DCG, Wintersemester 2024/25 Lecture 9 (6 pages)

Computing Delaunay Triangulations

Anne Driemel and Herman Haverkort updated: November 7, 2024

In this lecture we want to analyze the randomized incremental algorithm to compute the Delaunay triangulation, which we discussed in the previous lecture.

Before we can analyze the algorithm, there are two issues we need to solve:

(i) How to find the triangle in T that contains p_i?

(ii) How to bound the number of edge flips done by the recursive calls to LegalizeEdge ?

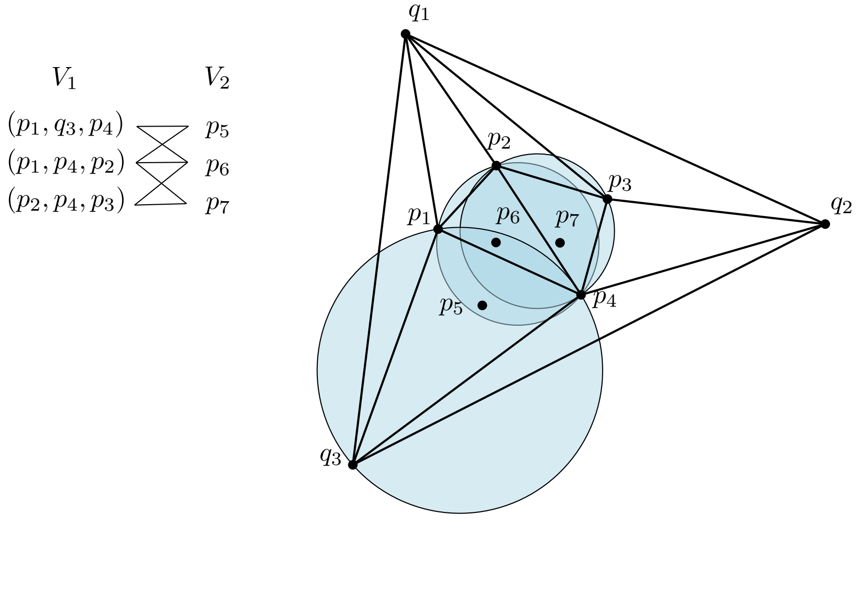
The first question is an example of a more general point location problem, which we will discuss in more detail later in the course. In this lecture we will circumvent the problem by using a conflict graph. For the second question, we have seen that the algorithm may perform many edge flips during one update step, potentially a linear number of them, so we need to be careful when analyzing them. For solving these issues we proceed as before, by maintaining a conflict graph.

1 Conflict graph

We first discuss what constitutes a conflict for our algorithm. We say a triangle (a, b, c) of the triangulation is in conflict with a point p, if adding this point to the triangulation would cause this triangle to be removed. This is the case if and only if the point p lies inside the circle through a, b, and c. Indeed, a triangle which is present in the Delaunay triangulation of $p_1, \ldots, p_{i-1}$, does not contain any of the points $p_1, \ldots, p_{i-1}$ in its circumcircle. Therefore, by Lemma 8.6, the condition that the next point p_i lies outside this circle is equivalent to the triangle being present in the Delaunay triangulation of the set $p_1, \ldots, p_i$.

We will change the algorithm from Lecture 8, so that we can maintain the conflict graph $G = (V_1 \cup V_2, E)$ of the triangles of the current triangulation (the set V_1) and the remaining points that still need to be added (the set V_2). An edge $(u, v) \in V_1 \times V_2$ is in E if u is in conflict with v.

Example 9.1. *The figure below shows an example of a Delaunay triangulation and the conflict graph $G = (V_1 \cup V_2, E)$ of the set of triangles V_1 with the points $V_2 = \{p_5, p_6, p_7\}$. Triangles with empty neighborhood are omitted.*

