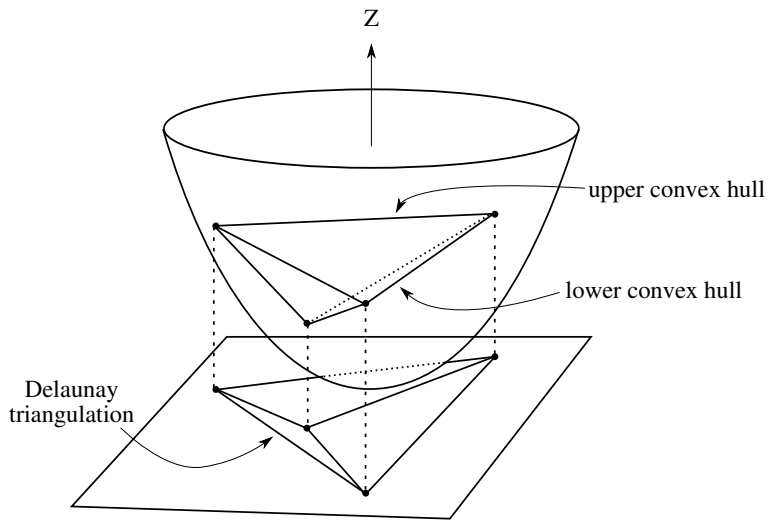
A Survey of Mesh Generation Techniques

Dirk Gerrits René Gabriëls Peter Kooijmans

June 1, 2006



2IL40 Advanced Algorithms 2005/2006
Department of Mathematics & Computer Science
Technische Universiteit Eindhoven

0531798 Dirk Gerrits <dirk@dirkgerrits.com>
0531594 René Gabriëls <r.gabriels@student.tue.nl>
0538533 Peter Kooijmans <p.j.a.m.kooijmans@student.tue.nl>

Contents

1 Introduction

Mesh generation is a topic studied by two kind of researchers: theoreticians (mainly mathematicians and computer scientists), and practitioners (mainly structural engineers). Theoreticians are mostly interested in the properties of the algorithms involved such as complexity and optimality, while the practitioners are also concerned with applicability of the algorithms to real world problems.

The theoretical study of mesh generation is rooted in the study of optimal triangulations, and hence is a part of the field of computational geometry. Results coming from this field are usually published in the International Journal of Computational Geometry and Applications or in Discrete and Computational Geometry. An overview of results in the field of computational geometry can be found in the book [3] which has several relevant chapters on the subject of mesh generation. More detailed results on mesh generation can be found in several survey papers, such as [1] and [5].

The practical study of mesh generation is rooted in doing computer simulations of physical systems, e.g. airflow around the wing of an airplane. In order to simulate a continuous physical domain, it has to be discretised in such a way that meaningful results can be obtained by applying mathematical techniques such as the finite element method (FEM). Mesh generation is a critical step in computer simulation, because the elements of the mesh and the size of the mesh determine the numerical properties of the outcome and the running time of the simulation. Results coming from this field are usually published in the proceedings of the International Meshing Roundtable or in the International Journal for Numerical Methods in Engineering. For a more in-depth treatment of the mesh generation in computer simulation, see [2] and [4].

1.1 Problem description

What the theoretical and practical fields have in common is the definition of a mesh: a partitioning of some geometric domain into certain finite elements, satisfying a number of constraints. The geometric domain is usually a point set or *polytope* (a generalisation of a polygon to any dimension). The finite elements are usually triangles or quadrilaterals in two, and tetrahedra or hexahedra in three dimensions. The constraints can be a lot of things, which we will see later.

There are two kind of meshes: *structured meshes* and *unstructured meshes*. Structured meshes are deformed grid-shaped meshes, and therefore have a fixed topology, while unstructured meshes can have a varying topology. Structured meshes are conceptually easier, but they can often not be fitted easily to complex domains. Sometimes, this can be remedied by decomposing the domain into simpler regions, and then for each of those regions generate a structured mesh.

We will focus on unstructured mesh generation, or more precisely, on the algorithms that generate those meshes. Traditionally, theoreticians studied only optimal triangulation problems, which are problems that require one to calculate an optimal *simplicial complex* given some fixed input. A simplicial complex is a set of *simplices* that intersect only by their common edges. A simplex in n dimensions is the most simple element that spans an n dimensional subspace, in two dimensions this is a triangle and in three dimensions a tetrahedron.

Mesh generation problems usually ask to insert new nodes where necessary as well, the

so called Steiner nodes. Mesh generation also does not restrict itself to simplices; often it is desirable to obtain a mesh consisting of quadrilaterals in two dimensions or hexahedra in three dimensions. Optimal results for mesh generation problems are much harder to obtain, hence the optimal solution is usually approximated by means of an approximation algorithm.

A restriction we will abide by is that we assume no other information about the domain is known. In practice however, some knowledge about the domain to be simulated is often used to guide the mesh generation algorithm in the shape and size of elements generated.

1.2 Outline

In this paper we will give an overview of the theoretical results, and only briefly mention some interesting practical "heuristics". Because there are so many orthogonal properties of the problems under study, describing the results in a suitable linear fashion is non-trivial. We will follow the same structure as the survey [1]. Because almost any algorithm is input dimension specific, we first split the problem space on that: there are separate sections for two dimensional and three dimensional meshing. For each of these sections, we split further on problem type: triangulation, mesh generation (which is Steiner triangulation) and heuristics. On the lowest level we split on input type and optimisation criteria.

To keep the size of this paper manageable, we will not list every algorithm and proof, but only give references to sources where they can be found. This will result in quite some references, so we will list references per relevant section or subsection only.

References

[1] M.W. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry*, number 4 in Lecture Notes Series on Computing, pages 47–123. World Scientific, second edition, 1995.

[2] P.J. Frey and P. George. *Mesh Generation: Application to finite elements*. Hermès Science Europe Ltd, 2000. 1-903398-00-2.

[3] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, first edition, 1997. 0-8493-8524-5.

[4] S.H. Lo. Finite element mesh generation and adaptive meshing. *Progress in Structural Engineering and Materials*, 4(4):381–399, 2002.

[5] S. Teng and C.W. Wong. Unstructured mesh generation: Theory, practice, and perspectives. *International Journal of Computational Geometry and Applications*, 10(3):227–266, 2000.

2 Two-dimensional meshing

The special case of two dimensional meshing is the most well studied domain in meshing. This section is divided into two parts: first we treat ordinary triangulation, and then Steiner triangulations which are the most representative for two dimensional mesh generation problems. The most likely inputs to a 2-dimensional meshing algorithm are:

- PSLG (planar straight line graph), which is the most general input possible. It consists of vertices and non-crossing edges between vertices, which may not be removed.
- Point-set, which consists of a set of points in the plane. It is a special case of the PSLG, namely without any edges.
- Polygon with holes, which also is a special case of a PSLG, although it doesn't allow any edges in the triangulation that are outside its hull. So triangulation algorithms for a PSLG cannot be used directly to triangulate a polygons with holes.
- Simple polygon, which is a special case of a polygon with holes, except that it doesn't have any holes.

There are other inputs possible, which constrain the input further in some way, e.g. convex and monotone polygons. In all cases we can describe the complexity of the input by the number of vertices, n, it contains. This is true because the input is always a planar graph, and therefore Euler's formula holds.

2.1 Triangulation

The problem of 2-dimensional triangulation asks for a simplicial complex that optimises a set of criteria, given one of the inputs described above. Note that a two dimensional simplex is simply a triangle, so we ask for a collection of non-overlapping triangles, with corners on the input vertices such that all vertices are covered.

There exist several optimisation criteria for the triangles generated: maximise the minimum angle of the corners, minimise the maximum angle of the corners, minimise the maximum circumcircle, minimise the maximum containment circle, and minimise the area ratio of inscribed circle and triangle. There are also criteria that do not look at the triangles, but at vertices or edges, such as: minimise the maximum vertex degree, and minimise the total edge length.

We start in section 2.1.1 with treating unoptimised triangulation, i.e. triangulation problems without optimisation criteria. In the sections 2.1.2 and 2.1.3, we introduce two important concepts, namely the Delaunay triangulation and the constrained Delaunay triangulation, that turn out to optimise several important criteria. In 2.1.4 we describe a number of algorithms to construct the (constrained) Delaunay triangulation. In 2.2.3, 2.1.6, and 2.1.7 we describe algorithms that can optimise interesting criteria not optimised by the (constrained) Delaunay triangulation. And finally, in section 2.1.8, we look at anisotropic triangulation.

