Point Location Problems

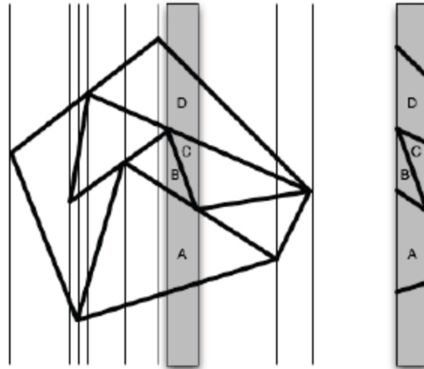
Last time we saw range query, where the query was a d-dimensional box.

Given an arbitrary piecewise linear decomposition of the plane (PSLG), or generalized – a set of segments, we want to know in which regions (or under which line segments) a query point resides.

The basic point location in 2-D: find the face that contains a particular query.

Inefficient query: just check the $O(n)$ faces (according to Euler's formula) for the query point.

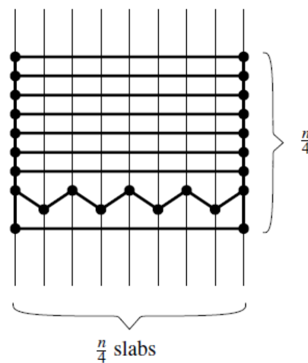
We want an efficient – both time and space – query algorithm, that uses $O(\lg n)$ time and $O(n)$ space.



Trivial algorithm:

- Sort the n segments with respect to the $2n$ endpoints, and cross a vertical line at each endpoint.
- Label all subdivisions of the faces by the faces they are contained in. Note that there is no intersection between two regions.
- In every vertical line there are at most n lines crossing it. So per strip (the grey in the figure above) we can do a binary search.
- Given a query point: identify the x-value of the point and the corresponding strip, and in it find the region. Total searches: 2 – i.e. $2\lg n = O(\lg n)$.

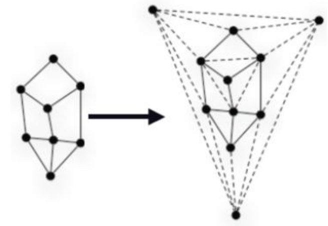
But the data structure is not efficient: $O(n^2)$ space. Example where that happens:



Another problem with this data structure is that it is not dynamic – linear time to update it.

Kirkpatrick's Algorithm

Assumption:



- An option is actually a triangulation. Each given planar graph can be represented as its triangulation with $O(n)$ triangles and $O(n)$ edges.
- Anywhere outside the object there is an infinite space. If a face is not a triangle, we can extend it to a triangle in that infinite region – and that would not change the linear complexity.

The algorithm:

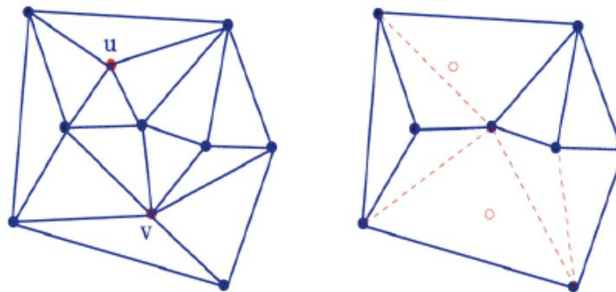
Let T be a map corresponding to a triangulation.

We start with T_0 that has all triangles of the complete triangulation of an object.

At each step we remove a portion of the triangles (some cn) until eventually at the level k we have 1 triangle, and $k = O(\lg n)$.

Each time we drop a vertex, we need to re-triangulate. We cannot drop arbitrary vertices, like vertices with very large degrees. For instance, if a vertex with $\sqrt{n}$ edges (i.e. is part of that much triangles) is dropped, then in the navigation phase, when we get to that level, we need to check, going down the structure, all edges - $\sqrt{n}$. Therefore we can remove only vertices with a **constant** degree. In other words: Each triangle in T_{1+1} overlaps $O(1)$ triangles of T_i.

