Surface Triangulation: A Survey

Subodh Kumar
Department of Computer Science
University of North Carolina
Chapel Hill NC 27599

July 3, 1996

Abstract

This paper presents a brief survey of some problems and solutions related to the triangulation of surfaces. A surface (a two dimensional manifold, in the context of this paper) can be represented as a three dimensional function on a planar disk. In that sense, the triangulation of the disk induces a triangulation of the surface. Hence the emphasis of this paper is on triangulation on a plane. Apart from the issues in triangulation, this survey talks about the known upper and lower bounds on various triangulation problems. It is intended as a broad compilation of known results rather than an intensive treatise, and the details of most algorithms are skipped.

1 Introduction

This survey assumes familiarity with the fundamental concepts of computational geometry. We define the triangulation problem as follows:

Input: i. A set S of points, $\{p_i\}$, such that each p_i lies on the surface
ii. A set of conditions, $\{\mathcal{C}_i\}$

Output: A set S' of triples $\{(p_{i_1}, p_{i_2}, p_{i_3})\}$ such that each $p_{i_j} \in S$ and $\mathcal{C}_i(S') = TRUE$, $\forall \mathcal{C}_i$.

The vertex pair $(p_i, p_j$ can also be thought of as an edge e_{ij}.

One condition that we always include (and will be assumed in the rest of the paper) is:

If two different edges e_ij and e_kl intersect, i, j, k, and l must not all be different.

Sometimes each pair (or edge) (p_i, p_j) is also given a weight, $\mathcal{W}$. The weight of the triangulation S' can then be computed as

$$\sum_i \left(\mathcal{W}(p_{i_1}, p_{i_2}) + \mathcal{W}(p_{i_2}, p_{i_3}) + \mathcal{W}(p_{i_1}, p_{i_3})\right), \quad (p_{i_1}, p_{i_2}, p_{i_3}) \in S'$$

An often used condition is: $\mathcal{W}(S')$ be the minimum over all valid triangulations. Such triangulation is referred to the minimum weight triangulation. The weight of an edge, very often, is just the geodesic (shortest distance along the surface [Nei66]) distance (ℓ_1, ℓ_2 or ℓ_∞) between the end points. Some other useful conditions are based on angles of the triangles, maximum weight of triangle edges etc. Also, only maximal triangulations are considered. The edges on the convex hull of S are included in the triangulation, unless we are triangulating a structure, for example, a polygon.

We do not lose much by concentrating only on triangulation in planar domains, since a surface can be broken into a small number of planar domains. Only a cursory description is presented here. For details see any text book, e.g. [Nei66], on differential geometry.

1.1 A surface is 2D

A surface M can be represented as map χ from a two dimensional disk D in $\mathcal{R}^2$ to $\mathcal{R}^3$. The map χ should be 1-to-1 and continuous. Thus, for each point $(u, v) \in D$ there exists a point (x, y, z) on M such that $\chi(u, v) = (x, y, z)$. For each point on a there must exist such a proper disk such that $\chi(D) \subset M$. Most surfaces of interested can be broken into a few (i.e. a finite number) such patches. Map of a triangle

$$\{(u_1, v_1), (u_2, v_2), (u_3, v_3)\} \in D$$

is

$$\{\chi(u_1, v_1), \chi(u_2, v_2), \chi(u_3, v_3)\} \in M.$$

1.2 Applications

The need for triangulation arises in a wide variety of application ranging from physics to meteorology to mathematics. Many applications need to find triangulation in 3D, i.e. tetrahedralization of a subset of $(\mathcal{R}^3)$, but this survey does not address the issues there (Refer [Yvi89]). Though I will note that many of the results presented here extend to 3D.

Robotics people use triangulation to plan the motion of a robot. A similar application lies in visibility computation, which, in turn, is used extensively in computer graphics to perform hidden surface elimination. Triangulation is also useful in rendering images, since current graphics hardwares can draw triangles rather efficiently. Triangulation are also used to compute magnetic fields on a given domain [CSS83]. Finite element analysis methods extensively use domain triangulation to make the problem tractable [Lis94, ZSZZ90]. Nearest neighbour computation can be speeded up using triangulation. Computer vision research benefits from the use triangulation to represent stereo data [LSFB88, FLB90]. Meteorologists routinely triangulate their domains to perform weather analysis. Mathematicians can efficiently perform the interpolation of multi-variate functions by triangulating the domains. Triangulations also appears in a number of algorithms in chaos theory to solve differential equations.

1.3 Organization

Rest of this paper is organized as follows. In section 2 some simple triangulation schemes are discussed. To find about triangulation of polygons see section 3. A brief discussion of surfaces is presented in section 4. Finally section 5 concludes the survey. Informal terms like 'left most' to mean 'the point with the minimum `X` coordinate' are used occasionally; the meaning should be clear from the context.

2 General Triangulation

Two popular schemes to compute legal triangulations are listed below.

2.1 Greedy Triangulation

One technique to triangulate a set of points is the greedy method [DG70]. It keeps adding edges that do not intersect any of the previously added edges, till no more new edges can be added:

$L \leftarrow Set\{(p_i, p_j)\}, \ \forall p_i, p_j \in S$
$GT \leftarrow \phi$
While $L \neq \phi$ do
begin
  $W \leftarrow (p_y, p_z) \mid \mathcal{W}((p_y, p_z)) = \min_{i,j} \mathcal{W}(p_i, p_j)$
  $GT \leftarrow GT \cup \{W\}$
  $L \leftarrow L \setminus \{W\} - \{m \in L | W \text{properly intersects } m\}$

end

This same basic structure has been implemented by a number of different researchers in different ways achieving different overall complexity. [Lin89] shows that if S is uniformly distributed in a unit square, the greedy triangulation of S can be computed in $O(nlog^{1.5}n)$ expected time. [Gol89] presents a very simple $O(n^2 \log n)$ deterministic algorithm:

$T_G \leftarrow \phi, T \leftarrow GDT(S, T_G)$
While more triangulation is needed, do
    find the shortest edge uv not in T_G such that u and v are visible to each other
$T_G \leftarrow T_G \cup \{uv\}$
$T \leftarrow GDT(s, tg)$
Output T_G.

[DDMW94] brings the random complexity down to $n \log n$. It terminates in $O(n \log n)$ expected time, if the input point set is randomly distributed over any convex shape.

[LL90] brought down the deterministic complexity to $O(n^2)$ using Voronoi diagram with barriers, an extension to Voronoi diagrams where a number of the output edges are specified in the input. Let $G = (V, E)$ be a planar straight line graph. For $v \in V$ the region $R(v)$ is defined to contain all points p in the plane such that the shortest, open straight-line segment between p and a vertex of G that does not intersect any edge of G is (p, v). The minimal set of straight-line segments and half-lines outside G that partitions the plane onto regions $R(v), v \in V$, is called the Voronoi diagram with barriers of G, or $Vorb(G)$. [LL90] is outlined below:

$T \leftarrow \phi$
Compute $Vorb(G)$
While T is not a complete triangulation do
begin
    Find the shortest diagonal d of $G \cup T$ (using $Vorb(G)$)
    $T \leftarrow T \cup d$
    Update $Vorb(G \cup T)$ (compute it from $Vorb(G \cup T \setminus \{d\})$
end
T is the triangulation.

[LL90] also presents an algorithm to optimally triangulate convex polygons in linear time.

2.2 Delaunay Triangulation

Another popular way of triangulating a set of polygons is by using Delaunay triangulation. Delaunay triangulation is the dual of the Voronoi diagram (also called Drichilet or Thiessen tessellation). The Voronoi diagram of a set of points is a planar subdivision, with one face per point such that for all points lying on a face, F_p, corresponding to the input point, p, p is the closest input point. Delaunay triangulation can be computed by drawing an edge between two input points if their Voronoi faces are adjacent.

