Voronoi and Delaunay Techniques

Henrik Zimmer

July 30, 2005

Contents

1 Abstract

Voronoi diagrams and Delaunay triangulated meshes are important tools. Their areas of usage range from the eye-pleasing 3D modeling part of Computer Graphics to the generation of meshes for equation representation and computation areas. Voronoi diagrams for example provide means of naturally partioning space into subregions with areas of applications such as spatial data manipulation, the modelling of spatial structures, surface reconstruction and so on. The Delaunay triangulation is a tool for obtaining good quality meshes desired both in Computer Graphics and when using meshes for equations and problem representation. Many complex high dimensional problems can be expressed with graphs and a good quality mesh lets us get closer to a numerically stable solution for these problems.

During the course of this paper I will treat the generation of two special kinds of meshes. Delaunay Triangulations and Voronoi Diagrams. I will start by giving a short overview of the concept of and the need for mesh quality. The emphasis will be on Voronoi Diagrams and Delaunay Triangulations. I will present the basic theory behind Delaunay and Voronoi, show the relation between them and give examples of different methods and algorithms that can be used to obtain them.

2 Meshes

2.1 What is a mesh

A mesh can generally be described as a number of points connected in some way by lines. The mesh of a sphere can for example be the wire-frame that represents the structure of a sphere (fig.1). In Computer Graphics the points and lines of a mesh are often referred to as edges and vertices. A mesh in its most general form is like an ordinary graph without any particular rules about connectivity, overlapping and so on. Most usages for meshes however require rules and structure. For the Computer Graphics area of usage e.g. when texurizing or shading a mesh, we require a mesh without overlapping edges and holes in order to obtain good results.

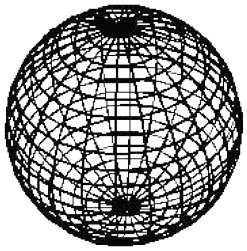

Figure 1: A Sphere wire-frame.

2.1.1 Structured and unstructured meshes

We talk about two classes of meshes, structured meshes and unstructured meshes. The structured meshes have the same topology as a square grid of triangles. In a structured mesh we have the set connectivity of the grid which this makes it easy to find the neighboring nodes of a given node, this can be done by addition. Unstructured meshes call for a list of neighbors to be stored for each node, this is due to the arbitrary connectivity between the nodes.

2.1.2 Generation

This paper will only treat the generation and treatment of unstructured meshes. The reason for using unstructured meshes is that the regularity of structured meshes is too limited. The solving of equation systems on structured meshes would be a lot easier due to the ease of finding the neighboring nodes. However unstructured meshes are indispensible when we want to use meshes for the modeling and representation of problems and equations with irregularly shaped domains.

2.2 Triangulation

2.2.1 Triangulation definition

The meaning of *triangulation* is to generate a mesh of triangles from a given set of points. The triangles are formed by edges between the points of the set. To use a triangulated mesh is a first step towards getting the rules and structure we desire for a good quality mesh.

In Computer Graphics or more generally in geometry a suitable triangulation would be one without overlapping edges and without a faulty degenerate structure. Formally, what is sought is:

A triangulation T of $\mathbb{R}^n$ is a subdivision of $\mathbb{R}^n$ into a set of n-dimensional *simplices* (triangles in 2D case), where

- any simplex face (triangle side) is shared by either one adjacent simplex (triangle) or none at all
- any bounded set in $\mathbb{R}^n$ intersects only finitely many simplices (triangles) in T

When considering vertices p of a vertex set P in space, the triangulation of P is taken as the triangulation of the *convex hull* of P, this gives us a convex outer edge of our triangulation.

2.2.2 Quality of meshes and good triangulation

One difficulty of generating meshes is to offer sufficient control over the mesh generating process, e.g. the sizes of the elements. The necessary quality requirements on meshes, when using them to solve and model problems, include: the mesh has to have a high resolution (small elements) in areas with a lot of information in order to accuratly describe the problem, the mesh should have a low resolution (large elements) in areas with little information in order to save computation time. It is necessary that the mesh generating process gives us elements of sufficiently high quality. For a numerical solution to be as accurate and stable as possible we need what can be refered to as round elements. Naturally triangles in a triangle mesh are not round, what is sought are triangles that have nor too large nor too small angles which leave as little room as possible for discretization and rounding errors. A common bound in two dimensions is to say, if no angle is smaller than Θ then no angle is larger than $\pi - 2\Theta$. That is why many methods want to bound the smallest angle. This is exactly what the Delaunay triangulation does, it even maximizes the minimum angle among all possible triangulations.

3 Voronoi and Delaunay

3.1 Voronoi diagram

The Voronoi diagram also refered to as *Voronoi tesselation* is a special kind of decomposition of metric space, determined by distances between discrete sets of points. The Voronoi diagram of a set of points is the division of a plane or space into regions for each point. The regions contain the part of the plane or space which is closer to that point than any other.

