DCG, Wintersemester 2024/25 Lecture 15 (5 pages)

Triangulation Hierarchy

Anne Driemel and Herman Haverkort updated: December 4, 2024

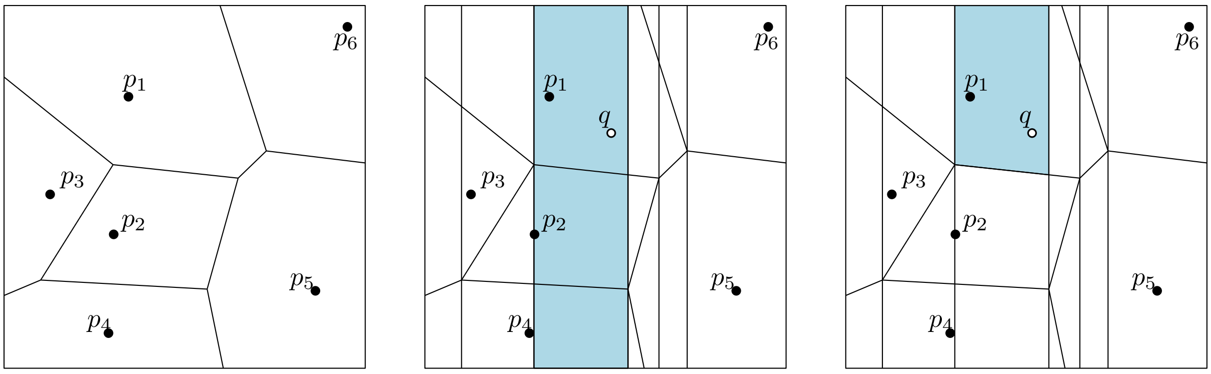
In many applications in computational geometry, we are given a subdivision of the plane, and we need to find the cell that contains a given point. An example of this is the subproblem we encountered in the first algorithm for computing the Delaunay triangulation in Lecture 9. When adding a new point to the triangulation, the algorithm needs to find the triangle of the current triangulation that contains the new point, in order to update the triangulation.

1 Nearest-neighbor searching

Another example is nearest-neighbor searching. Assume we are given a finite set P of points in the plane and, subsequently, we want to ask queries of the following type: given a point q in the plane, which is the point in P that is closest to q? We call this point the nearest neighbor of q.

How do we solve the problem? A naive solution could be to perform n distance computations for every query and to return the point for which the distance computation yields the minimum value. However, this takes $O(n)$ time per query. If we have to perform many queries, then it might be worthwile to compute the Voronoi diagram of P. But how do we find the cell that contains a query point fast? This is the point location problem in the subdivision defined by the Voronoi diagram.

A first solution to solve the point location problem could be to insert vertical lines at every vertex of the decomposition. In such a subdivision, we would be able to perform a binary search on the x-coordinate of the point, followed by a binary search on the segments intersecting the slab between two vertical lines that contains the point. See below for an illustration.



In the above example, the Voronoi diagram to the left is overlayed with vertical lines placed at each of the Voronoi vertices. This allows to find the vertical slab that contains q via binary search in $O(\log n)$ time and then the intersection of the slab with the Voronoi cell that contains the query point q, with a second binary search, in $O(\log n)$ time. The worst-case running time for a query is $O(\log n)$, but the space complexity may be quadratic in n due to the intersections of the vertical lines with the edges of the subdivision.

2 Triangulation hierarchy by Kirkpatrick

We now discuss a data structure that uses merely $O(n)$ space and allows to do point location queries in $O(\log n)$ time. The data structure takes as input a triangulation which has at most three vertices on the unbounded face. (This can also be applied to searching in a Voronoi diagram if the diagram is restricted to a bounded region, if one adds additional edges to turn the Voronoi diagram into a triangulation.)

The main idea of the data structure is to build a sequence of triangulations, reducing the number of vertices in each step, and then building a graph on top of the triangulations, thereby linking the triangles in these triangulations with each other, such that it is possible to search among them efficiently. We first prove some basic facts on the number of vertices of low degree in a triangulation, that we will use in the analysis.