Lemma: Any planar graph has an independent set of size $\geq \frac{n}{18}$, and every vertex of that set has degree of at most 8.



In the diagram above u, v are independent, and removing them removes a constant number of triangles.

Suppose we want to go from T_i to T_{i+1}. When we drop a vertex, we remove triangles and get a coarser mesh, and we need to triangulate it. If the dropped vertex v degree is d, we drop d triangles, and re-triangulate that mesh with at most $d-2$ triangles. We will cross triangles not between adjacent vertices.

Navigating the structure we get is $O(\lg n)$, and at each phase we do at most 8 tests.

By a greedy algorithm of each time taking the vertices with degrees 8 and descending we get the independent set.

Proof of lemma:

First, all vertices have degree at least 3. By Euler's formula: $\sum_{v \in V} \deg(v) = 2|E| = 6n - 12 < 6n$ (since $e = 3n - 6$).

- Claim 1: the number of vertices with degree ≤ 8 is at least $\frac{n}{2}$:

Assume by contradiction that the number of vertices with degree ≥ 9 is at least $\frac{n}{2}$, and denote that set S_1. We know that for all vertices $v \in V - S_1$ have $\deg(v) \geq 3$, denote that S_2. Adding up: $\sum_{v\in V} \deg(v) = \sum_{v\in S_1} \deg(v) + \sum_{v\in S_2} \deg(v) \geq 9 \cdot \frac{n}{2} + 3 \cdot \frac{n}{2} = 6n$, but that's a contradiction to the upper strict bound above.

Therefore we have a set $S \subseteq V$ such that $\forall v \in S: \deg(v) \leq 8$ and $|S| \geq \frac{n}{2}$.

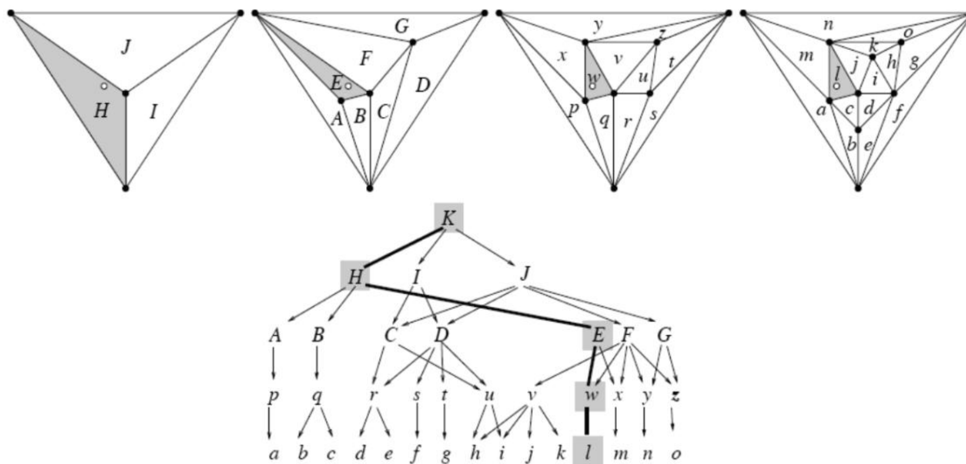
Now we greedily remove vertices with degrees 8. When removing such a vertex, we removed at most 9. We can repeat this process $\frac{|S|}{9}$ times, meaning $|S| \geq \frac{n}{18}$ – our independent set.

Construction of Hierarchy:

- We start with T_0 and an IS S_0 of size $\frac{n}{18}$ with max degree 8.
- In T_1 after removing S_0 we get $n - \frac{n}{18}$ vertices.
- In T_2: $n - \frac{n}{18} - \frac{n-\frac{n}{18}}{18}$.

We never pick the outer triangle's vertices. Therefore, after at most $k = \lg_{\frac{18}{17}} n \cong 12 \lg n$ levels we get to the only triangle Δabc. The tree of T_i is our search tree.