Delaunay triangulation of a point set can be found in $O(n \log n)$ [SH75] (this is also the lower bound on the problem). Voronoi diagram $Vor(S)$ of the point set S is computed as follows:

- Partition S into two subsets S_1 and S_2, of approximately equal sizes
- Compute $Vor(S_1)$ and $Vor(S_2)$ independently.
- Compute the polygonal chain σ separating S_1 and S_2.
- Discard all edges of $Vor(S_2)$ that lie to the left of σ, and all the edges of $Vor(S_1)$ that lie to the right of σ. The result is $Vor(S)$.

[Mau84] presents a randomized algorithm which has an expected running time of $O(n)$ if the input data set is uniformly distributed. It first partitions the input data set by drawing a grid with a small number of points per grid cell on average. This partitioning is done by the radix-sort algorithm [AHU74]. After the partitioning the algorithm processes each point

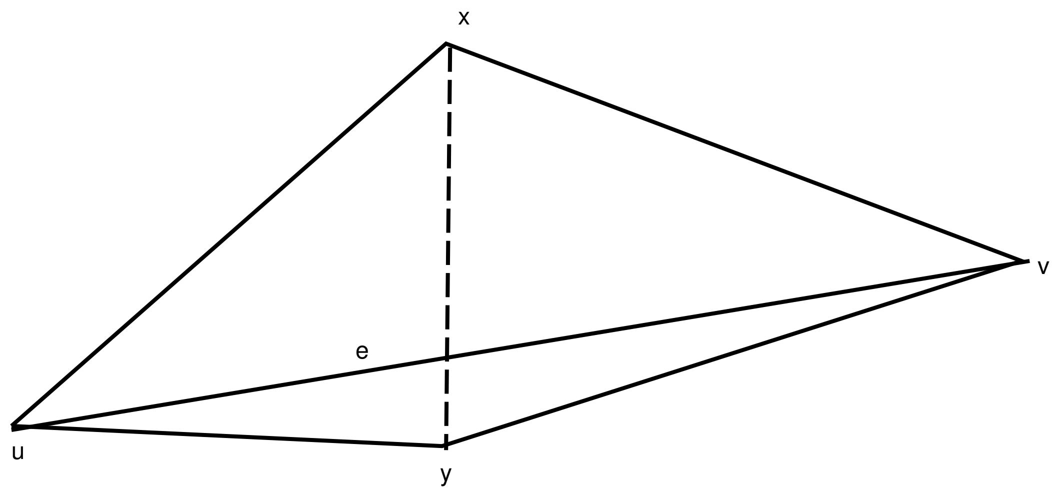


Figure 1: Flipping the diagonal

by rotating a ray about the point and connecting it to nearest neighbours till a 'conflicting' neighbour stops this process.

2.3 Constrained Delaunay Triangulation

The constrained (or generalized) Delaunay triangulation contains an edge uv iff vertex u is visible to vertex v and there exists a circle through u and v that contains no input point w, visible to edge uv. Vertex u is said to be visible to vertex v, if no input segment crosses the edge uv. Vertex w is said to be visible to edge uv if it is visible from any point on the edge uv.

A simple $O(n^2)$ algorithm to find constrained Delaunay triangulation is produced below:

- Start with a valid triangulation
  For any edge $e = uv$ not in input set and not on the hull, let Q be the quadrilateral $uxvy$ formed by merging the two triangles, uvy and xvu, on either side of e.
  Q does not belong to the triangulation if the minimum angle that e makes with the edges of Q is smaller than that the other diagonal xy makes.
- In such case switch the diagonal; generate triangles xyu and xvy.

This algorithm terminates after $O(n^2)$ flips. [LL86] presents a simple $O(n^2)$ algorithm to compute a valid starting triangulation. This algorithm can be modified to run in $O(n \log n)$ if the constraints are edges of a simple polygon.

[Che89] improves the upper bound to $O(n \log n)$ for arbitrary point set, thus hitting the lower bound of $\Omega(n \log n)$. It uses the divide and conquer approach:

- Sort the input points S by the `X` coordinate.
- Divide the region (of points) into vertical strips such that there is exactly one vertex per strip (This requires unique `X` coordinates. [MV93] addresses that problem effectively.)
- Triangulation is found for each strip and adjacent strips merged.
- Repeat till only one strip remains.

[CR90] reports an $O(n \log n)$ time incremental algorithm. Its strategy is to start with no constraints, and then add them, one at a time, to the current triangulation.

2.4 Optimal Triangulation

Optimal triangulation is the problem of finding a triangulation with the minimum total weight. The weight of an edge can be its Euclidean length. Optimal triangulation of a set S is denoted as $OT(S)$. [ACNS82] shows that every planar drawing of a graph with n vertices and m edges ($m \geq 4n$) contains 10^{13n} graphs with no edges crossing each other. This means that even if we could restrict our attention to the set of valid triangulations, there are an overwhelming number of possible triangulations to choose from. Though the minimum weight triangulation has not been proven to be NP-complete, there are no known polynomial time algorithms. All we can hope with polynomial time algorithms is get close to the minimum. A number of heuristics have been reported to work well. In fact the greedy triangulation and the Delaunay triangulation themselves can be considered heuristics for optimal triangulation. Let us denote the weight of these two triangulations for a given set of points as GT and DT respectively.

We define the quality $R(S')$ of a triangulation S' to be the ratio of $\mathcal{W}(S')/OT(S)$. We can quantify the quality of a class of triangulations $\mathcal{T}$ by $\max\{\mathcal{W}(S')/OT(S)\}, S' \in \mathcal{T}$.

Neither Greedy triangulation (GT) nor Delaunay triangulation (DT) are optimal [Llo77]. They are not even close to optimal – [MZ79] showed that $R(GT)$ is $\Omega(n^{\frac{1}{3}})$ and $R(DT)$ is $\Omega(\frac{n}{\log n})$. [Kir80] tightened the bound for $R(DT)$ to $\Omega(n)$. Of course, this is also the lower bound since any reasonable triangulation will be no worse, as no triangulation can have $< 3n$ edges. The length of each edge is less than the diameter of S, which in turn is less than half the length of Convex Hull of S [Sha78]. And since the edges of the hull lie in the optimal triangulation the ratio is not more than $O(n)$. In the special case when S is convex, [LL87] formally proves that $R(DT)$ is, in fact, $O(1)$.

[Lev87] tightened the bound for $R(GT)$ to $\sqrt{\frac{n}{2}}$. [Lin86] showed that though in worst case these triangulations can be quite un-optimal, on average they both behave well. [Lin86] proves that both $R(GT)$ and $R(DT)$ are $O(\log n)$ with high probability, using a maximal rectangle around the set S. If the length of this rectangle is L and the width is W,

$$Pr\left[R(DT) = O(\log n \frac{L}{W})\right] \geq 1 - \frac{n}{t}\mathrm{e}^{-\log n}$$

$$Pr\left[R(GT) = O(\log n (\frac{L}{W})^2)\right] \geq 1 - (1 + \frac{n}{t})\mathrm{e}^{-\log n}$$

2.5 Minimum Triangulation heuristics

A number of other heuristics have been proposed in literature [PH87, Lin87]. [Lin87] presents an $O(n^3)$ algorithm assures that the R-factor is $O(\log n)$ with probability greater than $1 - cn^{-\alpha}$, where $c > 0$ and $\alpha > 1$. This algorithm has the following basic steps:

- Find the convex hull, ($CH(S)$, of the input set S.
- Find the minimum length planar forest, F, connecting $CH(S)$ with the internal points.
- Find the minimum weight triangulation of the polygon $CH(S) \cup F$
- This triangulation with polygon edges added yields the final triangulation.