Lemma 15.1. *A triangulation with n vertices has at least $\lceil \frac{k-5}{k+1} n \rceil$ vertices of degree at most k, for any $k \geq 5$.*

Proof. The lemma is easy to verify for $n \leq 2$. Now suppose we have a triangulation with $n \geq 3$ vertices. Let f be the number of faces, and let e be the number of edges. Since the boundary of every face (include the outer face) consists of at least three edges, and each edge is on the boundary of only two faces, we have $3f \leq 2e$, and therefore:

$$f \leq \frac{2}{3}e.$$

Euler's formula tell us $e = f + n - 2$. Plugging in the above yields $e \leq \frac{2}{3}e + n - 2$, which we can rewrite as:

$$2e \leq 6n - 12.$$

Now let d_i denote the degree of the ith vertex. We know that the sum of vertex degrees is equal to $2e$, since every edge is incident to exactly 2 vertices, so each edge is counted twice in the sum of vertex degrees. Therefore:

$$\sum_{i=1}^{n} d_i \leq 6n - 12.$$

Let h be the number of vertices in the triangulation that have vertex degree greater than k. Then it holds for the sum of vertex degrees:

$$\sum_{i=1}^{n} d_i \geq h(k+1).$$

Thus we have:

$$6n - 12 \geq \sum_{i=1}^{n} d_i \geq h(k+1)$$

which implies

$$h \leq \frac{6n-12}{k+1}$$

Now, there are $n - h$ remaining vertices which have vertex degree at most k. By the above, we have

$$n - h \geq n - \frac{6n-12}{k+1} = \frac{k-5}{k+1}n + \frac{12}{k+1} \geq \frac{k-5}{k+1}n$$

Since the number of vertices of degree at most k is a natural number, we may round the lower bound to the next natural number. □

Corollary 15.2. *A triangulation with n vertices has at least $\lceil \frac{n}{2} \rceil$ vertices of degree at most $k = 11$.*

Definition 15.3 (Independent set)**.** *In a graph $G = (V, E)$ an independent set is a subset of the vertices $S \subseteq V$, such that no two vertices of S share an edge in G.*

Lemma 15.4. *There exists a natural number C, such that, given the DCEL of a triangulation of $n \geq C$ vertices, at most 3 of which are on the unbounded face, we can compute in $O(n)$ time an independent set of vertices S of size at least $m = \lceil \frac{n}{25} \rceil$, such that each vertex in S has degree at most $k = 11$ in G, and none of them lie on the unbounded face of G.*

Proof. We can compute the vertex degree of each vertex by counting the number of edges incident to it by using the *next* and *twin* pointers of the DCEL. Since the total number of vertex-edge incidence pairs is in $O(n)$, the total running time is linear in n. Thus, we can identify all vertices of degree at most k in $O(n)$ time. Let this set be W. By Corollary 15.2, we have for the size of this set $|W| \geq \lceil \frac{n}{2} \rceil$. Note that some vertices of W (at most three) may lie on the unbounded face.

We use a greedy algorithm to choose the independent set from W. In each step, we choose a vertex of W, which does not lie on the unbounded face of G, and we remove this vertex including all of its neighbors from W. Thus, in each step, we remove at most $k + 1$ vertices from W and increase the size of our independent set by 1.

