DELAUNAY TRIANGULATIONS IN THE PLANE

HANG SI

Contents

Introduction

This chapter introduces the most fundamental geometric structures – Delaunay triangulation as well as its dual Voronoi diagram. We will start with their definitions and properties. Then we introduce simple and efficient algorithms to construct them in the plane. We first learn a useful and straightforward algorithm – Lawson's edge flip algorithm – which transforms any triangulation into the Delaunay triangulation. We will prove its termination and correctness. From this algorithm, we will show many optimal properties of Delaunay triangulations. We then introduce a randomized incremental flip algorithm to construct Delaunay triangulations and prove its expected runtime is optimal.

1. Definitions and properties

This section introduces the Delaunay triangulation of a finite point set in the plane. It is introduced by the Russian mathematician Boris Nikolaevich Delone (1890–1980)

in 1934 [4]. It is a triangulation with many optimal properties. There are many ways to define Delaunay triangulation, which also shows different properties of it. This section will introduce the dual definition via the Voronoi diagram, the empty circumcircle definition, and the lifting transformation definition via convex hull.

1.1. **Voronoi diagrams.** Given a finite point set in the plane, the Voronoi diagram of this point set divides the plane according to the *nearest-neighbor rule*: Each point of this point set is associated with a region of the plane.

1.1.1. *Definitions.* Let S be a finite set of n points in a plane. The *Voronoi region* of a point $\mathbf{p} \in S$ is the set of points in the plane that are as close to $\mathbf{p}$ as to any other point in S, that is

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

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*:

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

In general, when S has $n \geq 3$ points. The Voronoi region of a point in S is a convex polygonal region, possibly unbounded, with at most $n - 1$ edges, see Figure 1 Left for an example.

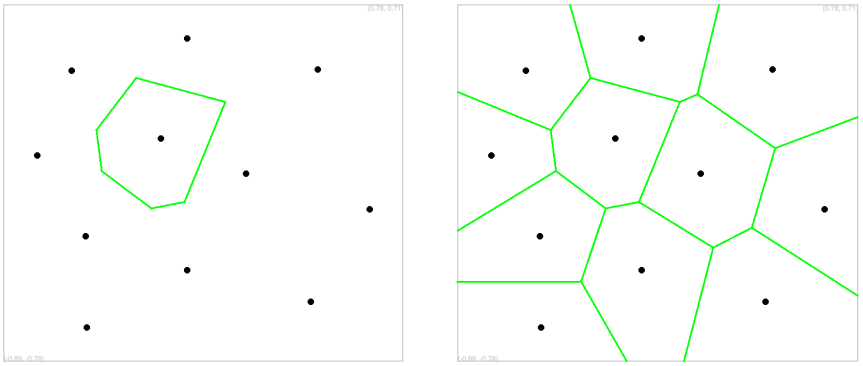

FIGURE 1. A Voronoi region (left) and the Voronoi diagram (right) of a two-dimensional point set.

Each point in the plane has at least one nearest point in S, so it belongs to at least one Voronoi region. It follows that the Voronoi regions cover the entire plane. Any two Voronoi regions must not overlap each other, i.e., they are either disjoint or share at their common face. The Voronoi regions together with their edges and vertices form the *Voronoi diagram* of S [15], see Figure 1 Right. Voronoi diagram is named after the Russian and Ukrainian mathematician Georgy Feodosevich Voronoy (1868–1908) [15]. It is also known as Dirichlet tessellations (after German mathematician Peter Gustav Lejeune Dirichlet (1805 – 1859)).

The Voronoi diagram of a set of n two-dimensional points 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.

Voronoi diagram arises in nature in various situations. It is one of the most fundamental data structures in computational geometry. For example, a point location data structure can be built on top of the Voronoi diagram to answer nearest neighbor queries, where one wants to find the object that is closest to a given query point. Aurenhammer [1] gives an excellent survey of Voronoi diagrams.

There are many algorithms for computing the Voronoi diagram of a set of n points in the plane. A naive algorithm is for each point $\mathbf{p}_i$, compute its Voronoi region $V(\mathbf{p}_i)$ by interesting the $n-1$ half-spaces $H_{p_q p_j}$, $\mathbf{p}_j \neq \mathbf{p}_i$. This algorithm takes at least $O(n^2)$ time. There are efficient algorithms that run in $O(n \log n)$ time. The first such algorithm is based on divide-and-conquer being Shamos and Hoey [14]. However, this algorithm is a bit complicated (in the merge step) to be implemented. A plane sweep algorithm developed by Fortune [7] is based on incremental construction. It is efficient and runs in $O(n \log n)$. It is also simple to implement. Later in this section, we will discuss another incremental algorithm that constructs the Voronoi diagram's dual structure.

1.1.2. *The dual diagram.* 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 edge, see Figure 2. If no four or more points share a common circle, this dual diagram is a two-dimensional simplicial complex that decomposes the convex hull of S. It is called the *Delaunay triangulation* of S, see Figure 2 Right.

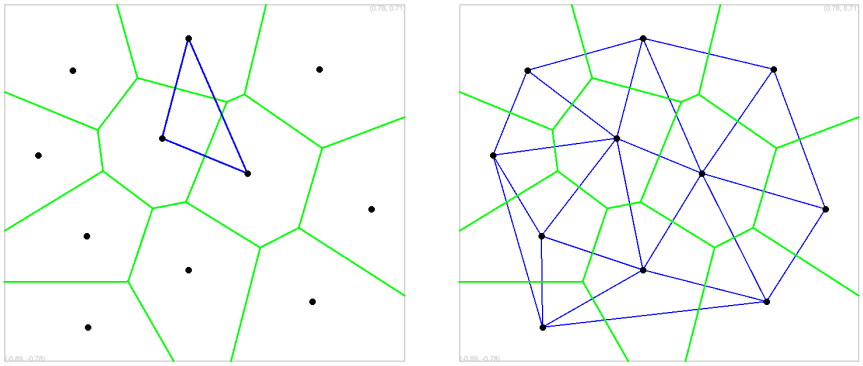

FIGURE 2. Three dual edges of the Voronoi edges (Left) and the dual diagram (Right). If no four or more points of the point set share a common circle, then this dual diagram is the Delaunay triangulation of this point set.

1.1.3. *Degeneracies and general position.* There is an ambiguity in the definition of Delaunay triangulation if four or more Voronoi regions meet at a common point. This case implies that there are four or more points of S lie on a common circle. Probabilistically, the chance of picking arbitrary four points on the circle is zero because the circle defined by the first three points has zero measure in $\mathbb{R}^2$. A common way to say the same thing is that four points lie on a common circle form a *special case*, or they are *degenerate*, see

Figure 3 Left. An arbitrary small perturbation suffices to remove the degeneracy and reduce the special to the general case, see Figure 3 Right.

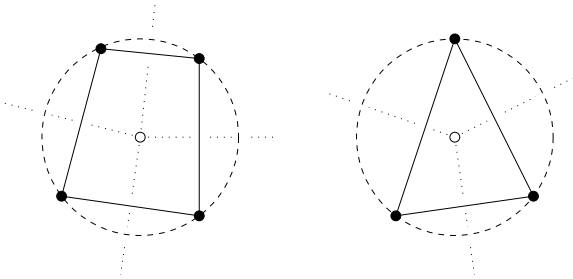

FIGURE 3. To the left, four dotted Voronoi edges meet at a common vertex, and the dual Delaunay edges bound a quadrilateral. To the right, in the general case, only three Voronoi edges meet at a vertex, and the Delaunay edges bound a triangle (Figure from [6]).

Since the Voronoi diagram of S is unique, if S is in general position, the Delaunay triangulation of S is also unique.

1.2. **The empty circumcircle property.** We now introduce another definition of Delaunay triangulation through the *empty circumcircle property*.

1.2.1. *Delaunay simplices.* Let S be a finite set of n vertices in the plane. Let σ be a simplex whose vertices are from S. The *circumcircle*, or *circumscribing circle* of σ is the circle that passes through all vertices of σ. A triangle has a unique circumcircle in the plane, while an edge can have infinitely many circumcircles, see Figure 4 Left. We say that a simplex σ has an *empty circumcircle* in S if it has a circumcircle that encloses no vertex of S, see Figure 4 Right. A simplex is *Delaunay* if it has an empty circumcircle.

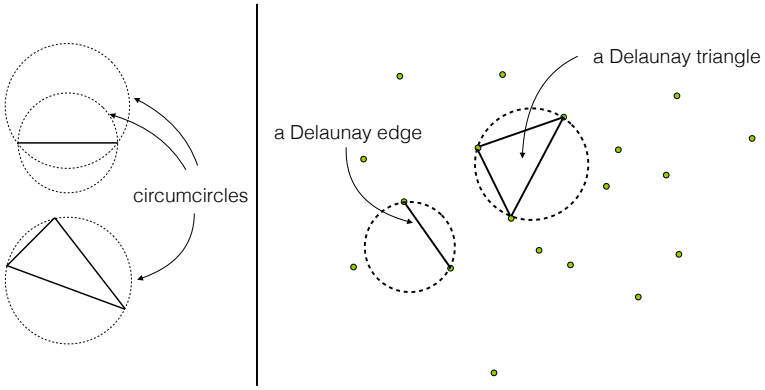

FIGURE 4. Empty circumcircles and Delaunay simplices.

