Computational Geometry

5. Point Location

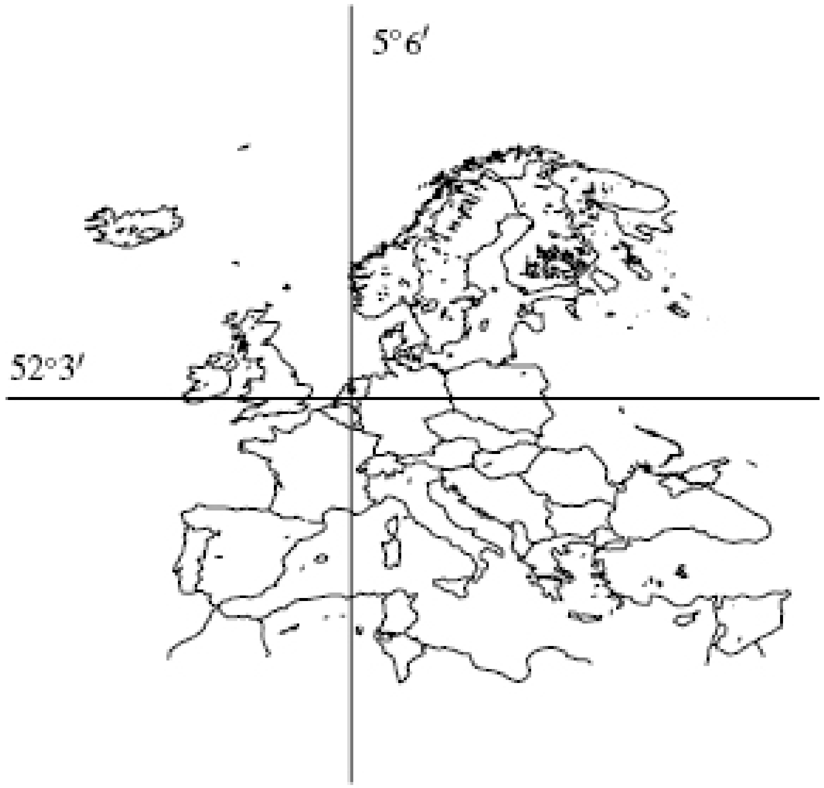
5.1 Motivation

- In Chapter 4 we had special search structures for range queries.
- In this Section will we focus on point queries.
- **Input:** A planar *tessellation* S, i.e. a partition of the plane into polygonal regions without intersections.
- **Goal** of a *point query* is to determine for a given point q the polygon $P \in S$, that contains q.

5.1 Motivation

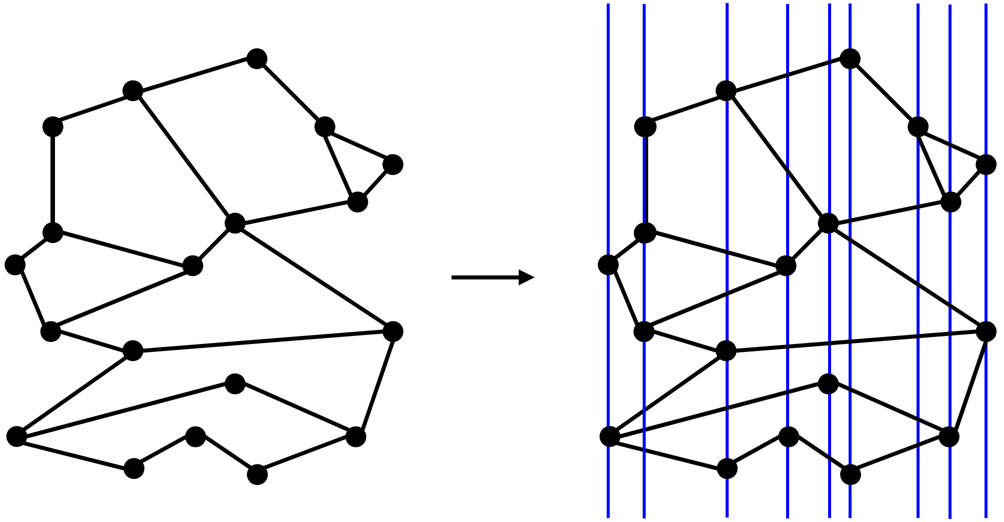
Example

- Find for given geographic coordinates the respective country in a map.
- Such applications with additional real-time requirements are usual for GIS- or navigation-systems.