2.1.1 Unoptimised triangulation

The simplest triangulation problem asks for any triangulation of its input, no criteria have to be optimised. The first question that arises is whether there always exists a triangulation for all types of input described above. This is indeed the case. For the most constrained case, namely simple polygons, this can be proved by showing that a diagonal always exists. It also turns out that the number of triangles and edges in any triangulation is always the same: there are $2n - 2 - k$ triangles and $3n - 3 - k$ edges, where k is the number of edges on the convex hull.

Calculating the triangulation for a point set is reducible to sorting numbers. So a lower bound on the running time of any point set triangulation algorithm is $\Omega(n \log n)$. It turns out

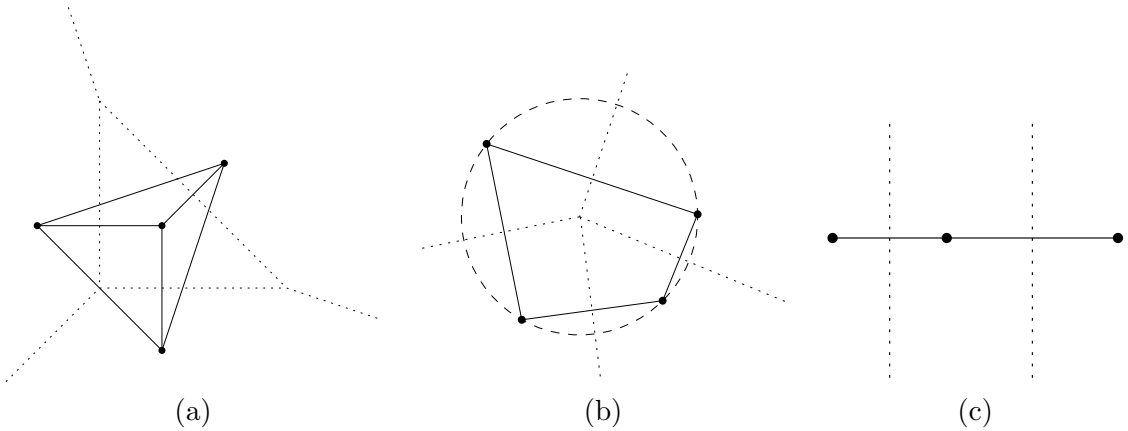


Figure 1: Voronoi diagrams (dotted) and their dual: (a) unambiguous case, (b) ambiguous case where four Voronoi sites are cocircular, (c) ambiguous case where all points are colinear

that this lower bound is tight: triangulations of point sets can be calculated in $O(n \log n)$. The same holds for the more general case of a PSLG. For the more constrained case of a simple polygon, an $O(n)$ algorithm exists. The same algorithm can also triangulate polygons with holes, but the running time increases to $O(n \log k)$, where k is the number of holes.

Theorem 1. *A point set or PSLG can always be triangulated in $O(n \log n)$ time. Polygons with holes can always be triangulated in $O(n \log k)$. Simple polygons can always be triangulated in $O(n)$.*

2.1.2 Delaunay triangulation (DT)

The *Delaunay triangulation* is an important concept in point set triangulations, because it optimises several criteria simultaneously. The Delaunay triangulation of a point set is the dual of the Voronoi diagram of the same point set. We will discuss what exactly that means, and find out about some of its interesting properties. Then we will discuss some construction algorithms. The discussion in sections 2.1.2 and 2.1.3 is based on chapters 1 and 2 of [2], and section 2.2 of [1]. The interested reader is directed to these references for proofs of the discussed properties.

Defintion 1. *The* Voronoi diagram *of a point set decomposes the plane into one* Voronoi region *for each point. A point p's Voronoi region contains all points that are at least as close to p as to any other point of the set.*

We obtain the dual of a Voronoi diagram by connecting two points if and only if their Voronoi regions share an edge. Figure 1(a) illustrates this, for the case that the dual consists of only triangles. When four or more points in the point set lie on a circle however, four or more Voronoi regions could meet in a single point and more general polygons result, see figure 1(b). Also, with all points on a line, we get polygons degenerated to line segments, see figure 1(c). When the point set is such that no four points lie on a circle and no three points lie on a line, we say the point set is in *general position.*

Defintion 2. *The* Delaunay triangulation *of a point set is the dual of the Voronoi diagram of the point set, if the point set is in general position.*

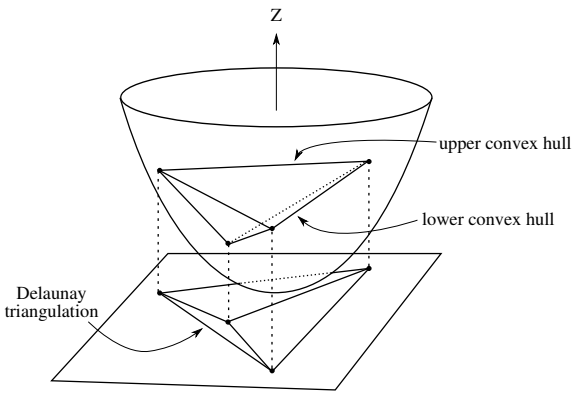


Figure 2: The relation between the 2-dimensional Delaunay triangulation and the lower convex hull of a 3-dimensional paraboloid. (Figure 3 of [1].)

Because the Voronoi diagram is uniquely defined, the Delaunay triangulation is also uniquely defined if the point set is in general position. We will assume for the rest of the discussion that the point set is in general position. This is no real restriction, because if an actual input set of points is not in general position, we can perturb the points slightly so that they are. The polygons in the dual will then be arbitrarily triangulated and points on a line will form very skinny triangles. Another solution for the four or more cocircular points problem would be to postprocess the triangulation by arbitrarily triangulating the remaining more general polygons.

The Delaunay triangulation can also be defined directly. An edge $\overline{ab}$ is part of the Delaunay triangulation, if and only if there exists a circumcircle through a and b, that does not contain any other point inside or on it. Edges satisfying this property are said to be *locally Delaunay*. A triangle abc is part of the Delaunay triangulation if and only if its circumcircle contains no other points inside or on it. This criterion is called the *Delaunay criterion*.

Another definition, which is isn't immediately obvious: the Delaunay triangulation is equivalent to the lower part of the convex hull of the points "lifted" to the paraboloid $(x, y, x^2 + y^2)$. See figure 2. This definition is interesting though, because it applies to any dimension, and allows an easy construction algorithm, as we will see later.

A Delaunay triangulation optimises a number of interesting properties:

- It maximises minimum angle, i.e. among all triangulations of the point set the Delaunay triangulation has the least small minimum angle. Note that it does not do the reverse, i.e. it does not minimise the maximum angle.

- It minimises the maximum circumcircle and containment circle, i.e. among all triangulations of the point set the Delaunay triangulation has the least big maximum circumcircle and containment circle.

The Delaunay criterion does not directly give rise to an algorithm that computes a triangulation satisfying the Delaunay criterion. A very simple approach is to generate an initial triangulation and improve it using the *edge flip algorithm*. The edge flip algorithm works by flipping the diagonal of a quadrilateral composed of two adjacent triangles, if the quadrilateral is convex but not locally Delaunay. The total running time of this algorithm is $O(n^2)$.

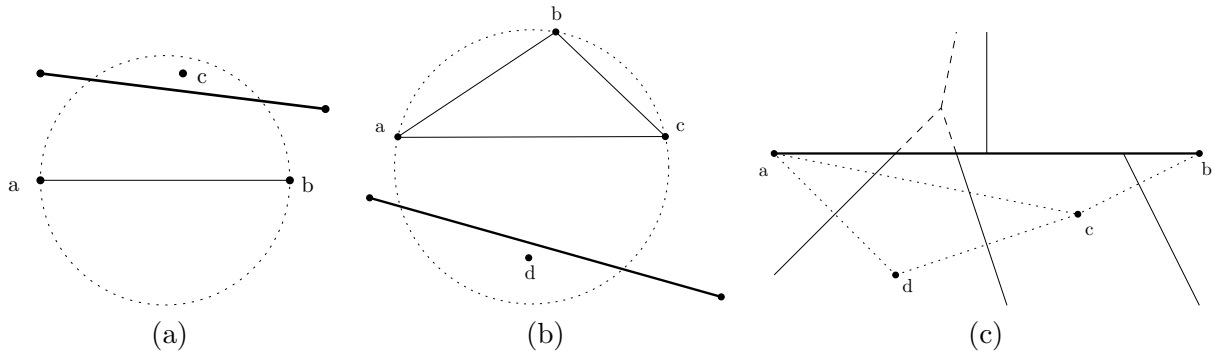


Figure 3: Constrained Delaunay triangulation (a) Edge $\overline{ab}$ is part of the CDT, because c is not visible from $\overline{ab}$ (b) Triangle abc is part of the CDT because d is not visible from its interior. (c) Bounded Voronoi diagram (solid lines) and associated CDT (dotted lines).