After $m - 1$ rounds, the size of W has decreased by at most $(m - 1)(k + 1)$ for $k = 11$ and the computed independent set has size $m - 1$. Then, we choose one more vertex from W, which should not be on the unbounded face. This would certainly be possible, if it holds that

$$|W| - 3 \geq (m - 1)(k + 1) + 1 \tag{1}$$

This is equivalent to

$$\frac{|W| - 4}{k + 1} + 1 \geq m$$

By Corollary 15.2, we have $|W| \geq \lceil \frac{n}{2} \rceil$ and further

$$\frac{|W| - 4}{k + 1} + 1 \geq \frac{\lceil \frac{n}{2} \rceil}{12} + \frac{2}{3} \geq \frac{n}{24} + \frac{2}{3}$$

For large enough n (say $n \geq 200$), it holds that

$$\frac{n}{24} + \frac{2}{3} \geq \left\lceil \frac{n}{25} \right\rceil$$

Thus, for $m = \lceil \frac{n}{25} \rceil$ the inequality in (1) is implied. □

2.1 Building the data structure

Now, to build the data structure, we start with the triangulation T_0, which is given as a DCEL, and we generate a sequence of triangulations $T_1, \ldots, T_k$, with decreasing number of vertices. Triangulation T_i is generated from T_{i-1} in the following way. If the number of vertices in T_{i-1} at least C, for a suitable constant C, then we apply Lemma 15.4 to compute an independent set of vertices, each of degree at most 11. We remove these vertices from T_{i-1} and triangulate the faces that have more than three edges on their boundary, by adding additional edges. If the number of vertices in T_{i-1} is less than C, then we remove all inner vertices and obtain the last triangulation T_k, which only contains the bounding triangle of T_0. Figure 1 shows an example of the sequence of triangulations computed by the algorithm.

In addition, we build a directed graph G, which has a node for each triangle of one of the triangulations. Two different triangles of triangulation T_i and T_{i-1} are connected by an edge if they have non-empty intersection. We define the node that corresponds to the bounding triangle that is contained in T_m as the root node of G.

Figure 2 shows the resulting graph of the data structure.

2.2 How to query the data structure

A query with a point q starts at the root node of G and follows a path to a node of a triangle in T_0, by following the triangles that contain q. In each step, the query checks all outgoing edges, if the query is contained in the corresponding triangle. Since every node has a constant number of outgoing edges, this can be done in constant time per node.

Figure 2 shows the data structure for the sequence of triangulations in Figure 1, and a query with a point q in this data structure.

2.3 Analysis

Theorem 15.5. *Given a triangulation with $n \geq 4$ vertices, at most 3 of which on the unbounded face, one can build in time $O(n)$ a data structure of size in $O(n)$, such that when queried with a point $q \in \mathbb{R}^2$, the data structure returns the triangle that contains q in $O(\log n)$ time.*

Proof. How many triangulations are created by the data structure? By Lemma 15.4 the number of vertices decreases by at least a factor of $\frac{1}{25}$ in each round. The process stops when at most C vertices are left. So we are looking for the smallest natural number r that satisfies

$$\left(\frac{24}{25}\right)^r n \quad \leq \quad C$$

This is equivalent to

$$r \cdot \log_2 \left(\frac{25}{24}\right) \quad \geq \quad \log_2 \left(\frac{n}{C}\right)$$

The smallest natural number r satifying this is

$$r \quad = \quad \left\lceil \frac{\log_2\left(\frac{n}{C}\right)}{\log_2\left(\frac{25}{24}\right)} \right\rceil \in O(\log n)$$

The construction time is linear in each step, so the total number of steps is bounded by

$$\sum_{i=0}^{r} \left(\frac{24}{25}\right)^r n \leq n \cdot \sum_{i=0}^{\infty} \left(\frac{24}{25}\right)^i \leq 25n$$

where the last step follows from the bound on the geometric series. This also bounds the space complexity.

The query time is linear in the maximum length of a path from the root to a triangle in T_0. By the above analysis, this is in $O(\log n)$. $\square$

