TRIANGULATIONS IN $\mathbb{R}^2$

HANG SI

Contents

Introduction

Triangulations appear in many different parts of mathematics and computer science since they are the natural way to decompose a region of space into smaller, easy-to-handle pieces. For example, all meshes in this lecture are triangulations of geometric objects which are topological metrical spaces and not necessarily in the Euclidean metric.

This chapter is devoted to the simplest object – triangulations in the plane. We first give the mathematical definitions of triangulations, and then learn how to efficiently construct them.

Typically, given a set of points in space, we would like to connect them in a nice way. The meaning of ?nice? depends on the applications in which the triangulations to be used. Geometrically, one would like to connect the nearest neighbours and to avoid small angles. This leads to the well-known Delaunay triangulations, which have many of these nice properties. We will study both theoretical and efficient algorithms to construct Delaunay triangulations in the plane.

The geometric structure of Delaunay triangulation is better understood by its relation to the convex hull of point sets in one dimensional higher. This leads to a general class of triangulations called weighted Delaunay triangulations in which the Delaunay triangulation is just a special object of it. Such triangulations have many optimal geometric and combinatorial properties. We will prove a fundamental properties about weighted Delaunay triangulation, proven by Edelsbrunner, the acyclic theorem. We will generalise the well-know randomized incremental flip algorithm – to construct weighted Delaunay triangulations and show it is correct for any dimensions.

1. Definitions and Combinatorial Properties

1.1. **Euclidean spaces.** Euclidean space is actually a flat and non-curvy space. In Euclidean geometry, the surface is always assumed to be flat. If it become curved or spherical, then it comes under non-Euclidean geometry. An example of two-dimensional Euclidean space is a piece of paper.

One way to think of a k-dimensional Euclidean space E^k is as a set of k-tuples $(x_1, \cdots, x_k)$ of real numbers $x_i, 1 < i < k$, called *points* satisfying certain relationships, expressible in terms of distance and angle. This distance function is called the *Euclidean metric*. It makes the Euclidean space a metric space. It is sufficient to define the Euclidean geometry.

We will use the standard way to define the Euclidean space as a *k-dimensional real vector space* $\mathbb{R}^k$ equipped with an inner product $\|\cdot\|$. Note that a Euclidean space is not technically a vector space but rather an affine space, on which a vector space acts by translations. There is no canonical choice of the origin in the space.

The 2-dimensional Euclidean space is called the *Euclidean plane*. The position of a point p in a Euclidean plane (E^2) is a two-dimensional real vector. The *Euclidean norm* (or magnitude) of a vector $p = (x, y)$ is defined as

$$\|p\| = \sqrt{x^2 + y^2}.$$

The *distance* between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the Euclidean plane is the *Eucdliean length* of the vector $p_1 - p_2$,

$$\mathrm{d}(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

The *angle* θ between two edges ab and ac in the Euclidean plane can be obtained through the Law of Cosine

$$\theta = \arccos \frac{\mathrm{d}(a,b)^2 + \mathrm{d}(a,c)^2 - \mathrm{d}(b,c)^2}{2 * \mathrm{d}(a,b) * \mathrm{d}(a,c)}.$$

We use the Euclidean distance function $\|\cdot\|$ to define an *open d-ball*,

$$\mathbb{B}^d(\mathbf{p}, r) = \{\mathbf{x} \in \mathbb{R}^d \,|\, \|\mathbf{x} - \mathbf{p}\| < r\},$$

as the set of all points closer than a given distance r from a given point $\mathbf{p}$. The *natural topology* of $\mathbb{R}^d$ is a system of open sets, where each open set is a union of open balls. An *open set* in $\mathbb{R}^d$ is any union of open balls in $\mathbb{R}^d$. Obviously, any finite intersection of a set of open balls is also an open ball (which may be an empty ball $\mathbb{B}^d(\mathbf{p}, 0)$).

1.2. **Simplicial complexes.** We use simpical complex as the fundamental tool to model geometric shapes and spaces. Because of their combinatorial nature, simplical complexes are perfect data structure for geometric modelling algorithms.

A point set in $\mathbb{R}^d$ is *convex* if with any two points $\mathbf{v}_i, \mathbf{v}_j$ it also contains the straight line segment $[\mathbf{v}_i, \mathbf{v}_j] = \{(1-\lambda)\mathbf{v}_i + \lambda\mathbf{v}_j \ : \ 0 \leq \lambda \leq 1\}$. The *convex hull* of a not necessarily convex set V, denoted as $\text{conv}(V)$, is the smallest convex set containing V.

$$\text{conv}(V) = \left\{ \sum_{i=0}^{k} \lambda_i \mathbf{v}_i \;\middle|\; \mathbf{v}_i \in V, \lambda_i \in \mathbb{R}, \lambda_i \geq 0, \sum_{i=0}^{k} \lambda_i = 1 \right\}.$$

Let V be a finite set of $k+1$ points. The *affine hull* of V, $\text{aff}(V)$, is the smallest affine subspace that contains V, i.e.,

$$\text{aff}(V) = \left\{ \sum_{i=0}^{k} \lambda_i \mathbf{v}_i \;\middle|\; \mathbf{v}_i \in V, \lambda_i \in \mathbb{R}, \sum_{i=0}^{k} \lambda_i = 1 \right\}.$$

The *dimension* of $\text{aff}(V)$ is the dimension of the corresponding linear subspace. A set of $k+1$ points $\mathbf{v}_0, ..., \mathbf{v}_k \in \mathbb{R}^d$ is *affinely independent* if its affine hull has dimension k.

A *geometric k-simplex* σ in $\mathbb{R}^d$ is the convex hull of a collection of $k+1$ affinely independent points in $\mathbb{R}^d$. The *dimension* of σ is $\dim(\sigma) = k$. For example, the (-1)-simplex is the empty set $(\emptyset)$, a 0-simplex is a vertex (or point), a 1-simplex is an edge, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron, see Figure 1.

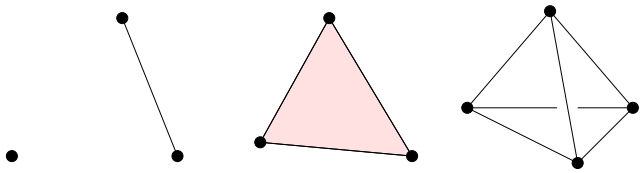


FIGURE 1. From left to right are a 0-simplex (a point), a 1-simplex (an edge), a 2-simplex (a triangle), and a 3-simplex (a tetrahedron).

A *face* τ of σ is the convex hull of any subset of the vertices of σ. It is again a simplex. $\tau = \emptyset$ and $\tau = \sigma$ are the two trivial faces of σ, all others are *proper* faces of σ. We write $\tau \leq \sigma$ or $\tau < \sigma$ if τ is a face or a proper face of σ. The number of l-faces of σ is equal to the number of ways we can choose $l+1$ from $k+1$ points, which is $\binom{k+1}{l+1}$. The total number of faces is

$$\sum_{l=-1}^{k} \binom{k+1}{l+1} = 2^{k+1}.$$

The union of all proper faces of a simplex σ is called the *boundary* $\text{bd}(\sigma)$ of σ. The *interior* $\text{int}(\sigma)$ of σ is $\sigma - \text{bd}(\sigma)$, see Figure 2. Note that a point (a 0-simplex) has no proper face, i.e., the boundary of a point is $\emptyset$. Therefore the interior of a point is the point itself.

A *geometric simplicial complex* $\mathcal{K}$ in $\mathbb{R}^d$ is a finite collection of simplices, such that any two simplices are either disjoint or meet in a common face which is also in $\mathcal{K}$. More

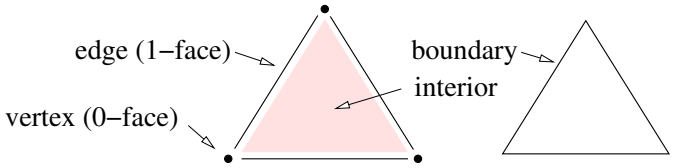
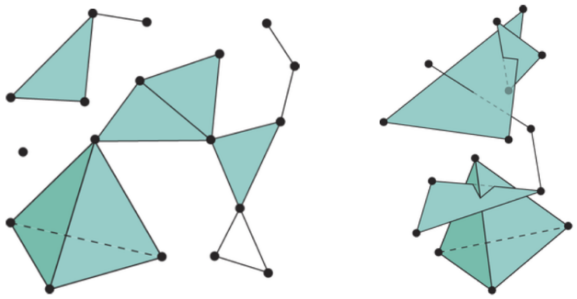


FIGURE 2. The boundary and interior of a 2-simplex (a triangle).

formally,

(i) any face of a simplex $\sigma \in \mathcal{K}$ is also in $\mathcal{K}$, and
(ii) the intersection of any two simplices $\sigma, \tau \in \mathcal{K}$ is a face of both σ and τ.

The first property implies $\emptyset \in \mathcal{K}$. The second property implies that any two different simplices in $\mathcal{K}$ have disjoint interiors, i.e., $\text{int}(\sigma) \cap \text{int}(\tau) = \emptyset$. The *dimension* $\dim(\mathcal{K})$ of $\mathcal{K}$ is the largest dimension of a simplex of $\mathcal{K}$. The *vertex set* $\text{vert}(\mathcal{K})$ of $\mathcal{K}$ is the set of all vertices of $\mathcal{K}$. Without the second property, $\mathcal{K}$ is an *abstract simplical complex*. Figure 3 illustrates a 3-dimensional geometric simplical complex and an abstract simplical complex, respectively.



