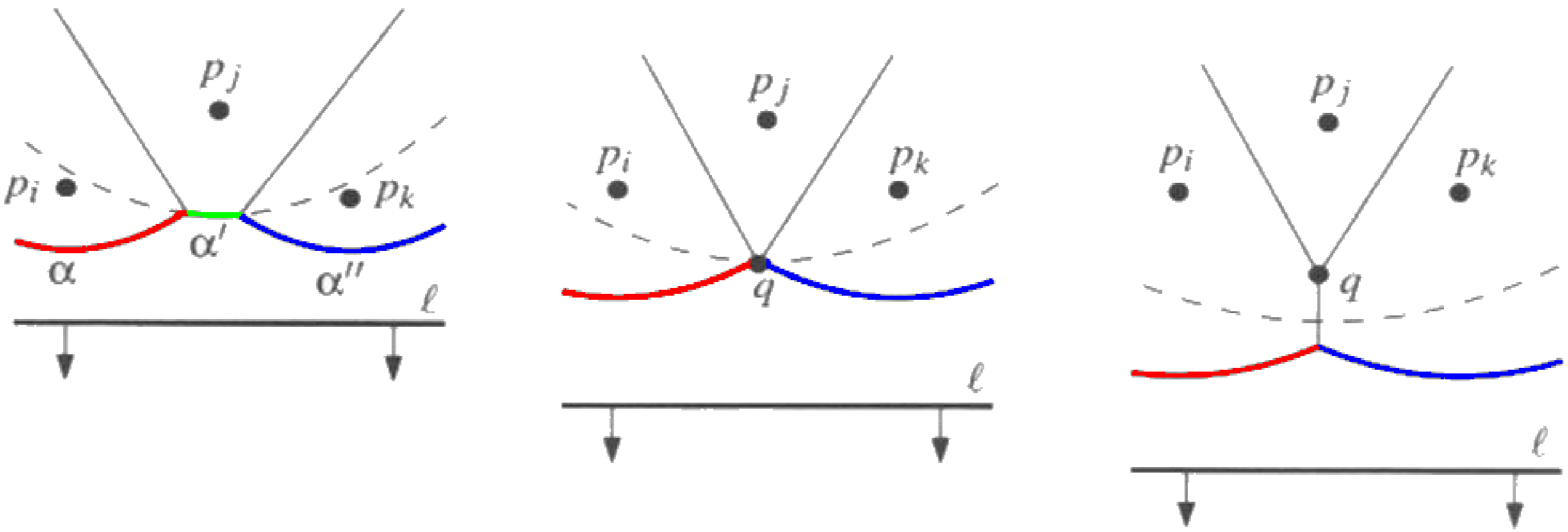




7.2 Voronoi-Algorithm

- The beach consists of at most $2n-1$ arcs, because every site-event adds one new arc and can split one existing arc into two pieces.

➡ There is a second type of events, where an arc degenerates to a point and then vanishes.



7.2 Voronoi-Algorithm

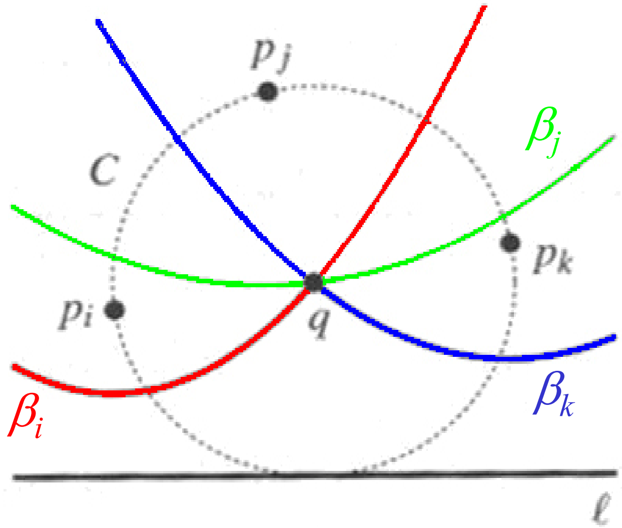
Lemma 5

An arc can only disappear from the beach at a *circle-event*, i.e. an $\boxed{\text{empty}}$ circle C through p_i, p_j, p_k of (previously) adjacent arcs touches ℓ.

