Chapter 6

Delaunay Triangulation: Incremental Construction

We have learned about the Lawson flip algorithm that computes a Delaunay triangulation of a given n-point set $P \subseteq \mathbb{R}^2$ by performing $O(n^2)$ Lawson flips. We have also seen in an exercise that there are point sets where it may take up to $\Omega(n^2)$ flips. Moreover, it can be implemented to run in $O(n^2)$ time.

Here, we will discuss a different algorithm. The final goal is to show that this algorithm can be implemented to run in $O(n \log n)$ time. Throughout, we assume that P is in general position (no 3 points on a line, no 4 points on a common circle), so that the Delaunay triangulation is unique (Corollary 5.19). There are techniques to deal with non-general position, but we don't discuss them here.

6.1 Incremental construction

The idea is to build the Delaunay triangulation of P by inserting one point after another according to an order $p_1, p_2, \ldots, p_n$ chosen uniformly at random. Note that this random order will only become relevant later in the runtime analysis.

To avoid special cases, we enhance the set P with three artificial points p_0, p_{-1} and p_{-2} "far out" such that the boundary of the convex hull of $P \cup \{p_0, p_{-1}, p_{-2}\}$ has only these three artificial points as vertices. The idea is to later remove the extra points and their incident edges to obtain $\mathcal{DT}(P)$.

The algorithm starts off with the Delaunay triangulation of the three artificial points, which consists of one big triangle enclosing all other points. In our figures, we suppress the far-away points, since they are merely a technicality. For $1 \leqslant s \leqslant n$, let $P_s = \{p_1, \ldots, p_s\}$ and $P^*_s = P_s \cup \{p_0, p_{-1}, p_{-2}\}$. Throughout, we maintain the Delaunay triangulation of the set P^*_{s-1} of points inserted so far, and when the next point p_s comes along, we update the triangulation to the Delaunay triangulation of P^*_s.

Let $\mathcal{DT}(s)$ denote the Delaunay triangulation of P^*_s. Now assume that we have already built $\mathcal{DT}(s-1)$, and we next insert p_s. Here is the outline of the update step.

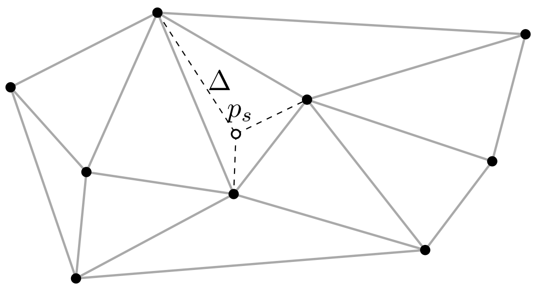


Figure 6.1: *Inserting* p_s *into* $\mathcal{DT}(s-1)$*: Step 1*

1. Find the triangle $\Delta = \Delta(p, q, r)$ of $\mathcal{DT}(s-1)$ that contains p_s, and replace it with the three triangles resulting from connecting p_s with all three vertices p, q, r; see Figure 6.1. We now have a triangulation $\mathcal{T}$ of P_s^*.