(1) (2)

FIGURE 3. (1) A three-dimensional geometric simplical complex. (2) An abstarct simplical complex. (Figures from Wikipedia)

A *subcomplex* is a subset of $\mathcal{K}$ that is a simplicial complex itself. Observe that every subset of a simplicial complex satisfies Condition (ii). To enforce Condition (i), we add faces and simplices to the subset. The *closure* of a subset $\mathcal{L} \subset \mathcal{K}$ is the smallest subcomplex that contains $\mathcal{L}$,

$$\text{Cl}\, \mathcal{L} = \{\tau \in \mathcal{K} \mid \tau \in \mathcal{L}\}.$$

A particular subcomplex is the *i-skeleton* $\mathcal{K}^{(i)}$ of $\mathcal{K}$, which consists of all simplices $\sigma \in \mathcal{K}$ whose dimension is i or less. Hence $\mathcal{K}^{(0)} = \text{vert}(\mathcal{K}) \cup \emptyset$.

1.2.1. *Stars and links.* We use special subsets to talk about the local structure of a simplical complex. These subsets may or may not be closed. The *star* of a complex τ consists of all simplices that contain τ, and the *link* consists of all faces of simplices in

the star that do not intersect τ, i.e.,

$$\begin{aligned} \operatorname{St}\tau &= \{\sigma \in K \mid \tau \leq \sigma\} \\ \operatorname{Lk}\tau &= \{\sigma \in \operatorname{Cl}\operatorname{St}\tau \mid \tau \cap \sigma = \emptyset\} \end{aligned}$$

Figure 4 illustrates these two definitions. Note that the star of τ is not a sub complex (check it), while the link of τ is (validate it).

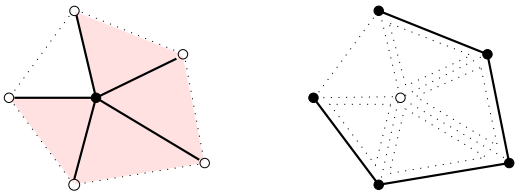


FIGURE 4. Star and link of a vertex. Left: the solid edges and the shaded triangles belong to the star of the solid vertex. Right: the solid edges and vertices belong to the link of the hollow vertex.

1.2.2. *Underlying spaces.* Let $\mathcal{K}$ be a simplicial complex. The *underlying space* $|\mathcal{K}|$ of $\mathcal{K}$ is the union of all simplices of $\mathcal{K}$, i.e., $|\mathcal{K}| = \bigcup_{\sigma\in\mathcal{K}} \sigma$. Note that the underlying space of any (geometric) simplical complex is a geometric domain (such as a polygon in the plane or a polyhedron in 3d). This domain consists of a point set which are from the simplices of the given simplicial complex.

We can give each simplex its natural topology as a subspace of $\mathbb{R}^d$. We can then give $|\mathcal{K}|$ a natural topology defined as: a subset A of $|\mathcal{K}|$ is closed iff $A \cap \sigma$ is closed for any $\sigma \in \mathcal{K}$. Alternatively, if $\mathcal{K}$ is finite, since $|\mathcal{K}|$ is embedded, we can consider the subspace topology on $\mathcal{K}$ induced from $\mathbb{R}^d$. Note that these two topologies are the same only if $\mathcal{K}$ is finite. Otherwise, the first one is finer and more general than the second one. However, in this class, we will talk about only finite simplicial complexes. Hence from now on we consider the underlying space $|\mathcal{K}|$ of any finite simplicial complex equipped with the natural induced homology form $\mathbb{R}^d$ (where K embeds into)

1.3. **Triangulations of point sets.** There are many possible simplicial complexes that have the same underlying space. A simplicial complex gives a combinatorial structure on its underlying space. It is then a *triangulation* of that space. We will use this idea to define triangulations of geometric objects, such as point sets and polygonal domains.

Let S be a finite point set in the plane, a *triangulation* of S is a 2-dimensional simplical complex $\mathcal{T}$ such that

(i) the vertices of $\mathcal{T}$ are in S; and
(ii) the underlying space $|\mathcal{T}|$ is the convex hull of S.

Figure 1.3 shows some triangulations of a point set. Note a triangulation does not need to use all vertices of the point set.

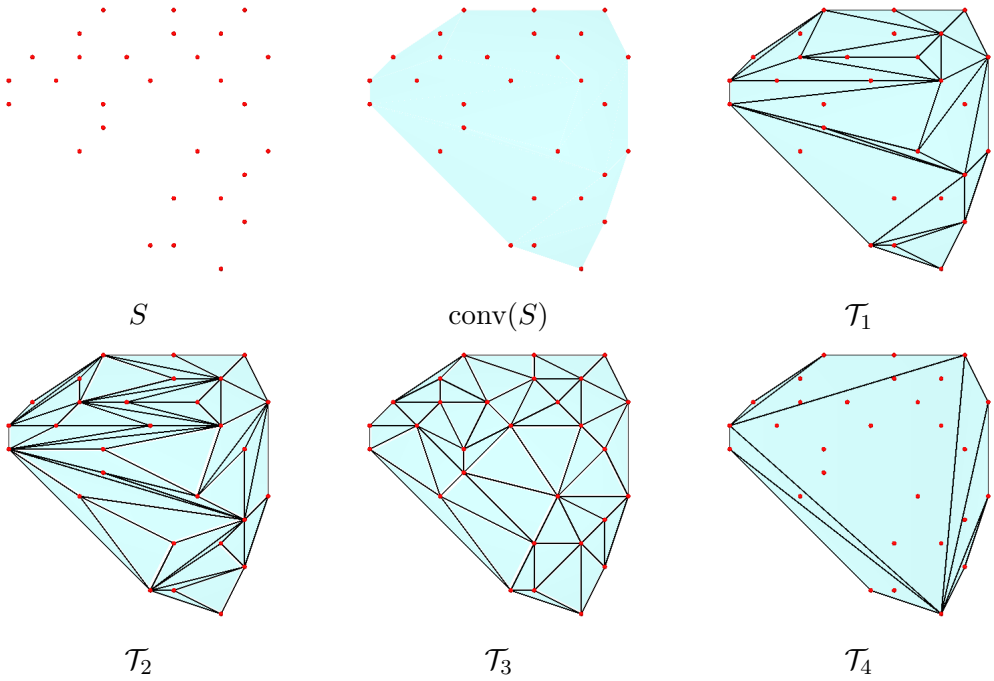


FIGURE 5. A 2d point set S, the convex hull $\mathrm{conv}(S)$, and different triangulations $\mathcal{T}_1, \ldots, \mathcal{T}_4$ of S.

1.4. **Planar graphs and Euler's formula.** A triangulation in the plane is a planar graph, but it may not be maximally connected. The face outside the convex hull of this triangulation is a k-polygon, and $k \geq 3$. Using the Euler formula, we can get upper bounds on the number of edges and faces of a triangulation in terms of the number of vertices of its vertex set.

A *graph* $\mathcal{G} = (V, E)$ is a set V of vertices, and a set E of edges, each a pair of vertices of V. A graph is *simple* if every edge has two distinct vertices and no two edges have the same vertices. A simple graph is *connected* if there is a *path* (a sequence of edges) that connecting every pair of its vertices. The smallest connect graph are *trees*, which are characterised by having a unique simple path between every pair of vertices. Removing any one edge disconnects the graph. A *spanning tree* of $\mathcal{G} = (V, E)$ is a tree (V, T) with $T \subset E$. It has the same vertex set as the graph and uses a minimal set of edges necessary to be connected. A graph is connected if and only if it has a spanning tree.

There are many drawings of a graph, some with and some without crossings. As illustrated in 6. A graph is *planar* if it has an embedding in the plane without crossing edges. Only graphs with relatively few edges can be drawn without crossings in the plane.

Let $\mathcal{G} = (V, E)$ be a planar graph. It decomposes the plane into a set of regions, which are called *faces* of $\mathcal{G}$. Let v, e, and f denote the number of vertices, edges, and faces of $\mathcal{G}$. Euler formula is a linear relation between these numbers.

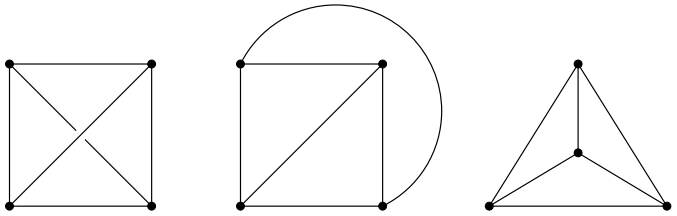


FIGURE 6. From left to right: a drawing that is not an embedding, and embedding with one curved edge, and a straight-line embedding.

Theorem 1.1 (Euler Formula)**.** *Every connected planar graph* $\mathcal{G} = (V, E)$ *satisfies*

$$v - e + f = 2.$$

This formula is well-known as the Euler formula for planar graphs and convex 3d polytopes. There are plenty of proofs, see a collection of different proofs of this formula by D. Eppstein's "Geometry Junkyard" [1]. Here we show a proof based on the dual graph and spanning trees.

Proof. Let $\mathcal{G} = (V, E)$ be a connected planar graph. Define the *dual graph* $\mathcal{G}^* = (V^*, E^*)$ of $\mathcal{G}$, such that every vertex in V^* corresponds to a face of $\mathcal{G}$, and every edge in E^* corresponds to two adjacent faces of $\mathcal{G}$. Let F be the set of faces of $\mathcal{G}$, V^* and F are bijective, and E^* and E are bijective, see Figure 7.