1.2.2. *Growing empty circles.* Given a finite point set S, how can we find all Delaunay simplics of a point set in the plane?

A simple way is to test every three points, whether they have an empty circumcircle or not. With n points, there are $\binom{n}{3}$ triangles. For every triangle in this set, it needs to test $n-3$ points to determine whether it is a Delaunay triangle. Therefore, it will take $O(n^4)$ time to find all Delaunay triangles. Moreover, it is unclear whether the set of Delaunay triangles belongs to a triangulation of this point set.

We describe an approach that was used by Delaunay himself. It is a process of "growing empty balls" within a set of points. It starts with an empty ball at any point $\mathbf{p} \in S$, see Figure 5 (1). Let this circle grow as long as it does not touch any other points of S. This process stops once it touches a point $\mathbf{q} \in S$, we get a Delaunay edge $\mathbf{pq}$ of S, see Figure 5 (3). Now we grow an empty circumcircle of $\mathbf{pq}$ by choosing one of the two possible directions along the bisector line of $\mathbf{pq}$, see Figure 5 (4). The growing of this circumcircle (of edge $\mathbf{pq}$) will stop either:

(i) it touches a third point $\mathbf{r} \in S$, or
(ii) it never touches any point of S.

In case (i), we find a Delaunay triangle $\mathbf{pqr}$. While in case (ii), the edge $\mathbf{pq}$ must be a convex hull edge of S, and we switch the search direction to the opposite. This process will continue as long as there are vertices of S which do not belong to any Delaunay triangle.

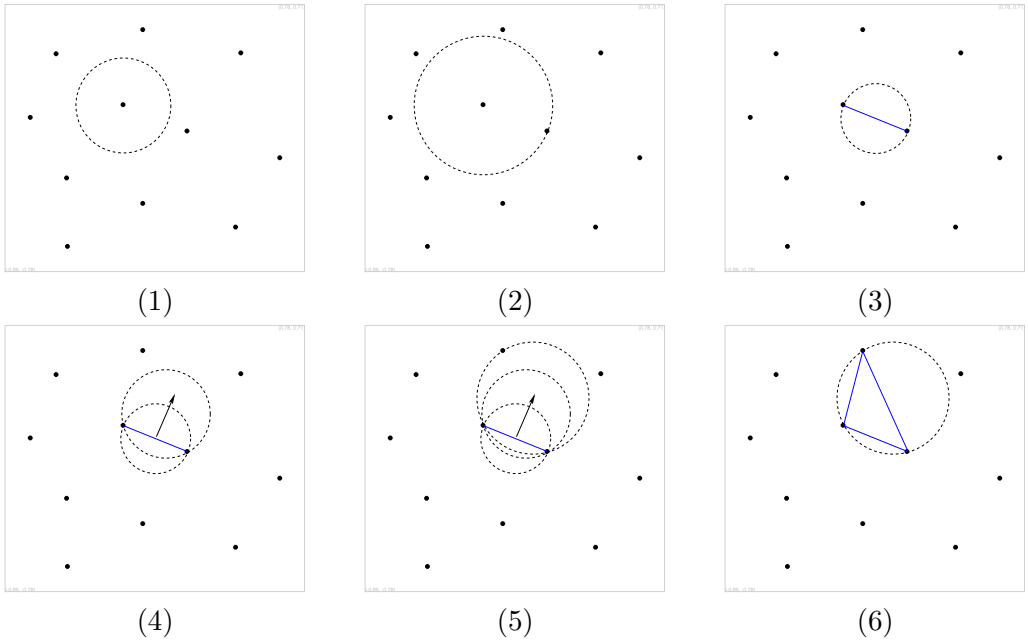

FIGURE 5. Growing empty circles to get Delaunay simplices.

This process will terminate, and we get a set of Delaunay triangles of S. Moreover, we know the following facts:

(1) at every Delaunay edge, there is either only one Delaunay triangle (it must be a convex hull edge) or there are two Delaunay triangles, and
(2) this set of Delaunay triangles must cover the convex hull of S.

If S is in general position, i.e., no four points of S lie on a common circle, then every triangle we found in the above process is unique. Therefore, we can claim that the set of Delaunay triangles and their edges and vertices form a S triangulation. It is called the *Delaunay triangulation* of S [4]. Figure 6 shows an example of the Delaunay triangulation of a set of points in the plane.

Delaunay triangulation is unique when S is in a general position. If S contains 4 or more points that lie on a common circle, all simplices formed by these vertices are Delaunay, and they overlap each other. Therefore, the set of all Delaunay simplices of S is not a simplical complex. In this case, one could still obtain a Delaunay subdivision by deleting all Delaunay simplifies overlapping each other.

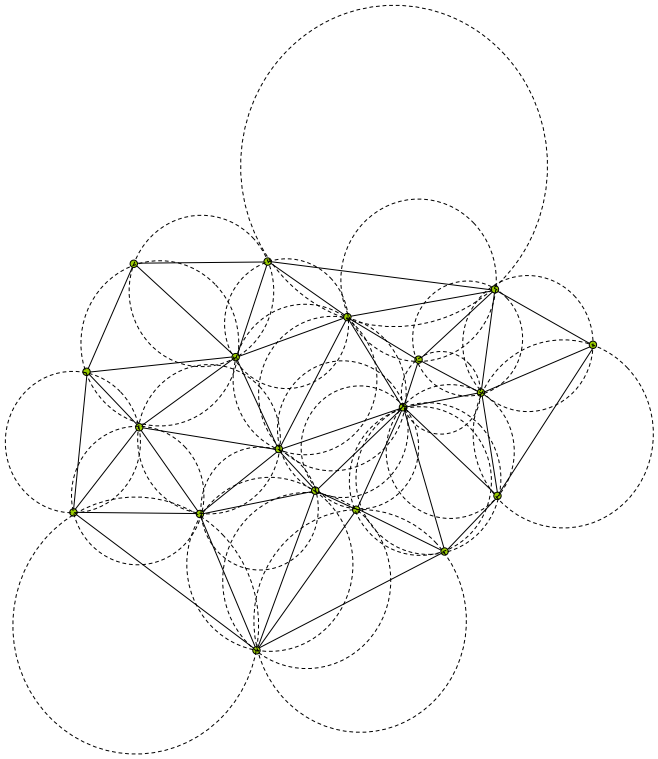

FIGURE 6. The empty circumcircle property of the Delaunay triangulation.

The above process indeed gives an algorithm to compute the Delaunay triangulation.

- Starting from any arbitrary point $\mathbf{p} \in S$. It takes $O(n)$ time to search the nearest point to find the first Delaunay edge $\mathbf{pq}$.
- From each Delaunay edge $\mathbf{pq}$, with a search direction (a normal of $\mathbf{pq}$), it searches a point which forms a Delaunay triangle with this edge. There are total $n-2$ points to be checked. If a point lies in the other halfspace opposite the search

direction, one can ignore it. If all points are ignored, we find it is a convex hull edge. Hence to find a convex hull edge takes $O(n)$ time. Otherwise, there are $n-2$ possible triangles to be checked. For each candidate triangle, we need to test at most $n-3$ points to check whether its circumcircle is empty or not. Hence, finding a Delaunay triangle from a given Delaunay edge takes $O(n^2)$ time.

There are at most $3n-6$ edges (by Euler's formula) in a n vertices triangulation. Hence in the worst case, this algorithm runs in $O(n^3)$ time.

Delaunay triangulation tends to connect points by their nearest neighbors. One of the properties of Delaunay triangulation is its relation to the Euclidean minimum spanning tree. Given a set of n points in the plane, consider a weighted graph whose edges are all $\binom{n}{2}$ (undirected) pairs of distinct points, and edge (p_i, p_j) has weight equal to the Euclidean distance from p_i to p_j . A *minimum spanning tree* is a set of $n-1$ edges that connect the n points (into a tree) such that the total weight of edges is minimized.

Theorem 1.1. *The Euclidean minimum spanning tree of a set of points S is a subgraph of the Delaunay triangulation of S.*

The proof of this theorem is leaved as an exercise.

1.2.3. *The* `in_circle` *predicate.* Given three non-collinear points $\mathbf{a} = (a_x, a_y), \mathbf{b} = (b_x, b_y), \mathbf{c} = (c_x, c_y)$ in the plane, the geometric predicate to test whether a point $\mathbf{d} = (d_x, d_y)$ 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 non-collinear points in the plane: counterclockwise or clockwise. Assume that the three points $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are in counterclockwise order in the plane. By our choice of the matrix A, then we have

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

of the circle passing through $\mathbf{a}, \mathbf{b}, \mathbf{c}$, see Figure 7.

Remark. If $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are in clockwise order in above, then the result sign must be reversed. On the other hand, if we swap two columns in matrix A, all the signs above must be reversed.

1.3. **The lifting transformation.** There is a fascinating relation between Delaunay triangulation in $\mathbb{R}^d$ and convex hull in $\mathbb{R}^{d+1}$. These two structures appear from quite different concepts. In this section, we will establish their relation through a lifting and projecting process.

