Delaunay triangulation

Maja Dall'Acqua

maja.dallacqua@studenti.unitn.it

Introduction

The Delaunay triangulation was first introduced by Boris Nukolaevich Delaunay in 1934 as a way to decompose a set of points into a triangular mesh with some specific geometric properties.
Indeed this algorithm is a geometric method used to create triangulations of a set of points in a plane, in other words it connects points with edges to form triangles such that no point lies inside the circumcircle of any triangle formed.
The Delaunay triangulation has found extensive use across scientific fields due to its geometric properties, which ensure well-conditioned triangles and optimal connectivity among points.

More specifically, the Delaunay algorithm can have application in many fields, as:

- computational geometry: it is used for various geometric algorithms and applications.
- mesh generation: it is used in finite element analysis and computer graphics to generate high-quality triangular meshes.
- GIS analysis: it is used in spatial analysis, terrain modeling and mapping applications.
- spatial analysis: together with the Voronoi Diagram, it is used also in the fields of facility location, clustering, and proximity analysis.

Definition

Given a set of points P, the Delaunay algorithm allows to connect all the points in a planar space forming a geometric structure composed of triangles. Before describing the algorithm itself it is important to define some concepts that will be useful in the following sections.

Angle vector: the angle vector of a triangulation T, namely $A(T)$, is an array composed of the angles of all the triangles created by the current triangulation, sorted by increasing order.

Angle-optimality: A triangulation T is called angle-optimal if the angle vector of T, corresponding to $A(T) = (\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{3m})$, is bigger than the angle vector of T', which is $A(T') = (\alpha'_1, \alpha'_2, \alpha_3', \dots, \alpha'_{3m})$, for all the triangulations T' of a set of points P.

Illegal edges: an edge is defined as illegal if we can locally increase the smallest angle by flipping that edge. This means that we compare the angle vectors derived from the two edges to observe whether the smallest angle increases or decreases after the flipping.
If the smallest angles of the two edges are equal, we compare the second smallest angles and so on until we can define a rank between the two triangulations.

Edge flipping: an edge e, which is one of the diagonals of a quadrilateral polygon, is replaced by the other diagonal e' of the same quadrilateral, dividing the polygon into two triangles different from the ones derived from diagonal e.

The edge flipping process is applied when there is an illegal edge in the triangulation.

We can observe these concepts in the current example (see *Figure 1*), where we consider the quadrilateral $abcd$ and its two diagonals, namely bd and ac.
Edge bd produces the angle vector $A(T) = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6)$, while edge ac, which is the flipped edge of bd, produces the angle vector $A(T') = (\alpha'_1, \alpha'_2, \alpha'_3, \alpha'_4, \alpha'_5, \alpha'_6)$.
When ranking the two triangulations, three cases may occur:

1. $\alpha_1 > \alpha'_1$: this means that the edge bd produces angles which are all bigger than the smallest angle of edge ac, so we can't locally increase the smallest angle. Therefore edge bd is already a legal edge and its derived triangulation is angle-optimal.

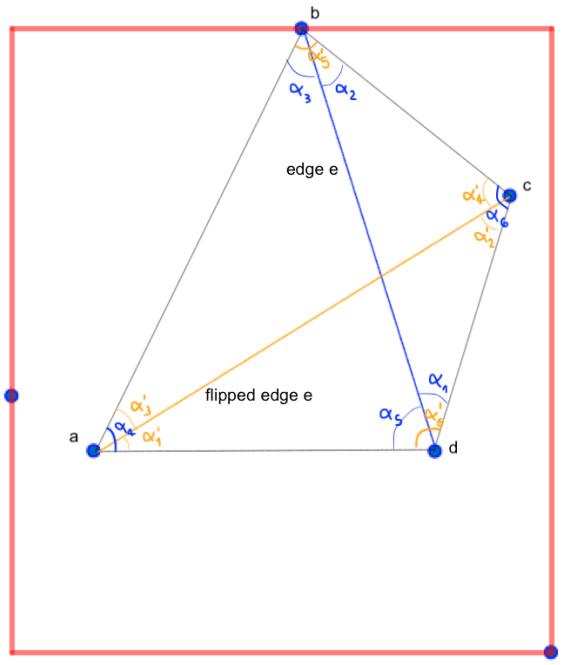
2. $\alpha_1 = \alpha'_1$: we move on to the second pair of angles. We try to compare α_2 with α'_2 and so on until we can rank the two triangulations.