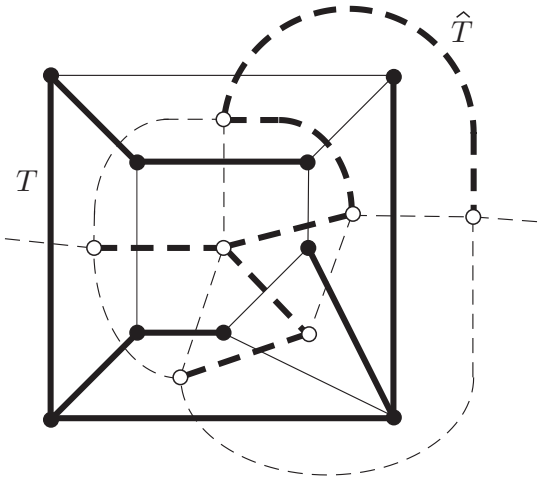


FIGURE 7. A proof of Euler formula using dual graph and spanning trees (Figure from D. Eppstein).

Choose any spanning tree $\mathcal{T}$ of $\mathcal{G}$. It has v vertices, and $v-1$ edges. The dual edges of $(\mathcal{G}-\mathcal{T})^*$ is also a spanning tree of $\mathcal{G}^*$. The two spanning trees together have $v-1+f-1$ edges, i.e., $v - 1 + f - 1 = e$, which the above formula follows. □

[1] `https://www.ics.uci.edu/~eppstein/junkyard/euler/`

Euler formula is a topological and combinatorial property of that graph. It can be used to show many interesting properties of planar graphs as well as three-dimensional convex polytopes.

A triangulation is a planar graph, but it may not be maximally connected. The face outside the convex hull of this triangulation is a k-polygon, and $k \geq 3$. Using the Euler formula, we can get upper bounds on the number of edges and faces of a triangulation in terms of the number of vertices of its vertex set.

Let n, e, and f be the number of vertices, edges, and triangles of a triangulation $\mathcal{T}$ of a point set S. In a triangulation $\mathcal{T}$, every triangle have three edges, every interior edge is shared by two faces, and every convex hull edge is shared by one face, we then have:

$$3f = 2e + k,$$

where k is the number of edges on the convex hull of $\mathcal{T}$. Since $k > 0$, then,

$$3f < 2e.$$

Using the Euler formula and the above inequality, we can bound the total number of edges and faces of $\mathcal{T}$, which are

$$\begin{aligned} e &< 3n - 6, \\ f &< 2n - 4. \end{aligned}$$

The *degree* (or *valency*) of a vertex u in a planar graph $\mathcal{G}$ is the number of edges at this vertex. Every edge of $\mathcal{G}$ has two distinct vertices (endpoints). The sum of vertex degrees is twice the number of edges, which must less than $6n - 12$. It follows that every planar graph has a vertex whose degree is less than 6.

1.4.1. *Euler Characteristic.* If a planar graph is not connected, i.e., it has more than one connected component. The Euler formula does not hold anymore. In general, the *Euler characteristic* of a d-dimensional simplical complex $\mathcal{K}$ is the alternating sum of the number of simplices

$$\chi = s_0 - s_1 + s_2 - \cdots + (-1)^d s_d,$$

where d is the dimension of $\mathcal{K}$ and s_i is the number of i-simplices in $\mathcal{K}$. It is common to omit the $(-1) - simplex$ from the sum.

Let $\mathcal{K}$ be a two-dimensional simplicial complex, $\chi = v - e + f$. We have seen that for any simply connected planar graph, $\chi = 2$. However, if a planar graph is not simply connected, i.e., it has more than one connected component, then the right hand side of the Euler formula is not necessarily 2 anymore.

2. Incremental Construction

In this section, we introduce a common and simple approach for building geometric structures called *incremental construction*. In such an algorithm objects are added one at a time and the structure is updated with each new insertion.

2.1. **Convex hull construction.** We will begin with a presentation of how to incrementally create the convex hull of a given point set. Then we show that triangulations are just byproducts of the convex hull construction.

Recall that the *convex hull* of a set of points is the smallest convex set that contains the points. The convex hull of a two-dimensional point set is a simple polygon. The graph of a simple polygon is a cyclic sequence of vertices (or edges). It can be represented by a doubly-linked data structure.

A common approach for building geometric structures is called *incremental construction*, that is objects (points) are added one at a time, and the structure (convex hull) is updated with each new insertion.

The basic idea of an incremental convex hull algorithm is that points (objects) are added one at a time, and the convex hull (geometric structure) is updated with each new insertion. A new point is processed in three steps.

1 Locate the *visible* edges of the convex hull for the point. If the point lies in the interior of the convex hull, there is no visible edge, and one can simply skip this point, and go to process the next one.
2 If the point lies outside of the current convex hull, construct a cone from the new point to all of its visible edges, see Figure 2.1 Left.
3 Delete the visible edges in Step 2, thus forming the convex hull with the new point and the previously processed points, see Figure 2.1 Right.

The basic algorithm is given in 9.

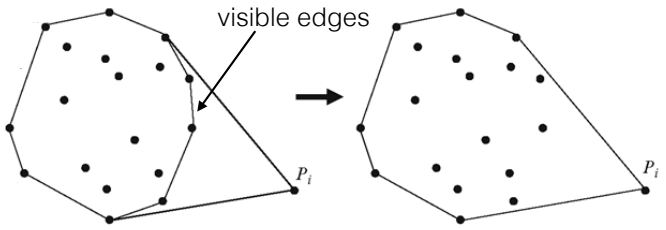


FIGURE 8. Incremental construction of convex hull.

Algorithm: IncrementalConvexHull(S)
Input: $S = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ is a set of n points in the plane;
Output: The convex hull of S;
1 initialize one triangle $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$;
2 **for** $i = 4$ to n **do**
3 **if** $\mathbf{p}_i$ lies outside conv$(\mathbf{p}_1, \ldots, \mathbf{p}_{i-1})$; **then**
4 let Q be the set of all visible edges of $\mathbf{p}_i$;
5 create a cone formed by $\mathbf{p}_i$ and edges in Q;
6 delete edges in Q;
7 **endif**
8 **endfor**

FIGURE 9. The incremental convex hull algorithm.

Remark. For the initial triangle to be valid, it is necessary to assume the general position, i.e., $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ are not collinear.

2.1.1. *The* `orient2d` *test.* The geometric test for visible edges is a very important predicates `orient2d` which takes three ordered points $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$ in the plane, and test whether they follow a counterclockwise or clockwise direction, see Figure 2.1.1. Note the orientation depends on the order in which the points are given.

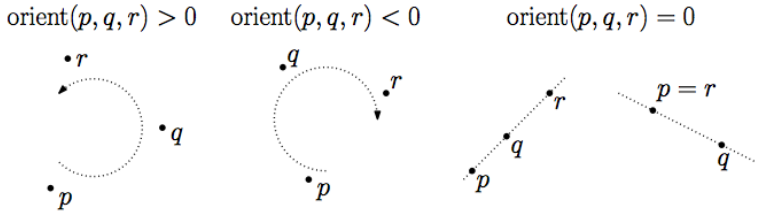


FIGURE 10. Orientations of three points in the plane (Figures by D. Mount).

Orientation is formally defined as the sign of the determinant of the points given in homogeneous coordinates, i.e.,

$$\texttt{orient2d}(\mathbf{p}, \mathbf{q}, \mathbf{r}) = \text{sign}(\det(A)),$$

where

$$A = \begin{bmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{bmatrix}$$

The result is positive if $\mathbf{p}, \mathbf{q}, \mathbf{r}$ follow a counterclockwise turn, it is negative they follow a clockwise turn. It is zero if the three vertices are collinear. Note that the result of $\det(A)$ is twice of the signed area of the triangle defined by the three points.

Remark. If two rows (or two columns) of A in above are interchanged to produce a matrix, B, then: $\det(A) = -\det(B)$. The signs of the above `orient2d()` test must be also inverted.

2.1.2. *Running time.* The complexity of this algorithm can be estimated in terms of n, the total number of points, and h, the number of points (edges) on the convex hull.

Since every step it needs to check whether the new point lies inside or outside the convex hull, which is h, the running time is clearly $O(nh)$. If h is small compared to n, for instance, it is bounded by a constant, then this algorithm runs in linear time $O(n)$. If h is large, such that $h = \Omega(n)$, then the time complexity of this algorithm is $O(n^2)$.

We can clearly, improve this algorithm by presorting the given set S. A simple way is to sort the point set along a fixed direction (for example, the x-axis), then use a (vertical) line that sweeps over the plane from left to right. This guarantees that each newly added point is outside the current hull. To efficiently obtain the set Q (in line 4), one could start from the last newly created hull edge e. Then the set of all hull edges which are visible by $\mathbf{p}_i$ can be collected by a breadth search from e. The sort of a set of vertices along the sweep line can be done in time $O(n \log n)$. The number of hull edges which are visible by each $\mathbf{p}_i$ is a constant independent of n. Thus this incremental algorithm with a line sweep sorting constructs the convex hull in $O(n \log n)$ time.

2.2. **A line-sweep triangulation algorithm.** By slightly modifying the incremental convex hull algorithm in Figure 9 one can obtain a triangulation of S as well. Instead of creating a cone (in line 5) and deleting the visible edges (in line 6), we create triangles by joining each visible edge in Q and the point $\mathbf{p}_i$. This resulting a triangulation (instead of the convex hull) of $\{\mathbf{p}_1, \ldots, \mathbf{p}_i\}$.

We then have a simple and efficient algorithm using line-sweeping [9] to construct triangulations for a set of points. The basic idea is to sort the point set along a fixed direction (for example, the x-axis), then use a (vertical) line that sweeps over the plane from left to right. The triangulation is created online during the line sweeping. An invariant is: at any moment in time, the partial triangulation contains all points to the left of the line. When the line hits a new vertex (an event), the triangulation is augmented by creating new triangles connecting to this new vertex. The algorithm is given in Figure 11.