2. Perform Lawson flips on $\mathcal{T}$ until $\mathcal{DT}(s)$ is obtained; see Figure 6.2

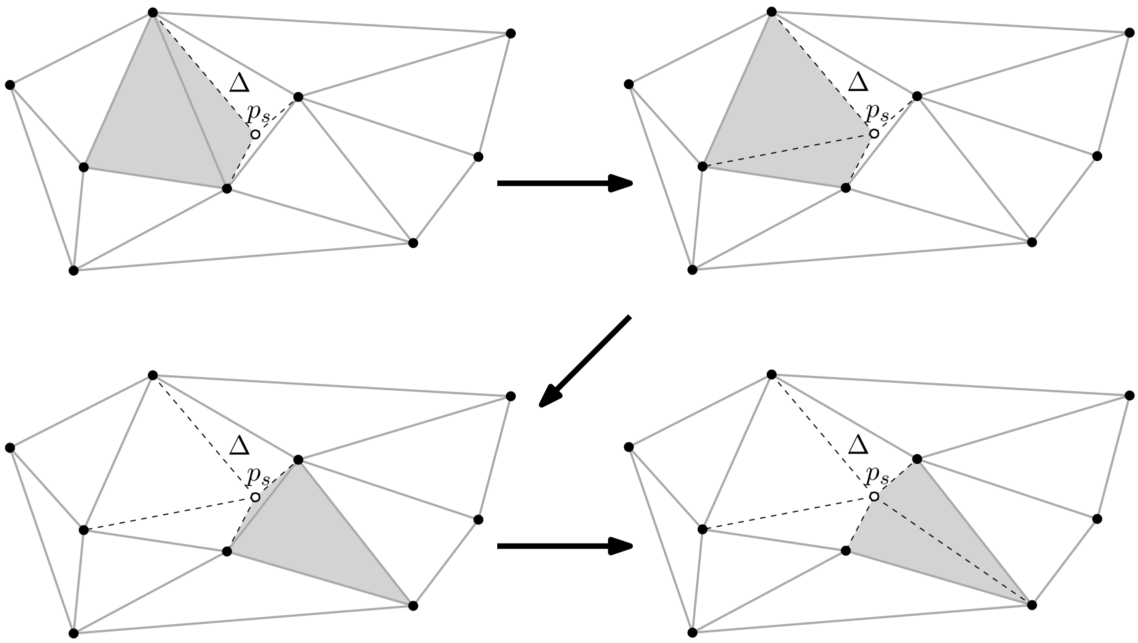


Figure 6.2: *Inserting* p_s *into* $\mathcal{DT}(s-1)$*: Step 2*

How to organize the Lawson flips. The Lawson flips can be organized quite systematically, since we always know the candidates for "bad" edges that may still have to be flipped. Initially (after step 1), only the three edges of Δ can be bad, since these are the only edges for which an incident triangle has changed (by inserting p_s in Step 1). Each of the three new edges is good, since the 4 vertices of its two incident triangles are not in convex position.

Now we have the following invariant (part (a) certainly holds in the first flip):

(a) In every flip, the convex quadrilateral Q in which the flip happens has exactly two edges incident to p_s, and the flip generates a new edge incident to p_s.

(b) Only the two edges of Q that are *not* incident to p_s can become bad after the flip.

We will prove part (b) in the next lemma. The invariant then follows since (b) entails (a) in the next flip. This means that we can maintain a queue of potentially bad edges that we process in turn. A good edge will be removed from the queue, and a bad edge will be flipped and replaced according to (b) with two new edges in the queue. In this way, we never flip edges incident to p_s; the next lemma proves that this is correct and at the same time establishes part (b) of the invariant.

Lemma 6.1. *Every edge incident to p_s that is created during the update is an edge of the Delaunay graph of P^*_s and thus an edge that will be in $\mathcal{DT}(s)$. It easily follows that edges incident to p_s will never become bad during the update step.*[1]

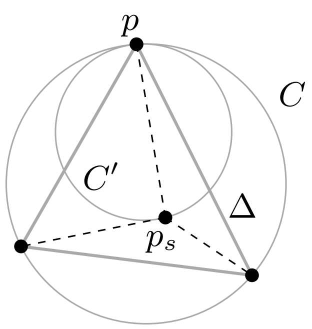
Proof. Let us consider one of the first three new edges, $\overline{p_s p}$, say. Since the triangle Δ has a circumcircle C strictly containing only p_s (Δ is in $\mathcal{DT}(s-1)$), we can shrink that circumcircle to a circle C' through p_s and p with no interior points, see Figure 6.3 (a). This proves that $\overline{p_s p}$ is in the Delaunay graph. If $\overline{p_s t}$ is an edge created by a flip, a similar argument works. The flip destroys exactly one triangle Δ of $\mathcal{DT}(s-1)$. Its circumcircle C contains p_s only, and shrinking it yields an empty circle C' through p_s and t. Thus, $\overline{p_s t}$ is in the Delaunay graph also in this case. □

6.2 The History Graph

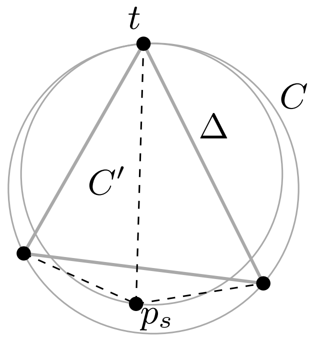
What can we say about the performance of the incremental construction? Not much yet. First of all, we did not specify how we find the triangle Δ of $\mathcal{DT}(s-1)$ that contains the point p_s to be inserted. Doing this in the obvious way (checking all triangles) is not good, since already the find steps would then amount to $\Theta(n^2)$ work throughout the whole algorithm. Here is a smarter method, based on the *history graph*.

