# Delaunay Triangulation Algorithm and Application to Terrain Generation

Faniry Harijaona Razafindrazaka (faniry@aims.ac.za)
African Institute for Mathematical Sciences (AIMS)

Supervised by: Doctor J W Sanders
International Institute for Software Technology
United Nations University, Macao

22 May 2009

*Submitted in partial fulfillment of a postgraduate diploma at AIMS*

# Abstract

We describe a randomized incremental algorithm for computing the Delaunay triangulation of a set of points and a recent technique of applying it to terrain generation. The algorithm is optimal, using a Directed Acyclic Graph (DAG)-based location structure for the incremental insertion which achieves an expected running time of $O(n \log n)$ and $O(n)$ expected storage. The analysis of the expected storage is simplified, and the algorithm is implemented and tested. The implementation is done, in its integrality, with CGAL-Python which is fast and friendly, followed by numerical statistics on different distributions. The terrain generation technique is based on gaussian functions and the plotting is made with the Open Source software Blender 3D which is probably its first combination with CGAL-Python. A high quality rendering of three simple terrains is shown as a demonstration of the terrain generator.

Mamaritra tetika fanamboarana ny fanatelozoroan' i Delaunay ao aminy velarana $\mathbb{R}^2$ amin'ny fomba kisendrasendra isika ary ny fampiasana azy io aminy famoronana vohitra. Ilay tetika dia maty paika izay mampiasa grafy misy toro-lalana ary tsy misy fihodonana aminy fanatsofoana ireo teboka isan'isany, ny fandalinana ny hafainganany kosa dia natao faran'izay tsotra. Ny famolavolana dia natao, aminy akapobeny, tamin'ny CGAL-Python izay aingana sady mora ampiasaina, ny vokatra numerika dia nomena ihany koa. Ilay fanamboarana vohitra dia tsy voafetra izay afaka mamboatra tendrombohitra arak'izay ilainy mpampiasa. Ilay vohitra dia natao kisary tamin'ilay fitaovana maimaimpona Blender 3D izay mety ho voalohany aminy fampiasana azy io sy CGAL-Python miaraka. Hisy vohitra roa haseho aminy endriny farany izay kanto mba anehoana ny fahatsaran'ilay fanamboarana.

## Declaration

I, the undersigned, hereby declare that the work contained in this essay is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

Faniry Harijaona Razafindrazaka, 22 May 2009

# Contents

# 1. Introduction

The Delaunay triangulation of a set of points is one of the classical problems in computational geometry. It was discovered in 1934 by the French mathematician Boris Nikolaevich Delone or Delaunay [2]. Many algorithms have since been proposed by computer scientists as well as mathematicians. In 1985, Guibas and Stolfi [10] proposed a Divide and Conquer algorithm which achieves the optimal bound of $O(n \log n)$. Later on, Steve Fortune [7] proposed a Sweepline algorithm for Voronoi diagrams which also achieves this bound. Unfortunately, they are hard to implement, so that engineers instead use incremental insertion algorithms because of their simplicity and the fact that they are not hard to implement. Guibas and Stolfi [10] proposed an incremental algorithm using a walking strategy to locate points which achieves $O(n^{1/2})$ per point location, and it was improved by Mucke [6] to an $O(n^{1/3})$ per point location. Although these algorithms provide good implementations, they are still sub-optimal algorithms. Thus we will present an optimal randomized incremental algorithm which uses a DAG-based data structure for the point location, using the analysis proposed in [9] and which solves the problem in $O(n \log n)$ expected time and $O(n)$ expected storage.

Nowadays, generating artificial terrains are indispensable. They are used in flight simulator, in making texture map to use as a background, in creating virtual worlds in game programming as well as in 3D animated movies. They are many ways to generate them, a simple randomize-and-smooth technique [13] can be used, or a more complicated way is based on fractal methods [16], whereas fluid simulation can also be used [16]. In this report, we present a recent technique based on gaussian functions and the Delaunay triangulation. The statistics parameters of the gaussian enable us to control the quality and the quantity of features in the heighten map.

The report follows the general approach of randomized incremental construction of the Delaunay triangulation, but differs in the following respects:

(i) the analysis of the expected storage is proved by backwards analysis;

(ii) the implementation is done with CGAL-python combined with Blender 3D which makes the drawing easier and produces results of high quality;

(iii) a technique of generating terrain based on the Delaunay triangulation is developed.

The work is divided in two parts. The first part introduces the definition of Delaunay triangulation and the theoretical analysis of the algorithm, while the second part is its application to terrain generation. The Delaunay triangulation is known to be the dual of the Voronoi diagram, as described in Chapter 2. We then study some properties of a Delaunay triangle from the *empty circle criterion* to the *local max-min angle criterion*. In Chapter 3, several algorithms are given and contrasted with the randomized incremental method, especially the Divide and Conquer algorithm [10] and the Plane Sweep algorithm [7]. The location structure is explained in Chapter 4 with the DAG-based datastructure, followed by pseudo code of the full algorithm. The analysis of the algorithm is simplified as much as possible, using basic tools from probability and backwards analysis. Finally, in Chapter 5, statistics on the speed of the point location are given, and the application of the Delaunay triangulation to the perspective view of topographic maps is explained. We then develop one method to generate terrain.

# 2. Voronoi diagram and Delaunay triangulation

In this section, we first introduce the notion of *Voronoi cells* and *half-planes*, and then give the duality properties of the Delaunay triangulation.

## 2.1 The Voronoi diagram

### 2.1.1 Voronoi Cells

Let $\mathcal{P} = \{p_1, p_2, p_3, \ldots, p_n\}$ be a set of points in the Euclidian plane which are called *the sites*. A region of the plane obtained by assigning every point to its nearest site $p_i$ is called the *Voronoi cell* $\mathcal{V}(p_i)$, that is,

$$\mathcal{V}(p_i) = \{x \in \mathbb{R}^2 : d(x, p_i) \leq d(x, p_j),\ \forall i \neq j\}.$$

It can be interpreted as the set of all points which are closer to $p_i$ than the other sites. Some points do not have an unique nearest site or nearest neighbor. The set of all points that have more than one nearest neighbor is called the Voronoi diagram $\mathcal{V}(\mathcal{P})$ for the set of sites.

### 2.1.2 Example

Consider three sites in the plane. Geometrically, the Voronoi diagram of these three sites is the perpendicular bisectors of the three sides of the triangle formed by the three (non-collinear) points which intersect at the circumcenter, the center of the unique circle that passes through the triangle's vertices. Thus the Voronoi diagram for three points must appear as in Figure 2.1.

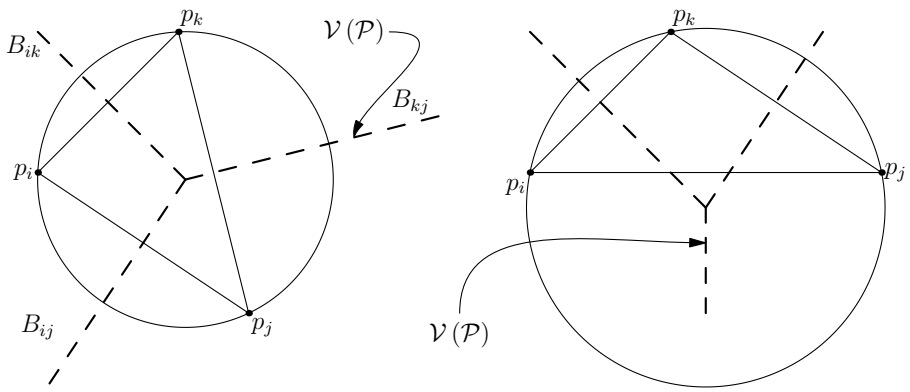



Figure 2.1: Three sites: bisectors meet at the circumcenter

### 2.1.3 Half-Planes

Let $H(p_i, p_j)$ be the closest half-plane bounded by $B_{ij}$ and containing $p_i$ (Figure 2.1). We can interpret $H(p_i, p_j)$ as,

$$H(p_i, p_j) = \{x \in \mathbb{R}^2 \mid d(x, p_i) < d(x, p_j)\}, \text{ for } i \neq j,$$

Using this notation for the definition of the Voronoi diagram, we have that

$$\mathcal{V}(p_i) = \bigcap_{j \neq i} H(p_i, p_j).$$

The intersection is taken over all $j$'s which are different from $i$. We will use this characterisation of the Voronoi diagram later on.

## 2.2 Dual of $\mathcal{V}(\mathcal{P})$

**Definition 2.2.1.** *The dual of a planar graph $G$ is constructed as follows: each face in $G$ is represented by a vertex, and for each edge $e \in E(G)$, we draw an edge between the vertices that represent the faces which share the edge $e$.*

**Definition 2.2.2.** *A triangulation of a set of point $\mathcal{P}$ is defined to be a planar subdivision $S$ such that adding an edge connecting two points of $\mathcal{P}$ which is not in $S$ will destruct its planarity.*

### 2.2.1 Construction

In 1934, Delaunay [2] proved that the dual graph of the Voronoi diagram drawn with straight lines produces a planar triangulation of the Voronoi sites $\mathcal{P}$, now called the *Delaunay triangulation* $\mathcal{D}(\mathcal{P})$. Figure 2.2 shows the Voronoi diagram of $n = 11$ sites and its corresponding dual graph, the Delaunay triangulation.

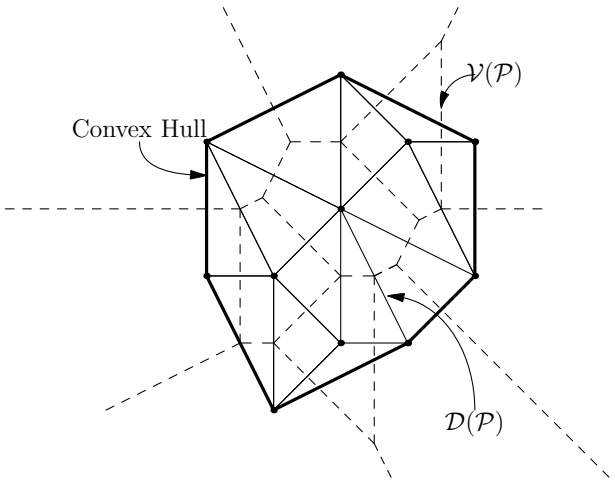



Figure 2.2: Voronoi diagram, Delaunay triangulation and the convex hull of $n = 11$ sites

### 2.2.2 Duality Properties

The duality of those graphs links them into a one to one correspondence between edges, vertices and faces which, thus, imply the following observations:

**D**1. $\mathcal{D}(\mathcal{P})$ is the straight line dual graph of $\mathcal{V}(\mathcal{P})$ (by definition);

**D**2. $\mathcal{D}(\mathcal{P})$ is a triangulation if no four points of $\mathcal{P}$ are cocircular (see Remark 2.3.2);