3. $\alpha_1 < \alpha'_1$: this means that the smallest angle produced by edge bd is smaller than the smallest angle in the triangulation of ac, so we can locally increase the angles by flipping the edge. Therefore in this case edge bd is illegal and we can adjust it by flipping it to edge ac, whose triangulation is angle-optimal.

In this specific case we have that $\alpha_1 > \alpha'_1$, which implies also that $A(T) > A(T')$.

Then we can say that the triangulation producing triangles abd and bcd is angle-optimal and the segment bd is a legal edge.

Figure 1: triangulation of a quadrilateral abcd.



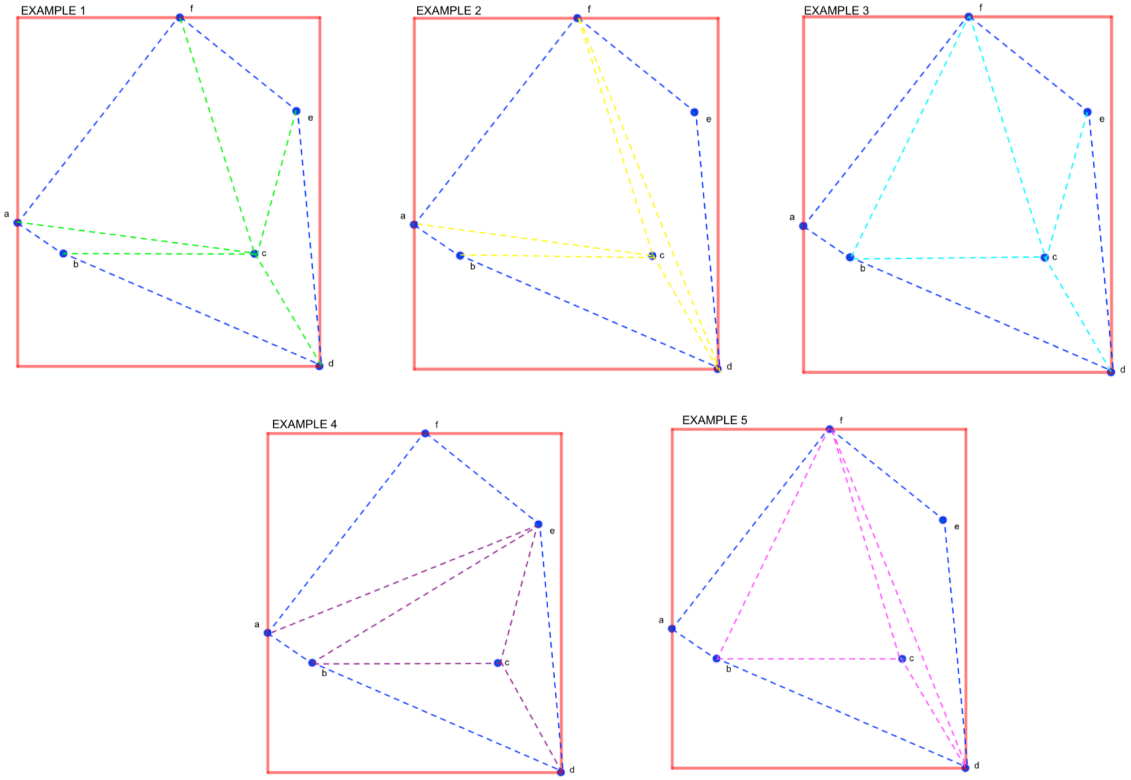
These concepts can now be applied to compute the Delaunay triangulation of a set of points P.

Delaunay algorithm's theorems

The Delaunay graph can be considered as an embedding of the dual graph of the Voronoi diagram. Indeed, we can construct the Delaunay diagram by connecting the nodes of the Voronoi cells with the adjacent ones using a straight-line segment and, at the same time, from the Delaunay graph we can build the Voronoi diagram.
This relation will be explained with more details at the end of the current section.

Let's now consider a random set of points $P = \{a, b, c, d, e, f\}$. From this set P there are many triangulations possible, but only one is also a Delaunay triangulation. To know which is the correct one, the algorithm applies Delaunay's theorems to all the possible triangulations and discards them based on the observance of such theorems.
Figure 2 shows some possible triangulations that can be applied to the set of points P.

Figure 2: some possible triangulations from the same set of points P



The main goal is then to understand which will be used by the algorithm to construct the Delaunay graph. We can answer this question by using the theorems of the Delaunay algorithm and displaying the circumcircles of the triangles involved, although we could already make a guess just by looking at *Figure 2*.
Indeed, we can easily see that most of the examples produce triangles with really small angles, which may probably be increased by flipping some of the edges. Then those won't probably be Delaunay triangulations. We can suppose that the correct Delaunay triangulation may be the one in *Example 3*, where the angles appear to be already big enough, but we need to prove it.
Therefore the current analysis is structured in the following way: first the theorems associated with the Delaunay triangulation will be discussed, then the lemmas derived from the algorithm implementation will be further analyzed. In both the sections some examples from the current set of points $P = \{a, b, c, d, e, f\}$ will be used.