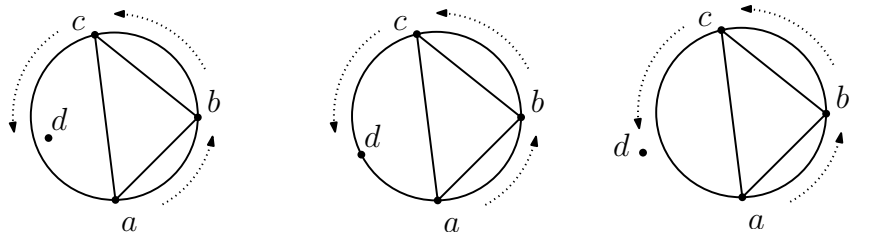

FIGURE 7. The `in_circle` test (Figure from Mount's CMSC754).

Let S be a finite 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 $\mathbf{p}'$ onto 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 8.

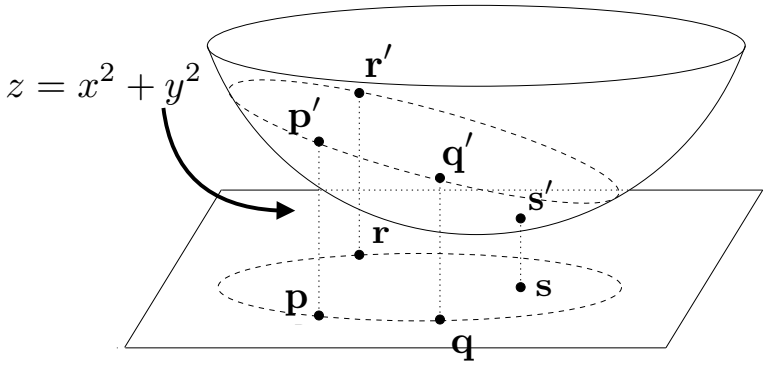

FIGURE 8. The lifting map: circles map to planes.

1.3.1. *Circles and Planes.* The following fact gives a relation between circles in $\mathbb{R}^2$ and planes in $\mathbb{R}^3$, see Figure 8.

Lemma 1.1. *Consider* 4 *distinct points* $\mathbf{p}, \mathbf{q}, \mathbf{r}, \mathbf{s}$ *in the plane in* $\mathbb{R}^2$*, and let* $\mathbf{p}', \mathbf{q}', \mathbf{r}', \mathbf{s}'$ *be their respective projections onto the paraboloid,* $z = x^2 + y^2$*. The point* $\mathbf{s}$ *lies within the circumcircle of* $\mathbf{p}, \mathbf{q}, \mathbf{r}$ *if and only if* $\mathbf{s}'$ *lies vertically below the plane in* $\mathbb{R}^3$ *passing through* $\mathbf{p}', \mathbf{q}', \mathbf{r}'$*.*

Proof. Let H be the plane passing through the lifted points $\mathbf{p}', \mathbf{q}', \mathbf{r}'$. H cuts the paraboloid into three parts, a patch below H, an ellipse in H, and a patch above H. We show that the projection of the ellipse in H is just the circumcircle of $\mathbf{p}, \mathbf{q}, \mathbf{r}$.

We find the equation of H as follows. Let H_t be the plane which is parallel to H and is tangent to the paraboloid. Let the tangent point be $\mathbf{c} = (c_x, c_y, c_x^2 + c_y^2)$. Using the fact, "the slope (normal) of a tangent plane is the gradient of the function at its tangent point". Let $f = x^2 + y^2 - z$, the gradient of f at $\mathbf{c}$ is: $\nabla f|_{\mathbf{c}} = (2c_x, 2c_y, -1)$. Then the equation of H_t is:

$$2c_x(x - c_x) + 2c_y(y - c_y) - (z - (c_x^2 + c_y^2)) = 0,$$

which is

$$z = 2c_x x + 2c_y y - (c_x^2 + c_y^2).$$

We shift H_t upwards a positive amount r^2 we get H,

$$z = 2c_x x + 2c_y y - (c_x^2 + c_y^2) + r^2.$$

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

$$x^2 + y^2 = 2c_x x + 2c_y y - (c_x^2 + c_y^2) + r^2.$$

This is equivalent to

$$(x - c_x)^2 + (y - c_y)^2 = r^2.$$

This is a circle centered at (c_x, c_y) with a radius r. It is the circumcircle of $\mathbf{p}, \mathbf{q}, \mathbf{r}$.

Thus, we see that the intersection of an arbitrary lower halfspace with the paraboloid, when projected onto the (x, y) plane, is the circle's interior. The point $\mathbf{s}$ lies within this circumcircle, if and only if its lifting $\mathbf{s}'$ onto the patch of paraboloid which lies below the plane H passing through $\mathbf{p}', \mathbf{q}', \mathbf{r}'$.

□

1.3.2. *Delaunay triangulation and Convex hull.* Due to the above fact, we can show the nice relation between Delaunay triangulation and convex hull.

Let S' be the set of all sites resulting from 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. (In other words, this plane separates the $\text{conv}(S')$ and the viewpoint at $(0, 0, -\infty)$.) 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 *simplicial complex* $\mathcal{T}$ which is the *Delaunay triangulation* of S, see Figure 9.

1.3.3. *Voronoi diagram and convex hull.* Given a point $\mathbf{p} = (a, b)$, the hyperplane $H(\mathbf{p})$ that is tangent to $\mathbf{p}$'s lifting, namely, $(a, b, a^2 + b^2)$, has the equation

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

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

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

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

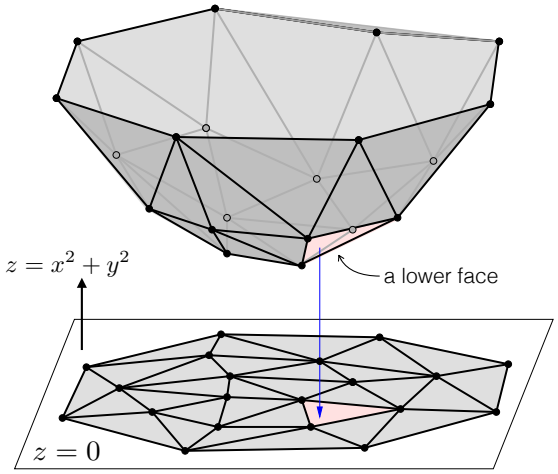

FIGURE 9. The projection of the lower faces of the convex hull is the Delaunay triangulation.

Consider two points $\mathbf{p}_1$ and $\mathbf{p}_2$ in the plane $z = 0$. We claim that $\mathbf{q}$ is closer to $\mathbf{p}_1$ if and only if at the position $\mathbf{q} = (\alpha, \beta)$, the plane $H(\mathbf{p}_1)$ lies *above* (closer to the paraboloid) than $H(\mathbf{p}_2)$. It simply follows from the above vertical distance formula.

Lemma 1.2. *Let $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ be a set of points in the plane $z = 0$. A point $\mathbf{q}$ belongs to the Voronoi cell of the point $\mathbf{p}_i$, if and only if and only if $H(\mathbf{p}_i)$ is the highest plane (seen from $z = +\infty$) at $\mathbf{q}$.*

Therefore, the Voronoi diagram of $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ is simply the vertical projection, down to $z = 0$, of the point-wise maxima of the downward-facing half spaces $H(\mathbf{p}_i)$. Or equivalent is the uppermost face of the arrangement defined by these planes.

2. Lawson's flip algorithm

This section introduces the locally Delaunay condition for edges and proves the Delaunay lemma, which shows a crucial local property of Delaunay triangulations. This Lemma suggests that one can construct Delaunay triangulation from any triangulation by a sequence of edge flips. A classical Lawson's flipping algorithm is introduced. This algorithm's correctness implies two fundamental results of planar triangulations, (1) among all triangulations of the same point set, and the Delaunay triangulation maximizes the minimum angle; and (2) the set of all triangulations of the point set is connected by edge flips.

2.1. The Delaunay lemma.

This section introduces a local condition for edges, shows it implies a triangulation is Delaunay.

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

(i) it is on the convex hull, or
(ii) it belongs two triangles, $t_{\mathbf{abc}}, t_{\mathbf{abd}} \in \mathcal{K}$, and $\mathbf{c}$ lies outside the circumcircle of $t_{\mathbf{abd}}$. (It is equivalent to say that $\mathbf{d}$ lies outside the circumcircle of $t_{\mathbf{abc}}$.)