**D**3. every triangle of $\mathcal{D}(\mathcal{P})$ corresponds to a vertex of $\mathcal{V}(\mathcal{P})$;

**D**4. each edge of $\mathcal{D}(\mathcal{P})$ corresponds to an edge of $\mathcal{V}(\mathcal{P})$;

**D**5. each node of $\mathcal{D}(\mathcal{P})$ corresponds to a region of $\mathcal{V}(\mathcal{P})$;

**D**6. The boundary of $\mathcal{D}(\mathcal{P})$ is the convex hull[1] of the sites $\mathcal{P}$ (Figure 2.2);

**D**7. The interior of each triangle face of $\mathcal{D}(\mathcal{P})$ contains no site.

**Remark 2.2.3.** *Properties* **D***6 and* **D***7 are the most interesting and can be verified in Figure 2.2. Property* **D***7 has its equivalence in the property of Voronoi diagram saying that the circumcircle of every three sites contains no other sites [15].*

## 2.3 The Delaunay triangulation

**Definition 2.3.1** (Illegal Edge)**.** *Let $\overline{p_ip_j}$ be the edge incident to the triangles $\Delta p_ip_jp_k$ and $\Delta p_ip_jp_l$ (Figure 2.3). We say that the edge $\overline{p_ip_j}$ is illegal if the point $p_l$ lies in the interior of the circumcircle of $\Delta p_ip_jp_k$ or $p_k$ in that of $\Delta p_ip_lp_j$.*

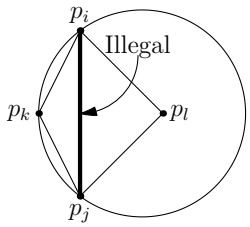



Figure 2.3: Illegal Edge

**Remark 2.3.2.** *This definition is valid only if the points $p_i, p_j, p_k$ and $p_l$ do not lie in a common circle. It gives a degeneracy and violates the uniqueness of the triangulation, since any cross edge is legal. The two graphs in Figure 2.4 are geometrically different but define each a Delaunay triangulation. In this*

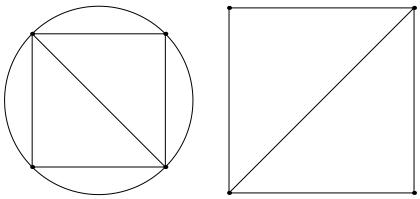


Figure 2.4: Two different Delaunay triangulation spanning the same set of points

*case, one can fix a rule such that a point lying on the circumcircle of a given triangle is set to be outside of this circle.*

[1]The convex hull of a set of points is the smallest convex set that includes all the points

**Definition 2.3.3** (The empty circle criterion)**.** *Given a triangulation $\mathcal{T}$ of a set of points $\mathcal{P}$, if the circumcircle of a triangle in the triangulation is an empty circle, we say that the triangle satisfies the empty circle criterion.*

This empty circle criterion is equivalent to the fact that every edge in the triangulation is legal. The triangle in Figure 2.3 do not satisfy the empty circle criterion.

**Theorem 2.3.4** (The empty circle theorem)**.** *Let $\mathcal{P}$ be a set of points in the plane. A triangulation $\mathcal{T}$ of $\mathcal{P}$ satisfies the empty circle criterion if and only if $\mathcal{T}$ is a Delaunay triangulation of $\mathcal{P}$.*

*Proof.* The sufficient condition is given by Remark **??**. Let us show that the triangulation $\mathcal{T}$ is in fact the dual of a Voronoi diagram and then conclude with Property **D**1.

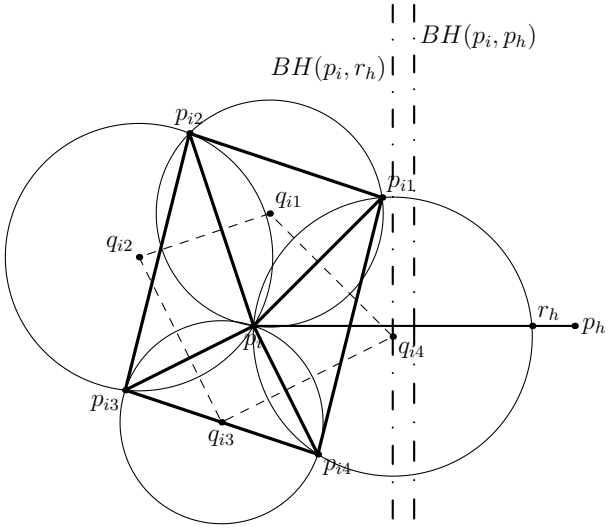



Figure 2.5: Illustration of the proof of the empty circle theorem[14]

Let $p_i \in \mathcal{P}$ and $p_{i1}, \ldots, p_{ik}$ be the vertices of the triangles sharing $p_i$. Moreover, let $q_{i1}, \ldots, q_{ik}$ be the centres of the circumcircles $C_{i1}, \ldots, C_{ik}$ corresponding respectively to the triangles $\Delta p_i p_{i1} p_{i2}, \ldots, \Delta p_i p_{ik} p_{i1}$ in counterclockwise order (Figure 2.5). Since $\overline{p_i p_{ij}}$ is orthogonal to $\overline{q_{ij} q_{ij-1}}$, $j = 1, \ldots, k$, the polygon with vertices $q_{i1}, \ldots, q_{ik}$ is given by $\bigcap_{j=1}^{k} H(p_i, p_{ij})$. By hypothesis, $C_{i1}, \ldots, C_{ik}$ are empty circles, which means that there exists at least one $p_h \in \mathcal{P} \setminus \{p_{i1}, \ldots, p_{ik}\}$ which lies outside these circles. The edge $\overline{p_i p_h}$ intersects a boundary of an arc of circumcircle $C_{ij}$ at a point called $r_h$. Then the boundary of $H(p_i, r_h)$ passes through $q_{ij}$ since $r_h \in C_{ij}$, and

$$\bigcap_{j=1}^{k} H(p_i, p_{ij}) \subset H(p_i, r_h) \subset H(p_i, p_h).$$

This holds for all $p_h$ in $P \setminus \{p_{i1}, \ldots, p_{ik}\}$. Therefore,

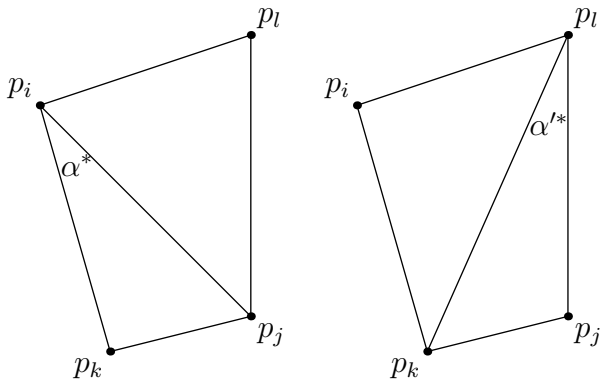
$$\bigcap_{j=1}^{k} H(p_i, p_{ij}) = \bigcap_{j \neq i} H(p_i, p_j) = \mathcal{V}(p_i).$$

Then, the polygon constructed by $q_{i1}, \ldots, q_{ik}$ is the Voronoi polygon of $p_i$. If we carry out the above procedure for every $p_i \in \mathcal{P}$, we obtain $\mathcal{V}(\mathcal{P})$. Since $\mathcal{T}$ is the dual of $\mathcal{V}(\mathcal{P})$, it follows from Property D1 that the given triangulation is the Delaunay triangulation of $\mathcal{P}$. □

**Definition 2.3.5** (The local max-min angle criterion)**.** *Given a triangulation $\mathcal{T}$, let $\overline{p_ip_j}$ be an edge in $\mathcal{T}$ which the triangle $\Delta p_ip_lp_j$ and $\Delta p_ip_jp_k$ share, and $Q$ the quadrilateral formed by these triangles. Furthermore, let $\alpha_i, 1 \leq i \leq 6$, be the six angles of $Q$ and $\alpha'_i, 1 \leq i \leq 6$, those of $Q'$ obtained by flipping the diagonal of $Q$. If*

$$\min_{1\leq j\leq 6} \alpha'_j \leq \min_{1\leq j\leq 6} \alpha_j,$$

*then we say that the edge $\overline{p_ip_j}$ satisfies the local max-min angle criterion. In Figure 2.6, we use the notations $\alpha^* = \min_{1\leq j\leq 6} \alpha_j$ and $\alpha'^* = \min_{1\leq j\leq 6} \alpha'_j$, from which we see that Figure 2.6 (a) satisfies the local max-min since $\alpha'^* \leq \alpha^*$,whereas Figure 2.6 (b) does not.*




(a) $Q$ (b) $Q'$

Figure 2.6: The local max-min angle criterion

**Remark 2.3.6.** *Finding the minimum angle of a quadrilater appears slightly complicated. In practice, we use the following relation [14].*

$$\alpha'^* \leq \alpha^* \iff p_l \text{ lies outside the circumcircle of } \Delta p_ip_kp_j.$$

**Theorem 2.3.7** (The local max-min angle theorem [14])**.** *Let $\mathcal{P}$ be a set of points in the plane. A triangulation $\mathcal{T}$ of $\mathcal{P}$ satisfies the max-min angle criterion if and only if $\mathcal{T}$ is a Delaunay triangulation of $\mathcal{P}$.*

*Proof.* Remark 2.2.3 and Remark 2.3.6 imply that if a triangle $\mathcal{T} \in \mathcal{D}(\mathcal{P})$, every internal edge satisfies the local max-min angle criterion. It remains to prove that if every internal edge of $\Delta p_ip_jp_k$ in $\mathcal{T}$ satisfies the max-min angle criterion, then $\mathcal{T} \in \mathcal{D}(\mathcal{P})$. Since, $\overline{p_ip_j}$ satisfies the local max-min angle criterion, $p_l$ is outside the circumcircle of $\Delta p_ip_jp_k$. Since the edges $\overline{p_ip_l}, \overline{p_jp_l}, \overline{p_ip_k}$ and $\overline{p_kp_j}$ satisfy the local max-min angle criterion, the triangles $\Delta p_ip_jp_k$ and $\Delta p_ip_jp_l$ cannot contain other sites. Therefore, there are no points in the circumcircle of $\Delta p_ip_jp_k$ and $\Delta p_ip_jp_l$. Applying the same procedure to every triangle, we can prove that every corresponding circumcircle is an empty circle. From Property **D**6, the triangulation $\mathcal{T}$ is $\mathcal{D}(\mathcal{P})$. □

# 3. The Algorithm

Having described the Delaunay triangulation, we introduce here an optimal algorithm which computes directly the Delaunay triangulation of a set of points in the plane. The algorithm is in the class of randomized incremental constructions and will be contrasted with other optimal algorithms.

### 3.0.1 Basic Concepts

Let $\mathcal{P}$ be a set of points in the plane and $\mathcal{D}(\mathcal{P}_r)$ a triangulation of $\mathcal{P}_r \subset \mathcal{P}$ at stage $r$. The triangulation $\mathcal{D}(\mathcal{P}_{r+1})$ is obtained by inserting a point $p_r$, randomly taken from $\mathcal{P} \setminus \mathcal{P}_r$, in $\mathcal{D}(\mathcal{P}_r)$. The algorithm locates $p_r$ using a DAG for point location based on the history of the Delaunay triangulation called *the Delaunay tree* (Section 4.2). At this stage two cases can happen: either the point lies on an edge or strictly inside the triangle. In the first case, we construct two edges connecting $p_r$ to the vertices of the two adjacent triangles, whereas in the second case, we construct three edges connecting the point and the three vertices of the triangle (Figure 3.1) . Since we look for a legal triangulation, we check if every edge of the containing triangle is valid with respect to the empty circle criterion. We then proceed recursively until all the points are inserted. At last, we get the Delaunay triangulation of $\mathcal{P}$. Figure 3.2 shows the insertion process.