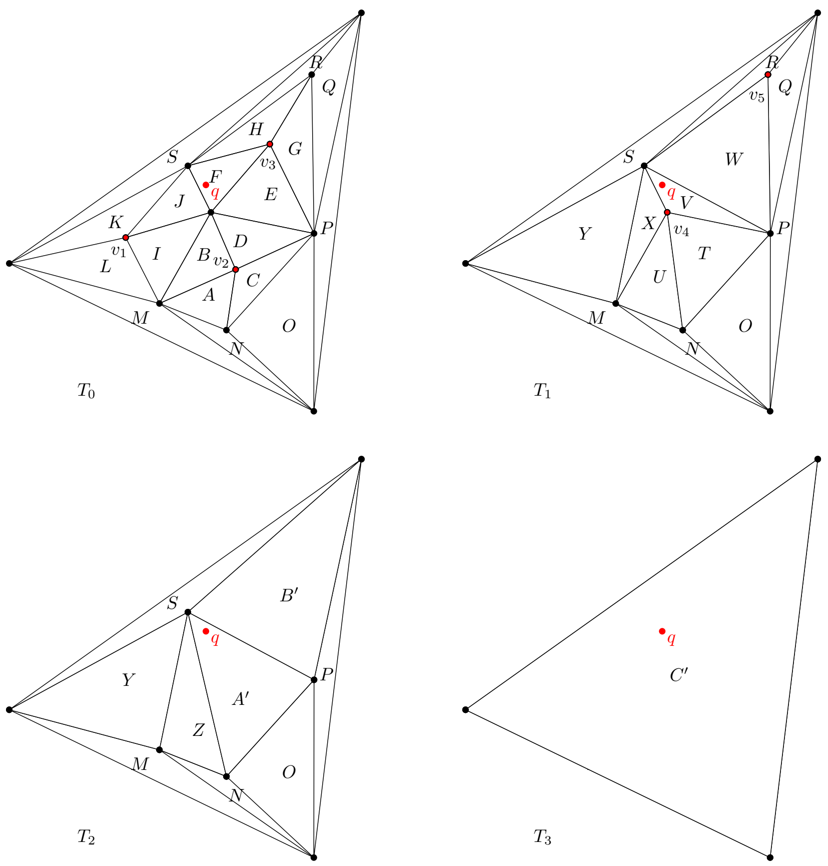


Figure 1: Sequence of triangulations computed by the algorithm. In the first step, the algorithm removes vertices v_1, v_2 and v_3. The triangles $A, B, C, D, E, F, G, H, I, J, K, L$ are replaced by the triangles T, U, V, W, X, Y. In the second step, the algorithm removes vertices v_4 and v_5. The incident triangles R, Q, T, U, V, W, X are replaced by Z, A', B'. The last triangulation in the sequence only contains the bounding triangle.

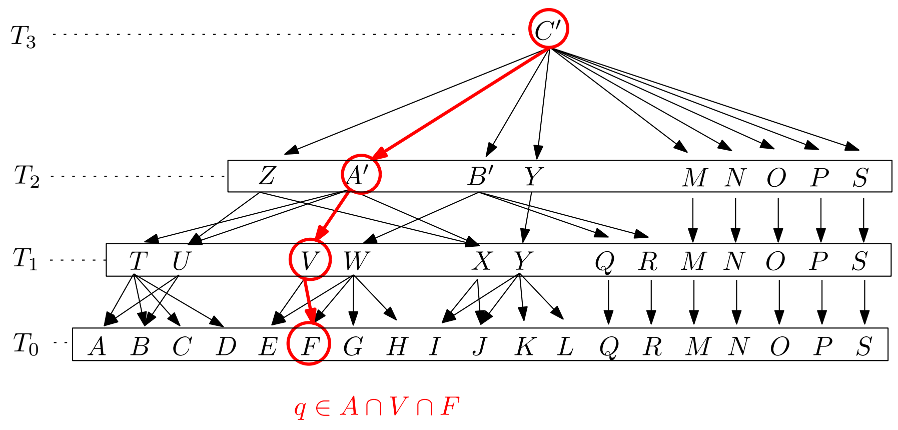

T_3
C'
T_2
Z A' B' Y M N O P S
T_1
T U V W X Y Q R M N O P S
T_0
A B C D E F G H I J K L Q R M N O P S
$q \in A \cap V \cap F$

Figure 2: Example of a query with a point q in the triangulation hierarchy.