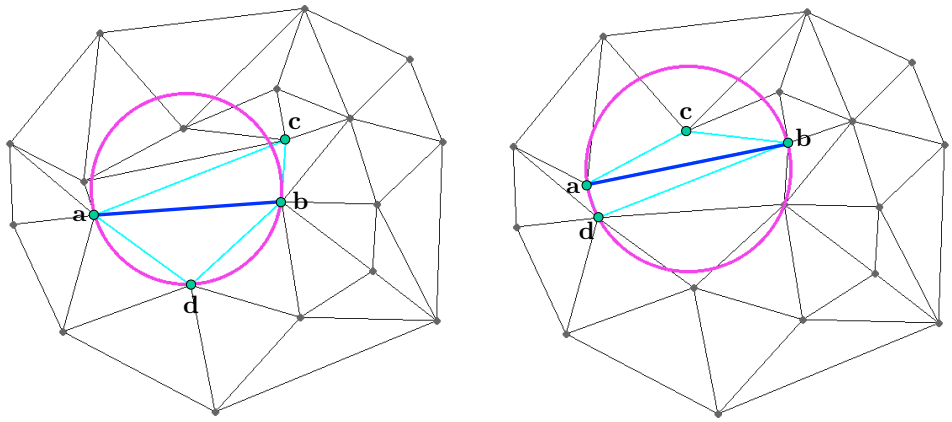

FIGURE 10. Locally Delaunay edges. In the left, the edge $e_{\mathbf{ab}}$ is locally Delaunay. In the right, the edge $e_{\mathbf{ab}}$ is not locally Delaunay.

This definition is illustrated in Figure 10. A locally Delaunay edge is not necessary a Delaunay edge. For an example, the edge $e_{\mathbf{ab}}$ in Figure 10 left is locally Delaunay but it is not Delaunay. The Delaunay lemma shows that if every edge is locally Delaunay, then the triangulation must be Delaunay.

Theorem 2.1 (Delaunay Lemma)**.** *If every edge of $\mathcal{K}$ is locally Delaunay, then $\mathcal{K}$ is the Delaunay triangulation of S.*

Proof. Consider any triangle $t_{\mathbf{abc}} \in \mathcal{K}$ and another vertex $\mathbf{p} \in \mathcal{K}$. We will show that if all edges of $\mathcal{K}$ are locally Delaunay, then $\mathbf{p}$ must lie outside (or on) the circumcircle of $t_{\mathbf{abc}}$.

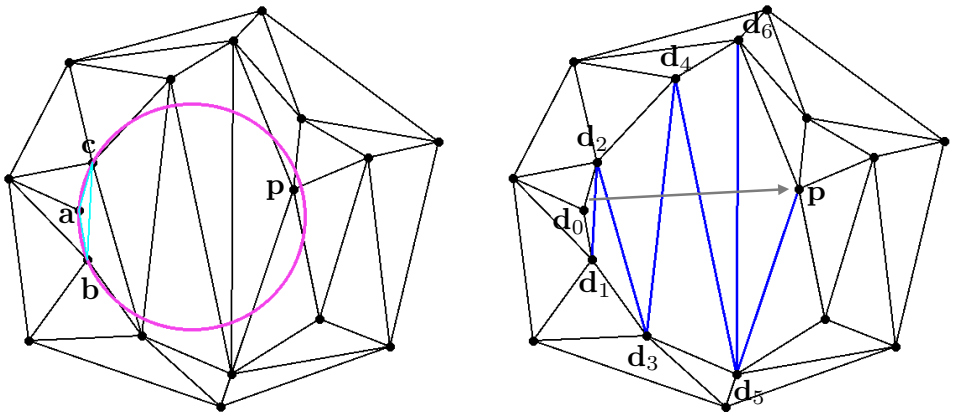

FIGURE 11. Proof of Delaunay lemma. Left: The circumcircle of a triangle $t_{\mathbf{abc}} \in \mathcal{K}$ contains a vertex $\mathbf{p} \in \mathcal{K}$. Right: The sequence of adjacent triangles.

Assume $\mathbf{p}$ lies in the inside of the circumcircle of $t_{\mathbf{abc}}$, see Figure 11 Left. Choose a point $\mathbf{x}$ in $t_{\mathbf{abc}}$. The line segment $l_{\mathbf{xp}}$ intersects a sequence of triangles in $\mathcal{K}$, which are:

$$t_{\mathbf{abc}} = t_{\mathbf{d}_0\mathbf{d}_1\mathbf{d}_2}, t_{\mathbf{d}_1\mathbf{d}_2\mathbf{d}_3}, \ldots, t_{\mathbf{d}_{m-2}\mathbf{d}_{m-1}\mathbf{p}},$$

such that every two adjacent triangles in this sequence shares a common edge. By assumption, all common edges in this sequence are locally Delaunay, see Figure 11 Right.

A key observation is the following fact. Since the circumcircle of $t_{\mathbf{abc}} = t_{\mathbf{d}_0\mathbf{d}_1\mathbf{d}_2}$ encloses $\mathbf{p}$, and since the edge $e_{\mathbf{d}_1\mathbf{d}_2}$ is locally Delaunay, then the circumcircle of its adjacent triangle $t_{\mathbf{d}_1\mathbf{d}_2\mathbf{d}_3}$ must also encloses $\mathbf{p}$, see Figure 12.

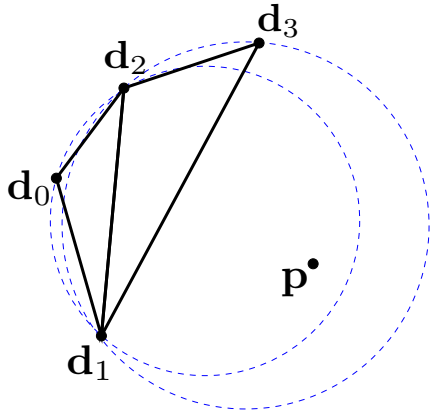

FIGURE 12. Proof of Delaunay lemma. The circumcircle of $t_{\mathbf{d}_0\mathbf{d}_1\mathbf{d}_2}$ encloses $\mathbf{p}$ implies that the circumcircle of its adjacent triangle, which is $t_{\mathbf{d}_1\mathbf{d}_2\mathbf{d}_3}$, in this sequence also encloses $\mathbf{p}$.

This can be proven using the circle and plane relation. $\mathbf{p}$ lies in the circumcircle of $t_{\mathbf{d}_0\mathbf{d}_1\mathbf{d}_2}$ implies that the lifted point $\mathbf{p}' \in \mathbb{R}^3$ lies below the plane h passing through $\mathbf{d}_0', \mathbf{d}_1', \mathbf{d}_2'$. The edge $e_{\mathbf{d}_1\mathbf{d}_2}$ is locally Delaunay implies that the lifted point $\mathbf{d}_3'$ lies above the plane h. Moreover, both $\mathbf{d}_3$ and $\mathbf{p}$ lie on the same side of the edge $e_{\mathbf{d}_1\mathbf{d}_2}$. Therefore $\mathbf{p}'$ must lie below the plane passing through the lifted points $\mathbf{d}_1', \mathbf{d}_2', \mathbf{d}_3'$. This shows that $\mathbf{p}$ must lie inside the circumcircle of $t_{\mathbf{d}_1,\mathbf{d}_2,\mathbf{d}_3}$.

By this fact, we will find that the circumcircle of the second last triangle, i.e., $t_{\mathbf{d}_{m-3}\mathbf{d}_{m-2}\mathbf{d}_{m-1}}$, in this sequence must contain $\mathbf{p}$. Therefore, the edge $e_{\mathbf{d}_{m-2}\mathbf{d}_{m-1}}$ is not locally Delaunay, it contradicts to our assumption.

Since we can chose any vertex $\mathbf{p} \in \mathcal{K}$. Hence we have shown that the triangle $t_{\mathbf{abc}}$ is indeed Delaunay in $\mathcal{K}$. Since we have chosen $t_{\mathbf{abc}}$ arbitrarily in $\mathcal{K}$, this means, all triangles in $\mathcal{K}$ must be Delaunay. □

2.2. **Edge flips and Lawson's algorithm.**

2.2.1. *Edge flips.* Let $\mathcal{K}$ be a triangulation of a point set S in the plane. Given an interior edge $e_{\mathbf{ab}} \in \mathcal{K}$, let the two triangles sharing at this edge are $t_{\mathbf{abc}}$ and $t_{\mathbf{abd}}$. If the union of these two triangles is a convex quadrangle, we can *flip* this edge $e_{\mathbf{ab}}$ to another edge $e_{\mathbf{cd}}$ within this the convex hull of $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$. Formally, this means we remove the

simplices $e_{\mathbf{ab}}$, $t_{\mathbf{abc}}$, and $t_{\mathbf{abd}}$ from $\mathcal{K}$, and add the simplices $e_{\mathbf{cd}}$, $t_{\mathbf{cda}}$, and $t_{\mathbf{cdb}}$ to the triangulation, see Figure 13.

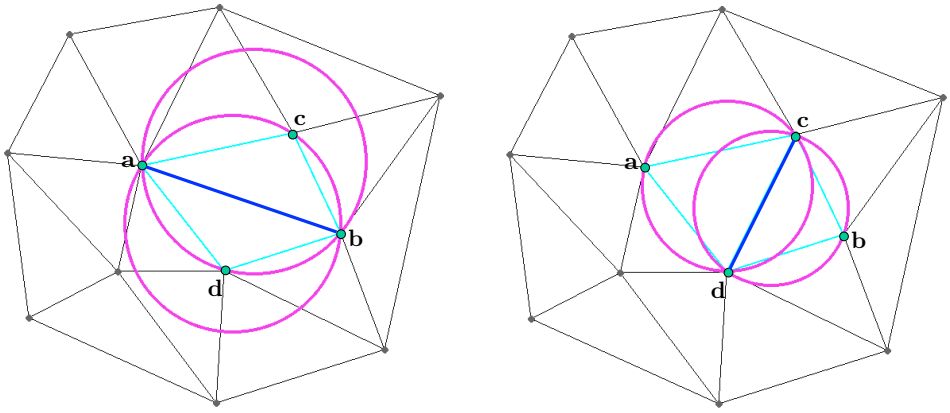

FIGURE 13. The edge flips between $e_{\mathbf{ab}}$ and $e_{\mathbf{cd}}$. Moreover, if $e_{\mathbf{ab}}$ is not locally Delaunay, then $e_{\mathbf{cd}}$ must be locally Delaunay.