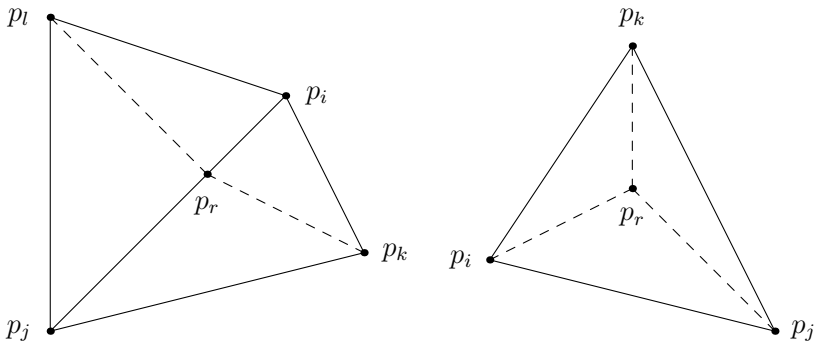



Figure 3.1: Two cases of the positions of $p_r$

### 3.0.2 Why Randomized Incremental Algorithm?

Interesting algorithms have been implemented. The most remarquable of them are:

- *Plane Sweep Algorithm*

  In 1985, Fortune invented a clever Plane Sweep algorithm which attains the optimal bound $O(n \log n)$. Plane-sweep algorithms pass a sweep line over the plane, leaving at any time the problem solved for the portion of the plane already swept and unsolved for the portion not yet reached. A plane-sweep algorithm for the Voronoi diagram would have the diagram constructed behind the line. It seems quite impossible, as Voronoi edges of a Voronoi region $\mathcal{V}(p)$ would be encountered by the sweep line $L$ before $L$ encounters the site $p$ responsible for the region. Fortune surmounted this seeming impossibility by an extraordinary clever idea [15].

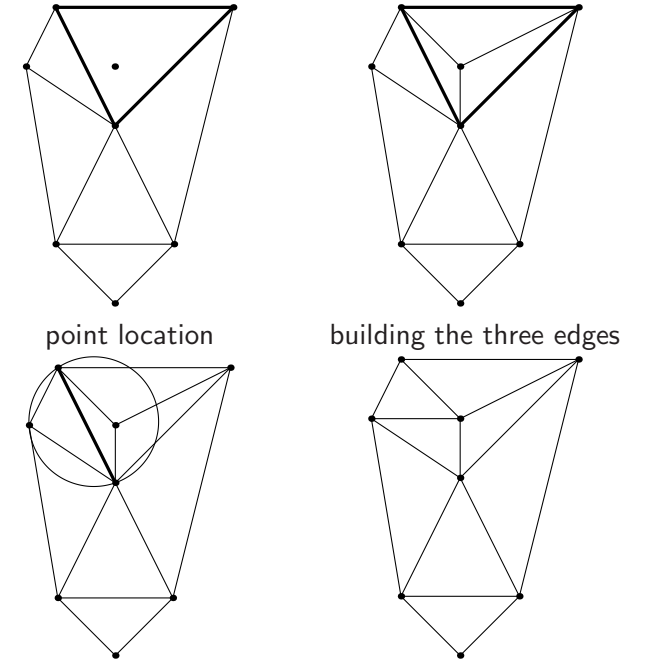



Figure 3.2: The insertion process

- *Divide and Conquer Algorithm*

  Given a set of points $\mathcal{P}$, instead of computing the Delaunay triangulation of $\mathcal{P}$, we sort the set of points $\mathcal{P}$ in $x$ coordinates and divide it in two subsets $\mathcal{P}_1$ and $\mathcal{P}_2$. The Delaunay triangulation $\mathcal{D}(\mathcal{P})$ is obtained by merging the two triangulations. Recursively applied, this method leads, in principle, to the consideration of an elementary problem. This algorithm attains the optimal running time $O(n \log n)$.

Unfortunately, the implementation of these algorithms still requires codes of significant programming complexity, which is why people prefer to use randomized incremental because it is less difficult to implement when the datastructure is correctly chosen; it is randomized, so that its complexity can be analysed using tools from probability theory [17]; it is "on line" in the sense that the input is inserted piece-by-piece and the computation is made accordingly without having the entire input available from the start. Finally, it can have an expected running time of $O(n \log n)$ when the insertion is made in random order [9].

# 4. Computation of the Delaunay Triangulation

We have now enough information to build the algorithm. We proceed in four phases. We first start with the initialisation of three dummy points which enclose all the points of $\mathcal{P}$; we then describe an efficient location technique to locate queries; we next discuss the *Legalization* which is a procedure verifying the empty circle criterion; and finally, give pseudo-code of the algorithm. The last section contains the analysis of the algorithm

## 4.1 Initialisation of the Three Dummy Points

Our approach in Section 3.0.1 is already in the step case, which means going from $\mathcal{D}(\mathcal{P}_r)$ to $\mathcal{D}(\mathcal{P}_{r+1})$. One question is left: what is the base case, since we do not have any triangle to make an insertion? There is a recent construction of a universal triangle $\Delta\Omega_1\Omega_2\Omega_3$ which encloses all the points such that $\Omega_1 = (3M, 0), \Omega_2 = (0, 3M)$ and $\Omega_3 = (-3M, -3M)$ where $M$ is the maximum in absolute value of any coordinate of a point in $\mathcal{P}$ [1]. In this report, we just make sure that these three points are far away in such a way that every edge of $\Delta\Omega_1\Omega_2\Omega_3$ always satisfies the local max-min criterion (Figure 4.1).

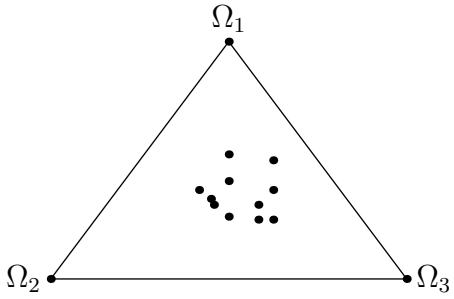



Figure 4.1: Initialisation of the three dummy points

## 4.2 The Delaunay Tree

The most complicated part of this algorithm is the point location based on a *Directed Acyclic Graph* (DAG) data structure, also called the Delaunay tree. The basic idea is that, whenever we replace some triangle $\Delta p_i p_j p_k$ by new triangles, we leave $\Delta p_i p_j p_k$ as part of the structure $DTree$, mark it as "old" and maintain a pointer from $\Delta p_i p_j p_k$ to each of the newly generated triangles that partially overlaps it. More explicitly, we start to locate the point by the root $\Omega_1\Omega_2\Omega_3$ in $DTree$. We check the three children of this root to know in which triangle the new inserted point is. We then recursively descend to the corresponding child. The point location ends when the triangle which contains the current inserted point is reached. Since the out-degree of a node is at most three, this step takes linear time in the number of nodes on the search path, or in the number of triangles stored in $\mathcal{D}(\mathcal{P})$ that contains $P$.

As it is often the case, an edge flip can affect the structure of $DTree$. Suppose that we have inserted a point $P$ in some triangle $\Delta_1$ which splits this one into three subtriangles $\Delta_{11}, \Delta_{12}, \Delta_{13}$ (or two if the inserted point lies on an edge). In $DTree$, we have to add three (or two) new leaves and then make a

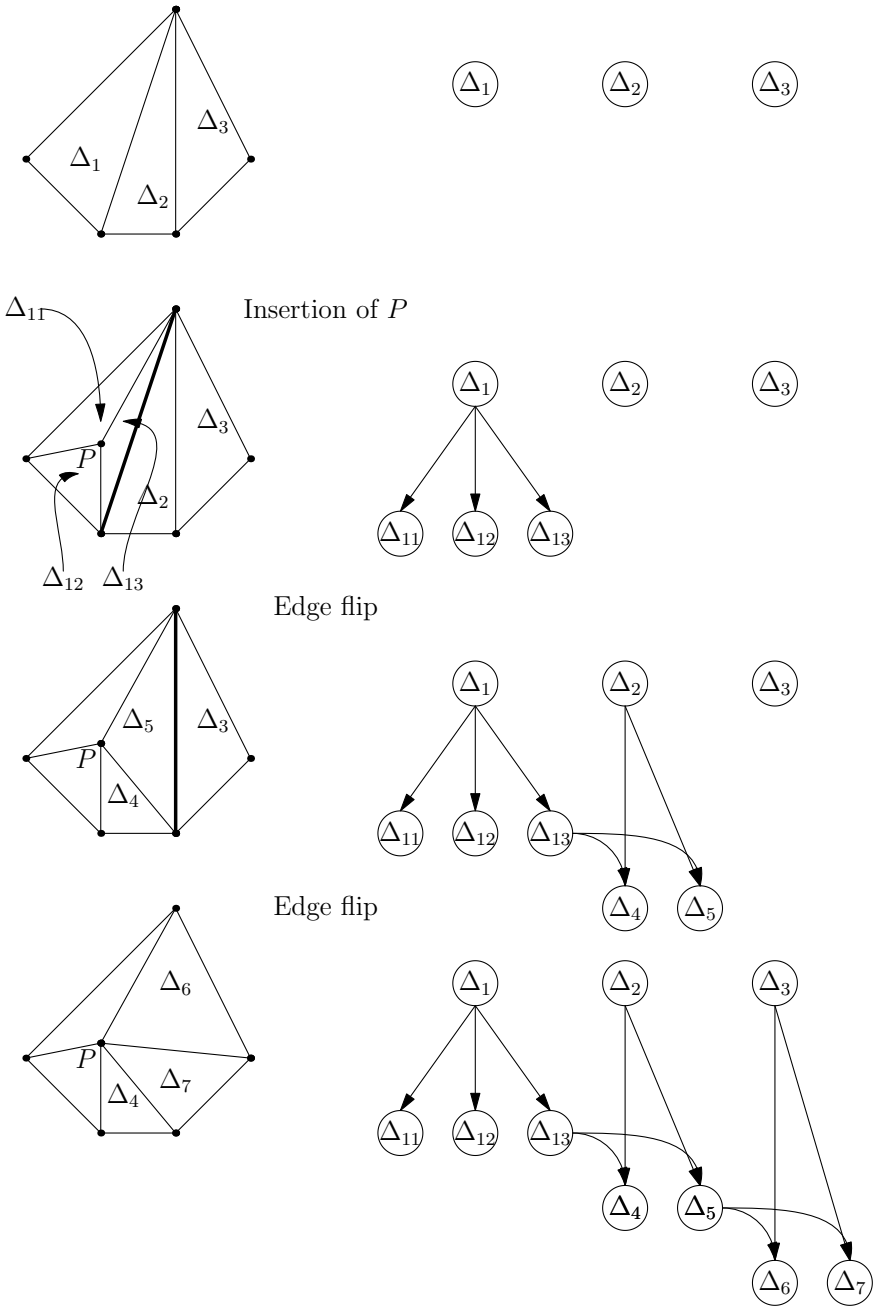



Figure 4.2: The change due to several edge flips on the data structure $DTree$

pointer from $\Delta_1$ to those leaves. In the same way, when we replace two triangles $\Delta_2$ and $\Delta_{13}$ by $\Delta_4$ and $\Delta_5$ by an edge flip, we create leaves for the two new triangles, and keep a pointer from $\Delta_2$ and $\Delta_{13}$ to the two new leaves. An example is illustrated in Figure 4.2 where we left out the other part of the structure which is not affected by the insertion of $P$.