Given a set $P = \{p_1, p_2, \cdots, p_n\}$ of n points in $\mathbb{R}^d$, the Voronoi Diagram $Vor(P)$ is the partition of $\mathbb{R}^d$ into n polyhedral regions, one for each p_i. Each region, known as a Voronoi cell denoted $vo(p)$, encloses and corresponds to one of the n points. The boundaries between the cells are the perpendicular bisectors of the lines joining the points, i.e. for each point p of the set and its corresponding cell, the cell only contains the points in $\mathbb{R}^d$ which are closer to p than all other

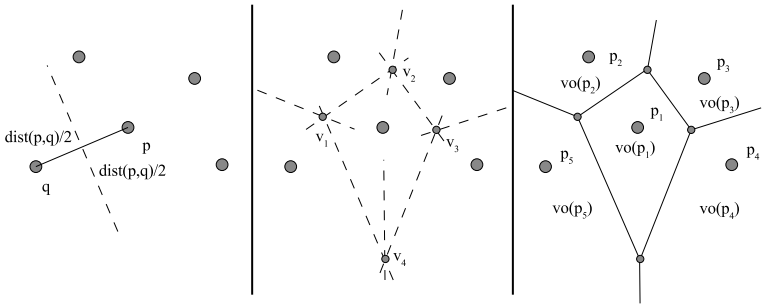

Figure 2: A two dimensional Voronoi diagram of 8 vertices

points. More precisely with $vo(p)$ being the Voronoi cell for point p and S the set of points then,

$$vo(p) = \{x \in \mathbb{R}^d | dist(x,p) \leq dist(x,q) \forall q \in S\{p\}\}$$

where $dist$ is the Euclidian distance funtion

$$dist : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$$

which gives the standard distance between two vectors. The vertices that are created in the intersections of these bisectors or boundaries are known as the *Voronoi vertices* (fig.3). We will later see that these play an essential part in the Delaunay triangulation. The former bisectors, the boundaries between two Voronoi cells are referred to as the *Voronoi edges*. Since every $vo(p)$ corresponds to a vertex p we have n Voronoi cells for a vertex set of n points, the Euler relation implies that there are $O(n)$ many Voronoi vertices and edges as well. Note: The edges of the voronoi diagram do not have to be bounded, unbounded edges are called *Voronoi rays*.

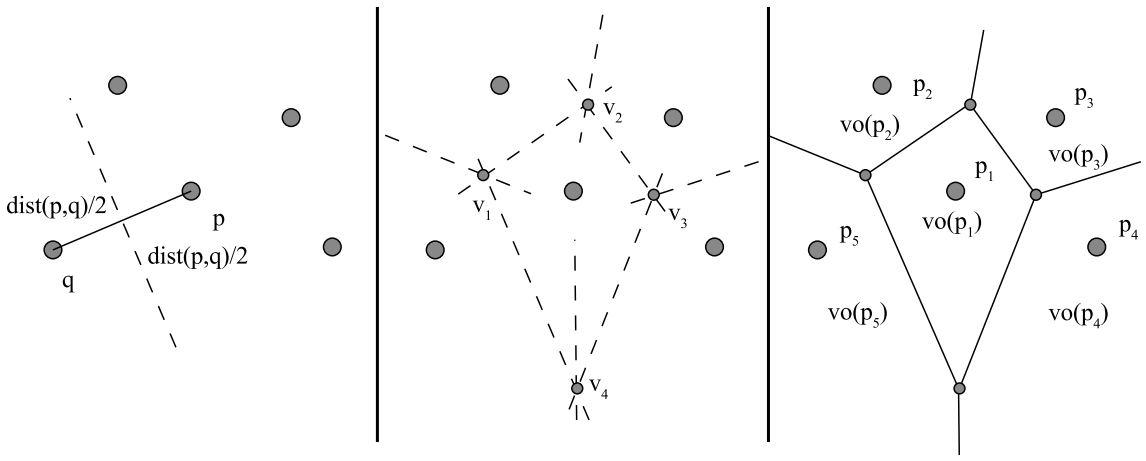

Figure 3: Step-by-step generation of a Voronoi Diagram from 5 vertices in 2D; left: the dashed *bisector* between p and q is formed; middle: Voronoi vertices v_i are formed at the bisector intersections; right: Bisector remnants are the Voronoi edges between the Voronoi cells $vo(p_i)$;

3.2 Delaunay triangulation

The *Delaunay triangulation* (fig.4) $Del(P)$ of a set of vertices $P = \{p_1, p_2, \cdots, p_n\}$, introduced by Delaunay in 1934, is very useful when working with meshes. One of its main advantages is that it

maximizes the minimum angle among all triangulations of a given vertex set. A triangulation is *Delaunay* when the Delaunay criterion is satisfied for all edges and triangles of the triangulation. By satisfying this criterion the triangulation gets the maximum minimum angle. This property gives us high mesh quality, which leads to stable calculations on the mesh and naturally a nicely triangulated mesh for shading and texturizing. Later we will take a look at the connection between the Delaunay triangulation and the voronoi Diagram mentioned before. This connection allows us to extract the triangulation from the diagram and vice versa. We will however in the Methods and Algorithms chapter see that there are methods to obtain them directly without the help of the other.

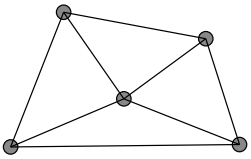

Figure 4: The Delaunay triangulation of a set of five vertices.