Algorithm: LineSweep(S, $\mathbf{s}$)
Input: A set S of n points in $\mathbb{R}^2$, $\mathbf{s}$ is the normal of sweep line;
Output: A triangulation $\mathcal{T}$ of S;
1 sort the points in S into a sequence $L := \{\mathbf{p}_1, \ldots \mathbf{p}_n\}$ along $\mathbf{s}$;
2 initialize $\mathcal{T}$ with only one triangle $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$;
3 **for** $i = 4$ to n **do**
4 let Q be the set of all visible edges of $\mathbf{p}_i$;
5 create new triangles to $\mathcal{T}$ by each edge in Q and $\mathbf{p}_i$;
6 **endfor**

FIGURE 11. The sweep line triangulation algorithm.

In line 1, the vertices in L are ordered in such a way, that no conflict will occur during the algorithm. A simple order is the lexicographic (dictionary) order along the x- or y-axis. For the initial triangle to be valid, it is necessary to assume the general position, i.e., $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ are not collinear. To efficiently obtain the set Q (in line 4), one could start from the last newly created triangle, which must contain a hull edge e. Then the set of all visible edges of $\mathbf{p}_i$ can be collected by a breadth search from e. Figure 12 illustrates an example of this algorithm.

The sort of a set of vertices along the sweep line can be done in time $O(n \log n)$. The number of visible edges of $\mathbf{p}_i$ is a constant independent of n. The total number of newly created triangles is less than $2n - 4$. Thus this line sweep algorithm constructs a triangulation in $O(n \log n)$ time.

3. Delaunay Triangulations

This section introduces the Delaunay triangulation of any point set in the plane. It is introduced by the Russian mathematician Boris Nikolaevich Delone (1890–1980) in 1934 [3]. It is a triangulation with many nice properties. There are many ways to define Delaunay triangulations. We first introduce them as duals of Voronoi diagrams, then introduce other equivalent definitions while showing their properties. We discuss efficient and simple algorithms to compute Delaunay triangulations.

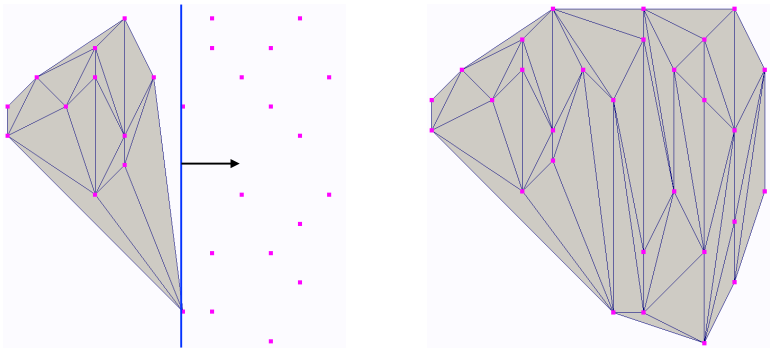


FIGURE 12. The line-sweep algorithm yo construct a triangulation.

3.1. **Voronoi diagrams.** Voronoi diagrams are named after the Russian and Ukrainian mathematician Georgy Feodosevich Voronoy (1868–1908) in 1907 [11]. It is also known as Dirichlet tessellations (after German mathematician Peter Gustav Lejeune Dirichlet (1805 – 1859)). Voronoi diagrams arise in nature in various situations. They are one of the most fundamental data structures in computational geometry.

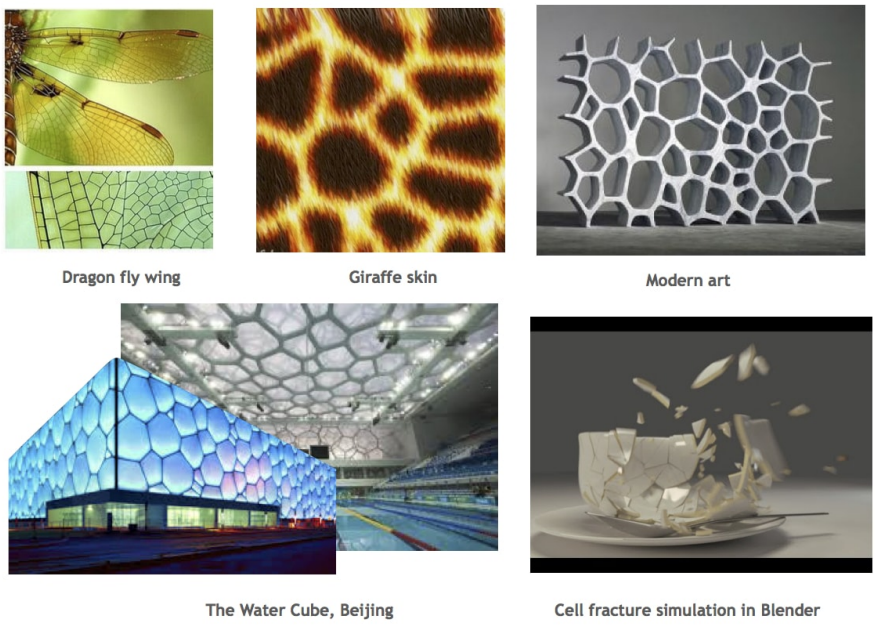


FIGURE 13. Voronoi diagrams.

Voronoi diagram divides the plane according to the *nearest-neighbour rule*: Each point is associated with region of the plane closet to it.

Consider the simplest case of two points $\mathbf{p}$ and $\mathbf{q}$ in the plane. The set of points that are at least as close to $\mathbf{p}$ as to $\mathbf{q}$ is the *half-space*:

$$H_{pq} = \{\mathbf{x} \in \mathbb{R}^2 \,:\, \|\mathbf{x} - \mathbf{p}\| \leq \|\mathbf{x} - \mathbf{q}\|\},$$

where $\|\cdot\|$ means Euclidean distance.

Let S be a set of n points (called *sites*) in $\mathbb{R}^2$. the *Voronoi region* of a site $\mathbf{p} \in S$ is the set of points $\mathbf{x} \in \mathbb{R}^2$ that are as close to $\mathbf{p}$ as to any other site in S, that is

$$V_p = \{\mathbf{x} \in \mathbb{R}^2 \,|\, \|\mathbf{x} - \mathbf{p}\| \leq \|\mathbf{x} - \mathbf{q}\|, \forall \mathbf{q} \in S\},$$

where $\|\cdot\|$ means Euclidean distance.

The Voronoi region of $\mathbf{p}$ is the intersection of a set of half-spaces of points at least at close to $\mathbf{p}$ as to $\mathbf{q}$, $H_{pq} = \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x} - \mathbf{p}\| \leq \|\mathbf{x} - \mathbf{q}\|\}$. It follows that it is a convex polygonal region, possibly unbounded, with at most $n - 1$ edges.

Each point $\mathbf{x} \in \mathbb{R}^2$ has at least one nearest site in S, so it belongs to at least one Voronoi region. It follows that the Voronoi regions cover the entire plane. Two Voronoi regions lie on opposite sides of the perpendicular bisector separating the two generating points. The Voronoi regions together with their edges and vertices form the *Voronoi diagram* of S [11], see Figure 14 Left.

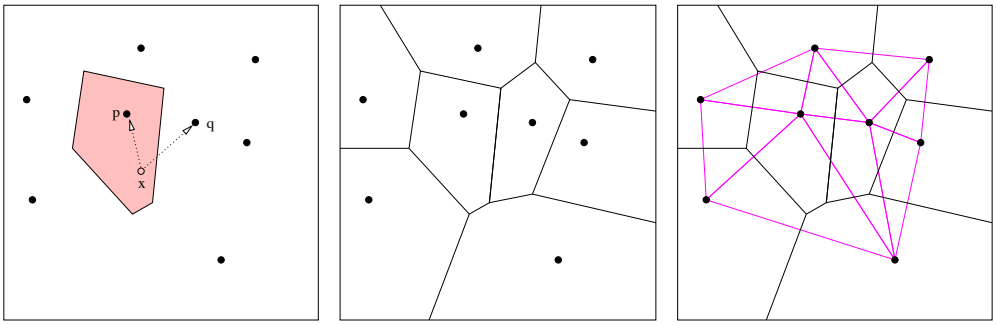


FIGURE 14. From left to right, a Voronoi cell, the Voronoi diagram and the dual Delaunay triangulation.

Voronoi diagrams have found numerous applications in Natural sciences and engineering. We just mention one case in geometry. A point location data structure can be built on top of the Voronoi diagram in order to answer nearest neighbor queries, where one wants to find the object that is closest to a given query point. Nearest neighbor queries have numerous applications. For example, one might want to find the nearest hospital, or the most similar object in a database.

A 2-dimensional Voronoi diagram is a planar graph with n regions and minimum vertex degree 3. Each of the e edges has two vertices, and each of the v vertices belongs to at least three edges. Hence $2e \geq 3v$. Euler formula $n + v - e = 2$ implies $e \leq 3n - 6$ and $v \leq 2n - 4$. In average, the number of edges of each Voronoi regions is less than 6.

We get a dual diagram if we draw a straight line connecting $\mathbf{p}$ and $\mathbf{q}$ in S if their Voronoi regions share a common line segment. In general, if no four sites of S share a common circle, i.e., S is in *general position*, the dual diagram is a 2-dimensional simplicial complex which decompose the convex hull of S. It is called the *Delaunay triangulation* of S, see Figure 14 Right.