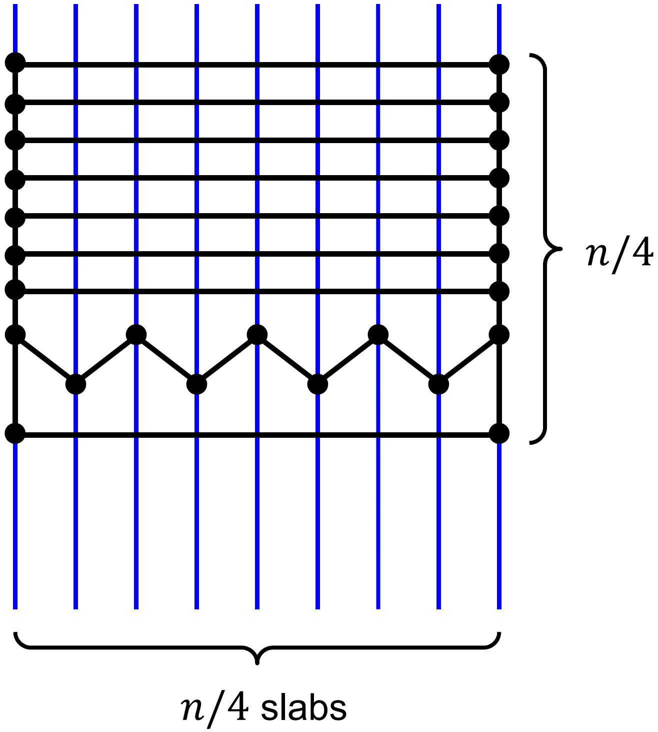
5.2 Trapezoidal Maps

- The search-structure are so-called *trapezoidal maps*.
  - Insert for every vertex p of S a **vertical line**.
  - This yields slabs of trapezoidal pieces,
    - i.e. bounded and unbounded trapezoids and triangles,
  - which can be sorted from top to bottom.
  - Thus, access is possible in $O(\log n)$.



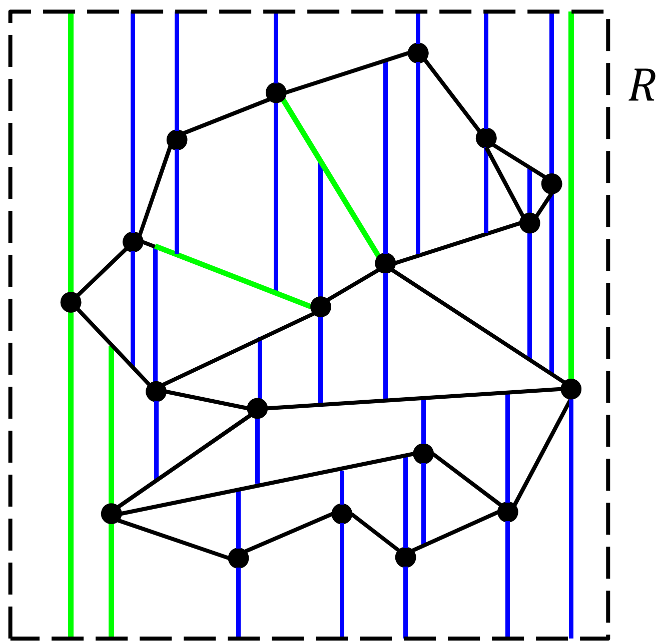
5.2 Trapezoidal Maps

- If a tessellation S contains n edges, a list sorted by x-coordinates contains $O(n)$ slabs.
- Every slab can consist of $O(n)$ trapezoids, requiring $O(n^2)$ of memory.
- For most applications this is not acceptable.



5.2 Trapezoidal Maps

- To reduce the memory consumption
  - the tessellation can be enclosed in a rectangle R containing all polygons of S, thus bounding all unbound slabs, and
  - the *vertical lines* through $p \in S$ go upwards and downwards only until they touch another segment of S or R.
- For simplicity we assume that all vertices have different x-coordinates, i.e. are in *general position*.
- Adjacent and collinear edges are called *sides* of a polygon Δ.
- The resulting tessellation of S is called a *trapezoidal map* $T(S)$.



5.2 Trapezoidal Maps

Lemma 1

For a planar tessellation S in general position and its trapezoidal map $T(S)$ every polygon $\Delta \in T(S)$ has one or two vertical and exactly two non-vertical sides.

Proof

1. First prove convexity of Δ.
   - Because the segments of S do not intersect, a vertex of Δ is either
     a) an endpoint of a segment of S,
     b) a point where a vertical segment starts or ends or
     c) a vertex of R.
   - Because of the vertical lines through the vertices of S there are no inner angles larger than π, so Δ is convex.

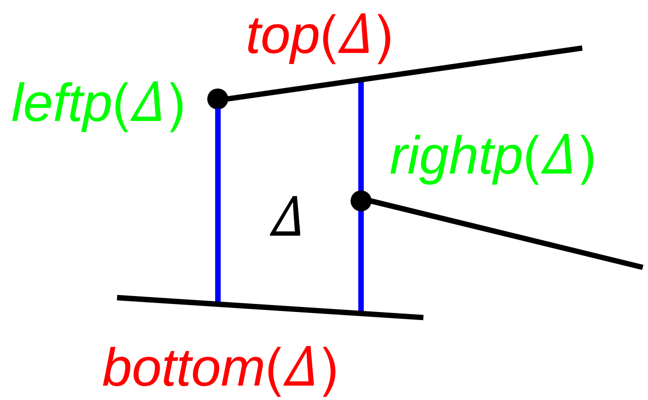
5.2 Trapezoidal Maps

2. Because of convexity Δ has at most two vertical sides.
3. Assume Δ has more than two non-vertical sides.
   1. Because of convexity there must be two of those which are adjacent and bound Δ either from above or from below.
   2. Because non-vertical sides are parts of line segments, these two sides meet in an endpoint/vertex of S.
   3. Because of the vertical lines at vertices the non-vertical sides bounding from above or below cannot be adjacent in Δ, which is a contradiction.
   - Thus, Δ has exactly two non-vertical sides, an upper and a lower non-vertical side.
4. Since Δ is bounded and convex, is has either two (trapezoid) or one (triangle) vertical sides. □