Definition 6.2. *For any given s, the* history graph $\mathcal{H}_{s-1}$ *of P^*_{s-1} is a directed acyclic graph whose vertices are all triangles that have ever been created during the incremental construction of $\mathcal{DT}(s-1)$. There is a directed edge from triangle Δ to Δ' whenever Δ has been destroyed and Δ' has been created in the same step, which implies that Δ overlaps with Δ' in its interior.*

[1]If such an edge was bad, it could be flipped, but then it would be "gone forever" according to the lifting map interpretation from the previous chapter, which means it could not have been part of the Delaunay graph.



(a) New edge $\overline{p_s p}$ incident to p_s created in Step 1



(b) New edge $\overline{p_s t}$ incident to p_s created in Step 2

Figure 6.3: *Newly created edges incident to* p_s *are in the Delaunay graph*

It follows that the history graph $\mathcal{H}_{s-1}$ contains triangles of outdegrees 3, 2 and 0. The ones of outdegree 0 are clearly the triangles of $\mathcal{DT}(s-1)$.

The triangles of outdegree 3 are the ones that have been destroyed during Step 1 of an insertion. For each such triangle Δ, its three outneighbors are the three new triangles that have replaced Δ, see Figure 6.4.

The triangles of outdegree 2 are the ones that have been destroyed during Step 2 of an insertion. For each such triangle Δ, its two outneighbors are the two new triangles created during the flip that has destroyed Δ, see Figure 6.5.

The history graph $\mathcal{H}_{s-1}$ can be built during the incremental construction at asymptotically no extra cost; but it may need extra space since it keeps all triangles ever created. Given $\mathcal{H}_{s-1}$, we can search for the triangle Δ of $\mathcal{DT}(s-1)$ that contains p_s, as follows. We start from the big triangle $\Delta(p_0, p_{-1}, p_{-2})$; this one certainly contains p_s. Then we follow a directed path in the history graph. If the current triangle still has outneighbors, we find the unique outneighbor containing p_s (recall that we assume general position) and continue the search there. If the current triangle has no outneighbors, it is in $\mathcal{DT}(s-1)$ and contains p_s. Thus, the time complexity of finding the triangle containing p_s is linear in the length of the path traversed in the history graph.

Types of triangles in the history graph. After each insertion of a point p_s, several triangles are created and added to the history graph. It is important to note that these triangles come in two types: Some of them are valid Delaunay triangles of $\mathcal{DT}(s)$, and they survive to the next stage of the incremental construction. Other triangles are immediately destroyed by subsequent Lawson flips, because they are not Delaunay triangles of $\mathcal{DT}(s)$.

Note that, whenever a Lawson flip is performed, one of the two triangles destroyed is always a "valid" triangle from a previous iteration, and the other one is an "ephemeral" triangle that was created in this iteration. The ephemeral triangle is always the one that has p_s, the newly inserted point, as a vertex.