2.6 Optimizing Triangle Properties

Given that the optimal triangulation is a tough problem to solve, we need to find approximations to the minimum total weight condition. Indeed many applications do not need the total weight of the triangles to be small; some need a small number of triangles, some others may need small triangles, and still other need triangles with small angles. The Delaunay triangulation of a set of points, for example, maximize the minimum angle of any triangle, thus assuring that triangles are not long and skinny in general.

[SH75] erroneously claimed that Delaunay triangulation was the minimum weight triangulation. [BEY91] proved a variety of properties about Delaunay triangulation. It assumes Poisson distribution on the input point set.

Let $\mathcal{S}$ be a $[0, \sqrt{n}] \times [0, \sqrt{n}]$ square. Its results (Table 1) hold for the part of the Delaunay triangulation that lies entirely inside the square.

Property	Expected value
Maximum vertex degree	$\theta(\frac{n}{\log\log n})$
Maximum length of an edge	$\theta(\log^{\frac{1}{2}} n)$
Minimum angle of a triangle	$\theta(n^{-\frac{1}{2}})$
Maximum angle of a triangle	$\pi - \theta(n^{-\frac{1}{5}})$

Table 1: Properties of Delaunay triangulation

[ETW90, ETW92] shows an algorithm that minimizes the maximum angle of any triangle in $O(n^2 \log n)$ time. It starts with an arbitrary triangulation and improves it iteratively by 'edge-insertion'. This algorithm can also be used to minimize the sorted angle vector, if the points are in general position. At the 'edge-insertion' step, all old edges that intersect the new edge are deleted, and the polygons on the two sides of the new edge are retriangulated. Although a local operation, [ETW90, ETW92] proves that a global optimum is reached. It is important how this new edge uv is chosen. If $\angle xuy$ is the largest angle of the triangulation, an 'ear cutting' procedure finds uv such that $uv \in S - \{x, u, y\}$, and $xy \cap uv \neq \phi$.

[ET91, ET93] minimizes, instead of the maximum internal angle of the triangles, the maximum *length* of the any triangle. This is done in $O(n^2)$ time. It starts by finding the convex hull $ch(S)$ and the relative neighbourhood graph $rng(S)$ of S. An edge uv belongs to $rng(S)$ if

$$|uv| \leq \min_{x \in S - \{a,b\}} \max\{|xu|, |xv|\}$$

$ch(S) \cup rng(S)$ decomposes the convex hull of S into simple polygonal regions. These can be triangulated in $O(n^2)$ to give the minmax length triangulation of S. Note that an optimal triangulation of each of these polygons can be computed in $O(n^3)$, but such triangulation does not give a globally optimal minimum length triangulation.

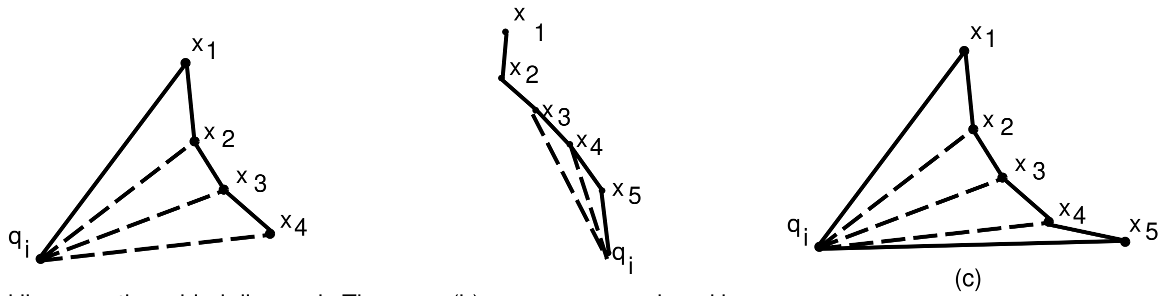
Delaunay triangulation is again found to be of use. [Raj91] proved that the Delaunay triangulation of a point set lies within a factor of $\frac{2}{\sqrt{3}}$ of the minimum length triangulation.

3 Polygon Triangulation

A polygon is a sequence of points (specified in either clockwise or anticlockwise order), or equivalently a sequence of edges. Only closed polygons are considered here. To triangulate such a polygon, we need to maintain The original set of edges must belong the triangulation. In addition, every point on the plane enclosed by P must also lie inside some triangle, and each triangle lies inside P.

This survey does not consider self intersecting polygons. Triangulation of such polygons does not fall under the spectrum of our definition of triangulation.

Unlike general points, every simple polygon can be triangulated. (In fact any planar straight line graph can be triangulated.) Still, historically, it was not easy to find it efficiently. This problem has a long and distinguished history. The triangulation of a simple polygon can be easily found in $O(n^3)$. For each of the $O(n^2)$ pairs of possible segments we can check if it intersects the polygon in $O(n)$ time. In fact we know of an $O(n^2)$ algorithm since at least the the beginning of the century [Len11]. If we are just interested in *legal* triangulation, we cannot expect to triangulate a simple polygon in better than $\Omega(n)$ time; all points need to be a part of the triangulation. For polygons with holes, [AAP86] proves a lower bound of $\Omega(n \log n)$.



(a) Dashed lines are the added diagonals The remaining polygon is formed from the old polygon by replacing $q_i x_1 x_2 x_3 x_4$ by $q_i x_4$

(b) $x_3 x_4 x_5 q_i$ replaced by $x_3 q_i$

Figure 2: Three cases of [GJPT78]

A line segment between two points of a simple polygon P is called a *diagonal* if it lies completely inside the polygon:

Consider the left most point u of a polygon P. Say, u is adjacent to points v and w, respectively. If the edge vw does not intersect P, it is a diagonal. Otherwise vertices of P lies inside the triangle uvw. The segment ut is a diagonal, if d is the farthest from the line passing through v and w, and lies on the same side of that line as u.

This means a diagonal can be found in $O(n)$ time, thereby implying that a polygon can be triangulated in $O(n^2)$ time.

For restricted cases, we can do better – clearly a convex polygon can be triangulated in $O(n)$. Each edge between points on the polygon is a diagonal. In fact a monotone polygon (a polygons that can be split into two monotone chains *wrt* any line) can be triangulated in $O(n)$ ([Gho83, Tou83]). A simple algorithm by [GJPT78] is outlined below. (We can assume without any loss of generality that the polygon is monotone in the `Y` coordinate.