No point of P lies in the interior of C.

Proof

- The only alternative would be that adjacent arcs β_i, β_k belong to the same parabola, which cannot happen because of Lemma 4.



- β_j can only degenerate, if the circle is empty. □

7.2 Voronoi-Algorithm

Lemma 6

All Voronoi-vertices can be detected by circle-events.

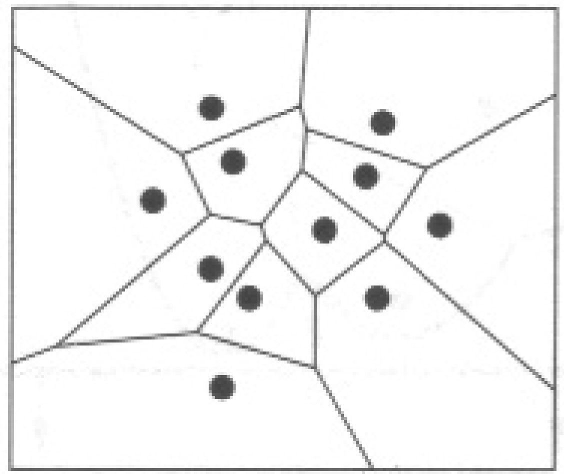
Proof

- We have to show, that the points p_i, p_j, p_k of the adjacent Voronoi-cells have generated three adjacent arcs before the circle-event.
  - Lifting the sweep-line by $\varepsilon > 0$ gives two empty circles tangential to ℓ interpolating p_i, p_j and p_j, p_k respectively.
  - Thus β_i, β_j and also β_j, β_k were adjacent and generated the circle-event.

$\square$

7.2 Voronoi-Algorithm

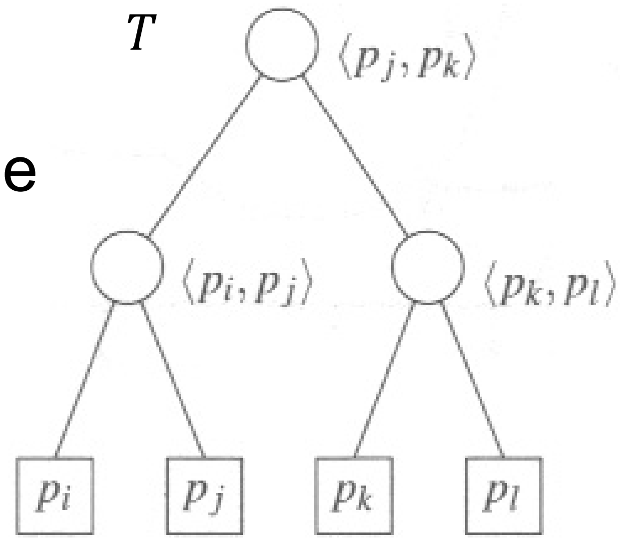
- The data structure for the algorithm consists of three components:
  - A ***priority-queue*** Q for site- and circle-events, where the priority is the y-coordinate.
    - It is initialized with the points from P.
  - A ***balanced search tree*** T, representing the structure of the beach.
    - So, for every event the corresponding arc can be determined in $O(\log n)$.
  - A ***doubly-linked edge list*** D, representing the Voronoi-Diagram.
    - To avoid half-lines it is embedded in a sufficiently large rectangle.