Because the Delaunay triangulation of a point set can be defined in terms of the Voronoi diagram of that point set or in terms of the lifting transformation, we can use algorithms to construct one of those and then easily obtain the Delaunay triangulation. This methods gives optimal running time. In section 2.1.4 we will describe direct algorithms that are optimal.

2.1.3 Constrained Delaunay triangulation (CDT)

Suppose we are given not only a point set S but also a set of edges L between some of those points, i.e. the input becomes a PSLG. We can then find a triangulation that contains all those edges, and is in some sense "closest" to the Delaunay triangulation of the points. Such a triangulation is called a *constrained Delaunay triangulation*. Following the discussion in chapter 2 of [2], we can fairly simply extent the notions introduced for Delaunay triangulation for this new problem.

Before we can define the constrained Delaunay triangulation of a PSLG, we have to define visibility between to points. Two points a and b are called *visible* from each other if the line segment $\overline{ab}$ does not cross any other edge, and no points other than a and b lie on it. Now, the constrained Delaunay triangulation can be defined directly by changing the direct definition of the Delaunay triangulation.

Defintion 3. *The* constrained Delaunay triangulation *contains an edge ab if and only if it is part of the input, or a is* visible *to b and there is a circle passing through a and b such that all other points inside or on the circle are not visible from any point on* $\overline{ab}$.

The Delaunay criterion can also be extended to the constrained case: a triangle is part of the constrained Delaunay triangulation if and only if all other points contained in its circumcircle are invisible from its interior. See figures 3(a) and 3(b) for two examples.

We defined the Delaunay triangulation as the dual of the Voronoi diagram. Similarly, there's a notion of a *bounded Voronoi diagram*[1] to which the constrained Delaunay triangulation is (almost) dual. The bounded Voronoi diagram is simply a Voronoi diagram constructed with a different distance measure between the points, one that takes obstruction of visibility by the input segments into account. A point p's bounded Voronoi region contains all the points for which p is the closest *visible* point. The edges of the constrained Delaunay triangulation are such that their end points's bounded Voronoi regions either share an edge, or they would

have shared an edge if the region wasn't obstructed by an edge in L. This last case is shown in figure 3 (c), where the CDT contains edge $\overline{ac}$ even though a and c's Voronoi regions don't meet, but they *would* meet (dashed lines) if it wasn't for constraint edge $\overline{ac}$.

The edge flipping algorithm can be generalised to PSLGs, simply by not allowing swapping of edges that were present in the input. The time complexity is still $O(n^2)$. Note that no border problems arise when applying the algorithm to an already triangulated polygon. But also here, we can do better.

2.1.4 DT & CDT construction algorithms

In the previous sections we saw simple edge flipping algorithms to compute the Delaunay and constrained Delaunay triangulation. But we can do better. There are several algorithms that have a running time of $O(n \log n)$, which is optimal. We will describe a sweepline algorithm.

Given a point set S and a line segment set L, one can construct a CDT in $O(n \log n)$ time using a *sweep line* algorithm. A vertical line is swept, from left to right, visiting the points in the set S and the begin and end points of the line segments in L in order of increasing x-coordinate. This ordering is computed in advance and clearly takes $O(n \log n)$ time. Also the line segments intersecting the sweep line are stored in a data structure, the *status structure*. This data structure should allow for the search, insert and delete operations to be performed in $O(\log n)$ time.

At each step of the algorithm the line segments intersecting the sweep line partition it into intervals. In such an interval there exists a convex chain of edges that form the CDT up till now. From each of these convex chains the rightmost vertex is stored. When the algorithm reaches a point p, the corresponding interval is computed, and then the point is connected to the rightmost vertex that was stored for this interval.

In [2] it is shown that there is a search in the status structure of the sweep line algorithm for each point in S and that there is an insertion and deletion pair for each line segment in L, which in all takes time $O(n \log n)$. Because fewer than $3n$ edges are added, and each edge is added in constant time, the total running time of the algorithm becomes $O(n \log n)$.

Theorem 2. *The Delaunay triangulation of a point set, and the constrained Delaunay triangulation of a PSLG can be calculated in* $\Theta(n \log n)$ *time.*

Another optimal algorithm to compute the CDT is a *divide & conquer* approach. First, the end points of the line segments are sorted by x-coordinates. Then the set of n end points is split by a vertical line into two subsets, each with at most $\lceil n/2 \rceil$ points. The CDT of both subsets is then computed and merged to create the final triangulation for the entire input.

The best known incremental construction algorithm is described in [2], and runs in $O(n \log n)$ expected time, which doesn't improve the running time of the deterministic algorithm.

2.1.5 Edge insertion

The edge flip algorithm seen in the previous two sections is an example of a local improvement algorithm. It finds a triangulation that maximises the minimum angle and always ends up in a global optimal. But other optimization criteria also exist and edge flipping will usually not find a global optimum for these criteria.

Another example of a local improvement algorithm, *edge insertion*, is a generalisation of the edge flip algorithm. It always finds an optimal solution for minimising the maximum angle, while the edge flip algorithm is not guaranteed to find a global optimum for this criteria.

The edge insertion algorithm adds an edge e to an existing triangulation and removes all edges of the triangulation that intersect with e. This results in two simple polygons, without diagonals, on either side of the just inserted edge e. These polygons are then triangulated again. The edge e is chosen to cut through the "worst" triangle, for a certain function f measuring the badness of triangle. If the newly created triangulation is worse than the original one, the original one is restored. As is proven in [1] this algorithm runs in $O(n^3)$ using $O(n^2)$ insertions. When the order of the insertions is chosen more carefully the running time can be reduced to $O(n^2 \log n)$.

2.1.6 Dynamic programming

Many optimal triangulation problems can be solved in polynomial time, by using a *dynamic programming* approach. Let f be a function that computes the quality of a triangulation of simple polygon.

Lemma 1. *The optimal triangulation for a quality measure f can be computed in $O(n^3)$ time if the function f is* decomposable*.*

Defintion 4. *Let f be a function. Given a simple polygon P and corresponding triangulation $\mathcal{T}$, let there be a diagonal (a, b) of P that splits the polygon into two simple polygons P_1 and P_2 with corresponding triangulations $\mathcal{T}_1$ and $\mathcal{T}_2$. The function f is said to be* decomposable *if:*

- *There is a function g, such that $f(\mathcal{T}) = g(f(\mathcal{T}_1), f(\mathcal{T}_2), a, b)$.*
- *g can be computed in $O(1)$ time.*
- *If $\mathcal{T}$ is just a single triangle, then $f(\mathcal{T})$ can be computed in $O(1)$ time.*

A few examples of decomposable functions are:

- The minimum (maximum) angle in a triangulation.
- The minimum (maximum) circumcircle of a triangle.
- The minimum (maximum) length of an edge in the triangulation.
- The minimum (maximum) area of a triangle.
- The sum of the edge lengths in the triangulation.

The time bound of $O(n^3)$ is proven in [1], and can be improved by using a *visibility graph* and only doing computations for diagonals.

Defintion 5. *The visibility graph of a polygon P has all vertices of P, and an edge between a and b, if a is visible from b in P.*

The visibility graph of a polygon P can be easily computed in $O(n^2)$ time, but more complicated algorithms exist that can do this in $O(n \log n + E)$.

Now, using the fact that a graph with E edges has at most $O(E^{3/2})$ triangles, we can compute the triangulation of any simple polygon, optimising any decomposable function in time $O(n^2 + E^{3/2})$, where E is the number of edges in the visibility graph of the simple polygon.