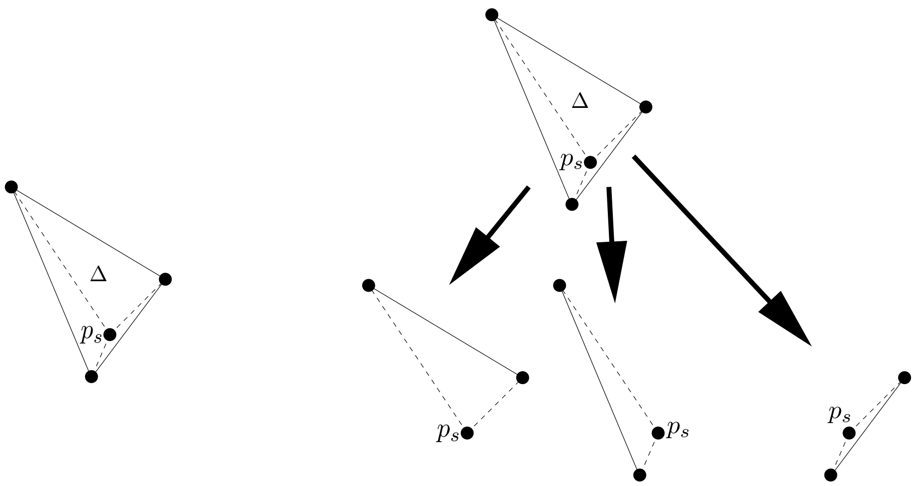


Figure 6.4: *The history graph: one triangle gets replaced by three triangles*

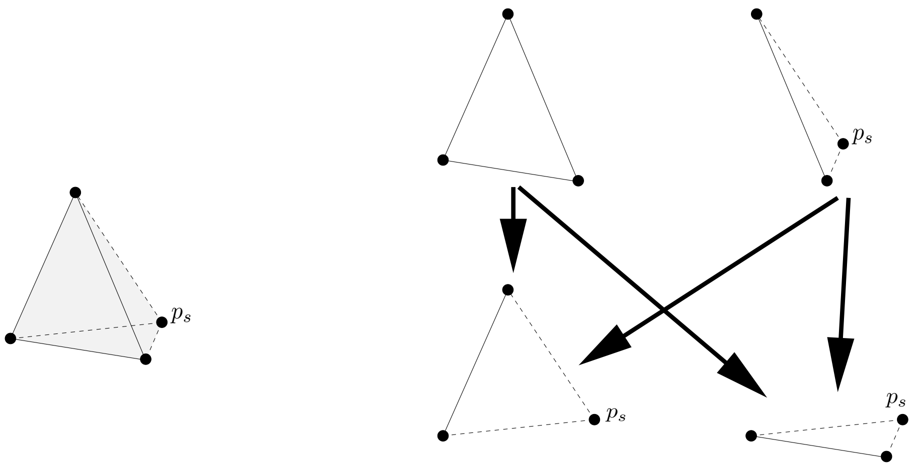


Figure 6.5: *The history graph: two triangles get replaced by two triangles*

6.3 Analysis of the algorithm

We start by making the following simple observation.

Observation 6.3. *Let* d_s *denote the degree of the vertex* p_s *in the triangulation* $\mathcal{DT}(s)$*. Given* $\mathcal{DT}(s-1)$ *and the triangle* Δ *of* $\mathcal{DT}(s-1)$ *that contains* p_s*, we can build* $\mathcal{DT}(s)$ *in time proportional to* d_s*. Moreover, the total number of new triangles (both "valid" and "ephemeral") created throughout this insertion is* $2d_s - 3$*.*

Indeed, since every Lawson flip increases the number of edges adjacent to p_s by exactly one, the number of flips is equal to the final degree d_s of p_s minus three. Step 1 of the update takes constant time and creates three new triangles; each flip in Step 2 can be implemented in constant time as well and creates two new triangles. Therefore, the running time is indeed proportional to d_s, and the number of created triangles is $3 + 2(d_s - 3) = 2d_s - 3$, as claimed.

Lemma 6.4. *Let* d_s *again denote the degree of the vertex* p_s *in the triangulation* $\mathcal{DT}(s)$*. Then, we have* $E[d_s] \leqslant 6$*.*

Proof. We use *backwards analysis* to bound the expected value of the random variable d_s. Since $\mathcal{DT}(s)$ is a triangulation with $s+3$ vertices, it follows from the Euler characteristic that it has $3(s+3)-6$ edges. If we exclude the three edges of the convex hull, we get that the degrees of all interior vertices in $\mathcal{DT}(s)$ add up to at most $2(3(s+3)-9) = 6s$. This implies that the expected degree of a random point of P_s (i.e., not including the artificial points p_0, p_{-1} or p_{-2}) in $\mathcal{DT}(s)$ is at most 6. □

By combining the above observations, we can also prove the following bound on the expected number of triangles created by the algorithm. Note that this is at the same time a bound on the expected size of the history graph.