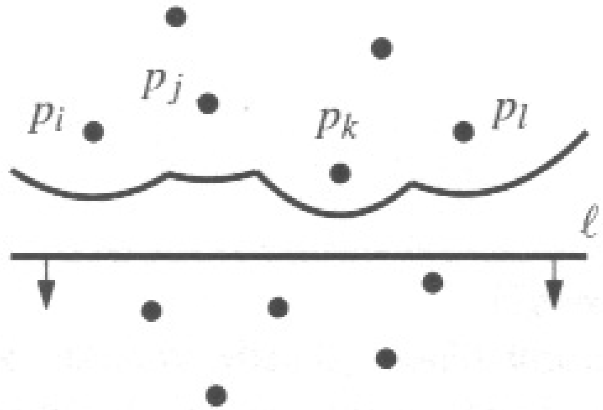


7.2 Voronoi-Algorithm

The balanced search tree

- The *leaves* of T represent the arcs of the beach from left to right and contain a
  - pointer to the generating point and a
  - pointer to the circle-event, removing the arc (if existent) and
  - pointers to neighboring arcs.
- The *inner knots* of T represent the transitions between arcs and contain
  - pointers to the generating points of the corresponding segments and a
  - pointer to the corresponding Voronoi-edge in D.

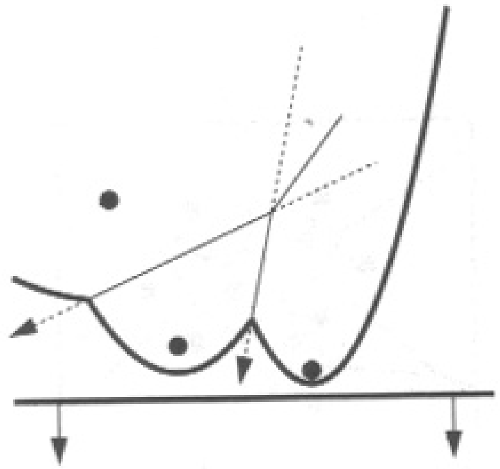




7.2 Voronoi-Algorithm

The priority-queue

- The site-events are known a priori.
- The circle-events must be computed on the fly.
  - For every triple of adjacent arcs a circle-event must be generated, if the transition points do not move away from each other.
  - The y-coordinate for converging edges can be computed.



- A site-event can prevent a previously identified circle-event, which must then be removed from the priority-queue.

7.2 Voronoi-Algorithm

Algorithm 1: Voronoi-Diagram(Point set P)

Input: Set $P = \{p_1, \dots, p_n\}$ of n distinct points in the plane.
Output: $\mathrm{Vor}(P)$ within a bounding-box as edge-list D.

Initialize the priority-queue Q with all site-events;
Initialize an empty search-tree T and an empty edge-list D;
while ($Q \neq \emptyset$) {
  Take event with largest y-coordinate from Q;
  if (The event is a site-event) **then** {
    HandleSiteEvent(p_i), where p_i is the corresponding site;
  } **else** {
    HandleCircleEvent(γ), where γ is the leaf in T representing the arc that will disappear;
}}
The remaining inner nodes of T correspond to the half-infinite edges, that need to be added: Compute a bounding box around all Voronoi-vertices and P and attach the half-infinite edges to the bounding box by updating D;

7.2 Voronoi-Algorithm

Algorithm 2: HandleSiteEvent(p_i)

if (T is empty) **then** Insert p_i to T and **return**;
else {
  Search in T for the leaf γ_j representing arc β_j vertically above p_i;
  if (γ_j points to a circle-event) **then** Remove it from Q;
  a) Replace γ_j in T by a sub-tree with three leaves: the middle stores the new site p_i, the other two p_j;
  b) Store tuples (p_j, p_i), (p_i, p_j) in two inner nodes of T;
  c) Re-balance T;
  Insert between $V(p_i)$ and $V(p_j)$ a new edge to D;
  if (β_i and its two right neighbors generate a circle-event) **then**
    Insert it to Q and add links to relevant leaves in T;
  if (β_i and its two left neighbors generate a circle-event) **then**
    Insert it to Q and add links to relevant leaves in T;
}