Theorem 1

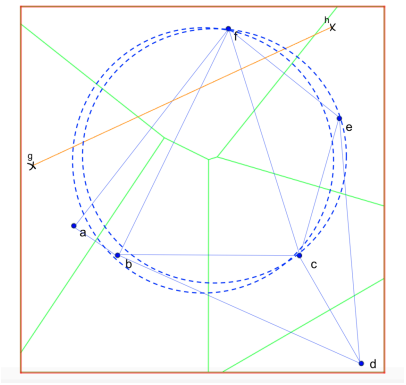
The first theorem states that the Delaunay graph is a planar graph, which means that there exists no edge crossing. We can prove this theorem by contradiction.

From the Voronoi diagram we know that the Delaunay graph has the following property:

> *an edge* $p_i p_j$ *is part of the Delaunay graph if and only if there is a closed disc* C_{ij} *with* p_i *and* p_j *on its boundary and no other site of P contained in it.*

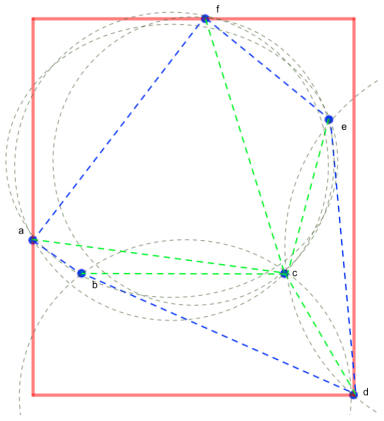
Suppose to have two different segments of the Delaunay diagram intersecting. Both p_l and p_k must lie outside the circle C_{ij} and outside the triangle formed by p_i, p_j and c_{ij}. This implies that $p_k p_l$ must intersect one of the edges of triangle t_{ij} incident to the center c_{ij}.
This intersection then contradicts the fact that the edges are contained in disjoint Voronoi cells. A visualization of this contradiction is displayed in *Figure 3*. There we see that the edge gh, which intersects edge fc, has both points g and h outside the triangles fce and fcb. But this implies that there are at least two Voronoi cells containing two points, which we know is not possible considering the properties of the Voronoi diagram.

Figure 3: existence of edge gh intersecting edge fc.



Considering *Example 1* (shown in *Figure 4*), we can see that the condition enounced by the first theorem is not respect in the triangles abc and fac, since the former has node e on the border of its circumcircle, while the latter contains node b in its circumcircle. Therefore we can already exclude *Example 1* and *Example 2* (since it also has triangle abc) as correct Delaunay triangulations.

Figure 4: circumcircles of Example 1 triangulation.



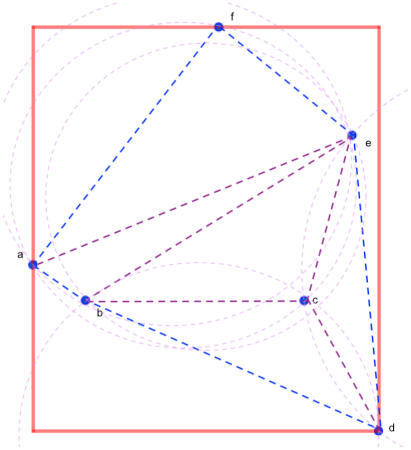
Theorem 2

> *A set of points P is in general position when there are at most three points on the boundary of a circle. In such cases all the vertices of the Voronoi diagram have degree three and therefore all the bounded faces of the Delaunay graph are triangles.*

When instead four points lie on the boundary of the circle, the polygon composed is not actually a triangle. In these cases we can still obtain a triangulation by adding edges to the Delaunay graph.

In *Example 4* (shown in *Figure 5*) we can see that on the boundary of the circumcircle of triangle afe lie four points, which are the vertices a, f, e and also the node c. This means that the polygon composed is not a triangle, but instead the polygon $acef$. Therefore, this cannot be a Delaunay triangulation, unless we decide to add one or more nodes to the graph. Moreover this example does not respect also Theorem 1 since the circumcircles of some of the triangles contain also other nodes.

Figure 5: circumcircles of Example 4 triangulation.