Example:

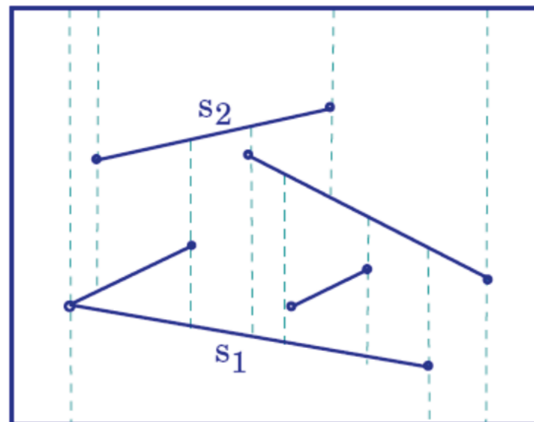


This data structure is in fact **linear**. Counting the elements in the layers of the tree:

$n(1 + 17/18 + (17/18)^2 + \cdots) \leq 18n$ (since it is a converging sum).

What's not good? Adding a new edge will mess up the data structure.

Trapezoidal Maps



A tessellation data structure that supports:

- Input of non-intersecting segments $S = \{s_1, \ldots, s_n\}$
- Query point p, report the segment directly above p.

This is a generalized case of a PSLG. As before, we draw vertical lines **but** we halt at the first segments encountered on the way. With n segments we have $2n$ vertices and $O(n)$ faces.

In order to avoid infinite faces we hold a bounding box around the segments. Assume we have a trapezoidal map, and started with n line segments. Shooting the vertical segments create new vertices. The new tessellation will have at most $6n + 4$ vertices and $3n + 1$ traps (trapezoids).

Proof:

For any segment, we have 2 vertices, each of them will have at most 2 new vertices (crossing the vertical lines), in total $6n$. to that we add the 4 vertices of the bounding box.

Proof that we have $3n + 1$ trapezoids:

Assume every trapezoid is labeled by its left boundary (dotted line), as its support edge on the left. Claim: each one of the right points of the segments can be a support of one trapezoid. We have n such points, so we have n trapezoids to start with. Every left point characterized at most 2 trapezoids – the one to the left and to the right of the line that goes through it. On top of that we add the left bounding box trapezoid, and get a total of $3n + 1$.

The incremental construction: at some stage i when we add a segment, it intersects some of the vertical lines, and we need to also shoot the vertical lines of its endpoints.

Complexity:

The final product is dependent on the order in which we add the segments to the structure.

Lemma: segment i's insertion takes $O(k_i)$ where k_i is the number of new trapezoids introduced from T_{i-1} to T_i.

Proof:

Assume s_i intersects K existing rays, the total of $K + 4$ rays need processing (the additional 4 are from the new ones we shoot from the endpoints of s_i).

If $K = 0$, it is only 4. Otherwise, we need to break it to upper and lower pieces. So we need to add to the DECL 2 vertices. In a DECL each ray processing will cost $O(1)$. In the worst case, $k_i = \Theta(i)$, and if it happens for all i we get a total of $\Theta(n^2)$.

But, with a randomized construction we will get $O(1)$ at each phase. For each s_i the expected number of trapezoids created is constant.

Random construction:

We are looking for $E[k_i]$ over all permutations of order of insertion of s_i to the structure. We will show $E[k_i] = O(1)$.

Proof:

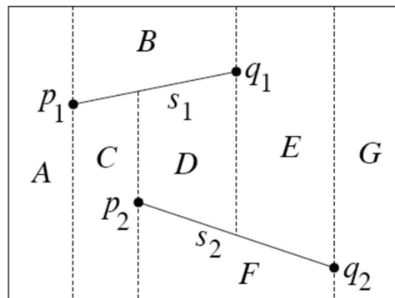
Assume T_i is a map after s_i's insertion. It is dependent on $s_1, \dots, s_i$, but is **not** dependent on the order they were inserted – as that part will always result with the same partial trapezoidal map.