7.2 Voronoi-Algorithm

Algorithm 3: HandleCircleEvent(γ)

```
1: a) Remove all circle-events from Q where β_j is involved,
      using the predecessor- and successor-pointers in T;
   b) Remove the leaf γ that represents the disappearing arc
      β_j from T;
   c) Update the inner nodes representing the transition
      points;
   d) Re-balance T;
2: Add a Voronoi-vertex and the new edge between V(p_i) and
   V(p_k) to D, where p_i and p_k generate the neighbor arcs;
3: if (The triples of consecutive arcs containing the
       disappeared arc as transition generate a new circle-
       event) then {
4:    a) Add these circle-events to Q;
      b) Update the pointers in T;
5: }
```

7.2 Voronoi-Algorithm

Proposition 7

Algorithm 1 takes $O(n \log n)$ run time and $O(n)$ memory.

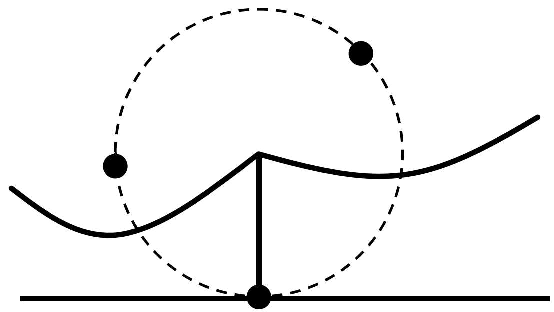
Proof

- The operations on T and Q (search, insert, delete) take each $O(\log n)$ and the initialization $O(n)$.
- Because every event uses a constant number of these operations, every event can be processed in $O(\log n)$.
- There are n site-events and at most $2n - 5$ (number of Voronoi-vertices) circle-events: total run time $O(n \log n)$.
- Because the number of arcs is bounded by $2n - 1$, the memory to store T and Q is linear. $\square$

7.2 Voronoi-Algorithm

Special cases

- If two points have the same y-coordinate, any order is possible.
  - If these are the first two points, there is no arc above the second.
- If two circle-events coincide, four points lie on one circle.
  - For simplicity do not consider this case separately and add an edge of length zero, which is removed in a post-processing step.
- If a site-event occurs at the transition of two arcs, one of the neighbor arcs is split to yield an arc of length zero.
  - This zero-length arc will generate a circle-event, that will remove the zero-length arc when it is processed.