Sort the vertices from top to bottom: $q_0, q_1, \ldots q_n$.
(These vertices are considered in that order. Note that sorting takes only $O(n)$)
Push q_0 and q_1 onto a stack.
Let $x_1, x_2, \ldots x_j$ be the content of the stack, and q_i be the next vertex to be processed
While there is something left to be triangulated, do
begin
    If q_i is adjacent to x_1 but not x_j (Fig.3(a))
        Add diagonals $(q_i, x_2), (q_i, x_3), \ldots q_i, x_j)$; $Stack \rightarrow \ (x_j, q_i)$
    else if q_i is adjacent to x_j but not x_1 (Fig.3(b))
    Repeat
        Add diagonal (x_{j-1}, q_i);
        Delete x_j from the stack;
        $j \leftarrow j - 1$;
    until j $= 1$ or the internal $\angle x_i \geq 180°$;
    Push q_i (onto the stack)
    else (q_i is adjacent to both x_1 and x_j) (Fig.3(c))
        Add diagonals $(q_i, x_2), (q_i, x_3), \ldots (q_i, x_{j-1})$ and stop.
end

For general simple polygons, the time complexity was not improved till quite recently when a number of $O(n \log n)$ algorithms [GJPT78, B.C82, HM83, FM84] were proposed in the late 70's and 80's. Still the gap between the known lower bound, $\Omega(n)$, and upper bounds kept challenging computational geometers for long. Finding a linear time algorithm remained open problem; [TV86]'s algorithm was mistakenly thought to be linear for some time. In 1988 [TV88] brought it down to $O(n \log \log n)$, showing that simple polygon triangle was easier than sorting. [CTV89] and [Cha90a] improved it to $O(n \log *n)$ subsequently. Finally [Cha91] showed that polygon triangulation can indeed be performed in asymptotically linear time. Unfortunately, in practice, this is not the recommended algorithm. We need further research to find simple an practical algorithm that will 'actually' run faster than the simple $O(n \log n)$ algorithms on data of interest. It turns out that if the polygon is allowed to have holes we cannot do better than $\Omega(n \log n)$ [AAP86].

Year	Complexity	Reference
1911	$O(n^2)$	[Len11]
1978	$O(n \log n)$	[GJPT78]
1983	$O(n \log r)$ (r = # concavities)	[HM83]
	$O(n \log s)$ (s = sinousity)	[CI84]
1988	$O(n + nt_0)$ (t_0 = # triangles with no edges on ∂P)	[Tou90]
	$O(n \log \log n)$	[TV88]
	$O(n \log *n)$ (randomized)	[CTV89]
1990	$O(n \log \log n)$	[KKT90]
	$O(n \log *n)$ (bounded integer domain)	[KKT90]
	$O(n \log *n)$ (deterministic)	[Cha90a]
1991	$O(n)$	[Cha91]

Table 2: History of Polygon Triangulation (modified from [O'R94])

3.1 Improvement in Algorithms

[GJPT78] triangulates a simple polygon by decomposing it into monotone polygons and then triangulate each of them separately using the algorithm described earlier. This decomposition uses the regularization procedure introduced in [LP77]. This procedure takes $O(n \log n)$ and adds non-intersecting diagonals to the polygon which do not cross the polygon boundary. Thus the total complexity of the algorithm remains $O(n \log n)$.

[B.C82]'s algorithm is particularly easy to implement. It finds a diagonal of the polygon P that divides it into two polygons $P1$ and $P2$, such that the number of points in $P1$ and $P2$ are each less than $\frac{2}{3}|P| + 2$. The algorithm finds such a diagonal in $O(|P|)$ time. Given that, a simple divide and conquer algorithm yields a total complexity of $O(n \log n)$.

[HM83] combines the steps of [GJPT78] into one sweep to yield an $O(n \log n)$ algorithm. It then improves it to $O(n \log r)$ (where r is the number of concave angles (internal angles $> 180°$) with in the input polygon) by restricting the sweep event points to only $O(r)$ points of the polygon. The sweep line also is no longer a simple straight line. A crooked line is carried, since some points do not get processed till the actual sweep line is well past them. This algorithm works even if the (simple) polygon has (polygonal) holes.

[CI84], like [HM83], depends on the complexity of the polygon. It defines a new measure *sinuosity*. Any simple polygon can be decomposed into alternating sequences of spiraling and antispiraling chains. The number of such chains is the sinuosity of the polygon. (In particular start shaped polygons have a sinuosity of 1.) Its complexity is $n \log s$.

The attempts to start with the 'montonization' or 'diagonal splitting' path failed till [CI84] and [FM84] showed the equivalence of trapezoidal decomposition with triangulation for simple

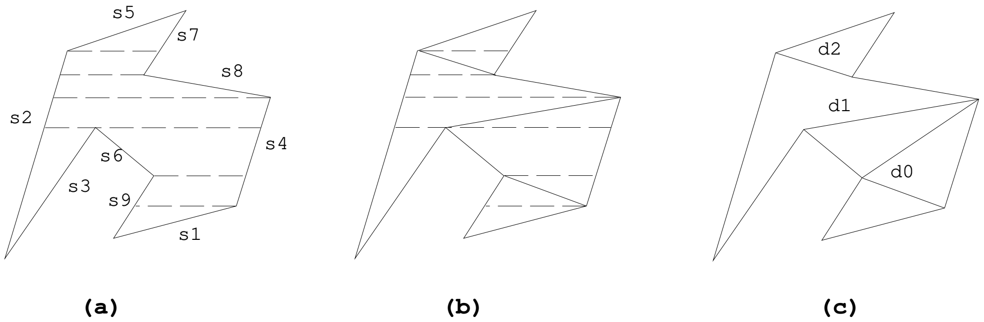


Figure 3: Generating monotone polygons from the trapezoid formation

polygons, and many recent efforts have been concentrated on polygon trapezoidalization (of course, a trapezoidalization can be used for monotone decomposition, as discussed below). Horizontal trapezoidalization is also called horizontal visibility map – it gives the edges on the left and the right of a vertex that are visible from it, via a horizontal ray.

[FM84] constructs trapezoids with two vertical sides and are determined by two vertices of the input polygon P. It is a deterministic sweep line algorithm to find the trapezoidalization in $O(n \log n)$. Once trapezoidalized, P is broken into monotone polygons and triangulated. Trapezoids that are determined by the two points (of the polygon) that are neither adjacent on the polygon nor on the trapezoid, can be triangulated by joining those two points. When all such trapezoids are exhausted, only monotone polygons remain to be triangulated.

[TV88] used trapezoidalization to show that triangulation is simpler than sorting. The complexity of his algorithm is $O(n \log \log n)$. This algorithms uses complex data structures and complicated steps, and is not easy to implement. Though practically of modest consequence, it was a major theoretical jump. This algorithm is based on a result that *Jordan sorting* can be done in linear time [HMRT86]. For a simple polygon P and a horizontal line L, the Jordan sorting problem is to sort the intersection points of ∂P (the boundary of P) and L by X coordinate, given as input only a list of the intersections in the order in which they occur clockwise around ∂P. (the list of vertices of P are not given.)

[TV88] starts with a vertex v of P that is neither the highest or the lowest vertex. (If there are no such vertices, the visibility computation is done.) It then determines the intersection points of ∂P with the horizontal line L through v. The corresponding points are then Jordon sorted and the visible pairs correspond to the consecutive intersection points along L. In fact these two steps are combined into one. Only a few intersections ([TV88] calls them *essential intersections*) before the sorting routine is called. If the routine fails, some more intersection points are evaluated and the process iterated. Now P is subdivided into smaller polygons along L and the their triangulation recursively computed.

[KKT90]'s $O(n \log \log n)$ algorithm is much simpler. In addition, this algorithm can be modified to run in $(n \log *n)$ if the coordinates of the points of the polygon are integers bounded by a fixed polynomial in n[1]. The basic idea is the same as [TV88], but the triangle splitting is achieved without any Jordan sorting or other complicated data structures of [TV88]. [KKT90] is based on the construction of the visibility partition of a polygonal chain P_0 from the partitions of some prefix chain P_1 of P_0 and the rest, $P_0 \setminus P_1$. For bounded integer coordinates, this algorithm build a data structure in linear time that can answer queries about horizontal neighbours in $O(\log^2 n)$.

[1] As they are in, for instance, many computer graphics applications, when points are specified in screen coordinates.