5.2 Trapezoidal Maps

- For $\Delta \in T(S)$ denote by
  - *top(Δ)* the upper and by
  - *bottom(Δ)* the lower

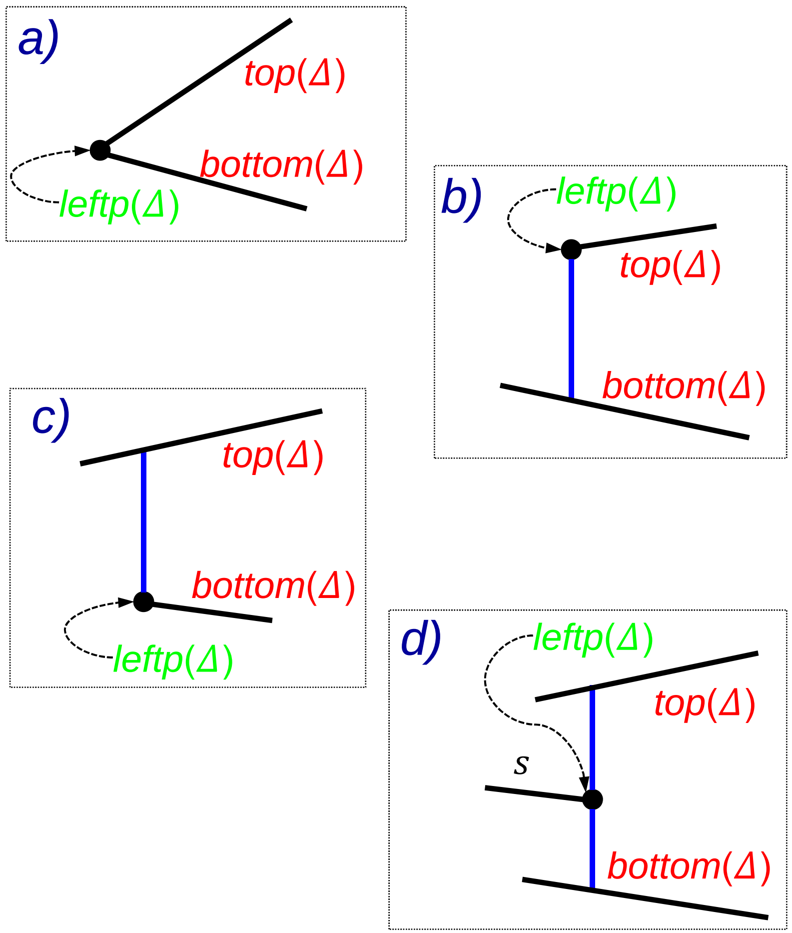
  non-vertical side.



- The left and right side of Δ is either a
  - vertical side, determined by a vertex in S, or
  - a vertex, in case of a triangle.
- For both cases there is a unique vertex in S that determines this side.
  - This vertex is denoted by *leftp(Δ)* and *rightp(Δ)*.

5.2 Trapezoidal Maps

- For a left side of Δ there are five cases:
  - a) *leftp*(Δ) is the left endpoint of *top*(Δ) and *bottom*(Δ),
  - b) *leftp*(Δ) is the left endpoint of *top*(Δ),
  - c) *leftp*(Δ) is the left endpoint of *bottom*(Δ),
  - d) *leftp*(Δ) is the right endpoint of another segment s, that lies left of Δ,
  - e) *leftp*(Δ) is the lower left corner of R.



5.2 Trapezoidal Maps

Lemma 2

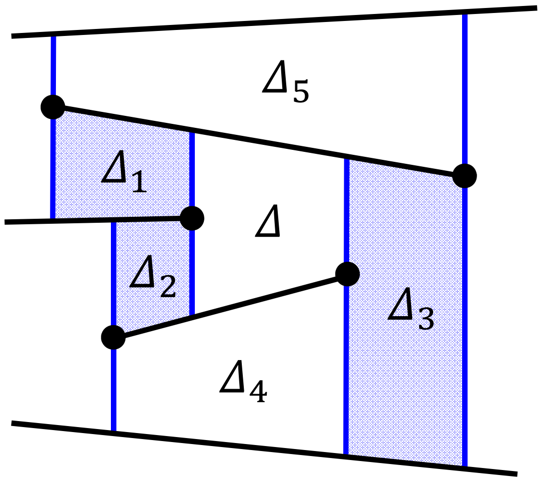
A trapezoidal map $T(S)$ of a planar tessellation S with n edges in general position has at most

- $6n + 4$ vertices and
- $3n + 1$ trapezoids.

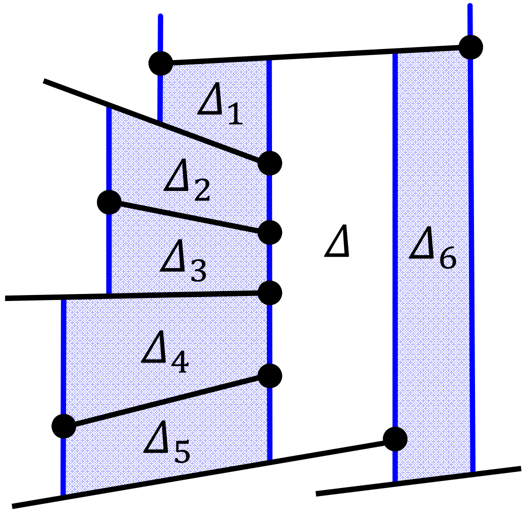
Proof: Exercise!