7.2 Voronoi-Algorithm

Complexity

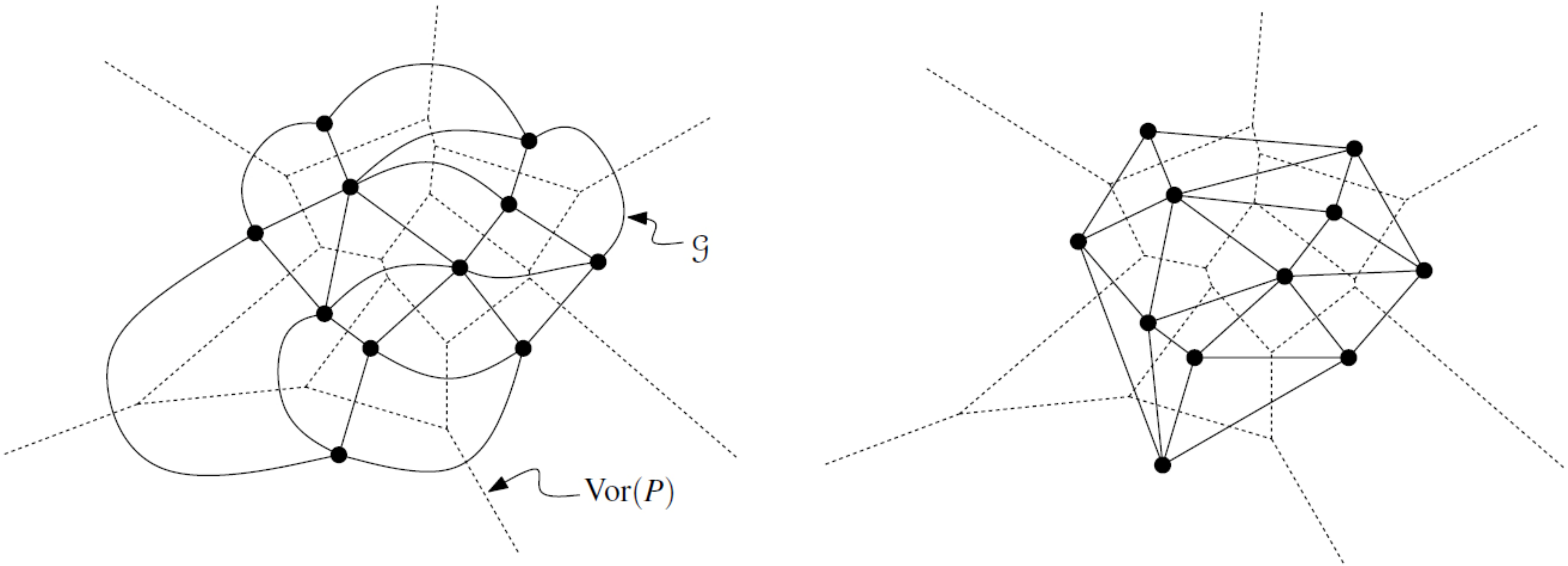
- The points p_i of P with unbounded $V(p_i)$ are the corners of the convex hull $\mathcal{CH}(P)$.
- These points can be determined in $O(n)$
  - Traverse a sufficiently large circle C around P (clockwise) and compute its intersections with Voronoi-edges.

➡ The Voronoi-diagram can be used to compute the convex hull of P.

➡ The complexity to compute the Voronoi-diagram of n points in the plane is $\Omega(n \log n)$.

7.3 Delaunay-Triangulation

- The dual graph $\mathcal{G}$ of a Voronoi-diagram is called *Delaunay-graph*. Boris Nikolajewitsch Delone (1890-1980)
- If no more than three points lie on a circle, the Delaunay-graph is a triangulation of P and its convex hull.



7.3 Delaunay-Triangulation

Proposition 8

1. Three points $p_i, p_j, p_k \in P$ belong to the same face of the Delaunay-graph, if and only if the circumscribed circle contains no further point of P (*Delaunay-condition*).
2. Two points $p_i, p_j \in P$ span an edge of the Delaunay-graph, if there is a circle through these points, containing no further point of P.

Proof

- The proof follows from Proposition 3.
- In 1. the center of the circumscribing circle is the Voronoi-vertex corresponding to this Delaunay-face.

7.3 Delaunay-Triangulation

Proposition 9

Among all possible triangulations the Delaunay-triangulation maximizes the smallest interior angle of all triangles.

Proof: Exercise.

7.3 Delaunay-Triangulation

- A simple algorithm to construct the Delaunay-triangulation.
  - First find a single triangle (or alternatively a bounding box subdivided into two triangles), containing all points of P.
  - Add the points of P in random order to the triangulation and restore the Delaunay-condition.
  - For the case that several points lie on a circle, the corresponding face can be triangulated arbitrarily.
    - There is more than one Delaunay-triangulation in this case.

7.3 Delaunay-Triangulation

Algorithm 4: Delaunay-Triangulation

Input: Set P of n distinct points in the plane.
Output: Delaunay-Triangulation D.