It is easy to show the following fact: If the edge $e_{\mathbf{ab}} \in \mathcal{K}$ is not locally Delaunay, then the sum of angles at its two opposite vertices $\mathbf{c}$ and $\mathbf{d}$, $\angle\mathbf{acb} + \angle\mathbf{adb} > 180^o$. (If this sum equals to 180^o, this means they are cocircluar). Then the angle sum at vertices $\mathbf{c}$ and $\mathbf{d}$, $\angle\mathbf{cad} + \angle\mathbf{cbd} < 180^o$. This implies that after the edge flip, the new edge $\mathbf{cd}$ is locally Delaunay.

2.2.2. *Description of the algorithm.* We can use edge flips as elementary operations to convert an arbitrary triangulation $\mathcal{K}$ to the Delaunay triangulation. This algorithm was first developed by C. Lawson [9, 10] [1].

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

FIGURE 14. The Lawson edge-flip algorithm.

The algorithm is described in Figure 14. It uses a stack to maintains edges of the triangulation, which need to be checked whether they are locally Delaunay or not. Initially, all edges of $\mathcal{K}$ are pushed on the stack. It then runs in a loop to process these

[1]The edge flip algorithm was proposed by Lawson in 1972 to prove a theorem that states that any two triangulations of a point set are connected by a sequence of edge flips. A later paper in 1977, he proved that the Delaunay triangulation could be obtained in this way.

edges in the stack. Each time, an edge $e_{\mathbf{ab}}$ is pop up from the stack. This edge may have already been flipped; then, it simply goes to the next edge. Otherwise, if this edge is not locally Delaunay, then apply an edge flip to remove this edge and create a new edge $e_{\mathbf{cd}}$, which is locally Delaunay. After this flip, the four edges at the boundary of the quadrilateral $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ might become not locally Delaunay. They are pushed into the stack. The algorithm stops when the stack is empty.

Figure 15 shows an example of an input triangulation and an output (which is the Delaunay triangulation) of this algorithm.

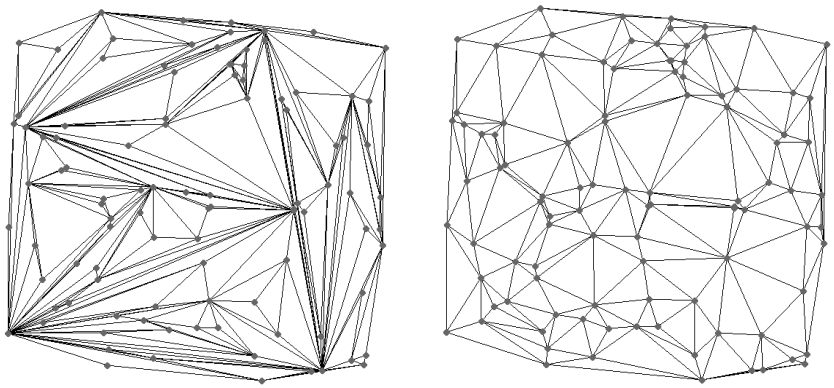

FIGURE 15. Lawson's flip algorithm takes an arbitrary triangulation (left) as input and returns the Delaunay triangulation (right).

2.2.3. *Correctness and termination.* The algorithm can be understood as gluing a sequence of tetrahedra. Flipping $e_{\mathbf{ab}}$ to $e_{\mathbf{cd}}$ is likely gluing a tetrahedron $t_{\mathbf{a'b'c'd'}}$ from below to the faces $t_{\mathbf{a'b'c'}}$ and $t_{\mathbf{a'b'd'}}$, see Figure 16.

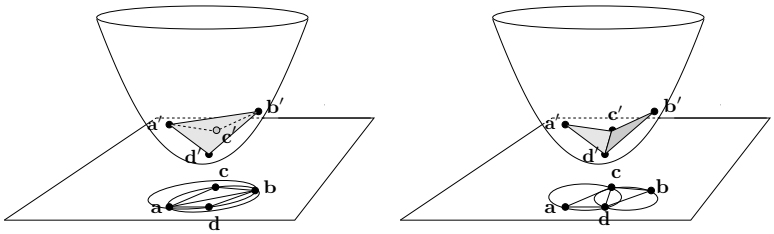

FIGURE 16. Left: a locally non-Delaunay edge corresponds to a non-convex edge in the lifted point set. Right: a locally Delaunay edge lies on the convex hull of the lifted point set. (Figures by B. Gärtner ETHZ)

Once we glue $t_{\mathbf{a'b'c'd'}}$ we cannot glue another tetrahedron right below the lifted edge $e_{\mathbf{a'b'}}$. In other words, once we flip $e_{\mathbf{ab}}$, we cannot introduce $e_{\mathbf{ab}}$ again by some other flip. This fact implies that this algorithm will eventually terminate when all locally non-Delaunay edges are flipped. By the Delaunay lemma, the triangulation is Delaunay.

The above fact also implies there are at most as many flips as edges are connecting n points, namely $\binom{n}{2}$. Each flip takes constant time. Hence the total running time is $O(n^2)$.

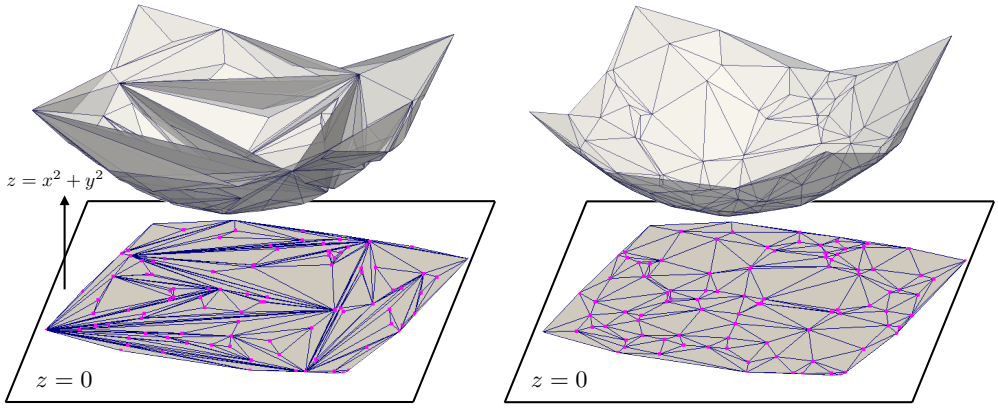

FIGURE 17. Top: Bottom: The lifted view of the Lawson's flip algorithm which transforms a non-convex surface (left) in 3d into a convex one (right).

This algorithm is indeed a convex optimization algorithm. It transforms a lifted surface triangulation, which is not convex, into a convex surface in $\mathbb{R}^3$ whose projection into the plane is the Delaunay triangulation of this point set. Figure 17 shows both the lifted and projected triangulations.

2.3. **Optimal properties of Delaunay triangulations.** The correctness of Lawson's flip algorithm implies several optimal properties of the Delaunay triangulation.

2.3.1. *The MaxMin angle property.*

Theorem 2.2. *Among all triangulation of a finite point set $S \subset \mathbb{R}^2$, the Delaunay triangulation maximizes the minimum angle.*

Proof. Each flip substitutes two new triangles for two old triangles. It, therefore, changes six of the angles, see Figure 18 Left. The six old angles are:

$$a_1, b_1, a_2, b_2, a_1 + a_2, b_1 + b_2$$

and the six new angles are

$$c_1, d_1, c_2, d_2, c_1 + c_2, d_1 + d_2$$

We show that an old angle is at least as small for each of the six new angles.

Both c_1 and a_2 are opposite the same edge $e_{\mathbf{bd}}$. The locally Delaunay property of the edge $e_{\mathbf{cd}}$ implies that $\mathbf{a}$ lies outside the circumcircle of $t_{\mathbf{cdb}}$. Therefore, the new angle c_1 must larger than the old angle a_2, see Figure 18 Right. By the same reason, we have $d_1 \geq a_1$, $c_2 \geq b_2$, and $d_2 \geq b_1$.