## 4.3 Legalization

After finding in which triangle $p_r$ is inserted, we have to construct three edges which connect $p_r$ to the three vertices of the triangle. At this stage, we need to test if all the edges of the initial triangle remain valid. Observe that an edge $\overline{p_ip_j}$ that was initially valid becomes invalid if one of the new triangle incident to it has changed by the insertion of a new point or an edge flip of some other triangles. We thus need to use the geometric predicate $\texttt{Incircle}(A, B, C, D)$ which returns true if $D$ lies in the circumcircle of $\Delta ABC$. This condition is equivalent to,

$$\text{Incircle}(A, B, C, D) \iff \det \begin{pmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{pmatrix} > 0 \qquad [10].$$

**Remark 4.3.1.** *In order to know if an edge $\overline{p_ip_j}$ is legal or not, it is sufficient to test whether $p_r$ lies in the circumcircle of the triangle $\Delta_i$, $i = 1, 2, 3$, lying in the other side of an edge $\overline{p_ip_j}$ of $\Delta p_ip_jp_k$ [9].*

Since the occurrence of an edge flip may affect the validity of the other edges, we need to build a sub-routine called `ValidEdge`, which takes a face and a vertex and verifies the empty circle criterion for the face adjacent and opposite to this vertex. In Algorithm 1, `flip` is a function which flips the diagonal of a quadrilater formed by the triangles $\Delta$ and $\Delta'$.

**Algorithm 1** ValidEdge$(\Delta, p_r, \mathcal{D}(\mathcal{P}))$

```
Let Δ_adj be the triangle opposite to p_r and adjacent to Δ
if InCircle(Δ_adj, p_r) then
    (* We have to make an edge flip *)
    flip(Δ, Δ_adj, p_r, D(P))
    Let Δ' and Δ'' be the two new triangles, recursively legalize them
    ValidEdge(Δ', p_r, D(P))
    ValidEdge(Δ'', p_r, D(P))
end if
```

Moreover, we need to introduce another geometric predicate discussed in [10], the $\texttt{CClockwise}(A, B, C)$ test which returns true if the triangle path from $A$ to $B$ to $C$ is oriented counterclockwise. In terms of coordinates, this can numerically be implemented as

$$\text{CClockwise}(A, B, C) \iff \det \begin{pmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{pmatrix} > 0.$$

Expanding the determinant and rearranging gives the simple condition,

$$(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A) > 0.$$

## 4.4 Efficient Incremental Algorithm

We give here a high-level description of the algorithm using the four processes described above.

**Algorithm 2** Delaunay Triangulation of $\mathcal{P}$

**Input:** A set $\mathcal{P}$ of $n+1$ points in the plane
**Output:** A Delaunay triangulation of $\mathcal{P}$

Initialise the triangulation $\mathcal{D}(\mathcal{P})$ to be $\Omega_1\Omega_2\Omega_3$
Make a random permutation $[p_1, p_2, p_3, \ldots, p_n]$ of $\mathcal{P}$
**for** $r \leftarrow 1$ to $n$ **do**
    (* Take the point $p_r$ which should not be already selected *)
    Find a triangle $\Delta$ which contains $p_r$
    **if** If $p_r$ lies in the interior of $\Delta$ **then**
        Divide $\Delta$ to $\Delta_1, \Delta_2$ and $\Delta_3$
        (* Check for the validity of edges *)
        `ValidEdge`$(\Delta_1, p_r, \mathcal{D}(\mathcal{P}))$
        `ValidEdge`$(\Delta_2, p_r, \mathcal{D}(\mathcal{P}))$
        `ValidEdge`$(\Delta_3, p_r, \mathcal{D}(\mathcal{P}))$
    **else**
        (*$p_r$ lies on an edge $e$ of $\Delta$*)
        Let $\Delta'$ be the triangle sharing the edge $e$ with $\Delta$, then divide $\Delta'$to $\Delta'_1, \Delta'_2$ and $\Delta$ to $\Delta_1, \Delta_2$
        (* Check for validity of edges *)
        `ValidEdge`$(\Delta'_1, p_r, \mathcal{D}(\mathcal{P}))$
        `ValidEdge`$(\Delta'_2, p_r, \mathcal{D}(\mathcal{P}))$
        `ValidEdge`$(\Delta_1, p_r, \mathcal{D}(\mathcal{P}))$
        `ValidEdge`$(\Delta_2, p_r, \mathcal{D}(\mathcal{P}))$
    **end if**
**end for**
Delete all triangles containing $\Omega_1$, $\Omega_2$ or $\Omega_3$ as vertices from $\mathcal{D}(\mathcal{P})$
**return** $\mathcal{D}(\mathcal{P})$

## 4.5 Correctness and Efficiency

The goal of this section is to prove that the Delaunay triangulation of a set of planar points constructed with the incremental randomized algorithm can be computed in $O(n \log n)$ expected time and $O(n)$ expected storage. Before starting, let $\mathcal{P}_r := \{p_1, \ldots, p_r\}$ be the set of points already inserted at the stage $r$ which contains $r$ elements and $\mathcal{D}(\mathcal{P}_r) := \mathcal{D}(\{\Omega_0, \Omega_1, \Omega_2\} \cup \mathcal{P}_r)$ the corresponding Delaunay triangulation.

**Lemma 4.5.1.** *(General results)*

1. *Let $\mathcal{P}$ be a set of $n$ points in the plane which are not all collinear, and let $k$ denote the number of points in $\mathcal{P}$ that lie on the boundary of the convex hull of $P$. Then any triangulation of $P$ has $2n-2-k$ triangles and $3n-3-k$ edges.*
2. *For $n \geq 3$, the number of vertices in the Voronoi diagram of a set of $n$ point sites in the plane is at most $2n-5$ and the number of edges is at most $3n-6$.*

*Proof.* The proof is not complicated, but quite long, using some results from graph theory. For more details, we refer to [1].

**Lemma 4.5.2.** *The expected degree of a random point of $\mathcal{P}_r$ is at most $6$.*

*Proof.* Let $\deg(p_r, \mathcal{D}(\mathcal{P}_r))$ be the degree of $p_r$ in $\mathcal{D}(\mathcal{P}_r)$. For the moment, we fixe the set $\mathcal{P}_r$ and make backwards analysis. It follows from the second result of Lemma 4.5.1 that $\mathcal{D}(\mathcal{P}_r)$ has at most $3(r+3)-6$ edges, where the extra three edges come from the initial triangle $\Omega_1\Omega_2\Omega_3$, and therefore, the Handshake Lemma [1] implies that the total degree of the vertices in $\mathcal{P}_r$ is less than $2[3(r+3)-9] = 6r$. We deduce that the expected degree of a random point of $\mathcal{P}_r$ is at most $6$. □

**Theorem 4.5.3.** *The expected number of flips done by* `ValidEdge` *after inserting* $p_r$ *is* $O(1)$.

*Proof.* The key observation is that the number of flips that we make when we insert $p_r$ is given by $\deg(p_r, \mathcal{D}(\mathcal{P}_r)) - 3$ (see Figure 4.3). Therefore,

$$\begin{aligned}
\mathsf{E}[\text{number of flips after inserting } p_r] &= \mathsf{E}[\deg(p_r, \mathcal{D}(\mathcal{P}_r))] - 3 \\
&= \frac{1}{r+3}\sum_{i=1}^{r+3} \deg(p_i, \mathcal{D}(\mathcal{P}_r)) - 3 \\
&, = \frac{1}{r+3}.6(r+3) - 3 \qquad \text{(according to Lemma 4.5.2)} \\
&\leq 3 \\
&= O(1).
\end{aligned}$$

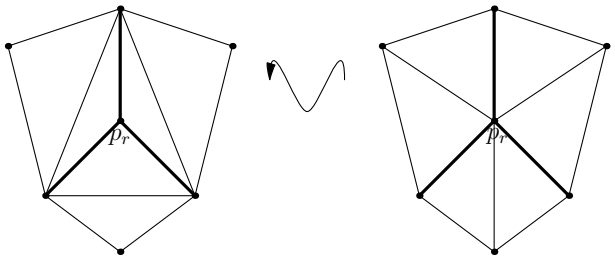



Figure 4.3: Backwards analysis on the degree of $p_r$

□

**Lemma 4.5.4.** *The expected number of triangles created by the full algorithm is at most* $9n + 1$.

*Proof.* The insertion of $p_r$ splits the current triangle to three or four new triangles, and gives the same number of edges incident to $p_r$. Every edge which is flipped in procedure `ValidEdge` creates two new triangles and two new edges incident to $p_r$, creating at most $2\deg(p_r, \mathcal{D}(\mathcal{P}_r)) - 3$ triangles. Hence,

$$\begin{aligned}
\mathsf{E}[\text{number of triangles created at step } r] &\leq \mathsf{E}[2\deg(p_r, \mathcal{D}(\mathcal{P}_r)) - 3] \\
&= 2\mathsf{E}[\deg(p_r, \mathcal{D}(\mathcal{P}_r))] - 3 \\
&\leq 2 \cdot 6 - 3 = 9,
\end{aligned}$$

[1] For any graph $G$, we have the relation

$$\sum_{v \in V(G)} \deg v = 2|E(G)|,$$

where $V(G)$ is the set of vertices of $G$ and $E(G)$ is the set of edges.

having also used Lemma 4.5.2.

The desired result is then obtained by summing over $r$ and using the linearity of the expectation and then adding the initial triangle $\Omega_1\Omega_2\Omega_3$. □

**Definition 4.5.5.** *The scope of a triangle is defined in [9] to be the number of sites contained in the interior of its circumcircle. The scope of a triangle can assume values between* $0$ *and* $n-3$*, and triangles with zero scope are the Delaunay triangles.*

**Lemma 4.5.6.** *[9] Let* $\Delta p_i p_j p_k$ *be a triangle with scope* $k$*. Then the probability that* $\Delta$[2] *appears as a Delaunay triangle during the incremental construction is given by*

$$\textit{Proba}(\Delta \textit{ appears as Delaunay triangle}) = \frac{6}{(k+1)(k+2)(k+3)}. \tag{4.1}$$

*Proof.* The only possibility for $\Delta$ to be Delaunay is that $p_i, p_j$ and $p_k$ are inserted first before any of the $k$ sites inside its circumcircle. This condition is equivalent to compute the event $E$: "insert $p_i, p_j$ and $p_k$ first (without any order) and independently on the time of insertion of the sites of its scope". Hence,

$$\text{Proba}(E) = \frac{3!k!}{(k+3)!} = \frac{6}{(k+1)(k+2)(k+3)}.$$

□

We next introduce the new variable $T_l$ defined to be the set of triangles $\Delta p_i p_j p_k$ whose scope is $l$, and we use the notation $T_{\leq m} = \bigcup_{d=0}^{m} T_d$.