1: Compute a random permutation $p_1, \ldots, p_n$ of the points in P;
2: Initialize D with a triangle $q_1 q_2 q_3$, enclosing all points;
3: **for** $(r = 1, \ldots, n)$ {
4:    Find triangle $p_i p_j p_k$ containing p_r;
5:    Find all Delaunay-conditions that are violated, using the adjacency of triangles;
6:    Remove these triangles from D and replace them by new triangles by connecting all points of these triangles to p_r by edges;
7: }
8: Flip edges $q_i u$ for all triangles $q_i uv$ and $q_i uw$, if q_i violates the Delaunay-condition for triangle uvw.
9: Remove the points q_1, q_2, q_3 and all their triangles from D;

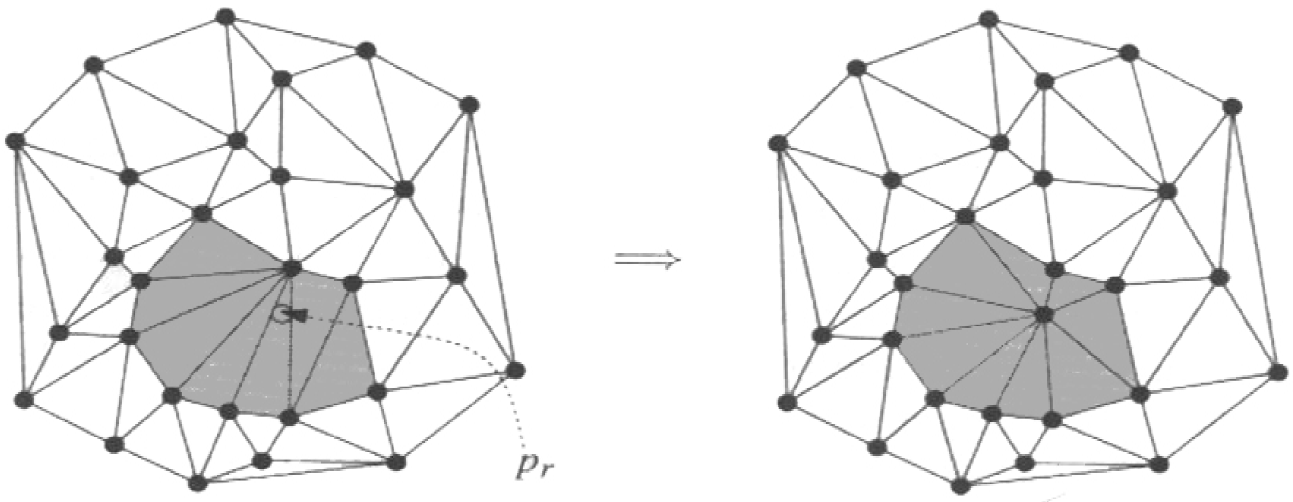
7.3 Delaunay-Triangulation

Proposition 11

Algorithm 4 computes a correct Delaunay-triangulation.

Proof

- We have to prove that the new triangles inserted in line 6 satisfy the Delaunay-condition.



7.3 Delaunay-Triangulation

- Assume one of the triangles t generated during the insertion of p_r does not satisfy the Delaunay-condition.
  - Then an edge between p_r and the polygon Δ, that was generated by removing the triangles, must be flipped.
  - This gives a triangle Γ of consecutive points p_i, p_j, p_k of Δ.

a) If $\Gamma \in D$ before the insertion, Γ was removed, because it violated the Delaunay-condition with p_r.