It follows that an edge flip in Lawson's algorithm does not decrease the smallest angle in a triangulation. Since we can transform any triangulation $\mathcal{K}$ of S into the Delaunay triangulation by a sequence of the same kind of flips, this implies that the smallest angle in $\mathcal{K}$ is no larger than the smallest angle in the Delaunay triangulation. □

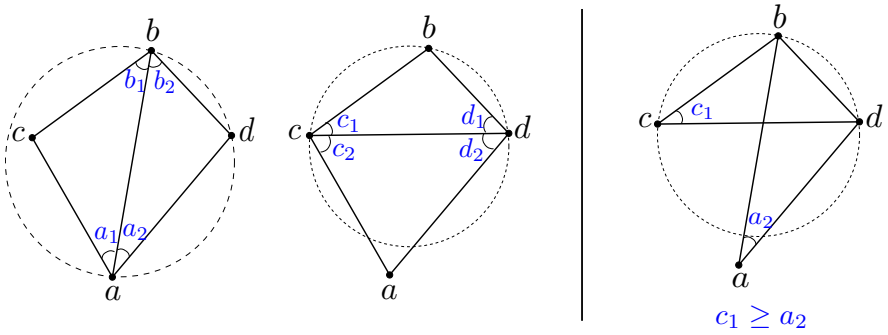

FIGURE 18. Flipping edge $e_{\mathbf{ab}}$ to $e_{\mathbf{cd}}$ improves the minimum angle.

2.3.2. *Dirichlet energy of piecewise linear interpolating functions.* Consider the problem of interpolating a two-dimensional function using a triangulation. Given a triangulation of this function's domain, one can construct the piecewise linear interpolating function to the given function values. The geometry of this interpolating function is a piecewise linear surface with triangles whose vertices are the vertices of the planar triangles. Figure 17 shows two piecewise linear interpolating functions for the quadratic function $f = x^2 + y^2$.

A point set has many different triangulations. A natural question is: which triangulation is the best for the piecewise linear interpolating of a given function?

The *Dirichlet energy* of a function $g : \Omega \to \mathbb{R}$ is the L^2 norm squared of the gradient of the function, i.e.,

$$E(g) = \frac{1}{2} \int_{\Omega} \|\nabla g(x)\|^2 \mathrm{d}x.$$

It is a measure of how variable a function is. More abstractly, it is a quadratic functional on the Sobolev space H^1. It is named after the German mathematician Peter Gustav Lejeune Dirichlet (1805 – 1859). The Dirichlet energy is intimately connected to Laplace's equation. Solving Laplace's equation $-\Delta u(x) = 0$ for all $x \in \Omega$, subject to appropriate boundary conditions, is equivalent to solving the variational problem of finding a function u that satisfies the boundary conditions and has minimal Dirichlet energy.

This energy depends on the triangulation (which is used to form the piecewise linear interpolating function). Rippa [12] proves the following theorem.

Theorem 2.3 (Rippa, 1990)**.** *The Delaunay triangulation minimizes Dirichlet energy of a piecewise linear interpolating function, for any fixed set of function values.*

The key to proving the above theorem is a geometry condition given by the following Lemma proven by Rippa [12]. It shows that within a convex quadrilateral, the Delaunay triangulation's local Dirichlet energy must smaller than the non-Delaunay triangulation. Therefore, by the termination of Lawson's algorithm, the Delaunay triangulation minimizes the global Dirichlet energy.

2.4. **The (undirected) flip graph.** One can use flips to traverse the set of all triangulations of a point set 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 19 shows an example. The termination of Lawson's flip algorithm implies that the flip-graph for any point set in the plane is connected, i.e., one can go from any triangulation of S to any other triangulation.

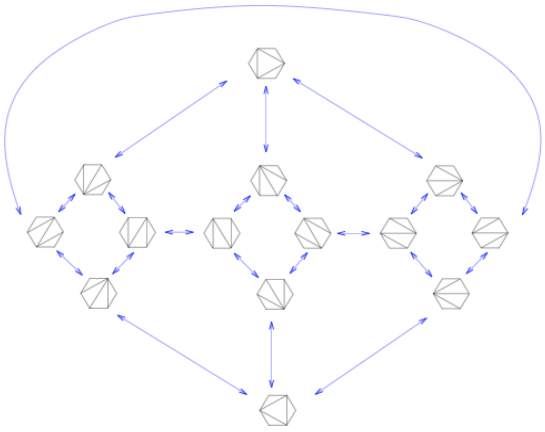

FIGURE 19. The flip graph of a set of the vertex set of a convex 6-gon.

Since this flip graph contains all triangulations of a point set, there are many interesting properties of this graph to be studied; refer to the work of Hurtado et al [8]. Here we show an example that needs $O(n^2)$ edge flips. This is a special construction (by P. Bose and F. Hurtado [2]) of a non-convex polygon shown in Figure 20. We have a sequence of $n-1$ ones and $n-1$ zeros. A flip is possible between a 1 triangle and a 0 triangle. The two adjacent numbers are switched.

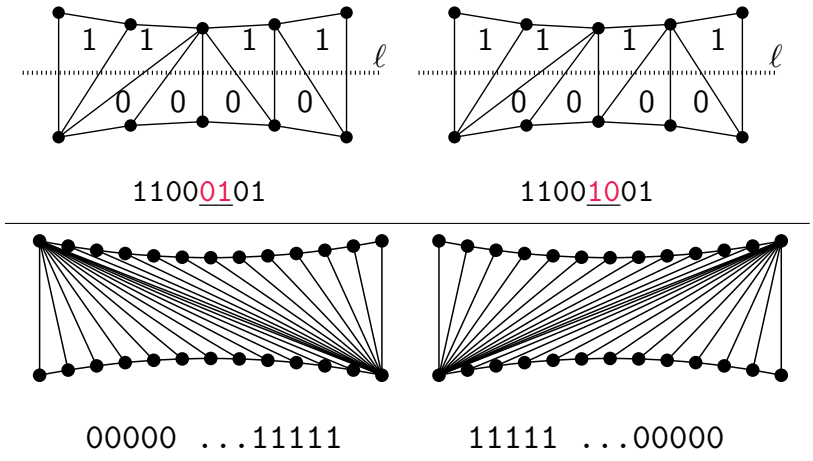

FIGURE 20. A lower bound example (Figures from A. Pilz (IST TU-Graz)).

Let the starting triangulation be the one on the bottom left, and the final triangulation is on the bottom right. There are $n-1$ zeros and $n-1$ ones. Therefore we need at least $(n-1)^2$ flips.

3. Randomized incremental flip algorithm

This section introduces an algorithm that constructs Delaunay triangulations incrementally, using edge flips and randomization. This algorithm is simple and easy to implement. Moreover, it is also efficient. The expected runtime is $O(n \log n)$. After explaining the algorithm, we present a detailed analysis of the expected running time.

3.1. **Inserting a vertex.** Let $\mathcal{T}$ be the Delaunay triangulation of a point set S in the plane. Let $\mathbf{p}$ be the new vertex to be inserted. Assume that $\mathbf{p}$ lies inside this triangulation. Then there exists a triangle $t_{\mathbf{abc}} \in \mathcal{T}$ which contains $\mathbf{p}$. We further assume that $\mathbf{p}$ lies strictly in the interior of this triangle. The simplest vertex insertion is just an elementary flip, 1-to-3 flip, which deletes the triangle $t_{\mathbf{abc}}$ and creates three new triangles: $t_{\mathbf{abp}}$, $t_{\mathbf{bcp}}$, and $t_{\mathbf{cap}}$ in $\mathcal{T}$, see Figure 21. The reverse of 1-to-3 flip is another elementary flip, the 3-to-1 flip, which delete a vertex.

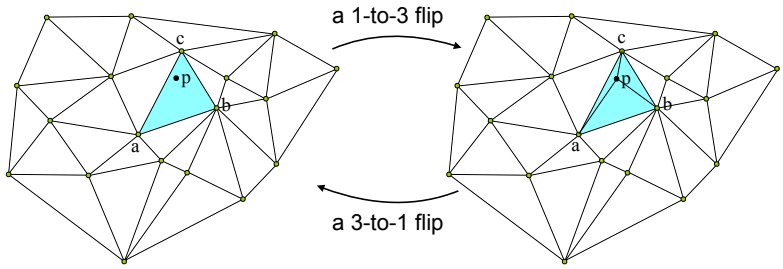

FIGURE 21. Two elementary flips in the plane. From left to right is a 1-to-3 flip which inserts a vertex. The reverse is a 3-to-1 flip which deletes a vertex.

A special (degenerate) case is when $\mathbf{p}$ lies on an edge of the triangle $t_{\mathbf{abc}}$. Then the above 1-to-3 flip will create a degenerate triangle (with a zero area). In this case, one can immediately perform a 2-to-2 edge flip to remove the degenerate triangle. This process replaces two triangles by four, i.e., it is a 2-to-4 flip. However, it is not elementary. It is a combination of a 1-to-3 flip (vertex insertion) and a 2-to-2 flip (edge swap).

3.2. **Description of the algorithm.** The primary 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. For simplicity, we start with a triangulation $\mathcal{D}_0$ that consists of a single and sufficiently large triangle $t_{\mathbf{xyz}}$. This triangle must enclose all vertices in its interior. The algorithm is a `for-loop`, adding the points in sequence. At the moment, we will assume the general position, i.e., each vertex $\mathbf{p}_i$ lies precisely in the interior of a triangle. After adding a vertex, the algorithm uses edge flips to satisfy the Delaunay lemma before the next point is added. The algorithm is given in Figure 22.