Lemma 6.5. *The expected number of triangles (both "valid" and "ephemeral") ever created by the algorithm is at most* $9n + 1$*.*

Proof. Before inserting any points from the set P, we only have the artificial triangle $\Delta(p_0 p_{-1} p_{-2})$. During the s-th iteration of the algorithm, when we insert the point p_s, we know from Observation 6.3 that the number of new triangles created is $2d_s - 3$. Combined with Lemma 6.4 we then get that the expected number of created triangles in iteration s is at most

$$E[2d_s - 3] = 2E[d_s] - 3 \leqslant 2 \cdot 6 - 3 = 9.$$

By linearity of expectation we thus get that the expected total number of created triangles is at most $9n + 1$. □

Note that we cannot say that all iterations create a number of triangles close to 9, i.e., there could be some very costly insertions. However, the average is constant which provides us with a linear expected total value.

Locating p_s in the history graph. We proceed now to the most difficult part of the analysis; that is, showing that locating the triangle that contains p_s in $\mathcal{H}_{s-1}$ costs logarithmic time in expectation. Note that the time required for locating p_s is proportional to the number of triangles in $\mathcal{H}_{s-1}$ that contain p_s (because every traversed triangle is, by definition, one that contains p_s). This set of traversed triangles also consists of "ephemeral" triangles that have been created and immediately destroyed in the same iteration. To make the analysis possible, we instead want to express the required running time for locating p_s in terms only of "valid" triangles.

Indeed, we can say that locating p_s costs time at most proportional to the number of valid triangles ever created that contain p_s in their circumcircle. This reformulation indirectly accounts also for ephemeral triangles that contain p_s; whenever we traverse an ephemeral triangle Δ in $\mathcal{H}_{s-1}$ while locating p_s, we can charge the time spent to the valid triangle Δ' that was destroyed together with Δ during the corresponding Lawson flip. It is clear that, in this way, the triangle Δ' is charged at most once. Provided that the corresponding Lawson flip was performed, we also know that p_s is indeed contained in the circumcircle of Δ'.

Instead of analyzing the running time for locating one point p_s in a particular iteration s, the goal is to understand the combined running time for locating all points $p_1, p_2, \ldots, p_n$ over all iterations. We can now express this combined running time as

$$O\left(n + \sum_{\Delta} |K(\Delta)|\right),$$

where the sum goes over all valid Delaunay triangles ever created in any iteration $s = 1, \ldots, n$ (i.e., we deliberately exclude the initial artificial triangle in the sum), and the set $K(\Delta)$ consists of all points from P that are contained in the circumcircle of Δ.

Note that in the case of $\mathcal{DT}(0)$, we have $|K(\Delta)| = n$ for the artificial triangle $\Delta = \Delta(p_0 p_{-1} p_{-2})$; whereas we know that $|K(\Delta)| = 0$ for all triangles Δ in the final Delaunay triangulation $\mathcal{DT}(n)$. In between, we would like the values to somehow interpolate nicely.

Lemma 6.6. *It holds that*

$$E\left[\sum_{\Delta} |K(\Delta)|\right] = O(n \log n).$$

Proof. Throughout, we will make use of the following four random sets and functions:

- $\tau_s = \{\Delta \in \mathcal{DT}(s)\}$, the set of Delaunay triangles in $\mathcal{DT}(s)$;
- $\tau^*_s = \tau_s \setminus \tau_{s-1}$, the set of newly created Delaunay triangles in $\mathcal{DT}(s)$;
- $\varphi_s(q) = |\{\Delta \in \tau_s : q \in K(\Delta)\}|$, the number of triangles in $\mathcal{DT}(s)$ whose circumcircle contains a point q;
- $\varphi^*_s(q) = |\{\Delta \in \tau^*_s : q \in K(\Delta)\}|$, the number of newly created triangles in $\mathcal{DT}(s)$ whose circumcircle contains a point q.