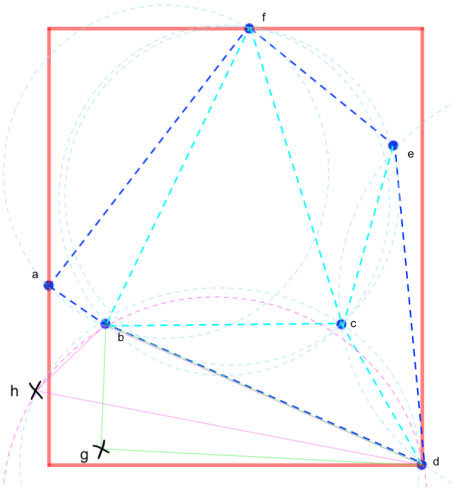


Theorem 3

> *Let P be a set of points in the plane. A triangulation T of P is legal if and only if T is a Delaunay triangulation.*

To prove this theorem we can show by contradiction that any legal triangulation is a Delaunay triangulation.
Assuming to have a triangulation T that is a legal triangulation but not a Delaunay triangulation. This would mean that in *Example 3* (displayed in *Figure 6*) there exists a triangle bcd such that the circumcircle C_{bcd} contains a point g that belongs to P in its interior. Let now e be the segment bd of triangle bdg that does not intersect triangle bcd. Of all the pairs gb, gd, gc in triangle t we need to select the one that maximizes the angle of bgd. To do so we actually need to choose a point h along the boundary of C_{bcd} and construct the triangle bdh adjacent to bcd along e. Since t is legal, also e is legal, therefore we can conclude that point h is not contained in the interior of C_{bcd}.
We can also see that the circumcircle $C(bdh)$ contains the part of the circumcircle $C(bcd)$ that is separated from bcd by e. Consequently the point g belong to the circumcircle $C(bdh)$. If we assume that the edge hd is the edge of bdh such that dhg that not intersect bhd we would actually see that the angle dgh is bigger than angle dgb by Thales' theorem, contradicting the definition of the pair (bcd, g).

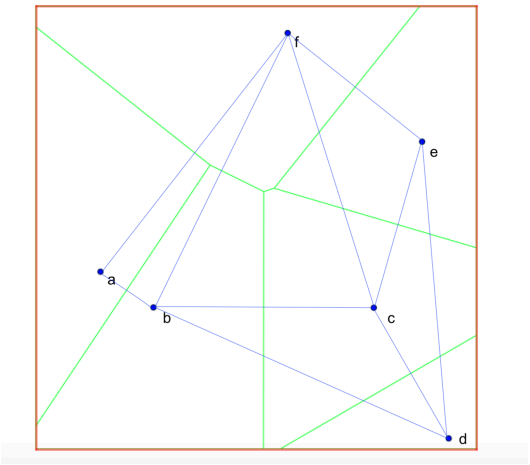
Figure 6: circumcircles of set P with new nodes g and h.



Through all these considerations we can now derive that any angle-optimal triangulation of P is actually a Delaunay triangulation. Furthermore we can also state that any Delaunay triangulation of P maximizes the minimum angle over all triangulations of P.

By analyzing the previous theorems we can conclude that the correct Delaunay triangulation of the set of points P is the one shown is *Example 3* of *Figure 2*. We confirm this by computing the Delaunay algorithm and the Voronoi diagram on GRASS, as displayed in *Figure 7*. In blue we see the edges of the Delaunay graph, which are the straight lines that connect the adjacent Voronoi cells, displayed in green.

Figure 7: Voronoi diagram and Delaunay graph of set P.



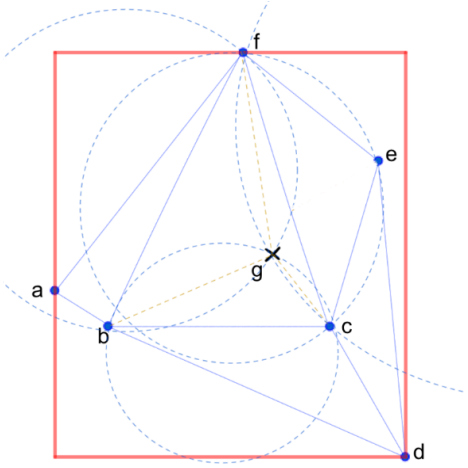
Delaunay algorithm's implementation

The Delaunay algorithm is implemented through a randomized incremental algorithm. When we want to add some artificial points to the set P and still maintain the Delaunay triangulation we need to consider some factors during the process.
Firstly, it is important to analyze the location of a new point g. Two main cases are possible to happen, such as:

- point g lies inside one of the Delaunay triangles, as shown in *Figure 8*. In this case we can draw the edges between point g and all the vertices of the triangle containing g, which leads in our example to the construction of the new triangles bgd, fgc, and bgc.
- point g falls on an edge of the triangulation, as shown in *Figure 9*, where g lies on the segment fc. In this case we need to add new edges from point g to the opposite nodes of the triangle sharing edge fc. This means that we construct four new triangles, namely bgf, bgc, cge and fge.