3.2.1 Delaunay criterion for edges and triangles

For a triangulation T of P to be Delaunay the Delaunay criterion or *empty circumcircle property* (fig.5) has to hold for all triangles and edges of the triangulation. A *circumcircle* is the circle that passes through the endpoints (vertices) u and v of the edge uv and endpoints u, v, w of a triangle uvw. When the *circumcircle* is empty, meaning that the circle passing through the endpoints of a triangle $t \in T$ or an edge $e \in T$ encloses no other vertex of the set V, t and e are said to be *Delaunay* and *locally Delaunay* respectively. It is obvious due to the nature of an edge that it can have infinitely many *circumcircles* where the *empty circumcircle property* only has to hold for one of them, where as for a triangle the circumcircle is uniquely defined by its three vertices. With the property of an edge or triangle being *Delaunay* Jonathan Richard Sewchuk

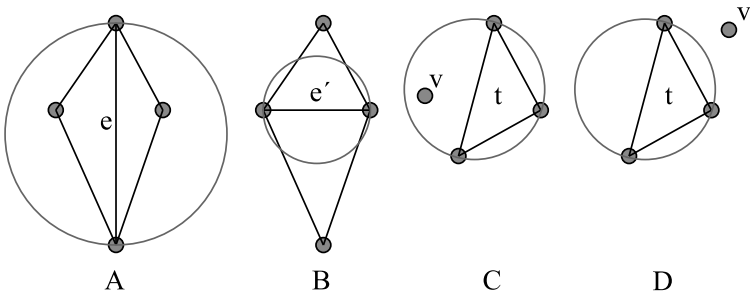

Figure 5: A: Edge e is not locally Delaunay, no empty circumcircle; B: e' is locally Delaunay; C: triangle t not Delaunay v lies within circumcircle; D: t is Delaunay.

proves [1] that: for a triangulation T, if all triangles $t \in T$ are Delaunay then all edges $e \in T$ are Delaunay; and vice versa. The idea of this proof is relatively easy. For the first direction, if we have a triangulation where all triangles are Delaunay, then all edges are part of a triangle which is Delaunay. A circumcircle can of course have other vertices *on* its boundary and still be empty. Now for each of the three edges of each triangle we therefore consider the same circumcircle as for the triangle itself, which is empty. In the other direction we assume that all edges of the Triangulation, T, are Delaunay, but for the sake of contradiction there is one triangle $t \in T$ which is not Delaunay. If t is not Delaunay then it has to have another vertex, v, within its radius but outside t itself. Now consider the edge e closest to v separating v from the interior of the triangle

t, there does not exist any circumcircle for e that contains neither v nor w, where w is the third corner of t opposite to v over the edge e.
The Delaunay triangulation $Del(V)$ of a vertex set V is unique, provided that at most three vertices $p \in V$ are co-circular [2]. When four vertices happen to be on the same circle, i.e. are co-circular, we are presented with two ways of forming triangles of the vertices, both are valid, but the triangulation will no longer be unique.

3.2.2 Egde flippings

Edge flipping is easily understood when considering part A and B of figure 5 above. Preforming an *edge flip* on the edge e shared by the two triangles t_1 and t_2, means to remove e and insert the other diagonal of the quadrilateral, yielding the new edge $e\prime$. An important lemma for *edge flippings* states that: if e is an edge of a triangulation T then, either e is locally Delaunay or e is can be flipped, yielding an edge which is locally Delaunay. The *edge flippings* are the foundation of many algorithms for the Delaunay triangulation, especially for the pure flipping algorithms mentioned later.

4 The Delaunay-Voronoi Duality

Before moving on to the algorithms and methods commonly used to obtain Delaunay triangulations we will more elaborately explain the connection between Delaunay triangulations and Voronoi diagrams. As mentioned earlier the Voronoi vertices (the intersection points of the Voronoi edges) play a central role in the Delaunay triangulation. As shown in figure 6 expanding a circle centered at a Voronoi vertex v, you get the *empty circumcircle* for pqr, which is the foundation of Delaunay triangulation. More specifically, the Voronoi Diagram $Vor(V)$ in $\mathbb{R}^2$ and the Delaunay Triangulation $Del(V)$, of a vertex set V, are *dual* to each other in graph theoretical sense.

By definition of duality of graphs, each of the graph vertices of a graph G corresponds to a face of the dual graph G^* each of whose vertices corresponds to a face of G. Also, two vertices in the dual graph G^* are connected by an edge if the corresponding faces in G are share a common edge. In our more specific case, every vertex $p \in V$ corresponds to a Voronoi cell $vo(p)$; every triangle $t \in Del(V)$ corresponds to a vertex $v \in Vor(V)$; every edge $e(p, q) \in Del(V)$ between two points in V corresponds to the boundary edge between the Voronoi cells $vo(p)$ and $vo(q)$.

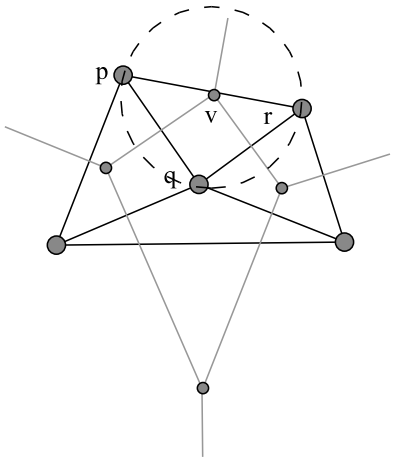

Figure 6: Delauany - Voronoi Duality. Triangulation shown on top of the Voronoi Diagram (faded), the triangle pqr corresponds to the vertex v which is the center of its circumcircle; the edges in the Voronoi diagram are exactly the half-way bisectors of the edges in the Delaunay triangulation.