5.2 Trapezoidal Maps

- Two trapezoids are called *adjacent* if they have a common vertical edge.
- A trapezoid in general position has at most four neighbors, two upper sharing *top*(Δ) and two lower sharing *bottom*(Δ).



Vertices in general position.



Vertices not in general position.

5.2 Trapezoidal Maps

- The data structure for a trapezoid Δ has pointers to
  - the points *leftp*(Δ), *rightp*(Δ),
  - the edges *top*(Δ), *bottom*(Δ),
  - the four adjacent trapezoids.
- The geometry of Δ can be computed from this in constant time.

5.3 Search Data Structure

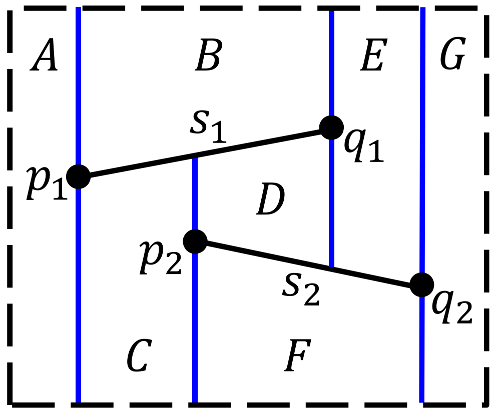
- The trapezoids will be the leaves of a search data structure D, consisting of an directed acyclic graph.
- The search data structure D is similar to a kd-tree (see Section 4), because it has two different types of knots:
  - x-knots, pointing to vertices of S and
  - y-knots, pointing to edges of S.

  ➡ In contrast to kd-trees these might occur in arbitrary order.

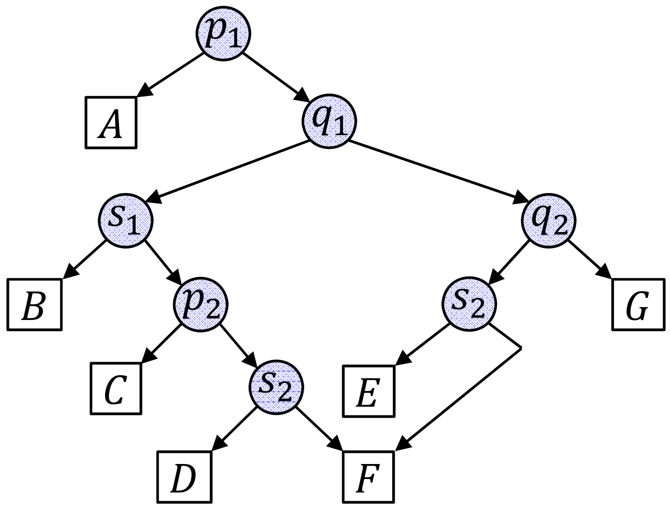
  ➡ It is possible that different knots point to the same trapezoid (see example in the next page).
- As for binary trees every inner knot has two outgoing edges.

5.3 Search Data Structure

- A *point query* for a point q starts at the root and ends in a leaf, corresponding to the trapezoid containing q.
- At a x-knot test if q lies left or right of the vertical line.
- At a y-knot test if q lies above or below the respective line segment.



Trapezoidal map.



Search data structure D.

5.3 Search Data Structure

- The construction of D is done incrementally by insertion of edges of S.
- In order for D not to degenerate to a linear list, the algorithm is randomized:
  - First compute a random permutation of edges $s_i \in S$ and
  - then insert the edges in this order.
- **Invariant:** After the i-th step of the algorithm the trapezoidal map T and the corresponding search data structure D for the tessellation $S_i := \{s_1, s_2, \ldots, s_i\}$ are completely initialized.

5.3 Search Data Structure

Algorithm 1: TrapezoidalMap(S)

Input: Set of non-crossing edges S.
Output: Trapezoidal map $T(S)$ and a search data structure D

1: Compute bounding box of S, use it to initialize T and D;
2: Compute random permutation $s_1, \ldots, s_n$ of edges of S;
3: **for** $(i \coloneqq 1, \ldots, n)$ **do** {
4: Compute all trapezoids $\Delta_0, \ldots, \Delta_k$ in T intersected by s_i;
5: **Update T :** Remove $\Delta_0, \ldots, \Delta_k$ from T and insert new trapezoids generated by s_i;
6: **Update D :** Remove the leaves for $\Delta_0, \ldots, \Delta_k$ from D and insert new trapezoids as leaves: Link the new leaves to inner knots by adding some new inner knots as described below;
7: }

Line 4: Use $T(Si)$ and subroutine FollowSegment on page 21.

Line 5: Operations on T, see page 22ff.

Line 6: Operations on D, see page 22ff.