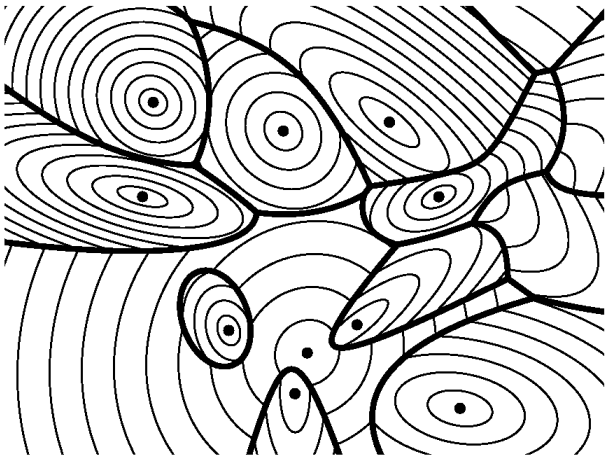


Figure 4: A 2-dimensional anisotropic Voronoi diagram. Thin arcs show isolines of each point's distance measure. (This is figure 3 from [3].)

2.1.7 Greedy method

Using a greedy algorithm to create a triangulation, one start with the empty triangulation, and at each iteration the shortest uncrossed edge is added. This leads to a triangulation where the sorted vector of edge lengths is lexicographically minimised [1]. The time complexity for this algorithm is $O(n^2)$ for points sets and this can be improved to $O(n)$ for convex polygons.

2.1.8 Anisotropic triangulation

It's not always the case that long, skinny triangles are unwanted. Sometimes we want our triangulation to be coarser in one direction than in another. That is, we want our mesh to have *anisotropic* properties. In this section we will look at *anisotropic Delaunay triangulation* as described in [3].

Defintion 6. *An* anisotropic Voronoi diagram *is a Voronoi diagram where each point has its own notion of distances. More formally, in d dimensions each point p has an associated d x d matrix M_p, which induces a distance function* $d_p(q_1, q_2) = \sqrt{(q_1 - q_2)^T M_p (q_1 - q_2)}$

The Voronoi region of a point $p \in S$ then contains all the points q for which $d_p(p, q)$ is less than $d_{p'}(p', q)$ for any other $p' \in S$. In general, this yields Voronoi regions bounded by arcs rather than straight edges. Figure 4 shows an example.

The dual of an ordinary Voronoi diagram of a point set in general position is the Delaunay triangulation. An anisotropic Voronoi diagram can have a much more complicated structure, and its dual isn't necessarily a triangulation. Theorem 7 of [3] describes a sufficient condition for the dual to be a triangulation. We'll paraphrase it below, but first we need some definitions.

The *wedge* between $v, w \in S$ is defined as the points q for which both the angle $\angle qvw$ as seen from v and the angle $\angle qwv$ as seen from w are less than 90°. An arc of an anisotropic Voronoi diagram is called *wedged* if for every pair of distinct points $v, w \in S$, all points of the arc lie in the wedge between v and w.

Lemma 2. *Let Ω be a polygonal subregion of the plane whose vertices are in S, and containing all other points of S in its interior. Let D be the anisotropic Voronoi diagram of S. We write*

$D|_\Omega$ as the part of D that lies within Ω. If all the Voronoi arcs and vertices in $D|_\Omega$ are wedged, then the dual of $D|_\Omega$ is a polygonalisation of Ω. If S is in general position, it's even a triangulation of Ω, the anisotropic Delaunay triangulation.

Ordinary Voronoi diagrams have a certain degree of locality. Inserting a single new point usually doesn't change the diagram too much beyond the vicinity of the point. This allows for fast incremental construction algorithms. Anisotropic Voronoi diagrams aren't so regular, and their cells aren't necessarily connected. A cell also don't necessarily contain the point in S from which it was generated. Such a cell is called an *orphan*.

Upon insertion of a new point, a "correct" incremental construction algorithm has to search through the whole diagram for points closer to the new point than to their old one, possibly making orphans in the process. However for the goal of meshing we want a Voronoi diagram that has no orphans, and it turns out we can use a sloppy incremental insertion algorithm that "forgets" to make new orphans.

This gives rise to the notion of a *loose (anisotropic) Voronoi diagram*. It can be incrementally constructed as outlined in section 8 of [3]. If the point set S conforms to the preconditions of Lemma 2 then the finally constructed loose anisotropic Voronoi diagram is equal to the "non-loose" anisotropic Voronoi diagram, and its dual is then, by the same lemma, the anisotropic Delaunay triangulation.

Theorem 3. *For a set S of n points the algorithm of section 8 of [3] has worst case time bounds of $\Omega(n^2)$ and $O(n^3)$. (to tighten this range is an open problem).*

It could be that our input is a PSLG, that is we have next to the point set S also a set of edges L that we want to have in the output. In that case we can first create the (loose) anisotropic Voronoi diagram and then refine it. The refinement process looks for edges $\overline{ab}$ in L that cross a Voronoi cell other than those of a and b. Such an edge is split by inserting a new point into S somewhere along the edge and updating the diagram accordingly.

When the diagram agrees with all the edges in L we can further refine the result for quality. We do this by inserting more points to eliminate orphans, poorly shaped triangles in the dual, and other irregularities. This is done by finding *violator* points in the space and putting a point into S at that location. A point q is a violator if:

- it lies on the shared edge of the Voronoi regions of points $v, w \in S$, but is not wedged between v and w, or
- it dualises to a triangle that is inverted, too large, or has an angle less than some given constant

That the refinement process eliminating all violators ends is the subject of section 10 of [3]. The resulting mesh has the wanted anisotropies and provable quality properties.

References

[1] M.W. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry*, number 4 in Lecture Notes Series on Computing, pages 47–123. World Scientific, second edition, 1995.

[2] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001. 0-521-79309-2.

[3] F. Labelle and J.R. Shewchuk. Anisotropic voronoi diagrams and guaranteed-quality anisotropic mesh generation. *Proceedings of the 19th Annual Symposium on Computational Geometry*, pages 191–200, 2003.

2.2 Mesh generation

We will now cover mesh generation, also called Steiner triangulation. Our discussion of this topic is based on section 2.3 of [3]. Contrary to ordinary triangulation problems, Steiner triangulation problems allow adding new Steiner vertices if necessary. The triangulation must respect the input though, meaning that existing vertices may not be deleted, and edges that are part of the input must be fully covered in the output.

As we already saw in section 2.1.1, any two dimensional input can be triangulated, so adding extra vertices is pointless without an optimisation criterion. One class of Steiner triangulation problems therefore ask for an optimal triangulation, given a fixed number of Steiner points. This is not very interesting from a practical perspective however, because having absolute bounds on the number of extra vertices used is often not as important as absolute bounds on the shape of the triangles in the resulting mesh. We will call these absolute bounds on the shape a *quality measure*. They give rise to a second class of problems, which asks for a minimum size triangulation satisfying the quality measure.

Steiner triangulation problems are very difficult to solve optimally, so researchers have turned to approximation algorithms. The goal is then to devise an algorithm that generates a triangulation that is as close as possible to the optimum triangulation for the given quality measure, preferably, within a constant factor.

Angles within a triangle are classified as follows: *obtuse angles*, which are larger than 90°, *right angles*, which are exactly 90°, and *acute angle*, which are smaller than 90°. The longest edge of a triangle is called its *hypotenuse*, the other two sides its *legs*.

The *aspect ratio* of a triangle is defined as the length of the hypotenuse divided by the height from the hypotenuse. The aspect ratio is a bound for the maximum and minimum angle in a triangle: if θ if the smallest angle in, then $|\frac{1}{sin\theta}| \leq$ aspect ratio $\leq |\frac{2}{sin\theta}|$. Also, if the aspect ratio is less than two, then all triangles are acute. A related concept to the aspect ratio is the length of the hypotenuse divided by the length of the shortest edge. This value is always smaller than the aspect ratio.

2.2.1 No small angles

This is the Steiner analogue to the Delaunay triangulation. But now we want absolute constraints on the minimum angle in the resulting triangulation. There are two caveats. First, the input may already contain small angles, e.g. on the border of the polygonal region to be triangulated. In that case, we would like to have bounds only for newly added angles. Second, the number of triangles generated may depend on the geometry of the input, as well as the size of the input. Consider for example a rectangle of size $1 \times A$, each triangulation bounding the minimum angle will need a number of triangles in the order of A. Results treated in this section will regard A as a constant.