There is an ambiguity in the definition of Delaunay triangulation if four or more Voronoi regions meet at a common point, i.e., four sites of S share a common circle. Probabilistically, the chance of picking four points on the circle is zero because the

circle defined by first three points has zero measure in $\mathbb{R}^2$. A common way to say the same thing is that four points in a common circle for a *degeneracy* or a *special case*, see Figure 15 Left. An arbitrary small perturbation suffices to remove the degeneracy and to reduce the special to general case, see Figure 15 Right.

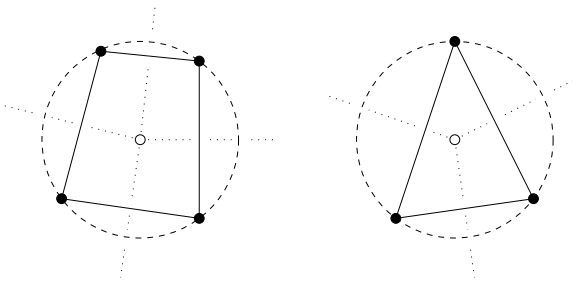


FIGURE 15. To the left, four dotted Voronoi edges meet at a common vertex and the dual Delaunay edges bound a quadrilateral. To the right, in general case, only three Voronoi edges meet at a vertex and the Delaunay edges bound a triangle (Figure from [6]).

3.2. **The empty circumcircle property.** Alternatively, Delaunay triangulations can be defined through the so-called *empty circumcircle property*. Recall that the *circumcircle*, or *circumscribing circle* of a simplex σ is the circle that passes through all vertices of σ. In plane, a triangle has a unique circumcircle, while an edge has infinitely many circumcircles, see Figure 16 Left.

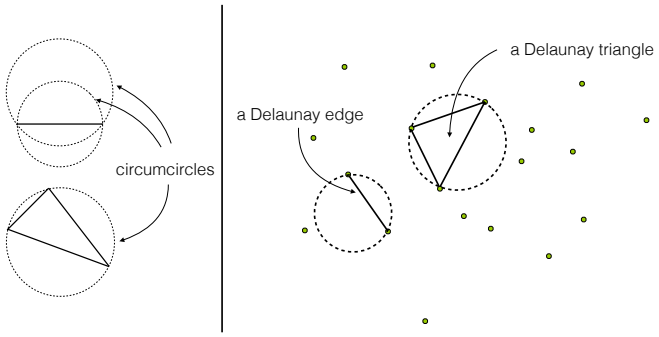


FIGURE 16. Empty circumcircles and Delaunay simplices.

Let S be a set of n points (called *sites*) in $\mathbb{R}^2$. A simplex σ whose vertices are in S is *Delaunay* if it has a circumcircle that encloses no site in S. We say, that σ has an empty circumcircle, see Figure 16 Right.

Question: given a set of points, how to find all Delaunay simplics?

We describe an approach which was used by Delaunay himself. It is a process of "growing empty balls" within a set of points.

If S is in general position, then the set of all Delaunay simplices of S form a simplical complex whose union is the convex hull of S, it is the *Delaunay triangulation* of S.

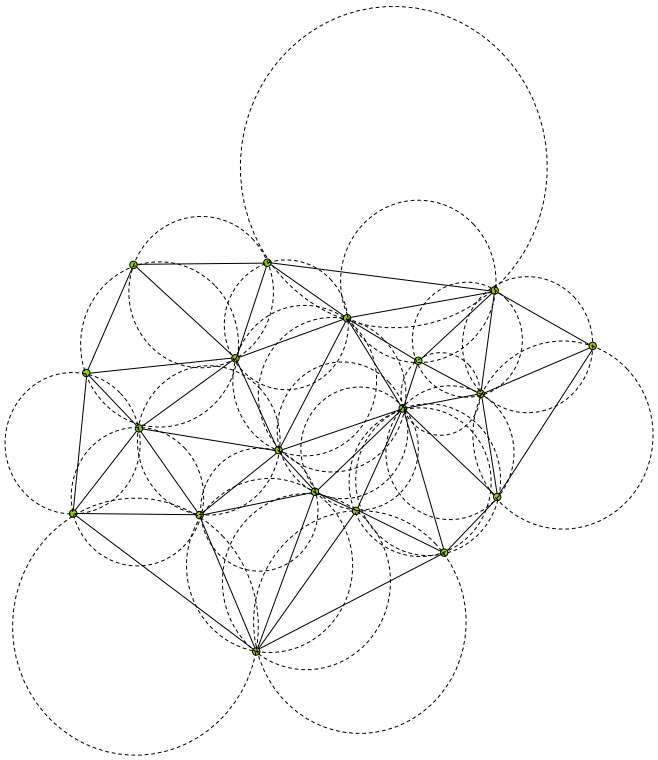


FIGURE 17. The empty circumcircle property of the Delaunay triangulation.

Note that if S is not in general position, then the set of all Delaunay simplices of S is not a simplical complex. There are simplicies overlapping each other. In this case, one could still obtain a Delaunay subduevsion by deleting all Delaunay simplifies which are overlapping each other.

3.2.1. *The* `in_circle` *test.* Given three non-collinear points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ in $\mathbb{R}^2$, the geometric predicate to test whether a point $\mathbf{d}$ lies inside, on, or outside the circumcircle of the triangle **abc** is:

$$\texttt{in\_circle}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) = \text{sign}(\det(A)),$$

where

$$A = \begin{bmatrix} a_x & a_y & a_x^2 + a_y^2 & 1 \\ b_x & b_y & b_x^2 + b_y^2 & 1 \\ c_x & c_y & c_x^2 + c_y^2 & 1 \\ d_x & d_y & d_x^2 + d_y^2 & 1 \end{bmatrix}$$

Note there are exactly two orientations of three points in the plane, which correspond to the left-handed rule or the right-handed rule, respectively. It is up to the user to decide which one is "positively" oriented (so the other one is "negatively" oriented).

Assume that the three points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are counterclockwise ordered in the plane (viewed from a point above this plane). Then we have

$$\begin{array}{rcll} \texttt{in\_circle}(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) & > & 0 & \longrightarrow \mathbf{d} \text{ lies inside ,} \\ \texttt{in\_circle}(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) & = & 0 & \longrightarrow \mathbf{d} \text{ is co-circular,} \\ \texttt{in\_circle}(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}) & < & 0 & \longrightarrow \mathbf{d} \text{ lies outside} \end{array}$$

of the circle passing through $\mathbf{a}, \mathbf{b}, \mathbf{c}$.

3.3. **The lifting transformation.** There is a fascinating relation between a Delaunay triangulation in 2d and a convex hull in 3d. These two structures appear from quite different concepts. In this section, we will establish their relation through a lifting and projecting process.

Let S be a set of n points in $\mathbb{R}^2$. We now consider, for each point $\mathbf{p} = (p_x, p_y) \in S$, a point $\mathbf{p}' = (p_x, p_y, p_z) \in \mathbb{R}^3$, where $p_z := p_x^2 + p_y^2$, i.e., $\mathbf{p}'$ is a point on the paraboloid $z = x^2 + y^2$ in $\mathbb{R}^3$, and $\mathbf{p}$ is the projection of $\mathbf{p}'$ into the plane by removing its z-coordinate. We call this map $f : \mathbf{p} \in \mathbb{R}^2 \to \mathbf{p}' \in \mathbb{R}^3$ the *lifting map*. The lifting map that takes a point in the plane to a paraboloid in $\mathbb{R}^3$, see Figure 18 Left.

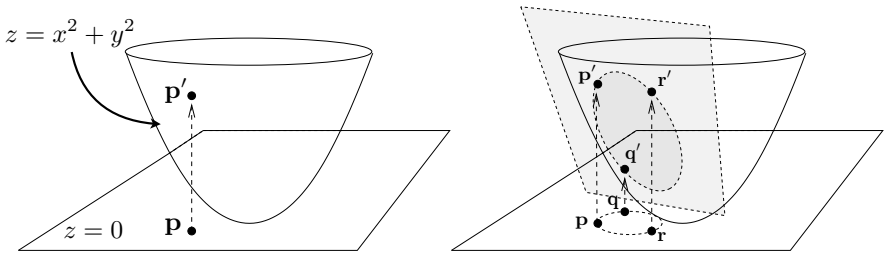


FIGURE 18. The lifting map: circles map to planes.

3.3.1. *Planes and circles.* We have the following fact which gives the relation between circles in $\mathbb{R}^2$ and plane in $\mathbb{R}^3$, see Figure 18 Right.

Lemma 3.1. *Consider 4 distinct points p, q, r, s in the plane, and let p', q', r', s' be their respective projections on to paraboloid, $z = x^2 + y^2$. The point s lies within the circumcircle of p, q, r if and only if s' lies on the lower side of the plane passing through p', q', r'.*

Proof. Consider an arbitrary plane in $\mathbb{R}^3$, which we assume is tangent to the paraboloid above the point $p := (a, b)$ in the plane. Then the plane equation is:

$$n_x(x - a) + n_y(y - b) + n_z(z - (a^2 + b^2)) = 0,$$

where (n_x, n_y, n_z) is the normal of this plane. One can find the normal components by the gradient of the equation $z = x^2 + y^2$. Thus $n_x = 2a$, $n_x = 2b$, and $n_z = -1$. Thus, the plane equation is

$$z = 2ax + 2by - (a^2 + b^2).$$

If we shift the plane upwards by some positive amount ρ^2 we get the plane

$$z = 2ax + 2by - (a^2 + b^2) + \rho^2.$$

Since $z = x^2 + y^2$, we can eliminate z, giving

$$x^2 + y^2 = 2ax + 2by - (a^2 + b^2) + \rho^2.$$

This is equivalent to

$$(x - a)^2 + (y - b)^2 = \rho^2.$$

This is a circle centered at (a, b) with a radius ρ. Thus, the intersection of a plane with paraboloid produces a closed space curve (which turns out to be an ellipse), which when projects back onto the (x, y) coordinate plane is a circle. Furthermore, the squared radius (ρ) of the circle equals to the vertical distance between the lifting of the (a, b) onto the paraboloid (whose distance is $a^2 + b^2$) and its lifting onto the plane (whose distance is $a^2 + b^2 + \rho^2$).