5.3 Search Data Structure

- The initial data structure $T(S_0 = \emptyset)$ contains only one trapezoid R.
  - Consequently, D has initially only one leaf.
- To get from $T(S_{i-1})$ to $T(S_i)$, all trapezoids $\Delta_0, \dots, \Delta_k$ that are intersected by s_i must be replaced.
  - $\Delta_0, \dots, \Delta_k$ are sorted from left to right.
  - Starting at Δ_0, the other trapezoids can be determined by adjacency:
    - If *rightp*(Δ_j) is below of s_i, Δ_{j+1} is the upper right neighbor of Δ_j.
    - If *rightp*(Δ_j) is above of s_i, Δ_{j+1} is the lower right neighbor of Δ_j.

5.3 Search Data Structure

- To determine Δ_0, do a point-query for the left endpoint p of s_i.
  - This is possible, because the search data structure is already completely constructed for all edges of S_{i-1}.
  - If p is not yet in $T(S_{i-1})$, the query returns the relevant trapezoid Δ_0.

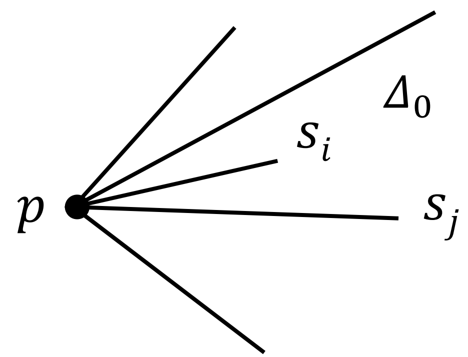
- **But:** It is possible that p exists as endpoint of another segment in $T(S_{i-1})$ on an x- or a y-knot:

1. In the query, p lies on the vertical segment of the respective x-knot.
   - In this case the search is continued in the right sub-tree to find Δ_0.

5.3 Search Data Structure

2. Analogously, the query point p lies on an edge $s_j \in S_{i-1}$ corresponding to a y-knot.

   - This is only possible, if p is an endpoint of $s_j \in S_{i-1}$.
   - In this case the slope of segment s_i with left endpoint is p must be tested:
     - If the slope of s_i is larger than the slope of s_j, continue the search in the upper sub-tree to find Δ_0.
     - Otherwise, continue in the lower sub-tree to find Δ_0.



5.3 Search Data Structure

- To determine the trapezoids that need to be removed use:

Algorithm 2: FollowSegment(T, D, s_i)

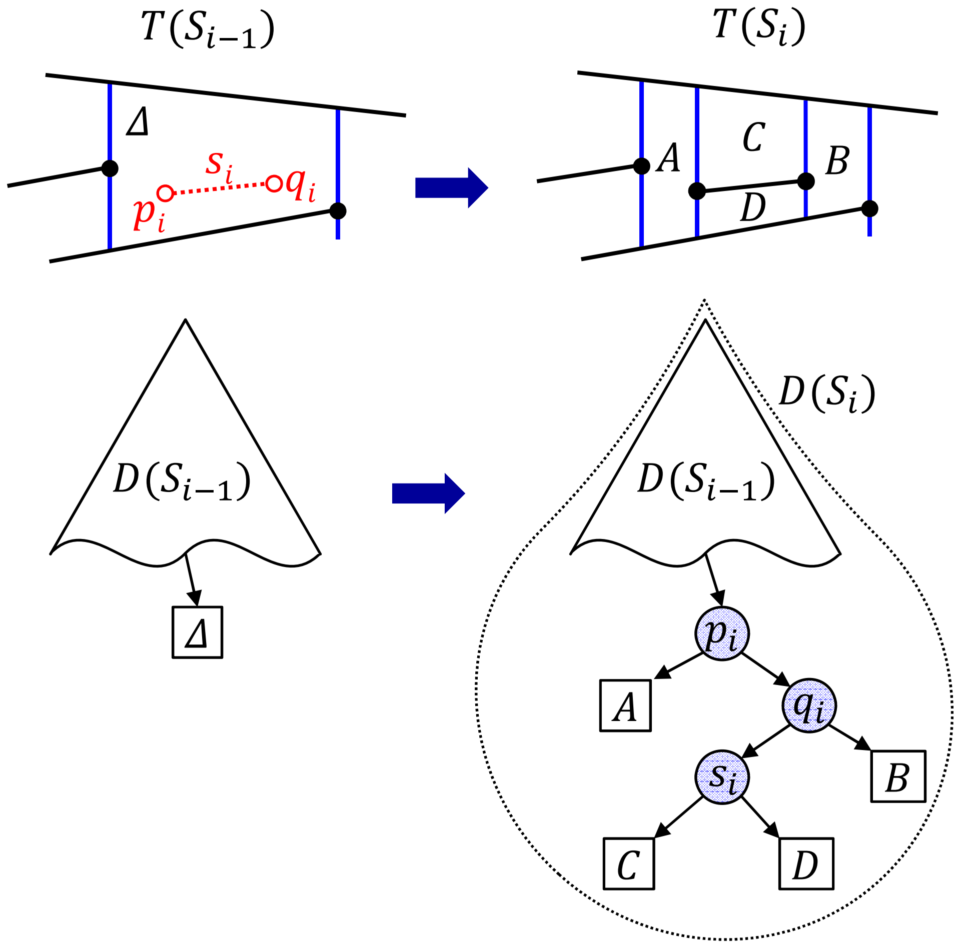
Input: Trapezoidal map T, search data structure D, new edge s_i
Output: Trapezoids $\Delta_0, \dots, \Delta_k$ intersected by s_i

```
Determine the left p and right q endpoint of s_i;
Search in D for p to determine Δ_0;
j := 0;
while (q is left if rightp(Δ_j)) do {
  if (rightp(Δ_j) is above of s_i) then {
    Δ_{j+1} := lower right neighbor of Δ_j;
  } else {
    Δ_{j+1} := upper right neighbor of Δ_j;
  }
  j++;
}
return Δ_0, ..., Δ_j;
```