[CTV89] is a randomized divide and conquer based algorithm that finds the visibility partition of the polygons *wrt* a random subset of edges. The polygon is then subdivided into smaller polygon about that partition using Jordan sorting. This subdivision is then used to recurse. The algorithm terminates in expected $O(n \log *n)$. It can also be used to check whether or not a given polygon is simple. [CCT92] reports a version that eliminates the use of Jordan sorting, and is similar in flavor to the result introduced next.

[Sei91] is a randomized incremental algorithm, much simpler than [CTV89]. It starts by randomly ordering the edges of the input polygon. These edges are then added to the trapezoidal data structure one by one in that order, updating the trapezoidalization each time (see Fig. 3.1). The trapezoids are decompose into monotone polygons in linear time – these polygons are computed from the trapezoidal decomposition by checking whether the two vertices of the original polygon lie on the same side. [Sei91] proves that if each permutation of $s_1, \ldots, s_n$ is equally likely then trapezoid formation takes $O(n \log^* n)$ expected time.

The linear time algorithm of [Cha91] also constructs the horizontal visibility map of the polygon. The basic idea is the same as that of merge sort. It first builds a tree bottom-up constructing 'coarse' visibility map. This process does not consider all edges. Then, in a second top-down phase, the visibility map is refined incrementally by merging submaps. The visibility map really has a tree structure and can be written as a collection of 'blobs' of roughly equal sizes, themselves connected like a tree. These blobs make up the coarse map, and this tree structure is kept of degree ≤ 4. This bound is maintained through out the algorithm, making sure that the merges do no take too long.

3.2 Polygon triangulation with guarantees

The $O(n^3)$ algorithm mentioned in the beginning can be modified to compute the minimum weight triangulation – that is not NP-complete for the polygon case [Kli80]. [Ram91] gives an $O(n \log n)$ algorithm for minimum weight triangulation if the the vertices have weights. A tight lower bound of $\Omega(n \log n)$ is also shown.

[ETW90, ETW92] can be used to minimize the maximum angle, while minimizes the maximum edge length. [LL92] shows an alternate scheme that takes only $O(n)$ time for simple polygon and $O(n \log n)$ for planar straight line graph, as opposed to [ET91, ET93]'s $O(n^2)$, but it does not guarantee minimality. It can be up to 3 times more than the minimum. In general, any *c-sensitive* triangulation of a planar point set approximates the minmax triangulation within a factor of $2(c+1)$. [LL92] also proves that the greedy triangulation and the Delaunay triangulation of a planar straight line graph are 4-sensitive and 1-sensitive respectively. c-sensitivity is defined as follows:

> Let G be a planar straight line graph and let $c > 0$ A diagonal a of G is said to be c-sensitive if for any other diagonal b that properly intersects a,
>
> $$\frac{d}{c} \leq |b|$$
>
> where $d =$ the distance of any endpoint of b to the closest endpoint of a. A partial triangulation of G of G is said to be c-sensitive if all diagonals forming it are c-sensitive.

The Delaunay triangulation of a polygon (a constrained Delaunay triangulation) was shown to be computable in $O(n \log n)$ by [LL86]. [KL93] improves that to a randomized algorithm with expected running time to be $O(n)$.

[KB92] minimizes the maximum degree of the triangulation of any planar graph. Such triangulation is specially useful for re-triangulation when local changes are made to the input point set. Unfortunately it turns out to be an NP-complete problem. But there exists a linear

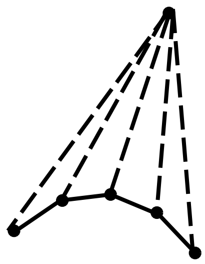


Figure 4: There exist only one triangulation

time algorithm that triangulates planar graphs with the maximum degree being only a constant factor of the lower bound.

3.3 Steiner Points

Sometimes, there exists a unique triangulation of a given polygon (Fig.3.3), which may not satisfy quality criteria. If we could introduce extra points inside such polygons, we may expect to find better triangulation, in terms of these quality criteria. These points are referred to as *steiner points*.

[KKT90] showed that the problem of triangulating a simple polygon with steiner points is linear time equivalent to the steiner free triangulation, if we are interested only in a legal triangulation.

[BGR88] computes a triangulation that has no obtuse triangles. It does this by overlaying a square grid on the polygon. It also proves that there always exists an obtuse triangulation if the vertices of the polygon lie on a square grid. General polygons can be triangulated with no angles smaller than $\tan^{-1}(\frac{1}{4})$; up to $O(n^2)$ triangles may be needed.

Using only $O(n)$ triangles [BDE92] guarantees that the smallest height of the triangle is approximately as large as allowed by the input polygon. Using $(n \log n)$ triangles $O(n^{\frac{3}{2}})$ for polygons with holes) they guarantee that no triangles have more than $150°$ internal angles. For convex polygons [BDE92] uses $O(n^{1.85})$ triangles and produces non-obtuse triangulation.

[BE91] triangulates a polygon into $O(n^2)$ non-obtuse triangles. This algorithm does not restrict the polygon to be simple – it triangulates polygons with holes. [BE91] also describes how to triangulate a convex polygon into $O(n^2)$ right triangles.

[BMR94] reports a $O(n^2 \log n)$ time algorithm that improves the bound for general polygons to $O(n)$ non-obtuse triangles. This algorithm first packs the polygon with non-overlapping discs that are tangent to each other and to the polygon edges. Each uncovered region of the polygon has four sides now. The algorithm adds edges between centers of the discs and points of tangencies. This subdivides the polygon into smaller polygons, which are triangulated independently.

[Mit93] triangulates a general planar straight line graph in $O(n^2 \log n)$ time adding $O(n^2 \log n)$ steiner points with no angle larger than $7\frac{\pi}{8}$. (There is a lower bound of $O(n^2)$ triangles for any triangulation that guarantees angles smaller than $180 - \epsilon, \epsilon > 0$ [BDE92].) It improves a given triangulation by starting at an obtuse angle and explores the triangulation in search for a sequence for steiner points that satisfies the angle condition locally.