In both cases we should consider that by adding a new node g to set P some of the previous triangulation may now be illegal, so they might need to be replaced through some edge flips. The process to legalize an edge is called recursively since a new illegal triangulation may happen after each edge flipping. The algorithm then stops when all the illegal edges are replaced.
We can try to see whether new illegal triangulations are formed in the current example. Considering *Case 1* we notice that there are two illegal triangulations, triangles fgc and fgb, since they both contain other points of P in their circumcircles, respectively node e and node a, as demonstrated in *Figure 8*.

Figure 8: case 1: circumcircles of set $P \cup g$, where node g is added inside the Delaunay triangulation.



So now we should apply some edge flips in order to obtain all legal triangulations for the set $P \cup g$, both when g is inside one of the triangles and when g is along the edge of one of the triangles. Let's start from the former.

Considering triangle fgc we could try to flip edge fc to edge ge in order to obtain new triangles feg and gec. We can see that the angles produced by using edge ge are bigger than the ones in edge fc, implying that the angle-optimal triangulation is the one obtained with triangles feg and gec[1]. At the same time, we can try to legalize the edge fb by flipping it such that we obtain the triangles abg and afg.
As for the first case, we see that the smallest angle of fb, which is angle afg is smaller than the smallest angle of ag, which is agb, implying that the legal triangles are then abg and afg. These changes are shown in *Figure 9*.

[1] Please note that the angle vectors were not directly calculated, so the triangulations were compared using visual estimates. This approach was applied to the current example and the following ones.

Now that we have legalized these edges, we should check whether the other edges have remained legal or there have been any changes. If all the edges of the new set of points *P* are legal, we can move to the last step of the algorithm, which consists in checking whether all the triangles of *P* are Delaunay triangles or not. We can do so by searching for any points in each circumcircle.
If the condition is satisfied we can conclude that every edge in the new graph is legal, so the algorithm stops and we have obtained a new triangulation after the addition of point g.
If not, we need to legalize the illegal edges by flipping them. We stop when all the edges in the new graph are legal and all the triangles are Delaunay triangles.

Figure 9: case 1: edge flipping in set P U g.

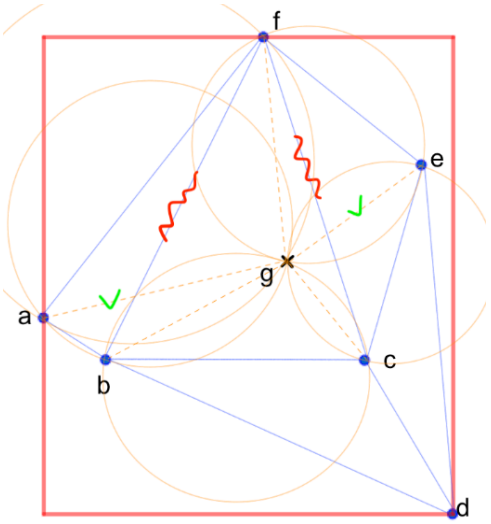
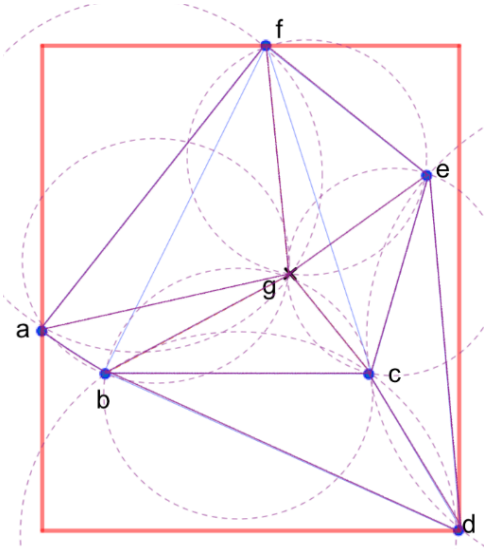


Figure 10: case 1: final Delaunay triangulation for set P U g.