**Lemma 4.5.7.** *[9] The number of triangles* $\Delta p_i p_j p_k$ *whose scope is* $m$ *is*

$$\|T_{\leq m}\| = O(n(m+1)^2). \tag{4.2}$$

*Proof.* This proof is an extended version of the one presented in [9] but the probabilistic technique is from Clarkson and Shor [17]. The idea is to take a random sample $\mathcal{R}$ of size $r$ $(r \geq 3)$ among the given sites, and denote by $\mathcal{D}(\mathcal{R})$ the set of Delaunay triangles formed by this sample. The expected number of such triangles is thus

$$\begin{aligned}
\mathbf{E}\left[\|\mathcal{D}(\mathcal{R})\|\right] &= \sum_{\Delta} \text{Proba}(\Delta \in \mathcal{D}(\mathcal{R})) \\
&= \sum_{l=0}^{n-3} \sum_{\Delta \in T_l} \text{Proba}(\Delta \in \mathcal{D}(\mathcal{R})) \\
&\geq \sum_{l=0}^{m} \sum_{\Delta \in T_l} \text{Proba}(\Delta \in \mathcal{D}(\mathcal{R})).
\end{aligned}$$

A necessary condition for $\Delta p_i p_j p_k$ to be in $\mathcal{D}(\mathcal{R})$ is that $p_i, p_j, p_k \in \mathcal{R}$, where as a sufficient condition is that none of the $l$ sites in the interior of its circumcircle is in $\mathcal{R}$. Denoting this event as $F$, we have

$$\text{Proba}(F) = \frac{\binom{r-3}{n-j-3}}{\binom{r}{n}} \geq \frac{\binom{r-3}{n-k-3}}{\binom{r}{n}} \text{ for } j \leq k.$$

[2] Here, the triangle $\Delta$ has $p_i, p_j$ and $p_k$ as vertices but we use this notation for simplicity

Since $\|\mathcal{D}(\mathcal{R})\|$ is at most $2r$ (first result of Lemma 4.5.1), we obtain

$$2r \geq \mathbf{E}\left[\|\mathcal{D}(\mathcal{R})\|\right] \geq \sum_{l=0}^{k} \|T_l\| . \frac{\binom{r-3}{n-k-3}}{\binom{r}{n}},$$

which amounts to

$$\|T_{\leq k}\| \leq \frac{2r\binom{r}{n}}{\binom{r-3}{n-k-3}} = \frac{2n\binom{r-1}{n-1}}{\binom{r-3}{n-k-3}} = 2nB(n-1, k+2, r-3, 2),$$

where

$$B(N, M, s, t) = \frac{\binom{s+t}{N}}{\binom{s}{N-M}}.$$

In order to obtain the best upper bound, we have to minimise $B(n-1, k+2, r-3, 2)$ with respect to $r$. In general, we have for successive values of $s$ the ratio [9].

$$B(N, M, s+1, t)/B(N, M, s, t) = t(N-M+1)/M.$$

Hence, the minimum of $B(N, M, s, t)$ for fixed $N, M$ and $t$ occurs when $s = [t(N-M+1)/M]$.

In our case, the function $B$ is minimised when

$$r = [2n/(k+2)] + 1. \tag{4.3}$$

Now,

$$B(n-1, k+2, r-3, 2) = \frac{n-1}{r-1} \cdot \frac{n-2}{r-2} \cdot \frac{n-3}{n-r} \cdot \frac{n-4}{n-r-1} \cdot \cdots \cdot \frac{n-k-2}{n-r-k+1}.$$

Observing that

$$\frac{n-1}{r-1} < \frac{n-2}{r-2}, \qquad r < n$$
$$fracn - in - (r+i-3) \leq \frac{n-k-2}{n-r-k+1}, \qquad 3 \leq i \leq k+2,$$

we find

$$B(n-1, k+2, r-3, 2) < \left(\frac{n-2}{r-1}\right)^2 \left(\frac{n-k-2}{n-r-k+1}\right)^k.$$

Using the minimum value $r$ in (4.3), we obtain

$$\begin{aligned} B(n-1, k+2, r-3, 2) &< \left(\frac{(n+2)(k+2)}{2n-2k-4}\right)^2 \left(1+\frac{2}{k}\right)^k \\ &< \left(\frac{n-2}{2}\right)^2 (k+2)^2 \left(1+\frac{2}{k}\right)^k \\ &= O((k+1)^2), \end{aligned}$$

having also used the fact that $k \leq n-3$.

□

**Theorem 4.5.8.** *[9] The Delaunay triangulation of a set $\mathcal{P}$ of $n$ points in the plane can be computed within $O(n \log n)$ expected running time using $O(n)$ expected storage.*

*Proof.* In Algorithm 2, if line 5 is not taken into account, then the running time of the algorithm is proportional to the number of created triangles. According to Lemma 4.5.4, we can conclude that the expected running time without the time for point location is $O(n)$.

The search structure $DTree$ could use more than linear storage. However, since every node of $DTree$ is formed by a triangle created by the algorithm, it follows from Lemma 4.5.4 that the corresponding expected number is $O(n)$ which gives the $O(n)$ expected storage.

To locate a point $p_r$, we have to visit every node of $DTree$ until we reach the current triangulation containing $p_r$. Therefore, the time to locate $p_r$ is $O(1)$ plus the number of visited triangles marked as "old" which contains $p_r$ and have been destroyed. Let $\Delta p_i p_j p_k$ be a triangle (Delaunay or not) and $\Delta_{adj} p_i p_j p_l$ its adjacent triangle. One of the following events can destroy the triangle $\Delta p_i p_j p_k$:

(i) a new point $p_r$ has been inserted inside (or on one of the edges of) $\Delta$ which subdivides it into three (or two) subtriangles;

(ii) an edge flip has replaced $\Delta$ and $\Delta_{adj}$ by the pair $\Delta'$ and $\Delta'_{adj}$.

In the first case, $\Delta$ was a Delaunay triangle before $p_r$ was inserted. In the second case, either $\Delta$ was a Delaunay and the opposite vertex $p_l$ was inserted, or $\Delta_{adj}$ was a Delaunay triangle and the opposite vertex $p_k$ was inserted, in which case $p_k$ must have been the newly inserted site, and $p_k$ and $p_r$ lie within the circumcircle of $\Delta_{adj}$.

In either case, the charged triangle should be Delaunay before the insertion step that caused its destruction, and $p_r$ lies in the circumcircle of that triangle. Observe also that a triangle $\Delta$ can be charged at most $k$ sites where $k$ is the scope of this triangle. Moreover, a necessary condition for a triangle $\Delta$ to be charged at all is that it arises as a Delaunay triangle at some stage during the incremental construction. Therefore, the total time for the point location is at most,

$$
\begin{aligned}
& O\left(n + \sum_{j=0}^{n-3} \sum_{\Delta \in T_j} j \cdot \text{Prob}(\Delta \text{ appears as a Delaunay triangle})\right) \\
&= O\left(n + 6 \sum_{j=0}^{n-3} \frac{j \cdot \|T_j\|}{(j+1)(j+2)(j+3)}\right) \text{ (according to Lemma 4.5.6)} \\
&= O\left(n + 6 \sum_{j \geq 0} \left[\frac{\|T_{\leq j}\|}{(j+1)(j+2)(j+3)} - \frac{\|T_{\leq j-1}\|}{(j+1)(j+2)(j+3)}\right]\right) \\
&= O\left(n + 12 \sum_{j \geq 0} \frac{\|T_{\leq j}\|}{(j+2)(j+3)(j+4)}\right) \\
&= O\left(n + 12n \cdot \sum_{j=0}^{n} \frac{1}{j+1}\right) \\
&= O(n \log n).
\end{aligned}
$$

□

**Remark 4.5.9.** *We have given here an amortized analysis of the algorithm. There are cases where the running time of the algorithm is $\Theta(n^2)$. Those cases happen when the order of insertion is taken into account. Consider, for example, a set of points $\mathcal{P}$ containing $n$ points such that $n/2$ points lie on the negative portion of the $x - axis$, and $n/2$ on the positive portion of the line $y = 1$ (Figure 4.5). The order of insertion follows the label of each points. In this case, when point $n/2 + k$ is inserted, Many Delaunay triangles will thus arise quadratically.*