5 Methods and Algorithms

5.1 Algorithms for obtaining Delaunay triangulations

There are many different algorithms that can be used to obtain the Delaunay triangulation, here I will present three of the algorithms most commonly used.
Due to the duality between Voronoi diagrams and Delaunay triangulations, given say the Voronoi diagram we can easily obtain the Delaunay triangulation. A wellknown approach to obtaining the the Voronoi diagram $Vor(V)$ of a set of vertices V is to use the *Sweep-line algorithm*, this is the most commonly used method for obtaining the Voronoi diagram. There are also ways of obtaining the Delaunay triangulation $Del(V)$ directly, the most common algorithm to use is probably the algorithm by Bowyer/Watson. The Boywer/Watson Algorithm falls into the category of *incremental insertion algorithms* for Delaunay triangulation there are also other classes such as pure *flipping algorithms* and the more exotic *projection algorithms* like the later explained $\mathbb{R}^{d+1}$ *projection algorithm*.

The *flipping* or *optimization algorithms* consist of two major components, first; given a vertex set V an arbitrary triangulation has to be generated, second; the triangulation has to be optimized into a Delaunay triangulation through flipping of non-Delaunay edges. Naturally the first step can be omitted when already presented with an arbitrary triangulation of the mesh. The edge flip algorithm is based on another lemma showed by Shewchuk [1]; if all edges of a triangulation T are locally Delaunay, then all edges of T are (globally) Delaunay. The flip algorithm of an arbitrary triangulation of n vertices terminates after $O(n^2)$ edge flips, yielding a new triangulation which is Delaunay. In the chapter *Edge flippings* we saw that, presented with an edge which is not locally Delaunay we can flip it obtaining an edge which is locally Delaunay. Proceeding this way for all non-locally Delaunay edges of a mesh we get a mesh which is *globally/everywhere* Delaunay.

Incremental insertion or just *insertion algorithms* incrementally insert the vertices we want to triangulate, one by one. After a point has been inserted into the triangulation the Delauany criterion is re-enforced. A so called *super triangle* (fig.7) is often formed to get a start triangulation into which the points can be inserted.

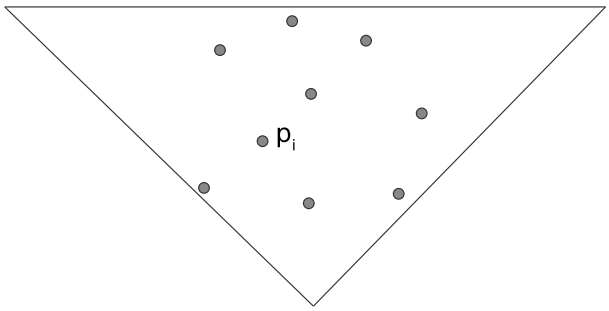

Figure 7: A super triangle, enclosing all the points p_i.

5.1.1 Bowyer/Watson

The algorithm of Bowyer/Watson falls into the category of incremental insertion algorithms. Most often the algorithm is know as just the Watson algorithm. If we are not given an initial triangulation it starts by forming the *super triangle* enclosing the vertex set V that we want to triangulate. The algorithm then proceeds by incrementally inserting the vertices $p \in V$ in the triangulation (fig.8). After every insertion step a search is made, to find the triangles whose circumcircles enclose p. (fig.8) These triangles are deleted, forming a polygon containing p called the *insertion polygon*. Edges between the vertices of the insertion polygon and p are inserted forming the new triangulation, which is Delaunay (fig.9). We have now, after inserting all the vertices $p \in V$ this way, obtained the Delaunay triangulation $Del(V)$. In the last step all we need to do is remove the

super triangle and its edges going into the triangulation.

In its simplest form the Watson algorithm can produce a faulty, degenerate mesh. In the step of finding the triangles whose circumcircles contain the inserted vertex, the algorithm can fail to delete one or more triangles effected by an insertion. This is due to a round-off error and it leaves one or more triangles in the insertion polygon which is supposed to be empty, when retriangulating this gives a faulty mesh. The Watson algorithm can be implemented to work robustly by using depth-first search from the super triangle in order to find the insertion polygon.

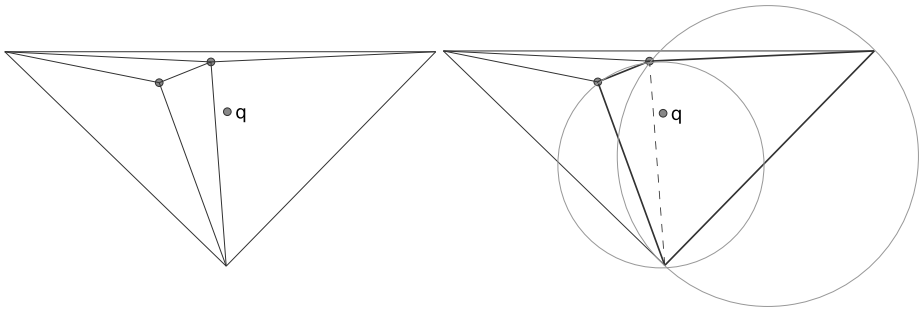

Figure 8: The Bowyer/Watson algorithm: After inserting a point q into a triangulation, we find the effected triangles and mark the insertion polygon (thick).