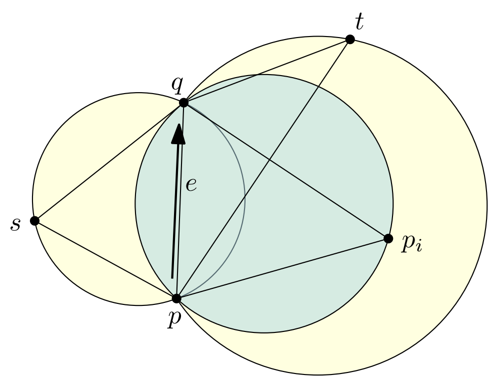


With G at hand, the first issue (i) above could be solved by checking the neighborhood of the point p_i in G, that is, the list *Conflicts*(p_i) of the triangles in the current triangulation T that are in conflict with p_i. However, we also have the second issue of bounding the number of flips. Ideally, we also want to avoid flipping and instead remove the set *Conflicts*(p_i) at once and re-triangulate the face that is created by removing this set of triangles. We can observe, that all new triangles to be added must be incident to p_i, since, by Lemma 8.6, any triangle that has the Delaunay property with respect to the set $p_1, \ldots, p_i$ which is not incident to p_i is already present in the Delaunay triangulation of $p_1, \ldots, p_{i-1}$. So the only triangles that we can add without breaking the Delaunay triangulation are triangles incident to p_i. This leads to the definition of horizon edges, which are the edges that form the new triangles with p_i.

Definition 9.2 (Horizon edge)**.** *Let T be a Delaunay triangulation stored in a DCEL and let p_i be a point in $\mathbb{R}^2$. Let e be a half-edge of T. If the face $IncidentFacet(e)$ is not in conflict with p_i whereas the face $IncidentFacet(Twin(e))$ is in conflict with e, then we call e a horizon edge.*

2 The algorithm

The algorithm is now as follows. In the first step, we compute a random permutation of the input points. Then, the algorithm computes a suitable bounding triangle t with vertices q_1, q_2, q_3, such that none of the vertices invalidates any of the Delaunay triangles of P. The bounding triangle is used to initialize the triangulation. The conflict graph is initialized by adding an edge between t and each of the points $p_1, \ldots, p_n$. When adding a point p_i, the algorithm computes the cycle of horizon edges that bounds the set of faces in *Conflicts*(p_i). The algorithm removes all triangles that are in conflict with p_i, and for each horizon edge, the algorithm inserts a new triangle that is formed by p_i and the horizon edge. Note the similarities to Algorithm 8.1 for computing convex hulls in $\mathbb{R}^3$. Even for updating the conflict graph, we can re-use Lines 16–23 of the algorithm from Lecture 8. We can show that it is sufficient to scan the set of points that are in conflict with one of the triangles incident to the horizon edge (or its twin edge), for computing the set of points that are in conflict with the new triangle created with this horizon edge. This is proven with the following lemma.



Lemma 9.3. *Let e be a horizon edge from p to q. Assume that a point p_i is not in conflict with triangle $t_1 = (p, q, s)$, but it is in conflict with triangle $t_2 = (q, p, t)$. The set of points in conflict with the new triangle spanned by p_i and e is a subset of Conflicts(t_1) ∪ Conflicts(t_2).*

Proof. Let ℓ be the line that contains p and q and consider the two closed halfspaces that are bounded by this line. Let h_s be the halfspace of the two that contains s and let h_t be the halfspace that contains t. Let $D(a,b,c)$ denote the disk that has a,b and c on its boundary. We claim that

$$D(q,p,p_i) \subseteq D(p,q,s) \cup D(q,p,t)$$

Indeed, this follows from the fact that

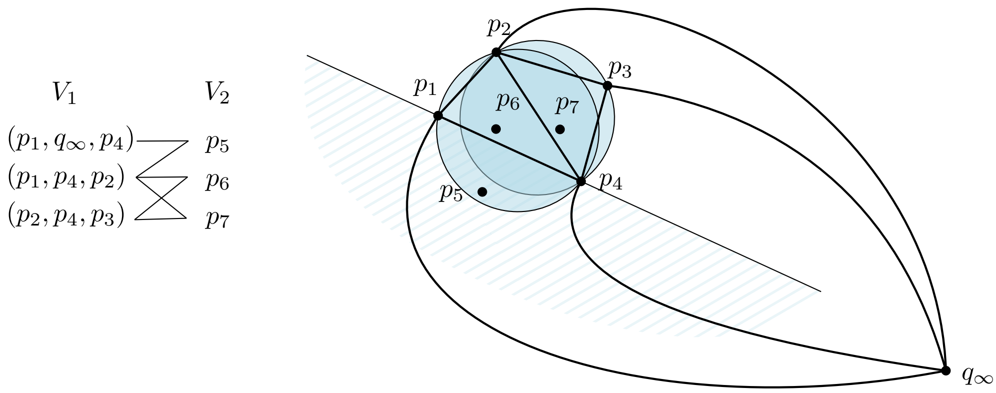
$$D(q,p,p_i) \cap h_s \subseteq D(p,q,s) \tag{1}$$

and that

$$D(q,p,p_i) \cap h_t \subseteq D(q,p,t) \tag{2}$$

and that $h_s \cup h_t = \mathbb{R}^2$. Note that all three disks contain p and q on their boundary. Now, Equation 1 follows from the fact p_i is not contained in $D(p,q,s)$and Equation 2 is true since p_i is contained in $D(q,p,t)$. □