5.3 Search Data Structure

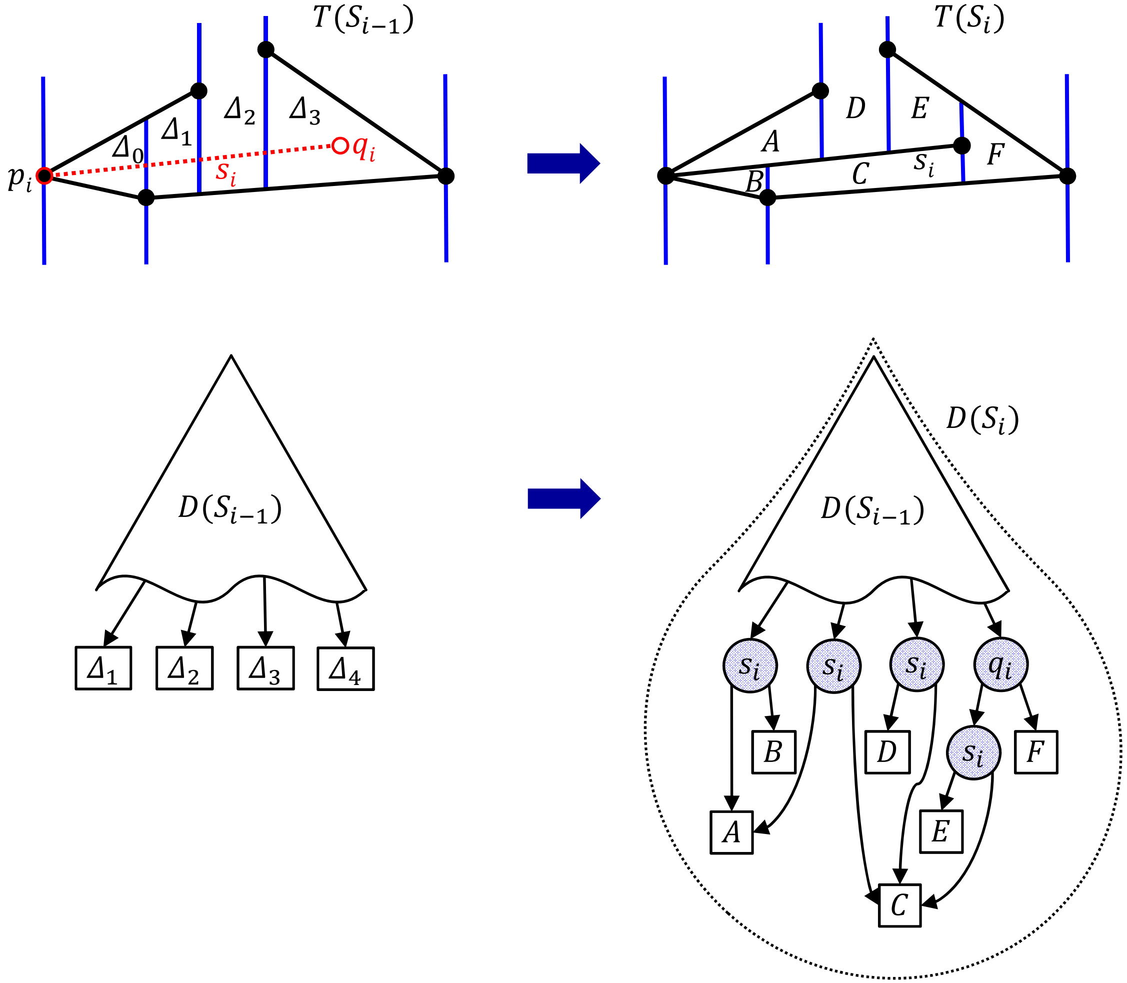
- Now, T and D must be updated.

- In the simplest case s_i is completely contained in one trapezoid Δ, so only Δ needs to be replaced.
- This gives
  - two new x-knots for p_i and q_i,
  - a new y-knot for s_i, and
  - four new trapezoids.



5.3 Search Data Structure

- In general proceed as follows:
  - Use the rules a) − c) on the next page.



5.3 Search Data Structure

a) If p_i lies in the interior of Δ_0,

   1. insert an x-knot to D and
   2. generate a new trapezoid left of p_i. (Analog for q_i in Δ_k).

b) For every intersected trapezoid $\Delta_j, j = 0, \dots, k$,

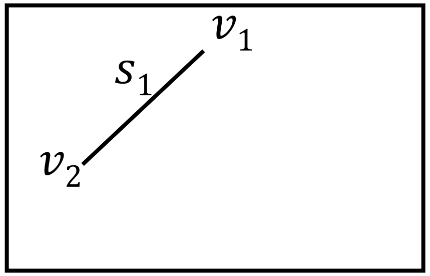
   1. insert a new y-knot for s_i to D and
   2. split Δ_j into an upper and a lower trapezoid.