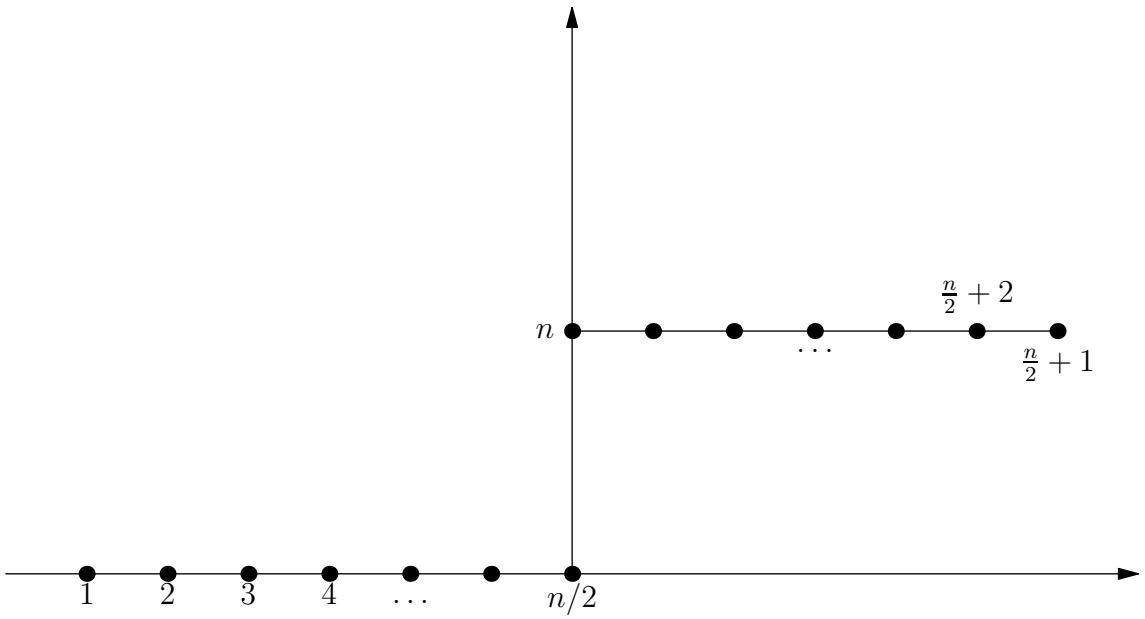



Figure 4.4: Worst case distribution

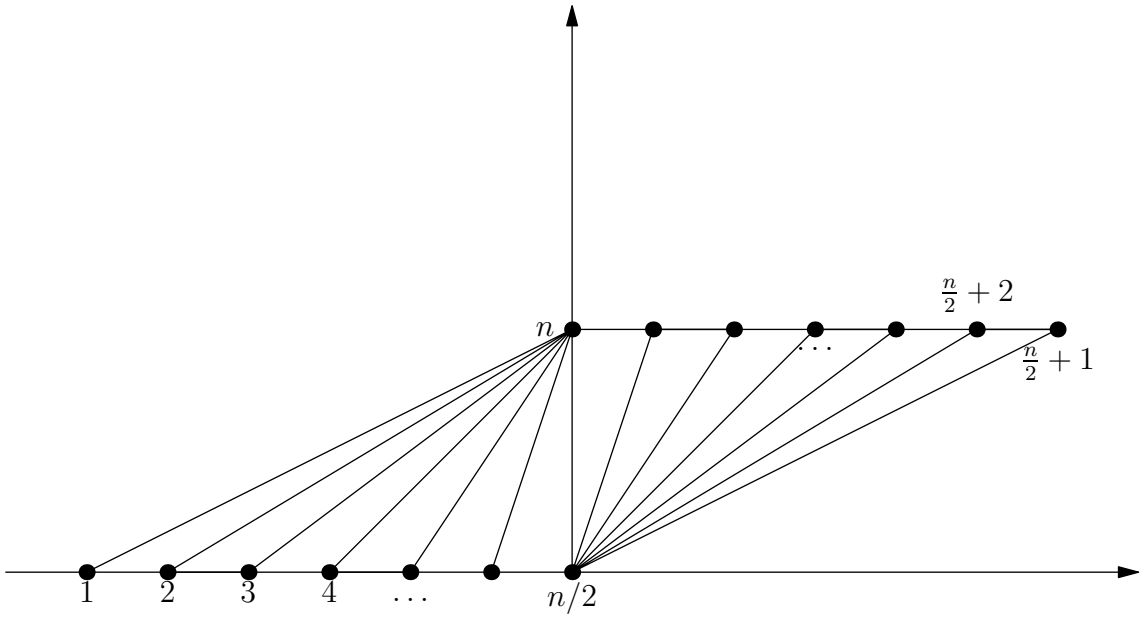



Figure 4.5: The final triangulation

# 5. Applications

The implementation of such optimal algorithms are quite complicated and need a considerable amount of lines of code. Since the edge flips can be done in $O(1)$ time (Theorem 4.5.3), we can, at least, test the speed of the Delaunay tree point location without this procedure. Some results on different prove that the implementation gives good results even for huge number of points. We have chosen the vertices-and-faces-based datastructure of CGAL-python. The last part will introduce the concept of terrain modelling.

### 5.0.1 Vertices-and-Faces-Based Datastructure

The implementation is made with CGAL-Python (`http://cgal-python.gforge.inria.fr/`) using vertices-and-faces-based datastructure. There are other structures which are standard in the field of computational geometry such as the quad-edge data structure of Guibas and Stolfi [10] or the doubly-connected edge list of Müller and Preparata [3]. However, our choice of CGAL is because it can be used with Python and the vertices-and-faces-based datastructure is easy to manipulate.

The triangulation can be seen as a container of faces and vertices maintaining incidence and adjacency relations among them. Each face keeps a pointer on its incident vertices and on its three adjacent faces. Each vertex gives access to one of its incident faces and through that face to the circular list of its incident faces (see Figure 5.1).

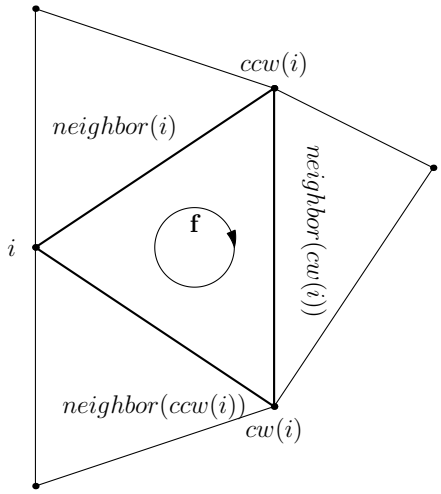



Figure 5.1: Adjacency relation between faces and vertices

### 5.0.2 Statistics

The following results are obtained using CGAL-Python library on Ubuntu 8.04.2, Intel Centrino Core @ Duo T6400 2.0 GHz, 2 MB L2 cache with 4 GB RAM, and interpreted with Ipython. The timing is given by the function `clock`. This test is only for the speed of the Delaunay tree point location on different distributions, which means that the legalization is not included and for simplicity, the case of a point lying on an edge is considered to be inside the tested triangle. The distributions are: random

points inside the unit square, the unit circle, a parabola, an ellipse and a ring. The number lies in the range 500, 5000 and 50000. Since the implementation is made with Python, the speed may be a little bit slower compared with the standards, but the results are satisfactory.

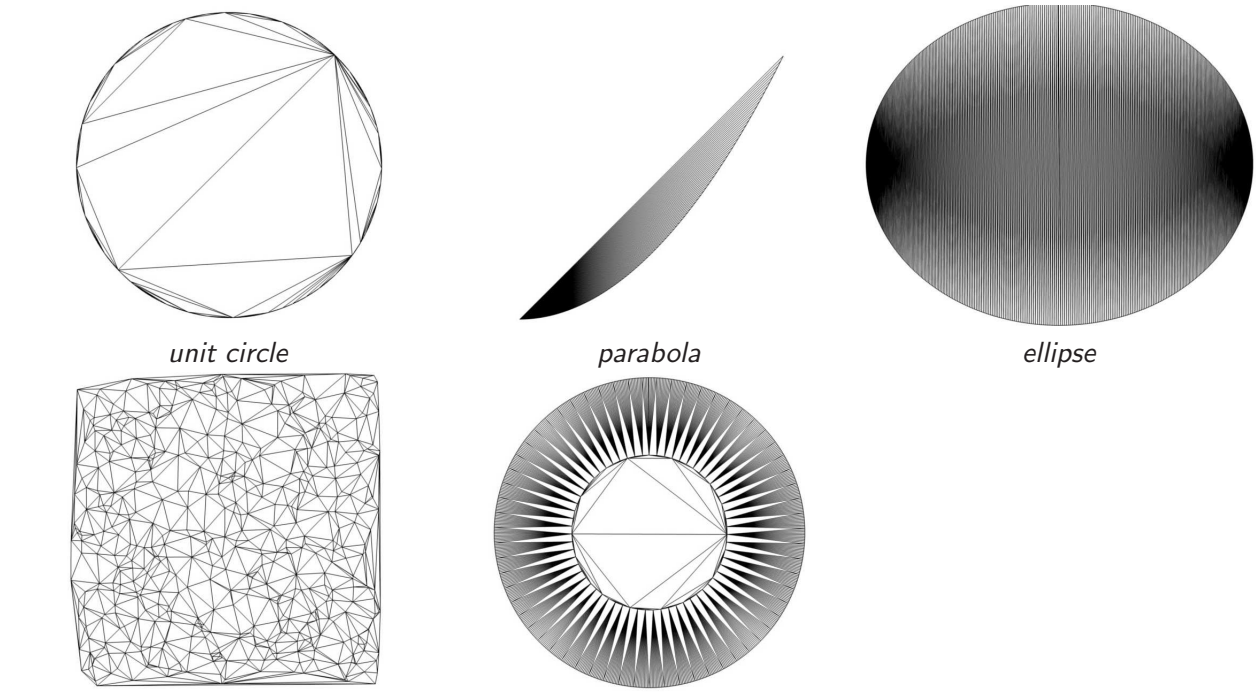



Figure 5.2: Several distributions

| distribution | size | time in s |
|---|---|---|
| random | 500 | 0.26 |
| | 5000 | 2.78 |
| | 50000 | 36.77 |
| circle | 500 | 0.29 |
| | 5000 | 3.61 |
| | 50000 | 52.57 |
| ellipse | 500 | 0.31 |
| | 5000 | 3.95 |
| | 50000 | 50.47 |
| parabola | 500 | 0.26 |
| | 5000 | 3.62 |
| | 50000 | 46.55 |
| ring | 500 | 0.44 |
| | 5000 | 5.67 |
| | 50000 | 119.77 |

Table 5.1: Speed of Delaunay tree on different distributions

## 5.1 Terrain Generation

In this section, we describe one application of the Delaunay triangulation: the perspective view of a terrain. A terrain can be viewed as a 2D surface embedded in 3D space. The procedure is the same as triangulating planar points, but each vertex will be associated with a certain height. The drawing is made using the Open Source software Blender 3D (www.blender.org).

### 5.1.1 What is a Terrain?

A terrain is a 2-dimensional surface in 3-dimensional space with a special property: Every vertical line intersects it at only one point. In other words, it is the graph of a function $f : A \subset \mathbb{R}^2 \to \mathbb{R}$ that assigns a height $f(p)$ to every point $p \in A$ of the terrain [1]. We can model a piece of the earth's surface as a terrain.

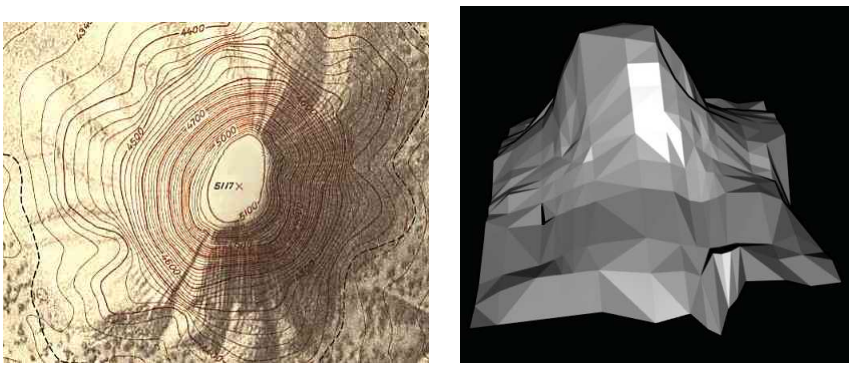
### 5.1.2 The Modelling Technique

Knowing the height of every point on the earth is impossible. This means that when we talk about some terrain, we only know the values of the function $f$ on a finite set $P \subset A$ of sample points. From this sample points, we approximate the height of the other points in the domain. One can think of building the Voronoi diagram of the site, and give to its Voronoi cells the height of the corresponding sites. However, this gives a discrete terrain, which is not a good approximation of the terrain. Hence, the approach is to project each 3D points in the plane, triangulate them with the algorithm and finally, put them at their corresponding heights. We will then get a *polyhedral terrain*, the graph of a continuous function that is piecewise linear, . The polyhedral terrain is used as an approximation of the original terrain [1].

Consider the following topographic map taken from

http://www.compassdude.com/i/topographic-map.jpg.

In Figure 5.3 (b), we draw an approximation of the terrain, having taking $288$ sample points from the map.



(a) topographic map (b) the approximative terrain

Figure 5.3: The procedure of construction

Note that the quality of the approximation is proportional to the number of the sample points. Unfortunately, taking several samples points is costly and not very helpful for those in game programming where generation of terrain should be fast and of good quality. This observation lead to the idea of creating a terrain generator. We will next present a recent technique to generate them.