Considering *Case 2* only the circumcircle of triangle fgb contains node a (see *Figure 11*), which means that the triangulation needs to be changed to a Delaunay one.
Moreover the only illegal edge is fb since node a is inside the circumcircle of triangle fbg. Considering the quadrilateral $afgb$ we can try to legalize edge fb by flipping it to edge ag. We notice that by doing it the smallest angle produced by edge fb, i.e. the angle afb, is smaller than the smallest angle of ag, which is agb. Therefore, we can conclude that to have a Delaunay triangulation we should consider the triangles afg and agb. Since all the other triangles are Delaunay triangles the algorithm stops and the graph shown in *Figure 13* is the final Delaunay triangulation for the set *P* and point g.

Figure 11: case 2: circumcircles of set P ∪ g, where node g is added along one edge of the Delaunay triangulation. .

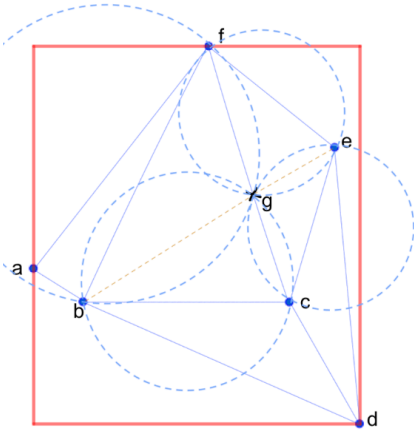


Figure 12: case 2: edge flipping in set P U g

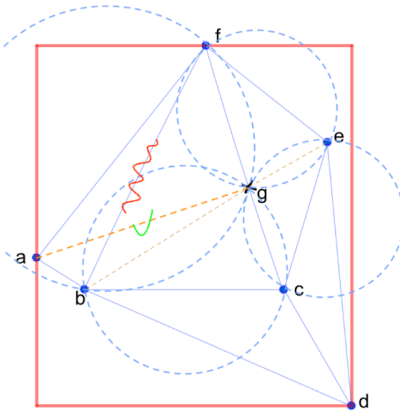
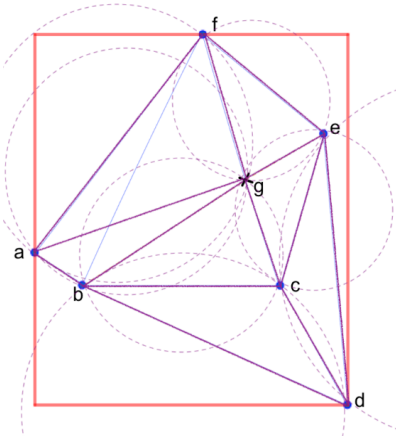


Figure 13: case 2: final Delaunay triangulation for set P U g.



Before exploring the Lemmas derived from the Delaunay triangulation we should consider one more element. Indeed when a new point is added to the graph we need to find the triangle containing that point. To simplify this search when we build the Delaunay graph we also build a point location structure D, which is a DAG (*Directed Acyclic Graph*) structure.

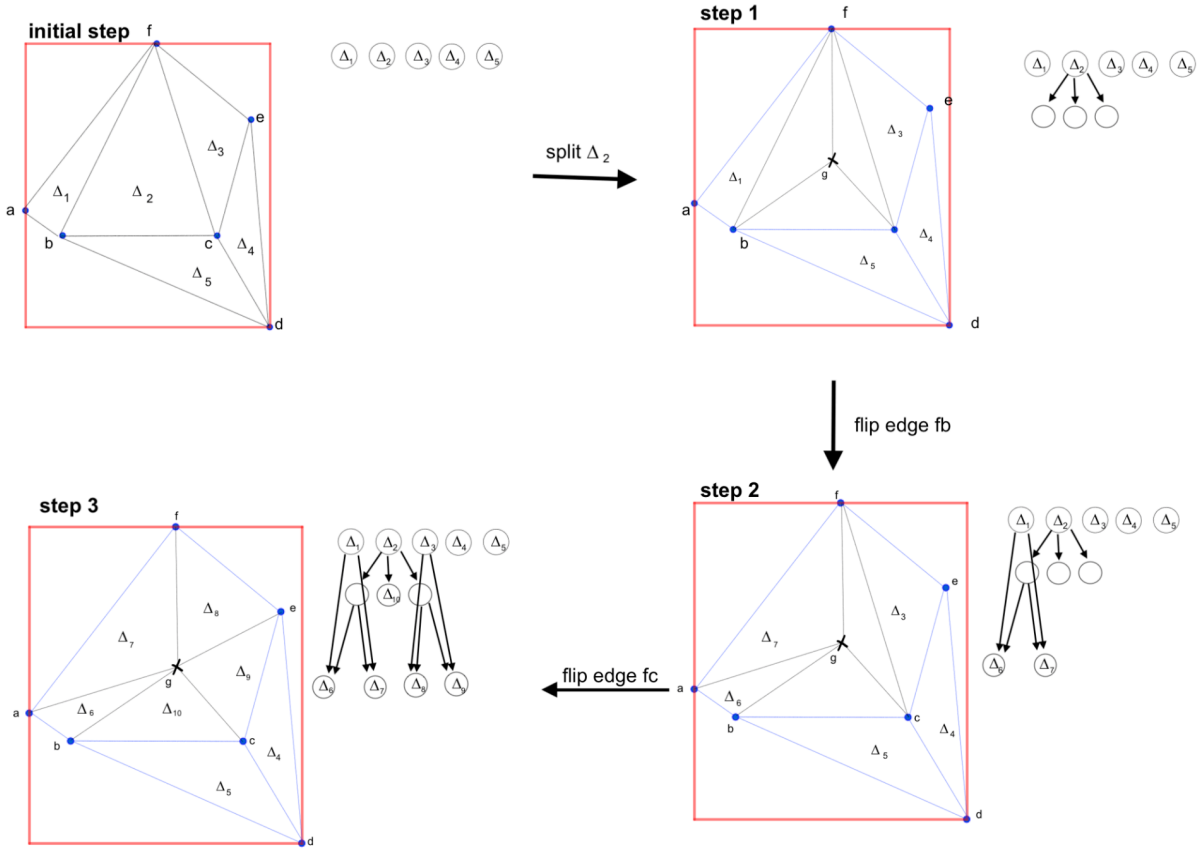
In D we have:

- the leaves as the triangles of triangulation T,
- the internal nodes as the triangles that were destroyed during the triangulation process.

D is then initialized as a single leaf node corresponding to triangle $p_0 p_{-1} p_{-2}$.

We recreated all the steps of the construction of D, considering the first case of the previous example, where a new point g is inserted into the graph P.

Figure 14: construction of the point location structure D.



Lemma 1

> *Every new edge created in the Delaunay triangulation algorithm or in the function to legalize an edge during the insertion of* p_r *is an edge of the Delaunay graph of* $P = \{p_{-2}, p_{-1}, p_0, \ldots, p_r\}$.

We can prove this Lemma by considering the previous case (*Figure 8*) in which node g is inserted inside the Delaunay graph, creating the new edges bg, fg, gc. If the triangle fbc is a triangle in the Delaunay triangulation before the addition of g then its circumcircle $C(fbc)$ contains no point h such that h is lower than g in its interior.
Then by shrinking $C(fbc)$ we can find another circumcircle C' through nodes b and g, that is contained in $C(fbc)$. Since we know that $C' \leq C(fbc)$, we also know that C is empty, which implies that the edge bg is a new Delaunay edge after the addition of g. We can apply these considerations also to the other edges obtained from the addition of g, which means that edges bg, fg and cg are legal edges of the set $P \cup g$.

Lemma 2

The expected number of triangles created by the algorithm is at most 9n + 1.

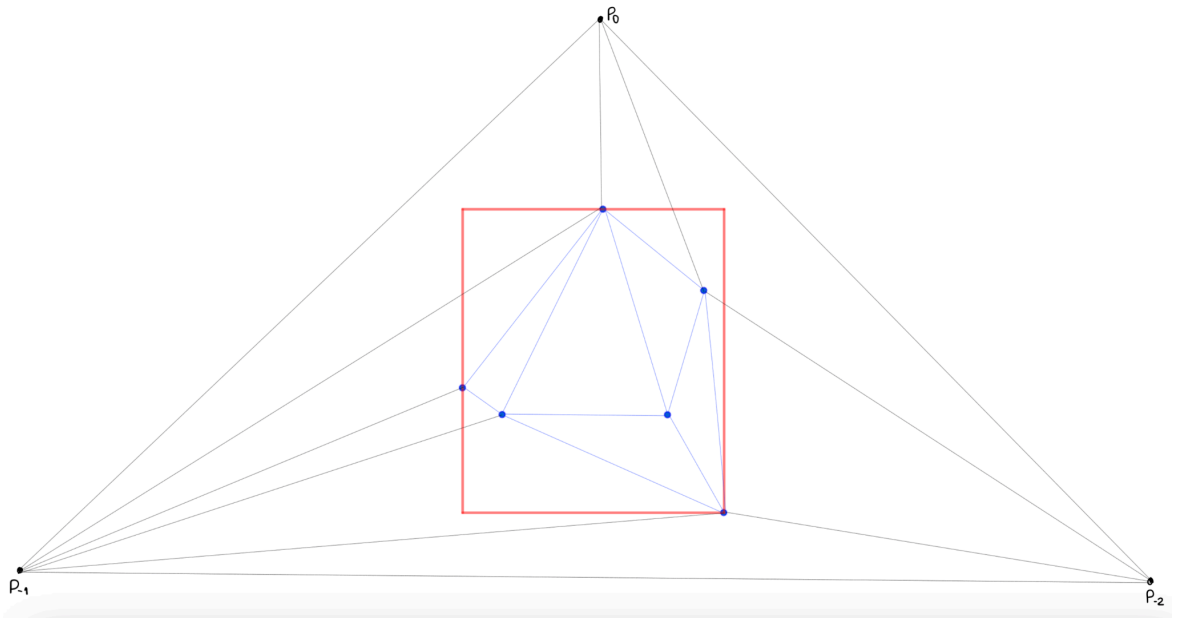
To prove this Lemma we need to consider various assumptions from backwards analysis.
In particular we need to remember the following steps:

1. the creation of a single triangle p_0, p_{-1}, p_{-2}, which contains all the points of set P and such that none of these three points lie inside one of the circumcircles of P. These points are treated symbolically to test illegal edges. (see *Figure 14*)
2. every point in $\{p_1, \dots, p_r\}$ has probability $\frac{1}{r}$ to be the last point,
3. in iteration r the point p_r is inserted and the triangle in which it is contained is splitted creating new triangles and edges,
4. the expected degree of point p_r is ≤ 6.

   This number is a generalization derived from Euler's formula $V - E + F = 2$.
5. the number of triangles t_r created at the insertion p_r is $\leq 2 * degree(p_r) - 3$. From this formula we can derive that t_r is $\leq 2 * 6 - 3$, so t_r is ≤ 9.

If we want to consider the overall number of triangles created by the algorithm we need to consider both the outer triangle $p_0 p_{-1} p_{-2}$ and the sum of the triangles created by all n points. So the expected overall number of triangles created by the algorithm is $\leq 9n + 1$, where n is the number of points on the plane.

Figure 15: triangulation of set P= {p_{-2}, p_{-1}, p_0, ..., p_r}.



Lemma 3

The computational complexity of the algorithm is $O(nlog(n))$.

This cost of the computation typically refers to algorithms like "incremental" or "divide and conquer" approaches that indeed achieve this time complexity. We can prove the lemma by analyzing each step of the algorithm and its computational cost.

The steps of the incremental algorithm are:

1. sorting: the input n points are sorted based on their coordinates. An efficient sorting algorithm takes complexity $O(nlog(n))$ for n points in the graph.
2. incremental insertion: a new point is added to the current graph.
3. affected triangles identification: the algorithm scans the triangles created by the insertion of the new point and decides whether each triangle needs to be updated or not. It takes time $O(log(n))$.
4. triangulation updating: the non-delaunay triangles are removed and substituted with new triangles through the edge flipping function. It takes time $O(log(n))$.

Steps 3 and 4 are repeated recursively until all the edges are legal and all the triangles in the graph are Delaunay triangles, only then it stops. This means that in the worst case scenario they are repeated n times.

Then the overall complexity of the algorithm is $O(nlog(n)) + O(nlog(n)) + O(nlog(n))$, where the first cost is referred to the sorting process, the second one is referred to the triangles identification for each point n and the third refers to the cost of updating the illegal triangles for each point n.
Since the complexity of the algorithm is calculated considering the largest element of the complexity analysis, we can conclude that the computational complexity of the randomized incremental algorithm to compute the Delaunay triangulation is $O(nlog(n))$.

Conclusion

We have seen how the Delaunay algorithm works and how we can construct the corresponding graph step by step. Of course, in reality the algorithm is much faster and it takes in consideration all the possible triangulations simultaneously of large sets of points.

To obtain the set of points P used in the examples of this report, the following commands should be run on GRASS after starting the software:

```
d.mon wx1 #start a new monitor
v.random output=random_points npoints=6 seed=15 #create the set P of random points
d.vect random_points icon=basic/circle color=blue size=7 #display set P
v.voronoi random_points out=voronoi_random_points #compute the Voronoi diagram of set P
v.delaunay random_points out=delaunay_random_points #compute the Delaunay triangulation of set P
d.vect -c voronoi_random_points type=boundary color=green #display the Voronoi diagram
d.vect -c delaunay_random_points  type=boundary color=blue #display the Delaunay graph
```