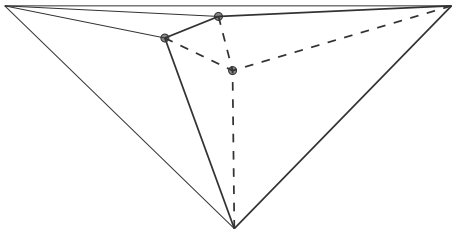

Figure 9: The Bowyer/Watson algorithm: After removing the interior (dashed in figure above) of the insertion polygon, we retriangulate by simply inserting the missing edges (thick dashed).

5.1.2 Lawson

Unlike the Watson algorithm the Lawson algorithm is not based finding circumcircles and deleting triangles to obtain an insertion polygon. Lawson's incremental insertion algorithm utilizes edge flippings to achive the same result and thus avoids a possibly faulty, degenerate mesh which can occur using Watsons method.

In the same way as before, if we are not given a start triangulation we use a *super triangle* enclosing all vertices in V. In every insertion step a vertex $p \in V$ is inserted. A simple retriangulation is made where the edges are inserted between p and the corner vertices of the triangle containing p (fig.11). For all new triangles formed the circumcircles are checked, if they contain any neighbouring vertex the edge between them is flipped. This process is executed recursively until there are no more faulty triangles and no more flips are required (fig.12). The algorithm then moves on, inserting another vertex from V until they have all been triangulated into a mesh, then, like before the super triangle and its edge are removed.

5.1.3 $\mathbb{R}^{d+1}$ projection algorithm

This algorithm transfers the problem of calculating the Delaunay triangulation onto the problem of calculating a *convex hull*. The name of the algorithm implies a projection, a projection onto

Bowyer-Watson Algorithm

- form *super triangle*, enclosing all points $p \in V$
- as long as not all vertices of V have been treated, do:
  1. insert vertex $p \in V$ into triangulation
  2. find circumcircles containing p with corresponding triangles
  3. remove triangles to get *insertion polygon*
  4. retriangulate *insertion polygon* by simply adding edges to p
- remove *super triangle*

Figure 10: Bowyer/Watson Algorithm in short

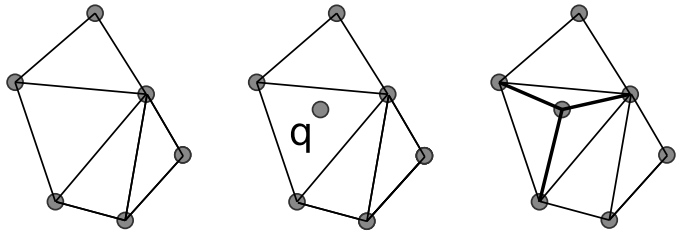

Figure 11: The Lawson algorithm: Inserting point q into a given Delaunay triangulation (no super triangle needed). First we simply retriangulate q into the triangle.

the $d+1$ dimension. What the algorithm basically does, what will be explained more theoretically below, is to project all points onto a one dimension higher paraboloid centered in the origin (fig.14). In this higher dimension the *convex hull* is calculated and the lower part of it projected back onto the first dimension, this gives the Delaunay triangulation (fig.15).
First we need to know what a convex hull is.

On the web, Mathworld at www.wolfram.com gives a definition of *convex hulls* as; *The convex hull of a set of points S in n dimensions is the intersection of all convex sets containing S.*
The 2D Delaunay triangulation $Del(V)$ can be interpreted in 3D. Due to this lifting from a lower dimension to a higher the algorithm is also known as the *lifting algorithm*. We have a Vertex set that we want to triangulate, V. First we introduce the syntax later used, the one dimension higher paraboloid, in this example 3D, has the form $f(x, y) = (x, y, x^2 + y^2)$ and is denoted Ω. The

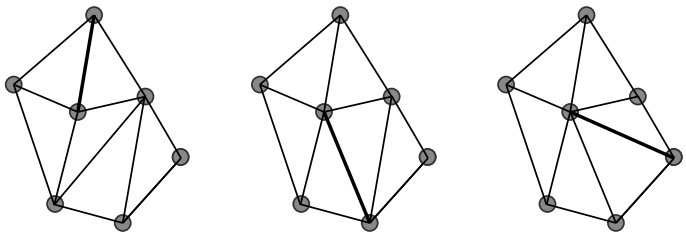

Figure 12: The Lawson algorithm: Flip non locally Delaunay edges of new triangles formed until there are no bad edges left.