After the insertion of the new vertex $\mathbf{p}_i$ into $\mathcal{D}_{i-1}$, Lawson's flip algorithm is used to transform $\mathcal{D}_{i-1}$ into the Delaunay triangulation of S_i. Recall that this algorithm flips all

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 $\mathcal{D}$ of S;
1 initialize $\mathcal{D}_0$ with only one larger triangle $t_{\mathbf{xyz}}$;
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 a 1-3 flip;
5 initial the stack L with link edges of $\mathbf{p}_i$;
6 LawsonFlip(L);
7 **endfor**
8 remove all triangles containing $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ from $\mathcal{D}_n$;

FIGURE 22. The incremental-flip algorithm.

edges which are not locally Delaunay in current triangulation. We will take advantage of the following facts after the insertion of $\mathbf{p}_i$:

- Any edge in $\mathcal{D}_{i-1}$, which is not a link edge of $\mathbf{p}_i$ remains locally Delaunay. The edges inside the star of $\mathbf{p}_i$ are locally Delaunay as well.
- Only link edges of $\mathbf{p}_i$ in $\mathcal{D}_{i-1}$ might not be locally Delaunay.

By these facts, the initial stack L only contains the link edges of $\mathbf{p}_i$. Lawson's flip algorithm guarantees that the termination of the flip process and the result is the Delaunay triangulation $\mathcal{D}_i$ of S_i. Figure 23 shows an example of this algorithm.

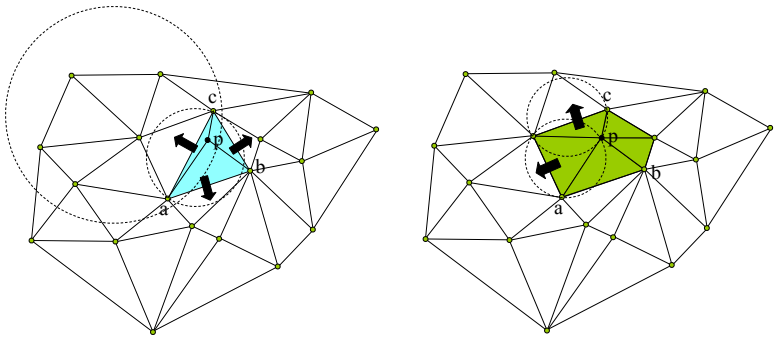

FIGURE 23. Recovery Delaunay property by the Lawson's flip algorithm.

3.3. **Worst-case running time.** 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.

This section focuses on (2) and considers the worst case. It estimates the maximum number of flips that may be performed within this algorithm.

Any triangle in $\mathcal{D}_{i-1}$ whose circumcircle does not contain the new vertex $\mathbf{p}_i$ remains a Delaunay triangle in $\mathcal{D}_{i-1}$. This fact shows that every new triangle in $\mathcal{D}_i$ must have $\mathbf{p}_i$ as a vertex. This implies that all flips occur right around $\mathbf{p}_i$. Each edge flip increases

the degree of $\mathbf{p}$ by 1. Hence the total number of edge flips for the insertion of $\mathbf{p}_i$ is proportional to the degree of $\mathbf{p}_i$, $\deg(\mathbf{p}_i)$.

It is possible that $\deg(\mathbf{p}_i) = \Theta(i)$. Figure 3.3 shows 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 $\Theta(i)$, hence the total number of flips is

$$\Theta(3 + 4 + \cdots + n) = \Theta(n^2).$$

This shows that if the vertices' insertion order is "poorly" chosen, the incremental flip algorithm can take $\Theta(n^2)$ time.

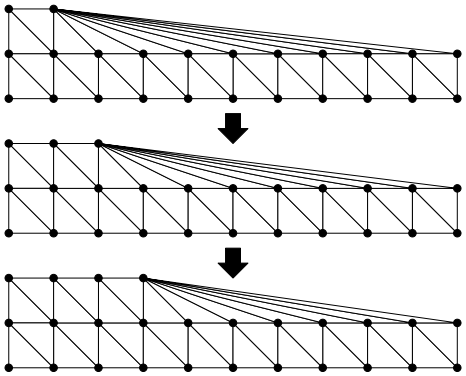

FIGURE 24. Each vertex insertion can cause $\Theta(n)$ flips (example from [**?**]).

3.4. **The expected number of flips.** This section shows that 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. Let $S^1, \ldots, S^m$ be the $m = n!$ permutations of n points of S. We assume that the probability of every permutation is equally likely to be chosen by the incremental flip algorithm. Then each S^i has probability $\frac{1}{n!}$. The sample space is a set of discrete outcomes $\mathcal{S} := \{S^i \,|\, i = 1, \cdots, n!\}$. We want to know what is the expected total number of flips of this algorithm. Let f_i be the number of total flips produced by the algorithm using S^i as the input sequence. Define the random variable $F : \mathcal{S} \to \mathbb{R}$ such that $F(S^i) = f_i$. The expectation (the average value) of the random variable F is:

$$E[F] := \sum_{i=1}^{m} f_i \frac{1}{n!} = \frac{1}{n!} \sum_{i=1}^{m} f_i.$$

The expected total number of flips of this algorithm is all the individual total number of flips $f_1 + f_2 + \cdots + f_m$ divided by $n!$.

3.4.1. *Backward analysis.* It is, in general, not possible to know the total number of flips produced by each input sequence f_i. The technique we use to analyze the algorithm is called the backward analysis developed by Seidel [13].

Consider inserting the last point $\mathbf{p}_n$. The sum of all possible last points' degrees is the same as the sum of the 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:

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

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:

$$\begin{array}{rcll} \sum_{i=1}^{m} f_i & = & f_m + f_{m-1} + \cdots + f_1, & (m = n! \text{ terms}) \\ & = & F_n + F_{n-1} + \cdots + F_1, & (n \text{ terms}) \\ & \leq & 3n \cdot (n-1)! + 3(n-1)(n-1)! + \cdots + 0, & (n \text{ terms}) \\ & \leq & 3n \cdot n \cdot (n-1)! & \\ & = & 3n \cdot n! & \end{array}$$

The expected number of edge flips for adding n points is

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

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 $O(n)$.

3.5. **Point location.** Point location is needed before we can insert a point into a triangulation. It is an essential fact for determining the total running time of this incremental algorithm.

3.5.1. *Straight line searching.* A simple point location scheme is "straight-line walk". More precisely, the algorithm is starting from an arbitrary triangle $\sigma \in \mathcal{D}_{i-1}$, and search the triangle τ that containing $\mathbf{p}_i$ by walking along the ray starting from an interior point of σ toward to $\mathbf{p}_i$, see Fig 25.

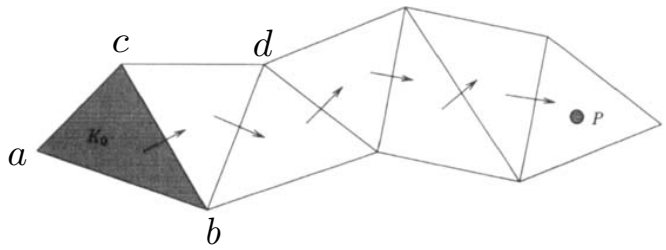

FIGURE 25. Search a point $\mathbf{p}$ by a line search starting from an arbitrary triangle. Here the starting triangle is $t_{\mathbf{abc}}$.

How much time will this searching algorithm use? We assume that each triangle is visited only once. Each point location will visit at most the number of triangles of the triangulation, which is less than $2n$. In the worst case, a single point location may visit $O(n)$ triangles. An example is shown by Devillers et al. [5, Fig. 1]. Hence the location of n points this algorithm may take $O(n^2)$ time. With this point location algorithm, the randomized incremental flip algorithm still has the expected $O(n^2)$ runtime.

3.5.2. *Using a history graph.* Most theoretical algorithms build additional data structure for point location. A way to improve the point location is to build a history graph point-location data structure (as a helper). A simple approach is based on maintaining the uninsured points in a set of *buckets*. Think of each triangle of the current triangulation as a *bucket* that holds all the uninsured points that lie inside this triangle. Whenever a triangle is split, or an edge is flipped, some old triangles are destroyed and replaced by new triangles. When this happens, lump together all the deleted triangles' points and re-distribute them into the new triangles. The number of new triangles is $O(1)$ (3 for split and 2 for a flip). This process requires $O(1)$ time for each point that is re-bucketed. The expected runtime of point location using "bucketing" is $O(n \log n)$, see [**?**].

Despite the optimal runtime guarantee of using a history graph, it needs to maintain an additional data structure within the algorithm. It requires some extra work to implement.

3.5.3. *Biased randomised insertion order (BRIO).* Amenta, et al [**?**] proposed the *Baised randomized insertion order* (BRIO). The main idea of BRIO is to remove enough randomness to improve performance significantly but leaves enough randomness so that the algorithms remain theoretically optimal.

Let P be a set of n points. The BRIO first sorts the points into $\lg n$ rounds and inserts rounds of points one after one. For simplicity, we assume n is a power of 2.

We choose each point independently with probability $1/2$ to be inserted in the final round to allocate points to rounds. We choose each of the remaining points independently with probability $1/2$ to be inserted in the next-to-last round, and so on. When we get to the first round, we choose any remaining points with probability one. The probability that a point is chosen in round $i > 0$ is $2^{i-1}/n$, and the remaining probability $1/n$ goes to the event that the point is chosen in round zero.