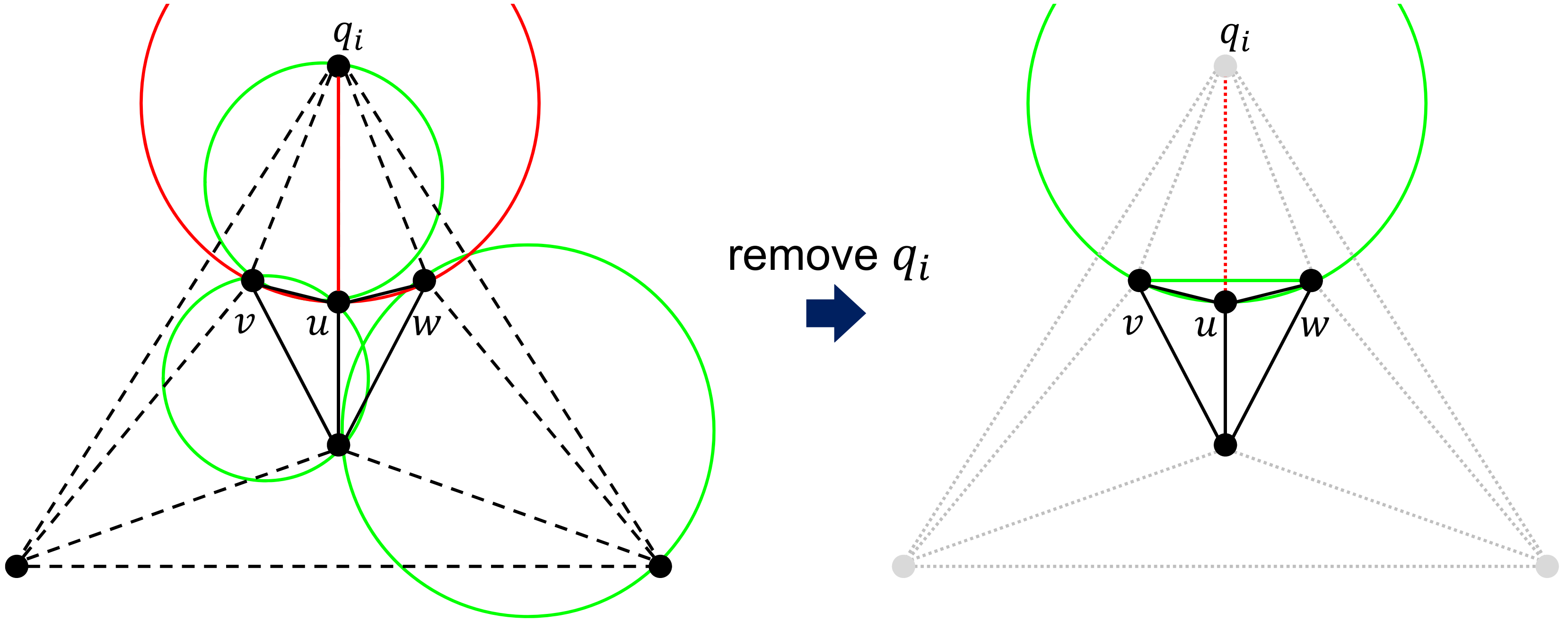
b) If $\Gamma \notin D$ before the insertion, it did not satisfy the Delaunay-condition before the insertion.

- Both cases are a contradiction, proving the proposition. $\square$

7.3 Delaunay-Triangulation

Remark for lines 8 and 9

- Upon removal of the corners q_i of the bounding triangle $q_1 q_2 q_3$, some edges $q_i u$ to these corners need to be flipped, i.e. edges vw need to be inserted.



7.3 Delaunay-Triangulation

- The run time of this algorithm (without any further improvements) is $O(n^2)$, because the search for a single triangle containing p_r takes $O(r)$.
  - Using a search structure (cf. Point-Location) the expected time for this can be reduced to $O(\log n)$.
  - The expected number of triangles that must be removed is constant in every step (see [1]), such that the total expected run time is $O(n \log n)$.
  - In any case the memory is of size $O(n)$.

7.4 Literature

[1] Marc de Berg et al., *Computational Geometry: Algorithms and Applications*, 2nd Edition, Springer, 2000, Chapters 7 and 9.

[2] M.I. Shamos and D. Hoey, *Closest-point problems*, Proc. 16th Annual IEEE Sympos. Found. Comput. Sci., pp 151-162, 1975.

[3] S.J. Fortune, *A sweepline algorithm for Voronoi Diagrams*, Algorithmica, 2:153-174, 1987.

[4] L.J. Guibas et al., *Randomized incremental construction of Delaunay and Voronoi diagrams*, Algorithmica, 7:381-413, 1992.