Lawson Algorithm

- form *super triangle*, enclosing all points $p \in V$
- as long as not all vertices of V have been treated, do:
  1. insert vertex $p \in V$ into triangulation
  2. triangulate new p (draw edges to p from enclosing triangle, creating triangles)
  3. for all new triangles t created recursively:
     - check circumcircle of t, if containing neighbouring vertex, *flip*
- remove *super triangle*

Figure 13: Lawson Algorithm in short

projections of the 2D points $p \in V$ onto Ω are denoted $\hat{p}$, where $\hat{p} = (p_x, p_y, p_x^2 + p_y^2)$. The tangent planes in the projected points $\hat{p}$ are denoted $\hat{p}^*$. Tangent planes, which are the higher dimension equivalents to 2D tangent lines, can be defined as:

Let (x_0, y_0) be any point on a surface function $z = f(x, y)$. Then the surface has a nonvertial tangentplane at (x_0, y_0) with equation; $z = f(x_0, y_0) + f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0)$ [Mathworld].
Our tangent planes $\hat{p}^*$ are thereby $z = 2p_x x + 2p_y y - p_x^2 - p_y^2$. For any three vertices $p, q, r \in V$

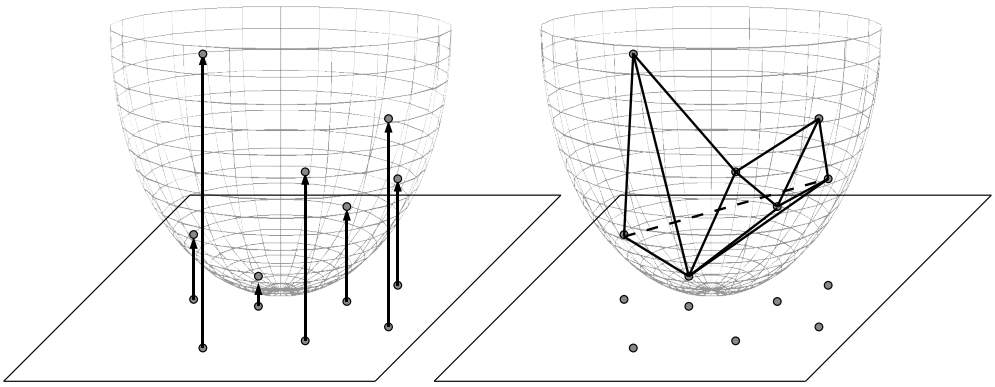

Figure 14: The projection algorithm: Project the points in $\mathbb{R}^2$ to the $\mathbb{R}^3$ paraboloid. Calculate the convex hull, here only shown the lower part. (Dashed lines lie behind the other)

where the corresponding Voronoi cells $vo(p), vo(q), vo(r) \in Vor(V)$ share a Voronoi vertex v, we consider the tangent planes $\hat{p}^*, \hat{q}^*, \hat{r}^*$ at the points $\hat{p}, \hat{q}, \hat{r}$ on Γ. $\hat{p}^*, \hat{q}^*, \hat{r}^*$ intersect at a point v^* in space, located above v. v^* is, due to the nature of the tangent planes, below its corresponding projection $\hat{v}$ on Ω.

Now imagine that we are standing at this point v^* and facing/looking at the paraboloid Ω. The outer edge, or boundary, of the points we see on Ω are exactly those, whose tangent planes pass through our location v^*. We call this boundary on Ω Γ. The points on Γ we denote $\hat{g}$ and their tangent planes are $\hat{g}^*$. It follows that the projection of Γ onto 2D gives us the points g at equal distance from v. That is a the points g form a circle centered at v. Since v is the shared vertex between three Voronoi cells $vo(p)$, $vo(q)$ and $vo(r)$, v is a Voronoi vertex and the circle around v is the *circumcircle* through the Delaunay triangle pqr.

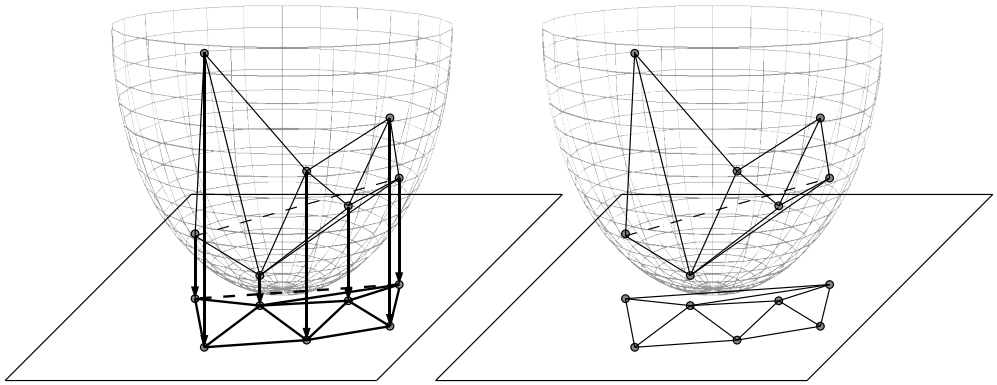

Figure 15: The projection algorithm: Project the convex hull edges back onto $\mathbb{R}^2$ and get the Delaunay triangulation. (Dashed lines lie behind the other)

Now how does this correspond to the convex hull? Because the point p, q and r define a unique circle, the circumcircle, the boundary Γ is in fact the intersection of the paraboloid with a plane through $\hat{p}$, $\hat{q}$ and $\hat{r}$. We know that p, q and r form a Delaunay triangle since their Voronoi cells share a Voronoi vertex v, this means that the circumcirle is free of all other points $w \in V$. In projective terms this means that these other points $w \in V$ outside pqr's circle are projected to points that lie above the plane through Γ, so on Ω we can cut away what is below this plane without removing any of the projected vertices $\hat{w}$. Each of these cuts removes a piece of the parabola Ω and leaves a facet, the triangle through $\hat{p}$, $\hat{q}$, $\hat{r}$. For all points p, q, r, $pqr \in Del(V)$ these triangle facets form the convex hull of the projected points $\hat{p}$, $p \in V$. What we get is that the convex hull facets and their boundaries are the Delaunay triangles projected onto Ω, and vice versa.