### 5.1.3 The Terrain Generator

### 5.1.4 Main Idea

The generation of the terrain is based on a very clever idea, where mountains are represented by the two dimensional gaussian,

$$f(x, y) = (-1)^M \times \exp(-\sigma_x(x - \mu_x)^2 - \sigma_y(y - \mu_y)^2). \tag{5.1}$$

Here,

- $M$ is a boolean which decides if the surface is a valley or a mountain;
- $h$ is the height of the mountain;
- $\sigma_x, \sigma_y$ define how spread out the mountain is (it can be interpreted as the inverse of the variance in statistics).
- $\mu_x, \mu_y$ is the centre of the mountain (it is the mean value in statistics);

### 5.1.5 The Process of Construction

We have built a class called `Terrain` which gives access to the following variables:

`nb_centre`: the number of mountains and valleys;

`centre`: array of two rows which defines the coordinate of each mountain;

`etal`: array of two rows which defines how spread out each mountain should be;

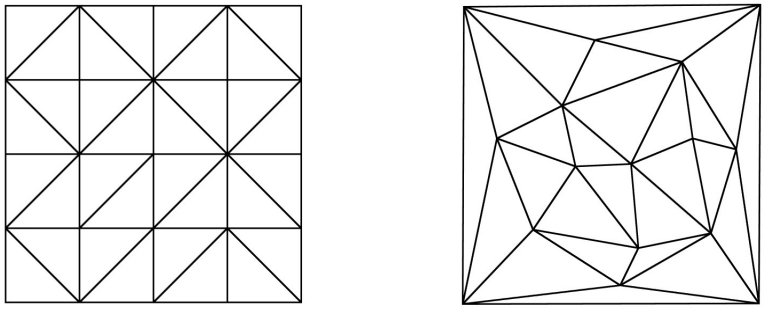
`height`: vector which defines the height of each mountain.

`mountain`: vector formed by booleans

The initial terrain is initialised by the function `Terrain_init` which takes as input`nb_centre` and the dimension of the grid point `xmin,xmax,ymin,ymax`. It returns an initialisation of the terrain by assigning random positions and random heights to each mountain. We then use the function `Mountain_construct` which takes an initialised terrain and returns the final terrain by replacing each mountain by the two dimensional gaussian .

At this stage, each point of the terrain is represented by three coordinates noted $(x, y, h)$ where $h$ is a function of $(x, y)$ generated by `Mountain_construct`. The user defines how many sample points have to be taken for the triangulation. We do not use any location structure, instead we look for the triangle with the bigger area and insert the point in the centroid of this triangle. Thus the running time

of locating the triangle and inserting the point is $O(1)$ when the list of triangles are sorted (in terms of area) in decreased order. The sorting takes $O(n)$ which gives an $O(n^2)$ of the total algorithm. Of course we can use a location structure but the quality of the triangulation is better with the centroid technique [19]. In Figure 5.4 (a), we triangulate a grid of 16 points, the same triangulation is done ni (b), but using the centroid technique. The advantage of (b) is that there are triangles which minimum angle is less than $45^\circ$. Note that at the beginning, we already have a triangulation formed by `xmin,xmax,ymin,ymax`,



(a) grid points (b) centroid technique

Figure 5.4: Quality of triangulation

so that we start the locating process with them. We can summarise the process by the following pseudo-code

**Algorithm 3** Terrain Generation

**Input:** `xmin,xmax,ymin,ymax,nb_centre`,the number of sample points $n$
**Output:** Perspective view of a generated terrain

1: Initialise the terrain with `Terrain_init`
2: Let $DT$ be the Delaunay triangulation formed by `xmin,xmax,ymin,ymax`
3: (*Construct the triangulation of the rectangle formed by `xmin,xmax,ymin,ymax,nb_centre`*)
4: **for** $r \leftarrow 1$ to $n$ **do**
5: **for** each face of $DT$ **do**
6: Take the one which has the maximum area
7: **end for**
8: (*Let $f$ be this face*)
9: Compute the centroid $P$ of $f$
10: Use the process from line 6 to line 11 in Algorithm 2 with $P$ and $f$
11: **end for**
12: Give a height to its point of $DT$ by `Mountain_construct`

### 5.1.6 Visualisation

Below are presented the images of three explicit terrains obtained using the algorithm.