The first solution to this problem was a grid based approach described in [1], to triangulate simple polygons. The triangles are also guaranteed to have no new angles smaller than $\tan^{-1}(\frac{1}{4}) \approx 14°$. No bounds are given on the size of the grid generated. The method roughly works as follows: overlay a fine enough square grid on the polygon, triangulate regions around

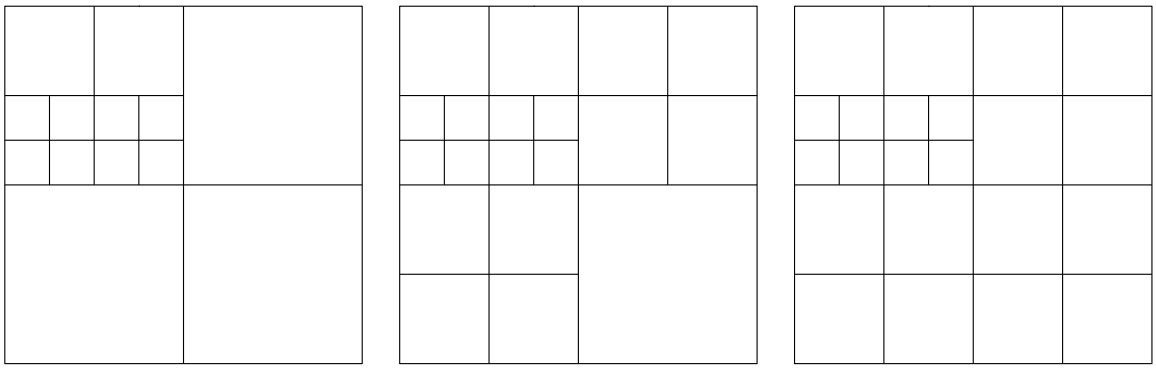


Figure 5: Quadtree subdivision: (a) unbalanced quadtree, (b) balanced quadtree, (c) strongly balanced quadtree

corners in the border, and then triangulate the remaining region inside the polygon. This approach also works for the more general case of PSLGs.

This method has been improved for point sets by using quad trees [5]. Quad trees use only fine grained squares where the input is fine grained, and therefore use less Steiner points than grid based approaches. The algorithm results in a mesh in which no angles less than 20° are present, while it uses $O(k)$ triangles, where k is the minimum number of triangles in a triangulation of the input with no angle smaller than 20°. These bounds are improved to 36° and 80° by constraining the balanced condition of the quad tree and by replacing squares with tiles with projections and indentations.

Defintion 7. *A* quadtree *is a tree of axis aligned squares, in which each square is either a leaf, or is split into four equal size, non-overlapping squares. It results in a recursive partition of a region, delimited by the root square. See figure 5 for examples.*

Each square can have at most four equal sized *neighbours*, one in each direction. If the diagonal siblings are also counted, a square has eight neighbours, usually called *extended neighbours*. Because the definition of quadtrees given above may result in very unbalanced trees (see figure 5(a)), a *balanced condition* is introduced. The balanced condition states that every side of an unsplit square (leaf nodes in the tree) has at most one vertex that splits it. See figure 5(b) for a balanced quadtree. An equivalent condition is that no two neighbouring leaf nodes may differ more than two in side length. This can be strengthened to extended neighbours, giving us a so called *strongly balanced* quadtree. See figure 5(c) for an example.

When the input is a point set the triangulation algorithm works as follows. It first builds a quadtree, starting with a large enough square containing all the points, e.g. the square concentric with, and twice the size of the bounding box. It stops splitting squares when all leaf nodes in the tree contain at most one point, and each leaf node that contains a point is surrounded by a 5 by 5 grid of equally sized leaf nodes that don't contain input points. The center of this grid is shown in figure 6(a). Now the corner of the containing square that is closest to the input point is located. In 6(a) this is the corner to the lower left of the input point. The quadtree is now warped by placing that corner node on the input point and reconnect the edges, like in figure 6(b).

The last step is to triangulate the ordinary squares and the warped squares. Ordinary squares are triangulated by adding an extra point in the center and, and connecting edges

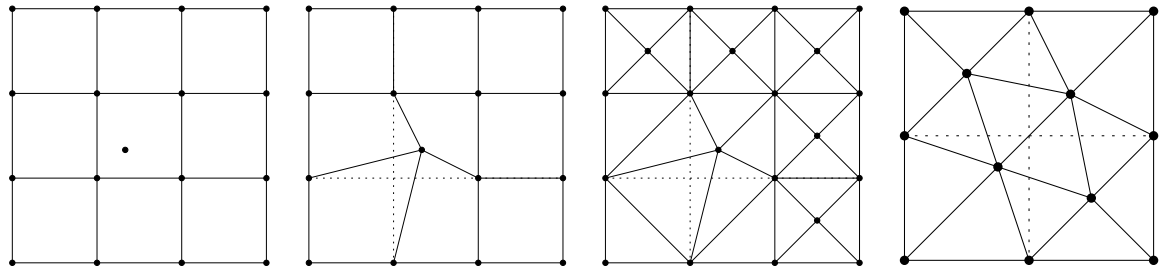


Figure 6: Quadtree triangulation: (a) the input point inside the square, (b) warp the quad tree by moving the closest corner from the input point to the location of the input point, (c) triangulate the warped quadtree (d) improved triangulation of the quadtree region around an input point.

from the corners to it. If one of the edges of the square is split (an edge can only be split at most once, due to the balance condition), an edge from this vertex to the center is also added. This results in at most 8 triangles, with angles of 90° and 45° degrees only. The warped squares are triangulated by adding the shortest diagonal. A case analysis for the generated triangles shows that there are no angles created that are smaller than 20°.

Theorem 4. *The number of triangles generated by the quadtree algorithm described above is $O(m)$, where m is the number of triangles in an optimal triangulation which has no angles smaller than 20°. The algorithm runs in $O(n \log n + k)$, where k is the output size, i.e. the number of triangles generated.*

For the proof, see [5]. They also prove that it holds for any angle smaller than 20°.

By using the strongly balanced condition instead of the ordinary balanced condition, we can improve these bounds by triangulating the squares a little smarter. The downside is that the constant factor in the number of triangles increases.

The algorithm works as follows. Because the extended balanced condition holds, each (warped) leaf square is adjacent to neighbours that differ at most a factor 4 in size. We can label the edges of a leaf square relative to the size of its neighbours. For each edge there are three possibilities: the adjacent leaf square is twice as small, equally sized, or twice as big. A square can now have $3^4 = 81$ possible labellings, of which 28 are possible, which can be reduced to 9 by removing rotations of the same labelling. [5] comes up with tiles for each of the 9 squares that have triangles of aspect ration $\frac{5}{3}$. The quadtree around an input point is generated such that we have a box around the input point that is small enough to not influence the angles in a triangulation using a tile that covers multiple squares.

Theorem 5. *The number of triangles generated by the improved quadtree algorithm described above is $O(m)$, where m is the number of triangles in an optimal triangulation which has angles bounded between 36° and 80°. The algorithm runs in $O(n \log n + k)$, where k is the output size, i.e. the number of triangles generated.*

Bounds of 51° for the minimum angle and 72° for the maximum angle could be possible without using a lot more triangles, but further improvement is thought to be difficult, because it requires a mesh in which all new vertices have degree 6.

A similar quadtree based algorithm, also described in [5], can generate a mesh for a polygonal region without small angles. The algorithm is more complicated, because it also

needs to warp and properly triangulate squares that are intersected by line segments from the input. The aspect ratio of any new triangle in the mesh generated by this algorithm is smaller than or equal to 5, and hence the smallest new angle is larger than 18°.

Theorem 6. *The number of triangles generated by the quadtree algorithm for polygons is $O(m)$, where m is the number of triangles in an optimal triangulation which has angles bounded above* 18°. *The algorithm runs in $O(n \log n + k)$, where k is the output size, i.e. the number of triangles generated.*

A modified version of the quadtree algorithm described above can also generate a mesh with no small angles for a PSLG. We will not describe it here, because a *Delaunay refinement* algorithm, achieves better bounds, while being conceptually simpler. A Delaunay refinement algorithm maintains a Delaunay triangulation of the input, and at each iteration adds a point at a strategic location, and the recomputes the Delaunay triangulation incorporating the new point. The procedure is ended whenever the mesh satisfies some property.

The first refinement method with guaranteed bounds on the shape of the triangles is described in [7] . The algorithm works on PSLGs, and operates by adding vertices to the center of a triangle with circumcircle radius bigger than the input feature size. The algorithm stops when no such triangles remain. The input feature size is defined as follows:

Defintion 8. *The* input feature size *is the smallest distance between two points in a point set; the smallest distance between any point and any edge not incident to it in a PSLG, with the exception that for polygons, only interior distances are considered.*

The algorithm can optionally be supplied with a user defined *grading function*, which specified when a generated triangle is small enough. This is useful when knowledge about the domain to be simulated is present.