There are many avaliable algorithms for implementing and obtaining the convex hull of a set, Matlab and Mathematica comes with convex hull methods and there are many other implementations avaliable such as *qhull*. We are now able to reduce our calculation of the Delaunay triangulation to two projections and a convex hull calculation. From the *qhull* homepage the complexity for the convex hull calculation using *qhull* is $O(nlogv)$, where n and v are the input and output vertices, in our case when all the points on the paraboloid are part of the triangulation this would be $O(nlogn)$, $n = v$. The projections are done in linear time which gives $O(nlogn)$ as the total complexity.

Projection Algorithm (lifting algorithm)

- project the d-dimensional vertices $p = (p_{x_1}, p_{x_2}, \cdots, p_{x_d}) \in V$ to $d+1$-dimensional points $\hat{p} = (p_{x_1}, p_{x_2}, \cdots, p_{x_d}, f(p_{x_1}, p_{x_2}, \cdots, p_{x_d}))$ on the paraboloid $f(p_{x_1}, p_{x_2}, \cdots, p_{x_d}) = {p_{x_1}}^2 + {p_{x_2}}^2 + \ldots + {p_{x_d}}^2$ centered at the origin.
- calculate the *convex hull* of the points to the paraboloid
- project the lowest part of the*convex hull* back on the d-th dimension
- d-dimension edges between the vertices now form the Delaunay triagulation of the set V

Figure 16: Projection Algorithm in short

5.2 Algorithms for obtaining Voronoi diagrams

By knowledge of the duality between the Delaunay triangulation and the Voronoi diagram, the diagram should be easily obtained by connecting the circumcircle centers from the triangulation. Usually this is not what is done, the Voronoi diagram can be used to obtain the Delaunay triangulation but we selldomly use this relation the other way around. If one wants to calculate the Voronoi diagram, there is a projection algorithm very similar to the one for Delaunay; the Voronoi diagram $Vor(V)$ of a vertex set P is the back-projection of the *upper envelopes* of the tanget planes $\hat{p}^*$ for all $p \in V$. More commonly used however is the *sweep-line algorithm.*

5.2.1 The Fortune sweep-line algorithm

The sweep-line algorithm by Fortune uses a sweep-line to traverse the points in a set. There are two types of constructive events where the components of the Voronoi diagram are born. A *point event* gives birth to a new Voronoi edge and a *circle event* gives birth to a new Voronoi Edge as well as a Voronoi vertex. Given a set of V vertices in the plane we consider a line l that sweeps over the plane top-down, this is called the sweep-line (fig.17). The half-plane above l we refer to as l^+. For a vertex $p_i \in V$ we let β_i denote the vertical parabola with p_i as its focus. β_i consists of the points in l^+ that are at equal distance from p_i and the sweep-line l. The beach-line (fig.18) is the lower

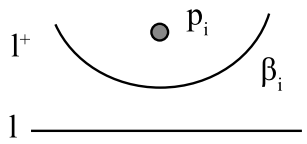

Figure 17: The vertical parabola β_i for vertex p_i are those points at equal distance to p_i and sweep-line l.

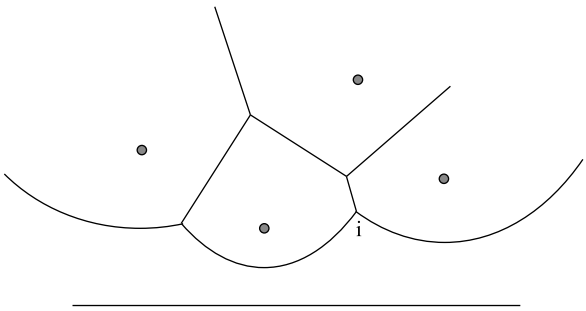

Figure 18: The sweep-line algorithm: The beach-line is the lower envelope of the parabolas β_i for the points p_i in l^+.

envelope of the parabolas of the points in l^+, basically the arc-tips of the lower parabolas. As the sweep-line moves down the beach-line naturally follows along. By the nature of the beach-line, as it moves down it gradually flattens. The Voronoi edges are born with with the occurance of a *point event.* In the instance that the sweep-line l has swept down and reached a new point p_n, a point event occurs (fig.19). First a vertical line is drawn from p_n straight up to the beach-line. This is the birth of a new Voronoi edge. The vertical line will expand into a new parabola β_n for the new point p, opening up a hole on the beach-line. The intersections of β_n with the rest of the beach-line will start tracing a new Voronoi edge. Voronoi vertices are created with the occurance of *circle events.* A *circle event* is when two traced Voronoi edges meet a common intersection point. Alternatively if we consider consecutive arc-triples on the beach-line, a *circle event* will occur in the intersection