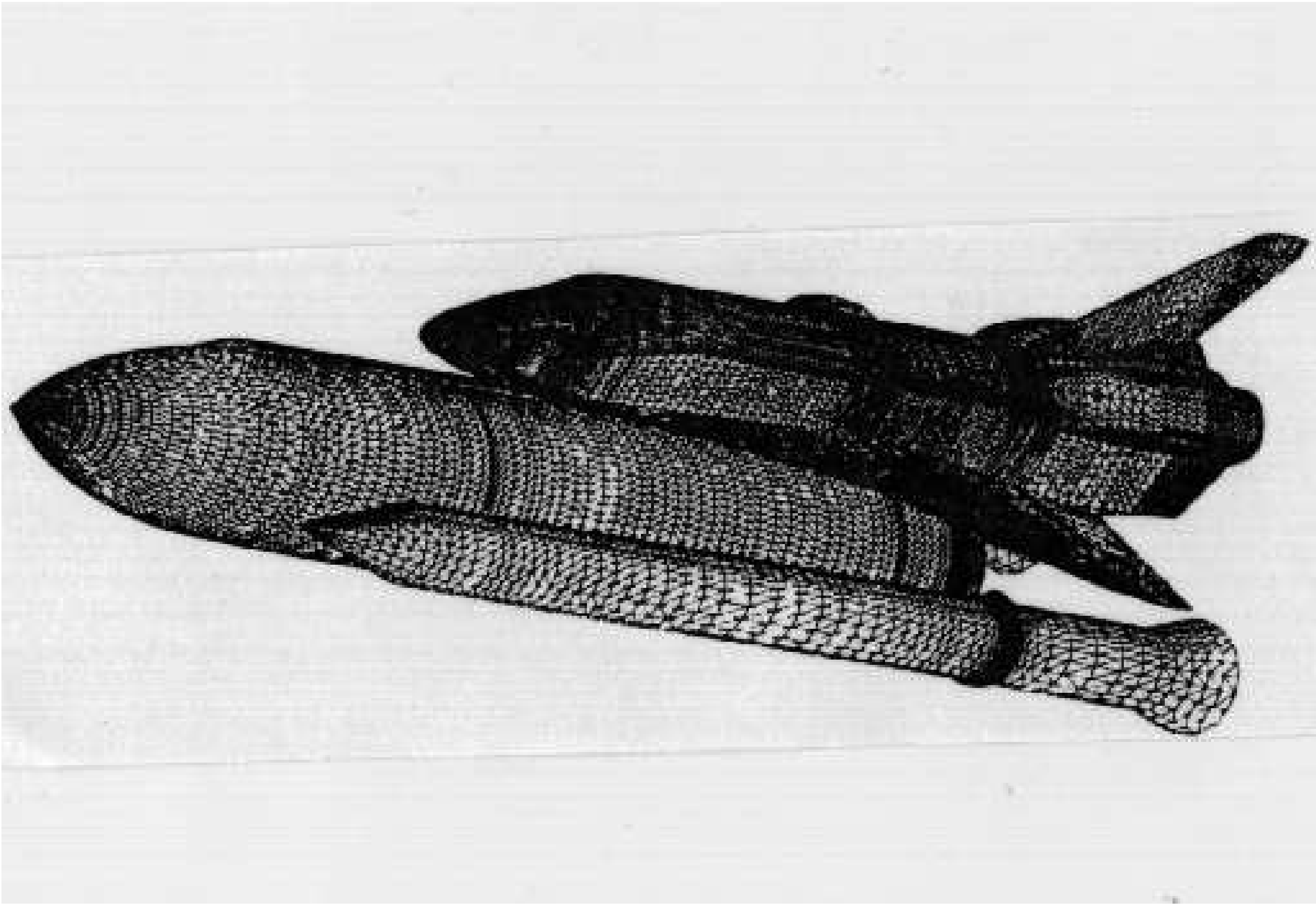


Figure 5: Example of a triangulated surface

3.4 Simpler Polygons

While one thread of research had been trying to improve the time complexity of simple polygon triangulation, another thread went off finding linear triangulation algorithms for restricted classes of polygons. As noted earlier, convex or monotone polygons can be triangulated in linear time. A star shaped polygon can also be triangulated in linear time using an ear cutting procedure.

[PS81] shows how to test if a given simple polygon is monotonic in $O(n)$ time. [TA82] show how to triangulate a class of polygons called *edge-visible* polygons in linear time. An edge visible polygon contains at least one edge such that every point inside the polygon is visible from some point on that edge. Given a general polygon, testing if it is simple does no seem to be all that simple: [Jar84] shows a lower bound of $O(n \log n)$.

4 Surfaces

The triangulation of a surface is just an extension of planar triangulations and finds application in computer vision, geometric modeling, geographic data representation, function interpolation, virtual reality etc. Typically, we sample the surface at a number of points and triangulate these points 4. Such surfaces are sometimes said to be 2.5 dimensional, as they are 2 dimensional object in $\mathcal{R}^3$ A large number of papers [Baj90, AES93, ea91, ea89, LR81, Luk93, LC93, Che93, SC88, SL87, FK90, RHD89, Roc87, AES91, FMM86] in computer graphics literature, concentrating on the rendering aspects of surfaces, use the triangulations of the surfaces. We typically look for three qualities in such surface triangulation algorithms:

1. Speed
2. quality of triangulation
3. generality of solution

[LR78] addresses all these issues. It triangulates a set of points and a bounding polygon. The bounding polygon is an important factor in surface triangulation because, typically surfaces are broken into smaller surfaces and each surface must be triangulated independently. It is required that the triangulations must match at each of these sub-surface boundaries. [LR78] computes such triangulation in $O(n \log n)$, and handles multiply connected regions. It divides the input data set into disjoint sections by splitting it along a line, and solves the problem on each of these sub-problems recursively. The split line is chosen so that it approximately lies in the 'middle' of the polygon. This is done by looking at the signed distance of such a line from the input point set, and taking a product. They report that this works quite well in practice. Finally this algorithm performs a post processing step flipping diagonals just as described earlier.

Many times a surface is decomposed into many different approximate triangulations – multi-resolution modeling [Tur92, HDD$^+$93, Var94] is one such example. It becomes imperative to ensure, then, that a triangulations does not alter the topology of the surface. [DC91] gives criteria for testing if two triangulations are of the same homology type. In other words, it answers if they are topologically similar.

[De 89] talks about representing different representations of the surface as a 'pyramid' of Delaunay triangulations that encodes triangulation of the surface at different levels of sampling on the surface. This algorithm builds the structure in $O(n^2)$.

4.1 Parametric Surfaces

Surfaces are typically presented using coefficients of a basis. Bernstein's basis is one of the most popular:

$$B_i^n(t) = \left(\begin{array}{c} n \\ i \end{array} \right) t^i (1-t)^{n-i}$$

Tensor product Bézier surfaces, for example, are extensively used in geometric modeling. A rational Bézier surface is given as $\mathbf{F}(u,v) = (X(u,v), Y(u,v), Z(u,v), W(u,v)) =$

$$(\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij}\mathbf{r}_{ij}B_i^m(u)B_j^n(v), \sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij}B_i^n(u)B_j^m(v)),$$

where $\mathbf{r}_{ij}$ are the control points in the object space, w_{ij} are the weights and B_i^m, B_j^n are the Bernstein polynomials. Given a triangulation in the (u,v) plane, that triangulation is imposed on the surface. Algorithms sometimes sample the surface on a set of grid points when the triangulation becomes trivial. Such tessellation is called uniform tessellation. Other algorithms sample the surface at arbitrary points – these are adaptive tessellations, and a general triangulation is required. Sometime triangulation is inherent in the sampling process, e.g. Coarse triangulation is generated and multiple levels of sampling is done. Samples within a triangle need to be triangulated within themselves. New samples are introduced inside each of these new triangles, and so on until a stopping criterion is reached.

[Roc87] samples the surface at a set of grid points. The grid size is determined at run time, and the triangulation changes as grid does. This algorithm handles trimmed surfaces also. A trimmed surface is a normal surface with some part trimmed out. Now, we not only have a set of points but boundary as well. [Roc87] does a monotonic decomposition of the boundary, and subdivides the surface such that each boundary is now monotonic. The problem can now be solved using grid point triangulation except the tessellation on the boundary. To solve this problem [Roc87] introduces steiner points where the grid intersects the boundary. If steiner points are not allowed, the problem can be as easily solved ([KM94]) by connecting the vertices of the bounding polygon to the closest grid points. The 'outermost' grid points are connected to the closest point on the polygon boundary.

Sometimes triangular surfaces are used to define general surfaces. All the issues of planar triangulation are equally relevant to these surfaces. An example of a triangular surface function is:

$$\psi(u, v, w) = \sum_{i+j+k_{i,j,k\geq 0}} G_{ijk} \frac{n!}{i!j!k!} u^i v^j w^k$$

$u + v + w = 1$.

4.2 Algebraic surfaces

Algebraic surfaces belong to the class of implicit surfaces which can be thought of as the set of points that form roots to an equation $f(x, y, z) = 0$. [HW90, Blo88] are some of the excellent sources on triangulation of implicit surfaces. [Blo88] surrounds the implicit surface with an octree [Mea82]. The surface is sampled at the corners of this octree. The algorithm proceeds to adaptively refine the octree and finally triangulates each node of the octree.