Thus, we conclude that the intersection of an arbitrary lower halfspace with the paraboloid, when projected onto the (x, y) plane is the interior of a circle.

Consider the lifting of p, q, r onto the paraboloid, lifted points p', q', r' define a plane, the intersection of the plane and the paraboloid is an ellipse, and the projection of this ellipse onto the (x, y) plane is just the circumcircle passing through p, q, r. The point s lies within this circumcircle, if and only if its lifting s' onto the paraboloid lies within the lower halfspace of the plane passing through p', q', r'. □

3.3.2. *Convex hulls and Delaunay triangulations.* Due to the above fact, we can easily show the nice relation between Delaunay triangulation and convex hulls.

Let S' be the set of all sites resulting obtained by the lifting map on S. The convex hull of S' is a 3d convex polytope, denoted as $\text{conv}(S')$. A *lower face* of $\text{conv}(S')$ is a face such that it is contained in a non-vertical plane in $\mathbb{R}^3$ and the whole polytope lies vertically above this plane. The projection of the set of lower faces of $\text{conv}(S')$ into the xy-plane gives a subdivision of the convex hull of S. If S is in general position, then this subdivision is a *simplical complex* $\mathcal{T}$ which is the *Delaunay triangulation* of S, see Figure 19.

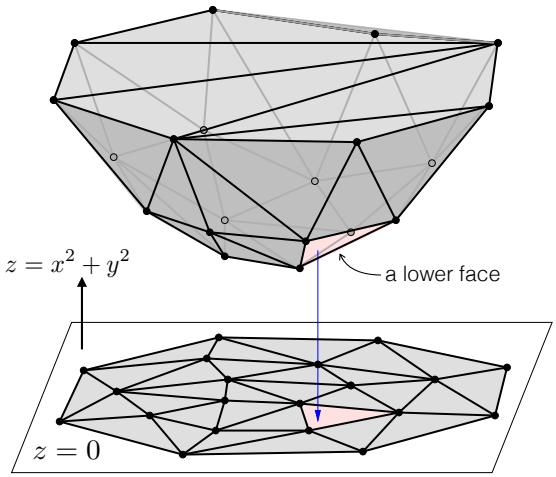


FIGURE 19. The projection of the lower faces of the convex hull is the Delaunay triangulation.

3.3.3. *Convex hulls and Voronoi diagrams.* Given a point $p = (a, b)$, the hyperplane $H(p)$ that is tangent to p's lifting, namely, $(a, b, a^2 + b^2)$, has the equation

$$z = 2ax + 2by - (a^2 + b^2).$$

Now, consider an arbitrary point $q = (\alpha, \beta)$ in the plane, the vertical distance from q to the paraboloid is $\alpha^2 + \beta^2$. What is the vertical distance from q to $H(p)$? By substituting (α, β) into the equation of $H(p)$, we get $2a\alpha + 2b\beta - (a^2 + b^2)$. Let $\Delta(p, q)$ denote the difference between these two vertical distances at point q,

$$\Delta(p, q) = \alpha^2 + \beta^2 - (2a\alpha + 2b\beta - (a^2 + b^2)) = (a - \alpha)^2 + (b - \beta)^2.$$

This shows that $\Delta(p, q)$ equals precisely the power of the two-dimensional Euclidean distance between p and q.

Consider two points p_1 and p_2 in the plane $z = 0$. We claim that q is closer to p_1 if and only if at the position $q = (\alpha, \beta)$, the plane $H(p_1)$ lies *above* (closer to the paraboloid) than $H(p_2)$. It simply follows from the above vertical distance formula.

Lemma 3.2. *Let $p_1, p_2, \ldots, p_n$ be a set of points in the plane $z = 0$. A point q belongs to the Voronoi cell of the point p_i, if and only if and only if $H(p_i)$ is the highest plane (seen from $z = +\infty$) at q.*

Therefore, the Voronoi diagram of $p_1, p_2, \ldots, p_n$ is simply the vertical projection, down to $z = 0$, of the point-wise maxima of the downward facing half spaces $H(p_i)$. Or equivalent , is the uppermost face of the arrangement defined by these planes.

3.4. **Lawson's edge flip algorithm.** This section introduces a local condition for edges, shows it implies a triangulation is Delaunay, and derives an algorithm based on edge flipping. The correctness of this algorithm implies two important results of planar triangulations, (1) among all triangulations of the same point set, the Delaunay triangulation maximises the minimum angle; and (2) the set of all triangulations of the point set is connected by edge flips.

3.4.1. *Locally Delaunay lemma.* Let $\mathcal{K}$ be a triangulation of a point set S in $\mathbb{R}^2$. An edge $\mathbf{ab} \in \mathcal{K}$ is *locally Delaunay* if either

(i) it is on the convex hull, or
(ii) it belongs two triangles, **abc** and **abd**, and **d** lies outside the circumcircle of **abc**.

This definition is illustrated in Figure 20. A locally Delaunay edge is not necessary a Delaunay edge. However, if every edge is locally Delaunay, then we can show that all are Delaunay edges.

Theorem 3.1 (Delaunay Lemma)**.** *If every edge of $\mathcal{K}$ is locally Delaunay, then $\mathcal{K}$ is the Delaunay triangulation of S.*

Proof. We use a new concept of distance from a circle. The *power distance* of a circle U with center u and radius r is: $\pi_U(x) := \|x - u\|^2 - r^2$. If x lies outside of this circle, then this distance is the square length of the tangent line segment connecting x and U. In any case, the distance is positive outside the circle, zero on the circle, and negative inside the circle.

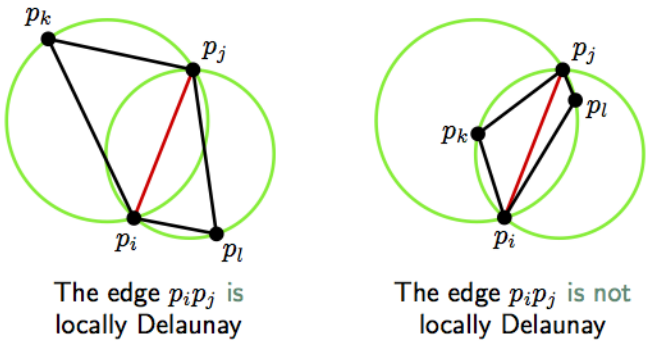


FIGURE 20. To the left $\mathbf{p}_i\mathbf{p}_j$ is locally Delaunay and to the right it is not.

Let $x \in \mathbb{R}^2$ be a fixed point. We say a triangle abc lies in-front of another triangle def if there is a half-line starting at x that first pass through abc and then through def, denoted as $abc \prec_x def$. The following fact is crucial to prove the Delaunay Lemma:

If $abc \prec_x abd$ and ab is locally Delaunay, then $\pi_{C_{abc}}(x) < \pi_{C_{abd}}(x)$, where C_{abc} is the circumcircle of abc, see Figure 21.

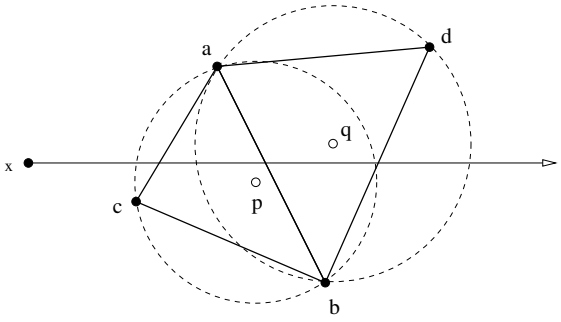


FIGURE 21. Edge ab is locally Delaunay implies that $\pi_{C_{abc}}(x) < \pi_{C_{abd}}(x)$.

Define $abc = t_0$ and write $\pi_{t_0}(x)$ for the power distance of x from the circumcircle of t_0. Assume z is a vertex lies inside the circumcircle of abc, then we could find a sequence of triangles starting from abc and ending at a triangle uvz, i.e.,

$$abc = t_0 \prec_x t_1 \prec_x \cdots \prec_x t_m = uvz.$$

Now we have $\pi_{t_m}(z) = 0$, and $\pi_{t_i}(z) > 0$, for $i = m-1, \ldots, 0$. But our assumption means $\pi_{t_0}(z) < 0$, a contradiction. □

3.4.2. *Edge flips.* Given an edge **ab**, let the two triangles sharing at **ab** are **abc** and **abd**. The union of these two triangles must be a convex quadrangle. We can *flip* **ab** to **cd**. Formally, this means we remove **ab**, **abc**, and **abd** from the triangulation, and add **cd**, **cda**, and **cdb** to the triangulation, see Figure 22.

If **ab** is not locally Delaunay, then **cd** is locally Delaunay. This fact can be easily shown by the following fact: If **ab** is not locally Delaunay, the sum $\angle acb + \angle adb > 180^o$. Then the sum $\angle cad + \angle cbd \leq 180^o$.