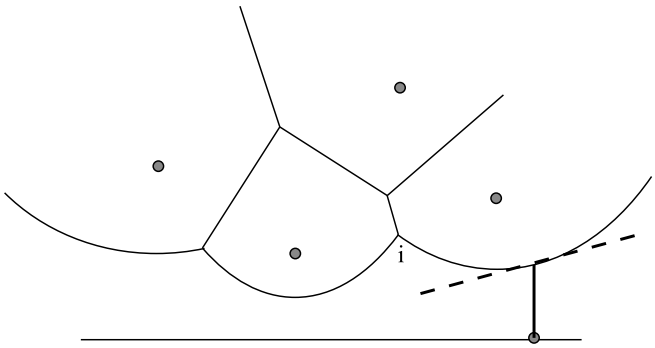

Figure 19: The sweep-line algorithm: A point event occurs when the sweep-line reaches a new point. The vertical (thick) line will expand into a parabola β for the new point. β will start tracing the new Voronoi edge, thick dashed line.

point when the two outer arcs overgrow the middle one. A triplet of consecutive arcs is the part of the beach-line where the parabolas β_i, β_j, β_k for the points p_i, p_j, p_k lie consecutively after each other. This intersection point where β_i meets β_k is the Voronoi vertex. A new Voronoi edge is also created since we now start tracing a new intersection line between β_i and β_k. The Voronoi vertex created in a circle event is exactly the mid-point of the circle through the three points, p_i, p_j, p_k whose parabolas formed the arc triplet, that is why it is called a circle event. This circle is actually the circumcircle through p_i, p_j, p_k.
Since these events are the constructive parts of this algorithm, we need to keep track of them. It is easy to know when a point event occurs, we just have to check when the sweep-line hits a new point. For circle events we keep track of the triplets of consecutive arcs on the beach-line, when a triplet becomes a tuplet we have circle event. When the sweep line has swept through the points and when the list of triplets is empty what remains is the Voronoi edges and vertices, the *Voronoi diagram* $Vor(V)$ of the set V.

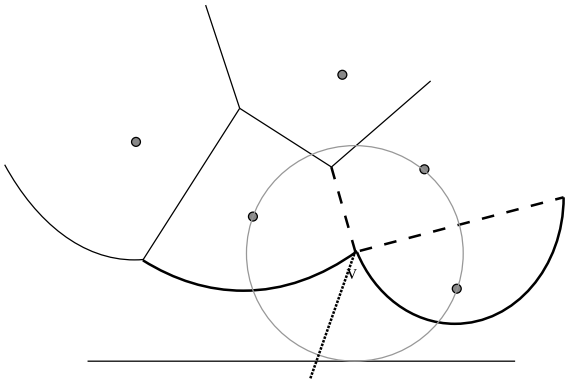

Figure 20: The sweep-line algorithm: A circle event has occured, a new Voronoi vertex v is created in the intersection of the two dashed Voronoi edges. A new Voronoi Edge, dotted, is born and will be traced between the two thick arcs.

5.3 Last words on the algorithms

Finally a few words on the general robustness and complexities of some of the algorithms.

The complexity of the flipping algorithm is $O(n^2)$ edge flips, after which it yields a Delaunay triangulation.

Sweep line algorithm

- use a line sweeping the plane from top-down, a *sweep line*, to traverse the points
- keep track of triplets of consecutive arcs on the beach line B in event queue
- when sweep line hits a new point p_i (*point event*)
  - add vertical line from p_i to B (as sweep line moves on, let the line grow into an arc β_i on B for point p_i
  - a new Voronoi edge is born in B^+
- when three consecutive arcs become two, i.e. the end points of the two overgrow the middle one (*circle event*)
  - the intersection point q is a Voronoi vertex
  - a new Voronoi edge is born
- when the sweep line has swept all the points and the event queue is empty, we are left with the Voronoi edges and vertices, the Voronoi diagram

Figure 21: Sweep line algorithm in short

The *Bowyer/Watson algorithm* is a mentioned, not robust against round-off errors in its simplest implementation, but by changing the search-method for finding the insertion polygon to a depth-first search, we end up with an insertion algorithm which is equally robust to the *Lawson algorithm*. In two dimensions these algorithms run in $O(n^2)$ edge flips time the average case is however $3n$ egde flips. A bound first obtained by Clarkson and Shor, binds the complexity for random insertion algorithms to $O(n \log(n))$, using a special point location scheme.

The complexity for the *projection or lifting algorithms* is as mentioned above linear in the projections and $O(n \log(n))$ for the convex hull calculation.

The *sweep line algorithm* for Voronoi diagrams runs in $O(n \log(n))$ time.

References

[1] J. Shewchuk.

The Delaunay Triangulation and Unstructured Mesh Generation.. Lecture Notes on Delaunay Mesh Generation 1:10-15, 1999

[2] L. De Floriani. *Delaunay Triangulation: definitions* Delaunay Triangulation: Part 1 3-4, 2003

[3] S.-W. Cheng. *Voronoi Diagram* Computational geometry handout 2004

[4] S.-W. Cheng. *Delaunay Triangulation* Computational geometry handout 2004

[5] K. Fukuda *Voronoi Diagram and Delaunay* Polyhedral Computation FAQ 2000

[6] *Voronoi tesselation* Unstructured Mesh Generation on Planes and Surfaces using Graded Triangulation 1997