5 Conclusion

This survey is an attempt to present the current state of the art in surface triangulation. The richness of literature on this topic almost precludes an exhaustive listing of results. I have tried to present what I think are some of the more important and application oriented results in this area. I hope this will help a researcher get an idea of the known techniques, and will allow an implementer to decide which algorithm is best suited for their application.

References

[AAP86] Ta. Asano, Te. Asano, and R. Y. Pinter. Polygon triangulation: Efficiency and minimality. *J. Algorithms*, 7:221–231, 1986.

[ACNS82] M. Ajtai, V. Chvatal, M. Newborn, and E. Szemeredi. Crossing-free subgraphs. *Annals of Discrete Math.*, 12:9–12, 1982.

[AES91] S.S. Abi-Ezzi and L.A. Shirman. Tessellation of curved surfaces under highly varying transformations. *Proceedings of Eurographics'91*, pages 385–97, 1991.

[AES93] S.S. Abi-Ezzi and L.A. Shirman. The scaling behavior of viewing transformations. *IEEE Computer Graphics and Applications*, 13(3):48–54, 1993.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Baj90] C.L. Bajaj. Rational hypersurface display. In *Symposium on Interactive 3D Graphics*, pages 117–27, Snowbird, UT, 1990.

[B.C82] B.Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982.

[BDE92] M. Bern, D. Dobkin, and D. Eppstein. Triangulating polygons without large angles. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 222–231, 1992.

[BE91] M. Bern and D. Eppstein. Polynomial-size nonobtuse triangulation of polygons. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 342–350, 1991.

[BEY91] M. Bern, D. Eppstein, and F. Yao. The expected extremes in a Delaunay triangulation. *Internat. J. Comput. Geom. Appl.*, 1:79–91, 1991.

[BGR88] B. S. Baker, E. Grosse, and C. S. Rafferty. Nonobtuse triangulation of polygons. *Discrete Comput. Geom.*, 3:147–168, 1988.

[Blo88] J. Bloomenthal. Polygonization of implicit surfaces. Tech. Report EDL-88-4, Xerox PARC, Palo Alto, CA, 1988.

[BMR94] M. Bern, S. Mitchell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 221–230, 1994.

[CCT92] K. L. Clarkson, R. Cole, and R. E. Tarjan. Randomized parallel algorithms for trapezoidal diagrams. *Internat. J. Comput. Geom. Appl.*, 2(2):117–133, 1992.

[Cha90a] B. Chazelle. Efficient polygon triangulation. *Preprint. Probably unpublished – precursor to [Cha90b]*, 1990.

[Cha90b] B. Chazelle. Triangulating a simple polygon in linear time. In *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 220–230, 1990.

[Cha91] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[Che89] L. Chew. Constrained delaunay triangulation. *Algorithmica*, 4:97–108, 1989.

[Che93] F. Cheng. Computation techniques on nurb surfaces. In *SIAM Conference on Geometric Design*, Tempe, AZ, 1993.

[CI84] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Trans. Graph.*, 3(2):135–152, 1984.

[CR90] A. K. Cline and R. J. Renka. A constrained 2-dimensional triangulation and the solution of closest node problems in the presence of barriers. *SIAM J. Numer. Anal.*, 27(5):1305–1321, 1990.

[CSS83] Z. J. Cendes, D. N. Shenton, and H. Shahnasser. Magnetic field computation using Delaunay triangulation and complementary finite element methods. *IEEE Trans. Magn.*, MAG-19(6), 1983.

[CTV89] K. Clarkson, R. E. Tarjan, and C. J. Van Wyk. A fast Las Vegas algorithm for triangulating a simple polygon. *Discrete Comput. Geom.*, 4:423–432, 1989.

[DC91] B. R. Donald and D. R. Chang. On the complexity of computing the homology type of a triangulation. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 650–662, 1991.

[DDMW94] M. T. Dickerson, R. L. S. Drysdale, S. A. McElfresh, and E. Welzl. Fast greedy triangulation algorithms. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 211–220, 1994.

[De 89] L. De Floriani. A pyramidal data structure for triangle-based surface representation. *IEEE Comput. Graph. Appl.*, 9:67–78, March 1989.

[DG70] Duppe and Gottschalk. Automatische interpolation von isolinen bei willkurlichen stutzpunkten. *Algrveine Vermessungsnachricten*, 77, 1970.

[ea89] T. Derose et al. Apex: two architectures for generating parametric curves and surfaces. *The Visual Computer*, 5:264–276, 1989.

[ea91] R. Bedichek et al. Rapid low-cost display of spline surfaces. In *Proceedings of advanced reserach in VLSI*, MIT Press, 1991.

[ET91] H. Edelsbrunner and T. S. Tan. A quadratic time algorithm for the minmax length triangulation. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 414–423, 1991.

[ET93] H. Edelsbrunner and T. S. Tan. A quadratic time algorithm for the minmax length triangulation. *SIAM J. Comput.*, 22:527–551, 1993.

[ETW90] H. Edelsbrunner, T. S. Tan, and R. Waupotitsch. An $O(n^2 \log n)$ time algorithm for the minmax angle triangulation. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 44–52, 1990.

[ETW92] Herbert Edelsbrunner, Tiow Seng Tan, and Roman Waupotitsch. $O(N^2 \log N)$ time algorithm for the minmax angle triangulation. *SIAM J. Sci. Statist. Comput.*, 13(4):994–1008, July 1992.

[FK90] D.R. Forsey and V. Klassen. An adaptive subdivision algorithm for crack prevention in the display of parametric surfaces. *Proceedings of Graphics Interface*, pages 1–8, 1990.

[FLB90] O. D. Faugeras, E. Le Bras-Mehlman, and J. D. Boissonnat. Representing stereo data with the Delaunay triangulation. *Artif. Intell.*, 44(1–2):41–87, July 1990.

[FM84] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Trans. Graph.*, 3(2):153–174, 1984.

[FMM86] D. Filip, R. Magedson, and R. Markot. Surface algorithms using bounds on derivatives. *CAGD*, 3:295–311, 1986.

[Gho83] S. K. Ghosh. A linear time algorithm for decomposing a monotone polygon into star-shaped polygons. In *Proc. 3rd Conf. Found. Softw. Tech. Theoret. Comput. Sci.*, pages 505–519, India, 1983.

[GJPT78] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett.*, 7:175–179, 1978.

[Gol89] S. Goldman. A space efficient greedy triangulation algorithm. *Inform. Process. Lett.*, 31(4):191–196, 1989.

[HDD+93] H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and W. Stuetzle. Mesh optimization. In *ACM SIGGRAPH*, pages 19–26, 1993.

[HM83] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In *Proc. 4th Internat. Conf. Found. Comput. Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 207–218. Springer-Verlag, 1983.

[HMRT86] K. Hoffman, K. Mehlhorn, P. Rosenthiel, and R. Tarjan. Sorting jordan sequence in linear time usuing level-link search trees. *Inform. and Control*, 68:170–184, 1986.

[HW90] Mark Hall and Joe Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics and Applications*, 10(6):33–42, November 1990.

[Jar84] J. W. Jaromczyk. Lower bounds for polygon simplicity testing and other problems. In *Proc. Math. Found. of Computer Science '84*, pages 339–347. Springer-Verlag, 1984.

[KB92] G. Kant and H. L. Bodlaender. Triangulating planar graphs while minimizing the maximum degree. In *Proc. 3rd Scand. Workshop Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 258–271. Springer-Verlag, 1992.

[Kir80] D. G. Kirkpatrick. A note on delaunay and optimal triangulations. *Info. Proc. Letters.*, 10:127–128, 1980.