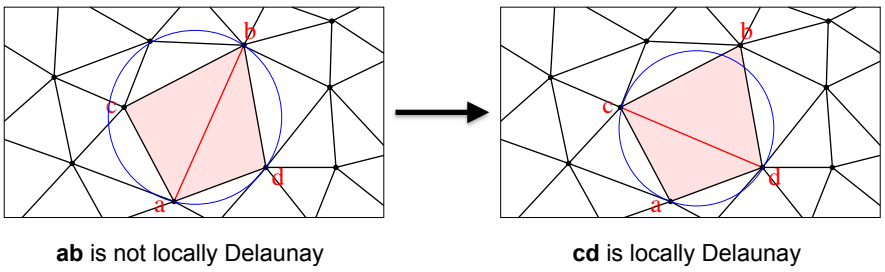


FIGURE 22. The edge flips between **ab** and **cd**.

3.4.3. *The algorithm.* We can use edge flips as elementary operations to convert an arbitrary triangulation $\mathcal{K}$ to the Delaunay triangulation. The algorithm uses a stack to maintains all edges which may be locally non-Delaunay. Initially, all edges of $\mathcal{K}$ are pushed on the stack.

Algorithm: LawsonFlip(L)
Input: a stack L of edges of a triangulation $\mathcal{K}$;
Output: the modified triangulation $\mathcal{K}$;
1 **while** $L \neq \emptyset$ **do**
2 pop an edge **ab** from L;
3 **if ab** is not locally Delaunay **then**;
4 flip **ab** to **cd**;
5 push edges **ac**, **cb**, **db**, **da** on L;
6 **endif**
7 **endwhile**

FIGURE 23. The Lawson edge-flip algorithm.

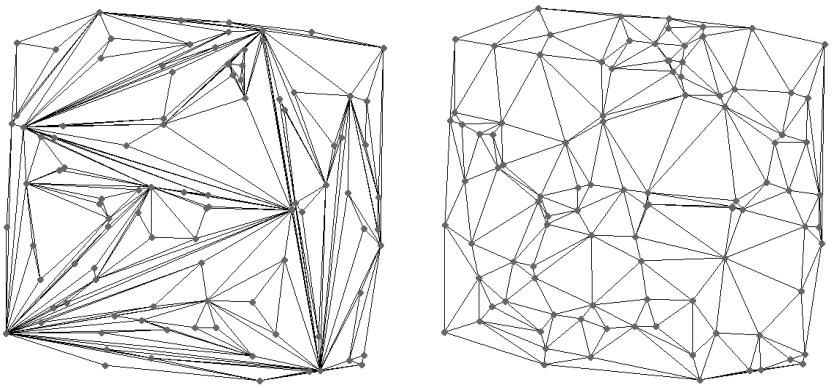


FIGURE 24. Lawson's flip algorithm takes an arbitrary triangulation (left) as input and returns the Delaunay triangulation (right).

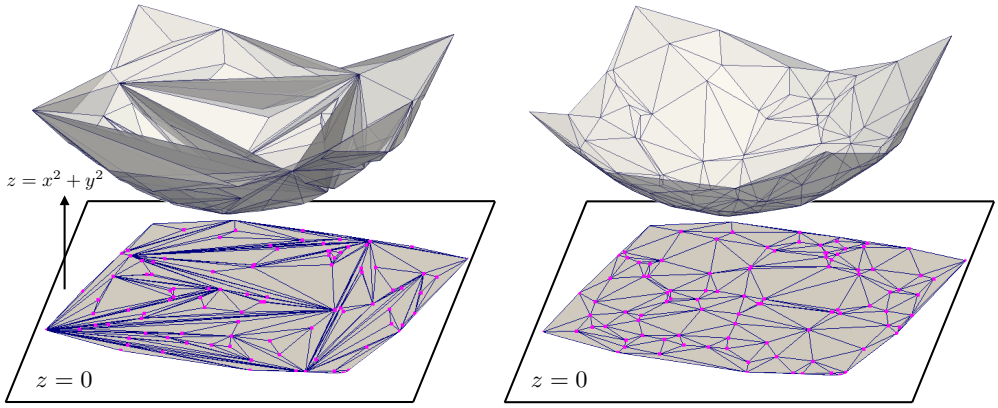


FIGURE 25. The lifted view of the Lawson's flip algorithm which transforms a non-convex surface (left) in 3d into a convex one (right).

3.4.4. *Termination and running time.* Flipping **ab** to **cd** is likely gluing a tetrahedron $\mathbf{a'b'c'd'}$ from below to $\mathbf{a'b'c'}$ and $\mathbf{a'b'd'}$, see Figure 25.

The algorithm can be understood as gluing a sequence of tetrahedra. Once we glue $\mathbf{a'b'c'd'}$ we cannot glue another tetrahedron right below $\mathbf{a'b'}$. In other words, once we flip **ab** we cannot introduce **ab** again by some other flip. This implies that the Lawson's edge-flip algorithm will eventually terminate when all locally non-Delaunay edges are flipped. By the Delaunay lemma, the triangulation is Delaunay. This also implies there are at most as many flips as there are edges connecting n points, namely $\binom{n}{2}$. Each flip takes constant time, hence the total running time is $O(n^2)$.

3.4.5. *The MaxMin angle property.* We illustrate an optimal property of the Delaunay triangulation.

Theorem 3.2. *Among all triangulation of a finite point set $S \subset \mathbb{R}^2$, the Delaunay triangulation maximises the minimum angle.*

Proof. A flip substitute two new triangles for two old triangles. It therefore changes six of the angles, see Figure 26. The six old angles are:

$$\alpha_1, \beta_1, \gamma_1 + \gamma_2, \alpha_2, \beta_2, \delta_1 + \delta_2,$$

and the six new angles are

$$\gamma_1, \delta_1, \beta_1 + \beta_2, \gamma_2, \delta_2, \alpha_1 + \alpha_2.$$

We show that for each of the six new angles there is an old angle that is at least as small. At first, $\gamma_1 \geq \alpha_2$, since both angles are opposite the same edge **bd**, and **a** lies outside the circumcircle of **bdc**. Similarly, $\delta_1 \geq \alpha_1$, $\gamma_2 \geq \beta_2$, $\delta_2 \geq \beta_1$, and for trivial $\alpha_1 + \alpha_2 > \alpha_1$, and $\beta_1 + \beta_2 \geq \beta_1$.

It follows that a flip does not decrease the smallest angle in a triangulation. Since we can transform any triangulation $\mathcal{K}$ of S to the Delaunay triangulation, this implies that the smallest angle in $\mathcal{K}$ is no larger than the smallest angle in the Delaunay triangulation. □

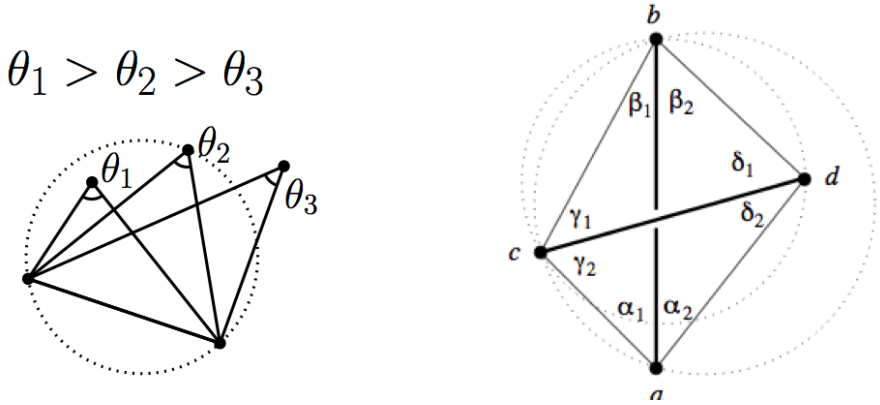


FIGURE 26. Flipping edge **ab** to **cd** changes six of the angles.

3.4.6. *The flip graph.* One can use flips to traverse the set of all triangulations of S. We can form a *flip-graph* $\mathcal{G}$ of S. Each triangulation is a node of $\mathcal{G}$, and each edge of $\mathcal{G}$ between two nodes u and v means there is a flip that changes the triangulation u to v. Figure **??** shows an example. The termination of Lawson's flip algorithm implies that the flip-graph is connected. Moreover, one can go from any triangulation of S to any other triangulation in less than $O(n^2)$ flips.

3.5. **Randomized incremental flip algorithm.** In this section, we introduce an algorithm that construct Delaunay triangulations incrementally, using edge flips and randomisation. After explain the algorithm, we present a detailed analysis of the expected running time.

The basic step of this algorithm is to interleave flipping edges and adding points. Denote the points in $S \subset \mathbb{R}^2$ as $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$, and assume general position. To reduce the outside to the inside case, we start with a triangulation $\mathcal{D}_0$ that consists of a single and sufficiently large triangle **xyz**. The algorithm is a `for-loop` adding the points in sequence. After adding a point, it uses edge flips to satisfy the Delaunay lemma before the next point is added. The algorithm is given in Figure 27.

Algorithm: IncrementalFlip($S = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$)
Input: a sequence S of n points in $\mathbb{R}^2$;
Output: the Delaunay triangulation $\mathcal{D}_n$ of S;
1 initialize $\mathcal{D}_0$ with only one larger triangle **xyz**;
2 **for** $i = 1$ to n **do**
3 find the triangle $\tau \in \mathcal{D}_{i-1}$ containing $\mathbf{p}_i$;
4 insert $\mathbf{p}_i$ by splitting τ into three triangles containing $\mathbf{p}_i$;
5 initial the stack L with three link edges of $\mathbf{p}_i$;
6 LawsonFlip(L);
7 **endfor**
8 remove all triangles containing **x**, **y**, and **z** from $\mathcal{D}_n$;

FIGURE 27. The incremental-flip algorithm.