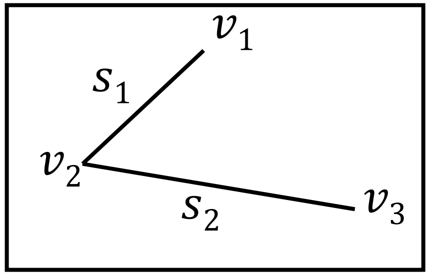
c) Where possible join new trapezoids by shortening of the vertical lines.

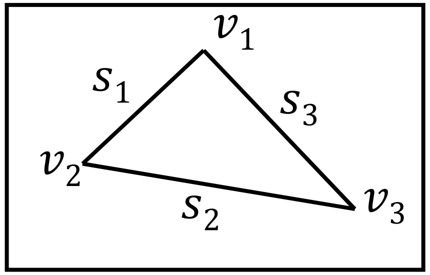
- The pointers to the new trapezoids (*leftp, rightp, top, bottom,* all four neighbors) need to be updated.
- All these operations can be done in $O(k)$.

5.3 Search Data Structure

Example: What are T and D in the following cases?







5.4 Run Time

Proposition 3

1. **Construction time:** Algorithm 1 computes for a planar tessellation S with n edges in general position a trapezoidal map $T(S)$ and the search data structure D in $O(n \log n)$ *expected time*.
2. **Memory:** The *expected* memory used is of size $O(n)$.
3. **Query time:** A point-query is computed in $O(\log n)$ *expected time*.

Proof

- Correctness of this algorithm follows from the invariant of the loop over the segments which are inserted.

5.4 Run Time

3. **Query time:** A point-query for an arbitrary point q in $T(S_i)$ can be computed in expected time $O(\log i)$.

- Let X_i denote the number of knots, by which the length of the search path to find q grows by the insertion of s_i.
- The expected length of the complete search path for q in $T(S_n)$ is

$$E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{n} X_i \cdot P(X_i > 0)\,.$$

- The length grows by at most three knots in every step, so $X_i \leq 3$ and $E(X_i) \leq 3P_i$, where P_i is the probability for $X_i > 0$, i.e.
  - the probability for the search path to grow.

5.4 Run Time

- What is an upper bound for P_i?
  - For the search path to q to grow in the i-th step, the trapezoid Δ_q in $T(S_{i-1})$ containing q is changed by the insertion of s_i.
  - This happens if and only if s_i determines a side of Δ_q, i.e.
    - $s_i = top(\Delta_q)$ or $s_i = bottom(\Delta_q)$, or
    - $s_i \ni leftp(\Delta_q)$ or $s_i \ni rightp(\Delta_q)$.
  - Because the set S_i has i edges and all are equally likely to be s_i, each of these four cases has a probability of at most $1/i$ and $P_i \leq 4/i$.
- This yields

$$E\left(\sum_{i=1}^{n} X_i\right) \leq \sum_{i=1}^{n} 3P_i \leq 12 \sum_{i=1}^{n} \frac{1}{i} = 12\, H_n,$$

where $\ln n < H_n < \ln n + 1$ (harmonic sequence).

- Thus the search for q takes $O(\log n)$ time steps.

5.4 Run Time

2. **Memory:** Due to Lemma 2, $T(S_i)$ has at most $3i + 1$ trapezoids.

➡ Prove that the expected memory of D is also linear.

- It depends on the number of trapezoids and inner knots of D.
- Denote by k_i the number of new trapezoids in step i.
- The number of new inner knots in step i is $k_i - 1$, see pages 23f.
- In the *worst case* $k_i = O(i)$, yielding memory of $O(n^2)$.
- **But:** For the *expected size* we get

$$O(n) + E\left(\sum_{i=1}^{n}(k_i - 1)\right) = O(n) + \sum_{i=1}^{n} E(k_i).$$

5.4 Run Time

- To determine the expected value of k_i use for $\Delta \in T(Si)$ and $s \in S_i$

$$\delta(\Delta, s) := \begin{cases} 1, & \text{if } \Delta \text{ disappears from } T(S_i), \text{when } s \text{ is removed} \\ 0, & \text{otherwise} \end{cases}.$$

- Because there are at most four segments (*top, bottom, leftp, rightp*) causing Δ to disappear, we get

$$\sum_{s \in S_i} \sum_{\Delta \in T(S_i)} \delta(\Delta, s) \leq 4|T(S_i)| = O(i).$$

- Averaging over all edges $s \in S_i$, yields the expected value

$$E(k_i) = \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in T(S_i)} \delta(\Delta, s) \leq \frac{O(i)}{i} = O(1).$$

5.4 Run Time

- Because the expected number of inserted trapezoids and also the expected number of new inner knots is constant for every step, the expected total memory is $O(n)$.

1. **Construction time:** Show that the construction takes $O(n \log n)$.
   - The construction consists of the initialization and
   - for each iteration a point query to find Δ_0 and the insertion of k_i trapezoids and $k_i - 1$ inner knots:

$$O(n) + O(1) + \sum_{i=1}^{n} \Big(O(\log i) + O\big(E(k_i)\big) \Big) = O(n \log n).$$

- This proves Proposition 3. □

5.4 Run Time

Remark

- The analysis of the runtime in Proposition 3 is based on the randomness in Algorithm 1.