The algorithm generates triangles with a minimum angle of 30 degrees, if the size of the boundary edges is between s and $\sqrt{3}s$, where s is the feature size of the input. The bound on the largest angle is no better than $180 - 2 * 30 = 120$. However, no bounds on the number of triangles used is guaranteed.

The algorithm described in [8] provides such a bound, but has weaker angle bounds in general. This algorithm adds vertices to the center of the circumcircle of the triangle with the smallest angle, and recomputes the Delaunay triangulation. Whenever a new point lies within the enclosing circle of an input edge, this point is not added however, but the input edge is split into two equally sized input edged, after which the Delaunay triangulation is updated. The splitting of triangles with small angles goes on until all triangles have a specified minimum angle. But this doesn't necessarily halts; but this turns out to be the case when the minimum angle specified is smaller than 20°. Furthermore, the following theorem holds:

Theorem 7. *The number of triangles generated by the refinement described above is $O(m)$, where m is the number of triangles in an optimal triangulation which has new angles bounded above some constant smaller than* 20°. *The algorithm runs in $O(k^2)$, where k is the output size.*

According to [8], the algorithm usually behaves well for angle bounds no larger than 30°, which makes this algorithm even more attractive. Another advantage of this algorithm over a quadtree based algorithm is the fact that this algorithm doesn't generate axis-aligned elements when the input is not axis aligned, which might harm the subsequent finite element method calculations. But these improvements are traded for a theoretical increase of running time.

2.2.2 No large angles

This problem is in some sense the dual of the previous problem: we want to obtain a mesh with no angles larger than some constant, with as few extra Steiner points as possible. Upper bounds of 90° are especially interesting, because some finite element methods behave nicely in that case. This problem is called the nonobtuse triangulation. It is also interesting, because a nonobtuse triangulation is also the Delaunay triangulation.

In the previous section, we already saw a few algorithms that gave bounds on the lower bounds as well as the upper bound of the angles generated: [1] gives an upper bound of 90° for PSLGs, but no bound on the mesh size, [5] gives an upper bound of 80° for point sets, and bounds the mesh size by a constant factor from the optimum. Because these algorithms also bound the minimum angle, the size of the resulting mesh is bounded not only by the size of the input, but also by its geometry.

The first quadtree algorithm of the previous section can be extended to generate only nonobtuse triangles, as described in [5]. This algorithm works by triangulating clusters that don't become well separated when a square in the quadtree repeatedly splits itself. This way, the geometry of the input doesn't influence the running time of the algorithm with an extra logarithmic factor.

Theorem 8. *A point set of n points can be triangulated with $O(n)$ nonobtuse triangles. The algorithm runs in time $O(n \log n + k)$.*

[2] describes a grid-based algorithm, giving no obtuse angles for a polygon with holes. The first step is to add a vertical line segment from each corner of the polygon to the opposite boundary, where the intersection generating a new point. This divides the polygon into vertical *slabs*. Then second step is to add a horizontal line segment from each point to the last vertical segment possible. The polygon is now partitioned into rectangles, quadrilaterals, right triangles and obtuse triangles. See figure 7(a) for an example of this initial partition.

The rectangles can be triangulated easily by adding a diagonal, because none of its edges is subdivided by some horizontal segment. Right and obtuse triangles are the most difficult to triangulate, because their legs may be subdivided by other edges. [2] describes a recursive procedure that works for all cases. Figure 7(b) shows the general idea of this procedure applied to triangle abc with only intersections at leg ac: the intersection that is closest to c is removed, which results in the same problem on triangle abc', but with one intersection less. The base case of the algorithm is easily handled by adding a vertical line through c.

What remains is the case of quadrilaterals, which can now be triangulated by adding the shortest diagonal and applying the procedure described above for the two triangles generated. The triangulation of an obtuse triangle with m intersections on its legs requires $O(m^2)$ triangles. But because every point in the initial partitioning induces at most one such intersection point for all triangles, the total number of triangles in the complete mesh is only quadratic.

Theorem 9. *The algorithm described above generates a nonobtuse mesh of a polygon, with size $O(n^2)$, where n is the number of corners in the polygon. It runs in $O(n \log n + k)$ where k is the number of triangles generated.*

[6] improves this bound by applying a circle-based algorithm. The polygon is filled with $O(n)$ circles, such that each region not covered is bounded by at most four edges or arcs. Then edges are added from the center of circles to points of tangency (points where other

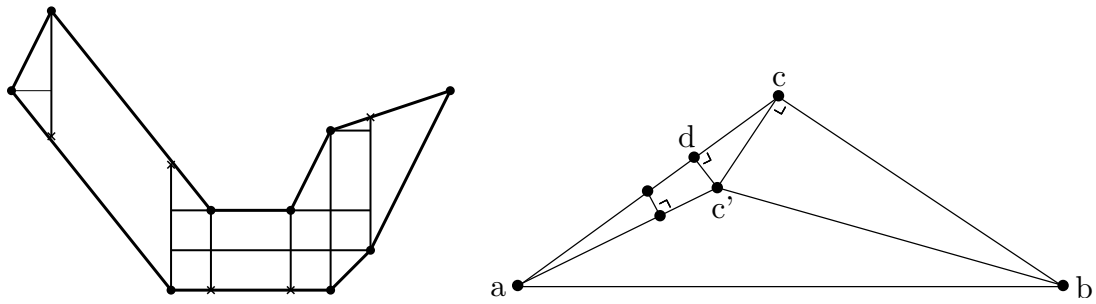


Figure 7: Grid based method to nonobtuse triangulation: (a) the first step in the process: extend vertices with vertical and then with horizontal lines, (b) tile to triangulate an obtuse triangle with intersections on its leg.

circles or edges touch the circle). This gives a subdivision of the original polygon, consisting of smaller polygons. These polygons can be triangulated without introducing extra vertices on the boundaries that do not bound the original polygon, guaranteeing that the triangulations of the smaller polygons fit together.

Theorem 10. *The algorithm described above generates a nonobtuse mesh of a polygon, with size $O(n)$, where n is the number of corners in the polygon. It runs in $O(n \log^2 n)$ time for polygons with holes, and in $O(n \log n)$ for simple polygons.*

It is currently unknown whether a PSLG can be triangulated without obtuse angles in polynomial time. The same holds for polygons, for which both the interior and exterior have to be triangulated. If only large angles are to be avoided, the first problem is solvable in polynomial time using a variety of algorithms described in [3].

2.2.3 Maximise the minimum height

The edge insertion algorithm of section can be used to find a triangulation of a point set that maximises the minimum angle. The result is not necessarily a global optimum, but it will be if the points are in general position.

If we add Steiner points we can improve the maximum minimum height. For example, if we have a regular n-gon and triangulate it, there must be a triangle which has two of its edges on the n-gon boundary. Such a triangle has a small height of $\sin(\frac{\pi}{n})$, but if we add a Steiner point to the center of the n-gon we can create triangles with height nearly 1.

For a point set, the algorithms above for no large angles already solves the maxmin height problem. For a polygon, the input feature size s gives an upper bound on the minimum height achievable. In fact, $\Omega(s)$ is achievable, with a linear number of triangles:

Theorem 11. *A polygon with holes can be triangulated into $O(n)$ triangles of height $\Omega(s)$.*

The algorithm works as follows: first acute corners of the polygon are cut off to be triangulated at the very end without Steiner points. Next, we imagine a grid with $\frac{s}{3}$ x $\frac{s}{3}$ squares overlaid on the polygon. Remove all horizontal and vertical lines of the grid that aren't incident to a square (adjacent to a square) containing a vertex. We end up with $O(n)$ boundary triangles and $O(n^2)$ interior rectangles, all with height $\Omega(s)$. We can carefully

merge some rectangles to reduce complexity and then use a sweepline algorithm to triangulate everything with linear complexity. We end up with $O(n)$ Steiner points, and $O(n)$ triangles of height $\Omega(s)$.

2.2.4 Minimum weight

Finding the minimum weight triangulation (MWT) of a point set, that is the triangulation with the least total edge length, is largely an open problem. It is neither known to be NP-complete, nor known to be solvable in polynomial time. If the weight isn't the edge length but an arbitrary function the problem *is* NP-complete, so most effort has been expended into finding approximations of the MWT.