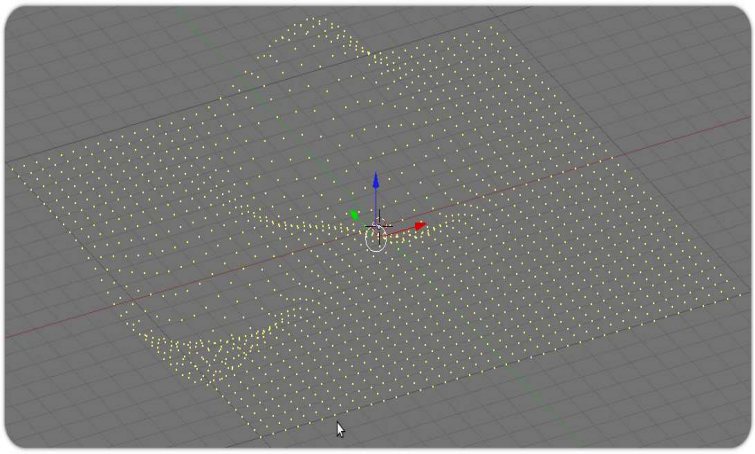


Figure 5.5: The target terrain formed by $1$ mountain and $3$ valleys

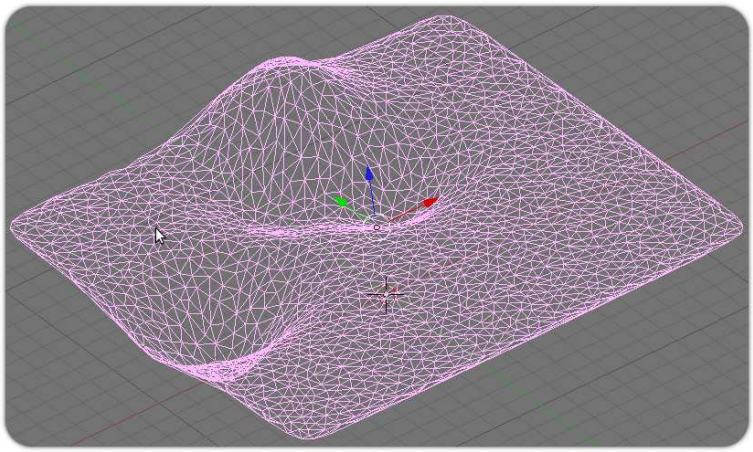


Figure 5.6: The terrain after the Delaunay triangulation

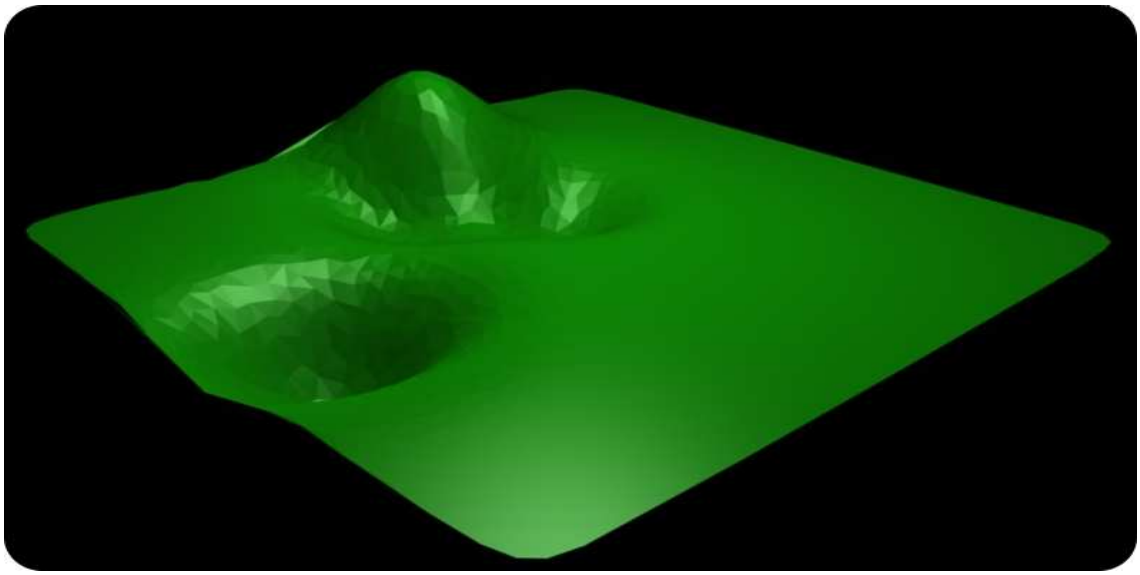


Figure 5.7: The terrain without mesh

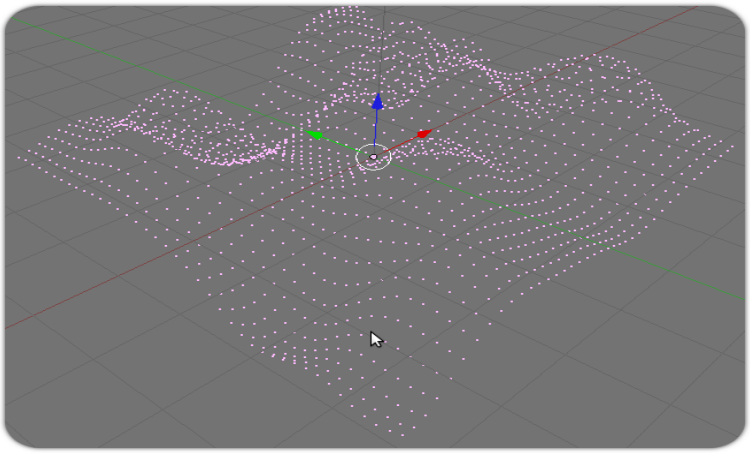


Figure 5.8: The target terrain formed by $20$ mountains and $10$ valleys

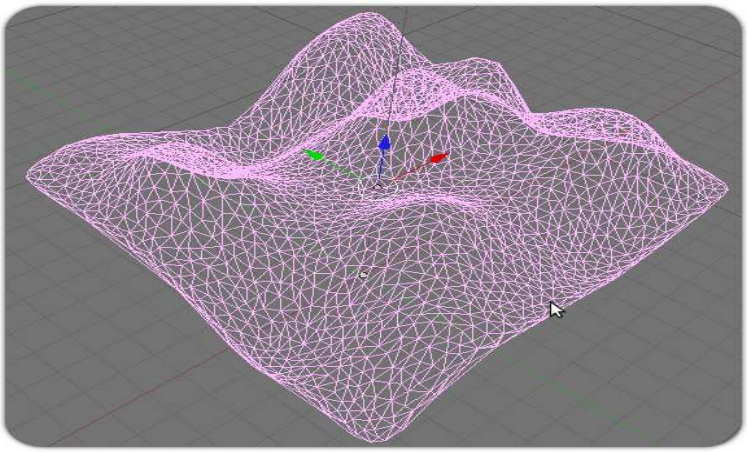


Figure 5.9: The terrain after the Delaunay triangulation

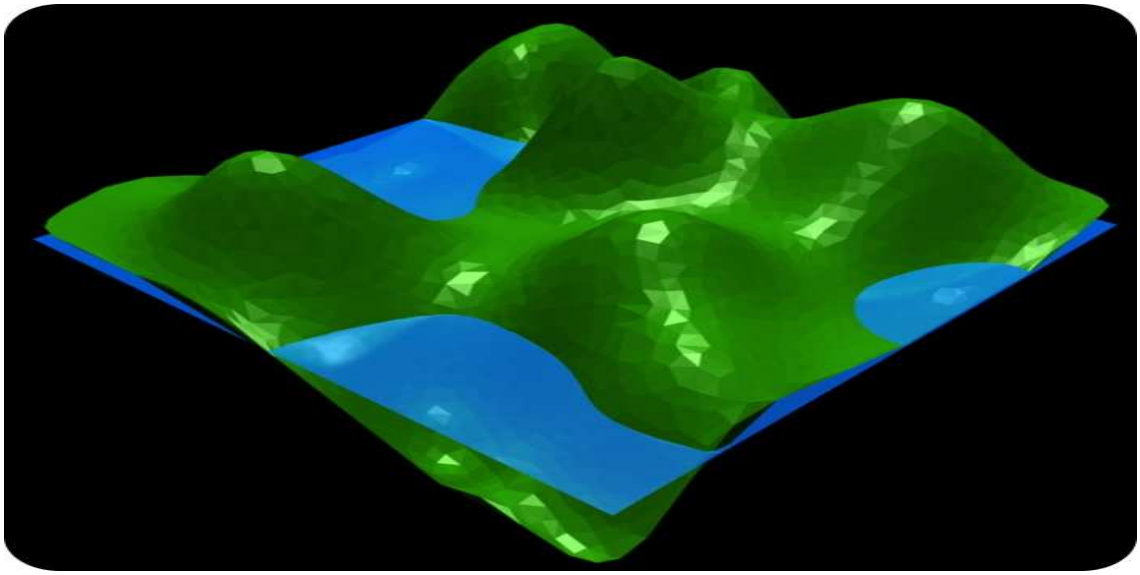


Figure 5.10: The terrain without mesh

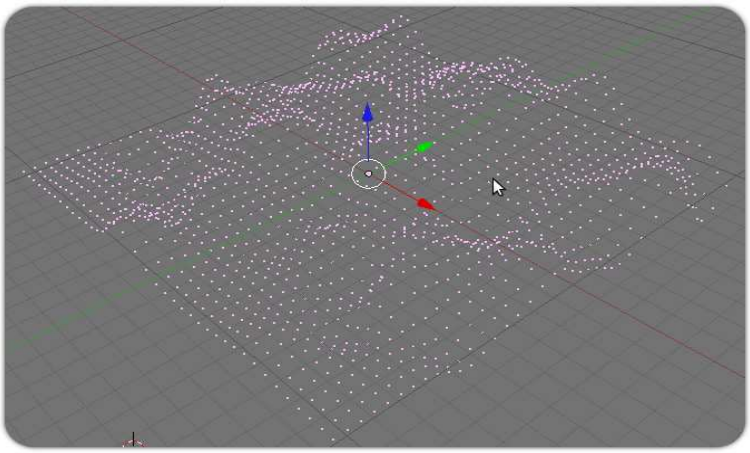


Figure 5.11: The target terrain formed by $50$ mountains and $20$ valleys

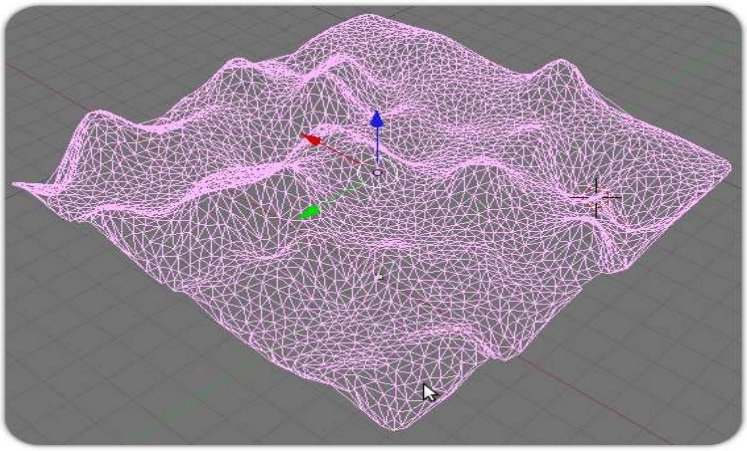


Figure 5.12: The terrain after the Delaunay triangulation

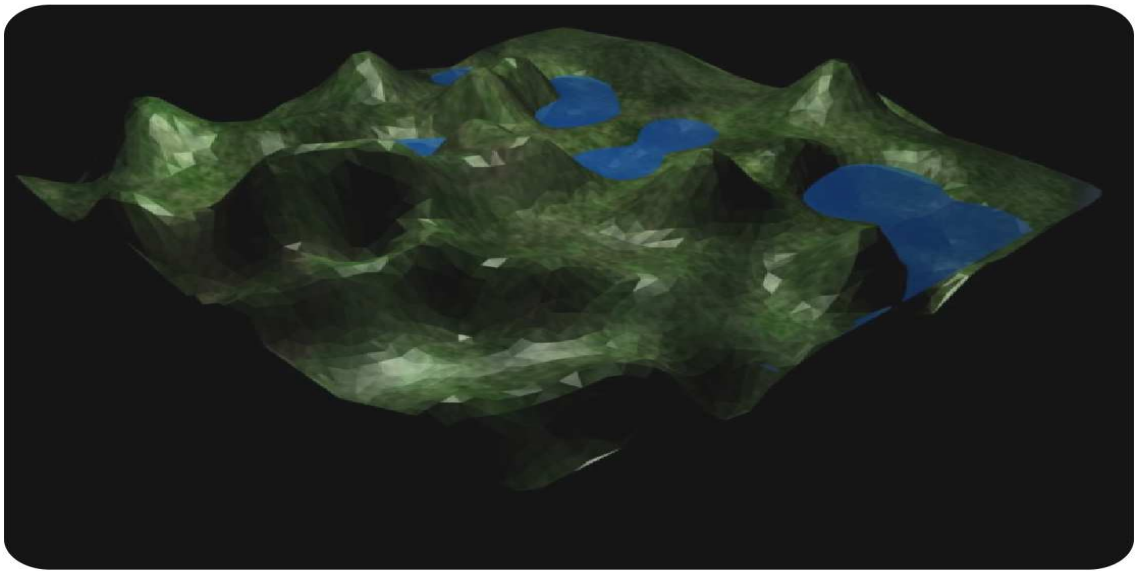


Figure 5.13: The terrain without mesh and using a texture

# 6. Notes and Comments

There are several applications of the Delaunay triangulation which are well known outside the field of computational geometry. Triangulations are used in finite element methods [8], in computer graphics, in image morphing [18] and many more. The randomized incremental algorithm that we have given here is due to Guibas et al. [9], but the analysis of the expected storage has been simplified and the algorithm implemented and tested. The Delaunay tree is not the fastest point location in the class of randomized incremental algorithms. Recently, Olivier Delliers [4] has proposed a new datastructure called the *Delaunay hierarchy* which is based on the nearest neighbor paradigm and uses only $3\%$ memory during the point location compared to the one presented here. Nowadays, the triangulations of a set of points are not sufficient, the reality implies introducing constraints in the input data which are edges and holes. Sometimes, triangulating a set of points with small angles are difficult to obtain in terms of quality, so that extra points, called Steiner points, are added during the triangulation. These new considerations introduce us to the concept of high order constrained Delaunay triangulation which is still a very active area of research in computational geometry.

The terrain generator is based on the idea of Jean Claude Isenman [12] with certain translations into Python. Generating a terrain is one part of the work, yet there are a lot of details to modelise, much more imagination to explore. The research will never end up as long as it imitates the reality, which is the main objective .

All the codes of this project are accessible at

`http://users.aims.ac.za/~faniry/essayproject.html`,

where they can be used freely under the GNU General Public Licence as published by the Free Software Foundation.

# Acknowledgements

This work would have never been achieved without the contribution of Lord Jesus Christ the Son of God, the valuable criticisms of my supervisor Dr Jeff W Sanders, the discussions with my tutor Miangaly Gaelle Andriamaro, the assistance of my family and the encouragement of my AIMS colleagues. I would have not bee able to study at AIMS without the help of Gerard Razafimanantsoa. Finally, I would like to say well done to the AIMS staff for the great project that they are executing.

# References


[1] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[2] Delone, B. N. Sur la sphére vide. *Bul. Acad. Sci. URSS, Class. Sci. Nat.* (1934), 793–800.

[3] D.E.Muller, and Preparata, F. Finding the intersection of two convex polyhedra. *Theoritic. Comput. Sci 7* (1978), 217–236.

[4] Devillers, O. The Delaunay Hierarchy. *Internat. J.Found. Comput. Sci. 13* (2002), 163–180.

[5] Edelsbrunner, H. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[6] E.P., M., and I. Saias I., Z. B. Fast Randomized Point Location Without Preprocessing in Two- and Three-dimensional Delaunay Triangulations. *CompGeom'96, Philadelphia* (1996), 274–283.

[7] Fortune, S. A sweepline algorithm for the Voronoi diagrams. *Algorithmica 2* (1987), 153–174.

[8] George, P.-L., and Borouchaki, H. *Delaunay triangulation and Meshing: application to finite elements*. HERMES, 1998.

[9] Guibas, L. J., Knuth, D. E., and Sharir, M. Randomized Incremental Construction of Delaunay and Voronoi Diagrams. *Algorithmica 7* (1992), 381–413.

[10] Guibas, L. J., and Stolfi, J. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graphics 4* (1985), 74–123.

[11] INRIA. http://cgal-python.gforge.inria.fr/. Computational Geometry Algorithm for Python.

[12] Isenmann, J. C. B. B. Generation de terrain et triangulation de Delaunay. *http://fearyourself.developpez.com/tutoriel/jeu/Delaunay/* (2006).

[13] Lecky-Thompson, and Guy W. Real-Time Realistic Terrain Generation. *Game Programming Gems* (2000), 484–498.

[14] Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 2000.

[15] O'Rourke, J. *Computational Geometry in C: Second Edition*. Cambridge University Press, 1998.

[16] Shankel, and Jason. Fractal Terrain Generation. *Game Programming Gems* (2000), 499–511.

[17] Teillaud, M. *Towards Dynamic Randomized Algorithms in Computational Geometry*. Springer-Verlag, 1993.

[18] Tikhonov, A. Image Morphing. *Computational Photography, Alexei Efros, CMU, Fall* (2007).

[19] Weatherill., N. P. Delaunay Triangulation in Computational Fluid Dynamics. *Computers and Mathematics with Applications* (1992), 129–150.

[20] Woudhouse., F. Terrain Generation Using Fluid Simulation. *http://www.gamedev.net/reference/articles/article2001.asp* (2003).