There is one caveat with this algorithm: It is not obvious that a suitable bounding triangle can be computed easily. Recall that we required that the vertices of the bounding triangle do not interfer with the Delaunay property of the triangles of T, but we do not know these triangles and their circumcircles in advance, and it would be too time-consuming to test all triangles of P. Instead of computing the vertices of the bounding triangle explicitely, we want to maintain the triangle symbolically. We want the vertices to be very far away from the point set P, as if all three of them lie at infinity. We have to specify how conflict lists are being computed in this case. Let e be a halfedge incident to a triangle that has one vertex on the bounding triangle. We say a point $p \in \mathbb{R}^2$ is in conflict with this triangle if p is contained in the halfspace bounded by the line that contains the edge e and which does not contain any of the points of P in its interior (Note the similarity to the conflict graph for computing the convex hull in $\mathbb{R}^2$). There are three triangles that have two vertices on the bounding triangle. Those triangles disappear when the three vertices of the bounding triangle approach infinity. Therefore, we remove them from T, and we "merge" the vertices q_1, q_2 and q_3 to one vertex q_∞ which is placed "at infinity". The figure below shows a drawing of the modified graph of the triangulation and the conflict graph.



Observation 9.4. *For $i \geq 3$, the modified graph of the triangulation stored in the DCEL is a vertex 3-connected planar graph with $i+1$ vertices (by including the vertex at infinity and its incident edges). By Theorem 7.1 (Steinitz), the graph is therefore isomorphic to the graph of a 3-dimensional convex polytope. Therefore, by Theorem 7.2, it follows that number of edges e is at most $3(i-1)$ and the number of faces f is at most $2(i-1)$.*

Algorithm 9.1

1: **procedure** DELAUNAY-TRIANGULATION(Set of points P in $\mathbb{R}^2$)
2: Let $p_1, \ldots, p_n$ be a random permutation of the points of P
3: Initialize the triangulation T as a DCEL with the triangle spanned by $\{p_1, p_2, p_3\}$
4: Add an additional vertex q_∞ “at infinity” to T and add edges (q_∞, p_1), (q_∞, p_2), (q_∞, p_3)
5: *// Initialise the conflict graph:*
6: **for** each face f of T and each $j \in \{4, ..., n\}$ such that p_j conflicts with f **do**
7: Add f and p_j to each other's conflict lists
8: **end for**
9: *// Incremental construction:*
10: **for** $i \leftarrow 4$ **to** n **do**
11: **if** $\mathit{Conflicts}(p_i) \neq \emptyset$ **then**
12: $H \leftarrow$ cycle of half-edges clockwise along boundary of union of $\mathit{Conflicts}(p_i)$
13: **for** $e \in H$ **do**
14: *// Create new triangular face:*
15: Create face f with boundary cycle of new half-edges $\mathit{Start}(e) \to p_i \to \mathit{End}(e)$
16: $\to \mathit{Start}(e)$, setting all attributes except the twin pointers;
17: *// Collect points that f may conflict with from the* old *faces incident on e:*
18: $T \leftarrow \mathit{Conflicts}(IncidentFacet(e)) \cup \mathit{Conflicts}(IncidentFacet(Twin(e)))$
19: *// Connect up f in the conflict graph G:*
20: **for** $p \in T$ **do**
21: **if** f is in conflict with p **then**
22: add f and p to each other's conflict lists
23: **end if**
24: **end for**
25: *// Insert f in the DCEL of T:*
26: Set twin pointers between e and its newly created twin
27: *// (the twin pointers of the other new edges will be set on Line 31)*
28: *// Make sure $\mathit{IncidentEdge}(\mathit{Start}(e))$ is an edge that is not about to be deleted:*
29: Let $\mathit{IncidentEdge}(\mathit{Start}(e))$ point to e
30: **end for**
31: Go around H again and set twin pointers between half-edges to/from p_i
32: *// Discard old facets:*
33: **for** $f \in \mathit{Conflicts}(p_i)$ **do**
34: Remove f from the conflict lists of all points it is in conflict with
35: Remove f and all incident half-edges and now-isolated incident vertices
36: **end for**
37: **end if**
38: **end for**
39: Remove the vertex q_∞ and its incident edges from T
40: **return** T
41: **end procedure**

Theorem 9.5. *Algorithm 9.1 computes the Delaunay triangulation of P.*

Proof. We show the following invariant with three parts at the end of each iteration of the main for-loop (where we consider Lines 3–8 to be iteration $i = 3$):

(i) T stores a Delaunay triangulation of the points $p_1, \ldots, p_i$ (together with the vertex q_∞ and its incident edges and incident triangles).