Any triangulation has a total edge length of $O(n)$ times the minimum. The Delaunay triangulation, as well as triangulations made by edge flipping aiming for minimum length, can be $\Omega(n)$ times the minimum. Edge insertion might give better results, but that remains an open problem. A greedy triangulation can be as bad as $\Omega(\sqrt{n})$, though $O(1)$ for a convex polygon instead of a point set. The best known scheme is to partition the convex hull into convex polygons, then using one of the above heuristics to triangulate these. This gives an $O(\log n)$ approximation, and can be implemented with a running time of $O(n^2 \log n)$

2.2.5 Quadrilateral Meshing

The circle based approach to triangulation described in section 2.2.2 can be adapted to work for quadrilateralisation too. One of these techniques is *Voronoï quadrilateralisation*, it makes use of the so called *geodesic Voronoï diagram.*

Defintion 9. *The geodesic Voronoï diagram of a set of point sites in a polygonal domain is a partition into cells. In each cell lie the points whose geodesic distance is closest to the cell's site.*

The algorithm will find a point set whose geodesic Voronoï diagram is a quadrilateralisation of the mesh. We modify the original circle packing algorithm. We will place a circle on each of the vertices of the boundary, centered at that same vertex with radius half the minimal distance between vertices. Next instead of filling the rest of the polygon's interior with circles, trying to create tangencies with the boundary, we try to make circles that are tangent to other circles that have their center on the boundary. Another constraint is that no tangent point between circles can lie on the boundary of the domain, but only in the domain interior. The result of all this will be a packing with three sided and four sided gaps. In [4] a time bound is proven on this construction.

Theorem 12. *Given a polygonal domain and a, in time $O(n \log n)$ we can find a packing as described above. The geodesic Voronoï diagram of the points of tangencies between the circles of the packing form a quadrilateral mesh of the domain.*

[4] also shows that the *power diagram* of the circles created by the above packing is the planar dual of the mesh we are looking for. The *power* of a circle with respect to a point is defined as follows:

Defintion 10. *The power of a circle with respect to a point is the squared radius of the circle minus the squared distance between the point an the circle's center.*

The power diagram can now also be defined:

Defintion 11. *The power diagram of a set of circles is partition of the plane into cells, where each cell consists of the points for which the power of the circle corresponding to that cell is greatest.*

Having these definitions the following theorem is proven in [4]:

Theorem 13. *The quadrilateral mesh as defined by theorem 12 contains an edge between two points if and only if the corresponding circles' cells share an edge in the power diagram of the circles in the packing and the circumcircles of the gaps.*

The above algorithm is expected to generate between $3n$ and $4n$ quadrilaterals, based on the estimate that the original circle packing algorithm for triangulations is expected to create between $20n$ and $30n$ triangles and 8 triangles from the original algorithm typically form one quadrilateral in this adapted algorithm.

Using the algorithm described above, we can also create a quadrilateralisation in which each quadrilateral has two opposite right angles. It works as follows. We form a quadrilateralisation as in the previous section, and we overlay the power diagram of the corresponding circle packing. Now for each quadrilateral Q we drop perpendiculars to all four of its sides from the Voronoï site that is contained within Q. This results in the following theorem from [4]:

Theorem 14. *In $O(n \log n)$ time we can partition any polygon into a mesh of $O(n)$ quadrilaterals each having two opposite right angles.*

Another type of "good" quadrilaterals are kites. Kites are convex quadrilaterals that have one axis of symmetry along one of their diagonals. They have nice theoretical properties. We will now present an algorithm that can construct a quadrilateralisation, where each quadrilateral is a kite. As in the previous algorithms we first create a circle packing. Then we connect tangent circles with a radial line segment through their point of tangency. In figure 8 and figure 7 of [4] it shown that this is always possible, with a case distinction on the 7 different situations that can occur, this is part of a proof of the following theorem:

Theorem 15. *In $O(n \log n)$ time we can partition any polygon into a mesh of $O(n)$ kites.*

The kite mesh generated by the previous algorithm can be further refined to create quadrilaterals with maximum angle 120°.

Theorem 16.

In $O(n \log n)$ time we can partition any polygon into a mesh of $O(n)$ quadrilaterals with maximum angle 120°.

This theorem is also proven in [4] with a case distinction on the different kites that can exist, and how these kites can then be subdivided to create the required quality quadrilaterals. See figure 9

References

[1] B.S. Baker, E. Grosse, and C.S. Rafferty. Nonobtuse triangulation of polygons. *Discrete & Computational Geometry*, 3:147–168, 1988.

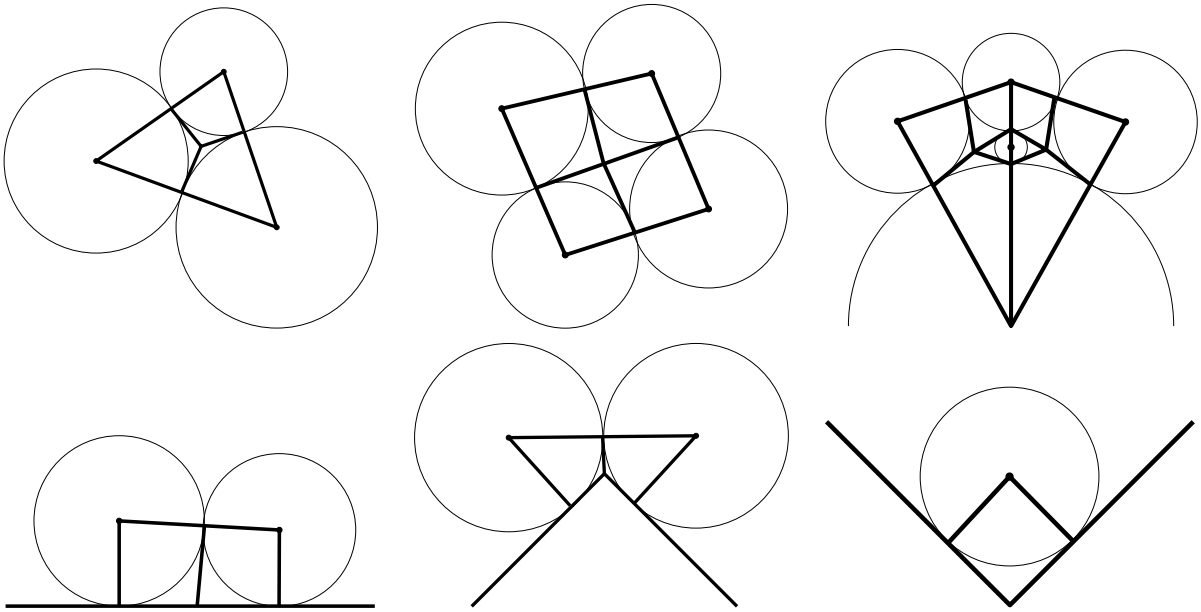


Figure 8: 6 of the 7 different situations that can occur when circle packing with kite mesh generation, these pictures come from figure 6 in [4].

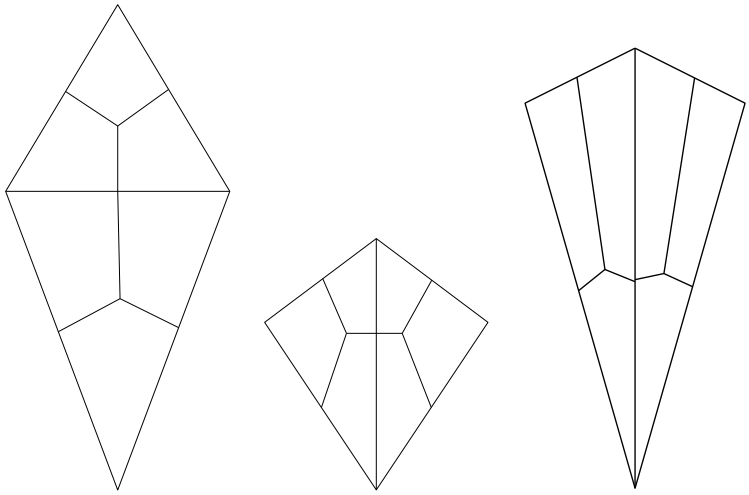


Figure 9: The 3 different kite types that can occur and the corresponding way to divide them into quadrilaterals with a maximum angle of 120°. This image comes from figure 9 in [4].

[2] M.W. Bern and D. Eppstein. Polynomial-size nonobtuse triangulation of polygons. *Proceedings of the 7th Annual Symposium on Computational Geometry*, pages 342–350, 1991.