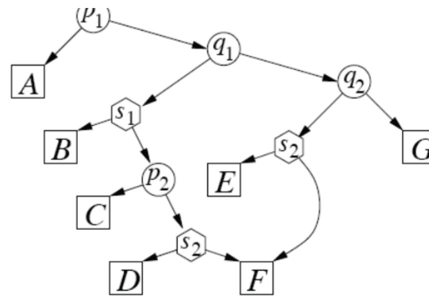
When we reshuffle them, the probability to choose any of them to be the last one is $\frac{1}{i}$. Say Δ is a trapezoid dependent on the last segment added, s, i.e. added when s was inserted as last segment. Let $\delta(\Delta, s) = 1$ if Δ was created as a result of adding s, or 0 otherwise (δ is an indicator value).

$E[k_i] = \sum_{s \in S_i} \sum_{\Delta \in T_i} \Pr[s] \times \delta(\Delta, s)$ – The sum over all probabilities of certain s chosen, times the number of trapezoids dependent on it. That value is: $E[k_i] = \sum_{s \in S_i} \sum_{\Delta \in T_i} \Pr[s] \times \delta(\Delta, s) = \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in T_i} \delta(\Delta, s) = \frac{1}{i} \sum_{\Delta \in T_i} \sum_{s \in S_i} \delta(\Delta, s)$.

Each Δ is dependent on 4 segments. 2 segments are top and bottom segments, and 2 are those with endpoints creating the vertical left and right edges of Δ. Therefore $\sum_{s \in S_i} \delta(\Delta, s) \leq 4$. Now: $E[k_i] = \frac{1}{i} \sum_{\Delta \in T_i} 4 = \frac{4}{i} |T_i| = \frac{O(i)}{i} = O(1)$ – constant □

The data structure:



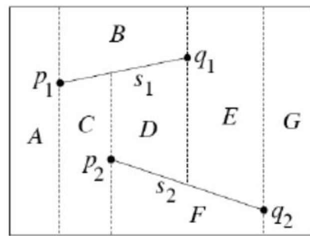


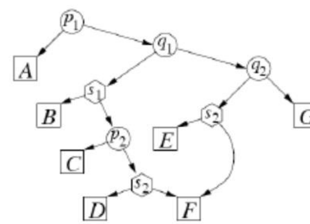
- X-nodes (circles): x-coordinate of segment endpoints. At each point we get to such node, we check whether p the query point is to the left or right to that node.
- Y-nodes (hexagons): pointer to a segment. At each such node, we check whether p is above or below the segment.
- It is a DAG, so no cycles.
- Each leaf is a trapezoid.

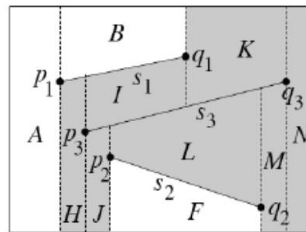
For efficiency, the height has to be $O(\lg n)$, and at every node we need to do a constant amount of work.

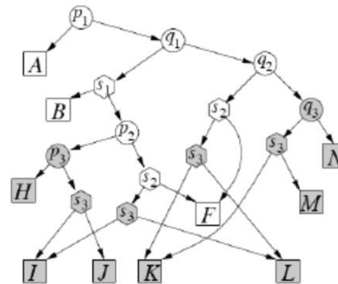
The construction is incremental, i.e. we start with an empty trapezoid and bring in the segments one at a time.

<u>Adding a segment</u>:









Since we have a DECL, we can check k_i times which edges our s_3 intersects (4 checks per each added trapezoid, to locate the edge s_3 intersects). We query the two endpoints of s_3, and the 2 trapezoids they reside in will be replaced (C, G above). In addition, the trapezoids in between will be changed. All changes are local.

Every old leaf (trapezoid) we replace, will create a local DAG with depth at most 3.

Every trapezoid that is completely cut, will be replaced by two trapezoids. It is cut by the segment – so E is replaced by a y node – the segment s_3, with two children - K, L.

When an endpoint region is cut, e.g. G, it is replaced by an x-node, with left and right children. The left child is a segment with two leafs, and the right – a leaf as well. so G was replaced by a depth 2 DAG. Same for C.

<u>Expected complexity</u>:

Space $O(n)$, query $O(\lg n)$.

Analysis: in the slides.