At the moment, we will assume the general position, i.e., each vertex $\mathbf{p}_i$ lies exactly in the interior of a triangle. The vertex insertion (line 4) is another elementary flip operation, which split a triangle τ into three is also an elementary flip called 1-to-3, see flip 28. The reserve of it is another elementary flip, 3-to-1 flip, which removes a vertex from a triangle. We will discuss vertex insertion in great detail in Section **??**.

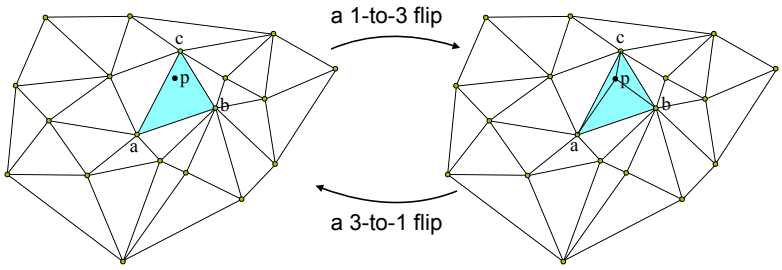


FIGURE 28. Vertex insertion by a 1-to-3 flip and deletion by a 3-to-1 flip.

3.5.1. *Number of flips.* The running time of this algorithm consists of two parts: (1) the time to locate the triangle τ containing the vertex $\mathbf{p}_i$; and (2) the time to perform flips. In this section, we study the total number of flips that will be done in this algorithm. We consider two cases: the worst case and the expected case when vertices are inserted in a random order.

Note that every new triangle in $\mathcal{D}_i$ has $\mathbf{p}_i$ as vertex. Indeed, if **abc** is a triangle in $\mathcal{D}_{i-1}$ and its circumcircle does not contain $\mathbf{p}_i$, then it must be also a triangle in $\mathcal{D}_i$. This implies that all flips during the insertion of $\mathbf{p}_i$ occur right around $\mathbf{p}_i$, see Figure 29. This implies the number of edges flips is related to the degree of $\mathbf{p}_i$, i.e. the number of edges which have $\mathbf{p}_i$ as a vertex. It is denoted as $\deg(\mathbf{p}_i)$. In the incremental flip algorithm, each edge flip increases the degree of $\mathbf{p}_i$ by 1. Since the initial degree of $\mathbf{p}_i$ is 3 (creating by splitting the triangle τ), then the number of edge flips to add $\mathbf{p}_i$ in $\mathcal{D}_i$ is equal to $\deg(\mathbf{p}_i) - 3$.

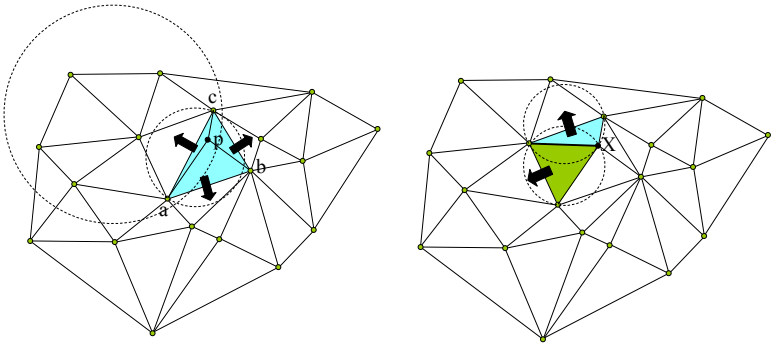


FIGURE 29. The Lawson's flip algorithm after the vertex insertion. All flipped edge must contain the new vertex $\mathbf{p}_i$.

We first consider the worst case. Figure 3.5.1 shows such an example. Assuming the sequence of vertices is ordered first from left to right, then from bottom to top. The degree of each successive vertices can be $\Theta(n)$, hence the total number of flips is $\Theta(n^2)$. This shows that if the insertion order of the vertices are chosen badly, the incremental flip algorithm can take $\Theta(n^2)$ time.

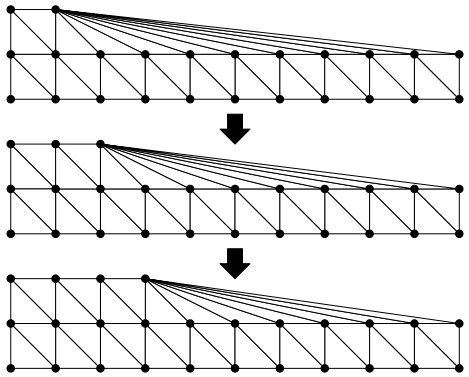


FIGURE 30. Each vertex insertion can cause $\Theta(n)$ flips (example from [**?**]).

3.5.2. *The expected number of flips.* We next show if the vertices are inserted in a random order, the expected total number of flips is only $O(n)$. Random does not mean arbitrary but rather that every permutation of the n points is equally likely. There are total $n!$ permutations. Let $S_1, \ldots, S_m$ be the $m = n!$ permutations of n points.

Let f_i be the total number of flips resulted by the incremental-flip algorithm on S_i. The technique we use to analyse the algorithm is called the backward analysis developed by Seidel [10].

Consider inserting the last point $\mathbf{p}_n$. The sum of degrees of all possible last points is the same as the sum of degrees of all points in $\mathcal{D}_n$ (due to the uniqueness of the Delaunay triangulation). The latter is equal to twice the number of edges, which is

$$\sum_{i=1}^{n} \deg(\mathbf{p}_i) \leq 2(3n-6) \leq 6n.$$

Note that each of the last point appears $(n-1)!$ times in all the $n!$ permutations. Therefore the number of flips for adding all last points is at most:

$$F_n \leq (6n - 3n)(n-1)! \leq 3n(n-1)!,$$

where the $-3n$ is due to the number of edges created by the point insertion is not counted as the number of edge flips.

Then the total number of flips is:

$$\begin{array}{rcl} \sum_{i=1}^{n} f_i & = & f_n + f_{n-1} + \cdots + f_1 \\ & \leq & 3n \cdot (n-1)! + 3(n-1)(n-1)! + \cdots + 0 \\ & \leq & 3n \cdot n \cdot (n-1)! \\ & = & 3n \cdot n! \end{array}$$

The expected number of edge flips for adding n points is

$$\mathrm{E}[\sum_{i=1}^{n} f_i] \leq 3n \cdot n! \cdot \frac{1}{n!} = 3n.$$

There is a simple way to say the same thing. If points are inserted in a random order, the expected number of flips for the last point is at most 3. Hence the total number of edge flips for adding n points is $O(n)$.

3.6. **Point location.** The point location is another important fact for running time. In this section, we first give a very simple scheme for point location. However, the worst case runtime of algorithm is $O(n)$ for locating one point. We then discuss several simple and practical ways to overcome this.

3.6.1. *Straight walk.* A simple point location scheme is called "straight walk". More precisely, the algorithm starting from an arbitrary triangle $\sigma \in \mathcal{D}_i$, and search the triangle τ that containing $\mathbf{p}_i$ by walking along the ray starting from an interior point of σ toward to $\mathbf{p}_i$.

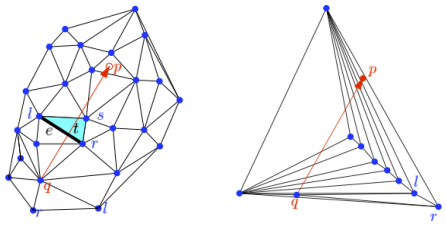


FIGURE 31. Locating a point by straight line walk. (Images from [4]).

How much time will this searching algorithm uses? We assume that each triangle is visited by only once. Then each point location will visit at most the number of triangles of the triangulation which is less than $2n$. One can show that this is possible by the triangulation shown in Figure 3.6.1 Right. Hence the location of n points by this algorithm may take $O(n^2)$ time. It is too slow. Hence, with this point location algorithm, the randomized incremental flip algorithm still has the expected $O(n^2)$ runtime.

3.6.2. *Jump and walk.* Despite the nice theoretical guarantee of runtime, it is not practical to use additional data structure for point location.

On the other hand, the simple walk-through algorithm could work efficiently if it could chose a "good" starting point which lies not too far away from the searching point, then the walk will not taking too many steps. A simple way to achieve this is to randomly sampling some points and select the nearest one, this is so-called the "jump-and-walk" scheme [8]. They proposed a "jump-and-walk" method chooses a random sample of $O(n^{1/3})$ triangles from the current triangulation, and choose the closest one as the starting triangle for locating points.

3.6.3. *Pre-sorting.* The most efficient way to realise this is to pre-sort the points according to their spatial location such that the point to be located is not far from the last inserted point. We will introduce several point sorting schemes in Section **??**.

References

[1] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.

[2] Long Chen and Jin-Chao Xu. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.

[3] B. N. Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[4] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in triangulation. *International Journal of Foundations of Computer Science*, 13(2):181–199, 2002. INRIA Tec. Report No. 4120, 2001.

[5] H. Edelsbrunner. An acyclicity theorem for cell complex in d dimension. *Combinatorica*, 10(3):251–260, 1990.

[6] Herbert Edelsbrunner. *Geometry and topology for mesh generation*. Cambridge University Press, Cambridge, England, 2001.

[7] Damrong Guoy. *Tetrahedral Mesh Improvement, Algorithms and Experiments*. PhD thesis, Computer Science in University of Illionois at Urbana-Champaign, 2001.

[8] Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proc. 12th annual Symposium on Computational Geometry*, pages 274–283, 1996.

[9] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM*, 25(10):739–747, October 1982.

[10] R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. In *1978-1988 Ten Years IIG*, pages 178–191, 1988.

[11] G. Voronoi. Nouvelles applications des parametrès continus à la théorie de formas quadratiques. *Reine Angew. Math.*, 133:97–178, 1907.