[3] M.W. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry*, number 4 in Lecture Notes Series on Computing, pages 47–123. World Scientific, second edition, 1995.

[4] M.W. Bern and D. Eppstein. Quadrilateral meshing by circle packing. *International Journal of Computational Geometry and Applications*, 10(4):347–360, 2000.

[5] M.W. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48:384–409, 1994.

[6] M.W. Bern, S.A. Mitchell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. *Proceedings of the 10th Annual Symposium on Computational Geometry*, pages 221–230, 1994.

[7] L.P. Chew. Guaranteed-quality mesh generation for curved surfaces. *Proceedings of the 9th Symposium on Computational Geometry*, pages 274–280, 1993.

[8] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. *Proceedings of the 4th annual ACM-SIAM Symposium on Discrete algorithms*, pages 83–92, 1993.

2.3 Heuristics

As we mentioned earlier, there is also a practical, engineering perspective to mesh generation. Before theoreticians studied mesh generation, engineers were already applying automated mesh generation algorithms to real world problems. These *heuristic algorithms* usually didn't have any theoretical guarantees on the quality and size of the mesh generated; they were selected because they often yielded meshes with good quality.

Heuristic algorithms sometimes also take knowledge of the simulation into account, to steer the mesh generation, e.g. by using more triangles in places where there will be more variation in the simulation. Because most practical problems in two dimensions are defined on polygons, most study has gone into that. The main classes of heuristic algorithms are:

- Grids: overlay a square grid over the input domain, then cut the grid off at the border of the domain, and finally triangulate the cut off grid.
- Quadtrees: they have been used in practical mesh generation tools before computational geometers started researching them. The methods employed are similar as the ones described in [1], but no performance guarantees are provided.
- Advancing front: first discretise the border of the input polygon properly, then work inwards, adding Steiner points and triangles on well chose places, e.g. to create a Delaunay triangulation.
- Mesh refinement: start with an initial triangulation, for example the constrained Delaunay triangulation, and keep refining the mesh until it satisfies some criterion. Refining the mesh might involve moving vertices, flipping edges, subdividing triangles, et cetera.

- Domain decomposition: first split a polygon into simpler polygons, e.g. convex polygons, and then triangulated with certain density using a simple triangulation algorithm

References

[1] M.W. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48:384–409, 1994.

3 Three-dimensional meshing

In this section we will go over the case of our domain to be meshed being 3-dimensional. Many techniques that work in 3 dimensions actually work in higher dimensions as well, but it's easier to discuss them in the specific case of 3 dimensions. It also happens to be the case most interesting to practical application.

The most likely inputs to a 3-dimensional meshing algorithm are:

- Point-set, which consists of a set of points in 3-dimensional space.
- Non-simple polyhedron, which could have holes so that it's boundary is not necessarily connected.
- Simple polyhedron, which is a special case of a non-simple polyhedron where holes are not allowed.

3.1 Tetrahedralisation

Not adding Steiner points when meshing is a more severe restriction in 3 as in 2 dimensions. In 2 dimensions, the simplest element is a triangle, and it's possible to divide any polygon into triangles. In 3 dimensions, the simplest element is a tetrahedron, but not every polyhedron admits a tetrahedralisation without the addition of Steiner points.

With the addition of Steiner points it *is* always possible though, and for a point set we won't even need to add Steiner points. One way of meshing a point set in 3 dimensions is with Delaunay tetrahedralisation [1].

3.1.1 Delaunay tetrahedralisation

The concept of a Voronoi diagram generalises easily to higher dimensions. A Voronoi region of a point $P \in S$ still contains the points that are closer to P than any other point in S. The difference is the shape of the regions. In 2 dimensions they will be (possibly unbounded) convex polygons, in 3 dimensions they will be convex polyhedra.

A point P where multiple Voronoi regions meet lies closest to multiple points of S, i.e. those points lie on a sphere with P at the center. There could be arbitrarily many points on a common sphere, but again we will assume our input to be in general position. In 3 dimensions, this means that no 5 points must lie on a common sphere and no 4 points on a common plane.

Defintion 12. *As in the 2-dimensional case, the* Delaunay tetrahedralisation *of a point set in 3 dimensions is defined as the dual of its Voronoi diagram. This dual has an edge between any two points sharing a Voronoi facet.*

When the point set is in general position, each Voronoi edge is shared by three facets, making the facets in the dual triangles. General position also implies that the Voronoi vertices are incident to 4 Voronoi regions, making the polyhedra in the dual indeed tetrahedra.

The Delaunay tetrahedralisation has properties similar to the 2-dimensional case. It is equivalent to the lower part of the convex hull of the points "lifted" to the 4-dimensional paraboloid $(x, y, z, x^2 + y^2 + z^2)$, for instance. However, its optimised qualities are not really the same:

- It does not maximise the minimum angle. "Slivers", i.e. very slim tetrahedra can occur. These can be removed in a subsequent refinement phase that inserts Steiner points.
- It minimises the maximum containment sphere, but it does not minimise the maximum circumsphere.

As far as construction goes, some of the Delaunay triangulation techniques also generalise to tetrahedralisation. Computing the lower convex hull of the lifted points can be done in $O(n^2)$ time and throwing the fourth coordinates away gives the Delaunay tetrahedralisation. Since the convex hull can have $\Omega(n^2)$ complexity, this method is optimal in the worst case. However, other more direct $O(n^2)$ algorithms exist. The edge flipping algorithm of section 2.1.2 and can be extended to 3 dimensions, as described in [1]. Incremental construction can also be adapted, as discussed in the same reference.

References

[1] H. Edelsbrunner. *Geometry and Topology for Mesh Generation.* Cambridge University Press, 2001. 0-521-79309-2.

3.2 Mesh generation

3.2.1 Hexahedralisation

As in 2 dimensions with quadrilaterals, meshing with hexahedra in 3 dimensions has many advantageous properties for numerical methods, but isn't as well-studied as meshing with tetrahedrons. Although many heuristics exist, theoretical results are fewer. This section will discuss the results described in [1], which includes an algorithm that meshes the interior of a surface of n quadrilaterals with $O(n)$ hexahedra.

The problem studied is to use hexahedra to fill a connected 3-dimensional region bounded by a surface of an even number n of quadrilaterals. The boundary must appear unchanged in the output, as faces of the generated hexahedra, but the interior can have Steiner points added where needed.

A theorem due to Mitchell and Thurston states that this problem can always be solved, but it doesn't give any bounds on the size of the generated mesh. The paper [1] does show a bound, leading to the following theorem:

Theorem 17. *Any simply connected three-dimensional domain with an even number n of quadrilateral boundary faces can be partitioned into a mesh of $O(n)$ hexahedra respecting the boundary.*

The paper also gives an algorithm to construct such a mesh. It works in two phases, first quadrilateralising the interior and then the "buffer layer" between the interior and the surface:

- First a surface S isomorphic to the boundary B is placed (slightly) more towards the interior of the region. The points of S are connected to the corresponding points in B, thus forming quadrilaterals between the two surfaces.

  S is then triangulated and its volume tetrahedralised. We can get $O(n)$ tetrahedra for example by connecting each triangle on S to a common interior vertex. The tetrahedra are then split into 4 hexahedra each, in such a way that the subdivisions of adjacent faces match. This also turns the quadrilaterals connecting B and S into pentagons, which will have to be fixed.

- The vertices and edges of B form a planar bipartite graph. Let U and V with $|U| < |V|$ be the sets of vertices forming the bipartition. Each of the pentagons connecting B and S has one edge that has one vertex in U and one vertex in S. When we split all such edges the faces connecting B and S turn into hexagons. After this the polyhedra making up the buffer layer consist of seven quadrilaterals and four hexagons each. The hexagons are split into 2 or 3 quadrilaterals each, in such a way that an odd number is divided into 2 and an odd number is divided into 3. Now all buffer cells are composed of 16 or 18 quadrilaterals, and are easily meshed into hexahedra.

References

[1] D. Eppstein. Linear complexity hexahedral mesh generation. *Proceedings of the Symposium on Computational Geometry*, pages 58–67, 1996.

4 Conclusions

We have tried to give a survey of the theoretical results in the field of (unstructured) mesh generation. The field is a vast multi-dimensional space, spanning many different inputs, optimisation criteria, and spatial domains. It's impossible to capture the entire field in its full complexity in a single paper and we limited ourselves to the most practically relevant and/or well-understood areas. We hope our survey provides a useful basis for starting a study of the field, and the interested reader is encouraged to explore the references.