Using this new notation, we can rewrite the expression from the lemma as follows:

$$\mathrm{E}\left[\sum_{\Delta}|K(\Delta)|\right] = \mathrm{E}\left[\sum_{s=1}^{n}\sum_{\Delta\in\tau_s^*}|K(\Delta)|\right] = \sum_{s=1}^{n}\mathrm{E}\left[\sum_{\Delta\in\tau_s^*}|K(\Delta)|\right]. \tag{6.7}$$

This works because each triangle created by the algorithm is created in some iteration and, hence, belongs to the set τ_s^* for some s.

We recall that for every s, the sum $\sum_{\Delta\in\tau_s^*}|K(\Delta)|$ counts (with multiplicity) the number of points in P that lie inside the circumcircles of the triangles in τ_s^*. Since these circumcircles belong to triangles of the Delaunay triangulation $\mathcal{DT}(s)$, they must not contain any points from the set P_s and, hence, all points lying inside these circumcircles belong to $P \setminus P_s$. Thus, we can further rewrite the sum in terms of the function φ_s^* as

$$\mathrm{E}\left[\sum_{\Delta}|K(\Delta)|\right] = \sum_{s=1}^{n}\mathrm{E}\left[\sum_{\Delta\in\tau_s^*}|K(\Delta)|\right] = \sum_{s=1}^{n}\mathrm{E}\left[\sum_{q\in P\setminus P_s}\varphi_s^*(q)\right]. \tag{6.8}$$

To further analyze the expected value in the above expression, we use conditional expectations. That is, we condition on P_s (i.e., the random set of points inserted in the first s iterations) being a specific set of s points from P and then, later, we take the weighted average of all such conditional expectations.

More formally, let us fix any concrete set $\hat{P}_s = \{\hat{p}_1, \ldots, \hat{p}_s\}$ that is a subset of P of size s, and define $\mathcal{E}$ to be the event that $P_s = \hat{P}_s$. Under the condition $\mathcal{E}$, the set of points $P_s = \hat{P}_s$ and thus also the set of triangles $\tau_s = \hat{\tau}_s$ and the function $\varphi_s = \hat{\varphi}_s$ are fixed; therefore, the random variable $\varphi_s^*(q)$ depends only on which of the points in $\hat{P}_s$ was inserted last. Since the order of insertion of $\hat{P}_s$ is still uniformly at random, a triangle $\Delta \in \hat{\tau}_s$ is incident with the random point p_s (and hence Δ is newly created in iteration s) with probability at most $3/s$ (only "at most" because the three incident vertices of Δ might be equal to the artificial points p_0, p_{-1}, p_{-2}). We therefore get

$$\begin{aligned}\mathrm{E}\left[\sum_{q\in P\setminus P_s}\varphi_s^*(q)\,\middle|\,\mathcal{E}\right] &= \sum_{q\in P\setminus\hat{P}_s}\mathrm{E}\left[\varphi_s^*(q)\mid\mathcal{E}\right] = \sum_{q\in P\setminus\hat{P}_s}\sum_{\substack{\Delta\in\hat{\tau}_s:\\ q\in K(\Delta)}}\underbrace{\Pr\left[\Delta\in\tau_s^*\mid\mathcal{E}\right]}_{\leqslant 3/s}\\ &\leqslant \frac{3}{s}\sum_{q\in P\setminus\hat{P}_s}\hat{\varphi}_s(q).\end{aligned} \tag{6.9}$$

Still conditioned on $\mathcal{E}$, we note that any element q of $P \setminus \hat{P}_s$ is equally likely to be the point p_{s+1}; that is, the point inserted in the next iteration. We therefore get that

$$\mathrm{E}[\varphi_s(p_{s+1}) \mid \mathcal{E}] = \frac{1}{n-s}\sum_{q\in P\setminus\hat{P}_s}\hat{\varphi}_s(q). \tag{6.10}$$

If we now combine equations (6.9) and (6.10), and then drop the condition $\mathcal{E}$ (which is justified by the law of total expectation), we obtain

$$E\left[\sum_{q \in P \setminus P_s} \varphi_s^*(q)\right] \leqslant \frac{3(n-s)}{s} E[\varphi_s(p_{s+1})]. \tag{6.11}$$