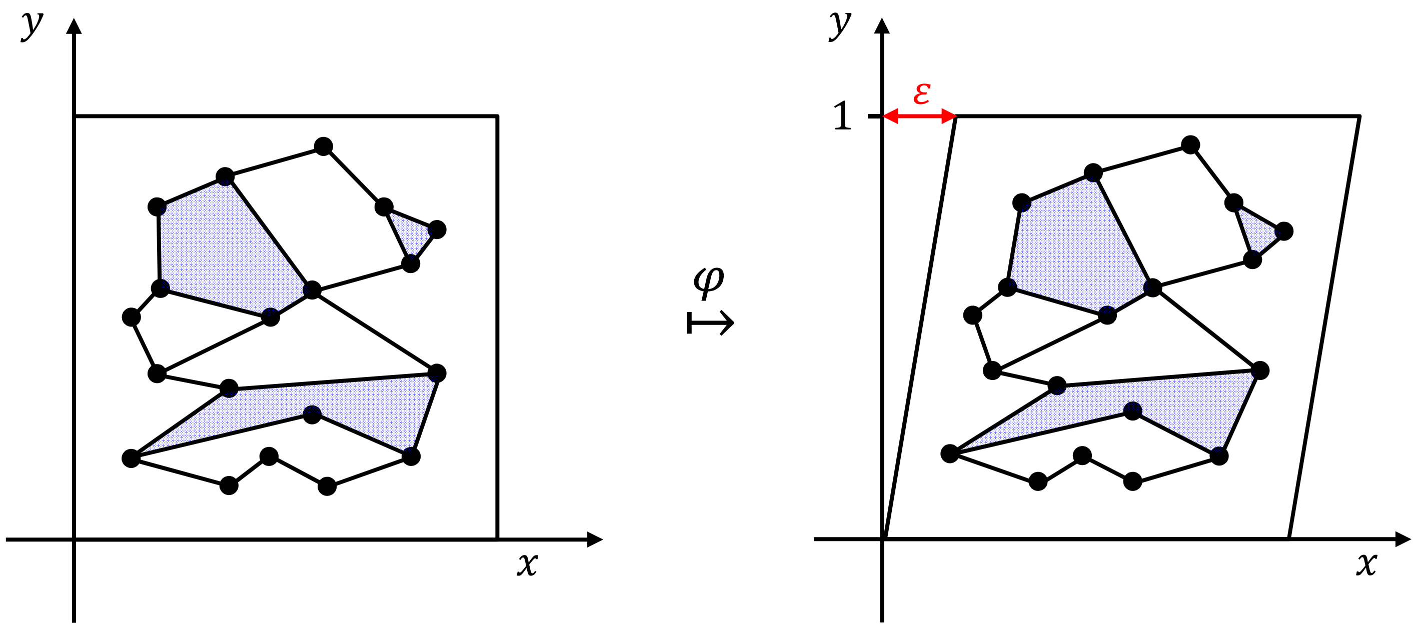
  It does not depend on the randomness in the input.

  Thus, there are no tessellations for which the run time is in principle better than expected.

5.5 Degenerate Cases

- Transformation of the vertices with a suitable shearing yields always points in general position, $\varepsilon > 0$:

$$\varphi : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x + \varepsilon y \\ y \end{pmatrix}.$$



5.5 Degenerate Cases

- Because many algorithms do not use geometric computations, but only relative positions, the shearing can be done symbolically.
  - In Algorithm 2 only two operations change due to the shearing ε:

  a) Is a point q left or right of a vertical line through p. ← Algorithm 2, line 4

  b) Is a point q above, below or on an edge s. ← Algorithm 2, line 5

    - ✶ This is only used when the vertical line through q intersects s.
- For case a) the x-coordinates $x_q + \varepsilon\, y_q$ and $x_p + \varepsilon\, y_p$ need to be compared.
  - Because $\varepsilon \to 0$, compare first only x_q and x_p.
  - If $x_q = x_p$, compare also y_q and y_p.

5.5 Degenerate Cases

- For case b) we have to test, if q is above a line segment s with endpoints (x_1, y_1) and (x_2, y_2), where

$$x_1 + \varepsilon\, y_1 \leq x_q + \varepsilon\, y_q \leq x_2 + \varepsilon\, y_2$$

and also $x_1 \leq x_q \leq x_2$, because of ✶.

  - If $x_1 = x_2$, we have $y_1 \leq y_q \leq y_2$ and q lies exactly on the segment.
  - If $x_1 < x_2$, nothing changes, because a shearing preserves the relations between points and lines.

➡ The approach using a shearing is equivalent with the lexicographical order.

5.6 Literature

[1] M. de Berg et al., *Computational Geometry: Algorithms and Applications*, 2nd Edition, Springer, 2000, Chapter 8.

[2] K. Mulmuley, *A fast planar partition algorithm*, International Journal of Symbolic Computation, 10: 253-280, 1990.

[3] R. Seidel, *A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons*, Comput. Geom. Theory Appl., 1:51-64, 1991.