(ii) The conflict graph G stores the conflicts between all triangles in T (including the infinite triangles) and the points $p_{i+1}, \ldots, p_n$.

(iii) A vertex p of T has an edge with the vertex q_∞ if and only if p lies on the boundary of the convex hull of $\{p_1, \ldots, p_i\}$.

In the beginning, T is initialized with the triangle spanned by three points p_1, p_2 and p_3 and the vertex q_∞. If we remove q_∞, then we obtain the Delaunay triangulation of $\{p_1, p_2, p_3\}$. All three parts of the invariant are satisfied by construction.

When adding a point p_i, we check its neighbours *Conflicts*(p_i) in the conflict graph. There are two cases:

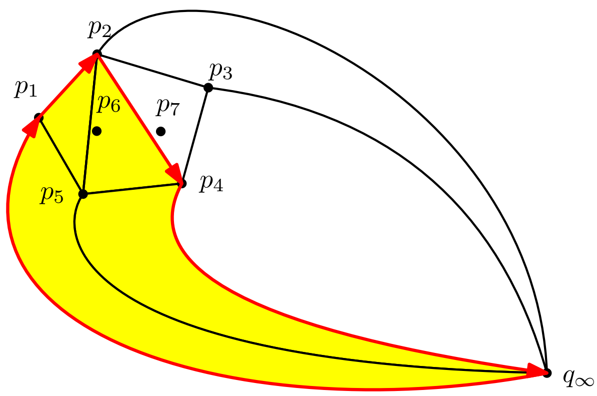
(a) either p_i lies in the convex hull of $p_1, \ldots, p_{i-1}$.

(b) or p_i lies outside the convex hull of $p_1, \ldots, p_{i-1}$.

In case (a), all infinite triangles connected to q_∞ must stay in T. Indeed, p_i cannot be in conflict with any of the infinite triangles, since in that case there would be a separating line between p_i and the convex hull of $p_1, \ldots, p_{i-1}$. The triangles of the Delaunay triangulation that p_i is in conflict with are removed and replaced by triangles incident to p_i. To see that this is the correct Delaunay triangulation, assume for the sake of contradiction that there would be a triangle in the Delaunay triangulation of $p_1, \ldots, p_i$ that is not incident to p_i and which is not present in the Delaunay triangulation of $p_1, \ldots, p_{i-1}$. This triangle has the Delaunay property with respect to $p_1, \ldots, p_{i-1}$ and therefore we get a contradiction with Lemma 8.6. Therefore, part (i) of the invariant holds at the end of iteration i. Part (ii) follows from Lemma 9.3 (one can check that the lemma still holds if we replace a disk with a halfspace in case a triangle is an infinite triangle). Part (iii) also holds, since we did not make any changes to the infinite triangles and the convex hull also remained the same.

In case (b), the point p_i must be in conflict with at least one of the infinite triangles. Therefore, it is contained in at least one of the halfspaces that have an edge of the convex hull on its boundary and no point of $p_1, \ldots, p_{i-1}$ in their interior. The corresponding edges of the convex hull form a convex chain, which will be removed from T by the algorithm, when removing the triangles that are in conflict with p_i. When adding the edges from p_i to T, the algorithm will add an edge between p_i and the infinite vertex q_∞. Since p_i lies on the boundary of the convex hull of $p_1, \ldots, p_i$, part (iii) of the invariant will be satisfied at the end of this iteration of the for-loop. The point p_i can also be in conflict with other triangles, which are not incident to q_∞. Those triangles will also be removed and replaced by triangles incident to p_i. Using the same argument as in case (a) one can check that parts (i) and (ii) of the invariant are also satisfied. □

The figure below shows an example of the new triangles created during the insertion of point p_5 (shaded yellow), and the cycle of horizon edges (in red). This example is an example of case (b) in the proof above.


p_1
p_2
p_3
p_4
p_5
p_6
p_7
q_∞