Now consider all points within each round. The simplest way is to insert them in an arbitrary order. For example, we can order the points randomly. It is possible to further improve the locality by organizing the points into *blocks*, which respect locality in space, such that near points are in the same block, then insert points block by block. There are many ways to order points in block, such as quad-tree or *kd*-tree. In the next section, we introduce an ordering of points based on the Hilbert curve.

The intuition of BRIO is that in the early rounds (with few points), the insertions tend to be sprinkled nearly randomly across all the data, producing a nicely balanced data structure. In contrast, the later rounds (with many points) are grouped by blocks, accessing local regions of the data structure mostly independently.

Notice that this ordering only indirectly attacks locality in the virtual memory structure layout, which is a fundamental problem. The hope is that inserting points with

the locality in virtual memory. However, this depends on the specific program's storage management scheme, which may be hard to predicate.

3.5.4. *The expected running time.* The expected running time of the randomized incremental flip algorithm using bucketing for point sorting is determined by (1) $O(n \log n)$ time for total point location together with (2) $O(n)$ time for flips. Hence its expected running time is $O(n \log n)$.

Exercises

1. Draw the Voronoi diagram and Delaunay triangulation for a set of 10 randomly distributed points in the plane.
2. Proposition: Every finite set of points in the plane has a Delaunay triangulation.
   (2.1) Prove this proposition.
   (2.2) Show an example that a point set in the plane has more than one Delaunay triangulation.
3. An acute triangle has all three angles less than $\pi/2$.
   (3.1) Prove that a triangulation $\mathcal{K}$ all of whose triangles are acute is the Delaunay triangulation of its vertex set.
4. Prove Theorem 1.1, which states that all edges of every Euclidean-distance weighted minimum spanning tree belong to the Delaunay triangulation of the same point set.
5. Given a Delaunay triangulation $\mathcal{K}$ and a point $\mathbf{p}$ inside one of the triangles of $\mathcal{K}$. A simple way to locate $\mathbf{p}$ is to start an arbitrary point $\mathbf{q} \in \mathcal{K}$, and search $\mathbf{p}$ by performing a line search, see Fig 25.

   If $\mathcal{K}$ is not a Delaunay triangulation, this algorithm may not find the point $\mathbf{p}$. An example of such triangulation and such a sequence are given in Fig. 26.

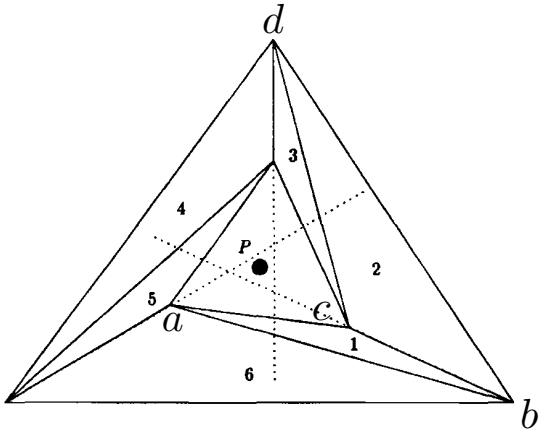

Figure 26. A cyclic configuration. Starting from the triangle $t_{\mathbf{abc}}$ to line search the point $\mathbf{p}$. The sequence of triangles $t_1, \ldots, t_6$ form a loop.

Question: How to overcome this situation and make this algorithm work correctly? (Hint: Read the reference by Devillers, O., Pion, S., and Teillaud, M. Walking in triangulation.)

6. The Bowyer-Watson algorithm [3, 16] is also an incremental algorithm to construct Delaunay triangulations. It is different from the incremental flip algorithm in the vertex insertion step after the point location of $\mathbf{p}_i$, see Figure 27:
   (1) it finds all triangles in the current triangulation whose circumcircles contain $\mathbf{p}_i$ in their interiors.
   (2) remove all these triangles, which creates an empty region, called *cavity*, in $\mathcal{D}_{i-1}$; and
   (3) $\mathcal{D}_i$ is created by filling the cavity with new triangles that contain the new vertex.

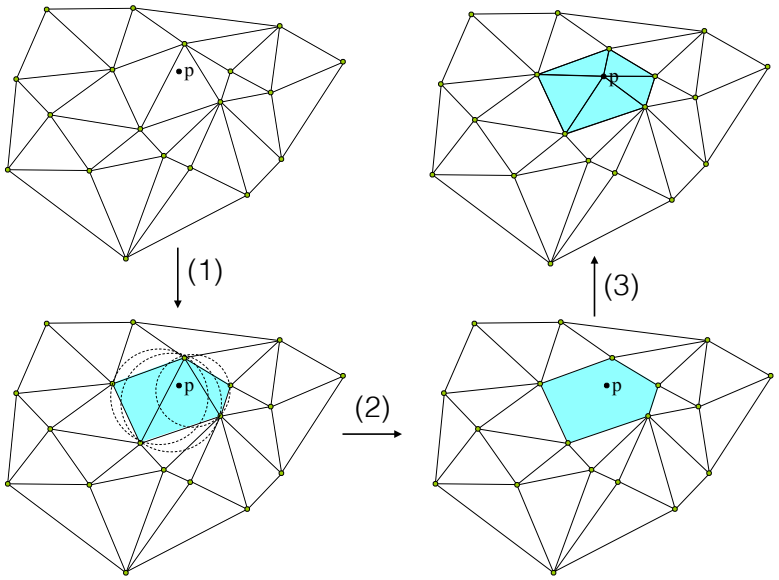

FIGURE 27. The vertex insertion steps of the Bowyer-Watson algorithm [3, 16].

   (6.1) Prove the correctness of the Bowyer-Watson algorithm for incremental constructing Delaunay triangulations in the 2d case. [Hint: use the idea of lifting map to show it].
   (6.2) What is the total cost (running time) for the vertex insertion steps for inserting all vertices.
   (6.3) Compare the efficiency of Bowyer-Watson algorithm and the incremental flip algorithm.
7. Let $D(\mathbf{c}, r)$ be a two dimensional disk with center $\mathbf{c} \in \mathbb{R}^2$ and radius $r \geq 0$. Let $\mathcal{D} = \{D_1, D_2, \cdots, D_n\}$ be a set of n disk in $\mathbb{R}^2$. Define the *Voronoi cell* $V_D(D_i)$ of the disk $D_i(\mathbf{c}_i, r_i) \in \mathcal{D}$ as:

$$V_D(D_i) = \{\mathbf{p} \,|\, \|\mathbf{p} - \mathbf{c}_i\| - r_i \leq \|\mathbf{p} - \mathbf{c}_j\| - r_j, \forall D_j \in \mathcal{D}\},$$

   where $\|\mathbf{x} - \mathbf{y}\|$ means the Euclidean distance between $\mathbf{x} and \mathbf{y}$. The *Voronoi diagram* of a set of disks $\mathcal{D}$ is then defined by collecting Voronoi cells of disks together with their edges vertices.

(7.1) Show the bisector between two disks: $D_1 = (\mathbf{p}, 0.2)$ and $D_1 = (\mathbf{q}, 0.5)$, where $\mathbf{p} = (1.0, 0)$ and $\mathbf{q} = (2.0, 0)$.

(7.2) Randomly generates 5 points in the plane and assign each point randomly a radius. Hint, you could use `Detri2` to generate this. Draw the Voronoi diagram for this set of disks.

8. Let S be a set of n points in $\mathbb{R}^2$. The *furthest-point Voronoi cell*, denoted as $V_f(\mathbf{p})$, of a point $\mathbf{p} \in S$ consists of all points at least as far from $\mathbf{p}$ as from any other point in S, i.e.,

$$V_f(\mathbf{p}) = \{\mathbf{p} \,|\, \|\mathbf{x} - \mathbf{p}\| \geq \|\mathbf{x} - \mathbf{q}\|, \forall \mathbf{q} \in S\}.$$

(8.1) Show that $V_f(\mathbf{p})$ is not empty only if $\mathbf{p}$ is a vertex of the convex hull of S.

(8.2) Draw the furthest Voronoi diagram for a set S of 10 points randomly distributed in the plane, and draw the dual, which is the *furthest-point Delaunay triangulation* of S.

References

[1] Franz Aurenhammer. Voronoi diagrams – a study of fundamental geometric data structures. *ACM Comput. Surveys*, 23:345–405, 1991.

[2] Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60 – 80, 2009.

[3] Adrian Bowyer. Computing Dirichlet tessellations. *Comp. Journal*, 24(2):162–166, 1981.

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

[5] 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.

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

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

[8] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.

[9] C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

[10] C. L. Lawson. Software for c^1 surface interpolation. *Mathematical Software III, Academic Press*, pages 164–191, 1977.

[11] 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.

[12] S. Rippa. Minimal roughness property for the Delaunay triangulation. *Computer Aided Geometric Design*, 7:489–497, 1990.

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

[14] M. I. Shamos and D. Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162, Oct 1975.

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

[16] David F. Watson. Computing the n-dimensional Delaunay tessellations with application to Voronoi polytopes. *Comput. Journal*, 24(2):167–172, 1981.