[KKT90] D. G. Kirkpatrick, M. M. Klawe, and R. E. Tarjan. Polygon triangulation in $O(n \log\log n)$ time with simple data structures. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 34–43, 1990.

[KL93] R. Klein and A. Lingas. A note on generalizations of Chew's algorithm for the Voronoi diagram of a convex polygon. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 370–374, Waterloo, Canada, 1993.

[Kli80] G. T. Klincsek. Minimum triangulation of polygonal domains. *Annals of Discrete Math.*, 9:121–123, 1980.

[KM94] S. Kumar and D. Manocha. Interactive display of large scale nurbs models. Technical Report TR94-008, Department of Computer Science, University of North Carolina, 1994.

[LC93] W.L. Luken and Fuhua Cheng. Rendering trimmed nurb surfaces. Computer science research report 18669(81711), IBM Research Division, 1993.

[Len11] N. J. Lennes. Theorems on simple polygon and polyhedron. *Amer. J. Math.*, 33:37–62, 1911.

[Lev87] C. Levcopoulos. An $\Omega(\sqrt{n})$ lower bound for the nonoptimality of the greedy triangulation. *Inform. Process. Lett.*, 25:247–251, 1987.

[Lin86] A. Lingas. *Info. Proc. Letters.*, 22(1):25–31, 1986.

[Lin87] A. Lingas. A new heuristic for minimum weight triangulation. *SIAM J. Algebraic Discrete Methods*, 8(4):646–658, 1987.

[Lin89] Andrzej Lingas. Greedy triangulation can be efficiently implemented in the average case. In *Graph-Theoretic Concepts in Computer Science. Proceedings.*, 1989.

[Lis94] Dani Lischinski. Incremental Delaunay triangulation. In Paul Heckbert, editor, *Graphics Gems IV*, pages 47–59. Academic Press, Boston, MA, 1994.

[LL86] D. T. Lee and A. K. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete Comput. Geom.*, 1:201–217, 1986.

[LL87] C. Levcopoulos and A. Lingas. On approximation behavior of the greedy triangulation for convex polygons. *Algorithmica*, 2:175–193, 1987.

[LL90] C. Levcopoulos and A. Lingas. Fast algorithms for greedy triangulation. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 238–250. Springer-Verlag, 1990.

[LL92] C. Levcopoulos and A. Lingas. C-sensitive triangulations approximate the minmax length triangulation. In *Proc. 12th Conf. Found. Softw. Tech. Theoret. Comput. Sci.*, New Delhi, India, 1992.

[Llo77] E. Lloyd. Dt and gt are non-optimal. In *Proc. 18th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 228–240, 1977.

[LP77] D.T. Lee and F.P. Preparata. Location of a point in a planar subdivision and its application. *SIAM J. Comput.*, 6:594–606, 1977.

[LR78] B. A. Lewis and J. S. Robinson. Triangulation of planar regions with applications. *Comput. J.*, 21:324–332, 1978.

[LR81] J.M. Lane and R.F. Riesenfeld. Bounds on polynomials. *BIT*, 2:112–117, 1981.

[LSFB88] E. Le Bras-Mehlman, M. Schmitt, O. D. Faugeras, and J. D. Boissonnat. How the Delaunay triangulation can be used for representing stereo data. In *Second International Conference on Computer Vision*, pages 54–63, New York, NY, 1988. IEEE.

[Luk93] W.L. Luken. Tessellation of trimmed nurb surfaces. Computer science research report 19322(84059), IBM Research Division, 1993.

[Mau84] A. Maus. Delaunay triangulation and the convex hull of n points in expected linear time. *BIT*, 24:151–163, 1984.

[Mea82] D. Meagher. Geometric modeling using octree encoding. *Comput. Graph. Image Process.*, 19(2):129–147, 1982.

[MF85] D. Montuno and A. Fournier. A linear time triangulation for simple polygons. Report ??, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1985.

[Mit93] Scott A. Mitchell. Refining a triangulation of a planar straight-line graph to eliminate large angles. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS 93)*, pages 583–591, 1993.

[MV93] J.-M. Moreau and P. Volino. Constrained Delaunay triangulation revisited. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 340–345, Waterloo, Canada, 1993.

[MZ79] G. K. Manacher and A. L. Zobrist. Neither the greedy nor the Delaunay triangulation of a planar point set approximates the optimal triangulation. *Inform. Process. Lett.*, 9:31–34, 1979.

[Nei66] B. O' Neill. *Elementary Differential Geometry.* Academic Press, 1966.

[O'R94] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[PH87] D. A. Plaisted and J. Hong. A heuristic triangulation algorithm. *J. Algorithms*, 8:405–437, 1987.

[PS81] F. P. Preparata and K. J. Supowit. Testing a simple polygon for monotonicity. *Inform. Process. Lett.*, 12:161–164, 1981.

[Raj91] V. T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 357–363, 1991.

[Ram91] P. Ramanan. A new lower bound technique and its application: tight lower bound for a polygon triangulation problem. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 281–290, 1991.

[RHD89] A. Rockwood, K. Heaton, and T. Davis. Real-time rendering of trimmed surfaces. In *Proceedings of ACM Siggraph*, pages 107–117, 1989.

[Roc87] A. Rockwood. A generalized scanning technique for display of parametrically defined surface. *IEEE Computer Graphics and Applications*, pages 15–26, August 1987.

[SC88] M. Shantz and S. Chang. Rendering trimmed nurbs with adaptive forward differencing. In *Proceedings of ACM Siggraph*, pages 189–198, 1988.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

[SH75] M. Shamos and D. Hoey. Closest pair problems. In *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 151–162, 1975.

[Sha78] M. Shamos. *Computational Geometry.* PhD thesis, Dept. Of Comput. Sci., Yale Univ., 1978.

[SL87] M. Shantz and S. Lien. Shading bicubic patches. In *Proceedings of ACM Siggraph*, pages 189–196, 1987.

[TA82] G. T. Toussaint and D. Avis. On a convex hull algorithm for polygons and its application to triangulation problems. *Pattern Recogn.*, 15:23–29, 1982.

[Tou83] G. T. Toussaint. A new linear algorithm for triangulating monotone polygons. Technical Report SOCS 83.9, McGill University, 1983.

[Tou90] G. T. Toussaint. An output-sensitive polygon triangulation algorithm. In *Proc. 8th Internat. Conf. on Comput. Graphics*, pages 443–446, 1990.

[Tur92] G. Turk. Re-tiling polygonal surfaces. In *ACM SIGGRAPH*, pages 55–64, 1992.

[TV86] R. E. Tarjan and C. J. Van Wyk. A linear-time algorithm for triangulating simple polygons. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 380–388, 1986. Erratum in SIGACT News 18:4 (1987), 63.

[TV88] R. E. Tarjan and C. J. Van Wyk. An $O(n \log \log n)$-time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143–178, 1988.

[Var94] A. Varshney. *Hierarchical Geometric Approximations*. PhD thesis, University of North Carolina, 1994.

[Yvi89] M. Yvinec. Triangulation in 2D and 3D space. In *Geometry and Robotics Workshop Proceedings, Toulouse France, 26–28 May 1988*, volume 391 of *Lecture Notes in Computer Science*, pages 275–291, Berlin, 1989. Springer-Verlag.

[ZSZZ90] Jian-Ming Zhou, Ke-Ran Shao, Ke-Ding Zhou, and Qiong-Hua Zhan. Computing constrained triangulation and Delaunay triangulation: A new algorithm. *IEEE Trans. Magn.*, 26(2):694–697, March 1990.