Euler's characteristic implies that the number of triangles increases by exactly two whenever a new point is inserted into a triangulation. Therefore, we always have $\varphi_s(p_{s+1}) + 2 = |\tau_{s+1}^*|$, which is simply saying that in iteration $s+1$, the number of destroyed Delaunay triangles plus two is equal to the number of newly created Delaunay triangles. Moreover, as already noticed earlier in this chapter, we have in fact $|\tau_{s+1}^*| = d_{s+1}$; that is, the number of new Delaunay triangles is equal to the degree d_{s+1} of the newly inserted point p_{s+1}. We may therefore rewrite inequality (6.11) as

$$E\left[\sum_{q \in P \setminus P_s} \varphi_s^*(q)\right] \leqslant \frac{3(n-s)}{s}(E[|\tau_{s+1}^*|]-2) = \frac{3(n-s)}{s}(\underbrace{E[d_{s+1}]}_{\leqslant 6}-2) \leqslant \frac{12(n-s)}{s}, \tag{6.12}$$

where we have used Lemma 6.4 in the last step.

We are finally able to plug (6.12) back into (6.8) in order to conclude the proof:

$$E\left[\sum_{\Delta} |K(\Delta)|\right] \leqslant \sum_{s=1}^{n} \frac{12(n-s)}{s} \leqslant 12n \sum_{s=1}^{n} \frac{1}{s} = O(n \log n). \qquad \square$$

The main theorem. Having the previous lemmas at hand, proving our main result is now straightforward.

Theorem 6.13. *The Delaunay triangulation of a set* P *of* n *points in the plane can be computed in* $O(n \log n)$ *expected time, using* $O(n)$ *expected space.*

Proof. The correctness of the algorithm follows from the correctness of the Lawson flip algorithm, and from the fact that we perform all possible Lawson flips in every iteration. For the space consumption, we note that only the history graph could use more than linear space, but Lemma 6.5 proves that its expected size is $O(n)$, which yields the desired bound.

To bound the running time of the algorithm, we first ignore the time used during the point location queries. Ignoring this, from Observation 6.3 we know that the running time is proportional to the number of triangles created. From Lemma 6.5 we again know that only $O(n)$ triangles are created in expectation. Hence, only $O(n)$ additional time is needed in expectation.

It remains to account for the point location queries. Recall that we do this by using the history graph. We start from its root, the triangle $\Delta(p_0, p_{-1}, p_{-2})$, and then traverse a path in this graph that finishes in a node corresponding to the triangle of $\mathcal{DT}(s-1)$ that contains p_s. Since the out-degree of all nodes in the history graph is $O(1)$, the running time of the point location query is proportional to the number of nodes visited. As explained earlier, the number of nodes visited in this way over the whole run of the algorithm is bounded by the expression $O(n + \sum_\Delta |K(\Delta)|)$. From Lemma 6.6 we get the required upper bound of $O(n \log n)$ on the expected value of that expression. □

Exercise 6.14. *For a sequence of* n *pairwise distinct numbers* $y_1, \ldots, y_n$ *consider the sequence of pairs* $(\min\{y_1, \ldots, y_i\}, \max\{y_1, \ldots, y_i\})_{i=0,1,\ldots,n}$ *(*$\min \emptyset := +\infty, \max \emptyset := -\infty$*). How often do these pairs change in expectation if the sequence is permuted randomly, each permutation appearing with the same probability? Determine the expected value.*

Exercise 6.15. *Given a set* P *of* n *points in convex position represented by the clockwise sequence of the vertices of its convex hull, provide an algorithm to compute its Delaunay triangulation in* $O(n)$ *time.*

Questions

28. *How can we efficiently compute the three artificial points* p_0, p_{-1} *and* p_{-2} *whose convex hull contains all points of* P*, while keeping their coordinates "small".*
29. *Describe the algorithm for the incremental construction of* $\mathcal{DT}(P)$: how do we find the triangle containing the point p_s to be inserted into $\mathcal{DT}(s-1)$? How do we transform $\mathcal{DT}(s-1)$ into $\mathcal{DT}(s)$? How many steps does the latter transformation take?
30. *What are the two types of triangles that the history graph contains?*